Leapfrog scheme

Nina Kristine Kylstad, Ingeborg Sauge Torpe

September 20, 2012

Exercise 17

We want to analyze the Leapfrog scheme by looking at the exact solution of the discrete equation. We consider the case where a is constant and $b = 0$, giving

$$u'(t) = -au(t)\,,\ u(0) = I \tag{1}$$

where I is some initial condition. We assume that the exact solution of the discrete equations is on the form

$$u^n = A^n \tag{2}$$

The leapfrog scheme for (1) can be written as

$$u^{n+1} = u^{n-1} - 2a\Delta t u^n \tag{3}$$

We insert (2) into (3):

$$\begin{aligned}
A^{n+1} &= A^{n-1} - 2a\Delta t A^n \\
\Rightarrow A^2 &= (2a\Delta t)A - 1 = 0 \\
\Rightarrow A &= \frac{-2a\Delta t \pm \sqrt{(2a\Delta t)^2 - 4 \cdot 1 \cdot -1}}{2} \\
&= -a\Delta t \pm \sqrt{(a\Delta t)^2 + 1}
\end{aligned}$$

We can see that the governing polynomial for A has two roots, A_1 and A_2. This means that A^n is a linear combination of A_1 and A_2,

$$A^n = C_1 A_1^n + C_2 A_2^n \tag{4}$$

where C_1 and C_2 are constants to be determined. The root A_1 is negative, and can therefore cause oscillations.

To find the constants C_1 and C_2, we use the initial condition I, and the value we obtain for u^1 by using the Forward Euler scheme:

$$u^1 = u^0 - \Delta t a u^0 = I(1 - \Delta t a)$$

To simplify, we let $x = \Delta t a$. The equation for A^n then becomes:

$$A^n = C_1(-x - \sqrt{x^2+1})^n + C_2(-x + \sqrt{x^2+1})^n \tag{5}$$

We can now find C_1 and C_2.

$$\begin{aligned}
& A^0 = C_1 + C_2 = I \\
\Rightarrow\ & C_1 = I - C_2 \\
& A^1 = C_1(-x - \sqrt{x^2+1}) + C_2(-x + \sqrt{x^2+1}) = I(1 - x) \\
\Rightarrow\ & C_2 = \frac{I(1 + \sqrt{x^2+1})}{2\sqrt{x^2+1}}
\end{aligned}$$

To test how the roots A_1 and A_2 affect the numerical solution, we find the values of $C_1 A_1^n$ and $C_2 A_2^n$ for increasing values of n. This is done in the program `dc_leapfrog_analysis.py` A sample of the output is as follows:

```
1x-193-157-247-37:Downloads ninakylstad$ python dc_leapfrog_analysis.py
n = 0
0.029289321903         0.070710678097         0.100000000000
n = 1
-0.029583679552         0.070007106761         0.099004983375
n = 2
0.029880995494         0.069310535961         0.098019867331
n = 3
-0.030181299462         0.068620896042         0.097044553355
n = 4
0.030484621484         0.067938118040         0.096078943915
n = 5
-0.030790991892         0.067262133681         0.095122942450
n = 6
0.031100441322         0.066592875367         0.094176453358
n = 7
-0.031413000718         0.065930276174         0.093239381991
n = 8
0.031728701336         0.065274269843         0.092311634639
n = 9
```

```
-0.032047574745         0.064624790777         0.091393118527
n = 10
0.032369652831         0.063981774028         0.090483741804
. . .
. . .
. . .
. . .
n = 391
-1.461411232877         0.001417169765         0.002004050106
n = 392
1.476098413941         0.001403068924         0.001984109474
n = 393
-1.490933201156         0.001389108386         0.001964367255
n = 394
1.505917077964         0.001375286756         0.001944821475
n = 395
-1.521051542715         0.001361602651         0.001925470178
n = 396
1.536338108818         0.001348054703         0.001906311429
n = 397
-1.551778304892         0.001334641557         0.001887343314
n = 398
1.567373674916         0.001321361872         0.001868563934
n = 399
-1.583125778390         0.001308214319         0.001849971412
n = 400
1.599036190484         0.001295197585         0.001831563889
```

As we can see from the output, the root A_1 begins oscillating right from the beginning. For low values of n, we see that $C_1 A_1^n$ is quite small, and therefore does not affect the solution much. However, it becomes larger as n becomes larger, and as we can see from the last 10 values of n shown here, $C_1 A_1^n$ becomes considerably larger than both $C_2 A_2^n$, and the exact analytical solution. Because of this, the numerical solution oscillates more and more with larger n.