ASSIGNMENT 7: THE HEAT EQUATION

November, 2010 Guanqing Ou and Arash Ushani

PROBLEM STATEMENT

Solve the heat equation,

$$U_t = U_{xx}, x \in [0,1], t \geq 0$$

where $U_t = \frac{dU}{dt}$ and $U_{xx} = \frac{d^2U}{dt^2}$, with boundary and initial conditions

$$U(0,t) = U(1,t) = 0, U\left(\frac{1}{2}, 0\right) = 1$$

using the finite difference method, collocation method, and Galerkin's method in space, and Euler's in time. Compare and discuss the accuracy and stability of these methods given that the heat equation has the exact solution

$$U(x,t) = \sum_{k=1}^{\infty} a_k e^{-k^2\pi^2 t} \sin(k\pi x)$$

where a_ks are chosen to satisfy the initial condition.

Essentially, we are interested in the behavior of a unit-length rod whose temperature is zero at every point except the center, and whose ends are approximated to always have temperature zero.; $U(x,t)$ is the temperature at any point x along the length of the rod at time t.

COMPUTATIONAL APPROACH

Finite-Difference Method

The finite difference method approximates the exact values of $U(x_i, t)$ with an evenly spaced grid of N points $U_i(t)$ on the domain $x = [0,1]$. At each U_i, $U_i' = U'(x_i, t)$, which is approximated as follows:

$$U_i' = \frac{U_{i+1} - 2U_i + U_{i-1}}{(\Delta x)^2} \quad (1)$$

If we express the temperatures at each time t as a vector $\vec{U} = \begin{bmatrix} U_1 \\ \vdots \\ \vdots \\ U_N \end{bmatrix}$, then we can define a matrix

$$A = \frac{1}{(\Delta x)^2}\begin{bmatrix} -2 & 1 & 0 & \dots & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & \vdots \\ 0 & 1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 1 & -2 & 1 \end{bmatrix}$$

such that

$$\frac{d\vec{U}}{dt} = A\vec{U}. \tag{2}$$

With this expression for the derivative of $\vec{U}$, we can apply Euler's method, which at the most basic level assumes that we can approximate exact values of a function $g(x_i, t)$ with $g_i(t)$where

$$g_i = g_{i-1} + \Delta t\, g'_{i-1}(t) \tag{3}$$

In the case of the heat equation,

$$\vec{U}_{i+1} = \vec{U}_i + \Delta t \frac{d\vec{U}_i}{dt} \tag{4}$$

After substituting our expression for $\frac{d\vec{U}}{dt}$ and factoring

$$\vec{U}_{i+1} = \vec{U}_i + \Delta t A\vec{U}_i = (I + \Delta t A)\vec{U}_i \tag{5}$$

Using the given initial conditions, we can find any $\vec{U}_i$:

$$\vec{U}_i = (I + \Delta t A)^i \vec{U}_0 \tag{6}$$

With this expression and known initial and boundary conditions, we can numerically solve the heat equation.

Collocation

The collocation method is built upon the assumption that $U(x,t)$ is the linear combination of a set of linearly independent basis functions. A set of N "collocation points", or grid points, is chosen on the domain such that

$$U(x,t) = \sum_{j=1}^{N} a_j(t)\phi_j(x) \tag{7}$$

where the a_js are coefficients and the ϕ_js are the basis functions. Basis functions can be algebraic or trigonometric functions of j and x, such as $x^j(1-x)$ and $\sin(j\pi x)$. In order to numerically solve for $U(x,t)$, we solve for the coefficients a_j. While we only solve for the coefficients for the collocation points, the resulting expression for $U(x,t)$can be used to approximate the solution at all points in the domain.

To numerically solve the heat equation, we substitute our expression for $U(x,t)$ into the ODE $U_t = U_{xx}$ for each of N points x_i in the domain, such that we have N equations for $i = 1, \dots, N$

$$\sum_{j=1}^{N} \dot{a_j}(t)\phi(x_i) = \sum_{j=1}^{N} a_j(t)\phi''(x_i) \tag{8}$$

In order to state this in such a way that we can apply Euler's method, we express equation 8 using vector notation:

$$B\vec{\dot{a}} = A\vec{a} \tag{9}$$

where $\vec{a} = \begin{bmatrix} a_1 \\ \vdots \\ a_N \end{bmatrix}$, $B\begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots \\ \phi_1(x_2) & \phi_2(x_2) & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$= , $A = \begin{bmatrix} \phi_1''(x_1) & \phi_2''(x_1) & \dots \\ \phi_1''(x_2) & \phi_2''(x_2) & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$.

This allows us to find an expression for the derivative of what we are solving for (the coefficients, a_j)

$$\vec{\dot{a}} = B^{-1}A\vec{a} \tag{10}$$

such that we can apply Euler's method in the same way as in the finite difference method:

$$\vec{a}_i = \vec{a}_i + \Delta t B^{-1}A\vec{a}_{i-1} = (I + \Delta t B^{-1}A)\vec{a}_{i-1} \tag{11}$$

$$\vec{a}_i = (I + \Delta t B^{-1}A)^i\vec{a}_0 \tag{12}$$

Galerkin

The Galerkin method, similar to the collocation method, assumes that $U(x,t)$ can be expressed as a linear combination of basis functions based on j and x. In addition to this, the Galerkin method assumes that there is some error that cannot be accounted for in this simplification, which is defined as the residual function, $r(x)$. Essentially, it is assumed that every $U(x,t)$ is a vector in 3D, and the span of the basis function only covers a plane in this 3D space. $r(x)$ is thus orthogonal to

the basis functions. We define $r(x) = U_{xx} - g(x)$ where $U_{xx} = g(x)$, and solve for the coefficients a_j such that we minimize the residual.

As with collocation, we lay down a grid of N points in the domain. Because $r(x)$ is defined to be orthogonal to the basis functions ϕ_i, this means that for exact solutions, the integral of the dot product of the residual and the basis functions is equal to zero: $\int_0^1 r(x)\phi_i(x)dx = 0$ for each grid point x_i, $i = 1, \dots, N$.

For the heat equation, $r(x) = U_{xx} - U_t$, and using our expression for $U(x,t)$ as the linear combination of basis functions, we can say

$$\int_0^1 \left(\sum_{j=1}^{N} \dot{a}_j(t)\phi_j(x)\,dx - \sum_{j=1}^{N} a_j(t)\phi_j''(x)\,dx \right) \phi_i(x)dx = 0$$

$$\text{for } i = 1, \dots, N$$

Since the integral of sums is equal to the sum of integrals, this can be simplified to

$$\sum_{j=1}^{N} \dot{a}_j(t) \int_0^1 \phi_j(x)\phi_i(x)dx = \sum_{j=1}^{N} a_j(t) \int_0^1 \phi_j''(x)\phi_i(x)dx.$$

Again, as with the collocation method, we can express this in vector matrix for $B\vec{\dot{a}} = A\vec{a}$, where $\vec{a} = \begin{bmatrix} a_1 \\ \vdots \\ a_N \end{bmatrix}$, $B = \begin{bmatrix} \int_0^1 \phi_1\phi_1{}_1 & \int_0^1 \phi_1\phi_2 & \dots \\ \int_0^1 \phi_2\phi_1 & \int_0^1 \phi_2\phi_2 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$, $A = \begin{bmatrix} \int_0^1 \phi_1''\phi_1 & \int_0^1 \phi_2''\phi_1 & \dots \\ \int_0^1 \phi_1''\phi_2 & \int_0^1 \phi_2''\phi_2 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$, which allows us to find an expression for $\vec{a}_i$.

With both the collocation and Galerkin method, the coefficients we solve for can then be substituted back into the equation $U(x,t) = \sum_{j=1}^{N} a_j(t)\phi_j(x)$ to solve for the temperature at all $x \in [0,1]$ along the heated rod at all time points $t \geq 0$.

IMPLEMENTATION

The above methods were implemented in MATLAB as shown below. A trigonmetric basis, $\phi_i = \sin(j\pi x)$, $N = 20$ grid points, with $\Delta t = 0.01$, and total time = 1 seconds were used.

Finite-Difference Method

The initial condition was defined as a column vector of 0s except at $x = \frac{1}{2}$, where $U\left(\frac{1}{2}, 0\right) = 1$. The equation $\vec{U}_i = (I + \Delta t A)^i \vec{U}_0$ is used to step the value of $U(x,t)$ forward through time.

```
function res = finitediff(u0, bound, x0, range, N, dt,time, c)
syms

    %% u_m = (I + dt*A)^m * u0
    dx = range/N;
    u_init = zeros(1,N);
    u_init(u0(1) * N) = u0(2);
```

The matrix A is defined using diagonals.

```
    A_constantpart = c/(dt^2);
    maindiag= (-2* ones(N,1));
    sidediag= ones(N-1,1);
   A_matrix = diag(maindiag)+diag(sidediag,-1)+diag(sidediag,1);
    A= A_constantpart* A_matrix;
```

$(I + \Delta t A)$ is defined.

```
    IplusdttimesA = eye(N)+ dt* A;
```

$\vec{U}_i$ is calculated for each time step *m*, and the temperatures are plotted.

```
    for m = 1:(time/dt)
       u=u_init*(IplusdttimesA)^m;
       plot(linspace(1,N,N), u);
       pause(.1)
    end
    res = 1;
end
```

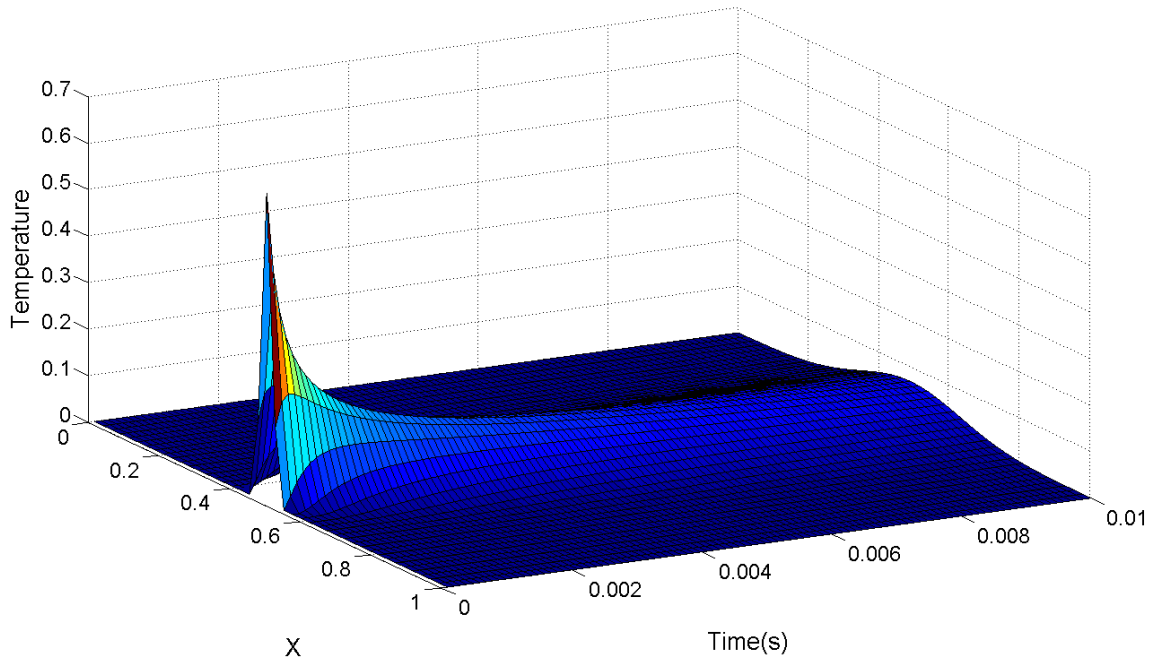

Figure 1. Graph showing the temperature along the rod through time, generated using the finite difference method.

Collocation

The function is takes the following inputs:

- the form of the basis function; initial conditions u0, which is an array with position and initial value at that position
- boundary values with the same format
- x0 and range, which allows us to define the domain
- N which defines how many elements the domain is split into
- dt and time, which define the time step and how many steps to go forward
- and c, a constant factor that can be included in the definition of the ODE, but which was just 1 in our case.

```
function res = collocation_heat(basis, u0, bound, x0, range, N, dt,time, c)
syms x t jay
```

Here we simply lay down the grid of x and j values based on the number of collocation points. We then define derivatives of the basis, as these will be used later to construct matrices A and B.

```
    phi = basis;
```

```
    dx = range/(N+1);
    xs = x0+dx: dx: x0+range-dx;
    jays = linspace(1,N,N);
    phiprime = diff(phi,x);
    phidprime = diff(phiprime,x);
```

To find the a0 using the initial conditions given, we know the temperature at all points x at time 0. Since we assume $U(x,0) = \sum_{j=1}^{N} a_j(0)\phi_j(x)$, and we can find $\phi_j(x)$, and solve for $a_j(0)$.

```
    u_initial = zeros(1,N);%initial conditions--general
    u_initial(floor(N* u0(1))) = u0(2);% initial conditions - with actual
values substituted

    b_temp= subs(phi, jay, jays);
    b= zeros(N,N);
```

"amatrix" and "b" are the variable names for matrices A and B. Both matrices are first defined symbolically, then appropriate values of js and xs are substituted in.

```
    amatrix_temp = subs(phidprime, jay, jays);
    amatrix =zeros (N,N);
    for i =1:N
        amatrix(i,:)= subs(amatrix_temp, x, xs(i));
        b (i,:) = subs(b_temp, x,xs(i));
    end
```

Here we solve for $a_j(0)$ as explained above. The for loop is used for both the definition of "amatrix" and "b" for convenience.

```
    a0 = linsolve(b,u_initial');
    %%B*adot= A*a; A= amatrix
    %%a_m = (I+dt*B^-1 * A)^m * a0

    a = zeros(N,N);
```

This for loop allows calculation of the coefficient matrix $a_j(t)$ at each time timestep m by taking the matrix $(I + \Delta t B^{-1}A)$ to the power of the timestep, then uses the coefficient vector to find an equation that can then be used to find the temperature at every point x at this particular time after zero.

```
    for m = 1:time/dt %m = time step that you're on
        time(m) = m*dt;
        a=(eye(N)+ (dt*b^(-1)*amatrix))^m*a0;
        eqtn_expression= b_temp*a;%U(x,t) = Sum (j= 1 to N) a_j(t)*phi_j(x))
        eqtn_func= matlabFunction(eqtn_expression);
        pts= eqtn_func(xs);
        pts_(m,:)= [bound(1) pts bound(2)];
    end
    xs_ = [x0 xs x0+range];
    surf(xs_, time, pts_);
    res = 1;
end
```

This implementation gives results that make sense in the context of the heat equation: the heat spreads along the rod, then dissipates (Figure 2).

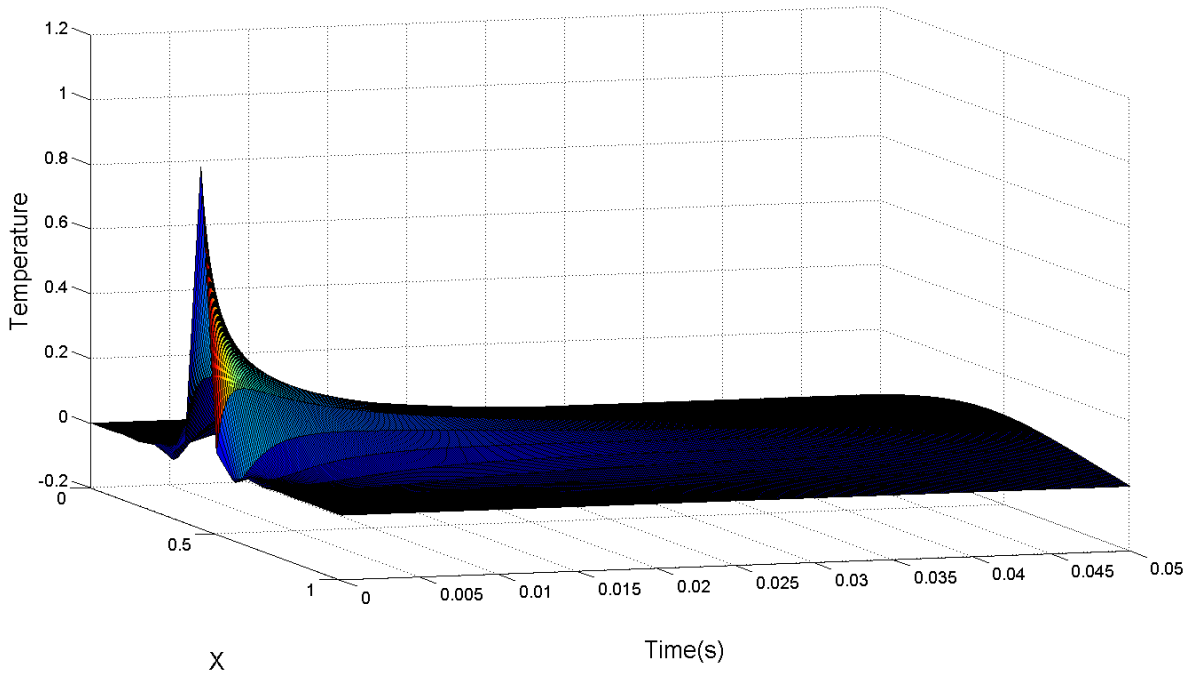

Figure 2. Graph showing the temperature along the rod through time, generated using the collocation method.

Galerkin

The Galerkin method is essentially implemented the same way, except matrices *A* and *B* consist of integrals instead of just derivatives of the basis function. This implementation led to results that were essentially the same as shown in Figure 2.

```
function res = galerkin_heat(basis, u0, bound, x0, range, N, dt,time, c)
syms x t jay jay2

    phi = basis;
    dx = range/(N+1);
    xs = x0+dx: dx: x0+range-dx;
    jays = linspace(1,N,N);
    phiprime = diff(phi,x);
    phidprime = diff(phiprime,x);
    b = zeros(N,N);
    a = zeros(N,N);
    b_temp= subs(phi, jay, jays);
    phi2= subs(phi, jay, jay2);
    xend=x0+range;
    phi=basis;

    integrand_uxtimesphix = u0(2) *phi;
    integral= int(integrand_uxtimesphix,x, x0,xend);
```

```
    u_initial = zeros(1,N);
    u_initial(floor(N* u0(1))) = u0(2);

    b_temp= subs(phi, jay, jays);
    b= zeros(N,N);

    amatrix_temp = subs(phidprime, jay, jays);
    amatrix =zeros (N,N);
```

Calculating initial conditions as described before:

```
    for i =1:N
        amatrix(i,:)= subs(amatrix_temp, x, xs(i));
        b (i,:) = subs(b_temp, x,xs(i));
    end
    a0 = linsolve(b,u_initial');
```

Populating matrices A and B.

```
    for i = 1:N
        for j = 1:N
            amatrix_integrand = matlabFunction(subs(phidprime*phi2, {jay
jay2},{i j}));
            b_integrand = matlabFunction (subs(phi*phi2, {jay jay2}, {i j}));
            amatrix(i,j) = quad(amatrix_integrand, x0,xend);
            b(i,j) = quad(b_integrand, x0,xend);
        end
    end
```

Using the expression $\vec{U}_{\mathrm{i}} = (I + \Delta t A)^{i}\vec{U}_0$ to calculate the temperature at each point along the rod at each time point.

```
    for m = 1:time/dt
        time(m) = m*dt;
        a=(eye(N)+ (dt*inv(b)*amatrix))^m*a0;
        eqtn_expression= b_temp*a; %U(x,t) = Sum (j= 1 to N) a_j(t)*phi_j(x))
        eqtn_func= matlabFunction(eqtn_expression);
        pts= eqtn_func(xs);
        pts_(m,:) = [bound(1) pts bound(2)];
    end
    xs_ = [x0 xs x0+range];
    surf(xs_, time, pts_);
    res=1;

end
```

STABILITY AND ACCURACY

Stability

For the finite difference method, we expect the stability to be determined by the eigenvalues of the matrices $(I + \Delta t B^{-1}A)$. For collocation and Galerkin, the matrix that determines stability is $(I + \Delta t B^{-1}A)$. We want $(I + \Delta t B^{-1}A)^m$ to approach zero as m approaches infinity, since this means that the coefficient matrix approaches zero, and all *U(x,t)* approach zero as time passes. This makes sense physically, as we would expect the heat to dissipate completely with time. In order for this to be true, all eigenvalues of the matrix $(I + \Delta t B^{-1}A)$ should have magnitude less than 1. Essentially, we can think of eigenvalues as how much the values that the matrix is multiplied by is scaled, so we can clearly see that if the eigenvalues are greater than 1, there will be growth instead of decay as time progresses.

The following code looks at the behavior of the maximum eigenvalues as timestep and the number of grid points are varied:

```
function res = stability7 (method, N, dt)
    if nargin < 3
        dt = [1e-4 .01];
    end
    if nargin< 2
        N =[2 20];
    end
    if nargin < 1
        method = @collocation_ht;
    end

    dts = dt(1):1e-3: dt(2);
    ns = N(1) : N(2);
```

In this step, for each combination of N and stepsize, the eigenvalue of the matrix $(I + \Delta t B^{-1}A)$ is calculated. "Method" is whichever method we are evaluating, and for this calculation, the output of the finite difference was set as $(I + \Delta t A)$, and the outputs of the collocation and Galerkin methods were set as $(I + \Delta t B^{-1}A)$.

```
    for nindex = 1:length(ns)
        for tindex = 1:length(dts)
            tstepsize=dts(tindex);
            EV= eig(method(ns(nindex), tstepsize));
            EVmax(nindex, tindex)=max(EV);
        end
    end
    surf(dts,ns,EVmax);
end
```

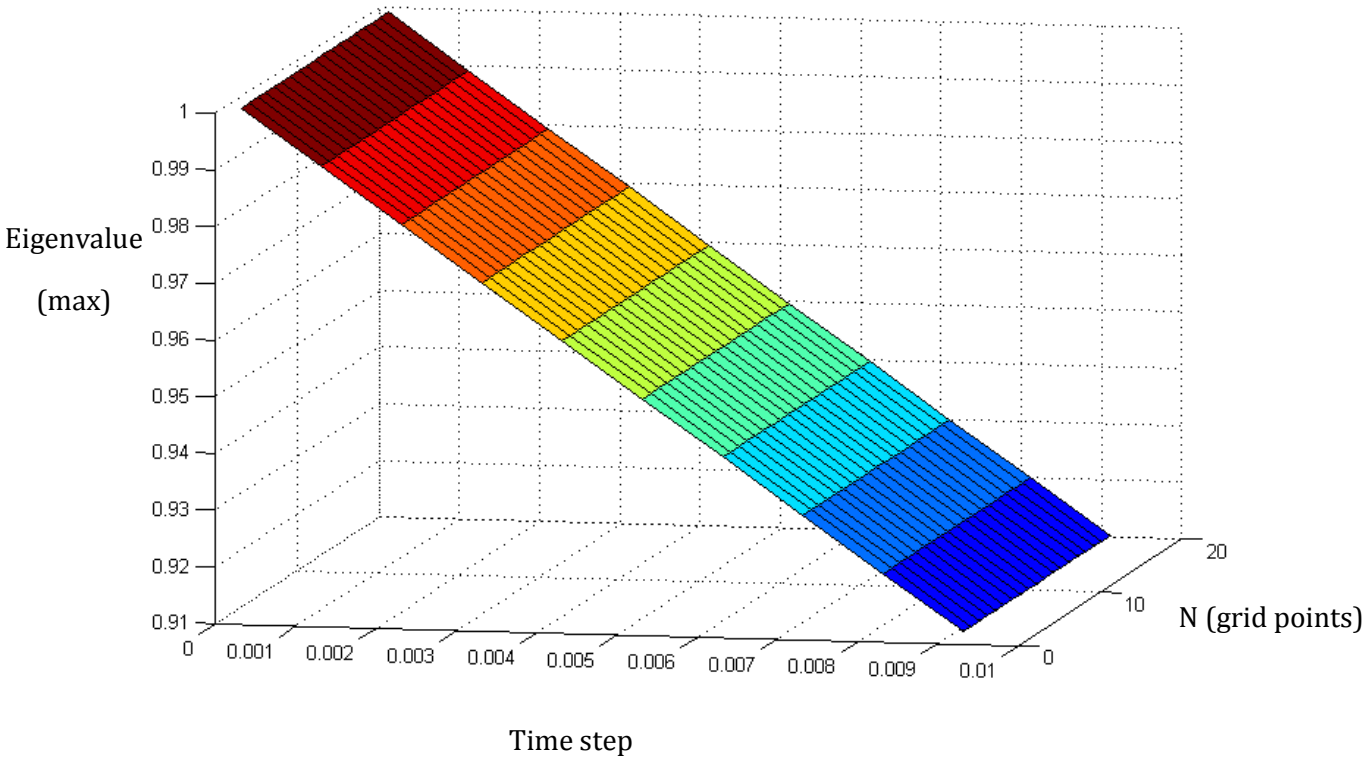

Figure 3. Graph showing the maximum eigenvalue found for the matrix $(I + \Delta t B^{-1} A)$ at each combination of a range of time step sizes and collocation grid sizes. The graph generated for the finite difference method followed the same trend

And Figure 3 shows the resulting trend. It appears that within our normal bounds (N between 2 and 20) and time steps varying from .0001 to 0.01, we should expect stable behavior from the collocation method. The same was found for the finite difference and Galerkin methods.

Accuracy

In order to look at accuracy, we use the given exact solution, $U(x,t) = \sum_{k=1}^{\infty} a_k e^{-k^2\pi^2 t} \sin(k\pi x)$. To solve for a_k, we use the initial conditions given to set *U(x,0)*, and divide the matrix containing $e^{-k^2\pi^2 t}\sin(k\pi x)$ with *x*, *k,* and *t* substituted by this initial conditions to find the coefficient matrix. Instead of infinity, we vary *k* over a wide range. This was implemented as follows:

```
function res = accuracy7(method, K, time, x0, range,u0,N)
syms k x t jay
```

Parameters:

```
    if nargin < 7
        N = 20;
    end
    if nargin < 6
```

```
    u0 = [1/2,1];
end
if nargin< 5
    range = 1;
end
if nargin < 4
    x0 = 0;
end

if nargin < 3
    time = .05;
end
if nargin< 2
    K = 60;
end
if nargin < 1
    method =@galerkin_heat;
end
```

The exact solution is calculated using the function “exactsolution7”, seen below.

```
exact = @exactsolution7;
```

The grid of x values calculated using the exact solution is generated based on the size of K. This is distinguished from N, which determines the grid size used in the numerical method.

```
ks = 1: K;
dx = range/(K-1);
xs = x0:dx:x0+range;
xs = linspace(x0,range);
```

The exact solution is calculated for each of the x values in the grid “xs”, and for each k value, at time zero.

```
for x = 1:length(xs)
    for k = 1: length(ks)
        B(x,:)= exact(ks,xs(x),0);
    end
end
```

This initial set of values is used with the initial conditions to solve for the array a_k, which can then be used to calculate the solution at any time t.

```
u_initial = zeros(length(xs),1);
u_initial(round(length(xs)* u0(1)),1) = u0(2);
ak = linsolve(B, u_initial);
 for x = 1:length(xs)
    B(x,:)= exact(ks,xs(x),time);
 end
 exactpts=B*ak;
```

The values at the middle of the rod is calculated at each time point.

```
exact_pt=exactpts(round(length(xs)* u0(1)));
```

Then, for each size of N, calculate the difference between the temperature at the middle of the rod as calculated by the exact solution and the numerical method of choice.

```
    for i = 1:N
        N (i) = i;
        syms jay x
        approx=collocation_heat(sin(jay*pi*x), [1/2 1], [0 0],
0,1,i, .0001,.05);
        diff(i)=abs(exact_pt- approx);
    end
   plot(N, diff);
   res =ak;

end

function res = exactsolution7(k,x,t)
res=exp(-k.^2 * pi^2*t).*sin(k*pi*x);
end
```

This yielded the following pattern for the Galerkin method:

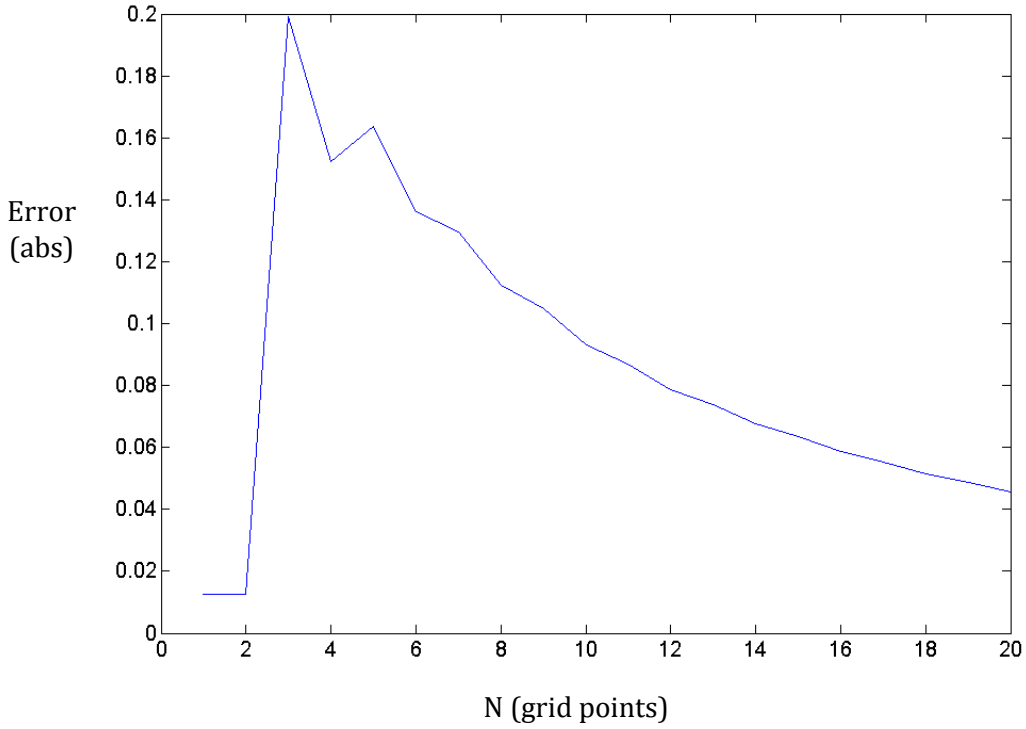

Figure 4. Graph of error between the temperature at the midpoint calculated using the exact solution and the Galerkin method. The same trend was found for the finite difference and collocation method.

It is likely that when N is too small, there is instability that leads to sinusoidal behavior, and the low error found in this range is the accidental result of the sine wave fitting the initial pattern of the solution to the heat equation. But once this range is passed, the trend is as we would expect: increasing the size of the grid increases the accuracy of the method.

CONCLUSION

Of the three methods presented here, it is clear that the finite-difference method is the fastest. The accuracy and stability of the three methods appear to be identical in the ranges investigated in this tutorial.