

Calculus Demonstrations using MATLAB

Peter K Dunn*

Department of Mathematics and Computing,
University of Southern Queensland,
Toowoomba, Queensland, 4350, Australia

Chris Harman†

Department of Mathematics and Computing,
University of Southern Queensland,
Toowoomba, Queensland, 4350, Australia

Abstract

We discuss ways in which technology can be used in the calculus learning process. In particular, five MATLAB programs are detailed for use by instructors or students that demonstrate important concepts in introductory calculus: Newton's method, differentiation and integration. Two of the programs are animated. The programs and the graphical user interface have been specifically designed to help the student understand the processes behind these important introductory concepts. Each program has a series of demonstrations that show unusual, difficult or important cases.

1 Introduction

The use of technology in teaching calculus has become a popular component of most introductory calculus classes. The University of Southern Queensland has chosen to use MATLAB in its introductory algebra and calculus units. In their first years, students learn algebra and calculus by incorporating MATLAB as a numerical and graphical tool, and they learn the basics of its language and graphical commands.

A recently conducted research project [1, 2] analysed the influences on attitudes and learning, and found that almost all students responded positively to MATLAB for ease of computation and graphing. In addition, the use of the software as a tool was found to have had a strong impact on the learning strategies adopted and on their confidence towards mathematics. Others have found similar effects of the influence of computing methods on learning mathematics.

*Tel: +61 7 4631 5527, Fax: +61 7 4631 5550, e-mail: dunn@usq.edu.au

†Tel: +61 7 4631 5530, Fax: +61 7 4631 5550, e-mail: harman@usq.edu.au

For example, [3] found that “Students with high computer-mathematics interaction feel that computers enhance mathematics learning by providing many examples, enable the user to focus on major ideas by reducing mechanical toil, . . . and find computers helpful in linking algebraic and geometric ideas”.

In most other courses reported in the literature that use a computer package (an algebraic one like MATHEMATICA, MAPLE or DERIVE, or a numerical one like MATLAB), the software is mainly used to do the work for the student. The software is used to save the student time in fiddly or prone-to-error calculations, or to do more complex problems that are impossible to do ‘by hand’. This is, of course, an important use of computers in the classroom.

The programs discussed in this paper, however, focus more on using the computer to *show* what is happening, rather than to *do* a particular task. It is stated in [4, p 81] that

It is just about impossible to illustrate Newton’s method adequately with freehand sketches using a piece of chalk. A computer printout on a transparency, or even an animation using a PC, can be a great help. Likewise Simpson’s rule, the Runge-Kutta method, and other numerical techniques can be best illustrated with a computer and an attractive display.

He also says that “new technology should be layered atop a traditional foundation” ([4], p 28). The programs discussed herein are therefore aids to learning the topics, and cannot possibly be effective without adequate teaching around them. We would certainly not advocate that the programs be used as the sole method of teaching the topics.

It can be argued that MATLAB can be incorporated into learning in three main ways.

Firstly, it can be used as a numerical and graphical tool to enable faster illustration of the applications and better visualisation of complex ideas such as three dimensional graphical representations. The approach in this case is to just use command line instructions in MATLAB to help develop the ideas [1, 2]. No programming skills are required.

Secondly, MATLAB can be incorporated into the material by using appropriate programs which can be used by the student to aid the learning of ideas and concepts. The programs are usually available to the student for their use or modification. This approach has been used in several books such as [5] and [6], and also by [7], where a subject based around MATLAB is discussed and many excellent ideas for incorporating MATLAB into teaching are presented. A rudimentary knowledge of programming is helpful or necessary in this approach.

Thirdly, MATLAB software can be used to *show* the procedure or concept in a more sophisticated way by using larger, stand-alone and robust programs with well designed graphical user interfaces. No programming skills are required in this approach. Of course, these three approaches can effectively complement one another.

In this paper, we discuss five MATLAB functions that use this third approach. The programs were developed for student use and for use in lectures to graphi-

cally demonstrate Newton's method, differentiation and integration techniques. Two of the programs are animated, showing each step of the algorithm so that the algorithm is more easily understood.

First, some general comments. In each of the programs, any function compatible with MATLAB requirements can be used. The ability to use any function makes the programs generically interesting and useful. The functions must be written in standard MATLAB code when entered, so the MATLAB 'dot'-notation is used. That is, we would write $(\mathbf{x}.^2) .* (\mathbf{x}+1)$ rather than $(\mathbf{x}^2) * (\mathbf{x}+1)$. This dot notation implies element-by-element calculation, meaning MATLAB performs the calculations on corresponding elements of vectors or matrices, rather than using matrix operations.

Naturally, using incomplete or invalid MATLAB code can cause errors. Also, unstable or difficult functions may also cause the programs to behave strangely.

Each of the programs use larger than standard labels on the figures, and thicker than standard lines. This has been done to enable the programs to be used in a lecture situation by the instructor, so that the students can still see the details on the screen.

2 Newton's Method

The MATLAB function `numnewton` shows an animated demonstration of Newton's method. The initial screen is shown in Figure 1.

The top part of the screen is the display, where the action takes place. At the bottom of the screen are parameters that can be set. At the left, we have entry boxes for the function in MATLAB notation, the derivative of the function (also in MATLAB notation), the limits of the display, and an estimate of the zero to be found. By default, the function solves $f(x) = x^2 = 0$ near $x = 0.5$, and displays between $x = -0.4$ and $x = 0.6$. Any (legitimate) MATLAB functions can be used in this entry box using the MATLAB 'dot'-operators.

Note that the inclusion of the derivative is optional. If a derivative is not supplied, a numerical (central) derivative is calculated. While not as accurate, it enables the program to be used in any case, even when the function is complicated, or the derivative is difficult or impossible to find.

To the right of these entry boxes is a slider that determines the speed of the animation. The slider can be moved using a mouse or the keyboard up and down arrows. **Slow** gives the instructor time to talk about the animation as it proceeds, and **Fast** quickly runs the animation.

The buttons to the far right are fairly clear: **OPTIONS** sets convergence criteria (discussed later); **GO** starts the animation; **PAUSE** pauses for the instructor to talk about what is happening (it changes to **RESUME** after being pressed); **QUIT** shuts down the program.

[Figure 1 about here.]

Figure 2 shows the program in action, after three iterations using the default settings. Colours are employed for clarity: The initial estimate for the solution is shown as a green square; the current solution is shown in red, and previous

iterations are shown in gray. The program demonstrates the approach of the method by showing the tangent lines giving the next approximate zero of the function. In the MATLAB command window, a summary of the iterations is provided, giving the estimate of the root, the function value and the derivative at each iteration. For the default screen, the output is:

X-Value ~~~~~	Function ~~~~~	Derivative ~~~~~
0.50000	0.2500000	1.00000
0.25000	0.0625000	0.50000
0.12500	0.0156250	0.25000
0.06250	0.0039062	0.12500
0.03125	0.0009766	0.06250
0.01562	0.0002441	0.03125
0.00781	0.0000610	0.01562
0.00391	0.0000153	0.00781
0.00195	0.0000038	0.00391
0.00098	0.0000010	

Convergence detected: Function value close to zero.

Note that the OPTIONS button can be used to set the convergence criteria to other values.

[Figure 2 about here.]

The numnewton program can be used to demonstrate many of the problems with Newton's method, such as the necessity of a good starting point, cycling, a zero derivative, or an infinite derivative. To facilitate this, a menu item above the figure is included (called DEMONSTRATIONS) which sets parameters that show these situations. (A good discussion of these is found in [8], page 79, from where some of the examples are drawn.) The user selects the DEMONSTRATIONS menu item, and then selects the situation to be shown. The program demonstrates the situation by going through the animation.

It has been pointed out (see, for example, [9, p 67]) that the use of technology has its 'snags'. They point out that software that glosses over inherent complications in algorithms can leave the student "blissfully unaware that such problems are present and have been overcome". By specifically including some demonstrations of the problems inherent in Newton's method, we hope to have avoided this problem. Both [9, p 98] and [4, p 26] mention problems with using the computer to do calculations. In [4], the author states that "a computer does things so fast that it trivialises them. When something is trivialised, then understanding is lost." For example, some of these problems are discussed in [6, p 161], but not in [5]. We hope that the use of the program showing what the algorithm is doing, explicitly demonstrating shortcomings and allowing the instructor the capacity to slow the demonstration down to talk about the steps will address these concerns.

The demonstrations available are as follows:

- **Cycling:** This option sets the function as $f(x) = x^3 - x - 3$, and starts iterations at $x = 0$. This sets the algorithm into a loop which never converges, but continues to revisit the same estimates over and over again.
- **Cycling (2):** This option sets the function as $f(x) = \sqrt{|x|}$, and starts iterations at $x = 1$. The algorithm continues to oscillate between $x = -1$ and $x = 1$.
- **No root exists:** This option sets the function as $f(x) = x^2 - 4x + 4.5$ and the starting estimate at $x = 0.5$. This function has no zeros at all, but a minimum at $x = 2$ for which $f(2) = 0.5$. The algorithm attempts to find a zero near $x = 2$ in vain.
- **No root exists (2):** This option sets the function as $f(x) = \exp(-x)$ and the starting estimate at $x = 4$. Obviously, no zeros exist, but since $f(x)$ is very small, convergence is (incorrectly) reported with the default convergence options set. This is useful for demonstrating that the computer doesn't always give the correct answer!
- **Divergence:** This option sets the functions as $f(x) = \arctan(x)$ and the starting point at $x = 1.4$ for which $\arctan(1.4) \approx 0.95$. While this may not seem unreasonable, the algorithm actually diverges from the root.
- **Divergence (2):** This option sets the functions as $f(x) = \operatorname{erf}(x)$ and the starting point at $x = 1$. The derivative is left blank, so that numerical derivatives are calculated. The algorithm diverges from the root at $x = 0$. (Note that $\operatorname{erf}(x)$ is a special function—the MATLAB *error function*.)
- **Bad starting value:** This option sets the function as $f(x) = x^3 - x - 1$ and the starting point as $x = -0.5$ for which $f(x) = -0.625$. Again, this doesn't seem unreasonable, but the algorithm wanders far and wide in search of the root at $x \approx 1.32$. It can be eventually found after about 18 iterations.
- **Zero derivative:** This option sets the function as $f(x) = \sin x$ and the starting value as $x = 4.36918237257240$. After one iteration, the algorithm is evaluating the derivative at $x = \pi/2$ which is, of course, zero.
- **Zero derivative (2):** This option sets the function as $f(x) = x^3 - 3x^2 + 3x$ and the starting value as $x = 1.7937$. After one iteration, the algorithm is evaluating the derivative at $x = 1$ where the derivative is zero.
- **Convergence to wrong root:** This option sets the function as $f(x) = \sin x$ and the starting point to $x = 1.25$. The function converges to the zero at $x = -2\pi$, probably not the zero being sought.
- **Convergence to wrong root (2):** This option sets the MATLAB function $f(x) = \operatorname{humps}(x)$ and the starting point to $x = -0.55$, near the zero at $x \approx -1.1316$. This MATLAB function has 'two strong maxima' near

$x = 0.3$ and $x = 0.9$, and zeros at $x \approx -0.13162$ and $x \approx 1.3$. Numerical derivatives are used. Despite starting at $x = -0.55$, the function converges to the zero at $x = 1.3$, probably not the zero being sought.

- **Infinite derivative:** This option sets the function as the (obviously contrived) function $f(x) = \text{sign}(x)\sqrt{|x|} + 0.5$ with a starting value of $x = -1$. After one iteration, the derivative is evaluated at $x = 0$, for which the derivative is infinite.

As an example of cycling, the final screen from the first demonstration is shown in Figure 3.

[Figure 3 about here.]

A graphical user interface can be used to alter the convergence criteria for $f(x)$, the absolute error $|x_n - x_{n-1}|$ and the maximum number of iterations. The default values for these parameters are 0.0001, 0.00001 and 10 respectively. These criteria can be set using the **OPTIONS** button or the **OPTIONS** menu item above the figure. A second menu-only option under the **OPTIONS** menu item sets the convergence options back to the default values.

3 Differentiation

The MATLAB function `numdiff` shows an animated demonstration of differentiation. The initial screen is shown in Figure 4.

[Figure 4 about here.]

The top part of the screen is the display, where the action takes place. At the bottom of the screen are parameters that can be set. At the left, we have entry boxes for the function in MATLAB notation, the limits of the display, and the point at which to differentiate. By default, the function differentiates $f(x) = x^2$ at $x = 1$, and displays between $x = 0$ and $x = 2$. Any (legitimate) MATLAB functions can be used.

To the right of these entry boxes is a slider that determines the speed of the animation as discussed before. The three buttons that appear to the far right also appear in the program `numnewton`, and perform the same jobs.

After pressing **GO**, the animation begins. The function itself is plotted in blue, and the program demonstrates the definition of differentiation

$$\frac{df}{dx} = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

by calculating and showing the secant passing through $f(x)$ and $f(x + \Delta x)$ as $\Delta x \rightarrow 0$. This secant line is plotted in red, and is shown to approach the tangent. In the MATLAB command window, the values of Δx are shown together with the approximate (numerical) derivative for each Δx . For the default function $f(x) = x^2$, the MATLAB command window gives the following:

Delta-x	Approximate Derivative
~~~~~	~~~~~

1	3
0.8	2.8
0.5	2.5
0.3	2.3
0.1	2.1
0.01	2.01
0.001	2.001
1e-05	2
1e-09	2

A diagram of the animation after three iterations is given in Figure 5.

[Figure 5 about here.]

As with `numnewton`, there is also a DEMONSTRATIONS menu item above the figure for interesting cases. The demonstrations available are as follows:

- **Infinite Derivative:** This option numerically differentiates  $f(x) = \text{sign}(x)\sqrt{|x|}$  at  $x = 0$ , where the derivative is infinite.
- **Infinite Derivative (2):** This option numerically differentiates the modified Bessel function of the first kind,  $f(x) = I_{0,2}(x)$  at  $x = 0$ , where the derivative is infinite.
- **Zero Derivative:** This option numerically differentiates  $f(x) = 1 - x^2$  at  $x = 0$ , where the derivative is 0.
- **Zero Derivative (2):** This option numerically differentiates  $f(x) = \sin x$  at  $x = \pi/2$ , where the derivative is 0.
- **Difficult Derivative:** This option numerically differentiates  $f(x) = \sin(1/x)$  at  $x = 0.09$ . Because this function has rapid oscillations as  $x \rightarrow 0$ , good estimates are only found for very small  $\Delta x$ . Indeed, even though the derivative is negative, the numerical derivatives are positive until  $\Delta x = 0.001$ .
- **Difficult Derivative (2):** This option numerically differentiates  $f(x) = \text{humps}(x)$  at  $x = 0.3$ . `humps` is a MATLAB function with a strong maximum near  $x = 0.3$ , and so small  $\Delta x$  are required to give a good estimate of the derivative. Indeed, even though the derivative is positive, the numerical derivatives are negative until  $\Delta x = 0.00001$ .

## 4 Approximating Integrals

The remaining three programs are, in many ways, very similar: They all demonstrate numerical methods of approximating integrals. Consequently, they contain many similar components that can be discussed together. There are some differences, however, that will be addressed as necessary.

The three programs are `numrect`, `numtrap` and `numsimpson`.

- **numrect**: This program approximates an integral with rectangles. It uses the concepts of a right sum and a left sum to approximate the integral (as used in [10], Chapters 3 and 7, and similar to the concepts of upper sum and lower sum used by, for example, [11], page 262). The right sum uses the right value of the function on each interval, while the left sum uses the left value of the function on each interval. The area under the curve can be seen to be sandwiched between the two sums. The screen is shown in Figure 6.

The left and right sums are given in different colours so that they are easily distinguished. On the figure, the values of the left and right sums are both given, calculated from the displayed rectangles.

- **numtrap**: The second program uses the trapezoidal rule to approximate integrals. The screen is shown in Figure 7. The program plots the function of interest in green, and then the trapezoids used to approximate the integral in red.
- **numsimpson**: This program uses Simpson's 1/3 rule to approximate integrals. The screen is shown in Figure 8. Again, the function is plotted in green, and the parabolas used to approximate the integral are shown in red.

In each program, the top part of the screen shows the graphics, and the bottom again contains the parameters. The function is given (again in standard MATLAB notation), plus the limits of integration and the number of regions for which to calculate. The number of regions to use can also be set by a slider, from 2 up to 50 (for larger intervals, the programs can become slow, and the graphics become too cluttered). For Simpson's method, the program ensures an *even* number of intervals is used.

The buttons to the right are again clear: **GO** draws the picture after any changes have been made, and **QUIT** quits the program.

[Figure 6 about here.]

[Figure 7 about here.]

[Figure 8 about here.]

The programs can be used by setting all the required parameters and pressing the **GO** button. Alternatively, after changing the number of intervals with the slider, the program immediately is activated. This enables the student or instructor to see the effect of increasing (or decreasing) the number of intervals on the integral by using the mouse (or keyboard arrow keys) to move the slider. For each program, output is given in the MATLAB command window, indicating the approximation to the integral with the given number of regions. As an example, the output from **numsimpson** is given for approximating one of the demonstrations. Convergence seems to be occurring. Similar output is generated for **numrect** and **numtrap** also.

```
Integrating 1 ./ ( 1 + x.^6 ) between -3 and 4
```

Number of Panels ~~~~~	Approximate Integral ~~~~~
2	4.5967547
6	2.3248453
10	2.1666795
14	2.1038790
18	2.0784916
22	2.0943325
26	2.0899679
30	2.0928144
34	2.0935154
38	2.0933077
42	2.0934796
46	2.0933896
50	2.0933810

As in the Newton's method program, some demonstrations are included in the DEMONSTRATIONS menu item for interesting cases. The following demonstrations appear in all three programs:

- Difficult Regions: sqrt(x): This option approximates  $\int_0^2 \sqrt{x} dx$ . Most methods have trouble with accuracy near  $x = 0$ .
- Difficult Regions: exp( abs(x) ): This option approximates  $\int_{-2}^2 \exp(|x|) dx$ . Most methods have trouble at  $x = 0$ , where the function is not smooth.
- Difficult Regions: 1 ./ ( 1 + x.^6 ): This option approximates  $\int_{-3}^4 1/(1 + x^6) dx$ . Most methods struggle with this function because of the sharp changes at  $x = \pm 1$ .
- Difficult Regions: besselj(0.2, x): This option integrates the modified Bessel function of the first kind,  $\int_0^1 I_{0.2}(x) dx$ . Most methods have trouble near  $x = 0$ .
- Difficult Regions: sin( 1./ x): This option approximates  $\int_{0.1}^1 \sin(1/x) dx$ . This function is well-known to be difficult to integrate, as the number of zeros in the function increases rapidly as  $x \rightarrow 0$ .

Each of the three functions share these demonstrations. In addition, numtrap includes two other demonstrations:

- Under Estimation: x.^2 + 4: This option approximates  $\int_{-1}^2 x^2 + 4 dx$ . The trapezoidal rule will always *under*-estimate this integral.
- Over Estimation: exp( -x ): This option approximates  $\int_{-4}^1 \exp(-x) dx$ . The trapezoidal rule will always *over*-estimate this integral.

The program `numsimpson` also includes an extra demonstration:

- Exact Answer:  $x^3 - x - 1$ . This option approximates  $\int_{-1}^2 x^3 - x + 1 dx$ . For any number of regions, Simpson's rule will give an exact answer for any cubic.

## 5 Obtaining and Using the Programs

The programs are all free, and operate under MATLAB versions 5 and 6. Each program requires just one (text) file, and all are very easy to install. The user must place a copy of the files anywhere on the MATLAB path. The current MATLAB path can be established by typing `path` at the MATLAB prompt. In addition, making changes to the MATLAB path can be done by typing `pathtool` at the MATLAB prompt. The local documentation should be consulted for setting the MATLAB path, since the details may be different for different platforms and different MATLAB versions.

The functions are then run by simply typing the function name (for example, `numnewton`) at the MATLAB prompt.

The files are stored at the first author's Web pages. They can be downloaded from

<http://www.sci.usq.edu.au/staff/dunn/matlabprograms.html>

The programs are copyright by the author and can be used freely, but cannot be exploited for commercial reasons.

## References

- [1] Cretchley, P., Harman, C., Ellerton, N., Fogarty, G., 1999, Computation, exploration, visualisation: Reaction to MATLAB in first-year mathematics. *Delta 99 Proceedings, Australian Symposium on Modern Undergraduate Mathematics*, pp. 81–86.
- [2] Cretchley, P., Harman, C., Ellerton, N., Fogarty, G., 2000, *Mathematics Education Research Journal*, **12**, 219–233.
- [3] Galbraith, P., Haines, C. and Pemberton, M., 1999, *Making the difference*, edited by J. M. Truran and K. M. Truran (Sydney: Mathematics Education Research Group of Australasia Inc.), pp. 215–222.
- [4] Krantz, S. G., 1998, *How to Teach Mathematics* (American Mathematical Society).
- [5] Lindfield, G. and Penny, J., 1995, *Numerical Methods using MATLAB* (Ellis Horwood).
- [6] Borse, G. J., 1997, *Numerical Methods with MATLAB: a resource for scientists and engineers* (PWS-Kent Publishing).

- [7] Colgan, L, 2000, *International Journal of Mathematical Education in Science and Technology*, **31**, 15–26.
- [8] Mathews, J. H., 1992, *Numerical Methods for Mathematics, Science and Engineering* (Prentice-Hall).
- [9] Burn, B., Appleby, J., and Maher, P. (eds), 1998, *Teaching Undergraduate Mathematics* (Imperial College Press).
- [10] Hughes-Hallet, D., Gleason, A. M., McCallum, W. G., Flath, D. E., Lock, P. F., Gordon, S. P., Lomen, D. O., Lovelock, D., Mumford, D., Osgood, B. G., Pasquale, A., Quinney, D., Tecosky-Feldman, J., Thrash, J. B., Thrash, K. R., and Tucker, T. W., 1998, *Calculus: Single and Multivariable* (Wiley and Sons).
- [11] Larson, R. E., Hostetler, R. P. and Edwards, B. H., 1994, *Calculus with Analytic Geometry* (DC Heath and Company).

## Figure Captions:

These are the figure captions, on a separate page at the end as required.

Figure 1: The initial screen for the `numnewton` program

Figure 2: After three iterations of the `numnewton` program solving  $f(x) = x^2$ , starting at  $x = 0.5$ . The program numerically determines the root using Newton's method, and animates the solution-finding algorithm.

Figure 3: A demonstration of Newton's method cycling through iterations when solving  $f(x) = x^3 - x - 3$  and starting at  $x = 0$ .

Figure 4: The initial screen for the `numdiff` program. The default is to differentiate  $f(x) = x^2$  at  $x = 1$ .

Figure 5: After three iterations of the `numdiff` program. The program numerically differentiates  $f(x) = x^2$  at  $x = 1$ , and animates the secant lines approaching the tangent line.

Figure 6: The `numrect` program, approximating  $\int_0^2 \sin(x^2) dx$  using six intervals.

Figure 7: The `numtrap` program approximating  $\int_0^2 \sin(x^2) dx$  using six intervals.

Figure 8: The `numsimpson` program approximating  $\int_0^2 \sin(x^2) dx$  using six intervals.