CS4210 Assignment 7 Due: 12/5/14 (Fri) at 6pm

You must work either on your own or with one partner. You may discuss background issues and general solution strategies with others, but the solutions you submit must be the work of just you (and your partner). If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group. Points may be deducted for poor style and reckless inefficiency.

Topics: Methods for Boundary Value Problems, B-spline review, matrix set-up

1 The Collocation Method

In this problem you are to compute an approximate solution $\tilde{u}(x)$ to the boundary value problem

$$-u''(x) \; + \; q(x)u(x) \; = \; r(x) \qquad a \le x \le b \tag{1}$$

$$u(a) \; = \; u_a \quad u(b) \; = \; u_b \tag{2}$$

using the *method of collocation*. The idea is to express $\tilde{u}(x)$ as a linear combination of simple basis functions. The coefficients that define the linear combination are then determined via a linear system that is obtained by imposing certain conditions on $\tilde{u}(x)$.

For basis functions we will use the B-splines $B_0(x), \ldots, B_{n+1}(x)$ that were introduced in A3. In particular, we will seek an approximate solution to (1) of the form

$$\tilde{u}(x) \; = \; \sum_{k=0}^{n+1} \alpha_k B_k(x)$$

where

$$B_k(x) \; = \; B_* \left(\frac{x - x_k}{h} \right)$$

with $h = (b-a)/(n-1)$ and

$$x_k = a + (k-1)h \qquad k = 0{:}n+1.$$

Recall that A3 was about interpolation with B-splines. In that assignment we also determined $\alpha_0, \ldots, \alpha_{n+1}$ via a linear system solve. The linear equations enforced interpolation conditions

$$\tilde{u}(x_i) \; = \; y_i \qquad i = 1{:}n.$$

and a pair of end conditions, e.g., $u''(a) = 0$, $u''(b) = 0$.

For the BVP (1)-(2) we proceed similarly. As in A3, the linear equations stipulate the properties that we want $\tilde{u}(x)$ to satisfy. The first of these is the left boundary condition:

$$\tilde{u}(a) \; = \; \tilde{u}(x_1) \; = \; u_a$$

Next, we insist that the differential equation is satisfied by $\tilde{u}(x)$ at the *collocation points* $x_1, \ldots, x_n$:

$$-\tilde{u}''(x_i) \; + \; q(x_i)\tilde{u}(x_i) \; = \; r(x_i) \qquad i = 1{:}n. \tag{3}$$

Lastly, we want to be sure that the right boundary condition is satisfied:

$$\tilde{u}(b) \; = \; \tilde{u}(x_n) \; = \; u_b$$

You job is to implement a function that does this:

```
  function alpha = BVP_Collocation(a,ua,b,ub,q,r,n)
% a<b and ua and ub are scalars.
% q and r are handles to functions q(x) and r(x) defined on [a,b].
% n is a positive integer, n >=2.
% alpha is a column (n+2)-vector with the property that if
%
%          utilde(x) = alpha(1)B_0(x) + ... + alpha(n+2)*B_{n+1}(x)
%
% then
%          utilde(a) = ua
%         -utilde''(z(i)) + q(z(i))u(z(i)) = r(z(i))     i=1:n
%          utilde(b) = ub
%
% where z = linspace(a,b,n) and
% B_{k}(z) is the B-spline Bstar((z-xk)/h) where xk = a+(k-1)h
% and h = (b-a)/(n-1).
```

(Sorry for the subscript-from-one annoyances.) A test script `ShowCollocation` is provided that can be used to compare your implementation with an analogous procedure based on the method of finite differences. (You will also need to download `BVP_FiniteDiff`).

In this problem we are NOT concerned with the efficient set up of the linear system that specifies the α's. Assignment A3 gave you enough practice with that, i.e., the exploitation of the local support feature of the B-spline basis. Clearly, that "technology" can be exploited here. Our goal is simply for you to appreciate the collocation framework by using it to solve a simple problem. So that you do not get bogged down in low-level details associated with the evaluation of the B_k and their derivatives, we supply the following function on the website:

```
  function [y,dy,ddy] = derBstar(z)
% z is a scalar
% y   = Bstar(z)
% dy  = Bstar'(z)
% ddy = Bstar''(z)
```

(Note that the equations in (3) involve second derivatives of the B_k evaluated at the x_i.) Again, don't spend time vectorizing or exploiting the local support properties of the basis function–just set up the linear system correctly and use \. Submit `BVP_Collocation` to CMS.

2 The Crank-Nicholson Method for the Heat Equation

Here is a simple version of the *heat equation*:

$$\frac{\partial}{\partial t}u(x,t) \;=\; \frac{\partial^2}{\partial x^2}u(x,t) \;+\; s(x,t) \qquad a \le x \le b, \quad t \ge 0 \tag{4}$$

Think of $u(x,t)$ as the temperature of a rod at time t where $s(x,t)$ is a given heat-source function. The temperature at the start is known,

$$u(x,t) = u^{(0)}(x) \qquad a \le x \le b$$

and remains the same at the endpoints

$$u(a,t) = u^{(0)}(a) = u_a \qquad u(b,t) = u^{(0)}(b) = u_b \qquad t \ge 0. \tag{5}$$

For us, the discretization of this problem involves two parameters. One involves space and one involves time:

$$h = (b-a)/(n-1) \qquad \Delta_t \;>\; 0.$$

The goal is to produce approximations

$$u_k^{(j)} \;\approx\; u(x_k, t_j)$$

where $x_k = a + (k-1)h$ for $k = 1{:}n$ and $t_j = j\Delta_t$ for $j = 0, 1, 2, \ldots$. Since the value of $u(x,t)$ is fixed at the endpoints (see equation (4)), we set

$$u_1^{(j)} = u_a \qquad u_n^{(j)} = u_b \qquad j = 0, 1, 2, \ldots$$

The *Crank-Nicholson* scheme relates the approximate solution at time t_{j+1} to the approximate solution at time t_j as follows:

$$\frac{u_k^{(j+1)} - u_k^{(j)}}{\Delta_t} = \frac{1}{2}\left(\frac{\dfrac{u_{k+1}^{(j)} - u_k^{(j)}}{h} - \dfrac{u_k^{(j)} - u_{k-1}^{(j)}}{h}}{h} + \frac{\dfrac{u_{k+1}^{(j+1)} - u_k^{(j+1)}}{h} - \dfrac{u_k^{(j+1)} - u_{k-1}^{(j+1)}}{h}}{h} \right) + s\left(x_k, t_j + \frac{\Delta_t}{2}\right)$$

Assume that we know $u_k^{(j)}$, $k = 1{:}n$ and want to compute $u_k^{(j+1)}$, $k = 1{:}n$. Of course, the endpoint values are known,

$$u_1^{(j)} = u_1^{(j+1)} = u_a \qquad u_n^{(j)} = u_n^{(j+1)} = u_b$$

so it is all about computing $u_2^{(j+1)}, \ldots, u_{n-1}^{(j+1)}$ from known stuff. Using the giant Crank-Nicholson divided difference recipe above, show that we have an $(n-2)$-by-$(n-2)$ linear system of the form

$$T \begin{bmatrix} u_2^{(j+1)} \\ \vdots \\ u_{n-1}^{(j+1)} \end{bmatrix} = \text{rhs that involves } u_1^{(j)}, \ldots, u_n^{(j)},\ h,\ \Delta_t,\ s\text{-evaluations}$$

Complete the following function so that it carries out a Crank-Nicholson step

```
  function uNext = CrankN(uNow,a,b,n,tc,deltaT,s)
% uNow is a column n-vector.
% a < b
% n is a positive integer
% tc is the ``current time''.
% deltaT >0 is the time step.
% s is a handle to a function of the form s(x,t)
% uNext is a column n-vector whose entries satisfy uNext(1) = uNow(1),
%     uNext(n) = uNow(n), and


%  uNew(k) - uNow(k)      1   (uNew(k+1)-2*uNew(k)+uNew(k-1))
%  -----------------  =  ---  -----------------------------   +
%      deltaT             2                h^2

%                         1   (uNow(k+1)-2*uNow(k)+uNow(k-1))
%                        ---  -----------------------------   + s(x(k),tc+deltaT/2)
%                         2                h^2

% for k=2:n-1 where

h = (b-a)/(n-1);
x = linspace(a,b,n);
```

It is fine for you NOT to exploit T's sparse structure. Just set it up explicitly and use \. A test script `ShowHeat` is provided. Submit `CrankN` to CMS.

3 The Shooting Method for a Two-Point Boundary Value Problem

The function `Cannon_v0(v0)` solves the A5 initial value problem

$$\begin{aligned} \dot{x} &= v(t)\cos(\theta(t)) \\ \dot{y} &= v(t)\sin(\theta(t)) \\ \dot{\theta} &= -g/v(t)\cos(\theta(t)) \\ \dot{v} &= -D(t)/m - g\sin(\theta(t)) \end{aligned}$$

where

$$D(t) = \frac{c\rho s}{2}((\dot{x}(t) + w)^2 + \dot{y}^2)$$

and $x(0) = 0$, $y(0) = 0$, $\theta(0)$, and $v(0) = v_0$ are the given initial conditions. For a given input `v0`, the function displays a table that reports just how far the cannonball travels for various initial angles and constant wind speeds, e.g.,

```
v0 = 50.000

Cannonball Distance as a function of initial angle A (degrees) and headwind w

  A      w = -20     w = -10       w = 0      w = 10      w = 20
----------------------------------------------------------------------------
 10       83.496      80.871      77.674      74.008      69.970
 15      119.756     114.473     108.154     101.090      93.474
 20      151.314     143.057     133.235     122.421     111.029
 25      177.530     166.358     153.130     138.620     123.540
 30      197.824     184.134     167.840     150.027     131.618
 35      211.767     196.244     177.446     156.887     135.705
 40      219.044     202.618     182.094     159.575     136.283
 45      219.433     203.202     182.118     158.356     133.729
 50      212.964     198.261     177.480     153.422     128.359
 55      199.771     187.671     168.312     144.925     120.099
 60      180.184     171.603     154.651     132.812     109.077
 65      154.354     150.089     136.694     117.485      95.906
 70      124.483     124.475     115.042      99.226      80.450
```

Develop an analogous function `Cannon_d(d)` that determines the required initial velocity v_0 so that the cannonball travels exactly distance d before landing. Sample output:

```
Required travel distance = 200.000

Required initial velocity as a function of initial angle A (degrees) and headwind w

     A      w = -20     w = -10       w = 0      w = 10      w = 20
   ----------------------------------------------------------------------------
    10       82.241      85.022      88.442      92.584      97.557
    15       67.241      69.858      73.252      77.549      82.915
    20       58.850      61.341      64.743      69.214      74.987
    25       53.638      56.048      59.478      64.174      70.367
    30       50.326      52.667      56.161      61.083      67.771
    35       48.317      50.607      54.206      59.389      66.698
    40       47.331      49.582      53.289      58.842      66.879
    45       47.258      49.484      53.347      59.348      68.290
    50       48.119      50.288      54.386      60.961      71.097
    55       50.035      52.181      56.575      64.015      75.496
    60       53.465      55.525      60.423      68.819      82.458
    65       59.164      61.121      66.178      76.804      93.710
    70       69.047      70.784      77.324      89.973     113.175
```

To do this you need to understand and make use of the MATLAB root-finder fzero. A simple example tells all. Suppose

```
function z = MyF(x,a)
z = a*x^2 - 20;
```

is available. We know that if $a = 2$ then this function has a single root in the interval [3,4]. The following script assigns the root to r:

```
a = 2;
L = 3;
R = 4;
Bracketing_Interval = [L,R]; % Contains the root of interest
r = fzero(@(x)  MyF(x,a),Bracketing_Interval)
```

Now here is what you do to compute the "magic v_0, i.e., the initial velocity so that when the cannonball lands, $x(t_{final}) = x_{final} = d$. Suppose F(v0,theta,w,d) is a function that runs ode45 with the same terminate-on-landing event function and with initial condition [0;0;theta;v0]. If F returns the value of $x_{final} - d$, then it evaluates to zero precisely when v_0 is the required initial velocity, i.e., the initial velocity that causes the terminating value of x to be d. Thus, to produce Cannon_d(d) you need to adjust Cannon_v0 so that (a) it includes the subfunction F just described and (b) it replaces the function d = HowFar(theta,w,v0) with a function v0 = HowFast(theta,w,d) that returns the required initial velocity. HowFast is essentially a one-liner that calls fzero. Think a little bit about the required bracketing interval that you pass to fzero. The smaller the interval the fewer the number of F-calls and that means a reduced number of f-evaluations overall. Include comments on how you pick the bracketing interval. Note: fzero is unhappy if there is no root in the bracketing interval. You may assume that the incoming d satisfies $10 \leq d \leq 500$. Submit Cannon_d to CMS.