MTH 452 Final Project Report

Hunter Darling

1. Solve the boundary problem using Finite Difference method:

$$\nabla^2 u + 2u = g, \text{ inside } \Omega = [0,1] \text{ X } [0,1]$$

$$u = 0, \text{ on the boundary of } \Omega$$

Where $g(x, y) = (xy + 1)(xy - x - y) + x^2 + y^2$. The exact solution $u = (xy(x-1)(y-1))/2$. Use the Gauss-Seidel procedure to solve the obtained linear equation.

This boundary value problem utilizes the 2D-Poisson equation and the finite difference discretization. By calculating the second order differences of x and y, we obtain a linear expression to evaluate and approximate solutions are different points. This is incorporated into the MATLAB code by defining g as the approximating function into which each x and y may be inputted to calculate approximations at each “grid point” on the surface.

The first things built in the algorithm are vectors for these x and y values. It is populated with values determined by the step sizes h and k. Then a matrix to hold the approximations at each grid point is built, and initialized to be full of 0s, its dimensions given by the choices of m and n, or, the number of steps. This matrix's interior grid points are then populated according to the finite difference method, and the boundary points by the boundary conditions stipulated. This yields a linear system which may then be solved by the Gauss-Seidel method, in which a linear system, $A\mathbf{x} = \mathbf{b}$, is solved by iterations given by:

$$(D - L)\mathbf{x}^{(k)} = U\mathbf{x}^{(k-1)} + \mathbf{b}$$

Where $\mathbf{b}$ is given by the solution to $A\mathbf{x} = \mathbf{b}$ by some initial approximations $\mathbf{x}^{(0)}$. D is diagonalized, L is lower triangular, and U is upper triangular, split from A. $\mathbf{x}^{(k-1)}$ are the approximated values at the $(k-1)^{th}$ iteration, which are then used to approximate the k^{th}. This can be further simplified into an expression for $\mathbf{x}^{(k)}$:

$$\boldsymbol{x}^{(k)} = T\boldsymbol{x}^{(k-1} + \boldsymbol{c}$$

Where $T = (D - L)^{-1}U$ and $\boldsymbol{c} = (D - L)^{-1}\boldsymbol{b}$.

The MATLAB code is built in direct coordination with the pseudocode found in the textbook for Algorithm 12.1 on page 738. Steps 1-5 define the parameters, and build the vectors and matrices, while steps 7-20 are recursive beneath step 6 and perform the Gauss-Seidel method.

Pseudocode

Poisson finite diff algorithm:

Define endpoints of boundary: a, b, c, d; define step sizes m, n; define TOL; define max iterations N

1. *Compute h, k*
2. *Initialize vector of x values; initialize vector of y values*
3. *Initialize matrix for approximations*
4. *Define lambda, mu, l = 1*
5. *While l <= N*
    a. *Do Gauss-Seidel method until NORM < TOL*
6. *Output approximations*

g_final :

Stipulate boundary conditions, in this case u = 0 all along the boundary

final_1 :

Calls poission2D and plots contours of approximations, exact solutions, and error plot

The following table contains the final approximation matrix values:

Table 1

w =

0	0	0	0	0	0	0	0	0	0	0
0	0.0062	0.0111	0.0146	0.0167	0.0174	0.0167	0.0146	0.0111	0.0062	0
0	0.0100	0.0178	0.0233	0.0267	0.0278	0.0267	0.0233	0.0178	0.0100	0
0	0.0112	0.0200	0.0262	0.0300	0.0312	0.0300	0.0262	0.0200	0.0112	0
0	0.0100	0.0178	0.0233	0.0267	0.0278	0.0267	0.0233	0.0178	0.0100	0
0	0.0062	0.0111	0.0146	0.0167	0.0174	0.0167	0.0146	0.0111	0.0062	0
0	0	0	0	0	0	0	0	0	0	0

The resulting approximations from Table 1 can be then represented by a contour plot (Figure 1), where they can then be compared to the contour plot of the exact solution (Figure 2), given by *u* as defined above. The error in the approximation is modeled in Figure 3.

Figure 1

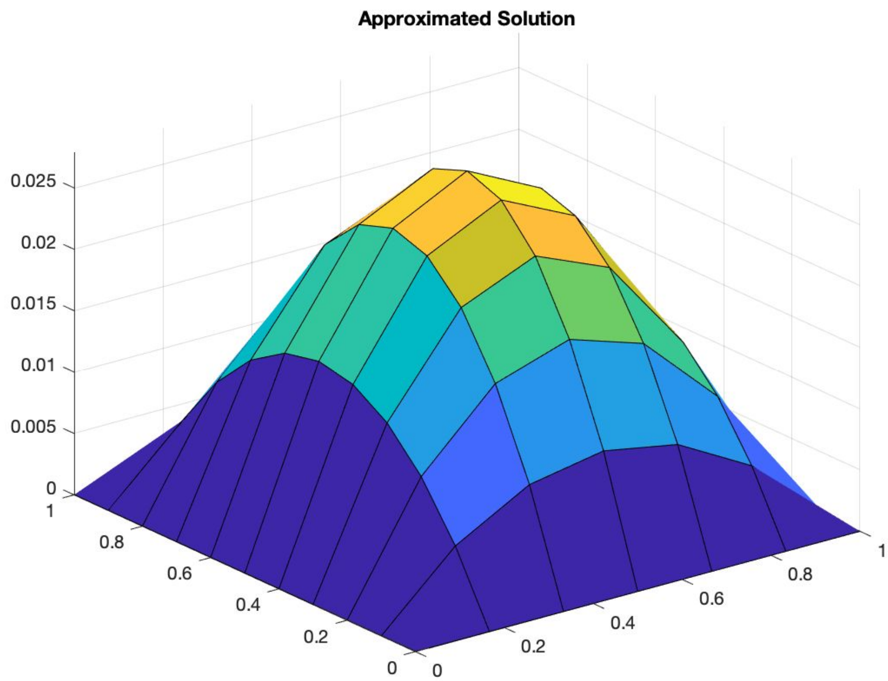


Figure 2

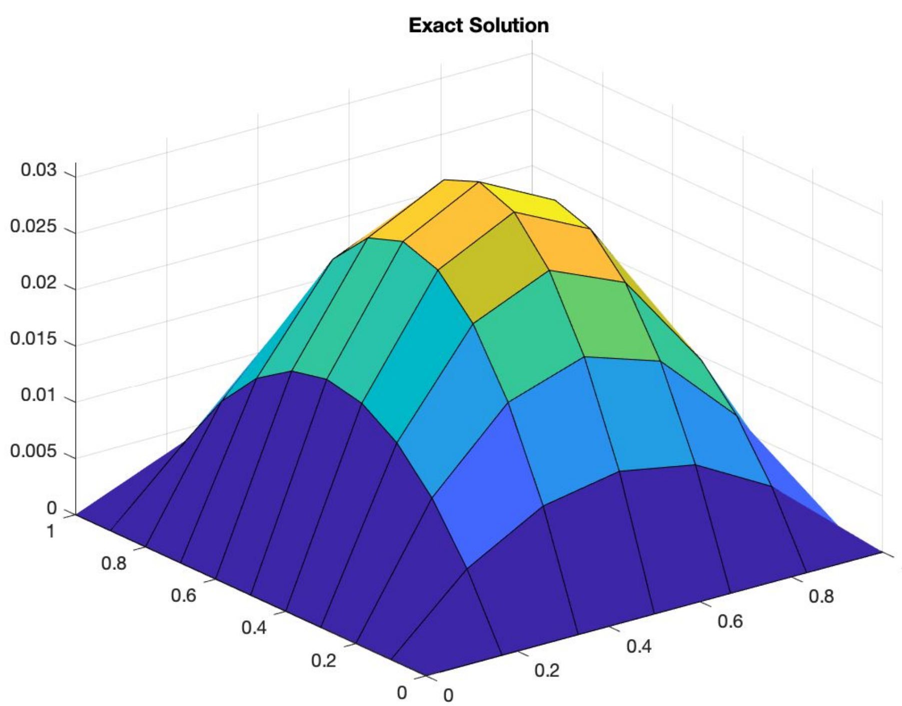
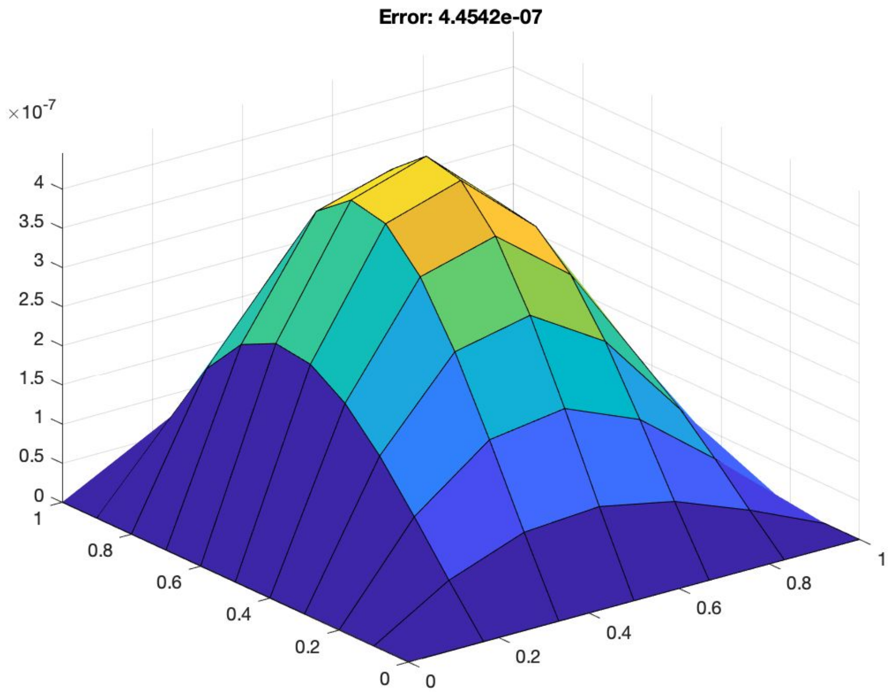


Figure 3



2. Code and run the Crank-Nicolson method with different choices of h and k for the following parabolic equation:

$$\partial u/\partial t = \partial^2 u/\partial x^2,\ 0 < x < 2,\ t > 0,$$

with boundary conditions

$$u(0,t) = u(2,t) = 0,$$

and initial condition

$$u(x,0) = \sin(\pi x(x - ½))$$

This problem utilizes the Crank-Nicolson method of approximation, in this case, for a parabolic partial differential equation. As in part 1, the finite difference is employed to discretize and simplify the problem so it may be solved computationally. This creates a tridiagonal linear system which can then be solved to retrieve approximations.

The Crank Nicolson method from algorithm 12.3 in the textbook (page 753) relies on a selection of integers N and m, which define the amount of steps to divide the time into and the amount of sample points to take from the defined range of x values, respectively. The x values are defined above and as this problem is open ended on the time interval (T is undefined) over which to sample, I chose $T = 2$, *i.e.* $0 < t < 2$. Within the algorithm, these step sizes are then calculated as $k = T/N$, and $h = l/m$, where l is the endpoint of the x values' range, in this case, 2.

The file *CNMethod_final* alters the parameters of the function slightly and instead accepts defined values of h and k, so the step size may be declared initially, rather than computed within the algorithm, and thus the number of steps are calculated from h and k, rather than vice versa. This allows for an iterative process to observe the changes in the approximations made by the algorithm as the step size decreases.

Pseudocode

Crank Nicolson:

Define endpoint of boundary l; define end time T; define alpha; define h, k

1. *Calculate m, N, from h, k, define lambda = (alpha^2 * k) / h^2, initialize final approx to 0*
2. *Build tridiag system using u(x, 0) as defined in problem*
    a. *Solve tridiag system*
3. *Output approx*

final_2 :

Calls CNMethod_final and plots approximation

Figure 4 illustrates the preceding algorithm run with *h = k = 0.1.*

Figure 4

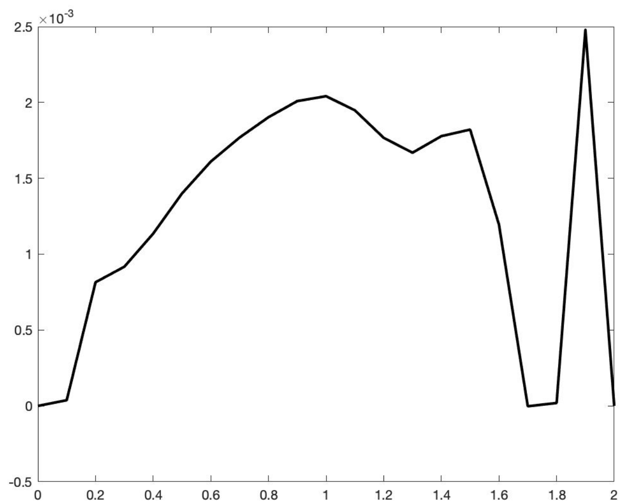


Figure 5 illustrates the same algorithm run with a decreased step size in the sampling of the *x* values, *h = 0.05*.

Figure 5

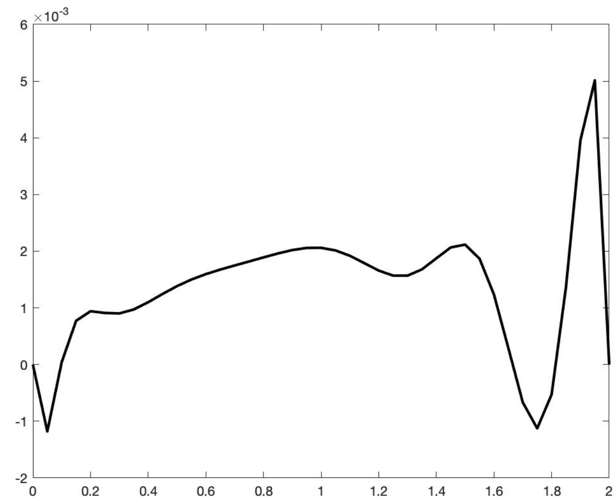


Figure 6 is the algorithm run with a decreased step size in the sampling of the *t* values, *k = 0.05.*

Figure 6

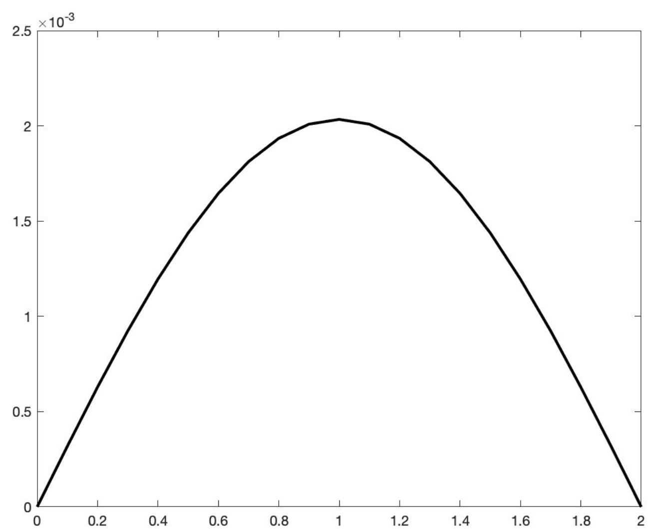


Figure 7 is a plot of the approximations from the algorithm run with $h = k = 0.05$.

Figure 7

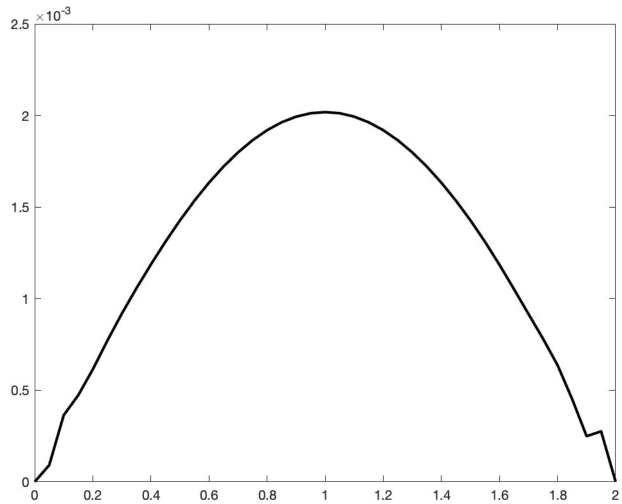
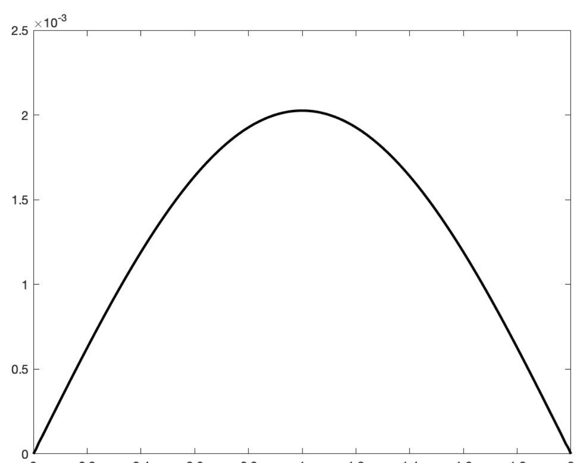


Figure 8 is the same as above, with $h = k = 0.01$.

Figure 8



From the above diagrams we can observe that the *h* and *k* values affect the smoothness and shape of the graph, respectively. As decreasing the step size *h* of the *x* value sample points increases the amount of data points used to approximate the curve, it becomes clear that the smaller the *h* value, the more smooth the approximation plot. Likewise, decreasing the step size *k* of the *t* values increases the amount of times the data is sampled, and lends to an more accurate approximation that converges towards the exact solution, a parabola.

Reference:

Burden, R. L., Faires, J. D., & Burden, A. M. (2016)*. Numerical Analysis (10th Edition).*