Lab8_sol

October 23, 2017

1 Sample solutions for exercises of Lab 8

Let us consider the following problem: find u such that

$$\frac{\partial u}{\partial t}(x,t) = \frac{1}{4}\frac{\partial^2 u}{\partial x^2}(x,t),\ x \in [-1,1],\ t \in (0,0.5] \tag{1}$$

$$u(-1,t) = 1, u(1,t) = 0,\ t \in (0,0.5] \tag{2}$$

$$u(x,0) = u_0(x),\ x \in [-1,1] \tag{3}$$

where u_0 is a given function

After following the steps of deriving the Explicit finite difference method for this problem, we get

$$U_{i0} = u_0(x_i),\ i = 0,\dots,n \tag{4}$$

$$U_{0k} = 1,\ k = 1,\dots,m \tag{5}$$

$$U_{nk} = 0,\ k = 1,\dots,m \tag{6}$$

$$U_{i,k+1} = \frac{1}{4}\frac{\Delta t}{\Delta x^2}U_{i-1,k} + (1 - \frac{1}{4}\frac{2\Delta t}{\Delta x^2})U_{ik} + \frac{1}{4}\frac{\Delta t}{\Delta x^2}U_{i+1,k}, k = 0,\dots,m-1,\ i = 1,\dots,n-1 \tag{7}$$

1.1 Exercise 1

Write a function that for given values of m and n and for given function u_0 returns the values U_{im}, $i = 0,\dots,n$ of the approximate solution obtained by explicit finite difference method. Test the correctness of your function in the case $m = 100, n = 10$ and $u_0(x) = \sin(\pi x) + \frac{1-x}{2}$, when the exact solution is $u(x,t) = e^{-\pi^2 t/4}\sin(\pi x) + \frac{1-x}{2}$.

Solution

```
In [1]: import numpy as np
        def lab8solver(m,n,u0):
            delta_t=0.5/m
            delta_x=2/n #(1-(-1))/n
            #define values of x_i
            x=np.linspace(-1,1,n+1)
            #define matrix U with dimension (n+1)x(m+1)
            U=np.zeros(shape=(n+1,m+1))
            #fill in the initial condition
```

```
        U[:,0]=u0(x)
        #use boundary conditions
        k=np.arange(1,m+1)
        U[0,k]=1
        U[n,k]=0
        #the coefficients a,b,c in the formula for U[i,k+1]
        #do not depend on time, so compute before starting a time cycle
        a=1/4*delta_t/delta_x**2
        b=1-2*a
        c=a
        #compute all other values
        i=np.arange(1,n)
        for k in range(0,m):
            U[i,k+1]=a*U[i-1,k]+b*U[i,k]+c*U[i+1,k]
        return U[:,m]
    #test the function with given data
    def u0(x):
        return np.sin(np.pi*x)+(1-x)/2
    m=100
    n=10
    approx_sol=lab8solver(m,n,u0)
    def u(x,t):
        return np.exp(-np.pi**2*t/4)*np.sin(np.pi*x)+(1-x)/2
    #compare answers with values of exact solution
    x=np.linspace(-1,1,n+1)
    print(np.max(np.abs(approx_sol-u(x,0.5))))
```

```
0.00925655857449
```

1.2 Exercise 2

The total error caused by replacing exact derivatives with finite difference approximations is $O(\Delta t + \Delta x^2)$, which usually implies that the error of the approximate solution is of the same order. This means, that if we increase m four times and n two times, then the total error should be reduced approximately four times. Verify the convergence rate by computing the errors in the settings of the previous exercise for $m = 4, 16, 64, 256$ and $n = 2, 4, 8, 16$.

In [2]:
```
repetitions=4
errors=np.zeros(repetitions)
for i in range(1,repetitions+1):
    n=2**i
    m=4**i
    approx_sol=lab8solver(m,n,u0)
    x=np.linspace(-1,1,n+1)
    errors[i-1]=np.max(np.abs(approx_sol-u(x,0.5)))
print(errors)
print(errors[:-1]/errors[1:])
```

```
[  3.56632987e-17   6.48611972e-02   1.53203711e-02   3.77115439e-03]
[  5.49840277e-16   4.23365705e+00   4.06251496e+00]
```

We see that the for the first two computations the relation does not hold (the second error is much larger than the first one), but starting from the third computation the errors start to be reduced approximately 4 times

1.3 Exercise 3

It turns out that explicit methods may be unstable for certain choices of parameters m and n. This means, that if m and n do not satisfy certain condition, the approximate solution may have arbitrarily large errors even when we let m and n to go to infinity. The sufficient condition of stability is that the coefficients a, b and c are all nonnegative. Repeat the computations of the previous exercise for $m = 2, 8, 32, 128$ and $n = 10, 20, 40, 80$ and compute the errors.

```
In [3]: repetitions=4
        errors=np.zeros(repetitions)
        for i in range(1,repetitions+1):
            n=2**i*5
            m=4**i//2
            approx_sol=lab8solver(m,n,u0)
            x=np.linspace(-1,1,n+1)
            errors[i-1]=np.max(np.abs(approx_sol-u(x,0.5)))
        print(errors)
        print(errors[:-1]/errors[1:])
```

```
[  1.22363261e-01   2.61927242e-02   1.88914222e+06   1.76749584e+75]
[  4.67165081e+00   1.38648768e-08   1.06882414e-69]
```

Errors are actually increasing when m and n increase. Let us see what are the values of a,b,c for each computation

```
In [4]: repetitions=4
        errors=np.zeros(repetitions)
        for i in range(1,repetitions+1):
            n=2**i*5
            m=4**i//2
            delta_t=0.5/m
            delta_x=2/n
            a=1/4*delta_t/delta_x**2
            b=1-2*a
            c=a
            print("i =",i,", a =",a,", b =",b,", c =",c)
```

```
i = 1 , a = 1.5624999999999998 , b = -2.1249999999999996 , c = 1.5624999999999998
i = 2 , a = 1.5624999999999998 , b = -2.1249999999999996 , c = 1.5624999999999998
i = 3 , a = 1.5624999999999998 , b = -2.1249999999999996 , c = 1.5624999999999998
i = 4 , a = 1.5624999999999998 , b = -2.1249999999999996 , c = 1.5624999999999998
```

As we see, the coefficient b is negative and the stability condition is not satisfied in this case.