# 040692 PR Practical in numerical Astronomy
# University of Vienna
# One-dimensional diffusion equation

Dawid Stepanovic

December 30, 2021

**One-dimensional diffusion equation:**

$$\frac{\partial \rho}{\partial t} = D\frac{\partial^2 \rho}{\partial x^2} \tag{1}$$

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:

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

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:

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

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 contains:**

- 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 the sum of the forward-time centered-space (FTCS) and backward-time centered-space (BTCS) schemes

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

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

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

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

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ -\alpha & 1+2\alpha & -\alpha & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -\alpha & 1+2\alpha & -\alpha \\ 0 & \dots & 0 & 0 & 1 \end{pmatrix}}_{\in \mathbb{R}^{N\times N}} \begin{pmatrix} \rho_0^{n+1} \\ \rho_2^{n+1} \\ \vdots \\ \vdots \\ \rho_{N-1}^{n+1} \\ \rho_N^{n+1} \end{pmatrix} = \begin{pmatrix} u_0^n \\ u_1^n \\ \vdots \\ \vdots \\ u_{N-1}^n \\ u_N^n \end{pmatrix}, \tag{6}$$

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 $j = 0$

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

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

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

Thus for Neumann boundary conditions, where

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

we have to solve the following matrix equation:

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

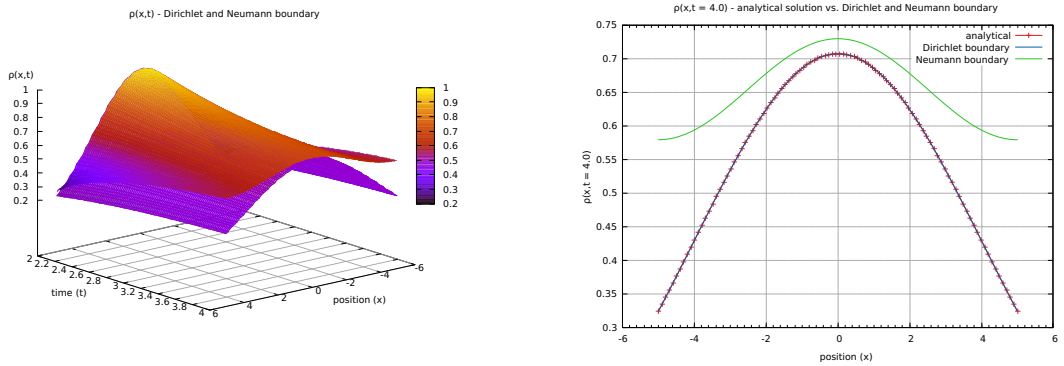
In opposite to the lecture notes we reformulate the linear system in such a way that the boundary specifications are included in the linear system and do not need to be updated separately. We further use the centered difference to approximate $\frac{\partial \rho(x_0,t))}{\partial x}$ and $\frac{\partial \rho(x_N,t))}{\partial x}$ (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

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

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

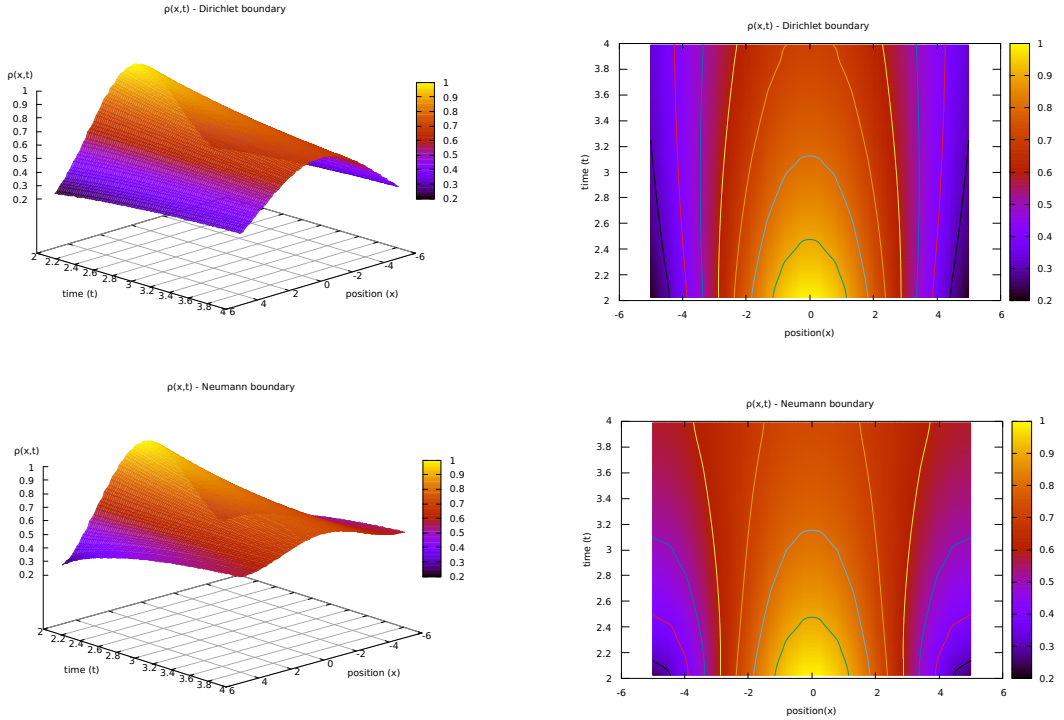
**The graphical representation of the solution for both boundary conditions at $t = 4.0$ and its comparison with the analytic solution.**

The results are visualized down below:

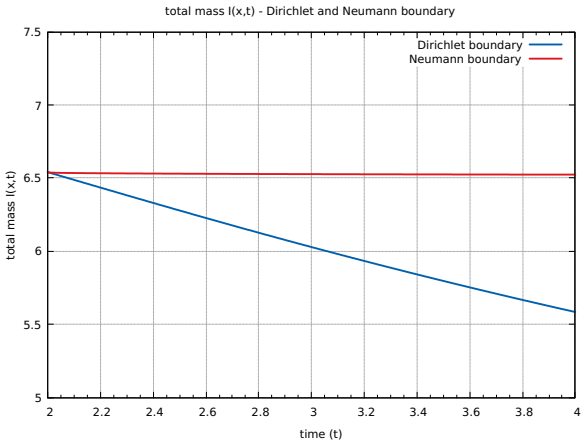



The left graphic shows the evolution of $\rho(x,t)$ and the right graphic represent the solution for $\rho(x,t)$ at $t = 4.0$. We see that the implementation by the Dirichlet boundaries are almost identical with the analytical solution, whereas the solution with Neumann boundaries differ clearly from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

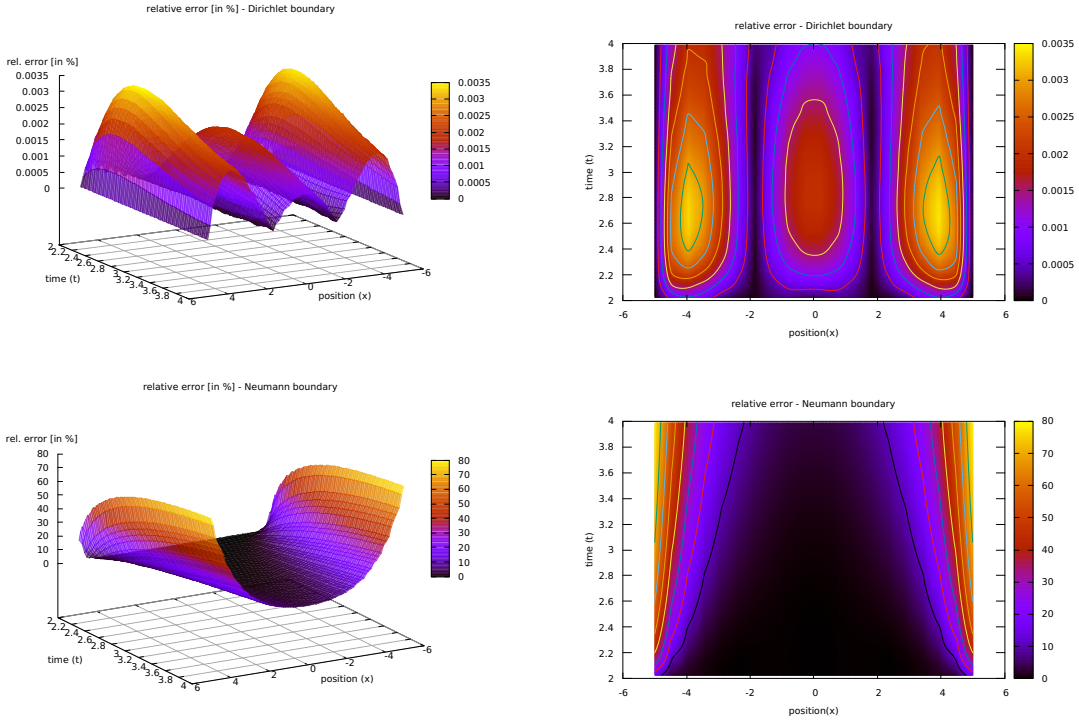
$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right). \tag{12}$$

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

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

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 **sgesv**. 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:**

```
program diffusion
  implicit none
  ! define parameters and variables
  integer, parameter :: N=100, Nmax=1000
  real*4,  parameter :: D=2.,t0=2.
  real*4   delta_t, delta_x, A, alpha, t, acc, L, eps
  real*4,  dimension(N) :: x, rho_new, rho, analytical, u, ucheck, ones, p, q, r, s, pivot
  real*4,  dimension(N) :: acc_array
  real*4,  dimension(Nmax) :: integral, tlist
  real*4,  dimension(Nmax,N) :: res, errorres
  real*4,  dimension(N,N) :: M
  integer  i, j, k, counter, rc, stepplot
  integer  BC

  ! read in the boundary conditions
  write(6,*) 'Enter the boundary conditions for the  one-dimensional diffusion equation: Dirichlet =
     1, Neumann = 2'
  read(5,*) BC

  L = 10.
  delta_x = L/(N-1)
  rho = 0.
  rho_new = 0.
  delta_t = (delta_x*delta_x)/(2*D)
  eps = 1.0e-6
  stepplot = 10

  ! set initial values
  do i=1,N
    x(i) = (i-1)*delta_x-L/2
  enddo

  do i=1,N
    rho(i) = initial(x(i))
  enddo

  res(1,:) = rho
  counter = 1
  t = t0 + delta_t
  tlist(1) = 2.
  integral(1) = simpson(rho,-5.,5.,N-1)

 ! boundary conditions
  if (BC.eq.1) then        ! Dirichlet BC
  do while(t .le. 4.0)
  ! define alpha and A = (1 + 2*alpha)
  alpha = delta_t*D/(2*delta_x*delta_x)
  A = (1 + 2*alpha)

  ! determin u the right-hand side
  do i=1,N
  if (i == 1) then
    u(i) = density(x(1),t)
  else if (i == N) then
    u(i) = density(x(N),t)
  else
    u(i) = alpha*rho(i-1)+(1-2*alpha)*rho(i)+alpha*rho(i+1)
  endif
  enddo

  ! determin the matrix for each time step in matrix and vector form
  ! matrix form
  M = 0.
  M = reshape(M,[N,N])
  do i = 1,N
  M(i,i) = A
  if(i .lt. N) then
    M(i,i+1) = -alpha
    M(i+1,i) = -alpha
  endif
  enddo

  M(1,1)   = 1.
  M(N,N)   = 1.
  M(1,2)   = 0.
  M(N,N-1)= 0.

  ! 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
89    ! solve the system by the thomas algorithm and by the lapack function for testing reasons
90    rho_new = thomasVec(p,q,r,u)
91    call sgesv(N, 1, M, N, pivot, u, N, rc)
92
93    ! run test between thomas algorithm and lapack function (eps = 1.0e-6 )
94    do i=1,N
95      if(abs(rho_new(i)-u(i)) .gt. eps) then
96        print*, 'Rho check encountered an error!'
97        stop
98      endif
99    enddo
100
101   ! calculate the analytical solution
102   do i=1,N
103   analytical(i) = density(x(i),t)
104   enddo
105
106   ! calculate the mass by the simpson rule and store the values for each time step
107   integral(counter+1) = simpson(rho_new,-5.0,5.0,N-1)
108   ! calulate the relative error for each time step and store the values
109   errorres(counter,:) = abs(((rho_new-analytical)/analytical)*100.0)
110
111   ! calculate the accuracy
112   do k=2,N-1
113     acc_array(k) = abs(rho_new(k)-rho(k))/minval([rho(k),rho_new(k)])
114   enddo
115   acc = maxval(acc_array)
116
117   ! store the rho values for the new time step
118   rho = rho_new
119
120   ! determine the accuracy configuration
121   if (acc.lt. 0.15) then
122     t = t + delta_t
123     counter = counter + 1
124     res(counter,:) = rho_new
125     tlist(counter) = t
126   else
127       delta_t = delta_t/2
128     t = t + delta_t
129   endif
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)
```

```
  call sgesv(N, 1, M, N, pivot, u, N, rc)

  ! run test between thomas algorithm and lapack function (eps = 1.0e-6)
  do i=1,N
    if(abs(rho_new(i)-u(i)) .gt. eps) then
      print*, 'Rho check encountered an error!'
      stop
    endif
  enddo

  ! calculate the analytical solution
  do i=1,N
    analytical(i) = density(x(i),t)
  enddo

  ! calculate the mass by the simpson rule and store the values for each time step
  integral(counter+1) = simpson(rho_new,-5.0,5.0,N-1)
  ! calulate the relative error for each time step and store the values
  errorres(counter,:) = abs(((rho_new-analytical)/analytical)*100.0)

  ! calculate the accuracy
  do k=2,N-1
      acc_array(k) = abs(rho_new(k)-rho(k))/minval([rho(k),rho_new(k)])
  enddo
  acc = maxval(acc_array)

  ! store the rho values for the new time step
  rho = rho_new

  ! determine the accuracy configuration
  if (acc.lt. 0.15) then
    t = t + delta_t
    counter = counter + 1
    res(counter,:) = rho_new
    tlist(counter) = t
  else
      delta_t = delta_t/2
    t = t + delta_t
  endif

  enddo
  else
    print*,'no valid boundary condition'
  endif

  ! print steps, rho(x,t=4.0), analytical solution (t=4.0), relative errors(t=4.0) and total mass
  print*,'steps:'
  print*, counter
  print*,'numerical solution for t=4.0:'
  print*, res(counter,:)
  print*,'analytical solution for t=4.0:'
  print*, analytical
  print*,'relative errors for t=4.0:'
  print*, errorres(counter-1,:)
  print*,'total mass as a function of x (simpsons rule):'
  print*, integral(1:counter)
  !print*, tlist(1:counter)

! print out the plotting variables
  open(11,file='graphplot.txt',status='unknown')
  open(12,file='integral.txt',status='unknown')
  open(13,file='error.txt',status='unknown')
  k = 0.
  do j=1,counter-1
  if ((mod(j,stepplot) == 0) .or. (t .lt. 4.0)) then
    do i=1,N
      write(11,*) x(i), tlist(j), res(j,i)
      write(13,*) x(i), tlist(j), errorres(j,i)
    enddo
  endif
  write(12,*) tlist(j), integral(j)
  enddo

  open(14,file='graphplotana.txt',status='unknown')
  open(15,file='graphplotana_N.txt',status='unknown')
  do j=1,N
    write(14,*) x(j), analytical(j)
    write(15,*) x(j), res(counter,j)
  enddo

contains
! analytical solution
function density(x,t) result(dens)
  implicit none
  real*4, parameter :: t0 = 2.0, D = 2.0
  real*4 t, x, dens
  dens = sqrt(t0/t)*exp(-(x*x)/(4*D*t))
end function density

! initial values
function initial(x) result(dens)
  implicit none
  real*4, parameter :: t0 = 2.0, D = 2.0
  real*4 x, dens
  dens = exp(-(x*x)/(4*D*t0))
end function initial
```

```
! thomas algorithm
function thomasVec(p,q,r,s) result(res)
  implicit none
  integer i, k
  real*4, dimension(N) :: p, q, r, s, res, inits
    integer, parameter :: N = size(s)

  inits = s
  do k = 2, N
    a    = p(k)/q(k-1)
    q(k) = q(k) - a*r(k-1)
    inits(k) = inits(k) - a*inits(k-1)
  enddo
  res(N) = inits(N)/q(N)
  do k = N-1,1,-1
    res(k) = (inits(k)-r(k)*res(k+1))/q(k)
  enddo
end function thomasVec

! simpson's rule that combines the 1/3 rule and the 3/8 rule
function simpson(f,a,b,N) result(integral)
implicit none
integer N, i
real*4  a, b, integral, s
real*4  h, x
real*4, dimension(N) :: f

s = 0.0
h = (b-a)/N
do i=3, N-4, 2
   s   = s + 2.0*f(i) + 4.0*f(i+1)
end do
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))
return
end function simpson
end program diffusion
```

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