Project 4, FYS 3150 / 4150, fall 2013

Odd Petter Sand and Nathalie Bonatout

November 2, 2013

All our source code can be found at our GitHub repository for this project: https://github.com/NathalieB/Project4/

1 Introduction

x

2 Theory and Technicalities

2.1 Closed form solution

We subtitute

$$v(x,t) = u(x,t) - u_s(x) = u(x,t) + x - 1$$

where $u_s(x) = 1 - x$ is the steady state solution that satisfies our boundary and initial conditions. We then set up the diffusion equation for $v(x,t)$:

$$\frac{\partial^2 v(x,t)}{\partial x^2} = \frac{\partial v(x,t)}{\partial t}$$

with known boundary conditions

$$v(0,t) = v(1,t) = 0 \qquad t \geqslant 0.$$

The initial condition $u(x,0) = 0$ then becomes

$$v(x,0) = u(x,0) - u_s(x) = 0 - (1-x) = x - 1 \qquad 0 < x < 1.$$

The solution of this equation is known from pp. 313-314 in the lecture notes, using $L = 1$:

$$v(x,t) = \sum_{n=1}^{\infty} A_n \sin(n\pi x) e^{-n^2\pi^2 t}.$$

We now find the Fourier series coefficients by partwise integration

$$A_n = 2\int_0^1 v(x,0)\sin(n\pi x)dx = 2\int_0^1 (x-1)\sin(n\pi x)dx$$

$$= 2\left(\left[-(x-1)\frac{1}{n\pi}\cos(n\pi x)\right]_0^1 - \int_0^1 -\frac{1}{n\pi}\cos(n\pi x)dx\right)$$

$$= \frac{2}{n\pi}\left([(1-x)\cos(n\pi x)]_0^1 + \left[\frac{1}{n\pi}\sin(n\pi x)\right]_0^1\right)$$

$$= \frac{2}{n\pi}(1+0) = \frac{2}{n\pi}$$

and finally, by substitution, our closed form solution is

$$u(x,t) = v(x,t) + u_s(x) = 1 - x + \sum_{n=1}^{\infty} \frac{2}{n\pi}\sin(n\pi x)e^{-n^2\pi^2 t}.$$

2.2 Algorithms

2.2.1 Explicit scheme

```
input: nSteps (# of interior points), time

deltaX = 1.0 / (nSteps + 1)
alpha = 0.5
deltaT = alpha * deltaX ^ 2
tSteps = 1.0 / deltaT

define v, vNext
```

```
for( i = 0; i < nSteps)
        x = (i + 1) * deltaX
        v[i] = v(x, 0)

for( t = 1; t <= tSteps)
        for( i = 1 --> nSteps )
                vNext[i] = (1 - 2 * alpha) * v[i]
                if( i > 0 ) : vNext[i] += alpha * v[i-1] // else += 0
                if( i < nSteps ) : vNext[i] += alpha * v[i+1] // else += 0
        v = vNext

for( i = 0; i < nSteps)
        x = (i + 1) * deltaX
        u[i] = 1 - x + v[i]

output: u
```

2.2.2 Implicit scheme

```
input: nSteps (# of interior points), time, tSteps

deltaX = 1.0 / (nSteps + 1)
deltaT = 1.0 / tSteps
alpha = deltaT / (deltaX ^ 2)

define v, vNext

for( i = 0; i < nSteps)
        x = (i + 1) * deltaX
        v[i] = v(x, 0)

a = -alpha          // diagonal element
b = 1 + 2*alpha     // off-diagonal element

for( t = 1; t <= tSteps)
        tridiagonalSolver(a, b, v, vNext)
        v = vNext

for( i = 0; i < nSteps)
        x = (i + 1) * deltaX
        u[i] = 1 - x + v[i]

output: u
```

2.2.3 Crank-Nicholson scheme

```
input: nSteps (# of interior points), time, tSteps

deltaX = 1.0 / (nSteps + 1)
deltaT = 1.0 / tSteps
alpha = deltaT / (deltaX ^ 2)

define v, w

for( i = 0; i < nSteps)
        x = (i + 1) * deltaX
        v[i] = v(x, 0)

a = 2 * (1 + alpha)     // diagonal element
b = -alpha              // off-diagonal element

for( t = 1; t <= tSteps)
        for( i = 0; i < nSteps )
                w[i] = 2 * (1 - alpha) * v[i]
                if( i > 0 ) : w[i] += alpha * v[i - 1] // else += 0
                if( i < nSteps - 1 ) : w[i] += alpha * v[i + 1] // else += 0
        tridiagonalSolver(a, b, w, v)

for( i = 0; i < nSteps)
        x = (i + 1) * deltaX
        u[i] = 1 - x + v[i]

output: u
```

2.3 Tridiagonal form of the implicit schemes

The methods to reformulate the problem into a tridiagonal matrix equation is described in great detail in pp. 308-312 in the lecture notes, so we will not reproduce them here.

In the case of the Crank-Nicholson sceme, it is easily seen by the definition of matrix addition that the sum of a diagonal matrix and a tridiagonal matrix $(2\mathbf{I} + \alpha\mathbf{B})$ is also tridiagonal. We can then note that the known vector $\mathbf{w}_{j-1} \equiv (2\mathbf{I} - \alpha\mathbf{B})\mathbf{v}_{j-1}$is easily calculated in every step by

$$w_i = 2(1 - \alpha)v_i + \alpha(v_{i-1} + v_{i+1})$$

hence there is no reason to calculate the inverse matrix $(2\mathbf{I} - \alpha\mathbf{B})^{-1}$ or demand that it be tridiagonal. The operation above is $\mathcal{O}(n)$, just as the tridiagonal solver itself, so this does not affect the scaling of the algorithm.

2.4 Truncation errors and stability

Simply use the table from p. 350 in the slides.

If time: Do the Taylor expand for explicit scheme and find the actual error, not the order: p. 346-347 in the slides. (Can use just the result in the C-N case.)

For explanations of the stability criteria, we again refer to the lecture notes (p. 307-309, 312).

3 Results and analysis

3.1 Explicit scheme

x

3.2 Implicit scheme

x

3.3 Cranck-Nicholson scheme

x

4 Conclusion

What we learned.

4.1 Critique

x