Using CAS calculators to teach and explore numerical methods

Alasdair McAndrew
Alasdair.McAndrew@vu.edu.au
School of Engineering and Science
Victoria University
PO Box 14428, Melbourne 8001
Australia

Abstract

We describe the use of CAS calculators in a numerical methods mathematics subject offered to third year pre-service teachers. We show that such calculators, although very low-powered compared with standard computer based numerical systems, are quite capable of handling text-book problems, and as such provide a very accessible learning environment. We show how CAS calculators can be used to implement some standard numerical procedures, and we also briefly discuss the pedagogical values of our approach.

1 Introduction

Numerical methods: the area of mathematics concerned with finding approximate solutions to intractable problems, has long been the preserve of high-powered computers and computer systems. And the increasing speed and power of computers, and the development of accessible software, has meant an increase in the availability of new methods. However, teaching of such material has either meant accessibility to computers and software, or restriction to simplified problems which can be done on a scientific calculator. The advent of CAS calculators, such as the TI-nspire and the Casio ClassPad, has meant that for the first time students have easy access to a powerful computing environment, and one also which they are likely to have seen at school.

Note that since this article was first written a third CAS calculator: the HP Prime, has entered the market. It is not yet clear what impact this will have on the use of such calculators in education.

At Victoria University, (Melbourne, Australia), a "Computational Methods" subject is offered as an elective for third year pre-service teachers who have chosen mathematics as their principal teaching "method". The students have had a solid grounding in calculus, statistics, and some algebra, and have satisfied the requirement by the local government body to be able to teach mathematics up to and including upper secondary levels. The students have also had

some exposure to the use of technology, in particular CAS calculators earlier in the course. The computational methods subject is designed to provide the students with a greater expertise in the use of CAS calculators—including programming—and also to introduce them to a mathematical discipline (numerical methods) which they are most likely to encounter as teachers or practitioners. Note that the current curricula mandates the use of CAS calculators at year 12 (the final year of secondary school), and so an expertise in their use is now necessary on the part of mathematics teachers

2 Numerical methods and CAS calculators

In the course only a small selection of topics is covered: time constraints prevent a "complete" introduction to numerical methods, and some important topics are excluded. We do not for example spend much time on rounding errors, eigensystems, or functional approximation. We aim to introduce basic methods to solve non-linear equations and of linear systems, interpolation and numerical differentiation, numerical integration, and the solution of initial value problems.

In this section we briefly discuss each of the above topics, and show how they can be developed and implemented on a CAS calculator. We note that earlier work, for example by Gander and Gruntz [**gand99**] discusses numerical computing from the perspective of a symbolic, rather than a purely numerical, system, no other research has investigated numerical methods on CAS calculators.

2.1 Non-linear equations

Aside from the quadratic formula, students tend to have little exposure to non-linear equations. And aside from the quadratic formula and its cousins, the cubic and quartic formulas (which are in any case too complicated to use in practice), there are few generic formulas for the solution of such equations. So a numerical approach is often the simplest. It is also not immediately obvious whether an equation has a solution expressible in closed form. For example, the equation:

$$x^5 + x - 1 = 0$$

has one real and four complex roots, and all can be obtained using algebraic tools. The real root has the closed form

$$(t^{1/3} + t^{-1/3} - 1)/3$$

where $t = (2\sqrt{69} + 25)/2$. However, the apparently similar equation:

$$x^5 + x^2 - 1 = 0$$

cannot be solved by standard algebraic means, and there is no straightforward method of expressing its solutions in closed form.

We consider two classes of root-finding algorithms:

1. "Bracketing methods". We start with two values a and b for which $f(a)f(b) < 0$, so that by the mean value theorem (assuming f is continuous on the closed interval $[a, b]$) there is a root ξ with $a \leq \xi \leq b$. A bracketing method successively shortens the interval around the root.

2. Derivative methods. These methods invoke the derivative f' of the function.
In each case we have a pair of previous values x_{n-1}, x_n or a single previous value x_n, and we produce a new value x_{n+1} which will be closer to the root than before.

The simplest bracketing method is the method of bisection: given a and b, set $t = (a+b)/2$. Then consider the sign of $f(t)$. We set which ever of a and b has the opposite sign to the value of t, and repeat as often as we need. Thus at each stage the length of the interval is halved. In terms of a sequence of values, if the current root is bracketed by $[x_{n-1}, x_n]$, then we define $t = (x_{n+1} + x_n)/2$ and

$$[x_n, x_{n+1}] = \begin{cases} [t, x_{n+1}] & \text{if } f(t)f(x_{n-1}) < 0, \\ [x_n, t] & \text{otherwise.} \end{cases}$$

Here's how it could be implemented on each calculator, for solving $x^5 + x^2 - 1 = 0$, given that there is a root between 0 and 1:

Using the TI-nspire CAS calculator	Using the Casio ClassPad calculator

Enter the function $\mathsf{f(x)} := \mathsf{x}^5 + \mathsf{x}^2 - 1$ and a little function called "bisect1" which will perform a single bisection step:
 	
Define bisect1(a) = Func
Local t
$\mathsf{t} := \frac{\mathsf{a[1]} + \mathsf{a[2]}}{2}$
If $\mathsf{f(t)} \cdot \mathsf{f(a[1])} < 0$ Then
Return {a[1], t}
Else
Return {t, a[2]}
EndFunc
	As with the TI-nspire we enter the function f(x), and also create a small bisect1 program with a single parameter a (which will be a list of two values):
	
	Local t
	(a[1]+a[2])⇒t
	If f(a[1])×f(t)<0
	Then
	Return {a[1],t}
	Else
	Return {t,a[2]}

With each calculator the function can now be called as many times as liked (watching the two endpoints getting closer together), or used within another program which either runs bisect1 a given number of times or (better) until the difference between the endpoints is less than a give value, such as 10^{-6}.

The standard derivative method is of course Newton's method (or the Newton-Raphson method), for which

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Since each CAS calculator can perform symbolic derivatives, implementation is straightforward.

Using the TI-nspire CAS calculator	Using the Casio ClassPad calculator

Define $f(x) = x^5 + x^2 - 1$
Define $nr(x) = x - \frac{f(x)}{\frac{d}{dx}f(x)}$
x:=0.8
For i,1,8,x:=nr(x):Disp x:EndFor
	`Define f(x)=x^5+x^2-1`
	`Define nr(x)=x-`$\frac{\texttt{f(x)}}{\texttt{diff(f(x),x)}}$
	`.8`
	`nr(ans)`

The ClassPad doesn't allow programming constructs (such as for-loops) outside of a program, so we can just start of a Newton-Raphson computation, and press the EXE key to repeat the previous command, and watch the values on the screen. Alternatively, we can use the Sequence module, and define the recursive sequence

$$a_{n+1} = nr(a_n), \quad a_0 = 0.8$$

and tabulate a few values.

Other methods, such as the secant method, regula falsi, can be easily implemented using very similar schemes to those shown.

We note that the calculators do of course have inbuilt routines for solving equations: for example the TI-nspire has "`nSolve`". However, the point is not simply to use the calculators as black boxes to find a solution, but teach the students the means by which such solutions are obtained. The students are encouraged to use the inbuilt routines to check their own solutions.

2.2 Systems of linear equations

We investigate two recursive methods for solving a linear system: the Gauss-Seidel method, and Jacobi's method. They are most easily described by an example, where equations are rewritten in recursive form:

$4x - y + 2z = 5$ $x + 5y - z = 7$ $2x - y + 6z = -8$ The original equations	$x_{n+1} = (5 + y_n - 2z_n)/4$ $y_{n+1} = (7 - x_{n+1} + z_n)/5$ $z_{n+1} = (-8 - 2x_{n+1} + y_{n+1})/6$ Gauss-Seidel iteration	$x_{n+1} = (5 + y_n - 2z_n)/4$ $y_{n+1} = (7 - x_n + z_n)/5$ $z_{n+1} = (-8 - 2x_n + y_n)/6$ Jacobi iteration

The main difference is that Gauss-Seidel always uses the most recently computed value of the variables, whereas Jacobi iteration uses the previous values for a complete iteration. Both methods are guaranteed to converge if the coefficient matrix is "diagonally dominant"; that is, the absolute value of the diagonal element is strictly greater than the sum of the absolute values of all other elements in its row. This condition will be satisfied by the given example. Suppose we write the equations in matrix form $A\mathbf{x} = \mathbf{b}$ and split A into three parts: the diagonal elements D, the upper triangular section U and lower triangular section L, so that $A = U + D + L$. From the examples above, it is not hard to show that the Gauss-Seidel and Jacobi iterations can be written as

$$\mathbf{x}_{n+1} = (D + L)^{-1}(\mathbf{b} - U\mathbf{x}_n), \qquad \mathbf{x}_{n+1} = D^{-1}(\mathbf{b} - (L + U)\mathbf{x}_n)$$

respectively.

Using the TI-nspire CAS calculator	Using the Casio ClassPad calculator

For the Gauss-Seidel method we start by entering three initial guesses for x, y and z , and then apply the iteration:
 	
$\mathsf{x} := 1 : \ \mathsf{y} := 1 : \ \mathsf{z} := 1$
For $\mathsf{i}, 1, 10 : \mathsf{x} := \dfrac{5 + \mathsf{y} - 2 \cdot \mathsf{z}}{4}$:
$\mathsf{y} := \dfrac{7 - \mathsf{x} + \mathsf{z}}{5} : \mathsf{z} := \dfrac{-8 - 2 \cdot \mathsf{x} + \mathsf{y}}{6}$:
Disp x, y, z : EndFor
	To use a loop for the Gauss-Seidel method, we need to write a program, called, say `gs`:
	
	`{1,1,1}⇒{x,y,z}`
	`For 1⇒i To 10`
	`approx((5+y-2× z)/4)⇒x`
	`approx((7-x+z)/5)⇒y`
	`approx((-8-2× x+y)/3)⇒z`
	`Print {x,y,z}`
	`Next`
	
	and this program can be run from within the Program module.

Using the TI-nspire CAS calculator	Using the Casio ClassPad calculator
For Jacobi's method, use the matrix version. First enter the matrix a of coefficients, then extract the diagonal d. The rest of the matrix $(L + U)$, will be obtained with $m - d$: d := diag(diag(m) m := a − d x := [[1,1,1]]' For i, 1, 10 : x := $\mathsf{d}^{-1} \cdot (\mathsf{b} - \mathsf{m} \cdot \mathsf{x})$: Disp x′ : EndFor	Jacobi's method can be implemented using the Sequence module; and entering the recursive definitions: $\mathsf{a}_{n+1} = (4 - \mathsf{b}_n + \mathsf{c}_n)/4$ $\mathsf{a}_0 = 1.0$ $\mathsf{b}_{n+1} = (2 - 2 \times \mathsf{a}_n - \mathsf{c}_n)/5$ $\mathsf{b}_0 = 1.0$ $\mathsf{c}_{n+1} = (11 - \mathsf{a}_n)/3$ $\mathsf{c}_0 = 1.0$ The sequence values can then be tabulated from within the Sequence module.

2.3 Interpolation

Interpolation is the problem of fitting a (piecewise) polynomial to a sequence of data points. Although we investigate cubic splines in the course, we just show here how to use Lagrangian and Newton interpolation.

Given a set of $n+1$ data points $\{(x_i, y_i), i = 0, 1, 2, \ldots, n\}$ we can fit an n-degree polynomial to it. One standard method is the Lagrangian polynomial, defined as:

$$L(x) = \sum_{k=0}^{n} \frac{(x - x_0)(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0)(x_k - x_1) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)} y_k.$$

Note that in the fraction, the top line consists of the product of all terms $(x - x_i)$ except for $(x - x_k)$, and the bottom line of the product of all non-zero terms $(x_k - x_i)$. This polynomial is more easily generated by first defining

$$q(x) = (x - x_0)(x - x_1)(x - x_2) \cdots (x - x_n)$$

and then

$$L(x) = \sum_{k=0}^{n} \frac{q(x)}{(x - x_k)q'(x_k)} y_k.$$

That these two definitions of $L(x)$ are equivalent is an elementary calculus exercise. Alternatively, we can set the polynomial to be

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

and generate n linear equations for a_i by substituting in turn $x = x_k$ and $p(x) = y_k$. Then a_i can be found by standard linear methods. Suppose for example we wish to fit a cubic polynomial to the four points

$$(x_i, y_i) = (-3, -61), (1, -5), (2, -1), (5, 83).$$

Using the TI-nspire CAS calculator	Using the Casio ClassPad calculator

First define the x and y values:
 	
$\mathsf{xs} := \{-3, 1, 2, 5\}$
$\mathsf{ys} := \{-61, -5, -1, 83\}$
 	
Now using the second method:
 	
$\mathsf{q(x)} := \mathsf{product(x - xs)}$
 	
$\mathsf{sum}\left(\frac{\mathsf{q(x)}}{(\mathsf{x} - \mathsf{xs}) \cdot \left(\frac{\mathsf{d}}{\mathsf{dx}}(\mathsf{q(x)})|(\mathsf{x} = \mathsf{xs})\right)} \cdot \mathsf{ys}\right)$
 	
The TI-nspire has very elegant list handling, where an operation on a list will automatically be done on every element of the list. So in the final sum, all operations on the lists xs and ys are done on each corresponding element individually and finally added.
To find the polynomial by linear methods:
 	
$\mathsf{m} := \{\mathsf{xs}^3, \mathsf{xs}^2, \mathsf{xs}, \mathsf{xs}^0\}'$
$\mathsf{c} := \mathsf{simult(vx}, \{\mathsf{ys}\}')$
$\{\{\mathsf{x}^3, \mathsf{x}^2, \mathsf{x}, 1\}\} \times \mathsf{c}$
	The commands for the ClassPad are very similar to those on the TI-nspire. With `xs` and `ys` defined:
	
	`DelVar x`
	`Define q(x)=prod(x-xs)`
	$\mathtt{sum}(\frac{\mathtt{q(x)}}{(\mathtt{x} - \mathtt{xs}) \times (\frac{\mathtt{d}}{\mathtt{dx}}(\mathtt{q(x)})|(\mathtt{x} = \mathtt{xs}))} \times \mathtt{ys})$
	
	And the use of matrices is similar, except that the ClassPad doesn't have the equivalent of a `simult` command for solving matrix equations. So we pre-multiply by the inverse instead:
	
	`listToMat(xs^3,xs^2,xs,xs^0)⇒xv`
	`vx^-1× listToMat(ys)⇒c`
	`[[x^3,x^2,x,1]]× c`
	`[[x^3,x^2,x,1]]× c`

If the x values are equally spaced, then a method called the *Newton-Gregory difference formula* can be used. Suppose the x values are given by $x_k = x_0 + kh$ (so h is the common difference), and the y values are as before $y_0, y_1, \ldots, y_n$. Create a table of all successive differences of the y values; each row of which is denoted Δ, Δ^2, down to Δ^{n-1} which will be a single value. If the first values of the differences are denoted $\Delta_0, \Delta_1, \Delta_2, \ldots, \Delta_{n-1}$, then the interpolating

polynomial can be written as

$$\sum_{k=0}^{n-1} \binom{z}{k} \Delta_k$$

where $z = (x - x_0)/h$. For example:

x :	-3	-1	1	3	5
y :	185	-31	-39	-367	-1159
Δ	-216	-8	-328	-792	
Δ^2	208	-320	-464		
Δ^3	-528	-144			
Δ^4	384				

The circled numbers are the values $\Delta_0, \Delta_1, \Delta_2, \Delta_3, \Delta_4$, from top to bottom. Then:

$$\begin{aligned} p(x) &= \binom{z}{0}185 - \binom{z}{1}216 + \binom{z}{2}208 - \binom{z}{3}528 + \binom{z}{4}384 \\ &= 384 - 216z + 208\frac{z(z-1)}{2} - 528\frac{z(z-1)(z-2)}{6} + 384\frac{z(z-1)(z-2)(z-3)}{24} \end{aligned}$$

Substituting $(x + 3)/2$ for z in the above expression, and simplifying, produces

$$x^4 - 11x^3 - 17x^2 + 7x - 19$$

as the required polynomial.

To implement this, we could use lists of differences, but instead we shall demonstrate the use of the spreadsheet on the calculators. Both calculators provide a spreadsheet; in contrast to a standard numerical spreadsheet, these spreadsheets also allow symbolic calculations. We note in passing that although there is a considerable literature on the use of spreadsheets in mathematics teaching and learning: for example [**bill07**, **van06**], there has been no discussion of the use of symbolic spreadsheets.

Start by entering the values 0–5 in the first row and the y values below the 0 in the first column:

	A	B	C	D	E
1	0	1	2	3	4
2	185				
3	−31				
4	−39				
5	−367				
6	−1159				

In cell B2, enter the expression "= A3 − A2" and copy it into the block of cells B2–E6:

	A	B	C	D	E
1	0	1	2	3	4
2	185	−216	208	−528	384
3	−31	−8	−320	−144	—
4	−39	−328	−464	—	—
5	−367	−792	—	—	—
6	−1159	—	—	—	—

Notice that the row A2–E2 now consists of all the differences Δ_k. Now in cell A7 enter the formula

$$= \text{nCr}\left(\frac{x+3}{2}, A1\right)$$

and copy that into cells B7–E7. These cells should now contain the polynomials

$$185, \quad -108(x+3), \quad 26(x+1)(x+3), \quad -11(x-1)(x+1)(x+3), \quad (x+3)(x+1)(x-1)(x-3).$$

Finally, add them all by entering "= sum$(A7 : E7)$" in an empty cell somewhere. This will produce the interpolating polynomial.

Similar methods can be used to implement Neville's method, or the method of divided differences (see Cheney and Kincaid [**chen12**] for discussions of these.) Note also that our description above was in fact for the Newton-Gregory *forward* difference method; there are also *backwards* and *central* difference methods.

2.4 Quadrature

Quadrature, or numerical integration, is a vital topic in any numerical methods course, and deals with finding approximate values of definite integrals

$$\int_a^b f(x)\,dx$$

where $f(x)$ has an anti-derivative not (easily) expressible in closed form. Examples are the elliptic integrals

$$\int_0^\phi \sqrt{1 - k^2 \sin^2 x}\,dx$$

for $k^2 < 1$, which arose initially in conjunction with determining the arc length of an ellipse, but have been shown since to have very deep properties connecting with many other branches of mathematics. There are a huge number of different quadrature methods, and many of the most useful approximate an integral with a finite sum of the form

$$w_0 f(x_0) + w_1 f(x_1) + \cdots + w_n f(x_n)$$

where each $x_i \in [a, b]$. The values w_i are called "weights" and the x_i values "abscissae" or "ordinates". In general the weights and ordinates are chosen so that the expression will be exact for a particular class of functions.

One set of quadrature formulas are the "Newton-Cotes" rules, where the x_i are chosen to be equidistant, and for a given n the weights are chosen so that the approximation is correct for all $f(x) = x^k$ for $k \leq n$. For example, for $n = 4$, we have

$$\int_a^b f(x)\,dx \approx w_0 f(a) + w_1 f(a+h) + w_2 f(a+2h) + w_3 f(a+3h) + w_4 f(b)$$

where $h = (b-a)/4$, and we choose w_k so that the expression is exact for $f(x) = 1, x, x^2, x^3, x^4$. Since the weights will be independent of the limits of integration, we can choose a and b so that $h = 1$:

$$\int_0^4 f(x)\,dx \approx w_0 f(0) + w_1 f(1) + w_2 f(2) + w_3 f(3) + w_4 f(4).$$

The weights can then be found by substituting each of x^k for $f(x)$ in this expression, and so obtaining linear equations for the w_i values, which can then be easily solved. Given that

$$\int_0^4 x^k\,dx = \frac{1}{k+1} 4^{k+1}$$

the linear equations will be

$$\begin{aligned}
w_0 + w_1 + w_2 + w_3 + w_4 &= 4 \\
w_1 + 2w_2 + 3w_3 + 4w_4 &= 8 \\
w_1 + 4w_2 + 9w_3 + 16w_4 &= \tfrac{64}{3} \\
w_1 + 8w_2 + 27w_3 + 64w_4 &= 64 \\
w_1 + 16w_2 + 81w_3 + 256w_4 &= \tfrac{1024}{5}
\end{aligned}$$

These can be solved to obtain

$$w_0,\ w_1,\ w_2,\ w_3,\ w_4 = \frac{14}{45},\ \frac{64}{45},\ \frac{8}{15},\ \frac{64}{45},\ \frac{14}{45}$$

and so a general form for this approximation is

$$\int_a^b f(x)\,dx \approx \frac{2h}{45}\big(7f(a) + 32f(a+h) + 12f(a+2h) + 32f(a+3h) + 7f(b)\big)$$

where as above $h = (b-a)/4$. This particular quadrature rule is known as the *Newton-Cotes rule of order four*, or *Boole's rule*.

We show how this rule, and clearly other Newton-Cotes rules, can be easily developed on a CAS calculator.

Using the TI-nspire CAS calculator	Using the Casio ClassPad calculator
$\mathtt{n} := 4$ $\mathtt{ys} := \mathtt{seq}((\int_0^{\mathtt{n}} \mathtt{x}^{\mathtt{k}}\mathtt{dx}), \mathtt{k}, 0, \mathtt{n})$ $\mathtt{m} := \mathtt{constructMat}((\mathtt{j}-1)^{\mathtt{i}-1}, \mathtt{i}, \mathtt{j}, \mathtt{n}+1, \mathtt{n}+1)$ $\mathtt{m}[1,1] := 1$ $\mathtt{w} := \mathtt{simult}(\mathtt{m}, \mathtt{list}\blacktriangleright\mathtt{mat}(\mathtt{ys})')$ $\mathtt{w}'$	This has to be written as a program. $\mathtt{seq}(\int((\mathtt{x}^{\mathtt{k}}), \mathtt{x}, 0, \mathtt{n}), \mathtt{k}, 0, \mathtt{n}) \Rightarrow \mathtt{y}$ $\mathtt{fill}(0, \mathtt{n}+1, \mathtt{n}+1) \Rightarrow \mathtt{m}$ For $1 \Rightarrow \mathtt{i}$ To $\mathtt{n}+1$ For $1 \Rightarrow \mathtt{j}$ To $\mathtt{n}+1$ $(\mathtt{j}-1)\hat{}(\mathtt{i}-1) \Rightarrow \mathtt{m}[\mathtt{i}, \mathtt{j}]$ Next Next $1 \Rightarrow \mathtt{m}[1,1]$ $\mathtt{m}^{-1} \times \mathtt{listToMat}(\mathtt{y}) \Rightarrow \mathtt{w}$ Return $\mathtt{trn}(\mathtt{w})$

The matrix M of coefficients can be defined by $m_{ij} = (j-1)^{i-1}$, assuming that 0^0 returns 1. Both the ClassPad and the TI-nspire return "undefined", so that the value m_{11} has to be entered separately.

Having created weights, we can use such a rule to evaluate a definite integral. For example, suppose we approximate

$$\int_0^1 e^{-x^2}\,dx$$

which has a value ≈ 0.746824132812, using a value of $h = 0.05$, so that there are 5 uses of Boole's rule. Given the weights in an array w, and h, we can implement the approximation using

$$\int_0^1 e^{-x^2}\,dx \approx h\sum_{k=0}^{4}\left(\sum_{i=1}^{5} w_i f\left(\frac{4k+i-1}{20}\right)\right).$$

Note that both calculators adopt indexing whereby the first element of a list is indexed with 1. This expression can be entered into the calculators almost unchanged, and the result is 0.746824132917 which is in error by only about 10^{-10}. And in fact, for a Newton-Cotes rule of order m, applied to the integral

$$\int_a^b f(x)\,dx$$

a total of n times, so that $h = (b-a)/(mn)$, can be implemented with

$$\frac{b-a}{mn}\sum_{k=0}^{n-1}\left(\sum_{i=1}^{m+1} w_i f\left(a+\frac{mk+i-1}{mn}\right)\right).$$

This is slightly inefficient in that some function values will be computed twice, however in practice this inefficiency is not noticeable.

As with solving equations, both calculators can solve integrals numerically using inbuilt routines, and as we noted previously students are encouraged to use these routines to check their own solutions, and to approximate the errors in their calculations.

2.5 Differential Equations

At the beginning of this subject, the students will have had some exposure to the concept of differential equations, and to some simply solvable types, as well as a little modelling. We only consider first order initial value problems of the general sort

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0.$$

The most basic form of solution is *Euler's method*, where the solution is given in the form of ordered pairs. Starting with (x_0, y_0) and a "step-size" h, then

$$x_{n+1} = x_n + h$$
$$y_{n+1} = y_n + hf(x_n, y_n).$$

A problem with Euler's method is that in general it is very inaccurate, and errors tend to accumulate with each step. For example, suppose we take the IVP

$$\frac{dy}{dx} = \frac{1}{2}xy + \frac{x}{3}, \quad y(0) = \frac{1}{3}$$

which can be easily solved to produce

$$y = e^{x^2/4} - \frac{2}{3}.$$

With a step size $h = 0.5$, we can compute $y(x_n)$ by the exact solution, and the approximate values y_n as computed by Euler's method:

x_n	$y(x_n)$	y_n	Error
0.0	0.333333	0.333333	0.0
0.5	0.397828	0.333333	0.064494
1.0	0.617359	0.458333	0.159025
1.5	1.088388	0.739583	0.348805
2.0	2.051615	1.266927	0.784688
2.5	4.104066	2.233724	1.870343
3.0	8.821069	4.046468	4.774601
3.5	20.714276	7.581319	13.132957
4.0	53.931483	14.798307	39.133176

The errors clearly increase. This can be shown again in the diagram in figure 1.

Students can play with Euler's method very easily. First, an exact solution can be computed:

Using the TI-nspire CAS calculator	Using the Casio ClassPad calculator
$\mathsf{f}(\mathsf{x},\mathsf{y}) := \frac{\mathsf{x}\cdot\mathsf{y}}{2} + \frac{\mathsf{x}}{3}$ $\mathsf{dsolve}\left(\mathsf{y}' = \mathsf{f}(\mathsf{x},\mathsf{y}) \text{ and } \mathsf{y}(0) = \frac{1}{3}, \mathsf{x}, \mathsf{y}\right)$	$\mathtt{Define}\ \mathsf{f}(\mathsf{x},\mathsf{y}) = \mathsf{x}\times\mathsf{y}/2 + \mathsf{x}/3$ $\mathtt{dSolve}(\mathsf{y}' = \mathsf{f}(\mathsf{x},\mathsf{y}), \mathsf{x}, \mathsf{y}, \mathsf{x} = 0, \mathsf{y} = 1/3)$

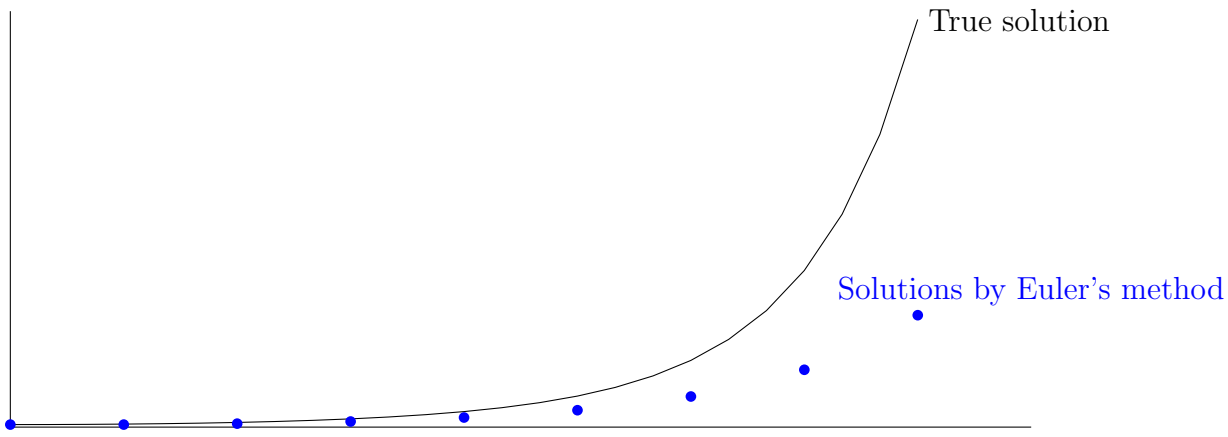

Figure 1: Euler's method

With each calculator, the solution can be turned into a function, say $s(x)$, which can be plotted. Euler's method can be implemented in a spreadsheet. Defining h as 0.5, a spreadsheet is created where column A contains the x values $0, 0.5, 1, 1.5, \ldots, 3.5, 4.0$, and cell B1 contains the value y_0; in this case $1/3$. In cell B2 enter "$= \texttt{B1} + \texttt{h} \times \texttt{f}(\texttt{A1}, \texttt{B1})$" and this formula can be copied down column B.

It is not hard to show how Euler's method can be improved; in fact Euler's method may be considered as first order Taylor series approximation; if

$$y(x+h) = y(x) + hy'(x) + \frac{h^2}{2}y''(x) + \frac{h^3}{6}y'''(x) + \cdots$$

then a truncation after the second term produces

$$\begin{aligned} y(x+h) &\approx y(x) + hy'(x) \\ &= y(x) + hf(x,y) \end{aligned}$$

which is Euler's method. Other methods provide accuracy equal to higher order Taylor series, and one very popular family of methods are the Runge-Kutta methods, where a high-order Taylor series is obtained by a judicious use of nested functions. These are extraordinarily difficult to develop—there is a great deal of complicated algebra involved—but the results can have a pleasing elegance. One fourth order method is defined as:

$$\begin{aligned} k_1 &= f(x_n, y_n), \\ k_2 &= f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right), \\ k_3 &= f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right), \\ k_4 &= f(x_n + h, y_n + hk_3) \\ y_{n+1} &= y_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

where as for Euler's method $x_{n+1} = x_n + h$. (One of the very few texts which provides a full algebraic construction of this method is the venerable text of Ralston & Rabinowitz [**rals65**].) Applying this to the above equations produces these values:

x_n	$y(x_n)$	y_n	Error
0.0	0.333333	0.333333	0.0
0.5	0.397828	0.397827	0.000001
1.0	0.617359	0.617345	0.000009
1.5	1.088388	1.088322	0.000044
2.0	2.051615	2.051205	0.000410
2.5	4.104066	4.101802	0.002264
3.0	8.821069	8.809320	0.011749
3.5	20.714276	20.654545	0.059731
4.0	53.931483	53.624082	0.307401

The errors are very much smaller than in Euler's method, even though they increase with each step. This is to be expected, and these errors can be made smaller either by using a smaller step size (with a step size of 0.1 and 40 steps, the approximate value at $x = 4.0$ is about 0.000848 in error), or by using two Runge-Kutta methods simultaneously, and adjusting the step size each step according to the errors between the two methods. Figure 2 shows the remarkable precision of a single Runge-Kutta method, even with a fairly large step size, as compared to Euler's method.

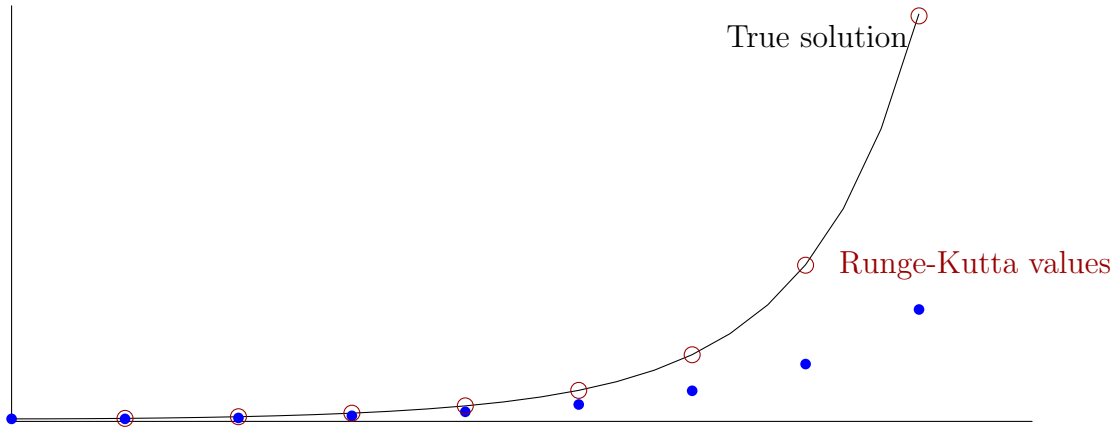

Figure 2: Runge-Kutta 4th order method

As for Euler's method, this can be implemented as a spreadsheet. Start by entering the x values (in our example from 0 to 4 in steps of $h = 0.5$) in column A and the value $y(0) = 1/3$ in cell B1. In cells C1 to F1 enter the values of the k_i: in C1 enter "`=f(a1,b1)`", in D1 enter "`=f(a1+h/2,b1+h/2*c1)`", in E1 enter "`=f(A1+h/2,B1+h/2*D1`" and in F1 enter "`=f(A1+h,B1+h*E1)`". Then in cell B2 enter "`=c1+h/6*(c1+2*d1+2*e1+f1)`". Then copy cells C1–F1 to cells C2–F2, and finally copy cells B2–F2 down as far as needed.

In both the TI-nspire and the Casio ClassPad there are methods for producing graphs similar to those shown in figures 1 and 2.

Newer versions of the calculators, or of their operating systems, include methods for computing Runge-Kutta or Euler steps. However, as for previous topics, we are keen to provide the students with some deeper knowledge about these methods' use and practice, hence we encourage creating the algorithms from scratch. It is also quite possible to write programs to implement these methods:

Using the TI-nspire CAS calculator	Using the Casio ClassPad calculator

Enter the function
$f(x,y) := \frac{x \cdot y}{2} + \frac{3}{x}$
and a program called "euler" which will perform as many iterations of Euler's method as required.
 	
Define euler(a,b,h,n) =
Prgm
Local xn,yn,xp,yp
xp := a
yp := b
Disp xp, yp
For i,1,n
xn := xp + h
yn := yp + h · f(xp, yp)
xp := xn
yp := yn
Disp xp, yp
EndFor
EndPrgm
	As with the TI-nspire we enter the function f(x,y), and also create a function `euler` which will have `a`, `b`, `h` and `n` as parameters:
	
	`Local xp,yp,xn,yn`
	`a⇒xp`
	`b⇒yp`
	`Print {xp,yp}`
	`For 1⇒i to n`
	`xp+h⇒xn`
	`yp+h×f(xp,yp)⇒yn`
	`approx(xn)⇒xp`
	`approx(yn)⇒yp`
	`b⇒yp`
	`Print {xp,yp}`
	`Next`

Note that in both programs, the values `xp` and `yp` represent the current values of x and y, and the values `xn`, `yn` the new values computed by Euler's method. It is a trivial matter to edit these programs to implement the Runge-Kutta method described above.

In the classes we show both approaches to students: spreadsheets and programs, and invite the students to choose their preferred method.

3 Conclusions

We have shown that many standard numerical tools can be developed and explored on a CAS calculator, and that in this respect the two current (as of the time of writing) competitors are equivalent in their functionality and their accessibility. And it is quite possible to go a great deal further: to investigate other methods than the few discussed above, for example topics such as error propagation, computation of eigensystems, or a deeper investigation into differential

equations. However a derivation of high-order Runge-Kutta methods, such as described by Gruntz [**grun95**] is beyond the power of a handheld device. Other topics (errors, eigensystems, approximation) can also be readily explored.

By their nature, numerical computations require a great deal of iterative computations with high precision real numbers. For this reason, most texts are based around a programming environment such as Matlab [**chap12**], or a programming language [**xu08**]. However, such environments are sometimes not easily accessible, or are very costly, or require a great deal of learning (especially in the case of a programming language) before students can feel comfortable with them. Using a CAS calculator, however, provides students with an environment which is portable, accessible, and one which the students may well have had prior exposure in their previous schooling.

We note also that the use of CAS (either on computers or calculators) as a pedagogical tool to introduce a topic, has a long history, see for example Heid [**heid98**, **heid13**]. Rather than use the CAS to explore difficult problems after the basic material has been mastered (this been a popular approach in tertiary teaching following Buchberger [**buch90**]), the CAS may be introduced at the very beginning. It has been shown that this use does not lead to any conceptual loss, or loss of understanding of a mathematical topic. It is this *ab initio* approach we espouse in this course. This approach also ties neatly in with constructive approaches to learning, in that use of a CAS not just as a black box but as a pedagogical learning tool allows the students' understanding and mastery to grow along with their mastery of the CAS.

Although this course has so far only run with small numbers of students, the reception has been extremely positive, and we expect that future cohorts will be as engaged with the material, and with its implementation, as students have been so far.

4 Acknowledgements

The author gratefully acknowledges the insightful comments of the reviewers, who suggested many ways that the original article could be improved.