Math 448 Final Project

Hunter Mitchell

May 6, 2020

1) Consider the partial differential (diffusion) equation

$$\partial_t u = D\partial_x^2 \tag{1}$$

for $x \in [0,1]$, and $t > 0$. The initial conditions are given by

$$u(x,0) = \begin{cases} x \text{ if } 0 \leq x \leq 1/2 \\ 1-x \text{ if } 1/2 < x \leq 1 \end{cases} \tag{2}$$

The boundary conditions are

$$u(0,t) = u(1,t) = 0 \tag{3}$$

Solve the problem (1)-(3) with $D = 1/2$ numerically using the explicit difference scheme

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = D\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2} \tag{4}$$

Take $\Delta x = 0.1$ and compute the numerical solutions for three values of $\Delta t : \Delta t = 1/50, 1/100, 1/200$.

(i) Check numerically the order of accuracy in time by plotting the error(the difference between the exact and numerical solutions for the above values of Δt.

(ii) Plot the exact and numerical solutions for the above values of Δt. (You should have one plot for each Δt showing the exact and numerical solutions on the same set of axis).

(iii) Comment on the results of the computations.

The exact solution of (1)-(3) can be obtained by separation of variables. The description of the method is available in many introductory PDE textbooks. For example, you can consult "Advanced Engineering Mathematics" by Erwin Kreyszig, Ch. 11. The exact solution $v(x,t)$ has the series representation

$$v(x,t) = \sum_{k=1}^{\infty} \frac{4}{(k\pi)^2} \sin(k\pi/2)\sin(k\pi x)e^{-D(k\pi)^2 t} \tag{5}$$

Solution: I began by reading section 9.1 of the book to better understand what the problem was asking. There was a corresponding algorithm that I then implemented in MATLAB. I had to add a 2 in the denominator of s to account for the $D = 1/2$ part. I also had to create a separate g function to calculate the initial conditions from Equation (2). The first and last values of the v vector are normally $a(t)$ and $b(t)$ but fortunately we know from Equation (3) that these both equal zero.

Now that we have a function to calculate the numerical solution, we need to create one for the analytical solution. For this, I just created a function that resembles Equation (5) and a function that loops through that to put it into a vector. Now we just have to make a main script that calculates the solutions for the different Δt values and compares them. I decided to use 10 steps in the t direction ($M = 10$) for my calculations. Here is my main code:

```
% Hunter Mitchell - Math 448 Final Question 1
% 5/4/20

M = 10; % number of steps in t-direction

% Calculate exact and numerical solutions
exactVector1 = calcExactVector(0.02,M);
exactVector2 = calcExactVector(0.01,M);
exactVector3 = calcExactVector(0.005,M);
numVector1 = calcNumericalSolution(9,0.02,M);
numVector2 = calcNumericalSolution(9,0.01,M);
numVector3 = calcNumericalSolution(9,0.005,M);


% Calculate the error vectors
errorVector1 =  abs(numVector1 - exactVector1)
errorVector2 = abs(numVector2 - exactVector2)
errorVector3 = abs(numVector3 - exactVector3)


% Plot the error vectors (part i)
figure(1)
plot(errorVector1,'r')
legend('delta t = 0.02')
figure(2)
plot(errorVector2,'b')
legend('delta t = 0.01')
figure(3)
plot(errorVector3,'g')
legend('delta t = 0.005')


% Plot the numerical vectors vs exact vectors (part ii)
figure(4)
plot(exactVector1,'r',numVector1,'b')
legend('Exact Solution','Numerical Solution')
figure(5)
plot(exactVector2,'r',numVector2,'b')
legend('Exact Solution','Numerical Solution')
figure(6)
plot(exactVector3,'r',numVector3,'b')
legend('Exact Solution','Numerical Solution')

```

Figure 1: Main Code

Here are my different functions:

```
% Calculates each exact solution and puts into vector
function vector = calcExactVector(k,M)
  x = 0;
  dx = 0.1;
  for i=1:11
    u(i) = calcExactSolution(x,k*M);
    x = x+dx;
  end
  vector = u;
end
```

Figure 2: Exact Vector Function

```
% Exact solution function
function totalSum = calcExactSolution(x,t)

  currentSum = 0;
  for k = 1:14
    temp = (4/(k*k*pi*pi))*sin(k*pi/2)*sin(k*pi*x)*e^(-0.5*k*k*pi*pi*t);
    currentSum += temp;
  end
  totalSum = currentSum;

end
```

Figure 3: Exact Solution Function

```
% Numerical solution function - based on algorithm from page 618
function result = calcNumericalSolution(n,k,M)

 h = 1/(n+1);
 s = k/(2*h*h); % must add 2 in denominator to account for D

 for i = 0:n+1
   w(i+1) = gfunction(i*h);
 end

 t = 0;

 %w

 for j = 1:M
   v(1) = 0;  % a(t) = 0
   v(n+2) = 0;   % b(t) = 0
   for i = 2:n+1
     v(i) = s*w(i-1)+(1-2*s)*w(i)+s*w(i+1); % update step
   end
   t = j*k;

   %j
   %t
   %v

   w=v;

 end

 result = w;
end
```

Figure 4: Numerical Solution Function

```
% Initial conditions from equation 2
function value = gfunction(x)
  if (x >= 0) && (x <= 0.5)
    value = x;
  elseif (x > 0.5) && (x <= 1)
    value = 1-x;
  end
end
```

Figure 5: G Function

i)These are the three error vector outputs. The first is for $\Delta t = 1/50$, the second is for $\Delta t = 1/100$, and the third is for $\Delta t = 1/200$.

```
>> Math448Final1
errorVector1 =
 Columns 1 through 8:
     0.00000   110.05333   226.88878   342.17780   430.74366   464.34894   430.74366   342.17780
 Columns 9 through 11:
   226.88878   110.05333     0.00000
errorVector2 =
 Columns 1 through 8:
   0.0000000   0.0020954   0.0033270   0.0050707   0.0061353   0.0059502   0.0061353   0.0050707
 Columns 9 through 11:
   0.0033270   0.0020954   0.0000000
errorVector3 =
 Columns 1 through 8:
   0.0000000   0.0007441   0.0014061   0.0018875   0.0021427   0.0022146   0.0021427   0.0018875
 Columns 9 through 11:
   0.0014061   0.0007441   0.0000000
```

Figure 6: Error Output Vectors

We can then plot each of these:

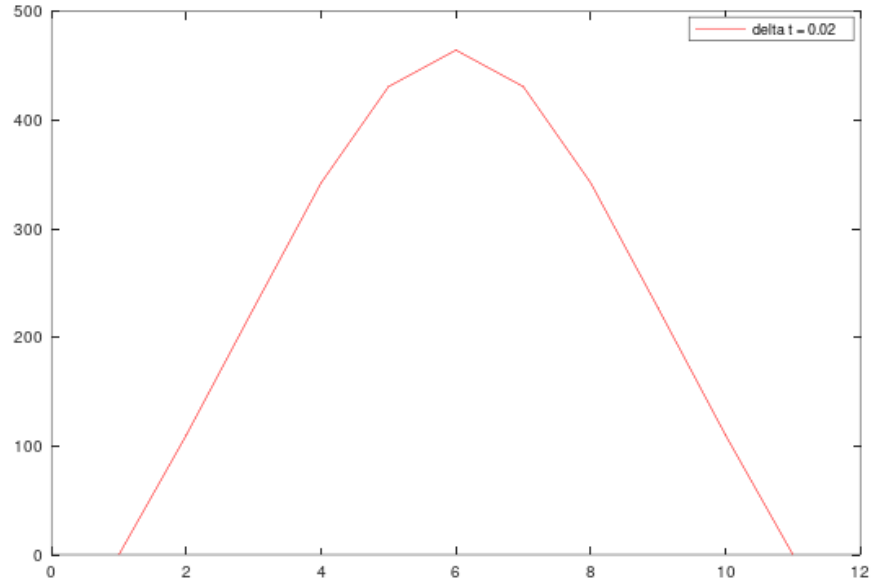

Figure 7: Error Vector for $\Delta t = 1/50$

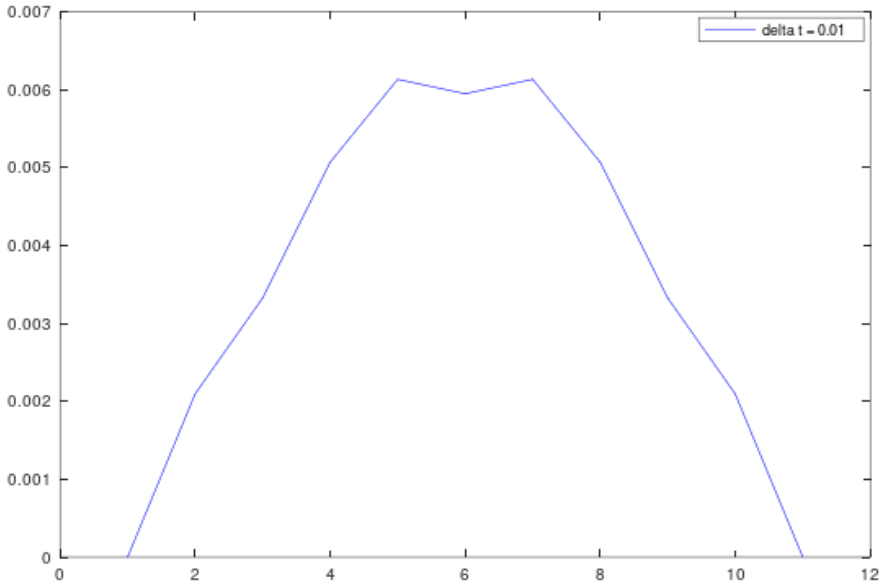

Figure 8: Error Vector for $\Delta t = 1/100$

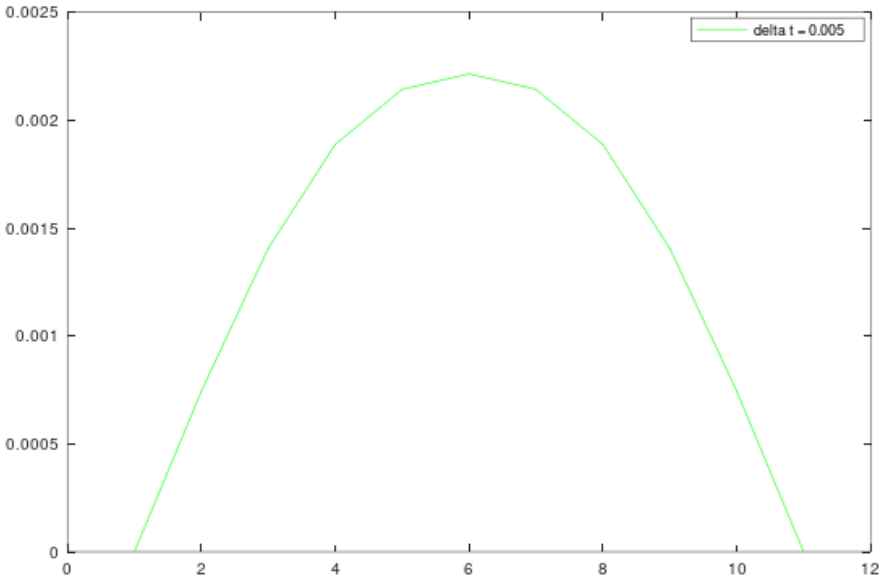

Figure 9: Error Vector for $\Delta t = 1/200$

ii) These are the plots of the numerical solutions vs the analytical solutions:

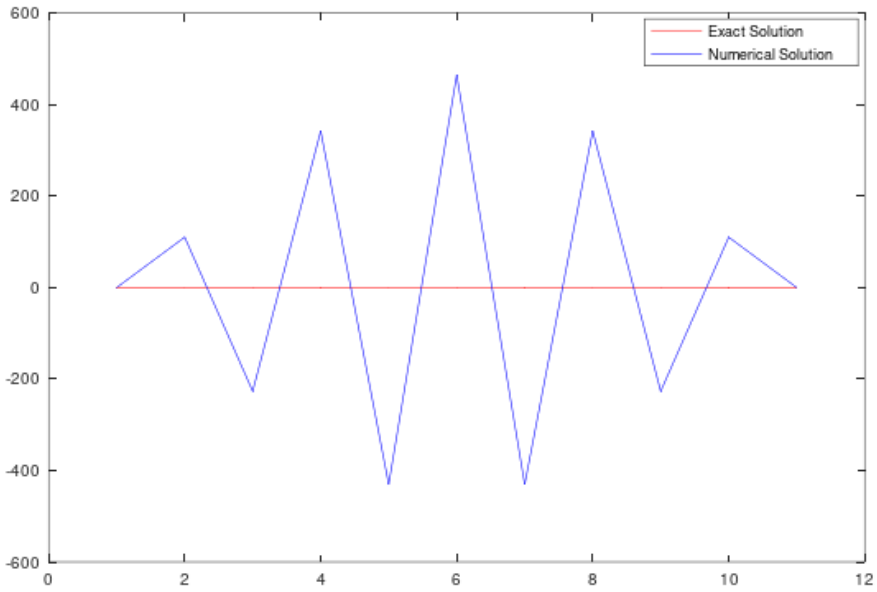

Figure 10: Numerical vs Analytical for $\Delta t = 1/50$

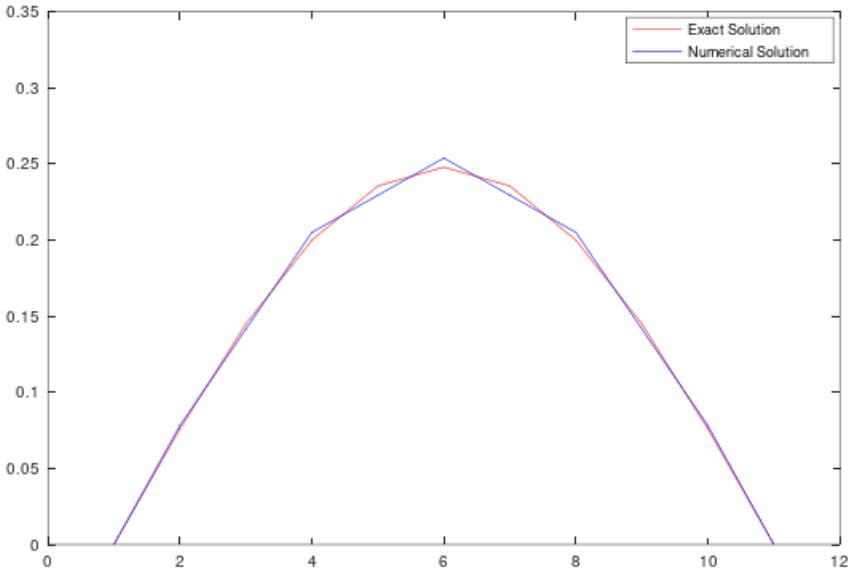

Figure 11: Numerical vs Analytical for $\Delta t = 1/100$

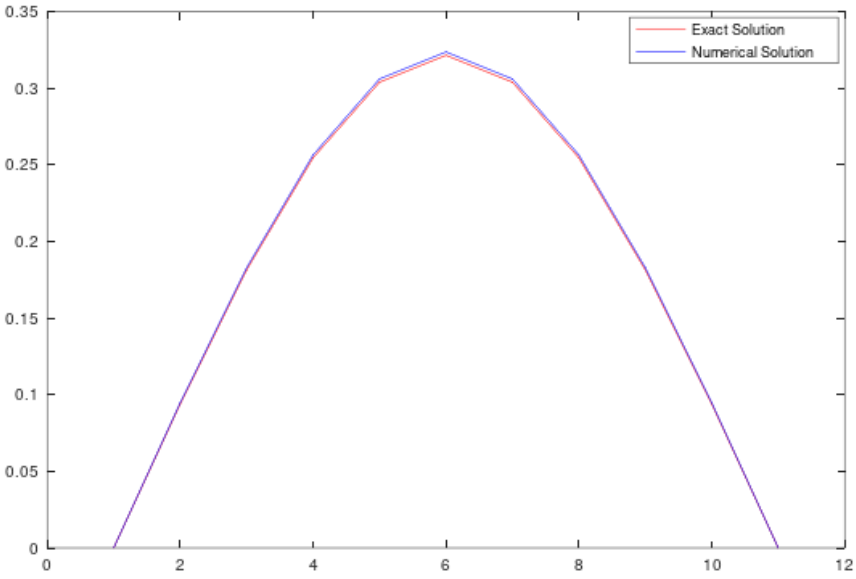

Figure 12: Numerical vs Analytical for $\Delta t = 1/200$

iii) Clearly, the best estimate for the exact solution uses $\Delta t = 1/200$. For this case, the error vector remains below 0.0025 the entire time and almost perfectly matches the analytical solution in Figure 12. The case of $\Delta t = 1/100$ also does pretty well. It matches the analytical solution with an error below 0.007 for each value. The numerical case of $\Delta t = 1/50$ is very off from the analytical solution. I read in the book that not every combination of step sizes works, so this must be an example of when it diverges. We also can see that each solution is more accurate towards the boundaries where $t = 0$ and $t = 1$. This is because we know these values, so our numerical solution will match better towards these points.

2) Solve Computer problem 1 in Section 8.3. Write you own program from scratch. Then compare with the exact analytic solution that can be obtained by the method of integrating factors studied in Math 315. The exact solution is

$$x(t) = 12\frac{e^t}{(e^t+1)^2}$$

Problem: Write a computer program to solve an initial-value problem $x' = f(t, x)$ with $x(t_0) = x_0$ on an interval $t_0 \leq t \leq t_m$ or $t_m \leq t \leq t_0$. Use the fourth order Runge-Kutta method. Test it on this example:

$$\begin{cases} (e^t+1)x' + xe^t - x = 0 \\ x(0) = 3 \end{cases}$$

Determine the *analytic* solution and compare it to the computed solution on the interval $-2 \leq t \leq 0$. Use $h = -0.01$.

Solution: I began by reading about the fourth order Runge-Kutta method in this section and noticing that Example 1 is very similar to the problem at hand. I first coded the constants that I knew would be needed, and also defined a t vector. Then I solved for x' from the equation in the problem and made it an anonymous function. The other anonymous function is the exact solution equation that was given. I then implemented the algorithm from Example 1 with a few necessary minor changes. I included a print statement to observe the outputs like the example. Now that the numerical part was done, I just needed to code the analytical part. This is just a simple loop that calculates it for each t value. Lastly, I compared both solutions side by side and plotted them on the same graph. This is the code:

```
% Hunter Mitchell - Math 448 Final Question 2 (8.3 CP 1) page 547
% 5/4/20

% Solving for x' gives our f(x,t)
f=@(x,t)(x-x*exp(t))/(1+exp(t));

% Analytic solution that was given
u=@(x) (12*exp(x))/((exp(x)+1)*(exp(x)+1));

% Define boundaries
a = -2;
b = 0;

% Calculate M from h
h = -0.01;
M = (a-b)/h;

% Create t-vector
t = b:h:a;

% Define starting x-value
x(1) = 3;

% Calculate numerical solution
for i=1:M

    F1 = h*(f(t(i),x(i)));
    F2 = h*(f((t(i)+h/2),(x(i)+F1/2)));
    F3 = h*(f((t(i)+h/2),(x(i)+F2/2)));
    F4 = h*(f((t(i)+h),(x(i)+F3)));

    x(i+1) = x(i) + (F1 + 2*F2 + 2*F3 + F4)*(1/6);
    e = abs(u(t(i)) - x(i+1));

    fprintf(' %20f \t %20f \t %20f \t %20f \n',i,t(i),x(i),e)
end


% Calculate exact solution
for i=1:M+1
    y(i)=u(t(i));
end

```

Figure 13: Code Part 1

```
43
44
45 for j=1:M+1
46     fprintf(' %20f \t %20f \n',x(j) ,y(j))
47 end
48
49
50 % Plot values
51 figure(1)
52 plot(t,x,'r')
53 hold on
54 plot(t,y,'b')
55 legend('Numerical','Analytical')
56 xlabel('t')
57 ylabel('x')
```

Figure 14: Code Part 2

The following output columns show the iteration, t-value, numerical value, and error, respectively.

```
 1.000000            0.000000            3.000000            0.000045
 2.000000           -0.010000            2.999955            0.000106
 3.000000           -0.020000            2.999819            0.000107
 4.000000           -0.030000            2.999593            0.000049
 5.000000           -0.040000            2.999276            0.000068
 6.000000           -0.050000            2.998869            0.000245
 7.000000           -0.060000            2.998371            0.000481
 8.000000           -0.070000            2.997783            0.000776
 9.000000           -0.080000            2.997104            0.001130
10.000000           -0.090000            2.996335            0.001542
11.000000           -0.100000            2.995475            0.002013
12.000000           -0.110000            2.994525            0.002542
13.000000           -0.120000            2.993485            0.003129
14.000000           -0.130000            2.992354            0.003773
15.000000           -0.140000            2.991133            0.004474
16.000000           -0.150000            2.989822            0.005233
17.000000           -0.160000            2.988421            0.006048
18.000000           -0.170000            2.986929            0.006918
19.000000           -0.180000            2.985347            0.007845
20.000000           -0.190000            2.983675            0.008826
21.000000           -0.200000            2.981913            0.009863
22.000000           -0.210000            2.980061            0.010953
23.000000           -0.220000            2.978120            0.012097
24.000000           -0.230000            2.976088            0.013294
25.000000           -0.240000            2.973966            0.014543
26.000000           -0.250000            2.971754            0.015844
27.000000           -0.260000            2.969453            0.017196
28.000000           -0.270000            2.967062            0.018599
29.000000           -0.280000            2.964582            0.020052
30.000000           -0.290000            2.962012            0.021553
31.000000           -0.300000            2.959352            0.023103
```

Figure 15: Iteration, t-value, numerical value, and error output

The following output columns show the numerical value, and the analytical value, respectively.

3.000000	3.000000
2.999955	2.999925
2.999819	2.999700
2.999593	2.999325
2.999276	2.998800
2.998869	2.998126
2.998371	2.997302
2.997783	2.996328
2.997104	2.995205
2.996335	2.993933
2.995475	2.992512
2.994525	2.990943
2.993485	2.989226
2.992354	2.987361
2.991133	2.985348
2.989822	2.983188
2.988421	2.980882
2.986929	2.978429
2.985347	2.975831
2.983675	2.973087
2.981913	2.970199
2.980061	2.967167
2.978120	2.963991
2.976088	2.960672
2.973966	2.957211
2.971754	2.953609
2.969453	2.949866

Figure 16: Numerical value and analytical value output

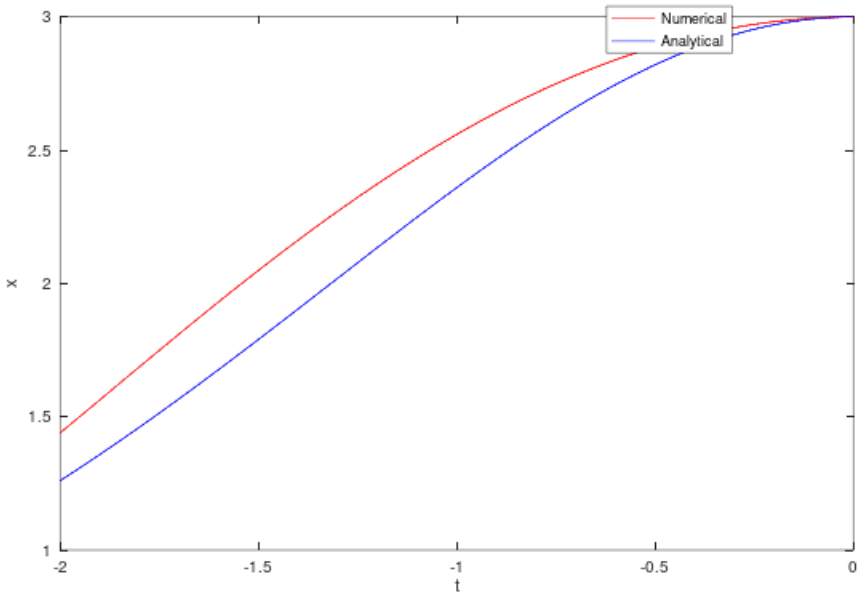

Figure 17: Numerical vs Analytical Plot

As we can see from the plot, the errors between our calculated numerical solution and exact analytical solution vary as we change t. As one would expect, the values are much closer as t approaches zero. This is because we know the value is 3 when $t = 0$. Overall, the Runge-Kutta method does a decent job of finding the solution to this initial value problem.

3) Solve Computer problem 1 in Section 6.13.

Problem: Write and test a fast Fourier transform code to compute the values of

$$p(x) = \sum_{j=0}^{N-1} a_j E_j(x)$$

assuming that $N = 2^m$ and that the coefficients a_j have been prescribed.

Solution: I began by researching examples of discrete Fourier Transforms. I started by implementing the algorithm on page 455. I was very lost at the beginning of this problem, since I thought that we also had to use the g function at the bottom of page 458. After realizing this wasn't necessary and that we could test the our code on any function, I made a lot of progress. The professor said that a triangular wave function would be a good function to test on the code,

so I implemented this as an anonymous function with a period of 2π. I then used MATLAB'S built in Fast Fourier Transform code to know what output I was looking for. I had to make a few small changes to the algorithm in order for this to work. For example, I had to add an input parameter and also I had to change the initialization step a bit so that I got the right starting vector. I also had to add one to a lot of the vector for loops, since Octave does not allow a subscript of 0. I finally got this to work and I'm very happy because it took me a long time. Here is my main code:

```
% Hunter Mitchell - Math 448 Final Question 3 (6.13 CP 1) page 459
% 5/4/20

% n = 7pi/2 will give 8 values
n = (7/2)*pi;


x = 0:pi/2:n


% Triangle wave function
f = @(x) (2/pi)*asin(sin(x));


y = f(x)


% plot triangle wave function
plot(x,y)


% FFT using matlab's builtin function
C1 = fft(y);


m = 3; % m = 3 will mean the length is 2^3 = 8

% FFT using my function
C2 = myfft(m,f);


% Multiply by m again
C2 = C2.*8

C1
C2

% Calculate the difference
E = abs(C1-C2);

E


```

Figure 18: Main Code

And here is my function:

```
% Fast Fourier Transform Algorithm adapted from page 455-456
function C = myfft(m,f)
  N = 2^m;
  w = exp(-2*pi*sqrt(-1)/N);
  for k=0:N-1
    Z(k+1) = w^k;
    C(k+1) = f(4*pi*k/N);
  endfor
  %Z
  %C
  for n=0:m-1
    for k=0:(2^(m-n-1)-1)
      for j=0:((2^n)-1)
        u = C((2^n)*k + j + 1);
        v = Z((j)*(2^(m-n-1))+1)*C((2^n)*k+(2^(m-1))+j+1);
        D(((2^(n+1)))*k + j + 1) = (u+v)/2;
        D(((2^(n+1)))*k + j + 1 + 2^n) = (u-v)/2;
      endfor
    endfor
    %D
    for j=0:N-1
      C(j+1) = D(j+1);
    endfor
  endfor
end
```

Figure 19: Fast Fourier Transform Code

This is a graph of the triangular wave function:

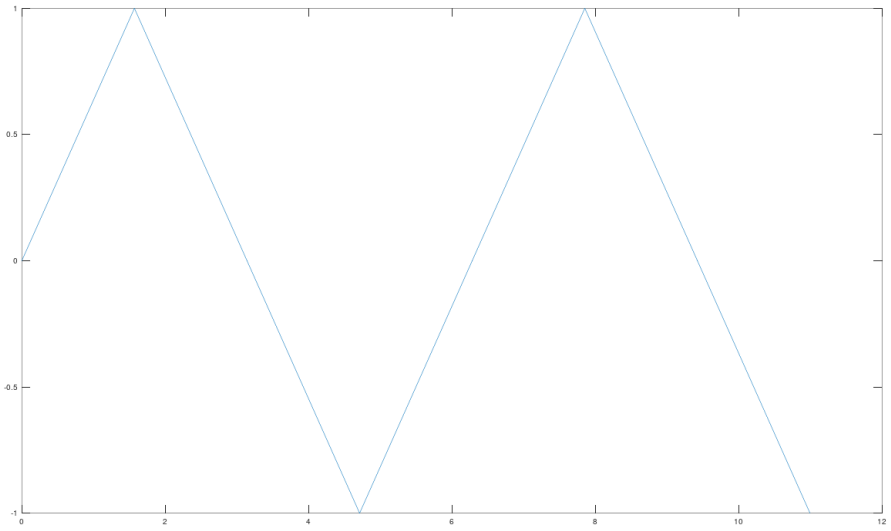

Figure 20: Numerical vs Analytical Plot

And this is the output. $C1$ is the output of the built in Fast Fourier Transform function, $C2$ is the output of my Fast Fourier Transform function, and E is the difference between the two.

```
C1 =

 Columns 1 through 4:

   0.00000 + 0.00000i   0.00000 + 0.00000i  -0.00000 - 4.00000i   0.00000 - 0.00000i

 Columns 5 through 8:

   0.00000 + 0.00000i   0.00000 + 0.00000i  -0.00000 + 4.00000i   0.00000 - 0.00000i

C2 =

 Columns 1 through 4:

   0.00000 + 0.00000i   0.00000 + 0.00000i   0.00000 - 4.00000i   0.00000 - 0.00000i

 Columns 5 through 8:

   0.00000 + 0.00000i   0.00000 + 0.00000i  -0.00000 + 4.00000i   0.00000 - 0.00000i

E =

 Columns 1 through 7:

   0.0000e+00   4.9304e-32   1.1331e-15   4.9304e-32   0.0000e+00   4.9304e-32   1.1331e-15

 Column 8:

   4.9304e-32
```

Figure 21: Output

As we can see, the difference between the computers output and my output is approximately zero. I assume that some of the values in the error vector are not exactly zero because of how MATLAB stores their float values. Either way, this algorithm does a very good job of finding the Fast Fourier Transform of a function, and I am happy that it worked so well!