Contents

1 Finite-difference scheme for the one-dimensional heat equation 2

2 Explicit scheme in time 3
  2.1 No vector notation ......................................................... 3
  2.2 Vector notations ............................................................... 4
  2.3 Matrix notations, using sparse matrices ............................... 5

3 Example from the Lecture Notes 8
  3.0.1 Convergence of the scheme ............................................ 8
  3.0.2 Comparison with the true solution .................................... 10
  3.0.3 Explosive scheme ......................................................... 11
1 Finite-difference scheme for the one-dimensional heat equation

In [3]: from time import time
    # For sparse matrices
    from scipy.sparse import dia_matrix
    from scipy.sparse.linalg.dsolve import spsolve

We consider here the heat equation on $[0, \infty) \times [0, \infty)$:

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

with boundary conditions $u(x, 0) = \sin(2\pi x)$ for all $x \in [0, \infty]$ and $u(0, t) = u(\infty, t) = 0$ for all $t \geq 0$. We apply a finite difference scheme, explicit in time and with central difference in space.
2 Explicit scheme in time

2.1 No vector notation

In [4]: sigma, barX, T = 0.2, 1., 1.
    nx = 100  # Number of grid points
    nt = 5000  # Number of time steps

In [5]: start_time = time()
    dx = barX / (nx - 1)  # Grid step in space
    dt = T / nt  # Grid step in time
    print 'Ratio $\frac{\Delta_T \sigma^2}{\Delta_x^2}$ = ', (sigma * sigma * dt / (dx * dx))

# Boundary conditions
    x = linspace(0.0, barX, nx)
    u = sin(2.0 * pi * x)

for n in range(0, nt):
    for j in range(1, nx - 1):
        u[j] += dt * (0.5 * sigma * sigma) * 
        (u[j - 1] - 2 * u[j] + u[j + 1]) / (dx ** 2)

    # Plot every pp time steps
    pp = 100
    if (n % pp == 0):
        if (n % (10 * pp) == 0):
            plotlabel = "t = %.1f" % (n * dt)
            plot(x, u, label=plotlabel, color=get_cmap('copper')(float(n) / nt))

xlabel(u'$x$', fontsize=20)
title(u'One-dimensional heat equation')
legend()
show()
    temp = time() - start_time
    print("--- Computation time: %s seconds ---" % temp)

Ratio $\frac{\Delta_T \sigma^2}{\Delta_x^2} = 0.078408$
2.2 Vector notations

In [6]: start_time = time()

\[ dx = \frac{\text{barX}}{\text{nx} - 1} \]  # Grid step in space
\[ dt = \frac{T}{nt} \]  # Grid step in time

# Boundary conditions
\[ x = \text{linspace}(0.0, \text{barX}, \text{nx}) \]
\[ u = \sin(2.0 \ast \pi \ast x) \]
\[ \text{rhs} = \text{zeros}(\text{nx}) \]

for n in range(0, nt):
    \[ \text{rhs}[1:-1] = dt \ast \left(0.5 \ast \sigma \ast \sigma\right) \ast \left(u[2:] - 2.0 \ast u[1:-1] + u[2:]) \right) / (dx \ast dx) \]
    u += rhs

# Plot every pp time steps
\[ pp = 100 \]
if (n % pp == 0):
if (n % (10 * pp) == 0):
    plotlabel = "t = %1.2f" % (n * dt)
    plot(x, u, label=plotlabel, color=get_cmap('copper')(float(n) / nt))

xlabel(u'$$x$$', fontsize=14)
title(u'One-dimensional heat equation')
legend()
show()
temp = time() - start_time
print("--- Computation time: %s seconds ---" % temp)

--- Computation time: 0.532827854156 seconds ---

2.3 Matrix notations, using sparse matrices

We are interested here in solving the matrix system in $\mathbb{R}^N$:

$$\operatorname{Diag}(-2, 1, 1) u = (dx)^2 (1, \ldots, 1)'$$.

In [8]: N = 1000
dx = 1. / (N - 1)  # Space step size

--- Computation time: 0.532827854156 seconds ---
x = linspace(0.0, 1.0, N)

# Definition of the tridiagonal matrix
Tmatrix = [ones(N), -2.0 * ones(N), ones(N)]
nonzeropositions = array([-1, 0, 1])
iterationMatrix = dia_matrix((Tmatrix, nonzeropositions), shape=(N, N))

# Schematic representation of the diagonal matrix
figure()
spy(iterationMatrix)
title('Tridiagonal matrix')
draw()

rhs = -ones(N) * dx * dx  # Right-hand side

# Solving the linear system
spSolution = spsolve(iterationMatrix, rhs)
comptime = time() - t
print("Computation time using sparse library: %s seconds" % comptime)

# In order to compare with the full resolution
fullIterMatrix = iterationMatrix.todense()
t = time()
linAlgSolution = linalg.solve(fullIterMatrix, rhs)
comptime = time() - t
print("Computation time using standard linear algebra: %s seconds" % comptime)

# Solution plot
figure()
plot(x, spSolution, 'k-')
title('Solution of the matrix equation')
xlabel(u'$x$')
ylabel(u'$u$')
show()

Computation time using sparse library: 0.0011830329895 seconds
Computation time using standard linear algebra: 0.0667409896851 seconds
3 Example from the Lecture Notes

We consider here the example in the lecture notes, namely the heat equation $\partial_\tau u(\tau, x) = \frac{\sigma^2}{2} \partial_{xx} u(\tau, x)$ on $[0, \infty) \times [0, 1]$ with boundary condition $u(0, x) = 2x1_{x \in [0, \frac{1}{2}]} + 2(1 - x)1_{x \in [\frac{1}{2}, 1]}$.

We first rewrite the explicit scheme as a function, taking the boundary condition as argument.

As seen in the lecture notes, the CFL condition, ensuring convergence of the scheme, is $cfl \leq 1$, where $cfl := \delta_t / \delta_x^2$.

In [9]: def explicitSchemeHeatEquation(sigma, barX, T, m, n, BC, *extraArguments):
    dx = barX / (m - 1)  # Grid step in space
    dt = T / n            # Grid step in time

    # Boundary conditions
    xx = linspace(0., barX, m)
    uu = BC(xx)
    rhs = zeros(m)
    for l in range(0, n):
        rhs[1:-1] = dt * (0.5 * sigma * sigma) * \
                     (uu[:-2] - 2.0 * uu[1:-1] + uu[2:]) / (dx * dx)
        uu += rhs
    return xx, uu

In [10]: # Boundary conditions in the example from the lecture note

    def boundaryConditionf(zz):
        uu = []
        for z in zz:
            if z < 0.5:
                uu.append(2. * z)
            else:
                uu.append(2. * (1. - z))
        return np.asarray(uu)

3.0.1 Convergence of the scheme

In [11]: sigma, barX, T = sqrt(2.), 1., 0.001
    nx = 100    # Number of grid points in space
    nt = 10000  # Number of time steps
In [12]: TT = arange(0.01, 0.5, 0.05)
   plt.figure(figsize=(8, 5))
   
   for T in TT:
       xx, uu = explicitSchemeHeatEquation(
           sigma, barX, T, nx, nt, boundaryConditionf)
       plt.plot(
           xx, uu, color=get_cmap('afmhot')(T), label="t = %1.2f" % T)
       plt.legend()
   
   dx = barX / (nx - 1)
   dt = T / nt
   cfl = sigma * sigma * dt / (dx * dx)
   
   plt.plot(xx, boundaryConditionf(xx))
   
   plt.title("Solutions of the heat equation. CFL= %s " % cfl)
   plt.xlabel(u'x', fontsize=12)
   plt.show()
3.0.2 Comparison with the true solution

The true solution has the explicit form:

\[ u(\tau, x) = \frac{8}{\pi^2} \sum_{n \geq 1} \frac{\sin(n\pi x)}{n^2} \sin\left(\frac{n\pi}{2}\right) e^{-n^2\pi^2 \tau}. \]

In [13]: def TrueSolution(x, tau, nMax):
    
    temp = 0.
    for n in range(1, nMax + 1):
        temp = temp + sin(n * pi * x) * sin(0.5 * n * pi) * exp(-n * n * pi * pi * tau) / (n * n)
    return 8. * temp / (pi * pi)

In [14]: nx = 100  # Number of grid points in space
    nt = 10000  # Number of time steps

    plt.figure(figsize=(8, 5))
    tau = 0.1
    xx, uu = explicitSchemeHeatEquation(
        sigma, barX, tau, nx, nt, boundaryConditionf)
    trueSols = [TrueSolution(x, tau, 20) for x in xx]
    plt.plot(xx, uu, 'r+', markersize=10, label="FD solution")
    plt.plot(xx, trueSols, 'ro', mfc='none',
             markersize=10, label="True solution")
    plt.legend()
    plt.title("Solutions of the heat equation. \tau = %.2f\) \( \% \tau")
    plt.xlabel("x", fontsize=12)
    plt.show()
3.0.3 Explosive scheme

In [19]: sigma, barX, T = sqrt(2.), 1., 0.001

nx = 100  # Number of grid points in space
nt = 100  # Number of time steps

In [20]: TT = arange(0.1, 0.5, 0.1)
plt.figure(figsize=(8, 5))
for T in TT:
    xx, uu = explicitSchemeHeatEquation(
        sigma, barX, T, nx, nt, boundaryConditionf)
    plt.plot(
        xx, uu, color=get_cmap('afmhot')(T), label="t = %1.2f" % T)
plt.legend()

dx = barX / (nx - 1)
dt = T / nt
cfl = sigma * sigma * dt / (dx * dx)
plt.title("Solutions of the heat equation. CFL= %s " % cfl)
plt.show()