

Isi Denklemi

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

olarak gosterilen denklem fizikte isi denklemi olarak bilinir, u fonksiyonu iki degiskenlidir $u(x, t)$. Ornek icin bu denklemin cozumunu tek boyutta gosterecegiz, yani bir genisligi onemli olmayan bir demir cubugu uzerinde isinin dagilmasi konusuna bakacagiz, boyutu temsil icin x degiskeni kullanılacak. t degiskeni zamani temsil ediyor olacak. Baslangic sartlari (initial conditions) olarak isinin $t=0$ aninda demir cubuk uzerinde x 'e bagli bir sinus fonksiyonu ile dagildigini farzedecegiz, sinir sartlari ise (boundary conditions) cubugun iki ucunun sifir derecede tutulmasi olacak. Sonucta isinin nereye gidecegini tahmin ederek te soyleyebiliriz – isi demirin iki ucundan kacarak tum cubuk boyunca sifir dereceye inecektir.

Ustteki denklem bir kısmi diferansiyel denklemdir (partial differential equation).

Matematiksel cozumler ya analitik, ya da yaklasiksal olur. Biz bu ornegi cozmek icin yaklasiksal, hesapsal bir teknik kullanacagiz. Elimizde bir diferansiyel denklem varsa cozum bulmak demek bir fonksiyon bulmak demektir, bir sayi degil; yaklasiksal yontemle de oyle bir u fonksiyonu bulacagiz ki, test / belli noktalarda gercek fonksiyonla olabildigince ayni sonuclar verecek.

Cozumde sinirli farklar (finite differences) denen bir metot kullanılacak. Bu yaklasiksal metotta calculus'un sonsuz ufakliklar icin kullanılan turevleri, bildigimiz sayisal cikartma islemi uzerinden tanimlanan “farkliliklara” donusecekler. Mesela d^2/dx^2 nedir? x 'e gore turevin turevidir, hesapsal olarak ise farkin farkidir. Sonsuzluktan yaklasiga soyle geceriz: Eger $u_{j,i}$ bir 2 boyutlu dizin uzerinde u fonksiyonunun sayisal degerlerini tasiyor olsaydi, ve j, i indis degerleri t, x 'i temsil ediyorlar ise, x uzerinden birinci turev yani birinci fark (first difference) soyle olur:

$$\frac{u_{j,i+1} - u_{j,i}}{h}$$

h hangi degiskenin farkini aliyorsak, o farkin buyuklugunu tanimlayan aralik degeridir, $h = \Delta x$, ve $u_{j,i+1} = u(t, x + \Delta x)$.

İkinci fark, farkın farkıdır:

$$\begin{aligned} & \frac{1}{h} \left[\left(\frac{u_{j,i+1} - u_{j,i}}{h} \right) - \left(\frac{u_{j,i} - u_{j,i-1}}{h} \right) \right] \\ &= \frac{u_{j,i+1} - 2u_{j,i} + u_{j,i-1}}{h^2} \end{aligned} \quad (1) \quad (\text{diff})$$

Bu carpımı tüm i değerleri için ve matris üzerinden temsil etmenin yolu şudur: Bir ikinci farklılıklar matrisi A yaratırız:

$$A = \frac{1}{\Delta x^2} \begin{bmatrix} -2 & 1 & 0 & 0 \dots 0 & 0 & 0 \\ 1 & -2 & 1 & 0 \dots 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 \dots 1 & -2 & 1 \\ 0 & 0 & 0 & 0 \dots 0 & 1 & -2 \end{bmatrix}$$

Ve u değerlerinin bir vektör içine çekeriz:

$$U_j = \begin{bmatrix} u_{j,0} \\ u_{j,1} \\ u_{j,2} \\ \vdots \\ u_{j,n} \end{bmatrix}$$

AU_j carpımının ^{diff} denklemindeki toplamları her u için teker teker vereceğini görebiliriz. İndislerden j zaman, i mesafedir, yani üstteki denklem şimdi sadece mesafeyi yani x 'i parçalara bölmüştür.

Zamanı da modele dahil edelim ve çözümü elde etmeye uğralım. İki denkleminin tamamını şimdiye kadar elde ettiklerimizi kullanarak ve ayrışal olarak yazalım:

$$\frac{U_{j+1} - U_j}{\Delta t} = AU_j \quad (2) \quad (\text{main})$$

$\frac{\partial^2 u}{\partial x^2} \approx AU_j$, ve $\frac{\partial u}{\partial t} \approx (U_{j+1} - U_j)/\Delta t$ olarak alındı. U_j tanımındaki j indisi zaman için kullanılıyor, mesafe yani x 'i temsil eden indislerin tamamı U 'nın içinde var zaten.

Yaklaşıksal tekniklerden Crank-Nicholson'a göre AU_j 'i ardi ardına iki zaman

indisi üzerinden hesaplanan bir ortalama olarak temsil edebiliriz, yani

$$AU_j \approx \frac{1}{2}(AU_{j+1} + AU_j)$$

Niye bu acilim yapıldı? Cunku elimizde U_{j+1} ve U_j degerleri var, bu degerleri tekrar ortaya cikararak bir “denklemler sistemi” yaratmis olacagiz, iki bilinmeyen icin iki formül yanyana gelebilecek ve cozume erisilebilecek.

Ustteki formulu ^{main} denklemindeki AU_j degerleri icinkullanalım ve tekrar duzenleyelim.

$$\frac{\Delta t}{2}AU_{j+1} + \frac{\Delta t}{2}AU_j = U_{i+1} - U_i$$

$$U_{i+1} - \frac{\Delta t}{2}AU_{j+1} = U_i + \frac{\Delta t}{2}AU_j$$

$$(I - \frac{\Delta t}{2}A)U_{j+1} = (I + \frac{\Delta t}{2}A)U_i$$

Artık bu formulu lineer cebirden bilinen $Ax = b$ formuna sokarak cozebiliriz. Forma gore formülün sag tarafı b olur, sol tarafta parantez içi A olacak, U_{j+1} ise bilinmeyen x olacak (bizim x 'ten farklı). Hesapsal kodlar bir dongu icinde, her zaman dilimi icin bilinmeyen U_{j+1} degerini bulacak. Dongunun sonunda yeni U_{j+1} eski U_j olacak ve hesap devam edecek.

Sinir Sartlari

Her iki ucta u 'nun sifir olma sarti uygulamali matematikte Dirichlet sinir sarti olarak biliniyor. Bu sart A matrisinin olusturulmasi sirasinda kendiliginden oluyor. Ufaltilmis bir D2 matrisi üzerinde gosterme gerekirse,

$$\begin{bmatrix} 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \end{bmatrix}$$

degerlerinin her satirinin ^{diff} denklemini temsil ettigini soylemistik. Eger sartlarimizdan biri u_1 ve u_5 'un sifir olmasi ise, carpim sirasinda ona tekabül eden D2'nin en soldaki ve en sagdaki kolonların tamamen sifir yapmamiz yeterli olurdu, cunku carpim sirasinda U_j icinde o kolonlar u_1 ve u_5 ile carpilip

onu sifir yaparlardi. O zaman yeni matris soyle olurdu:

$$\begin{bmatrix} 0 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 0 \end{bmatrix}$$

Bu isler. Alternatif olarak sifir kolon yerine, o kolonlari tamamen matrizen atabilirdik, ayni sekilde u degerlerini uretirken birinci ve sonuncu degerleri de atmamiz gerekirdi, nasil olsa onlar “bilinmeyen” degisken degiller. Bu yeni matris soyle olurdu:

$$\begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix}$$

Alttaki kod icinde $\mathbf{x} = \mathbf{x}[1:-1]$ ibaresi x ve dolayli olarak u ’nun ilk ve son degerlerini atmak icin kullanilmakta.

Kod

” ” ”

This program solves the heat equation

$$u_t = u_{xx}$$

with dirichlet boundary condition

$$u(0, t) = u(1, t) = 0$$

with the Initial Conditions

$$u(x, 0) = 10 * \sin(\pi * x)$$

over the domain $x = [0, 1]$

The program solves the heat equation using a finite difference method where we use a center difference method in space and Crank–Nicolson in time.

” ” ”

```
import scipy as sc
import scipy.sparse as sparse
import scipy.sparse.linalg
import numpy as np
import pylab as pl
import matplotlib.pyplot as plt
import time
```

```

# Number of internal points
N = 200

# Calculate Spatial Step-Size
h = 1/(N+1.0)

# Create Temporal Step-Size, TFinal, Number of Time-Steps
k = h/2
TFinal = 1
NumOfTimeSteps = int(TFinal/k)

# Create grid-points on x axis
x = np.linspace(0,1,N+2)
x = x[1:-1]

# Initial Conditions
u = np.transpose(np.mat(10*np.sin(np.pi*x)))

# Second-Derivative Matrix
data = np.ones((3, N))
data[1] = -2*data[1]
diags = [-1,0,1]
D2 = sparse.spdiags(data, diags, N, N)/(h**2)

# Identity Matrix
I = sparse.identity(N)

# Data for each time-step
data = []

for i in range(NumOfTimeSteps):
    # Solve the System:
    #
    #  $(I - k/2*D2) u_{new} = (I + k/2*D2)*u_{old}$ 
    #
    A = (I - k/2*D2)
    b = (I + k/2*D2)*u

```

```

        u = np.transpose(np.mat(sparse.linalg.spsolve(A, b)))
        data.append(u)

exit()

FPS = 20
MovieLength = 10

def plotFunction( frame ):
    plt.plot(x, data[int(NumOfTimeSteps*frame/(FPS*MovieLength))])
    plt.axis((0,1,0,10.1))

def CreateMovie(plotter, numberOfFrames, fps=10):
    import os, sys
    import matplotlib.pyplot as plt
    plt.ion()
    for i in range(numberOfFrames):
        plotter(i)
        plt.draw()
        time.sleep(0.2)
        plt.hold(False)

# Generate the movie
CreateMovie(plotFunction, int(MovieLength*FPS), FPS)

Ustteki data degiskeni icinde zamana gore degisen tum u vektorleri bulun-
abilir. Bu vektorleri grafik olarak gostermek icin alttaki kod eklenebilir:

import numpy as np
import scipy.linalg

# Number of internal points
N = 200

# Calculate Spatial Step-Size
h = 1/(N+1.0)
k = h/2

```

```

x = np.linspace(0,1,N+2)
x = x[1:-1] # get rid of the '0' and '1' at each end

# Initial Conditions
u = np.transpose(np.mat(10*np.sin(np.pi*x)))

# second derivative matrix
I2 = -2*np.eye(N)
E = np.diag(np.ones((N-1)), k=1)
D2 = (I2 + E + E.T)/(h**2)

I = np.eye(N)
data = []

TFinal = 1
NumOfTimeSteps = int(TFinal/k)

for i in range(NumOfTimeSteps):
    # Solve the System:
    # (I - k/2*D2) u_new = (I + k/2*D2)*u_old
    A = (I - k/2*D2)
    b = np.dot((I + k/2*D2), u)
    u = scipy.linalg.solve(A, b)
    data.append(u)

FPS = 20
MovieLength = 10

import matplotlib.pyplot as plt
import time

def plotFunction( frame ):
    plt.plot(x, data[int(NumOfTimeSteps*frame/(FPS*MovieLength))])
    plt.axis((0,1,0,10.1))

def CreateMovie(plotter, numberOfFrames, fps=10):
    plt.ion()

```

```

    for i in range(numberOfFrames):
        plotter(i)
        plt.draw()
        time.sleep(0.2)
        plt.hold(False)

# Generate the movie
CreateMovie(plotFunction, int(MovieLength*FPS), FPS)

```

Python kodlari zip icinde hfdiff2.py dosyasindadir. Kullanilan matrislerde bol sifir oldugu icin seyrek (sparse) matris teknikleri kullanilabiliyor, sparse metotlari kullanan kodlar ise J. Wiens kodlarindan alinan hfdiff.py icinde bulunabilir.

Kaynaklar

Wiens, J., <http://www.jkwiens.com/2010/01/02/finite-difference-heat-equation-using-numpy>

Stackoverflow, <http://stackoverflow.com/questions/4843034/application-of-boundary-conditions-in-finite-difference-solution-for-the-heat-equ>

Strang, G., Computational Science and Engineering