HW6 Continuum Dynamics

Fei Ding

November 18, 2009

Problem1
Figure 1 showed the contour plot of potential.

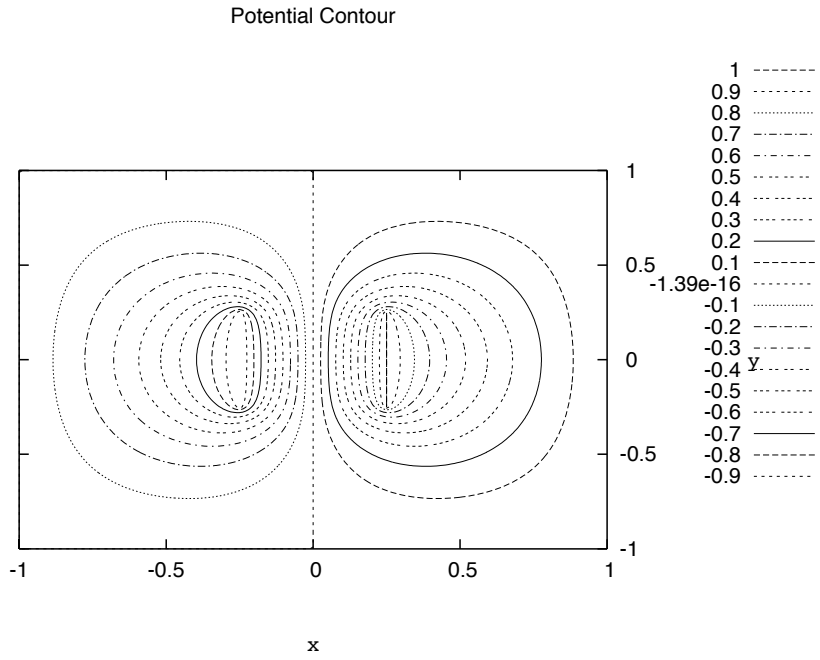

Figure 1: Potential contour for a 2D view

The equation the potential satisfy is the Laplace equation, $\nabla^2\phi = 0$.

Termination criterion. At first, I used the maximum element of the matrix $\mathbf{V} - \mathbf{V_{old}}$ as the error estimation , but it didn't work well. Then I used Frobenius norm of the matrix $\mathbf{V} - \mathbf{V_{old}}$ as the error estimation, the definition is as following,

$$\|\mathbf{A}\| = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 \right)^{1/2}$$

Figure 2-4 showed the error estimate in V vs. iteration for N =20,40,80 using Jacobi method, Gauss-Seidel scheme and checkerboard SOR, respectively. It is clearly that Gauss-Seidel converges faster than Jacobi roughly by a factor of 2, while the SOR converges much faster than the previous two methods. I used Chebyshev acceleration to determine the value of ω in this part. That is I changed the value of ω for each updating.

From figure 5, it is seen that the Jacobi updating doesn't work for SOR. From figure 6, the looping in coordinates updating converges faster than checkerboard updating for SOR. For this part I used N=40 and $\omega = \frac{2}{1+\sqrt{1-\rho^2_{Jacobi}}}$, where $\rho_{Jacobi} = 1 - \frac{\pi^2}{2J^2}$, $J = 2N + 1$ for all the three updating schemes in order to compare them. However, the interesting thing was that I found the checkerboard updating converged a little faster here using constant ω than using Chebyshev acceleration in the previous part. The reason I can think about is that because here the boundary conditions are not only on the four sides, so the ρ_{Jacobi} is not so accurate. Or maybe the small difference (529 iterations $vs.$ 515 iterations) in the converge didn't really tell useful information except for computation fluctuations.

From the table below we can see iteration numbers required by the three different methods for N=20,40,80.

	N=20	N=40	N=80
Jacobi	1508	5761	21962
Gauss-Seidel	803	3020	11486
Checkerboard SOR	263	529	1071

The iteration number of Jacobi and Gauss-Seidel method increased roughly by 4 with the grid numbers $L(L = 2N + 1$here) increased by 2. While the iteration number of Checkerboard SOR increased roughly by 2 with the with the grid numbers L increased by 2. Thus, the iteration number is $O(L^2)$ for Jacobi and Gauss-Seidelmethod, O(L) for SOR method. Considering for each iteration, there should be L^2 loops. So the SOR is an $O(L^3)$ algorithm, while Jacobi and Gauss-Seidel is $O(L^3)$

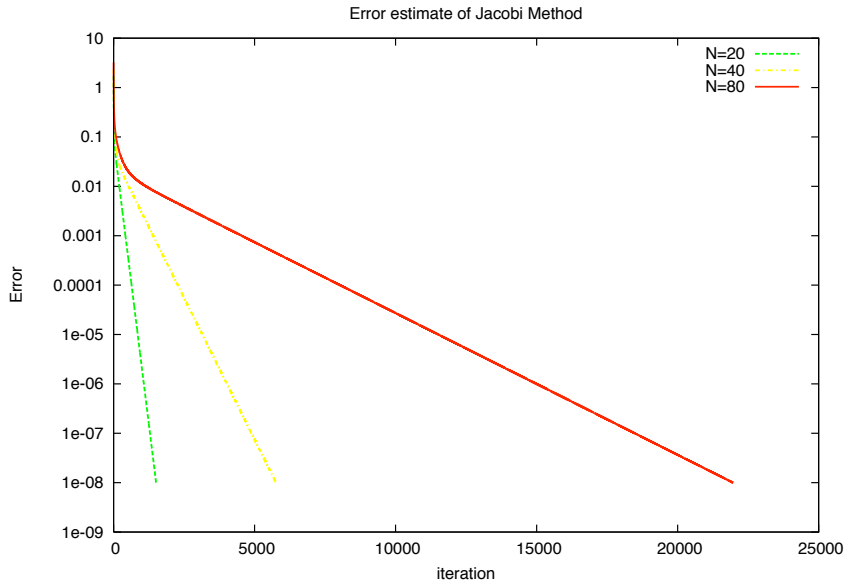

Figure 2: Error estimate of Jacobi Method

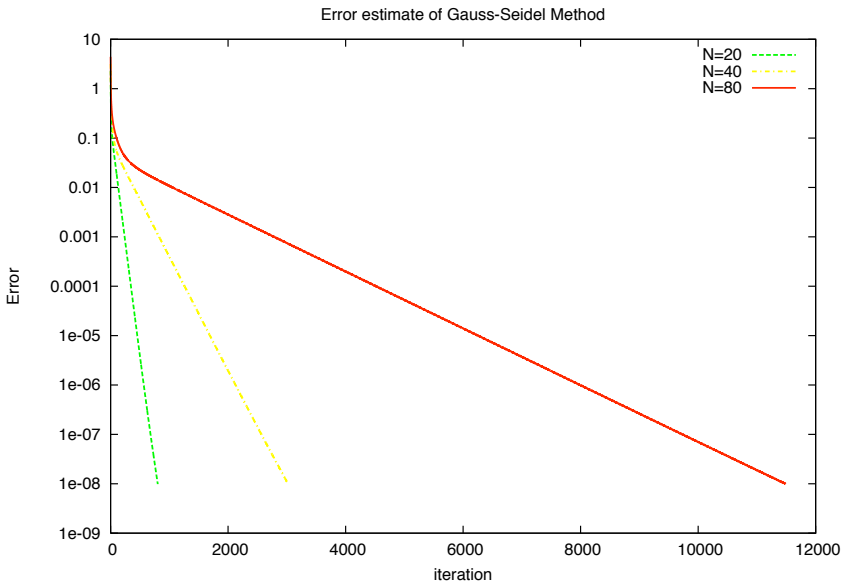

Figure 3: Error estimate of Gauss-Seidel Method

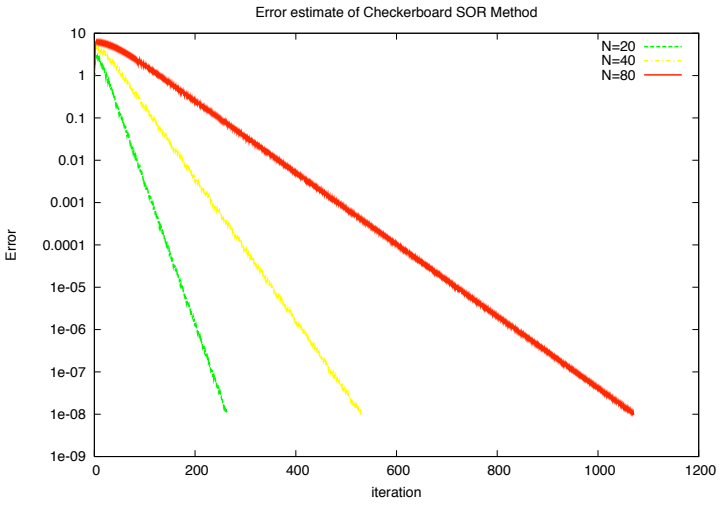

Figure 4: Error estimate of Checkerboard SOR Method

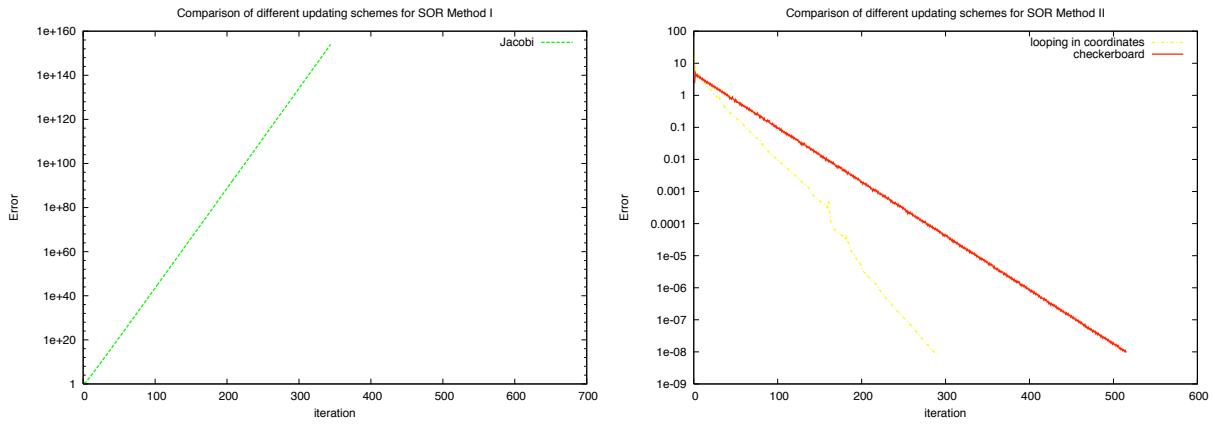

Figure 5: Comparison of different updating for SOR Method

Problem2

The heat equation $\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}$, apply FTCS to solve the problem and got the Time evolution of Temperature distribution showed in figure 6. The step function at x=0 spread out as time goes by. $\delta x = 0.001m$, $\delta t = 0.5s$, grid number $2N + 1 = 2001$, so the scale of sea and sky is roughly from -1 to 1. I set Dirichlet boundary condition for space, that is fixed value at end points. Because I ran the program up to $t = 320s$, the step function hasn't reached the boundary yet. On figure 6 I zoomed into the range [-0.05,0.05] to give a better view of the result.

According to the initial condition

$$T(x, 0) = T_L \theta(x < 0) + T_R \theta(x > 0)$$

the analytic solution is

$$T(z, t) = \frac{T_L + T_R}{2} + \frac{T_R - T_L}{2} erf(\frac{x}{\sqrt{4Dt}})$$

Figure 7 showed that the numerical solution agrees well with the analytic solution at t=160s.

Stability. Figure 8 and figure 9 showed stability against changes in δx and δt, respectively. According to stability condition, $\alpha = 8 * 10^{-7} \leq \frac{(\Delta x)^2}{2\Delta t}$, for fixed $\delta t = 0.5$, the critical $\delta x = 0.00089442719$, figure 8 showed that the small change in δx (0.4%) produced large unstable. While for δx above the critical value the result is stable and nice. For fixed $\delta x = 0.001$, the critical $\delta t = 0.625$, so the value I chose $\delta t = 0.5$ is quite safe and did give nice plot. The small change in δt(0.8%) also produced large unstable. And the unstable for change in δx is larger than in δt. The interesting thing is that while $\delta t = 0.625$, the plot turned out to be same value for every two neighboring grid points, which makes sense because unstable means the value jumps up and down quite crazy, while for the critical point the values are about to jump but stay in the same position for the neighboring points.

Figure 10 showed the ocean's thermal expansion. The expansion decelerated as time increased.

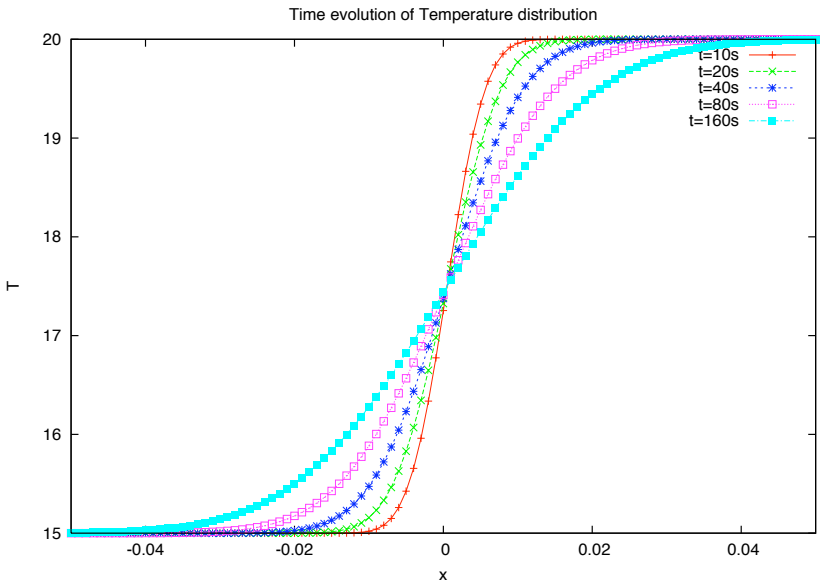

Figure 6: Time evolution of Temperature distribution

Codes

Problem1

```
 program Jacobi
IMPLICIT NONE
integer, parameter::N=100
integer:: i,j,l,neg,pos,NUM
double precision:: delta,norm,sum2,x,y
double precision, dimension(-N:N, -N:N):: v
double precision, dimension(-N:N):: prev, curr !in order to perform Jacobi Method, record the previous i
delta=1d0/N  !grid spacing
neg=nint(-0.25d0/delta) !the grid position of left plate
pos=nint(0.25d0/delta)  !the grid position of right plate
!initial distribution of v
do j=-N, N
   do l=-N, N
      if (j==neg .AND. l>=neg .AND. l<=pos) then
         v(j,l)=-1d0
      else if (j==pos .AND. l>=neg .AND. l<=pos) then
         v(j,l)=1d0
      else
         v(j,l)=0d0
      end if
   end do
end do
do i=-N, N
   prev(i)=0d0 !the initial value of prev array=v(j,-N)=0
   curr(i)=0d0 !main purpose is to set the boundary value for curr array
end do
norm=1d0
```

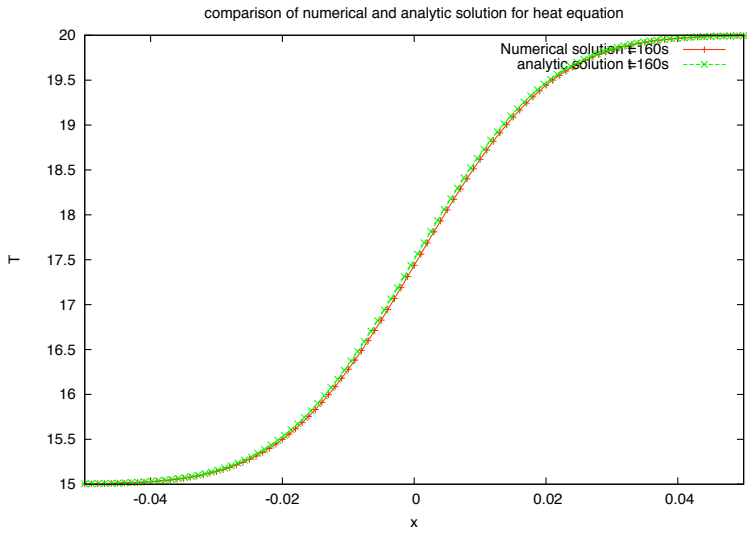

Figure 7: Compare with analytical solution

```
NUM=0
do while(norm>=1d-8)
   NUM=NUM+1 !iteration number counting
   sum2=0d0
   do l=-(N-1), (N-1) !sweep along rows
      do j=-(N-1), (N-1)
         curr(j)=v(j,l) !use curr array to record old v(j,l)
         if (j==neg .AND. l>=neg .AND. l<=pos) then
            continue
         else if (j==pos .AND. l>=neg .AND. l<=pos) then
            continue
         else
           v(j,l)=0.25d0*(v(j+1,l)+curr(j-1)+v(j,l+1)+prev(j))
         end if
         sum2=sum2+(v(j,l)-curr(j))**2!square sum of difference matrix
      end do !j loop
      do i=-N,N
         prev(i)=curr(i) !use prev array to record the old row
      end do
    end do !l loop
    norm=dsqrt(sum2)!Frobenius norm of difference matrix
    write(2,*) NUM,norm
end do !do while loop
print *,NUM
!write data into file
do j=-N,N
   do l=-N,N
      x=j*delta
      y=l*delta
```

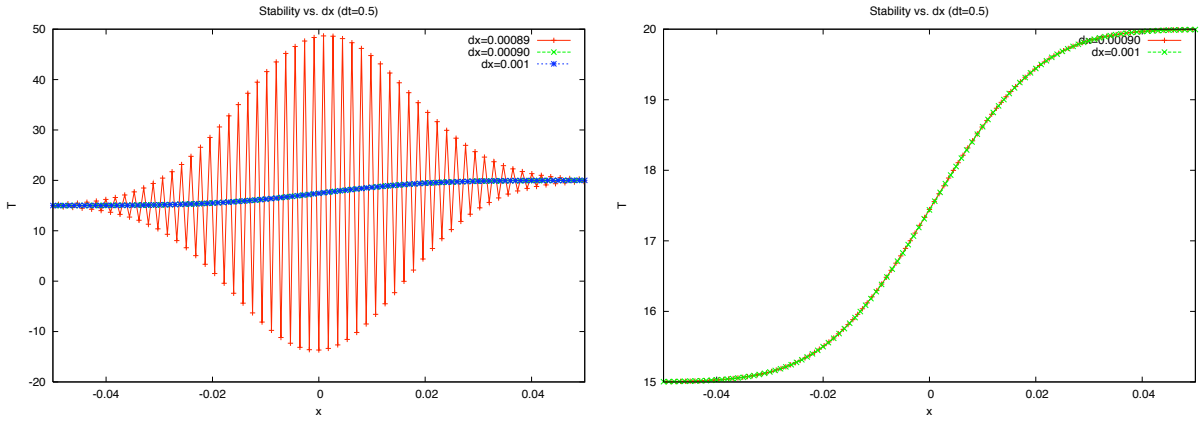

Figure 8: Stability vs. dx

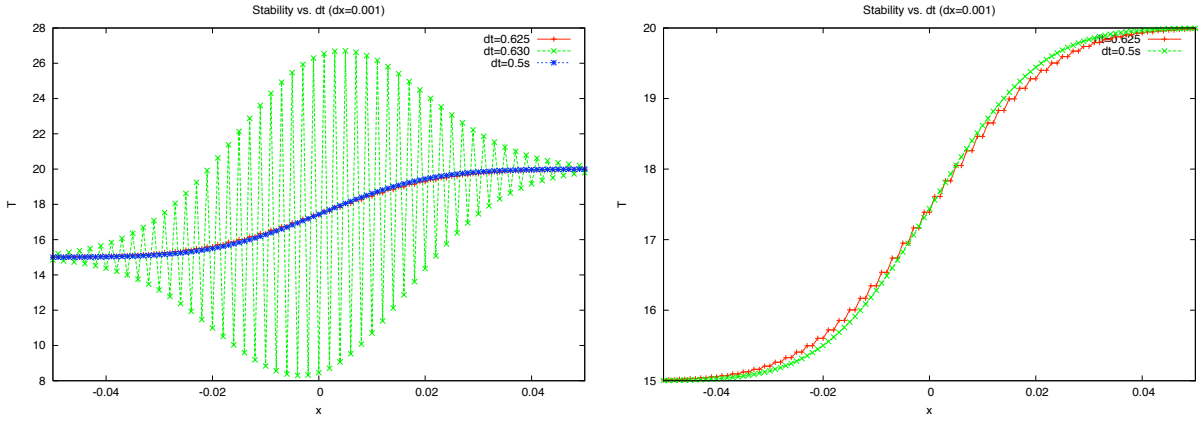

Figure 9: Stability vs. dt

```
      write(1,'(3F16.10)') x,y,v(j,l)
   end do
      write(1,*) !add space to plot contour
end do
end program
```

```
program GaussSeidel
IMPLICIT NONE
integer, parameter::N=80
integer:: i,j,k,l,neg,pos,NUM
double precision:: delta,old,sum2,norm
double precision, dimension(-N:N, -N:N):: v
delta=1d0/N  !grid spacing
neg=nint(-0.25d0/delta) !the grid position of left plate
pos=nint(0.25d0/delta)  !the grid position of right plate
!initial distribution of v
do j=-N, N
   do l=-N, N
      if (j==neg .AND. l>=neg .AND. l<=pos) then
         v(j,l)=-1d0
```

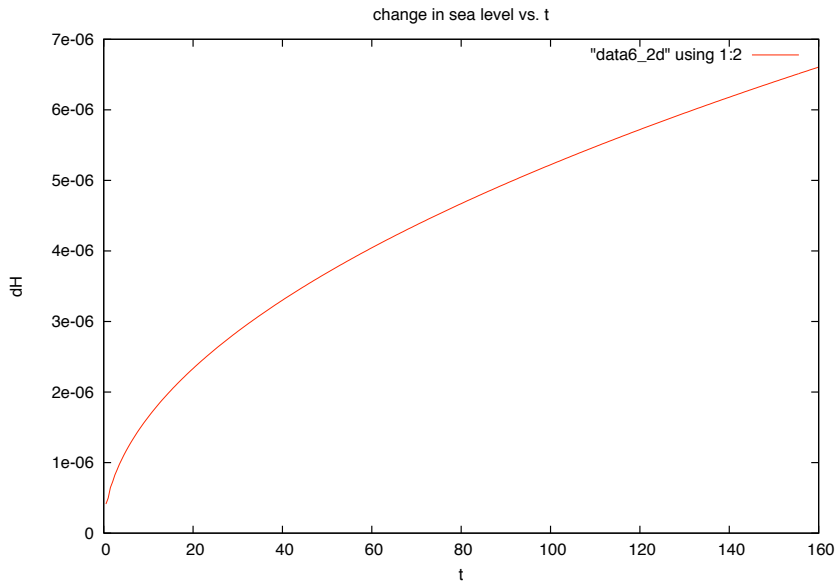

Figure 10: Chage in sea level vs. T

```
      else if (j==pos .AND. l>=neg .AND. l<=pos) then
         v(j,l)=1d0
      else
         v(j,l)=0d0
      end if
   end do
end do
norm=1d0
NUM=0
do while(norm>=1d-8)
   NUM=NUM+1 !iteration number counting
   sum2=0d0
   do l=-(N-1), (N-1) !sweep along rows
      do j=-(N-1), (N-1)
         old=v(j,l) !record the old value of v(j,l)
         if (j==neg .AND. l>=neg .AND. l<=pos) then
            continue
         else if (j==pos .AND. l>=neg .AND. l<=pos) then
            continue
         else
           v(j,l)=0.25d0*(v(j+1,l)+v(j-1,l)+v(j,l+1)+v(j,l-1))
         end if
         sum2=sum2+(v(j,l)-old)**2!square sum of difference matrix
      end do !j loop
    end do !l loop
    norm=dsqrt(sum2)!Frobenius norm of difference matrix
    write(1,*) NUM,norm
end do !do while loop
print *,NUM
end program
```

```
program SOR
IMPLICIT NONE
integer, parameter::N=20
integer:: i,j,k,l,neg,pos,NUM
double precision:: delta,old,vstar,sum2,norm
double precision:: PI,rjac,w
double precision, dimension(-N:N, -N:N):: v
PI = 4.d0*datan(1.d0)
delta=1d0/N  !grid spacing
neg=nint(-0.25d0/delta) !the grid position of left plate
pos=nint(0.25d0/delta)  !the grid position of right plate
!initial distribution of v
do j=-N, N
   do l=-N, N
      if (j==neg .AND. l>=neg .AND. l<=pos) then
         v(j,l)=-1d0
      else if (j==pos .AND. l>=neg .AND. l<=pos) then
         v(j,l)=1d0
      else
         v(j,l)=0d0
      end if
   end do
end do
norm=1d0
NUM=0
rjac=1-PI**2/(2d0*dble(2*N+1)**2) !Jacobi spectral radius
w=1d0
do while(norm>=1d-8)
   NUM=NUM+1 !iteration number counting
   sum2=0d0
   do l=-(N-1), (N-1) !sweep along rows
      do j=-(N-1), (N-1)
         old=v(j,l) !record the old value of v(j,l)
         if (j==neg .AND. l>=neg .AND. l<=pos) then
            continue
         else if (j==pos .AND. l>=neg .AND. l<=pos) then
            continue
         else
            if (mod(j+l,2) ==mod(NUM,2)) then !checkerboard updating odd-even
               vstar=0.25d0*(v(j+1,l)+v(j-1,l)+v(j,l+1)+v(j,l-1))
               v(j,l)=w*vstar+(1d0-w)*old
            end if
         end if
         sum2=sum2+(v(j,l)-old)**2!square sum of difference matrix
      end do !j loop
    end do !l loop
    norm=dsqrt(sum2)!Frobenius norm of difference matrix
    write(1,*) NUM,norm
    !Chebyshev Acceleration
    if (NUM==1) then
       w=1d0/(1d0-rjac**2/2d0)
    else
       w=1d0/(1d0-rjac**2*w/4d0)
```

```
    end if
end do !do while loop
print *,NUM
end program
```

```
program SORJacobi
IMPLICIT NONE
integer, parameter::N=40
integer:: i,j,l,neg,pos,NUM
double precision:: delta,norm,vstar,sum2
double precision:: PI,rjac,w
double precision, dimension(-N:N, -N:N):: v
double precision, dimension(-N:N):: prev, curr !in order to perform Jacobi Method, record the previous i
PI = 4.d0*datan(1.d0)
delta=1d0/N  !grid spacing
neg=nint(-0.25d0/delta) !the grid position of left plate
pos=nint(0.25d0/delta)  !the grid position of right plate
!initial distribution of v
do j=-N, N
   do l=-N, N
      if (j==neg .AND. l>=neg .AND. l<=pos) then
         v(j,l)=-1d0
      else if (j==pos .AND. l>=neg .AND. l<=pos) then
         v(j,l)=1d0
      else
         v(j,l)=0d0
      end if
   end do
end do
do i=-N, N
   prev(i)=0d0 !the initial value of prev array=v(j,-N)=0
   curr(i)=0d0 !main purpose is to set the boundary value for curr array
end do
norm=1d0
NUM=0
rjac=1d0-PI**2/(2d0*dble(2*N+1)**2)
w=2d0/(1+dsqrt(1-rjac**2))
do while(norm>=1d-8 .AND. NUM<10000) !Add a new looping condition in case the method does't converge
   NUM=NUM+1 !iteration number counting
   sum2=0d0
   do l=-(N-1), (N-1) !sweep along rows
      do j=-(N-1), (N-1)
         curr(j)=v(j,l) !use curr array to record old v(j,l)
         if (j==neg .AND. l>=neg .AND. l<=pos) then
            continue
         else if (j==pos .AND. l>=neg .AND. l<=pos) then
            continue
         else
           vstar=0.25d0*(v(j+1,l)+curr(j-1)+v(j,l+1)+prev(j))
           v(j,l)=w*vstar+(1d0-w)*curr(j)
         end if
         sum2=sum2+(v(j,l)-curr(j))**2!square sum of difference matrix
      end do !j loop
      do i=-N,N
```

```
         prev(i)=curr(i) !use prev array to record the old row
       end do
     end do !l loop
     norm=dsqrt(sum2)!Frobenius norm of difference matrix
     write(1,*) NUM,norm
end do !do while loop
print *,NUM
end program
```

Problem2

```
program ocean
IMPLICIT NONE
integer,parameter:: N=1000,M=320
integer::i,j
double precision::ul=15d0, ur=20d0
double precision:: delx, delt,delv
double precision::alpha=8d-7, beta=207d-6
double precision,dimension(-N:N):: u,uold
delx=1d0/N
delt=0.5d0
!delt=0.630 !test of stability
!delx=0.0009d0!test of stability
!initial T for sea
do i=-N,0
   uold(i)=ul
   u(i)=ul
end do
!initial T for sky
do i=1,N
   uold(i)=ur
   u(i)=ur
end do
delv=0d0
do j=1,M !loop of time
   do i=-(N-1),N-1 !loop of space
      u(i)=uold(i)+delt*alpha*((uold(i+1)-2*uold(i)+uold(i-1))/delx**2)
      !calculate deltaV for each disc for sea
      if (i<=0) then
         delv=delv+delx*beta*(u(i)-uold(i))
      end if
   end do
   write(2,*) j*delt,delv
   !store old value of Temprature distribution
   do i=-N,N
      uold(i)=u(i)
   end do
end do
delv=0d0
write(1,*) "Temprature distribution @ t=",M*delt
do i=-N,N
   write(1,*) i*delx,u(i)
end do
```

```
end program
```