UCB Math 228A, Fall 2014: Homework Set 4

Due Nov. 3, 2014

Code Submission: E-mail all requested and supporting MATLAB files to Luming at `lwang@berkeley.edu` as a zip-file named lastname_firstname_4.zip, for example luming_wang_4.zip.

1. The *Cash-Karp method* below is a 4/5-order embedded Runge-Kutta scheme. Recall that the vectors b^T and $\hat{b}^T$ in the last two rows are used to produce two solution updates,

$$u_{n+1} = u_n + h \sum_{j=1}^{s} b_j k_j, \qquad \hat{u}_{n+1} = u_n + h \sum_{j=1}^{s} \hat{b}_j k_j, \tag{1}$$

where $\hat{u}$ is used only for estimation of the local trunction error $\tau \approx \|\hat{u}_{n+1} - u_{n+1}\|_\infty$.

0						
1/5	1/5					
3/10	3/40	9/40				
3/5	3/10	-9/10	6/5			
1	-11/54	5/2	-70/27	35/27		
7/8	1631/55296	175/512	575/13824	44275/110592	253/4096	
	2825/27648	0	18575/48384	13525/55296	277/14336	1/4
	37/378	0	250/621	125/594	0	512/1771

Implement a MATLAB function `rkck.m` with the syntax

```
function [t,u]=rkck(f,tlim,u0,abstol,hmin,hmax)
```

for solving ODEs using the Cash-Karp scheme with step size control. The input parameters are the right-hand side function `f(t,u)`, the start and end times `tlim=[t0,t1]`, the initial solution `u0`, the absolute tolerance `abstol`$=\delta$, and the smallest and largest acceptable timesteps `hmin`, `hmax`. The outputs are the times `t` and the solutions `u` at the actual timesteps (including `t0` and `t1`).

For the step size control, set the initial step size $h = h_{\max}$, and use the following strategy:

1. Take a step using the scheme and estimate τ
2. If $\tau \le \delta h$ accept the step, otherwise reject the step
3. Assuming that $\tau(h) = Kh^5$, find a new step size h_{new} such that $\tau(h_{\text{new}}) = 0.5 \cdot \delta h_{\text{new}}$
4. Set $h \leftarrow \min(\min(\max(h_{\text{new}}, 0.1h), 4h), h_{\max})$
5. If $h < h_{\min}$, terminate with error message
6. If needed, adjust h so that the final step will end exactly at the final time $t = t_1$
7. Repeat until final time reached

Test the function using the script `erk1ml` on the course web page, only replacing the call to `ode45` with your function `rkck`. Set `abstol=1e-3`, `hmin=1e-6`, and `hmax=1.0`.

2. Write a MATLAB function with the syntax

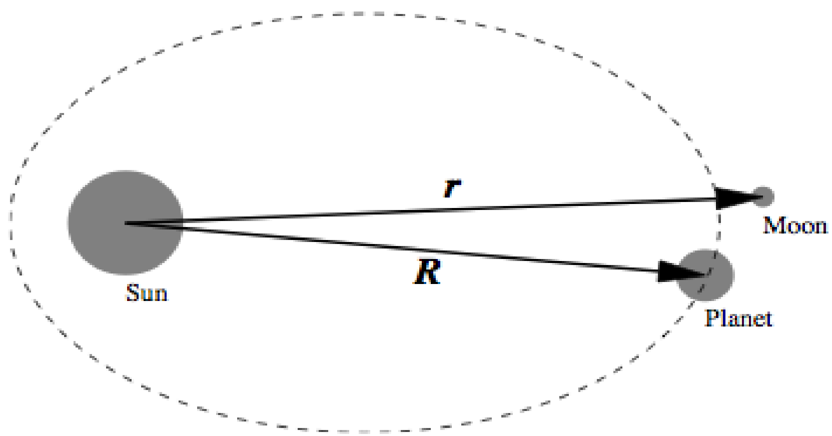
```
function [t,u]=orbit(m,T,u0)
```

which uses `rkck` from problem 1 to solve for the position up to time T of a planet $\boldsymbol{R} = (X, Y)$ of mass m_2 and a moon $\boldsymbol{r} = (x, y)$ of mass m_3, starting from the initial conditions $u_0 = [X, Y, x, y, \dot{X}, \dot{Y}, \dot{x}, \dot{y}]^T$. Assume that only gravitational forces are present and that the sun of mass m_1 is fixed (see figure below). The (attractive) gravitational force between two bodies of mass $m, \tilde{m}$ at $\boldsymbol{x}, \tilde{\boldsymbol{x}}$ is given by

$$\boldsymbol{F} = -m\tilde{m}\frac{\boldsymbol{x} - \tilde{\boldsymbol{x}}}{\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|^3}$$

and the motion $\boldsymbol{x}(t)$ of a body of mass m is given by Newton's second law, $m\ddot{\boldsymbol{x}} =$ the total force acting on the body. Use `abstol=1e-6`, `hmin=1e-6`, and `hmax=1.0`. Test the function using the commands

```
m=[1,1/100,1/8000]; T=30; u0=[5;0;5.1;0;0;.2;0;.3];
[t,u]=orbit(m,T,u0);
figure(1),plot(u(1,:),u(2,:),u(3,:),u(4,:)),axis equal
figure(2),semilogy(t(1:end-1),diff(t))
```



3. **a)** Implement a MATLAB function

```
function c=mkfdstencil(x,xbar,k)
```

which computes the coefficients c_i for a finite difference approximation $u^{(k)}(\bar{x}) \approx \sum_i c_i u(x_i)$.

b) Implement a MATLAB function

```
function u=bvp1(x,f,sigma,beta)
```

which solves the boundary value problem

$$u''(x) = f(x), \quad u'(a) = \sigma, \quad u(b) = \beta \tag{2}$$

using 3-point finite differences on a nonuniform grid. The inputs are the grid points `x`, the right-hand side function `f(x)`, and the boundary values. The output `u` is the solution vector. Test the function using the commands

```
x=linspace(0,1,100).^2;
f=@(x) sin(2*pi*x).*exp(x);
u=bvp1(x,f,0,0);
plot(x,u)
```

c) Implement a MATLAB function

```
function [e1,e2,slope1,slope2]=bvpconv(ns)
```

which solves the problem in `b` using $f(x) = e^x$, $\sigma = -5$, $\beta = 3$, $a = 0$, $b = 3$, and grids with n points for each values n in the vector `ns`. Use the following two point distributions:

1. `x=3*linspace(0,1,n).^2;`
2. `x=3*sort(rand(1,n)); x(1)=0; x(n)=3;`

The outputs `e1,e2` should be the inifinity norms of the errors for the two grid types (compare with the true solution), and `slope1,slope2` should be the estimated convergence rates (the slopes of the error versus $1/n$ in a log-log graph). Test the function using the commands

```
ns=round(logspace(1,3,50));
[e1,e2,s1,s2]=bvpconv(ns);
loglog(1./ns,e1,1./ns,e2)
[s1,s2]
```