Amath 586: Homework 2

Due on April 16, 2015

Brian de Silva

Problem 1

The proof of convergence of 1-step methods in Section 6.3 shows that the global error goes to zero as $k \to 0$. However, this bound may be totally useless in estimating the actual error for a practical calculation.
For example, suppose we solve $u' = -10u$ with $u(0) = 1$ up to time $T = 10$, so the true solution is $u(T) = e^{-100} \approx 3.7 \times 10^{-44}$. Using forward Euler with a time step $k = 0.01$, the computed solution is $U^N = (.9)^{1000} \approx 1.75 \times 10^{-46}$, and so $E^N \approx 3.7 \times 10^{-44}$. Since $L = 10$ for this problem, the error bound (6.16) gives

$$\|E^N\| \le e^{100} \cdot 10 \cdot \|\tau\|_\infty \approx 2.7 \times 10^{44} \|\tau\|_\infty. \tag{1}$$

Here $\|\tau\|_\infty = |\tau^0| \approx 50k$, so this upper bound on the error does go to zero as $k \to 0$, but obviously it is not a realistic estimate of the error. It is too large by a factor of about 10^{88}.
The problem is that the estimate (6.16) is based on the Lipschitz constant $L = |\lambda|$, which gives a bound that grows exponentially in time even when the true and computed solutions are decaying exponentially.

(a) Determine the computed solution and error bound (6.16) for the problem $u' = 10u$ with $u(0) = 1$ up to time $T = 10$. *Correction: Ignore the following statement, since it's not correct...* Note that the error bound is the same as in the case above, but now it is a reasonable estimate of the actual error.

(b) A more realistic error bound for the case where $\lambda < 0$ can be obtained by rewriting (6.17) as

$$U^{n+1} = \Phi(U^n)$$

and then determining the Lipschitz constant for the function Φ. Call this constant M. Prove that if $M \le 1$ and $E^0 = 0$ then

$$|E^n| \le T\|\tau\|_\infty$$

for $nk \le T$, a bound that is similar to (6.16) but without the exponential term.

(c) Show that for forward Euler applied to $u' = \lambda u$ we can take $M = |1 + k\lambda|$. Determine M for the case $\lambda = -10$ and $k = 0.01$ and use this in the bound from part (b). Note that this is much better than the bound (1). *But note that it's still not a very sharp bound.*

Solution:

(a) We have

$$\tau^n = \frac{1}{2} k u''(t_n) = \frac{1}{2} k (100u) = 50 k e^{10 t_n}$$

so

$$\|\tau\|_\infty = \frac{1}{2} e^{100}.$$

The solution computed by Forward Euler will be

$$\begin{aligned} U^{n+1} &= U^n + k(10U^n) \\ &= (1 + 10k)U^n \\ &= 1.1 U^n \\ \Rightarrow U^n &= (1.1)^n. \end{aligned}$$

For $n = 1000$ this gives $U^n = (1.1)^{1000} \approx 2.47 \times 10^{41}$. At $t = 10$, the exact solution is $u(10) = e^{100} \approx 2.69 \times 10^{43}$ which implies the absolute error in our approximation is $\approx 2.66 \times 10^{43}$. Since the Lipschitz constant for this problem is $L = |\lambda| = 10$ the error bound (1) gives

$$|E^n| \le e^{L t_n} \cdot 10 \cdot \|\tau\|_\infty = e^{100} \cdot 5 \cdot e^{100} \approx 3.6 \times 10^{87}.$$

Problem 1 continued on next page...

(b) Suppose $M \leq 1$ is a Lipschitz constant for Φ and $E^0 = 0$. Then since

$$U^{n+1} = \Phi(U^n)$$

and

$$u(t_{n+1}) = \Phi(u(t_n)) + k\tau^n,$$

we have

$$\begin{aligned}
E^{n+1} &= \Phi(U^n) - \Phi(u(t_n)) - k\tau^n \\
\Rightarrow |E^{n+1}| &\leq |\Phi(U^n) - \Phi(u(t_n))| + |k\tau^n| \\
&\leq M|E^n| + |k\tau^n| \\
&= M|\Phi(U^{n-1}) - \Phi(u(t_{n-1})) - k\tau^{n-1}| + k|\tau^n| \\
&\vdots \\
\Rightarrow |E^n| &\leq M^n|E^0| + k\sum_{m=1}^{n} M^{n-m}|\tau^{m-1}| \\
&= k\sum_{m=1}^{n} M^{n-m}|\tau^{m-1}| \ \leq k\sum_{m=1}^{n} |\tau^{m-1}| \\
&\leq k\sum_{m=1}^{n} \|\tau\|_\infty \ \leq nk\|\tau\|_\infty \\
&\leq T\|\tau\|_\infty
\end{aligned}$$

for $nk \leq T$.

(c) The Forward Euler method for this problem is given by

$$U^{n+1} = U^n + k\Psi(U^n, t_n, k)$$

where $\Psi(U^n, t_n, k) = f(U^n) = \lambda U^n$. Hence

$$\Phi(U^n, t_n, k) = U^n + k\lambda U^n = U^n(1 + k\lambda).$$

Then for any U^* we have

$$\begin{aligned}
|\Phi(U^n, t_n, k) - \Phi(U^*, t_n, k)| &= |(1 + k\lambda)U^n - (1 + k\lambda)U^*| \\
&\leq |1 + k\lambda||U^n - U^*|
\end{aligned}$$

so we can take $M = |1 + k\lambda|$ as a Lipschitz constant for Φ. When $\lambda = -10$ and $k = 0.01$, $M = |1 - 0.1| = 0.9$ Using the error bound from (b) (and observing that $E^0 = 0$ and $M < 1$), this tells us that

$$|E^n| \leq T\|\tau\|_\infty = 10 \cdot \frac{1}{2}e^{100} = 5e^{100} \approx 1.34 \times 10^{44}.$$

Problem 2

Which of the following Linear Multistep Methods are convergent? For the ones that are not, are they inconsistent, or not zero-stable, or both?

(a) $U^{n+2} = \frac{1}{2}U^{n+1} + \frac{1}{2}U^n + 2kf(U^{n+1})$

(b) $U^{n+1} = U^n$

(c) $U^{n+4} = U^n + \frac{4}{3}k(f(U^{n+3}) + f(U^{n+2}) + f(U^{n+1}))$

(d) $U^{n+3} = -U^{n+2} + U^{n+1} + U^n + 2k(f(U^{n+2}) + f(U^{n+1}))$.

Solution:
A linear multistep method is zero-stable if its characteristic polynomial $\rho(\zeta)$ satisfies the root condition, is consistent if and only if $\rho(1) = 0$ and $\rho'(1) = \sigma(1)$, where $\sigma(\zeta)$ is the other characteristic polynomial of the method, and is convergent if and only if it is both zero-stable and consistent.

(a) For this method

$$\rho(\zeta) = \zeta^2 - \frac{1}{2}\zeta - \frac{1}{2} = (\zeta - 1)(\zeta + \frac{1}{2}),$$
$$\sigma(\zeta) = 2\zeta.$$

Then $\rho(1) = 0$ and $\rho'(1) = 2(1) - \frac{1}{2} = \frac{3}{2} \neq 2 = \sigma(1)$, so the method is inconsistent. However, the roots of ρ are $-\frac{1}{2}$ and 1 so it is zero-stable. Overall the method is not convergent since it is inconsistent.

(b) Here we have

$$\rho(\zeta) = \zeta - 1$$
$$\sigma(\zeta) = 0.$$

Hence $\rho(1) = 0$, but $\rho'(1) = 1 \neq 0 = \sigma(1)$ so the method is inconsistent. The only root of ρ is 1 so the root condition is satisfied and the method is zero-stable. In conclusion the method is not convergent since it is inconsistent.

(c) In this case

$$\rho(\zeta) = \zeta^4 - 1 = (\zeta^2 - 1)(\zeta^2 + 1),$$
$$\sigma(\zeta) = \frac{4}{3}(\zeta^3 + \zeta^2 + \zeta).$$

$\rho(1) = 0$ and $\rho'(1) = 4(1)^3 = 3 \cdot \frac{4}{3} = \sigma(1)$ so the method is consistent. The roots of ρ, $\pm 1, \pm i$ satisfy the root condition, so the method is stable is well. Therefore this linear multistep method is convergent.

(d) This method has

$$\rho(\zeta) = \zeta^3 + \zeta^2 - \zeta - 1 = (\zeta^2 - 1)(\zeta + 1),$$
$$\sigma(\zeta) = 2\zeta^2 + 2\zeta.$$

We have $\rho(1) = 0$ and $\rho'(1) = 3(1)^2 + 2(1) - 1 = 4 = 2 + 2 = \sigma(1)$ so this method is consistent. However it is not zero-stable as 1 is a double root which lies on the unit circle. Hence it is not convergent.

Problem 3

(a) Determine the general solution to the linear difference equation $2U^{n+3} - 5U^{n+2} + 4U^{n+1} - U^n = 0$.

Hint: One root of the characteristic polynomial is at $\zeta = 1$.

(b) Determine the solution to this difference equation with the starting values $U^0 = 11$, $U^1 = 5$, and $U^2 = 1$. What is U^{10}?

(c) Consider the LMM

$$2U^{n+3} - 5U^{n+2} + 4U^{n+1} - U^n = k(\beta_0 f(U^n) + \beta_1 f(U^{n+1})).$$

For what values of β_0 and β_1 is local truncation error $\mathcal{O}(k^2)$?

(d) Suppose you use the values of β_0 and β_1 just determined in this LMM. Is this a convergent method?

Solution:

(a) This linear difference equation has the following characteristic polynomial,

$$\rho(\zeta) = 2\zeta^3 - 5\zeta^2 + 4\zeta - 1 = (\zeta - 1)(2\zeta^2 - 3\zeta + 1) = (\zeta - 1)^2(2\zeta - 1).$$

We have a repeated root so the general solution is of the form

$$U^n = c_1 + nc_2 + (\frac{1}{2})^n c_3. \tag{2}$$

(b) The initial data $U^0 = 11$, $U^1 = 5$, and $U^2 = 1$ gives us a system of equations which we may solve to determine $c_1,\ c_2$, and c_3. Substituting the known values for $U^0,\ U^1$, and U^2 into (2), we get

$$\begin{aligned} c_1 + c_2 \qquad\qquad &+c_3 = 11 \\ c_1 + c_2 \qquad\qquad &+\frac{1}{2}c_3 = 5 \\ c_1 + 2c_2 \qquad\qquad &+\frac{1}{4}c_3 = 1 \end{aligned}$$

$$\Rightarrow c_1 = 3, c_2 = -2, c_3 = 8.$$

Thus the particular solution is given by

$$U^n = 3 - 2n + 8(\frac{1}{2})^n = 3 - 2n + 2^{3-n}$$

$$\Rightarrow U^10 = 3 - 2(10) + 2^{-7} = \frac{1}{128} - 17.$$

Problem 3 continued on next page...

(c) In order to construct a method which has $O(k^2)$ local truncation error, we need to force the $O(1)$ and $O(k)$ terms of the local truncation error to vanish (the expansion is given in Section (5.9.1)). Note, the $O(k^{-1})$ them is determined by the coefficients on the U^j terms. Since $\rho(1) = 2 - 5 + 4 - 1 = 0$ this term is already taken care of. The $O(1)$ term is

$$u'(t_n) \sum_{j=0}^{3} (j\alpha_j - \beta_j),$$

where α_j is the coefficient on U^{n+j}. Forcing this term to vanish is equivalent to enforcing

$$\beta_0 + \beta_1 = 4 - (2 \cdot 5) + 6 = 0.$$

The $O(k)$ term is

$$u''(t_n) \sum_{j=0}^{3} (\frac{1}{2} j^2 \alpha_j - j\beta_j).$$

Setting this equal to 0 gives the condition

$$\beta_1 = 1.$$

Solving these two equations yields $\beta_0 = -1,\ \beta_1 = 1$;

(d) In (a) we saw $\rho(\zeta) = (\zeta - 1)^2(2\zeta - 1)$ which has a double root of 1, which lies on the unit circle. The root condition is not satisfied so the method is not zero-stable. This can also be seen from the general solution derived in (b). Thus the method is not convergent (although it is consistent).

Problem 4

Consider the midpoint method $U^{n+1} = U^{n-1} + 2kf(U^n)$ applied to the test problem $u' = \lambda u$. The method is zero-stable and second order accurate, and hence convergent. If $\lambda < 0$ then the true solution is exponentially decaying.
On the other hand, for $\lambda < 0$ and $k > 0$ the point $z = k\lambda$ is never in the region of absolute stability of this method (see Example 7.7), and hence it seems that the numerical solution should be growing exponentially for any nonzero time step. (And yet it converges to a function that is exponentially decaying.)

(a) Suppose we take $U^0 = \eta$, use Forward Euler to generate U^1, and then use the midpoint method for $n = 2,\ 3,\ \ldots$. Work out the exact solution U^n by solving the linear difference equation and explain how the apparent paradox described above is resolved.

(b) Devise some numerical experiments to illustrate the resolution of the paradox.

Solution:

(a) When applied to the test problem $u' = \lambda u$ the midpoint method can be written

$$U^{n+1} = U^{n-1} + 2k\lambda U^n \Rightarrow U^{n+2} - 2k\lambda U^{n+1} - U^n = 0.$$

The characteristic polynomial is then

$$\rho(\zeta) = \zeta^2 - 2k\lambda\zeta - 1$$

Problem 4 continued on next page...

which has the following roots

$$\zeta_{1,2} = k\lambda \pm \sqrt{(k\lambda)^2 + 1}.$$

If we use one iteration of Forward Euler to obtain U^1 with an (exact) initial guess of $U^0 = \eta$, then

$$U^1 = \eta + k\lambda\eta = \eta(1 + k\lambda).$$

The general solution of the midpoint method is

$$U^n = c_1(\zeta_1)^n + c_2(\zeta_2)^n,$$

so we can determine c_1 and c_2 using the particular values of U^0 and U^1 we are working with. To this end,

$$\begin{aligned}
U^0 &= \eta = c_1 + c_2 \\
&\Rightarrow c_1 = \eta - c_2 \\
U^1 &= (k\lambda + \sqrt{(k\lambda)^2 + 1})c_1 + (k\lambda - \sqrt{(k\lambda)^2 + 1})c_2 \\
&\vdots \\
&\Rightarrow c_1 = \frac{\eta}{2} \cdot \frac{1 + \sqrt{1 + (k\lambda)^2}}{\sqrt{1 + (k\lambda)^2}},\ c_2 = \frac{\eta}{2} \cdot \frac{-1 + \sqrt{1 + (k\lambda)^2}}{\sqrt{1 + (k\lambda)^2}}.
\end{aligned}$$

Thus

$$U^n = \frac{\eta}{2} \cdot \frac{1 + \sqrt{1 + (k\lambda)^2}}{\sqrt{1 + (k\lambda)^2}} \cdot (k\lambda + \sqrt{(k\lambda)^2 + 1})^n + \frac{\eta}{2} \cdot \frac{-1 + \sqrt{1 + (k\lambda)^2}}{\sqrt{1 + (k\lambda)^2}} \cdot (k\lambda - \sqrt{(k\lambda)^2 + 1})^n.$$

If $\lambda < 0$ and $k > 0$ then second root satisfies, $|\zeta_2 = k\lambda - \sqrt{(k\lambda)^2 + 1}| > 1$. Normally this would indicate that the method should be unstable on the problem at hand, since ζ_2 is exponentiated at each iteration. However further analysis reveals that for k small enough, this is not the case. First observe that c_2 is $O(k^2)$ for k small since

$$\begin{aligned}
c_2 &= \frac{\eta}{2} \cdot \frac{-1 + \sqrt{1 + (k\lambda)^2}}{\sqrt{1 + (k\lambda)^2}} = \frac{\eta}{2} \cdot \frac{-1 + (1 + \frac{1}{2}(k\lambda)^2 + O((k\lambda)^4)}{\sqrt{1 + (k\lambda)^2}} \\
&= \frac{\eta}{2} \cdot \frac{\frac{1}{2}(k\lambda)^2 + O((k\lambda)^4}{\sqrt{1 + (k\lambda)^2}} = \frac{\eta}{2} \cdot (\frac{1}{2}(k\lambda)^2 + O((k\lambda)^4) \cdot (1 - \frac{1}{2}(k\lambda)^2 + O((k\lambda)^4)) \\
&= O(k^2),
\end{aligned}$$

where we have used the Taylor expansions for $(1 + x^2)^{\frac{1}{2}}$ and $(1 + x^2)^{\frac{1}{2}}$ for small $|x|$. Now we turn our attention to the root ζ_2. Suppose we are interested in computing an approximation to the test problem at a fixed time $T = nk$. Again making use of a Taylor expansion, we have

$$\begin{aligned}
\zeta_2 &= k\lambda - \sqrt{1 + (k\lambda)^2} = k\lambda - (1 + \frac{1}{2}(k\lambda)^2 + O((k\lambda)^2) \\
&= -1 + k\lambda + O(k^2) \\
&\Rightarrow zeta_2^n = (-1 + k\lambda + O(k^2))^n \\
&= (-1)^n(1 + |\lambda|k + O(k^2))^n \\
&= (-1)^n(1 + \frac{|\lambda|T}{n} + O(k^2))^n \approx (-1)^n(1 + \frac{|\lambda|T}{n})^n.
\end{aligned}$$

As $k \to 0$, since $T = kn$ is fixed, $n \to \infty$ and $(-1)^n(1 + \frac{|\lambda|T}{n})^n \approx (-1)^n e^{|\lambda|T}$. This implies that ζ_2^n is bounded as n gets large and k small. Since $c_2 = O(k^2)$, for small k (and large n), we have

$$\begin{aligned} c_2\zeta_2^n &= \frac{\eta}{2} \cdot \frac{-1 + \sqrt{1 + (k\lambda)^2}}{\sqrt{1 + (k\lambda)^2}} \cdot (k\lambda - \sqrt{1 + (k\lambda)^2})^n = O(k^2) \cdot (k\lambda - \sqrt{1 + (k\lambda)^2})^n \\ &\approx O(k^2)e^{|\lambda|T} = O(k^2). \end{aligned}$$

Hence the term which had the potential to disrupt our approximation by blowing up is actually very small for small k. The other root is less than 1 in modulus, so it causes no instability, so the midpoint method works on the test problem.

(b) See the attached Juliabox notebook.

Problem 5

Perform numerical experiments to confirm the claim made in Example 7.10.

Solution:
See the attached Juliabox notebook.

Amath 586 HW2 Julia Notebook

April 17, 2015

1 Amath 586 HW 2

Brian de Silva

1.1 Problem 4(b)

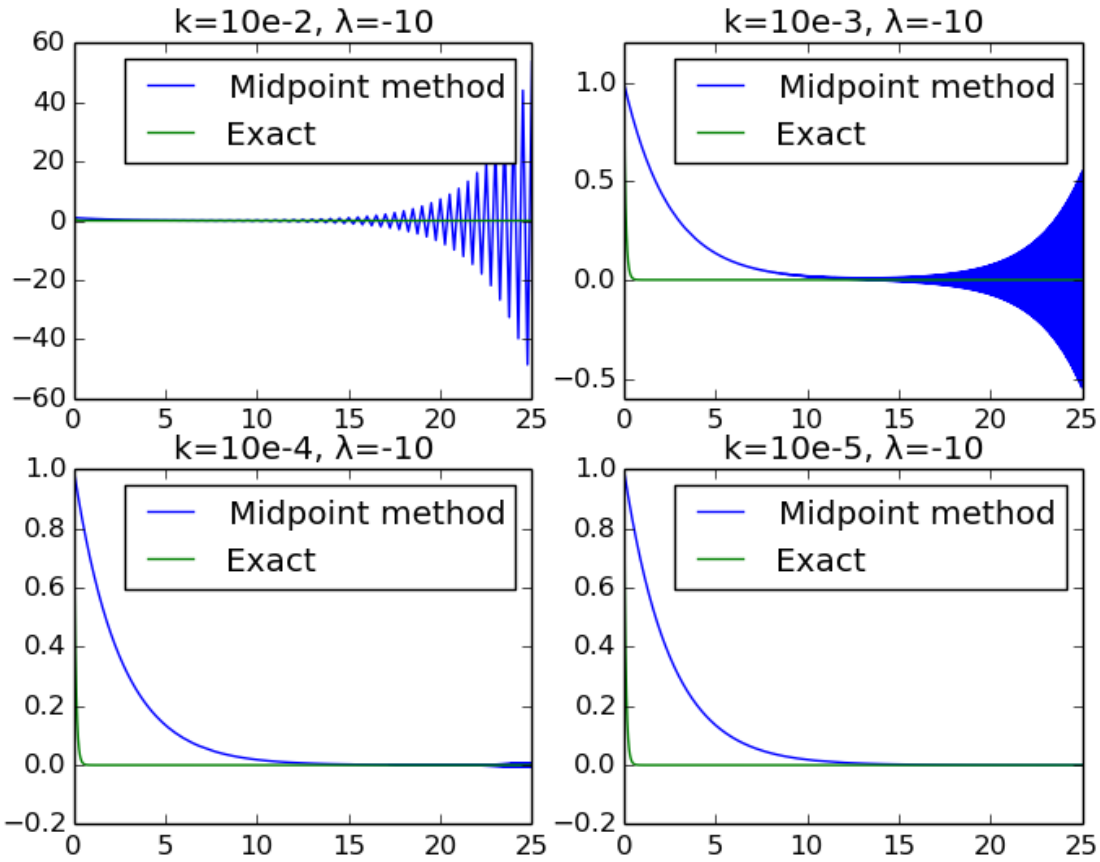
In [1]:
```
# Implement the midpoint method and import a plotting package

function midpointMethod(f,u0,T,n)
    # Numerically solves u'(t) = f(u(t)), u(t_0) = u_0 on [t_0,T] so that U^n ≈ u(T=t[U+2099])
    # Uses Forward Euler to obtain U^1

    U = zeros(n+1);
    U[1] = u0;
    k = 1/n;

    # Use one step of FE to get U^1
    U[2] = U[1] + k*f(U[1]);

    for j=2:n
        U[j+1] = U[j-1] + 2*k*f(U[j])
    end

    return U;
end;

using PyPlot;
```

```
INFO: Loading help data...
```

First we check that the midpoint method exhibits stability when applied to the test problem for small enough k and determine how small k must be to stabilize the method.

In [106]:
```
# Set some parameters for the test problem u'(t) = λu(t) ⟹ u(t) = exp(λt)
η = 1;
λ = -10;
T = 25;
f = u -> λ*u;
u0 = 1;

u_exact = t -> exp(λ*t);
fine_grid = linspace(0,T,100000);
```

```
# Apply method for various values of n (= T/k)
U100 = midpointMethod(f,η,T,100);
U1000 = midpointMethod(f,η,T,1000);
U10000 = midpointMethod(f,η,T,10000);
U100000 = midpointMethod(f,η,T,100000);

# Plot the results (analysis below)
PyPlot.figure()
PyPlot.subplot(2,2,1)
PyPlot.plot(linspace(0,T,101),U100,fine_grid,u_exact(fine_grid))
PyPlot.title("k=10e-2, λ=-10")
PyPlot.legend(["Midpoint method","Exact"])

PyPlot.subplot(2,2,2)
PyPlot.plot(linspace(0,T,1001),U1000,fine_grid,u_exact(fine_grid))
PyPlot.title("k=10e-3, λ=-10")
PyPlot.legend(["Midpoint method","Exact"])

PyPlot.subplot(2,2,3)
PyPlot.plot(linspace(0,T,10001),U10000,fine_grid,u_exact(fine_grid))
PyPlot.title("k=10e-4, λ=-10")
PyPlot.legend(["Midpoint method","Exact"])

PyPlot.subplot(2,2,4)
PyPlot.plot(linspace(0,T,100001),U100000,fine_grid,u_exact(fine_grid))
PyPlot.title("k=10e-5, λ=-10")
PyPlot.legend(["Midpoint method","Exact"]);
```



We see that the method is in fact stable for k small enough. Unfortunately, "small enough" in this case seems to be on the order of 10^{-4} or 10^{-5}. Even taking $k = 10^{-4}$ we still begin to see some oscillaiton in the solution close to $t = 10$, suggesting that if we took T to be larger, we would need an even smaller value of k to ensure stability.

1.2 Problem 5

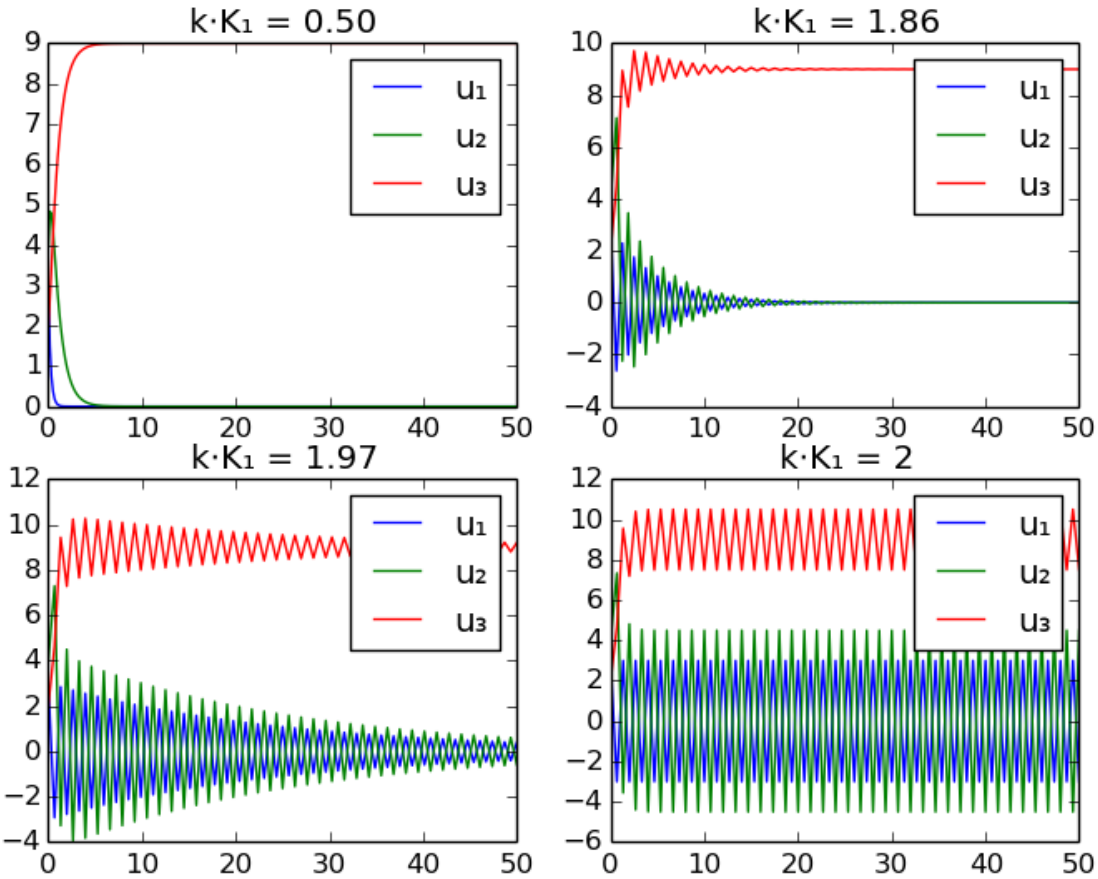
Here we wish to confirm the claim made in Example 7.10, that Euler's method is stable when it is applied to the problem presented in (7.10) exactly when $k \cdot \max(K_1, K_2) \leq 2$ and is explosively unstable even for k only slightly greater than this bound.

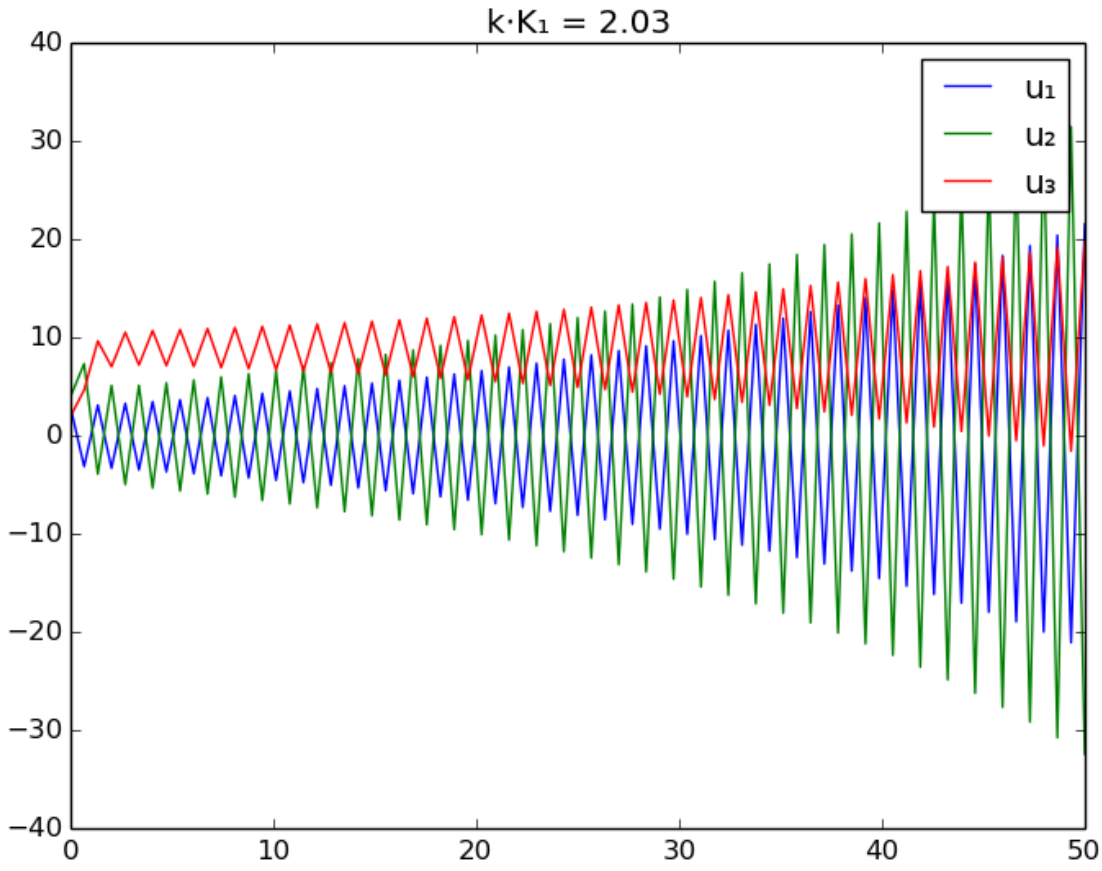
```
In [1]: # Euler's method implementation for linear systems
        # Solves the problem u' = Au, u(0) = u0

        function eulerSystem(A,u0,T,n)
        # Numerically solves u'(t) = Au(t), u(t0) = u0 on [t0,T] so that U^n ≈ u(T=t[U+2099])

            U = zeros(size(A)[1], n+1);
            U[:,1] = u0;
            k = T / n;

            for j = 1:n
                U[:,j+1] = U[:,j] + k*A*U[:,j];
            end
```

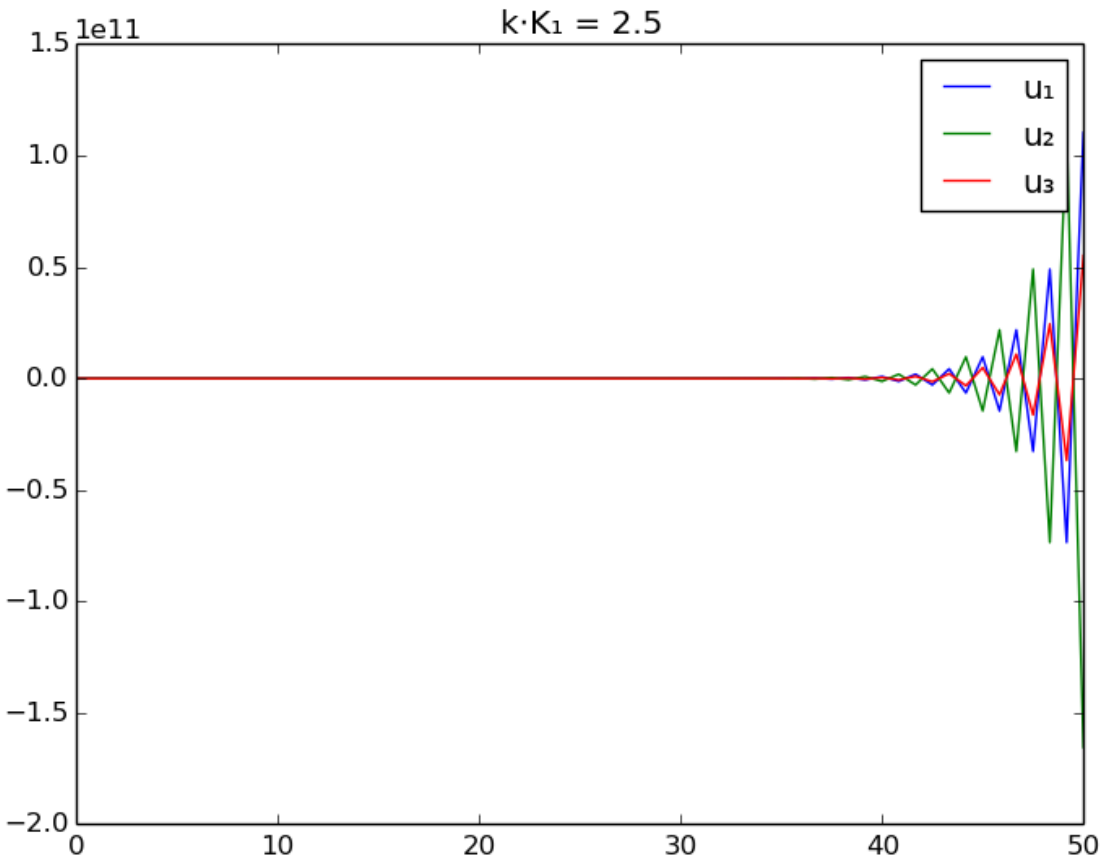
```
    return U
end;
```

In [11]:
```
# Define A and choose initial conditions (I went with those from Figure (7.4))
K₁ = 3;
K₂ = 1;
T = 50;

A = [-K₁ 0 0;
     K₁ -K₂ 0;
     0 K₂ 0];

η = [3;4;2];


# Apply Euler's method for various values of k (note: we should witness
# explosive instability as soon as k exceeds 2/3, i.e k·K₂ > 2)

# k·K₁ = 1/2 < 2
n = 300;
k = T / n;
U300 = eulerSystem(A,η,T,n)
PyPlot.subplot(2,2,1)
PyPlot.plot(linspace(0,T,n+1),U300[1,:]',linspace(0,T,n+1),U300[2,:]',linspace(0,T,n+1),U300[3
PyPlot.title("k·K₁ = 0.50")
PyPlot.legend(["u₁","u₂","u₃"]);

# k·K₁ = 1.5 < 2
n = 80;
k = T / n;
U80 = eulerSystem(A,η,T,n)
PyPlot.subplot(2,2,2)
PyPlot.plot(linspace(0,T,n+1),U80[1,:]',linspace(0,T,n+1),U80[2,:]',linspace(0,T,n+1),U80[3,:]
PyPlot.title("k·K₁ = 1.86")
PyPlot.legend(["u₁","u₂","u₃"]);

# k·K₁ = 1.97 < 2
n = 76;
k = T / n;
U76 = eulerSystem(A,η,T,n)
PyPlot.subplot(2,2,3)
PyPlot.plot(linspace(0,T,n+1),U76[1,:]',linspace(0,T,n+1),U76[2,:]',linspace(0,T,n+1),U76[3,:]
PyPlot.title("k·K₁ = 1.97")
PyPlot.legend(["u₁","u₂","u₃"]);

# k·K₁ = 2
n = 75;
k = T / n;
U75 = eulerSystem(A,η,T,n)
PyPlot.subplot(2,2,4)
PyPlot.plot(linspace(0,T,n+1),U75[1,:]',linspace(0,T,n+1),U75[2,:]',linspace(0,T,n+1),U75[3,:]
PyPlot.title("k·K₁ = 2")
PyPlot.legend(["u₁","u₂","u₃"]);
```

```
# k·K_1 = 2.03 > 2
n = 74;
k = T / n;
U74 = eulerSystem(A,η,T,n)
PyPlot.figure()
PyPlot.plot(linspace(0,T,n+1),U74[1,:]',linspace(0,T,n+1),U74[2,:]',linspace(0,T,n+1),U74[3,:]
PyPlot.title("k·K_1 = 2.03")
PyPlot.legend(["u_1","u_2","u_3"]);

# k·K_1 = 2.5 > 2
n = 60;
k = T / n;
U60 = eulerSystem(A,η,T,n)
PyPlot.figure()
PyPlot.plot(linspace(0,T,n+1),U60[1,:]',linspace(0,T,n+1),U60[2,:]',linspace(0,T,n+1),U60[3,:]
PyPlot.title("k·K_1 = 2.5")
PyPlot.legend(["u_1","u_2","u_3"]);
```




k·K₁ = 2.03
u₁
u₂
u₃
40
30
20
10
0
−10
−20
−30
−40
0
10
20
30
40
50



We see in the plots above that as soon as $k{\cdot}\max(K_1, K_2) = k \cdot K_1 > 2$ Euler's method becomes unstable. When the product is $\frac{1}{2}$ it seems to be performing very well. As we increase $k \cdot K_1$, we begin to see some oscillations for smaller time, but these are slowly damped out indicating stability. The oscillations persist longer and longer the more we increase $k \cdot K_1$, until, finally, when $k \cdot K_1 = 2$ no damping occurs whatsoever. The numerical solutions oscillate regularly about the true solutions without increasing nor decreasing in amplitude at all as time is advanced. We are just on the cusp of instability in this case. As soon as $k \cdot K_1$ is nudged past 2, instability begins manifesting itself as the oscillations increase in amplitude with time.