

# 040692 PR Practical in numerical Astronomy

Dr. Eduard Vorobiev

## Assignment

Dawid Stepanovic

December 30, 2021

### One-dimensional diffusion equation:

$$\frac{\partial \rho}{\partial t} = D \frac{\partial^2 \rho}{\partial x^2} \quad (1)$$

Solve numerically the one-dimensional diffusion equation (1) in the box  $-5 \leq x \leq +5$  using the Crank-Nicholson scheme with the initial condition set by the following equation:

$$\rho(x, t_0) = \exp\left(-\frac{x^2}{4Dt_0}\right) \quad (2)$$

For the diffusion coefficient, choose  $D = 2$ , the starting time  $t_0 = 2.0$ , and the number of grid cells in the x-direction  $N = 100$ . For the boundaries choose the Neumann and Dirichlet conditions. For the former, assume that the gradient of the density across the boundaries is equal to zero, while for the latter use the analytic solution of the form:

$$\rho(x, t) = \sqrt{\frac{t_0}{t_n}} \exp\left(-\frac{x^2}{4Dt}\right), \quad (3)$$

where  $t_n$  is the current time. To preserve accuracy, the maximum relative change in density every time step should not exceed 15 %, such that

$$\max_j \left( \frac{|\rho_j^{n+1} - \rho_j^n|}{\min(\rho_j^{n+1}, \rho_j^n)} \right) < 0.15.$$

### The report should contain:

- A brief introduction on the Crank-Nicholson method and the way you implemented it.
- The graphical representation of the solution for both boundary conditions at  $t = 4.0$  and its comparison with the analytic solution.
- The total mass inside the computational domain as a function of time in the interval  $t \in [2.0, 4.0]$  for each boundary condition. Provide an explanation for the observed trends.
- The analysis of the relative error between the numerical and exact solutions as a function of position  $x$  (also in the graphical form). Where are the minimum and maximum relative errors found (provide values)?

### A brief introduction on the Crank-Nicholson method and the way you implemented it.

We divide the spatial domain into  $N - 1$  intervals of length  $\Delta x = \frac{L}{N-1}$ , while using  $N$  equally distant grid points such that  $x_i = (i - 1)N - \frac{L}{2}$  is the  $i$ -th centered grid point around zero, where  $L$  is total length. Equation (1) is defined in the domain  $(x, t) \in [-5, 5] \times T$ , where the temperature  $T$  is limited to  $T \in [2, 4]$ . For the initial time step we chose the Courant stability condition for the diffusion equation, which is given by  $\Delta t \leq \frac{\Delta x^2}{2D}$ . We adapt the notation such that  $\rho(x_i, t_n) \equiv \rho_i^n$ , where  $i \in N$  and  $t_n \in T$  ( $t_n$  depends on the accuracy setting), and write the Crank-Nicholson scheme as presented in the lecture notes as the sum of the forward-time centered-space (FTCS) and backward-time centered-space (BTCS) schemes

$$\frac{\rho_j^{n+1} - \rho_j^n}{\Delta t} = \frac{D}{2} \left( \frac{\rho_{j+1}^{n+1} - 2\rho_j^{n+1} + \rho_{j-1}^{n+1}}{\Delta x^2} + \frac{\rho_{j+1}^n - 2\rho_j^n + \rho_{j-1}^n}{\Delta x^2} \right) + \mathcal{O}((\Delta x)^2). \quad (4)$$

Solving equation (4) for the new temperatures in terms of old temperatures we get

$$-\alpha\rho_{j-1}^{n+1} + (1 + 2\alpha)\rho_j^{n+1} - \alpha\rho_{j+1}^{n+1} = \underbrace{\alpha\rho_{j-1}^n + (1 - 2\alpha)\rho_j^n + \alpha\rho_{j+1}^n}_{=: u_j^n}, \quad (5)$$

where  $\alpha = \frac{\Delta t D}{2(\Delta x)^2}$ . Supposing a Dirichlet boundary conditions approach, we have to solve the following matrix equation

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ -\alpha & 1 + 2\alpha & -\alpha & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -\alpha & 1 + 2\alpha & -\alpha \\ 0 & \dots & 0 & 0 & 1 \end{pmatrix}}_{\in \mathbb{R}^{N \times N}} \begin{pmatrix} \rho_0^{n+1} \\ \rho_1^{n+1} \\ \vdots \\ \vdots \\ \rho_N^{n+1} \end{pmatrix} = \begin{pmatrix} u_0^n \\ u_1^n \\ \vdots \\ \vdots \\ u_N^n \end{pmatrix}, \quad (6)$$

where the first and the last equation is the boundary condition with  $u_0^n = \rho(x_0, t_n)$  and  $u_N^n = \rho(x_N, t_n)$  as defined in equation (2).

For the Neumann boundary conditions (i.e. in our case for insulated boundaries) we need a special version of equation (5). We use therefor the fact that  $\rho_{-1}^{n+1} \equiv \rho_1^{n+1}$  and get for the left boundary at  $j = 0$

$$(1 + 2\alpha)\rho_0^{n+1} - 2\alpha\rho_1^{n+1} = \underbrace{(1 - 2\alpha)\rho_0^n + 2\alpha\rho_1^n}_{=: u_0^n}. \quad (7)$$

For the right boundary we use the fact that  $\rho_{N-1}^{n+1} \equiv \rho_{N+1}^{n+1}$  at  $j = N$  and get

$$(1 + 2\alpha)\rho_{N-1}^{n+1} - 2\alpha\rho_N^{n+1} = \underbrace{(1 - 2\alpha)\rho_N^n + 2\alpha\rho_{N-1}^n}_{=: u_N^n}. \quad (8)$$

Thus for Neumann boundary conditions, where

$$\frac{\partial \rho(x_0, t)}{\partial x} = 0 \quad \text{and} \quad \frac{\partial \rho(x_N, t)}{\partial x} = 0, \quad (9)$$

we have to solve the following matrix equation:

$$\underbrace{\begin{pmatrix} 1+2\alpha & -2\alpha & 0 & \dots & 0 \\ -\alpha & 1+2\alpha & -\alpha & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -\alpha & 1+2\alpha & -\alpha \\ 0 & \dots & 0 & -2\alpha & 1+2\alpha \end{pmatrix}}_{\in \mathbb{R}^{N \times N}} \begin{pmatrix} \rho_0^{n+1} \\ \rho_2^{n+1} \\ \vdots \\ \vdots \\ \rho_{N-1}^{n+1} \\ \rho_N^{n+1} \end{pmatrix} = \begin{pmatrix} u_0^n \\ u_1^n \\ \vdots \\ \vdots \\ u_{N-1}^n \\ u_N^n \end{pmatrix}, \quad (10)$$

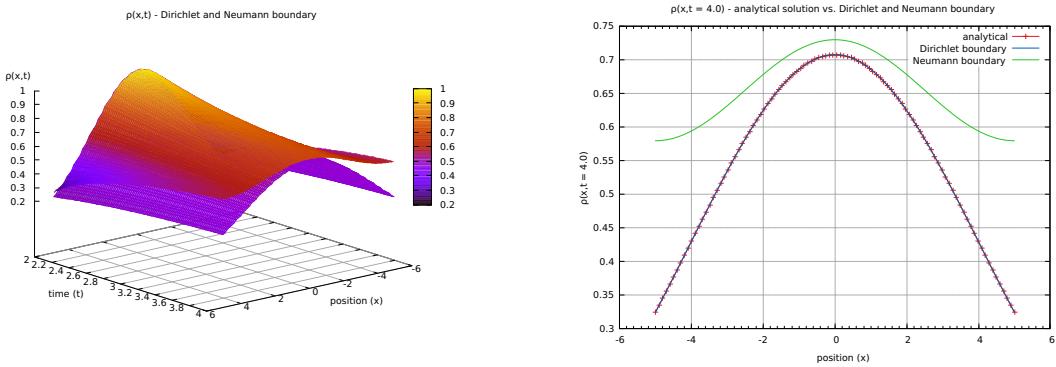
In opposite to the lecture notes we reformulate the linear system in such a way that the boundary specifications are included in the linear system and do not need to be updated separately. We further use the centered difference to approximate  $\frac{\partial \rho(x_0, t)}{\partial x}$  and  $\frac{\partial \rho(x_N, t)}{\partial x}$  (instead of forward or backward difference to represent the Neumann boundary condition at the left and the right end of the domain), by constructing the so-called "false points" such that

$$\frac{\partial \rho(x_0, t_{n+1})}{\partial x} = \frac{\rho_1^{n+1} - \rho_{-1}^{n+1}}{2\Delta x} + \mathcal{O}(\Delta x^2) = 0 \text{ and } \frac{\partial \rho(x_N, t_{n+1})}{\partial x} = \frac{\rho_{N-1}^{n+1} - \rho_{N+1}^{n+1}}{2\Delta x} + \mathcal{O}(\Delta x^2) = 0. \quad (11)$$

However, since both linear systems have a tridiagonal structure we can use the Thomas algorithm to solve the system for every time step  $t_n$ .

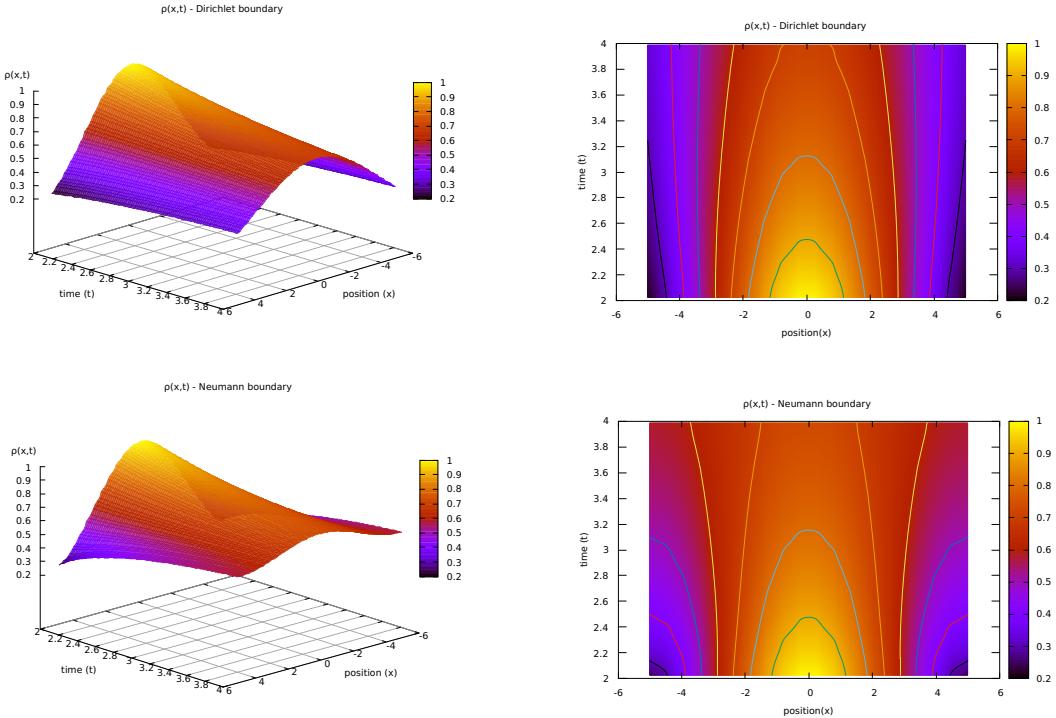
**The graphical representation of the solution for both boundary conditions at  $t = 4.0$  and its comparison with the analytic solution.**

The results are visualized down below:

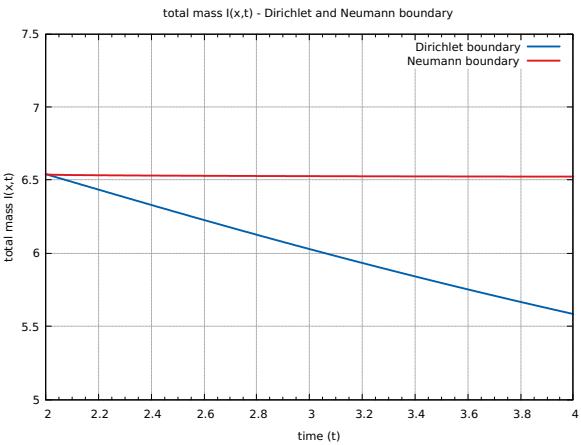


The left graphic shows the evolution of  $\rho(x, t)$  and the right graphic represent the solution for  $\rho(x, t)$  at  $t = 4.0$ . We see that the implementation by the Dirichlet boundaries are almost identical with the analytical solution, whereas the solution with Neumann boundaries differ clearly from the aforesaid.

For a better representation we have plotted the Dirichlet and the Neumann boundary approach as a 3D plot and as contour plot that show  $\rho(x, t)$  over the time period  $t \in [2.0, 4.0]$ .



**The total mass inside the computational domain as a function of time in the interval  $t \in [2.0, 4.0]$  for each boundary condition. Provide an explanation for the observed trends.**



We see that the total mass  $I(x,t)$  inside the computational domain as a function of time in the interval  $t \in [2.0, 4.0]$  for the Neumann boundary remains constant, whereas the total mass  $I(x,t)$  of the Dirichlet boundary declines over time. The reason is that a Dirichlet boundary condition is one which allows for flow to occur at a boundary. A Neumann boundary condition does not allow for flow to occur at a boundary. To calculate the integral of the total mass  $I(x,t)$  we use

Simpson's 1/3 rule, which is given by

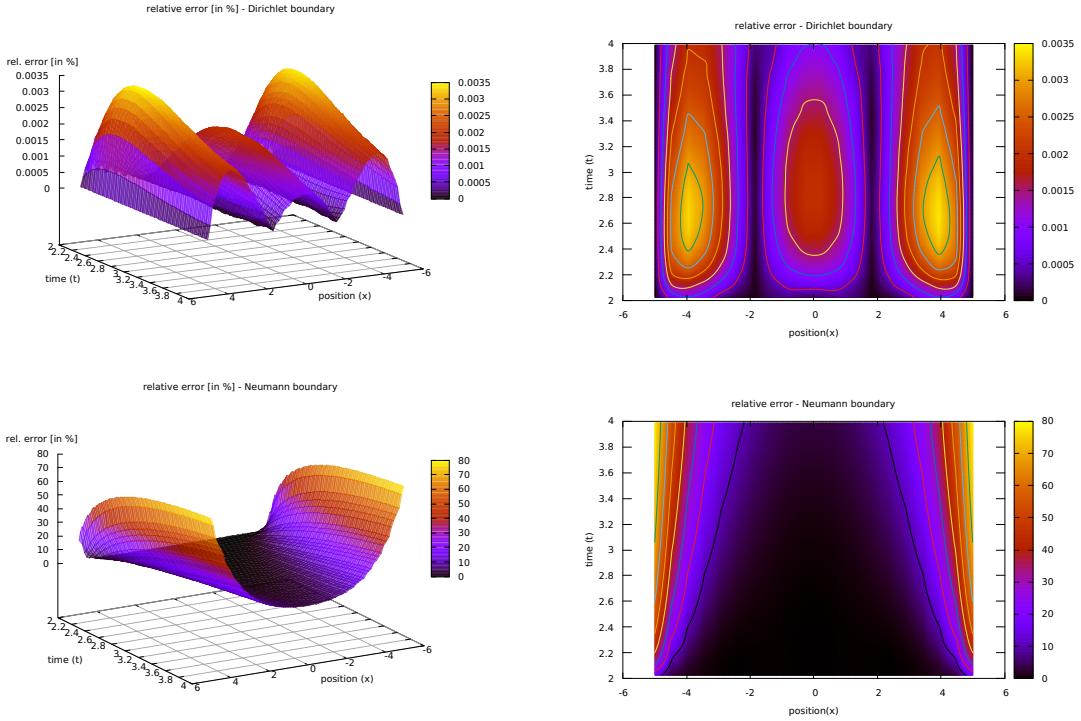
$$\int_a^b f(x)dx \approx \frac{b-a}{6} \left( f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right). \quad (12)$$

Because we have 99 grid points by construction and thus an odd number of integration points we use the Simpson's 3/8 rule, which is given by

$$\int_a^b f(x)dx \approx \frac{b-a}{8} \left( f(a) + 3f\left(\frac{2a+b}{2}\right) + 3f\left(\frac{a+2b}{2}\right) + f(b) \right), \quad (13)$$

for the last integration point to get a better approximation.

**The analysis of the relative error between the numerical and exact solutions as a function of position  $x$  (also in the graphical form). Where are the minimum and maximum relative errors found (provide values)?**



We see that the relative errors for the Dirichlet boundary approach at  $x = -2$  and  $x = 2$  and at the boundaries (obviously because of the chosen boundary conditions) are constantly almost zero over time. The largest deviation for the Dirichlet boundaries occurs at the positions  $x = -4$  and  $x = 4$  at approx.  $t = 2.6$ . For the Neumann boundary condition the relative errors appear at the boundaries as highest and increase over time.

#### Further remarks:

We also have implemented a small sanity-check that solve the linear system by the lapack function **gesv**. To compile the program one need to include the lapack routines. The plots where made

with GNUPLOT (Version 5.2).

Compile program: **gfortran diffusion.f90 -llapack -lblas -L~/lapack -O3 -W**

Output of the program diffusion.f90: **graphplot.txt** including the calculated densities  $\rho(x, t)$ , **integral.txt** including the the total mass  $I(x, t)$ , **graphplotana.txt** including the analytical solution and **graphplotana\_N.txt** including the  $\rho(x, t_N)$  for  $t_n = 4.0$ .

### Fortran code:

```

1  program diffusion
2    implicit none
3    ! define parameters and variables
4    integer, parameter :: N=100, Nmax=1000
5    real*4, parameter :: D=2.,t0=2.
6    real*4, dimension(N) :: x, rho_new, rho, analytical, u, ucheck, ones, p, q, r, s, pivot
7    real*4, dimension(N) :: acc_array
8    real*4, dimension(Nmax) :: integral, tlist
9    real*4, dimension(Nmax,N) :: res, errorres
10   real*4, dimension(N,N) :: M
11   integer i, j, k, counter, rc, stepplot
12   integer BC
13
14
15 ! read in the boundary conditions
16 write(6,*) 'Enter the boundary conditions for the one-dimensional diffusion equation: Dirichlet ='
17 !      Neumann = 2'
18 read(5,*) BC
19
20 L = 10.
21 delta_x = L/(N-1)
22 rho = 0.
23 rho_new = 0.
24 delta_t = (delta_x*delta_x)/(2*D)
25 eps = 1.0e-6
26 stepplot = 10
27
28 ! set initial values
29 do i=1,N
30   x(i) = (i-1)*delta_x-L/2
31 enddo
32
33 do i=1,N
34   rho(i) = initial(x(i))
35 enddo
36
37 res(1,:) = rho
38 counter = 1
39 t = t0 + delta_t
40 tlist(1) = 2.
41 integral(1) = simpson(rho, -5.,5.,N-1)
42
43 ! boundary conditions
44 if (BC.eq.1) then          ! Dirichlet BC
45 do while(t .le. 4.0)
46 ! define alpha and A = (1 + 2*alpha)
47 alpha = delta_t*D/(2*delta_x*delta_x)
48 A = (1 + 2*alpha)
49
50 ! determin u the right-hand side
51 do i=1,N
52   if (i == 1) then
53     u(i) = density(x(1),t)
54   else if (i == N) then
55     u(i) = density(x(N),t)
56   else
57     u(i) = alpha*rho(i-1)+(1-2*alpha)*rho(i)+alpha*rho(i+1)
58   endif
59 enddo
60
61 ! determin the matrix for each time step in matrix and vector form
62 ! matrix form
63 M = 0.
64 M = reshape(M,[N,N])
65 do i = 1,N
66   M(i,i) = A
67   if(i .lt. N) then
68     M(i,i+1) = -alpha
69     M(i+1,i) = -alpha
70   endif
71 enddo
72
73 M(1,1) = 1.
74 M(N,N) = 1.
75 M(1,2) = 0.
76 M(N,N-1)= 0.
77
78 ! vector form

```

```

78 ones = 1.
79 p = -alpha*ones
80 q = A*ones
81 r = -alpha*ones
82 p(1) = 0.
83 r(N) = 0.
84 q(1) = 1.
85 q(N) = 1.
86 r(1) = 0.
87 p(N) = 0.
88 ! solve the system by the thomas algorithm and by the lapack function for testing reasons
89 rho_new = thomasVec(p,q,r,u)
90 call gesev(N, 1, M, N, pivot, u, N, rc)
91
92 ! run test between thomas algorithm and lapack function (eps = 1.0e-6 )
93 do i=1,N
94   if (abs(rho_new(i))-u(i)) .gt. eps) then
95     print*, 'Rho check encountered an error!'
96     stop
97   endif
98 enddo
99
100 ! calculate the analytical solution
101 do i=1,N
102   analytical(i) = density(x(i),t)
103 enddo
104
105 ! calculate the mass by the simpson rule and store the values for each time step
106 integral(counter+1) = simpson(rho_new,-5.0,5.0,N-1)
107 ! calculate the relative error for each time step and store the values
108 errores(counter,:)= abs((rho_new-analytical)/analytical)*100.0
109
110 ! calculate the accuracy
111 do k=2,N-1
112   acc_array(k) = abs(rho_new(k)-rho(k))/minval([rho(k),rho_new(k)])
113 enddo
114 acc = maxval(acc_array)
115
116 ! store the rho values for the new time step
117 rho = rho_new
118
119 ! determine the accuracy configuration
120 if (acc.lt. 0.15) then
121   t = t + delta_t
122   counter = counter + 1
123   res(counter,:)= rho_new
124   tlist(counter) = t
125 else
126   delta_t = delta_t/2
127   t = t + delta_t
128 endif
129
130 enddo
131 else if (BC.eq.2) then      ! Neumann BC
132 do while(t.le. 4.0)
133   ! define alpha and A = (1 + 2*alpha)
134   alpha = delta_t*D/(2*delta_x*delta_x)
135   A = (1 + 2*alpha)
136
137   ! determin u the right-hand side
138   do i=1,N
139     if (i == 1) then
140       u(i) = (1-2*alpha)*rho(i)+2*alpha*rho(i+1)
141     else if (i == N) then
142       u(i) = 2*alpha*rho(i-1)+(1-2*alpha)*rho(i)
143     else
144       u(i) = alpha*rho(i-1)+(1-2*alpha)*rho(i)+alpha*rho(i+1)
145     endif
146   enddo
147
148   ! determin the matrix for each time step in matrix and vector form
149   ! matrix form
150   M = 0.
151   M = reshape(M,[N,N])
152   do i = 1,(N)
153     M(i,i) = A
154     if(i.lt. N) then
155       M(i,i+1) = -alpha
156       M(i+1,i) = -alpha
157     endif
158   enddo
159
160   M(1,2) = -2.*alpha
161   M(N,N-1)= -2.*alpha
162
163   ! vector form
164   ones = 1.
165   p = -alpha*ones
166   q = A*ones
167   r = -alpha*ones
168   p(1) = 0.
169   r(N) = 0.
170   r(1) = -2.*alpha
171   p(N) = -2.*alpha
172
173   ! solve the system by the thomas algorithm and by the lapack function for testing reasons
174   rho_new = thomasVec(p,q,r,u)

```

```

175 call sgessv(N, 1, M, N, pivot, u, N, rc)
176
177 ! run test between thomas algorithm and lapack function (eps = 1.0e-6)
178 do i=1,N
179   if (abs(rho_new(i)-u(i)) .gt. eps) then
180     print*, 'Rho check encountered an error!'
181     stop
182   endif
183 enddo
184
185 ! calculate the analytical solution
186 do i=1,N
187   analytical(i) = density(x(i),t)
188 enddo
189
190 ! calculate the mass by the simpson rule and store the values for each time step
191 integral(counter+1) = simpson(rho_new,-5.0,5.0,N-1)
192 ! calculate the relative error for each time step and store the values
193 errorres(counter,:)= abs(((rho_new-analytical)/analytical)*100.0)
194
195 ! calculate the accuracy
196 do k=2,N-1
197   acc_array(k) = abs(rho_new(k)-rho(k))/minval([rho(k),rho_new(k)])
198 enddo
199 acc = maxval(acc_array)
200
201 ! store the rho values for the new time step
202 rho = rho_new
203
204 ! determine the accuracy configuration
205 if (acc.lt. 0.15) then
206   t = t + delta_t
207   counter = counter + 1
208   res(counter,:) = rho_new
209   tlist(counter) = t
210 else
211   delta_t = delta_t/2
212   t = t + delta_t
213 endif
214
215 enddo
216 else
217   print*, 'no valid boundary condition'
218 endif
219
220 ! print steps, rho(x,t=4.0), analytical solution (t=4.0), relative errors(t=4.0) and total mass
221 print*, 'steps:'
222 print*, counter
223 print*, 'numerical solution for t=4.0:'
224 print*, res(counter,:)
225 print*, 'analytical solution for t=4.0:'
226 print*, analytical
227 print*, 'relative errors for t=4.0:'
228 print*, errorres(counter-1,:)
229 print*, 'total mass as a function of x (simpsons rule):'
230 print*, integral(1:counter)
231 !print*, tlist(1:counter)
232
233 ! print out the plotting variables
234 open(11,file='graphplot.txt',status='unknown')
235 open(12,file='integral.txt',status='unknown')
236 open(13,file='error.txt',status='unknown')
237 k = 0.
238 do j=1,counter-1
239   if ((mod(j,stepplot) == 0) .or. (t .lt. 4.0)) then
240     do i=1,N
241       write(11,*) x(i), tlist(j), res(j,i)
242       write(13,*) x(i), tlist(j), errorres(j,i)
243     enddo
244   endif
245   write(12,*) tlist(j), integral(j)
246 enddo
247
248 open(14,file='graphplotana.txt',status='unknown')
249 open(15,file='graphplotana_N.txt',status='unknown')
250 do j=1,N
251   write(14,*) x(j), analytical(j)
252   write(15,*) x(j), res(counter,j)
253 enddo
254
255 contains
256 ! analytical solution
257 function density(x,t) result(dens)
258   implicit none
259   real*4, parameter :: t0 = 2.0, D = 2.0
260   real*4 t, x, dens
261   dens = sqrt(t0/t)*exp(-(x*x)/(4*D*t))
262 end function density
263
264 ! initial values
265 function initial(x) result(dens)
266   implicit none
267   real*4, parameter :: t0 = 2.0, D = 2.0
268   real*4 x, dens
269   dens = exp(-(x*x)/(4*D*t0))
270 end function initial
271
```

```

272 ! thomas algorithm
273 function thomasVec(p,q,r,s) result(res)
274   implicit none
275   integer i, k
276   real*4, dimension(N) :: p, q, r, s, res, inits
277   integer, parameter :: N = size(s)
278
279   inits = s
280   do k = 2, N
281     a = p(k)/q(k-1)
282     q(k) = q(k) - a*r(k-1)
283     inits(k) = inits(k) - a*inits(k-1)
284   enddo
285   res(N) = inits(N)/q(N)
286   do k = N-1,1,-1
287     res(k) = (inits(k)-r(k)*res(k+1))/q(k)
288   enddo
289 end function thomasVec
290
291 ! simpson's rule that combines the 1/3 rule and the 3/8 rule
292 function simpson(f,a,b,N) result(integral)
293 implicit none
294 integer N, i
295 real*4 a, b, integral, s
296 real*4 h, x
297 real*4, dimension(N) :: f
298
299 s = 0.0
300 h = (b-a)/N
301 do i=3, N-4, 2
302   s = s + 2.0*f(i) + 4.0*f(i+1)
303 end do
304 integral = (s + f(1) + 4.0*f(2))*h/3.0 + (3*h/8)*(f(N-3)+ 3*f(N-2) + 3*f(N-1) + f(N))
305 return
306 end function simpson
307 end program diffusion

```

**Fortran code 1:** One-dimensional diffusion equation