Bathagaray, J. P.

AMSC/CMSC 460 Section 0401 (Fall 2017)

FINAL EXAM - TAKE HOME ASSIGNMENT

Instructions:

- You have to hand in your submission on paper. Besides this, you must submit all your Matlab codes you used to `jpb@umd.edu` by December 11 at 11AM.
- The second part of the final exam consists in a quiz. The questions in the quiz are going to be related to this part. The only material you are allowed to use is your assignment.
- At the end of the class of Dec 11, you have to hand in both your assignment and your quiz.
- You don't need to write every piece of code from scratch. You may use any code from the book's toolbox or from the course page. However, you should understand the algorithms behind these codes.

---

1. (15 pts) The first derivative of a function f at a point a can be approximated by the difference quotient

$$d_h f(a) = \frac{f(a+h) - f(a-h)}{2h}$$

(a) Use a Taylor expansion to show that the error $e(h) = f'(a) - d_h f(a)$ in this approximation can be bounded by

$$|e(h)| \leq \frac{M}{6} h^2, \tag{1}$$

where M is a bound on the 3rd derivative of f.

(b) Write a MATLAB function `function d = derivative(f,a,n)` that approximates the first derivative of the function `f` with the above expression for $h = 10^{-k}$ with $k = 1 : n$. The output `d` should be a vector of length n.

(c) Use this function to approximate the 1st derivative of $f(x) = \cos(6x)$ at $a = 1$ with $n = 16$. Print the vector d and the actual errors $|e(h)|$.

(d) Explain why the error does not behave as expected. To this end, use (1) and account for round-off error. Indeed, we are evaluating

$$\overline{d_h} f(a) = \frac{\overline{f}(a+h) - \overline{f}(a-h)}{h}$$

for small h, where $\overline{f}(a+h)$ and $\overline{f}(a-h)$ are the computed values of $f(a+h)$ and $f(a-h)$, respectively. Assume the computed function value $\overline{f}(z)$ is computed according to

$$\overline{f}(z) = \mathtt{fl}(f(z)),$$

where $\mathtt{fl}(f(z))$ is the floating point representation of $f(z)$. Using this assumption, show that

$$|d_h f(a) - \overline{d_h} f(a)| \leq |f'(a)| \frac{\mathtt{eps}}{h} \quad \text{(the inequality here is an approximate inequality)},$$

and find a bound for the error

$$|f'(a) - \overline{d_h} f(a)|.$$

What is the order (with respect to eps) of the optimal choice for h? What is the order of the absolute error using the optimal h? Compare your estimates with your results from part (c).

---

2. (15 pts) *Closed Newton-Cotes formulas.* To integrate a function f over an interval $[a, b]$, the Newton-Cotes approach is based on evaluation of the integrand at equally spaced points. In particular, the *closed* Newton-Cotes formulas include evaluation at the endpoints a and b. So, this method consists in dividing $[a, b]$ into n equal subintervals and then to estimate $\int_a^b f$ by the integral of the interpolating polynomial of degree n. Namely, defining $x_k = a + kh$ for $k = 0, \ldots, n$, and $h = (b-a)/n$, the closed Newton-Cotes formula of degree n is given by

$$Q_n^{NC}[f] = \int_a^b p_n(x) dx,$$

where p_n is the polynomial interpolating f at the nodes $\{x_k\}_{k=0,\ldots,n}$.

For example, the closed Newton-Cotes formula of degree 1 is the trapezoid rule, and the closed formula of degree 2 is Simpson's rule[1].

(a) The interpolating polynomial p_n can be written as

$$p_n(x) = c_1 x^n + c_2 x^{n-1} + \ldots + c_n x + c_{n+1}.$$

[1] Moreover, the closed Newton-Cotes formula of degree 3 and 4 are usually called *Simpson's 3/8 rule* and *Boole's rule*, respectively.

Write a Matlab function `c = polynomial(f,a,b,n)` that computes the coefficients $\{c_n\}$ of the interpolating polynomial. You may want to use the Matlab command `vander` for this purpose.

(b) Noticing that, for all $k = 0, \ldots, n$ it holds that

$$\int_a^b x^k dx = \frac{b^{k+1} - a^{k+1}}{k+1},$$

and using the function from part (a), write a Matlab program `q = NewtonCotes(f,a,b,n)` that, taking as inputs a function `f`, the endpoints of the interval `a`, `b` and the degree `n`, gives as an output `q`= $Q_n^{NC}[f]$.

(c) Test your code computing the errors for $\int_0^\pi x \sin(x) dx$ using `n`= $1, \ldots, 10$.

---

3. (15 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.
(Hint: you may use as a starting point the code we displayed in class on November 8)

(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 $t = 20$.

---

4. (15 pts)[2] The function $y(x)$ is defined on the interval $0 \le x \le 1$ by

$$y'' = y^2 - 1,$$
$$y(0) = 0,$$
$$y(1) = 1.$$

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

$$y'' = y^2 - 1,$$
$$y(0) = 0,$$
$$y'(0) = \eta,$$

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

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

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.

(c and d) Nonlinear finite differences. Partition the interval into $n+1$ equal subintervals with spacing $h = 1/(n+1)$:

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

Replace the differential equation with a nonlinear system of difference equations involving n unknowns, $y_1, y_2, \ldots, y_n$:

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

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.

(c) 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.

(d) 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.