

# Modeling Heat Flow \*

Padma Raghavan

January 15, 2009

## 1 Modeling Heat Flow

An incredibly large number of different physical phenomena are governed by “partial differential equations” or PDEs. Examples include heat transfer, fluid-flow, and the elastic deformation of materials. Engineering design and scientific research rely to a large extent on modeling such phenomena to help design new products, discover new applications of materials etc. For example, modeling heat-transfer plays a role from semiconductor design to the design of nuclear reactors. Modeling the bending of materials (elasticity) has applications in the design of nanoscale structures to rockets.

The “heat-equation” makes an interesting example. It represents the larger class of PDE applications but it is simple enough to describe at a high level. Its solution requires a collection of “basic operations” or kernels – these kernels are some of numeric algorithms we have studied. There are also several alternative algorithms for each basic operation and the “best” might be problem dependent. In short it is a simple representative example that embodies many of the challenges that you will find in more complex scientific and engineering applications. The next several sections introduce this application from a numeric perspective.

### 1.1 On continuous problems solved discretely

Lets start with a function  $f(x)$  in one variable and the approximation of its derivative. If  $f(x)$  is a smooth function, we can use its Taylor’s series expansion to derive difference rules for its derivatives. The Taylor’s expansion for  $f(x)$  is:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \dots \quad (1)$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \dots \quad (2)$$

---

\* ©Copyright 2005, by Padma Raghavan.

Solving for  $f'(x)$  in Equation 1 gives the forward difference formula

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

Solving for  $f'(x)$  in Equation 2 gives the backward difference formula:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}.$$

Subtracting the second series from the first gives the centered difference formula:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{(2h)}.$$

You can also use the two series to get a difference formula for the second derivative  $f''(x)$ . Adding the two series gives:

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

It is easily observed that both difference formulas for derivatives, have associated “discretization” errors. This error is quite different from and in addition to the “round-off” errors that are inherent in the floating-point representation of the real number line. Furthermore, the methods can be “stable” or “unstable.” Integration is a smoothing process and is typically stable; however, for differentiation small perturbations in the data can cause large changes in the result. These types of errors and stability issues can have significant implications for the application.

## 1.2 The Heat Equation: An Introduction

Consider the diffusion of heat in a bar of length  $L$  whose left end is maintained at a temperature of  $\alpha$  and the right end at  $\beta$ . Let the initial temperature distribution be given by  $g(x)$ . Assume the bar is thin enough so you can consider it a one-dimensional object – modeled as a length along the  $x$ -axis. Now to some notation:

- Let  $u$  be a function of time  $t$  and space  $x$ .
- Let  $u_x$  denote  $\partial u / \partial x$  and let  $u_t$  denote  $\partial u / \partial t$ .
- Let  $u_{xx} = \partial^2 u / \partial x^2$  and  $u_{xy} = \partial^2 u / \partial x \partial y$  etc.
- The temperature at point  $x_i$  at time  $t_m$  is denoted by  $u(t_m, x_i)$
- We will also use  $u_i^m$  interchangeably with  $u(t_m, x_i)$  to denote the temperature at point  $x_i$  at time  $t_m$ .

The heat equation models the temperature as it changes over time:

$$\begin{aligned} u_t &= cu_{xx}, \quad 0 < x < L, \quad t > 0 \\ u(t, 0) &= \alpha, \quad u(t, L) = \beta, \quad \text{boundary condition} \\ u(0, x) &= g(x), \quad \text{initial condition} \end{aligned}$$

The constant  $c$  in the first equation governs the rate of diffusion and depends on the physical properties such as specific heat, conductivity, density etc. The solution  $u$  to this PDE gives the temperature distribution as a function of both space and time. Our goal is to compute  $u(t, x)$  given the PDE  $u_t = cu_{xx}$  and the initial and boundary conditions.

This is perhaps the simplest example of a PDE. A more realistic form would use 2 or more space dimensions but the 1-dimensional model is the best starting point to illustrate the numeric solution process.

Recall that we have a difference formula for approximating the second derivative of a function  $f(x)$ . Let the interval  $[0, L]$  be partitioned into  $n + 1$  subintervals of size  $\Delta x$  such that  $x_0 = 0$ ,  $x_1 = 1\Delta x$ ,  $x_i = i\Delta x$ , and finally  $x_{n+1} = (n + 1)\Delta x = L$ .

Now at time  $t$ ,

$$u_{xx}(t, x_i) \approx \frac{u(t, x_{i+1}) - 2u(t, x_i) + u(t, x_{i-1}))}{(\Delta x)^2}$$

where  $x_i = i\Delta x$ ,  $0 \leq i \leq n + 1$ .

Using this in the heat equation we get:

$$u_t(t, x_i) = c/(\Delta x)^2 \{u(t, x_{i+1}) - 2u(t, x_i) + u(t, x_{i-1})\}$$

Once again, we can use a difference formula for  $u_t$ . Let the time dimension be discretized in intervals of  $\Delta t$  and let  $t_m = m\Delta t$ . Using the forward difference formula,  $u_t(t_m, x_i) = (1/(\Delta t))\{u(t_{m+1}, x_i) - u(t_m, x_i)\}$ . Using this equation in the left hand side gives:

$$(1/(\Delta t))\{u(t_{m+1}, x_i) - u(t_m, x_i)\} = c/(\Delta x)^2 \{u(t, x_{i+1}) - 2u(t, x_i) + u(t, x_{i-1})\}.$$

**Let  $u(t_m, x_i)$  be denoted by  $u_i^m$ .** The last equation can be rewritten as:

$$u_i^{m+1} = u_i^m + \mu \{u_{i+1}^m - 2u_i^m + u_{i-1}^m\}, \quad \mu = \frac{c\Delta t}{(\Delta x)^2}.$$

The boundary conditions give  $u_0^m = \alpha$  and  $u_{n+1}^m = \beta$  for  $m = 0, 1, 2, \dots$ . The initial condition gives  $u_i^0 = g(x_i)$  for  $i = 1, 2, \dots, n$ . Now the solution can be computed one time step at a time. That is compute  $u_i^1$  for  $i = 1, 2, \dots, n$ , use these values to compute  $u_i^2$  for  $i = 1, 2, \dots, n$  and so on. Figure 1 illustrates this process. This solution method is called **explicit** because the values of  $u_i^{m+1}$  are obtained in terms of known values of  $u_i^m$ . Even in this simple 1-dimensional example we used several numeric methods such as the discretization of a continuous function and numeric approximations to a derivative. Additionally, at a time step  $m + 1$ , we obtain a vector  $u^{m+1}$  with component  $u_i^{m+1}$  for  $i = 0, \dots, (n + 1)$ . This is a vector with  $n + 2$  components obtained from the vector at the previous time-step  $u^m$ . Can we write this in matrix -vector terms as  $u^{m+1} = Au^m$ ? Is so, what is the matrix  $A$  and how many arithmetic operations are required per time-step?

**Exercise:** Use the following Matlab function to run simulations for different values of the parameters.

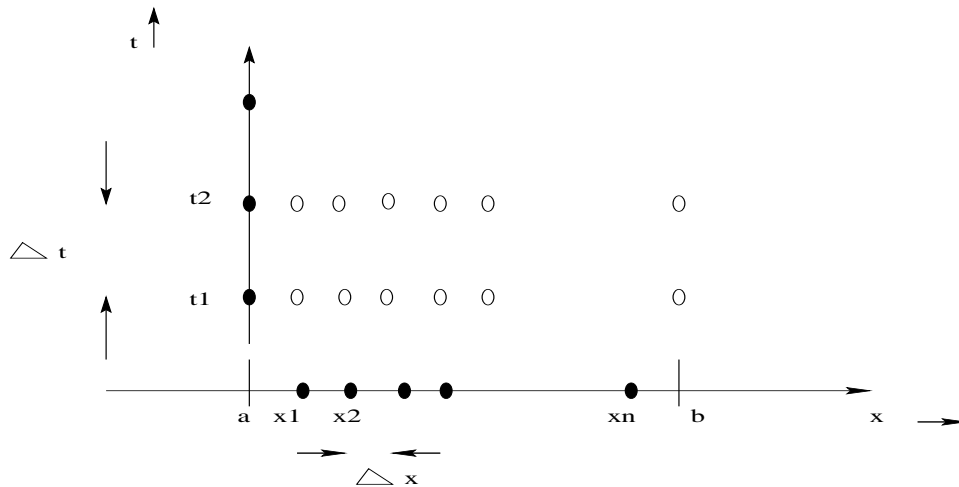


Figure 1: One-dimensional heat-equation, space and time step steps.

```
function ut=one_dim_explicit(c,deltat,deltax,alpha, beta, ntsteps, plotinterval)
%%% This function calculates the evolution of the temperature distribution
% of a one-dimensional object of unit length
% with the constant of diffusion c.
%%% The left end is at alpha, and the right end is at beta degrees
%%% The initial conditions given are by g(x)= sine(pi x);
%%% The time step is deltat
%%% The space step is deltax
%%% The program computes a total of ntsteps
% and plots the temperature every plotinterval steps.
%%% Sample program for CSE/Math 456 written by Padma Raghavan
%%% begin initialization
n = 1/ deltax-1; A =zeros(n+2); x(1) = 0.0;
for i=2:(n+1)
    A(i,i-1) = 1; A(i,i+1) = 1; A(i,i) = -2;
    x(i) = deltax*(i-1); um(i) = sin((x(i)*pi));
end
um(1) = alpha; um(n+2) = beta; x(n+2) = deltax*(n+1); um = um';
mu = c* deltat/(deltax)*(deltax);
fprintf('Initialization: delta t =%f, (deltax*deltax)/(2*c) =%f\n',
deltat, (deltax*deltax)/(2.0*c));
%%% end initialization
%%% plot intial temperature distribution
plot(x, um); title(['plot at time_step ' int2str(0)])
%%% begin simulation
    for time_step=1:ntsteps
```

```

umplus1 = um + mu *A* um;
if (mod(time_step, plotinterval) == 0)
    plot(x, umplus1);
    title(['plot at time_step ' int2str(time_step)])
    disp('Strike a key to continue');
    pause;
end
um = umplus1; total_time = deltat*time_step;
end
%% end simulation

```

## 2 The Explicit Solution of the Heat Equation in Two-Dimensions

Consider heat transfer in a thin-plate which can be modeled as a 2-dimensional object. For simplicity, let the plate be square with unit-length along each dimension. Assume the four edges are maintained at four different constant temperatures  $\alpha, \beta, \gamma$ , and  $\tau$ . Let the initial temperature distribution be given by  $g(x, y)$ . Now the temperature at a point on the plate at an instant in time  $u(t, x, y)$  is a function of two space variables,  $x$  and  $y$  and one time variable  $t$ . The PDE has the form:

$$\begin{aligned}
 u_t &= c(u_{xx} + u_{yy}), \quad 0 < x, y < 1, \quad t > 0 \\
 u(t, 0, y) &= \alpha, \quad u(t, 1, y) = \beta, \quad u(t, x, 0) = \gamma, \quad u(t, x, 1) = \tau, \quad \text{boundary condition} \\
 u(0, x, y) &= g(x, y), \quad \text{initial condition}
 \end{aligned}$$

Consider discretization in space. Perhaps the simplest way is to imagine a  $k \times k$  grid laid out on the unit square with grid-spacing of  $h$  units along either dimension. Now there are  $k^2$  grid points on the plate and these are points at which we will compute the evolution of temperature over time. By increasing  $k$  (alternatively picking a smaller  $h$ ) we can increase the number of grid points and hence get a finer discretization. Figure 2 shows a  $7 \times 7$  grid with  $h = 1/6$ . Let  $u(t, i, j)$  denote the value of the function at grid point  $(i, j)$  at time  $t$ .

Now  $h$  corresponds to  $\Delta x$  in the previous one-dimensional case. Proceeding as before (focus only on the  $x$ -axis):

$$u_{xx}(t, i, j) \approx 1/(h^2)\{u(t, i + 1, j) - 2u(t, i, j) + u(t, i - 1, j)\}.$$

Likewise, for  $u_{yy}$ :

$$u_{yy}(t, i, j) \approx 1/(h^2)\{u(t, i, j + 1) - 2u(t, i, j) + u(t, i, j - 1)\}.$$

In the time-domain we can use the difference formula for  $u_t$ . Let the time dimension be discretized in intervals of  $\Delta t$  and let  $t_m = m\Delta t$ . Now

$$u_t(t_m, i, j) = \frac{u(t_{m+1}, i, j) - u(t_m, i, j)}{\Delta t}.$$

Combining this equation with the earlier ones, we get:

$$u(t_{m+1}, i, j) = u(t_m, i, j) + \frac{c\Delta t}{(h^2)} \{u(t_m, i+1, j) + u(t_m, i-1, j) + u(t_m, i, j+1) + u(t_m, i, j-1) - 4u(t_m, i, j)\}$$

As earlier, by writing  $u(t_m, i, j)$  as  $u_{i,j}^m$  and  $\mu = \frac{c\Delta t}{h^2}$  we get:

$$u_{i,j}^{m+1} = u_{i,j}^m + \mu \{u_{i+1,j}^m + u_{i-1,j}^m + u_{i,j+1}^m + u_{i,j-1}^m - 4u_{i,j}^m\}$$

Once again, at each grid-point values of  $u$  at a previous time-step are used to compute values at the current time-step. This is the explicit scheme for a 2-dimensional problem. The boundary conditions used in these examples are very simple. In general the boundary conditions can be more complicated and could involve derivatives. For example, one edge could be a perfect insulator and hence the rate of change of temperature at that edge would be zero.

**Exercise.** Run the Matlab code for the 1-dimensional problem as:

`one_dim_explicit(1, .00001, .01, 20, 0, 10000, 100)`. The parameters are  $c, \Delta t, \Delta x, \alpha, \beta$ , the total number of time steps, and the number of time steps after which the temperature is shown as a plot. The initial temperature distribution is given by a `sine` function.

Try `one_dim_explicit(1, .6, .01, 20, 0, 100, 1)`. You will see unstable behavior.

Change  $\Delta x$  to a larger value (say .1), you will see how the discretization looks coarse, i.e., the local truncation errors are higher.

### 3 The Explicit Solution of the Heat Equation

As described earlier, the heat equation in one-dimension can be written as:

$$u_t(t, x_i) = c/(\Delta x)^2 \{u(t, x_{i+1}) - 2u(t, x_i) + u(t, x_{i-1})\}$$

If the time dimension is discretized in intervals of  $\Delta t$  and  $t_m = m\Delta t$ , using a forward difference formula for  $u_t$  gives:

$$(1/(\Delta t)) \{u(t_{m+1}, x_i) - u(t_m, x_i)\} = c/(\Delta x)^2 \{u(t, x_{i+1}) - 2u(t, x_i) + u(t, x_{i-1})\}.$$

Writing  $u(t_m, x_i)$  as  $u_i^m$ , we get:

$$u_i^{m+1} = u_i^m + \mu\{u_{i+1}^m - 2u_i^m + u_{i-1}^m\}, \quad \mu = \frac{c\Delta t}{(\Delta x)^2}.$$

This equation can be rewritten in matrix-vector terms.

Let  $\mathbf{u}$  denote the vector  $[u_0, u_1, u_2, \dots, u_{n+1}]^T$  and let  $\mathbf{A}$  denote an  $(n+2) \times (n+2)$  matrix of the form:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Now the iterations can be written as  $\mathbf{u}^{m+1} = \mathbf{u}^m + \mu\mathbf{A}\mathbf{u}^m$ , starting with

$$\mathbf{u}^0 = [\alpha, \mathbf{g}(\mathbf{x}_1), \mathbf{g}(\mathbf{x}_2), \dots, \mathbf{g}(\mathbf{x}_n), \beta]^T.$$

This is an operation of the form “vector = vector + matrix  $\times$  vector.” This is perhaps the simplest operation involving a matrix and a vector. Most scientific and engineering applications require that an operation of this form be performed repeatedly in various contexts. Note that most of the elements of  $A$  are zeroes and the matrix is hence a *sparse* matrix. The operation discussed above can be implemented to utilize sparsity and made to operate only on the nonzeros. Thus one time-step would required approximately  $O(n)$  operations as opposed to  $O(n^2)$  if the matrix were dense.

In two dimensions, the equation at  $(i, j)$  is:

$$u(t_{m+1}, i, j) = u(t_m, i, j) + \frac{c\Delta t}{(h^2)}\{u(t_m, i+1, j) + u(t_m, i-1, j) + u(t_m, i, j+1) + u(t_m, i, j-1) - 4u(t_m, i, j)\}$$

As earlier, by writing  $u(t_m, i, j)$  as  $u_{i,j}^m$  and  $\mu = \frac{c\Delta t}{h^2}$  we get:

$$u_{i,j}^{m+1} = u_{i,j}^m + \mu\{u_{i+1,j}^m + u_{i-1,j}^m + u_{i,j+1}^m + u_{i,j-1}^m - 4u_{i,j}^m\}$$

This equation can also be written in matrix vector form. Number the  $k \times k$  grid points in “natural order,” starting with 1 in the bottom left for  $(0,0)$ , and moving to the right and then up, see Figure 2 for a  $7 \times 7$  example. The matrix  $\mathbf{A}$  will have the form shown in Figure 2, with  $-4$  in the diagonal positions and 1’s in the off diagonal positions. The vector  $\mathbf{u}^0$  will have be initialized to take into account the initial and boundary conditions.

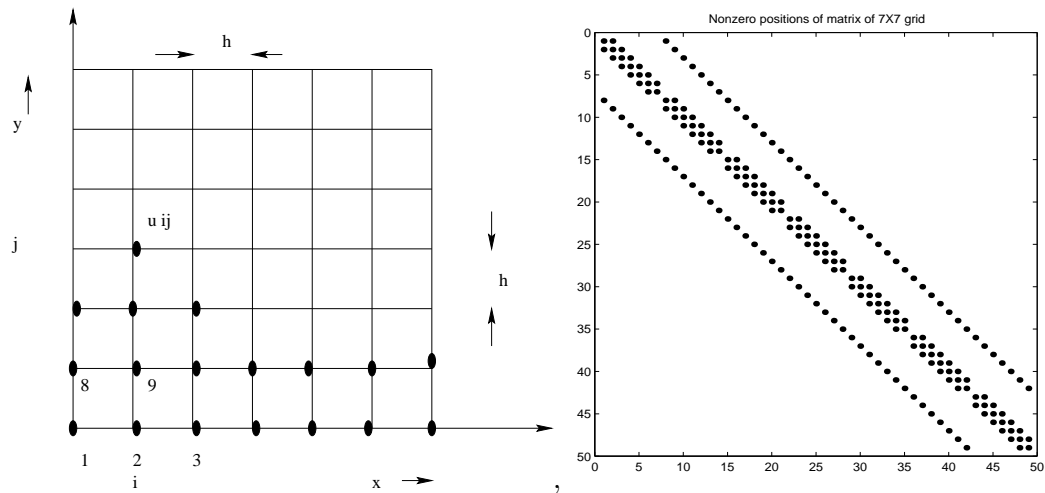


Figure 2: Two-dimensional discretization for the heat equation; the matrix is shown in the lower figure.

### 3.1 Errors

At each grid point, the different equation used for  $u_t$  is first order accurate, i.e., has errors of the form  $\Delta t$ . However, the difference equation for  $u_{xx}$  or  $u_{yy}$  is second order accurate, i.e., has errors of form  $O(h^2)$ . The overall method is thus first-order accurate in time and second order accurate in space. You may think that if the time step and the space step are sufficiently small, then the global error will tend to zero. Unfortunately, this is not the case. Analysis (which is beyond the scope of this class shows) that the explicit method in 1-dimension can become unstable unless  $\Delta t \leq \frac{\Delta x^2}{2c}$ . If you tried the example, you would have noticed unstable behavior for:

```
one_dim_explicit(1, .6, .01, 20, 0, 100, 1).
However, it is stable for:
one_dim_explicit(1, .01, .01, 20, 0, 100, 1). See Figure 3.
```

The stability result shows that time-steps have to be very small to stay in the stable range. Let  $N$  be the vector/matrix dimension for a given model and  $T$  the total number of time-steps. Now the overall cost is given by  $O(T * N)$  and even for relatively small windows of time,  $T$  will have to be considerably large to satisfy the stability condition.

## 4 Modeling Heat Flow Using an Implicit Method

The “implicit” method is derived by using the “backward difference formula.” In 1-dimension, the left hand side is:

$$u_t(t_{m+1}, x_i) = (1/(\Delta t))\{u(t_{m+1}, x_i) - u(t_m, x_i)\}.$$

We use the difference formula for  $u_{xx}(t_{m+1})$ , in the right hand side, to get the following expression (recall that we used a difference formula for  $u_{xx}$  at time  $t_m$  for the explicit method).

$$u_i^m = (1 + 2\mu)u_i^{m+1} - \mu\{u_{i+1}^{m+1} + u_{i-1}^{m+1}\}, \quad \mu = \frac{c\Delta t}{(\Delta x)^2}.$$

If this equation is rewritten in matrix-vector terms, we get  $\mathbf{A}\mathbf{u}^{m+1} = \mathbf{u}^m$  where the matrix  $\mathbf{A}$  is of the form:

$$\begin{bmatrix} (1 + 2\mu) & -1 & 0 & 0 & \cdots & 0 \\ -1 & (1 + 2\mu) & -1 & 0 & \cdots & 0 \\ 0 & -1 & (1 + 2\mu) & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & -1 & (1 + 2\mu) \end{bmatrix}$$

To give you the big picture, I have ignored components  $\mathbf{u}_0, \mathbf{u}_{n+1}$  which are trivially set according to the boundary conditions. Note that now we no longer have a simple matrix-vector multiplication. Instead we have to solve a linear system, one where the matrix is “tri-diagonal.”

In 2-dimensions, the discretization is again modified using the backward difference formula in the left hand side and expressing the right hand side in terms of  $u_{xx}$  and  $u_{yy}$  at time  $m + 1$ . As earlier, by writing  $u(t_m, i, j)$  as  $u_{i,j}^m$  and  $\mu = \frac{c\Delta t}{h^2}$  we get:

$$u_{i,j}^m = u_{i,j}^{m+1} - \mu\{u_{i+1,j}^{m+1} + u_{i-1,j}^{m+1} + u_{i,j+1}^{m+1} + u_{i,j-1}^{m+1} - 4u_{i,j}^{m+1}\}$$

Once again, using the natural numbering of the grid, we can come up with an  $K^2 \times K^2$  matrix  $\mathbf{A}$  such that  $(\mathbf{I} + \mu\mathbf{A})\mathbf{u}^{m+1} = \mathbf{u}^m$ . If the matrix  $\mathbf{B} = (\mathbf{I} + \mu\mathbf{A})$  then  $\mathbf{B}$  has the zero-nonzero pattern shown in Figure 2 with  $(1 + 4\mu)$  along the diagonal and  $-1\mu$  along the off-diagonal nonzeros.

Now the cost per time-step is considerably more because you have to solve a linear system using a method such as Gaussian elimination or Cholesky factorization. Furthermore, such methods will have to utilize the sparsity of the matrix otherwise the cost per time-step would be  $O(N^3)$  where  $N$  is the dimension of the matrix. The storage is also significant, if you store all the zeroes in the matrix then even for a  $10000 \times 10000$  grid, the matrix would be dimension  $10^{10}$  and would require storage of the order of  $8 \times 10^{20}$ . The situation would be considerably worse in 3-dimensions. But the advantages are considerable, the accuracy is still first order in time and second order in space but the method is unconditionally stable! So time-steps can be larger than for the explicit method. Because of this there are a wide variety of methods for solving *sparse linear systems* and it is a very active area of research.

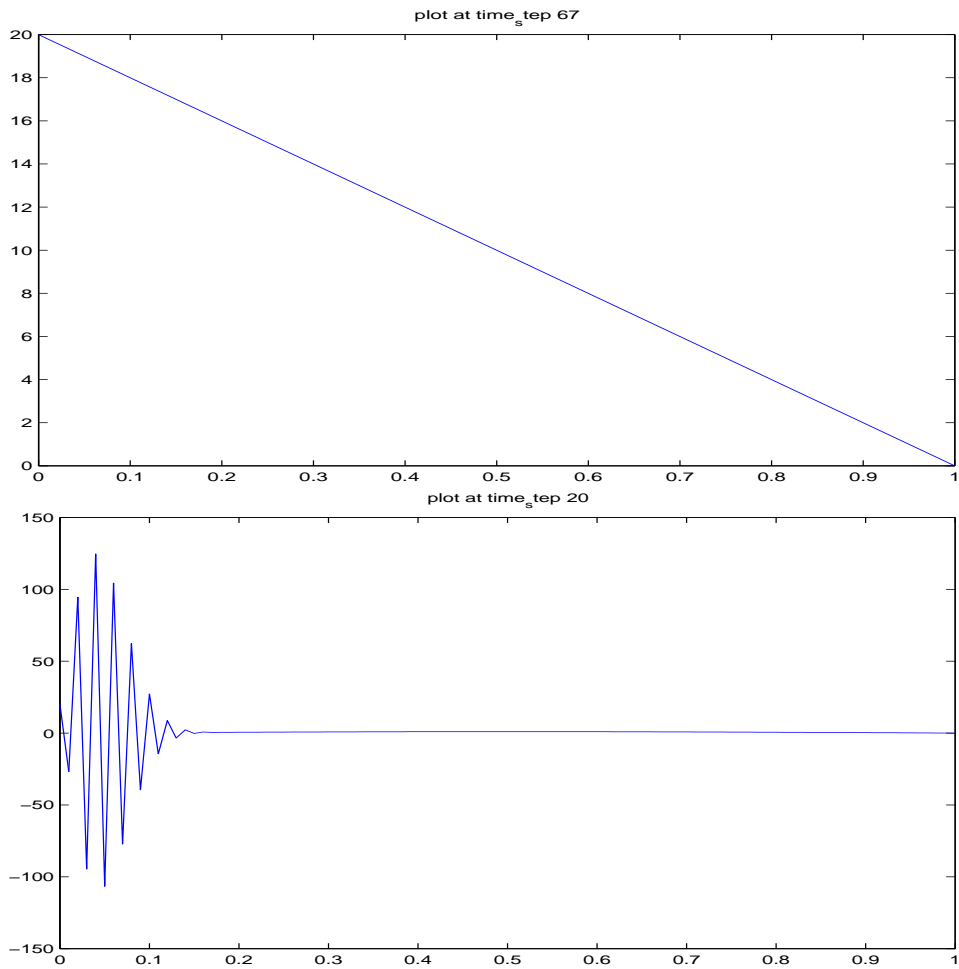


Figure 3: One-dimensional explicit method; stable for  $\delta t = .01$  but unstable for  $\delta t = .6$ .

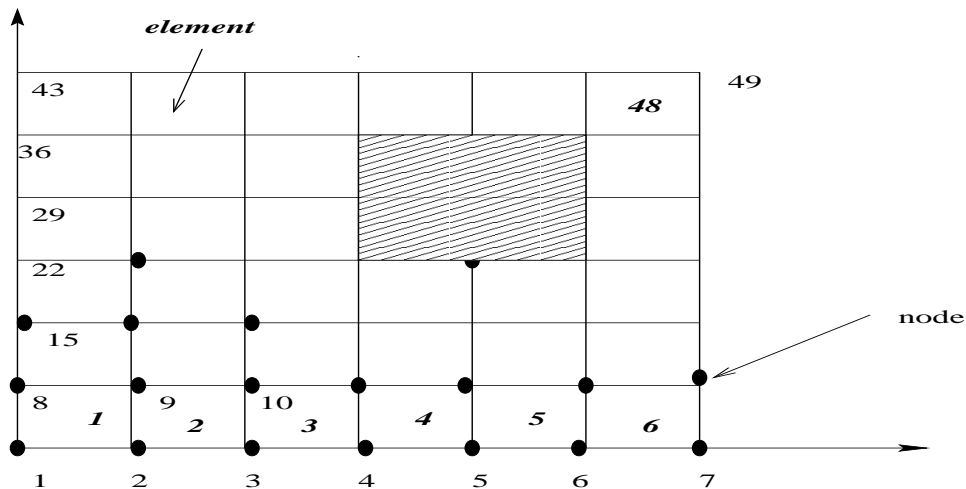


Figure 4: FEM for a regular grid.

## 5 Finite Element Methods (FEM)

So far we have discussed “finite-difference” methods for solving PDEs. Perhaps an even more powerful and more popular paradigm is that of “finite element” methods. Once again it is a computational framework for deriving numeric solutions to PDEs.

Here is a brief overview that should be enough to understand solution techniques. Consider the heat equation as before (in 2-dimensions) but now in steady-state form, i.e.,  $u_t = 0$ . The equation simplifies to  $u_{xx} + u_{yy} = 0$  with additional boundary and initial conditions. Let the domain of interest be the unit square as earlier. The domain is now divided into *elements*. See the example in Figure 4; it is similar to the earlier discretization but you must now focus on the elements. Each element is a small square and there are  $7 \times 7$  nodes or grid-points as earlier. The value of  $u$  is expressed as simple polynomial that is a linear combination of a set of functions of  $x$  and  $y$  (basis functions). As before FEM transforms the PDE into a linear system of the form  $\mathbf{Ax} = \mathbf{b}$ . Elements of  $\mathbf{A}$  are derived by computing a set of definite integrals over elements of the FE mesh. A grid node exchanges information (read as– is connected to) with all other grid points with which it shares an element. Now each node will have nine neighbors (including itself).

For most applications the FE mesh is not regular as shown in the example. In fact one of the strengths of the scheme is the ability to *refine* areas of interest, i.e., have a fine mesh at some regions and a coarser mesh at others. Furthermore the “mesh” can have triangular elements in the 2-dimensional case and tetrahedra in the 3-dimensional case. Computing FE meshes for irregular domains is an art in itself. The point to be noted is that the linear system to be solved typically has a sparse matrix; however the sparsity pattern is not regular. See Figure 5 for a mesh and the sparsity structure of the associated matrix.

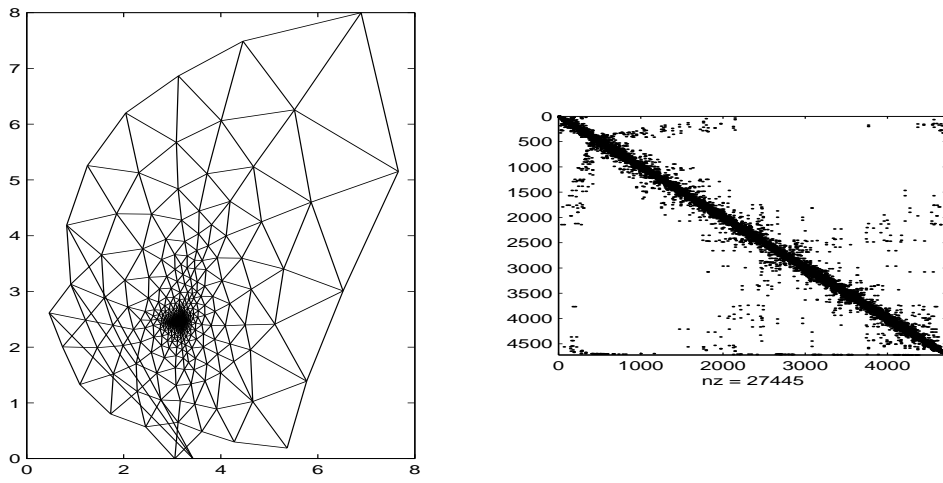


Figure 5: A real FE mesh and its associated matrix.