

LabK_JustRelax

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:

$$\begin{aligned} u_{xx} + u_{yy} &= 0 && \text{Laplace} && u_{xx} + u_{yy} = g(x,y) && \text{Poisson} \\ u_{xx} + u_{yy} + f(x,y)u &= g(x,y) && \text{Helmholtz} \end{aligned}$$

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

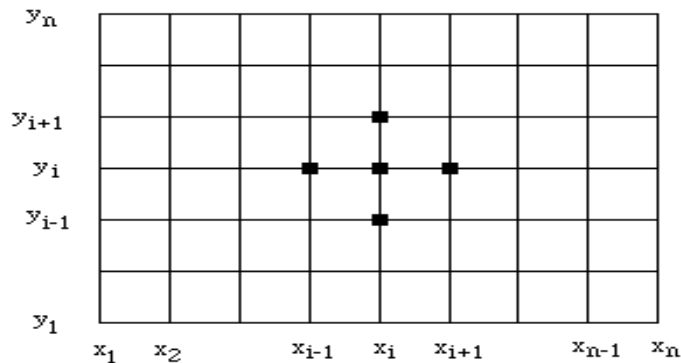
The formula for approximating $f''(x)$ is obtained from

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

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 $R = \{(x,y) : 0 \leq x \leq a, 0 \leq y \leq b \text{ where } b/a = m/n\}$ is subdivided into $(n-1) \times (m-1)$ squares with side h where $a = nh, b = mh$ as shown below:



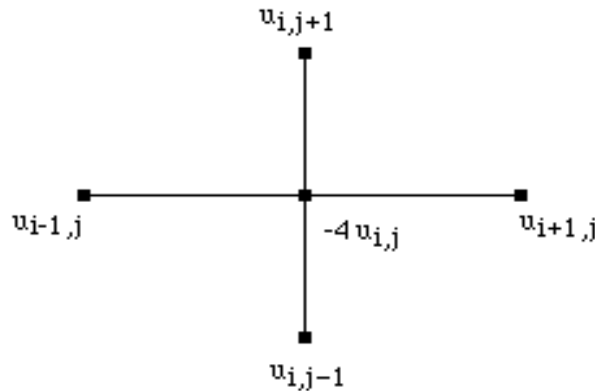
To solve Laplace's equation we impose the condition

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

which is valid at all interior grid points $(x,y)=(x_i,y_j)$ for $i=2,3,4,\dots,n-1$ and $j=2,3,4,\dots,m-1$. 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:

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

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



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

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

The Iterative Solution Method

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

$$\begin{aligned} u_{1j} & \text{ for } 2 \leq j \leq m-1 \text{ (on the left)} \\ u_{i1} & \text{ for } 2 \leq i \leq n-1 \text{ (on the bottom)} \\ u_{nj} & \text{ for } 2 \leq j \leq m-1 \text{ (on the right)} \\ u_{in} & \text{ for } 2 \leq i \leq n-1 \text{ (on the top)} \end{aligned}$$

We can then write our equation in form suitable for

iteration as

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

where

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

for $2 \leq i \leq n-1$ and $2 \leq j \leq m-1$.

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 until the residual term r_{ij} is **reduced to zero** (i.e., $|r_{ij}| < \epsilon$ 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 $1 \leq \omega \leq 2$. The optimal choice for ω is given by

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

* **Do exercises #57-#58** *

Poisson's and Helmholtz's Equations

The iterative formulas for these two equations are:

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

where for Poisson:

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

and for Helmholtz:

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

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

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

 * Do exercises #59-#60 *

Sample Programs --- Laplace Equation

```
function fla1,x
return,20
end
```

```
function fla2,x
return,180
end
```

```
function fla3,x
return,80
end
```

```
function fla4,x
return,0
end
```

```
function 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 = floor(a/h)+1
m = floor(b/h)+1
ave = (a*(fla1(0)+fla2(0)) $
      + b*(fla3(0)+fla4(0)))/(2.0*a+2.0*b)
U = ave*(fltarr(n,m)+1.0)
for jj=0,m-1 do begin
  U(0,jj) = fla3(h*(jj-1))
  U(n-1,jj) = fla4(h*(jj-1))
endfor
for i=0,n-1 do begin
  U(i,0) = fla1(h*(i-1))
  U(i,m-1) = fla2(h*(i-1))
end
U(0,0) = (U(0,1) + U(1,0))/2.0
U(0,m-1) = (U(0,m-2) + U(1,m-1))/2.0
U(n-1,0) = (U(n-2,0) + U(n-1,1))/2.0
U(n-1,m-1) = (U(n-2,m-1) + U(n-1,m-2))/2.0
w = 4.0/(2.0+sqrt(4.0-(cos(!pi/(n-1))+ cos(!pi/(m-1)))^2))
err = 1.0 & cnt = 0
while ((err GT tol) AND (cnt LE max1)) do begin
  err = 0.0
  for jj=1,(m-2) do begin
    for i=1,(n-2) do begin
      relx = w*(U(i,jj+1)+U(i,jj-1)+U(i+1,jj)+ U(i-1,jj))- $
            4.0*U(i,jj))/4.0
      U(i,jj) = U(i,jj) + relx
      if (err LE abs(relx)) then begin
        err=abs(relx)
      endif
    endfor
  endfor
  cnt = cnt+1
endwhile
return,U
end

```

```

pro lpsolve
;   ELLIPTIC EQUATIONS.
;   Dirichlet solution for the Laplace equation
;   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)
surface,reverse(U,1)
end

```

Alternative Program (no functions + matrix shifts)

```

pro relax_ex1, steps
;set potential array size
pin=fltarr(41,41)
;set boundary values
pin(0:40,0)=20+fltarr(41)
pin(0:40,40)=180.0+fltarr(41)
pin(0,0:40)=transpose(80.0+fltarr(41))
pin(40,0:40)=transpose(0.0+fltarr(41))
b=pin NE 0.0
b(40,0:40)=transpose(1+fltarr(41))
p=pin
window,xsize=410,ysize=410
loadct,37
for i=1,steps do begin
  print,i
  p=0.25*(shift(p,1,0)+shift(p,-1,0)+ $
          shift(p,0,1)+shift(p,0,-1))
  p=p*(1-b)+pin
  surface,reverse(p,1)
endfor
end

```

```

pro relax_ex, steps
;set potential array size
pin=fltarr(41,41)
;set boundary values
pin(0:40,0)=20+fltarr(41)
pin(0:40,40)=180.0+fltarr(41)
pin(0,0:40)=transpose(80.0+fltarr(41))
pin(40,0:40)=transpose(0.0+fltarr(41))
b=pin NE 0.0
b(40,0:40)=transpose(1+fltarr(41))
p=pin
window,xsize=410,ysize=410
loadct,37
for i=1,steps do begin
  print,i

```

```

    p=0.25*(shift(p,1,0)+shift(p,-1,0)+ $
                shift(p,0,1)+shift(p,0,-1))
    p=p*(1-b)+pin
    pot=rebin(p,410,410)
    tvscl,pot
    pot=0
endfor
end

```

Random Walk Solution of Laplace's Equation

The relaxation method was based on the fact that the solution of Laplace's equation in 2-dimensions at the point (x,y) is given by

$$V(x,y) = \frac{1}{4} \sum_{i=1}^4 V(i)$$

where $V(i)$ is the value of the potential at the i^{th} nearest neighbor. We saw that this solution is equivalent to the result that the potential at any point equals the average of the potential on a sphere in 3-dimensions (circle in 2-dimensions) about that point.

This relation can be given a probabilistic interpretation in terms of random walks, i.e., a "walk" in two dimensions (in this case) where the probability of taking a step in any direction is equal.

A sample random walk program is shown below:

```

pro ranwalk
size=401
ar=intarr(size+10,size+10)
x=(size-1)/2
xst=x
y=(size-1)/2
yst=y
L=1
ar(x,y)=100
window,0,title='Random
Walk',xs=400,ys=400,xpos=100,ypos=100
loadct,37
for j=1L,100000L do begin

```

```

ch=randomu(seed)
x=x+L*(ch LT .25 )
x=x-L*((ch GE.25) AND (ch LT .5))
y=y+L*((ch GE .5) AND (ch LT .75))
y=y-L*((ch GE .75) AND (ch LE 1.00))
j=j+100001L* (sqrt((float(x-xst))^2 +(float(y-yst))^2) GT
200.0)
if (j LE 100000L) then begin
  ar(x,y)=100
  if ((j mod 1000L) EQ 0) then begin
    tvscl,ar
  endif
endif
endfor
end

```

Suppose that many random walkers (drunks) are at the point (x,y) and each walker "jumps" to one of its four neighbors (on a square grid) with equal probability $p=1/4$. The above relation says that the average potential found by the walker after jumping one step is the potential at (x,y) .

The random walk algorithm for computing the solution to Laplace's equation can be stated as:

[1] Begin at point (x,y) where the value of the potential is desired, and take a step in a random direction.

[2] Continue taking steps until the walker reaches a boundary (where we have a specified boundary value) point i .

[3] Repeat steps [1] and [2] n times and sum the potential values $V_b(i)$ found at the boundary each time.

[4] The value of the potential at the point (x,y) is estimated by

$$V(x,y) = \frac{1}{n} \sum_{i=1}^n V(i)$$

where n is the total number of random walkers.

The disadvantage of the random walk method is that it

requires many walkers to obtain a good estimate of the potential at each point. However, if the potential is needed at only a small number of points, then the random walk method might be more efficient than the relaxation method which requires the potential to be calculated at all points within the boundary.

 * **Do exercise #61** *

Another case where the random walk method is appropriate is when the geometry of the boundary is fixed, but the potential in the interior for a variety of different boundary potentials is needed. In this case the quantity of interest is

$$G(x, y, x_b, y_b) = \text{number of time walker from point } (x, y) \text{ reaches boundary point } (x_b, y_b)$$

The random walk algorithm is equivalent to the relation

$$V(x, y) = \frac{1}{n} \sum_b G(x, y, x_b, y_b) V(x_b, y_b)$$

where the sum is over all points on the boundary. We can use the same function G for different distributions of the potential on a given boundary. G is an example of a **Green's Function**.