Estimating the heat equation

Kurtis Doobay

Introduction

The heat equation is one of simplest partial differential equations and was the first one I learned in university. Instead of solving for the continuous solution, I will be numerically solving for discrete times and positions. In this paper I will present three schemes appropriately labeled as such: 1, 2 and Richardson. The scheme 1 will be derived using a one-sided derivative for $\frac{\partial u}{\partial t}(x,t)$, and a two-sided derivative $\frac{\partial^2 u}{\partial x^2}(x,t)$ giving an error of $O(dx^2 + dt)$. To decrease the order of the error, scheme 2 will use a midpoint approximation for $\frac{\partial u}{\partial t}(x, t + dt)$, the two-sided derivative $\frac{\partial^2 u}{\partial x^2}(x, t + dt)$, and have dt and dx to be in the same order of magnitude to give an error of $O(dt^2)$. Finally, Richardson's Extrapolation will be used to refine the scheme 2 solution. Richardson's is applied over two forms of scheme 2, one of approximated in $\frac{dt}{2}$, the other approximated in dt with a cubic spline to estimate all $\frac{dt}{2}$ midpoints with the respective degree of error. A combination of these two will return a scheme with an error $O(dt^4)$.

Deriving the schemes

Scheme 1

Iterative Formula

The one-dimensional heat equation is defined as $\frac{\partial u}{\partial t}(x,t) = \alpha * \frac{\partial^2 u}{\partial x^2}(x,t)$, where α is the rate of diffusion. The initial conditions $\alpha = 1; u(0,t) = u(L,t) = 0; u(x,0) = \sin x$ gives enough information for $L = \pi$. In both schemes, we have , $\frac{\partial^2 u}{\partial x^2}(x,t) = u_{xx}(x,t) \approx \frac{u(x-dx,t) - 2u(x,t) + u(x+dx,t)}{dx^2}$. In scheme 1, the one-sided derivative is used as $\frac{\partial u}{\partial t}(x,t) = u_t(x,t) \approx \frac{u(x,t+dt) - u(x,t)}{dt}$.

Solving for an iterative formula:

$$\frac{\partial u}{\partial t}(x,t) = \frac{\partial^2 u}{\partial x^2}(x,t) \approx \frac{u(x,t+dt) - u(x,t)}{dt} = \frac{u(x-dx,t) - 2u(x,t) + u(x+dx,t)}{dx^2}$$

$$u(x,t+dt) = u(x,t) + dt\frac{u(x-dx,t) - 2u(x,t) + u(x+dx,t)}{dx^2}$$

Given our initial conditions $\forall t;\ u(0,t) = u(L,t) = 0$, we can solve for any point on defined on (x,t).

Matrix Form

Using this iterative formula, we can solve in the form of a matrix. Let $u_x(t)$ be a matrix of x values incremented by dx at time position t of size 1 by $\frac{L}{dx} - 1$.

$$A = \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix}, r = \frac{dt}{dx^2}, I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}$$

Where A is a square matrix and I is the identity matrix both of the size $\frac{L}{dx} - 1$. Note for A, we do not include the end points where $u(0,t) = u(L,t) = 0$. The matrix formula for solving $u(x, t + dt)$ in the form of the matrix $u_x(t + dt)$ is:

$$u_x(t + dt) = (I + r * A) \cdot u_x(t)$$

Scheme 2

Attempting approximation using symmetric derivative

For S2, the symmetric derivative is $\frac{\partial u}{\partial t}(x,t) \approx \frac{u(x,t+dt)-u(x,t-dt)}{2dt}$.

Solving for an iterative formula:

$$\frac{\partial u}{\partial t}(x,t) = \frac{\partial^2 u}{\partial x^2}(x,t) \approx \frac{u(x,t+dt) - u(x,t-dt)}{2dt} = \frac{u(x-dx,t) - 2u(x,t) + u(x+dx,t)}{dx^2}$$

$$u(x,t+dt) = u(x,t-dt) + 2dt\frac{u(x-dx,t) - 2u(x,t) + u(x+dx,t)}{dx^2}$$

But this iterative formula has a dependence at $t - dt$ for the right-hand side, which we do not have at $t = 0$!

Matrix solution using midpoint approximation

Therefore, we will instead solve for every dt locations using these new equations with a dependence of only t. I would like to thank this video for helping me conceptualize the midpoint approximation: https://www.youtube.com/watch?v=D-huCvF15-g.

$$\frac{\partial u}{\partial t}(x, t + \frac{dt}{2}) \approx \frac{u(x,t+dt) - u(x,t)}{dt}$$

$$\frac{\partial^2 u}{\partial x^2}\left(x, t + \frac{dt}{2}\right) \approx \frac{u\left(x - dx, t + \frac{dt}{2}\right) - 2u\left(x, t + \frac{dt}{2}\right) + u\left(x + dx, t + \frac{dt}{2}\right)}{dx^2}$$

$$\frac{u(x,t+dt) - u(x,t)}{dt} = \frac{u\left(x - dx, t + \frac{dt}{2}\right) - 2u\left(x, t + \frac{dt}{2}\right) + u\left(x + dx, t + \frac{dt}{2}\right)}{dx^2}$$

$$u(x,t+dt) = u(x,t) + dt\frac{u\left(x-dx,t+\frac{dt}{2}\right) - 2u\left(x,t+\frac{dt}{2}\right) + u\left(x+dx,t+\frac{dt}{2}\right)}{dx^2}$$

Before we can solve this, we need to estimate $u(x,t+\frac{dt}{2})$ in terms of dt. With the exact formula given for $u(x,0) = \sin x$, we have the luxury of being able to set dt. With a sufficiently small enough dt, we can use the average of the points to estimate the value, such that $u\left(x,t+\frac{dt}{2}\right) \approx \frac{u(x,t)+u(x,t+dt)}{2}$.

$$u(x,t+dt) - dt\frac{u(x-dx,t+dt) - 2u(x,t+dt) + u(x+dx,t+dt)}{2dx^2}$$
$$= u(x,t) + dt\frac{u(x-dx,t) - 2u(x,t) + u(x+dx,t)}{2dx^2}$$

From this formula, let of construct a matrix to solve for $u(x,t+dt)$. Let $u_x(t)$ be a matrix of x values incremented by dx at time position t of size 1 by $\frac{L}{dx} - 1$.

$$A = \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix}, r = \frac{dt}{2 * dx^2}, I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}$$

Where A is a square matrix and I is the identity matrix both of the size $\frac{L}{dx} - 1$. The formula in the form of a matrix is as follows:

$$(I - r * A) \cdot u_x(t+dt) = (I + r * A) \cdot u_x(t)$$

Because $(I + r * A) \cdot u_x(t)$ reduces to a matrix of size 1 by $\frac{L}{dx} - 1$, we can use row reduction to solve for $u_x(t+dt)$.

Order of Scheme 2

Error on both terms can be calculated using Taylor series.

About $t + \frac{dt}{2}$; $u = u\left(x,t+\frac{dt}{2}\right), a = t + \frac{dt}{2}$, let $u' = \frac{\partial u}{\partial t}\left(x,t+\frac{dt}{2}\right)$

$$u(x,t) = u - \frac{dt * u'}{2} + \frac{dt^2 * u''}{2^2 * 2!} - \frac{dt^3 * u'''}{2^3 * 3!} + \frac{dt^4 * u^{(4)}}{2^4 * 4!} - \frac{dt^5 * u^{(5)}}{2^5 * 5!} + \cdots$$

$$u(x,t+dt) = u + \frac{dt * u'}{2} + \frac{dt^2 * u''}{2^2 * 2!} + \frac{dt^3 * u'''}{2^3 * 3!} + \frac{dt^4 * u^{(4)}}{2^4 * 4!} + \frac{dt^5 * u^{(5)}}{2^5 * 5!} + \cdots$$

$$\frac{u(x,t+dt) - u(x,t)}{dt} = u' + \frac{dt^2 * u'''}{2^2 * 3!} + \frac{dt^4 * u^{(5)}}{2^4 * 5!} \ldots = u' + O(dt^2)$$

About x; $u = u(x,t+\frac{dt}{2})$, let $u' = \frac{\partial u}{\partial x}\left(x,t+\frac{dt}{2}\right)$

$$u\left(x,t+\frac{dt}{2}\right)=u$$

$$u\left(x-dx,t+\frac{dt}{2}\right)$$
$$=u-dx*u'+\frac{dx^2*u''}{2!}-\frac{dx^3*u'''}{3!}+\frac{dx^4*u^{(4)}}{4!}-\frac{dx^5*u^{(5)}}{5!}+\frac{dx^6*u^{(6)}}{6!}\dots$$

$$u\left(x+dx,t+\frac{dt}{2}\right)$$
$$=u+dx*u'+\frac{dx^2*u''}{2!}+\frac{dx^3*u'''}{3!}+\frac{dx^4*u^{(4)}}{4!}+\frac{dx^5*u^{(5)}}{5!}+\frac{dx^6*u^{(6)}}{6!}\dots$$

$$\frac{u\left(x-dx,t+\frac{dt}{2}\right)-2u\left(x,t+\frac{dt}{2}\right)+u\left(x+dx,t+\frac{dt}{2}\right)}{dx^2}$$
$$=u''+\frac{2*dx^2*u^{(4)}}{4!}+\frac{2*dx^4*u^{(6)}}{6!}+\cdots=u''+O(dx^2)$$

Plugging in the equations:

$$\frac{du}{dt}(x,t)-\frac{d^2u}{dx^2}(x,t)$$
$$=\frac{u(x,t+dt)-u(x,t)}{dt}$$
$$-\frac{u\left(x-dx,t+\frac{dt}{2}\right)-2u\left(x,t+\frac{dt}{2}\right)+u\left(x+dx,t+\frac{dt}{2}\right)}{dx^2}$$
$$=\left(\frac{\partial u}{\partial t}+\frac{dt^2}{2^2*3!}*\frac{\partial^3 u}{\partial t^3}+\frac{dt^4}{2^4*5!}*\frac{\partial^5 u}{\partial t^5}+\cdots\right)$$
$$-\left(\frac{\partial^2 u}{\partial x^2}+\frac{2*dx^2}{4!}*\frac{\partial^4 u}{\partial x^4}+\frac{2*dx^4}{6!}*\frac{\partial^6 u}{\partial x^6}+\cdots\right)$$
$$=\left(\frac{dt^2}{2^2*3!}*\frac{\partial^3 u}{\partial t^3}+\frac{dt^4}{2^4*5!}*\frac{\partial^5 u}{\partial t^5}+\cdots\right)-\left(\frac{2*dx^2}{4!}*\frac{\partial^4 u}{\partial x^4}+\frac{2*dx^4}{6!}*\frac{\partial^6 u}{\partial x^6}+\cdots\right)$$
$$=O(dt^2+dx^2)$$

Because the degree of error is the same from both sources, to minimize error they should be similar in size.

$$dt=dx;\frac{du}{dt}(x,t)-\frac{d^2u}{dx^2}(x,t)=O(dt^2)$$

So, the degree of error of the approximation is $O(dt^2)$.

Richardson's Extrapolation

Derivation

To reduce the degree of error, we apply Richardson's extrapolation. Let $\varphi(h)$ be the scheme 2 estimation and let $h = dt$. We know that the expanded error term is $O(h^4)$ from when we calculated the order of scheme 2. Let A be the exact value, $\varphi(h)$ and $\varphi\left(\frac{h}{2}\right)$ estimates A as such:

$$\varphi(h) = A + a_1 h^2 + O(h^4)$$

$$\varphi\left(\frac{h}{2}\right) = A + \frac{a_1 h^2}{4} + O(h^4)$$

There exists a combination of $\varphi(h)$ and $\varphi\left(\frac{h}{2}\right)$ that can reduce the h^2 error term to 0 while estimating A. A constructed matrix of $\varphi(h)$ and $\varphi\left(\frac{h}{2}\right)$ is as such:

$$\left[\varphi(h) \quad \varphi\left(\frac{h}{2}\right)\right] \cdot x = \begin{bmatrix} A & A \\ a_1 h^2 & \frac{a_1 h^2}{4} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} A \\ 0 \end{bmatrix}$$

After row reduction, we get the combination of $\frac{4}{3} * \varphi\left(\frac{h}{2}\right)$ and $-\frac{1}{3} * \varphi(h)$. Plugging it in, we get:

$$\frac{4 * \varphi\left(\frac{h}{2}\right) - \varphi(h)}{3} = \frac{4-1}{3} A + \frac{1-1}{3} a_1 h^2 + O(h^4) = A + O(h^4)$$

As such, using a combination of $\frac{4*\varphi(h)-\varphi(h)}{3}$ reduces the error of S2 from $O(h^2)$ to $O(h^4)$.

Application

While the Richardson's Extrapolation works well for continuous solutions, it poses a problem when solving numerically in an array. $\varphi\left(\frac{h}{2}\right)$ is defined for $x, t = n * \frac{dt}{2}; n \in N$, but $\varphi(h)$ is only defined for $x, t = n * dt; n \in N$, approximately 1/4th as many points (for clarification on the 1/4th, $\varphi(h)$ is both half as dense along x and t). To increase the density of $\varphi(h)$, we apply a cubic spline to estimate all $x, t = m * \frac{dt}{2}; m = 2 * n + 1; n \in N$ values. In my estimation, I calculated the midpoints along t first before x, but it should not matter which order.

Graphs

The following plots assign $dt = 0.1; dx = \frac{\pi}{10}$.

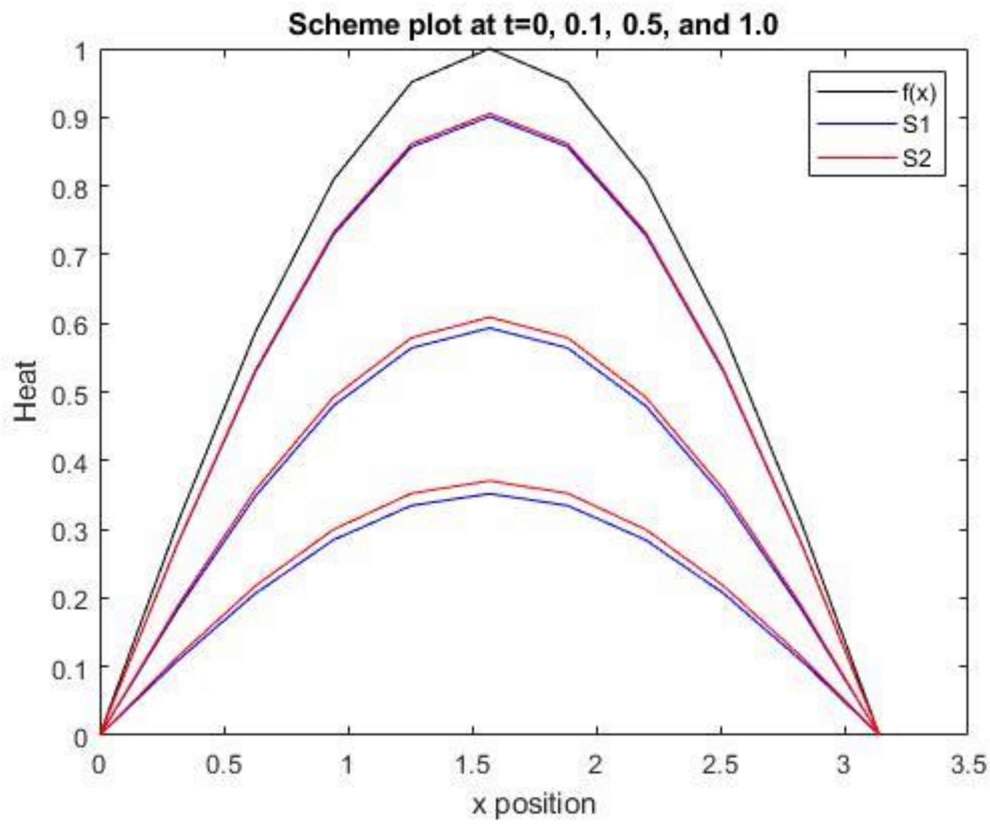

Figure 1:S1 and S2. The difference in the degree of error is so large it's visible.

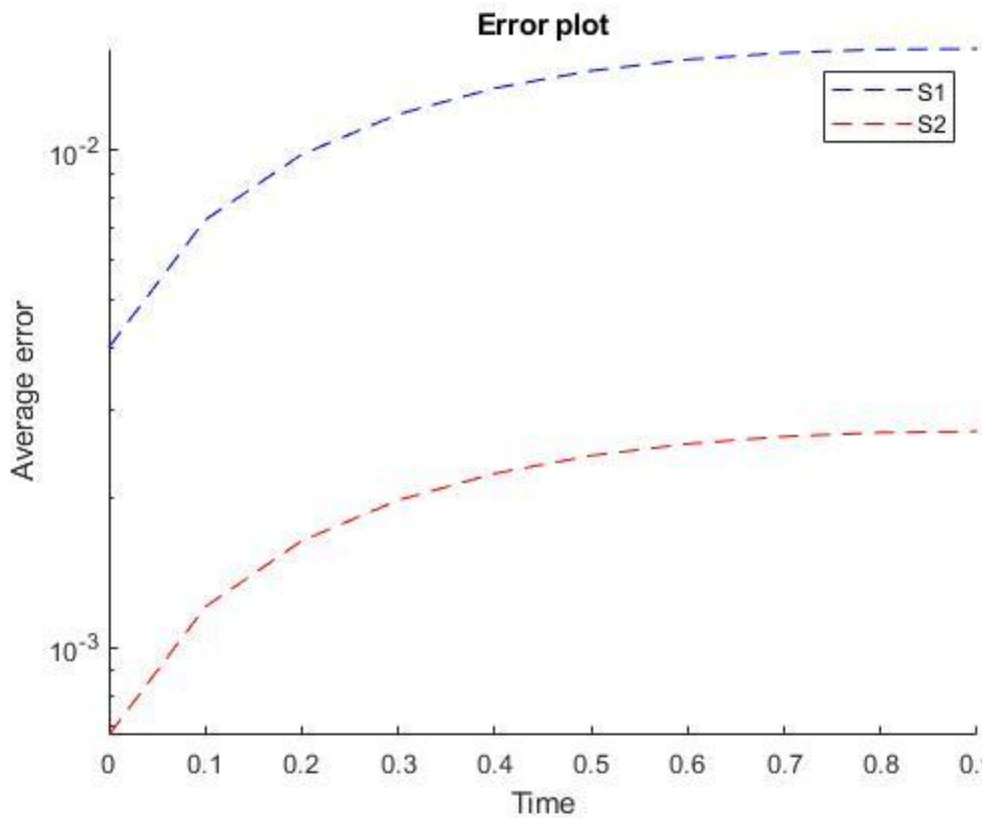

Figure 2:Error plot of S1 and S2.

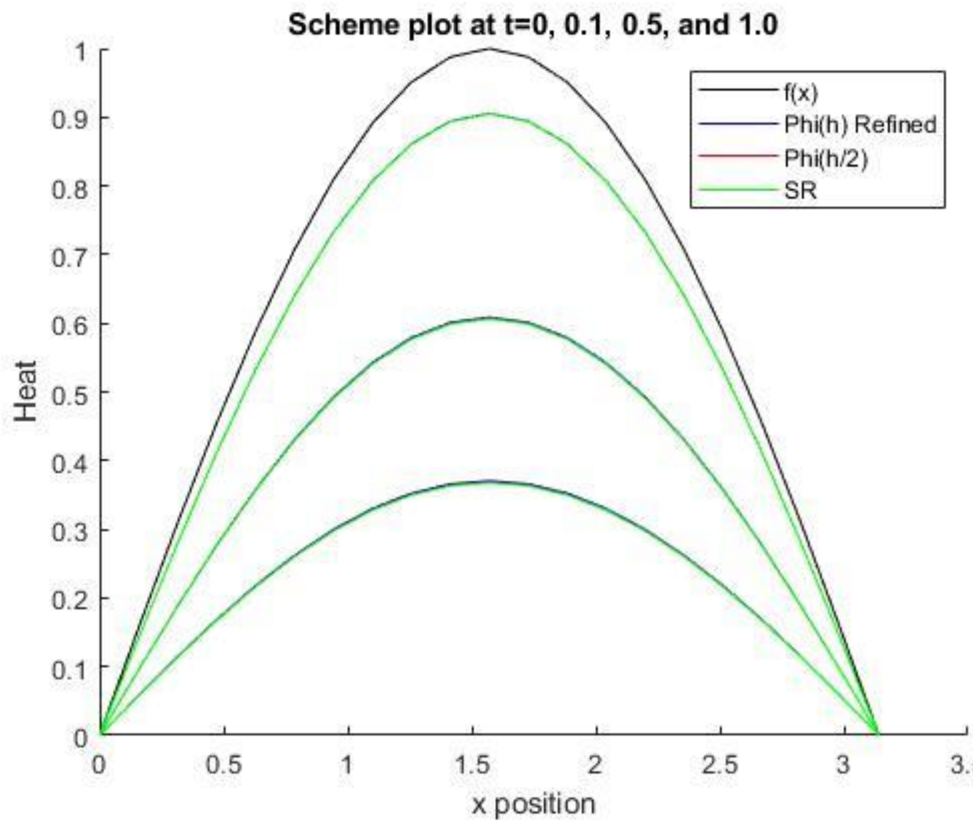

Figure 3: Plot of the refined Phi(h), Phi(h/2), and SR. Notice that the plots are all almost overlapping.

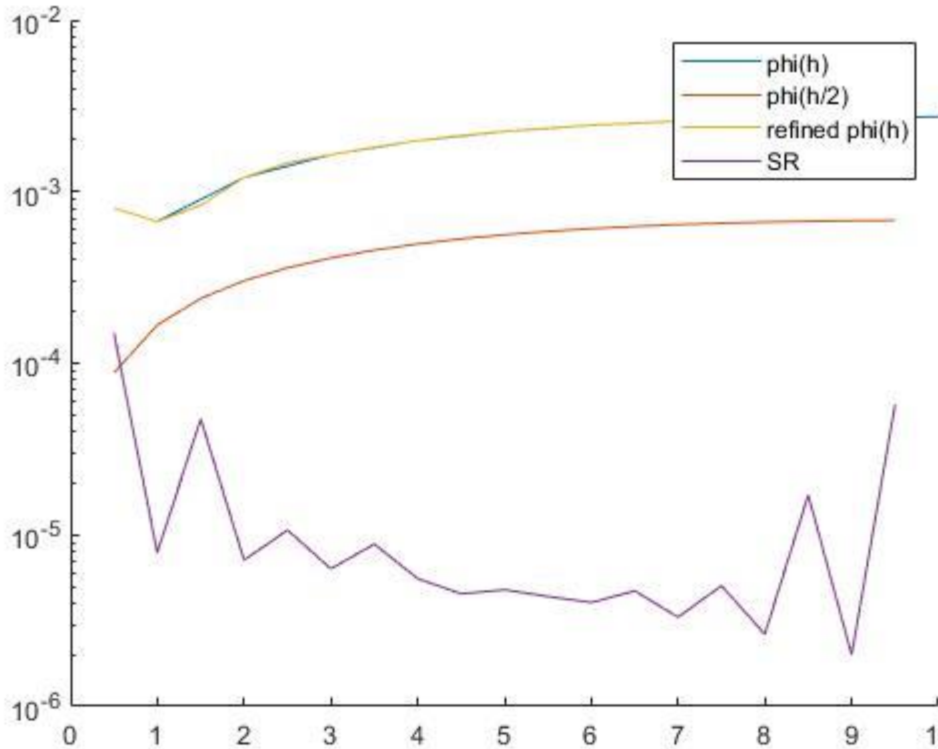

Figure 4: The degree of error for Phi(h), Phi(h/2), the refined Phi(h), and SR. The initial peak in SR is due to the scheme midpoint error being larger at dt. Notice the outlier in the beginning of the refined Phi(h). In this graph it's clear how accurate SR is.

Conclusion

In this paper there are three estimation schemes for the heat equation, the first of $O(dx^2 + dt)$, the second of $O(dt^2)$, and the third (Richardson) of $O(dt^4)$. I can see it possible to reduce the error even further with repeated Richardson applications, and we can approach the exact solution taking $O(dt^{n+2})$ steps every iteration. Throughout this project I had a lot of problems and errors, and an annoying issue where my errors were much larger than they should have been. At the end of the project I resolved all my issues and had a working product. I enjoyed the semester with Dr. Islas, and learned a lot from the class.

Attached Programs

```
function PP1
    % The exact solution to the heat equation
    u = @(x,t) exp(-t).* sin(x);
    % u(x,0)
    f = @(x) sin(x);

    % Scheme parameters
```

```
    N = 10; L = pi; dx = L/N; dt = 0.1; tmax = 1/dt;

    % The initial value for the schemes
    U_0 = f(dx * (1:N-1));

    S1_U = generateS1(U_0,dx,dt,N,tmax);
    S2_U = generateS2(U_0,dx,dt,N,tmax);

    E1 = calculateError(S1_U,u,dx,dt,N,tmax);
    E2 = calculateError(S2_U,u,dx,dt,N,tmax);

    % Plotting
    % We need to plot for timeslots 0.1, 0.5 and 1
    chi = dx * (0:N);

    L1 = plot(chi,[0,U_0(1,:),0],'k');



    % Scheme plot
    figure(1);
    hold on;
    title('Scheme plot at t=0, 0.1, 0.5, and 1.0');
    ylabel('Heat');
    xlabel('x position');
    % S1
    L2 = plot(chi,[0,S1_U(0.1/dt,:),0],'b');
    plot(chi,[0,S1_U(0.5/dt,:),0],'b');
    plot(chi,[0,S1_U(1/dt,:),0],'b');

    % S2
    L3 = plot(chi,[0,S2_U(0.1/dt,:),0],'r');
    plot(chi,[0,S2_U(0.5/dt,:),0],'r');
    plot(chi,[0,S2_U(1/dt,:),0],'r');
    hold off;

    legend([L1,L2,L3],'f(x)','S1','S2');
    % Error plot
    figure(2);
    hold on;
    title('Error plot');
    ylabel('Average error');
    xlabel('Time');
    semilogy(dt * (0:tmax-1),E1, '--b');
    semilogy(dt * (0:tmax-1),E2, '--r');
    legend('S1','S2');
    set(gca,'yscale','log')
    hold off;
end
function PP2
    % The exact solution to the heat equation
    u = @(x,t) exp(-t).* sin(x);
```

```
% u(x,0)
f = @(x) sin(x);

% Scheme parameters
N = 10; L = pi; dx = L/N; dt = 0.1; tmax = 1/dt;

% The initial value for the schemes
U_0 = f(dx * (1:N-1));
U2_0 = f(dx/2 * (1:2*N-1));

% First derivation
Phih = generateS2(U_0,dx,dt,N,tmax);
Phih2 = generateS2(U2_0,dx/2,dt/2,2*N,2*tmax);
PhihR = coarseRefined(Phih,U_0,dx,dt,N,tmax);
SR = (4*Phih2-PhihR)/3;

% Errors
E1 = calculateError(Phih,u,dx,dt,N,tmax);
E2 = calculateError(Phih2,u,dx/2,dt/2,2*N,2*tmax-1);
E3 = calculateError(PhihR,u,dx/2,dt/2,2*N,2*tmax-1);
E4 = calculateError(SR,u,dx/2,dt/2,2*N,2*tmax-1);

figure(1);
hold on;
title('Scheme plot at t=0, 0.1, 0.5, and 1.0');
ylabel('Heat');
xlabel('x position');
chi = dx/2 * (0:2*N);
% Initial value
L1 = plot(chi,[0,U2_0(1,:),0],'k');
% PhiR(h)
L2 = plot(chi,[0,PhihR(0.2/dt,:),0],'b');
plot(chi,[0,PhihR(1/dt,:),0],'b');
plot(chi,[0,PhihR(2/dt,:),0],'b');
% Phi(h/2)
L3 = plot(chi,[0,Phih2(0.2/dt,:),0],'r');
plot(chi,[0,Phih2(1/dt,:),0],'r');
plot(chi,[0,Phih2(2/dt,:),0],'r');
% SR
L4 = plot(chi,[0,SR(0.2/dt,:),0],'g');
plot(chi,[0,SR(1/dt,:),0],'g');
plot(chi,[0,SR(2/dt,:),0],'g');
legend([L1,L2,L3,L4],'f(x)','Phi(h) Refined','Phi(h/2)','SR');
hold off;

% Error plot
figure(2);
hold on;
plot(1:tmax,E1);
plot((1:(2*tmax-1))/2,E2);
plot((1:(2*tmax-1))/2,E3);
plot((1:(2*tmax-1))/2,E4);
```

```
    legend('phi(h)','phi(h/2)','refined phi(h)','SR');
    set(gca,'yscale','log')

end
function S1_U = generateS1(S1_0,dx,dt,xmax,tmax)
    % The iterative matrix for generating S1
    A = eye(xmax-1) + dt/(dx^2)*(-2*eye(xmax-1)+diag(ones(xmax-2,1),1) +
diag(ones(xmax-2,1),-1));

    % Initilize S1_U
    S1_U = zeros(tmax,xmax-1);
    S1_U(1,:) = S1_0*A;

    for i = 2:tmax
        S1_U(i,:) = S1_U(i-1,:) * A;
    end
end
function S2_U = generateS2(S2_0,dx,dt,xmax,tmax)
    % The left hand side iterative matrix
    A_L = eye(xmax-1)-dt/(2*dx^2)*(-2*eye(xmax-1)+diag(ones(xmax-2,1),1)
+ diag(ones(xmax-2,1),-1));
    % The right hand side
    A_R = eye(xmax-1)+dt/(2*dx^2)*(-2*eye(xmax-1)+diag(ones(xmax-2,1),1)
+ diag(ones(xmax-2,1),-1));

    S2_U = zeros(tmax,xmax-1);
    matrix = [A_L,(S2_0*A_R)'];
    S2_U(1,:) = RNG(matrix);

    for i = 2:tmax
        matrix = [A_L,(S2_U(i-1,:)*A_R)'];
        S2_U(i,:) = RNG(matrix);
    end
end
function x = RNG(A)
%
% The forward elimination is attained by the following loops
%
[M N] = size(A);
%
for n=1:N-2
    for m=n+1:M
        A(m,:) = A(m,:) - A(m,n)/A(n,n)*A(n,:);
    end
end
% and the backward substitution by the following loop
x(M) = A(M,N)/A(M,M);
for m=M-1:-1:1
    x(m) = ( A(m,N) - sum( A(m,m+1:M).*x(m+1:M) ) )/A(m,m);
end
function E = calculateError(SU,u,dx,dt,xmax,tmax)
```

```
    % t is bounded between 0:tmax, but at t=0 the error is 0
    % x is bounded between 1:xmax-1

    % initializing
    E = zeros(1,tmax);

    for i = 1:tmax
        diff = u(dx*(1:xmax-1),(i)*dt) - SU(i,:);
        E(i) = norm(diff,inf);
    end
end
function S2R = coarseRefined(S2,S2_0,dx,dt,xmax,tmax)
    % Appending the base condition to spline between
    S2 = [S2_0;S2];
    [M N] = size(S2);

    % We need an intial condition between t=0 and dt
    for tau = 1:M
        S2R_0(tau,:) = splineMidpoints(dx*(0:xmax),[0,S2(tau,:),0]);
    end
    [M N] = size(S2R_0);

    for chi = 1:N
        S2R(:,chi) = splineMidpoints(dt*(0:tmax),S2R_0(:,chi)')';
    end
    S2R = S2R(2:end,:);
end
function S = cubicSpline(xn,yn)
    % construct the matrix for zn
    % z1 = zn+1 = 0
    hn = xn(2:end) - xn(1:end-1);
    wn = (yn(2:end) - yn(1:end-1))./hn;

    A = diag(hn(2:end-1),1) + diag(hn(2:end-1),-1) + 2.*(diag(hn(1:end-1)
+ hn(2:end)));
    A = [A,6.*(wn(2:end)-wn(1:end-1))'];
    zn = [0,RNG(A),0];

    an = (zn(2:end) - zn(1:end-1))./(6.*hn);
    bn = zn./2;
    cn = wn-hn./6.*(zn(2:end) + 2.*zn(1:end-1));
    S = [an',bn(1:end-1)',cn',yn(1:end-1)'];
end
function set = splineMidpoints(xn,yn)
    S = cubicSpline(xn,yn);
    set = [];

    % yn(1) == 0 if we're dealing with dx, we want to add the first
    % position if we're dealing with dt
    if (yn(1) ~= 0)
        set = [set,yn(1)];
    end
```

```matlab
    for alpha = 1:length(xn)-1
        % the average value, we want approximations of the midpoint from
        % the phi(h) estimation
        chi = (xn(alpha)+xn(alpha+1))/2;
        % x-xi
        delta = chi - xn(alpha);

        gamma = S(alpha,:) * [delta^3;delta^2;delta;1];

        % Inserting the non-edge values


        set = [set,gamma];
        if (alpha ~= length(xn)-1)
            set = [set,yn(alpha+1)];
        end
    end
    % yn(end) == 0 if we're dealing with dx, we want to add the first
    % position if we're dealing with dt
    if (yn(end) ~= 0)
        set = [set,yn(end)];
    end
end
```