Partial Differential Equations

Many problems in applied science, physics and engineering are modeled mathematically with partial differential equations (PDEs), which are ODEs involving more than one independent variable. There are three types of PDEs. First is the 1-dimensional diffusion equation:

$$\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2} \to T_t = \kappa T_{xx}$$

The form given above is from the theory of heat transport. The variable T(x,t) is the temperature at position x at time t. The constant is the thermal diffusion coefficient. This is an example of a **parabolic** equation. The time-dependent Schrodinger equation is another example of a diffusion equation(imaginary diffusion constant):

$$i\hbar\psi_t = \frac{\hbar^2}{2m}\psi_{xx} + V(x)\psi$$

The second type of PDE is the one-dimensional wave equation:

$$A_{tt} = v^2 A_{xx}$$

where A(x,t) is the wave amplitude and v is the wave speed. This is an example of a hyperbolic equation.

The third type of PDE is Poisson's equation. In two-dimensions it takes the form:

$$\Phi_{xx} + \Phi_{yy} = -\frac{1}{\varepsilon_0}\rho(x,y)$$

where $\Phi(x,y)$ is the electrostatic potential or its physical equivalent in some system, $\rho(x,y)$ is the charge density and ε_0 is the permittivity of free space. If $\rho(x,y) = 0$ then we have Laplace's equation. This is an example of an **elliptic** equation.

Notice that in each of these examples we have two independent variables, either (x,t) or (x,y) . All the methods we discuss are extendible to higher dimensions, but are easier to understand in the two-dimensional case we discuss first. Formally, a second-order PDE in two independent variables of the form

$$aA_{xx} + bA_{xy} + cA_{yy} + dA_x + eA_y + fA(x,y) + g = 0$$

is classified as

$$\text{hyperbolic}(b^2 - 4ac > 0) \quad , \quad \text{parabolic}(b^2 - 4ac = 0) \quad , \quad \text{elliptic}(b^2 - 4ac < 0)$$

The diffusion and wave equations are similar in that they are usually solved as initial value problems. For the diffusion equation, we might be given an initial temperature distribution T(x,t=0) and want to find T(x,t) for all t > 0. Similarly, for the wave equation we could start with an initial amplitude A(x,t) and velocity dA(x,t=0)/dt of a wave pulse and be asked to find the shape of the wave pulse, A(x,t) for all t > 0.

Besides initial conditions, we need to specify boundary conditions. Say our solution is constrained to the region of space between $x=-L/2 \to x=+l/2$. Boundary conditions are then imposed at these endpoints.

For the diffusion equation we might fix the temperature at the boundaries(Dirichlet Boundary conditions),

$$T(x=-L/2,t)=T_a \quad , \quad T(x=L/2,t)=T_b$$

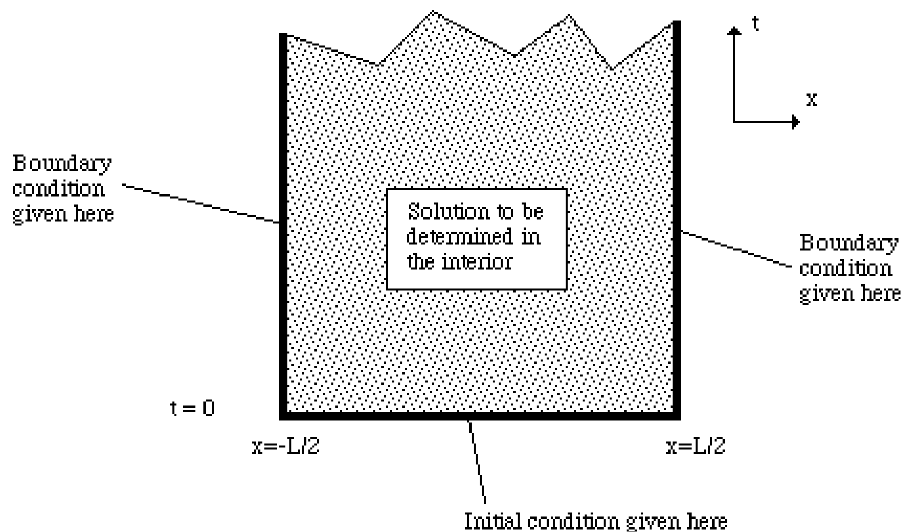
An alternative would be to fix the flux(normal derivative) at the boundaries(Neumann Boundary conditions),

$$-\kappa \left.\frac{dT}{dx}\right|_{x=-L/2}=F_a \quad , \quad -\kappa \left.\frac{dT}{dx}\right|_{x=L/2}=F_b$$

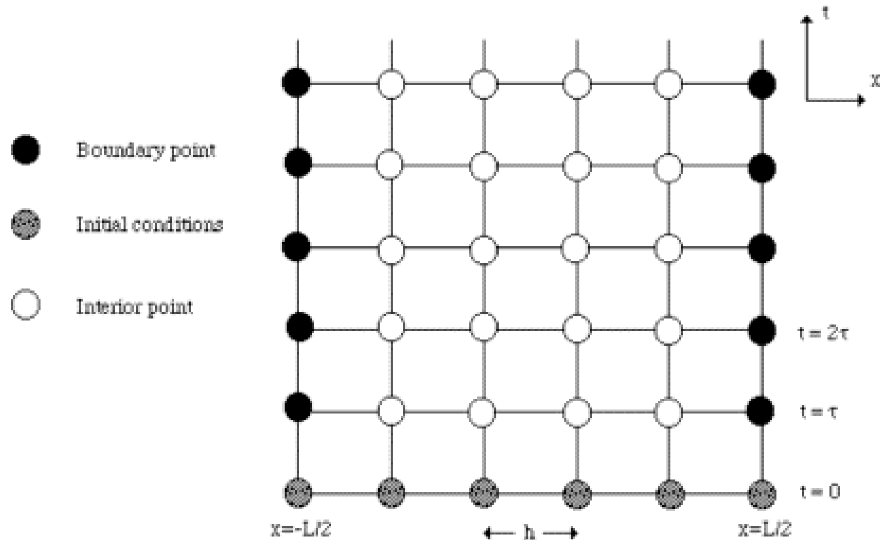
A third type of boundary condition is common in numerical simulations. We equate the function at the two ends(Periodic Boundary conditions)

$$T(x=-L/2,t)=T(x=L/2,t) \text{ and } \left.\frac{dT}{dx}\right|_{x=-L/2}=\left.\frac{dT}{dx}\right|_{x=L/2}$$

So we visualize initial value problems on the x-t plane, as shown below.



The interior region is open-ended; we compute T(x,t) or A(x,t) as far into the future as we want. As with ODEs we discretize time as $t_n = (n-1)\tau$ where τ is the time step and n=1,2,3,4, ... Similarly, we discretize space as $x_i = (i-1)h - L/2$ where h is the grid spacing and i= 1,2,3, ...,N and the grid spacing is h=L/(N-1). A schematic of an initial value problem in discretized form is shown below.



Our job is to determine the unknown values at the interior points (open circles) given the initial conditions (gray circles) and the boundary conditions(filled circles).

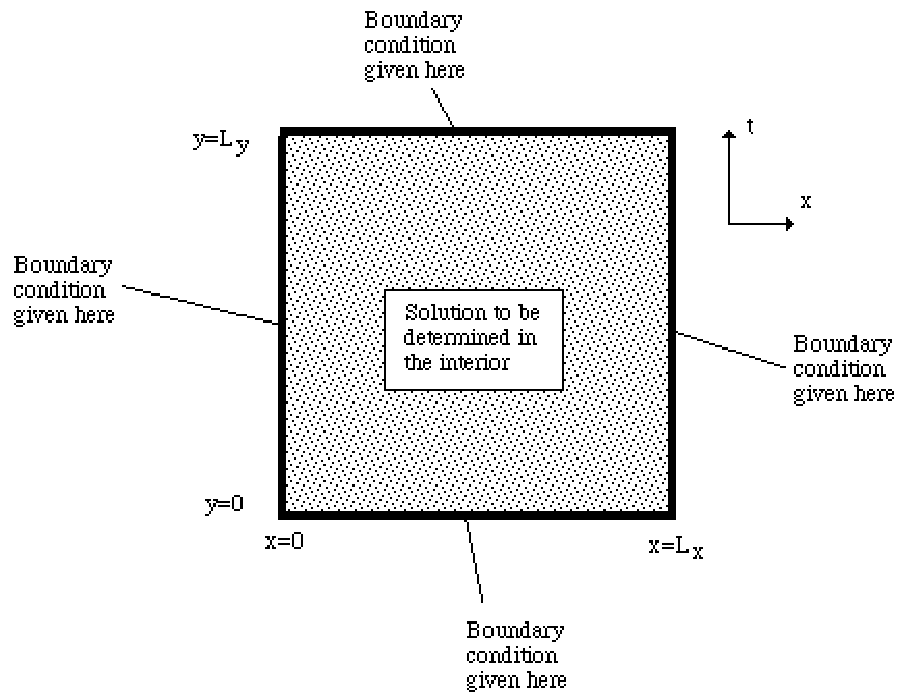
Initial value problems are often solved using marching methods. Starting from the initial condition, we compute the solution one time step into the future. Using this result the solution at the next time step is computed. The algorithm proceeds in this manner and marches forward in time.

Elliptic equations, such as Poisson's equation, in electrostatics, are not initial value problems, but strictly **boundary value problems.** For example, we may be told the potential on the four sides of a rectangle,

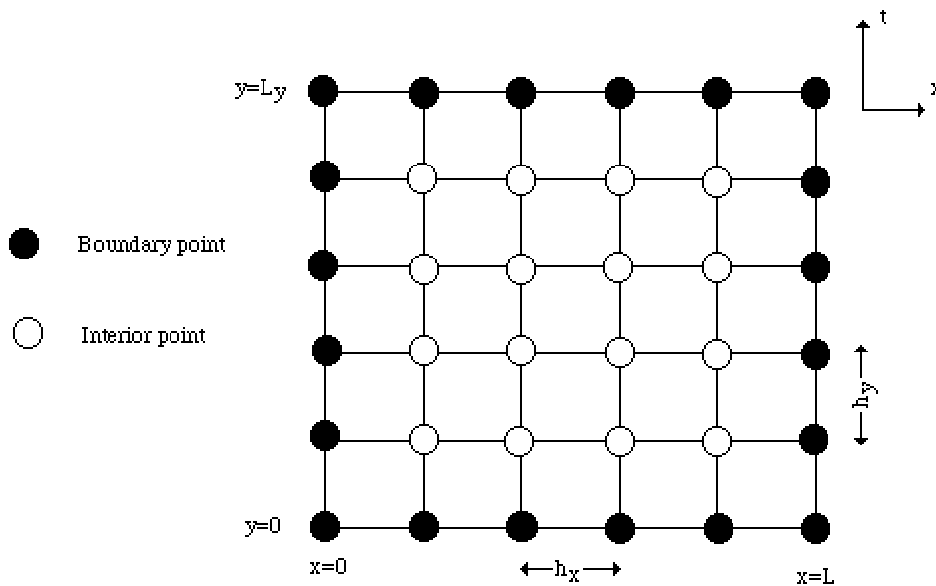
$$\Phi(x=0,y)=\Phi_1\,,\;\Phi(x=L_x,y)=\Phi_2\,,\;\Phi(x,y=0)=\Phi_3\,,\;\Phi(x,y=L_y)=\Phi_4$$

and be asked to solve for $\Phi(x,y)$ at all points inside the rectangle(see figure below). We discretize space as

$x_i = (i-1)h_x$, $y_j = (j-1)h_y$ where h_x and h_y are the x and y grid spacings.



Our task is now to determine Φ at the interior points given the constraints specified by the boundary conditions as in the figure below.



Algorithms for solving boundary value problems are sometimes called **jury** methods. The potential at an interior point is

influenced by all the boundary points; the solution in the interior is a weighted result that reconciles all the demands (constraints) imposed by the boundary.

In this laboratory we will study only one type of partial differential equation, namely, the elliptic equations.

Elliptic Partial Differential Equations (PDEs)

As examples of elliptic PDEs we consider the Laplace, Poisson, and Helmholtz equations. In Cartesian coordinates, the Laplacian of a function is given by

$$\nabla^2 u = u_{xx} + u_{yy}$$

The three equations are of the form:

$$u_{xx} + u_{yy} = 0 \quad \text{Laplace} \qquad u_{xx} + u_{yy} = g(x,y) \quad \text{Poisson}$$

$$u_{xx} + u_{yy} + f(x,y)u = g(x,y) \quad \text{Helmholtz}$$

It is often the case that the boundary values for the functions g and f are known at all points on the sides of a rectangular region R in the plane. In these cases, we can use finite-difference techniques to solve the PDE.

The Laplace Difference Equation

The formula for approximating f'(x) is obtained from

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

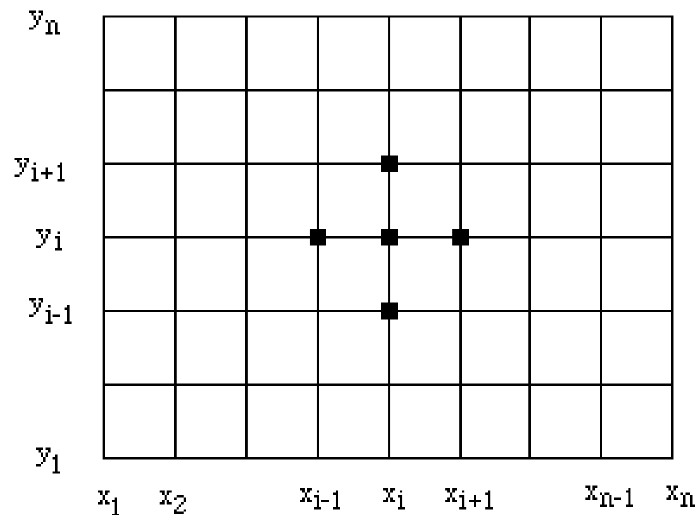
which leads to an approximation for the Laplacian given by

$$u_{xx} + u_{yy} = \frac{u(x+h,y) + u(x-h,y) + u(x,y+h) + u(x,y-h) - 4u(x,y)}{h^2}$$

We assume that the rectangle

$$R = \left\{ (x,y) : 0 \le x \le a \,,\, 0 \le y \le b \ \text{ where } \ b/a{=}m/n \right\}$$

is subdivided into $(n-1)\times(m-1)$ squares with side h where $a = nh, b = mh$ as shown below:



To solve Laplace's equation we impose the condition

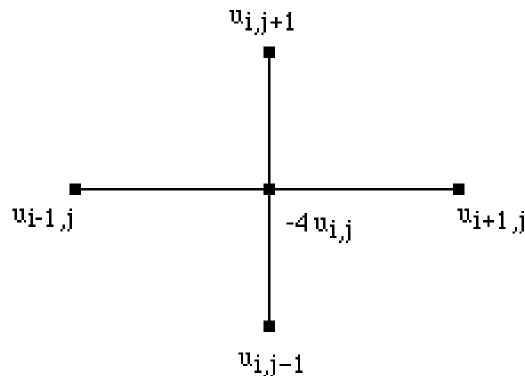
$$\frac{u(x+h,y)+u(x-h,y)+u(x,y+h)+u(x,y-h)-4u(x,y)}{h^2}=0$$

which is valid at all interior grid points $(x,y)=(x_i,y_j)$ for

i = 1,2,3,....,n-1 and j = 2,3,4,....,m-1. The grid points are uniformly spaced $x_{i+1}=x_i+h$, $y_{i+1}=y_i+h$. We then get the 5-point difference formula for Laplace's equation:

$$\nabla^2 u_{ij}=\frac{u_{i+1j}+u_{i-1j}+u_{ij+1}+u_{ij-1}-4u_{ij}}{h^2}=0$$

This formula relates u_{ij} to its four neighboring values u_{i+1j} , u_{i-1j} , u_{ij+1} , u_{ij-1} as shown below:



Rearranging we get the Laplace computational formula (for the relaxation method):

$$u_{ij} = \frac{1}{4}\left[u_{i+1j} + u_{i-1j} + u_{ij+1} + u_{ij-1}\right]$$

The Iterative Solution Method

Suppose that the boundary values u(x,y) are known at the following grid points:

u_{1j} for $2 \le j \le m$-1 (on the left)

u_{i1} for $2 \le i \le n$-1 (on the bottom)

u_{nj} for $2 \le j \le m$-1 (on the right)

u_{in} for $2 \le i \le n$-1 (on the top)

We can then write our equation in form suitable for iteration as

$$u_{ij} = u_{ij} + r_{ij}$$

where

$$r_{ij} = \frac{1}{4}\left[u_{i+1j} + u_{i-1j} + u_{ij+1} + u_{ij-1} - 4u_{ij}\right]$$

for 2 ≤ i ≤ n-1 and 2 ≤ j ≤ m-1.

Starting values for all interior grid points must be supplied. A good choice is the average of the (2n+2m+4) boundary values (call it K). One keeps iterating this equation until the residual term r_{ij} is **reduced to zero**, that is, $|r_{ij}| < \varepsilon$ for all interior grid points.

The **speed of convergence** for reducing all of the residuals to zero is increased by using the method called **successive over-relaxation** (SOR). The SOR method uses the iteration formula $u_{ij} = u_{ij} + \omega r_{ij}$ where the parameter ω lies in the range 1 ≤ ω ≤ 2. The optimal choice for ω is given by

$$\omega = \frac{4}{2 + \sqrt{4 - \left[\cos\frac{\pi}{n-1} + \cos\frac{\pi}{m-1}\right]^2}}$$

Poisson's and Helmholtz's Equations

The iterative formulas for these two equations are:

$$u_{ij} = u_{ij} + r_{ij}$$

where for Poisson:

$$r_{ij} = \frac{1}{4}\left[u_{i+1j} + u_{i-1j} + u_{ij+1} + u_{ij-1} - 4u_{ij} - h^2 g_{ij}\right]$$

and for Helmholtz:

$$r_{ij} = \frac{1}{4}\frac{\left[u_{i+1j} + u_{i-1j} + u_{ij+1} + u_{ij-1} - (4 - h^2 f_{ij})u_{ij} - h^2 g_{ij}\right]}{4 - h^2 f_{ij}}$$

In addition, we can develop a 9-point difference formula, which greatly improves the accuracy of the Laplace computation.

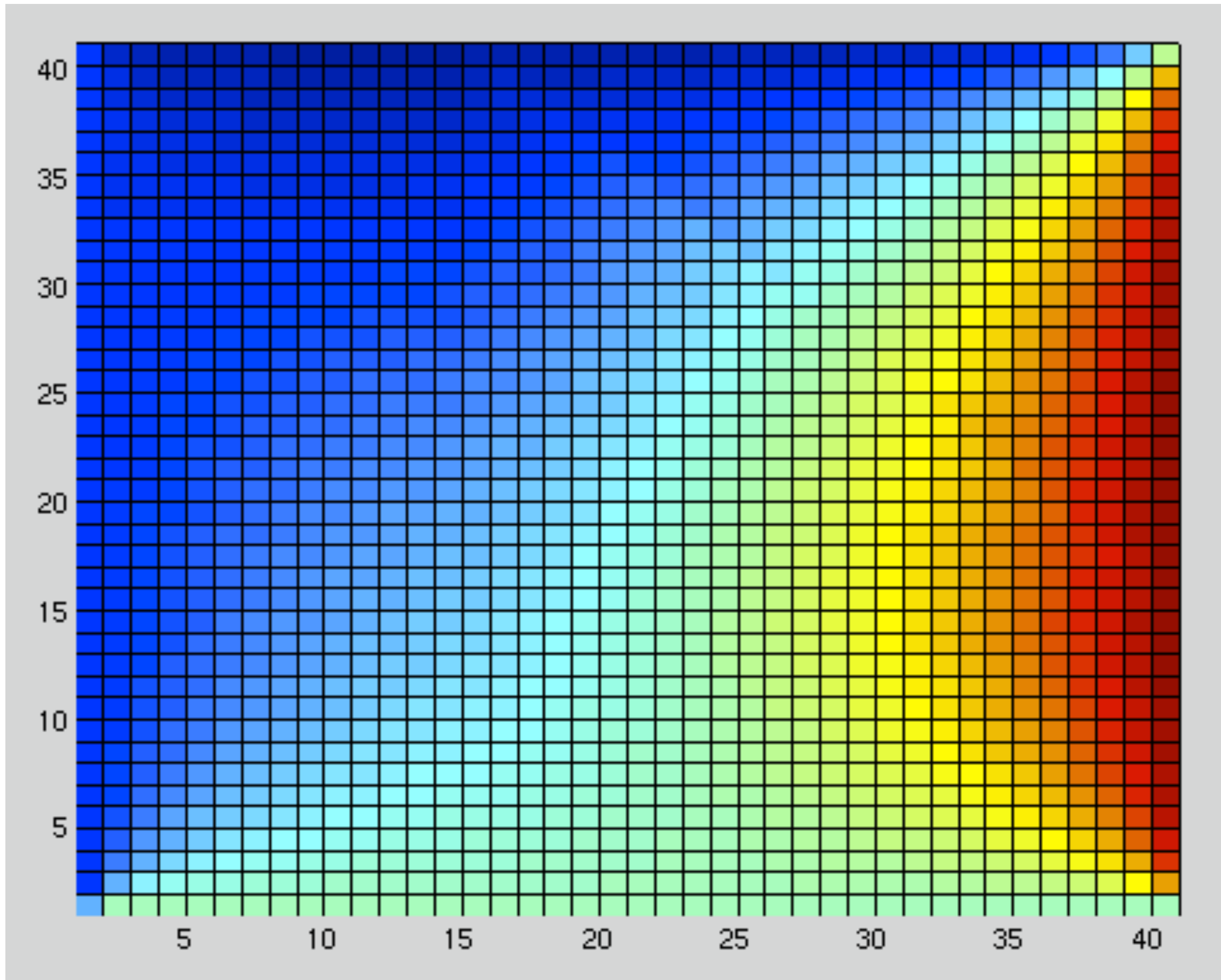
$$\nabla^2 u_{ij} = \frac{u_{i+1j-1} + u_{i-1j-1} + u_{i+1j+1} + u_{i-1j+1} + 4u_{i+1j} + 4u_{i-1j} + 4u_{ij+1} + 4u_{ij-1} - 20u_{ij}}{6h^2} = 0$$

Sample Programs --- Laplace Equation

```
function z = fla1(x)
z = 20;

function z = fla2(x)
z = 180;

function z = fla3(x)
z = 80;

function z = fla4(x)
z = 0;

function U = dirich(a,b,h,tol,max1)
% Dirichlet solution to Laplace's equation.
% f1  is a boundary function, input.
% f2  is a boundary function, input.
% f3  is a boundary function, input.
% f4  is a boundary unction, input.
% a   is the width of [0 a], input.
% b   is the width of [0 b], input.
% h   is the step size, input.
% tol is the tolerance, input.
% max1 is the maximum number of iterations, input.
n = fix(a/h)+1;
m = fix(b/h)+1;
ave = (a*(fla1(0)+fla2(0)) ...
    +  b*(fla3(0)+fla4(0)))/(2*a+2*b);
U = ave*ones(n,m);
for jj=1:m
  U(1,jj) = fla3(h*(jj-1));
  U(n,jj) = fla4(h*(jj-1));
end
for i=1:n
  U(i,1) = fla1(h*(i-1));
  U(i,m) = fla2(h*(i-1));
end
```

```
U(1,1) = (U(1,2) + U(2,1))/2;
U(1,m) = (U(1,m-1) + U(2,m))/2;
U(n,1) = (U(n-1,1) + U(n,2))/2;
U(n,m) = (U(n-1,m) + U(n,m-1))/2;
w = 4/(2+sqrt(4-(cos(pi/(n-1))+cos(pi/(m-1)))^2));
err = 1;
cnt = 0;
while ((err>tol)&(cnt<=max1))
 err = 0;
 for jj=2:(m-1)
  for i=2:(n-1)
   relx=w*(U(i,jj+1)+U(i,jj-1)+U(i+1,jj)+U(i-1,jj)-4*U(i,jj))/4;
     U(i,jj) = U(i,jj) + relx;
     If (err<=abs(relx))
       err=abs(relx);
     end
  end
 end
 cnt = cnt+1;
end

% ELLIPTIC EQUATIONS.
% Dirichlet solution for the Laplace equation
% with the boundary values:'
% u(x,0) = f1(x), u(x,b) = f2(x)   for 0 ≤ x ≤ a,
% u(0,y) = f3(y), u(a,y) = f4(y)   for 0 ≤ y ≤ b,
% A numerical approximation is computed over the rectangle
%         0 ≤ x ≤ a ,   0 ≤ y ≤ b.
% function z = f1(x)
% z = 20;
% function z = f2(x)
% z = 180;
% function z = f3(y)
% z = 80;
% function z = f4(y)
% z = 0;
% Place the endpoint of [0,a] in  a.
% Place the endpoint of [0,b] in  b.
% Place the step size in  h.
% Place the tolerance in  tol.
% Place the maximum number of iterations in  max1
a  =  4.0;
b  =  4.0;
h  =  0.1;
tol = 0.001;
max1  = 25;
```

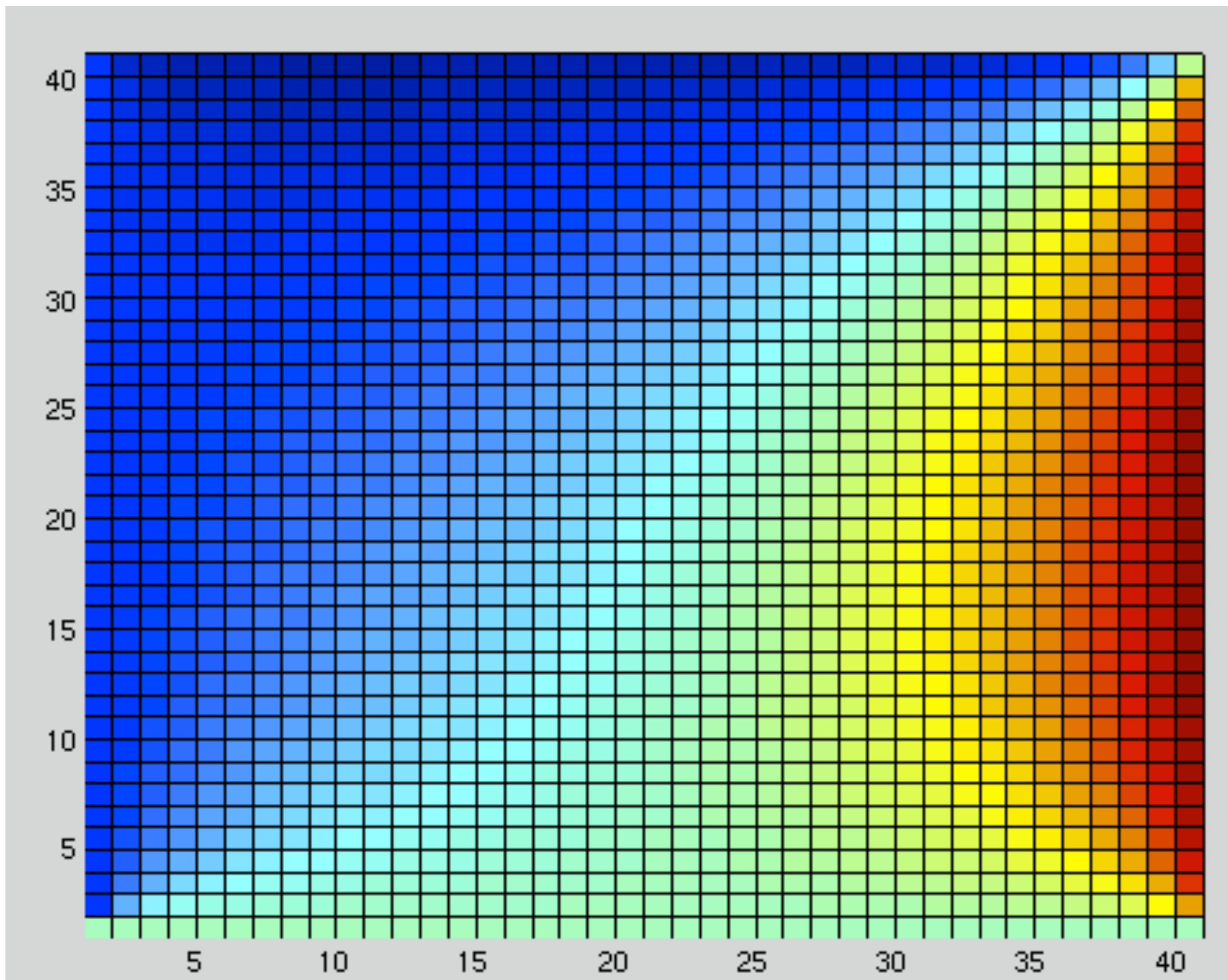
```
% Proceeding with the iteration.
U = dirich(a,b,h,tol,max1);
pcolor(U);

>> tic,laplsolve,toc
Elapsed time is 0.157554 seconds.
```



Alternative Program (no functions + matrix shifts)

```
% set potential array size
p=zeros(41,41);
% set boundary values
p(1,1:41)=20*ones(1,41);
p(41,1:41)=180*ones(1,41);
p(1:41,1)=80*ones(41,1);
p(1:41,41)=zeros(41,1);
% iterate
for i=1:1000
```

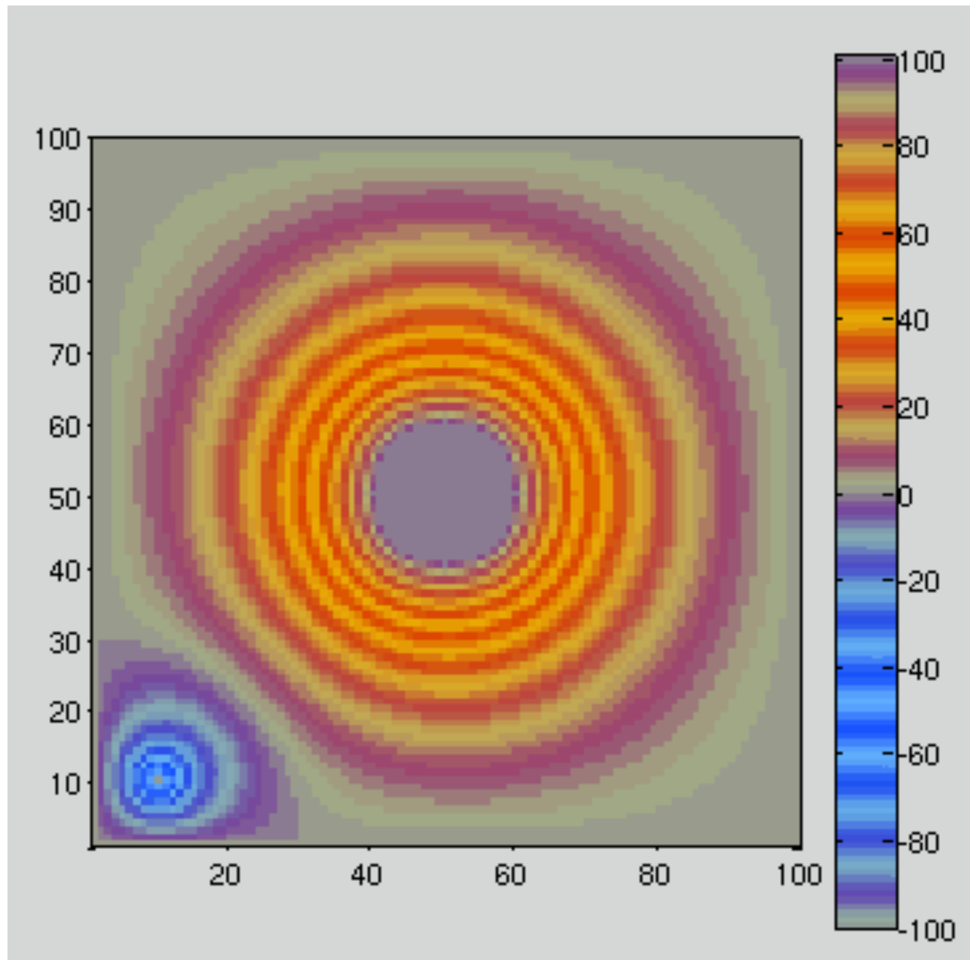
```
 p=0.25*(p(:,[41,[1:40]])+p(:,[[2:41],1])+p([41,[1:40]],:)+ ....
       p([[2:41],1],:));
 % keep boundaries constant
 p(1,1:41)=20*ones(1,41);
 p(41,1:41)=180*ones(1,41);
 p(1:41,1)=80*ones(41,1);
 p(1:41,41)=zeros(41,1);
end
pcolor(p')

>> tic, laplrelax,toc
Elapsed time is 0.261400 seconds.
```




```
% Matlab code for simple relaxation method
```

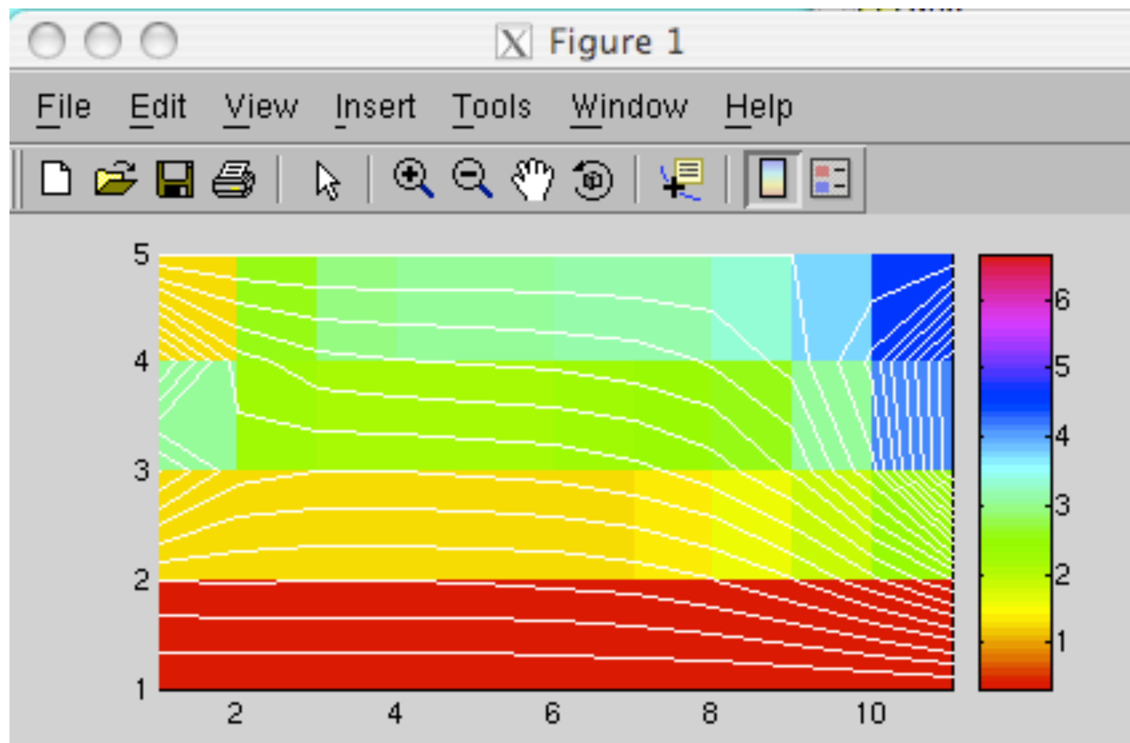
```
% classrelax8.m
% initial potential values
% or boundary values
n=100
p=zeros(n);
% set boundary values
p(1,1:n)=zeros(1,n);
p(n,1:n)=zeros(1,n);
p(1:n,1)=zeros(n,1);
p(1:n,n)=zeros(n,1);
for i=(n/2-15):(n/2+15)
 for j=(n/2-15):(n/2+15)
    r2=(i-n/2)^2+(j-n/2)^2;
    if ((r2 <= n/10) & (r2 >=(n/10-1)))
      p(i,j)=100;
    end
  end
end
p(n/10,n/10)=-100;
% iteration
for i=1:1000
  i
  p=0.25*(p(:,[n,[1:(n-1)]])+p(:,[[2:n],1])+ ....
                               p([n,[1:(n-1)]],:)+p([[2:n],
1],:));
% reset changed boundaries
% set boundary values
p(1,1:n)=zeros(1,n);
p(n,1:n)=zeros(1,n);
p(1:n,1)=zeros(n,1);
p(1:n,n)=zeros(n,1);
for i=(n/2-15):(n/2+15)
 for j=(n/2-15):(n/2+15)
    r2=(i-n/2)^2+(j-n/2)^2;
    if ((r2 <= 100) & (r2 >=81))
      p(i,j)=100;
    end
  end
end
p(n/10,n/10)=-100;
end
figure('Position',[100 10 400 400])
x=[1:n];
y=[1:n];
pcolor(x,y,p)
hold on
```

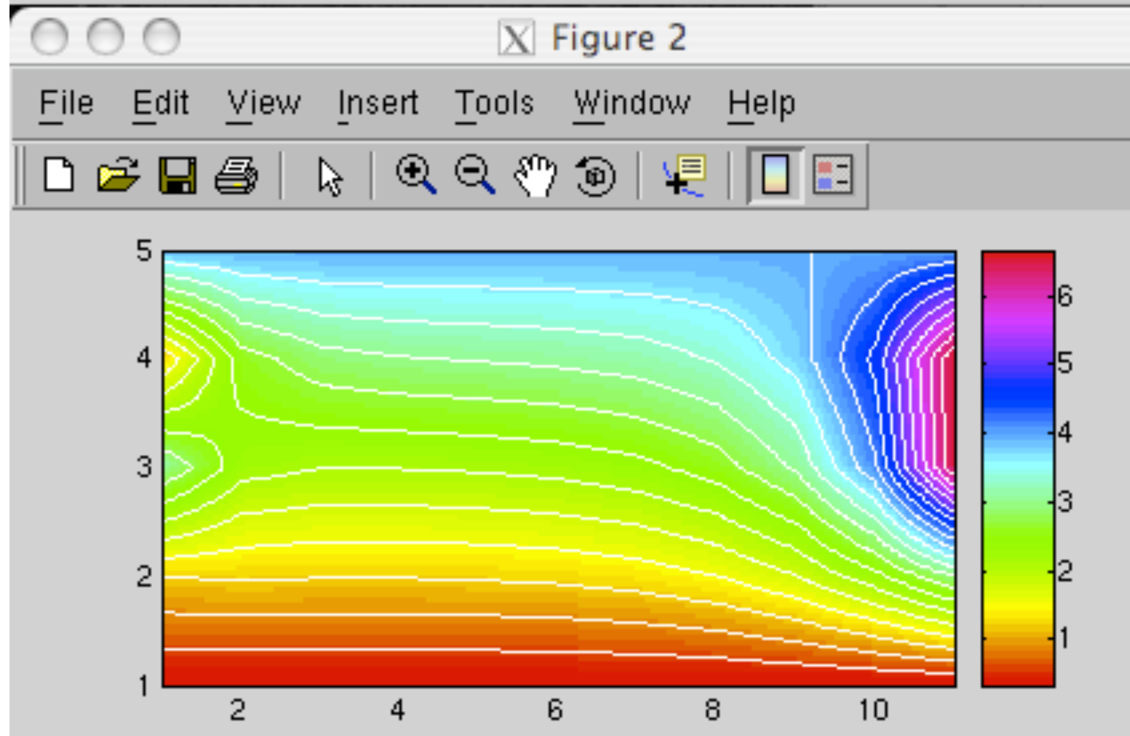
```
colormap(waves)
shading flat
axis('square')
colorbar
hold off
```




```
% Matlab code for simple relaxation method
% classrelax511.m
% initial potential values
% or boundary values
p=zeros(5,11);
% set boundary values
p(1,1:11)=zeros(1,11);
p(5,1:11)=4*ones(1,11);
p(2,1)=1;
p(3,1)=3;
p(4,1)=1 ;
p(2,11)=3;
```

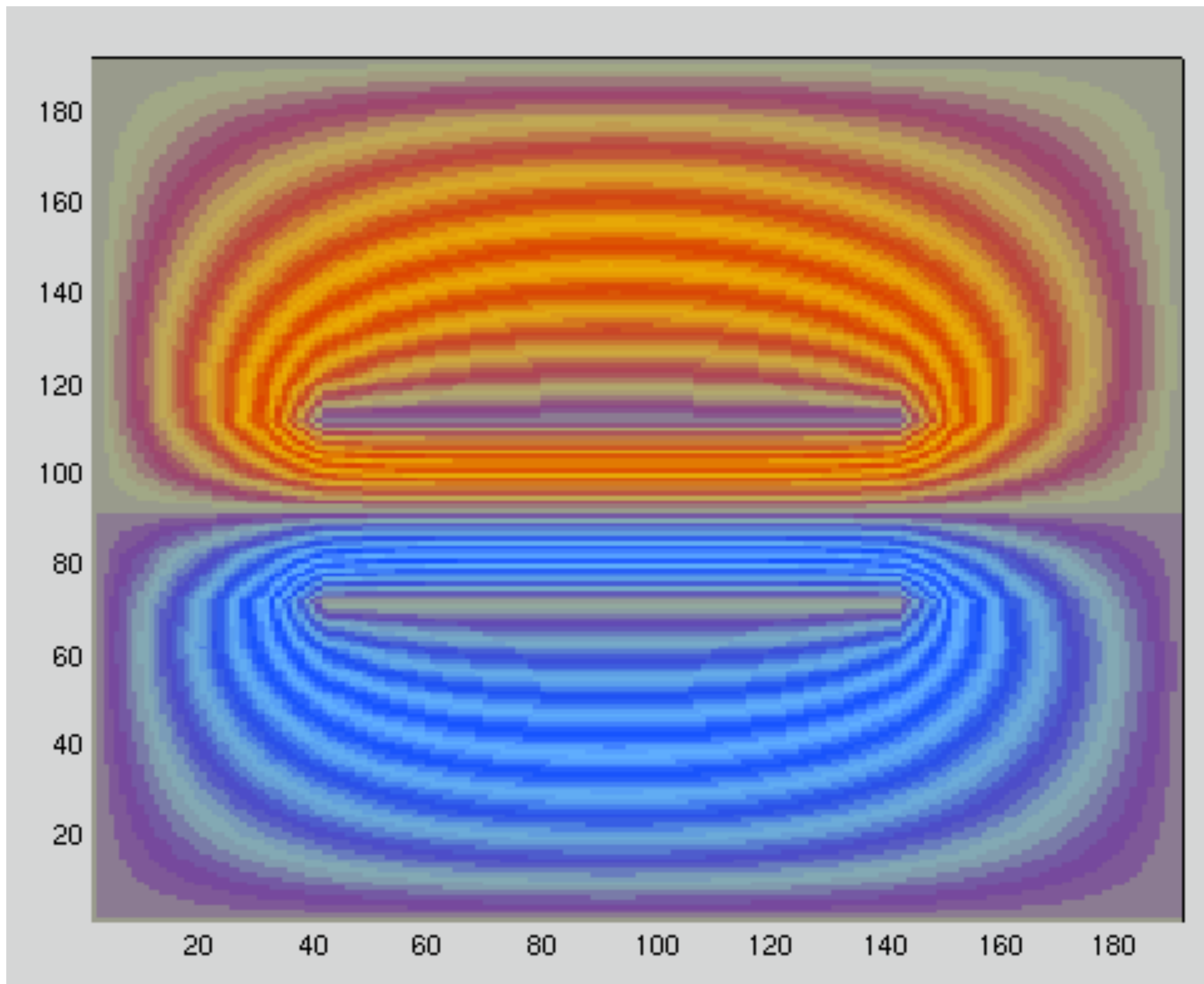
```
p(3,11)=7;
p(4,11)=7;
% iteration
for i=1:100
  i
  p=0.25*(p(:,[11,[1:10]])+p(:,[[2:11],1])+ ....
                              p([5,[1:4]],:)+p([[2:5],1],:));
% reset changed boundaries
% set boundary values
p(1,1:11)=zeros(1,11);
p(5,1:11)=4*ones(1,11);
p(2,1)=1;
p(3,1)=3;
p(4,1)=1 ;
p(2,11)=3;
p(3,11)=7;
p(4,11)=7;
end
figure('Position',[100 10 440 200])
x=[1:11];
y=[1:5];
pcolor(x,y,p)
hold on
colormap(hsv)
shading flat
contour(x,y,p,20,'w')
colorbar
hold off
pause
% interpolate 5x11 --> 100x220
tx=1:.05:11;
ty=1:.05:5;
[X,Y] = meshgrid(tx,ty);
pot=interp2(x,y,p,X,Y);
% colormap image
figure('Position',[100 250 440 200])
pcolor(tx,ty,pot)
hold on
colormap(hsv)
shading flat
contour(tx,ty,pot,20,'w')
colorbar
hold off
```






```
% Matlab code for simple relaxation method
% relax3.m
% initial potential values
p=zeros(20);
% boundary values
```

```
p(1,1:20)=zeros(1,20);
p(20,1:20)=zeros(1,20);
p(1:20,1)=zeros(20,1);
p(1:20,20)=zeros(20,1);
p(12,5:15)=15*ones(1,11);
p(8,5:15)=-15*ones(1,11);
% iteration
for i=1:1000
  i
  p=0.25*(p(:,[20,[1:19]])+p(:,[[2:20],1])+ ....
              p([20,[1:19]],:)+p([[2:20],1],:));
  % reset changed boundaries
p(1,1:20)=zeros(1,20);
p(20,1:20)=zeros(1,20);
p(1:20,1)=zeros(20,1);
p(1:20,20)=zeros(20,1);
p(12,5:15)=15*ones(1,11);
p(8,5:15)=-15*ones(1,11);
end
% interpolate 20x20 --> 200x200
x=[1:20];
y=[1:20];
t=1:.1:20;
[X,Y] = meshgrid(t,t);
pot=interp2(x,y,p,X,Y);
% colormap image
pcolor(pot)
colormap(waves)
shading flat
```




```
% Matlab code for simple relaxation method
% relax31cmct.m
% initial potential values
p=zeros(20);
% boundary values
p(1,1:20)=zeros(1,20);
p(20,1:20)=zeros(1,20);
p(1:20,1)=zeros(20,1);
p(1:20,20)=zeros(20,1);
p(8,10)=3; p(12,10)=-1;
% iteration
for i=1:1000
  i
  p=0.25*(p(:,[20,[1:19]])+p(:,[[2:20],1])+ ....
                    p([20,[1:19]],:)+p([[2:20],1],:));
% reset changed boundaries
p(1,1:20)=zeros(1,20);
```

```
p(20,1:20)=zeros(1,20);
p(1:20,1)=zeros(20,1);
p(1:20,20)=zeros(20,1);
p(8,10)=3 ;
p(12,10)=-1 ;
end
% interpolate 20x20 --> 200x200
x=[1:20];
y=[1:20];
t=1:.1:20;
[X,Y] = meshgrid(t,t);
pot=interp2(x,y,p,X,Y);
% colormap image
pcolor(pot)
colormap(waves)
shading flat
```