Numerical Method Solutions to Partial Differential Equation

Timothy Holmes

Department of Mathematics, DePaul University

June 3, 2018

Abstract

Partial Differential Equations (PDEs) are complex problems that are hard to solve analytically. Numerical methods are used to allow for solving these problems faster and also solving problems that can't be done analytically. There are many different methods used to solve PDEs. The methods analyzed in this paper include an explicit method, implicit method, and the Crank-Nicolson method. The explicit method is an unstable method when the proper conditions are not satisfied. Both the implicit method and the Crank-Nicolson method are always stable when $D > 0$, $h > 0$, and $k > 0$. However, the implicit methods truncation error grows faster than the Crank-Nicolson method. The methods with the best run time are the explicit method, implicit method, and Crank-Nicolson method, respectively.

1 Introduction

Partial differential equations (PDEs) have a significantly important use over the span of many different fields were they tend to have many different applications. These partial differential equations can be used in engineering, physics, and finance, to name a few, in order to solve very hard problems that might otherwise be impossible to solve without them. There are two different ways PDEs can be solved for, analytically and numerically. One problem with an analytical methods is it can be very time consuming, where a computer can do it much faster with little error. Another problem is when the problem becomes complex enough the problem can no longer be solved using an analytical method, therefore, a numerical method is used to solve it. There are several different numerical methods for solving PDEs, each with their own unique characteristic and specific function that they are good for. However, this paper could't possible cover all of the different methods to solving PDEs, they are far too complex and lengthy. Therefore, this paper will specifically observe the Finite Difference Method. Using the Finite Difference Method, three different types of PDEs will be calculated; parabolic, hyperbolic, and elliptic. The Finite Differencing Method has three sub methods that are used to solve various types of PDEs.

The finite-differencing methods (FDM) is a numerical method that is used to solve PDEs in mathematics. The method is used to approximate differential equations approximating derivatives. The three sub-methods previously stated are; the explicit method that uses a forward differencing, the implicit method that uses backwards differencing, and the Crank-Nicolson method that uses central differencing.

2 method

To test these methods, a specific problem needs to be addressed. As previously stated, these methods are very popular in different disciplines. For the sake of simplicity, the heat equation will be the specific problem that is observed in this paper. The heat equation is given by,

$$u_t = Du_{xx}. \tag{1}$$

The subscripts x and t are independent variables. Moreover, x represents the temperature x measured along a one-dimensional rod, D is just a constant relating to the material of the rod, and t is time.

2.1 Explicit method

Some PDEs have specific boundary conditions that they need to satisfy. PDEs with boundary conditions most often show up in science and engineering problem, in this case, the heat equation is a PDE that needs to satisfy certain boundary conditions. If $u_{xx} = 0$ then using the forward differencing method, the first derivative can be approximated by

$$u_i' \approx \frac{u_{i+1} - u_i}{2h}. \tag{2}$$

The second derivative can be approximated by

$$u_i'' \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}, \tag{3}$$

where $h = x_{i+1} - x_i$. Now for the heat equation, the spatial part, u_{xx}, has already been solved for. Therefore, the time part u_t need to be found. The first derivative of the forward differencing method gives

$$u_t(x,t) \approx \frac{1}{k}(u(x,t+k) - u(x,t)). \tag{4}$$

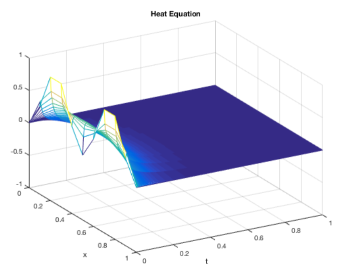
Substituting this equation into the heat equation ultimately gives us

$$u_{i,j+1} = (1-2s)u_{i,j} + s(u_{i+1,j} + u_{i-1,j}) \tag{5}$$

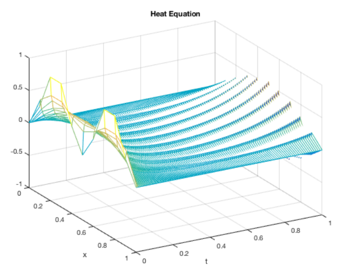
where

$$s = \frac{Dk}{h^2}.$$

The models below were generated by Matlab code that used the equations above.



(a) $k = 0.0040$



(b) $k > 0.005$

Figure 1: Initial conditions are the same while the k values vary. figure (a) shows an example when the method is stable, figure (b) is and example when k is too large and is unstable.

The explicit method is confined to small k steps since if k wasn't small the error would grow to large. When the error grows to large the model generated is unreadable and no information is able to be interpreted from the graph. Therefore, this method is stable when $D > 0$ and $s < 1/2$. Stability of this method completely depends on the step sizes in choice. Altering the algorithm to record the time taken after initial conditions yields 0.0343875 seconds. However, this method will be quicker than other methods since this method reaches its limit much quicker.

2.2 Implicit method

The implicit method is called *implicit* since it approximates derivatives at the next iterative step for time. Therefore, instead of approximating at the current time step t_k this method approximates at the next time step t_{k+1}. For this method, u_{xx} stays the same but u_t changes. Using the backwards-difference formula u_t becomes,

$$u_t = \frac{1}{k}(u(x,t) - u(x,t-k)) + \frac{k}{2}u_{tt}(x,c_0) \tag{6}$$

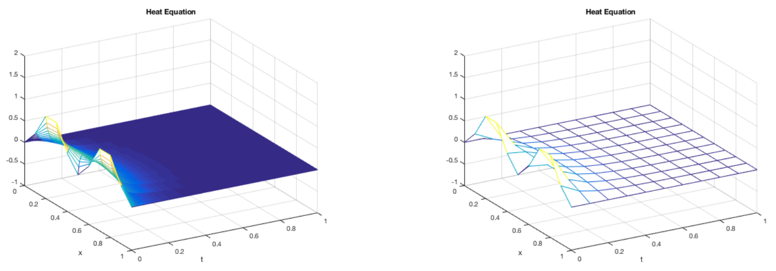
where $t - k < c_0 < t$. Substituting the difference formula into the heat equation we eventually get

$$su_{i+1,j} + (1+2s)u_{ij} - su_{i-1,j} = u_{i,j-1} \tag{7}$$

where

$$s = \frac{Dk}{h^2}.$$

Using the equations above to write an algorithm in Matlab generates the two figures below. For the small k the figure generated had a lot more detail in it. However, for the case when k was large, the individual grids could be pointed out and is less detailed.



(a) Small k value (b) Large k value

Figure 2: Overview of testing out different k values for the implicit method. Varying the k value by entering extreme values shows that this method is stable.

Unlike the previous method, the implicit method remains stable for all values of k and as long as the condition $D > 0$. While it is true that this method is stable, it is also true that the error increases as the value of k increases. The error grows by $\mathcal{O}(k) + \mathcal{O}(h^2) \approx \mathcal{O}(k)$. This method take a little longer to run than the previous method. For the same conditions it took this algorithm 5.9365 seconds to generate a figure.

2.3 Crank-Nicolson method

The Crank-Nicolson method is the combination of the two previous methods, explicit method and implicit method. u_t becomes

$$\frac{1}{k}(u_{ij} - w_{i,j-1}). \tag{8}$$

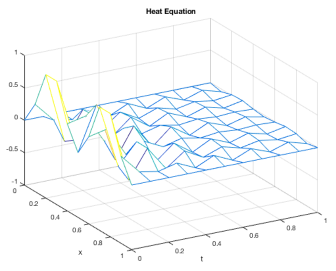
Then the u_{xx} becomes

$$\frac{1}{2}\left(\frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{h^2}\right) + \frac{1}{2}\left(\frac{u_{i+1,j-1} - 2u_{i,j-1} + u_{i-1,j-1}}{h^2}\right). \tag{9}$$

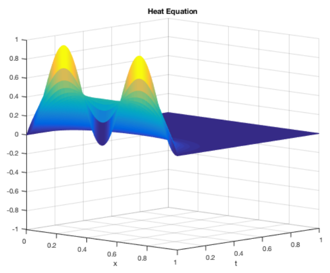
This can then be reduces to

$$2u_{ij} - 2u_{i,j-1} = s[u_{i+1,j} - 2u_{ij} + u_{i-1,j} + u_{i+1,j-1} - 2u_{i,j-1} + u_{i-1,j-1}]. \quad (10)$$

Combining these two methods and writing some code in Matlab generates the two figures below.



(a) Small space steps



(b) Large space steps

Figure 3: Figure (a) shows the Crank-Nicolson Method with low space steps and figure (b) shows the method with high space steps.

This method, like the last one is stable when the conditions $D > 0$, $h > 0$, and $k > 0$ are meet. The truncation error for this method is given by $\mathcal{O}(k^2)+\mathcal{O}(h^2)$. The error for this method always results in a lower error than the implicit method. This method is also more accurate than the explicit method. Therefore, out of all the previous methods, this one is the most sought after. However, running this method in Matlab did take a considerable amount of time with the same initial conditions. It took 10.8403 seconds for this method to generate a figure.

3 Conclusion

In conclusion, the explicit method, implicit method, and Crank-Nicolson Method are all great methods for various different things. The explicit method is great to use when there is not need for a large time step, k. When a large k isn't needed this algorithm is going to give you the quickest result with little error. However, that is not always the case, and the accuracy of this method isn't going to be the greatest since it has a small k value. The implicit method is the best option if you want to solve a PDE that needs a large k and a median time to calculate. There are often times when the computation in thought is going to be a lot more power intensive. Moreover, saving time on the calculation may be worth the error, this is what this method is great for. However, if time is not an issue and error might be the best option is the Crank-Nicolson Method. This method is the combination of the two previous methods. It has the lowest error and the time step can be large, as seen previously. This method did take up the most time of all the computational methods. These are all great methods, but depending on what the parameters and expected outputs the importance of each method varies.

References

Numerical Analysis by Timothy Sauer

Appendix

```
% Program 8.1 Forward difference method for heat equation
% input: space interval [xl,xr], time interval [yb,yt],
% number of space steps M, number of time steps N
% k = yt-yb/N, h = xr-xl/M
% Example usage: w=heatfd(0,1,0,1,10,250)
%
function heatfd(xl,xr,yb,yt,M,N)
tic
f=@(x) sin(2*pi*x).^2;
l=@(t) 0*t;
r=@(t) 0*t;
D=1; % diffusion coefficient
h=(xr-xl)/M; k=(yt-yb)/N; m=M-1; n=N;
sigma=D*k/(h*h);
a=diag(1-2*sigma*ones(m,1))+diag(sigma*ones(m-1,1),1);
a=a+diag(sigma*ones(m-1,1),-1); % define matrix a
lside=l(yb+(0:n)*k); rside=r(yb+(0:n)*k);
w(:,1)=f(xl+(1:m)*h)'; % initial conditions
for j=1:n
w(:,j+1)=a*w(:,j)+sigma*[lside(j);zeros(m-2,1);rside(j)];
end
w=[lside;w;rside]; % attach boundary conds
x=(0:m+1)*h;t=(0:n)*k;
mesh(x,t,w') % 3-D plot of solution w
title('Heat Equation')
xlabel('x')
ylabel('t')
view(60,30);axis([xl xr yb yt -1 1])
time = toc;
fprintf('time %g \n',time)
end
```

```
% Program 8.2 Backward difference method for heat
    equation
% input: space interval [xl,xr], time interval [yb,yt],
% number of space steps M, number of time steps N
% output: solution w
% Example usage: w=heatbd(0,1,0,1,10,10)
function heatbd(xl,xr,yb,yt,M,N)
tic
f=@(x) sin(2*pi*x).^2;
l=@(t) 0*t;
r=@(t) 0*t;
D=1; % diffusion coefficient
h=(xr-xl)/M; k=(yt-yb)/N; m=M-1; n=N;
sigma=D*k/(h*h);
a=diag(1+2*sigma*ones(m,1))+diag(-sigma*ones(m-1,1),1);
a=a+diag(-sigma*ones(m-1,1),-1); % define matrix a
lside=l(yb+(0:n)*k); rside=r(yb+(0:n)*k);
w(:,1)=f(xl+(1:m)*h)'; % initial conditions
for j=1:n
w(:,j+1)=a\(w(:,j)+sigma*[lside(j);zeros(m-2,1);rside(j)]);
end
w=[lside;w;rside]; % attach boundary conds
x=(0:m+1)*h;t=(0:n)*k;
mesh(x,t,w') % 3-D plot of solution w
title('Heat Equation')
xlabel('x')
ylabel('t')
view(60,30);axis([xl xr yb yt -1 2])
time = toc;
fprintf('time %g \n',time)
end
```

```
% Program 8.4 Crank-Nicolson method
% with Dirichlet boundary conditions
% input: space interval [xl,xr], time interval [yb,yt],
% number of space steps M, number of time steps N
% output: solution w
% Example usage: w=crank(0,1,0,1,10,10)
function w=crank(xl,xr,yb,yt,M,N)
f=@(x) sin(2*pi*x).^2;
l=@(t) 0*t;
r=@(t) 0*t;
D=1; % diffusion coefficient
h=(xr-xl)/M;k=(yt-yb)/N; % step sizes
sigma=D*k/(h*h); m=M-1; n=N;
a=diag(2+2*sigma*ones(m,1))+diag(-sigma*ones(m-1,1),1);
a=a+diag(-sigma*ones(m-1,1),-1); % define tridiagonal matrix a
```

```
b=diag(2-2*sigma*ones(m,1))+diag(sigma*ones(m-1,1),1);
b=b+diag(sigma*ones(m-1,1),-1); % define tridiagonal matrix b
lside=l(yb+(0:n)*k); rside=r(yb+(0:n)*k);
w(:,1)=f(xl+(1:m)*h)'; % initial conditions
for j=1:n
sides=[lside(j)+lside(j+1);zeros(m-2,1);rside(j)+rside(j+1)];
w(:,j+1)=a\(b*w(:,j)+sigma*sides);
end
w=[lside;w;rside];
x=xl+(0:M)*h;t=yb+(0:N)*k;
mesh(x,t,w');
title('Heat Equation')
xlabel('x')
ylabel('t')
view (60,30); axis([xl xr yb yt -1 1])
time = toc;
fprintf('time %g \n',time)
end
```