Numerical Analysis: an Inquiry Based Approach
(Instructor Edition)

Bjork

June 2, 2015
1 Introduction

This set of notes was developed with the intent to help the students encounter various techniques from the field of numerical analysis. The primary goal is to have the students code computer programs that implement various classic algorithms from this discipline of mathematics. The author tried to keep the notes free from specific references to any particular software, though the course for which these notes were developed used Maple\textsuperscript{TM} 17.

Instructors should be careful to provide more instruction for software specific issues that could arise. For example, though Maple\textsuperscript{TM} has a very nice linear algebra package, it is unhappy with using the built in summation inside a loop (too many levels of recursion). However, the built in \texttt{add()} command posses no problems inside a loop. Even though there were frustrations inherent to discovering the commands that produced error free programs, the fact my students found these ended up being one of the greatest successes of the course.

The notes are organized in such a way that each section is a project. Some take a week, others a touch longer. I took two class periods for my oral midterm exam. We finished with the Gauss-Seidel technique on the very last day of the semester.

I split my class into groups of two or three, and kept the groups for the entire semester. I put people with similar computer science knowledge in the same group. This ensured that they would all participate in the coding. Each project had to be written up as a project, and I selected one group per project to present. I graded the presenters on both the content and the form of the presentation. The class was also graded on participation during the presentation. Each group was also required to turn in their projects to me electronically so that I could grade each project on soundness. The final exam was an oral presentation of a randomly selected project from the semester.

As a final comment, I am releasing these notes in the hope to get them class tested by others before submitting them for refereeing. Since numerical analysis is on a two year rotation at my institution, I welcome all the feedback I can get on these notes so that I may improve them. Thank you for your help in this endeavor.
2 Opening Act

We first explore a few of the tools you will use throughout the semester.

Problem 1. Consider the function

\[ f(x) = \frac{x \cos(x) - \sin(x)}{x - \sin(x)}. \]

After having stored this function, begin exploring values of this function. Make sure you try different kinds of numbers (integers, rationals, decimals, irrationals, transcendentals ...). Report on your findings, paying special attention to the formatting of the answer.

Now that you have some idea of how our software answers your prompts, let’s start our exploration. We are interested in using numerical methods to find

\[ \lim_{x \to 0^+} f(x). \]

Problem 2. Based on your knowledge of Calculus, make a conjecture as to the actual value of this limit. Is \( f \) continuous at 0?

- Write a loop that computes 20 values that approach the limit.
- Write a loop that stops once your approximations are within \( 10^{-5} \) of your earlier guess. Are there any dangers inherent to writing this kind of a loop?
- How many times does the program go through the loop? Can it be made faster? Are there any limitations?
- Use this tool to run a similar analysis on the behavior of \( g(x) = x^2 \sin(x) \) near 0.

Teaching Notes: This section is really designed to get the students started with programming mathematics. My students came with a very wide curve of programming abilities: some had no programming background whatsoever, most were currently enrolled in a Visual Basic course, while a couple were very knowledgeable (one was a professional web designer). Make sure that they thoroughly explore the software, and code both for and while loops.
3 Finding your Zero

Since solving equations of the type \( f(x) = a \) is equivalent to finding the roots of the function \( g(x) = f(x) - a \), root finding quickly becomes an important field of study. The next few activities will introduce you to some methods by which such roots can be approximated.

3.1 Project: Slow and steady wins the race

Problem 3. Suppose you know that \( f : \mathbb{R} \to \mathbb{R} \) is a function such that \( f(-1) = 2 \) and \( f(1) = -4 \).

1. Does this information tell you anything about the roots of \( f \)? Carefully discuss and justify your claims. You may want to recall some theorems from Calculus.

2. Assuming that the function meets any requirements that were found above, what could you conclude if you knew that \( f(0) = 1 \).

3. Consider the function \( f(x) = x^3 + x - 1 \) on the interval \( I = [0, 1] \). Compute \( f(0) \) and \( f(1) \). Show that \( f \) has a root in \( I \).

4. Compute \( f\left(\frac{1}{2}\right) \). On which interval of length \( \frac{1}{2} \) does \( f \) have a root?

5. Find an interval of length \( \frac{1}{4} \) on which \( f \) has a root.


Polynomials are a common class of very well behaved functions. Many tricks in algebra are focused on finding their roots. Unfortunately, it is a deep theorem of abstract algebra that no formula for roots exists for polynomials of degree 5 or higher. This is where numerical methods come in.

Problem 4. Let \( f(x) = x^7 - 3x^2 + x + \frac{1}{2} \). After graphing your function, find each root to within a tolerance of \( 10^{-4} \). Your solutions should be fully explained and include your coding of your earlier algorithm. Check visually that your root is approximately correct.

Problem 5. Suppose you were to use this algorithm on a continuous function With a root in the interval \([1, 3]\). How many loops would your algorithm require before finding a root to within \( 10^{-5} \) precision?

Suppose you were to use this algorithm on a continuous function With a root in the interval \([a, b]\). How many loops would your algorithm require before finding a root to within \( \varepsilon \) precision?

Teaching Notes: A typical algorithm for the bisection method from my students did not take precautions against which end point had the positive image and which had the negative. Students realized their code was not good enough when they tried to run their code for Problem 4 on Problem 5.
3.2 Finding your Zen: the fixed point method

Definition 6. Suppose that \( f \) is a function. A number \( a \) is called a \textbf{fixed point} for \( f \) whenever \( f(a) = a \).

Problem 7. 1. Give an example of a function with a fixed point.
2. Give an example of a function without a fixed point.
3. Suppose that \( f \) has a fixed point at \( a \). Define a function \( g \) such that \( g(a) = 0 \). You cannot use \( a \) in your definition of \( g \), and \( g \) must be non-zero.
4. Suppose \( f(a) = 0 \). Define a function \( g \) such that \( a \) is a fixed point of \( g \). You cannot use \( a \) in your definition of \( g \), and \( g(x) \neq x \).

Finding roots and finding fixed points are equivalent. In some circumstances, finding fixed points can be much faster than finding roots.

Problem 8. Let \( f(x) = (\sin(x) + \cos(x))/2 \).
   
   • Use a graph to make a reasonable guess at to where a fixed point \( a_0 \) might be.
   • Write the first few terms of the sequence \( \{a_n\}_{n=0}^{\infty} \) where \( a_i = f(a_{i-1}) \) for each \( i \geq 1 \).
   • Report your preliminary findings. Make a conjecture as to how to find fixed points.
   • Let \( g(x) = \sin(x)/x \). Can you find a fixed point for \( g \) starting from the point \( a = 0.4 \)? What if \( a = 0.1 \)? What if \( a = 1.3 \)? What if \( a = \pi \)? What impact does this have on your conjecture?
   • Write and code a fixed point finding algorithm that has appropriate fail safes.

Problem 9. 1. Let \( f(x) = e^{e^{-x}} \). Use your fixed point finding algorithm to try to find a fixed point. Explain how the starting point affected your algorithm outcomes.
2. Let \( g(x) = e^{-x} - 3 \ln(x) \). Use a fixed point method to find a root for \( g(x) \).

When the algorithm works, it works very well. It would be quite useful to know that the algorithm will converge before running our program. The following theorem gives us a sufficient criteria for convergence:

Theorem 10 (Fixed-Point Theorem). Suppose that \( g : [a, b] \to [a, b] \) is continuous on \( [a, b] \). Furthermore, if \( g' \) exists on \( (a, b) \) and for all \( x \in (a, b) \), \( |g'(x)| \leq k \) for some \( k < 1 \), then

1. the sequence \( \{a_n\}_{n=0}^{\infty} \) where \( a_i = f(a_{i-1}) \) converges to a unique fixed point \( p \) for any \( a_0 \in [a, b] \).
2. for each \( n \geq 1 \) we have the following bounds on the error:

\[
|a_n - p| \leq k^n \max\{a_0 - a, b - a_0\}
\]

and

\[
|a_n - p| \leq \frac{k^n}{1 - k} |a_1 - a_0|.
\]
Teaching Notes: I did not go through the proof of this result in my class, but such a mini-lecture is completely appropriate at this point in the class.
### 3.3 A blast from the past

Recall the following theorem from Calculus:

**Theorem 11 (Taylor’s Theorem).** Suppose \( f : \mathbb{R} \to \mathbb{R} \) is a function that admits \( n \) continuous derivatives on the interval \([a, b]\) and that \( f^{(n+1)} \) exists on \([a, b]\). Let \( c \in [a, b] \). Then for each \( x \in [a, b] \) there is a number \( z_x \) between \( c \) and \( x \) such that

\[
 f(x) = f(c) + f'(c)(x - c) + \frac{f''(c)}{2!}(x - c)^2 + \cdots + \frac{f^{(n)}(c)}{n!}(x - c)^n + \frac{f^{(n+1)}(z_x)}{(n+1)!}(x - c)^{n+1}
\]

**Problem 12.**

1. Let \( f(x) := e^x + 2^{-x} + 2 \cos(x) - 6 \). Show that \( f \) satisfies the statement of Taylor’s theorem for \( n = 1 \), and write the conclusion of Taylor’s theorem for the interval \([1, 2]\) where \( c \) is a good guess to the value of a root of \( f \).

2. Suppose that \( x \) is the actual root of \( f \). If \( c \) is a good guess to the value of \( x \), then \( (x - c)^2 \) is much smaller than \( (x - c) \). We will drop the last term of the polynomial to get the following expressions: \( 0 \approx f(c) + (x - c)f'(c) \). Use this to approximate the root.

3. Is this new approximation better than your first guess which was \( c \)? What possible concerns could make this process fail? How would you safeguard against them?

4. Iterate this process to generate an algorithm for approximating roots. Use this algorithm to find a root of \( f \) up to a tolerance of \( 10^{-5} \).

5. Code an algorithm based on this fixed point method that finds the roots of a function and includes all the appropriate fail-safes.

The method found in Problem 11 is called Newton’s method. It depends heavily on having a good enough initial approximation. The method will either converge quickly, or fail spectacularly.

We now have three algorithms to find the roots of a function: the bisection method, the fixed point iteration, and Newton’s method.

**Problem 13.** For the following functions, try each algorithm to find all roots in the given interval. Compare and contrast how each algorithm handles the search. If an algorithm cannot be used, explain why not. All numerical approximations should be within a tolerance of \( 10^{-6} \).

1. \( f(x) = x \cos(2x) - (x - 1)^2 \), for \( 0 \leq x \leq 1 \).

2. \( g(x) = x^2 - 4x + 2 - \ln(x) \), for \( 0 \leq x \leq 4 \).

3. \( h(x) = 2x^2 - e^x \), for \( -1 \leq x \leq 4 \).

4. \( k(x) = x + 2 \cos(x) - e^x \), for \( -1 \leq x \leq 1 \).
**Teaching Notes:** This last problem is the large one. The students need to provide a thorough explanation: the presentation is paramount at this point. Make sure the students realize that while some algorithms find roots, others find fixed points for a related function. Fixed point iteration fails on most (all?) of the above functions. You may want to supply a function for which fixed point iterations actually succeed.
4 Calculus my calculator can’t do: Numerical Integration

4.1 Get your trapeze on

Recall that a trapezoid (trapezium or trapeze in other parts of the world) is a quadrilateral with two parallel sides.

Problem 14. Suppose that a trapezoid has parallel sides labeled by \( b \) and \( B \), and that the distance between these sides is given by \( h \) (called the height). Prove that the area of the trapezoid is given by

\[
\frac{h}{2} (B + b).
\]

Problem 15. Let \( f(x) := e^{x^2} \).

- Use the area of a single trapezoid to estimate the area below \( f \) over the interval \( I = [0, 1] \).
- Split the interval \( I \) into two equal pieces, and use the area of the two trapezoids to estimate the area below \( f \) over the interval \( I \).
- Repeat this process by using the area of 4 trapezoids to estimate the area below \( f \) over the interval \( I \).
- Generalize your findings by stating a formula for estimating \( \int_0^1 f(x) \, dx \) with \( n \) many trapezoids.
- Use your formula to estimate \( \int_0^1 f(x) \, dx \) with 73 trapezoids.

Often, using straight lines to approximate curves gives a larger error than using quadratics.

Problem 16. Let \( g(x) := \frac{x}{\ln(x)} \). Let \( I \) be the interval \([e, e+2]\). Define the following polynomial on \( I \):

\[
p(x) := \frac{(x-(e+1))(x-(e+2))}{-2} g(e) + \frac{(x-e)(x-(e+2))}{-1} g(e+1) + \frac{(x-e)(x-(e+1))}{2} g(e+2)
\]

1. Compute the values of \( p(e) \), \( p(e+1) \), \( p(e+2) \) and plot \( g(x) \) and \( p(x) \) on the same graph.

2. Find an approximation to \( \int_e^{e+2} g(x) \, dx \) by integrating \( p(x) \).

You have just used a polynomial of degree 2 to approximate a function you were asked to integrate. Such polynomials were first used by Lagrange to fit through certain points.
4.2 Threading a function through points

Definition 17. Suppose you are given \( n+1 \) many points \((x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\). Then the \( n \)th Lagrange interpolating polynomial \( p_n(x) \) is given by

\[
p_n(x) := y_0 L_{n,0}(x) + \cdots + y_{n+1} L_{n,n}(x) = \sum_{k=0}^{n} y_k L_{n,k}(x)
\]

where for each \( k = 0, 1, \ldots, n \) we have

\[
L_{n,k} = \frac{(x - x_0) \cdot (x - x_1) \cdot \ldots \cdot (x - x_{k-1}) \cdot (x - x_{k+1}) \cdot \ldots \cdot (x - x_n)}{(x_k - x_0) \cdot (x_k - x_1) \cdot \ldots \cdot (x_k - x_{k-1}) \cdot (x_k - x_{k+1}) \cdot \ldots \cdot (x_k - x_n)}
= \prod_{i=0,1,\ldots,n \text{ and } i \neq k} \frac{x-x_i}{x_k-x_i}
\]

Problem 18. Let \( g(x) := \frac{x}{\ln(x)} \).

1. Compute \( L_{1,0}(x) \) and \( L_{1,1}(x) \) for the points \((e, g(e))\) and \((e + 2, g(e + 2))\). Use these to write the first Langrange interpolating polynomial \( p_1(x) \).

2. Find the second Lagrange interpolating polynomial for the points \((e, g(e)), (e + 1, g(e + 1))\) and \((e + 2, g(e + 2))\) by first computing \( L_{2,0}(x), L_{2,1}(x) \) and \( L_{2,2}(x) \).

3. Divide the interval \([e, e + 2]\) into four equal length intervals, and find a degree 4 interpolating polynomial that threads through the \( g \)-images of the endpoints of these intervals.

4. Use this degree 4 polynomial to approximate \( \int_e^{e + 2} g(x) \, dx \).

Problem 19. Write a code that accepts an interval \([a, b]\), a number \( n + 1 \) of points \((x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\) inside this interval, and that will generate the \( n \)th Lagrange interpolating polynomial \( p_n(x) \). Verify that this code works by running it on the givens of the previous problem.

Teaching Notes: Since we are about to embark on solving initial value problems, we need a tool to interpolate points that generate an approximate solution. The code generated in Problem 19 will be used throughout the next section. Other interpolating techniques could be added at this point, but you may not get to the section on Linear Algebra if you dwell on interpolation too long.
5 Initial Value Problems and Numerical Solutions

Many problems in the real world deal with finding the solution to a differential equation which involves some initial condition. Even well posed problems that have a unique solution may not be easy to find. This chapter looks at numerical approximation to solutions to such problems.

5.1 Euler, the master of us all

The first method we will explore does not generate a solution per se. It generates a mesh of points that the approximate solution hits. Once we have these points, we can use our interpolating polynomials to get a decent approximate solution.

Problem 20. Let \( y(t) := \ln(x) + 2 \).

1. Show that \( y \) is a solution to the initial value problem

\[
y' = \frac{1}{t^2} - \frac{y}{t}, \quad 1 \leq t \leq 2, \quad y(1) = 2
\]

2. Divide the domain of \( y \) into four equal length pieces. Call the endpoints \( t_0, \ldots, t_4 \) and the length of each piece the step size \( h \).

3. Show that for each \( i = 0, \ldots, 3 \), we have

\[
y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2} y''(z_i)
\]

for some \( z_i \in [t_i, t_{i+1}] \).

4. By dropping the last term in the previous equation, obtain approximations \( w_0, \ldots, w_4 \) for the values of \( y(t_0), y(t_1), \ldots, y(t_4) \).

5. Use an appropriate interpolating polynomial to approximate the solution \( y \) to the initial value problem.

The method describe in the above problem is called Euler’s method.

Problem 21. Write a code that will provide \( n + 1 \) equally spaced approximate values to the solution of the well posed initial value problem

\[
\frac{dy}{dt} = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha.
\]

Problem 22. Find a polynomial \( p(x) \) which is an approximate solution to the initial value problem

\[
y' = \frac{1 + t}{1 + y}, \quad 1 \leq t \leq 2, \quad y(1) = 2, \text{ with step size } h = \frac{1}{2}.
\]

**Teaching Notes:** Since the actual solution is provided in the opening problem, there is the danger that the code generated by the student will merely use the derivation function itself, rather than the more general differential equation. The last problem forces the students to alter their code if this was the case.
5.2 Taylor strikes again

Problem 23. Let \( y(t) := (t + 1)^2 - \frac{e^t}{2} \).

1. Show that \( y \) is a solution to the initial value problem

\[
y' = y - t^2 + 1, \ 0 \leq t \leq 2, \ y(0) = \frac{1}{2}
\]

2. By differentiating both sides of the above differential equation, express \( y' \), \( y'' \) and \( y''' \) in terms of \( y \).

3. Divide the domain of \( y \) into four equal length pieces. Call the endpoints \( t_0, \ldots, t_4 \) and the length of each piece the step size \( h \).

4. Show that for each \( i = 0, \ldots, 3 \), we have

\[
y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2} y''(t_i) + \frac{h^3}{3!} y'''(t_i) + \frac{h^4}{4!} y^{(4)}(z_i)
\]

for some \( z_i \in [t_i, t_{i+1}] \).

5. By dropping the last term in the previous equation, and using the relation between \( y \) and its derivatives found in the first part of this problem, obtain approximations \( w_0, \ldots, w_4 \) for the values of \( y(t_0), y(t_1), \ldots, y(t_4) \).

6. Use an appropriate interpolating polynomial to approximate the solution \( y \) to the initial value problem.

7. Compare the accuracy of this approximation with the results from Euler’s method.

The method described in the above problem is called Taylor’s method of order 4.

Problem 24. Write a pseudo-code that will provide \( k + 1 \) equally spaced approximate values to the solution of the well posed initial value problem

\[
\frac{dy}{dt} = f(t, y), \ a \leq t \leq b, \ y(a) = \alpha.
\]

by using Taylor’s method of order \( n \).

Problem 25. Use Taylor’s method of order 3 to find an approximating polynomial \( p(x) \) to the solution to the differential equations

1. \( y' = \sin(t) + e^{-t}, \ 0 \leq t \leq 1 \) where your step size \( h = .25 \) and \( y(0) = 0 \).

2. \( y' = \frac{1+t}{1+y}, \ 1 \leq t \leq 2, \ y(1) = 2, \) with step size \( h = \frac{1}{4} \).
5.3 Bigger, Faster, Stronger: the Runge-Kutta methods

Even though the previous methods work very well, they are seldom used. The main drawback comes in having to compute derivatives for \( f(t, y) \). Carle Runge and Martin Wilhelm Kutta developed an array of algorithms that sidestep this issue. These are called Runge-Kutta methods.

**Algorithm 26** (Midpoint Method).

\[
\begin{align*}
    w_0 &= \alpha \\
    w_{i+1} &= w_i + hf(t_i + \frac{h}{2}, w_i + \frac{h}{2}f(t_i, w_i)) \quad \text{for each } i = 0, 1, \ldots N - 1.
\end{align*}
\]

**Algorithm 27** (Modified Euler’s Method).

\[
\begin{align*}
    w_0 &= \alpha \\
    w_{i+1} &= w_i + \frac{h}{2} \left[ f(t_i, w_i) + f(t_{i+1}, w_i + hf(t_i, w_i)) \right] \quad \text{for each } i = 0, 1, \ldots N - 1.
\end{align*}
\]

**Algorithm 28** (Heun’s Method).

\[
\begin{align*}
    w_0 &= \alpha \\
    w_{i+1} &= w_i + \frac{h}{2} \left[ f(t_i, w_i) + 3f \left( t_i + \frac{2}{3}h, w_i + \frac{2}{3}hf(t_i, w_i) \right) \right] \quad \text{for each } i = 0, 1, \ldots N - 1.
\end{align*}
\]

All three above algorithms are classified as two-step methods. The bound on the truncation errors is of the same order as the order-2 Taylor method from the previous section.

**Problem 29.** Write codes that implement these three algorithms.

**Problem 30.** Consider the following differential equation:

\[ y' = \frac{y^2}{1 + t^2}, \quad 1 \leq t \leq 2, \quad y(1) = \frac{-1}{\ln 2}, \quad \text{with } h = 0.1 \]

1. Generate points \( w_0, w_1, \ldots, w_{10} \) that approximate a solution to the equation by using the Midpoint method.

2. Generate points \( w_0, w_1, \ldots, w_{10} \) that approximate a solution to the equation by using the Modified Euler method.

3. Generate points \( w_0, w_1, \ldots, w_{10} \) that approximate a solution to the equation by using Heun’s method.

4. The actual solution to this differential equation is \( y(t) = \frac{-1}{\ln(t+1)} \). Create a table that shows the errors for each method at each point.

5. Obtain and graph the three interpolating polynomials.

**Teaching Notes:** You could also have the students compare these solutions to what Euler’s method would produce. If your class has enough time left in the term, you could introduce 4-step Runge-Kutta methods at this point before moving on to the next section.
6 Linear Algebra in the Computer Age

We turn our attention to a different branch of mathematics: solving systems of linear equations.

6.1 Maple Linear Algebra Package worksheet

Teaching Notes: When using Maple to do linear algebra, include the command

```
with(LinearAlgebra);
```

before attempting specific command.

Try your hand at the following:

- Create a blank (all entries are zero) 4 × 4 matrix, and save it as $A$.
- Store the value 3 into $A[1, 3]$.
- Store successive increasing integers into the diagonal entries of $A$.
- Capture the values of the diagonal entries of $A$ into a vector $b$.
- Create a matrix $C$ that is 4 × 4 all with successive entries from the Fibonacci sequence.
- Replace the diagonal entries of $C$ with the diagonal entries of $A$.

6.2 Why can't we just find the right answer?

Consider the following system of equations:

\[
\begin{align*}
-x_1 & - 2x_2 - x_3 - 4x_4 + 3x_5 = 1 \\
-x_1 & - 3x_2 - x_3 + 2x_4 = 0 \\
-2x_1 & + 5x_2 - 2x_3 + x_4 + 2x_5 = -1 \\
3x_1 & + 2x_2 - x_3 & - x_5 = 1 \\
4x_1 & & - 4x_3 + 2x_4 + x_5 = 2
\end{align*}
\]

Problem 31.  
- Express the previous system of equations in the form $Ax = b$, where $A$ is a 5 × 5 matrix, $x, b \in \mathbb{R}^5$.

- Recall what techniques you know to solve such a system. What are some reasons you might not want to use these techniques here?

- Solve the first equation for the variable $x_1$, the second equation for the variable $x_2$, and so forth.

- Rewrite the system of equation in the form $x = Tx + c$, where $T$ is a 5 × 5 matrix, $x, c \in \mathbb{R}^5$. What is the relationship between $A$ and $T$?
• Code a procedure that will accept an \( n \times n \) matrix \( A \), an \( n \)-dimensional vector \( b \), and return an \( n \times n \) matrix \( T \) and an \( n \)-dimensional vector \( c \) such that the systems \( Ax = b \) and \( x = Tx + c \) are equivalent.

• Starting with the vector \( w_0 := (0,0,0,0,0) \), compute \( w_1 := Tw_0 + c \). By following the iteration \( w_{i+1} = Tw_i + c \), compute \( w_1, w_2, w_3 \) and \( w_4 \).

• What do these findings suggest?
One way to say that the iteration was successful is stop once relative change in each coordinate is below a pre-prescribed tolerance.

1. For each coordinate of \( w_3 \) and \( w_4 \), compute the absolute difference for the two vectors in that coordinate.

2. If \( x := (x_1, x_2, \ldots x_n) \) is a vector, define \( \|x\|_\infty := \max\{|x_i| : 0 \leq i \leq n\} \). Compute

\[
\frac{\|w_4 - w_3\|_\infty}{\|w_4\|_\infty}
\]

3. Write a code that iterates a search for a solution to the equation \( Ax = b \) and stops at a tolerance of \( 10^{-5} \). Make sure your code can switch the form of \( Ax + b \) into the new form \( x = Tx + c \).

4. There was nothing special about using the vector \( (0,0,0,0,0) \) as the initial guess. Run your code with different initial guesses, and report your findings on the number of iterations it took your code to stop.

The technique explored in the above problem is called the Jacobi iterative method for finding solutions to systems of equations.

**Problem 32.** Solve the following system of equations using Jacobi’s method:

\[
\begin{align*}
x_1 + 2x_2 &= 1 \\
-x_1 + 2x_2 + 3x_3 &= 1 \\
x_2 + 3x_3 - 2x_4 &= 1 \\
2x_3 - x_5 &= 1 \\
x_4 + 2x_5 - x_6 &= 1 \\
2x_5 + 3x_6 &= 1
\end{align*}
\]

**Teaching Notes:** Since Maple had built-in matrix multiplication, I did not have my students code it at this time. I left it for the next section where they coded the Gauss-Seidel algorithm. If you want to have the students code the multiplication by hand for this algorithm, just move the next problem up a section.
6.3 Once again, with feeling...

For our next technique, we need to delve into the mechanics of matrix multiplication.

Let

\[
A := \begin{bmatrix}
1 & 1 & 2 \\
-1 & 3 & 2 \\
2 & -1 & 0
\end{bmatrix}, \quad B := \begin{bmatrix}
-1 & 1 \\
-1 & 5 \\
7 & 3
\end{bmatrix}.
\]

Problem 33. Recall the mechanics of matrix multiplications.

1. By hand, carry out the multiplication \(A \cdot B\).

2. Write an algorithm that carries out this multiplication one entry at a time.

3. Modify your algorithm to accept an \(m \times n\)-matrix \(A\) and an \(n \times l\) matrix \(B\), and returns the product of the two as an \(m \times l\) matrix \(C\).

Consider once again the system

\[
\begin{align*}
-x_1 - 2x_2 - x_3 - 4x_4 + 3x_5 &= 1 \\
-x_1 - 3x_2 - x_3 + 2x_4 &= 0 \\
-2x_1 + 5x_2 - 2x_3 + x_4 + 2x_5 &= -1 \\
3x_1 + 2x_2 - x_3 - x_5 &= 1 \\
4x_1 - 4x_3 + 2x_4 + x_5 &= 2
\end{align*}
\]

Problem 34. The Gauss Seidel iteration method is nothing more than a minor change to the code for the Jacobi iteration method.

- After having written the new system \(x = Tx + c\), starting from the vector \(w_0 := (0, 0, 0, 0, 0)\), compute \(w_1 := Tw_0 + c\) using your new matrix multiplication algorithm.

- Notice that the coordinates of \(w_1\) are computed one at a time. Suppose that you replaced the first coordinate of \(w_0\) with the newly computed first coordinate of \(w_1\) prior to computing the second coordinate of \(w_1\). Which second coordinate is closer to the actual value of the second coordinate of the solution vector \(x\)?

- By replacing the coordinates in \(w_0\) by the newly computed coordinates, Gauss and Seidel noticed the convergence of the iteration was accelerated. Modify your Jacobi iteration code to take this this replacement into account.

- Compare the convergence rate of your new algorithm with that used by Jacobi.

Problem 35. Solve the following system of equations using the Gauss-Seidel algorithm:

\[
\begin{align*}
x_1 + 2x_2 &= 1 \\
-x_1 + 2x_2 + 3x_3 &= 1 \\
x_2 + 3x_3 - 2x_4 &= 1 \\
2x_3 - x_5 &= 1 \\
x_4 + 2x_5 - x_6 &= 1 \\
2x_5 + 3x_6 &= 1
\end{align*}
\]

Compare your result to what you found using Jacobi’s method.