MATH 5620 NUMERICAL ANALYSIS II
HOMEWORK 3, DUE FRIDAY FEBRUARY 19 2010

Problem 1 B&F 5.9.1 a and 5.9.3 a (Runge-Kutta 4 for systems).
Problem 2 B&F 11.1.2 a,b (Linear shooting method)
Problem 3 B&F 11.2.4 a,c (Nonlinear shooting method)
Problem 4 B&F 11.3.2 a,b (Linear finite differences)
Problem 5 B&F 11.4.4 a,c (Nonlinear finite differences)

Implementation tips (Please read!)

- Problems 1–3 are simpler to implement if you write first a "vectorized" version of the Runge-Kutta method of order 4 that can deal with first order systems of the kind

$$\begin{cases} \mathbf{y}' = \mathbf{f}(t, \mathbf{y}), \text{ for } t \in [a, b] \\ \mathbf{y}(a) = \alpha \end{cases}$$

where $\mathbf{y}(t) \in \mathbb{R}^n$ is the solution vector, $\alpha \in \mathbb{R}^n$ is the initial condition and $\mathbf{f} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$. Such a function would have the interface:

```
% function y = rk4(a,b,n,y0,f)
%
% Runge-Kutta method for systems
%
%    y' = f(t,y),   t in [a,b]
%    y(a) = y0
%
% Inputs
%  a,b      time interval
%  n        number of subintervals
%  y0       initial condition (vector of length m)
%  f        function handle defining the problem
%             f = @(t,y) ....
%           where t = time, y = vector of length m
%           and output is a vector of length m
%
% Outputs:
%  y        iterate history (m by n+1 matrix)
```

- For 11.2.4 a,c (11.4.4 a,c) it is helpful to compare the results for same method/problem to those in 11.2.3 a,c (11.4.2 a,c).
- For Problems 2 and 3:
  - Instead of rewriting the algorithms in the book, use the vectorized version of Runge-Kutta of order 4 you wrote for Problem 1. Steps 2–4 in Algorithm 11.1 and Steps 4–6 in Algorithm 11.2 are simply a call to the routine from Problem 1.

  - Do not be alarmed if you obtain slightly different results from those in the book. This is because the Runge-Kutta method used in Algorithms 11.1 and 11.2 is modified to take advantage of the particular structure of the problem. Some reference numbers with the (simpler) approach I propose are posted in the class website.
  - **Note** For the linear shooting method, the class textbook takes a linear combination of two different solutions, but requires you to give the constant $r(t)$ term separately. Both ways of deriving the solution should give identical results (within machine precision).
- For Problems 4 and 5:
  - It is much simpler to use sparse matrices to construct the tridiagonal systems. For example the discretization L of y'' on a uniform grid can be obtained in Matlab by the command

    ```
    L = (1/h^2)*spdiags(ones(n,1)*[1,-2,1],-1:1,n,n);
    ```

  - Replace the tridiagonal linear system solve steps by Matlab's backslash in Algorithms 11.3 and 11.4. Such systems are relatively cheap to solve when you specify them as sparse matrices.
  - The behavior of `spdiags` can be tricky if your sub/super diagonals vary. This matlab code:

    ```
    Q = [ 1  3  0
          2  3 -1
          3  3 -2
          0  3 -3 ];
    A = spdiags(Q,-1:1,4,4);
    ```

    constructs the matrix with entries:

    $$A = \begin{bmatrix} 3 & -1 & & \\ 1 & 3 & -2 & \\ & 2 & 3 & -3 \\ & & 3 & 3 \end{bmatrix}$$

    So `spdiags` uses only the upper part of subdiagonals and the lower part of superdiagonals.
  - If you dont want to use `spdiags` you can create an n by n all zeros sparse matrix with `A=sparse(n,n);` and then fill it entry by entry.