Modelling and Simulation
EEEN 30150

Electronic, Electrical and Communications Engineering

Dr Paul Curran

Room 145, Engineering and Materials' Science Centre.

paul.curran@ucd.ie

+353-1-7161846

Overview of Topics

Solution of equations by iteration.

Optimisation.

Solution of system of linear equations.

Solution of ordinary differential equations: initial value problems.

Numerical Quadrature.

Solution of ordinary differential equations: boundary value problems.

Solution of partial differential equations.

Section 7: Partial Differential Equations

- Method: Finite Differences

Workload: 6 lectures + 1 laboratory + Autonomous learning

The Wave Equation

One of the most important scientific problems in terms of its historical significance in the development of science and mathematics is concerned with the transverse vibrations of a narrow gauge string of length l fixed at both ends. The equation describing these vibrations, $y(x,t)$, in an idealised scenario (assuming these vibrations to be of small amplitude for example) is called the *wave equation*.

$$\frac{\partial^2 y}{\partial x^2} = \frac{\rho A}{T_0}\frac{\partial^2 y}{\partial t^2}$$

The Wave Equation

The equation is more commonly written as.

$$\frac{\partial^2 y}{\partial x^2} = \frac{1}{v^2}\frac{\partial^2 y}{\partial t^2} \qquad \frac{1}{v^2} = \frac{\rho A}{T_0}$$

The parameter ρA is the mass per unit length. The parameter T_0 is the tension. The ratio $\rho A/T_0$ is equated to $1/v^2$ because the resulting parameter v (also commonly denoted c) it transpires has units of velocity. Of the several solutions to the equation available that generally credited to d'Alembert most clearly explains the name, wave equation, and most clearly justifies the introduction of the parameter v.

Normalisation

It is commonly possible to introduce some form of scaling such that an equation takes on a normalised form. In the case of the wave equation a little thought shows that the equation itself is suggesting that our choice of units for time is inappropriate (or rather not optimal). If we time scale, letting

$$\tau = vt$$

we obtain the wave equation again, but now with simpler parameters:

$$\frac{\partial^2 y}{\partial x^2} = \frac{\partial^2 y}{\partial \tau^2}$$

The Wave Equation

When the string is fixed at both ends we must impose the boundary conditions: $y(0,\tau)=y(l,\tau)=0, \quad \tau \geq 0$

We must also generally specify the initial state of the string, i.e. its initial transverse deflection $y(x,0)$ and the initial velocity of this deflection $y'(x,0)$. Many stringed instruments are either bowed, strummed or plucked in such a manner that just before release the string may be considered to be momentarily unmoving. This justifies the following initial condition:

$$y'(x,0)=\frac{\partial}{\partial t}y(x,0)=v\frac{\partial}{\partial \tau}y(x,0)=0 \quad , \; 0 \leq x \leq l$$

The Wave Equation

The resulting normalised problem to be solved is a second order linear partial differential equation with simple coefficients subject to boundary conditions and initial conditions:

$$\frac{\partial^2 y}{\partial x^2}=\frac{\partial^2 y}{\partial \tau^2}$$

$$y(0,\tau)=y(l,\tau)=0, \quad \tau \geq 0,$$
$$y(x,0)=f(x) \quad , \; 0 \leq x \leq l,$$
$$\frac{\partial}{\partial \tau}y(x,0)=0 \quad , \; 0 \leq x \leq l$$

$f(x)$ is a function of x defined for $0 \leq x \leq l$ which describes the initial deflection of the string.

Method of Finite differences

The method of finite differences for PDEs is, broadly speaking, the same as for boundary value problems for ODEs. Indeed this is the main reason why the latter was included in the module. On this occasion however it is necessary to break up the range both of positions and times. The range of values of x (positions) to be considered will be assumed finite, being $[a,b]$ (or $[0,l]$ in the case of the wave equation). Break up this range into N equal sub-intervals so that each sub-interval has length $h = (b-a)/N$. The range of values of τ (times) will in general not be finite. Nevertheless we must determine a time step-size, k, and seek to evaluate the solution at the positions $x_i = a+ih = ih$ (in this case) and the times $\tau_j = jk$ only.

Method of Finite differences

The critical idea is to apply similar approximations (the central differences) as we used in the case of the boundary value problems for ODEs.
Accordingly let: $y_{ij} = y(a+ih, jk) = y(ih, jk)$

$$\left(\frac{\partial y}{\partial x}\right)_{ij} = \frac{y_{i+1,j} - y_{i-1,j}}{2h} \qquad \left(\frac{\partial y}{\partial \tau}\right)_{ij} = \frac{y_{i,j+1} - y_{i,j-1}}{2k}$$

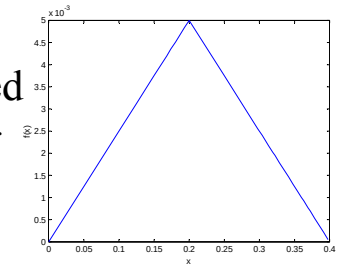
$$\left(\frac{\partial^2 y}{\partial x^2}\right)_{ij} = \frac{y_{i+1,j} - 2y_{i,j} + y_{i-1,j}}{h^2} \qquad \left(\frac{\partial^2 y}{\partial \tau^2}\right)_{ij} = \frac{y_{i,j+1} - 2y_{i,j} + y_{i,j-1}}{k^2}$$

The Wave Equation

In this first case let us take a string of length $l = 40$ cm = 0.4 m. Let us pick as the initial deflection a rather idealised pluck:

$$f(x) = \begin{cases} 0.025x & \text{for } 0 \le x \le 0.2 \\ 0.01 - 0.025x & \text{for } 0.2 \le x \le 0.4 \end{cases}$$

Obviously the string cannot be deformed in this way. But we are dealing with an idealised model, the string is not actually described by the wave equation, so another layer of idealisation hardly matters.



The Wave Equation

Let us pick $h = l/10 = 0.04$ m so that at each time jk we will have an approximate solution for 11 uniformly-spaced values of x including the mid-point. The unknown values for each special time $\tau = jk$ are therefore:

$$y_{0,j} = y(0, jk),\ y_{1,j} = y\left(\tfrac{l}{10}, jk\right), \ldots, y_{9,j} = y\left(\tfrac{9l}{10}, jk\right),\ y_{10,j} = y(l, jk)$$

The task is to set up equations which permit us to determine each of these 11 unkown values of the deflection y at each of the special normalised times.

The Wave Equation

The initial condition on y and the boundary conditions are:

$$y(0,\tau) = y(l,\tau) = 0, \quad \tau \geq 0,$$
$$y(x,0) = f(x) \quad , \; 0 \leq x \leq l$$

and these give us the following simple results:

$$y_{0,j} = y_{10,j} = 0 \quad \text{for all} \quad j$$
$$y_{i,0} = f(ih) \quad \text{for all} \quad i$$

$$y_{0,0} = 0,\ y_{1,0} = 0.001,\ y_{2,0} = 0.002,\ y_{3,0} = 0.003,$$
$$y_{4,0} = 0.004,\ y_{5,0} = 0.005,\ y_{6,0} = 0.004,\ y_{7,0} = 0.003,$$
$$y_{8,0} = 0.002,\ y_{9,0} = 0.001,\ y_{10,0} = 0$$

The Wave Equation

The normalised wave equation itself gives:

$$\frac{\partial^2 y}{\partial x^2} = \frac{\partial^2 y}{\partial \tau^2} \qquad \left(\frac{y_{i+1,j} - 2y_{i,j} + y_{i-1,j}}{h^2}\right) = \left(\frac{y_{i,j+1} - 2y_{i,j} + y_{i,j-1}}{k^2}\right)$$

$$y_{i,j+1} = 2y_{i,j} - y_{i,j-1} + \left(\frac{k^2}{h^2}\right)\left(y_{i+1,j} - 2y_{i,j} + y_{i-1,j}\right)$$

$$y^{(j)} = \begin{bmatrix} y_{1,j} \\ y_{2,j} \\ \vdots \\ y_{8,j} \\ y_{9,j} \end{bmatrix}$$

This is the set of sample values at time jk which are *not know*. Of course $y_{0,j}$ and $y_{10,j}$ are known from the boundary conditions.

The Wave Equation

Let $\sigma^2 = \dfrac{k^2}{h^2}$ $\qquad y_{i,j+1} = 2y_{i,j} - y_{i,j-1} + \sigma^2\left(y_{i+1,j} - 2y_{i,j} + y_{i-1,j}\right)$

These equations may be very neatly expressed as:

$$y^{(j+1)} = Ay^{(j)} - y^{(j-1)} + b^{(j)}$$

$$A = \begin{bmatrix} 2(1-\sigma^2) & \sigma^2 & 0 & \cdots & 0 & 0 \\ \sigma^2 & 2(1-\sigma^2) & \sigma^2 & \cdots & 0 & 0 \\ 0 & \sigma^2 & 2(1-\sigma^2) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2(1-\sigma^2) & \sigma^2 \\ 0 & 0 & 0 & \cdots & \sigma^2 & 2(1-\sigma^2) \end{bmatrix} \qquad b^{(j)} = \begin{bmatrix} \sigma^2 y_{0,j} \\ 0 \\ 0 \\ \vdots \\ 0 \\ \sigma^2 y_{10,j} \end{bmatrix}$$

The Wave Equation

The boundary conditions give $b^{(j)} = 0$ for all j so that:

$$y^{(j+1)} = Ay^{(j)} - y^{(j-1)}$$

This is a *second order recursion*, meaning that the next value depends upon the *two* preceding values. A problem arises: the recursion can start to predict the answer for $j = 2$ onwards, once it is known for $j = 0$ and $j = 1$. The initial conditions give it for $j = 0$, but how are we to determine it for $j = 1$ and thereby be able to start the recursion?

The Wave Equation

Of course there is one more initial (or $\tau = 0$ boundary) condition that we have yet to impose, namely:

$$\frac{\partial}{\partial \tau} y(x,0) = 0 \quad , \; 0 \le x \le l$$

The key idea is to employ this condition to obtain a credible approximation for the answer for $j = 1$ and thereby start the recursion. We could employ the *forward difference* approximation to obtain:

$$\frac{y(ih,k) - y(ih,0)}{k} = 0 \quad \text{for all } i \qquad y_{i,1} = y_{i,0}$$

The Wave Equation

This proves to be hopeless, giving results which are in general highly inaccurate. Essentially the problem is that the forward difference approximation employed for the first partial derivative w.r.t. τ has a huge error of order $O(k)$ whereas the central difference approximations, within the same numerical scheme, have more acceptable errors of order $O(h^2)$ and $O(k^2)$ respectively.

The Wave Equation

There is a standard, clever trick for acquiring an improved $O(k^2)$ approximation in this case:

$$y(x_i,k) = y(x_i,0) + k\frac{\partial y}{\partial \tau}(x_i,0) + \frac{k^2}{2}\frac{\partial^2 y}{\partial \tau^2}(x_i,0) + \mathrm{O}(k^3)$$

The wave equation itself gives:

$$\frac{\partial^2 y}{\partial \tau^2}(x_i,0) = \frac{\partial^2 y}{\partial x^2}(x_i,0) = \frac{y_{i+1,0} - 2y_{i,0} + y_{i-1,0}}{h^2} + O(h^2)$$

$$y_{i,1} = y_{i,0} + k\frac{\partial y}{\partial \tau}(x_i,0) + \frac{k^2}{2}\left(\left(\frac{y_{i+1,0} - 2y_{i,0} + y_{i-1,0}}{h^2}\right) + O(h^2)\right) + O(k^3)$$

The Wave Equation

$\sigma^2 = \frac{k^2}{h^2}$

$$\frac{\partial y}{\partial \tau}(x_i,0) = \frac{y_{i,1} - y_{i,0} - \frac{\sigma^2}{2}(y_{i+1,0} - 2y_{i,0} + y_{i-1,0})}{k} + O(k^2)$$

The approximation is therefore:

$$\frac{\partial y}{\partial \tau}(x_i,0) = \frac{y_{i,1} - y_{i,0} - \frac{\sigma^2}{2}(y_{i+1,0} - 2y_{i,0} + y_{i-1,0})}{k}$$

as this is zero by the initial condition on the partial derivative we finally obtain

$$y_{i,1} = y_{i,0} + \frac{\sigma^2}{2}(y_{i+1,0} - 2y_{i,0} + y_{i-1,0})$$

The Wave Equation

$$y_{i,1} = y_{i,0} + \frac{\sigma^2}{2}\left(y_{i+1,0} - 2y_{i,0} + y_{i-1,0}\right)$$

These equations also may be very neatly expressed as: $y^{(1)} = Cy^{(0)} + q$

$$C = \begin{bmatrix} 1-\sigma^2 & \frac{\sigma^2}{2} & 0 & \cdots & 0 \\ \frac{\sigma^2}{2} & 1-\sigma^2 & \frac{\sigma^2}{2} & \cdots & 0 \\ 0 & \frac{\sigma^2}{2} & 1-\sigma^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1-\sigma^2 \end{bmatrix}, \quad q = \begin{bmatrix} \frac{\sigma^2}{2} y_{0,0} \\ 0 \\ 0 \\ \vdots \\ \frac{\sigma^2}{2} y_{10,0} \end{bmatrix}$$

and again the simple boundary conditions yield that the vector q is zero giving: $y^{(1)} = Cy^{(0)}$

The Wave Equation

I should note that the matrices A and C are related by $A = 2C$. Obviously this is a rather useful observation. The reason for this happy coincidence becomes fairly clear when we consider the two recursions:

$$y_{i,j+1} = 2y_{i,j} + \sigma^2\left(y_{i+1,j} - 2y_{i,j} + y_{i-1,j}\right) - y_{i,j-1}$$

$$y_{i,1} = y_{i,0} + \frac{\sigma^2}{2}\left(y_{i+1,0} - 2y_{i,0} + y_{i-1,0}\right)$$

The Wave Equation

There is of course one outstanding issue before we can begin to solve, we have not chosen k. To make an informed choice of k can in general be a rather difficult problem, but there is a result which we will come to later and which in practice gives an upper bound on the acceptable value. As we do not wish to investigate this matter yet let us just take $k = 0.01$ for this example, a value which I happen to know is acceptable.

The Wave Equation

$$y^{(1)} = Cy^{(0)} \qquad \sigma^2 = \frac{k^2}{h^2} = \left(\frac{0.01}{0.04}\right)^2 = \frac{1}{16}$$

$$C = \begin{bmatrix} \frac{15}{16} & \frac{1}{32} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{32} & \frac{15}{16} & \frac{1}{32} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{32} & \frac{15}{16} & \frac{1}{32} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{32} & \frac{15}{16} & \frac{1}{32} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{32} & \frac{15}{16} & \frac{1}{32} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{32} & \frac{15}{16} & \frac{1}{32} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{32} & \frac{15}{16} & \frac{1}{32} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{32} & \frac{15}{16} & \frac{1}{32} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{32} & \frac{15}{16} \end{bmatrix} \qquad y^{(0)} = \begin{bmatrix} 0.001 \\ 0.002 \\ 0.003 \\ 0.004 \\ 0.005 \\ 0.004 \\ 0.003 \\ 0.002 \\ 0.001 \end{bmatrix}$$

Matlab: The Wave Equation

$$y^{(1)} = Cy^{(0)} \qquad y^{(j+1)} = Ay^{(j)} - y^{(j-1)}$$

Matrices A and C are both sparse with each having less than 31% of their elements being non-zero. As it happens they are also both tridiagonal, symmetric. Accordingly we should store them in Matlab as sparse matrices.

```
>> N = 10, len = 0.4    set parameters N and len = length l
>> h = (len-0)/N, k = 0.01      set parameters h and k
>> e = ones(N-1,1);
>> s2 = (k^2)/(h^2);            set parameter s2 = σ²
>> A = spdiags([s2*e 2*(1-s2)*e s2*e], [-1 0 1], N-1, N-1);
>> C = spdiags([(s2/2)*e (1-s2)*e (s2/2)*e], [-1 0 1], N-1, N-1);
```

Note placing of several assignments on one line.

The Wave Equation

I note that I have opted for a rather complicated method of generating matrices A and C. Actually I would be better placed just generating one of these matrices in this somewhat explicit manner and then employing the equation $A = 2C$ to generate the second.

The Wave Equation

$$y^{(1)} = Cy^{(0)} \qquad y^{(j+1)} = Ay^{(j)} - y^{(j-1)}$$

We will need to store two vectors, $y^{(j-1)}$ and $y^{(j)}$, in order to implement the second order recursion. I choose to create two Matlab variables, **yprev** and **ycurrent**, to store these.

```
>> f = [0.001:0.001:0.004]'; , yprev = [f; 0.005; flipud(f)];   initialise vector yprev as y(0)

>> ycurrent = C*yprev;          initialise vector ycurrent as y(1)

>> ytemp = (A*ycurrent )– yprev;     find next y and store in temporary variable ytemp

>> yprev = ycurrent;   update yprev

>> ycurrent = ytemp;  update ycurrent
```

Note transpose ' to ensure column vector and flip up down.

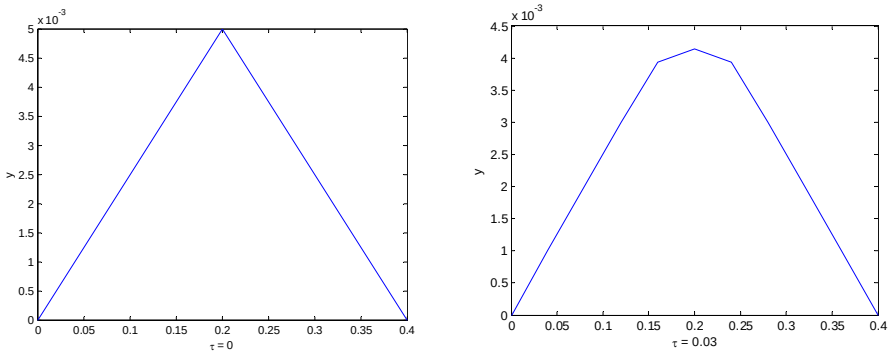
The Wave Equation

$$y^{(1)} = Cy^{(0)} \qquad y^{(j+1)} = Ay^{(j)} - y^{(j-1)}$$

$$y^{(0)} = \begin{bmatrix} 0.001 \\ 0.002 \\ 0.003 \\ 0.004 \\ 0.005 \\ 0.004 \\ 0.003 \\ 0.002 \\ 0.001 \end{bmatrix} \quad y^{(1)} = \begin{bmatrix} 0.0010 \\ 0.0020 \\ 0.0030 \\ 0.0040 \\ 0.0049 \\ 0.0040 \\ 0.0030 \\ 0.0020 \\ 0.0010 \end{bmatrix} \quad y^{(2)} = \begin{bmatrix} 0.0010 \\ 0.0020 \\ 0.0030 \\ 0.0040 \\ 0.0048 \\ 0.0040 \\ 0.0030 \\ 0.0020 \\ 0.0010 \end{bmatrix} \quad y^{(3)} = \begin{bmatrix} 0.0010 \\ 0.0020 \\ 0.0030 \\ 0.0040 \\ 0.0045 \\ 0.0040 \\ 0.0030 \\ 0.0020 \\ 0.0010 \end{bmatrix}$$

The Wave Equation

We have a means of solving the wave equation but the results as we have obtained them (namely lists of numbers) are not readily interpreted. It is more common to present the results as plots for various values of time.



The Wave Equation

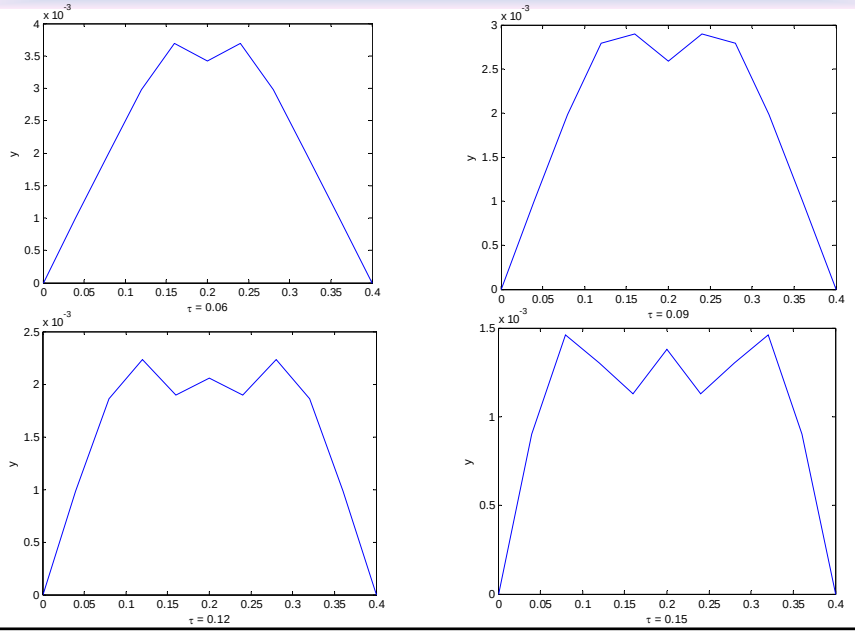
```
>> plot([0:len/N:len],[0;ycurrent;0])  plot current y vs x. Note addition of boundary values (both zero) to vector ycurrent.

>> xlabel('\tau = ?')     \tau produces symbol τ in x-label. Must enter the appropriate value for each plot.

>> ylabel('y')
```

The somewhat subtle point here is to recall that the vector **ycurrent** is a column vector which contains the 9 approximate solutions at the current time for the 9 *interior points* or *nodes*. To obtain the 11 approximate solutions at the current time for the 11 points we must add the known boundary values (both zero) to each end of **ycurrent**.

The Wave Equation



The Wave Equation

We see the gradual transformation of the original deflection but there are clearly at least three identifiable issues with this presentation of the data. Firstly the default plot command is joining data points by line segments and it becomes clear rather quickly that with only 11 data points per time step this is giving a rather poor sense of the actual solution. Secondly the default plot command is choosing the scaling of the vertical axis to suit the range of the data. This means that the axis scale changes from plot to plot so that one must take care in comparing the plots. Thirdly, it is really boring.

The Wave Equation

The first two issues are simply resolved. The x-range for every plot is [0,0.4]. The y-range for the first plot is [0,0.005] and is less for all of the other plots considered.

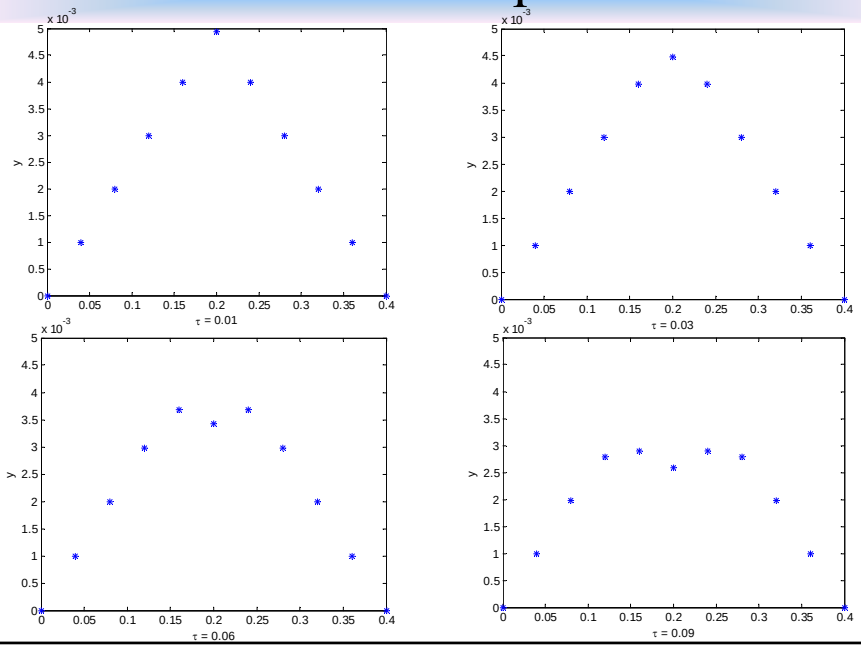
```
>> plot([0:len/N:len],[0;ycurrent;0],'*')        plot current y vs x marking data points with asterisks but not joining by line segments.

>> axis([0 0.4 0 0.005]          set x-range to [0,0.4] and y-range to [0,0.005] on plot

>> xlabel('\tau = ?')     \tau produces symbol τ in x-label. Must enter the appropriate value for each plot.

>> ylabel('y')
```


The Wave Equation



The Wave Equation

To deal with the third issue one may create a Matlab movie using the Matlab command **getframe**.

```
>> f = [0.001:0.001:0.004]'; , yprev = [f; 0.005; flipud(f)];
>> ycurrent = C*yprev;
>> plot([0:len/N:len],[0;yprev;0])
>> axis([0 len -0.02 0.02]      y will not remain positive, must adjust axis to permit negative values.
>> M(1) = getframe;   place plot as first frame in movie.
>> plot([0:len/N:len],[0;ycurrent;0])
>> axis([0 len -0.02 0.02]
>> M(2) = getframe;   place plot as second frame in movie
```


The Wave Equation

We may now use a **for** loop and the **getframe command** to generate another 148 frames and having created the movie the Matlab command **movie** permits it to be played.

```
>> for j = 3:150
ytemp = (A*ycurrent)-yprev;
yprev = ycurrent, ycurrent = ytemp;
plot([0:len/N:len],[0;ycurrent;0])
axis([0 len -0.02 0.02]
M(j) = getframe;        place plot as j^th frame in movie.
end
>> movie(M);     play movie once at 12 frames per second default.
```

The Wave Equation

If you wish to play the movie using some other package (e.g. Powerpoint) it must be converted to a more universal format. As we saw recent Matlab versions permit conversion to Audio Video Interleave (.avi) format using the **movie2avi** command.

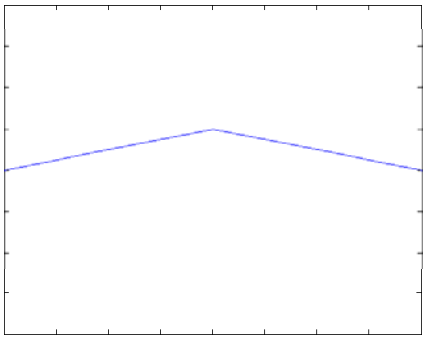
```
>> movie2avi(M,'WaveEquation1.avi')        convert Matlab movie to .avi format and store in file WaveEquation1.avi.

>> movie2avi(M,'WaveEquation1.avi','compression','none') convert movie to .avi but do not compress.
```

Recall Windows Media Player will commonly have problems playing the file if compression is used because the necessary codec (**co**mpression/**dec**ompression software) is unavailable. To avoid these issues convert without compression.

The Wave Equation

These commands lead to the following .avi file imported into Powerpoint:



There is an obvious problem here, the animation plays but the axes do not appear correctly. A minor adjustment to the code is required.

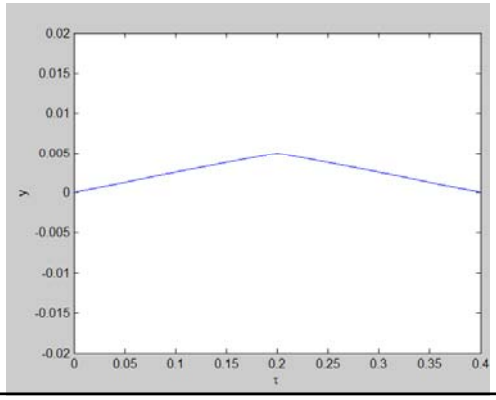
The Wave Equation

Before the first plot command create a *figure handle*:

```
>> handle = figure(1)          create figure handle.
```

Use the figure handle with each application of the getframe command:

```
>> M(j) = getframe(handle)
```



M-files

There are a reasonably large number of Matlab commands involved in generating the animated solution to the wave equation. When the number of commands starts to become large it starts to make sense to consider collecting these commands together, in short it is time to use a function **M-file**. Recall function M-files are collections of commands which can be grouped together and simply run in a batch. You can also take the opportunity to create a more general solution, allowing for example a different choice for N and k. Recall also that the **New M-File** button on the toolbar opens up the text editor for writing new M-files. Alternatively select **New** from the **File** menu. The **Open file** button opens the text editor to faciliate editing of an old M-file. Alternatively select **Open…** from the **File** menu.

Function M-files

The usual comments apply to the development of function M-files. You should be careful in your choice of names, picking names which at least somewhat describe what the function M-file does, rather than meaningless names like **f**. Likewise, just as regular Matlab functions have help files which are printed by means of the **help** command, you can and should write similar help files which will similarly be accessed for your functions. You should extensively comment your file (you should do this whatever code you are writing). If constants are to be set they should be set once at the very start of the file with clear comments indicating their meaning and the reason for the choice of value. You should write error checks at the start of the file to ensure acceptable input arguments and generate error messages if unacceptable values are presented.

Function M-files

I wrote a function M-file to collect all of the commands used for solving the wave equation with the special initial and boundary conditions considered above and generating a movie to facilitate animation of the solution. Here it is:

```
function  M = WaveEquation1(len,N,k,defmax)
%  WaveEquation1 (len,N,k,defmax): produces movie for animation of solution of transverse vibrations
%  of narrow gauge string fixed at both ends with normalized time, ideal
%  initial deflection (tent map) of maximum value defmax and zero initial velocity. len is
%  length of string, N+1 is number of x-points generated at each discrete
%  time, k is time step.

FrameNumber = 150; , Scale = 2;    % set a default number of frames generated and axis size
h = (len-0)/N;              % set x-step size
e = ones(N-1,1);
s2 = (k^2)/(h^2);          % set paramater s2 equal to sigma squared
```

Function M-files

```
A = spdiags([s2*e 2*(1-s2)*e s2*e],[-1 0 1],N-1,N-1);   % create sparse matrix A
C = spdiags([(s2/2)*e (1-s2)*e (s2/2)*e],[-1 0 1],N-1,N-1);   % create sparse matrix C
f = [h:h:(len/2)-h]';     % function M-file assumes tent map as initial deflection
yprev = defmax*[f;len/2;flipud(f)]/(len/2);    % initialise vector of N-1 interior points to tent map values
ycurrent = C*yprev;    % use special method for approximate solution after first time step
handle = figure(1);     % create figure handle to ensure proper treatment of axes
plot([0:len/N:len],[0;yprev;0]);
xlabel('x');                % Note change of label
ylabel('y');
axis([0 len -Scale*defmax Scale*defmax])
M(1) = getframe(handle);    % first frame equals initial condition
plot([0:len/N:len],[0;ycurrent;0]);
xlabel('x');
ylabel('y');
axis([0 len -Scale*defmax Scale*defmax])
M(2) = getframe(handle);    % second frame based on data acquired by special method
```


Function M-files

```
for count = 3:FrameNumber
   ytemp = (A*ycurrent)-yprev;   % find next y and store
   yprev = ycurrent;    % update previous y
   ycurrent = ytemp;   % update current y
   plot([0:len/N:len],[0;yprev;0]);
   xlabel('x');
   ylabel('y');
   axis([0 len -Scale*defmax Scale*defmax])
   M(count) = getframe(handle);
end
```

I saved this function M-file as WaveEquation1.m in my work folder. In this way I can achieve everything which we did above to generate movie M with the single command:

```
>> M = WaveEquation1(0.4,10,0.01,0.005)
```

Function M-files

Recall how the function M-file is created using the **function** command. Recall how the input arguments are passed to it and how the output arguments are passed from it. Recall how the help file is created, it comprises all of the text that follows the function definition line up to the first blank line. This text is preceeded by the % symbol so that Matlab knows that it comprises commentary not commands. The text editor automatically colours text following the % symbol and the % symbol itself green. It also automatically colours the commands **function**, **for** and **end** blue. Lastly it automatically indents the commands between the **for** and **end** commands. The help file for my function is available as for any other Matlab function:

```
>> help WaveEquation1
```


Function M-files

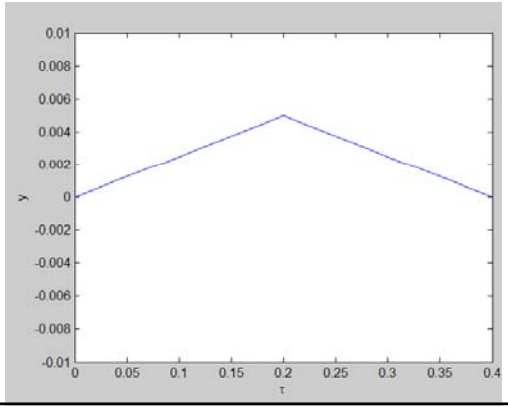
In creating this function M-file I took the opportunity afforded to write something more general than we considered above. It will work for any length of string, for any choice of N, the number of sub-intervals into which I break the string, and for any choice of k, the normalized time step. It does assume a particular form for the initial deflection but the maximum deflection is no longer restricted to 0.005. There are two special numbers embedded in the code: 150 is the number of frames which will be generated and 2 is the axis scale, meaning that the vertical axis for each frame will be set to -2 times max initial deflection to +2 times max initial deflection. Although 150 and 2 are arbitrary numbers they are named and set at the very start of the function M-file.

The Wave Equation

The gain from all of this effort is that with one edit (setting FrameNumber = 300) and two commands

```
>> M = WaveEquation1(0.4,20,0.005,0.005);
>> movie2avi(M,'WaveEquation3.avi','compression','none');
```

We have an animation over the same time frame as before but with 21 points per discrete time and time step of 0.005:

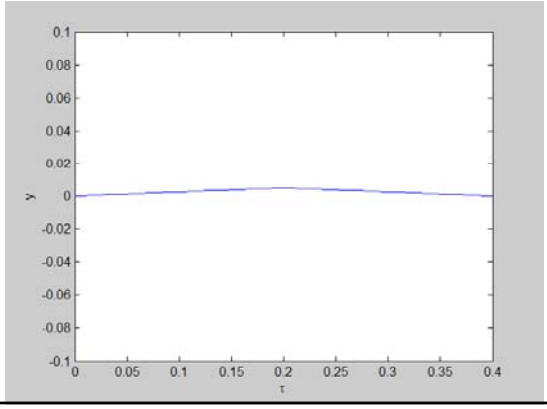


The Wave Equation

Suppose we leave $k = 0.01$ (setting FrameNumber = 15 and Scale = 20) but try for a larger $N = 50$

```
>> M = WaveEquation1(0.4,50,0.01,0.005);
>> movie2avi(M,'WaveEquation4.avi','compression','none');
```

The approximate solution is unstable and does not even remotely resemble the true solution.



Function M-files

I should note that when I executed the previous set of commands I actually ended up with a .avi file where the axes were very badly drawn and labeled. The problem was that at that point I had overwritten the Matlab variable **M** several times and it had become corrupted. I executed the following command:

```
>> clear M
```

before re-executing the commands of the previous slide in order to eliminate the problem. Many Matlab practitioners would advocate using the **clear** command far more often than I have done.

Instability

The problem of instability arises because of the recursion:

$$y^{(j+1)} = Ay^{(j)} - y^{(j-1)}$$

which lies at the heart of the algorithm. It is generally true of a recursion that it may be *stable*, meaning that, irrespective of the initial values $y^{(0)}$ and $y^{(1)}$, the vectors do not grow inordinately large, or it may be *unstable*, meaning that, for at least one choice of the initial values, they do.

Instability

To illustrate this point consider the following simple example:

$$y^{(j+1)} = Ay^{(j)} - y^{(j-1)} \quad , \quad A = 2I \quad , \quad y^{(0)} = 0 \quad , \quad y^{(1)} \neq 0$$

$$y^{(2)} = 2y^{(1)} \quad , \quad y^{(3)} = 2\left(2y^{(1)}\right) - y^{(1)} = 3y^{(1)}$$

$$y^{(4)} = 2\left(3y^{(1)}\right) - 2y^{(1)} = 4y^{(1)} \quad , \quad y^{(5)} = 2\left(4y^{(1)}\right) - 3y^{(1)} = 5y^{(1)}$$

The pattern of the solution has become obvious:

$$y^{(j)} = jy^{(1)}$$

So that clearly the vector $y^{(j)}$ becomes larger and larger with increasing j.

Instability

The second order recursion:

$$y^{(j+1)} = Ay^{(j)} - y^{(j-1)}$$

can be in general transformed to a first order recursion:

$$\begin{bmatrix} y^{(j)} \\ y^{(j+1)} \end{bmatrix} = \begin{bmatrix} 0 & I \\ -I & A \end{bmatrix} \begin{bmatrix} y^{(j-1)} \\ y^{(j)} \end{bmatrix} \quad , \quad v^{(j+1)} = \begin{bmatrix} 0 & I \\ -I & A \end{bmatrix} v^{(j)}$$

and stability is then concerned with whether the size of the vector $v^{(j)}$ grows without bound or not. Of course we have to be quite precise in terms of what we mean by *size*.

Instability

In practice we insist that the *size* of the vector be defined so that it possesses a certain set of properties, properties which we consider to be the minimal set for the word *size* to be appropriate. Moreover instead of using the word *size* we use the word *norm*, a term introduced to mathematics by Dirichlet. Denoting the norm of vector v by $\|v\|$ the associated *norm* of matrix Q is defined and denoted by:

$$\|Q\| = \sup_{v \neq 0} \left(\frac{\|Qv\|}{\|v\|} \right)$$

So the norm of the matrix identifies the maximum stretch of the vector, v, achieved by the action upon it of the matrix Q.

Lax-Richtmyer/Practical Stability

Obviously the solution of the recursion $v^{(j+1)} = Qv^{(j)}$ is

$$v^{(j)} = Q^j v^{(0)}$$

The recursion is said to *practically stable* or *Lax-Richtmyer stable* if there is some finite bound K such that

$$\left\|Q^j\right\| \leq K \quad \text{for all} \quad j \geq 0$$

from which it follows that

$$\left\|v^{(j)}\right\| \leq K\left\|v^{(0)}\right\| \quad \text{for all} \quad j \geq 0$$

i.e. that the vector $v^{(j)}$ does not grow without bound.

Lax-Richtmyer/Practical Stability

If $v^{(0)}$ is chosen to be an eigenvector of the matrix Q with associated eigenvalue λ then the solution is

$$v^{(j)} = \lambda^j v^{(0)}$$

which clearly does not remain bounded if λ has modulus greater than 1. We find that the recursion is not practically stable if any eigenvalue of the matrix Q has modulus exceeding 1. This it turns out is nearly equal to an alternative definition of practical stability. It can be shown that the recursion is practically stable iff all of the eigenvalues of Q have modulus ≤ 1 and those which have modulus = 1 are *non-degenerate* (i.e. their geometric multiplicity equals their algebraic multiplicity).

Lax-Richtmyer/Practical Stability

To justify the equation of the previous slide note that

$$v^{(j)} = Q^j v^{(0)}$$

$v^{(1)} = Qv^{(0)} = \lambda v^{(0)}$ by definition of λ and $v^{(0)}$.

$$v^{(2)} = Qv^{(1)} = Q\lambda v^{(0)} = \lambda Q v^{(0)} = \lambda^2 v^{(0)}$$

$$v^{(3)} = Qv^{(2)} = Q\lambda^2 v^{(0)} = \lambda^2 Q v^{(0)} = \lambda^3 v^{(0)}$$

The pattern has become apparent and is readily proven by induction.

Practical Stability vs Convergence

On the face of it determining that the recursion is Lax-Richtmyer stable does not seem to fully engage with the question of whether the approximation actually yields the correct answer. Two things can happen to the errors as we move forward through the iteration scheme. Either the errors, which are originally small, remain small or they start to accumulate, becoming larger with each step. This is what happened above. We say the finite difference scheme is *convergent* provided errors do not propagate or accumulate in this way. So if we select h and k to give small errors initially we continue to have similarly small errors throughout.

Section: 7 Partial Differential Equations

Practical Stability vs Convergence

Just because the vectors $y^{(j)}$ are not *blowing up* does not obviously mean that they are necessarily close to the desired answer, i.e. that errors do not accumulate. Obvious or not, there is an important result called either the *Lax Equivalence Theorem* or, less commonly, the *Lax-Richtmyer Equivalence Theorem* which states that for a certain important class of problems Lax-Richtmyer or practical stability *does* in fact imply convergence. In fact the theorem also says that convergence implies practical stability. So practical stability and convergence are *equivalent* for this class of problems.

Lax-Richtmyer/Practical stability

You will see it claimed that calculating the eigenvalues and determining whether those of modulus 1 are non-degenerate can be a challenging problem when the matrix Q becomes larger and that the task of determining Lax-Richtmyer stability is, as a result, rather onerous in practice. For this reason one commonly performs a *Von Neumann Stability Analysis* because this analysis is much easier to achieve and often gives answers rather close to those which we are seeking. This claim is overstated. For a large number of problems performing an exact stability analysis is not much more difficult than performing a Von Neumann stability analysis and understanding why it works is actually easier. We will therefore elect to do both.

Von Neumann Stability Analysis

The recursion in the original form in which we acquired it was:

$$y^{(j+1)} = Ay^{(j)} - y^{(j-1)} + b^{(j)}$$

$$A = \begin{bmatrix} 2(1-\sigma^2) & \sigma^2 & 0 & \cdots & 0 & 0 \\ \sigma^2 & 2(1-\sigma^2) & \sigma^2 & \cdots & 0 & 0 \\ 0 & \sigma^2 & 2(1-\sigma^2) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2(1-\sigma^2) & \sigma^2 \\ 0 & 0 & 0 & \cdots & \sigma^2 & 2(1-\sigma^2) \end{bmatrix} \qquad b^{(j)} = \begin{bmatrix} \sigma^2 y_{0,j} \\ 0 \\ 0 \\ \vdots \\ 0 \\ \sigma^2 y_{10,j} \end{bmatrix}$$

The Von Neumann Stability Analysis is based essentially on two ideas. Firstly we just ignore the first and last equations which are rather different from the others because of boundary effects. Von Neumann does justify ignoring these equations but the practical application of the method comprises simply ignoring them. Secondly we propose a particular "solution".

Von Neumann Stability Analysis

Based on Fourier theory, Von Neumann can identify the particular "solution"

$$y^{(j)} = \lambda^j \begin{bmatrix} e^{iKx_1} \\ e^{iKx_2} \\ e^{iKx_3} \\ \vdots \\ e^{iKx_{N-1}} \end{bmatrix}$$

where my notation is revealing its flaw as in this case i reverts to denoting the square root of -1. λ is a scalar. As usual x_1, … denote the N-1 interior values of x at which the solution is to be found. K is arbitrary.

Von Neumann Stability Analysis

Expanding the equations we have:

$$\begin{aligned}
\lambda^{j+1}e^{iKx_1} &= \lambda^j\left(2\left(1-\sigma^2\right)e^{iKx_1} + \sigma^2 e^{iKx_2}\right) - \lambda^{j-1}e^{iKx_1} + \sigma^2 y_{0,j} \\
\lambda^{j+1}e^{iKx_2} &= \lambda^j\left(\sigma^2 e^{iKx_1} + 2\left(1-\sigma^2\right)e^{iKx_2} + \sigma^2 e^{iKx_3}\right) - \lambda^{j-1}e^{iKx_2} \\
\lambda^{j+1}e^{iKx_3} &= \lambda^j\left(\sigma^2 e^{iKx_2} + 2\left(1-\sigma^2\right)e^{iKx_3} + \sigma^2 e^{iKx_4}\right) - \lambda^{j-1}e^{iKx_3} \\
&\vdots \\
\lambda^{j+1}e^{iKx_{N-2}} &= \lambda^j\left(\sigma^2 e^{iKx_{N-3}} + 2\left(1-\sigma^2\right)e^{iKx_{N-2}} + \sigma^2 e^{iKx_{N-1}}\right) - \lambda^{j-1}e^{iKx_{N-2}} \\
\lambda^{j+1}e^{iKx_{N-1}} &= \lambda^j\left(\sigma^2 e^{iKx_{N-2}} + 2\left(1-\sigma^2\right)e^{iKx_{N-1}}\right) - \lambda^{j-1}e^{iKx_{N-1}} + \sigma^2 y_{N,j}
\end{aligned}$$

Because the interior x-values are uniformly spaced the second implies the third, fourth, fifth, … , $(N\text{-}2)^{\text{th}}$. Morover, in the spirit of the Von Neumann analysis, we will simply ignore the first and last.

Von Neumann Stability Analysis

The single remaining equation:

$$\lambda^{j+1}e^{iKx_2} = \lambda^j\left(\sigma^2 e^{iKx_1} + 2\left(1-\sigma^2\right)e^{iKx_2} + \sigma^2 e^{iKx_3}\right) - \lambda^{j-1}e^{iKx_2}$$

permits simplification:

$$\lambda^2 = \lambda\left(\sigma^2 e^{-iKh} + 2\left(1-\sigma^2\right) + \sigma^2 e^{iKh}\right) - 1$$

$$\lambda^2 - \lambda\left(2\left(1-\sigma^2\right) + 2\sigma^2\cos(Kh)\right) + 1 = 0$$

$$\lambda^2 - 2\lambda\left(1 - 2\sigma^2\sin^2\left(\frac{Kh}{2}\right)\right) + 1 = 0$$

Von Neumann Stability Analysis

With each step this special "solution" is multiplied by the scalar λ. If the modulus of λ exceeds 1 then this special "solution" grows without bound. If on the other hand the modulus does not exceed 1 then this particular solution is bounded. Von Neumann concludes that an approximate condition for practical stability is that the roots of the quadratic:

$$\lambda^2 - 2\lambda\left(1 - 2\sigma^2\sin^2\left(\frac{Kh}{2}\right)\right) + 1$$

have modulus not exceeding 1 for arbitrary K.

Von Neumann Stability Analysis

$$\lambda^2 - 2\lambda\left(1 - 2\sigma^2 \sin^2\left(\frac{Kh}{2}\right)\right) + 1 = \lambda^2 - 2(1-\alpha)\lambda + 1$$

where $\alpha = 2\sigma^2 \sin^2\left(\frac{Kh}{2}\right)$ so that $0 \le \alpha \le 2\sigma^2$

The roots are readily found: $\lambda = 1 - \alpha \pm \sqrt{\alpha(\alpha - 2)}$

If $0 \le \alpha \le 2$ they become: $\lambda = 1 - \alpha \pm i\sqrt{\alpha(2 - \alpha)}$

and it is easily confirmed that both have modulus 1.

If $2 < \alpha$ one becomes: $\lambda = 1 - \alpha - \sqrt{\alpha(\alpha - 2)}$

which is real < -1 so that its modulus exceeds 1.

Von Neumann Stability Analysis

We conclude that we should require $\sigma^2 \le 1$ or $\frac{k^2}{h^2} \le 1$

Initially for the wave equation we chose:

$$h = \frac{len}{N} = \frac{0.4}{10} = 0.04 \quad , \quad k = 0.01 \qquad \frac{k^2}{h^2} = \left(\frac{0.01}{0.04}\right)^2 = \frac{1}{16} < 1$$

But in the unstable example we chose:

$$h = \frac{len}{N} = \frac{0.4}{50} = 0.008 \quad , \quad k = 0.01 \qquad \frac{k^2}{h^2} = \left(\frac{0.01}{0.008}\right)^2 = \frac{25}{16} > 1$$

The upshot of the analysis is the long-awaited upper bound on the time step *k,* namely $k \le h$.

Exact Stability Analysis

We are concerned with the eigenvalues of the matrix Q where:

$$Q = \begin{bmatrix} 0 & I \\ -I & A \end{bmatrix}$$

Let λ be an eigenvalue with associated eigenvector:

$$v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$Qv = \begin{bmatrix} 0 & I \\ -I & A \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} v_2 \\ -v_1 + Av_2 \end{bmatrix} = \lambda \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

Exact Stability Analysis

Expanding yields:

$$v_2 = \lambda v_1 \quad , \quad \left(\lambda^2 I - \lambda A + I\right) v_1 = 0.$$

where I as usual denotes the identity matrix. Now expand the unknown vector and recall A:

$$v_1 = \begin{bmatrix} v_{1,1} \\ v_{1,2} \\ \vdots \\ v_{1,N-1} \end{bmatrix} \qquad A = \begin{bmatrix} 2(1-\sigma^2) & \sigma^2 & 0 & \cdots & 0 & 0 \\ \sigma^2 & 2(1-\sigma^2) & \sigma^2 & \cdots & 0 & 0 \\ 0 & \sigma^2 & 2(1-\sigma^2) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2(1-\sigma^2) & \sigma^2 \\ 0 & 0 & 0 & \cdots & \sigma^2 & 2(1-\sigma^2) \end{bmatrix}$$

Exact Stability Analysis

We obtain:

$$\begin{aligned}
&\left(\lambda^2+1\right)v_{1,1}-\lambda\left(\sigma^2 v_{1,0}+2\left(1-\sigma^2\right)v_{1,1}+\sigma^2 v_{1,2}\right)=0\\
&\left(\lambda^2+1\right)v_{1,2}-\lambda\left(\sigma^2 v_{1,1}+2\left(1-\sigma^2\right)v_{1,2}+\sigma^2 v_{1,3}\right)=0\\
&\left(\lambda^2+1\right)v_{1,3}-\lambda\left(\sigma^2 v_{1,2}+2\left(1-\sigma^2\right)v_{1,3}+\sigma^2 v_{1,4}\right)=0\\
&\qquad\vdots\\
&\left(\lambda^2+1\right)v_{1,N-2}-\lambda\left(\sigma^2 v_{1,N-3}+2\left(1-\sigma^2\right)v_{1,N-2}+\sigma^2 v_{1,N-1}\right)=0\\
&\left(\lambda^2+1\right)v_{1,N-1}-\lambda\left(\sigma^2 v_{1,N-2}+2\left(1-\sigma^2\right)v_{1,N-1}+\sigma^2 v_{1,N}\right)=0
\end{aligned}$$

where the two *fictitious* elements $v_{1,0}$ and $v_{1,N}$ are defined to be 0.

Exact Stability Analysis

We may give the general formula:

$$\left(\lambda^2+1\right)v_{1,j}-\lambda\left(\sigma^2 v_{1,j-1}+2\left(1-\sigma^2\right)v_{1,j}+\sigma^2 v_{1,j+1}\right)=0$$

or alternatively:

$$v_{1,j+1}=-v_{1,j-1}+\left(\frac{\lambda^2-2\left(1-\sigma^2\right)\lambda+1}{\lambda\sigma^2}\right)v_{1,j}$$

for $1 \le j \le N\text{-}1$.

This type of equation is called a *second order linear recursion*.

Exact Stability Analysis

When you have seen recursions before they will almost certainly have been in the form where the first few terms were given (the first two terms in the case of a second order recursion) and the recursion then permits calculation of the subsequent terms. Essentially this is the description of an initial value problem for recursions. The problem of the previous slide is different, it is a *boundary value problem* for recursions.

Exact Stability Analysis

Experience has taught that linear recursions generally permit solutions in the form of powers, i.e.

$$v_{1,j} = r^j$$

Putting this expression into the recursion gives:

$$r^{j+1} = -r^{j-1} + \left(\frac{\lambda^2 - 2\left(1-\sigma^2\right)\lambda + 1}{\lambda\sigma^2} \right) r^j$$

i.e. that exponent r must be a root of the quadratic equation:

$$r^2 - \eta r + 1 = 0 \quad , \quad \eta = \left(\frac{\lambda^2 - 2\left(1-\sigma^2\right)\lambda + 1}{\lambda\sigma^2} \right)$$

Exact Stability Analysis

Being quadratic this equation will generally have two distinct roots, r_1 and r_2. Moreover, because the recursion is linear the general solution to the recursion is:

$$v_{1,j} = c_1 r_1^j + c_2 r_2^j$$

for arbitrary constants c_1 and c_2. The boundary conditions $v_{1,0} = v_{1,N} = 0$ give:

$$c_1 + c_2 = 0 \quad , \quad c_1 r_1^N + c_2 r_2^N = 0$$

$$c_2 = -c_1 \quad , \quad c_1 \left(r_1^N - r_2^N \right) = 0$$

Exact Stability Analysis

If c_1 is zero then $v_{1,j} = 0$ for all j. It follows that vector v_1 and therefore vector v_2 and therefore vector v is zero. This is impossible if v is an eigenvector. We conclude that c_1 is non-zero and therefore that:

$$r_1^N = r_2^N$$

It follows that r_2/r_1 is an Nth root of unity. Since r_1 and r_2 are assumed distinct this ratio is not equal to 1, i.e.

$$r_2 = r_1 e^{\frac{i2n\pi}{N}} \quad 1 \le n \le N-1.$$

Exact Stability Analysis

We have: $r^2 - \eta r + 1 = (r - r_1)(r - r_2)$

giving: $r_1 + r_2 = \eta \quad , \quad r_1 r_2 = 1$

$$1 = r_1^2 e^{\frac{i2n\pi}{N}} = \left(r_1 e^{\frac{in\pi}{N}}\right)^2 \quad \text{for} \quad 1 \le n \le N-1$$

$$r_1 e^{\frac{in\pi}{N}} = \pm 1 \quad \text{i.e.} \quad r_1 = e^{-\frac{in\pi}{N}}, e^{\frac{i(N-n)\pi}{N}}$$

$$\eta = 2\cos\left(\frac{n\pi}{N}\right) \quad , \quad 2\cos\left(\frac{(N\text{-}n)\pi}{N}\right) = -2\cos\left(\frac{n\pi}{N}\right)$$

Exact Stability Analysis

We may conclude that:

$$\eta = \frac{\lambda^2 - 2(1-\sigma^2)\lambda + 1}{\lambda\sigma^2} = 2\cos\left(\frac{n\pi}{N}\right) \quad \text{for } 1 \le n \le N-1$$

giving a description of the eigenvalues of Q as the $2(N\text{-}1)$ roots of the $N\text{-}1$ quadratic equations:

$$\lambda^2 - 2\left(1 - \sigma^2 + \sigma^2 \cos\left(\frac{n\pi}{N}\right)\right)\lambda + 1 = 0 \quad \text{for } 1 \le n \le N-1$$

Exact Stability Analysis

As each of the quadratic equations is real and as the product of the two roots is unity, it follows that both roots have modulus not exceeding 1 only if they are equal or complex, in which case they have modulus equal to 1. Accordingly the actual condition for stability is that:

$$-1 \le \left(1 - \sigma^2 + \sigma^2 \cos\left(\frac{n\pi}{N}\right)\right) \le 1 \quad \text{for } 1 \le n \le N-1$$

$$-2 \le -\sigma^2\left(1 - \cos\left(\frac{n\pi}{N}\right)\right) \le 0 \quad \text{for } 1 \le n \le N-1$$

Exact Stability Analysis

The second inequality is guaranteed. The first requires:

$$2\sigma^2 \sin^2\left(\frac{n\pi}{2N}\right) \le 2 \quad \text{for } 1 \le n \le N-1$$

$$\sigma^2 \sin^2\left(\frac{n\pi}{2N}\right) \le 1 \quad \text{for } 1 \le n \le N-1$$

Obviously this holds if $\sigma^2 \le 1$, i.e. if the Von Neumann stability condition holds. But it can hold in cases where the Von Neumann stability condition fails.

Exact Stability Analysis

For large N there is very little difference between the actual stability condition and the Von Neumann condition. For small N however there can be a significant difference. If $N = 2$ for example, Von Neumann requires:

$$\sigma^2 \leq 1$$

and the exact condition is:

$$\sigma^2 \sin^2\left(\frac{\pi}{4}\right) \leq 1 \quad , \quad \text{i.e.} \quad \sigma^2 \leq 2$$

a significantly larger range.

The 2-D Wave Equation

The 2-D wave equation is commonly written as.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{1}{v^2}\frac{\partial^2 u}{\partial t^2}$$

It describes the transverse displacement of an idealised two dimensional membrane subject to the assumption that this displacement is rather small. If the 1-D wave equation can credibly claim to offer some form of model for the vibration of a string, then the 2-D wave equation can equally claim to offer a similar model for the vibration of the membrane of a drum.

The 2-D Wave Equation

The same time scaling $\tau = vt$ as used previously is effective again giving the simpler equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{\partial^2 u}{\partial \tau^2}$$

As before we will refer to τ as *normalised time*.

The 2-D Wave Equation

If we wish to consider drums then we note that in many drums (*e.g.* kettle drum/timpani) the membrane is a disc which is fixed along its circular boundary. It seems tolerably obvious to consider an alternative choice of coordinates. Rather than opting for the Cartesian coordinates x, y and normalised time τ, we choose to consider polar coordinates r, θ and normalised time τ. A standard transformation yields the 2-D wave equation relative to these new coordinates:

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2} = \frac{\partial^2 u}{\partial \tau^2}$$

The 2-D Wave Equation

We assume that the drum is struck dead centre in such a manner as to create an initial symmetric deflection, i.e. a deflection which is independent of the angle θ. Needless to say this is hardly a realistic assumption. We will also assume that at the moment when we begin the simulation the membrane is still, i.e. that the velocity is zero across the whole membrane.

The 2-D Wave Equation

The resulting normalised problem to be solved is a second order linear partial differential equation subject to boundary conditions and initial conditions:

$$\frac{\partial^2 u}{\partial r^2}+\frac{1}{r}\frac{\partial u}{\partial r}+\frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2}=\frac{\partial^2 u}{\partial \tau^2}$$

$$u(a,\theta,\tau)=0,\quad 0\le\theta\le 2\pi\,,\ \tau\ge 0$$

$$u(r,\theta,0)=f(r)\ ,\ 0\le r\le a\,,\ 0\le\theta\le 2\pi$$

$$\frac{\partial u(r,\theta,0)}{\partial \tau}=0\ ,\ 0\le r\le a\,,\ 0\le\theta\le 2\pi$$

With $f(r)$ a function in r which denotes the initial deflection and with a denoting the radius.

Finite differences: Polar Coordinates

We must break up the space and normalised time into discrete steps. Break the angular range $[0,2\pi]$ into M equal steps and consider the approximate solution at the angles $m\Delta$ where $\Delta = 2\pi/M$ for integer $m = 0, 1, \ldots, M\text{-}1$ only. Break normalised time into steps k and consider the approximate solution at normalised times jk only for non-negative integer j. We wish to break the radial range $[0,a]$ into N equal steps however the divisions by r and r^2 in the equation imply that a choice of discrete radii which includes 0 may lead to numerical difficulties.

Finite differences: Polar Coordinates

One of many ideas offered to circumvent this problem is to consider the approximate solution at the radii $(i-\frac{1}{2})h$ for integer $i = 1, 2, \ldots, N$. Hence h must be chosen such that:

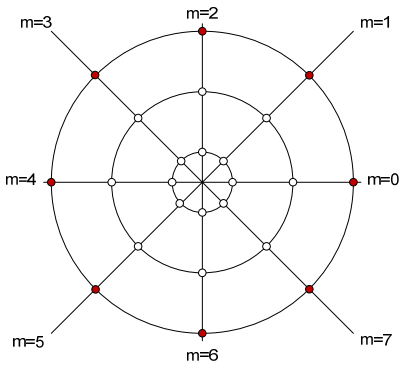
$$(N-\tfrac{1}{2})h = a \qquad \text{giving} \qquad h = \frac{2a}{2N-1}$$

We now introduce the standard short-hand notation:

$$u(r_i, \theta_m, \tau_j) = u((i-\tfrac{1}{2})h, m\Delta, jk) = u_{i,m,j}$$

Finite differences: Polar Coordinates

To illustrate we show the break up of space using $M = 8$ angular and $N = 3$ radial steps, with $r = 0$ excluded.



Section: 7 Partial Differential Equations

Finite differences: Polar Coordinates

The critical idea, as before, is to apply central differences. Again the radial coordinate is a little different so we consider it last. The normalised time is simplest yielding:

$$\left(\frac{\partial^2 u}{\partial \tau^2}\right)_{i,m,j} = \frac{u_{i,m,j+1} - 2u_{i,m,j} + u_{i,m,j-1}}{k^2} \quad \text{for} \quad 1 \le j$$

Of course the initial condition gives:

$$u_{i,m,0} = u\left(\left(i - \tfrac{1}{2}\right)h, m\Delta, 0\right) = f\left(\left(i - \tfrac{1}{2}\right)h\right)$$

Finite differences: Polar Coordinates

The angular coordinate yields:

$$\left(\frac{\partial^2 u}{\partial \theta^2}\right)_{i,m,j} = \frac{u_{i,m+1,j} - 2u_{i,m,j} + u_{i,m-1,j}}{\Delta^2} \quad \text{for} \quad 1 \le m \le M-2$$

If this formula is employed for the first value of $m = 0$ then it makes reference to $u_{i,-1,j}$. The angle referred to by the illegal index -1 is one angular step *before* the angle 0. This is the same as the angle $(M\text{-}1)\Delta$. In short:

$$u_{i,-1,j} = u_{i,M-1,j}$$

Section: 7 Partial Differential Equations

Finite differences: Polar Coordinates

$$\left(\frac{\partial^2 u}{\partial \theta^2}\right)_{i,m,j} = \frac{u_{i,m+1,j} - 2u_{i,m,j} + u_{i,m-1,j}}{\Delta^2} \quad \text{for} \quad 1 \le m \le M-2$$

Similarly if this formula is employed for the last value of $m = M\text{-}1$ then it makes reference to $u_{i,M,j}$. The angle referred to by the illegal index M is M angular steps *after* the angle 0. This is the angle $M\Delta = 2\pi$, i.e. it is the same as the angle 0. In short:

$$u_{i,M,j} = u_{i,0,j}$$

Finite differences: Polar Coordinates

I deduce that, provided I am mindful of the periodic constraints:

$$u_{i,-1,j} = u_{i,M-1,j} \qquad u_{i,M,j} = u_{i,0,j}$$

I may employ the central difference formula:

$$\left(\frac{\partial^2 u}{\partial \theta^2}\right)_{i,m,j} = \frac{u_{i,m+1,j} - 2u_{i,m,j} + u_{i,m-1,j}}{\Delta^2}$$

for *all* indices *m* from 0 to *M*-1.

Finite differences: Polar Coordinates

Now for the radial coordinate I have:

$$\left(\frac{\partial u}{\partial r}\right)_{i,m,j} = \frac{u_{i+1,m,j} - u_{i-1,m,j}}{2h} \quad \text{for} \quad 2 \le i \le N-1$$

$$\left(\frac{\partial^2 u}{\partial r^2}\right)_{i,m,j} = \frac{u_{i+1,m,j} - 2u_{i,m,j} + u_{i-1,m,j}}{h^2} \quad \text{for} \quad 2 \le i \le N-1$$

Moreover, since I chose the discrete radial values such that the last lies on the boundary, the boundary condition yields:

$$u_{N,m,j} = 0$$

Finite differences: Polar Coordinates

A problem arises with regard to $i = 1$. If the central difference formula is applied in this case it yields:

$$\left(\frac{\partial u}{\partial r}\right)_{1,m,j} = \frac{u_{2,m,j} - u_{0,m,j}}{2h} \qquad \left(\frac{\partial^2 u}{\partial r^2}\right)_{1,m,j} = \frac{u_{2,m,j} - 2u_{1,m,j} + u_{0,m,j}}{h^2}$$

i.e. it makes reference to an illegal index $i = 0$. However, consideration of the sample plot of the spatial nodes given above reveals that the node being referred to by the illegal triplet $(0,m,j)$ is actually the node on the same diameter line as the node $(1,m,j)$ but on the other side of the origin. In short it is the node indexed by the legal triplet $(1,m+(M/2),j)$ *provided* M is even. Accordingly I take M even and use:

$$u_{0,m,j} = u_{1,m+\frac{M}{2},j}$$

The 2-D Wave Equation

The normalised wave equation itself gives:

$$\frac{\partial^2 u}{\partial \tau^2} = \frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2}$$

$$\left(\frac{u_{i,m,j+1} - 2u_{i,m,j} + u_{i,m,j-1}}{k^2}\right) = \left(\frac{u_{i+1,m,j} - 2u_{i,m,j} + u_{i-1,m,j}}{h^2}\right) +$$

$$\left(\frac{1}{\left(i-\frac{1}{2}\right)h}\right)\left(\frac{u_{i+1,m,j} - u_{i-1,m,j}}{2h}\right) + \left(\frac{1}{\left(i-\frac{1}{2}\right)h}\right)^2\left(\frac{u_{i,m+1,j} - 2u_{i,m,j} + u_{i,m-1,j}}{\Delta^2}\right)$$

where references to illegal values of indices i (= 1) and m (= 0 or M) are treated as discussed above.

The 2-D Wave Equation

Rearrangement yields: $\sigma^2 = \dfrac{k^2}{h^2}$

$$u_{i,m,j+1} = 2u_{i,m,j} + \sigma^2\left(u_{i+1,m,j} - 2u_{i,m,j} + u_{i-1,m,j}\right) +$$

$$\sigma^2\left(\frac{u_{i+1,m,j} - u_{i-1,m,j}}{2i-1}\right) + \sigma^2\left(\frac{u_{i,m+1,j} - 2u_{i,m,j} + u_{i,m-1,j}}{\left(i-\frac{1}{2}\right)^2\Delta^2}\right) - u_{i,m,j-1}$$

for $j \geq 1$, $1 \leq i \leq N$-1, $0 \leq m \leq M$-1. As before we introduce a vector comprising all of the unknown numbers $u_{i,m,j}$ for given j, i.e. $(y^{(j)})^{\mathrm{T}}$ equals

$$\begin{bmatrix} u_{1,0,j} & u_{1,1,j} & \cdots & u_{1,M-1,j} & u_{2,0,j} & u_{2,1,j} & \cdots & u_{2,M-1,j} & \cdots & u_{N-1,0,j} & u_{N-1,1,j} & \cdots & u_{N-1,M-1,j} \end{bmatrix}$$

The 2-D Wave Equation

These equations may be very neatly expressed as:

$$y^{(j+1)} = Ay^{(j)} - y^{(j-1)}$$

Matrix A is expressible as $2C$ where matrix C has the highly structured form

$$\begin{bmatrix}
\left(1-\sigma^2\right)I - \left(\frac{2\sigma^2}{\Delta^2}\right)Q & \sigma^2 I & 0 & \cdots & 0 & 0 \\
\frac{\sigma^2}{3}I & \left(1-\sigma^2\right)I - \left(\frac{2\sigma^2}{3^2\Delta^2}\right)Q & \frac{2\sigma^2}{3}I & \cdots & 0 & 0 \\
0 & \frac{2\sigma^2}{5}I & \left(1-\sigma^2\right)I - \left(\frac{2\sigma^2}{5^2\Delta^2}\right)Q & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 2\left(1-\sigma^2\right)I - \left(\frac{2\sigma^2}{(2N-5)^2\Delta^2}\right)Q & \left(\frac{N-2}{2N-5}\right)\sigma^2 I \\
0 & 0 & 0 & \cdots & 2\left(\frac{N-2}{2N-3}\right)\sigma^2 I & \left(1-\sigma^2\right)I - \left(\frac{2\sigma^2}{(2N-3)^2\Delta^2}\right)Q
\end{bmatrix}$$

The 2-D Wave Equation

In the expansion of matrix C, to avoid clutter I denotes the MxM identity matrix and Q denotes the special matrix:

$$Q = 2I - \left(S + S^{-1}\right)$$

where:

$$S = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

The 2-D Wave Equation

If you care to check the above expansions you may note that there should apparently be a difficulty in the exceptional case of $i = 1$ where the illegal indexing triplet $(0,m,j)$ arises and must be appropriately treated. However, in this case the equation actually becomes:

$$u_{1,m,j+1} = 2u_{1,m,j} + \sigma^2\left(u_{2,m,j} - 2u_{1,m,j} + u_{0,m,j}\right) +$$
$$\sigma^2\left(u_{2,m,j} - u_{0,m,j}\right) + \sigma^2\left(\frac{u_{1,m+1,j} - 2u_{1,m,j} + u_{1,m-1,j}}{\left(\frac{1}{2}\right)^2\Delta^2}\right) - u_{1,m,j-1}$$

The coefficient of the problematic term $u_{0,m,j}$ becomes $\sigma^2 - \sigma^2 = 0$, i.e. the term does not occur and therefore its unorthodox treatment is not required.

The 2-D Wave Equation

Again if you care to check the above expansions you should also note the boundary condition $u_{N,m,j} = 0$ must be recalled and employed when the exceptional case $i = N\text{-}1$ is considered.

The 2-D Wave Equation

$$y^{(j+1)} = Ay^{(j)} - y^{(j-1)}$$

As in the 1-D wave equation this is a *second order recursion*, meaning that the next value depends upon the *two* preceding values. The same problem arises: the recursion can start to predict the answer for $j = 2$ onwards, once it is known for $j = 0$ and $j = 1$. The initial conditions give it for $j = 0$, but we must determine it for $j = 1$ by some other means if we are to start the recursion. We resolve this issue by employing the second initial condition in the manner of the 1-D wave equation.

The 2-D Wave Equation

$$u_{i,m,1} = u(r_i, \theta_m, k) =$$

$$u(r_i, \theta_m, 0) + k\frac{\partial u(r_i, \theta_m, 0)}{\partial \tau} + \frac{k^2}{2}\frac{\partial^2 u(r_i, \theta_m, 0)}{\partial \tau^2} + \cdots$$

$$\cong u_{i,m,0} + \frac{k^2}{2}\left(\frac{\partial^2 u(r_i, \theta_m, 0)}{\partial r^2} + \frac{1}{r_i}\frac{\partial u(r_i, \theta_m, 0)}{\partial r} + \frac{1}{r_i^2}\frac{\partial^2 u(r_i, \theta_m, 0)}{\partial \theta^2}\right)$$

$$= u_{i,m,0} + \frac{k^2}{2}\left(\frac{u_{i+1,m,0} - 2u_{i,m,0} + u_{i-1,m,0}}{h^2}\right) + \frac{k^2}{2}\left(\frac{1}{(i-\frac{1}{2})h}\right)\left(\frac{u_{i+1,m,0} - u_{i-1,m,0}}{2h}\right)$$

$$+ \frac{k^2}{2}\left(\frac{1}{(i-\frac{1}{2})h}\right)^2\left(\frac{u_{i,m+1,0} - 2u_{i,m,0} + u_{i,m-1,0}}{\Delta^2}\right)$$

The 2-D Wave Equation

Some elementary manipulation reveals that these equations may be expressed in the elegant form: $y^{(1)} = Cy^{(0)}$

where matrix C is equal to the matrix defined above. Accordingly, it remains "only" to compute the initial vector of unknowns $y^{(0)}$, to choose values for N, M and Δ, recalling that we wish M to be even, and to run the recursion.

Matlab: The 2-D Wave Equation

$$y^{(1)} = Cy^{(0)} \qquad y^{(j+1)} = Ay^{(j)} - y^{(j-1)}$$

```
>> N = 10, M = 16, radius = 1        set parameters N, M and radius
>> h = (2*radius)/((2*N)-1), Delta = (2*pi/M), k = 0.01
        set parameters h, Delta and k
>> s2 = (k^2)/(h^2);                 set parameter s2 = σ^2
>> e0 = ((1-s2)*ones(N-1,1))-((4*s2/(Delta^2))./([1:2:(2*N)-3].^2))';
>> e1 =(s2*[1:N-1]./[1:2:(2*N)-3])';
>> em1 = (s2*[1:N-1]./[3:2:(2*N)-1])';
>> R = spdiags([em1 e0 e1], [-1 0 1], N-1, N-1);
```

Section: 7 Partial Differential Equations

Matlab: The 2-D Wave Equation

$$y^{(1)} = Cy^{(0)} \qquad y^{(j+1)} = Ay^{(j)} - y^{(j-1)}$$

```
>> e = ones(M,1);
>> Stot = spdiags([e e zeros(M,1) e e], [-M+1 -1 0 1 M-1], M, M);
Stot equals S+S^-1
>> D = (2*s2/(Delta^2))*diag(1./([1:2:(2*N)-3].^2));
>> C = kron(R,eye(M))+kron(D,Stot);
>> A = 2*C;
```

I note the use of the *Kronecker product*, implemented by the command **kron**.

Aside: Kronecker Product

When a large matrix is naturally described in terms of blocks the Kronecker product can be useful in creating it. For example suppose a large matrix M is described in terms of a smaller matrix H and zero blocks as:

$$M = \begin{bmatrix} H & 0 & H \\ 0 & H & 0 \\ H & 0 & 0 \end{bmatrix}$$

Then, if H has already been created, M is created by the following code:

```
>> V = [1 0 1;0 1 0;1 0 0];    identifies position of H blocks
>> M = kron(V,H);
```


Matlab: The 2-D Wave Equation

Obviously we will need to implement using a function M-file:

```
function [Storage,Mov]=TwoDWaveEqn(N,M)
% [Storage,Mov] = TwoDWaveEqn(N,M) solves and creates movie of 2-D wave equation in polar coordinates on cicular region of radius equal to 1. N = number of radial steps, M = number of angular steps.
% Initial conditions comprise an initial deflection equal to a bump function and an initial velocity of zero throughout.


k = 0.01;  % arbitrary specification of normalised time step.
Number_Iterations = 200;   % specify number of time steps for which recursion is to be solved.
h = 2/((2*N)-1);  % specification of radial step. Radial points = (i-0.5)h.
Delta = 2*pi/M;          % specification of angular step.
s2 = (k^2)/(h^2);   % set parameter s2 = sigma squared.
rho = 4*s2/(Delta^2);   % set parameter.
```

Matlab: The 2-D Wave Equation

Next task of function M-file is to create matrices A and C.

```
% Must create matrices A and C as sparse matrices.

e0 = ((1-s2)*ones(N-1,1))-(rho./([1:2:(2*N)-3].^2))';
e1 = (s2*[1:N-1]./[1:2:(2*N)-3])';
em1 = (s2*[1:N-1]./[3:2:(2*N)-1])';
R = spdiags([em1 e0 e1],[-1 0 1],N-1,N-1);
e = ones(M,1);
Stot = spdiags([e e zeros(M,1) e e],[-M+1 -1 0 1 M-1],M,M);   % Stot = S + inv(S)
D = (rho/2)*diag(1./([1:2:(2*N)-3].^2));

C = kron(R,eye(M))+kron(D,Stot);
A = 2*C;
```


Matlab: The 2-D Wave Equation

Must create initial bump and initialise problem.

```
radius_vec = h*([1:N]'-0.5);
theta_vec = Delta*[0:M-1]';
bump_function = -[(1/100)*exp(-0.25./(0.25 - (radius_vec(1:N/2).^2)));zeros((N/2)-1,1)];

yprev = kron(bump_function,ones(M,1));   % initial deflections.
ycurrent = C*yprev;  % deflections after one time step.
L = length(ycurrent);

Storage = zeros(L,Number_Iterations+2);   %  set aside space to store data.
Storage(:,1) = yprev;
Storage(:,2) = ycurrent;   %  Store initial deflection and deflection at next time step.
```

Matlab: The 2-D Wave Equation

Main loop

```
for count = 1:Number_Iterations
   ytemp = (A*ycurrent) - yprev;  %  apply recursion
   yprev = ycurrent;  % update previous
   ycurrent = ytemp;  % update current
   Storage(:,count+2) = ycurrent;  % store data
end


Storage = [Storage;zeros(M,Number_Iterations+2)];    % Add boundary values to
solution data
```


Matlab: The 2-D Wave Equation

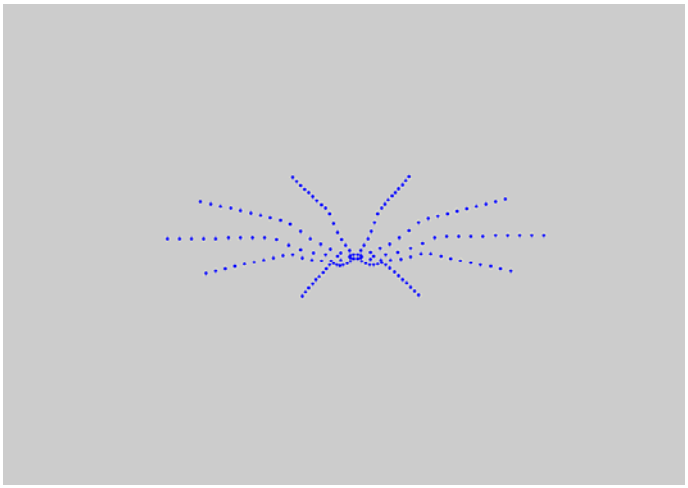
Visualisation

```
X = kron(radius_vec,cos(theta_vec));  % create vector of x-coordinates of nodes
Y = kron(radius_vec,sin(theta_vec));   % create vector of y-cordinates of nodes


for count = 1:Number_Iterations+2
   handlefig = figure('Position',[100 100 850 600]);
   plot3(X,Y,Storage(:,count),'.')
   axis([-1 1 -1 1 -2/100 2/100]);
   axis off
   Mov(count) = getframe(handlefig);
   close(handlefig)
end
```

Matlab: The 2-D Wave Equation

```
>> [Storage,Mov]=TwoDWaveEqn(16,10);
>>  movie2avi(Mov,'TwoDWavEqn.avi', 'compression', 'none');
```



The Heat Equation

The heat equation describes the change of temperature at a point in a region with time as the heat diffuses through the region by the process of *heat conduction*. The equation therefore describes diffusion of heat under rather idealised conditions, the loss or gain of heat due to radiation/irradiation is not modelled, nor is heat convection. In a fluid where molecules are free to move there are two mechanisms for the diffusion of heat: heat conduction where high energy molecules collide with lower energy molecules and impart some energy to them and *advection*, where molecules physically diffuse or relocate carrying thermal energy with them. Heat convection comprises the sum of the diffusions of heat through these two mechanisms.

The Heat Equation

In a solid material advection can scarcely occur at all so that it becomes reasonable to ignore convection. It becomes plausible to argue that in the interior at least conduction comprises the principle mechanism for the diffusion of heat. Many more practically inclined commentators argue for and employ the name *heat conduction equation* rather than heat equation. The classical name has stood for so long and is so widespread however that we will adopt it, its debatable aptness notwithstanding.

The Heat Equation

The temperature at a point (x,y,z) in the region of interest and at the time t is commonly denoted $u(x,y,z,t)$. Of course temperature is not defined at a literal *point* since it is an average. Nonetheless we may regard $u(x,y,z,t)$ as informing us of the average thermal energy of all of the particles very close to the point (x,y,z) at time t. As the heat diffuses through the region by the process of heat conduction, the heat equation predicts the spatio-temporal change of the temperature:

$$\frac{\partial u}{\partial t} = \frac{k}{c_p \rho}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) = \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right)$$

The Heat Equation

This PDE is a little too much to take on at present. Let us assume that the temperature profile is such that u does not vary with y and z. Obviously this is commonly an unjustifiable assumption but it does have the virtue of making the problem much easier. Accordingly the temperature is now the unknown function of two variables, $u(x,t)$, satisfying the one-dimensional heat equation:

$$\frac{\partial u}{\partial t} = \frac{k}{c_p \rho}\frac{\partial^2 u}{\partial x^2} = \alpha \frac{\partial^2 u}{\partial x^2}$$

where α is the *thermal diffusivity* of the material in the neighbourhood of point x having units of m^2/s.

Normalisation

Again it is possible to consider a change of time scale, letting

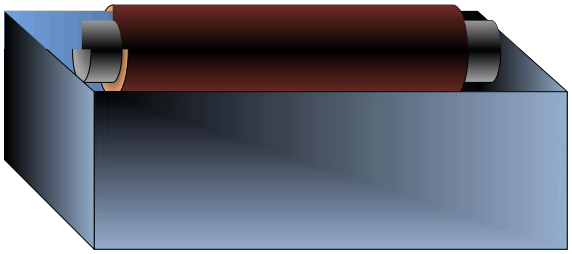
$$\tau = \alpha t$$

so that we again obtain the one-dimensional heat equation, but now with simpler parameters:

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}$$

The Heat Equation

It is customary in first discussing the solution by numerical means of the heat equation to consider a rather idealised problem and I propose to follow this custom. Accordingly, consider a metal rod of length l which is laterally insulated and where both ends are maintained at a constant temperature of 20°C.



The Heat Equation

We must find the temperature profile of the rod given that the initial temperature profile is described by: $u(x,0) = 20 + 15\sin\left(\frac{\pi x}{l}\right)$

The problem therefore is to solve: $\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}$

subject to boundary conditions:

$$u(0,\tau) = u(l,\tau) = 20^{\circ}\mathrm{C} \quad \text{for all } \tau \geq 0$$

and to initial condition:

$$u(x,0) = 20 + 15\sin\left(\frac{\pi x}{l}\right) \quad \text{for } 0 \leq x \leq l$$

Method of Finite differences

Break the rod down into N equally sized pieces. Let $h = l/N$ be the length of each of these pieces. Let k be a small timestep. We look for the approximate solution at positions $x = ih$ and scaled-times $\tau = jk$ only. Using central differences let:

$$u_{ij} = u(ih, jk) \qquad \left(\frac{\partial u}{\partial x}\right)_{ij} = \frac{u_{i+1,j} - u_{i-1,j}}{2h}$$

$$\left(\frac{\partial u}{\partial \tau}\right)_{ij} = \frac{u_{i,j+1} - u_{i,j-1}}{2k} \qquad \left(\frac{\partial^2 u}{\partial x^2}\right)_{ij} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}$$

The Heat Equation

The normalised heat equation gives:

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2} \qquad \left(\frac{u_{i,j+1} - u_{i,j-1}}{2k}\right) = \left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}\right)$$

$$u_{i,j+1} = u_{i,j-1} + \left(\frac{2k}{h^2}\right)\left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}\right)$$

We obtain a second order recursion where the approximate values at time step j+1 depend on the approximate values at the two previous time steps. To start this iteration we need initial conditions, i.e. values at the 0th time step, which we have, and we need values at the 1st time step, which we do not have.

Method of Finite differences

If instead we use the less accurate forward difference approximation for the time derivative:

$$\left(\frac{\partial u}{\partial \tau}\right)_{ij} = \frac{u_{i,j+1} - u_{i,j}}{k}$$

$$\left(\frac{u_{i,j+1} - u_{i,j}}{k}\right) = \left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}\right)$$

$$u_{i,j+1} = u_{i,j} + \left(\frac{2k}{h^2}\right)\left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}\right)$$

This time we have a first order recursion which we can in fact solve, since we have the necessary initial conditions.

Method of Finite differences

However, a Von Neumann Stability Analysis of this second scheme yields the approximate stablity condition:

$$\frac{k}{h^2} \leq \frac{1}{2}$$

The problem here is that we will have to take timestep k which are no more than half the square of the x-steps h. For accuracy h must be reasonably small so that its square becomes very small indeed. Crank and Nicolson proposed a rather simple but clever idea for circumventing this problem.

Crank-Nicolson Method

The idea is to use the forward difference approximation for the time derivative (since the central difference leads to a recursion which is difficult to start) and continue to use a central difference approximation for second spatial derivative, but this time to employ the average of the value at the j^{th} and $(j+1)^{\text{th}}$ time steps:

$$\left(\frac{u_{i,j+1}-u_{i,j}}{k}\right)=\left(\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{2h^2}\right)+\left(\frac{u_{i+1,j+1}-2u_{i,j+1}+u_{i-1,j+1}}{2h^2}\right)$$

Crank-Nicolson Method

A Von Neumann Stability Analysis in this case imposes *no conditions* on k/h^2 for stability. We therefore expect to be able to use reasonably large timesteps. We do pay a price however.

$$u^{(j)}=\begin{bmatrix} u_{1,j} \\ u_{2,j} \\ \vdots \\ u_{N-2,j} \\ u_{N-1,j} \end{bmatrix}$$

Again this is the set of sample values at time jk which are not know. Of course $u_{0,j}$ and $u_{N,j}$ are known from the boundary conditions, being 20°C in this case.

The Heat Equation

$$u_{i,j+1} - \left(\frac{k}{2h^2}\right)\left(u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}\right) = u_{i,j} + \left(\frac{k}{2h^2}\right)\left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}\right)$$

Let $r = \frac{k}{h^2}$ then the recursion can be rewritten in the elegant form: $A_1 u^{(j+1)} + b_1^{(j+1)} = A_2 u^{(j)} + b_2^{(j)}$

$$A_1 = \begin{bmatrix} 1+r & -\frac{r}{2} & 0 & \cdots & 0 \\ -\frac{r}{2} & 1+r & -\frac{r}{2} & \cdots & 0 \\ 0 & -\frac{r}{2} & 1+r & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1+r \end{bmatrix} \qquad A_2 = \begin{bmatrix} 1-r & \frac{r}{2} & 0 & \cdots & 0 \\ \frac{r}{2} & 1-r & \frac{r}{2} & \cdots & 0 \\ 0 & \frac{r}{2} & 1-r & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1-r \end{bmatrix}$$

The Heat Equation

$$b_1^{(j+1)} = \begin{bmatrix} -\frac{r}{2}u_{0,j+1} \\ 0 \\ 0 \\ \vdots \\ -\frac{r}{2}u_{N,j+1} \end{bmatrix} \qquad b_2^{(j)} = \begin{bmatrix} \frac{r}{2}u_{0,j} \\ 0 \\ 0 \\ \vdots \\ \frac{r}{2}u_{N,j} \end{bmatrix} \qquad b_2^{(j)} - b_1^{(j+1)} = \begin{bmatrix} 20r \\ 0 \\ 0 \\ \vdots \\ 20r \end{bmatrix} \text{ here}$$

$$A_1 u^{(j+1)} = A_2 u^{(j)} + b_2^{(j)} - b_1^{(j+1)}$$

We note that, starting from the initial and boundary conditions all of the terms on the right will be known. Accordingly the *Crank-Nicolson Method* has left us with the task of solving a system of linear equations and we have already seen how to do this.

Matlab: The Heat Equation

Let us take $N = 10$ and $k = 0.005$. Assume also that the length of the rod is 0.5 m. Hence $h = 0.5/N = 0.05$ and $r = k/h^2 = 2$. Find initial conditions:

```
>> len = 0.5;
>> N = 10, h = len/N, k = 0.005, r = k/(h^2);
>> x = [h:h:0.5-h]';
>> u = 20 + 15*sin(pi*x/len)
```

$$u^{(0)} = \begin{bmatrix} u_{1,0} \\ u_{2,0} \\ \vdots \\ u_{8,0} \\ u_{9,0} \end{bmatrix} = \begin{bmatrix} 24.6353 \\ 28.8168 \\ 32.1353 \\ 34.2658 \\ 35.0000 \\ 34.2658 \\ 32.1353 \\ 28.8168 \\ 24.6353 \end{bmatrix}$$

Matlab: The Heat Equation

Create sparse matrices A_1 and A_2

```
>> e = ones(N-1,1);
>> A1 = spdiags([-(r/2)*e (1+r)*e -(r/2)*e],[-1 0 1],N-1,N-1);
>> A2 = spdiags([(r/2)*e (1-r)*e (r/2)*e],[-1 0 1],N-1,N-1);
```

Create vector $b = b_2 - b_1$, which, as indicated, is independent of j for given boundary conditions.

```
>> b = [20*r; zeros(N-3,1); 20*r];
```

The recursion to be solved is $A_1 u^{(j+1)} = A_2 u^{(j)} + b$

Matlab: The Heat Equation

For first step we must solve the system of equations

$$A_1 u^{(1)} = A_2 u^{(0)} + b$$

```
>> vtemp = (A2*u) + b;
```

$$A_1 u^{(1)} = \begin{bmatrix} 44.1815 \\ 27.9537 \\ 30.9474 \\ 32.8694 \\ 33.5317 \\ 32.8694 \\ 30.9474 \\ 27.9537 \\ 44.1815 \end{bmatrix}$$

Solve by Gauss-Seidel iteration.

$$u^{(1)} = \begin{bmatrix} 23.8087 \\ 27.2446 \\ 29.9713 \\ 31.7220 \\ 32.3252 \\ 31.7220 \\ 29.9713 \\ 27.2446 \\ 23.8087 \end{bmatrix}$$

Matlab: The Heat Equation

Obviously we will have to implement a Gauss-Seidel iteration and it seems reasonable to do that in a function M-file called, for example, GaussSeidel.m This function M-file should have two input arguments, A a sparse matrix and b, possibly a sparse vector. It should have one output argument, the vector v satisfying $Av = b$.

```
>> u = GaussSeidel(A1,vtemp);
```

We now iterate the last two Matlab commands.

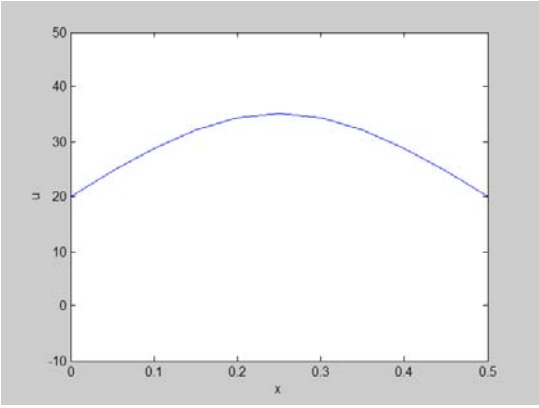
Matlab: The Heat Equation

This iteration yields the rather uninteresting data:

$$u^{(2)} = \begin{bmatrix} 23.1295 \\ 25.9527 \\ 28.1932 \\ 29.6317 \\ 30.1274 \\ 29.6317 \\ 28.1932 \\ 25.9527 \\ 23.1295 \end{bmatrix} \quad u^{(3)} = \begin{bmatrix} 22.5715 \\ 24.8913 \\ 26.7322 \\ 27.9142 \\ 28.3215 \\ 27.9142 \\ 26.7322 \\ 24.8913 \\ 22.5715 \end{bmatrix} \quad u^{(4)} = \begin{bmatrix} 22.1129 \\ 24.0190 \\ 25.5317 \\ 26.5030 \\ 26.8376 \\ 26.5030 \\ 25.5317 \\ 24.0190 \\ 22.1129 \end{bmatrix}$$

Matlab: The Heat Equation

Creating a movie and converting to .avi yields:



Obviously animation is not a particularly interesting way of presenting this data.

Matlab: The Heat Equation

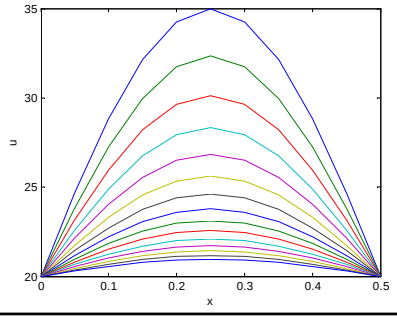
We may present the data in more interesting ways. Suppose I have created the vector **u** containing the initial values at the interior points.

```
>> Storage = zeros(N-1,15);  set space aside to store data for first 15 timesteps
>> Storage(:,1) = u;  put initial interior point values into first column
>> for count = 2:15
vtemp = (A2*u)+b;
u = GaussSeidel(A1,vtemp);  iterate
Storage(:,count) = u;  place current interior point values into current column as identified by count
end
```


Matlab: The Heat Equation

The solutions at all points for given time are found by appending the known boundary values to either end of the vector of values at the interior points.

```
>> Z = [20*ones(1,15);Storage;20*ones(1,15)];  append known boundary values to either end of interior solutions
>> plot([0:h:len],Z)
```

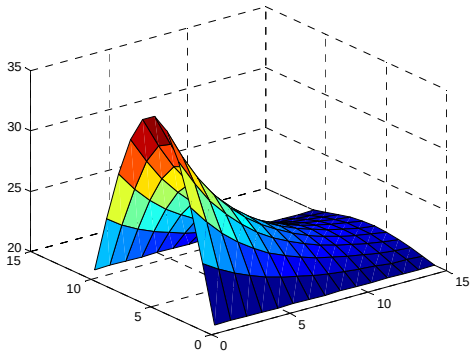


different colours mark the passage of time.

Matlab: The Heat Equation

Alternatively a 3D plot can be created.

```
>> surf(Z)
```



different colours denote height (or temperature value in this case). The blockiness of the plot is due to the relatively course x-step size.

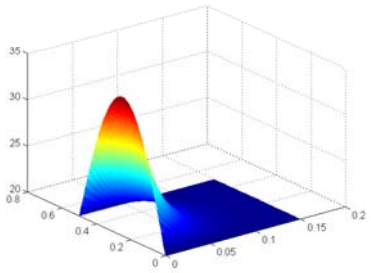
Matlab: The Heat Equation

An obvious problem, apart from its courseness, with the 3D plot obtained is that the axes are not what we would like. The x and y axes are being plotted versus index (i.e. x runs from 1 to 11, 11 being the number of x values per time. y runs from 1 to 15, 15 being the number of time steps taken). Moreover these axes are not labeled. The z axis tickmarks correctly indicate a range from 20 to 35 for the temperature value, but there is no axis label to indicate what is being plotted. Although the boundary lines for the individual surface sections are more honest in terms of clearly presenting the courseness of the approximation, in my opinion they become rather annoying as the grid becomes finer.

Matlab: The Heat Equation

To address these problems firstly repeat calculations with a finer grid, i.e. smaller step sizes, $N = 40$, $k = 0.0005$ and taking FrameNumber = 300 time steps.

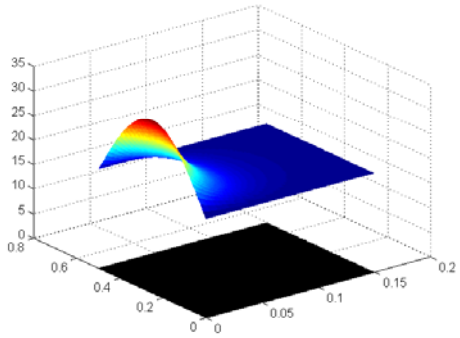
```
>> [T,X]=meshgrid([0:k:(FrameNumber-1)*k],[0:len/N:len]);
>> surf(T,X,Z, 'LineStyle', 'none')        the axes now have the
correct range and the boundary lines are removed
```



Matlab: The Heat Equation

I like the idea of putting a flat surface plot beneath, but I cannot directly turn off the boundary lines.

```
>> hold on
>> handlecolor = pcolor(T,X,Z)        flat surface plot beneath
```

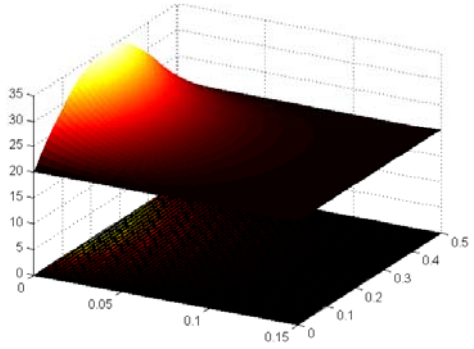


handlecolor will be a *handle* for accessing the flat surface graphics object created which will allow me to turn off lines later.

Matlab: The Heat Equation

Obviously the axis range is inappropriate, the colour is very dark and the orientation is not optimal

```
>> axis([0 0.15 0 0.5 0 35])
>> colormap hot        uses more yellow and red
```



Change of orientation or viewing angle achieved by the **Rotate 3D** tool on the figure window toolbar.

Matlab: The Heat Equation

The labels are very easily fixed.

```
>> xlabel('\tau'), ylabel('x'), zlabel('u')
```

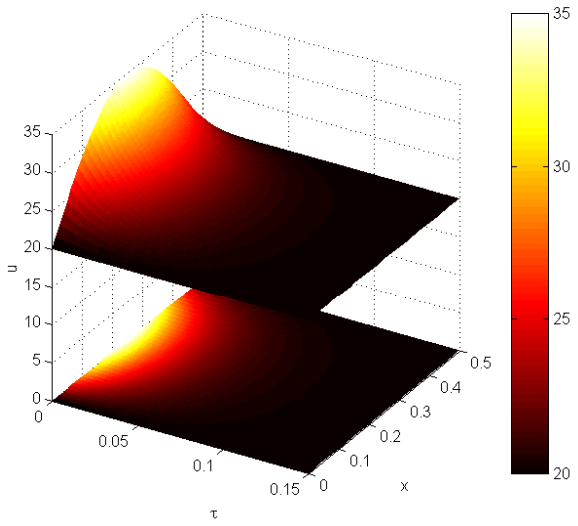
Finally I want to get rid of the boundary lines for the sections on the flat surface. I have a handle, **handlecolor**, for this object which I acquired when I created it. Accordingly I can set its LineStyle property to none, thereby switching off the lines.

```
>> set(handlecolor, 'LineStyle', 'none')    turn off boundary lines
```

Matlab: The Heat Equation

With one last command we obtain the following:

```
>> colorbar
```



Quasilinear second order PDEs

A PDE is said to be *second order* if the higest order partial derivatives involved are second order. A PDE is said to be *quasilinear* if it is linear in the highest order derivatives. The most general quasilinear second order PDE in two independent variables t and x has the form:

$$A\frac{\partial^2 u}{\partial t^2} + 2B\frac{\partial^2 u}{\partial t\,\partial x} + C\frac{\partial^2 u}{\partial x^2} = F\left(t, x, u, \frac{\partial u}{\partial t}, \frac{\partial u}{\partial x}\right)$$

There is a further classification of PDEs of this kind, they are said to be *hyperbolic*, *parabolic* or *elliptic*.

Quasilinear second order PDEs

A second order, quasilinear PDE in two independent variables t and x is said to be *hyperbolic* or of *hyperbolic type* if:

$$AC - B^2 < 0$$

The wave equation is hyperbolic because:

$$\frac{\partial^2 y}{\partial x^2} - \frac{1}{v^2}\frac{\partial^2 y}{\partial t^2} = 0 \qquad A = -\frac{1}{v^2}\ ,\ B = 0\ ,\ C = 1$$

$$AC - B^2 = -\frac{1}{v^2} < 0$$

Quasilinear second order PDEs

A second order, quasilinear PDE in two independent variables t and x is said to be *parabolic* or of *parabolic type* if:

$$AC - B^2 = 0$$

The heat equation is parabolic because:

$$\alpha\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t} \qquad A = 0\ ,\ B = 0\ ,\ C = \alpha$$

$$AC - B^2 = 0$$

Quasilinear second order PDEs

A second order, quasilinear PDE in two independent variables t and x is said to be *elliptic* or of *elliptic type* if:

$$AC - B^2 > 0$$

We have yet to consider an example of an elliptic equation. We have however seen that the numerical procedures for solving hyperbolic and parabolic PDEs, whilst sharing many features, are nonetheless distinct. We have to present another set of procedures for solving elliptic PDEs and do so through the vehicle of considering arguably the most famous such equation, *Laplace's equation*.

Laplace's Equation

Laplace's equation is: $$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = 0$$

In the event that the temperature settles to a constant value at each point in space Laplace's equation emerges from the heat equation by formally asserting that u does not depend on time t:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = \frac{1}{\alpha}\frac{\partial u}{\partial t}$$

Accordingly Laplace's equation is the *steady-state heat equation*. It transpires however that Laplace's equation has much more widespread use.

Laplace's Equation

Assuming all fields are static (i.e. not varying with time) and that a region is free, both of electric charge and of electric current, Maxwell's theory of electromagnetics implies that the electric field, $\boldsymbol{E}$, in the region satisfies the following equations:

$$\nabla \cdot \boldsymbol{E} = 0 \quad , \quad \nabla \times \boldsymbol{E} = 0$$

You may not have seen this notation before, but that does not really matter as I am not persisting with it. The second of these equations asserts that the curl of the vector field **E** is zero. The vector field is said to be *irrotational* and a very important mathematical result declares that, on a simply-connected region, every irrotational vector field is the gradient of a scalar field which is called the *potential*. A vector field which is the gradient of a scalar field is called *conservative*.

Laplace's Equation

So, assuming all fields are static and that a region is free, both of electric charge and of electric current, then there exists a scalar function called the *electric potential* and denoted φ such that: $\nabla \cdot \boldsymbol{E} = 0 \quad , \quad \boldsymbol{E} = -\nabla\varphi = -\text{grad}(\varphi)$

We saw the gradient back when we were considering optimisation. Translating into simpler notation:

$$\boldsymbol{E} = -\begin{bmatrix} \dfrac{\partial \varphi}{\partial x} \\ \dfrac{\partial \varphi}{\partial y} \\ \dfrac{\partial \varphi}{\partial z} \end{bmatrix} \quad , \quad \frac{\partial}{\partial x}\left(-\frac{\partial \varphi}{\partial x}\right) + \frac{\partial}{\partial y}\left(-\frac{\partial \varphi}{\partial y}\right) + \frac{\partial}{\partial z}\left(-\frac{\partial \varphi}{\partial z}\right) = 0$$

Laplace's Equation

Accordingly the electric potential satisfies Laplace's equation:

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} + \frac{\partial^2 \varphi}{\partial z^2} = 0$$

and from it we can determine the electric field:

$$\boldsymbol{E} = -\begin{bmatrix} \dfrac{\partial \varphi}{\partial x} \\ \dfrac{\partial \varphi}{\partial y} \\ \dfrac{\partial \varphi}{\partial z} \end{bmatrix}$$

Laplace's Equation

You may be aware that the fundamental law of electrostatics, *Coulomb's law*, and the fundamental law for gravitational attraction, *Newton's universal law of gravitation*, have a very similar form, both being inverse square laws. It is not therefore too surprising to learn that the gravitational field associated with a body has virtually exactly the same properties as the electric field associated with a charge. In particular it is conservative, so that it can be expressed as minus the gradient of a scalar function, the gravitational potential and this scalar potential function satisfies Laplace's equation.

Example:

A square metal plate has edge 1 m. The voltage at the bottom, left and right hand edges of the plate is held at 0 V (since these edges are grounded). The voltage at the top edge is given by the formula:

$$u(x,1) = 100\sin(\pi x)$$

Find the voltage throughout the plate. Note that voltage equals *electric potential difference*, namely the difference between the electric potential at the point and that at some reference point. Here we may take the reference point to be any point on the bottom, left or right hand edges since the potential difference is zero there. In other words for this problem we may take it that the voltage at a point equals the electric potential at that point.

Laplace's Equation

Accordingly, given that we are in 2 dimensions the voltage satisfies Laplace's equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

subject to the boundary conditions:

$$u(0,y) = u(1,y) = 0 \text{ V} \quad \text{for all } 0 \le y \le 1$$

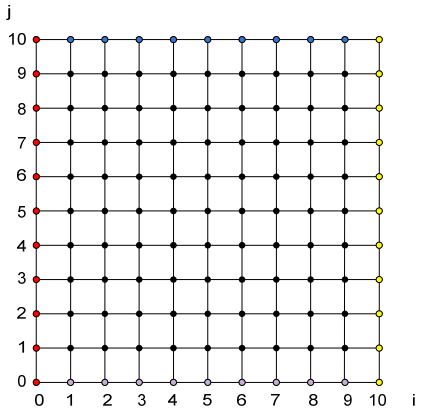
$$u(x,0) = 0 \text{ V} \quad \text{for all } 0 \le x \le 1$$

$$u(x,1) = 100\sin(\pi x) \text{ V} \quad \text{for all } 0 \le x \le 1$$

Laplace's Equation

We start in the usual way, covering the plate with a grid, i.e. break it into many smaller squares. Let $h = 1/N$ be the width of each of these smaller squares. Let $k = 1/N$ be the height of each of these smaller squares.

For example, let $N = 10$.



Method of Finite differences

We look for the approximate solution at positions $(x,y) = (ih,jk)$ only, i.e. at the grid points shown on the previous slide. The boundary conditions give the correct values for all of the boundary points of the grid. Using central differences let:

$$u_{ij} = u(ih, jk)$$

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_{ij} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \qquad \left(\frac{\partial^2 u}{\partial y^2}\right)_{ij} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{k^2}$$

Method of Finite differences

We obtain the approximation:

$$\left(\frac{u_{i,j+1}-2u_{i,j}+u_{i,j-1}}{k^2}\right)+\left(\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h^2}\right)=0$$

or given the choice $k = h = 1/N$:

$$u_{i,j+1}-2u_{i,j}+u_{i,j-1}+u_{i+1,j}-2u_{i,j}+u_{i-1,j}=0$$

$$u_{i,j+1}+u_{i,j-1}+u_{i+1,j}+u_{i-1,j}-4u_{i,j}=0$$

We have a very different type of equation to the previous cases. For the case $N = 10$ we have 81 interior points on the grid where the boundary conditions do not explicitly give the solution.

Method of Finite differences

We have to write out the equation for each of these 81 interior points. Let us take the order (1,1), (2,1), … , (9,1), (1,2), (2,2), … , (9,2), (1,3) , … , (1,9), … , (9,9).

$$\begin{aligned}
u_{1,2}+u_{1,0}+u_{2,1}+u_{0,1}-4u_{1,1}&=0 && (1,1)\\
u_{2,2}+u_{2,0}+u_{3,1}+u_{1,1}-4u_{2,1}&=0 && (2,1)\\
&\vdots && \vdots\\
u_{9,2}+u_{9,0}+u_{10,1}+u_{8,1}-4u_{9,1}&=0 && (9,1)\\
u_{1,3}+u_{1,1}+u_{2,2}+u_{0,2}-4u_{1,2}&=0 && (1,2)\\
u_{2,3}+u_{2,1}+u_{3,2}+u_{1,2}-4u_{2,2}&=0 && (2,2)\\
&\vdots && \vdots\\
u_{9,3}+u_{9,1}+u_{10,2}+u_{8,2}-4u_{9,2}&=0 && (9,2)
\end{aligned}$$

Method of Finite differences

We badly need to see a pattern. Let is introduce vectors of values at interior nodes for fixed j, much like we had before:

$$u^{(1)} = \begin{bmatrix} u_{1,1} \\ u_{2,1} \\ \vdots \\ u_{9,1} \end{bmatrix}, \quad u^{(2)} = \begin{bmatrix} u_{1,2} \\ u_{2,2} \\ \vdots \\ u_{9,2} \end{bmatrix}, \quad \cdots, \quad u^{(9)} = \begin{bmatrix} u_{1,9} \\ u_{2,9} \\ \vdots \\ u_{9,9} \end{bmatrix}$$

Rewrite the equation:

$$\left(u_{i-1,j} - 4u_{i,j} + u_{i+1,j}\right) + u_{i,j-1} + u_{i,j+1} = 0$$

The bracketed terms are either elements of $u^{(j)}$ or boundary values.

Method of Finite differences

Stacking the equations for fixed j as i runs from 1 to 9 gives:

$$\begin{bmatrix} -4 & 1 & 0 & \cdots & 0 & 0 \\ 1 & -4 & 1 & \cdots & 0 & 0 \\ 0 & 1 & -4 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -4 & 1 \\ 0 & 0 & 0 & \cdots & 1 & -4 \end{bmatrix} u^{(j)} + u^{(j-1)} + u^{(j+1)} = \begin{bmatrix} -u_{0,j} \\ 0 \\ 0 \\ \vdots \\ 0 \\ -u_{10,j} \end{bmatrix}$$

Following Kreyzig let us denote the matrix appearing here by B.

Method of Finite differences

Now stacking the previous blocks of equations for j running from 1 to 9 gives:

$$\begin{bmatrix} B & I & 0 & \cdots & 0 & 0 \\ I & B & I & \cdots & 0 & 0 \\ 0 & I & B & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & B & I \\ 0 & 0 & 0 & \cdots & I & B \end{bmatrix} \begin{bmatrix} u^{(1)} \\ u^{(2)} \\ u^{(3)} \\ \vdots \\ u^{(8)} \\ u^{(9)} \end{bmatrix} = \begin{bmatrix} b_1 - u^{(0)} \\ b_2 \\ b_3 \\ \vdots \\ b_8 \\ b_9 - u^{(10)} \end{bmatrix} \qquad b_j = \begin{bmatrix} -u_{0,j} \\ 0 \\ 0 \\ \vdots \\ 0 \\ -u_{10,j} \end{bmatrix}$$

Following Kreyzig let us denote the matrix appearing here by A. Evidently we are faced with a sparse system of linear equations.

Method of Finite differences

As the system of equations is sparse and as it is relatively large (81 equations in this case) an indirect or iterative method for solving these equations is preferable as it avoids back-substitution. In particular we will consider the Gauss-Seidel method. In this context, where we are employing the Gauss-Seidel method to solve a system of linear equations which has arisen in attempting to solve a PDE by finite differences, the method is also known as *Liebmann's method*.

Method of Finite differences

Using the boundary conditions:

$$b_j = \begin{bmatrix} -u_{0,j} \\ 0 \\ 0 \\ \vdots \\ 0 \\ -u_{10,j} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad u^{(0)} = \begin{bmatrix} u_{1,0} \\ u_{2,0} \\ \vdots \\ u_{9,0} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad u^{(10)} = \begin{bmatrix} u_{1,10} \\ u_{2,10} \\ \vdots \\ u_{9,10} \end{bmatrix} = \begin{bmatrix} 30.9017 \\ 58.7785 \\ 80.9017 \\ 95.1057 \\ 100.0000 \\ 95.1057 \\ 80.9017 \\ 58.7785 \\ 30.9017 \end{bmatrix}$$

Matlab: Laplace's Equation

To begin we initialise some parameters:

```
>> len = 1; , N = 10; , h = len/N;  M = (N-1)^2;  initialise
parameters, M equals number of interior points in grid
```

We must set up the matrix **A** and the vector **b**. **A** is sparse and it is nearly tridiagonal, so a way of setting up **A** is as follows:

```
>> e = ones(M,1);  create vector of M ones
>> A = spdiags([e e -4*e e e],[-(N-1) -1 0 1 (N-1)],M,M);
```

Sparse matrix **A** has -4 as every element on main diagonal and 1 as every element on super- and sub-diagonals and (N-1), -(N-1) diagonals.

Method of Finite differences

For example, for N = 4, this code produces the following matrix (albeit in sparse form):

$$\left[\begin{array}{ccc|ccc|ccc}
-4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & -4 & \boxed{1} & 0 & 1 & 0 & 0 & 0 \\
\hline
1 & 0 & \boxed{1} & -4 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & -4 & \boxed{1} & 0 & 1 \\
\hline
0 & 0 & 0 & 1 & 0 & \boxed{1} & -4 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4
\end{array}\right]$$

Method of Finite differences

A is therefore correct apart from the indicated 1s which are in error. These erroneous 1s occur in predictable locations and can easily be set to the correct value of 0 in every case.

```
>> i_index = [N-1:N-1:M-(N-1)];  row numbers where first of each pair of erroneous 1s occurs
>> A(i_index,i_index+1) = 0;
>> A(i_index+1,i_index) = 0;
```

Matlab: Laplace's Equation

For this example, with the zero conditions on three boundaries, the equations take the form $Au = b$, u being the vector of unknown values at the interior points, A being the matrix just created and b being the vector

```
>> utop = 100*sin(pi*[h:h:len-h]'/len);
>> b = [zeros(M-N+1;-utop];  create b
```

$$b = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ -u^{(N+1)} \end{bmatrix}$$

Note the transpose to make utop a column vector. Now solve $Au = b$ using Gauss-Seidel.

Matlab: Laplace's Equation

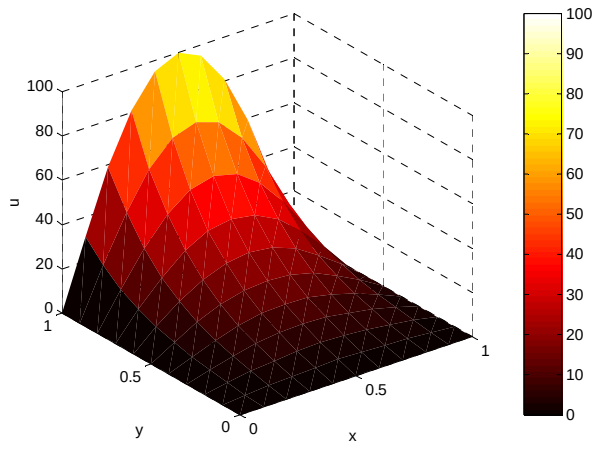
We obtain for the interior points:

$$\begin{bmatrix} u_{1,9} & \cdots & u_{9,9} \\ \vdots & & \vdots \\ u_{1,1} & \cdots & u_{9,1} \end{bmatrix} = \begin{bmatrix} \left(u^{(9)}\right)' \\ \vdots \\ \left(u^{(1)}\right)' \end{bmatrix} =$$

$$\begin{bmatrix} 22.5898 & 42.9684 & 59.1409 & 69.5244 & 73.1022 & 69.5244 & 59.1409 & 42.9684 & 22.5898 \\ 16.4892 & 31.3644 & 43.1693 & 50.7486 & 53.3602 & 50.7486 & 43.1693 & 31.3644 & 16.4892 \\ 12.0027 & 22.8304 & 31.4234 & 36.9404 & 38.8415 & 36.9404 & 31.4234 & 22.8304 & 12.0027 \\ 8.6910 & 16.5314 & 22.7535 & 26.7483 & 28.1248 & 26.7483 & 22.7535 & 16.5314 & 8.6910 \\ 6.2302 & 11.8505 & 16.3108 & 19.1744 & 20.1612 & 19.1744 & 16.3108 & 11.8505 & 6.2302 \\ 4.3791 & 8.3296 & 11.4647 & 13.4775 & 14.1711 & 13.4775 & 11.4647 & 8.3296 & 4.3791 \\ 2.9567 & 5.6240 & 7.7408 & 9.0999 & 9.5682 & 9.0999 & 7.7408 & 5.6240 & 2.9567 \\ 1.8238 & 3.4690 & 4.7747 & 5.6130 & 5.9018 & 5.6130 & 4.7747 & 3.4690 & 1.8238 \\ 0.8693 & 1.6536 & 2.2760 & 2.6755 & 2.8132 & 2.6755 & 2.2760 & 1.6536 & 0.8693 \end{bmatrix}$$

Matlab: Laplace's Equation

Presenting results as before shows a rather crude approximation for such a course grid:



Matlab: Laplace's Equation

A finer grid corresponding to $N = 50$ gives (with far more numerical effort):