Ay 190 Assignment 16

The Diffusion Equation

March 20, 2013

1 Diffusion in 1D

The 1D diffusion equation

$$\frac{\partial y}{\partial t} - D\frac{\partial^2 y}{\partial x^2} = 0$$

where D is a diffusion coefficient, has the analytic solution

$$y(\mathbf{x}, t) = \left(\frac{t_0}{t_0 + t}\right)^{1/2} \exp -\frac{|\mathbf{x} - \mathbf{x}_0|^2}{4D(t_0 + t)}.$$

The diffusion equation with $D = 1$ and $t_0 = 1$ was solved numerically and compared to the analytic solution.

1.1 The FTCS method

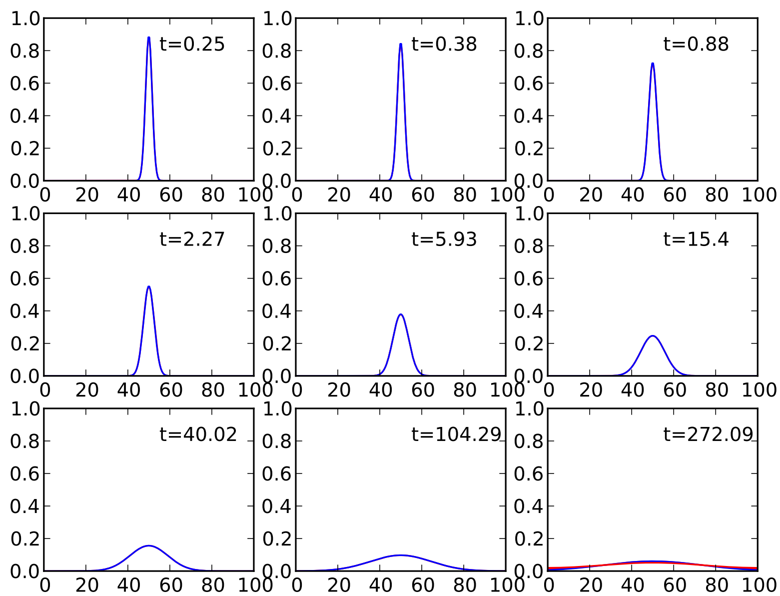

Figure 1: Evolution of the numerical (red) and analytic (blue) solution. The numerical solution was computed using a stepsize $\Delta x = 0.503$ and a time step $\Delta t = 0.126$ (chosen to satisfy the FTCS stability condition $\Delta t \leq \Delta x^2/2D$) and the FTCS method to discretize the spatial derivative. Initially, there is generally good agreement with the analytical solution, but the error grows at later times (see Figure 2 for details).

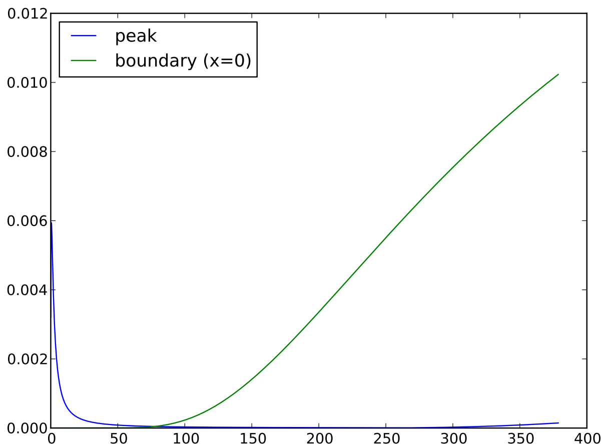

Figure 2: Error in the numerical solution. The error was calculated at two points, the peak of the gaussian (x=50) and at the boundary (x=0). Initially, the numerical solution underpredicts the height of the gaussian, but produces a more accurate solution, then at around $t = 300$, begins to diverge again, this time overpredicting the height. At the boundaries, the converse is true; the initial error is very small, but then increases at later times.

2 Diffusion in 2D

The diffusion equation in 2D is

$$\frac{\partial f}{\partial t} - D\left(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}\right) = 0.$$

2.1 Initial Gaussian pulse

The 2D diffusion equation with D=1 was solved with the initial condition

$$f(x, y, t = 0) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x - x_0)^2 + (y - y_0)^2}{2\sigma^2}\right]$$

with $x_0 = y_0 = 50$ and $\sigma = 10$.

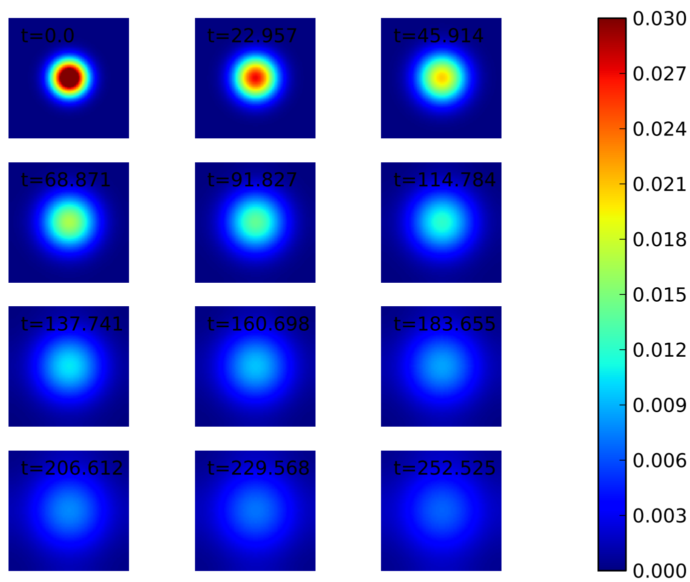

Figure 3: Evolution of the numerical solutionto the 2D diffusion equation for an initial gaussian pulse over the domain [0,100] × [0,100]. The solution was computed using stepsizes of $\Delta x = \Delta y = 1.010$ and a time step of $\Delta t = 0.255$, chosen to satisfy the more stringent 2D FTCS stability condition $\Delta t \leq \Delta x^2/4D$. The initial pulse gradually disappears as it diffuses outwards.

2.2 Initial ring-like pulse

The 2D diffusion equation with $D = 1$ was again solved but with the initial condition

$$f(x, y, t = 0) = \begin{cases} 1 & 0.05 \leq \sqrt{(x - 0.5)^2 + (y - 0.5)^2} \leq 0.1 \\ 0 & elsewhere \end{cases}$$

i.e. a ring of width 0.05 centered at $(0.5, 0.5)$, on the domain [0,1] × [0,1].

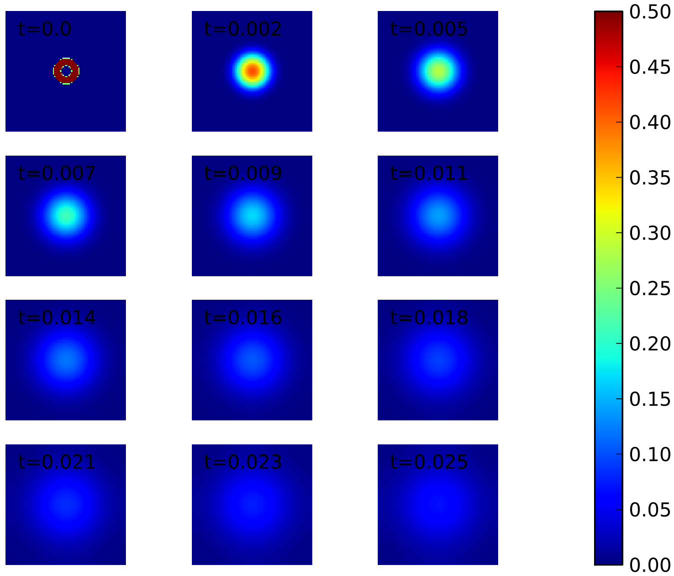

Figure 4: Evolution of the numerical solution to the 2D diffusion equation for an initial ring-like pulse over the domain [0,1] × [1,0]. The solution was computed using stepsizes of $\Delta x = \Delta y = 0.01$ and a time step of $\Delta t = 0.00003$, chosen to satisfy the 2D FTCS stability condition. As before, the initial pulse gradually disappears as it diffuses outwards.

Appendices

Referenced `http://pauli.uni-muenster.de/tp/fileadmin/lehre/NumMethoden/WS0910/ScriptPDE/Heat.pdf` for additional info on numerical solutions to the 1D and 2D diffusion equation.

The Python modules and scripts used in this assignment include

A 1D Diffusion: `diffuse1D.py` (see page 3-)

B 2D Diffusion: `diffuse2D.py` (see page 3-)

A 1D Diffusion

```
from pylab import *
import sys,math

# ==============================================================================
# SUPPORTING FUNCTIONS

def analytic(t,t0,x,x0,D):
    return (t0/(t0+t))**0.5 * exp( -(x-x0)**2 / (4*D*(t0+t)) )


def calc_rhs(D,y,dx):
    """
    Returns the spatial portion of the diffusion equation, calculating the second
    derivative of y using the centered finite difference method.
    """
    ddy=(y[2:]-2*y[1:-1]+y[:-2])/(dx*dx)
    return D*ddy*dt


def set_bound(y):
    y[0] = y[1]
    y[n-1] = y[n-2]
    return y


# ==============================================================================
# SOLVE THE 1D DIFFUSION EQUATION

n = 200
x = linspace(0,100,n)
dx = x[1]-x[0]
dx2 = dx*dx

x0 = 50.0
D = 1.0
t0 = 1.0
t = 0.0
```

```
nt = 3000

# initial conditions
y = exp( -(x-x0)**2 / (4.0*D*t0) )
dt = dx2/(2*D) # max timestep before FTCS becomes unstable
print 'dt =',dt,'dx =',dx

nplot=0
ion()
plot(x,y,"r-")
plot(x,analytic(t,t0,x,x0,D),"bx-")
show()

ie=len(x)/2 # peak of gaussian where error analysis done
err_peak=zeros(nt)
err_side=zeros(nt)
for it in range(nt):
    y[1:-1] += calc_rhs(D,y,dx)
    y = set_bound(y)
    t += dt

    yana=analytic(t,t0,x,x0,D)

    #if it % 10 == 0:
    if it == int(10**(nplot/2.4)) and nplot <= 8:
        nplot+=1
        subplot(3,3,nplot)
        clf()
        plot(x,y,'r')
        plot(x,yana,'b')
        ylim([0,1])
        text(55,0.8,'t={0}'.format(round(t,2)))
        draw()

    err_peak[it]=abs(yana[ie]-y[ie])
    err_side[it]=abs(yana[0]-y[0])

ioff()
plot(x,y,"r")
savefig('1D_evol.pdf')
show()

plot(arange(0,nt*dt,dt),err_peak)
plot(arange(0,nt*dt,dt),err_side)
legend(['peak','boundary (x=0)'],loc='best')
savefig('1D_error.pdf')
show()
```

B 2D Diffusion

```
#!/usr/bin/env python
#from __future__ import division
from matplotlib.patches import Patch
from pylab import *


# ==============================================================================
# SUPPORTING FUNCTIONS

def gauss(x,y,x0,y0,sig):
    """
    Returns a 2D gaussian centered at (x0, y0) and with width sig.
    Assumes that x and y are in the form returned by numpy.meshgrid().
    Used to set initial condition.
    """
    return exp( -((x-x0)**2 + (y-y0)**2)/(2*sig**2) ) / (sig*sqrt(2*pi))


def ring(x,y):
    """
    Returns a ring of width 0.05 stretching from 0.05 to 0.1 from the center at
    (0.5,0.5).  Assumes that x and y are in the form returned by numpy.meshgrid().
    Used to set intiial condition.
    """
    val=sqrt( (x-0.5)**2 + (y-0.5)**2 )
    z=array([array([1.0 if 0.05 <= r <= 0.1 else 0.0 for r in row]) for row in val])
    return z


def calc_ddxy(Z,nx,ny,dx2):
    """
    Returns the spatial second derivative of the diffusion equation, using the
    centered finite difference method.
    """
    ddy=array([(Z[i,2:]-2*Z[i,1:-1]+Z[i,:-2])/(dx*dx) for i in range(1,nx-1)])
    ddx=array([(Z[2:,i]-2*Z[1:-1,i]+Z[:-2,i])/(dx*dx) for i in range(1,ny-1)])

    return ddy+ddx.T


def set_bound(Z,nx,ny):
    # boundaries
    Z[0,0] = Z[1,1]
    Z[0,:] = Z[1,:]
    Z[:,0] = Z[:,1]
    Z[nx-1,ny-1] = Z[nx-2,nx-2]
    Z[nx-1,:] = Z[nx-2,:]
    Z[:,ny-1] = Z[:,ny-2]
```

```
    return Z


# ================================================================================
# SOLVE THE 2D DIFFUSION EQUATION

D = 1.0
nit = 1000
nx = 100
ny = 100
"""
x0 = 50
y0 = 50
sig = 10.0
x = linspace(0, 100, nx)
y = linspace(0, 100, ny)
X,Y = meshgrid(x, y)
Z = gauss(X,Y,x0,y0,sig)
"""
x = linspace(0,1,nx)
y = linspace(0,1,nx)
X,Y = meshgrid(x,y)
Z = ring(X,Y)

dx = x[1]-x[0]
dx2 = dx*dx
idx2 = 1.0/dx2
dt = dx2/(4*D) # max timestep before FTCS becomes unstable
print 'dx =',dx,'\ndt =',dt

RHS = zeros((nx,ny))

ion()
pcolor(X, Y, Z, vmin=0.0, vmax=0.5)
colorbar()
nplot=0
for it in range(nit):
    Z[1:-1,1:-1] += D*calc_ddxy(Z,nx,ny,dx2)*dt
    Z = set_bound(Z,nx,ny)

    """
    if it % 50 == 0:
        clf()
        pcolor(X, Y, Z, vmin=0.0, vmax=0.5)
        colorbar()
        text(10,80,'t={0}'.format(round(it*dt,3)))
        #fname = 'frame%04d.png'%it
        #print 'Saving frame', fname
        #savefig(fname)
        draw()
    """
```

```
    if it % 90 == 0 and nplot < 15:
        nplot+= 1 if (nplot+1) % 4 != 0 else 2
        subplot(4,4,nplot,aspect='equal')
        pcolor(X, Y, Z, vmin=0.0, vmax=0.5)
        axis('off')
        text(0.1,0.8,'t={0}'.format(round(it*dt,3)))
        draw()

ioff()
savefig('2Dring_evol.pdf')
show()
```