Borthagaray, Juan P.

AMSC/CMSC 460 Section 0201 (Spring 2019)

FINAL EXAM

Instructions:

- You must submit all the Matlab codes you used through the course ELMS/CANVAS page by May 17 at 10AM.
- Besides this, you have to hand in your submission on paper at MATH 4309 by May 17 at 1PM.
- There will be a 10 % penalty for each hour, or fraction of an hour, late.
- You do not need to write every piece of code from scratch. You may use any code from the book's toolbox or from the course page.

---

1. (30 pts)[1] (a) Create a Matlab function `perspline` by modifying `splinetx` to replace the not-a-knot end conditions with periodic boundary conditions. This requires that the given data have

$$y_n = y_1$$

and that the resulting interpolant P be periodic. In other words, for all x,

$$P(x + \Delta) = P(x),$$

where $\Delta = x_n - x_1$. The algorithms for `spline` involve calculations with y_k, h_k, and δ_k to produce slopes d_k. With the periodicity assumption, all of these quantities become periodic functions, with period $n-1$, of the subscript k. In other words, for all k,

$$\begin{aligned} y_k &= y_{k+n-1}, \\ h_k &= h_{k+n-1}, \\ \delta_k &= \delta_{k+n-1}, \\ d_k &= d_{k+n-1}. \end{aligned}$$

This makes it possible to use the same calculations at the endpoints that are used at the interior points in the nonperiodic situation. The special case code for the end conditions can be eliminated and the resulting M-files are actually much simpler.

Indeed, the slopes satisfy a system of simultaneous linear equations for $k = 2, \ldots, n-1$:

$$h_k d_{k-1} + 2(h_{k-1} + h_k)d_k + h_{k-1}d_{k+1} = 3(h_k\delta_{k-1} + h_{k-1}\delta_k).$$

With periodicity, this becomes

$$h_1 d_{n-1} + 2(h_{n-1} + h_1)d_1 + h_{n-1}d_2 = 3(h_1\delta_{n-1} + h_{n-1}\delta_1)$$

at $k = 1$ and

$$h_1 d_{n-1} + 2(h_{n-1} + h_1)d_n + h_{n-1}d_2 = 3(h_1\delta_{n-1} + h_{n-1}\delta_1)$$

at $k = n$. The resulting matrix has two nonzero elements outside the tridiagonal structure. The additional nonzero elements in the first and last rows are $A_{1,n-1} = h_1$ and $A_{n,2} = h_{n-1}$.

Demonstrate that your function works correctly on

```
x = 0:pi/4:2*pi;
y = cos(x).^2;
u = 0:pi/50:2*pi;
v = perspline(x,y,u);
plot(x,y,'o',u,v,'-')
```

(b) Let P be the spline interpolating polynomial. Recall that, over the subinterval $[x_k, x_{k+1}]$, we use the notation

$$y_k = P(x_k), \quad y_{k+1} = P(x_{k+1}), \quad d_k = P'(x_k), \quad d_{k+1} = P'(x_{k+1}), \quad s = x - x_k, \quad h = x_{k+1} - x_k,$$

and therefore we have

$$P(s) = \frac{3hs^2 - 2s^3}{h^3}y_{k+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3}y_k + \frac{s^2(s-h)}{h^2}d_{k+1} + \frac{s(s-h)^2}{h^2}d_k.$$

Show that the integral of P over the subinterval $[x_k, x_{k-1}]$ is

$$\int_0^h P(s)ds = h\left(\frac{y_{k+1} + y_k}{2}\right) - h^2\left(\frac{d_{k+1} - d_k}{12}\right). \tag{1}$$

---

[1] This problem is based on exercises 3.13, 6.20 and 6.21 in Moler's book.

(c) Modify your code from part (a) to create `persplinequad`, that integrates discrete data using periodic spline interpolation. More precisely, `Q = persplinequad(x,y)` should compute

$$\int_{x_1}^{x_n} P(x)dx,$$

where P is the periodic spline through the data points. Formula (1) should be useful for your implementation.

Test the performance of your function on

```
x = 0:pi/4:2*pi;
y = cos(x).^2;
persplinequad(x,y);
```

---

2. (25 pts) We consider the following model of pasture yield y as a function of the growing time t:

$$y(t) = \frac{\beta}{1 + e^{\lambda_1 - \lambda_2 t}}. \tag{2}$$

This model, that corresponds to a so-called *sigmoidal growth curve*, is linear in the parameter β and nonlinear in the parameters λ_1, λ_2.

(a) The file `pasture_yield.txt` contains a data set: the first column corresponds to t and the second to $y(t)$. Modify the Matlab script and `expfitfun.m` to produce the least squares fitting of these data. The parameters λ_1 and λ_2 are of the orders of 1 and 10^{-2}, respectively.
(You may use as a starting point the code `Separable_least_squares.m` we displayed in class and available at the course page.)

(b) For the values of $\beta, \lambda_1, \lambda_2$ you found in (a), use the secant method to find the value of t such that $y(t) = 30$.

(c) (This part is independent of the previous two) Suppose that, in model (2), we know that $\beta = 100(0.1 - \lambda_2)$ and that $\lambda_1 = 1$. Find the value of λ_2 that maximizes the pasture yield at $t = 20$.

---

3. (45 pts)[2] The function $y(x)$ is defined on the interval $0 \le x \le 1$ by

$$\begin{aligned} y''(x) &= y^2(x) - 1, \\ y(0) &= 0, \\ y(1) &= 1. \end{aligned} \tag{3}$$

This problem can be solved four different ways. Plot the four solutions obtained on a single figure, using `subplot(2,2,1),...,` `subplot(2,2,4)`.
(a) **Shooting method.** Suppose we know the value of $\eta = y'(0)$. Then we could use an ordinary differential equation solver like `ode23tx` or `ode45` to solve the initial value problem

$$\begin{aligned} y'' &= y^2 - 1, \\ y(0) &= 0, \\ y'(0) &= \eta, \end{aligned}$$

on the interval $0 \le x \le 1$. Each value of η determines a different solution $y(x;\eta)$ and corresponding value for $y(1;\eta)$. The desired boundary condition $y(1) = 1$ leads to the definition of a function of η:

$$f(\eta) = y(1;\eta) - 1.$$

Write a Matlab function whose argument is η. This function should solve the ordinary differential equation initial problem and return $f(\eta)$. Then use `fzero` or `fzerotx` to find a value η_* so that $f(\eta_*) = 0$. Finally, use this η_* in the initial value problem to get the desired $y(x)$. Report the value of η_* you obtain.

(b) **Quadrature.** Observe that $y'' = y^2 - 1$ can be written

$$\frac{d}{dx}\left(\frac{(y')^2}{2} - \frac{y^3}{3} + y\right) = 0.$$

[2] This problem is based on exercise 7.22 in Moler's book.

This means that the expression

$$\kappa = \frac{(y')^2}{2} - \frac{y^3}{3} + y$$

is actually constant. Because $y(0) = 0$, we have $y'(0) = \sqrt{2\kappa}$. So, if we could find the constant κ, the boundary value problem would be converted into an initial value problem. Integrating the equation

$$\frac{dx}{dy} = \frac{1}{\sqrt{2(\kappa + y^3/3 - y)}}$$

gives

$$x = \int_0^y h(y,\kappa)dy$$

where

$$h(y,\kappa) = \frac{1}{\sqrt{2(\kappa + y^3/3 - y)}}.$$

This, together with the boundary condition $y(1) = 1$, leads to the definition of a function $g(\kappa)$:

$$g(\kappa) = \int_0^1 h(y,\kappa)dy - 1.$$

You need two Matlab functions, one that computes $h(y,\kappa)$ and one that computes $g(\kappa)$. They can be two separate m-files, but a better idea is to make $h(y,\kappa)$ a function within $g(\kappa)$. The function $g(\kappa)$ should use `quadtx` to evaluate the integral of $h(y,\kappa)$. The parameter κ is passed as an extra argument from g, through `quadtx`, to h. Then `fzerotx` can be used to find a value κ_* so that $g(\kappa_*) = 0$. Finally, this κ_* provides the second initial value necessary for an ordinary differential equation solver to compute $y(x)$. Report the value of κ_* you obtain.

In parts (c, d and e) we consider a **Finite Difference method.** Partition the interval into $n+1$ equal subintervals with spacing $h = 1/(n+1)$:

$$x_i = ih, \quad i = 0,\ldots,n+1.$$

We now aim to solve the differential equation by finding n unknowns, $y_1, y_2, \ldots, y_n$, such that $y_i \approx y(x_i)$. To impose the first identity in (3) at the interior nodes $x_1,\ldots,x_n$, we approximate

$$y''(x_i) \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}. \tag{4}$$

(c) Show that, if y has as many continuous derivatives as needed, then

$$\left| y''(x_i) - \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} \right| \le Ch^2.$$

What does the constant C above depend on?

Using (4), we are led to the following nonlinear system of difference equations:

$$y_{i+1} - 2y_i + y_{i-1} = h^2(y_i^2 - 1), \quad i = 1,\ldots,n.$$

The boundary conditions are $y_0 = 0$ and $y_{n+1} = 1$.

A convenient way to compute the vector of second differences involves the n-by-n tridiagonal matrix A with -2's on the diagonal, 1's on the superand subdiagonals, and 0's elsewhere. You can generate a sparse form of this matrix with

```
e = ones(n,1);
A = spdiags([e -2*e e],[-1 0 1],n,n);
```

The boundary conditions $y_0 = 0$ and $y_{n+1} = 1$ can be represented by the n-vector b, with $b_i = 0$, $i = 1,\ldots,n-1$, and $b_n = 1$. The vector formulation of the nonlinear difference equation is

$$Ay + b = h^2(y^2 - 1),$$

where y^2 is the vector containing the squares of the elements of y, that is, the Matlab element-by-element power `y.^2`. There are at least two ways to solve this system.

(d) **Linear iteration.** This is based on writing the difference equation in the form

$$Ay = h^2(y^2 - 1) - b.$$

Start with an initial guess for the solution vector y. The iteration consists of plugging the current y into the right-hand side of this equation and then solving the resulting linear system for a new y. This makes repeated use of the sparse backslash operator with the iterated assignment statement

```
y = A\(h^2*(y.^2 - 1) - b)
```

It turns out that this iteration converges linearly and provides a robust method for solving the nonlinear difference equations. Report the value of n you use and the number of iterations required.

(e) **Newton's method.** This is based on writing the difference equation in the form

$$F(y) = Ay + b - h^2(y^2 - 1) = 0.$$

Newton's method for solving $F(y) = 0$ requires a many-variable analogue of the derivative $F'(y)$. The analogue is the Jacobian, the matrix of partial derivatives

$$J = \frac{\partial F_i}{\partial y_j} = A - h^2 \mathrm{diag}(2y).$$

In Matlab, one step of Newton's method would be

```
F = A*y + b - h^2*(y.^2 - 1);
J = A - h^2*spdiags(2*y,0,n,n);
y = y - J\F;
```

With a good starting guess, Newton's method converges in a handful of iterations. Report the value of n you use and the number of iterations required.

(f) **An optimization problem.** We now consider the equation

$$\begin{aligned} y''(x) &= \alpha y^2(x) - 1, \\ y(0) &= 0, \\ y(1) &= \sin(\pi\alpha/2), \end{aligned} \tag{5}$$

where α is a given positive number. Modify your code from part (e) to create a Matlab function that, given a number `alpha`, solves (5) using $n = 101$, and gives as output `y(51)`$\approx y(0.5)$. Use this function to determine the value of $\alpha \in [0, 1]$ that makes $y(0.5)$ as large as possible.