MATH 5620 NUMERICAL ANALYSIS II
HOMEWORK 4, DUE MONDAY MARCH 19 2012
(TWO PAGES)

Problem 1 B&F 11.1.4 a,b (Linear shooting method)
Problem 2 B&F 11.2.4 a,b (Nonlinear shooting method)
Problem 3 B&F 11.3.4 a,b (Linear finite differences)
Problem 4 B&F 11.4.4 a,b (Nonlinear finite differences)

Implementation tips (Please read!)

- Problems 1–2 are simpler to implement if you use 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 (in Matlab) is provided in the class website (`rk4.m`).
- For Problems 1 and 2:
  - Instead of rewriting the algorithms in the book, use the vectorized version of Runge-Kutta of order 4 that is provided. 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 are equivalent and should give identical results (within machine precision).
- For Problems 3 and 4:
  - 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 the system matrix is sparse.

- 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 do not 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 (with a for loop).