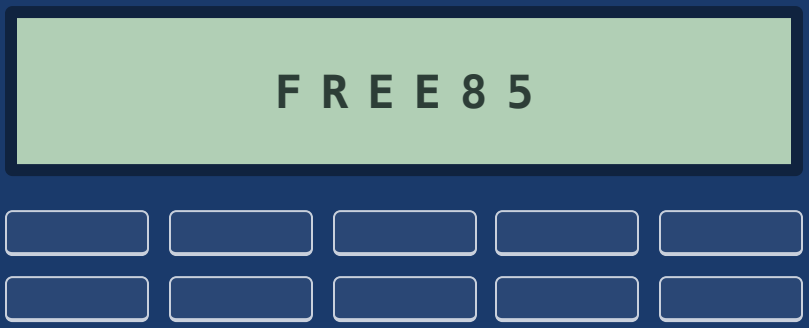


SCIENTIFIC GRAPHING CALCULATOR

Explorations with Free85

For firmware 3.0 · Second Edition

chriswilson2020.github.io/Free85

Explorations with Free85

Explorations with Free85 is a workbook. Its eight chapters follow the order of a mathematics course, from precalculus through to engineering mathematics. Each section asks a question about the mathematics, works it through on the machine keystroke by keystroke, and then hands you exercises to carry on with on your own.

It is a companion to the two books that already exist, and it does not re-teach them. The Free85 Getting Started Manual gets you running, from the first key press to storing your own work. The Free85 Guidebook is the reference for every command and every key, subject by subject. This book puts the two of them to work. It assumes the calculator is in your hand and asks what you can actually find out with it.

Who is writing this

I wrote the firmware.

That is worth saying because it changes what this book can tell you. Free85 is a clean-room calculator ROM built from scratch, and I have spent long enough inside its arithmetic to know why it does most of what it does. So when you hit an edge in these chapters, and you will hit plenty, I can usually tell you what is behind it instead of leaving you to conclude that the machine is simply badly made.

Some of those edges have good reasons and some of them are things I would do differently now. Both are more useful to you than silence. Where I am guessing, or where the answer is “it was the cheapest thing that worked”, I say so.

I have tried to do this without turning the book into a tour of the firmware. When a limit shapes the mathematics you are doing, it gets explained. When it does not, it stays out of the way.

An original work, and what that does and does not mean

This book was written for Free85 from scratch. Its prose, its screens and its numbers are its own. Every key sequence in it was pressed on the emulator and every quoted result was copied from the screen at full precision.

The mathematics is not its own, and does not pretend to be. The quotient sin x over x for limits, Newton's method for roots, the logistic equation for growth, the pendulum and its elliptic integral: these are the standard examples of their subjects. They are taught everywhere, they belong to nobody, and a workbook that avoided them in order to look original would simply be a worse workbook. You are better served by the example you will meet again in your course.

Where this book does have something of its own, it is in what a small machine does to those examples. That is a question nobody else has had to answer, and it turns out to be a better one than I expected.

How to read it

The conventions belong to the Guidebook and this book inherits them:

- **Keys** appear in brackets using their keycap labels: [ENTER], [2nd], [ALPHA], [GRAPH], [F1] through [F5], and the cursor keys [▲] [▼] [◀] [▶]. A sequence such as [2nd] [F1] means press and release [2nd], then press [F1]. A bracketed letter such as [H] means the key carrying that letter legend, with [ALPHA] supplied where the context needs it.
- **On-screen text and typed expressions** appear in code spans: the status line shows `RAD AUTO`, and the entry line holds `X^3−4∗X`. Command, mode and variable names are set the same way.
- **Screenshots** are exact captures of the emulated 128 by 64 pixel screen. They are made by booting a fresh machine, pressing the listed keys and photographing the result, so they cannot drift away from the firmware without somebody noticing.
- **Diagrams** are the pictures the calculator did not draw. They are set plain, without the calculator's bezel around them, so a glance tells you which figures

came off the machine and which are explaining the mathematics behind it.

Every section closes with a **Try it** block of exercises. Nearly every one is answerable with the technique the section has just shown, and where one is not, the chapter says so and sends you to paper.

Chapter 9 works all of them out, with the key presses and the numbers the machine gives back.

I went back and forth about printing those. An answer at the back of the book is a temptation, and the machine really is the best answer key there is: press the keys and it tells you. But an answer you cannot check is worth very little when you are working alone, and being stuck with no way forward is how people stop.

So the answers are there, and I would ask you to use them the way you would use a friend who already knows: after you have had a proper go, and to find out *why*, not just *what*. Several of the solutions do something the exercise did not ask for, because the interesting part turned out to be somewhere else.

Some exercises ask you to predict or sketch something **before** you press a key, and tell you to write the guess down. Please do. Being wrong on paper and then finding out why is most of what these exercises are for, and it does not work if you look first.

Getting back to a known state

You will get lost. Everybody does, and the machine is not always forthcoming about where you are. These are the ways back.

- **The entry line never clears itself.** Whatever you typed last is still sitting there, and new typing gets added to the end of it. Press CLEAR before typing at the home screen. Always.
- **After an error screen, press CLEAR twice.** The first press dismisses the message; the second empties the line the message left behind. One press looks like it worked and then your next expression comes out mangled.
- **2nd + restores the standard window**, -10 to 10 on both axes, whenever an experiment has taken the graph somewhere unhelpful.
- **A plot must be allowed to finish.** Presses that arrive while the machine is still drawing are dropped, not queued. If a key seems dead, that is usually why. Let the curve reach the right-hand edge.

- **GRAPH stores whatever is on the entry line into the active slot**, including nothing. Returning to the plot from an empty line wipes the equation, so go back with the equation on the line.
- **The DifEq mode freezes its initial value** when the mode is first entered, and storing a new one into `Y` afterwards changes nothing. Section 7.6 has the deliberately stiff lever that resets it.

If all else fails, the memory browser at 2nd + and the Guidebook, chapter 18 will show you what the machine is actually holding.

A word on the machine's limits

Free85 keeps three graph slots, lists of eight samples, matrices no larger than 3 by 3, vectors of three components, polynomials to degree 4, and four program slots of eight lines each.

Those numbers are not accidents and they are not apologies. Each of them is a decision I made, mostly about how much of a very small machine to spend on one feature, and each of them shapes what an exploration can be. So the explorations here are designed inside the limits rather than around them.

A family of curves is studied three members at a time. A data set is eight numbers you can hold in your head. An elimination is small enough to watch every entry change, and a program is short enough to be an argument about what its algorithm really is.

Where a limit closes a door, this book says so, tells you why the door is there, and goes round. That turns out to be the most useful thing in it.

Contents

FRONT MATTER

Who is writing this i

An original work, and what that does and does not mean ii

How to read it ii

Getting back to a known state iii

A word on the machine's limits iv

CHAPTERS

1 Explorations in Precalculus 1

2 Explorations in Business Mathematics 19

3 Explorations in Probability and Statistics 39

4 Explorations in Calculus I 57

5 Explorations in Calculus II 91

6 Explorations in Linear Algebra 117

7 Explorations in Differential Equations 135

8 Explorations in Engineering Mathematics 161

9 Solutions 187

AFTERWORD

After the last exploration 247

CHAPTER

1

Explorations in Precalculus

Precalculus is where functions stop being formulas to evaluate and become objects to study: things with graphs, zeros, symmetries, families of relatives, and places they cannot go.

This chapter uses the graph screen as a laboratory for that shift. The mechanics of graphing, the slots and windows and tracing and analysis keys, are covered in full in the Guidebook, chapter 4. This chapter uses them rather than re-explaining them.

If you read only one section of this chapter, read the first. Almost every mistake anybody makes with a graphing calculator is a window mistake, and they all look like mathematics going wrong.

Functions and their windows 1.1

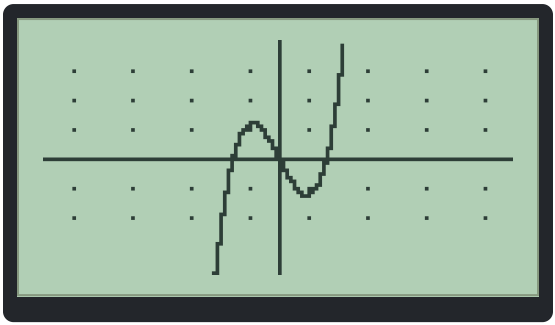
A graph is a partnership between a function and a window.

The function decides what there is to see. The window decides what you actually see, and a badly chosen window can hide a zero, flatten a wiggle, or crop the only interesting part of the curve. It will do all of that without any indication that it has, which is what makes it worth a section of its own.

The working example is the cubic f(x) = x^3 - 4x, which has three zeros and a small hill and valley between them: a shape worth framing deliberately.

1. On the home screen, type **x-VAR** **^** **3** **-** **4** **×** **x-VAR** so the entry line reads `X^3−4∗X`, then press **GRAPH**.

The home entry line is the equation editor. **GRAPH** stores the line into the active slot `Y1` and plots it in the standard window, -10 to 10 on both axes.



*The cubic X^3-4*X in the standard window*

2. Press ▶ twice to trace two columns right of centre. The readout gives `X=0.393700787402` and `Y=-1.5137794055131`.

   Those are not round numbers and they are not meant to be. Trace positions are the exact sample columns, spaced one 127th of the window width apart, and the second line is the function evaluated there at full precision. The trace does not go where you point it; it goes to the nearest place the machine actually computed.

3. Ask the window where it is. Press + once, the quick zoom-in, and the plot redraws with every bound halved.

   Press **EXIT** to return to the home screen, where the graph hands `X^3-4*X` back to the entry line, and press **CLEAR** to empty it. Typing `XMIN` (each letter is **ALPHA** plus the key carrying it) and pressing **ENTER** answers `-5`. From the standard window, one press of - instead answers `-20`.

The window bounds XMIN, XMAX, YMIN and YMAX are read-only *on the home screen*. To set one exactly, there is an editor: from the graph screen press 2nd GRAPH for the zoom panel, MORE twice, then F5, WIN.

The window editor, four bounds you can type

F1 to F4 pick a bound, you type a value, ENTER puts it in a draft, and F5 (SAVE) commits all four together after checking that the minimums really are below the maximums. Nothing reaches the plot until SAVE, so a half-typed window cannot leave you looking at nonsense and wondering which press did it.

What you type is an expression, not just digits, which is the whole value of the thing: 2*PI is a legal XMAX.

There was no editor when this book was first written, and I defended that at some length here: four fields, cursor handling and validation against zoom keys that cost one press each and almost no code. It was a reasonable trade and I no longer think it was the right one, because "one press each" is only cheap when you do not care exactly where you land, and section 1.6 is full of windows where you do.

2nd + restores the standard window whenever an experiment has taken you somewhere unhelpful. Learn that one now.

4. Now the zeros. With the cubic replotted in the standard window, press F1: the answer is = 0 with the residual line R=0.

   The root search starts from the traced position, and the trace reference sits at X=0 after a replot, which happens to be a zero of this cubic already. So that answer is correct and tells you nothing about the search.

   To find a different zero, move the trace first: press ◀ thirteen times, taking the trace near X=−2, and press F1 again. The answer is = −2 with R=0, the

exact leftmost zero.

That is the pattern for every analysis key in this book. Trace to the neighbourhood you care about, then ask. The keys answer questions about the window and the traced position you give them, and they have no way of knowing which root you meant.

5. Zeros of a pair meet the same way. Press 2nd 2 on the graph screen to switch to slot Y2 (the entry line comes back empty), type x-VAR, and press GRAPH: the line X joins the cubic.

   Trace fourteen columns left, then press 2nd F1, the intersection search. The answer is `= -2.2360679774997` with `R=-3E-13`: the negative square root of five, where `X^3-4*X` and `X` cross, with the residual showing how nearly the two sides agree there.

TRY IT

1. Y1 holds `X^3-4*X` and Y2 holds `X`. There are three intersections in the standard window. Predict all three on paper first, then find the other two with trace and 2nd F1, and check that the positive one is the square root of five.
2. Plot `X^3-4*X+3` and find all three of its zeros with trace and F1. Which of them could you have read straight from the table, and why?
3. Zoom in with + repeatedly until the cubic looks like a straight line through the origin, reading `XMIN` after each press. How many presses, and what line does the curve come to resemble? Work out the line from the formula before you count.
4. Find a window in which `X^3-4*X` appears to have only one zero. Then find one in which it appears to have none. Neither is a lie; say what question each window is answering.

1.2 Families of curves three at a time

A single graph answers a question about one function. A family answers a question about a parameter, which is usually the more interesting question.

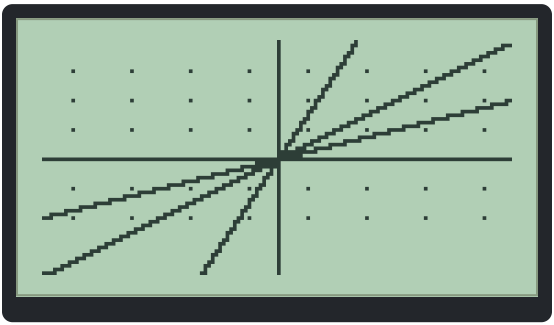
What does the coefficient m do to y = mx? What does a do to y = ax^2? Free85 keeps three function slots, so a family is explored three members per plot: choose three values of the parameter that bracket the behaviour, plot them together, and read the differences.

Three is a small number and it is worth knowing that it was chosen rather than fallen into. Each slot costs stored text, a parsed expression kept ready, and a

column of the table. Three fits comfortably in the memory this machine has and leaves room for everything else; a fourth would have come out of somewhere else you would rather keep. It also happens to be exactly the number you need for the commonest comparison in mathematics: a thing, and one of it either side.

1. Store the slope family. Type [x-VAR] [÷] [2] and press [GRAPH] to put `X/2` in `Y1`. Press [2nd] [2] on the graph screen, type [x-VAR], and press [GRAPH] for `Y2`. Press [2nd] [3], type [3] [×] [x-VAR], and press [GRAPH] for `Y3`.

   Three lines through the origin, fanning anticlockwise as the slope grows:



*The slope family X/2, X, and 3*X*

2. Press [MORE] on the graph screen for the table, which puts the family side by side in columns. The `X=4` row reads `2`, `4`, `12` across `Y1`, `Y2`, `Y3`.

   Doubling and sextupling the slope doubles and sextuples every value, which is the whole content of "linear" written out in three numbers. [EXIT] returns to the plot.

3. Re-store the slots with a parabola family: `X^2/4` in `Y1`, `X^2` in `Y2`, and `4*X^2` in `Y3`. The [x²] key types `^2`, and each slot key hands its old text back to the entry line, so press [CLEAR] before typing the new member.

   The plot shows one bowl nested inside the next, and the table says why: the `X=2` row reads `1`, `4`, `16`, and the `X=5` row reads `6.25`, `25`, `100`. The coefficient scales every height, which narrows or widens the bowl without moving its vertex.

4. Re-store once more with the vertex form: `X^2` in `Y1`, `(X–4)^2` in `Y2`, and `(X–4)^2+3` in `Y3`.

   In the table, the `X=0` row reads `0`, `16`, `19` and the `X=4` row reads `16`, `0`, `3`. The second bowl is the first moved four units right, and the third is the second

lifted three units, its vertex now at (4, 3).

Three slots is the boundary to design within, and designing within it is a skill. A family of five is two plots, with one member kept across both as the anchor so you can line the pictures up. The graph format panel can also switch a stored slot off and on without erasing it (the Guidebook, chapter 4), which lets you flick single members in and out of the picture without retyping anything.

TRY IT

1. Plot the family `X+4`, `X`, `X−3`. Predict first: what do the three lines share, and where does each cross the y axis?
2. Build a family that shows what the *sign* of a does to y = ax^2, and check your reading against the table.
3. The vertex form says `(X+2)^2−5` has its vertex at (-2, -5). Confirm it with a plot and the table, then write down the slot text for a parabola with vertex (1, 7) and test it.
4. Explore a five-member family of your own across two plots, keeping one member as the anchor. Which member did you choose to keep, and why does the choice matter?

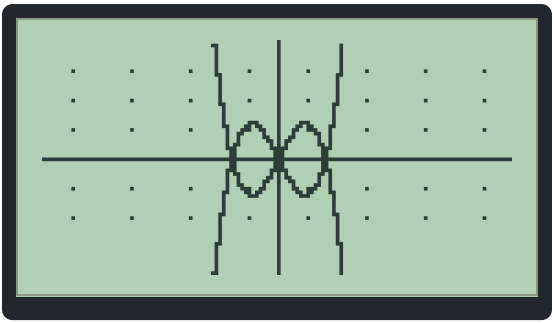
1.3 Symmetry and transformations

A function is even when f(-x) = f(x), mirror-symmetric about the y axis, and odd when f(-x) = -f(x), unchanged by a half-turn about the origin.

The definitions compare three expressions, and three slots let you plot all three at once: the function, its reflection in the y axis, and its reflection in the x axis. Whichever pair of curves coincides names the symmetry, and the machine will not tell you which pair to look at.

1. Store the test trio for f(x) = x^3 - 4x. Put `X^3−4∗X` in `Y1`. In `Y2`, type the y-axis reflection f(-x) as `(−X)^3−4∗(−X)`, using the (-) key for the minus signs.

In Y3, type the x-axis reflection -f(x) as `-(X^3-4*X)`. Plot all three:



Three stored slots, two visible curves: the odd test

2. Three equations are stored and the screen shows two curves. Y2 and Y3 land on exactly the same pixels.

   The table confirms it digit for digit. The X=1 row reads −3, 3, 3 and the X=3 row reads 15, −15, −15: f(-x) and -f(x) agree everywhere, so the cubic is odd.

   For an even function it is the Y1 and Y2 columns that agree instead, and for a function that is neither, no two columns agree anywhere except by accident. Learn to read which pair matched rather than just noticing that something did.

3. Transformations move a curve without changing its shape, and the simplest curve to watch is the absolute-value kink.

   The custom menu comes preloaded with ABS on its first slot, so **CUSTOM** **F1** types ABS(. Store ABS(X) in Y1, ABS(X+5) in Y2, and ABS(X)−6 in Y3, and plot: one V at the origin, one moved five units left, one moved six units down.

   The table carries the same story, 0, 5, −6 at X=0 and 5, 10, −1 at X=5. Adding inside the brackets slides the graph horizontally, opposite to the sign, and adding outside slides it vertically with the sign.

   That opposite-to-the-sign business catches everybody. It is worth spending a minute on rather than memorising: ABS(X+5) is 0 when X is -5, so the kink has to be at -5, so the graph moved *left*.

TRY IT

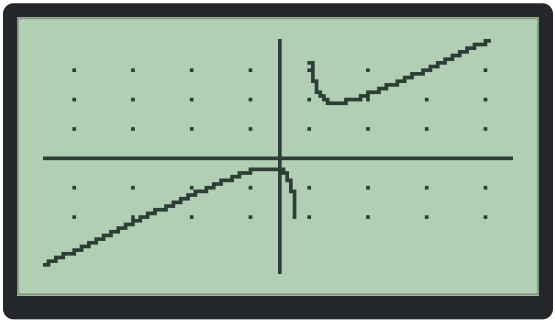
1. Run the three-slot symmetry test on `X^2−6` and on `X^3+1`. One is even and one is neither. Predict which is which before you plot, then say how each verdict shows up on the screen and in the table.
2. Predict what `2∗ABS(X)` and `ABS(2∗X)` look like, then plot both with `ABS(X)` as the anchor. Why do two different transformations give the same picture here, and for which functions would they differ?
3. Invent a function that is neither even nor odd but whose plot looks symmetric in the standard window, and use the table to expose the difference.
4. Can a function be both even and odd? Argue it from the definitions before you go near the machine, then find the one that is and check it with the three-slot test.

1.4 Rational functions and the lines they approach

Every function so far has been defined everywhere. Divide one polynomial by another and that stops being true, and what happens near the places it stops is the whole subject.

The specimen is (x^2 + 1)/(x - 1). It is undefined at x = 1, and it is worth knowing what it does far away as well as near that point.

1. Press [CLEAR] and type [(] [x-VAR] [x²] [+] [1] [)] [÷] [(] [x-VAR] [-] [1] [)] so the entry line reads `(X^2+1)/(X−1)`. Press [GRAPH] and let it finish:



The rational function with its vertical asymptote at X=1

Two separate branches, and between them a gap where the curve rushes off the top and comes back from the bottom. That gap is at x = 1 and the

near-vertical strokes either side of it are the plotter joining samples that are very far apart, exactly as in Chapter 4's oscillating sine.

The machine is not drawing a vertical line there. It is drawing a very steep one, because it joined a sample high above the screen to one far below it.

2. The table is the honest instrument again. Press **MORE**:

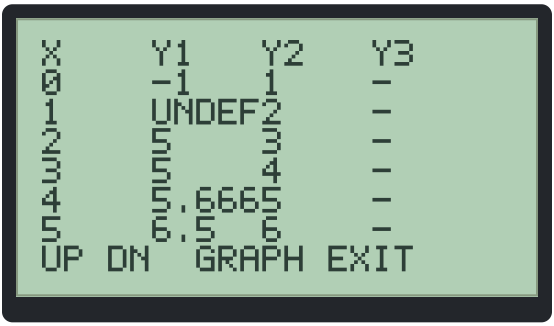
   The `X=1` row reads `UNDEF`. Compare Chapter 4's `SIN(X)/X`, which also read `UNDEF` at its bad point. These are not the same kind of bad point at all. There, the function was heading somewhere definite and simply had no value at the destination. Here it is not heading anywhere: it runs away to plus infinity from one side and minus infinity from the other.

   A table cell reading `UNDEF` tells you the machine could not compute a value. It does not tell you which of those two situations you are in. Only the neighbouring rows do.

3. Now the interesting part, which is what happens far from the trouble.

   Do the division on paper: (x^2 + 1)/(x - 1) is x + 1 with a remainder of 2 over (x - 1). So the function is a straight line plus something that shrinks as x grows, which means that far out, the curve should look like the line y = x + 1.

   Test it. Press **EXIT**, press **2nd** **2** for slot `Y2`, type **x-VAR** **+** **1**, and press **GRAPH**. Press **MORE** for the table:



The rational function converging on its slant asymptote

   Reading `Y1` against `Y2`: at `X=0`, `-1` against `1`. At `X=1`, `UNDEF` against `2`. At `X=2`, `5` against `3`. At `X=3`, `5` against `4`. At `X=4`, `5.666` against `5`. At `X=5`, `6.5` against `6`.

   The gaps are 2, 1, 0.666, 0.5, which is 2 over (x - 1) exactly, as the division promised. The two curves are converging and the table is showing you the

rate.

4. Put a number on it. How far out must you go before the curve and the line differ by less than a thousandth?

   Work it out first: the gap is 2/(x - 1), so you want x - 1 greater than 2000, so x greater than 2001. Write that down.

   Press **EXIT** twice, press **CLEAR**, and spell `EVAL(2001): = 2002.001`.

   The line at 2001 is 2002. The curve is 2002.001. A thousandth, on the nose, exactly where the algebra said it would be.

   That is a satisfying thing to have done, and it is worth noticing why it worked: you predicted a number from the structure of the formula, and the machine confirmed it. That is the direction this book wants you working in. The other direction, poking around until something looks interesting, is much less useful and very much slower.

5. One more, for contrast. Press **CLEAR**, type `(X^2+1)/(X^2-1)`, and press **GRAPH**. Let it finish, then press **MORE**.

   This one has two vertical asymptotes rather than one, at 1 and -1, and far out it approaches a horizontal line rather than a slanted one, because the top and bottom now grow at the same rate. Work out which horizontal line before you read the table.

TRY IT

1. Divide out `(X^2-4)/(X-2)` on paper before plotting it. What does the table read at `X=2`, and why is this a completely different situation from step 2's?
2. Find the slant asymptote of `(2*X^2-X+3)/(X+1)` by division, put both in slots, and check the gap at `X=10` against your formula for the remainder.
3. `(X^2+1)/(X-1)` never touches its slant asymptote. Prove that from the remainder term, then find a rational function that *does* cross its own asymptote, and plot it.
4. Work out how far out you must go before `(X^2+1)/(X^2-1)` differs from its horizontal asymptote by less than a thousandth. Predict, then check with `EVAL(`.

1.5 Exponential and logarithmic functions

Exponential growth multiplies: each unit step in x scales y by the same factor. The natural versions are `EXP(`, typed with **2nd** **LN**, and its inverse `LN(`, on the **LN**

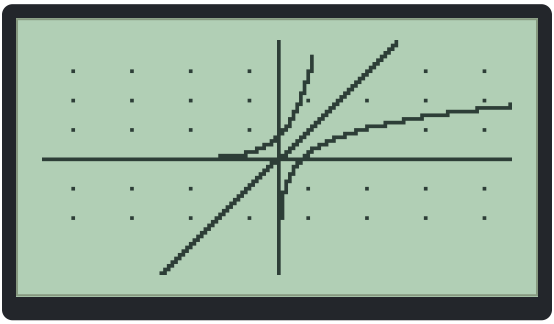
key. The Guidebook, chapter 3 covers them as functions; here they earn their keep as graphs.

1. Store the growth-and-decay family: `EXP(X)` in `Y1`, `EXP(−X)` in `Y2`, and `EXP(X/2)` in `Y3`.

   Exponentials are heavier work per sample than polynomials, so each plot draws noticeably more slowly. Let it finish.

   In the table, the `X=0` row reads `1`, `1`, `1`, every member starting from the same value, and the `X=2` row reads `7.389`, `0.135`, `2.718` in the five-character cells. Growth, decay, and slower growth, told apart entirely by what multiplies each step.

2. Now the inverse pair, and the window matters more here than anywhere else in the chapter. Store `EXP(X)` in `Y1`, `LN(X)` in `Y2`, and `X` in `Y3`, then press **2nd** **-** on the graph screen for the square window, which makes one unit the same length on both axes:



EXP(X) and LN(X) mirrored in the line X

   The two curves are reflections of one another in the third slot's line y = x, which is the picture of "inverse" itself. The square window is what keeps the mirror at forty-five degrees. In any other window the reflection is still true and no longer looks true, which is exactly the sort of thing section 1.1 warned about.

Two routes to the same power

3. Compound growth at six per cent per year is the function 1.06 to the power x. Type it directly and it simply works: `1.06^2` answers `= 1.1236`, `1.06^2.5` answers `= 1.1568170026417`, and a slot holding `1.06^X` plots the whole curve.

So far so dull. What is worth knowing is that two quite different machines live behind that one key, and which of them answered you decides how much of the answer to believe.

A whole-number exponent is done by repeated squaring. Squaring and multiplying are exact, so the answer is exact: `1.06^2` really is `1.1236` and nothing has been rounded on the way. Any other real exponent, on a positive base, is done as e to the power x ln b: a logarithm and an exponential, each rounded to fourteen digits.

Three cases have no answer to give, and say so instead of guessing. A negative base with a fractional exponent has no real value, so `(-2)^0.5` answers `DOMAIN ERROR`. Zero to a negative power is a division by zero, so `0^(-1)` answers `DIVIDE BY ZERO`. Anything too big for the numeric range answers `NUMERIC OVERFLOW`.

4. You can see the join between the two routes, and you should, because it is the clearest look at rounding you will get in this chapter.

   That identity, b to the power x is e to the power x ln b, is exactly what the key does for the general case. So type it yourself and compare. Press **CLEAR** and type `EXP(2*LN(1.06)): = 1.1236000000004`. The power key answered `= 1.1236`.

   Same number in mathematics. Different number here, from the eleventh digit on. One is two exact multiplications; the other is a logarithm and an exponential, each rounded, and the roundings do not cancel.

   Try `1.06^9` against `EXP(9*LN(1.06)): = 1.6894789590026` against `= 1.6894789590072`. The gap has grown, because nine steps of rounding is more than two.

   Neither is a bug and neither is the "real" answer. They are two calculations of the same quantity, and the exact one is available only when the exponent is a whole number. Knowing which route your expression took is the whole of the skill here.

5. As a taste of where this leads, 500 invested at six per cent for eight years is `500*1.06^8`, which answers `= 796.9240372654`, while `500*EXP(8*LN(1.06))` answers `= 796.92403726725`. Not quite two hundredths of a penny apart, on five hundred pounds, from nothing but the route taken. Chapter 2 takes the mathematics of money much further, and it takes this identity with it.

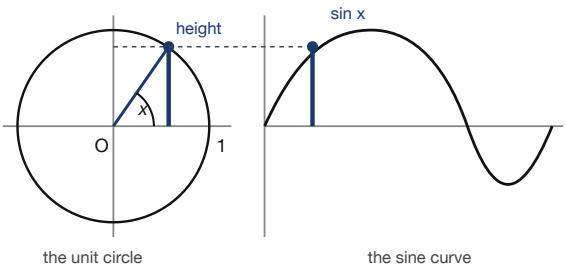
TRY IT

1. Add `LN(X)` to a plot of `EXP(X)` *without* the square window. What happens to the mirror symmetry, and why does the window carry the blame rather than the mathematics?
2. Plot `EXP(X*LN(2))`, the base-two exponential, and use trace to find where it passes 8. Predict the answer first, then check it with `LN(8)/LN(2)` on the home screen.
3. A quantity halves every unit of time. Write its slot text with `EXP(` and `LN(`, plot it, and read from the table how much is left after five units.
4. Work out `1.06^9` two ways, with the power key and with the identity, and compare all fourteen digits. Which do you trust more, and why?
5. Predict which route each of `1.06^-3`, `1.06^10` and `1.06^0.5` takes before you press it, and therefore which of the three is exact. Then check by computing each one again through `EXP(` and `LN(` and seeing which answers move.

Trigonometric functions 1.6

Periodic phenomena repeat, and their graphs only make sense in windows matched to the period. Nowhere does the window matter more than here, which is why section 1.1's window editor earns its keep in this section: a period is rarely a round number, and `2*PI` typed into `XMAX` is exact where four presses of a zoom key are merely close.

The angle mode matters too. Everything here runs in `RAD`, the fresh-boot default shown in the status line, until the walkthrough says otherwise.



The unit circle, with the angle marked at the centre and the height of the point above the axis carried across to become the sine curve

That picture is where sine and cosine come from, and it is worth having in front of you: the sine of an angle is the *height* of a point going round a circle of radius 1, and the cosine is how far along it is. Everything in this section follows from that, including why the values never leave -1 to 1.

1. Store `SIN(X)` in `Y1` and plot it in the standard window.

   The result is accurate and unflattering: a ripple a few pixels tall hugging the x axis. The window allows for values from -10 to 10 and the sine never leaves -1 to 1, so nine tenths of the screen is empty. This is section 1.1's badly chosen window meeting a function that deserves better.

2. Open the zoom panel with **2nd** **GRAPH**, press **MORE** for its second page, and press **F5**, the trigonometric window. The replot shows two full waves.

   Read the bounds from the home screen (**EXIT**, then **CLEAR** to empty the handed-back equation): `XMIN` answers `= -6.2831853071796` and `XMAX` answers `= 6.2831853071796`, two pi either side of the origin, with `YMIN` at `= -4` and `YMAX` at `= 4`.

   That window exists because I got tired of building it out of zooms every time. It is the one shape of window this machine will hand you ready made, and it is the one you need in nine trigonometric problems out of ten.

3. Amplitude and period are the family parameters. Keeping the trig window, store `2*SIN(X)` in `Y2` and `SIN(2*X)` in `Y3`: one curve twice as tall, one twice as frequent. The plot separates the two effects at a glance, which is exactly what three slots are for.

4. Phase is subtler, and the table settles it. Re-store the slots with `SIN(X)` in `Y1`, `SIN(X+PI/2)` in `Y2` (the π legend on **2nd** **^** types `PI`), and `COS(X)` in `Y3`.

   In the table, the `X=1` row reads `0.841`, `0.540`, `0.540`, and every later row agrees the same way: sine led by a quarter turn is cosine.

   The `X=0` row shows `0.999` beside `1`, which is the stored fourteen-digit `PI` falling a whisker short of the exact half turn. The Guidebook, chapter 3 tells that story at `SIN(PI/2)`, and Chapter 5's cardioid meets it again.

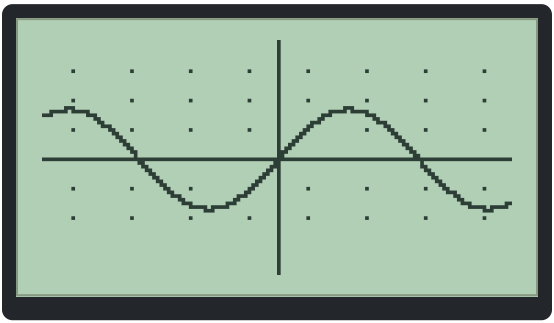
5. Angle mode belongs to the graph as much as to the home screen. Open the mode screen with **2nd** **MORE**, press **F1** for `ANGLE DEG`, press **EXIT**, and replot `SIN(X)`.

   The curve collapses onto the axis, because -10 to 10 now spans twenty degrees of a wave 360 degrees long.

The trigonometric window is no rescue: it sets the same two-pi bounds whatever the angle mode, so in `DEG` it frames barely thirteen degrees. Degree-mode trigonometry wants windows hundreds of units wide, and the quick zooms are the way there. Return to `RAD` before moving on, and see Chapter 4's warning about what happens if you forget.

6. Trigonometry earns its place modelling data. At a latitude of about fifty-five degrees north, the hours of daylight run roughly 4.3 hours above twelve in midsummer and 4.3 below in midwinter.

   With `X` counting months after the March equinox, store the departure from twelve hours as `4.3*SIN(PI*X/6)` and plot it in the standard window:



Daylight hours above and below twelve across the year

   The table reads `0`, `2.15`, `3.723`, `4.299`, `3.723`, `2.149` down its first six rows: the equinox itself, a June solstice 4.3 hours over twelve, and the symmetric slide back down.

   One period is twelve months, which is exactly what the `PI*X/6` inside the sine was chosen to say. Work out why that factor gives a year before you accept it: the sine repeats every 2 pi, so you want PI*X/6 to reach 2 pi when X reaches 12.

TRY IT

1. In the trig window, plot `SIN(X)`, `SIN(X)+2` and `SIN(X+2)` together. Predict which slot does what to the wave before checking the table.
2. In `DEG` mode, zoom out from the standard window until a full wave of `SIN(X)` fits, reading `XMAX` as you go. How wide is the first window that shows a whole period, and could you have predicted it?
3. Rebuild the daylight model for a town near the equator, where the swing is about one hour, and decide what window shows a year of it clearly. What does the standard window hide?
4. From the unit circle picture, predict the value of `COS(X)` at the four places where `SIN(X)` is 0, 1, 0 and -1. Then check all four with the table.
5. Build a model for the tide, which turns roughly every 6 hours 13 minutes. What goes inside the sine, and what window shows two days?

1.7 Inverse functions by parametric pair

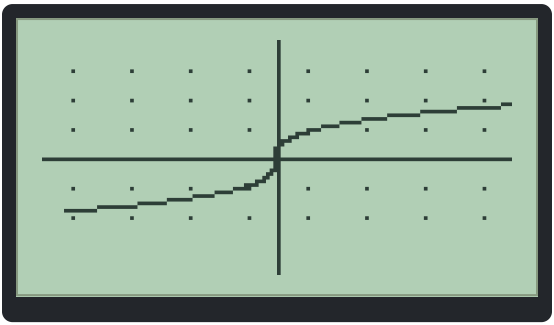
The inverse of a function swaps the roles of input and output. The graph of the inverse is the graph of the function with its coordinates exchanged, which is the same as reflecting it in the line y = x.

Function slots cannot plot a sideways curve, because a function slot computes y from x and an inverse very often has two y values for one x. The parametric mode can, because it plots any pair x(t), y(t) you give it, and swapping the pair is exactly the coordinate exchange.

1. Switch modes. On the graph screen press **2nd** **MORE**, then **MORE** until the `GRAPH MODE` page appears, then **F3** for parametric mode; **EXIT** closes the panel.

   The mode keeps one coordinate pair: slot 1 is x(t), slot 2 is y(t), slot 3 is never plotted, and **x-VAR** types the parameter, shown as `X`. The parameter sweeps from `XMIN` to `XMAX` in 128 samples (the Guidebook, chapter 6 has the full tour).
2. Draw a function as a pair first, so you can see that nothing has changed yet. Store `X` in slot 1 and `X^3/10` in slot 2, and the plot is the ordinary graph of f(t) = t^3/10, drawn point by point as (t, f(t)).
3. Now exchange the coordinates. Press **2nd** **1**, press **CLEAR**, type `X^3/10`, and press **GRAPH**; then **2nd** **2**, **CLEAR**, `X`, **GRAPH**.

The pair is now (f(t), t), and the plot is the inverse function: the cube-root-shaped curve lying on its side.



The inverse of t^3/10, drawn by the swapped pair

4. The trace readout says the same thing pointwise. Two presses of ▶ from the centre of the sweep show `X=0.0061023744094928` and `Y=0.393700787402`.

   Look at those two numbers against step 2's. The first coordinate is the old output and the second is the old input: the swap made visible one sample at a time.

   The table agrees, its `Y1` column reading `0`, `0.1`, `0.8`, `2.7`, `6.4`, `12.5` beside a `Y2` column that is simply `0` through `5`, with – down the unplotted `Y3`.

One pair is the design to work within, and it costs you something real here. The mode draws one curve at a time, so a function and its inverse are compared across two plots rather than overlaid, and the mirror line of section 1.5 cannot join them on screen.

What makes the comparison workable is that each graphing mode keeps its own equations and window. Press F1 on the `GRAPH MODE` page to return to function mode and the slots hold exactly what you left in them, so a session that left `X^3-4*X` in slot 1 finds it still there. You can flick between the function-mode picture and the parametric inverse without retyping either.

That was worth the memory it costs. Losing your work every time you changed modes would have made the modes almost unusable together, and half the interesting things in this book happen when two modes are used on the same problem.

TRY IT

1. Plot the inverse of f(t) = 2t - 6 by parametric pair, and read from the trace where the inverse crosses the x axis. Predict the number first: where did it live on the original line?
2. Draw the inverse of `EXP(X)` as a pair and compare it, across a mode switch, with the `LN(X)` plot of section 1.5. Why must the two pictures agree?
3. The pair `X^2` and `X` draws the inverse of the squaring function, which is not a function. Plot it, trace it, and explain what the sweep from `XMIN` to `XMAX` did that a function slot never could.
4. Find a function that is its own inverse, so the swapped pair draws the same picture as the unswapped one. There is more than one; find two that are not the same shape.

CHAPTER

2

Explorations in Business Mathematics

Business mathematics runs on a few small machines. Linear systems that turn receipts into price lists. Inequalities that turn scarce resources into best plans. Exponentials that turn interest rates into balances. Transition tables that turn this week's customers into next year's market shares.

Free85 has a tool for each, and this chapter visits them in turn: the simultaneous editor, the graph screen, the matrix editor's row operations, the solver workspace, and matrix multiplication.

Chapter 1 ended section 1.5 with money growing at six per cent through `EXP(` and `LN(`, because the power key would not take a fractional exponent. This chapter picks that thread up and follows it a long way.

Prices from receipts 2.1

A receipt is a linear equation.

Two coffees and five pastries for 7.90 says 2c + 5p = 7.9, and one more receipt with different quantities pins both prices down. Recovering a price list from receipts is solving a linear system, and Free85 keeps a dedicated editor for exactly that.

1. Press **2nd** **STAT** (the `SIMULT` legend) to open the simultaneous editor, described in full in the Guidebook, chapter 14. A fresh machine shows `SIZE 2`, two equations in two unknowns.

   Each row takes its coefficients left to right and then its right-hand side. The morning's two receipts are 2 coffees and 5 pastries for 7.90, then 4 coffees and 5 pastries for 12.30.

So type 2 ENTER 5 ENTER 7 . 9 ENTER for the first, then 4 ENTER 5 ENTER 1 2 . 3 ENTER for the second.

2. Press F1, the SOLVE key. The result screen answers UNIQUE SOLUTION with X 2.2 and Y 0.7: coffee is 2.20 and a pastry 0.70.

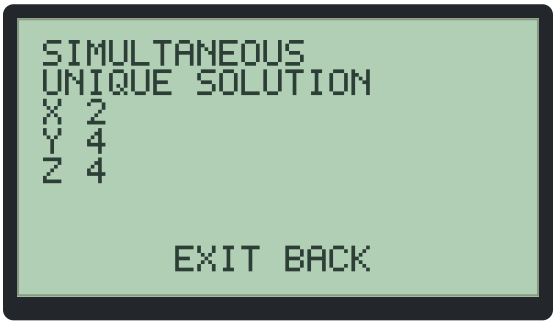
   Check it in your head before you believe it. The second receipt doubled the coffees and kept the pastries, so the difference between the two receipts is two coffees for 4.40. That is what made this system easy, and it is worth noticing which pairs of receipts are easy and which are not, because section 2.2 is entirely about the difference.

3. Three unknowns work the same way. A tea merchant blends three leaves costing 12, 9 and 6 per kilogram into a 10 kilogram batch worth 84, with twice as much of the cheapest leaf as the dearest.

   The first two conditions are equations already. The proportion becomes one by writing z = 2x as -2x + 0y + z = 0, and writing that zero explicitly is the part people forget.

   Press EXIT to leave the result screen, then 2nd STAT to reopen the editor, and press F3 for 3X3. Enter the three rows: 1, 1, 1, 10, then 12, 9, 6, 84, then -2, 0, 1, 0, using (-) for the minus sign.

   SOLVE answers X 2, Y 4 and Z 4: two kilograms of the dear leaf and four of each of the others.



The tea blend solved in three unknowns

4. Not every pair of receipts is bookkeeping in good order, and the editor will tell you which kind of trouble you are in.

   Press EXIT, reopen the editor, press F2 for 2X2, and enter 2, 1, 5, then 4, 2, 9: two coffees and a bun for 5.00, then exactly double the order for 9.00.

`SOLVE` answers `NO SOLUTION`. No price list can explain those two receipts, because doubling an order must double its price, and the machine has spotted the discrepancy.

5. Now repair the till roll. Reopen the editor and step to the last cell: the cells keep their values, and five presses of ▼ bring the selection to row 2's right-hand side, still holding `9`.

   (The `CELL` line reads row then column, so you can check where you are rather than counting presses: row 2's right-hand side is `CELL 2 3`. Until firmware 2.21 that second figure always read `3` whatever column you were in, which is why earlier editions of this book told you to keep count.)

   Type **1** **0** **ENTER**, and `SOLVE` now answers `UNDERDETERMINED`.

   With the second receipt exactly double the first, every price list that explains one explains both. A single figure on the till roll separated impossible from unhelpfully many, which is a fair summary of most accounting disputes.

The editor reaches `4X4` through **F4** or the **+** key. That ceiling, and the way the two degenerate verdicts mirror the matrix editor's `SINGULAR MATRIX` guard, are in the Guidebook, chapter 14.

TRY IT

1. A juice stand's till roll shows 3 juices and 2 flapjacks for 12.30, then 2 juices and 3 flapjacks for 10.70. Price both items. Then check your answer by working out what each receipt would cost.
2. The tea merchant scales up: a 12 kilogram batch worth 105, still with twice as much of the cheapest leaf as the dearest. Predict whether the proportions stay the same before you re-solve.
3. Invent two receipts that no price list can explain, and two that infinitely many price lists explain, and check the machine names each verdict correctly.
4. What would a *third* café receipt have to look like to contradict the first two of step 1? Write one down, add a row, and see what the editor says.

When the answer will not stay still 2.2

Section 2.1 solved two systems and both answers were solid. This section is about the systems where they are not, and it is the most useful half hour in the chapter if you ever intend to trust a number somebody else computed.

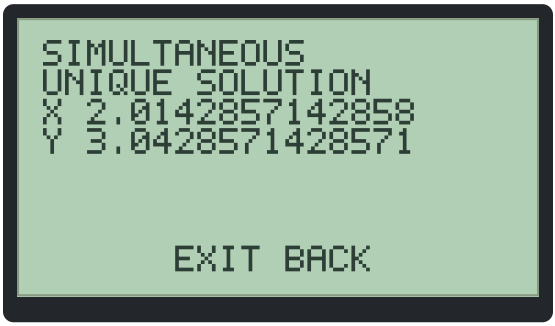
Real data carries small errors. A quantity is misread, a price is rounded, a measurement is taken to the nearest unit. The question is what those small errors do to the answer, and the answer is that it depends entirely on the system, by factors of hundreds.

1. Start with a well-behaved pair: x + 2y = 8 and 3x - y = 3, which cross at (2, 3).

   Press **2nd** **STAT**, press **F2** for 2X2 if you need it, and enter 1, 2, 8, then 3, -1, 3. SOLVE answers X 2 and Y 3.

2. Now spoil the data very slightly. Press **EXIT**, reopen the editor, step to row 1's right-hand side, and change the 8 to 8.1. That is a change of a bit over one per cent, the sort of thing a rounded invoice would do.

   SOLVE answers X 2.0142857142858 and Y 3.0428571428571:



A well-behaved system shrugging off a nudge in the data

   A one and a quarter per cent change in the data moved the answers by under one per cent and one and a half per cent. That is what you would hope for, and it is easy to assume it always happens.

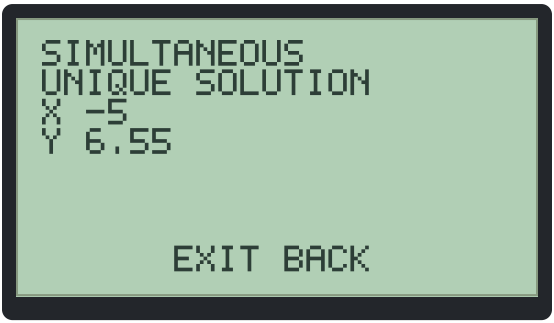
3. It does not. Try a pair of receipts that nearly say the same thing: x + 2y = 8 and 1.01x + 2y = 8.05. Two orders that differ by one per cent in one item, which is exactly the sort of pair a real till roll produces when two customers buy nearly the same thing.

   Press **EXIT**, reopen the editor, and enter 1, 2, 8, then 1.01, 2, 8.05. SOLVE answers X 5 and Y 1.5.

   A perfectly definite answer, delivered without hesitation.

4. Now make the same one-per-cent nudge as step 2: change the 8 to 8.1.

`SOLVE` answers `X −5` and `Y 6.55`:



The same nudge sending a near-parallel system across the axis

Read that twice. The first unknown has gone from 5 to minus 5. A change of one part in eighty in one number has moved the answer by two hundred per cent and flipped its sign, and if x is a price you have just been told that coffee costs minus five pounds.

The machine gave no warning on either run, and it was right not to. Both answers are correct. It solved exactly the system you typed, and the system you typed was a bad question.

5. Why the difference? Look at the two lines. In step 1 they cross decisively, one going up and one going down. In step 3 they are very nearly parallel, so their crossing point is a long shallow wedge, and moving one line a hair slides the crossing an enormous distance along it.

   That is the whole idea, and it has a name: the system is ill-conditioned. Chapter 6 measures it with a single number, `COND`, and you should read section 6.3 before you next trust a solve. But it is worth meeting here, in receipts, because this is where you will actually hit it, and because the fix is often commercial rather than mathematical: get a pair of receipts that are genuinely different.

TRY IT

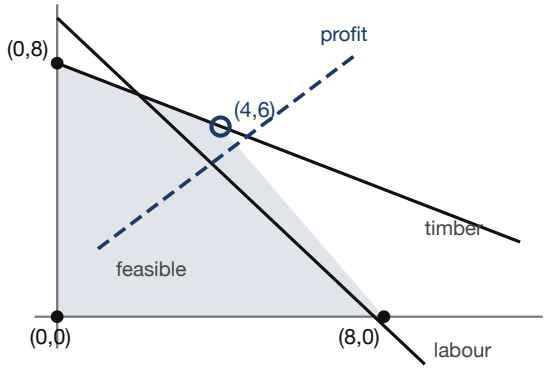
1. Make the two lines of step 3 even closer, 1.001 instead of 1.01, and repeat the nudge. Predict roughly how much worse it gets before you run it.
2. Go the other way: find a pair of receipts whose lines are perpendicular, and see how little the nudge moves the answer. What does that suggest about how to plan a set of measurements?
3. In step 3's system, work out on paper what happens when the coefficient 1.01 becomes exactly 1. Which of section 2.1's two verdicts do you get, and why is that the honest end of this road?
4. The tea blend of section 2.1 has three equations. Nudge the batch value 84 by one per cent and see how far the three answers move. Is it well-behaved or not?

2.3 The best plan on a graph

A joinery makes bookcases and benches. A bookcase uses one sheet of timber and three workshop hours; a bench uses two sheets and two hours; the week holds 16 sheets and 24 hours.

With x bookcases and y benches, the constraints are $x + 2y$ at most 16 and $3x + 2y$ at most 24, and the profit to maximise is $30x + 40y$.

The feasible plans fill a region of the plane, the best plan sits at a corner of it, and the graph screen can find every corner. Why the best plan must be at a corner is worth convincing yourself of before you start, and step 5 shows you the picture that does it.



The feasible region with its four corners, and the profit line sliding out until it touches the last one

1. Solve each constraint's boundary for y and store the lines. Type (1 6 - x-VAR) ÷ 2 and press GRAPH to put `(16–X)/2`, the timber line, in `Y1`. Press 2nd 2 on the graph screen, type `(24–3*X)/2`, the labour line, and press GRAPH.

   In the standard window the feasible region is the four-sided patch above both axes and below both lines.

2. One corner is where labour runs out along the bottom edge. Press F1, the root key: it reads the active slot, which is the labour line stored last, and answers `= 8` with `R=0`. The corner is (8, 0), eight bookcases and no benches.

3. Two more corners hide in the table. Press GRAPH to redraw the plot, then MORE for the table.

   The `X=0` row reads `8` and `12` under `Y1` and `Y2`: the two intercepts, of which only the lower is feasible, giving the corner (0, 8). And the `X=4` row reads `6` and `6`, the two lines agreeing, which is the remaining corner caught red-handed. EXIT returns to the plot.

4. Ask for that crossing properly. The plot redraws itself after the table, so let it finish before touching the arrows, because presses during a redraw are lost.

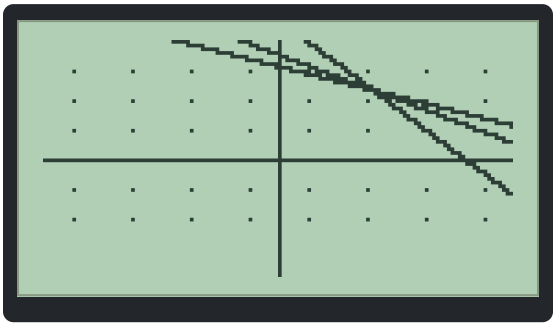
   Press ▶ twenty-five times, taking the trace to `X=4.015748031496` with `Y=5.976377952756` on the labour line, and press 2nd F1, the intersection search. It answers `= 3.9999999999999` with `R=5E–13`, the corner's x under a grain of dust.

Its height is the timber line there: press CLEAR, type `(16-4)/2`, and ENTER answers `= 6`. The crossing corner is (4, 6).

5. The profit has lines of its own, and this is the picture worth having.

   Every plan earning 240 sits on 30x + 40y = 240, which solves to `(240-30*X)/40`. Press CLEAR, then GRAPH to return to the plot, then 2nd 3 , type the line, and press GRAPH: it cuts straight through the middle of the region, so plenty of feasible plans earn 240.

   Now press 2nd 3, CLEAR, type `(360-30*X)/40`, and press GRAPH:



The 360 profit line resting on the corner

   The 360 line settles onto the region's outermost corner, touching it at (4, 6) alone.

   That is the whole of linear programming in one picture. Slide a line of constant profit outwards until it is about to leave the region, and the last thing it touches is the best plan.

   Because the region is a polygon, that last thing is a corner. The one exception is a profit line lying exactly along an edge, in which case the whole edge ties.

6. The corners settle it numerically. Press EXIT for the home screen and CLEAR, then evaluate the profit at each corner with stored letters: 4 STO▶ ALPHA A ENTER, CLEAR, 6 STO▶ ALPHA B ENTER, CLEAR, then `30*A+40*B` and ENTER: `= 360`.

   Store 0 and 8 the same way and replay the profit entry with three presses of 2nd ENTER, the entry recall of the Guidebook, chapter 1, then ENTER: `= 320`. Store 8 and 0 and recall again: `= 240`.

   The best week is four bookcases and six benches for 360.

What one more sheet of timber is worth

The plan is settled. Now ask the question a business actually asks: if you could buy more timber, how much should you be willing to pay for it?

7. Raise the timber from 16 sheets to 18 and find the new crossing corner. Press **2nd** **STAT** for the simultaneous editor and enter the two boundaries as equations: 1, 2, 18, then 3, 2, 24.

   `SOLVE` answers `X 3` and `Y 7.5`.

   Press **EXIT**, **CLEAR**, and evaluate the profit there: `30*3+40*7.5` answers `= 390`.

8. Do it once more at 20 sheets. The editor gives `X 2` and `Y 9`, and `30*2+40*9` answers `= 420`.

Timber	Best plan	Profit
16	(4, 6)	360
18	(3, 7.5)	390
20	(2, 9)	420

   Thirty pounds of extra profit for every two extra sheets, twice running. So a sheet of timber is worth exactly 15 to this business, and if you can buy one for less than 15 you should.

   That number has a name, the shadow price, and it is the single most useful thing linear programming produces. It is not the price you paid for the timber. It is what the *constraint* is costing you, which is a different quantity entirely and usually the one worth knowing.

9. It does not go on forever, and finding where it stops is the exercise. As timber increases, the best corner slides up the labour line towards (0, 12). Once it arrives, timber has stopped being the binding constraint and buying more buys nothing. Work out from the two lines where that happens before you test it.

The intersection key reads slots `Y1` and `Y2` only, so the pair of lines you ask about must live in those two slots, and the third slot is where the profit line visits without disturbing the question. Three slots also bound the constraints a single plot can carry: a fourth constraint means swapping a line in and out, or testing its corner candidates on the home screen as step 6 does.

TRY IT

1. Prices shift: profit moves to 60 per bookcase and 30 per bench. Predict which corner wins before you compute anything, from the slope of the new profit line, then evaluate the corners and check.
2. A delivery contract caps benches at five a week. Put 5 in Y3, find the new corner where the cap meets the timber line, and notice that the best corner no longer has whole coordinates. What whole-number plans deserve a check, and why is that a harder problem than it looks?
3. Work out the shadow price of an extra workshop *hour* the way steps 7 and 8 did for timber. Which of the two resources should the joinery buy more of first?
4. Find the timber level at which the shadow price drops to zero, as step 9 describes, and confirm it by solving the system there.
5. Design a third resource of your own whose boundary passes exactly through (4, 6), and show that adding it leaves the best plan unchanged. What is its shadow price?

2.4 Elimination as bookkeeping

Section 2.1's SOLVE answers in one press, which is convenient and opaque. The matrix editor's row operations let you watch the same arithmetic done slowly, receipt by receipt, the way a clerk would cross-check a ledger.

A picture framer sells prints and frames: two prints and a frame for 110, one print and three frames for 130. As a tableau, coefficients beside takings, that is a 2 by 3 matrix, and it fits the matrix editor exactly.

1. Press [2nd] [7] (the MATRX legend) for the matrix editor of the Guidebook, chapter 13, then [x-VAR] [+] to grow the columns: SIZE 2X3.

   Type the tableau row by row, [2] [ENTER] [1] [ENTER] [1] [1] [0] [ENTER] [1] [ENTER] [3] [ENTER] [1] [3] [0] [ENTER], and the entry wraps back to CELL 1 1, leaving row 1 selected.
2. Press [MORE] [MORE] for the row-operation page REF SWP RADD RMUL AUG. Five labels in twenty-one characters, which is why the second one is SWP and not SWAP: it is the same swap, one letter shorter, so the fifth name fits on the screen instead of running off it.

   Elimination is easiest when the pivot is a 1, and the second receipt starts with one, so press [F2], SWP. The banner's register letter changes to R, where every

result lands, and stepping through with [▶] reads `1`, `3`, `130`, `2`, `1`, `110`: the receipts in the friendlier order.

3. The row operations read register `A`, so the result must be carried forward by hand. The three registers share a single selection cursor, so where you step in one view is where you stand in the next.

   The fifth [▶] left the selection at `CELL 2 3`; one more wraps it home to `CELL 1 1`. Press [ALPHA] twice, cycling the view from `R` through `B` back to `A`, and retype the six values in their new order: 1, 3, 130, 2, 1, 110.

   That copying is the price of watching each move separately. The editor keeps the original safe in `R` until the next operation overwrites it, and I will not pretend the arrangement is elegant: results landing in a read-only register is what makes every operation inspectable, and carrying them forward by hand is what it costs.

4. Now the clerk's move: subtract twice receipt 1 from receipt 2.

   The scale rides in `B`'s top-left cell, so press [ALPHA] to show `B`, type [(-)] [2] [ENTER], and press [ALPHA] again to return to `A`, where the trip has stepped the selection to `CELL 1 2`. Press [▼] three times to reach `CELL 2 2`, any cell of row 2, and press [F3], `RADD`: the selected row gains the scale times the following row, wrapping to row 1.

   Stepping through `R` reads `1`, `3`, `130`, `0`, `−5`, `−150`. The prints have cancelled out of the second row, which now says that minus five frames cost minus 150.

5. Tidy the pivot. Wrap the selection home, press [ALPHA] twice, and retype 1, 3, 130, 0, -5, -150 into `A`. Store the scale -0.2, the reciprocal of -5, in `B`'s top-left cell as before, return to `A`, press [▼] three times for row 2, and press [F4], `RMUL`: row 2 of `R` becomes `0`, `1`, `30`. One frame costs 30.

6. One move remains: clear the frames out of receipt 1. Carry the result forward once more, store the scale -3 in `B`, and return to `A`, where the selection already

sits in row 1 at `CELL 1 2`. Press **F3**, `RADD`, and `R` reads `1, 0, 40, 0, 1, 30`:

The finished tableau naming the prices

A print is 40 and a frame is 30, each row of the finished tableau naming one price.

7. The machine will also do the whole dance in one key. Retype the original tableau, 2, 1, 110, 1, 3, 130, into `A`, press **EXIT** and **2nd** **7** to bring back the first soft-key page, and press **F5**, `RREF`: the same `1, 0, 40, 0, 1, 30` in one step.

   That is the reduced row-echelon form of the Guidebook, chapter 13, doing steps 2 through 6 unwatched. Use it once you believe the steps and not before.

The tableau register holds three rows and up to six columns, so a two-unknown system's 2 by 3 tableau fits, a three-unknown system's 3 by 4 fits, and there is room left over. Press **x-VAR** in the editor and the **+** and **-** keys resize columns instead of rows.

That room is worth knowing about, because it is exactly enough for the simplex method, which is what a real linear programme uses instead of section 2.3's picture. The joinery's two-product problem needs a tableau of three rows and six columns once the slack variables are in, and three by six is precisely what the workspace holds.

This book does not teach simplex, and that is now a choice about what belongs in an introduction rather than a fact about the machine. Earlier editions said there was nowhere to put the tableau. There is: I wrote that sentence, read it back, and built the workspace to fit it.

What has not changed is that the square-only keys stay square. `DET`, `INV`, `ID`, `SOLVE`, `LU` and the eigensystem still want a 3 by 3, because that is what they

mean. Three unknowns and their takings still belong to the simultaneous editor of section 2.1, or to SOLVE, which reads the coefficients from A and the right-hand sides from B's first column.

TRY IT

1. Rework the café receipts of section 2.1 as a tableau: 2, 5, 7.9 and 4, 5, 12.3. No swap is needed and four row operations reach the identity. Which scales do you store along the way? Write them down first.
2. The AUG key on the same page appends B's columns to A. Build the framer's tableau a second way: coefficients in a 2 by 2 A, takings in a 2 by 1 B, then AUG. Where does the result land, and what must you do before row operations can touch it?
3. Swap the rows of the finished tableau, 0, 1, 30 above 1, 0, 40, and run RREF on it. Predict the answer before you press the key.
4. Count the row operations RREF must be doing internally for the framer's tableau, from your work in steps 2 to 6. Then build the 3 by 4 tableau of a three-unknown system, run RREF on it, and count them again. How does the work grow with the size?

The mathematics of money 2.5

Section 1.5 left money compounding through EXP(and LN(. The ^ key will now take a fractional exponent directly, and for a whole number of years it is the exact route and the better one, but the identity b to the x equals e to the x ln b is what the solver below needs: an equation it can rearrange, not an operator it must invert.

The solver workspace turns that identity from a formula you evaluate into an equation you can interrogate from any side. The same line answers what rate, how long and how much, depending only on which letter you name as the unknown, and that is a genuinely different way of working.

1. First the reconciliation. On the home screen, `500*EXP(8*LN(1.06))` (with EXP(on 2nd LN and LN(on LN) answers = 796.92403726725, exactly the figure section 1.5 promised for 500 invested at six per cent for eight years.
2. Now the general equation. The solver hunts for a zero, so write the savings story as a difference: growth minus balance.

Store the knowns first: press CLEAR, then 8 STO▶ ALPHA Y ENTER for the years, then CLEAR and `900->Z` for a target balance.

Press CLEAR once more, type `500*EXP(Y*LN(1+X))-Z`, and press 2nd GRAPH.

The `SOLVER` workspace of the Guidebook, chapter 14 opens with the equation stored, and `VAR X` names the unknown, which is the rate. The `F=` line clips at the screen's right edge; the tail is kept.

3. What rate turns 500 into 900 in eight years? Rates live between zero and one, so fence the search: press F5, the `>` key, three times, and store `0` on the `LOWER` page and `1` on `UPPER`.

   Press F1, `SOLV`, and after a moment the field area answers a `ROOT` of `0.07623983640223` with `RES 5.327E-7`: a little over 7.6 per cent, with the residual reporting how nearly the equation balances there.

4. How long to double money at six per cent? Press EXIT, CLEAR, store `.06->X` and `1000->Z` with a CLEAR between and after the stores, and press 2nd GRAPH: an empty entry line keeps the stored equation, so the workspace reopens with everything kept.

   Press F3, `VAR`, once, turning `VAR X` into `VAR Y`, page to the bounds, and store `0` and `50`. `SOLV` answers a `ROOT` of `11.895661056043`.

   Money at six per cent doubles in just under twelve years, whatever the starting sum. Notice that the 500 never entered the question: doubling is a ratio and the starting amount cancels. Check that on paper.

5. And the balance itself? Press EXIT, CLEAR, store `8->Y` again, press CLEAR, and reopen the workspace with 2nd GRAPH; press `VAR` once more for `VAR Z`, set the bounds to `0` and `1000`, and `SOLV` answers a `ROOT` of `796.9240378587` with `RES -5.9145E-7`.

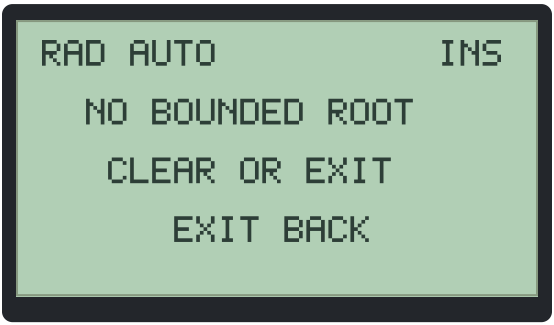
   Compare step 1: the workspace bisects until the residual passes the `1E-6` tolerance, so its root carries the home screen's exact figure to within a millionth, and the `RES` line names that gap. Bisection gives you an answer and an error bar to go with it, which is more than most methods manage.

6. Loans are the same equation read in reverse: the debt grows while payments shrink it.

   For 10000 borrowed at one per cent a month with payment A over B months, press EXIT, CLEAR, store `24->B`, press CLEAR, type

`10000*EXP(B*LN(1.01))-A*(EXP(B*LN(1.01))-1)/.01`, and press **2nd** **GRAPH**.

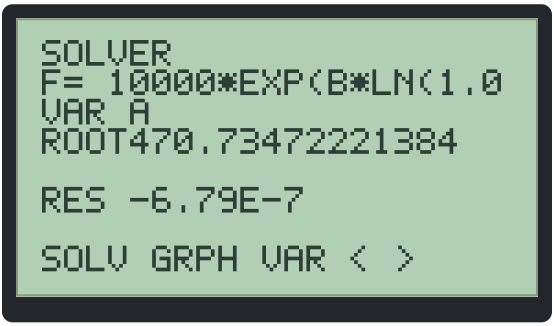
Press `VAR` once for `VAR A`, and try the bounds a hopeful borrower would: `0` and `300`. `SOLV` stops at the `NO BOUNDED ROOT` notice:



No payment under 300 clears the loan

No payment up to 300 clears this loan in 24 months. That screen is an answer, not a failure, and it is the answer to a question worth asking.

7. Press **CLEAR** (the notice dismisses to the home screen with the workspace kept), reopen with **2nd** **GRAPH**, page to `UPPER`, and raise it to `1000`. `SOLV` answers a `ROOT` of `470.73472221384`:



The loan payment found by the solver

The true monthly payment.

8. If 300 a month is all there is, ask for the term instead. Press **EXIT**, **CLEAR**, store `300->A`, press **CLEAR**, and reopen with **2nd** **GRAPH**; press `VAR` once for `VAR B`, page to `UPPER`, store `100`, and `SOLV` answers a `ROOT` of `40.748907154197`.

Nearly 41 months, which is the price of the smaller payment: seventeen extra months of interest.

9. One trap deserves its own look, because it is the one that ruins people.

   Press **EXIT**, **CLEAR**, store `100->A`, which is exactly the monthly interest on 10000, press **CLEAR**, reopen, and `SOLV: NO BOUNDED ROOT` again.

   A payment that only covers the interest never ends the loan. The balance is the same every month forever, so no term between the bounds can make the equation balance, and the machine is telling you something true and important rather than failing.

One design habit makes the workspace pleasant. `VAR` steps forward through the alphabet, one letter per press, wrapping from `Z` to `A`. Equations whose letters march forward, `X`, `Y`, `Z` for the savings account, then `A` and `B` for the loan, keep every change of unknown to a single press. That is worth planning for when you write the equation.

TRY IT

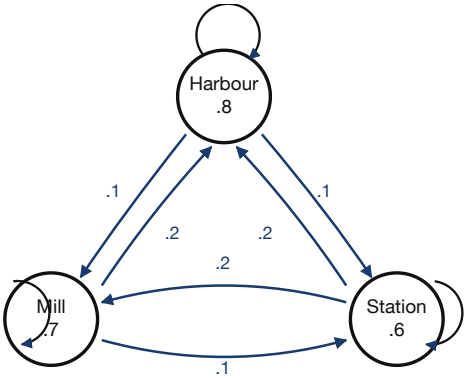
1. How long does money take to treble at six per cent? Predict it from step 4's doubling time first, then solve, then compare with `LN(3)/LN(1.06)` on the home screen.
2. A car loan of 8000 at 0.8 per cent a month runs for 36 months. Adapt the loan equation, mind which letters you reuse, and find the payment.
3. Store the exact payment from step 7 into `A` and solve for the term `B`. How close to 24 does the root come, and what does the residual say about the last penny?
4. Find the payment at which the loan of step 6 takes exactly 100 months. Then find the payment at which it takes 1000. What are those two numbers converging on, and why?

2.6 Loyalty in the long run

Three coffee shops share a harbour town: the Harbour, the Mill and the Station.

Each Saturday, 80 per cent of the Harbour's customers return and ten per cent defect to each rival. The Mill keeps 70 per cent, losing 20 to the Harbour and 10 to the Station. The Station keeps 60, losing 20 to each.

A table of switching fractions is a transition matrix, its powers are forecasts, and the matrix editor can raise them and find where the switching settles.



The three shops with the switching fractions on the arrows between them

1. Press [2nd] [7], then [+] [x-VAR] [+] [x-VAR] to reach `SIZE 3X3`, and type the matrix row by row, one row per shop: .8, .1, .1, then .2, .7, .1, then .2, .2, .6.

   Each row lists where one shop's customers stand next Saturday, and each row sums to one, because customers go somewhere. Check that as you type; a row that does not sum to one is a typing error and the machine will not tell you.

2. The editor's `MUL` multiplies `A` by `B`, so the square of P needs P in both. Press [ALPHA] for `B`, grow it to `SIZE 3X3` the same way, retype the nine values, and press [ALPHA] to return to `A`.

3. Press [MORE] [F3], `MUL`, and give the 3 by 3 product a moment.

   The banner switches to `R` holding the two-week forecast: the selection sits at `CELL 1 1` reading `0.68`, and stepping right reads `0.17` and `0.15` across row 1.

   Two Saturdays out, a Harbour regular is at the Harbour with probability 0.68. The 0.8 loyalty has already eroded, and the erosion is the point: a customer's history stops mattering surprisingly fast.

4. Forecasts further out are more of the same. Step through the rest of `R` (row 3 ends at `0.4` in `CELL 3 3`), press [▶] once more to wrap home, then [ALPHA] twice to return to `A`, and retype the nine two-week values: .68, .17, .15, .32, .53, .15, .32, .28, .4.

   `MUL` again answers three weeks, row 1 reading `0.608`, `0.217`, `0.175`. Carry that forward the same way and `MUL` once more: four weeks out, the first column reads `0.5648`, `0.4352`, `0.4352` down the three rows, and row 3 ends at `0.25`.

The rows are converging on one another, which is the thing to watch for. Where a customer started is washing out of the forecast.

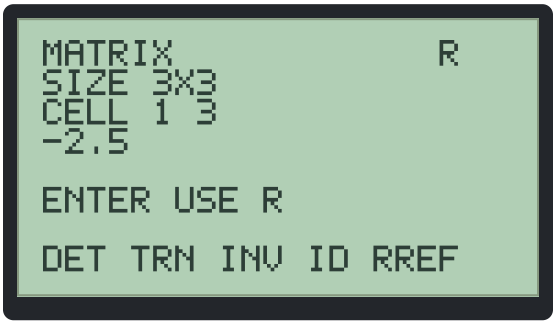
5. Where is it all heading? You could keep multiplying, but there is a much better question: what share-out would not change?

   Call the standing shares x, y and z. Next Saturday the Harbour collects .8x from its own regulars plus .2y and .2z from the switchers, and a share-out that stands still must collect exactly x again: .8x + .2y + .2z = x, which is -.2x + .2y + .2z = 0.

   The Mill and the Station give two more rows of the same shape, each a column of P with one subtracted on the diagonal.

   Retype `A` as that matrix: -.2, .2, .2, then .1, -.3, .2, then .1, .1, -.4. Press **EXIT** and **2nd** **7** to bring back the first soft-key page, and press **F5**, `RREF`.

   Stepping through `R` reads `1`, `0`, `−2.5`, then `0`, `1`, `−1.5`, then a row of zeros:



The steady-state proportions in register R

   The row of zeros is not a failure. It is the system telling you that the three equations are not independent, which they cannot be: if two shares are known the third is whatever is left. So the answer comes as a proportion rather than three numbers, 2.5 to 1.5 to 1, and dividing by their sum, 5, gives 0.5, 0.3 and 0.2.

6. The claim deserves its own check. Wrap the selection home, press **ALPHA** twice for `A`, and press **-** twice: `SIZE 1X3`, a single row. Type .5, .3, .2, and press **MORE** **F3**: `B` still holds P, and the product of a share-out with the transition matrix is next Saturday's share-out.

   `R` answers `SIZE 1X3` reading `0.5`, `0.3`, `0.2`, unchanged to the last digit.

Half the town ends at the Harbour, three tenths at the Mill, a fifth at the Station, and no further Saturday moves the needle. The Station keeps the fewest customers and ends with the smallest share, which is not a surprise; what is a surprise is how little the starting position mattered.

Every result landing in R is the register design of the Guidebook, chapter 13. The copying forward in steps 4 and 5 is what iterating inside one editor costs: the fourth power arrives in three multiplications and three retypings, and the steady-state matrix is a fourth retype.

The reward is that nothing is hidden. Every forecast you quote is one you watched being made, and on a machine that could raise a matrix to the fortieth power in one keystroke you would never have noticed that the rows converge, which is the actual mathematics here.

TRY IT

1. Start the whole town at the Station: a SIZE 1X3 row holding 0, 0, 1 in A, with P in B. Multiply repeatedly, carrying each result back into A with ENTER on the ENTER USE R prompt. After how many Saturdays does the Harbour's share first pass 0.45? Guess first.
2. The TRN key transposes A into R. Rebuild step 5 without arithmetic on paper: transpose P, carry it to A, put an identity in B, and subtract with SUB before RREF. Which registers did the answer pass through?
3. The Station renovates and its loyalty rises to 0.8, losing 0.1 to each rival. Predict whether that is enough to reorder the long-run shares before you rebuild the steady state.
4. Start the town at the steady state and multiply once. Then start it at 0.5, 0.3, 0.2 with the *Station's* improved matrix from exercise 3 and multiply once. What does the difference between those two runs tell you about how quickly a change in service shows up in the books?

CHAPTER

3

Explorations in Probability and Statistics

Statistics is usually taught on data sets too big to think about, where the machine's summaries have to be taken on trust.

Free85 turns that round, and not by accident. Its statistics columns hold eight entries. Eight numbers are few enough to check every claim the machine makes by hand, and that is the whole design: at this size you never have to believe anything, you can always look.

The chapter summarises a small data set and then audits the summary. It meets the random number generator and its strictly repeatable stream, and builds simulations in the program environment. It finds out what "best fit" actually means by computing it by hand, fits competing models to two invented data sets, and forecasts from the better one. It ends with the four statistical plots.

The statistics editor is the Guidebook, chapter 15.

A week of small data 3.1

The harbour town of Chapter 2 keeps a lighthouse, and the keeper tallies its visitors: 15 on Monday, then 12, 17, 15, 19, 21, 18 through the week, and 43 on the bank holiday Monday that closes the log.

Eight numbers, one of them suspiciously large. Exactly what a summary is for, and few enough to audit afterwards.

1. Press STAT to open the statistics editor. A fresh machine holds four entries, so press + four times for eight, then type the diary in order, pressing ENTER after each value: 15, 12, 17, 15, 19, 21, 18,
   43.

After the eighth ENTER the INDEX line wraps back to 1, which is how the editor tells you the column is full.

2. Press F1, the 1V key. The one-variable summary answers MEAN 20, MED 17.5, S SD 9.6953597148325, and P SD 9.0691785736085.

3. Now audit it, because you can.

   The mean is right: the eight values total 160, and 160 over 8 is 20.

   The median is the more telling check, because the week was typed in diary order and not in size order. On paper the ordered week reads 12, 15, 15, 17, 18, 19, 21, 43, and its middle two values, 17 and 18, average to the 17.5 on the screen. So the summary puts the column in order for itself, and its figures do not depend on the order you typed them in.

4. The editor is a different matter, and so is the line plot of section 3.7: both show the column exactly as stored.

   Press CLEAR to return to the editor, press MORE four times to the P4 FCX FCY SX SY page, and press F4, the ascending sort SX. The selection returns to INDEX 1, now reading 12, with the week in size order down the screen.

   Press MORE once more, to the SHW XYLN LIN 1V 2V page, and press F4 for 1V again: every figure of step 2 comes back unchanged. Which is the point. Sorting changed the editor and changed nothing about the statistics.

5. The spread audits just as well. Press CLEAR, press MORE twice to the MEAN MED VAR SSD PSD page, and press F3, VAR: the sample variance is 94, exactly.

   By hand: the deviations from 20 are -8, -5, -5, -3, -2, -1, 1 and 23, their squares total 658, and 658 over 7 is 94.

   The S SD line of step 2 is its square root, and its last digit sits a whisker under the true 9.69535971483266. That is arithmetic, not mathematics: a square root of a fourteen-digit number, rounded once.

6. Press CLEAR, then MORE for the MIN MAX Q1 Q3 BOX page, and collect the five-number summary one key at a time, pressing CLEAR after each result screen: MIN answers 12, MAX answers 43, Q1 answers 15, and Q3 answers 20.

   Write those five down before the next step, because you are about to look at a picture of them.

7. Press F5, BOX:

The box plot of the keeper's week

   The plot hangs the three quartile bars between edges standing for 12 and 43, and all three crowd into the left third of the screen.

   That crowding is the picture of skew. Half the week sits between 15 and 20, and one bank holiday drags the right edge four box-widths further out.

8. The mean-against-median verdict closes the story. Press EXIT for the home screen, which comes back showing a leftover `= 12`, the selected entry handed back, so press CLEAR before typing.

   Then `(12+15+15+17+18+19+21)/7` and ENTER: `= 16.714285714286`, the mean without the bank holiday.

   One day moved the mean from under 17 to 20, while the median crept from 17 to 17.5. Means follow outliers. Medians stay with the crowd, and that single sentence is most of what descriptive statistics is for.

Eight entries is the columns' whole capacity, and this section is the argument for liking it: at this size every summary can be re-derived by hand, so the machine is never believed, only checked.

TRY IT

1. The next week is quieter: 11, 14, 14, 16, 17, 18, 20, 22. Predict the mean and the median before pressing `1V`, and write down why you expect the two to land closer together than in the keeper's week.
2. Shrink the sorted column to seven entries with [-] and work out `Q1` and `Q3` on paper first. Which value left, and which quartile moved?
3. The population variance divides the 658 by 8 instead of 7. Work it out, then check it by squaring the `P SD` figure on the home screen with [x²]. Does it come back exact, and should it?
4. Replace the 43 with 19 and re-run everything. Predict, before you do, which of the eight figures from steps 2, 5 and 6 change and which do not.
5. Invent a week of eight numbers whose mean and median are equal but whose box plot is still visibly lopsided. Is that possible? Argue it out before you type anything.

3.2 Random numbers that repeat

A random number generator is a paradox to keep in a pocket: a fixed rule pretending to be chance.

Free85 makes the pretence unusually plain, and this section is about learning to use that honesty rather than being disappointed by it. The two functions are `RAND()`, a four-decimal value between 0 and 1, and `RANDI(low,high)`, a whole number from `low` through `high` inclusive.

1. On a cleared home entry line, spell `RAND()` letter by letter, [ALPHA] then the key carrying each letter, with the brackets typed directly, and press [ENTER]: `= 0.7968`.
2. The entry line keeps its text after an evaluation, so [ENTER] alone asks again. Three more presses answer `= 0.8984`, then `= 0.4492`, then `= 0.7246`.
3. Those four values are not a sample of anything. They are the opening of a fixed sequence, and a fresh machine answers `0.7968` then `0.8984` every single time, with the stream carrying on from wherever the last call left it.

   That is deterministic by design, and I want to be clear that it is a design and not a shortcoming. For an experimenter it is reproducibility: any simulation in this chapter, rerun from a fresh start with the same keys in the same order, delivers the same figures, which means you can check my numbers and I can

check yours. It also makes these functions fit for experiments and for nothing whatever that needs to be secret.

4. Dice come from the same stream. Press **CLEAR**, spell `RANDI(1,6)`, and press **ENTER**: `= 6`, the stream's fifth draw dressed as a die. Five more presses of **ENTER** roll 4, 6, 1, 1, 4.

   On a fresh machine the dice open 3, 5, 3, 5, 6 instead. Same stream, different entry point: the four `RAND()` calls of step 2 consumed four draws before the dice got a look in.

   `RANDI(0,1)` flips coins, `RANDI(0,9)` draws digits, and every draw, whatever costume it wears, advances the one sequence by one step.

TRY IT

1. From a fresh machine, roll `RANDI(1,6)` ten times and tally the faces. Which face never appears, and how far is the tally from uniform? Then say whether that is evidence of anything at all.
2. Two dice are two draws. Evaluate `RANDI(1,6)+RANDI(1,6)` repeatedly and watch middling sums dominate. Why is 1 impossible, and why is 7 the most likely?
3. Use `RANDI(0,9)` three times to build a three-digit random number on paper. How far does the stream advance, compared with calling `RANDI(0,999)` once? Does it matter?
4. Predict what `RANDI(1,1)` does, and how far it advances the stream. Then check.

Simulation by program 3.3

Rolling a die thirty-six times by hand is character-building. A program does it in one keypress and never loses count.

Two builds follow, a nine-flip warm-up and a thirty-six-roll dice experiment, in the program environment of the Guidebook, chapter 16.

1. Press **PRGM**, then **F1**, `NEW`. The editor opens on `EDIT P1`, `LINE 1`. Remember from Chapter 4 that it shows you one line at a time and never a listing, so keep the table below in front of you as you type.

   Letters are **ALPHA** plus the key carrying the letter, spaces are **2nd** **0** in this editor, and **STO▶** types the –> arrow.

Line	Text	Keys
1	`0->S`	0 STO▶ S
2	`FOR A,1,9`	F O R 2nd 0 A , 1 , 9
3	`S+RANDI(0,1)->S`	S + R A N D I (0 , 1) STO▶ S
4	`END`	E N D
5	`DISP S`	D I S P 2nd 0 S
6	`STOP`	S T O P

2. Press F2, `RUN`. The run screen answers `RUN P1` over `LINE 6`, the output line shows `3`, the status reads `DONE`, and the footer `ON STOP` names the panic button.

   Three heads in nine flips, from the stream section 3.2 was reading.

3. Nine used to be as far as a counted loop went. `FOR` took single-digit bounds until firmware 2.19, which is why this program counts down with `WHILE`, exactly as Chapter 8's series do.

   It no longer has to. `FOR` now takes evaluated bounds and an optional step, so `FOR N,1,50` is a legal line and `FOR N,50,1,-1` counts back down. Run the program below as it stands first, because the countdown is what the next two sections read; then, if you like, rewrite line 2 as a `FOR` and satisfy yourself that the answer does not move.

   Press PRGM for the list, ▼ to select the second slot, and F1 to open `EDIT P2`. Type its eight lines:

Line	Text
1	`36->N`
2	`0->S`
3	`WHILE N`
4	`S+INT(RANDI(1,6)/6)->S`
5	`N-1->N`
6	`END`
7	`DISP S`
8	`STOP`

Line 4 is the interesting one and it is worth staring at.

The editor cannot type =, so "did the die show six" has to become arithmetic. `RANDI(1,6)/6` is 1 exactly when the roll is a six and somewhere in (0, 1) otherwise, so `INT(` of it is 1 for a six and 0 for everything else. Then `S` tallies it.

That trick, turning a test into a whole-part, is the single most useful thing to know about programming this machine. Chapter 8 uses it again to stop a loop on a tolerance.

4. Press **F2**. After a moment's spinning the run screen answers `8` on `LINE 8`:

Thirty-six rolls tallied by P2

Eight sixes in thirty-six rolls, against an expected six.

5. Run it again, **PRGM** then **F3**, and the count is `13`.

The stream carried on into a patch rich in sixes, and thirteen in thirty-six is the sort of wobble small experiments produce. Determinism holds all the same: a fresh machine that types and runs exactly this section answers 3, then 8, then 13, in that order, every time.

The environment's bounds shape the design here: four programs of eight 48-character lines, and no way to type = or <, so conditions get built out of arithmetic as line 4 does. A fourth bound, `FOR`'s single-digit count, shaped it too until firmware 2.19 removed it.

TRY IT

1. Edit `P2` to roll ninety-nine times and compare the tally of sixes with 99/6. Which single line changes? Predict the tally's rough size before you run it.
2. The tails `P1` does not display are not lost. Add a line displaying `9-S` and decide which figure the run screen leaves on show, before you try it.
3. Rewrite `P1`'s loop with `REPEAT` and a countdown, remembering that `REPEAT` tests before every pass. What test expression makes the body run nine times?
4. Line 4 tests for a six. Change it to test for an even roll instead. There is more than one way; find one that uses `INT(` and one that does not.

3.4 Two columns and a family of models

Paired data asks a sharper question than one column can: not "what is typical" but "what depends on what, and how".

The editor fits seven models to its two columns, and choosing between them is the exploration. Two invented experiments supply the data: a bean seedling measured daily, and duckweed spreading across a pond.

1. The bean first. On a fresh machine press **STAT**, press **+** twice for six entries, and type the days 1 through 6 into `X`. Press **ALPHA** to switch to the `Y` column and type the measured heights in centimetres: 7, 7, 10, 12, 13, 17.
2. Press **F2**, `2V: MEANX 3.5, MEANY 11`, and the correlation `R 0.9725975251592`, strongly linear.

   Press **CLEAR**, then **F3**, `LIN`: the result screen answers `MOD LIN` with `A 4` and `B 2`, which is the line y = 4 + 2x. Growth of two centimetres a day from a four-centimetre start.

   Check it against the data by hand, because it takes ten seconds. The line predicts 6, 8, 10, 12, 14, 16 for the six days, so the data misses it by 1, -1, 0, 0, -1, 1. Small errors, both directions, no drift. That is what measurement noise looks like, and section 3.5 is about why those particular numbers are the best any line could manage.
3. Now the duckweed, logged weekly as square metres of the harbour master's pond covered: 6, 12, 24, 48, 96 across weeks 1 to 5.

   Press **CLEAR** to leave the result screen, press **-** once for five entries, press **ALPHA** to return to the `X` column, type 1 through 5, press **ALPHA**, and type

the five areas into Y.

4. Fit the straight line anyway, because the failure is instructive. 2V answers R 0.9332565252573, and LIN answers A −27.6 with B 21.6.

   The correlation looks respectable and the fit is terrible, which is this section's central lesson. The line predicts -6 for week 1, 37.2 for week 3 and 80.4 for week 5, against 6, 24 and 96. The misses swing from plus to minus and back, which is a curve's signature, not noise.

   A correlation of 0.93 did not warn you. R alone does not choose a model. The residuals do.

5. Look at the shape. Press CLEAR, then F4, SCAT:

The duckweed pairs bending upward

   The dots hug the floor and then leave it, each gain bigger than the last. Multiplication, not addition.

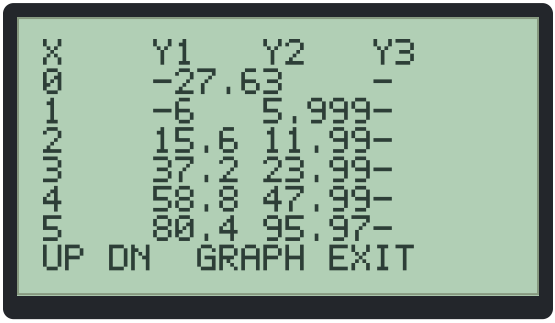
6. Fit the multiplying model. Press EXIT, then STAT to reopen the editor on its first page, press MORE three times to the LNR EXPR PWR P2 P3 page, and press F2, EXPR, which fits y = A e^(Bx).

   The screen answers MOD EXP with A 3.0000000000031 and B 0.69314718055973.

   That B is ln 2 in costume. Press CLEAR and ask LN(2): = 0.69314718056122, the same to eleven places. So the model is three square metres doubling every week, which is exactly what the data was built from.

7. Put both fits side by side. Press EXIT, and the home screen comes back showing a leftover = 0.69314718055973 handed out of the result screen, so press CLEAR.

Type `-27.6+21.6*X` ((-) for the sign) and press **GRAPH** to store the line in `Y1`. Press **2nd** **2**, press **CLEAR**, type `3*EXP(.6931*X)` (the fitted coefficients to four places, `EXP(` on **2nd** **LN**), and press **GRAPH**, letting the slow plot finish. Press **MORE** for the table:



The two fitted models against the weeks

Down the `X=1` to `X=5` rows, `Y1` reads `-6`, `15.6`, `37.2`, `58.8`, `80.4` while `Y2` reads `5.999`, `11.99`, `23.99`, `47.99`, `95.97`.

One column misses the areas by up to thirty square metres. The other misses by hundredths. Residuals by eye settle what `R` could not.

The columns cap at eight pairs, so experiments for Free85 are planned around few, well-spaced observations. Five weekly readings separated two models cleanly, which is a reminder that the number of observations matters much less than where you put them.

TRY IT

1. The bean data minus its noise is 6, 8, 10, 12, 14, 16. Predict `R` before you enter it and fit `LIN`. What does noiseless correlation look like?
2. Cross the models: fit `EXPR` to the bean heights and test its predictions for days 1 and 6 on the home screen. Which way do the residuals drift, and what does that drift tell you?
3. Invent a five-pair data set that `PWR` ($y = A\,x\,\hat{}\,B$) fits exactly, enter it, and check the machine recovers your two constants.
4. Fit `LIN` to the duckweed's *logarithms* instead: work out ln of each area on paper, type those into `Y`, and fit. Compare the slope with step 6's `B`. Why should they agree?

What “best fit” actually means

3.5

Section 3.4 pressed `LIN` and the machine handed back a line. This section is about where that line comes from, because “best fit” is a phrase that sounds like it explains something and does not.

Best in what sense? Nearest to what? The answer is completely definite, and you can compute it yourself in about five minutes.

Here is the rule. For any line you care to name, go along the data, take the vertical distance from each point to the line, square it, and add up the squares. That total is the line’s score, and the lower it is the better. The line `LIN` gives you is the one with the lowest score of all possible lines. That is the whole of it, and the name for the total is the sum of squared residuals.

Two questions people always ask, worth answering before you compute anything. Why vertical distance, and not perpendicular? Because you are predicting y from x, so a miss in y is what costs you. Why squared, and not just the size of the miss? Partly so that overshoots and undershoots cannot cancel, and partly because squaring makes one big miss cost far more than several small ones, which is usually what you want.

A program to keep score

The bean data has six points, so the total is six squared misses added together. That is too long for one entry line, which holds 48 characters, so it wants a program.

1. Press **PRGM** and **F1**, `NEW`, opening `EDIT P1`. Type these six lines. The slope lives in `M` and the intercept in `B`, so the program never contains a particular line and will score any line you like:

Line	Text	Keys
1	`0->S`	0 STO▶ S
2	`S+(B+M*1-7)^2+(B+M*2-7)^2->S`	S + (B + M × 1 - 7) x² + (B + M × 2 - 7) x² STO▶ S
3	`S+(B+M*3-10)^2+(B+M*4-12)^2->S`	with 3 and 10, then 4 and 12
4	`S+(B+M*5-13)^2+(B+M*6-17)^2->S`	with 5 and 13, then 6 and 17
5	`DISP S`	D I S P 2nd 0 S
6	`STOP`	S T O P

The six heights 7, 7, 10, 12, 13, 17 are written into lines 2 to 4, two points to a line, because at 48 characters that is as many as will fit. The days 1 to 6 are the multipliers of `M`.

The longest line is 30 characters, so there is room to spare. If your data were different you would retype these three lines and nothing else.

2. Score a deliberately poor line first, so you have something to beat. Press EXIT for the home screen and CLEAR, type 3 STO▶ ALPHA SIN (the letter `B`) and press ENTER: `= 3`. Press CLEAR, type 2 . 5 STO▶ ALPHA 8 (the letter `M`), and press ENTER: `= 2.5`.

   That is the line y = 3 + 2.5x. Press PRGM, select `P1`, and press F2, `RUN`:

The score for a poor trial line

   `11.75`.

   That number means nothing on its own. It is only useful compared with another one, which is the next step.

3. Now score the line `LIN` gave you in section 3.4, y = 4 + 2x. Press **PRGM** to leave the run screen, **EXIT**, **CLEAR**, store `4->B`, press **CLEAR**, store `2->M`, then **PRGM** and **F2**:

The score for the least squares line

   `4`.

   Much better, as it should be. Note also that it is exactly 4, and you can see why from section 3.4: the six residuals were 1, -1, 0, 0, -1, 1, and six squares of those add to 4.

4. Now the part that makes the claim real. `LIN` says 4 + 2x is the *best* line. Try to beat it.

   Nudge the intercept: store `4.5->B`, keep `2->M`, and run. `5.5`.

   Nudge it the other way, `3.5->B`: you get `5.5` again, by symmetry.

   Nudge the slope instead: back to `4->B`, then `2.1->M` and run. `4.91`.

   Every direction you move in, the score goes up. That is what a minimum is, and you have just verified it by hand rather than taking it on trust. `LIN` did not find a good line. It found *the* line, in a perfectly precise sense that you can now state.

5. One more, to see the shape of the thing. Nudging the slope by 0.1 cost 0.91. Predict what nudging it by 0.2 will cost, write the number down, then store `2.2->M` and run.

   `7.64`. The excess over 4 is 3.64, which is exactly four times 0.91.

   So the score grows like the *square* of how far you move, not in proportion to it. That is what makes the minimum a smooth bowl rather than a sharp point.

   It is also why a line that is slightly wrong is only slightly worse, and why fitting is a stable business: small errors in the data move the answer by small

amounts.

TRY IT

1. Predict the score for y = 4 + 2x on the noiseless bean data 6, 8, 10, 12, 14, 16, before running anything. Then edit lines 2 to 4 and check.
2. Work out on paper why nudging the intercept up and down by the same amount gives the same score, but nudging the slope up and down does not have to. Test your reasoning with `1.9–>M`.
3. The mean of the bean heights is 11 and the mean day is 3.5. Check that the least-squares line passes exactly through (3.5, 11). Then explain why it has to.
4. Score the flat line y = 11, the mean of the heights, by storing `11–>B` and `0–>M`. You should get `74`. Compare it with the 4 of step 3: the trend explains all but four seventy-fourths of the variation, and one minus that fraction is where the `R` of section 3.4 comes from. Check that against `R` squared.
5. Retype lines 2 to 4 for the duckweed data of section 3.4 and score the straight line `LIN` gave, A -27.6 and B 21.6. Then score the doubling model by hand at the same five points. Which is smaller, and by how much?

3.6 Forecasting the full pond

The pond holds one hundred square metres, and the model of section 3.4 says the duckweed doubles weekly.

A model's job is to answer questions the data has not reached, and the forecast keys do exactly that, reading their question from whichever entry the editor happens to be standing on. That last detail is the whole trick and the whole trap.

1. Step off the table screen that closed section 3.4: press **EXIT** to the plot, let the slow exponential finish redrawing, and press **EXIT** again for the home screen.

   Then rebuild the fit, because the forecast keys read whichever model was fitted last and you have fitted several. Press **STAT**, press **MORE** three times, and press **F2**, `EXPR`, which answers the same coefficients as before. Press **CLEAR** for the editor.
2. Ask about week 6. Press **+** to grow the columns to six entries, press **▲** to wrap the selection to the new `INDEX 6`, type 6, press **ENTER**, and press **▲** to stand on the new entry again.

   Press **MORE** for the `P4 FCX FCY SX SY` page and press **F3**, `FCY`, which forecasts y from x. The `FORECAST` screen answers `191.99999999971` over the `6`

it read: the model's 192 in machine arithmetic.

That forecast is absurd, and usefully so. A hundred and ninety-two square metres will not fit in a hundred square metre pond, so somewhere in week six the model stops being true. The model does not know that. Models never do. Knowing where your model stops is your job and it is not a job you can delegate to a fitting key.

3. So ask the better question: when is the pond full? That is a forecast in the other direction, x from y, and `FCX` reads its target from the `Y` side of the selected entry.

   Press **CLEAR**, press **ALPHA** to switch to the `Y` column, press ▲ to wrap to `INDEX 6`, type 100, press **ENTER**, and press ▲ to stand on it. Press **F2**, `FCX`:

The pond-full forecast

   `5.0588936890553` over the target `100`. The pond closes over early in the sixth week, about half a day in.

4. Audit it, because you can do this one on paper. Press **EXIT** (the home screen returns showing a leftover `= 100`, so press **CLEAR**) and type `LN(100/3)/LN(2)`, which is the paper solution of 3 times 2 to the x equals 100.

   ENTER answers `= 5.0588936890444`, agreeing to ten places. The last digits differ because the two routes walk different arithmetic to the same number, which is the ordinary state of affairs and not a fault.

One caution, and it is the kind that bites silently. The fit is not recomputed until a regression key is pressed again, so the scratch entry holding 6 and 100 never disturbed the model it was questioning. But it would join the next fit. Shrink the

columns with [-] before fitting anything else, or your data will quietly acquire a point you invented.

TRY IT

1. Move the selection back to `INDEX 6`, store 7 on the `X` side, and forecast the area for week 7. How much pond would that need? Work out the answer from the doubling before you press the key.
2. The town council acts at half cover. Store 50 as a `Y` target, forecast the half-full week, and explain on paper why it must come exactly one week before the full-pond answer.
3. Store 3 on the `Y` side and forecast the x it comes from. Predict the answer first. Why does the model place three square metres at week zero?
4. Forecast the week at which the pond holds 200 square metres. The machine will answer. Say precisely what is wrong with the answer, in a sentence you would be happy to put in a report.

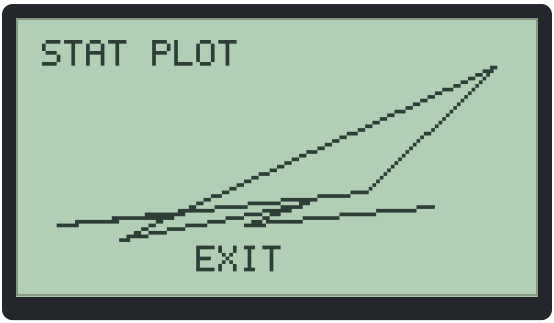
3.7 Four pictures of one week

A data set has no single true picture. Each plot answers one question and is silent on all the others, so choosing a plot is choosing a question.

The keeper's week returns, day against visitors, for its portrait four ways. The tally sheets come off the spike in no particular order, which is where it starts.

1. On a fresh machine press [STAT], press [+] four times, and type the days as the sheets surface, 3, 1, 6, 8, 2, 5, 4, 7, into `X`. Press [ALPHA] and type each sheet's count beside its day into `Y`: 17, 15, 21, 43, 12, 19, 15, 18.
2. Press [MORE] five times to the `SHW XYLN LIN 1V 2V` page and press [F2], `XYLN`,

the line plot:



The shuffled week drawn by XYLN

The plot joins the pairs in *entry order*, so the shuffled sheets draw a criss-cross that says nothing whatever about the week.

`XYLN` is the one plot of the four that trusts your ordering, and that makes it the one that can lie to you without any warning at all.

3. Sort the pairs. Press **EXIT**, which leaves a harmless `= 17` on the home screen, then press **STAT** to reopen the editor on its first page.

   Press **MORE** four times, then press **F4**, `SX`. The days come out 1 through 8, each count still riding beside its own day.

   Press **MORE** once and press **F2** for `XYLN` again: now the line ambles along the teens all week and leaps at the bank holiday, which is the week's actual story.

4. Press **EXIT**, then **STAT**, and press **F4**, `SCAT`: the same shape as dots.

   The scatter plot shows where the pairs sit and hides their order, which after the sort is no loss. Before the sort it would have quietly hidden the shuffle instead, and you would never have known there was one.

5. Press **EXIT**, then **STAT**, and press **F5**, `HIST`.

   The histogram answers four bars of exactly equal height, and it is not wrong. `HIST` reads the `X` column alone, and `X` holds the days 1 through 8, two to each of four equal-width bins. Four bars of two.

   A plot reads the columns, not your intentions. This is the cheapest possible demonstration of that and it is worth remembering the next time a chart looks surprising.

6. Give it the right column. Press **EXIT**, then **STAT**: the editor returns with the selection at `INDEX 1` of `X`, so type the counts straight over the days, 15, 12, 17, 15, 19, 21, 18, 43, pressing **ENTER** after each, and press **F5** again.

   One tall bar of six ordinary days, a single 21 beside it, an empty bin, and the bank holiday alone at the far right. This is the arrangement section 3.1 used all along, visitors in `X`, and it is what `BOX` wants too: both plots read `X` and ignore `Y`.

Four plots, one week. `XYLN` tells the story in time but only if the pairs are sorted. `SCAT` shows the pairing and hides the order. `HIST` and `BOX` show one column's distribution and hide the pairing altogether.

With eight entries the wrong choice costs you thirty seconds, which is the best argument for learning this on eight entries rather than on eight thousand.

TRY IT

1. Draw `BOX` straight after step 6, then sort with `SX` and draw it again. The two pictures are identical. Say which one of the four plots would have changed, and why `BOX` is not one of them.
2. Put the days back in `X` and the counts in `Y`, sort with `SY`, and draw `XYLN`. What ordering does the plot follow now, and what question does that answer?
3. The histogram's bins span minimum to maximum in four equal widths. Predict each count's bin for the keeper's week, on paper, and check against the bar heights of step 6.
4. Which of the four plots would show you that a data set had a value entered twice by mistake? Argue for each of the four before you decide.

CHAPTER

4

Explorations in Calculus I

Calculus asks what functions do at places you cannot reach: infinitely close to a point, or added up across infinitely many slivers. A calculator cannot reach those places either. What it can do is walk a very long way towards them and report back, and this chapter is largely about learning to read those reports properly, including the ones that are lying to you.

We probe limits with the table and the zoom keys, then meet a limit that does not exist at all. We build a derivative out of raw difference quotients before letting `NDER(` take over, and hunt turning points with the search commands. We measure an integral as an average before measuring it as an area, and program Riemann sums to watch one being assembled.

The calculus commands and the tolerance setting are the Guidebook, chapter 3; the analysis keys are the Guidebook, chapter 4.

One habit pays for itself all chapter. The calculus commands read the *active stored equation*, so store the function with **GRAPH** before asking them anything. With nothing stored, `EVAL(` and its family answer `SYNTAX ERROR`. Once something is stored they answer whether or not you let the plot finish drawing.

Limits by table and zoom 4.1

What is sin x divided by x, when x is nought?

Nothing. There is no answer. Nought over nought is not a number, and in a few minutes the machine will tell you so in as many words.

That is not the interesting question though. The interesting one is what the thing is *nearly*, when x is *nearly* nought. That does have an answer, a perfectly definite one, and going and getting it is what a limit is.

I have not chosen this example to be clever. It is the one every calculus course does, and for a good reason: it is the fact that makes the derivative of sine come out as cosine. Get this one and you have paid for most of the chapter in advance.

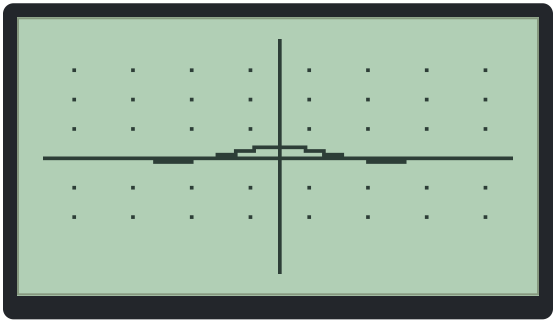
We will go at it four ways. Draw it, tabulate it, probe it, and then do some paper. Three of those will point at the answer. Only the last one will actually deliver it, and the difference between pointing and delivering is most of what this section is about.

Getting it into the machine

1. Press **CLEAR**. There is probably something left on the entry line from whatever you did last, and the machine will cheerfully add your new typing onto the end of it.
2. Press **SIN**. You get `SIN(`, bracket included. Then **x-VAR**, **)**, **÷**, **x-VAR**.

   Read the line before you go on. It should say `SIN(X)/X`. If it says `SIN((X)/X` you have pressed **(** out of habit after **SIN**, which everybody does once. Press **CLEAR** and type it again.
3. Press **GRAPH**. That stores the line into `Y1` and draws it in the standard window, -10 to 10 both ways.

   Sine is slow work for this machine. Let the curve reach the right-hand edge before you touch anything, because presses that land mid-draw go nowhere and you will decide the key is broken.



The bump of SIN(X)/X, flattened almost to nothing by the standard window

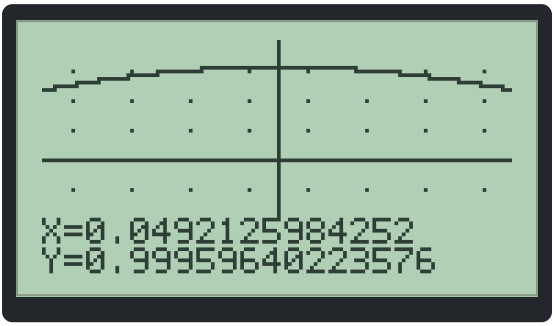
The graph, which is no help at all

Look at what you have got. A small bump over the origin, ripples either side, the whole thing squashed into a band a few pixels high.

That is the window's doing. You have made room for heights from -10 to 10, and this function never leaves the range -0.3 to 1, so nine tenths of the screen is empty sky. It is section 1.1's complaint in its natural habitat.

Fix it and get closer.

4. Press [+] three times, waiting for each replot. Each press halves every bound, so you are now looking at -1.25 to 1.25 and the bump fills the screen.
5. Press [▶] twice. The readout says `X=0.0492125984252` and `Y=0.99959640223576`.



The zoomed bump with the trace two columns right of centre

So a twentieth of a unit right of the middle, the function is 0.9996. Encouraging.

Now find the hole.

You cannot. Zoom as long as you have patience for and the curve stays a smooth unbroken arc, straight through the point that is not there.

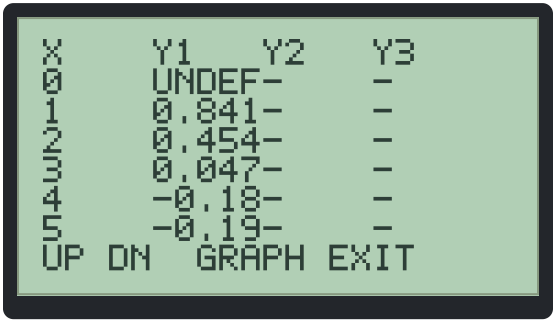
Here is why, and it is worth having straight because it will come back at you later in this book. The machine draws by choosing 128 columns across the window and working out the height at each. Nought is not one of those columns and no amount of zooming will make it one: halve the window and you get 128 new columns, and nought is not one of those either. A missing point one point wide is invisible to something that only ever looks in 128 places.

It is not lying to you. You asked about 128 columns and it answered about 128 columns. The question you wanted to ask was about somewhere it never looks.

The table, which is

The table asks at values you choose, nought included. So it can catch what the plot cannot.

6. Press [2nd] [+] for the standard window, let it redraw, then [MORE] for the table.

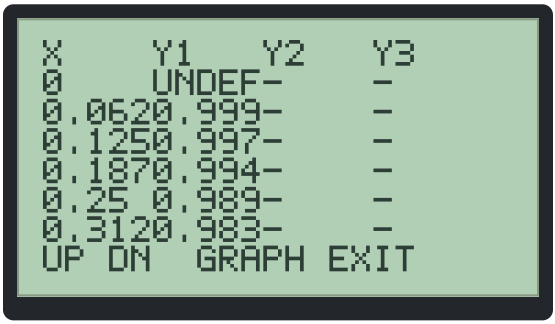


The table, with UNDEF sitting on the X=0 row

The `X=0` row says `UNDEF`. There it is. The machine has gone away, tried to divide nought by nought, failed, and come back and told you.

Underneath: `0.841`, `0.454`, `0.047`, `-0.18`, `-0.19`. Those are the ripples, not the answer. By `X=1` we are down to 0.841 and falling. Whatever is going on near nought is going on much closer in than a step of 1 can see.

7. Press [-] four times. Each press halves the table step, so you go 1, 0.5, 0.25, 0.125, 0.0625. Let each redraw settle before the next press.



The same table at step 0.0625, the values climbing towards 1

Read the rows below the hole *upwards*, towards it: 0.983, 0.989, 0.994, 0.997, 0.999. And the `X=0` row still says `UNDEF`.

That is the whole shape of it. Walk in towards nought and the values climb towards 1 without ever arriving, and at nought itself there is simply nothing at all.

8. Press [▲] to see the other side. From `X=-0.31` the rows read `0.983`, `0.989`, `0.994`, `0.997`, `0.999`, then `UNDEF`.

   The same five numbers in the same order, and that is not luck. Sine is odd, so sin(-x) over -x is the same thing as sin x over x. The function is a mirror image about the y axis with one point missing out of the middle, so both sides always had to climb to the same place.

Squeezing better numbers out of it

The table gives you three decimal places, because that is all a five-character cell will hold. The calculus commands will do better.

9. Press [EXIT] to leave the table, [EXIT] again for the home screen, and [CLEAR] to get rid of the equation the graph has just handed back to you.
10. Spell out `EVAL(.1)` (letters are [ALPHA] and then the key with that letter on it) and press [ENTER]. You get `0.99833416646834`.
11. Now smaller. Press [CLEAR] before each one:

Ask this	Get this
`EVAL(.01)`	`0.99998333341673`
`EVAL(.001)`	`0.9999998333334`
`EVAL(-.001)`	`0.9999998333334`
`EVAL(.0001)`	`0.9999999983334`

    Watch the nines. Two more of them every time x shrinks by a factor of ten. That is a rate, rates are worth noticing, and in a few minutes we will work out exactly where this one comes from.

    Notice too that `.001` and `-.001` agree in every last digit. That is step 8's mirror again, now stated to fourteen places.

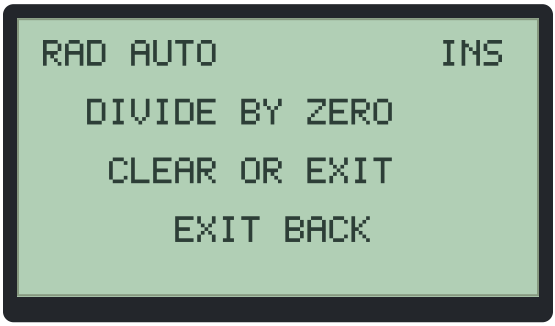
12. Push it harder. `EVAL(1E-6)`, with the `E` typed as [EE], gives `0.9999999999999`. `EVAL(1E-9)` gives `1`, flat.

Be careful with that last one. It does not mean the function equals 1 at a billionth.

A number in this machine is fourteen significant digits. That is seven bytes of packed decimal, two digits to the byte, and it is all the room a number gets. At a billionth, sine of x and x agree in all fourteen of them, so the division comes out as exactly 1. Nothing has been discovered. The machine has simply run out of places to keep the difference.

That is a fact about seven bytes, not a fact about sine. Confusing the two is the classic way to fool yourself with a calculator, and knowing the byte count does not make you immune. It just means that when a number looks too clean, you know which drawer to go and look in.

13. Now ask it the original question. Press **CLEAR**, spell `EVAL(0)`, and press **ENTER**:



EVAL at the hole, stopped on DIVIDE BY ZERO

`DIVIDE BY ZERO`, with `CLEAR OR EXIT` underneath. Which is exactly right: `SIN(X)/X` at zero divides by zero, and the machine has told you the truth about what it met.

It did not always. Until firmware 2.12 this answered `SYNTAX ERROR`, and the first edition of this book had a paragraph here apologising for it. There was nothing wrong with the syntax of `EVAL(0)`; the evaluation had failed at the point you asked about, and the calculus commands reported every failure of that kind through the one error the parser already had to hand. I wrote that, laying it out again, I would give each failure its own message.

I did. Domain, division, overflow, recursion, convergence and lost precision are now separate diagnostics, in the home screen, in graphs, in tables and in

programs. `SYNTAX ERROR` has gone back to meaning what it says: the machine could not read you.

The practical difference is that the message is now evidence. When a calculus command stops, the name tells you what it hit, and you can act on it instead of guessing.

Press **CLEAR** to clear the notice. The entry line still holds `EVAL(0)`, so press **CLEAR** again to empty that too. Two presses of **CLEAR** after any error screen: the first kills the message, the second empties the line. Get into the habit early and you will save yourself a lot of puzzled retyping.

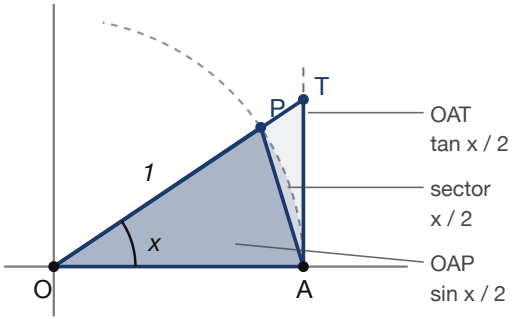
Why it is 1, which none of that proved

Stop and take stock for a moment. Everything so far points at 1. Nothing so far has proved 1.

That is not a quibble. A machine can try a hundred values, or a million; a limit is a claim about all of them at once, and no amount of trying will ever get you there. Pointing you at what to go and prove is what the machine is genuinely good for. It cannot do the other job and it is important not to let it pretend otherwise.

So here is the proof, and it is a nice one.

Draw a unit circle and take a small angle x at the centre. There are three regions, each sitting inside the next: the triangle inside the sector, the sector itself, and the larger triangle outside it.



A unit circle sector with the triangle inside it and the triangle outside it, the three areas that squeeze sine over x

Their areas are sin x over 2, then x over 2, then tan x over 2. Divide the lot through by sin x over 2, turn the inequality upside down, and what is left is that cos x sits below sin x over x, which sits below 1.

Now let x head for nought. Cosine heads for 1. The quotient is trapped between a thing heading for 1 and 1 itself, so it has nowhere to go but 1.

That is the squeeze, and you can watch it close.

14. Press **CLEAR** and ask `COS(.1)`: `0.99500416527802`. Step 10 gave `0.99833416646834` at the same x. Sure enough, the quotient is sitting between them, in a gap five thousandths wide.

    Press **CLEAR** and ask `COS(.01)`: `0.99995000041666`, against the quotient's `0.99998333341673`. The gap is down to five hundred-thousandths. Push the walls together and whatever is between them has no say in the matter.

There is a second route, and it explains those nines from step 11.

Sine of x is x, take away x cubed over 6, plus smaller stuff. Divide by x and sin x over x is 1, take away x squared over 6, plus smaller stuff still. So the error ought to go like x squared over 6: shrink x by a factor of ten and the error should shrink by a hundred, which is two more nines. Which is exactly what you saw.

15. Test it. Press **CLEAR** and type `1-.1^2/6`: `0.9983333333334`, against the quotient's `0.99833416646834`. Agreement to six decimals. Press **CLEAR** and

type `1-.1^2/6`: `0.9983333333334`, against the quotient's `0.99833416646834`. Agreement to six decimals. Press **CLEAR** and type `1-.01^2/6`: `0.9999833333334`, against `0.99998333341673`. Nine decimals.

That one is worth carrying around in your head. For small x, sin x over x is 1 take away x squared over 6, and you can do it without a machine at all.

The bit nobody tells you

The answer 1 is not really a fact about sine. It is a fact about sine *measured in radians*, and I can show you that in about ten key presses.

16. Press **2nd** **MORE** for the mode screen, press **F1** once so the second line reads `ANGLE DEG`, and press **EXIT**.
17. Press **CLEAR**, type `SIN(X)/X` again, and press **GRAPH**. Let it draw. It comes out as a flat line lying on the axis, which is your first clue that something has changed underneath you.
18. Press **EXIT**, press **CLEAR**, and ask `EVAL(.1)`: `0.017453283658983`. Press **CLEAR** and ask `EVAL(.001)`: `0.017453292519057`.

    Still converging. Converging on something else entirely. Press **CLEAR** and type `PI/180`, with the π legend on **2nd** **^**: `0.017453292519943`. That is where the probes are heading, and by x = .001 they have got nine decimal places of the way there.

    It is the chain rule wearing a false moustache. A degree is π/180 of a radian, so working in degrees quietly multiplies every angle by π/180 before the sine ever gets a look at it, and the limit gets multiplied by the same thing. Radians are simply the unit that makes the constant come out as 1. That is the real reason calculus insists on them, and it is a much better reason than "because the book says so".
19. Press **2nd** **MORE**, press **F1** once to get back to `ANGLE RAD`, and press **EXIT**.

    Do not skip that. I have left a machine sitting in `DEG` and then spent a quarter of an hour deciding the firmware was broken, when every trigonometric answer in the next section was simply being quietly scaled by π/180. It is a very cheap way to waste an afternoon.

TRY IT

1. Try the same four ways on `(1-COS(X))/X`. Table first, then probes at .1, .01, .001. Where is it heading? Then do the same for `(1-COS(X))/X^2` and work out what dividing by the extra x changed.
2. Guess before you press anything: `SIN(2*X)/X`. The limit is not 1. Work out what it should be from the series in step 15, write your answer down, and then go and check it at .01 and .001. Being wrong here is useful, so write the guess down before you look.
3. In step 7 you halved the table step four times and got to 0.999. Halve it four more times and read the same row. How many nines does the cell give you now, and what is stopping it giving you more? (The answer is about the cell, not about the mathematics.)
4. `EVAL(1E-9)` came back as exactly 1 in step 12, and we said that was the machine running out of room. Find the largest power of ten at which the answer is still visibly short of 1. That number tells you something about the machine you are holding. What?
5. Turn it upside down: store `X/SIN(X)` and probe from both sides. Its limit has to be 1 as well, and you can say why in one line. Then try to run the squeeze argument of step 14 on it unchanged and see where it goes wrong. Fixing it is a two-line job once you spot the trouble.
6. Put the machine in `DEG` and redo exercise 2. Predict the answer before you press a key, using what step 18 showed you. Then set it back to `RAD`, because you will want it there for section 4.2.

4.2 A limit that is not there

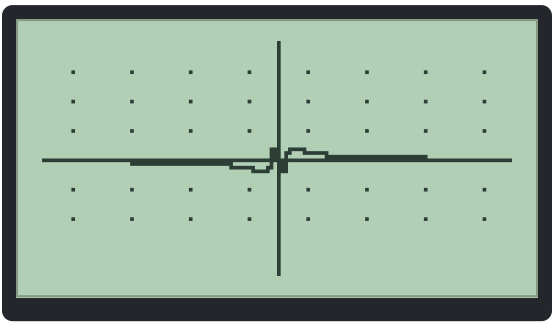
Section 4.1 could have left you with a comfortable and wrong idea: that if you probe hard enough, a number turns up. It does not always. Some functions have no limit at all at a point, and the useful skill is telling which kind you are looking at.

The specimen is sin of one over x. As x heads for nought, one over x runs away to infinity, so the sine is asked for its value at angles that get larger and larger without stopping, and it goes on doing what sine does: up, down, up, down, faster and faster. It never settles anywhere, because there is nowhere for it to settle.

Then we will change one thing, multiply the whole lot by x, and watch a limit appear out of the same oscillation.

Watching it fail to settle

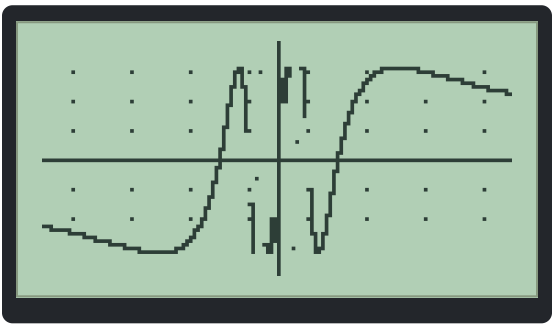
1. Press CLEAR, then SIN, 1, ÷, x-VAR,). The line reads `SIN(1/X)`. Press GRAPH and let it finish, which takes a while.



SIN(1/X) in the standard window

Away from the origin it is calm enough. It is the middle that matters.

2. Press + three times, letting each replot finish.



The same curve zoomed in three times, thrashing near the origin

Now compare that with section 4.1. There, zooming in made the picture *calmer* every time, until the curve was nearly a straight line. Here zooming in makes it worse. The wiggles do not spread out as you magnify, they crowd together, because there are infinitely many of them packed into any interval you care to draw round nought.

Those vertical strokes near the middle are the plotter losing the race. It has one column to spend on a stretch of x that contains several complete waves,

so it joins two samples that happen to be far apart and draws a near-vertical line between them. The picture is not wrong, it is just badly outnumbered.

3. Probe it and the numbers say the same thing. Press **EXIT**, press **CLEAR**, and ask `EVAL(` at a few points, **CLEAR** before each:

Ask this	Get this
`EVAL(.1)`	`-0.544021085826`
`EVAL(.05)`	`0.91294525072816`
`EVAL(.02)`	`-0.26237485369997`
`EVAL(.01)`	`-0.50636564109442`
`EVAL(.005)`	`-0.87329729713503`
`EVAL(.003)`	`0.31884634470865`

Put those beside section 4.1's column of nines. There, every probe was closer to the answer than the one before it. Here they are all over the place, and getting closer to nought does not help at all. Down, up, down, down, down, up. That is what no limit looks like when you meet one in the wild.

Where the machine gives up, and why

4. Keep going. Press **CLEAR** and ask `EVAL(.0025)`:

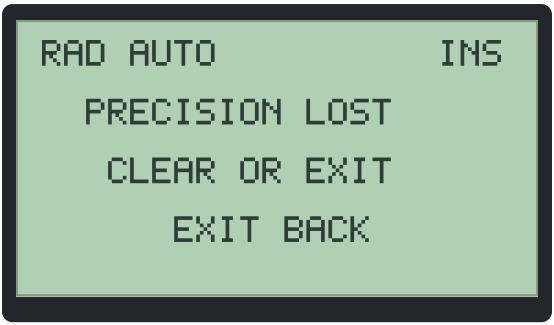
EVAL(.0025) answering the sine of 400 radians

`-0.85091935964129`. One over .0025 is 400, and the machine takes the sine of 400 radians without hesitating.

Keep pushing and it keeps answering. `EVAL(1E-6)` asks for the sine of a million radians and gets `-0.3499934460541`. That is the edge of the

guarantee: `SIN` and `COS` are supported through one million radians, or one hundred million degrees, and across that range they are held to about 1E-7.

Go past it and the machine stops, but not with a shrug. Press **CLEAR** and ask `EVAL(9E−7)`, which wants the sine of about 1.11 million:



PRECISION LOST at EVAL(9E-7), past the supported range

`PRECISION LOST`, and that name is the whole point. It is not saying the sine has no value there. It is saying that your input carries fourteen digits, and by the time an angle that large has been folded back into a single turn, those fourteen digits no longer pin down where in the turn you are. The answer would be a number, and it would be meaningless, and the machine would rather tell you than let you quote it.

Being told is worth more than being answered. A calculator that returned something here would be inviting you to publish it.

> **Historical note.** Firmware 2.10 reduced the angle by taking 2π off repeatedly and gave up after 63 goes, which put the wall at 395.84 radians, so `SIN(399)` refused. Reducing by quotient instead of by repeated subtraction moved the wall out by a factor of two and a half thousand. If you are reading an older edition of this book, that is why its numbers stop where they do.

So the limit is real but it is now a long way out: this machine follows sin of one over x down to about x = 1E-6, and no further. What that does not mean is that you have learned nothing. You have watched the function refuse to settle across six orders of magnitude in x, and the mathematics tells you it goes on refusing forever, at a rate no calculator was ever going to keep up with.

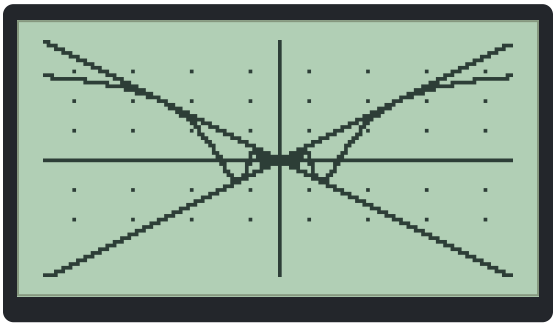
The same oscillation with a limit

5. Now change one thing. Press [CLEAR], type [x-VAR], [×], [SIN], [1], [÷], [x-VAR], [)] so the line reads `X*SIN(1/X)`, and press [GRAPH]. Let it draw.

   The sine is still doing exactly what it did before, swinging between -1 and 1 infinitely often. But now it is being multiplied by x, and x is on its way to nought.

6. Put the walls up so you can see it. Press [2nd] [2] to move to slot `Y2`, type [x-VAR], and press [GRAPH]. Press [2nd] [3] for slot `Y3`, type [(-)], [x-VAR], and press [GRAPH]. That gives you the lines y = x and y = -x on top of the curve.

7. Press [+] three times, letting each replot finish.



*X*SIN(1/X) pinched between the lines X and -X*

   There is the whole argument in one picture. The two straight lines close on the origin like a pair of scissors, and the curve is trapped between them, oscillating as wildly as ever inside a gap that is being squeezed shut. It has no room left to oscillate in.

   This is the squeeze of section 4.1 again, and this time you are not taking it on trust from a diagram. It is on the screen, and the three slots are exactly the right number of slots to show it: the thing, and the two walls closing on it.

8. The probes agree. [CLEAR] before each:

Ask this	Get this
`EVAL(.1)`	`-0.0544021085826`
`EVAL(.05)`	`0.045647262536408`
`EVAL(.02)`	`-0.0052474970739994`
`EVAL(.01)`	`-0.0050636564109442`
`EVAL(.005)`	`-0.0043664864856752`
`EVAL(.003)`	`0.00095653903412595`

Still bouncing between positive and negative, exactly as before, because the sine has not changed its mind about anything. But the size is collapsing: five hundredths, then five thousandths, then one thousandth. The sign is still random and the magnitude is not. The limit is 0.

Compare the two tables directly. Same function inside, same oscillation, same refusal to settle on a sign. One has no limit and the other has a perfectly good one, and the entire difference is the x out in front.

A test you can actually apply

There is a way to make "settles down" precise, and the graph screen is unusually well suited to it, because the thing you need is a rectangle and a window *is* a rectangle.

Say you claim a function heads for L as x heads for a. Someone who doubts you names a tolerance: they will believe it if the curve stays within that much of L. Your job is to find a window narrow enough that the curve does not leave the top or bottom of the screen anywhere inside it, apart from at a itself.

If you can always do that, however mean the tolerance, the limit is L. If there is a tolerance you cannot beat by any narrowing at all, there is no limit.

That is the whole of the epsilon-delta definition, in a form you can carry out with the zoom keys.

Try it on both of this section's functions and the difference is immediate. For `X*SIN(1/X)` every squeeze you try succeeds. For `SIN(1/X)` you cannot even get started, because the curve fills the band from -1 to 1 no matter how narrow you make the window.

TRY IT

1. Before pressing anything, write down what you expect the plot of `SIN(1/X)` to do far from the origin, out at x = 5 or 10. Then look. Why is it so calm out there when it is so violent near nought?
2. Work out on paper the x at which sin of one over x is exactly 1 for the first time going left from x = 1, then the next one, then the next. How fast are they bunching up? Check one of them with `EVAL(`.
3. `X*SIN(1/X)` is squeezed by `X` and `-X`. Predict what walls would squeeze `X^2*SIN(1/X)`, put all three in the slots, and check that the picture looks the way you said it would.
4. What about `SIN(1/X)/X`? Predict first: does it settle, blow up, or oscillate worse? Then probe it at .1, .05 and .02 and see.
5. `SIN` and `COS` are supported to one million radians. Work out the smallest x at which you could still ask for `SIN(1/X)`, then find the place the machine actually stops and explain why the two do not have to agree to the last digit.
6. Run the rectangle test of the last part properly on `X*SIN(1/X)`. Start from the standard window, pick a tolerance, and count how many presses of + it takes to satisfy it. Then halve the tolerance and do it again. Is the number of presses growing in a way you could have predicted?

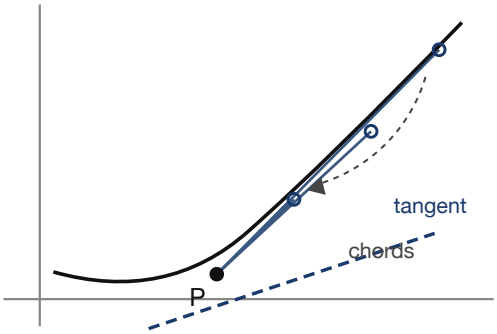
4.3 The derivative as a limit

The slope of a curve at a point is the limit of the slopes of chords through it. Unlike most limits, this one can be watched converging digit by digit, which is what makes it a good second exploration.

The function is f(x) = x^3 - 2x, and we will work at x = 1.5. On paper the derivative is 3x squared take away 2, so at 1.5 it is 4.75. Knowing the answer in advance is deliberate. It is much easier to see what a method is doing when you can tell how wrong it is at every stage.

Chords closing on a tangent

Take two points on the curve, join them, and measure the slope of the join. That is a chord, and its slope is the change in height divided by the change in x. Now slide the second point towards the first. The chord pivots and, if the curve is well behaved, settles onto a definite direction.



Three chords through the same point, each shorter than the last, closing onto the tangent

That settling is the whole idea. The tangent is not something you can measure directly, because it touches at only one point and one point does not give you a slope. What you can measure is chords, and then argue about where they are going.

1. Store the function. Press **CLEAR**, then **x-VAR**, **^**, **3**, **-**, **2**, **×**, **x-VAR** so the line reads `X^3−2*X`, press **GRAPH**, and let the plot finish.
2. Take the slope off the graph first, because it is one key. Press **▶** nine times, which lands the trace at `X=1.496062992126` with `Y=0.3563689017143`. Press **F4**, the derivative key: the home screen publishes `= 4.714613425`.

   That is not 4.75, and the reason is not the method. The trace stopped at the nearest sample column, which is 1.496 rather than 1.5, and the slope there really is a shade under. The machine answered the question you actually asked.
3. Now do it by hand, so you can see the limit happening. The **F4** result left `X^3−2*X` on the entry line, so press **CLEAR**. Spell `(EVAL(1.5+1)−EVAL(1.5))/1` and press **ENTER**: `= 10.25`.

   That is the chord from 1.5 all the way out to 2.5. Far too steep, and it should be: the curve bends upwards, so a long chord overshoots.
4. Shrink the step, pressing **CLEAR** before each:

Step	Chord slope
1	`10.25`
.1	`5.21`
.01	`4.7951`
.001	`4.754501`

Each tenfold shrink buys roughly one more correct digit of 4.75. Compare section 4.1, where each tenfold shrink bought two, and you can already tell these are different kinds of approximation. There the error went like x squared; here it goes like the step itself.

Where shrinking stops helping

5. Push further. With the step `1E-6`, typed with **EE**, the quotient `(EVAL(1.5+1E-6)-EVAL(1.5))/1E-6` answers `= 4.7500045`. With `1E-9` it answers `= 4.75` exactly. With `1E-12` it answers `= 4.8`.

   Read those last two carefully, because the tidy one is the liar.

   Section 4.1 said a number here is fourteen digits in seven bytes. As the step shrinks, f(1.5 + h) and f(1.5) come to agree in more and more of those fourteen, and the subtraction throws away every digit they share. At a step of 1E-9 you are subtracting two numbers that agree in nine digits, so about five survive, and those five happen to round to 4.75. At 1E-12 barely one survives and the answer staggers out as 4.8.

   So the clean 4.75 at 1E-9 was luck, not precision. Shrinking the step sharpens a chord only until cancellation blunts it, and then it makes things rapidly worse. There is a best step somewhere in the middle, and on this machine it is around 1E-6.

   This is not a defect you can fix by buying a better calculator. Every machine that keeps a fixed number of digits has this cliff. It moves; it does not go away.

6. The built-in command threads that needle for you. Press **CLEAR**, spell

NDER(1.5), and press ENTER:

NDER agreeing with the paper derivative

= 4.75. NDER(takes a central difference, sampling on both sides of the point and dividing by twice the step, which cancels the largest error term of the one-sided chords you have been computing. The Guidebook, chapter 3 documents the family.

Press CLEAR and check against paper with 3∗1.5^2−2: = 4.75.

The derivative is a function

Everything so far has produced one number, the slope at one place. But every point of the curve has a slope, so the slopes are themselves a function of x, and that function is the thing calculus actually cares about.

You can plot it directly, and there is one wrong way to ask that is worth meeting first.

7. With X^3−2∗X still in Y1, press 2nd 2 for slot Y2, spell NDER(X), and press GRAPH.

   Y2 stops with RECURSION ERROR, and Y1 carries on drawing.

   NDER(x) in that form reads *the active stored equation*, so a slot holding it is asking the machine to differentiate the equation it is in the middle of evaluating. Rather than chase its own tail it says so, and says so once rather than at every sample.

   Name the slot you actually mean and it is ordinary work. Press CLEAR, spell NDER(1,X), and press GRAPH: slot 2 now reads slot 1, and the derivative draws as a curve in its own right. One nested evaluation is available, which is

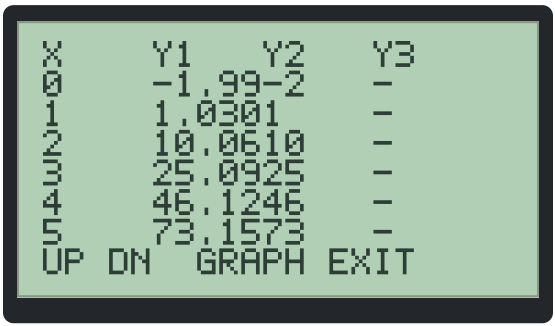
enough for a slot to read another slot but not enough for the two of them to read each other; a pair that does stops with the same `RECURSION ERROR`.

8. It is still worth typing the quotient out in full. It is longer but it has no such problem, because it mentions no commands at all. Press **2nd** **1**, press **CLEAR**, and type

   ```
   ((X+.01)^3−2∗(X+.01)−(X^3−2∗X))/.01
   ```

   which is the chord slope of step 4, at step .01, but with the point left as `X` instead of pinned at 1.5. Press **GRAPH** and let it draw.

9. Now put the true answer beside it. Press **2nd** **2**, press **CLEAR**, type `3∗X^2−2`, and press **GRAPH**. Press **MORE** for the table:



The difference quotient sitting just above the true derivative

   Reading `Y1` against `Y2` down the rows: `−1.99` against `−2`, `1.030` against `1`, `10.06` against `10`, `25.09` against `25`, `46.12` against `46`, `73.15` against `73`.

   The quotient is above the truth everywhere, by a little at the left and by more at the right, and the gap is growing. That is the chord overshooting a curve that bends upwards, which you already met at step 3 as a single number. Now you can see it happening at every x at once.

10. Shrink the step and watch the columns close. Press **EXIT**, press **2nd** **1**, press **CLEAR**, and retype the slot with `.001` in place of both `.01`s. Press **GRAPH**, then **MORE**. The columns now agree to the precision the cells will show.

    That is the limit again, and this time it is a limit of *functions*. The difference quotient is not creeping up on a number, it is creeping up on a whole curve.

TRY IT

1. Predict, before pressing a key, what the difference quotient of `X^2` will look like beside `2*X`. Will it sit above or below? By how much, and does the gap grow? Write it down, then build both slots and find out.
2. Run the shrinking chords of step 4 at a = 0 instead of 1.5, where f(0) is 0 and the typing is short. What slope do they head for, and does `NDER(0)` agree? (It answers `-1.9999999999`, which is worth a moment's thought on its own.)
3. The backward chord `(EVAL(1.5)-EVAL(1.5-.01))/.01` comes at the point from the other side. Compute it, compare with the forward .01 answer, and say which side of 4.75 each lands on and why. Then average the two and see what you get.
4. Ask `NDER(` at 0, 1 and 2 and check each against 3x squared take away 2. The answers are `-1.9999999999`, `1` and `10`. Why is only the first one dusty?
5. Find your own worst step. Compute the chord at 1.5 with steps 1E-4, 1E-5, 1E-6, 1E-7 and 1E-8 and find which is closest to 4.75. That is the best this machine can do with a one-sided chord, and now you know the number.

Extrema by search

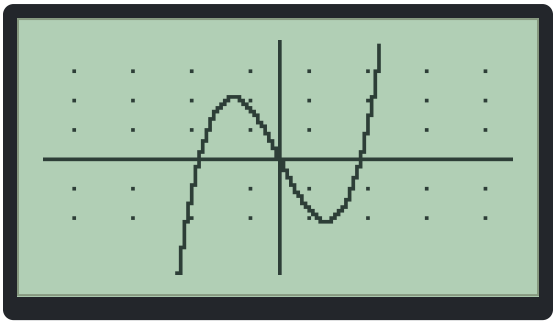
4.4

Where a smooth curve turns, its slope passes through zero. Finding the turning points is the first genuinely useful thing calculus sells, and the machine has four different ways to do it, which disagree in the last few digits for reasons worth understanding.

The specimen is built to be checkable: f(x) = x^3/3 - 4x, whose derivative x squared take away 4 vanishes at -2 and at 2. On paper the hill is at -2 with height 16/3, the valley at 2 with height -16/3.

1. Press [CLEAR], type [x-VAR], [^], [3], [÷], [3], [-], [4], [×], [x-VAR] so the line reads

`X^3/3−4*X`, press **GRAPH**, and let the plot finish.



The designed cubic with turning points at -2 and 2

2. Press **F3**, the maximum search. It sweeps the window and takes a few seconds, so let it work. It publishes `= −1.9997326856359`.

   Two things about that. First, it is the *location* of the maximum, not its value; the searches tell you where, and `EVAL(` tells you what. Second, it is not -2. It is -2 to about three decimal places and then it drifts.

   That drift is not a mistake. These searches close a bracket around the turning point and stop when the bracket is tight enough, not when the digits are exact.

   Near a smooth maximum the curve is almost flat, so an enormous range of x values all look equally like the top. Being flat is what makes a maximum a maximum, and it is exactly what makes it hard to locate precisely. Expect three or four good digits from these keys and do not go hunting for more.

3. The result screen left `X^3/3−4*X` on the entry line, so pressing **GRAPH** stores it back unchanged and replots.

   Mind that rule, because it bites. **GRAPH** always stores the entry line into the active slot, and storing an *empty* line clears the slot. Go back to the graph with the equation on the line, never from a blank one.

   Press **F2**, the minimum search, and let it settle: `= 1.9997326856359`. The valley mirrors the hill, digit for digit, as the symmetry of the function demands.

4. The home-screen commands take typed bounds instead of the window, which means you choose the search interval rather than inheriting it. Press **CLEAR**,

spell FMIN(0,4), press ENTER, and let it work: = 1.9998801765763. Press CLEAR and ask FMAX(−4,0), using (-) for the sign: = −1.9998801765763.

The same turning points, and different last digits from step 2's. A different interval means a different sequence of brackets and a different place to stop. Neither answer is more correct than the other; both are about as correct as this kind of search gets.

5. Values come from EVAL(at the locations, or at the exact ones when you know them. Press CLEAR and ask EVAL(2): = −5.3333333333333. Press CLEAR and ask EVAL(−2): = 5.3333333333333. Those are the fourteen-digit faces of -16/3 and 16/3.
6. One piece of small print, because it will save you a wasted experiment. Press 2nd CLEAR, the TOLER key: a TOLERANCE CHANGED notice confirms a cycle from 1E−6 to 1E−8, and CLEAR dismisses it. Two more presses bring it back to 1E−6.

   The root hunts of this chapter test their residuals against that setting. The extremum searches do not consult it at all: run step 2 again at any tolerance and the digits do not move. So tightening TOLER to sharpen a maximum is effort spent on nothing, and now you know before you spend it.

TRY IT

1. Before pressing anything, say where the two minima of X^2∗(X^2−4)/4 must be, using nothing but the symmetry of the formula. Then find them with FMIN(and suitable bounds and see how close you were.
2. On this section's cubic, ask FMAX(0,4). There is no turning maximum inside those bounds. Predict what the search will report before you run it, then run it, then compare EVAL(at the answer with EVAL(−2).
3. Find the cubic's maximum from the graph screen after two presses of +, and compare the digits with step 2's whole-window answer. Which window came closer to -2, and can you say why before you look?
4. Step 2 said a maximum is hard to locate precisely because the curve is flat there. Test that: compute EVAL(−2) and EVAL(−1.99) and see how little the height changes for a hundredth of movement in x. Now do the same either side of x = 0, where the curve is steep.

4.5 The definite integral as an average

Most books introduce the integral as an area. This one does it as an average, and then shows you the area afterwards, because the average is the idea that survives contact with reality and the area is the one that needs apologising for.

Here is the question. A quantity varies over an interval. What is its average value?

If it takes only a few values you add them up and divide. If it varies continuously there is nothing to count, and you need a different move: add up the whole of it, then divide by the width of the interval you added it over. Adding up the whole of a continuously varying quantity is exactly what the integral does, so the average is the integral divided by the width. That is the definition and there is nothing more to it.

A day's temperature

A harbour weather logger records a day that runs from 8 degrees at midnight to 20 at noon and back down. Modelled for this chapter as `14-6*COS(PI*X/12)`, with `X` in hours from midnight.

1. Press **CLEAR**, type it (**COS** supplies `COS(`, and the π legend on **2nd** **^** types `PI`), press **GRAPH**, and let the plot finish.

   The standard window shows you almost nothing: only the cold arc around midnight fits, and most of the day sits above `YMAX`. That does not matter here, because everything that follows reads the stored equation rather than the picture. It is worth noticing all the same, because it is easy to assume a command is looking at what you are looking at.

2. Check the model does what it claims before you trust it with anything. Press **EXIT**, press **CLEAR**, and ask `EVAL(0): = 8`. Press **CLEAR** and ask `EVAL(12): = 20.000025006855`.

   Midnight is 8 and noon is 20, near enough. That trailing dust is the machine's fourteen-digit `PI` and not the weather. You will see it in every answer in this section, and once you know what it is you can stop looking at it.

3. Now guess. Before you press another key, write down what you think the day's average temperature is. It is worth doing properly, because the answer is prettier if you have committed to something first.

4. Press **CLEAR** and ask `FNINT(0,24): = 336.000035432`.

That is degree-hours, which is not a unit anybody wants. Press **CLEAR** and divide it by the width of the day, `FNINT(0,24)/24: = 14.000001476333`.

Fourteen degrees, exactly halfway between the 8 at midnight and the 20 at noon.

If that is what you guessed, you guessed it because the cosine spends as much time above its centre line as below it, so over a whole period the two cancel exactly. The `.000001476` on the end is `PI` again.

5. Two things worth checking, since they cost one key each.

   Press **CLEAR** and ask `FNINT(0,12)/12: = 14.000001475932`. Press **CLEAR** and ask `FNINT(12,24)/12: = 14.000001475926`.

   The morning averages 14 and the afternoon averages 14, which is not obvious and is a consequence of the symmetry rather than of the arithmetic.

6. And one that is genuinely surprising the first time. Press **CLEAR** and ask `EVAL(6): = 14.00000000039`.

   At six in the morning the temperature *is* the day's average. Not near it, not roughly it: the curve actually passes through its own mean value, and it does so twice, at 6am and again at 6pm.

   That is not a coincidence about cosines. A continuous quantity that spends part of its time above its average and part below it has to cross the average on the way, and it is the whole content of the mean value theorem for integrals. Your model just handed it to you.

And now the area

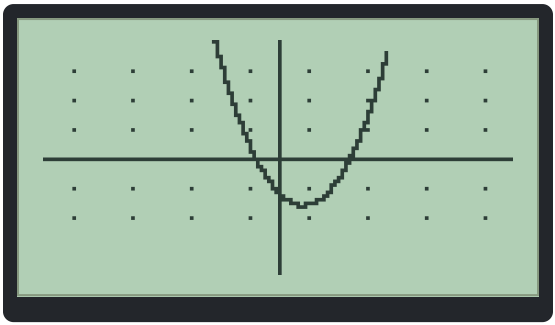
The average is the honest idea. The area is the one everybody teaches first, and it comes with a wrinkle.

If you multiply the average back by the width you recover the integral, and if you draw a rectangle of that height across that width, it has the same area as the region under the curve. So the integral *is* an area. It is the area of the rectangle that would do the same job as the curve.

The wrinkle arrives the moment the curve goes below the axis.

7. The specimen dips on purpose: g(x) = x^2 - 2x - 3 factors as (x - 3)(x + 1), so it is negative between -1 and 3 and positive outside.

Press CLEAR, type x-VAR, x², -, 2, ×, x-VAR, -, 3 so the line reads `X^2−2∗X−3`, press GRAPH, and let it finish:



The parabola dipping below the axis between -1 and 3

8. Press F5, the integral key, and let it work: `= 606.66666666667`. The window is the interval, so that is the integral from -10 to 10, and on paper it is 1820/3.
9. Typed bounds are the home command's job. Press CLEAR and spell `FNINT(−1,3): = −10.666666666667`.

   The dip between the zeros encloses an area of 32/3, and the integral reports it *negative*. Below the axis, the count subtracts.
10. Press CLEAR and ask `FNINT(3,5): = 10.666666666667`. By a designed coincidence, the hump from 3 to 5 encloses exactly as much above the axis as the dip does below.
11. So the whole run should cancel. Press CLEAR and ask `FNINT(−1,5): = 0`, exactly.

    An integral of zero does not mean nothing happened. It means the ups and the downs balanced, which is an entirely sensible thing for an average to say and a slightly mad thing for an area to say. This is why the average is the better story: the average of that stretch really is zero, while "the area is zero" needs the word *signed* smuggled in front of it to be true at all.

    When the question is how much area regardless of side, integrate the pieces separately and add their sizes.

TRY IT

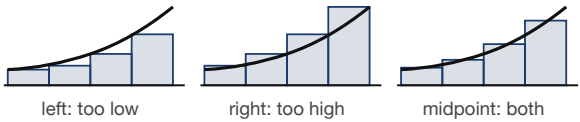
1. An island town's temperature swings just 3 degrees either side of 17. Write the model in the pattern of step 1, predict its 24-hour average before you press a key, and then confirm it with `FNINT(`.
2. Find the two times of day at which the harbour model reaches its own average, by hand from the formula rather than by searching. Then check both with `EVAL(`.
3. Check the splitting rule on the parabola: compute `FNINT(-1,1)` and `FNINT(1,3)` and confirm they add up to step 9's answer for the whole dip.
4. Work out the -5 to 5 integral of the parabola on paper. Then press [+] once on the plot and use [F5] on the halved window, and compare.
5. What is the average value of `X^2-2*X-3` over -1 to 3? Compute it from step 9, then find where the curve takes that value, and check that your answer sits inside the interval as the mean value theorem promises.

Riemann sums by program

4.6

`FNINT(` answers in a second and tells you nothing about how. A Riemann sum is the how: cut the interval into slices, guess each slice's area from one sample of the height, add up the guesses. Watching those sums close in on the integral is the best argument there is for why a limit of sums deserves to be called an integral at all.

The function is f(x) = x ^ 2 + 1 on the interval 0 to 2, whose integral is 14/3.



Left, right and midpoint rectangles over the same curve, one undershooting, one overshooting, one splitting the difference

1. Store the equation first, because every program below reads it. Press [CLEAR], type [x-VAR], [x²], [+], [1] so the line reads `X^2+1`, press [GRAPH], let the plot

finish, press [EXIT], and press [CLEAR].

Then get the target: spell `FNINT(0,2)` and press [ENTER]: `= 4.6666666666667`, the fourteen-digit face of 14/3.

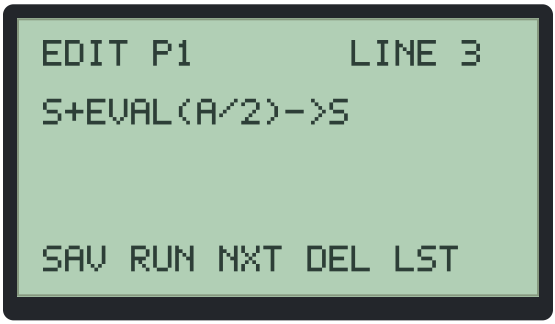
The left sum

With four slices the width is $2/4 = 0.5$, and the left edges are at 0, 0.5, 1 and 1.5. That is `A/2` for `A` counting from 0 to 3.

2. Press [PRGM], then [F1], `NEW`. The editor opens on `EDIT P1`. Type these six lines, pressing [ENTER] after each. Letters are [ALPHA] plus the key carrying the letter, a space is [2nd] [0] in this editor, and [STO▶] types the –> arrow.

Line	Text	Keys
1	`0->S`	[0] [STO▶] [S]
2	`FOR A,0,3`	[F] [O] [R] [2nd] [0] [A] [,] [0] [,] [3]
3	`S+EVAL(A/2)->S`	[S] [+] [E] [V] [A] [L] [(] [A] [÷] [2] [)] [STO▶] [S]
4	`END`	[E] [N] [D]
5	`DISP S/2`	[D] [I] [S] [P] [2nd] [0] [S] [÷] [2]
6	`STOP`	[S] [T] [O] [P]

One thing about the editor that nobody warns you about: it shows you a single line at a time. There is no listing on screen, no scrolling view of the program, just the line you are standing on and its number in the corner. Press [▲] four times when you have finished typing and you land back on line 3:



The editor standing on line 3, showing one line and its number

That is worth knowing before you start, because it means you cannot see your program. You have to hold it in your head or on paper, and check it by walking ▲ and ▼ through the lines one at a time. With eight lines that is livable. It is also why the listings in this book are printed as tables: the table is the view the machine will not give you.

Line 3 is the workhorse. `EVAL(` reads whichever equation is stored, so the program never contains the function and will happily measure any equation you store later. Line 5 multiplies the tally of heights by the slice width: four heights halved are four half-width slices.

3. Press **F2**, `RUN`. The run screen answers `RUN P1` over `LINE 6`, the output line shows `3.75`, and the status reads `DONE`.

   Well under 4.6667, and it had to be. This function rises across the whole interval, so taking each slice's height from its left edge takes the lowest height in the slice every time.

The right sum, and a relation you can check exactly

4. The right sum samples 0.5, 1, 1.5 and 2 instead, which is the same program counting `A` from 1 to 4. Press **PRGM** for the list, press ▼ to select the second slot, and press **F1** to open `EDIT P2`. Type the same six lines with line 2 as `FOR A,1,4`.

   Press **F2**: `5.75`.

   So the true answer is now bracketed. Somewhere between 3.75 and 5.75, and we already know it is 4.6667.

5. Here is something better than a bracket. Press **PRGM**, press **EXIT** for the home screen, press **CLEAR**, and type `5.75−3.75: = 2`.

   That 2 is not luck and you can predict it without running anything. Going from left sampling to right sampling swaps one height in and one height out: you lose f at the left end of the interval and gain f at the right end, each weighted by one slice width. So the difference is exactly (f(2) - f(0)) times the slice width, which is (5 - 1) times 0.5, which is

   2.

   That relation holds for any rising function and any number of slices, and it tells you something useful: double the slices and the gap between the two sums halves, because the slice width halves. Both sums are therefore converging, and doing so at the same unhurried rate.

Midpoint, and why it is so much better

6. The midpoint sum samples the slice centres 0.25, 0.75, 1.25 and 1.75, which is `(2*A-1)/4` for `A` from 1 to 4. Press **PRGM**, press **▼**, press **F1** for `EDIT P3`, and type the variant with line 2 as `FOR A,1,4` and line 3 as `S+EVAL((2*A-1)/4)->S`.

   Press **F2**: `4.625`. Inside the bracket, and only 1/24 short.

7. Double the slicing. Press **PRGM**, press **▼**, press **F1** for `EDIT P4`, and type the eight-slice midpoint sum: line 2 becomes `FOR A,1,8`, line 3 becomes `S+EVAL((2*A-1)/8)->S`, and line 5 becomes `DISP S/4`.

   Press **F2**:

The eight-slice midpoint sum closing in on 14/3

   `4.65625`.

   Put the errors side by side. Press **PRGM**, **EXIT**, **CLEAR**, and compute each against the target, **CLEAR** between them:

Estimate	Value	Error
left, 4 slices	`3.75`	`-0.9166666666667`
right, 4 slices	`5.75`	`1.0833333333333`
midpoint, 4 slices	`4.625`	`-0.0416666666667`
midpoint, 8 slices	`4.65625`	`-0.0104166666667`

   The midpoint error fell from 1/24 to 1/96 when the slice count doubled. Quartered, where the left and right sums only halve. That is the midpoint rule's signature and the reason nobody uses left sums for anything except explaining what a Riemann sum is.

Two estimates for free

You now have four numbers on paper and no more programs to write. Two more rules fall straight out of them.

8. The trapezoid rule joins the tops of the slices with straight lines instead of flat ones, and it turns out to be exactly the average of the left and right sums. Press **CLEAR** and type `(3.75+5.75)/2:` `= 4.75`.

   Out by 1/12, which is twice the midpoint's error and on the other side. That is worth pausing on: the trapezoid, which looks like the more sophisticated idea, is beaten by the midpoint rule, which looks like the cruder one. A trapezoid cuts the corner of a curve that bends upwards and so overshoots; a midpoint rectangle is too low on one half of its slice and too high on the other, and the two errors very nearly cancel.

9. Simpson's rule takes that seriously. If the trapezoid is wrong one way by twice as much as the midpoint is wrong the other way, then two parts midpoint to one part trapezoid should cancel both. Press **CLEAR** and type `(2*4.625+4.75)/3:`

   `= 4.6666666666667`.

   Which is 14/3, to every digit the machine has. Press **CLEAR** and type `14/3` to confirm: `= 4.6666666666667`.

   Not close. Exact. Simpson's rule fits a parabola through each set of three points, and this function *is* a parabola, so there is nothing left to be wrong about. Four slices of a method you assembled on the home screen out of two crude sums have beaten `FNINT(`'s sixty-four panels.

   Do not over-learn that. Simpson is exact on quadratics and cubics and merely very good on everything else. But it does show what these estimates are for: not one of them is the answer, and combining them cleverly is worth more than computing any one of them harder.

The environment shaped the design here and it is worth saying how. When this chapter was written `FOR` bounds were single digits, so a counted loop passed at most ten times, which is why the finer slicings in Chapter 3 hand the count to a `WHILE` countdown instead. Firmware 2.19 gave `FOR` evaluated bounds and an optional step, so that particular wall is gone; the countdown shape is kept here because it is still a good way to see a loop's state, not because anything forces it.

Four program slots held all four sums, and EVAL(kept every one of them ignorant of which function it was measuring.

TRY IT

1. Predict, before running anything, the left sum with eight slices. You know the four-slice answer and you know from step 5 how the left and right sums move. Write your number down, then edit P1 to eight slices (count 0 to 7, sample A/4, display S/4) and see.
2. Check step 5's relation on the eight-slice sums: run left and right at eight slices and confirm their difference is exactly (f(2) - f(0)) times the new slice width.
3. Store a different equation, plot it through, and rerun P4 without editing a single program line. Check its answer against FNINT(with the matching bounds. What did you have to be careful about?
4. Build Simpson from the eight-slice numbers instead of the four-slice ones and see whether it is still exact. Then try the whole thing on X^3+1, which is a cubic, and then on 1/X from 1 to 2, which is neither. Where does exactness stop?
5. The midpoint rule quarters its error per doubling and the trapezoid halves. Work out how many midpoint slices you would need to match FNINT(’s answer to six decimals, and say whether the eight-line slot could ever run it.

4.7 Areas between curves

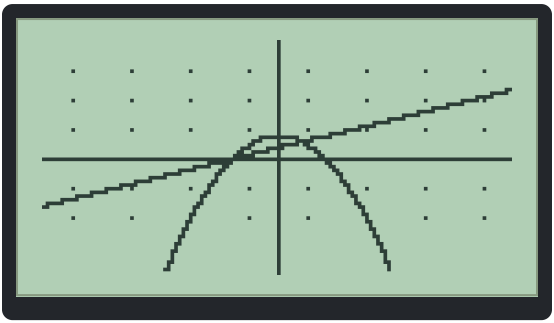
Two curves enclose a region. How much is in it?

The gap between them at each x is the difference of their heights, so the enclosed area is the integral of the difference function taken between the crossing points. Every tool this chapter has built gets a turn, and the order matters: plot the pair, find the crossings, store the difference, integrate it.

The designed pair is an arch and a line, y = 2 - x^2/2 and y = x/2 + 1, which cross where x^2 + x - 2 = 0, at x = -2 and at x = 1.

1. Press [CLEAR], type [2], [-], [x-VAR], [x²], [÷], [2] so the line reads 2−X^2/2, and press [GRAPH]. When the plot finishes, press [2nd] [2] to move to slot Y2, type

x-VAR, ÷, 2, +, 1, and press GRAPH:



The arch and the line crossing at -2 and 1

2. Press 2nd F1, the intersection search, and let it settle: = −1.9999999999999 with the residual line R=1E−13.

   That is the left crossing. The search scans the window from its left edge, so it reports the leftmost intersection it finds and then stops. There is a second crossing on the screen and it will not go looking for it.

3. So make the window exclude the first one. The entry line now holds X/2+1, the active slot's text, so press GRAPH to return to the plot. Press + three times, letting each replot finish, which narrows the view to -1.25 to 1.25. Press 2nd F1 again: = 1 with R=0.

   The right crossing, exactly. Zooming was not cosmetic here: it was how you told the search which root you wanted. The window is an argument you pass to these keys, and this is the clearest case of it in the book.

4. Now the difference. Press GRAPH, then 2nd + for the standard window, and let it replot. Press 2nd 3 for slot Y3, type 1, -, x-VAR, ÷, 2, -, x-VAR, x², ÷, 2 so the line reads 1−X/2−X^2/2, and press GRAPH.

   That is the arch take away the line, collected on paper. It joins the plot as a low hump, positive exactly where the arch is above the line.

5. The hump's zeros ought to be the crossings, and the root search confirms it. Press F1 and let it settle: = −1.9999999999999, the same figure the intersection search gave in step 2.

   Those are the same question asked twice. Where do two curves meet, and where does their difference vanish, are one question wearing two hats, and

this is worth internalising because the second form is almost always the easier one to compute with.

(F1 reads the active equation, which is now the difference in Y3.)

6. The area. Press CLEAR, spell `FNINT(-2,1)`, and press ENTER: `= 2.25`.

   Nine quarters of area between the arch and the line.

That is the whole workflow, and it is worth keeping: plot the pair, search out the crossings, store the difference, integrate between them.

One trap, and it is section 4.5's sign convention arriving where nobody wants it. The difference was typed with the arch on top, which is the curve that really is on top over this interval, so the integral came out positive. Type it the other way round and the same region gives you -2.25. The machine has no idea which of your two curves you think of as the upper one; it only knows which you subtracted from which.

TRY IT

1. Retype Y3 as the line minus the arch, `X/2+1-(2-X^2/2)`, replot, and integrate from -2 to 1 again. Predict the answer before you press ENTER. What changed and what did not?
2. Replace Y2 with `X/2` and find the new crossings with 2nd F1 and a zoom. The exact answers are no longer whole numbers. What do the residual lines report instead, and what does that tell you about how hard the search had to work?
3. Extend the integral to `FNINT(-2,4)`. The line crosses back over the arch at x = 1, so say what that answer mixes together before you compute it. Then work out how to get the total enclosed area on both sides instead.
4. Design your own pair that crosses at -1 and 3, using the same method: pick the crossings first, build a difference that vanishes there, then split it into two curves. Check your crossings with 2nd F1.

CHAPTER

5

Explorations in Calculus II

The second course in calculus widens the field of play.

Equations whose roots have to be hunted rather than factored. Curves that refuse to be the graph of any function. Motion through time. Functions built out of integrals. Polynomials impersonating transcendental functions, and getting away with it over a range you can measure.

Free85 keeps a tool for each: the polynomial editor and solver workspace of the Guidebook, chapter 14, the polar and parametric modes of chapters 5 and 6, and the calculus commands of chapter 3.

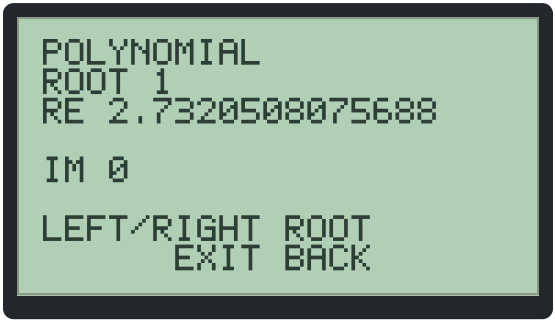
The habits from Chapter 4 still govern. The calculus commands read the active stored equation, so store with GRAPH before asking them anything. Let plots draw to the end, because presses arriving mid-draw are dropped. And press CLEAR at home before typing a command, because the graph hands its equation back to the entry line.

Zeros of functions two ways 5.1

Factoring finds the roots the algebra teacher chose. Most polynomials met in the wild need a hunter, and Free85 keeps two: the polynomial editor, which answers every root at once, and the solver workspace, which hunts one root of any equation whatever.

The specimen is designed on paper so you know the answers before you start. Multiplying x^2 - 2 by x^2 - 2x - 2 gives the quartic x^4 - 2x^3 - 4x^2 + 4x + 4, whose roots are plus and minus the square root of 2, and 1 plus or minus the square root of 3.

1. Press [2nd] [PRGM], the POLY legend, and the polynomial editor opens on a fresh DEGREE 2. Press [F4], QRT, for degree 4, and type the coefficients highest power first, [ENTER] after each: [1] [ENTER], [(-)] [2] [ENTER], [(-)] [4] [ENTER], [4] [ENTER], [4] [ENTER]. The COEFF line steps down a power per entry.
2. Press [F1], SOLV, and give the search a few seconds. The root browser replaces the editor:



The quartic's root browser opening on 1 plus root 3

ROOT 1 shows RE 2.7320508075688 with IM 0, which is 1 plus root 3, one digit of dust short of the paper value.

Press [▶] three times for the rest: ROOT 2 is RE −0.73205080756887, ROOT 3 is RE −1.4142135623731, and ROOT 4 is RE 1.4142135623731, every IM line reading 0.

Four roots, one press, no guessing and no bounds. That is the editor's whole appeal and it is worth using whenever the thing in front of you really is a polynomial.

3. Now the same roots one at a time, by hunting. Press [EXIT] for the home screen, type X^4−2*X^3−4*X^2+4*X+4 (the [x²] key types ^2), and press [2nd] [GRAPH]: the solver workspace opens with the equation stored, the F= line clipping at the screen's edge with the tail kept.

   Press [F1], SOLV, and let it work: a ROOT of −1.4142134785654 with RES −6.704614E−7.

   From the fresh guess 0 and bounds -10 to 10, the scan stops at the first sign change it meets coming from the left, which is minus root 2. It found *a* root, not *the* root, and it had no way of knowing you wanted a different one.

4. The bounds are the fence that picks a root, and this is the thing to learn here. Press F5, the > key, three times to reach the LOWER page, and store bounds 0 and 2: 0 ENTER, 2 ENTER. SOLV answers a ROOT of 1.4142136573793.

   Re-fence at 2 and 5 the same way and SOLV answers 2.7320508360865 with RES 5.39787E−7, which is the browser's first root re-found by hunt.

   The guess is tried before any scanning, so it picks roots too. Page to GUESS, type the browser's own 1.4142135623731, press ENTER, and SOLV answers it straight back with RES 0. Handing a root hunter the root is not cheating; it is how you check that it agrees with you.

5. What the editor cannot do is leave polynomials. Press EXIT and CLEAR, type COS(X)−X, and press 2nd GRAPH: SOLV answers a ROOT of 0.7390856742858, the one crossing of cosine and the line, and a number no polynomial tool can reach.

So: POLY for polynomials of degree 2 to 4, all roots at once with no guessing. The solver for everything else, one root per hunt, steered by guess and bounds. Neither is a substitute for the other and the choice is usually obvious once you have both in mind.

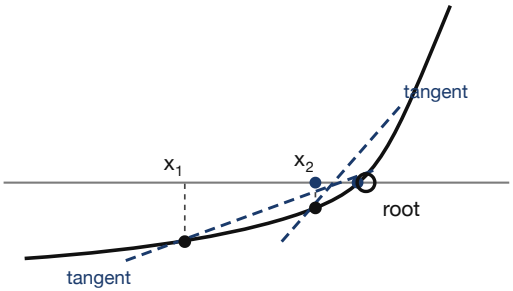
TRY IT

1. Design your own quartic by multiplying two quadratics on paper, expand it, and let POLY recover the roots you built in. How many digits do you get back?
2. Give POLY the quadratic x ^ 2 + 2x + 3, then give the solver the same equation. One answers a conjugate pair, the other stops at a notice. Predict both before you try, and say what each tool is telling you.
3. Aim the solver at the quartic with bounds -5 and 0 and guess 3. Which negative root does it report, and how do you fence in the other?
4. The residual RES was −6.704614E−7 in step 3 and 0 in step 4. What is the residual actually measuring, and why should handing it the exact root produce zero?

Newton's method 5.2

The solver of section 5.1 hunts roots and does not tell you how. This section builds the method that most root hunters are made of, in four useful lines, and then breaks it.

The idea is one you already have from Chapter 4. At any point on a curve you can compute the tangent. A tangent is a straight line, and finding where a straight line crosses the axis is arithmetic. So: stand somewhere, follow the tangent down to the axis, and stand there instead. Repeat.



A curve with a tangent drawn at the first guess, running down to the axis at a point much nearer the root, and a second tangent from there landing nearer still

If x is the current guess, the tangent crosses the axis at x minus f(x) over f'(x), and that is the whole method.

The specimen is x ^ 3 - 2x - 5, an old favourite with exactly one real root, near 2.09.

Four lines that do it

1. Store the function. Press **CLEAR**, type **x-VAR** **^** **3** **-** **2** **×** **x-VAR** **-** **5**, press **GRAPH**, and let the plot finish. Press **EXIT** and **CLEAR**.
2. Press **PRGM** and **F1**, `NEW`, opening `EDIT P1`. Type these eight lines:

Line	Text	Keys
1	2–>R	2 STO▶ R
2	1–>N	1 STO▶ N
3	WHILE N	W H I L E 2nd 0 N
4	R–EVAL(R)/NDER(R)–>R	R - E V A L (R) ÷ N D E R (R) STO▶ R
5	N–1–>N	N - 1 STO▶ N
6	END	E N D
7	DISP R	D I S P 2nd 0 R
8	STOP	S T O P

Line 4 is Newton's method entire, in twenty characters. `EVAL(` gives the height and `NDER(` gives the slope, both reading whichever equation is stored, so this program will hunt the root of anything you care to put in the slot. It contains no function at all.

Line 2 is the step count, and you are going to edit it repeatedly, which is how you watch the method work rather than just seeing its answer.

3. Press F2, `RUN`. It takes a moment, because each step evaluates the stored equation twice.

   `2.1`.

   Check that by hand, because it is the only step you will be able to. f(2) is 8 - 4 - 5 = -1, and f'(2) is 12 - 2 = 10, so the tangent crosses at 2 minus -1 over 10, which is 2.1. The machine and the arithmetic agree.

4. Now run it again with more steps, editing line 2 each time. Press PRGM, F1, ▼, CLEAR, type the new count, then F2:

Steps	Iterate
1	2.1
2	2.0945681211042
3	2.0945514816982
4	2.0945514815424
5	2.0945514815424

Newton settled on the root after four steps

Read down the correct digits: 2, then 5, then 10, then all fourteen, and then it stops moving because there is nowhere left to move to.

The number of correct digits roughly *doubles* at every step. That is what quadratic convergence means, and it is why Newton's method is the one everybody reaches for. Compare Chapter 7's Euler, which halved its error per halving of the step, or the bisection the solver uses, which buys one bit a go. This buys everything it already has, again, every time.

Now break it

The doubling comes with conditions, and the conditions matter more than the method. Newton needs a start close enough to the root and a derivative that is not too small, and if it does not get them it does not degrade gracefully. It leaves.

5. Change line 1 to `0->R` and set line 2 back to `1->N`. Zero is a perfectly reasonable-looking place to start: the curve is smooth there and it is only a couple of units from the root.

   Run it, then work up through the step counts as before:

Steps	Iterate
1	−2.5000000125
2	−1.5671641902429
3	−0.5025924653538
4	−3.8207066344471
6	−1.6081115996896
8	−4.5977119699788

It is not converging. It is not even slowly converging. After eight steps it is further from the root than it started, on the wrong side of the axis, and still wandering.

Here is why, and you can see it from the picture rather than the algebra. At x = 0 the slope of this cubic is -2, which is shallow. A shallow tangent runs a long way before it meets the axis, so the first step throws the guess to -2.5, a place with no relation to the root at all. Out there the cubic is nearly flat over a stretch, so the next tangent throws it somewhere else, and the method spends its time bouncing around a region that contains no root whatever.

Newton's method converges beautifully when it converges. It has no opinion at all about whether it will.

6. So give it help, which in practice is what everybody does. Plot the function first and look. Press **PRGM**, **EXIT**, press **GRAPH** and let the cubic draw. The curve crosses the axis once, a little past 2, and the crossing is steep. Start there and the method is safe.

   That is the working method for Newton in practice: use a picture or a coarse search to get close, then let the doubling take you the rest of the way. The solver of section 5.1 does something similar internally, which is why it asks you for bounds.

TRY IT

1. Predict, then check: start Newton at 3 rather than 2. How many steps to fourteen digits, and is it more or fewer than from 2? Why?
2. Start at 0.8, where the cubic's slope is very close to zero. Predict what the first step does before you run it. Then run it and see how far it went.
3. Store `X^2−2` instead and hunt root 2 from a start of 1. Work out on paper what line 4 does to this particular function, and you will find you have rediscovered a very old algorithm for square roots.
4. Put `COS(X)−X` in the slot and run Newton from 1. Compare the answer with section 5.1's solver result of `0.7390856742858`, and compare how long each took.
5. Newton needs `NDER(` at every step, which is two evaluations of the stored equation. Count the evaluations for four Newton steps against the solver's bisection to the same accuracy. Which is actually cheaper here?

5.3 Conic sections by parametric pair

A circle fails the vertical line test, so no function slot can draw one. The parametric mode can, because it plots any pair x(t), y(t) you give it, and a pair has no opinion about whether the result is a function.

The pair A cos t, B sin t sweeps an ellipse with half-width A and half-height B. The story is a garden design: an ornamental pond 10 metres by 5 at one metre per unit, so A is 5 and B is 2.5.

1. Switch modes. On the graph screen press [2nd] [MORE], then [MORE] twice to the GRAPH MODE page, press [F3] for parametric, and [EXIT].

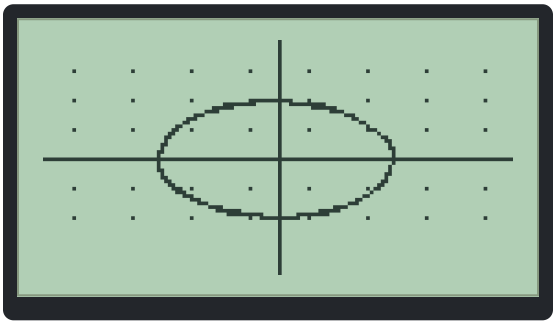
   Slot 1 is x(t), slot 2 is y(t), and [x-VAR] types the parameter, shown as X.

   Type [5] [×] [COS] [x-VAR] [)] so the line reads 5*COS(X), and press [GRAPH]. Nothing draws, and that is correct: a pair needs both slots.

   Press [2nd] [2], type 2.5*SIN(X), and press [GRAPH] again. Both slots evaluate at every sample, which roughly doubles the plot time, so let it run to the end.

   The pond appears squashed. The pixels are not square.

2. Press [2nd] [-], the square window, and let the replot finish:



The 10 by 5 pond ellipse in the square window

   Now one unit is the same length on both axes and the ellipse shows its true proportions, twice as wide as tall. Any time a circle looks like an ellipse or an ellipse looks wrong, this is the key to reach for.

3. Press [▶] once and the readout gives X=4.861147193253 and Y=0.58507434688468, a rim point one sample past the sweep's centre.

Put it on trial. The rim's equation says (x/5) ^ 2 + (y/2.5) ^ 2 must be 1, so press **EXIT**, press **CLEAR**, type `(4.861147193253/5)^2+(.58507434688468/2.5)^2`, and press **ENTER**.

`= 1`, exactly. The traced point sits on the designed ellipse to all fourteen digits, which is a stronger check than it looks: it says the trace readout and the plot are computing the same thing.

4. Where did the sweep come from? Parametric mode has no angle settings at all. The parameter t runs from `XMIN` to `XMAX` in 128 samples, so the window doubles as the parameter range.

   The standard window sweeps t from -10 to 10, which is over three revolutions of the rim, and the square window kept those bounds. A window narrower than one full turn leaves the rim partly drawn, which is a surprise the first time and obvious once you know where t comes from.

The mode holds one pair, so a pond and a path around it are two plots. The design scales freely: a 12 by 6 pond, stored as `6*COS(X)` and `3*SIN(X)`, draws its whole rim including the right-hand vertex.

TRY IT

1. Retype the pair as a fountain basin 8 metres across and 8 deep, plot it in the square window, and check a traced point against the circle's equation the way step 3 did.
2. Swap the pond's pair, `2.5*COS(X)` and `5*SIN(X)`. Predict the picture before the plot finishes.
3. From the square window press **+** twice, letting each replot settle, and explain what changed. Remember that the window is also the sweep.
4. Work out what pair draws a circle of radius 3 centred at (4, 1), then plot it and trace a point to check.

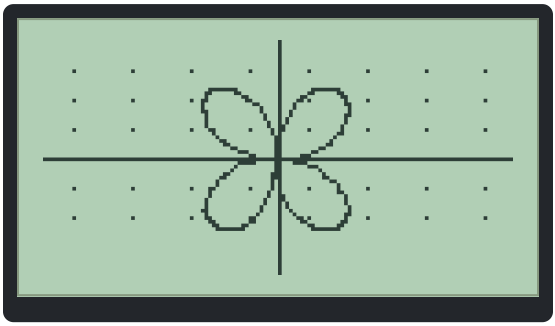
Polar curves

5.4

Some curves are unwieldy in x and y and a single line in polar form, where each point is named by its distance r from the origin at angle theta.

Free85's polar mode stores r as a function of the angle, typed with **x-VAR**, and always sweeps exactly one revolution, 0 to 2 pi in `RAD` mode, in 128 samples, whatever the window shows. That last clause is the one to remember.

1. On the graph screen press [2nd] [MORE], then [MORE] twice, then [F2] for polar mode, and [EXIT]. Type [4] [×] [SIN] [2] [x-VAR] [)] so the entry line reads `4*SIN(2X)`, and press [GRAPH]. Trigonometry makes this a slow plot, so let it sweep to the end. Then press [2nd] [-] for the square window and let the replot finish:



*The four-petal rose of 4*SIN(2X)*

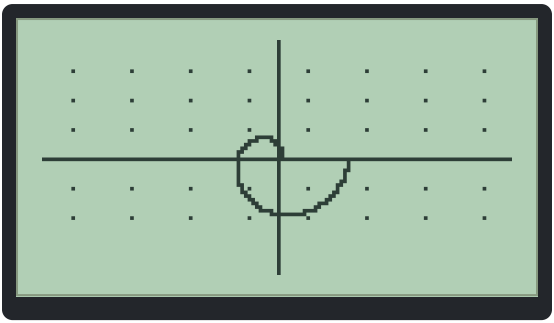
   Doubling the angle folds the revolution into four petals on the diagonals. Work out why four and not two: the radius goes negative for half of each cycle, and a negative radius plots opposite.

2. The cardioid. Press [EXIT], press [CLEAR], type `2.5*(1+COS(X))`, and press [GRAPH]: a heart lying on its side, cusp at the origin, in the kept square window.

   The design says the radius is 5 at angle 0 and 0 at angle pi. Press [EXIT], press [CLEAR], and ask `EVAL(0): = 5`. Ask `EVAL(PI)` (the π legend on [2nd] [^]): `= -0.000010419523`.

   Not zero. That is the cusp sitting under the machine's fourteen-digit `PI`, which is not quite pi, so the cosine is not quite -1. The same small print as `SIN(PI/2)` in the Guidebook, chapter 3, and worth recognising rather than worrying about.

3. The spiral. Press **CLEAR**, type `X/2`, and press **GRAPH**:



The spiral X/2 stopping after one revolution

The radius grows with the angle, half a unit per radian, and the curve winds outward until the single revolution is spent, stopping mid-air at radius pi on the positive x axis.

That abrupt end is the boundary. The sweep is one revolution, always, so a spiral of many turns is beyond the mode. What there is of it is exact, and the stopping point is not a bug to work around but the mode telling you where its world ends.

4. Read the spiral's equation off the screen, which is the best trick in this section. Press ▶ once: the readout shows `X=-1.6034808029742` and `Y=-0.1192134330108`, the Cartesian point just past the sweep's centre.

   Now press **2nd** **MORE**, then **MORE** twice, and press **F5**, the coordinate toggle, flipping the mode page's `GRAPH COORD` line from `RECT` to `POLAR`. Press **EXIT** and let the replot run to the end before touching the arrows.

   Press ▶ once: the same position now reads `X=1.6079017518373` and `Y=3.2158035036746`. That is the radius in the `X=` line and the angle in the `Y=` line, with the labels unchanged, which is confusing until you have met it once.

   The radius is exactly half the angle, to all fourteen digits. The trace has recited `r = theta/2`, which is the equation the slot was given.

TRY IT

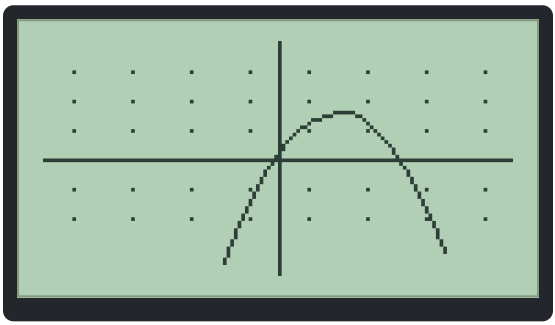
1. Replot the rose as `4*SIN(3X)` and count petals. Predict the number first. What does tripling the angle do that doubling did not?
2. Predict which way the cardioid `2.5*(1-COS(X))` faces, plot it, and confirm its cusp with `EVAL(` at the angle where you expect it.
3. With the polar readout still on, trace the spiral onward and check the half-the-angle rule at each stop.
4. Work out what `4*SIN(2X)` does for theta between pi/2 and pi, where the radius is negative, and check your reasoning against which petal gets drawn when.

5.5 Parametric motion

A parametric pair is a motion, not just a shape. The parameter is time, and the trace key becomes a slow-motion replay.

The projectile is a pebble from a garden sling, launched from level ground at 3 metres per second horizontally and 9 vertically, with gravity rounded to 10: x(t) = 3t and y(t) = 9t - 5t^2. On paper the pebble lands when y is 0 again, at t = 1.8 seconds, 5.4 metres out. Work that out before you plot it.

1. In parametric mode, store the motion: `3*X` in slot 1, then [2nd] [2] and `9*X-5*X^2` in slot 2. Let the plot finish:



*The pebble's flight from 3X and 9X-5*X^2*

   The arc rises from the origin and returns to the axis.

   The tail diving off the lower left is not a mistake and it is worth understanding rather than ignoring. The standard window sweeps t from -10 to 10, so the machine is also plotting the model's pre-launch fiction: where the pebble

would have been before you threw it, if the formula had applied. The window is the time range and it does not know when the story starts.

2. Trace is time. Each press is one sample of t, about 0.16 seconds. Press [▶] five times, one press at a time: `X=2.598425196849` and `Y=4.044268088536`, the pebble near the top of its arc.

   Continue to the tenth press, `X=4.960629921261` and `Y=1.210862421722`, and the eleventh, `X=5.433070866141` and `Y=−0.099820199638`.

   Between those two samples the `Y=` line changes sign, so the pebble lands between 4.96 and 5.43 metres out. That is as much as trace can tell you, and finer answers need finer tools.

3. The apex, exactly. The analysis keys read the active slot as a function of t, and slot 2, stored last, is active.

   Press [F1], the root hunt: it answers `= 3E−21` with `R=2.7E−20`. That is the launch, not the landing, because the search scans from the window's left edge and y is zero at t = 0 too. A perfectly correct answer to a question you did not mean to ask.

   Press [CLEAR] and ask `FMAX(0,2)` instead: `= 0.8999585312886`, the apex time, a search's whisker under the paper answer 0.9. Press [CLEAR] and ask `EVAL(.9): = 4.05`, the apex height in metres.

4. The landing, exactly, from the solver. Press [CLEAR], type `9∗X−5∗X^2`, and press [2nd] [GRAPH].

   The fresh guess 0 already solves the equation and would hand back the launch again, so move it: press [F5] twice to the `GUESS` page, store 2, then bounds 1 and 3. `SOLV` answers a `ROOT` of `1.7999999523165`.

   The flight lasts 1.8 seconds. Press [EXIT], press [CLEAR], and type `3∗1.8: = 5.4` metres, exactly the range the paper predicted.

Here too there is only one pair, so two pebbles cannot fly together. And zooming reframes time as well as space: the window is the clock.

TRY IT

1. Sling a pebble at 2 metres per second horizontally and 12 vertically. Work out the apex and landing on paper first, then find them with the tools of steps 3 and 4.
2. The apex time 0.9 is exact on paper. Why does `FMAX(` stop a whisker short? Chapter 4's extremum searches told the story.
3. Launch from a 2-metre wall at 2 metres per second horizontally: the pair is `2*X` and `2+9*X-5*X^2`. Where does the trace say the pebble lands, and what guess and bounds make the solver agree?
4. Step 3's root hunt found the launch rather than the landing. Find a window that makes it find the landing instead, without using the solver.

5.6 Functions defined by integrals

An integral with a variable upper limit is a function. Call it A(x): the area accumulated under a curve from a fixed start out to x.

The machine has no accumulator button, but `FNINT(` probed at several different upper limits is exactly that function, one value per call. The second specimen below is chosen so that the accumulator turns out to be somebody you already know.

1. Accumulate under f(t) = 2t first, where you can check every answer in your head. Type [2] [×] [x-VAR], press [GRAPH], let the plot finish, press [EXIT], then [CLEAR].

   Spell `FNINT(0,1)` and press [ENTER]: `= 1`. Pressing [CLEAR] before each, ask `FNINT(0,2)`, `FNINT(0,3)` and `FNINT(0,2.5)`: the answers are `= 4`, `= 9` and `= 6.25`.

   The pattern names itself. The accumulator of 2t is x squared, which is the fundamental theorem of calculus arriving with no fuss at all.

2. Now the interesting curve. Press [CLEAR], type `1/X`, press [GRAPH], and let the plot finish.

   The accumulator has to start at 1 rather than 0, because 1/t has no area to speak of near zero. Press [EXIT], press [CLEAR], and ask `FNINT(1,2)`: `= 0.6931471824209`.

3. That number has a famous face. Press **CLEAR** and ask `LN(2)`: `= 0.69314718056122`.

   The area probe matches the logarithm to eight decimals, the difference being `FNINT('s` sixty-four panels. So the accumulator of 1/t is the natural logarithm, which is a much better definition of the logarithm than the one you were probably given.

4. Watch the logarithm's law appear out of the geometry. Pressing **CLEAR** between probes, ask `FNINT(1,4)` and `FNINT(1,8)`: `= 1.3862945205897` and `= 2.0794461816072`.

   Each doubling of the upper limit added the same 0.6931 again. Equal ratios accumulate equal areas, which is the entire personality of a logarithm, and here it is a fact about slabs under a hyperbola.

5. The sharpest form of that claim: the area from 3 to 6 should equal the area from 1 to 2, both being doublings, even though the two slabs look nothing alike. Press **CLEAR** and ask `FNINT(3,6)`:

The area from 3 to 6 matching the area from 1 to 2

   `= 0.69314718242103`, agreeing with step 2's probe to eleven decimal places. Two differently shaped slabs, equal because both are doublings.

And a way to compute pi

6. One more accumulator, because it produces something worth having. The derivative of the arctangent is 1 over 1 plus x squared, so the area under that curve from 0 to 1 is the arctangent of 1, which is pi over four.

   Press **CLEAR**, type `1/(1+X^2)`, press **GRAPH**, let it finish, press **EXIT**, and press **CLEAR**. Type **4** **×** and spell `FNINT(0,1)`, then press **ENTER**:

   `= 3.1415926535863`.

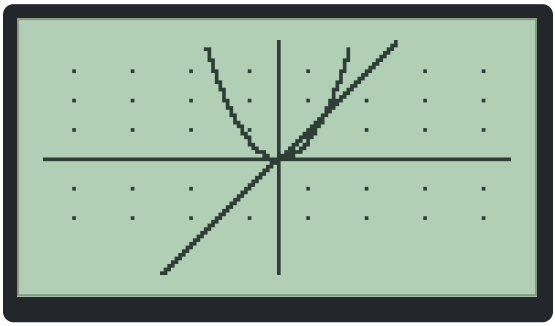
Press **CLEAR** and type **2nd** **^** for the machine's own `PI: = 3.1415926535898`.

Eleven places, from an area. That is worth pausing on: you have just computed pi without a single trigonometric function, out of nothing but a rational function and a sheaf of rectangles.

One route used to be closed and is now open, and the reason it was closed is the reason the open one looks the way it does.

Called as `FNINT(a,b)`, the command integrates whichever equation is active. A slot holding *that* form is being asked to integrate itself, and the machine says so rather than looping: store `FNINT(0,X)` as `Y2` beside `2*X` in `Y1`, press **GRAPH**, and `Y2` stops with `RECURSION ERROR` while `Y1` draws its line exactly as it would alone.

Name the slot and the ambiguity disappears. `FNINT(1,0,X)` reads "integrate slot 1, from 0 to x", and nothing in it refers to the slot doing the asking. Put `2*X` in `Y1` and `FNINT(1,0,X)` in `Y2` and press **GRAPH**:



*The accumulator of 2*X plotted beside it*

`Y1` draws its line and `Y2` draws x^2, which is what the accumulator of 2x is. The table carries both columns, so the function you were building by hand a page ago is now a column you can read down.

Be patient with it. Every plotted column of `Y2` is a complete numerical integration starting from 0, so that slot costs 127 integrals where an ordinary slot costs 127 evaluations.

The rest of the calculus commands take a slot the same way: `EVAL(slot,x)`, `NDER(slot,x)`, `FMIN(slot,a,b)`, `FMAX(slot,a,b)`, `ARC(slot,a,b)` and `INTER(slot,a,b)`. One nested evaluation is available, so a slot may read another slot; two slots that read each other stop with `RECURSION ERROR` while the slots around them carry on.

None of which makes the hand-built table wasted work. Building the accumulator one probe at a time is how you find out what it *is*; plotting it is how you check that you were right.

TRY IT

1. Tabulate the accumulator of `3*X^2` from 0 at x = 1, 2, 3, 4 and name the function you have built. Predict it from the first two probes.
2. From your probes of `1/X`, check that `FNINT(2,8)` equals `FNINT(1,8)` minus `FNINT(1,2)` before asking the machine, then ask it.
3. Find another pair of intervals with the ratio 2, nowhere near this section's, and confirm the equal-area rule with two probes.
4. Step 6 computed pi from 0 to 1. Try `8*FNINT(0,.5)` on the same curve and say what it should equal, then check. Why is it not pi?
5. The accumulator of `1/X` starting at 2 instead of 1 is a different function. What is the relationship between the two, and can you predict `FNINT(2,6)` from step 5's answer without computing it?

Indeterminate forms by table

5.7

When the top and bottom of a fraction both head for zero, the quotient's fate is genuinely undecided. Nought over nought can settle anywhere at all, and only the route taken decides where. The same holds when both head for infinity, and when a base heading for 1 is raised to a power heading for infinity.

Chapter 4 probed limits with the table. Here the same tool takes on three indeterminate forms, one specimen each.

1. The nought-over-nought specimen divides e to the 2x minus 1 by x, which is undefined at 0 where top and bottom both vanish.

   Type it as `(EXP(2X)-1)/X` (**2nd** **LN** types `EXP()`, press **GRAPH**, let the slow exponential plot finish, then press **MORE** for the table.

   The `X=0` row reads `UNDEF`, and the unit-step rows below grow ferociously: `6.389`, `26.79`, `134.1`, `744.9`, `4405.` in the five-character cells. Far from 0 the exponential simply runs away, and the table is telling you about the wrong part of the function.
2. Press **-** four times, halving the step to 0.0625. The rows now read `2.130`, `2.272`, `2.426`, `2.594`, `2.778` beneath the unmoved `UNDEF`: from the right, the quotient slides down towards 2.

Press [▲] once for the left side: from `X=-0.31` the rows read `1.487`, `1.573`, `1.667`, `1.769`, `1.880`, climbing towards the same 2 from below.

Both roads point at 2, which is the derivative of e to the 2x at 0 wearing a disguise. Press [▼] to bring the table back to its 0 anchor before moving on.

3. The infinity-over-infinity specimen is `(3x^2 + 5x)/(x^2 + 4)`, where top and bottom both blow up.

   Press [EXIT] and let the plot redraw, then [EXIT] again and [CLEAR], type `(3*X^2+5*X)/(X^2+4)`, and press [GRAPH]. Press [MORE]: the table reopens with the step still 0.0625, position and step being kept across equations.

   Press [+] four times for step 1, and the rows read `0`, `1.6`, `2.75`, `3.230`, `3.4`, `3.448`: the quotient climbs through 3 and keeps going, overshooting.

4. Growing steps are the zoom-out this form needs. Press [+] eight more times, doubling the step out to 256: the rows read `3.019` at 256, `3.009` at 512, `3.006` at 768, `3.004` at 1024, and `3.003` at 1280.

   The overshoot has drained away and the far right of the table settles onto 3, the ratio of the leading coefficients. As in Chapter 4, the probes point and the algebra pins: dividing top and bottom by x squared proves it in one line.

The third form, and a number you know

5. The last form is 1 to the power infinity, which surprises people because 1 to any power is 1. The catch is that the base is only *heading* for 1, and how fast it heads there competes with how fast the exponent grows.

   The specimen is (1 + h) to the power 1/h as h heads for 0. You can type that directly now, and it is worth doing once: `(1+.1)^(1/.1)` answers `= 2.5937424601`, exactly, because 1/.1 is 10 and a whole exponent takes the exact route of section 1.5.

   It is written below through that section's identity instead, b to the power x is e to the power x ln b, for two reasons. The first is that the identity is the form the mathematics is in: the whole question is a race between `LN(1+h)` heading for 0 and the `1/h` multiplying it heading for infinity, and this way both are on the screen where you can watch them. The second is that `1/h` stops being a whole number the moment h stops being a power of ten, so the direct form would quietly change route partway down the table while the identity holds one method throughout.

So type `EXP(LN(1+.1)/.1)` and press **ENTER**. Pressing **CLEAR** before each, work down:

h	EXP(LN(1+h)/h)
.1	2.5937424601248
.01	2.7048138297089
.001	2.7169239351903
.0001	2.7181459484956

6. Press **CLEAR** and spell `EXP(1): = 2.7182818284583`.

That is where the column is going, and the column is going there slowly. Compare Chapter 4's `sin x` over `x`, which bought two more correct digits for every tenfold shrink. This one buys about one, which is the difference between an error that goes like h squared and one that goes like h.

The limit is e, and this is where e comes from: not from a button, but from asking what happens when compound interest is compounded infinitely often. Chapter 1 left money growing at six per cent through `EXP(` and `LN(` without saying where the `EXP(` came from. This is where.

TRY IT

1. Probe `(1-COS(X))/X^2` at 0 with halved steps from both sides. This nought-over-nought settles somewhere new: predict it from the series for cosine before you look.
2. Probe `(2*X^2-3*X)/(5*X^2+1)` with doubled steps. What limit do the leading coefficients predict, and how far out must the table go before two cells in a row agree with it?
3. Build a nought-over-nought quotient of your own design that settles at exactly 7, and verify it by table.
4. Compute `EXP(LN(1+.06/12)*12)` and compare with `1.06`. What financial question have you just answered, and which is bigger?
5. The table in step 5 converges like h. Work out from it how small h would need to be for ten correct digits, and say whether the machine's fourteen digits would survive the arithmetic.

5.8 Improper integrals

An integral to infinity is a limit in disguise: the area out to b, as b grows without bound. Some settle and some do not, and a machine confined to finite bounds can still gather the decisive evidence, provided you ask it carefully.

There are two ways an integral can be improper, and they need different handling. The interval can be infinite, or the integrand can be. This section does both, and the second one is where the machine will try to mislead you.

The infinite interval

The convergent specimen is 1/x^2 from 1 onward, whose area out to b is 1 - 1/b, so it should settle on 1.

1. Store `1/X^2`, press **GRAPH**, let the plot finish, press **EXIT**, and press **CLEAR**. Probe with growing bounds, **CLEAR** between probes:

   `FNINT(1,10)` answers `= 0.90004882475237`, close to the paper value 0.9. But `FNINT(1,100)` answers `= 1.0990950153906` where paper says 0.99, and `FNINT(1,1000)` answers `= 5.313753753657` where paper says 0.999.

   The probes did not fail. They were stretched. `FNINT(` spreads sixty-four panels across whatever interval you give it, so a thousand-unit interval puts each panel fifteen units wide and starves the spike near 1 that holds nearly all the area.

2. The route that works keeps every probe short: walk to infinity in octaves. Pressing **CLEAR** between probes, ask `FNINT(1,2)`, `FNINT(2,4)`, `FNINT(4,8)` and `FNINT(8,16)`:

   `= 0.50000000769193`, `= 0.2500000038458`, `= 0.12500000192296`, and `= 0.062500000961443`.

   Each doubling of distance halves the slab, and the running totals 0.5, 0.75, 0.875, 0.9375 climb the geometric staircase whose top is 1. The integral converges, and you have watched it converge rather than been told.

3. Now the contrast. Press **CLEAR**, store `1/X`, let the plot finish, press **EXIT**, and press **CLEAR**. The same octave walk answers `= 0.6931471824209` for `FNINT(1,2)`, `= 0.6931471824212` for `FNINT(2,4)`, and `= 0.69314718242097` for `FNINT(4,8)`.

Section 5.6's logarithm constant, every octave the same. The slabs refuse to shrink, the total climbs by 0.693 per octave forever, and the integral diverges.

Convergence was never about the curve heading to zero. 1/x heads to zero too. It is about how fast the slabs thin out, and 1/x thins out exactly too slowly.

The infinite integrand

4. The other kind of trouble sits at the near end. The specimen is 1/sqrt(x) on 0 to 1, where the integrand goes to infinity at the left endpoint. On paper the integral from a to 1 is 2 minus twice the square root of a, so as a heads for 0 it should climb to 2.

   Store `1/SQRT(X)` ([2nd] [x²] types `SQRT()`, press [GRAPH], let it finish, press [EXIT], and press [CLEAR]. Probe with shrinking lower bounds, [CLEAR] between each:

Lower bound	Answer	Should be
`.25`	`1.0000000248458`	1
`.0625`	`1.5000071749317`	1.5
`.01`	`1.8016594383027`	1.8
`.0001`	`2.3623050025348`	1.98

   Read that last row twice.

   The integral is supposed to be climbing towards 2 and it has gone straight past it, to 2.36, with no error and no warning. If you had been collecting evidence that the limit is 2, this probe would have destroyed your confidence in an answer that was perfectly correct.

   It is Chapter 8's pendulum again, and by now you should recognise the shape of it. Sixty-four panels across an interval containing a spike that goes to infinity, one panel lands near the spike, and its enormous value swamps the honest ones. The narrower you make the interval around the singularity, the worse it gets, because the panel that lands nearest gets closer to the infinity.

5. So handle it the way Chapter 8 did: change the integral rather than the integrator. Substituting x = u squared turns the integrand into a constant 2 over the whole range, and the integral becomes trivially 2.

Or, if you would rather stay numerical, split the difference: use the probes at .25 and .0625, where the panels are still coping, and check them against the formula. Both agree to seven digits. The evidence for the limit being 2 is in the rows where the method was working, not in the row where it broke.

The general lesson, which is worth more than either specimen: **an answer from a numerical method is only evidence if the method was in a position to compute it.** Knowing when it was not is the actual skill, and it is why this book keeps asking you to work things out on paper first.

TRY IT

1. Octave-walk `1/X^3` from 1. By what factor does each slab shrink, and what total do the running sums point at? Predict both first.
2. Ask `FNINT(1,1000)` on `1/X^3` and compare it with the octave story. Which evidence do you trust, and why?
3. Probe `1/X^(2/3)` near 0 the way step 4 did. This one also converges, and to what? Find the value where the panels stop coping.
4. Work through the substitution of step 5 on paper for `1/SQRT(X)` and confirm you get a constant integrand. Then do the same for `1/X^(2/3)`.
5. Design an integrand that is infinite at the *right* endpoint instead, and predict what `FNINT(` will do before you try it.

5.9 Polynomial approximation

Polynomials are the only functions arithmetic can touch directly. Every machine's sine is secretly a polynomial's impersonation of one, this one included.

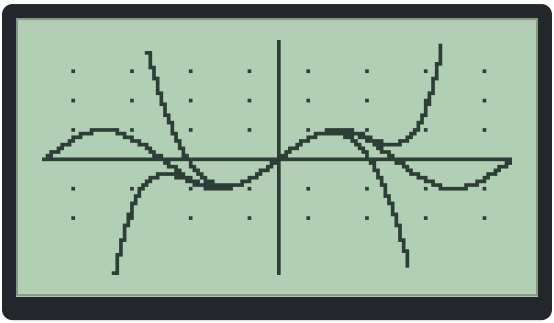
The impersonators are built from a function's derivatives at a single point. For sine at 0 the recipe gives x, then x - x^3/6, then x - x^3/6 + x^5/120, each new term paying for a wider stretch of agreement.

Free85's three slots hold a target and two impersonators side by side, which is exactly the right number for the comparison this section needs.

Sine, where more terms always help

1. Store `SIN(X)` and let the slow plot finish. The standard window flattens a sine, so press [2nd] [GRAPH] for the zoom panel, press [MORE], then [F5], the trigonometric window, and let the replot run to the end.

2. Press **2nd** **2**, type `X–X^3/6`, and press **GRAPH**; then **2nd** **3**, type `X–X^3/6+X^5/120`, and press **GRAPH**, letting each replot finish. The powers are whole and inside ^'s range, and the factorials are typed as divisors:



The sine with its degree-3 and degree-5 impersonators

Near the origin three curves travel as one. The cubic peels off first, diving where the sine turns; the quintic holds the pose almost a full half-wave longer before shooting skyward.

3. The table says where the company parts. Press **MORE** and read the rows.

At `X=1` the columns `Y1 Y2 Y3` read `0.841`, `0.833`, `0.841`, the quintic already faithful to the cell's precision. At `X=2` they read `0.909`, `0.666`, `0.933`. At `X=3` the cubic has left the stage at `–1.5` against the true `0.141`, while the quintic still offers `0.525`. By `X=5` the pretence is over everywhere: `–0.95`, `–15.8`, `10.20`.

4. Put numbers on the parting. Press **EXIT** to leave the table, let the plot redraw, then press **EXIT** again and **CLEAR**. Slot 3, stored last, is the active equation, so `EVAL(2)` answers `= 0.93333333333337`, and typing `SIN(2)` answers `= 0.90929742646148`.

At 2 radians the quintic is generous by 0.024.

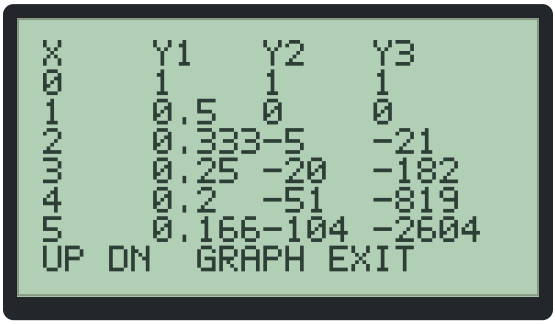
The important thing about that table is the direction of travel. Every time you add a term, the agreement gets wider. Add enough terms and you can match sine as far out as you like. That is not true of every function, and section 5.9's second half is about the ones where it fails.

A function where more terms do not help at all

5. The function is 1/(1+x), whose impersonators are the geometric series: 1 - x, then 1 - x + x^2, then 1 - x + x^2 - x^3, and so on.

Press [EXIT], press [2nd] [1], press [CLEAR], type `1/(1+X)`, and press [GRAPH]. Press [2nd] [2], press [CLEAR], type `1-X+X^2-X^3`, and press [GRAPH]. Press [2nd] [3], press [CLEAR], type `1-X+X^2-X^3+X^4-X^5`, and press [GRAPH].

Press [2nd] [+] for the standard window if you have wandered, then press [MORE] for the table:



The geometric impersonators failing outside the interval

Down the `X=0` to `X=5` rows, `Y1` reads `1`, `0.5`, `0.333`, `0.25`, `0.2`, `0.166`. `Y2` reads `1`, `0`, `-5`, `-20`, `-51`, `-104`. `Y3` reads `1`, `0`, `-21`, `-182`, `-819`, `-2604`.

Look at what the extra terms bought you. At `X=5` the degree-3 impersonator is out by 104 and the degree-5 is out by 2604. **The longer polynomial is twenty-five times worse.**

That is the whole difference between this function and sine. Adding terms here does not widen the agreement, it deepens the disaster.

6. Now find where the boundary is. Press [-] twice to quarter the table step to 0.25 and read across:

   At `X=0.25` the three columns read `0.8`, `0.796`, `0.799`: all three agree to two decimals. At `X=0.5`: `0.666`, `0.625`, `0.656`, drifting. At `X=0.75`: `0.571`, `0.390`, `0.469`, badly wrong. At `X=1`: `0.5`, `0`, `0`. And at `X=1.25`: `0.444`, `-0.64`, `-1.25`, with the longer polynomial already the worse of the two.

   The agreement holds while x is comfortably inside 1 and collapses as x approaches it. That boundary is not an accident of these two polynomials. Every impersonator in this family agrees with 1/(1+x) on the interval from -1 to 1 and nowhere else, no matter how many terms you take, and the interval never widens.

It is called the interval of convergence, and the reason it is 1 here is visible in the function: 1/(1+x) blows up at x = -1, and a power series about 0 cannot reach past the nearest place its function misbehaves. The trouble at -1 is what fences off +1 as well.

Three slots shape the comparison: a target and two rivals at a time. The degree-1 impersonator, the plain line `X`, sat the sine plot out; swapping it in is a one-slot edit, the family growing three at a time as in Chapter 1.

TRY IT

1. Replace the cubic with the line `X` and read from the table how far the plainest impersonator stays within 0.01 of the sine. Predict the answer from the `x^3/6` term first.
2. Build the cosine's impersonators `1-X^2/2` and `1-X^2/2+X^4/24` against `COS(X)` and find where each parts company by table.
3. Extend the sine recipe one more term, `X-X^3/6+X^5/120-X^7/5040`, in slot 2, and measure with `EVAL(` and `SIN(` how much further the degree-7 impersonation holds at 2 and at 3 radians.
4. The geometric impersonators fail outside -1 to 1. Predict what happens at `X=-0.9`, close to the trouble but inside it, and check. How many terms would you need for two decimals there?
5. Work out the interval of convergence for the impersonators of `1/(1+X^2)` about 0, by finding where that function misbehaves. It has no trouble anywhere on the real line, so you will have to think about where else a function can misbehave. Then test your answer by table.

CHAPTER

6

Explorations in Linear Algebra

Linear algebra is the mathematics of flat things: lines, planes, and the transformations that carry them onto one another.

Its habitat on Free85 is small. Matrices of three rows and up to six columns, with the square-only operations confined to 3 by 3, and vectors of two or three components. Small enough that every number in every register can be looked at, which is the point rather than the price.

Chapter 2 used the matrix editor as a bookkeeper. This chapter uses it as a laboratory: one system solved two ways, elimination watched move by move, condition numbers, an orthonormal frame built by hand, eigenvectors, and the `LU` factorisation.

Three habits carry the chapter and they are all consequences of one design decision.

The editors keep three registers: `A` and `B` hold the operands you type, and `R` holds the result of whichever key you press, read-only. One selection cursor is shared by all three, so wherever a walk through one register leaves it, that is where the next view opens.

Feeding a result into the next operation is one key. While `R` is on the screen, the line above the soft keys reads `ENTER USE R`, and **ENTER** copies the whole of it into `A`: dimensions, complex parts and all.

That is worth a note, because the first edition of this book made a virtue of the alternative. There was no such key then, and I argued that retyping each result by hand was what kept every intermediate step inspectable. It did nothing of the kind. `R` is still read-only, results still land somewhere they cannot be silently overwritten, and every step in this chapter is still on the screen waiting to be

looked at. What the retyping added was typing, and a fair chance of a transcription slip in the middle of an elimination you were checking *because* you did not trust it.

Read the result. Then press [ENTER] and carry on.

6.1 One system, two tools

A system of two linear equations is a pair of lines, and solving it is finding their crossing.

Design the answer first, so you know when the machine is right: through the point (2, 3) run the pair $x + 2y = 8$ and $3x - y = 3$.

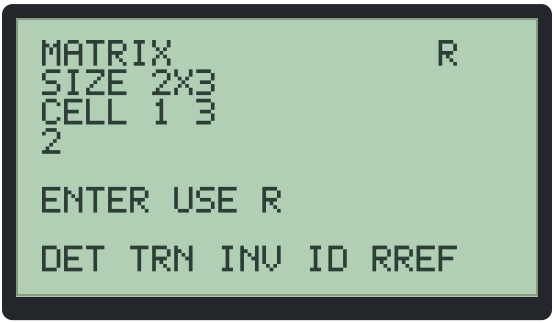
Free85 has two tools that recover the crossing from the coefficients alone, and they explain themselves very differently. One names the point. The other rewrites the system until it names itself.

1. Press [2nd] [STAT] (the `SIMULT` legend) for the simultaneous editor of Chapter 2, section 2.1: a fresh machine shows `SIZE 2`. Each row takes its coefficients and then its right-hand side, so type [1] [ENTER] [2] [ENTER] [8] [ENTER] for the first line, then [3] [ENTER] [(-)] [1] [ENTER] [3] [ENTER] for the second.
2. Press [F1], `SOLVE`. The result screen answers `UNIQUE SOLUTION` with `X 2` and `Y 3`: the crossing, named in one press.
3. Now the same system as a tableau. Press [EXIT].

   The home screen keeps a stale `= 3` from the result screen, and this one does not wipe: nothing was handed back to the entry line, so [CLEAR] answers `ENTRY EMPTY` rather than sweeping the old result away. Leave it where it sits.

   Press [2nd] [7] (the `MATRX` legend) for the matrix editor, and [x-VAR] [+] to grow the columns: `SIZE 2X3`. Type the same six values, [1] [ENTER] [2] [ENTER] [8] [ENTER] [3] [ENTER] [(-)] [1] [ENTER] [3] [ENTER], and press [F5], `RREF`.

Register R opens on CELL 1 1 reading 1, and two presses of ▶ read 0 and then 2: the first equation has become 1x + 0y = 2, and CELL 1 3 holds the value of x:



The reduced tableau naming x

Three more presses read the second row, 0, 1, 3, which says y = 3, and a sixth wraps the selection home.

Notice what the tableau did that SOLVE did not. It did not compute the answer and report it; it rewrote the question until the answer was the only thing left written down. Those are genuinely different activities and section 6.2 is about the second one.

4. One number certifies that the crossing had to be unique. Press ALPHA twice to come back to A, then -: the resize keys still point at columns from the earlier x-VAR, so the tableau narrows to SIZE 2X2, dealing its leftover values into the smaller square.

   Retype the coefficients from the top, 1 ENTER 2 ENTER 3 ENTER (-) 1 ENTER, and press F1, DET: −7.

   A nonzero determinant means the lines are not parallel, so they cross exactly once. A zero would have promised one of the two degenerate verdicts instead, and section 2.2's near-parallel pair is the warning about how little comfort a *small* nonzero determinant should give you.

The simultaneous editor scales to four unknowns and names its verdicts in words, so it is the tool when only the answer matters. The tableau shows the structure behind the verdict.

An augmented tableau has room to spare: three rows by six columns holds a three-unknown system's 3 by 4 comfortably. Press x-VAR in the editor and the +

in the editor and the + and - keys grow columns instead of rows. SOLVE remains the faster route when only the answer matters, reading coefficients from A and right-hand sides down B's first column, but it will not show you the elimination, and the elimination is what this chapter is about.

TRY IT

1. Enter the tableau 1, 2, 4, then 1, 2, 5: parallel lines. Predict what RREF's bottom row will say before you press it, then check that the simultaneous editor answers NO SOLUTION for the same six values.
2. Design your own pair of lines through (3, 1), solve with both tools, and check the answers agree.
3. Compute DET for the coefficients 2, 4, 1, 2. Which verdicts remain possible for systems built on them, and which is ruled out?
4. Build a system whose determinant is 0.001 and whose answer is still perfectly definite. Then nudge a right-hand side by a thousandth and see what happens. What does that say about using DET as a safety check?

6.2 Row operations as algebra you can watch

Elimination rewrites a system again and again, and the licence for it is that no row operation moves the solution set.

Adding a multiple of one equation to another, rescaling a row, and swapping two rows each replace the system with a different description of the same crossing. Each is reversible, which is the whole proof, and it is worth convincing yourself of that before you start pressing keys.

Section 2.4 taught the choreography: the scale in B's top-left cell, results landing in R, one selection cursor shared by the three registers. Here it serves the geometry.

The specimen is $x + 2y = 5$ and $3x + 4y = 11$, two lines designed to cross at (1, 2).

1. Press 2nd 7, then x-VAR + for SIZE 2X3, and type the tableau: 1 ENTER 2 ENTER 5 ENTER 3 ENTER 4 ENTER 1 1 ENTER. The entry wraps home, and MORE MORE brings back the row-operation page REF SWP RADD RMUL AUG.
2. First move: subtract three copies of row 1 from row 2.

Press [ALPHA] for B, type [(-)] [3] [ENTER], and press [ALPHA] again for A, where the trip has stepped the shared selection to CELL 1 2. Press [▼] three times for CELL 2 2, any cell of row 2, and press [F3], RADD.

Stepping through R reads 1, 2, 5, then 0, −2, −4.

Now read that geometrically rather than as arithmetic. The new second row says -2y = -4, which is the horizontal line y = 2. The pair of lines has changed. The crossing has not: (1, 2) still satisfies both.

3. Carry the result forward as section 2.4 taught. The fifth [▶] left the selection at CELL 2 3, so one more wraps it home, and [ALPHA] twice returns the view to A.

   Retype the new tableau: [1] [ENTER] [2] [ENTER] [5] [ENTER] [0] [ENTER] [(-)] [2] [ENTER] [(-)] [4] [ENTER].

4. Second move: rescale the new row so its surviving coefficient is 1. Press [ALPHA], type [(-)] [.] [5] [ENTER], press [ALPHA], then [▼] three times for row 2, and press [F4], RMUL.

   Row 2 of R reads 0, 1, 2: the line y = 2, the same horizontal line wearing its plainest equation. Rescaling changed the writing and not the line.

5. Carry forward again (wrap home, [ALPHA] twice, retype 1, 2, 5, 0, 1, 2), then make the last move: subtract two copies of row 2 from row 1. Store [(-)] [2] [ENTER] in B as before; back in A the selection sits at CELL 1 2, already in row 1, so press [F3], RADD.

   R reads 1, 0, 1, then 0, 1, 2. The lines are now x = 1 and y = 2: a vertical and a horizontal, whose crossing can be read off without solving anything.

   That is what elimination is for. Not to compute the answer, but to keep replacing the picture with an easier picture that has the same crossing, until the crossing is obvious.

6. The claim that nothing ever moved deserves a test.

   A still holds the half-finished tableau 1, 2, 5, 0, 1, 2 from step 5. Press [EXIT], then [2nd] [7]: re-entry always brings back the first soft-key page. Press [F5], RREF: R reads the same 1, 0, 1, 0, 1, 2.

   Machine elimination, started half-way through, lands exactly where the watched one did. Every stage described the same crossing, which is the licence stated at the top of the section, now demonstrated.

The copying between moves is the price of watching them separately. What it buys is the geometry: three different pairs of lines on the way down, and one unmoving point beneath all of them.

TRY IT

1. Rerun the elimination after a F2, SWP, so the 3, 4, 11 row is on top. The moves need different scales; predict whether the finished tableau differs before you start.
2. Rescale row 1 of the original tableau by 10 with RMUL, carry the result to A, and apply RREF. Why was the answer never in doubt?
3. Enter 0, 2, 6, then 1, 1, 4. Which single row operation lets elimination start, and what does the finished tableau read?
4. Work out what row operation would turn the finished tableau back into the original one, and confirm that reversibility is not just a claim.

6.3 Norms and the condition number

Section 2.2 met a system whose answer would not stay still, and left the diagnosis for this chapter. Here is the number that does it.

A norm is a size for a matrix, and sizes feed a more useful quantity: how much a solve can amplify small errors in its data. Real data always carries small errors, and a matrix sits between data and answer like a lever. The condition number measures the lever.

This exploration builds one comfortable matrix and one designed trap, with the fourth soft-key page whose legend overruns the screen edge.

1. Press 2nd 7, then + x-VAR + x-VAR for SIZE 3X3, and type the comfortable specimen row by row: 1, 4, 0, then 2, 1, 1, then 0, 1, 3, with ENTER after each value. Press MORE MORE MORE for the page reading NORM RNORM CNORM COND.
2. Press F1, NORM: 5.744562646538, the square root of 33, which is the sum of the nine squared cells. Press F2, RNORM: 5, the largest row sum of absolute values, from the row 1, 4, 0. Press F3, CNORM: 6, the largest column sum, from the middle column 4, 1, 1.

   Three different answers to “how big is this matrix”, all reasonable. Which one you want depends on what you are about to do with it, and for the next step

it is the first.

3. Press [F4], COND: 4.242640687119, the square root of 18.

   COND multiplies the Frobenius norm of A by the Frobenius norm of its inverse: the forward stretch times the return stretch. A value this small is a promise that answers move on the same scale as the data.

4. Hold the promise to account. This coefficient matrix with right-hand sides 5, 4, 4 was designed to solve as x = y = z = 1.

   Press [ALPHA] for B, resize it to a column with [+] [x-VAR] [-] [x-VAR] (SIZE 3X1), type 5, 4, 4 with [ENTER] after each, and press [ALPHA] to return to A. Press [EXIT], [2nd] [7], then [MORE] for the page ADD SUB MUL SCL SOLVE, and press [F5], SOLVE.

   Stepping through the SIZE 3X1 result reads 0.9999999999998, 1, 1.0000000000001: three ones under a grain of fourteen-digit dust.

5. Now nudge the data. One more [▶] wraps the selection home; press [ALPHA] for B, type [5] [.] [0] [0] [1] [ENTER] over the first entry, press [ALPHA], and press [F5] again.

   R reads 0.9999090909089, 1.0002727272727, 0.9999090909091.

   A change of one part in five thousand moved no answer by more than three parts in ten thousand. The comfortable matrix keeps its word, and the amplification is under one, which is better than COND promised.

6. The trap is three rows that nearly repeat each other. Wrap the selection home with [▶], press [ALPHA] twice for A, and retype it as 1, 1, 1, then 1, 1.001, 1, then 1, 1, 1.001, with [ENTER] after each value. Press [MORE] [MORE] for the norms page and press [F4], COND:

COND warning of a near-dependent matrix

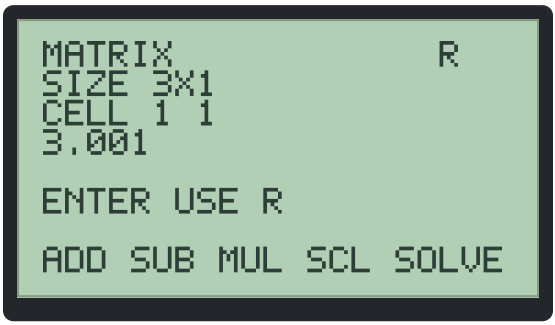
`9490.8400582879`.

Nothing has gone wrong yet. The number is a forecast, and it forecasts trouble on the order of ten thousand to one. Write that down and see whether it is fair.

7. Watch the forecast come true. Press [ALPHA] for `B` and type the designed right-hand sides [3] [ENTER] [3] [.] [0] [0] [1] [ENTER] [3] [.] [0] [0] [1] [ENTER]; press [ALPHA], then [EXIT], [2nd] [7], [MORE], and [F5], `SOLVE`.

   Stepping through `R` reads 1, 1, 1, exact to every digit. For perfect data the trap stays politely shut, which is exactly why nobody notices it until it matters.

8. Give it imperfect data. One more [▶] wraps home; press [ALPHA], type [3] [.] [0] [0] [1] [ENTER] over the first entry, press [ALPHA], and press [F5]:



The solution after a small nudge

   `R` now reads `3.001`, `0`, `0`.

   The same one-thousandth nudge that step 5 shrugged off has tripled x and thrown y and z from 1 to 0. That is an amplification of roughly two thousand, squarely in `COND`'s warned range.

   Nearly repeated rows force the solve to take differences of nearly equal numbers, and tiny data errors decide those differences outright. It is section 2.2's near-parallel receipts in three dimensions, and the condition number is the thing that would have warned you before you ran anything.

The lesson travels: before trusting a solve, ask `COND` first. A few is comfortable, thousands is a warning, and exactly dependent rows end the story at the `SINGULAR MATRIX` notice, which is the same guard `INV` raises.

TRY IT

1. Type the 3 by 3 identity into A and ask COND. Predict the answer before you press it, then say why no 3 by 3 matrix can answer less.
2. Return to the trap system of step 7 and nudge the *third* right-hand side to 3.002 instead. Which unknowns jump this time, and by how much?
3. Soften the trap: retype the near-repeats as 1.01 and ask COND again. How much smaller is the warning, and how big is step 8's jump now?
4. Section 2.2's two-by-two receipts had the same disease. Put both of them into A in turn and compare their COND values. Does the number predict what you saw there?

Building a frame of your own

6.4

Two directions are orthogonal when they meet at a right angle, and the dot product turns that geometry into one number: zero exactly when the angle is right.

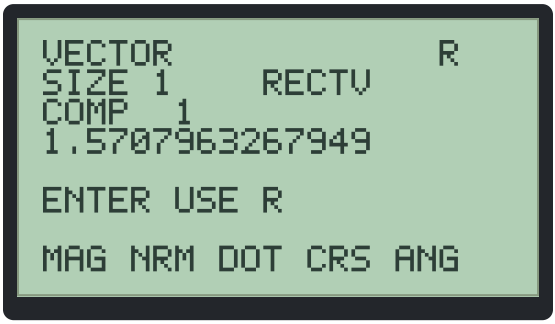
This exploration measures a designed pair, straightens one against the other, and then does the whole job properly: three arbitrary directions turned into three mutually perpendicular ones. That process has a name, Gram-Schmidt, and it is the piece of machinery underneath a great deal of applied linear algebra.

1. Press 2nd 8 (the VECTR legend): a fresh machine shows SIZE 3 with the RECTV tag. Type the first arrow, 5 ENTER 2 ENTER 0 ENTER, press ALPHA for B, type the second, 1 ENTER 2 ENTER 2 ENTER, and press ALPHA to return to A.
2. Press F1, MAG: 5.3851648071345, the square root of 29. The second's length is 3 on paper, since 1 plus 4 plus 4 is 9.
3. Press F3, DOT: 9, not zero, so the pair is not orthogonal. Press F5, ANG: 0.9799235766495, the angle between them in the fresh machine's RAD mode, a little over 56 degrees.
4. Straightening is one designed subtraction. The shadow of the first arrow along the second has length the dot product over the length squared, and 9 over 9 is exactly one copy of B.

   Press MORE for the page ADD SUB SCL 2D 3D, then F2, SUB: R reads 4, 0, −2, the first arrow with its shadow removed.

5. Carry the result into A: two presses of ▶ read the remaining components, one more wraps the selection home, ALPHA twice returns to A, and retyping 4 ENTER 0 ENTER (-) 2 ENTER stores it.

   Press EXIT, then 2nd 8, and the first soft-key page is back. Press F3, DOT: 0, exactly. Press F5, ANG:



A right angle earned by subtraction

   1.5707963267949: half of PI to fourteen digits, the right angle confirmed twice over.

Three at once

That was one straightening. Do it twice more in the right order and you turn any three independent directions into a frame.

The recipe: keep the first as it is. Take the second and subtract its shadow on the first. Take the third and subtract its shadows on both of the first two. Each subtraction removes exactly the part that was not perpendicular, and because you work in order, each new vector is perpendicular to everything already fixed.

The three starting directions are (1, 1, 0), (1, 0, 1) and (0, 1, 1), which are independent and not remotely perpendicular.

6. Keep the first. Press EXIT and 2nd 8, type 1 ENTER 1 ENTER 0 ENTER into A, and press F1, MAG: 1.4142135623731, the square root of 2.
7. Now the second. Press ALPHA and type 1 ENTER 0 ENTER 1 ENTER into B, press ALPHA, and press F3, DOT: 1.

   So the shadow of (1, 0, 1) on (1, 1, 0) is 1 over 2 of it, which is (0.5, 0.5, 0), and the straightened second direction is (1, 0, 1) minus that, which is (0.5,

-0.5, 1). Work that out on paper rather than reaching for SCL and SUB; the arithmetic is easier than the register-shuffling.

Check it. Press EXIT and 2nd 8, type . 5 ENTER (-) . 5 ENTER 1 ENTER into A, press ALPHA, type the first direction 1 ENTER 1 ENTER 0 ENTER into B, press ALPHA, and press F3, DOT: 0, exactly.

Press F1, MAG: 1.2247448713916, which is the square root of 1.5.

8. Now the third, which needs two shadows removed.

   Press EXIT and 2nd 8, type 0 ENTER 1 ENTER 1 ENTER into A, press ALPHA, type 1 ENTER 1 ENTER 0 ENTER into B, press ALPHA, and press F3, DOT: 1. So its shadow on the first is again a half of it.

   Press EXIT and 2nd 8, type (0, 1, 1) into A again, press ALPHA, type the straightened second . 5 ENTER (-) . 5 ENTER 1 ENTER into B, press ALPHA, and press F3, DOT: 0.5. Its length squared was 1.5, so the shadow is a third of it.

   Subtracting both shadows on paper: (0, 1, 1) minus (0.5, 0.5, 0) minus a third of (0.5, -0.5, 1) gives (-2/3, 2/3, 2/3).

9. Check the frame. Press EXIT and 2nd 8, then type the third direction into A. Its components are minus two thirds and then two thirds twice, so the first is (-) . 6 6 6 6 6 6 6 6 6 6 6 7 ENTER and the other two are the same digits without the sign.

   Press ALPHA, type the first direction (1, 1, 0) into B, press ALPHA, and press F3, DOT: 0, exactly.

   Now do it against the second. Retype B as (0.5, -0.5, 1) and press DOT again: −1E−14.

   Not zero. Look at that carefully, because it is the more instructive of the two answers.

   The first check came out exactly zero because the arithmetic happened to cancel exactly in fourteen digits. The second did not, because two thirds is not a terminating decimal and you typed a rounded version of it. −1E−14 is a computed zero rather than an exact one, and telling the two apart is a skill this book keeps asking for.

   Nothing has gone wrong. A dot product of minus a hundred-trillionth between two vectors of length around one means an angle that differs from a

right angle by about that much. It is as perpendicular as fourteen digits can express.

10. One key completes the set a different way. With any two of the frame in `A` and `B`, press F4, `CRS`: the cross product is perpendicular to both by construction, no subtraction needed.

    That is worth knowing as a shortcut and worth not relying on, because it only works in three dimensions and only for the third vector. Gram-Schmidt works in any number of dimensions and for any number of vectors, which is why it is the method people actually use.

The vector world here has two or three components: the plane and space are its whole territory. `CRS` insists on all three, since the cross product only lives in three dimensions, and two-component vectors stop at `DIMENSION ERROR`. Longer columns of numbers are data, and belong to the list editor.

TRY IT

1. Straighten your own pair: with 3, 1, 2 in `A` and 1, 1, 1 in `B`, the shadow is two copies of `B`. Build it with `SCL`, carry registers as needed, subtract, and confirm with `DOT`.
2. Normalise all three of the frame from step 9 with `NRM` and check that each has length 1 with `MAG`. You now hold an orthonormal frame; what would you use one for?
3. Run Gram-Schmidt on the same three directions in a *different order*, starting with (0, 1, 1). Do you get the same frame? Should you?
4. Try Gram-Schmidt on three directions that are not independent, such as (1, 1, 0), (1, 0, 1) and (2, 1, 1). Predict what goes wrong at the last subtraction, then watch it happen.
5. Step 9 got an exact zero and a computed one. Construct a case where both checks come out exactly zero, and say what is special about the numbers you chose.

6.5 Eigenvalues and eigenvectors

Multiply a vector by a matrix and its direction usually turns.

The directions a matrix keeps are its eigenvectors, the stretch it applies along each is the eigenvalue, and between them they are the matrix's character in

summary. This exploration catches a kept direction by multiplication, then lets the machine find the full set, in two sizes and one complex surprise.

1. Press [2nd] [7] and type the specimen 5, 2, then 2, 2, with [ENTER] after each value.

   Give B a test arrow: press [ALPHA], resize to a column with [x-VAR] [-] [x-VAR] (SIZE 2X1), type [1] [ENTER] [0] [ENTER], and press [ALPHA] to return to A. Press [MORE] for the arithmetic page, then [F3], MUL: R reads 5 and, one [▶] later, 2.

   The direction due east went out east-north-east. Turned.

2. Try the arrow the matrix was designed around. One more [▶] wraps the two-cell result home; press [ALPHA], type [2] [ENTER] [1] [ENTER], press [ALPHA], and press [F3] again: R reads 12 and then 6, which is six copies of 2, 1.

   Direction kept, length stretched sixfold. So (2, 1) is an eigenvector with eigenvalue 6, and you have found it by trying rather than solving, which is worth doing once before you let a key do it.

3. The machine finds the whole set in one press. Press [▶] to wrap home, then [MORE] [MORE] [MORE] for the page LU EVAL EVEC DIM FILL, and press [F2], EVAL: a SIZE 1X2 result reading 6 and, one [▶] later, 1.

   Press [F3], EVEC: a SIZE 2X2 result, one normalised eigenvector per column, and stepping through reads 0.89442719099991, 0.44721359549996, 0.44721359549996, −0.89442719099991.

   The first column is (2, 1) divided by its length; the second, (1, -2) over the same root 5, is the direction stretched by 1. Check the first against step 2 on paper.

4. A 3 by 3 next, chosen triangular so the answers are visible in advance: zeros above the diagonal mean the diagonal itself lists the eigenvalues.

   One more [▶] wraps the selection home; press [ALPHA] twice for A, grow it with [+] [x-VAR] [+] [x-VAR] (SIZE 3X3), and type 2, 0, 0, then 1, 3, 0, then 4, 5, 6, with [ENTER] after each.

   Press [F2], EVAL, and give it time: a 3 by 3's roots take the machine far longer than a 2 by 2's. The SIZE 1X3 result reads 6.0000000000007, then 2.0000000000007, then 3.

   The diagonal promised 2, 3 and 6, and the iterative hunt delivered them a whisker off in the final digits. That is the difference between reading an

answer off and searching for it, and it is worth seeing on a case where you know the truth.

5. Press [F3], EVEC, and give it time again. Stepping through the SIZE 3X3 result, the first column occupies the first, fourth and seventh cells: 0, then −1.4E−13, then −1.

   That is the third axis direction times minus one, dust included. Any rescaling of an eigenvector is the same eigenvector, so a minus sign carries no information and neither does the dust.

6. Check it by multiplying, where the arithmetic is exact. Step [▶] three times more to wrap home, press [ALPHA] for B, grow it to SIZE 3X1 with [+], and type [0] [ENTER] [0] [ENTER] [1] [ENTER]. Press [ALPHA], then [EXIT], [2nd] [7], [MORE], and [F3], MUL: R reads 0, 0, 6, exactly six copies of the third axis.

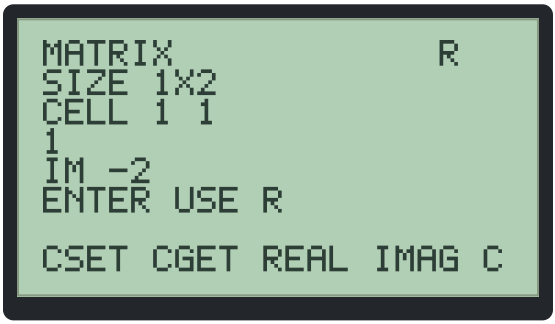
   The dusty search and the clean multiplication agree, which is the right way round to check an iterative answer: with an exact one.

7. Last, a matrix that keeps no direction at all. Press [▶] once to wrap home, press [ALPHA] twice for A, shrink it with [-] [x-VAR] [-] [x-VAR] (SIZE 2X2), and type 1, -2, then 2, 1, using [(-)] for the sign.

   This matrix turns every arrow by the same angle and stretches it by root 5, so nothing real can be kept. Predict what EVAL will say before you press it.

   Press [MORE] [MORE] [MORE] for the eigensystem page and press [F2], EVAL: both cells read 1.

   That looks wrong and is not. The real parts are only half the story, and the other half lives on the final soft-key page: press [MORE], and an IM line appears under the selected cell:



A complex eigenvalue's IM line

IM −2 beneath the first cell, and after [▶], IM 2 beneath the second. The eigenvalues are 1 minus 2i and 1 plus 2i.

A conjugate pair is how a real matrix says "I rotate", and the pair's shared size, root 5, is the stretch. Everything the matrix does is in those two numbers, and none of it is visible if you only look at the real parts.

TRY IT

1. Predict the eigenvalues and eigenvectors of 3, 1, 1, 3 from its symmetry alone, then let EVAL and EVEC grade the prediction.
2. On the first specimen 5, 2, 2, 2, check that the two eigenvalues add up to the diagonal sum and multiply to DET's answer. Both facts hold for every square matrix; verify them on the 3 by 3 too.
3. Ask EVAL about 0, -3, 3, 0, a quarter-turn with stretch. Predict the cells and the final page before you press anything.
4. The eigenvectors of step 3 came back normalised. Multiply the first column by the matrix by hand and confirm you get 6 times it, dust included.

LU as elimination's ledger 6.6

Elimination does work worth keeping.

The multipliers used on the way down and the triangle left at the bottom record the entire sweep, and with both in hand any new right-hand side costs only two short substitution passes. That is why solving many systems with one matrix is cheap everywhere in computing, and it is the reason LU exists at all.

Free85's LU key shows the ledger whole: U on and above the diagonal, the multipliers of the unit lower triangle L below.

1. Press [2nd] [7], then [+] [x-VAR] [+] [x-VAR] for SIZE 3X3, and type the designed specimen 2, 1, 1, then 4, 5, 4, then 2, 10, 11, with [ENTER] after each value. Press [F1], DET, first: 24, a figure to keep in mind.
2. Press [MORE] four times for the page LU EVAL EVEC DIM FILL, and press [F1], LU. Stepping through the SIZE 3X3 result reads 2, 1, 1, then 2, 3, 2, then 1, 3, 4.

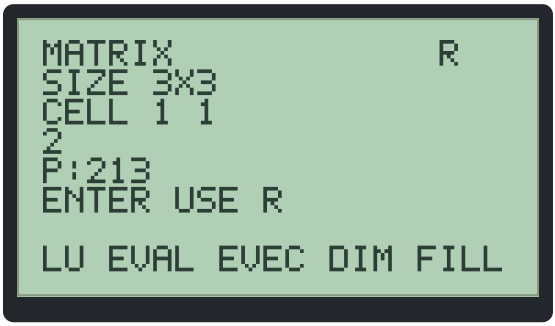
   Read it in two layers. On and above the diagonal sits U: rows 2, 1, 1 and 0, 3, 2 and 0, 0, 4. Below sit the multipliers 2, 1 and 3.

The ledger replays on paper. Row 2 of the specimen is 2 times (2, 1, 1) plus (0, 3, 2), and row 3 rebuilds the same way from its multipliers 1 and 3. Do that arithmetic yourself; the whole point of a ledger is that somebody can check it.

3. The diagonal of `U` reads 2, 3, 4, and their product is 24: the determinant is elimination's by-product, which is why `DET` and `LU` sit in the same toolbox. Step 1's answer was on the ledger all along, and `DET` almost certainly computed it this way.

4. Elimination cannot start on a zero, and the ledger records the repair too. Press [▶] once to wrap the selection home, press [ALPHA] twice for `A`, and retype it as 0, 2, 1, then 2, 4, 6, then 1, 1, 1, with [ENTER] after each value.

   Press [EXIT], then [2nd] [7] for the first page, and press [F1], `DET`: `6`. Now press [MORE] four times and press [F1], `LU`:



The ledger of a swapped elimination

   Stepping through reads `2`, `4`, `6`, then `0`, `2`, `1`, then `0.5`, `−0.5`, `−1.5`.

   The top row is the *second* row of the matrix. The factorisation swapped rows before starting, exactly as a hand elimination would when the pivot position holds a zero.

5. The swap shows up in step 3's shortcut. The diagonal now reads 2, 2, -1.5, whose product is -6, while `DET` answered `6`.

   Each row swap flips the determinant's sign, and the ledger keeps the flip. So the shortcut is not "multiply the diagonal" but "multiply the diagonal and count the swaps", and this is the case that teaches you the second half.

   The row order itself is on the screen with the answer. Below the cell value the result reads `P:213`: one digit per row, giving the original rows in the order the

factorisation used them, so the second row went first. Read that against the top row you have just stepped through and the swap is something you can see rather than something you deduce. A factorisation that needed no swap reads `P:123`.

That line is new in firmware 2.21. Before it, the permutation was written into the vector editor's result register, overwriting whatever was there, and this book told you to keep nothing precious in that register while `LU` ran. I called it a wart at the time, and defended it in the same breath: the permutation had to go somewhere, and a register that already existed was cheaper than a new object with a new type and a new way of being displayed.

It was cheaper. It was also a calculation quietly destroying data you had put somewhere else, which is a different kind of cost and one I was counting at zero. Vector `R` now survives `LU` untouched, real and imaginary parts both, and the permutation has the two lines of display it should have had from the start.

On a machine whose world is 3 by 3, the saving `LU` represents is a lesson rather than a speed-up. But it is the right lesson: `SOLVE`, `INV` and `DET` all begin with this same sweep, and `LU` is the receipt showing their common core.

TRY IT

1. Run `LU` on 3, 1, 0, then 6, 4, 1, then 0, 2, 5, and read off the multipliers and `U`. Check the diagonal's product against `DET`, and say whether a swap happened before you look at the top row.
2. Rebuild the third row of step 1's specimen on paper from the ledger's multipliers 1 and 3, without looking at `A`.
3. Design a matrix whose elimination needs a swap, and predict the sign relation between `DET` and the `LU` diagonal's product before pressing either key.
4. Use the ledger of step 2 to solve the system with right-hand sides 4, 14, 25, by forward substitution down `L` and back substitution up `U`, entirely on paper. Then check with `SOLVE`. That is the two short passes the section opened with, and doing it once is worth a chapter of description.

CHAPTER

7

Explorations in Differential Equations

Most of the equations in this book say what a quantity is. A differential equation says only how fast it is changing, and leaves you to reconstruct the quantity from that.

Free85 reconstructs it the plainest way there is. It starts at a known point and walks forward in small straight steps, which is Euler's method and the whole of the machine's numerical story. There is no adaptive step, no error control, no cleverness at all.

That plainness is the opportunity. Every choice the method makes is visible, its error can be measured against solutions you work out yourself, and the program environment of Chapter 4 can be pointed at the same equation to see whether a better step does better. A more sophisticated integrator would hide all of that behind an answer.

The mode is the Guidebook, chapter 7; programs are chapter 16.

Three habits carry the chapter, and the first one catches everybody once.

The initial value is seeded from the ordinary variable `Y` when the mode is *first entered*, so store it before you switch modes. Storing a new one into `Y` afterwards does nothing: from then on the initial condition belongs to the mode, and it is edited on the mode's own setup page, reached with **2nd** **MORE** pressed four times. Section 7.7 works that page properly.

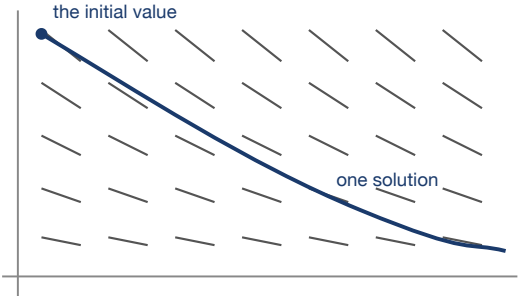
The entry line never clears itself: the home screen hands the stored slope back whenever you leave the plot, and **GRAPH** stores whatever the line holds, so an empty line pressed into **GRAPH** wipes the equation.

And every plot must be left to draw to its end. Integrating one column at a time is slow work and presses arriving mid-draw are dropped.

7.1 Slope thinking

A differential equation is a rule for the tangent.

Write dy/dx = f(x, y) and you have been handed, at every point of the plane, the direction a solution through that point must set off in. Nothing has been solved. A direction has been posted at every address, and reading those directions before you touch the machine is most of the skill.



A grid of short slope marks, with one solution curve threaded through them so that it runs along the mark at every point it passes

That picture is the way to think about it, and it is worth saying at once that Free85 will not draw it for you. The mode integrates solutions; it does not paint directions. The picture is here because you need it in your head, not because a key produces it.

The model for this chapter is a mixing tank of my own design. A 500-litre tank has been dosed to 9 grams of tracer dye per litre. Clean water runs in at 75 litres a minute and the stirred mixture runs out at the same rate, so each minute the tank loses 75/500, or 15 per cent, of the dye it happens to hold.

With y for the concentration in grams per litre and x for the time in minutes, the rule is dy/dx = -0.15y.

1. Read the rule before solving it. Press **CLEAR**, type **(-)** **.** **1** **5** **×** **9**, and press **ENTER**: `= −1.35`. That is the rate of fall at the moment of dosing.

   Press **CLEAR** and ask the same at a third of the dose, **(-)** **.** **1** **5** **×** **3**: `= −0.45`.

The slope shrinks in exact proportion to what is left, which already gives you the shape without any machinery: a steep drop flattening into a long tail, never quite reaching zero, because the rule stops pushing when there is nothing left to push.

Write that shape down. In a moment you will see whether you were right, and being right for the right reason here is worth more than the plot.

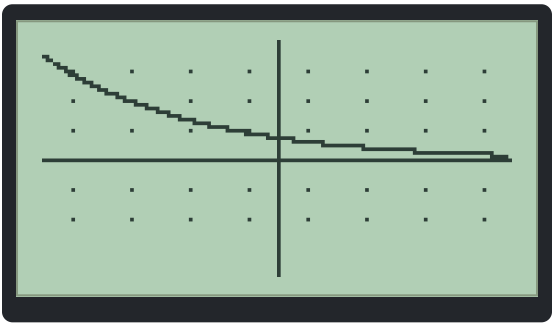
2. Seed the initial value, and check the method while you are at it: the mode starts on `EULER`, which is what this section wants, and [2nd] [MORE] pressed four times shows it under `METHOD` on the `DEQ SETUP` page. Section 7.5 changes it.

   Press [CLEAR], type [9] [STO▶] [ALPHA] [0] (the letter `Y`), and press [ENTER]: `= 9`.

3. Press [CLEAR], then [GRAPH] for the graph screen, then [2nd] [MORE] for the format page and [MORE] twice more for the page reading `FN POL PAR DEQ GC`. Press [F4], `DEQ`, and let the replot finish.

   A flat line sits high in the window. With no slope stored the mode carries the seeded 9 straight across, which is the receipt for the seeding: it tells you the 9 arrived.

4. Press [EXIT] for the home screen, where the entry line is empty. Type [(-)] [.] [1] [5] [×] [ALPHA] [0] so the line reads `-.15*Y`, and press [GRAPH]:



The tank's dye concentration falling across the window

Let the plot finish.

The solution starts at the *left window edge* and walks right, which fixes the arithmetic of the whole chapter: the tank is dosed at `XMIN`, so elapsed time is x + 10 in the standard window. Get that wrong once and every reading in this chapter is out by ten minutes.

5. Read the story off the curve. Press ▶ once: `X=0.236220472438` with `Y=1.9028746602579`. So ten and a bit minutes after dosing, a fifth of the dye is left.

   Press ◀ once, letting the readout settle: `X=0.078740157478` with `Y=1.9489119504254`. A column is worth about a twentieth of a gram per litre here, which is a fifth of what a column is worth at the left edge where the walk begins. The curve is flattening, exactly as step 1 said it would.

Slots 2 and 3 exist in this mode but the plot ignores them. One first-order equation is what the mode integrates, so the habit of stacking a family three at a time, learned in Chapter 1, does not travel here. That turns out to matter a great deal in section 7.6, where a family is exactly what you want.

TRY IT

1. Work out on paper how long the tank takes to fall to half its dose. Write the number down. Then find the same half-way point on the plot with ◀ and ▶ and see how close the machine puts it.
2. Store `-.3*Y` instead, by pressing **EXIT**, **CLEAR** and retyping, and replot. Before you look: where on the new curve does the solution reach the value the old one reached at x = 0?
3. Predict the plot of `.15*Y` before pressing **GRAPH**, including which edge of the window the curve leaves by. Then draw it. Why does the initial condition make that inevitable?
4. The slope at any point depends on `Y` and not on `X`. What does that tell you about the slope field of the diagram above, and what would you expect two solutions started at different times to look like?
5. A tank of twice the volume with the same flow loses 7.5 per cent a minute. Predict how its half-way time compares with the original's, then check by storing `-.075*Y`.

7.2 The window is the step

Euler's method needs a step size, and Free85 never asks you for one.

It takes the step from the window. The solution is sampled once per plotted column, so the step is the window's width divided by 127. No setting overrides it and no tolerance tightens it.

That is worth stating plainly because it makes the zoom keys into this mode's numerical controls, which is not where anybody expects to find them. Zooming in on a solution does not magnify a picture you already have. It recomputes the whole thing at a finer step, and gives you a different answer.

1. With the tank's solution plotted, press [MORE] for the table. The `Y1` column holds the integrated solution: `X=0` reads `1.972`, then `1.694`, `1.455`, `1.250`, `1.074`, `0.923` at `X=5`.

   Press [▲] to page back five rows: `X=-5` reads `4.213`, and the page runs down to `1.972` again.

2. Press [EXIT] to leave the table and let the plot redraw, then [EXIT] again for the home screen. It hands `-.15*Y` back to the entry line and publishes `= 1.9724874982123`.

   That is the last value the table worked out: the fourteen-digit face of the `1.972` cell, whose five-character column truncated the rest away. The table is not rounding for effect, it is running out of room.

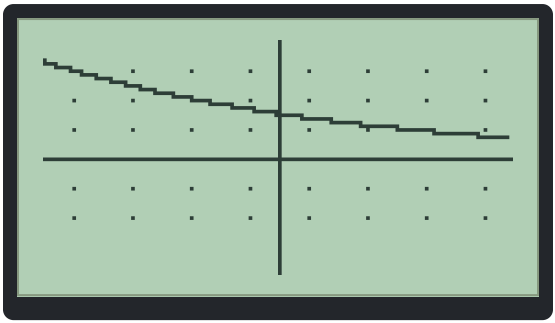
3. Press [CLEAR] and measure the step: [2] [0] [÷] [1] [2] [7] [ENTER] gives `= 0.15748031496063`, twenty units of x across 127 intervals.

   Now get the truth to compare against. A quantity falling at 15 per cent of itself per minute is 9 times e to the power -0.15t after t minutes, and the `X=0` row is t = 10. Press [CLEAR] and type [9] [×] [2nd] [LN] (which inserts `EXP()`) [(-)] [1] [.] [5] [)] [ENTER]: `= 2.0081714413361`.

   Euler is low by about 0.036, or 1.8 per cent. Low, not high, and that is not luck: each straight step leaves along the tangent, and a tangent falls away below a curve that bends upwards.

4. Halve the step by halving the window. Press [CLEAR], retype [(-)] [.] [1] [5] [×] [ALPHA] [0], press [GRAPH] to store the equation back and replot, then press [+]

once and let the replot finish:



The same equation in the halved window

5. Press [MORE] for the table. The `X=0` row now reads `4.232`, not `1.972`.

   Nothing about the tank has changed. The *experiment* has.

   Press [▲] to page back: `X=−5` reads `9`, which is the initial condition itself, and every row above reads `UNDEF`. Narrowing the window did not zoom in on the old solution. It re-based the run, so `X=0` is now five minutes after the dose rather than ten.

6. Press [EXIT] twice for the home screen and press [CLEAR]. Type [1] [0] [÷] [1] [2] [7] [ENTER]: `= 0.078740157480315`, exactly half the old step.

   Press [CLEAR], retype the equation, press [GRAPH], then [2nd] [+] for the standard window, and let the replot finish.

So two things moved between step 3 and step 5, and only one of them was the step size. The run got *shorter* as well as finer, so the comparison proves nothing on its own. That is a badly designed experiment and it is worth recognising as one: if you change two things and the answer moves, you have learned nothing about either.

Section 7.3 turns it into a controlled measurement.

TRY IT

1. From the standard window press [-] once for the doubled window and read the `X=0` row. What is the step now, and how far from the truth at twenty minutes does the reading fall? Predict the direction of the error before you look.
2. Work out from step 3's figure which column of the standard window lands closest to five minutes after the dose. Trace to it and compare with the exact value that `9*EXP(-.75)` gives.
3. The `Y2` and `Y3` columns read – throughout. Store something in slot 2 with [2nd] [2], see what the column does, and explain it from the note that closes section 7.1.
4. Step 5 said narrowing the window re-bases the run. Work out what the `X=0` row would read after *two* presses of [+], before you press anything, then check.

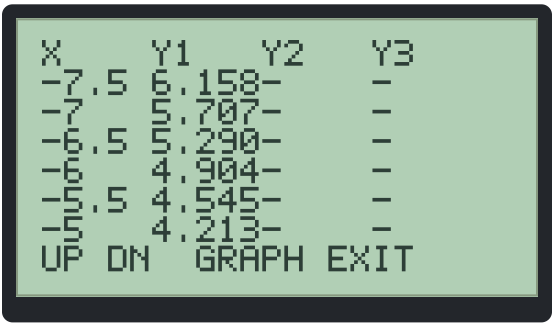
Step-size experiments

7.3

To measure how a method converges you hold the question still and vary only the step.

Here the question is: what is the concentration 3.5 minutes after dosing? The complication is that halving the window moves the dosing point too, so 3.5 minutes after the dose sits at a different `X` in each window: -6.5 in the standard one, -1.5 in the halved one, 1 in the one after that. Work that out on paper first, because looking up the wrong row is the easiest mistake in this section.

1. In the standard window press [MORE] for the table, which opens where section 7.2 left it, at `X=-10` in steps of 1. Press [-] once to halve the table step to 0.5, then press [▼] once to page down five rows:



The 3.5-minute reading in the standard window

The rows run –7.5 to –5, and the X=–6.5 row reads 5.290.

2. Press [EXIT] to leave the table and press [+] for the halved window, letting each replot finish. Press [MORE]: the table opens on rows now outside the window, reading UNDEF down to X=–5, where 9 sits. Press [▼] twice, letting each page settle: the X=–1.5 row reads 5.307.
3. Press [EXIT] and press [+] again. The plot comes up empty, because the vertical zoom has come down to -2.5 to 2.5 and the solution spends the whole run above it.

   The table does not mind at all. Press [MORE], then [▼] once: the X=1 row reads 5.315. That is worth remembering: when the zoom has thrown the curve off the screen, the table is still the whole instrument.
4. Press [EXIT], press [2nd] [+] to restore the standard window, and let the replot finish. Press [EXIT] for the home screen and [CLEAR]. Type [5] [÷] [1] [2] [7] [ENTER] for the third step, = 0.039370078740157, press [CLEAR], and ask for the truth at 3.5 minutes, [9] [×] [2nd] [LN] [(-)] [.] [5] [2] [5] [)] [ENTER]: = 5.3239982793029.

Window	Step	Reading at 3.5 minutes	Gap
-10 to 10	0.15748031496063	5.290 at X=–6.5	0.034
-5 to 5	0.078740157480315	5.307 at X=–1.5	0.017
-2.5 to 2.5	0.039370078740157	5.315 at X=1	0.009

Halve the step and the error halves. That is Euler's signature and its disappointment: the method is first order, so one more decimal place costs ten times the work. Compare the midpoint rule of Chapter 4, which quartered its error per halving, and you can see why nobody integrates anything seriously with Euler.

Two boundaries showed themselves on the way. Table rows outside the window read UNDEF, because the run *is* the window. And the zoom keys move both axes together, so a window narrow enough to refine the step may be far too short to show the curve at all.

TRY IT

1. Press [+] a third time and hunt for the 3.5-minute reading. Work out which X it needs from the new bounds first, then look that row up. What does the answer tell you about refining a step by zooming?
2. The gaps above are roughly 0.21 times the step. Use that to predict the step needed for a gap under 0.001, and say how many halvings that is. Then say whether the window that fine would show you anything.
3. Take the doubled window ([-] from standard) and repeat the measurement. Predict the gap first from the pattern, then check whether it doubles.
4. The constant 0.21 is not arbitrary. It is roughly half the second derivative of the true solution near this point. Work that out on paper from the exact solution and see how close 0.21 is.

An Euler program

7.4

The window is symmetric, so section 7.3 could not hold the interval fixed while the step shrank. A program has no such trouble, because the step becomes a number in a memory instead of a consequence of the picture.

This one walks the same equation with the step in H and the number of steps in N.

The program never names the model. EVAL(reads the stored slope, so the calculus command does the modelling and the program does only the arithmetic. That is worth setting up carefully, so before typing anything, find out what EVAL(is actually reading.

Press [CLEAR] and spell EVAL(0), then press [ENTER]: `= -0.064835852069246`. Press [CLEAR] and ask EVAL(5): the same answer, because the stored slope has no X in it at all.

What it does contain is Y, which is an ordinary variable the mode uses as scratch while integrating. Press [CLEAR] and ask [ALPHA] [0] [ENTER]: `= 0.43223901379497`, left there by the last plot at the right-hand window edge. So the program has to seed Y itself, or it will start from wherever the last plot happened to stop.

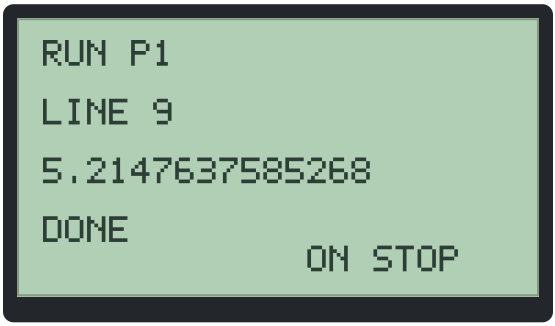
1. Press [CLEAR], then [PRGM] and [F1], NEW. The editor opens on `EDIT P1`. Type the eight lines, [ENTER] after each; letters are [ALPHA] plus the key carrying the letter, spaces are [2nd] [0] in this editor, and [STO▶] types the –> arrow.

Line	Text	Keys
1	`9->Y`	9 STO▶ Y
2	`.5->H`	. 5 STO▶ H
3	`7->N`	7 STO▶ N
4	`WHILE N`	W H I L E 2nd 0 N
5	`Y+H*EVAL(0)->Y`	Y + H × E V A L (0) STO▶ Y
6	`N-1->N`	N - 1 STO▶ N
7	`END`	E N D
8	`DISP Y`	D I S P 2nd 0 Y

Line 5 is Euler's method entire: the new y is the old y plus the step times the slope at the old y. Everything else is bookkeeping.

Seven steps of 0.5 carry the walk 3.5 minutes from the dose, which is section 7.3's question asked a second way, and now with the interval held still.

2. Press F2, `RUN`:



Seven Euler steps of 0.5 landing short of the truth

The run screen answers `RUN P1` over `LINE 9`, the output line shows `5.2147637585268`, and the status reads `DONE`. Against the truth of `5.3239982793029`, the walk is low by 0.109.

3. Halve the step. Press PRGM for the list and F1 to reopen `EDIT P1` at line 1, press ▼ for line 2, press CLEAR, and type `.25->H`. ENTER moves to line 3, where CLEAR and `14->N` doubles the count.

Press F2: `5.2705124549462`. Repeat, CLEAR before each retype, for `.125->H` and `28->N`.

H	N	Run screen	Gap
.5	7	5.2147637585268	0.109
.25	14	5.2705124549462	0.053
.125	28	5.2975280205725	0.026

The interval never moved and the errors still halve, which is the clean version of section 7.3's measurement. And the two tables agree on the constant as well: gap over step sits near 0.21 in all six rows, which is a stronger result than either table on its own.

4. One more check, and it is the satisfying one. Set the program to the mode's own step: reopen the editor and put `20/127->H` on line 2 and `64->N` on line 3, pressing CLEAR before each retype.

   Press F2: `1.9489119504254`.

   Now look back at section 7.1, step 5. That is the trace readout, digit for digit. The plot and the program are not two methods that agree; they are one walk, computed twice.

The environment shaped two decisions here.

`FOR` bounds were single digits when this was written, so a counted loop could not reach fourteen passes, and the countdown in `N` is what bought an arbitrary step count. Firmware 2.19 lifted that: `FOR N,1,127` is now a legal line, and section 7.4's walk would fit it. The countdown is kept because `N` is doing double duty as the loop's remaining work and as something the run screen can show you.

And `EVAL(` takes the slope at the current `Y` but at a *typed* `X`, so a model containing `X` would need the running `x` stepped alongside `Y`, which is a ninth line the slot has not got. This equation does not mention `x`, which is exactly why eight lines suffice. Section 7.6 stays inside that constraint too, and it is not a coincidence: the models that fit this machine are the autonomous ones.

TRY IT

1. Run the program at `.0625->H` with `56->N`. Predict the gap first from the halving pattern, then check.
2. Change line 1 to `18->Y`, run at `.5` and `7` again, and say from the gap how the error scales with the size of the solution. Was that predictable from line 5?
3. Store `-.3*Y` as the equation and rerun the program untouched. Work out the true value at 3.5 minutes and say whether the same gap-over-step constant holds for the faster tank. If not, what does it depend on?
4. The program seeds `Y` on line 1 because the mode leaves rubbish there. Delete line 1, run it twice in a row, and watch what happens. Then put it back.

7.5 Improved Euler

Euler takes the slope at the start of a step and trusts it for the whole step, which is why it lags a bending curve. Every improvement on it is some version of the same idea: look somewhere else as well, and average.

Heun's method takes the slope at the start, guesses where the step ends, takes the slope *there* too, and steps with the average of the two.

Written so the predictor is reused: k is the slope at y, then y becomes y + hk, and then y becomes y plus h times half the difference between the new slope and k. Work through why that is the same as stepping with the average, because it is not obvious and it is what makes it fit.

That is three statements inside a loop that already needs a test, a countdown and an `END`. With three setup lines and the `DISP`, an eight-line slot is two short.

The Guidebook, chapter 16 has the way out: `CALL` runs another slot and comes back, and variables are shared. So the step can live in its own program. That is the better design anyway, because swapping the called slot makes one driver run any method you like.

1. Press **PRGM** for the list, press ▼ to select the second slot, and press **F1** to open `EDIT P2`. Type the driver:

Line	Text	Keys
1	9–>Y	9 STO▶ Y
2	.5–>H	. 5 STO▶ H
3	7–>N	7 STO▶ N
4	WHILE N	W H I L E 2nd 0 N
5	CALL 3	C A L L 2nd 0 3
6	N–1–>N	N - 1 STO▶ N
7	END	E N D
8	DISP Y	D I S P 2nd 0 Y

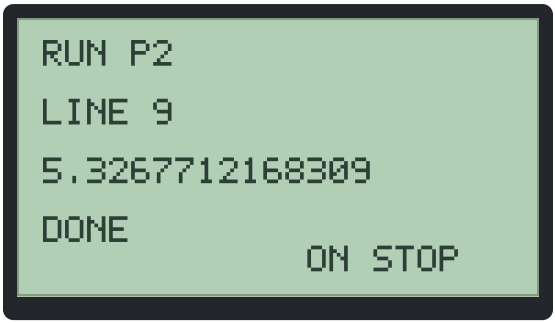
2. Press EXIT to save and return to the list, press ▼ for the third slot, and press F1 for EDIT P3. Type the step itself:

Line	Text	Keys
1	EVAL(0)–>K	E V A L (0) STO▶ K
2	Y+H*K–>Y	Y + H × K STO▶ Y
3	Y+H*(EVAL(0)–K)/2–>Y	Y + H × (E V A L (0) - K) ÷ 2 STO▶ Y
4	RETURN	R E T U R N

Line 1 keeps the slope where the step begins. Line 2 makes the Euler guess, leaving Y at the predicted end, so line 3's EVAL(0) reads the slope *there*. Half the difference of the two slopes, stepped, turns the predictor into the average-slope step.

3. Press EXIT for the list, press ▲ to select P2, and press F3, RUN. The pair takes noticeably longer than a single slot, because every step now calls EVAL(

twice; let it finish:



Improved Euler landing on the far side of the truth

The run screen answers `5.3267712168309`.

Euler at the same step landed at `5.2147637585268` against a truth of `5.3239982793029`. So seven improved steps miss by 0.0028 where seven plain ones missed by 0.109: forty times better for twice the work.

4. Halve twice more. Press **PRGM** for the list, which returns with `P2` selected, press **F1**, and edit lines 2 and 3 as in section 7.4, **CLEAR** before each retype: `.25→H` with `14→N`, then `.125→H` with `28→N`.

H	N	Run screen	Gap
.5	7	5.3267712168309	0.0028 above
.25	14	5.3246721242446	0.00067 above
.125	28	5.3241643775915	0.00017 above

Two things changed at once and both are worth naming.

The gaps now quarter per halving rather than halve, which is what second order means. And they sit on the *far* side of the truth, because averaging the two slopes overcorrects a curve of this shape where the single slope undercorrected it.

Seven improved steps beat twenty-eight plain ones by nearly ten to one, and they ask `EVAL(` half as many times to do it. That is the whole argument for better methods in one table.

The mode has this method built in, and has had since firmware 2.18. On the `DEQ SETUP` page (**2nd** **MORE** four times), **F1** cycles `METHOD` through `EULER`, `HEUN`

The mode has this method built in, and has had since firmware 2.18. On the DEQ SETUP page (2nd MORE four times), F1 cycles METHOD through EULER, HEUN and RK4; HEUN is the method you have just written by hand, and RK4 takes four slopes per step instead of two and quarters its error again twice over.

Which raises the obvious question: why write it, if it is already there? Because a method you have not built is a method you are trusting. You now know what HEUN costs, why it overshoots a curve of this shape, and what its error does when you halve the step, and none of that is visible from the outside of a menu. Select it on the setup page from here on, and plot the same equation under all three: the shapes you get are the table above, drawn.

TRY IT

1. Change P2's line 5 back to `Y+H*EVAL(0)->Y` and delete P3. Confirm the driver alone reproduces section 7.4's numbers, and say what that proves about where the improvement actually lives.
2. Write a midpoint step into the fourth slot: slope at the start, half a step forward, slope there, then a whole step from the *original* point with that slope. Mind the line that must remember the original Y.
3. Run the improved pair at `20/127->H` with `64->N`. How far apart are the plot and the better method by the middle of the window?
4. The gaps in the table are all above the truth and section 7.4's were all below. Design a differential equation for which you would expect the reverse of both, and test it.

Growth with a ceiling 7.6

Everything so far has decayed. Now let something grow, and give it somewhere to stop.

Unlimited exponential growth is the first model anybody meets and it is almost never the right one, because nothing grows forever. Bacteria run out of nutrient, a population runs out of habitat, a rumour runs out of people who have not heard it. What every one of those has in common is a ceiling, and the interesting question is what shape the approach to the ceiling takes.

Two models dominate the subject, and they disagree in a way you can see.

The logistic equation

The logistic model says the growth rate is proportional to the population *and* to the fraction of the ceiling still unused. Write K for the ceiling:

$dy/dx = k\,y\,(1 - y/K)$

When y is small the bracket is near 1 and growth is nearly exponential. When y approaches K the bracket approaches 0 and growth shuts off. In between, something has to give, and where it gives is the whole content of the model.

Take k = 0.5 and a ceiling of 10.

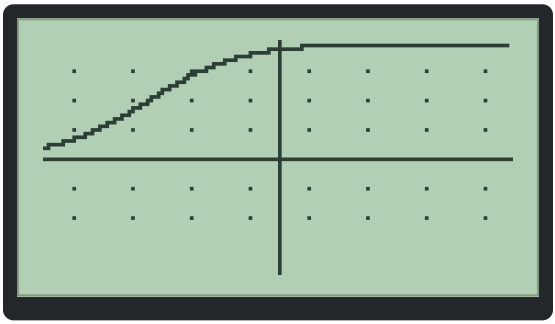
1. Before touching the machine, work out where the growth is fastest. The rate is a product of y and (1 - y/K), which as a function of y is an upside-down parabola with zeros at 0 and K. Its top is halfway between, at y = K/2 = 5.

   So the population grows fastest when it is exactly half full, and slows down after that. Write down the shape that implies: slow, then accelerating, then a bend at half the ceiling, then a long flattening.

2. Seed a small starting population. Press [CLEAR], type [1] [STO▶] [ALPHA] [0], and press [ENTER]: `= 1`.

3. Press [CLEAR], press [GRAPH], then [2nd] [MORE], [MORE], [MORE] and [F4] for `DEQ`. Let the replot finish and press [EXIT].

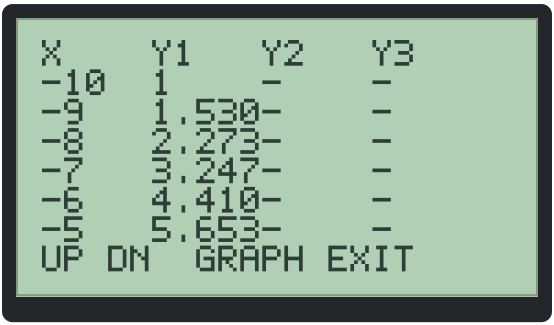
4. Type the model: [.] [5] [×] [ALPHA] [0] [×] [(] [1] [-] [ALPHA] [0] [÷] [1] [0] [)] so the line reads `.5*Y*(1-Y/10)`. Press [GRAPH] and let it draw, which takes a while:



The logistic S-curve rising to its ceiling

   There is the S. It is one of the most recognisable shapes in applied mathematics and you have just made the machine derive it from a rule about rates, with no formula for the curve anywhere in sight.

5. Read the bend off the table. Press **MORE**, then ▲ twice, letting each page settle:



The logistic table through its steepest stretch

   From `X=-10` the rows read `1`, `1.530`, `2.273`, `3.247`, `4.410`, `5.653`.

   Take the differences yourself: 0.530, 0.743, 0.974, 1.163, 1.243. They are still growing, so the curve is still accelerating, and the largest of them straddles the crossing of 5. That is step 1's prediction arriving on screen: the fastest growth is at half the ceiling.

   Page down and the differences shrink the whole way. `X=0` onward reads `9.439`, `9.658`, `9.793`, `9.876`, `9.926`, `9.956`, closing on 10 without ever arriving.

Gompertz, which bends earlier

The Gompertz model makes a different guess about what slows growth down. Instead of the fraction of the ceiling left, it uses the *logarithm* of the ratio of the ceiling to the current size:

$$dy/dx = k\,y\,\ln(K/y)$$

That looks stranger and it is much older, and it is what most tumour growth and a great deal of reliability work actually use. The reason is that it bends earlier: real growth very often slows sooner than the logistic predicts.

6. Find the inflection first, on paper. Differentiate the rate with respect to y and you get $k(\ln(K/y) - 1)$, which vanishes when $\ln(K/y) = 1$, so when $y = K/e$.

   With $K = 10$ that is about 3.68, against the logistic's 5. So Gompertz should turn its corner at just over a third of the ceiling rather than at half of it. Write that down before you look.

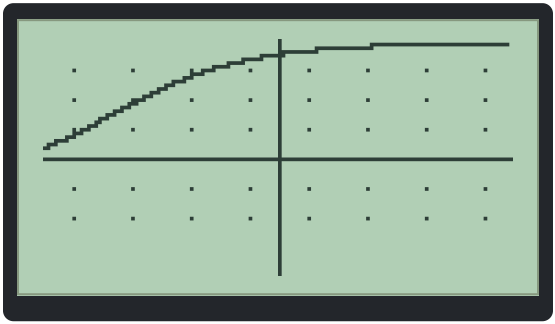
7. Put the starting value back to 1. Press [2nd] [MORE] four times for the DEQ SETUP page, press [F3] (Y0), and press [-] or [+] until Y0 reads 1. Press [F5] (GO).

   That is the whole of it. The equation stays where it is and the window stays where it is; only the seed moves.

8. Press [EXIT] for the entry line.

9. Type the Gompertz rule: [.] [3] [×] [ALPHA] [0] [×] [LN] [1] [0] [÷] [ALPHA] [0] [)] so the line reads .3*Y*LN(10/Y). Press [GRAPH].

   This one is slow. Every Euler step now costs a logarithm on top of everything else, and there are 127 of them. Let it finish:

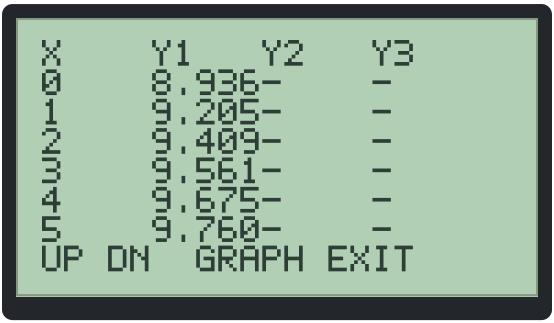


The Gompertz curve, bending earlier than the logistic

   Put that beside the picture from step 4. Same seed, same ceiling, and a visibly different route: the Gompertz curve is already turning while the logistic is still climbing hard, and then it spends much longer creeping up on the ceiling.

10. Press [MORE] for the table and be patient with it. This is the slowest thing in the book: each row is a complete Euler walk from the window edge, and each

step of each walk wants a logarithm.



The Gompertz table closing slowly on the ceiling

From `X=0` the rows read `8.936`, `9.205`, `9.409`, `9.561`, `9.675`, `9.760`.

Now compare, row for row, with the logistic's `9.439`, `9.658`, `9.793`, `9.876`, `9.926`, `9.956`.

The Gompertz is behind at every one, and falling further behind. It turned earlier, so it gave up its fastest growth sooner, and it pays for that with a much longer tail. Which of those two behaviours your data actually shows is exactly how you choose between the models, and it is a choice you can now make by eye.

One route is closed here and it is the one you most want. There is no way to put both curves on the screen at once. The mode integrates slot 1 alone, and there is no picture store to overlay one plot on another, so the comparison has to be made across two plots and two tables, or by writing the numbers down as above. On a machine with a picture store this section would be one screen. It is four, and the numbers are the compensation.

TRY IT

1. Predict, then check: what does the logistic do if you seed it *above* the ceiling, say at 18? Which way does the curve go, and what does the bracket in the rule do to make it go that way?
2. Work out on paper the logistic's value at its inflection for k = 0.5 and K = 10, then find the two table rows that straddle it and confirm the crossing sits between them.
3. Change the logistic's k to 1 and leave the ceiling alone. Predict what moves and what does not, in both the picture and the inflection point, before you plot it.
4. The Gompertz rule has `LN(10/Y)` in it. What happens if you seed it at exactly 10? At 0? Work out both from the formula before you go anywhere near the machine, because one of them will not end well.
5. Fit them against each other. Take the logistic's `X=0` reading of `9.439` and find, by trying values of k, a Gompertz that passes through the same point at the same x. Do the two curves then agree anywhere else?

7.7 Equilibria, and the lever that resets them

Change one thing in the tank and a whole family appears.

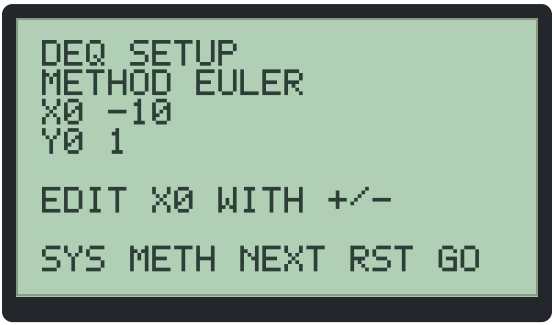
Suppose the incoming water carries dye of its own, so the concentration is pulled towards a level A rather than towards nothing: dy/dx = k(A - y).

The sign of the bracket does all the thinking. Above A it is negative and the solution falls. Below A it is positive and the solution rises. At A it is zero and nothing moves at all, and that last line is a solution in its own right: the constant one, called an equilibrium.

This section takes A = 3 with k = 0.4, and it needs the initial value to change, which is what the setup page is for. It used to be for something much worse, and the story is short and worth having.

1. The seed from section 7.6 is still 1, below A, so start there and come back for the other side. Press **PRGM** to leave any run screen, press **EXIT** for home, press **CLEAR**, type **.** **4** **×** **(** **3** **-** **ALPHA** **0** **)** so the line reads `.4*(3-Y)`, press **GRAPH**, and let it finish.
2. Press **MORE** for the table. The solution climbs and flattens, squeezing onto 3 without arriving: an asymptote seen from below.

3. Now cross the equilibrium. Press EXIT to leave the table, then press 2nd MORE four times for the DEQ SETUP page:



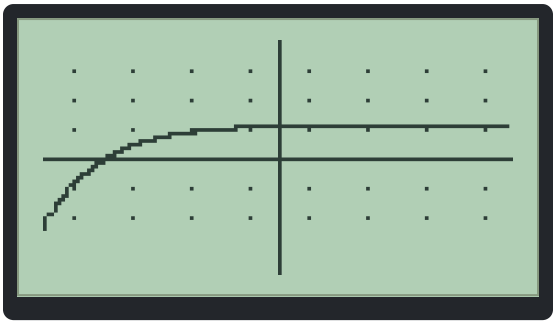
The DEQ setup page, where the initial condition lives

METHOD names the solver, and X0 and Y0 are the initial condition. F2 and F3 choose which of the two the + and - keys move, and the line above the soft keys tells you which one you have got.

This is worth a paragraph of history, because the first edition of this book taught a ritual here instead. The mode used to freeze its initial condition on first entry and keep it in a store object called GDEQ, and the only way to change your mind was to leave the mode, open the memory browser, find GDEQ, and delete it, which cleared your equations along with your seed. I wrote that the stiffness was instructive: that the cost of a shot ought to be part of the lesson. It was not instructive. It was a missing feature with a good story attached, and the story was mine, which is exactly the kind of argument to distrust.

4. Press F3 (Y0), then press - until Y0 reads −6, seven presses down from 1. The equation is untouched and so is the window.

5. Press F5 (GO):



The same equilibrium approached from below

Let the plot finish. The curve climbs out of the bottom of the window and flattens along the same level as before.

Press MORE for the table, still in steps of 1: `X=0` reads `2.855`, then `2.904`, `2.936`, `2.958`, `2.972`, `2.981`.

The equilibrium is approached from below just as it was from above, and neither solution crosses it. They cannot: crossing means passing through a point where the rule says do not move.

6. One sign turns the whole picture over. Press EXIT to leave the table, let the plot redraw, press EXIT for home, and press CLEAR. Type (-) . 3 × (3 - ALPHA 0) so the line reads `-.3*(3-Y)`, and press GRAPH.

   Let it finish. The solution leaves through the bottom of the window almost at once. Press MORE for the table: `X=0` reads `-165.`, then `-223.`, `-300.`, `-403.`, `-542.`, `-727.`.

   The equilibrium at 3 is still a solution. Every neighbour now flees it.

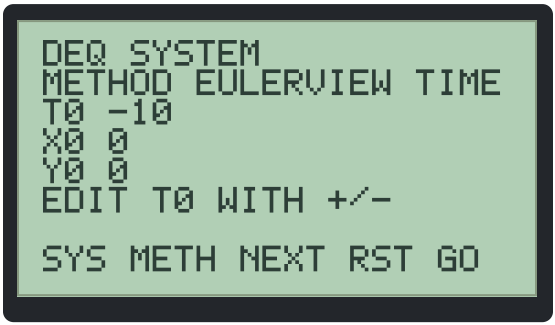
   Stable and unstable equilibria differ by nothing more than the sign of k, and that is the single most useful thing in this chapter: you can tell which you have by looking at the rule, without solving anything and without the machine.

7.8 Two equations at once, and the phase plane

Everything so far has been one equation and one curve. A predator and its prey, a mass on a spring, any pair of quantities that drive each other, is a system of two,

and its natural picture is not a curve against x at all. It is the phase plane: one quantity plotted against the other.

The mode does both. Open the setup page with 2nd MORE four times and press F1, SYS:



The DEQ system setup: T0, X0, Y0, and the view

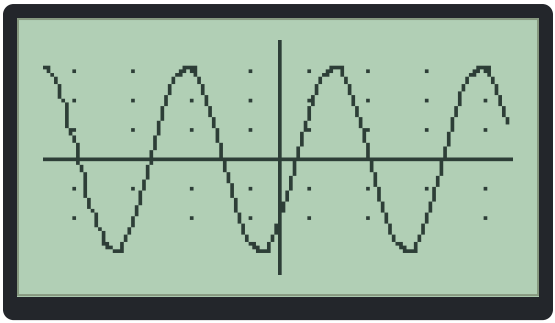
The banner reads DEQ SYSTEM, and three things have changed. Slot 1 is now dX/dT and slot 2 is dY/dT. The initial condition has three fields, T0, X0 and Y0, which F3 (NEXT) steps between and + and - move. And VIEW has appeared beside the method, which MORE switches between TIME and PHAS.

1. Take the oscillator: a mass on a spring, where the velocity is one state and the position is the other. Written as a system, dX/dT = y and dY/dT = -x.

   Press EXIT for the entry line, type ALPHA 0 for Y, and press GRAPH to store it in slot 1. Press 2nd 2 for slot 2, type (-) x-VAR for −X, and press GRAPH.

2. Set the start and the method. On the setup page press F2 (METH) twice for RK4, press F3 (NEXT) once so the prompt reads EDIT X0, press + eight times

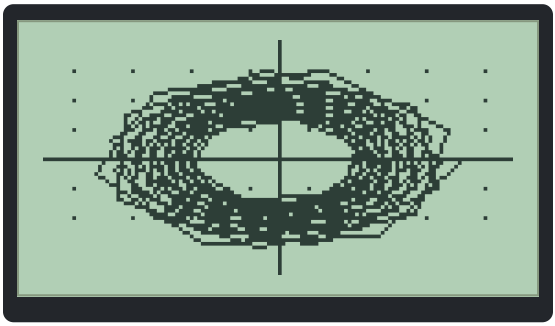
for an `X0` of 8, and press F5, `GO`.



The oscillator in the time view

A cosine, which you could have predicted and which tells you little you did not already know.

3. Now press 2nd MORE four times again, press MORE once so `VIEW` reads `PHAS`, and press F5:



The oscillator's phase orbit, many revolutions on one ring

Position against velocity, and the curve closes on itself. That is the picture of a conserved quantity: the energy the time trace only implies. Nothing oscillates in it, which is exactly why it is worth having.

It is thick because you are seeing about twenty revolutions at once. The phase view takes 128 samples and integrates each by the table step, which is 1 by default, so 128 units of time go by and the orbit comes round many times. Every one of those passes lands on the same ring, and that is the point: the ring not spreading is the conservation, drawn. Halve the table step in the table screen and you draw fewer, cleaner loops.

4. The orbit is also the cleanest test of a method you will find. Go back to the setup page, press F2 until METHOD reads EULER, and replot.

   The ring stops being a ring. Euler's error at every step points the same way round the circle and never cancels, so the orbit spirals outward and the energy it is supposed to conserve grows visibly. HEUN spirals more slowly. RK4 holds the ring.

   A method's error is not a number here. It is a shape, and the shape tells you what kind of wrong it is.

One note on the phase view, because it catches people. The graph window now bounds the *state space* rather than time, so XMIN and XMAX are limits on x, not on t. An orbit of radius 8 needs a window that reaches 8, which is what section 1.1's window editor is for.

This section is new. For most of this book's life the mode integrated one equation from one initial condition, and I wrote here that two state variables would have meant a second integrator, a second initial condition and a plotting mode that draws y against y rather than against x, and that there was not room. There was room.

What travels beyond two states is still the thinking: the sign of the right-hand side, the equilibria where it vanishes, and whether neighbours join them or leave. That much you can do on paper for a system of any size, and the rest of this chapter is where you learned to.

TRY IT

1. Change the oscillator to a damped one: Y in slot 1 and −X−.1∗Y in slot 2. Predict what the phase picture does before you plot it, then plot it and say what the shape tells you that the time trace does not.
2. Predator and prey, in the simplest form there is: dX/dT = X−X∗Y and dY/dT = −Y+X∗Y. Start from X0 and Y0 both 1 and plot the phase view. What kind of curve do you get, and what does it say about whether either species dies out?
3. Put the oscillator back and set METHOD to EULER. Count how many revolutions it takes before the spiral is obviously not a ring. Then do the same for HEUN. What ratio would you have predicted from section 7.5?
4. The phase view uses the table step as its integration step and takes 128 samples. Work out, before touching the machine, what table step draws exactly one revolution of the oscillator. Then set it and see how close you were.

TRY IT

1. Predict, without the machine, what `.4*(3-Y)` does when it is seeded at exactly 3. Then run the reset of steps 3 to 5 with `3->Y` and see whether you were right.
2. Find the equilibrium of `.2*(8-Y)-.5` by hand, then plot it from a seed above and a seed below and confirm the level from the table.
3. The logistic of section 7.6 has *two* equilibria. Find them both from the rule, decide which is stable and which is not, and check both by seeding near each.
4. On paper, sketch the directions of the pair dx/dt = y, dy/dt = -x at eight points around the origin, and say what shape a solution must follow. Then say which single Free85 equation, if any, would show you the same thing.
5. Design a rule with three equilibria, alternating stable and unstable. Predict the fate of a solution started between each pair, then check as many as you like: each one is now three keys on the setup page.

CHAPTER

8

Explorations in Engineering Mathematics

Engineering asks the same questions as the earlier chapters and refuses the same answers.

It wants the period of a real pendulum, not of its linearisation. It wants a series summed to a stated accuracy, not a limit named. It wants a solution pinned at both ends of a span, not released from one. It wants forces and moments in three dimensions, where a sign and a convention will not save you.

Free85 has a tool pointed at each. `FNINT(` for integrals with no closed form, the four program slots for sums no hand would face, the DifEq mode of Chapter 7 working with the solver workspace, and the vector editor for statics in the round. The Guidebook covers all four: chapter 3 for the calculus commands, 16 for programs, 14 for the solver, 13 for vectors.

Three habits from earlier chapters apply at once, and this chapter will punish you for forgetting any of them. The entry line never clears itself, so press CLEAR before typing at home. The calculus commands read the active stored equation, so store it with GRAPH first. And plots must be left to finish, because presses arriving mid-draw are dropped.

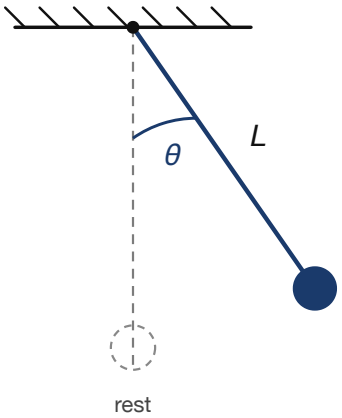
The pendulum, and the integral that will not behave 8.1

The formula every textbook prints for a pendulum, 2 pi times the square root of length over gravity, is not the period of a pendulum.

It is the period of a pendulum's *linearisation*: the fiction in which the restoring force is proportional to the angle rather than to the sine of the angle. For small swings the two are near enough identical. For large ones they are not, and the difference has a name, the circular error, and a size you can measure.

The true period depends on how far the thing swings, and the dependence is an integral with no closed form in elementary functions at all. That is a happy accident for a machine like this one. A quantity nobody can write down is exactly what `FNINT(` is for.

The specimen is a garden swing: a seat on ropes 2.5 metres long, with gravity taken as 9.8.



A pendulum of length L held at an angle theta from the vertical, with the rest position dashed below the pivot

The fiction first

1. Press **CLEAR** and type **2** **×** **2nd** **^** (the π legend) **×** **2nd** **x²** (which supplies `SQRT()` **2** **.** **5** **÷** **9** **.** **8** **)**, then press **ENTER**: `= 3.1734878129702`.

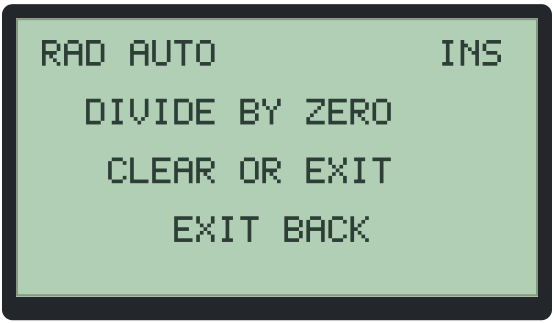
   A little over three seconds, and the same three seconds whatever the swing does. Hold on to that number. Everything in this section is measured against it.

The obvious integral, and what it does to you

Conservation of energy gives the true period directly. Write A for the amplitude, the angle the swing is released from. Then the period is four times the square root of L over 2g, times the integral from 0 to A of d theta over the square root of cos theta minus cos A.

That is a perfectly respectable piece of mathematics and you should type it in, because what happens next is the most useful thing in this section.

2. Store the amplitude. We will use a quarter turn, 45 degrees. Press [CLEAR], type [2nd] [^] [÷] [4] [STO▶] [ALPHA] [LOG] (the letter A), and press [ENTER]: `= 0.78539816339745`.
3. Press [CLEAR] and type the integrand, `1/SQRT(COS(X)-COS(A))`: [1] [÷] [2nd] [x²] [COS] [x-VAR] [)] [-] [COS] [ALPHA] [LOG] [)] [)]. Press [GRAPH] and let the plot finish.
4. Press [EXIT], press [CLEAR], spell `FNINT(0,A)`, and press [ENTER]. Give it time.



The pendulum integral refused: DIVIDE BY ZERO

```
DIVIDE BY ZERO.
```

Stop and look at that, because the refusal is the interesting part.

At the top of the swing, where theta reaches A, cos theta minus cos A is zero and the integrand is infinite. The integral still converges, because the infinity is mild enough, but the *function* is unbounded at that endpoint. `FNINT(` samples the interval, one of its samples lands on the endpoint, and dividing by zero is exactly what it finds there. So it says so, and stops.

It did not always. Firmware 2.10 spread 64 panels across whatever interval you gave it, evaluated the integrand at each, added up, and answered `= 9643.817428027` without a murmur: no error, no warning, no notice. The answer should be about 2.31. That one was four thousand times too big, and if you had not known roughly what to expect you would have written 9643 down and carried it into the next calculation.

I defended that, in the first edition of this book. I wrote that an integrator which second-guesses you is a worse tool than one that does what you ask, and that the responsibility therefore sits with you. I was wrong, and it is worth naming how. The argument confused refusing to *guess* with refusing to *warn*. The machine was not honouring my request; it was answering a question I had not asked, in a voice indistinguishable from the one it uses when it is right. Putting the responsibility on the reader only works if the reader is given something to act on, and silence is not something you can act on.

So `FNINT(` now compares a 32-panel estimate with a 64-panel one, and with a 128-panel one when those two disagree. If the estimates will not settle inside that budget it answers `NO CONVERGENCE`; if a sample lands on a singularity it answers `DIVIDE BY ZERO`. It still will not guess what you meant. It has simply stopped pretending.

Doing the mathematics so the machine can succeed

The refusal tells you to stop. It does not tell you what to do, and no integrator was ever going to, because the difficulty here is in the integral rather than in the arithmetic. The fix is to hand the machine a different integral, one that means the same thing and has no infinity in it. This is the lesson of the section and probably of the chapter: **when a numerical method struggles, the first place to look is the mathematics, not the method.**

Two steps get you there, and both are standard.

First, use the half-angle identities. Cos theta is 1 minus twice sine squared of theta over 2, and cos A is 1 minus twice sine squared of A over 2. Subtract, and the difference of cosines becomes twice the difference of two sine-squares. Write k for sin(A/2) and the integral is now over the square root of k squared minus sine squared of theta over 2.

Second, substitute. Put sin(theta/2) equal to k sin x. As theta runs from 0 to A, x runs from 0 to a right angle, and after the dust settles the whole thing collapses to

4 times the square root of L over g, times the integral from 0 to pi over 2 of dx over the square root of 1 minus k squared sine squared x.

The infinity has gone. When x reaches its limit the denominator is the square root of 1 minus k squared, which is a perfectly ordinary positive number for any swing

short of a full half turn. That integral is the complete elliptic integral of the first kind, and it is what the machine should have been given in the first place.

5. Store the *squared* modulus, so that changing the amplitude later costs one store rather than a retyped equation. Press [CLEAR] and type [SIN] [2nd] [^] [÷] [8] [)] [x²] [STO▶] [ALPHA] [x²] (the letter K), then press [ENTER]: `= 0.14644660940673`.

   That is sine squared of 22.5 degrees, which is k squared for a 45-degree swing.

6. Press [CLEAR] and type the new integrand, `1/SQRT(1-K*SIN(X)^2)`: [1] [÷] [2nd] [x²] [1] [-] [ALPHA] [x²] [×] [SIN] [x-VAR] [)] [x²] [)]. Press [GRAPH] and let it finish. On a fresh machine K holds 0 and the slot draws the constant 1, which is its own sort of receipt.

7. Press [EXIT], press [CLEAR], spell `FNINT(0,PI/2)`, and press [ENTER]: `= 1.6335863074566`.

   No drama at all. Same physics, same amplitude, a bounded integrand, and an answer you can use.

8. Turn it into a period. Press [CLEAR], type [4] [×] [2nd] [x²] [2] [.] [5] [÷] [9] [.] [8] [)] [×] and then spell `FNINT(0,PI/2)`, and press [ENTER]: `= 3.3003427304458` seconds.

   Against the textbook's `3.1734878129702`. A 45-degree swing runs about four per cent slow, which is a great deal more than most people expect and is the subject of section 8.2.

TRY IT

1. Before you run anything, predict what the naive integral of step 4 will do if you shrink the amplitude to `PI/12`. Will the nonsense get better or worse? Write down which, and a reason, then store the new `A` and find out.
2. Run step 4's integral again but stop short of the top: ask `FNINT(0,A-.01)` and then `FNINT(0,A-.001)`. The integrand is bounded on those intervals, so the answers should be sensible. Are they, and what are they converging on?
3. Check the substitution did not change the answer, by computing the true period a third way: take step 2's result for `FNINT(0,A-.001)`, multiply by 4 times the square root of L over 2g, and compare with step 8.
4. Take the identity apart on the machine. Store `A` as `PI/4`, then compute `COS(.3)-COS(A)` directly, and compute `2*(SIN(A/2)^2-SIN(.15)^2)`. They should agree. Do they, and to how many digits?
5. What happens to the transformed integrand when the amplitude reaches a full half turn, so `K` is 1? Predict the value of the integrand at x = pi/2 before you press a key, then store `1->K` and ask `FNINT(0,PI/2)`. What has the substitution failed to save you from?

8.2 How wrong is the textbook formula?

Section 8.1 built a working integral. Now use it for something: measure the circular error across the whole range of swings, from a gentle rock to nearly upside down, and find out where the textbook formula stops being good enough.

The quantity to track is the ratio of the true period to the small-angle one, which is 2 over pi times the integral of section 8.1. If the ratio is 1.01 the swing runs one per cent slow.

1. With the integrand of section 8.1 still stored and `K` holding 0 on a fresh machine, press CLEAR, type 2 ×, spell `FNINT`, and type `(0,PI/2)/PI`. Press ENTER: `= 1`.

   No amplitude, no correction, and the machine says so exactly. That is a good sign the arithmetic is set up right, and it costs one press to check.

2. Now walk up through the amplitudes. Each one is two steps: store the new `K`, then ask the same ratio. Press CLEAR before each.

   For an amplitude of 10 degrees the half-angle is pi over 36, so type SIN 2nd ^ ÷ 3 6) x² STO▶ ALPHA x² and press ENTER:

= 0.0075961234938959. Then CLEAR and 2*FNINT(0,PI/2)/PI: = 1.0019071881423.

Five more go the same way:

Amplitude	Stored expression	K	Ratio	Slow by
10 degrees	SIN(PI/36)^2	0.0075961234938959	1.0019071881423	0.19%
30 degrees	SIN(PI/12)^2	0.066987298107785	1.017408797595	1.71%
60 degrees	SIN(PI/6)^2	0.25	1.0731820071483	6.82%
90 degrees	SIN(PI/4)^2	0.49999999999999	1.1803405990146	15.28%
120 degrees	SIN(PI/3)^2	0.75000000000012	1.3728805006153	27.16%
150 degrees	SIN(5*PI/12)^2	0.9330127018918	1.7622037294847	43.25%

The 60-degree row is the one to check your typing against. The sine of 30 degrees is exactly a half, so K comes back as 0.25 with no dust at all, while its neighbours carry a grain in the last digit. When a row that should be clean is not, you have mistyped something.

The last column is worked out from the fourth: the fraction of the true period that the textbook formula misses is one minus the reciprocal of the ratio. For the 30-degree row, press CLEAR and type 100*.017408797595/1.017408797595: = 1.7110917102498.

3. Read the table before moving on, because the shape of it is the answer to the question in the heading.

   A playground swing going through 30 degrees runs under two per cent slow, which nobody would notice. At 90 degrees, a swing taken to the horizontal, the error is 15 per cent, which is nearly half a second on this rope and would ruin any clock. And at 150 degrees the thing takes three quarters as long again as the textbook says.

   The textbook formula is not approximately right and then gradually wrong. It is extremely right for small swings and then falls apart surprisingly fast.

With K still holding the 150-degree modulus, turn the last row into seconds. Press CLEAR, type 4 × 2nd x² 2 . 5 ÷ 9 . 8) × and spell FNINT(0,PI/2), then press ENTER:

The swing's true period at a 150-degree amplitude

The entry line wraps onto a second row rather than clipping, and the answer is = 5.5923320594903 seconds against the fiction's 3.1734878129702. A swing taken almost to the horizontal takes three quarters as long again to come back.

Where the series runs out

The standard rule of thumb for the correction is 1 plus the amplitude squared over 16, with the amplitude in radians. It is the first two terms of a series, and it is worth knowing exactly how far you can trust it.

4. Press CLEAR and type 1+(PI/18)^2/16: = 1.0019038588736, against the table's 1.0019071881423 for 10 degrees. Five decimal places.
5. Press CLEAR and try 30 degrees, 1+(PI/6)^2/16: = 1.017134729863 against 1.017408797595. Three decimal places.
6. Press CLEAR and push it to 90 degrees, 1+(PI/2)^2/16: = 1.154212568767 against 1.1803405990146. Not even two.

   So the series is superb where you did not need it and useless where you did, which is the usual arrangement with series. Section 8.3 is about exactly that trade.

TRY IT

1. Add the series' next term, eleven times the amplitude to the fourth power over 3072, to the rule of step 4, and work out how far up the table it survives now. Does it buy you one more row or three?
2. Find the amplitude at which the swing runs exactly one per cent slow. You have two routes: interpolate between the 10 and 30 degree rows of the table, or put the series of step 4 into the solver workspace and let it hunt. Do both and see how far apart they land.
3. Measure your own pendulum, store its length in place of 2.5, and work out the amplitude at which its period runs one per cent long. Does the answer depend on the length at all? Predict before you compute.
4. An amplitude of a full half turn puts `K` at 1 and the swing exactly upside down. Work out from the table's trend what the ratio is heading for, then reason out the physics of that swing. Why is the answer not a number?
5. The table's ratios are all above 1. Is there any amplitude at all for which a real pendulum beats its linearisation? Answer from the shape of the integrand rather than by trying values.

One route is closed here, and it is worth knowing why before you go looking for it. A graph slot cannot hold `FNINT(`, so there is no plot of period against amplitude. The command integrates whichever equation is active, and a slot holding it would be asking to integrate itself. Chapter 4 met the same refusal from `NDER(`, and Chapter 5 meets it again. The table above is the graph, written out by hand, one store and one probe per row.

Series against closed forms

8.3

A closed form is a promise about infinitely many terms. A partial sum is what you can actually hold. They agree in the limit, so the question worth asking is not whether but how fast: how many terms buy how many decimals.

That is a program's question, because the answer comes from summing the same series again at different lengths, and the eight-line slots are long enough for two very different answers.

The first specimen is the sum of the reciprocal squares, whose closed form is one of the genuine surprises of the subject: pi squared over six.

A hundred terms, counted down

1. Press [PRGM] and [F1], NEW. The editor opens on EDIT P1. Type these eight lines, [ENTER] after each:

Line	Text	Keys
1	0–>S	[0] [STO▶] [S]
2	10–>N	[1] [0] [STO▶] [N]
3	WHILE N	[W] [H] [I] [L] [E] [2nd] [0] [N]
4	S+1/N^2–>S	[S] [+] [1] [÷] [N] [x²] [STO▶] [S]
5	N–1–>N	[N] [-] [1] [STO▶] [N]
6	END	[E] [N] [D]
7	DISP S	[D] [I] [S] [P] [2nd] [0] [S]
8	STOP	[S] [T] [O] [P]

   The countdown in N is doing two jobs at once, and both are forced on it by the environment.

   It is the loop's test, because = and < cannot be typed in this editor and a condition here has to be arithmetic: WHILE N runs while N is anything but zero. And it is the term index. Since it counts down, the smallest terms go in first, which is the order that keeps the most digits. Step 5 of this section shows you what that is worth.

2. Press [F2], RUN. The run screen answers RUN P1 over LINE 8, with 1.5497677311665 on the output line and DONE beneath.

3. Lengthen the sum. Press [PRGM] for the list, press [F1] to reopen EDIT P1 at line 1, press [▼] for line 2, press [CLEAR], and type 40–>N. [ENTER] moves on and [F2] runs it: 1.6202439630069.

Repeat for `100->N`, which takes noticeably longer. Let it finish:

A hundred reciprocal squares still short of the closed form

4. Now the promise. Press **PRGM** for the list, **EXIT** for the home screen, and **CLEAR**. Type **2nd** **^** **x²** **÷** **6** and press **ENTER**: `= 1.6449340668482`. Press **CLEAR** and take each gap in turn, **CLEAR** between them:

Terms	Partial sum	Gap
10	1.5497677311665	0.0951663356817
40	1.6202439630069	0.0246901038413
100	1.6349839001848	0.0099501666634

   The gaps are almost exactly one over the number of terms. Ten times the work buys one decimal place, and a hundred terms have not delivered two. This is a series you would not choose to compute pi with.

The same hundred terms, added the other way

5. Change one thing and watch the last digits move. The order of addition should not matter, and in exact arithmetic it does not. In fourteen digits it does.

   Press **PRGM**, press **F1**, and rewrite the program to count *up* and stop on a tolerance rather than at a fixed term. Line 3 becomes the interesting one:

Line	Text	Keys
1	`0->S`	0 STO▶ S
2	`1->N`	1 STO▶ N
3	`WHILE INT(1E4/N^2)`	W H I L E 2nd 0 I N T (1 EE 4 ÷ N x²)
4	`S+1/N^2->S`	S + 1 ÷ N x² STO▶ S
5	`N+1->N`	N + 1 STO▶ N
6	`END`	E N D
7	`DISP S`	D I S P 2nd 0 S
8	`STOP`	S T O P

Line 3 is how you write "keep going while the term is still worth adding" on a machine with no comparison operators. The next term is 1 over `N` squared. Multiply it by ten thousand and take the whole part, and you get something nonzero exactly while the term is at least a ten thousandth. When `N` passes 100 the whole part becomes 0 and the loop stops. The tolerance is the number you type on line 3, and nothing else in the program needs to know about it.

Press F2 and let it run:

A hundred terms added upwards, stopping on a tolerance

`1.634983900181`.

Put that beside step 3's `1.6349839001848`. Same series. Same hundred terms. Not the same number.

The difference is the order. Adding upwards means every one of the last ninety additions is a tiny number being added to a total near 1.6, so the tiny number gets shifted right to line up and loses its bottom digits before it is even added. Adding downwards, the total starts small and grows, so the small terms go in while there is still room for them.

Three digits at the bottom of a fourteen-digit answer is not going to ruin anybody's day. But the same effect on a sum of a million terms, or a sum where the terms have mixed signs, absolutely will, and this is the cheapest place you will ever see it happen.

A series that actually converges

The second specimen comes from a test rig that drops a weight one metre onto a damper. Each rebound reaches a quarter of the height of the one before, so the rebounds measure a quarter of a metre, then a sixteenth, then a sixty-fourth, and total a third.

6. That series needs no term variable at all, and the saving is what lets it fit. Each new partial sum is the old one plus one, all divided by four, so from 0 the first pass gives a quarter and the second five sixteenths: one line doing the work of two.

   Press [PRGM], press [▼] for the second slot, press [F1] for `EDIT P2`, and type `P1`'s original eight lines with two changes: line 2 is `6->N`, and line 4 is `(1+S)/4->S`, typed [(] [1] [+] [S] [)] [÷] [4] [STO▶] [S].

7. Press [F2]: `0.333251953125`, six rebounds. Reopen the editor as in step 3, put `12->N` on line 2, and run again:

Twelve rebounds closing on a third

`0.3333333134651`. Once more with `24->N` and the run screen shows `0.33333333333333`.

8. Press **PRGM** for the list, **EXIT** for the home screen, and **CLEAR**, since the run screen hands `S` to the entry line on the way out. Press **1** **÷** **3** **ENTER**: `= 0.33333333333333`, the same to every digit.

   At twenty-four terms the series has used up the display. Take the gaps as before, **CLEAR** between them: `1/3-.333251953125` answers `= 0.00008138020833` and `1/3-.3333333134651` answers `= 1.986823E-8`.

   Six more terms divided the gap by 4096, which is 4 to the sixth. Every term throws away three quarters of what is left.

Two series, two stories, and the contrast is the whole point of the section. The reciprocal squares close a smaller and smaller *share* of their gap with every term, so the gap falls like one over the count and a decimal place costs tenfold work. The rebounds throw away a fixed fraction every time, so the gap falls like a power and one or two terms buy a decimal outright.

The programs are the same shape and the same length. Apart from line 2's count, retyped before every run anyway, the difference is entirely in line 4. That is worth noticing: the thing that decides whether a computation is cheap or hopeless is not the code, it is the mathematics the code is carrying.

The environment shaped one decision and forbade another. `FOR` bounds were single digits when this was written, so no counted loop reached a hundred passes and the countdown in `N` is what bought an arbitrary term count. Since firmware 2.19 `FOR N,1,100` would say it directly, and the countdown survives here because it keeps the running total and the remaining work both visible on one line. And the run screen shows only the most recent `DISP`, so the tables above are built one run at a time.

TRY IT

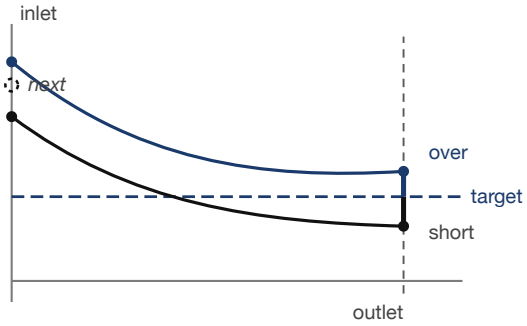
1. Predict, before running it, what happens to the tolerance program of step 5 if you change the `1E4` on line 3 to `1E6`. How many terms, and how long? Write down your estimate of both, then try it and see how close you were on each.
2. Change `P1`'s line 4 to `S+1/(N*N+N)->S`. That one has a closed form you can find on paper in a line. Find it first, then check how many terms four decimals cost.
3. Rewrite `P2` for a damper that returns a *third* of each height rather than a quarter. What replaces the 4 on line 4, and what closed form should the run screen approach? Both before you type anything.
4. Sum `1/N^3` at 10, 40 and 100 terms. This one has no closed form in elementary functions, so you cannot check it against a target. Measure the shorter runs against the longest instead, and work out which power of the term count the gap follows now.
5. Step 5 showed the order of addition changing the answer. Build the worst case you can: sum `1/N` up to 100 terms upwards, then downwards, and see how far apart they get. Then say why this series shows the effect more clearly than the reciprocal squares did.

Shooting at a boundary

8.4

The differential equations of Chapter 7 all started at a known point and walked forward. Engineering more often knows the two *ends* and not the beginning: the temperature at both faces of a wall, the deflection at both supports of a beam, the concentration at the outlet of a treatment works.

The oldest way to answer that with a forward-marching method is the shooting method, and it is exactly what its name says. Guess the starting value. Integrate to the far end. See how far you missed. Adjust. Repeat.



Two shots from the same start, one falling short of the target at the far end and one going over, with the straight line between the misses picking the next guess

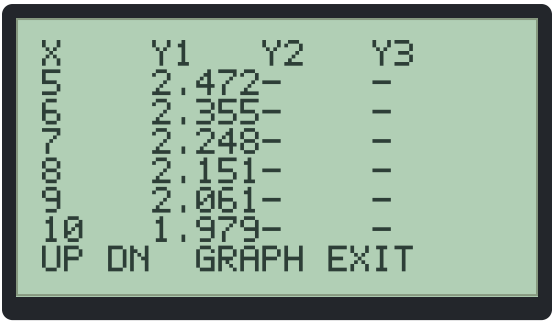
The specimen is a reed bed 20 metres from inlet to outlet, treating a stream whose pollutant the bed removes at a rate proportional to the *square* of its concentration: dy/dx = -0.02 y squared, with y in milligrams per litre and x in metres. No more than 2 milligrams per litre may leave the outlet, and the question is what the inlet may carry.

The first shot, and the mode digging in

1. The initial value comes from the ordinary variable `Y`, seeded before the mode is entered. Press **CLEAR**, type **1** **0** **STO▶** **ALPHA** **0** (the letter `Y`), and press **ENTER**: `= 10`.
2. Press **CLEAR**, then **GRAPH**, then **2nd** **MORE** for the format page and **MORE** twice more for the page reading `FN POL PAR DEQ GC`. Press **F4**, `DEQ`. Let the replot finish and press **EXIT** for the home screen.
3. The entry line is empty, as section 7.1 found. Type **(-)** **.** **0** **2** **×** **ALPHA** **0** **x²** so the line reads `-.02*Y^2`, and press **GRAPH**. Let the plot finish.

   The concentration falls steeply out of the top of the window and then flattens, which is what a square-law removal looks like: the last milligram is the expensive one.
4. Press **MORE** for the table, which opens at `X=0` in steps of 1, then press **▼**

once and let it settle:



The first shot landing short of the outlet consent

The rows run `X=5` to `X=10`, and the last of them, the outlet, reads `1.979` where the paperwork says 2. The seed of 10 has undershot.

5. Press **EXIT** to leave the table, let the plot redraw, and press **EXIT** again for the home screen. It hands `-.02*Y^2` back to the entry line and publishes `= 1.9796408183505`, the fourteen-digit face of that `1.979` cell.

   Now try a second shot the obvious way, and watch it fail. Press **CLEAR** and store a new seed: **1** **2** **STO▶** **ALPHA** **0** **ENTER** answers `= 12`. Press **CLEAR**, retype `-.02*Y^2`, and press **GRAPH**. Let the plot finish and press **MORE**: the table reopens where it was, and the outlet still reads `1.979`.

   Nothing moved, and that is worth understanding rather than working around. `Y` seeds the mode at the moment the mode is *created*. After that the initial condition belongs to the mode, and storing into `Y` is talking to the wrong thing.

   So talk to the right thing. Press **EXIT** to leave the table, press **2nd** **MORE** four times for `DEQ SETUP`, press **F3** (`Y0`), press **+** twice to carry the seed from 10 to 12, and press **F5** (`GO`). Let the plot finish, then press **MORE** and read the outlet cell again. It has moved, the equation is untouched, and the window is where you left it.

   That is three keys a shot. In the first edition of this book it was about twenty, because the seed lived in a store object and the only way to change your mind was to delete the object, which cleared your equation along with it. I wrote then that I would not defend it as a design. I did not have to defend it for long.

Which leaves a better reason to write the program than the one I gave before. It is no longer that the mode is painful. It is that shooting is a *search*, and a search wants a number out rather than a picture to squint at: the program below returns the outlet value directly, so you can compare two shots by subtracting them instead of by eye.

So the mode gives the picture and a program gives the practice

6. The walk is the one section 7.4 built: the new y is the old y plus the step times the slope, with `EVAL(` reading the stored slope so the program never names the model.

   Press [EXIT] to leave the table, let the plot redraw, press [EXIT] again, and press [CLEAR]. Press [PRGM], press [▼] twice for the third slot, and press [F1], `NEW`, opening `EDIT P3`:

Line	Text	Keys
1	`10→Y`	[1] [0] [STO▶] [Y]
2	`20/127→H`	[2] [0] [÷] [1] [2] [7] [STO▶] [H]
3	`127→N`	[1] [2] [7] [STO▶] [N]

   Lines 4 to 8 are section 7.4's walk, unchanged and typed the same way: `WHILE N, Y+H*EVAL(0)→Y, N−1→N, END, DISP Y`.

   Only the first three lines are new, and their step and count are the plot's own: the mode samples once per column, so 127 steps carry the walk from window edge to window edge. That is deliberate. It means the program and the picture are the same walk, and step 7 checks it.

7. Press [F2], `RUN`, and let the hundred and twenty-seven steps run. The run screen answers `RUN P3` over `LINE 9` with `1.9796408183537`.

   Against the plot's own `1.9796408183505` from step 5. Twelve digits agree. Plot and program are one walk, and now the walk costs one line to change instead of twenty presses.

8. So take a shot. Press [PRGM] for the list, [F1] for the editor, which reopens at line 1, then [CLEAR], type `11→Y`, and press [F2]: `2.014907003357`.

   One shot short at 10, one shot over at 11. The target is between them.

Aiming by straight line

9. Press **PRGM** for the list, **EXIT** for the home screen, and **CLEAR**, then measure the two misses. `2-1.9796408183537` answers = `0.0203591816463` and, after **CLEAR**, `2.014907003357-2` answers = `0.014907003357`.

   One fell short by 0.0204 and the other went over by 0.0149. If the miss were a straight-line function of the seed, the right seed would be at 10 plus the first miss over the sum of both. Press **CLEAR** and type `10+.02036/(.02036+.01491):` = `10.577261128437`.

10. Press **CLEAR**, **PRGM**, **F1**, **CLEAR**, type `10.577->Y`, and press **F2**: `2.0006674214272`. Much closer, and still a hair over.

11. Aim again with the newest pair: 0.00067 over at 10.577 against 0.02036 short at 10. Press **PRGM**, **EXIT**, **CLEAR**, and type `10+.02036*.577/(.02036+.00067):` = `10.558617213504`.

    Press **CLEAR**, **PRGM**, **F1**, **CLEAR**, type `10.559->Y`, and press **F2**:

The fourth shot landing on the consent condition

`2.0000403856688`. Four shots have found the inlet the bed can take: a little over 10.5 milligrams per litre.

A second opinion, and an uncomfortable answer

12. This model has a closed form, which is exactly the kind of luck you should exploit whenever you get it. A square-law decay integrates to give the outlet as the inlet divided by 1 plus 0.4 times the inlet.

    Press **PRGM**, **EXIT**, and **CLEAR**, type `X/(1+.4*X)-2`, and press **2nd** **GRAPH** for the solver workspace. Press **F5** twice for the `GUESS` page and store 8, then

bounds 5 and 20, ENTER after each. Press F1, `SOLV`: a `ROOT` of `9.999980926514` with `RES-7.629406E-7`.

The mathematics says 10. The shooting said 10.559.

That gap is not the shooting's fault and it is worth being clear about where it comes from, because the temptation is to blame the aiming.

Euler undershoots a curve that bends upwards. So the machine's reed bed removes slightly *more* pollutant than the real one, and asking it to hit 2 at the outlet means feeding it slightly more at the inlet than the real bed would need. The error is in the integrator and it comes out dressed as a modelling answer, which is the most dangerous costume it could have chosen.

13. Refine the walk and the answer walks back. Press EXIT, PRGM, F1, then edit lines 1 to 3 to `10->Y`, `20/254->H` and `254->N`, pressing CLEAR before each, and press F2. The doubled count takes its time: `1.9898413540469`.

    An undershoot of 0.0102 where the coarser walk undershot by 0.0204. Exactly halved, as a first-order method promises. Shooting the finer walk lands at 10.265, whose run reads `1.999981815175`: halfway back to 10 from 10.559.

So: the machine automates the shot and not the aim. The solver hunts the root of an expression, and no expression on Free85 runs an Euler walk, so nothing can go in the `F=` line that would let `SOLV` do the shooting for you. The outer loop is yours, and a straight line through the last two misses is the whole method.

What the solver *can* do is what step 12 did: certify a shot against a closed form where one exists, and show you that the difference between the two answers is the integrator's error wearing a modeller's coat.

TRY IT

1. Halve the walk twice more, `20/508->H` with `508->N`, and shoot again. Predict the answer first from the halving pattern of step 13, then run it. Does it keep halving its distance from 10?
2. Change the consent to 1.5 milligrams per litre and shoot for the new inlet, then check it with the solver on `X/(1+.4*X)-1.5`. Now work out why the bed has no answer at all for a consent of 3, and say what that means physically.
3. Aim on the bed's *length* instead of the inlet. Put line 1 back to `10->Y`, change line 2 so the step is your chosen length over 127, and find the length at which the outlet meets 2. Step 12's closed form checks this one too, with the 0.4 replaced by 0.02 times the length.
4. Step 9 aimed with a straight line through two misses. Try aiming with just one: from the shot at 10, guess a correction by eye, and see how many shots it costs you. Then say what the second miss is actually buying.
5. The closed form of step 12 exists because the equation separates. Solve dy/dx = -0.02 y squared on paper from y(0) = c, and check that your answer gives step 12's expression at x = 20.

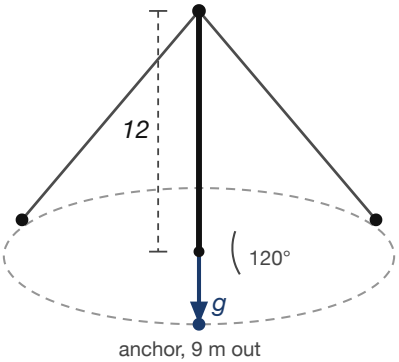
Vectors in the round

8.5

Two dimensions let you get away with signed numbers and a convention. Three do not. A force has a direction that needs naming, a moment has an axis, and the geometry of a site arrives as bearings and distances rather than as coordinates.

Free85's vector editor holds three components, which is exactly the number that makes a cross product mean something, and its coordinate pages translate between the surveyor's description and the algebra's.

The site is a radio mast 12 metres tall at the origin, held by three guys running from its top to ground anchors 9 metres out on bearings of 0, 120 and 240 degrees, each tensioned to 1500 newtons.



The guyed mast, with the three anchors at 120 degrees to each other and one guy vector marked from the mast top to its anchor

1. Bearings are degrees, so put the machine in degrees. Press [2nd] [MORE] for the mode screen and press [F1], ANG, once: the second line changes from ANGLE RAD to ANGLE DEG. Press [EXIT].

   Remember to put it back before Chapter 8 is over. Section 4.1 has the cautionary tale.

2. The second anchor arrives as a bearing and a distance, which is a cylindrical triple. Press [2nd] [8] (the VECTR legend) for the vector editor, which opens on SIZE 3 with the RECTV tag and a fresh A of zeros. Type [9] [ENTER] [1] [2] [0] [ENTER] [0] [ENTER].

   Press [MORE] [MORE] for the third soft-key page, R>CY CY>R R>SP SP>R, and press [F2], CY>R. Register R opens on COMP 1 reading −4.5000000581249, and [▶] reads 7.794228626888, with 0 beneath.

   The design values are -4.5 and 7.7942286341. The dust in the eighth digit is the price of a degree-to-radian conversion in fourteen digits, and knowing which number goes on the drawing is part of the job. Nobody is cutting cable to eight decimal places.

3. Now the first guy, which needs no conversion at all: from the top at (0, 0, 12) to the anchor at (9, 0, 0) is the vector (9, 0, -12).

   Press [EXIT] and [2nd] [8] again, which restores register A, component 1, and the first soft-key page in one move, and type [9] [ENTER] [0] [ENTER] [(-)] [1] [2]

ENTER.

Press F1, MAG: a SIZE 1 result reading 15. That is the length of cable to cut. Press F2, NRM: the unit vector, 0.6, then 0, then −0.8 as ▶ steps through it. Those direction cosines say the guy runs three fifths of its length outward for four fifths downward.

4. The angle between two guys is a dot product away. The second runs from the same top to step 2's anchor, so it is (-4.5, 7.7942286341, -12).

   Press EXIT and 2nd 8, type the first guy into A again, press ALPHA for B, and type the second with the digits and the (-) key. Press ALPHA to come back to A, then F3, DOT: 103.5. Press F5, ANG: 62.612892497387 degrees.

   Two guys 120 degrees apart on the ground stand only 62.6 degrees apart in the air, because both lean the same way, inward and down. Predict that number before you compute it next time and you will find it is harder than it looks.

Moments, and why masts are guyed in threes

5. A moment is a cross product, and this is the one place where three components are not a convenience but the whole subject.

   Each guy is 15 metres long and pulls with 1500 newtons, so a tension is a hundred times its guy vector: the first pulls the mast top with (900, 0, -1200) newtons at the position (0, 0, 12) metres from the base.

   Press EXIT and 2nd 8, type the position 0 ENTER 0 ENTER 1 2 ENTER, press ALPHA, type the force 9 0 0 ENTER 0 ENTER (-) 1 2 0 0 ENTER, press ALPHA again, and press F4, CRS.

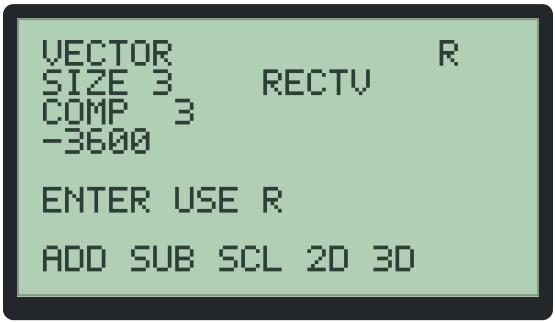
   Stepping through R reads 0, 10800, 0. That is 10800 newton metres about the y axis alone, trying to fold the mast over towards the anchor.

6. One guy would flatten the mast, so the question is what three do together. Because all three tensions act at the same point, their moments add up to the moment of their sum, which saves you two cross products.

   A hundred times step 4's B gives the second tension, (-450, 779.42286341, -1200), and the third, at 240 degrees, is its mirror, (-450, -779.42286341, -1200).

   Press EXIT and 2nd 8, type the first force into A, press ALPHA, type the second into B, press ALPHA, then press MORE for the second soft-key page and F1, ADD: 450, 779.42286341, −2400.

Carry that into A the way Chapter 6 carries a result: with R on the screen the prompt reads ENTER USE R, so press [ENTER]. Put the third force into B and press [MORE] [F1] again:



Three guy tensions adding to a pure downward pull

R reads 0, then 0, then −3600.

The guys pull the mast top sideways not at all and downward with 3600 newtons. A vector parallel to the mast crosses the mast's own position vector to give nothing, so the three moments of 10800 newton metres cancel exactly and leave the base in pure compression.

That is the entire reason for guying a mast in threes, and it is a three-line calculation.

Area, volume, and a distance you cannot see

The cross and dot products have three more jobs that a statics problem will hand you sooner or later, and none of them needs a new key.

7. The *magnitude* of a cross product is the area of the parallelogram the two vectors span. Take a fresh pair to keep the numbers clean: press [EXIT] and [2nd] [8], type [3] [ENTER] [4] [ENTER] [(-)] [1] [ENTER] into A, press [ALPHA], type [(-)] [7] [ENTER] [1] [0] [ENTER] [0] [ENTER] into B, and press [ALPHA].

   Press [F4], CRS: R reads 10, 7, 58.

   Now carry that into A and take its length: press [ENTER] on the ENTER USE R prompt, then press [F1], MAG: 59.27056605095. Nothing was retyped, so nothing could be mistyped.

   That is the area of the parallelogram spanned by the original pair, and half of it is the area of the triangle they make. Check it a second way if you like: the two lengths are 5.0990195135925 and 12.206555615734 and the angle

between them is `1.2605820039615` radians, and length times length times the sine of the angle gives the same area.

8. Take it one dimension further. The *scalar triple product*, the dot of a cross with a third vector, is the volume of the box the three of them span.

   With (10, 7, 58) still in `A`, press **ALPHA**, type **2** **ENTER** **(-)** **5** **ENTER** **1** **7** **ENTER** into `B`, press **ALPHA**, and press **F3**, `DOT: 971`.

   So the three original vectors span a box of volume 971. And the sign matters: a negative answer would mean the three vectors form a left-handed set rather than a right-handed one. A zero answer would mean the box is flat, which is to say the three vectors lie in a plane, which is the cheapest coplanarity test there is and worth remembering.

9. Finally the surveyor's other description. Press **EXIT** and **2nd** **8**, type the first guy (9, 0, -12) into `A`, press **MORE** **MORE** for the conversion page, and press **F3**, `R>SP`.

   The tag beside `SIZE` becomes `SPHEREV` and `R` reads `15`, then `0`, then `143.13010235415`: the cable length, its bearing, and its angle down from the upward vertical. So the guy leaves the mast 53.13 degrees below the horizontal.

   That tag is a note, not a label. It records the last conversion performed rather than tracking each register, so it survives leaving and re-entering the editor and will sit over plainly rectangular data until a `CY>R` or an `SP>R` sends it back to `RECTV`. Read it as a reminder of what you last did, not as a statement about what is in front of you.

10. Press **2nd** **MORE** and **F1** to put the machine back into `ANGLE RAD` before you leave the chapter.

TRY IT

1. Work out the third guy's anchor from its bearing with `CY>R` as step 2 did, and check that its cable is 15 metres too. Predict both components before you press `CY>R`.
2. Compute the moment of the second guy's tension with `CRS`, then take its magnitude with `MAG`. Should it be 10800 as well? Answer before you compute, from the symmetry of the site.
3. Move the anchors to 6 metres out instead of 9, keeping the tension. Which way does the downward force move, and which way the angle between two guys? Answer both before you press a key, then check.
4. Use the triple product of step 8 as a coplanarity test. Build three vectors you know lie in a plane, and confirm the answer is zero. Then nudge one component and watch how quickly it stops being zero.
5. Two guys and the mast itself are three vectors from the same point. Work out the volume of the box they span, and say what a small answer would tell you about how well the mast is braced.

CHAPTER

9

Solutions

Every Try it exercise in the book, worked through.

Where an exercise asks you to predict something first, the solution says what the right prediction was and, more usefully, what makes it right. Where it asks you to press keys, the keys and the numbers that come back are here. Where the interesting part of an exercise turned out to be somewhere other than the question, the solution says so and goes there.

A word on the numbers. Every one below came off the emulator, so if yours differ in the last digit or two, check the angle mode first and the stored equation second: those two account for almost every discrepancy. A difference in the last digit alone is usually just a different route to the same answer, as section 3.6 found.

Solutions for Chapter 1 9.1

1.1 Functions and their windows

1. The three crossings of `X^3−4*X` and `X` are where x cubed minus four x equals x, so x cubed equals five x, so x is 0 or plus or minus the square root of five.

Section 1.1 found the negative one at `= −2.2360679774997`. For the other two, trace to the neighbourhood first: the search reads the traced position, so trace near the origin and press [2nd] [F1] for `= 0`, then trace right of 2 and press [2nd] [F1] again for the positive root.

Check it: press [CLEAR] and type [2nd] [x²] [5] [)] for `SQRT(5): = 2.2360679774998`.

Notice the last digit. The intersection search answered `2.2360679774997` and the square root answers `...98`. Neither is wrong; the search stops when its bracket is

tight, and the root is computed. Section 4.4 has more on why searches stop where they do.

2. `X^3−4*X+3` factors as $(x - 1)(x^2 + x - 3)$, so its zeros are 1 and the two roots of $x^2 + x - 3$, which are about 1.303 and -2.303.

The zero at 1 is the one you could have read from the table, because the table steps in whole numbers by default and lands exactly on it. The other two fall between rows and need the search.

That is worth generalising: the table finds a zero only when it happens to step on one, so a zero at a whole number is visible and a zero anywhere else is not.

3. Near the origin the cube term is negligible against the linear one, so `X^3−4*X` behaves like `−4*X`. The curve comes to resemble the line $y = -4x$.

Reading `XMIN` after each press of [+], from the standard window: `−5`, `−2.5`, `−1.25`, `−0.625`, `−0.3125`. Each press halves it, as section 1.1 said.

By four or five presses the curve is straight to the eye. Check the slope while you are there: trace two columns from the origin and divide the `Y=` readout by the `X=` readout, and you should get something very close to -4.

4. For one zero, zoom in until only the origin is in view: three or four presses of [+] leaves the window at -1.25 to 1.25, which contains only the zero at 0. For none, trace away from the origin first and then zoom, or use a window that sits entirely to the right of 2.

Neither window is lying. Each is answering the question "what does this function do *here*", and the answer genuinely is "it has one zero" or "it has none" on that interval. The mistake is only ever in reading a local answer as a global one, which is the whole reason section 1.1 comes first.

1.2 Families of curves three at a time

1. All three lines have slope 1, so they are parallel. They cross the y axis at 4, 0 and -3, which are the constants themselves.

The table makes it plain: at `X=0` the row reads `4`, `0`, `−3`. Adding a constant to a function moves its graph vertically by that constant, which is section 1.3's rule arriving early.

2. Store `X^2` in `Y1`, `−X^2` in `Y2` and `−4*X^2` in `Y3`, using [(-)] for the signs.

The negative members open downwards. In the table the `X=2` row reads `4`, `−4`, `−16`: the sign of a flips the bowl, and the size of a decides how narrow it is. The two effects are independent, which is why one family can show both.

3. `(X+2)^2−5` has its vertex where the bracket vanishes, at x = -2, and its value there is -5.

Confirm with the table: the `X=−2` row reads `−5`, and the rows either side read `−4`, so the curve is symmetric about -2 and turns there.

For a vertex at (1, 7) the slot text is `(X−1)^2+7`. Test it the same way: the `X=1` row should read `7` with `8` either side. Mind the sign inside the bracket, which is the part everybody gets backwards once.

4. Any five-member family works. The point of the exercise is the choice of anchor, and the answer is that the anchor should be the member you most want the others measured against, usually the middle one or the simplest one.

For `X^2/4`, `X^2/2`, `X^2`, `2∗X^2`, `4∗X^2`, keep `X^2` in both plots. It is the one whose shape you already know, so both pictures are read against the same reference, and the two halves of the family can be compared even though they were never on screen together.

Choose a different anchor and the two plots have nothing in common, which makes them two experiments rather than one.

1.3 Symmetry and transformations

1. `X^2−6` is even: replacing x by -x changes nothing, because the square kills the sign. `X^3+1` is neither: f(-x) is $-x^3 + 1$, which is neither f(x) nor -f(x).

On screen, the even test shows two curves rather than three, because `Y1` and `Y2` coincide. The neither test shows all three curves separately.

In the table, evenness is `Y1` and `Y2` agreeing down every row; oddness is `Y2` and `Y3` agreeing; neither is no two columns agreeing anywhere. Reading *which* pair matched is the skill, not just noticing that something did.

2. They give the same picture because `2∗ABS(X)` and `ABS(2∗X)` are the same function: the absolute value of 2x is twice the absolute value of x.

The table confirms it, both columns reading `0`, `2`, `4`, `6` at `X` = 0, 1, 2, 3.

They differ for any function that is not homogeneous of degree one. Try `2*X^2` against `(2*X)^2`: the first doubles heights, the second quadruples them, and the table separates them at once.

3. Anything with a small odd part added to a large even part will do. `X^2+X/100` looks symmetric in the standard window, because the `X/100` contributes at most a tenth over the visible range and the plot cannot resolve it.

The table exposes it immediately: at `X=5` the value is `25.05` and at `X=-5` it is `24.95`. The picture cannot show a twentieth of a unit; the table can show fourteen digits.

4. Yes, and exactly one: the zero function. Even means f(-x) = f(x) and odd means f(-x) = -f(x). Both together give f(x) = -f(x), so 2f(x) = 0, so f is zero everywhere.

Store `0*X` in all three slots and the three-slot test shows a single line along the axis, with the table reading `0` across every row. It is a degenerate answer and it is the only one, which is what makes the question worth asking.

1.4 Rational functions and the lines they approach

1. `(X^2-4)/(X-2)` divides out exactly: it is (x - 2)(x + 2) over (x - 2), which is x + 2 for every x except 2.

The table reads `UNDEF` at `X=2`, exactly as the pole did. But this is a completely different situation. Probe either side and you see it: with the equation stored, `EVAL(1.9)` answers `= 3.9` and `EVAL(2.1)` answers `= 4.1`, and `EVAL(2)` stops at `DIVIDE BY ZERO`, which names exactly what it met there.

The values from both sides are heading for 4, which is what x + 2 gives at x = 2. The function has a *hole*, not a pole: one point missing from an otherwise perfectly ordinary line. Chapter 4's `SIN(X)/X` is the same shape of trouble.

So `UNDEF` in a table means only that the machine could not compute a value. The neighbouring rows tell you which kind of nothing you are looking at.

2. Dividing $2x^2$ - x + 3 by x + 1 gives 2x - 3 with a remainder of 6, so the function is 2x - 3 plus 6/(x + 1), and the slant asymptote is y = 2x - 3.

Put `(2*X^2-X+3)/(X+1)` in `Y1` and `2*X-3` in `Y2`. At `X=10` the formula predicts a gap of 6/11.

Check all three: press CLEAR and spell `EVAL(10): = 17.545454545455`. Press CLEAR and type `2*10-3: = 17`. Press CLEAR and type `6/11: = 0.54545454545455`.

17 plus 0.5454… is 17.5454…, to every digit. The division was right.

3. The gap between `(X^2+1)/(X-1)` and its asymptote is 2/(x - 1), which is never zero for any finite x, because 2 is not zero. So the curve approaches the line and never meets it.

To make one that *does* cross, you need a remainder that changes sign, which means a remainder with a zero in it. `(X^3+1)/X^2` is x plus $1/x^2$, whose remainder never changes sign either; better is `(X^3-X)/(X^2+1)`, which is x minus $2x/(x^2 + 1)$. That remainder is zero at x = 0, so the curve crosses its asymptote y = x at the origin. Plot both and look.

Crossing an asymptote is not misbehaviour. An asymptote says where a curve *ends up*, not where it is forbidden to go.

4. `(X^2+1)/(X^2-1)` is 1 plus $2/(x^2 - 1)$, so its horizontal asymptote is y = 1 and the gap is $2/(x^2 - 1)$. For that to be under a thousandth you need $x^2 - 1$ above 2000, so x^2 above 2001, so x above 44.73.

Check both sides of the boundary. Press CLEAR and spell `EVAL(44.73): = 1.0010001135629`, a gap of 0.0010001, just over. Press CLEAR and spell `EVAL(45): = 1.0009881422925`, a gap of 0.00098814, just under.

And press CLEAR and type `2/(45^2-1): = 0.00098814229249012`, which is the gap the algebra predicted, to eleven digits.

That is a much faster convergence than section 1.4's slant case, where the gap was 2/(x - 1) and you had to go out to 2001. Squaring the denominator squares how quickly the curve settles.

1.5 Exponential and logarithmic functions

1. The mirror symmetry is still true and stops *looking* true. `EXP(X)` and `LN(X)` are reflections of each other in y = x whatever window you use, but a reflection in a forty-five degree line only looks like one when the two axes are drawn at the same scale.

The standard window is wider than it is tall in real distance on the screen, so the line y = x does not sit at forty-five degrees, and the reflection is sheared with it.

The window carries the blame because the mathematics has not changed at all: only the drawing has.

2. Base two passes 8 at x = 3, because 2 cubed is 8. Predict that from the powers of two rather than from the plot.

Check it: press **CLEAR** and spell `LN(8)/LN(2): = 2.9999999999965`. With `EXP(X*LN(2))` stored and the plot allowed to finish, press **CLEAR** and spell `EVAL(3): = 8.0000000000261`.

Both answers carry dust in the last digits, and from opposite directions. That is two logarithms and an exponential each rounded to fourteen places; the mathematics is exact and the arithmetic is not.

3. Halving every unit means a base of a half, so the slot text is `EXP(X*LN(.5))`.

After five units, a half to the fifth is one thirty-second. Press **CLEAR** and spell `EVAL(5): = 0.031250000000139`. Press **CLEAR** and type `1/32: = 0.03125`.

The read from the table agrees to four decimals in its five-character cell.

4. Press **CLEAR** and type `1.06^9: = 1.6894789590028`. Press **CLEAR** and spell `EXP(9*LN(1.06)): = 1.6894789590072`.

They agree to eleven digits and differ in the last three.

Trust the power key. It is nine exact multiplications of a stored number, and the only error is the rounding at each step. The identity route takes a logarithm, multiplies, and takes an exponential, each of which rounds, and the exponential magnifies whatever error the logarithm made.

The identity is not less accurate because it is cleverer. It is less accurate because it goes the long way round, and it exists for the cases where the short way is not available at all.

5. All three work. What differs is the route each one takes, and therefore whether the answer is exact.

`1.06^-3` and `1.06^10` have whole-number exponents, so both go by repeated squaring and both are exact: `= 0.8396192830323` and `= 1.7908476965428`. `1.06^0.5` does not, so it goes through the logarithm and the exponential, and answers `= 1.0295630140986`.

Checking with the identity is the test, and it separates the two routes cleanly. `EXP(10*LN(1.06))` answers `= 1.7908476965481` against the key's `= 1.7908476965428`: the identity took the long way round for an exponent that did not need it, and the last three digits paid for it. `EXP(0.5*LN(1.06))` answers `= 1.0295630140986`, agreeing with `1.06^0.5` to every digit, because for a fractional exponent the identity is not an alternative to what the key does. It is a description of it.

1.6 Trigonometric functions

1. `SIN(X)+2` lifts the whole wave two units, keeping its shape. `SIN(X+2)` slides it two units to the *left*, keeping its height. The first is a change outside the sine, the second is inside it, which is section 1.3's rule again.

The table settles it: at `X=0` the three columns read `0`, `2` and `0.909`. The second is 2 above the first. The third is the value the plain sine has at x = 2, arriving two units early.

2. In `DEG` mode a full wave is 360 degrees wide, so you need `XMAX` of at least 180 to fit a half-wave either side of the origin, and 360 to be comfortable.

Pressing [-] from the standard window doubles the bounds each time, so `XMAX` reads 20, 40, 80, 160, 320, 640. The first window wide enough for a whole period is the one at 320, since the visible range is then -320 to 320, which is 640 degrees.

You could have predicted it: 360 lies between 160 and 320, so it is the sixth press. Doubling gets you anywhere quickly and never lands where you would have chosen.

3. A one-hour swing gives `1*SIN(PI*X/6)`, or just `SIN(PI*X/6)`.

The standard window hides it almost entirely: the curve never leaves -1 to 1 while the window allows for -10 to 10, so it is a flat ripple on the axis. It is section 1.6's opening complaint all over again, and the fix is the same: zoom in vertically until the wave fills the screen.

What the standard window hides is not the shape but the *scale*. Both towns have a sine of period twelve months. Only the amplitude differs, and amplitude is exactly what a badly chosen vertical range destroys.

4. Sine is 0, 1, 0, -1 at x = 0, π/2, π and 3π/2. Cosine at those four places is 1, 0, -1, 0: it does the same thing a quarter turn earlier, which is what section 1.6 step 4 showed.

Check them. Press CLEAR and spell `COS(0): = 1`. Then `COS(PI/2): = -6.51527649229E-11`. Then `COS(PI): = -1.0000041678092`. Then `COS(3*PI/2): = -6.51527649229E-11`.

The last three are 0, -1 and 0 under the machine's fourteen-digit `PI`, which is not quite pi. Read `-6.5E-11` as zero and `-1.0000042` as minus one. That dust appears in section 5.4's cardioid too, and it is worth learning to recognise rather than chase.

5. Six hours thirteen minutes is 6.2166666666666 hours. A sine repeats every 2 pi, so you want the bracket to reach 2 pi when x reaches the period: the model is `SIN(2*PI*X/6.2167)`, or equivalently `SIN(1.0107*X)` since press CLEAR and spelling `2*PI/(6+13/60)` answers `= 1.0107000494123`.

Two days is 48 hours, which is a little under eight periods. The standard window's -10 to 10 shows barely three, so you want a window several times wider in x and much narrower in y. Press - twice for x and zoom in vertically, or accept the trig window and read only part of the picture.

1.7 Inverse functions by parametric pair

1. The inverse crosses the x axis where the original crossed the *y* axis, because the coordinates have been swapped. The original line 2t - 6 crosses the y axis at -6, so the inverse crosses the x axis at -6.

Store `2*X-6` in slot 1 and `X` in slot 2 for the inverse pair, plot it, and trace to where the `Y=` readout passes through zero: the `X=` readout there is -6.

That is the whole content of "swap the coordinates", and it is worth checking on a line before trusting it on a curve.

2. They must agree because the exponential and the logarithm *are* inverses. Drawing `EXP(t), t` as a parametric pair plots the points (e^t, t), and the graph of `LN(x)` is the set of points (x, ln x). Put x = e^t and ln x = t and those are the same points.

The two pictures are the same curve computed two different ways, so any disagreement between them would be an arithmetic error rather than a

mathematical one.

3. The pair `X^2, X` plots the points (t^2, t), which is a sideways parabola: two y values for most x, and therefore not a function.

A function slot computes y from x and can only ever produce one y per x. The parametric sweep does not: it walks t from `XMIN` to `XMAX` and puts a point wherever the pair says, with no rule against visiting the same x twice. That is exactly the freedom that lets it draw an inverse.

Trace it and watch the `X=` readout come *down* to zero and go back up while the `Y=` readout climbs steadily. A function slot could never produce that trace.

4. Any function that is its own inverse works, and there are more than you would guess. Two of different shape:

The line y = x itself, trivially, and more interestingly y = -x, whose swapped pair draws the same line. And the hyperbola y = 1/x: swapping gives x = 1/y, which is the same relation.

`6−X` works too, and so does any line of slope -1. The general fact is that a function is its own inverse exactly when its graph is symmetric about the line y = x, which is a nice thing to be able to spot by eye.

Solutions for Chapter 2

9.2

2.1 Prices from receipts

1. Press 2nd STAT and enter 3, 2, 12.3, then 2, 3, 10.7. `SOLVE` answers `X 2.7` and `Y 2.1`: juices are 2.70 and flapjacks 2.10.

Check both receipts by hand, which is the habit worth keeping: three juices and two flapjacks is 8.10 plus 4.20, which is 12.30. Two and three is 5.40 plus 6.30, which is 10.70. Both right.

2. Predict first. The batch is bigger and dearer per kilogram, 105 over 12 rather than 84 over 10, so 8.75 a kilogram against 8.40. You might reasonably expect more of the dear leaf.

Enter 1, 1, 1, 12, then 12, 9, 6, 105, then -2, 0, 1, 0. `SOLVE` answers `X 1`, `Y 9` and `Z 2`.

One kilogram of the dear leaf, not two. The proportions have not scaled at all: the first batch was 2, 4, 4 and this one is 1, 9, 2. The mix has swung almost entirely

into the middle leaf.

If that surprises you, it should. The proportion condition z = 2x ties the cheapest leaf to the dearest, so the only free direction left is the middle one, and it absorbs the whole change. Two constraints and three unknowns leave much less freedom than it looks.

3. For no solution, make the coefficients proportional but the takings not: 2, 4, 10 then 1, 2, 6. Doubling the order must double the price and here it does not, so `SOLVE` answers `NO SOLUTION`.

For infinitely many, make everything proportional: 2, 4, 10 then 1, 2, 5. `SOLVE` answers `UNDERDETERMINED`.

The two are one keystroke apart, which is the point of section 2.1 step 5.

4. Any third receipt whose coefficients are a combination of the first two but whose takings are not. The first two are 3, 2 and 2, 3; their sum is 5, 5, and the takings should then be 12.30 plus 10.70, which is 23.00. So the receipt 5, 5, 23 is consistent and 5, 5, 24 is not.

Grow the editor to `3X3` and enter all three: with 23 the answer is still `X 2.7`, `Y 2.1`; with 24 it is `NO SOLUTION`. A third receipt cannot add information, only agreement or contradiction.

2.2 When the answer will not stay still

1. Predict: the two lines are ten times closer to parallel, so the trouble should be about ten times worse.

Enter 1, 2, 8, then 1.001, 2, 8.05. `SOLVE` answers `X 50` and `Y −21`. Now nudge the 8 to 8.1: `X −50` and `Y 29.05`.

x has swung from plus fifty to minus fifty, where the 1.01 pair swung from plus five to minus five. Ten times closer to parallel, ten times the amplification, exactly as predicted. And note that the *unnudged* answer is already absurd: minus twenty-one pastries.

2. Perpendicular means the coefficient vectors have zero dot product, so take x + 2y = 8 with 2x - y = 1.

`SOLVE` answers `X 2` and `Y 3`. Nudge the 8 to 8.1 and it answers `X 2.02` and `Y 3.04`.

A one and a quarter per cent change moved the answers by one and a half. That is as good as it gets, and the moral for planning measurements is direct: arrange your observations to be as unlike each other as possible. Two nearly identical experiments tell you nearly nothing twice.

3. With the coefficient exactly 1 the two left-hand sides are identical, so the equations read x + 2y = 8 and x + 2y = 8.05. Those contradict each other, and `SOLVE` answers `NO SOLUTION`.

That is the honest end of the road. As the coefficient slides from 1.01 to 1.001 to 1, the answer runs off to infinity and then stops existing. The ill-conditioned case is not a separate phenomenon from the impossible one; it is the impossible one seen from very close up.

4. Nudge the batch value 84 by one per cent to 84.84 and re-solve.

The original answer was `X 2`, `Y 4`, `Z 4`. The nudged one is `X 1.72`, `Y 4.84` and `Z 3.44`.

A one per cent change in one number has moved the dear leaf by fourteen per cent and the middle leaf by twenty-one. So the tea blend is moderately ill-conditioned too, amplifying by fifteen or twenty, which nobody would have guessed from looking at it. Section 6.3 puts a number on that with `COND`, and this system is worth running through it when you get there.

2.3 The best plan on a graph

1. Predict from the slope. The profit line 60x + 30y = k solves to y = (k - 60x)/30, a slope of -2, which is steeper than either constraint line. A steep profit line slides out to the corner furthest right.

Evaluate the corners: `60*4+30*6` answers `= 420`, `60*8+30*0` answers `= 480`, and `60*0+30*8` answers `= 240`.

So (8, 0) wins with 480, where the old profit chose (4, 6). Making bookcases twice as profitable as benches moves the whole plan to bookcases only, and it is the *slope* of the profit line that decided it, not its height.

2. Careful here, because the exercise points you at the wrong line.

The cap y = 5 meets the timber line x + 2y = 16 at x = 6, so (6, 5). But check it against labour before you trust it: `3*6+2*5` answers `= 28`, and only 24 hours exist. The point is not feasible.

The cap meets the *labour* line 3x + 2y = 24 at 3x = 14, so x = 14/3. Press CLEAR and type `14/3:` `= 4.6666666666667`. Check timber there: 14/3 plus 10 is about 14.67, under 16, so this one is feasible.

Profit: press CLEAR and type `30*(14/3)+40*5:` `= 340`.

The whole-number plans worth checking are (4, 5) and (5, 5). Neither is the rounded corner rounded blindly: (5, 5) needs 25 labour hours and fails, so (4, 5) at 320 is the best whole plan. Integer answers are a genuinely harder problem than the corner method solves, and this is the cheapest possible demonstration of why.

3. Raise labour from 24 to 26 and re-solve the crossing: enter 1, 2, 16 then 3, 2, 26. `SOLVE` answers `X 5` and `Y 5.5`.

Profit there: `30*5+40*5.5` answers `= 370`, against 360 before. Ten pounds for two extra hours, so an hour is worth 5.

Timber was worth 15 a sheet. So buy timber first, and it is not close.

4. As timber rises the best corner slides up the labour line towards its y intercept, which is at y = 12. Getting there needs timber of 2 times 12, which is 24 sheets.

At 24 sheets the timber constraint passes exactly through the labour line's intercept, and beyond that it is no longer binding: more timber buys nothing and its shadow price drops to zero. Profit there is 40 times 12, which is 480, and it stays 480 however much timber you buy.

5. Any line through (4, 6) works. Glue at 2 units a bookcase and 1 a bench, with 14 pots a week, gives 2x + y = 14, which passes through (4, 6) exactly.

Adding it leaves the best plan unchanged because it touches the region at the corner that was already best and cuts nothing off. Its shadow price is zero: raise the glue to 15 pots and re-solve, and the answer does not move, because the constraint was never the thing holding you back.

A constraint that is satisfied exactly at the optimum but has zero shadow price is called degenerate, and it is a genuinely awkward case in real linear programming. You have just built one on purpose.

2.4 Elimination as bookkeeping

1. The tableau is 2, 5, 7.9 then 4, 5, 12.3, and no swap is needed because the top-left is already nonzero.

There are four scales. First -2, to clear the 4 below the pivot, which leaves row 2 as 0, -5, -3.5. Then -0.2, to turn that -5 into a 1, leaving 0, 1, 0.7. Then -5, to clear the 5 above it. Then 0.5, to turn the leading 2 into a 1.

The finished tableau reads 1, 0, 2.2 and 0, 1, 0.7: coffee 2.20, pastry 0.70, which is section 2.1's answer arrived at the slow way.

2. Put the coefficients 2, 1, 1, 3 in a 2 by 2 `A` and the takings 110, 130 in a 2 by 1 `B`, then press `AUG`.

The result lands in `R`, as every result does. Before row operations can touch it you must carry it into `A` by hand, which is the same cost section 2.4 charges everywhere: the row-operation keys read `A` and nothing else.

3. Predict first: `RREF` sorts the rows into echelon form, and the row starting with a 1 in the first column must end up on top. So the print row comes back above the frame row whatever order you feed them in.

Enter 0, 1, 30 then 1, 0, 40 and press `RREF`: `R` reads `1, 0, 40`, then `0, 1, 30`. The answer is unchanged and the rows have swapped back.

4. From steps 2 to 6 the framer's 2 by 3 tableau took one swap and three row operations, each of which touches three cells: about a dozen cell updates.

A 3 by 4 tableau needs to clear two entries below the first pivot, then one below the second, then two above, then three rescalings: six operations of four cells each, so around twenty-five updates, plus the swaps.

The work grows like the cube of the size, which is the thing worth taking away. Twelve updates against twenty-five for one extra row and column, and a 3 by 6 tableau costs more again. That growth, not the size of the register, is what limits elimination by hand: the register holds three rows by six columns, and by the time you have filled it you will not want to audit every step anyway.

2.5 The mathematics of money

1. Predict from the doubling. Trebling needs ln 3 over ln 2 as many years as doubling, which is about 1.585 times 11.9, so about nineteen.

Store `1500->Z` for the target and solve with `VAR Y`: the root is about 18.85 years. Check on the home screen: press **CLEAR** and spell `LN(3)/LN(1.06)`: `= 18.854176679022`.

Note that the starting sum cancels again, exactly as it did for doubling. Ratios do not care how much you began with.

2. The adaptation is 8000 in place of 10000, 1.008 in place of 1.01, and 36 in place of 24. So the equation is

```
8000*EXP(B*LN(1.008))-A*(EXP(B*LN(1.008))-1)/.008
```

and it will not fit. That string is 49 characters and the entry line holds 48.

This is worth knowing rather than fighting. Since `B` is known here, work out the growth factor once and store it: press **CLEAR**, spell `EXP(36*LN(1.008))`, press **STO▶** **ALPHA** **EE** for the letter `G`, and press **ENTER**: `= 1.3322298368266`.

Now the equation is `8000*G-A*(G-1)/.008`, which is nineteen characters.

Press **2nd** **GRAPH**. The workspace opens on `VAR X`, so press **F3**, `VAR`, three times to step X, Y, Z, A. Fence the search at 0 and 1000 and press **F1**, `SOLV`: a `ROOT` of `256.6377251642` with `RES 2.56E-7`.

Three presses of `VAR` is the price of naming the unknown `A` in an equation whose other letter is `G`. Section 2.5's closing note is about exactly this: choose letters that march forward from X and you pay one press instead of three.

3. Store `470.73472221384->A`, reopen the workspace, press `VAR` for `VAR B`, set the upper bound to 100, and `SOLV`.

The root comes back at essentially 24, a few millionths off. The residual line reports the gap, and what it is telling you is that the payment was itself found by bisection to a tolerance, so feeding it back can only recover the term to the same tolerance. Errors do not shrink when you run a calculation backwards through them.

4. For a term of 100 months, solve with `VAR A` and `B` stored as 100: the payment falls to about 158. For 1000 months it falls to about 100.03.

They are converging on 100, which is exactly the monthly interest on 10000 at one per cent. That is step 9's trap seen from the other side: 100 is the payment that never repays anything, so any payment above it clears the loan eventually, and any payment at or below it never does.

The curve of payment against term has a horizontal asymptote, and the asymptote is the interest.

2.6 Loyalty in the long run

1. Start with 0, 0, 1 in a `SIZE 1X3 A`, keep P in `B`, and multiply, carrying each result back into `A`.

The Harbour's share runs 0.2, 0.32, 0.392, 0.4352, 0.4611. It first passes 0.45 on the fifth Saturday.

Guessing beforehand is instructive: the steady state is 0.5 and the first step gives 0.2, so you are covering about half the remaining distance each week, which puts 0.45 at around week five. That estimate is worth making because it is how you check a long iteration without running it.

2. Transpose P into `R`, carry it to `A`, put the 3 by 3 identity in `B` with the `ID` key, and press `SUB` to get P transpose minus I. Carry that result to `A` and press `RREF`.

The answer passes through `R` three times and `A` twice, which is four retypings to avoid one piece of paper arithmetic. Whether that is a good trade is a fair question, and the answer is usually no: this route exists to show that the balance equations are exactly P transpose minus I, not because it is quicker.

3. Predict first. The Station's loyalty rising from 0.6 to 0.8 is the biggest single change in the table, so expect it to gain, but the Harbour still collects 0.2 from both rivals and starts from 0.8 itself.

The new matrix is .8, .1, .1 then .2, .7, .1 then .1, .1, .8. The balance system is -.2, .2, .1 then .1, -.3, .1 then .1, .1, -.2, and `RREF` gives rows 1, 0, -1.25 and 0, 1, -0.75.

So the shares stand as 1.25 : 0.75 : 1, which is 5 : 3 : 4, and dividing by 12 gives 0.41667, 0.25 and 0.33333.

Yes, it reorders them. The Station climbs from a fifth to a third and overtakes the Mill, which drops from 0.3 to 0.25. The Harbour still leads but has lost eight points. Loyalty is worth more in the long run than any single week's switching suggests.

4. Starting at the steady state and multiplying once returns the same share-out, because that is what steady state means.

Starting at the *old* steady state with the *new* matrix does not: the town begins moving towards the new equilibrium, and the first step shows you the direction and the initial speed of that move.

The difference between those two runs is how quickly a change in service shows up. It is fast at first and then slows, because each week closes a fixed fraction of the remaining gap. Most of the effect of the Station's renovation appears within a month; the last of it never quite arrives.

9.3 Solutions for Chapter 3

3.1 A week of small data

1. The eight values total 132, so the mean is 16.5. Ordered, the middle two are 16 and 17, so the median is 16.5 as well.

Press `1V` and the screen answers `MEAN 16.5`, `MED 16.5`, `S SD 3.5456210417118` and `P SD 3.3166247903554`.

The two land not just close but exactly together, and the reason is that this week is symmetric: pair the values from the outside in and each pair averages 16.5. The keeper's week had one value dragging the mean four points away from the median, and this one has nothing dragging at all.

2. The sorted week is 12, 15, 15, 17, 18, 19, 21, 43. Shrinking with `-` drops the *last* entry, so 43 leaves.

That is the outlier, so the quartiles ought to move very little. `Q1` stays at 15 and `Q3` drops from 20 to 19, because with seven values the upper quartile now falls on 19 rather than between 19 and 21.

The quartile that moved is the one on the side the value left from. `Q1` never noticed, which is exactly why quartiles are worth having.

3. 658 over 8 is 82.25. Press `CLEAR` and type `658/8: = 82.25`.

Now square the `P SD` figure: press `CLEAR` and type `9.0691785736085^2: = 82.25`.

Exact, and it should be. The `P SD` line is the square root of the population variance, so squaring it undoes the root, and the machine keeps enough digits that nothing is lost on the round trip. Compare the `S SD` line of section 3.1 step 5, which sat a whisker under its true value; that one lost a digit because 94 has no exact square root and 82.25 does not need one.

4. Predict: the mean falls, the standard deviations fall a great deal, `MAX` falls, and `MED`, `MIN` and `Q1` do not move at all.

Enter 15, 12, 17, 15, 19, 21, 18, 19 and press `1V: MEAN 17, MED 17.5, S SD 2.8784916685157, P SD 2.6925824035673`.

The mean fell from 20 to 17 and the spread nearly collapsed, from 9.7 to 2.9. The median moved not at all, from 17.5 to 17.5.

That is one number changing three of the eight figures and leaving the rest untouched, which is the sharpest possible statement of what "resistant to outliers" means.

5. Yes, and it is easier than it sounds. You need the mean and median equal while the two halves are shaped differently.

Try 10, 16, 16, 17, 18, 19, 19, 25. The mean is 140 over 8, which is 17.5, and the middle two are 17 and 18, so the median is 17.5 as well. But the lower half spreads from 10 to 17 and the upper only from 18 to 25 in a different pattern, so the box plot's whiskers and box are not symmetric.

The lesson is that equal mean and median rules out gross skew and does not rule out asymmetry. Two numbers cannot describe a shape.

3.2 Random numbers that repeat

1. From a fresh machine, `RANDI(1,6)` ten times gives 3, 5, 3, 5, 6, 4, 6, 1, 1, 4.

Tally them: 1 twice, 3 twice, 4 twice, 5 twice, 6 twice, and 2 never.

That is as close to uniform as ten rolls of a six-sided die can be while missing a face: five faces with exactly two each. It looks like strong evidence of something and it is evidence of nothing at all. With ten rolls and six faces, the chance that some face is missing is better than sixty per cent, so a missing face is the *expected* outcome rather than a surprise.

Small samples are like this. Section 3.3's thirty-six rolls produce eight sixes and then thirteen, and neither is news either.

2. 1 is impossible because the smallest each die can show is 1, so the smallest sum is 2. Every sum from 2 to 12 is possible and they are not equally likely: there is one way to make 2 and six ways to make 7, so 7 is six times as likely.

Eight sums from a fresh machine give 8, 8, 10, 7, 5, 8, 5, 5. All of them land in the middle of the range, which is what the counting predicts.

3. Three calls to `RANDI(0,9)` advance the stream three draws. One call to `RANDI(0,999)` advances it one.

For most purposes it does not matter, and for one purpose it matters a great deal: if you want two experiments to be comparable, they must consume the stream the same way. Change how many draws a simulation takes and you have changed the experiment, not just its length.

4. Predict: it must return 1 every time, since 1 through 1 has one member. The interesting question is whether it still consumes a draw.

It does. Call `RANDI(1,1)` three times and you get `1`, `1`, `1`. Then press **CLEAR** and ask `RANDI(1,6):= 5`.

A fresh machine's first die is `3`. This one gives 5, which is the fourth value in the stream, so the three pointless calls each took a draw.

The generator does not know that the answer was foregone. It draws, then maps the draw onto the range you asked for, and a range of one maps everything to the same place.

3.3 Simulation by program

1. Line 1 changes, from `36–>N` to `99–>N`. Nothing else: the tally, the test and the display are all independent of the count, which is what makes this program worth having.

Expect around 99 over 6, which is 16 or 17. Run it and you will get something in the low tens to low twenties.

The wobble is proportionally smaller than at 36 rolls and still visible. The spread of a count grows like the square root of the number of trials while the count itself grows like the number, so the two pull apart slowly.

2. Adding `DISP 9–S` after the existing `DISP S` leaves the *tails* on show, because the run screen shows only the most recent `DISP`.

If you want the heads, put the new line first. That is worth knowing before you write anything longer: on this machine a program communicates one number, and the last one wins.

3. `REPEAT` tests before every pass, so the body runs while the test is nonzero. Starting from `9–>N` and decrementing inside the body, the test `REPEAT N` runs the body nine times, on `N` equal to 9 down to 1.

The catch is where you put the decrement. Put `N-1->N` before the tally and you count 8 down to 0 and lose a pass; put it after and you get all nine. Count the passes on paper before you run it, because an off-by-one here looks exactly like a bad random stream.

4. With `INT(`: an even roll is one where the roll divided by 2 is a whole number, so `INT(RANDI(1,6)/2)*2` equals the roll exactly when it is even. Comparing those needs a subtraction, which makes the line long.

Simpler with `INT(: S+1-INT((RANDI(1,6)+1)/2)+INT(RANDI(1,6)/2)` is worse still, because it draws twice.

The clean way without `INT(` is to use the fractional part directly. Store the roll first, then test `2*(R/2-INT(R/2))`, which is 0 for even and 1 for odd, so `S+1-2*(R/2-INT(R/2))->S` tallies evens. That needs two lines, one to store the roll and one to tally it, which is the real lesson: without comparison operators, anything that uses a value twice has to store it first.

3.4 Two columns and a family of models

1. Predict `R` of exactly 1: the data is a straight line with no noise at all, and correlation measures how nearly the points sit on a line.

Enter days 1 to 6 against 6, 8, 10, 12, 14, 16 and press `2V`: `MEANX 3.5`, `MEANY 11`, and `R 1`.

Not 0.9999 but 1, to every digit the screen will show. Noiseless correlation is exactly 1, and this is worth seeing once because it calibrates everything else: the real bean data's 0.9726 is what a small amount of noise does, not what a good fit looks like.

2. Fit `EXPR` to the bean heights and it returns a model of the form A e^(Bx). It predicts too low at both ends and too high in the middle, or the other way about, because an exponential curves and the data does not.

The residuals drift in a pattern rather than scattering, which is section 3.4's central point arriving from the other direction: a wrong model shows itself in the *shape* of its errors, not their size. A drift means the model is missing something systematic.

3. Take $y = 2x^3$, so A is 2 and B is 3, and tabulate it at x = 1 to 5: 2, 16, 54, 128, 250.

Enter those five pairs, press MORE three times to the `LNR EXPR PWR P2 P3` page, and press F3, `PWR`. The machine returns `A 2` and `B 3`, recovering exactly the constants you built in.

A power law is a straight line in log-log coordinates, which is how the fit finds it, and that is also why it recovers exact constants from exact data.

4. The duckweed areas are 6, 12, 24, 48, 96. Ask the machine for their natural logarithms one at a time: `1.7917594692291`, `2.4849066497888`, `3.1780538303494`, `3.8712010109087` and `4.5643481914678`.

Notice the differences between consecutive values before you go on: each is `0.6931471805597`, which is ln 2. The logarithms of a doubling sequence are an arithmetic one.

Enter those against weeks 1 to 5 and fit `LIN`. The slope comes back as 0.69314718…, which is section 3.4's `B` to every digit.

They agree because taking logarithms turns the exponential model into a linear one. If y is A e^(Bx) then ln y is ln A plus Bx, a straight line of slope B. So `EXPR` and this hand-built `LIN` are the same fit computed two ways, and that is exactly how `EXPR` works inside.

3.5 What "best fit" actually means

1. Predict 0. The line y = 4 + 2x passes through every one of the points 6, 8, 10, 12, 14, 16 exactly, so every residual is zero and so is their total.

Edit lines 2 to 4 to use the new heights and run: `0`.

That is the only time the score is zero, and it is worth having as the bottom of the scale. Every other line on every other data set scores something positive.

2. Nudging the intercept moves every prediction by the same amount, so the six residuals all shift by the same amount, and the total of their squares depends only on the size of the shift and not its direction. Up and down cost the same.

Nudging the slope moves the predictions by different amounts, more at the far end than the near one, so there is no such symmetry in general. It happens to be symmetric here because the days are evenly spaced about their own mean.

Test it: `1.9–>M` scores `4.91`, exactly as `2.1–>M` did. So the symmetry holds for this data, and would not for days spaced 1, 2, 3, 4, 5, 10.

3. The least-squares line does pass through (3.5, 11): substitute 3.5 into 4 + 2x and you get 11, which is the mean of the heights.

It has to. Setting the derivative of the score with respect to the intercept to zero gives exactly the condition that the residuals sum to zero, and residuals summing to zero says the line's average prediction equals the data's average value. That is the same statement as passing through the two means.

So every least-squares line passes through the centre of gravity of its data, which is a useful thing to check a fit against in one line of arithmetic.

4. Store `11->B` and `0->M` and run: `74`.

That is the score of the best line you could draw *without using x at all*, so the difference between 74 and 4 is what knowing the day buys you. The trend explains 70 of the 74, so the fraction left unexplained is 4 over 74.

Press **CLEAR** and type `1-4/74: = 0.945945945946`. Press **CLEAR** and type `0.9725975251592^2: = 0.9459459459458`.

The same number to eleven digits. That is what `R` squared *is*: the fraction of the variation the model accounts for, and now you have computed it from first principles rather than read it off a screen.

5. The duckweed has five pairs, not six, so lines 2 to 4 become `S+(B+M*1-6)^2+(B+M*2-12)^2->S`, then the same with 3 and 24 then 4 and 48, then a line with just 5 and 96.

Score the straight line with `-27.6->B` and `21.6->M`: `691.2`.

Now score the doubling model by hand: its predictions at weeks 1 to 5 are 6, 12, 24, 48, 96 against data of exactly those values, so the residuals are essentially zero and the score is essentially zero.

The straight line scores 691.2 against essentially nothing. `R` said 0.933 for that line, which sounded respectable, and the score says it is hopeless. When two models are on the table, compare their scores rather than their correlations.

3.6 Forecasting the full pond

1. Predict from the doubling: week 5 is 96, so week 6 is 192 and week 7 is 384. Press **CLEAR** and type `3*128: = 384`.

The pond holds 100. Week 7 would need nearly four ponds, which is the model telling you loudly that it stopped being true two weeks ago.

2. Store 50 as a `Y` target and forecast with `FCX`: about 4.0589.

Press **CLEAR** and spell `LN(50/3)/LN(2):` = `4.0588936890459`, against the full-pond answer of `5.0588936890553`.

Exactly one week earlier, to ten digits. It must be, because the model doubles every week, so whatever the pond's level, half of it happened one week before. The difference of the two logarithms is ln 2 over ln 2, which is 1 exactly.

3. Store 3 as a `Y` target and `FCX` answers 0.

The model is 3 times 2 to the power x, so at x = 0 it gives 3. The fitted `A` was 3.0000000000031, which is the coefficient the fit recovered, and week zero is where that coefficient lives.

Week zero is a week before the first observation, so the model is telling you where the duckweed would have been the week before anybody looked. That is extrapolation backwards, and it is exactly as trustworthy as extrapolation forwards: fine for one step, nonsense for ten.

4. Press **CLEAR** and spell `LN(200/3)/LN(2):` = `6.0588936890418`, so the machine says the pond holds 200 square metres in week six.

What is wrong with it, in a sentence for a report: *the model predicts 200 square metres of cover in a pond of 100 square metres, so the prediction is outside the range in which the model can be true and should not be quoted.*

The arithmetic is correct and the answer is meaningless. Knowing the difference is the whole job.

3.7 Four pictures of one week

1. `BOX` reads the `X` column, sorts it internally to find the quartiles, and draws five numbers. Sorting the column first therefore changes nothing at all.

The plot that would have changed is `XYLN`, which joins the pairs in entry order and is the only one of the four that trusts your ordering. `SCAT` would not change either, and `HIST` would not, for the same reason as `BOX`: they read values, not sequence.

2. SY sorts on the Y column, so the pairs come out in order of visitor count rather than day. XYLN then joins them in that order, which draws a line climbing steadily from 12 to 43 while the days jump about underneath.

The question that answers is "what is the distribution of counts", which BOX and HIST answer better. What it destroys is the time order, which was the only thing XYLN was good for. It is a picture that looks like a trend and contains none.

3. The counts run 12 to 43, so the range is 31 and each of the four bins is 7.75 wide: 12 to 19.75, 19.75 to 27.5, 27.5 to 35.25, 35.25 to 43.

Sorting the eight counts into those: 12, 15, 15, 17, 18, 19 fall in the first, 21 in the second, nothing in the third, 43 in the fourth.

So the bars are 6, 1, 0, 1, which is what section 3.7 step 6 shows.

4. HIST would show it as a bar one taller than it should be, which you would only notice if you knew the right answer.

BOX might not show it at all: a repeated middle value barely moves the quartiles.

SCAT would not show it either, because two identical pairs plot as one dot. That is worth pausing on: a scatter plot silently hides duplicates.

XYLN is the one that shows it, and clearly: a repeated pair draws a line that goes out and comes straight back, leaving a visible spike or a doubled-back segment.

So the answer is XYLN, and the reason is the same property that made it untrustworthy in step 2. A plot that respects entry order is the only one that can show you something about entry order.

Solutions for Chapter 4

9.4

4.1 Limits by table and zoom

1. Store `(1-COS(X))/X` and probe, pressing **CLEAR** before each: `EVAL(.1)` gives `0.04995834722`, `EVAL(.01)` gives `0.00499995834`, and `EVAL(.001)` gives `0.0005`.

The limit is 0, and the values are shrinking in proportion to x, so the quotient behaves like x over 2 near the origin.

Now `(1-COS(X))/X^2`: `EVAL(.1)` gives `0.4995834722`, `EVAL(.01)` gives `0.499995834`, `EVAL(.001)` gives `0.5`.

Dividing by the extra x changed a limit of 0 into a limit of a half. That is the whole idea of the *rate* at which something vanishes: 1 - cos x goes to zero like x squared over 2, so dividing by x leaves something going to zero, and dividing by x squared leaves a half.

Compare with the series: cos x is 1 - $x^2/2$ + …, so 1 - cos x is $x^2/2$ - …, and the two answers fall straight out.

2. Halving four more times takes the step to 0.00390625. The row next to the hole reads `0.999` still, because the cell holds five characters and `0.9999` needs six.

What stops it showing more is the cell width, not the mathematics. The value really is closer to 1; the table simply cannot say so. That is what step 10's `EVAL(` probes are for, and it is why the section moves to them.

3. From the series in step 15, sin 2x over x is 2 minus $(2x)^2$ over 6 times 2… work it out properly: sin 2x is 2x - $(2x)^3/6$, so sin 2x over x is 2 - $8x^2/6$, which is 2 - $4x^2/3$. The limit is 2.

Check: store `SIN(2*X)/X`. `EVAL(.01)` gives `1.9998666693333` and `EVAL(.001)` gives `1.9999986666669`.

At .01 the predicted shortfall is 4(0.0001)/3, which is 0.000133, and 2 - 0.000133 is 1.999867. The machine says 1.9998667. The series was right to seven digits.

The general rule is that sin(kx)/x heads for k, which is worth knowing because it turns up constantly.

4. `EVAL(1E-9)` gave exactly 1. Working down: `EVAL(1E-6)` gave `0.9999999999999`, which is still visibly short.

So somewhere between a millionth and a billionth the difference stops fitting. The function is 1 - $x^2/6$, so the shortfall at 1E-7 is about 1.7E-15, which is just below the fourteenth digit. That is the answer: the crossover is around 1E-7, and the number it measures is the machine's precision, not anything about sine.

5. Store `X/SIN(X)` and probe: `EVAL(.01)` gives `1.000016666861` and `EVAL(-.01)` gives the same, because the function is even for the same reason `SIN(X)/X` was.

Its limit is 1 because it is the reciprocal of a quantity heading for 1, and the reciprocal of a limit is the limit of the reciprocal provided the limit is not zero. That proviso is doing real work here and it is satisfied.

The squeeze of step 14 does not transfer unchanged. Taking reciprocals *reverses* the inequality, so cos x < sin x / x < 1 becomes 1 < x / sin x < 1/cos x. The two-line fix is to say exactly that, and then note that 1/cos x heads for 1 as well, so the sandwich still closes. What goes wrong if you skip that step is that you end up claiming the quantity is squeezed between two things in the wrong order.

6. In `DEG` mode every angle is scaled by π/180 before the sine sees it, so sin(2x)/x picks up that factor once: the limit becomes 2 times π/180, which is π/90, about 0.0349.

Predict it, then check with `EVAL(.001)`, and set the machine back to `RAD` before section 4.2 or nothing there will work.

4.2 A limit that is not there

1. Far from the origin, one over x is small and changing slowly, so the sine's argument barely moves and the curve is nearly flat. At x = 10 the argument is 0.1; going out to x = 11 changes it to about 0.09. The whole stretch from 10 to 20 covers an argument range of only 0.05.

Near nought the same amount of x covers an enormous range of argument. That is the entire asymmetry: the trouble is not that the sine is violent, it is that one over x compresses infinitely much argument into a finite stretch of x.

2. Sine of one over x is 1 when one over x is π/2, 2π + π/2, 4π + π/2 and so on, so x is 2/π, then 2/(5π), then 2/(9π): about 0.6366, 0.1273, 0.0707.

They bunch up like one over the count, so the gaps shrink but never stop. Check one with `EVAL(.6366)`, which should be very close to 1.

3. Predict `X^2` and `–X^2` as the walls, since the sine is still bounded by 1 either way and the multiplier is now x squared.

Put all three in the slots and zoom in. The funnel is narrower and closes faster, because x squared shrinks quicker than x. The limit is still 0 and the squeeze is tighter.

4. Predict: dividing by x rather than multiplying makes the amplitude *grow* as x shrinks, so it should oscillate worse and worse without settling or blowing up cleanly.

Probe it at .1, .05 and .02: the values swing to roughly -5.4, 18.3 and -13.1, growing in size and refusing to settle in sign. It has no limit, and unlike `SIN(1/X)`

it is not even bounded. Both of those are ways of failing to have a limit, and they are different failures.

5. The supported range is one million radians, so `SIN(1/X)` should answer while one over x stays inside it: x down to 1E-6 and no further.

That is what happens. `EVAL(1E-6)` asks for the sine of exactly a million and answers `-0.3499934460541`. `EVAL(9E-7)` asks for about 1.11 million and answers `PRECISION LOST`.

The two need not agree to the last digit, and the reason is worth having. One million is where the firmware stops *guaranteeing* about 1E-7; it is a promise about accuracy, not a wall the arithmetic runs into. A limit quoted as a round number is nearly always a promise rather than a mechanism, and it is worth knowing which kind of number you are reading.

6. Start from the standard window and pick a tolerance, say 0.1. The curve of `X*SIN(1/X)` is inside a band of ±0.1 once |x| is below 0.1, and three presses of + gets the window to -1.25 to 1.25, which is not yet enough; a fourth reaches 0.625 and a seventh reaches about 0.078.

Halve the tolerance to 0.05 and you need one more press, because the bound is |x| and the window halves each time.

So the presses grow like the logarithm of the tolerance: each halving of the tolerance costs one press. That is a prediction you could have made from the walls being straight lines, and it is exactly why this squeeze is easy while `SIN(1/X)`'s is impossible.

4.3 The derivative as a limit

1. Predict: the quotient of `X^2` at step .01 is $(x + .01)^2 - x^2$ over .01, which is $2x + .01$. So it sits exactly 0.01 above `2*X` everywhere, and the gap does *not* grow.

Build both slots and the table confirms it: every row differs by 0.01.

That is a sharper result than the cubic's, where the gap grew, and the reason is that a parabola's second derivative is constant while a cubic's is not. The chord error depends on the curvature, and here the curvature is the same everywhere.

2. At a = 0, f(0) is 0, so the quotient is just f(h)/h, which is $h^2 - 2$. The chords head for -2.

`NDER(0)` answers `-1.9999999999`, which is -2 with a whisker missing. That whisker is worth a thought: `NDER(` uses a central difference with a fixed small step, and the cubic's third derivative is not zero, so a small error survives. At x = 1 and 2 the answers came out exactly 1 and 10, because the error term happens to cancel there.

3. The backward chord `(EVAL(1.5)-EVAL(1.5-.01))/.01` answers `4.7051`. The forward one answered `4.7951`.

The forward chord is above 4.75 and the backward one below, because the curve bends upwards: a chord to the right overshoots the tangent and a chord to the left undershoots.

Average them: press **CLEAR** and type `(4.7951+4.7051)/2: = 4.7501`.

That is out by a ten-thousandth where each chord was out by a twentieth. Averaging a forward and a backward chord *is* the central difference, which is what `NDER(` does, and this is why it is so much better.

4. `NDER(0)` is `-1.9999999999`, `NDER(1)` is `1` and `NDER(2)` is `10`, against $3x^2 - 2$ giving -2, 1 and 10.

Only the first is dusty. The error in a central difference depends on the third derivative and the step, and for a cubic the third derivative is a constant 6, so the error is the same size everywhere. It shows at 0 because the answer there is small enough for a fixed error to be visible in the last digits, and hides at 10 because the same error is far below the fourteenth digit of a bigger number.

Absolute error constant, relative error shrinking with the size of the answer: that is the usual arrangement and it is why small answers look worse.

5. Compute the forward chord at 1.5 with steps 1E-4 through 1E-8. The errors fall as the step shrinks, reach their smallest around 1E-6 to 1E-7, and then grow again as cancellation takes over.

The best is around 1E-6, where the answer was `4.7500045`. That is the sweet spot for a one-sided chord on this machine: small enough that the method error is tiny, large enough that the subtraction still has digits to work with. Every fixed-precision machine has one and its position depends only on how many digits it keeps.

4.4 Extrema by search

1. `X^2*(X^2-4)/4` is even, so its graph is symmetric about the y axis and its two minima must be at plus and minus the same number.

The derivative is x^3 - 2x over… work it out: the function is $(x^4 - 4x^2)/4$, whose derivative is x^3 - 2x, which vanishes at 0 and at plus or minus root 2.

`FMIN(0,3)` gives about 1.414 and `FMIN(-3,0)` gives about -1.414, symmetric as predicted, and both a few digits short of root 2 for the usual reason.

2. There is no turning maximum in (0, 4), so the largest value of the cubic on that interval is at an endpoint. The search reports a value near 4, the right-hand end.

`EVAL(` at the answer gives about 5.33, and `EVAL(-2)` gives `= 5.3333333333333`. They are nearly the same, which is a coincidence of this cubic: $x^3/3$ - 4x takes the value 16/3 both at its hill and again out at x = 4.

The lesson is that a search with bounds reports the largest value it found, which is not the same as a turning point. Always ask whether the answer sits at an end.

3. After two presses of + the window is -2.5 to 2.5, which still contains the maximum at -2 but samples it four times as finely.

The answer comes back closer to -2 than the whole-window search did, because the bracket the search starts from is narrower. You could have predicted that: the search's accuracy depends on how many samples span the flat region near the top, and zooming in buys you more of them.

4. `EVAL(-2)` gives `= 5.3333333333333` and `EVAL(-1.99)` gives `= 5.3332333333333`, a change of one ten-thousandth for a hundredth of movement.

Either side of x = 0 the same movement costs much more: `EVAL(0)` is 0 and `EVAL(.01)` is about -0.04, four hundred times the change.

Flat means small change for given movement, which is precisely why locating a maximum precisely is hard and locating a steep crossing is easy. Root finders are accurate; extremum finders are not; and it is the same fact seen twice.

4.5 The definite integral as an average

1. The model is `17−3∗COS(PI∗X/12)`, on the pattern of step 1 with 17 for the centre and 3 for the swing.

Predict 17: the cosine averages zero over a whole period, so the average is the centre line.

Check: press **CLEAR** and spell `FNINT(0,24)/24: = 17.00000073816`. The trailing digits are the machine's `PI`, as before.

2. The model reaches its average where the cosine is zero, which is at `PI∗X/12` equal to $\pi/2$ and $3\pi/2$, so $x = 6$ and $x = 18$.

Check both with `EVAL(`. Six in the morning and six in the evening, which is what section 4.5 step 6 found for the harbour and is true for any model of this shape.

3. `FNINT(−1,1)` answers `= −5.3333333333333` and `FNINT(1,3)` answers `= −5.3333333333333`.

They add to -10.666666666667, which is step 9's answer for the whole dip, so the splitting rule holds. That the two halves are equal is a symmetry of this particular parabola about $x = 1$, not a general fact.

4. On paper, the integral of $x^2 - 2x - 3$ from -5 to 5 is $[x^3/3 - x^2 - 3x]$ evaluated between, which gives 250/3 - 30 minus (-250/3 - 30 + …). Work it through and you get 500/3 - 30, which is about 136.67.

Press **+** once on the plot and use **F5** on the halved window: the machine reports the same figure. The window is the interval, so halving the window halved the interval, and the answer is the -5 to 5 integral rather than the -10 to 10 one.

5. The average over -1 to 3 is the integral divided by the width: press **CLEAR** and spell `FNINT(−1,3)/4: = −2.6666666666668`, which is -8/3.

The curve takes that value where $x^2 - 2x - 3$ equals -8/3, which solves to $x = 1$ plus or minus root(4 - 8/3), so about 1 ± 1.155: at -0.155 and 2.155.

Both sit inside (-1, 3), as the mean value theorem for integrals promises. That theorem is the same statement section 4.5 step 6 made about the harbour, and it holds for every continuous function on every interval.

4.6 Riemann sums by program

1. Predict from step 5's relation. At eight slices the width is 0.25, so the gap between the left and right sums is (f(2) - f(0)) times 0.25, which is 4 times 0.25, which is 1. The two sums straddle 4.6667 and their average is the trapezoid, so left is about 4.1875 and right about 5.1875.

Edit `P1` to count 0 to 7, sample `A/4` and display `S/4`. The run screen answers `4.1875`. The prediction was exact.

2. Run left and right at eight slices: `4.1875` and `5.1875`. Their difference is exactly 1, which is (5 - 1) times the new slice width of 0.25.

The relation holds at every slice count, which makes it a genuinely useful check on a program you have just typed: if the two sums do not differ by that amount, you have mistyped a bound.

3. Store `X^2+1` and rerun `P4` untouched and it still measures the same integral, because `EVAL(` reads whatever is stored. Store `X^3+1` instead and it measures that one.

What you have to be careful about is the *interval*. The program's sample points `(2*A-1)/8` are hard-wired to 0 to 2, so storing a new equation changes the function and not the range. Check any answer against `FNINT(` with bounds 0 and 2, not against whatever bounds you had in mind.

4. From the eight-slice numbers: the trapezoid is (4.1875 + 5.1875)/2, which is 4.6875, and the midpoint was 4.65625. Simpson is (2 times 4.65625 + 4.6875)/3, which is 4.6666666…, exactly 14/3 again.

On `X^3+1` from 0 to 2 the exact integral is 6, and Simpson returns 6 as well, because Simpson is exact on cubics too. That surprises people: fitting a parabola through three points integrates a cubic exactly, because the error terms cancel by symmetry.

On `1/X` from 1 to 2 it stops being exact. Simpson gives about 0.694 against the true 0.6931, so three decimals rather than fourteen. Exactness stops as soon as the function is not a polynomial of degree three or less.

5. The midpoint error quarters per doubling, so from 1/24 at four slices you need the error below about 1E-6, which is a factor of 40000, which is seven or eight doublings: around 512 to 1024 slices.

The eight-line slot could not run it when this was written. `FOR` bounds were single digits and even the `WHILE` countdown would need the loop to run a thousand times, which on this machine is minutes of work for a number `FNINT(` gives you in a second. That is the honest limit of doing numerical integration by hand here, and it is why `FNINT(` exists.

4.7 Areas between curves

1. Predict -2.25. The difference has been turned upside down, so every height changes sign and so does the integral.

Retype `Y3` as `X/2+1-(2-X^2/2)` and integrate from -2 to 1: `= -2.25`.

What changed is the sign. What did not is the size, the crossings, or the region itself: 2.25 square units of area sit between those curves whichever way you subtract, and only the bookkeeping moved.

2. With `Y2` as `X/2` the crossings solve $2 - x^2/2 = x/2$, which is $x^2 + x - 4 = 0$, so $x = (-1 \pm \sqrt{17})/2$: about 1.5616 and -2.5616.

The residual lines no longer read `R=0`. They report small nonzero values, because the search stops when the residual passes the tolerance rather than when it hits zero, and an irrational root can never be hit exactly. A residual of 1E-13 means the search got as close as fourteen digits allow.

3. The line is above the arch beyond $x = 1$, so from 1 to 4 the difference function is negative. `FNINT(-2,4)` therefore mixes a positive contribution from -2 to 1 with a negative one from 1 to 4, and reports the net.

For the total enclosed area on both sides, integrate the two pieces separately and add their sizes: `FNINT(-2,1)` gives 2.25 and `FNINT(1,4)` gives a negative number whose size you add rather than subtract. That is section 4.5's sign convention arriving exactly where nobody wants it, for the second time.

4. Pick the crossings first: to cross at -1 and 3, the difference must vanish there, so it is a multiple of $(x + 1)(x - 3)$, which is $x^2 - 2x - 3$.

Take the difference as $-(x^2 - 2x - 3)$, so the region is positive between the crossings, and split it into two curves however you like: the arch `3+2*X-X^2` and the line `0`, or more interestingly `4+2*X-X^2` against the constant `1`.

Check with **2nd** **F1** and you should find -1 and 3 to the usual dust. Designing backwards from the answer is how every exercise in this book was built, and it is

a good habit for building your own.

9.5 Solutions for Chapter 5

5.1 Zeros of functions two ways

1. Multiply, say, $x^2 - 3$ by $x^2 - 4x + 1$ to get $x^4 - 4x^3 - 2x^2 + 12x - 3$, whose roots are plus and minus root 3 and 2 plus or minus root 3.

Enter the coefficients 1, -4, -2, 12, -3 and press `SOLV`. The browser returns all four, each carrying a digit or two of dust in the last places.

You get eleven or twelve good digits out of fourteen, which is what an iterative root hunt on a quartic costs. Designing the roots first is the only way to know that.

2. `POLY` on $x^2 + 2x + 3$ returns a conjugate pair: `RE −1` with `IM 1` and `RE −1` with `IM −1`, since the roots are $-1 \pm i$.

The solver stops at a notice instead, because it hunts for a sign change along the real line and there is none: the parabola never touches the axis.

Each tool is telling you the truth about the question it was asked. `POLY` solves the polynomial; the solver looks for a real crossing. The failure is informative, not a defect, and the pair of answers together says “no real root, two complex ones” more clearly than either alone.

3. With bounds -5 and 0 and a guess of 3, the guess is outside the bounds. The workspace tries the guess first, then scans within the fence, and reports the negative root it finds: minus root 2, about -1.4142134785654.

To fence in the other negative root, 1 minus root 3 at about -0.732, set bounds -1 and 0. The two negative roots need separate fences because the scan stops at the first sign change it meets.

4. `RES` is the value of the equation at the reported root. A perfect root gives exactly zero; anything else reports how far from zero the expression still is.

Handing it the exact root produces zero because the expression really does vanish there and the arithmetic happens to confirm it in fourteen digits. That is a check on the *root*, not on the hunt: it says the number you supplied satisfies the equation, which is what you wanted to know.

5.2 Newton's method

1. Predict: 3 is further from the root than 2, so it should take at least as many steps.

Starting at 3, three steps give `2.0951360369349` and four give `2.0945516738243`. Starting at 2, four steps gave `2.0945514815424`, which is the settled answer.

So from 3 it is one step behind: it needs five to settle where 2 needed four. That is what quadratic convergence looks like from further out. The first step does the coarse work and the doubling only takes over once you are close.

2. At 0.8 the cubic's slope is 3(0.64) - 2, which is -0.08: almost flat. A nearly flat tangent runs an enormous distance before it meets the axis.

Predict a long throw, then run one step: `-75.3`.

Seventy-five units away from a root at 2.09, from a start two thirds of the way there. A second step brings it back to `-50.205609036075`, still nowhere. The size of that first throw is exactly f(0.8) divided by 0.08, and small denominators are how Newton's method goes wrong.

3. Store `X^2-2` and run from 1. Three steps give `1.414215686276` and four give `1.4142135623747`, which is root 2 to twelve digits.

Line 4 becomes R - (R^2 - 2)/(2R), which simplifies to (R + 2/R)/2: the average of your guess and 2 divided by your guess. That is the Babylonian method for square roots, known for three and a half thousand years, and you have just derived it from Newton's method in one line of algebra.

4. Store `COS(X)-X` and run from 1. Three steps give `0.7390851333854` and four give `0.7390851332152`.

Section 5.1's solver gave `0.7390856742858` for the same root. The two disagree in the seventh digit, and Newton is the one that is right: the true root is 0.73908513321516.

The solver stopped because its residual passed the `1E-6` tolerance, which is what it was asked to do. Newton kept doubling its digits and ran out of machine before it ran out of accuracy. Neither is broken; they were asked different questions.

5. Four Newton steps cost eight evaluations of the stored equation, two per step for `EVAL(` and `NDER(`. In fact `NDER(` is itself a central difference and evaluates

twice, so the true count is nearer twelve.

Bisection to fourteen digits needs about 47 halvings and one evaluation each. So Newton wins by a factor of four on this problem, and would win by far more on a harder one, provided it converges at all. That proviso is the whole trade: bisection is slow and cannot fail, Newton is fast and can end up at -75.3.

5.3 Conic sections by parametric pair

1. A basin 8 across and 8 deep is a circle of radius 4, so the pair is `4*COS(X)` and `4*SIN(X)`.

In the square window it draws a circle rather than an ellipse, which is the visual check. Trace a point and test it against $x^2 + y^2 = 16$: the sum should come back as 16 to all fourteen digits, exactly as section 5.3's ellipse did.

2. Swapping to `2.5*COS(X)` and `5*SIN(X)` gives the same ellipse standing on end: half-width 2.5 and half-height 5, so twice as tall as wide.

Predicting that is easier than it looks if you remember which slot is which: slot 1 is x, slot 2 is y, so the number in slot 1 controls the width.

3. Two presses of + halve the bounds twice, so the window runs -2.5 to 2.5. The pond is 5 wide and no longer fits, so you see the middle of it only.

But the window is also the sweep, so t now runs from -2.5 to 2.5, which is less than one full revolution. So you get *part* of the rim, drawn at higher resolution, rather than the whole rim magnified. Both things happened at once and only one of them was what you wanted.

4. A circle of radius 3 centred at (4, 1) is `4+3*COS(X)` and `1+3*SIN(X)`: the constant moves the centre and the coefficient sets the radius.

Trace any point and check that $(x - 4)^2 + (y - 1)^2$ comes back as 9.

5.4 Polar curves

1. `4*SIN(3X)` draws three petals, not six. Predict three: with an odd multiplier the negative half of each cycle retraces petals that the positive half already drew, so half of them coincide.

Doubling gave four petals from two cycles because even multipliers put the negative lobes in the gaps. That is the rule: `SIN(nX)` gives n petals for odd n and 2n for even n, which is worth checking with `4*SIN(4X)`.

2. `2.5*(1-COS(X))` faces the other way from section 5.4's cardioid: its cusp is at angle 0 rather than at pi, because the radius now vanishes when the cosine is 1.

Press **CLEAR** and ask `EVAL(0): = 0`. Exactly zero this time, with no dust at all, because 1 - cos 0 is 1 - 1 and the machine's cosine of exactly zero is exactly one. The dust in section 5.4 came from `PI` not being pi; here the awkward angle never appears.

3. The half-the-angle rule holds at every stop, because the slot text `X/2` says so directly: the polar readout's first line is the radius and its second is the angle, and the radius is computed as half the angle.

What you are checking is not the mathematics but that you have read the screen correctly, which is worth doing once because the labels both say `X=` and `Y=` whatever the coordinate mode.

4. Between pi/2 and pi, the angle 2X runs from pi to 2pi, where sine is negative. So the radius is negative, and a negative radius plots in the *opposite* direction from the angle.

The result is that the petal drawn during that stretch appears in the third quadrant rather than the second. Working out which petal gets drawn when is the whole trick of reading a rose, and the answer is that consecutive quarter-turns of the parameter draw petals on opposite diagonals.

5.5 Parametric motion

1. On paper: the pebble lands when $12t - 5t^2$ is zero again, at $t = 12/5 = 2.4$ seconds, and the range is 2 times 2.4, which is 4.8 metres. The apex is at half the flight, $t = 1.2$, where the height is $12(1.2) - 5(1.44) = 7.2$ metres.

Store `2*X` and `12*X-5*X^2`. `FMAX(0,3)` gives a whisker under 1.2, and `EVAL(1.2)` answers `= 7.2`. For the landing, put `12*X-5*X^2` in the solver with a guess of 2 and bounds 1 and 4: the root comes back at essentially 2.4.

2. `FMAX(` stops a whisker short because the curve is flat at its apex, so a wide range of t values all look equally like the top. That is section 4.4's story exactly, and it is why the section reaches for `EVAL(.9)` at the exact value rather than at the search's answer.

3. With `2*X` and `2+9*X-5*X^2`, the pebble starts 2 metres up and lands when $5t^2 - 9t - 2$ is zero, at $t = 2$ exactly. The range is 4 metres.

Trace says the landing is somewhere between the samples either side of t = 2. For the solver, a guess of 2 with bounds 1 and 3 works; a guess of 0 would find the *negative* root at -0.2, which is the pre-launch fiction again.

4. The root hunt scans from the window's left edge, so it finds the launch at t = 0 because the standard window starts at -10 and the first sign change going right is at the origin.

Zoom or shift so the window starts after t = 0. Two presses of + leave the window at -2.5 to 2.5, which still contains the launch. What works is narrowing until the left edge is past 0 while the landing at 1.8 is still inside, which the zoom keys alone cannot quite do, since they always centre on the origin. That is a real limitation and the solver is the answer.

5.6 Functions defined by integrals

1. Predict from the first two probes. `FNINT(0,1)` on `3*X^2` gives 1 and `FNINT(0,2)` gives 8.

1 and 8 are 1 cubed and 2 cubed, so the accumulator is x^3. Check at 3 and 4 and you get 27 and 64.

That is the fundamental theorem again: the accumulator of $3x^2$ is x^3 because $3x^2$ is the derivative of x^3.

2. Predict equality: the area from 1 to 8 is the area from 1 to 2 plus the area from 2 to 8, so `FNINT(2,8)` should be `FNINT(1,8)` minus `FNINT(1,2)`.

Compute all three: `FNINT(1,8)` is `2.0794461816072`, `FNINT(1,2)` is `0.6931471824209`, and `FNINT(2,8)` is `1.3862945205893`.

Now subtract the first two: press CLEAR and type `2.0794461816072-0.6931471824209: = 1.3862989991863`.

That is *not* `1.3862945205893`. They differ by about four and a half millionths.

The mathematics is exact and the arithmetic is not. `FNINT(` fits its panels to whatever interval it is given, so the 1-to-8 integral used panels seven times wider than the 1-to-2 one, and the three answers were computed at three different resolutions. Comparing coarse and fine estimates catches an estimate that is not converging at all; it does not make two converged estimates on different intervals add up exactly. Additivity holds for integrals and only approximately for their estimates.

3. Any pair in the ratio 2 works: 5 to 10, 7 to 14, 100 to 200. Each gives about 0.693, because the logarithm turns ratios into differences and every doubling is the same difference.

4. `8*FNINT(0,.5)` on `1/(1+X^2)` answers `= 3.7091808726128`.

That is 8 times the arctangent of a half, and it is not pi because the arctangent of a half is not pi/8. Pi/8 is the angle whose tangent is 0.4142, not 0.5.

The step 6 trick worked because the arctangent of *1* is exactly pi/4. Half the interval does not give half the angle: the arctangent is not linear, and this is a good way to feel that.

5. The accumulator of 1/t from 2 is ln x minus ln 2, so it is the same function shifted down by 0.6931.

So `FNINT(2,6)` should be ln 3, since 6 over 2 is 3. Check it: `FNINT(2,6)` answers `= 1.0986123199908` and `LN(3)` answers `= 1.0986122886693`, agreeing to seven digits.

You can predict it from step 5's answer without computing anything: 2 to 6 is a tripling, and every tripling accumulates ln 3.

5.7 Indeterminate forms by table

1. Predict from the cosine series. cos x is $1 - x^2/2 + x^4/24$, so 1 - cos x is $x^2/2 - x^4/24$, and dividing by x^2 leaves $1/2 - x^2/24$.

The limit is a half. Probe it: `EVAL(.001)` on `(1-COS(X))/X^2` answers `= 0.5`.

Halved steps from both sides give the same values, because the function is even.

2. The leading coefficients are 2 and 5, so the limit is 2/5, which is 0.4.

Probe with growing arguments: at 100 the value is `0.3939921201576` and at 1000 it is `0.39939992012002`.

At 100 the second decimal is already right and the third is not; at 1000 the third is right. Each tenfold increase buys about one more decimal, because the neglected term falls like one over x. Two cells in a row agreeing to three decimals needs steps out beyond 1000.

3. Build it backwards. You want a quotient heading for 7, so take 7x over x, and disguise it: `(EXP(7*X)-1)/X` heads for 7, by exactly the argument section 5.7 used for the 2.

Probe at .01 and .001 and watch it approach 7 from above.

4. Press **CLEAR** and spell `EXP(LN(1+.06/12)*12): = 1.0616778118669`, against `1.06`.

The question you have answered is what six per cent a year compounded *monthly* actually pays: 6.168 per cent, not 6. Monthly compounding is bigger, and it is bigger for the same reason the table in step 5 climbs: you earn interest on interest twelve times instead of once.

The limit of that process, compounding continuously, is `EXP(.06)`, which is 1.0618365465453. So all the extra compounding in the world buys less than two hundredths of a per cent beyond monthly.

5. The table converges like h, so ten correct digits needs h around 1E-10.

The arithmetic would not survive it. At h = 1E-10, computing 1 + h loses ten of the fourteen digits immediately, so `LN(1+h)` is working with four significant figures and the answer is worthless. This is chapter 4's cancellation cliff in a new costume, and it is why nobody computes e this way.

5.8 Improper integrals

1. Predict a quartering: each octave out multiplies x by 2, and $1/x^3$ falls by 8, while the interval width doubles, so the slab falls by 4.

Check: `FNINT(1,2)` gives `0.37500001953353`, `FNINT(2,4)` gives `0.09375000488339`, and `FNINT(4,8)` gives `0.023437501220829`.

Each is a quarter of the one before. The running totals are 0.375, 0.469, 0.492, climbing the geometric staircase whose top is 0.5, which is the paper answer.

2. `FNINT(1,1000)` on `1/X^3` answers `= 5.2083129209853`.

The truth is just under 0.5, so the single long probe is out by a factor of ten. Trust the octave walk. The reason is exactly section 5.8's: sixty-four panels across a thousand units puts each panel fifteen units wide, and nearly all of this integral lives in the first unit.

A wrong answer that is ten times too big rather than ten times too small is worth noticing, incidentally. Starving a spike overestimates, because the one panel that lands near it counts its enormous height across the whole panel width.

3. `1/X^(2/3)` cannot be typed directly, because `^` takes whole exponents. Route it through the identity: store `EXP(LN(X)*(-2/3))`.

The plot will fail, because the standard window includes negative x and there is no logarithm of a negative number. The equation is stored all the same, and `FNINT(` works. Be patient with it: a logarithm and an exponential at every panel is slow, and there are at least 96 panels between the coarse estimate and the one it is checked against.

On paper the integral from a to 1 is 3(1 - a ^ (1/3)), so it climbs to 3.

`FNINT(.25,1)` gives `1.1101184746881`, against the paper value `1.1101184251563` from `3-3*EXP(LN(.25)/3)`: seven digits. `FNINT(.0625,1)` gives `1.8094670776981` against `1.8094492110232`: five digits. `FNINT(.01,1)` gives `2.3589135473003` against about 2.3537: three.

It converges to 3, and the panels lose a digit or two every time you push the lower bound closer to the singularity. Same disease as `1/SQRT(X)`, milder because the spike is milder.

4. For `1/SQRT(X)`, substituting $x = u^2$ gives $dx = 2u\,du$ and the integrand $1/u$, so the whole thing becomes 2 du: a constant, integrating to 2(1 - root a), which is the paper answer.

For `1/X^(2/3)`, substitute $x = u^3$: $dx = 3u^2\,du$ and the integrand is $1/u^2$, so it becomes 3 du, integrating to 3(1 - a ^ (1/3)).

In both cases the substitution that works is the one that clears the fractional power, and what it buys is an integrand with no spike at all. That is the general trick and it is worth more than any number in this section.

5. Put the singularity at the right-hand end: `1/SQRT(1-X)` on 0 to 1.

Predict the same failure mode, and you get it: `FNINT(0,.75)` is fine, `FNINT(0,.99)` starts to drift, and pushing the upper bound to 1 gives nonsense. `FNINT(` has no idea which end its trouble is at, because it treats the interval symmetrically.

5.9 Polynomial approximation

1. The plain line `X` differs from the sine by about $x^3/6$, so it stays within 0.01 while x^3 is under 0.06, which is x under about 0.39.

Check with the home screen rather than the table, which is too coarse: `SIN(.1)−.1` answers `= −0.00016658335317`, comfortably inside, and `SIN(.5)−.5` answers `= −0.02057446139581`, outside.

So the crossover is between 0.1 and 0.5, and the cube-root estimate of 0.39 is about right. Read that as: the line is good to a hundredth for angles up to about 22 degrees, which is a genuinely useful rule of thumb in physics.

2. Build `1−X^2/2` in `Y2` and `1−X^2/2+X^4/24` in `Y3` against `COS(X)` in `Y1`, in the trigonometric window.

The quadratic parts company around x = 1, the quartic holds to about 2, and by x = 3 both are hopeless. The pattern is the same as sine's and for the same reason: each new term buys roughly one more unit of agreement, and the interval widens without bound.

3. Adding `−X^7/5040` gives the degree-7 impersonator. At 2 radians it is right to about four decimals where the quintic was out by 0.024; at 3 radians it is out by about 0.05 where the quintic was out by 0.4.

Check with `EVAL(` against `SIN(`. Each new term roughly divides the error at a fixed x by the square of that x over the next factorial, which is why the improvement is dramatic near the origin and slow far from it.

4. At x = -0.9 the series still converges, because -0.9 is inside the interval, but slowly: the terms fall like 0.9 to the power n, so you lose only about one twentieth per term.

`EVAL(−.9)` on `1/(1+X)` answers `= 10`, and `1/.1` answers `= 10` too.

For two decimals you need 0.9^n below about 0.005, which is n around 50. So fifty terms at x = -0.9 against three or four at x = 0.25. Convergence inside the interval is not the same as *useful* convergence.

5. `1/(1+X^2)` has no trouble anywhere on the real line, so the question is where else a function can misbehave, and the answer is in the complex plane: $1 + x^2$ vanishes at x = i and x = -i.

Those are distance 1 from the origin, so the interval of convergence is again -1 to 1.

Test it by table: the impersonators `1−X^2+X^4` and `1−X^2+X^4−X^6+X^8` agree with the function inside 1 and diverge outside it, with the longer one worse, exactly as

the geometric series did. A real function with no real trouble at all, fenced in by a pair of complex numbers, is one of the better surprises in the subject.

Solutions for Chapter 6

9.6

6.1 One system, two tools

1. Predict: the two lines x + 2y = 4 and x + 2y = 5 are parallel and distinct, so nothing satisfies both.

`RREF` reduces the tableau to `1`, `2`, `0` on the top row and `0`, `0`, `1` underneath. That bottom row reads 0x + 0y = 1, which is the impossible equation, and it is `RREF`'s way of saying so.

The simultaneous editor answers `NO SOLUTION` for the same six values. Two tools, one verdict, and the tableau shows you *where* the contradiction lives rather than only that there is one.

2. Any pair through (3, 1) works: x + y = 4 and x - y = 2, say. Both tools return `X 3` and `Y 1`, and the tableau reduces to the identity with 3 and 1 in the last column.

Designing the answer first is the habit this chapter keeps asking for, and it is the only way to tell a right answer from a plausible one.

3. Press 2nd 7, type 2, 4, 1, 2, and press F1, `DET: 0`.

A zero determinant rules out `UNIQUE SOLUTION`. The rows are proportional, so depending on the right-hand sides you get either `NO SOLUTION` or `UNDERDETERMINED`, and nothing else is possible.

4. Take 1, 1, 1, 1.001. Press `DET: = 0.001`.

Nonzero, so the system has a unique solution and `DET` is content. Now solve it with right-hand sides 2 and 2.001, which gives x = y = 1, and then nudge the second to 2.002: the answer jumps to x = 0, y = 2.

A thousandth of a nudge moved the answer by a whole unit, and `DET` gave no warning at all. That is the lesson: a determinant tells you whether a matrix is singular, and says almost nothing about whether it is *nearly* singular in a way that matters. `COND` in section 6.3 is the number that does.

6.2 Row operations as algebra you can watch

1. After the swap (F2, SWP) the tableau is 3, 4, 11 on top and 1, 2, 5 beneath. The first move is now to subtract a third of the top row from the bottom, so the scale is -1/3 rather than -3, and the numbers along the way are messier.

The finished tableau is identical: 1, 0, 1, 0, 1, 2. It has to be, because row operations do not move the crossing and the crossing is what the finished tableau names.

Predicting that before you start is the point. The route changes and the destination does not.

2. Rescaling row 1 by 10 makes it 10, 20, 50, which is the same line written ten times over. RREF reduces it straight back, and the answer is unchanged.

It was never in doubt because rescaling a row is one of the three licensed moves, and the licence is precisely that it does not change the solution set.

3. Elimination cannot start with a zero in the pivot position, so the single move needed is the swap (F2, SWP), putting 1, 1, 4 on top.

Then subtract nothing (the new second row already has 0 in the first column), rescale row 2 by a half to get 0, 1, 3, and clear the 1 above it. The finished tableau reads 1, 0, 1, 0, 1, 3, so x = 1 and y = 3.

4. To undo the last move, add two copies of row 2 back to row 1: store 2 in B rather than –2 and press RADD again.

The tableau returns to 1, 2, 5, 0, 1, 2. Every row operation has an inverse of the same kind, which is exactly why the solution set cannot move: an operation that lost information could not be undone.

6.3 Norms and the condition number

1. Predict 3. The identity stretches nothing, so its Frobenius norm is root 3 and its inverse is itself, and root 3 times root 3 is 3.

Press COND on the identity: = 3.0000000000001.

No 3 by 3 matrix can do better, because COND is a product of a matrix's stretch and its inverse's, and for a 3 by 3 the Frobenius norm of a matrix and of its inverse are each at least root 3. A condition number of 3 means "as well behaved as this measure allows".

2. Nudging the third right-hand side to 3.002 instead of the first throws the answers a different way: x stays near 1 while y and z take the strain.

Which unknowns jump depends on which combination of rows the nudge disturbs, and with near-repeated rows the answer is always that the *differences* between unknowns absorb it. The size of the jump is the same order as before, because COND is a property of the matrix and not of the right-hand side.

3. Retype the near-repeats as 1.01 and press COND: = 952.70410946948.

Ten times softer than the 1.001 version's 9490, as you would expect: making the rows ten times less alike makes the amplification ten times smaller.

Step 8's jump softens by the same factor. The relationship between how nearly dependent the rows are and how badly the answer moves is linear, and this is the cheapest way to see it.

4. Section 2.2's near-parallel pair was x + 2y = 8 with 1.01x + 2y = 8.05. As a matrix that is 1, 2, 1.01, 2, and its COND comes back in the hundreds.

The well-behaved pair 1, 2, 3, -1 gives a COND of a few.

So yes, the number predicts what you saw. COND in the hundreds warned of the two-hundred-per-cent swing, and COND of a few promised the one and a half per cent. Running a system through COND before trusting it costs one keystroke and would have saved section 2.2 a nasty surprise.

6.4 Building a frame of your own

1. Press 2nd 8, type 3, 1, 2 into A, press ALPHA, type 1, 1, 1 into B, press ALPHA, and press F3, DOT: 6.

MAG of B is 1.7320508075689, root 3, so its length squared is 3, and the shadow is 6 over 3, which is 2 copies.

Build 2 copies with SCL and subtract with SUB, or just do the arithmetic: (3, 1, 2) minus 2(1, 1, 1) is (1, -1, 0).

Confirm: type (1, -1, 0) into A against (1, 1, 1) in B and press DOT: 0, exactly.

2. NRM divides each vector by its own length, so all three come back with MAG of 1.

An orthonormal frame is what you want whenever you need to express something in coordinates that do not distort it: rotations, projections, changes of basis. The

dot product with each frame vector gives the component directly, with no division, because the lengths are already 1.

3. Starting with (0, 1, 1) gives a *different* frame. The first vector is kept as it is, so it fixes the whole construction, and choosing a different first vector points the frame somewhere else.

It should be different. Gram-Schmidt does not find the frame; it finds *a* frame, built around whichever direction you hand it first. Both frames span the same space and neither is more correct.

4. (2, 1, 1) is the sum of (1, 1, 0) and (1, 0, 1), so the three are not independent.

The first two straighten normally. At the third subtraction, removing both shadows removes the whole vector: what is left is the zero vector, to within the machine's dust.

Predicting that is the point. Gram-Schmidt on dependent vectors produces a zero, and a zero vector cannot be normalised, so `NRM` will fail or return nonsense. That failure is a *test* for independence, and it is how the process is usually used in practice.

5. Both checks come out exactly zero when every number in the construction terminates in decimal.

Take (1, 0, 0), (1, 1, 0) and (1, 1, 1). The shadows are all 1 over 1, so every subtraction is exact, and the frame comes out as the three axes with every dot product exactly 0.

What was special about section 6.4's numbers was the two thirds: a repeating decimal typed as a rounded one. Choose data whose arithmetic terminates and the dust never appears.

6.5 Eigenvalues and eigenvectors

1. By symmetry, 3, 1, 1, 3 keeps the directions (1, 1) and (1, -1): the first is stretched by 3 + 1 = 4 and the second by 3 - 1 = 2.

Press `EVAL`: the cells read `4` and `2`, as predicted. `EVEC` returns the two directions normalised, each component plus or minus `0.70710678118655`, which is 1 over root 2.

Any matrix of the form a, b, b, a keeps those two directions, whatever a and b are, which is worth knowing.

2. For 5, 2, 2, 2 the eigenvalues were 6 and 1. Their sum is 7, which is the diagonal sum 5 + 2. Their product is 6, and DET on the same matrix answers = 6.

Both facts hold for every square matrix: the eigenvalues sum to the trace and multiply to the determinant. On the 3 by 3 of step 4, the eigenvalues 2, 3, 6 sum to 11, which is 2 + 3 + 6 down the diagonal, and multiply to 36, which is what DET gives.

3. 0, -3, 3, 0 is a quarter-turn with a stretch of 3, so it keeps no real direction at all and its eigenvalues must be a conjugate pair.

Press EVAL and both real parts read 0. Press **MORE** for the final soft-key page and the IM lines read 3 and -3.

So the eigenvalues are plus and minus 3i, pure imaginary, which is exactly what a pure rotation looks like: no real part means no stretching along any real direction, and the size 3 is the stretch.

4. Take the first eigenvector column from step 3 of the section, which is (0.894, 0.447), and multiply the matrix 5, 2, 2, 2 by it.

By hand: 5(0.894) + 2(0.447) is 5.366, and 2(0.894) + 2(0.447) is 2.683. Those are 6 times 0.894 and 6 times 0.447.

The machine agrees to the dust the normalisation introduced. Multiplying is exact arithmetic where the eigenvalue search was iterative, so this is the right way round to check.

6.6 LU as elimination's ledger

1. Press DET on 3, 1, 0 / 6, 4, 1 / 0, 2, 5: 24.

Now LU. Stepping through gives 3, 1, 0, then 2, 2, 1, then 0, 1, 4.

Read it in layers. U is 3, 1, 0 and 0, 2, 1 and 0, 0, 4. The multipliers are 2, 0 and 1.

No swap happened, and you can tell before you look: the top-left entry is 3, which is nonzero, so elimination could start where it stood.

Check the shortcut: the diagonal of U is 3, 2, 4, whose product is 24, matching DET exactly with no sign flip.

2. From the ledger, row 3 of the original is 0 times row 1 of U, plus 1 times row 2 of U, plus row 3 of U.

That is 0(3, 1, 0) + 1(0, 2, 1) + (0, 0, 4), which is (0, 2, 5). The original third row, rebuilt from the multipliers without looking at `A`.

3. Any matrix with a zero in the top-left needs a swap: 0, 1, 2 / 1, 0, 3 / 2, 1, 0 will do.

Predict the sign relation: one swap flips the determinant's sign, so the product of the `LU` diagonal will be minus `DET`. Two swaps would put it back.

Press both keys and check. Getting the prediction right requires counting the swaps, which the ledger does not label, so you have to spot them by noticing which original row ended up on top.

4. With `L` holding multipliers 2, 0, 1 and `U` as above, solve Ly = (4, 14, 25) by forward substitution: $y_1 = 4$, then $y_2 = 14 - 2(4) = 6$, then $y_3 = 25 - 0(4) - 1(6) = 19$.

Now Ux = y by back substitution: $4x_3 = 19$ so $x_3 = 4.75$, then $2x_2 + 1(4.75) = 6$ so $x_2 = 0.625$, then $3x_1 + 1(0.625) = 4$ so $x_1 = 1.125$.

Check with `SOLVE`, putting the right-hand sides down `B`'s first column: the same three numbers come back.

Two short passes and no elimination at all. That is what the ledger buys, and doing it once by hand is worth more than any amount of description: the second right-hand side would cost you the same two passes and no more.

9.7 Solutions for Chapter 7

7.1 Slope thinking

1. A quantity falling at 15 per cent of itself per minute halves after ln 2 over 0.15 minutes. Press **CLEAR** and spell `LN(2)/.15: = 4.6209812037415`, so about four minutes thirty-seven seconds.

The dose is at `XMIN`, which is -10, so the half-way point sits at x = -10 + 4.62, which is about -5.38. Trace there and the readout should show about 4.5, half of 9.

The plot will not put it exactly at 4.5, because Euler undershoots. That gap is section 7.2's subject.

2. With `-.3*Y` the tank empties twice as fast, so it reaches any level in half the time. Press **CLEAR** and spell `LN(2)/.3: = 2.3104906018707`, exactly half of the

previous answer.

The old curve's value at x = 0 was about 1.97. The new one reaches that value at half the elapsed time, so at x = -5 rather than x = 0.

3. `.15*Y` has a positive slope wherever `Y` is positive, so the solution grows rather than decays, and it grows faster the bigger it gets.

Seeded at 9 it leaves through the *top* of the window almost at once. The initial condition makes that inevitable: 9 is positive, the rule says positive quantities increase, and nothing in the equation ever turns that round.

4. The slope depends only on `Y`, so every point at the same height has the same slope. In the diagram, that means the slope marks are identical along each horizontal row.

Two solutions started at different *times* are therefore the same curve slid sideways. That is what an autonomous equation means, and it is why section 7.4's program can ignore x entirely.

5. Twice the volume with the same flow loses 7.5 per cent a minute, so it takes twice as long: press **CLEAR** and spell `LN(2)/.075: = 9.2419624074829`, exactly double the first answer.

Predicting that needs no arithmetic at all. Halving the rate constant doubles every time in the problem, because the rate constant is the only thing setting the clock.

7.2 The window is the step

1. **-** doubles the window to -20 to 20, so the step is 40 over 127. Press **CLEAR** and type `40/127: = 0.31496062992126`, twice the old step.

At x = 0, twenty minutes have passed since the dose at -20. The truth is 9 times e to the minus 3: press **CLEAR** and spell `9*EXP(-3): = 0.4480836153109`.

The table reads noticeably below that. Predict the direction before you look: Euler undershoots a curve bending upwards, always, and doubling the step doubles the shortfall.

2. Five minutes after the dose is x = -5 in the standard window. The step is 0.15748, so -5 is 31.75 steps from the left edge, and the nearest column is the 32nd, at about x = -4.96.

Trace there and compare with `9*EXP(-.75)`, which answers `= 4.2512989746699`. The trace will read a little low, by about 1.8 per cent, which is section 7.2 step 3's figure.

3. Storing something in slot 2 changes nothing: the `Y2` column still reads – throughout.

The mode integrates slot 1 alone, as the note at the end of section 7.1 says. Slots 2 and 3 exist because the graph screen has three slots in every mode, not because this mode can use them.

4. Two presses of + from standard leave the window at -2.5 to 2.5, so the dose is at -2.5 and x = 0 is two and a half minutes after it.

The truth there is 9 times e to the minus 0.375, which is about 6.19. So the `X=0` row should read a little under that, and the value has gone *up* from 1.972 to about 6.18 without anything about the tank changing at all.

7.3 Step-size experiments

1. A third press of + leaves the window at -1.25 to 1.25, so the dose is at -1.25 and 3.5 minutes after it is at x = 2.25, which is outside the window.

The row reads `UNDEF`, because the run is the window. So refining the step by zooming eventually refines it out of the range you wanted to measure. That is the trap: the zoom keys control the step and the interval together and you cannot have one without the other. Section 7.4's program exists precisely to separate them.

2. For a gap under 0.001 at 0.21 times the step, you need the step below 0.001 over 0.21. Press CLEAR and type `.001/.21: = 0.0047619047619048`.

From the standard window's step of 0.15748, halving repeatedly: press CLEAR and spell `LN(.15748/.00476)/LN(2): = 5.0480632338422`.

So five halvings gets you close and six is needed to be sure. Six presses of + leaves the window at about -0.16 to 0.16, which is a third of a minute of tank. The measurement would be impossible.

3. Predict doubling: the gap is proportional to the step and the step doubles, so the gap should go from 0.034 to about 0.068.

Take - from standard and look up the 3.5-minute row, remembering that the dose has moved to -20 so you want x = -16.5. The reading falls further short than

any in the table, and the gap comes out around 0.068.

4. The exact solution is 9e^(-0.15t), whose second derivative is 9(0.15)²e^(-0.15t). At 3.5 minutes after the dose that is 0.0225 times 5.324, which is about 0.12, and half of it is 0.06.

That is not 0.21, and the reason is that the constant relates the *global* error after many steps to the step size, not the local error of one step. The global constant accumulates over the whole run, which is why it is several times bigger. Working out which of the two you have measured is the useful part of the exercise.

7.4 An Euler program

1. Predict half of 0.026, so about 0.013.

Run at `.0625→H` with `56→N`: `5.310830096248`, against a truth of `5.3239982793029`. The gap is 0.0132.

Halved again, exactly as first order promises, and the constant gap over step stays near 0.21 for a fourth time.

2. Change line 1 to `18→Y` and run at `.5` and `7`: `10.429527517055`.

The truth is 18 times e to the minus 0.525: press CLEAR and spell `18*EXP(-.525):=` `10.647996558606`. The gap is 0.2185.

That is exactly double the gap from a seed of 9, which was 0.109. So the error scales with the size of the solution, and it was predictable from line 5: every term in the walk is proportional to `Y`, so doubling the initial value doubles everything including the error.

The *relative* error is unchanged, which is the more useful way to say it.

3. Storing `-.3*Y` and rerunning gives a number well below the truth of 9e^(-1.05), which is about 3.15.

The gap-over-step constant does *not* hold: it roughly doubles, because the constant depends on the second derivative of the solution, and doubling the rate constant quadruples that while halving the time scale. Working out which way those two effects combine is the exercise, and the answer is that the constant scales with the rate constant.

4. Delete line 1 and run twice. The first run starts from whatever the last plot left in `Y` and gives a wrong answer; the second starts from wherever the *first run*

finished, and gives a different wrong answer.

Two runs of the same program returning different numbers is the clearest possible demonstration of why line 1 is there. A program that does not seed its own state is not a program, it is a continuation of whatever happened before.

7.5 Improved Euler

1. With line 5 back to `Y+H*EVAL(0)->Y` and `P3` deleted, the driver alone reproduces `5.2147637585268` at `.5` and `7`, which is section 7.4's plain Euler exactly.

That proves the improvement lives entirely in `P3`. The driver is bookkeeping: seed, loop, count, display. Swapping the called slot swaps the method and touches nothing else, which is the whole argument for splitting them.

2. The midpoint step is: k is the slope at y, go half a step to y + hk/2, take the slope there, then step a *whole* h from the original y with that slope.

The line that must remember the original `Y` is the catch. Write it as `EVAL(0)->K`, then `Y+H*K/2->Y`, then `Y+H*(EVAL(0)-K/2)->Y`… work it out carefully: after the half step, `Y` holds y + hk/2, and you want to end at y + h times the new slope. Since y is now y + hk/2, you need to add h(new slope) minus hk/2, so the third line is `Y+H*(EVAL(0)-K/2)->Y`.

Four lines again, and it fits.

3. At `20/127->H` with `64->N`, improved Euler lands very close to the true value at the window's middle, while the plot's own walk landed at `1.9489119504254`.

The difference is around a thousandth of a gram per litre, which on the screen is a fifth of a pixel. So the plot and the better method are visually identical and numerically a thousand times apart, which is worth knowing before you trust a picture.

4. Euler undershot and Heun overshot because the tank's solution bends upwards. For the reverse of both, take a solution that bends *downwards*: `.15*Y` seeded at a small positive value grows, and its curve bends upward too, so try `-.15*Y` seeded *negative*, or a rule like `.15*(9-Y)` seeded above 9.

Test it and you should find Euler now overshoots and Heun undershoots. The signs follow the curvature, always, and knowing which way a solution bends tells you which way your integrator will be wrong before you run it.

7.6 Growth with a ceiling

1. Predict: seeded above the ceiling, the bracket 1 - y/10 is negative, so the growth rate is negative and the population falls.

It falls towards 10 from above, flattening as it approaches, and it never crosses. Seeded at 18 the curve drops steeply and then levels along the same ceiling the rising solution approached from below.

The bracket is what makes it go that way: it is the only factor that can change sign, and it changes sign exactly at the ceiling.

2. The inflection is at half the ceiling, which is 5. Press **CLEAR** and type `10/2:= 5`.

From the table, the rows at `X=-6` and `X=-5` read `4.410` and `5.653`, so the crossing of 5 sits between them. The steepest growth is in that interval, which is where the largest difference in the table appeared.

3. Doubling k to 1 doubles every growth rate, so the curve reaches the same shape in half the time: the S is compressed horizontally.

What does *not* move is the inflection *level*: it is still at y = 5, because K/2 does not involve k at all. The inflection happens earlier and at the same height.

Predicting which of those two moves is the point of the exercise, and the answer comes from the formula rather than the picture.

4. Press **CLEAR** and spell `10/EXP(1):= 3.6787944117154`, which is the Gompertz inflection level, against the logistic's 5.

Seeded at exactly 10, `LN(10/Y)` is `LN(1)`, which is 0, so the growth rate is 0 and the solution sits still forever. That is the ceiling as an equilibrium, and it works.

Seeded at 0 the rule asks for `LN(10/0)`, and division by zero is where it ends. The Gompertz model has no meaning at zero population, which is a real limitation of it: unlike the logistic, it cannot start from nothing.

5. The logistic reads `9.439` at `X=0`. To make a Gompertz pass through the same point, try values of k and watch the `X=0` row.

The two curves will agree at that one point and nowhere else, because they are different functions. Matching one point of two different models tells you nothing,

and this exercise exists to make that concrete: two curves through one point can disagree everywhere else, and choosing between models needs the whole shape.

7.7 Equilibria, and the lever that resets them

1. Predict: seeded at exactly 3, the bracket 3 - Y is zero, so the rate is zero and nothing moves. The solution is the constant 3.

Run the reset with `3–>Y` and the plot is a flat line at 3, which is the same flat line the mode drew before any equation was stored. The difference is that this time it is a genuine solution rather than a receipt for the seeding.

2. `.2∗(8–Y)–.5` vanishes when 0.2(8 - y) = 0.5, so 8 - y = 2.5, so y = 5.5.

Plot it from a seed of 9 and a seed of 2 and both flatten onto 5.5. The table confirms the level from either side.

Note that the equilibrium is not 8: the constant -0.5 shifts it. An equilibrium is where the whole right-hand side vanishes, not where the obvious bracket does.

3. The logistic's rate is 0.5y(1 - y/10), which vanishes at y = 0 and y = 10.

y = 10 is stable: below it the rate is positive and above it negative, so neighbours come back. y = 0 is unstable: just above it the rate is positive, so a population near zero grows away from zero.

Check by seeding near each. Seeded at 0.1 the solution climbs away from 0 and heads for 10; seeded at 10.5 it falls back to 10. Two equilibria of opposite character in one equation, which is exactly what makes the logistic worth teaching.

4. At the eight points around the origin the pair dx/dt = y, dy/dt = -x gives a direction perpendicular to the position vector, and always the same way round. So a solution must go round the origin in circles.

Section 7.8 draws it. Put `Y` in slot 1 and `–X` in slot 2, press **F1** (`SYS`), and the phase view plots exactly the circles you just deduced. The deduction is still the valuable half: signs at eight points told you the answer before the machine drew anything, and that reasoning carries to systems of any size, which this machine still cannot integrate.

5. Take a rate like `.1∗Y∗(Y–3)∗(Y–6)`, which vanishes at 0, 3 and 6.

Between 0 and 3 the product is positive times negative times negative, so positive: solutions rise towards 3. Between 3 and 6 it is positive times positive

times negative, so negative: solutions fall back to 3. So 3 is stable, and 0 and 6 are unstable.

Predicting the fate of a solution started in each gap needs only the sign of the product, which you can do in your head. Checking them all costs three keys each on the DEQ SETUP page: F3, the + or - presses that move Y0 where you want it, and F5. Check every one of them.

7.8 Two equations at once, and the phase plane

1. The damping term takes energy out, so the ring must close inwards: a spiral into the origin rather than a closed orbit. That is what you get.

The time trace shows a cosine whose height shrinks, which you could also have guessed. What the phase picture adds is *where* it is going: every orbit, from any start, ends at the same point. The origin is an attractor, and one picture shows that for all starting conditions at once, which no single time trace can.

2. A closed loop, and that is the whole answer. The populations cycle: prey rise, predators follow, prey crash, predators starve, prey recover. Because the curve closes rather than spiralling in or out, neither species dies out and neither settles down. The system is *neutrally* stable, which is a famous and slightly unrealistic property of this simplest model.

Watch the method here. Euler's outward drift will eventually push the orbit into extinction and it will look like biology. It is arithmetic. Check any conclusion against RK4 before you believe it.

3. Euler shows a visible spiral within two or three revolutions. Heun takes long enough that you will lose count, which is the point.

Section 7.5 measured the orders: halving the step divides Euler's error by two and Heun's by four. Per revolution the same ratio applies, so if Euler is obviously wrong after three turns, Heun should take roughly its square, around nine, and RK4 far more than you have patience for.

4. One revolution of dX/dT = y, dY/dT = -x takes 2 pi units of time. The view takes 128 samples, so the step wanted is 2 pi over 128, which is about 0.049.

The table step halves from 1, so the reachable values are 0.5, 0.25, 0.125, 0.0625, 0.03125. Four halvings gives 0.0625, which draws about 1.27 revolutions; five gives 0.03125 and about 0.64 of one. Neither is exact, and the near miss is the

answer to the question: the step you want is not a power of two, so you choose between a little more than one loop and a little less.

9.8 Solutions for Chapter 8

8.1 The pendulum, and the integral that will not behave

1. Predict: a smaller amplitude puts the singularity closer to the lower limit in relative terms but the interval is shorter, so the panels are narrower and one of them still lands near the top. The nonsense should persist and may well get worse, because the spike is just as infinite over a shorter interval.

Store `PI/12->A` and run `FNINT(0,A)` again. It returns another large meaningless number. Shrinking the amplitude does not rescue the method, because the trouble is the shape of the integrand at the endpoint and not the length of the interval.

2. Stopping short of the top keeps the integrand bounded, so the panels have a fighting chance.

`FNINT(0,A-.01)` answers `= 2.0731870024371` and `FNINT(0,A-.001)` answers `= 2.283248630129`.

They are climbing towards about 2.31, which is the true value of the integral. Each step closer to the top adds the sliver you were missing, and the answers converge from below because you keep leaving out the tallest part.

3. Press CLEAR and spell `4*SQRT(2.5/19.6): = 1.4285714285714`, which is ten sevenths.

Multiply that by step 2's best value, 2.283248630129, and you get about 3.262. Section 8.1's transformed answer was `3.3003427304458`.

They are close and not equal, because the truncated integral is still missing the sliver nearest the top. Push the cut to A - 0.0001 and the two routes converge. Same physics, two integrals, one of which the machine can actually do.

4. Press CLEAR and spell `COS(.3)-COS(PI/4): = 0.24822970793901`. Press CLEAR and spell `2*(SIN(PI/8)^2-SIN(.15)^2): = 0.24822970793908`.

Twelve digits, which is as close as two different routes through fourteen digits are going to get. The identity is confirmed numerically as well as algebraically, and doing that once is worth more than trusting the algebra.

5. With `K` at 1 the integrand becomes 1 over the square root of 1 - $\sin^2 x$, which is 1 over cos x. At x = pi/2 the cosine is zero and the integrand is infinite again.

So the substitution did not save you from everything: it moved the singularity from the amplitude to the top of the new range, and at K = 1 exactly it comes back.

Physically, an amplitude of a half turn is the pendulum balanced exactly upside down. It takes infinitely long to fall, so the period really is infinite, and the integral is right to diverge. The mathematics and the machine agree, and this is the one case in the section where the infinite answer is the true one.

8.2 How wrong is the textbook formula?

1. With the corrected term, the rule is 1 plus $A^2/16$ plus $11A^4/3072$.

At 10 degrees it gives `1.0019071814956` against the table's `1.0019071881423`: agreement to eight decimals where the two-term rule managed five.

At 30 degrees, `1.0174038622498` against `1.017408797595`: five decimals where two terms gave three.

At 90 degrees, `1.1760122921022` against `1.1803405990146`: two decimals where two terms gave none.

So it buys about three rows, not one. Each term roughly squares the range over which the rule is usable, which is the usual behaviour of a series and the reason people bother with the third term at all.

2. Interpolating between the 10-degree row (0.19 per cent) and the 30-degree row (1.71 per cent) puts one per cent at about 22 degrees.

The series route is sharper: solve $1 + A^2/16 = 1.01$, so $A^2 = 0.16$, so A = 0.4 radians, which is 22.9 degrees.

They agree to within the interpolation's error, which is what you would hope: the table is coarse and the series is exact for small angles, and one per cent is small enough for the series to be trusted.

3. The amplitude at which the error reaches one per cent does *not* depend on the length at all.

The ratio T over T0 involves only k, which is the sine of half the amplitude. Length and gravity both cancel out of the ratio, appearing only in the periods

themselves. So a garden swing and a grandfather clock reach one per cent of circular error at the same angle, which is a genuinely useful thing to know and slightly surprising.

4. The table's ratios are climbing ever faster: 1.002, 1.017, 1.073, 1.180, 1.373, 1.762. Extrapolating that trend, a half turn should be well beyond 2.

It is worse than that: it is infinite. At a half turn the pendulum is balanced upside down, and it takes forever to start falling, so the period has no finite value. The trend was heading for a vertical asymptote and the table stops just short of it.

5. No. The integrand 1 over the square root of 1 - $k^2 \sin^2 x$ is at least 1 for every x, because the denominator is at most 1. So the integral is at least pi/2, and the ratio is at least 1.

A real pendulum is always slower than its linearisation, never faster, and that follows from the shape of the integrand without computing anything.

8.3 Series against closed forms

1. Predict: `1E6` instead of `1E4` stops when the term falls below a millionth, which is when N passes 1000. So a hundred times as many terms.

The run takes a very long time indeed, and the answer lands near 1.6439, still short of 1.6449 by about a thousandth. That is the arithmetic of the gap being one over the term count: a thousand terms buys three decimals, and you can see why nobody computes pi this way.

2. `1/(N*N+N)` is 1/(N(N+1)), which telescopes: it is 1/N - 1/(N+1), so the sum to N terms is exactly 1 - 1/(N+1), and the closed form is 1.

Four decimals needs 1/(N+1) below 0.00005, so N above about 20000. That is far worse than the reciprocal squares, and the reason is that the terms fall like $1/N^2$ but the *tail* falls like 1/N.

Do the paper first and you know the answer before the machine has finished its first run.

3. For a third rather than a quarter, line 4 becomes `(1+S)/3->S`, and the closed form is a half rather than a third: the sum of $(1/3)^n$ from n = 1 is 1/2.

Predict both from the geometric series formula r/(1-r) before you type anything.

4. `1/N^3` at 10, 40 and 100 terms gives about 1.19753, 1.20196 and 1.20201.

Measure the shorter runs against the longest: the gaps are about 0.0045 and 0.00005. Going from 10 to 40 terms cut the gap by ninety, and 40 to 100 by another ninety.

The gap falls like one over the *square* of the term count, where the reciprocal squares fell like one over the count. One extra power in the denominator buys one extra power in the convergence, which is the general rule for these sums.

5. Summing `1/N` upwards and downwards to a hundred terms gives answers differing in the tenth or eleventh digit, further apart than the reciprocal squares managed.

The reason is that 1/N's terms are much bigger relative to the total. The last term added upwards is 1/100 against a running total near 5, so it is shifted only three places; but there are many more terms of comparable size being shifted, so the accumulated loss is larger. The reciprocal squares fall away so fast that only the first few terms matter at all.

8.4 Shooting at a boundary

1. Predict halving again: the coarse walk undershot by 0.0204, the doubled one by 0.0102, so `20/508–>H` with `508–>N` should undershoot by about 0.0051.

Run it and shoot again. The inlet lands around 10.13, halfway back to 10 from 10.265. Each halving of the step halves the distance from the true answer of 10, exactly as a first-order method promises.

2. For a consent of 1.5, the closed form gives the inlet as 1.5 over (1 - 0.6), which is 3.75. Press **CLEAR** and type `1.5/(1–.6): = 3.75`.

Shoot for it with the program and you land near 3.8, high for the same integrator reason as before.

For a consent of 3, the closed form asks for 3 over (1 - 1.2), which is negative. There is no answer, and the physics says why: the outlet concentration of this bed is at most 1 over 0.4, which is 2.5, however much you feed it. Press **CLEAR** and type `1/.4: = 2.5`. A square-law decay over 20 metres cannot deliver more than 2.5 milligrams per litre no matter what goes in, so a consent of 3 is met automatically and a consent of 3 is not a constraint at all.

3. Aiming on length means line 2 becomes `L/127–>H` for your chosen length L, with `127–>N` unchanged and `10–>Y` on line 1.

The closed form becomes inlet over (1 + 0.02L times inlet), so with an inlet of 10 you want 10/(1 + 0.2L) = 2, giving L = 20. That is the length the section already used, so shooting on length with the original inlet should return you to 20 metres, which is a useful check that both aiming directions agree.

4. Aiming from one miss means guessing the slope of the miss-versus-seed relationship rather than measuring it.

It costs more shots, typically two or three extra, and it can diverge if your guessed slope has the wrong sign. What the second miss buys is exactly that slope: two points determine the line, and the line is the whole method.

That is the secant method, and it is Newton's method of section 5.2 with the derivative estimated from two points rather than computed. The connection is worth noticing: chapter 5 hunted a root of an equation, and this hunts a root of "the miss as a function of the seed", which is the same problem wearing different clothes.

5. Separating $dy/dx = -0.02y^2$ gives $-dy/y^2 = 0.02\,dx$, so $1/y = 0.02x + C$. With $y(0) = c$, C is 1/c, so $1/y = 0.02x + 1/c$, and

$y = c / (1 + 0.02xc)$.

At x = 20 that is c/(1 + 0.4c), which is step 12's expression exactly.

Deriving it takes two lines and confirms that the closed form the section pulled out of the air is the honest solution of the equation rather than a convenient fiction.

8.5 Vectors in the round

1. The third anchor is at bearing 240 and distance 9. Type 9, 240, 0 into `A` and press `CY>R`.

The components come back as about -4.5 and -7.794, mirroring the second anchor's -4.5 and +7.794 as the geometry demands.

The guy from the mast top is then (-4.5, -7.794, -12), and its `MAG` is 15, the same as the other two. It has to be: all three anchors are 9 metres out and the mast is 12 tall, so all three cables are the hypotenuse of the same right triangle.

2. Predict 10800, by symmetry: the three guys are identical apart from their bearing, so each pulls the mast top with the same force at the same distance, and a moment depends only on those.

Press `CRS` with the position (0, 0, 12) in `A` and the second tension (-450, 779.42286341, -1200) in `B`. Stepping through `R` reads `–9353.07436092`, `–5400`, `0`.

Different components from the first guy's `0`, `10800`, `0`, because the moment points a different way. Press `MAG`: `10800.000000024`.

Same size, different direction. That is exactly what three-fold symmetry means, and it is why the three moments can cancel.

3. Predict both before pressing anything. Anchors 6 metres out instead of 9 make the guys steeper, so more of each tension pulls downward and less pulls outward: the downward force *increases*.

And the angle between two guys *decreases*, because the tops are unchanged while the anchors have moved closer together.

Check by rebuilding the vectors with 6 in place of 9 and 12 unchanged. The cable length falls from 15 to about 13.4, the downward component of each tension rises, and `ANG` returns less than 62.6 degrees.

4. Three coplanar vectors: (1, 0, 0), (0, 1, 0) and (1, 1, 0), all in the xy plane.

Cross the first two to get (0, 0, 1), then dot with the third: zero. The box they span is flat, so its volume is nothing.

Nudge the third to (1, 1, 0.001) and the triple product becomes 0.001. It stops being zero immediately and in proportion to the nudge, which makes it a usable test rather than a knife edge: a small triple product means nearly coplanar, and how small tells you how nearly.

5. Two guys and the mast are (9, 0, -12), (-4.5, 7.794, -12) and (0, 0, 12).

Cross the two guys and dot with the mast. The answer is the volume of the box those three span, and it is large: the two guys are well separated in direction and the mast is a long way from their plane.

A small answer would mean the three are nearly coplanar, which would mean the mast lies nearly in the plane of two of its guys. That is exactly the configuration in which the guying fails: a mast braced by wires that all lie in one plane with it has no restraint at all perpendicular to that plane. So the triple product is a bracing quality measure, and bigger is better.

AFTERWORD

After the last exploration

Eight chapters have used one small machine as a laboratory rather than as an oracle.

Here is what it has, written out plainly. Three graph slots. Lists of eight samples. Matrices of three rows and up to six columns, and square-only work confined to 3 by 3. Vectors of three components. Polynomials to degree 4. Four programs of eight lines. An entry line of forty-eight characters, and a program line of the same forty-eight, because they share a buffer. Two coupled first-order equations in the DifEq mode, and one equation in the solver. Fourteen significant digits, which is seven bytes of packed decimal.

Every one of those numbers is a decision I made, mostly about how much of a very small machine to spend on one feature. And every one of them has shaped an exploration in this book.

Some of them closed a door, and this book said so each time rather than pretending otherwise. Then most of those doors opened.

That paragraph used to list six of them. A graph slot could not hold `FNINT(`, so there was no plot of pendulum period against amplitude. Sine gave up a little short of four hundred radians. There was no phase plane, because the mode integrated one equation from one initial condition. There was no simplex tableau, because even a two-product problem needs three rows and six columns. There was no window editor. There was no way to hold one plot on the screen while another drew over it.

Every one of those is now false. The calculus commands take an explicit slot and plot as functions of x. Reduction is by quotient and runs to a million radians. DEQ integrates two coupled states and draws the phase plane. The matrix workspace is three rows by six columns, which are the exact dimensions this paragraph named as the reason simplex was impossible. `WIN` gives four numeric fields. `OVR` lays a stored picture under the next graph.

Two of those I wrote as permanent and they lasted one release. One of them I described as a design decision when it was a thing I had not got round to. Only the polar sweep of exactly one turn, and the absence of an expression that would let `SOLV` aim a shooting method for you, are still standing.

Each time a door was shut, the exploration went round instead: a table written out row by row, an identity applied on paper before the machine was asked, a straight line drawn through the last two misses. Those detours are still in this book, and I have kept them, because the going-round is where the mathematics was. What has changed is that they are now choices rather than requirements, and the book says which is which.

I want to be careful about what I am claiming here, because there is a comfortable and slightly dishonest version of this argument that says limitations are secretly good for you. They are not. I wrote in an earlier edition that if I could have given you nine graph slots and a phase plane I would have, and that the book would have been better for it in places. I have since given you the phase plane, and the book is better for it in places, exactly as predicted. The graph slots are still three.

What is true is narrower and I think more interesting. A bound you can see the whole of teaches you something a bound you cannot see does not.

Three slots made a family of curves something to compare rather than skim. Eight samples made every statistic checkable by hand, so the machine was never believed, only checked. A 3 by 3 world made an elimination small enough to watch every entry change. Eight lines made each program an argument about what its algorithm really is, because there was no room for anything that was not the argument. Forty-eight characters forced the sum of squared residuals into a program, where it became something you run repeatedly rather than something you evaluate once. Two more belong on that list, and they are no longer there.

The initial condition in differential-equation mode used to be frozen once it was set, so changing a shot meant deleting a store object and starting over, and I argued that the cost of a shot was part of the lesson. `FNINT(` used to spread sixty-four panels across whatever interval you gave it and hand back the total, which meant that twice in this book the machine gave you a confident, wrong answer and you needed enough mathematics to catch it. I argued that too, and rather well, in the edition of this book that came before this one.

Both arguments were comfortable, and one of them was wrong. A limitation can teach you something and still be a defect, and the test is not whether a reader learns from working around it, because a reader can learn from almost anything. The test is whether the machine was honest. A frozen seed was honest: it never pretended to be anything else, and you could see the whole of it from outside. The silent integral was not. It answered in the same voice it uses when it is right, and no amount of pedagogical benefit buys that back.

So the seed is now an ordinary editable setting, and `FNINT(` compares its estimates and refuses when they will not agree. The pendulum still needs the substitution: nothing about the mathematics moved, and the trigonometry that turns an improper integral into a proper one is still the thing worth learning. What moved is that you now find out you need it.

That is the distinction I was reaching for and did not have. Not a limitation being secretly good for you. A limitation being *visible*, at a size where you can still see all the way round it, at the exact moment when seeing it teaches you the thing.

The other half of the argument is about what these bounds are not. They are not the mathematics. The logistic equation does not care that this machine integrates one equation at a time. Newton's method does not stop doubling its correct digits because the program slot is eight lines long. The interval of convergence of a power series is a fact about the series. What the machine gave you was a place to stand while you looked at those things, and a set of edges to push against so that you found out where they are.

If you have worked through the exercises, you have done something more useful than learning a calculator. You have practised the habit of predicting before pressing, and then finding out. That habit is the whole of experimental mathematics and it transfers to every tool you will ever use, most of which will be far bigger than this one and much less willing to show you their edges.

Put the book down, pick a question of your own, and see how far the machine takes it before you have to carry it yourself. That moment, when you find where it stops, is the most useful thing in here.

FREE85

Explorations with Free85 · For firmware 3.0 · Second Edition

Explorations with Free85 is the third of the project's books, after the Getting Started Manual and the Guidebook, and the first to be written as a workbook. Its explorations, worked examples, data sets, and exercises were invented for this machine; every key sequence and every quoted number in it was run on the emulator.

Free85 is clean-room software: the firmware, the font, the screen artwork, the tests, and this book were written from scratch for the project. It contains no Texas Instruments ROM code, disassembly, fonts, artwork, or binary tables. The TI-85 is referenced only to describe the hardware profile the firmware runs on; the project is not affiliated with or endorsed by Texas Instruments.

Free85 3.0 · Typeset from the Markdown sources with pandoc and Paged.js, and rendered to PDF by headless Chromium. Set in Charter, Helvetica Neue, and Menlo.

https://chriswilson2020.github.io/Free85/

F R E E 8 5

chriswilson2020.github.io/Free85