Introducción al Cálculo Científico - 2002

Trabajo Práctico Nro. 2 - Fecha de entrega: martes 22/10

Para los siguientes problemas se requiere trabajo de *lápiz y papel*, escribir algunos scripts y funciones de MATLAB/OCTAVE, y ejecutar algunas instrucciones en MATLAB/OCTAVE. Presentar de manera concisa lo realizado y los resultados obtenidos. Incluir en la presentacion los scripts y funciones.

1. Dados tres puntos $x_1 < x_2 < x_3$ y una función $f \in C^4([x_1, x_3])$, sea q el polinomio cuadrático que interpola f en los nodos x_1, x_2, x_3.

(a) Determinar la forma de Newton de q y probar que

$$q'(x_2) = f[x_3, x_2]\frac{x_2 - x_1}{x_3 - x_1} + f[x_2, x_1]\frac{x_3 - x_2}{x_3 - x_1}$$
$$q''(x_2) = 2f[x_1, x_2, x_3].$$

Aquí los términos $f[z_1]$, $f[z_1, z_2]$, $f[z_1, z_2, z_3]$ denotan las *diferencias divididas* de f y se definen recursivamente del siguiente modo (ver §2.4.1 del libro)

$$f[z_1] = f(z_1)$$
$$f[z_1, z_2, \ldots, z_k] = \frac{f[z_2, \ldots, z_k] - f[z_1, z_2, \ldots, z_{k-1}]}{z_k - z_1}$$

Se puede ver que si p_{n-1} es el polinomio que interpola a una función f en $x_1, x_2, \ldots, x_n$, entonces

$$p_{n-1}(x) = f[x_1] + f[x_1, x_2](x - x_1) + f[x_1, x_2, x_3](x - x_1)(x - x_2) + \cdots$$
$$+ f[x_1, x_2, \ldots, x_n](x - x_1)(x - x_2)\cdots(x - x_{n-1}),$$

es decir, los coeficientes a_i de la representación de Newton son

$$a_1 = f[x_1] \qquad a_2 = f[x_1, x_2] \qquad \cdots \qquad a_n = f[x_1, x_2, \ldots, x_n].$$

(b) Escribir expresiones explícitas para $q'(x_2)$ y $q''(x_2)$ en términos de $f(x_1)$, $f(x_2)$, $f(x_3)$. Estas expresiones se conocen con el nombre de aproximaciones por *diferencias finitas* de $f'(x_2)$ y $f''(x_2)$.

(c) Utilizando expansiones en serie de Taylor alrededor de $x = x_2$, y la forma explícita del punto anterior para $q'(x_2)$ y $q''(x_2)$, mostrar las estimaciones de error

$$|f'(x_2) - q'(x_2)| \leq Ch^2 \max_{x_1 \leq x \leq x_3} |f'''(x)|$$
$$|f''(x_2) - q''(x_2)| \leq Ch \max_{x_1 \leq x \leq x_3} |f'''(x)|.$$

Aquí h denota el máximo entre $x_2 - x_1$ y $x_3 - x_2$.

(d) Considerar el caso de nodos *equiespaciados*, es decir $x_2 - x_1 = x_3 - x_2 = h$, y calcular $q'(x_2)$ y $q''(x_2)$. Estas expresiones se llaman diferencias finitas *centradas*.

(e) Utilizar la expansión de Taylor alrededor de $x = x_2$ (sigamos considerando nodos equiespaciados), pero esta vez con más términos, para derivar la cota para el error mejorada

$$|f''(x_2) - q''(x_2)| \leq Ch^2 \max_{x_1 \leq x \leq x_3} |f^{(4)}(x)|.$$

2. Escribir un *script* para verificar el funcionamiento de `pwLadapt` con la función $f(x) = \sqrt{x}$ en el intervalo $[0, 1]$. Específicamente, para cualquier valor de los parámetros δ y h_{min}, el script debería:

(a) Utilizar `pwLadapt`, `pwL`, y `pwLeval` para hallar y evaluar la interpolante lineal a trozos adaptiva $L(x)$.

(b) Graficar $f(x)$, $L(x)$ y los puntos de interpolación x, y en el mismo dibujo

(c) Encontrar el máximo error e_i entre $L(x)$ y $f(x)$ en cada subintervalo (x_i, x_{i+1}). Para ello, demostrar que el error máximo e_i en el intervalo (x_i, x_{i+1}) está dado por la expresión

$$e_i = \frac{(\sqrt{x_{i+1}} - \sqrt{x_i})^2}{4(\sqrt{x_{i+1}} + \sqrt{x_i})}.$$

Ejecutar el script varias veces para diferentes valores de δ y h_{min}. Hacer algún comentario acerca de la distribución de abcisas de los puntos de la grilla obtenida (para ello seguir los lineamientos del problema 3.1.7 del libro.)

¿Cómo se comparan los valores de e_i y de δ? Recordemos que `pwLadapt` fue diseñado para *"asegurar"* que $e_i \leq \delta$ para todo i. ¿Es esto siempre cierto? Explicar por qué.

3. Completar la siguiente función `CubicSpline.m`

```
function [a, b, c, d] = CubicSpline( x, y, derivada, muL, muR)
% Entrada:
%   x, y: n-vectores columnas, n >= 4 y x(1) < x(2) < ... < x(n)
%   derivada: el orden de la derivada que se utiliza en los extremos (1 ó 2)
%   muL, muR: valores de la derivada primera (segunda) en x(1) y en x(n)
% Salida:
%   a, b, c, d:  (n-1)-vectores columna que definen el spline.
%      En [x(i),x(i+1)] el spline S(z) esta dado por
%      a(i) + b(i) (z-x(i)) + c(i) (z-x(i))^2 + d(i) (z-x(i))^2 (z-x(i+1))
% Utilización:
%   [a, b, c, d] = CubicSpline(x, y, 1, muL, muR)
%         calcula el spline que tiene _primera_ derivada igual a
%         muL  y  muR  en  x(1)  y  x(n)  respectivamente
%   [a, b, c, d] = CubicSpline(x, y, 2, muL, muR)
%         calcula el spline que tiene _segunda_ derivada igual a
%         muL  y  muR  en  x(1)  y  x(n)  respectivamente
%   [a, b, c, d] = CubicSpline(x, y)
%         calcula el spline "not-a-knot", es decir aquél que tiene
%         derivada _tercera_ continua en x(2) y x(n-1)
```

4. Sean `x = 0:8` e `y = [1 1 1 d 1 1 1 1 1]`, puntos de interpolación donde d toma valores cercanos a 1, digamos `d = 0.5, 1, 1.5`.

(a) Encontrar, utilizando las funciones diseñadas para interpolar (Capítulo 2), el polinomio de grado ≤ 8 que interpola los puntos dados.

(b) Encontrar, utilizando la función `CubicSpline` del ejercicio anterior, el spline *not-a-knot* que interpola los datos dados.

(c) Graficar los resultados obtenidos. ¿Cuál método es menos sensible a perturbaciones del valor de y en un solo punto? ¿Qué método (spline cúbico o polinomio de grado superior) será preferible para interpolar datos experimentales?

5. En tipografía computada existe el problema de encontrar una interpolante suave a puntos que se encuentran sobre una curva del plano. Esta curva no necesariamente puede ser representada por una función de la variable x porque puede haber varios puntos con la misma abcisa x (ejemplo: una curva con forma de S). Una forma de encarar el problema es la de numerar los puntos $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ a medida que recorremos la curva. Sea d_i la distancia entre (x_i, y_i) y (x_{i+1}, y_{i+1}), $i = 1 : n-1$. Sea $t_i = d_1 + d_2 + \cdots + d_{i-1}$, $i = 2 : n-1$ ($t_1 = 0$). Supongamos que $S_x(t)$ es un spline interpolante de $(t_1, x_1), \ldots (t_n, x_n)$ y que $S_y(t)$ es un spline interpolante de $(t_1, y_1), \ldots (t_n, y_n)$. Luego la curva $\Lambda = \{(S_x(t), S_y(t)) : t_1 \leq t \leq t_n\}$ es suave y pasa por los n puntos dados. Escribir una función `[xi, yi] = SplineInPlane(x, y, m)` que devuelve en `xi(1:m)`, `yi(1:m)` las coordenadas x-y de m puntos que se encuentran en la curva Λ. Probar esta función con algún ejemplo.