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,

$$
\begin{align*}
    u_{n+1} &= u_n + h \sum_{j=1}^{s} b_j k_j, \\
    \hat{u}_{n+1} &= u_n + h \sum_{j=1}^{s} \hat{b}_j k_j,
\end{align*}
$$

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

\[
\begin{array}{ccccccccc}
0 & 1/5 & 1/5 & 3/10 & 3/40 & 9/40 & 3/5 & 3/10 & -9/10 & 6/5 \\
1 & -11/54 & 5/2 & -70/27 & 35/27 & 7/8 & 1631/55296 & 175/512 & 575/13824 & 35/27 \\
& & & & & & 2825/27648 & 0 & 18575/48384 & 13525/55296 & 253/4096 \\
& & & & & & 37/378 & 0 & 250/621 & 125/594 & 512/1771 \\
\end{array}
\]

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

```matlab
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 $\tau$
2. If $\tau \leq \delta h$ accept the step, otherwise reject the step
3. Assuming that $\tau(h) = Kh^5$, find a new step size $h_{new}$ such that $\tau(h_{new}) = 0.5 \cdot \delta h_{new}$
4. Set $h \leftarrow \min(\min(\max(h_{new}, 0.1h), 4h), h_{max})$
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

```matlab
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 $R = (X,Y)$ of mass $m_2$ and a moon $r = (x,y)$ of mass $m_3$, starting from the initial conditions $u_0 = [X,Y,x,y,X,Y,x,y]^T$. 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 $x, \tilde{x}$ is given by

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

and the motion $x(t)$ of a body of mass $m$ is given by Newton’s second law, $m\ddot{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

```matlab
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

```matlab
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

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

which solves the boundary value problem

$$u''(x) = f(x), \quad u'(a) = \sigma, \quad u(b) = \beta$$

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
\[ x = \text{linspace}(0,1,100)^2; \]
\[ f = \theta(x) \sin(2\pi x) \cdot \exp(x); \]
\[ u = \text{bvp1}(x,f,0,0); \]
\[ \text{plot}(x,u) \]

c) Implement a MATLAB function

\[
\text{function } [e1,e2,slope1,slope2] = \text{bvpconv}(ns)
\]

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. \( x = 3 \cdot \text{linspace}(0,1,n)^2; \)
2. \( x = 3 \cdot \text{sort}(\text{rand}(1,n)); \quad x(1)=0; \quad x(n)=3; \)

The outputs \( e1, e2 \) should be the infinity norms of the errors for the two grid types (compare with the true solution), and \( \text{slope1, slope2} \) 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

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

**Code Submission:** E-mail all requested and supporting MATLAB files to David at anderson@math.berkeley.edu as a zip-file named `lastname_firstname_5.zip`, for example `anderson_david_5.zip`. 