Lista 3

Introdução à Análise Numérica
Interpolação

Lucas Emanuel Resck Domingues

Outubro de 2020

2. Vamos utilizar o método das diferenças de Newton, encontrar os coeficientes c_n e finalmente encontrar $P(k)$.

   P é um polinômio de grau n, e sabemos que ele passa pelos pontos

$$(1, -1), (2, -2), \cdots, (n, -n), (0, (-1)^n)$$

   sendo o último ponto justificado por $P(0) = (-1)^n$ (termo independente). Então o polinômio escrito na forma do método das diferenças fica

$$\begin{aligned} P(x) &= c_0 + c_1(x - x_1) + \cdots + c_n(x - x_1) \cdots (x - x_n) \\ &= c_0 + c_1(x - 1) + \cdots + c_n(x - 1) \cdots (x - n) \end{aligned}$$

   A Tabela 1 mostra o cálculo dos coeficientes c_i pelo método das diferenças de Newton.

   Dessa forma, nós concluímos que $c_0 = c_1 = -1$ e que $c_2 = \cdots = c_{n-1} = 0$. Apenas nos resta encontrar c_n, que, seguindo a tabela, será uma função de n. Até agora, temos

$$P(x) = -1 - 1(x - 1) + c_n(x - 1) \cdots (x - n)$$

   Sabendo que $P(0) = (-1)^n$, vemos que

$$\begin{aligned} P(0) &= (-1)^n \\ c_n(-1) \cdots (-n) &= (-1)^n \\ c_n(-1)^n n! &= (-1)^n \\ c_n &= \frac{1}{n!} \end{aligned}$$

x	$y = \Delta^1$	Δ^2	Δ^3	Δ^4	
1	$\mathbf{-1}$				
		$\mathbf{-1}$			
2	-2		$\mathbf{0}$		
		-1		$\mathbf{0}$	
3	-3		0		
		-1		0	
4	-4		0		
		-1		0	$\cdots$
5	-5		0		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$n-1$	$-(n-1)$		0		
		-1		$C(n)$	
n	$-n$		$B(n)$		
		$A(n)$			
0	$(-1)^n$				

Tabela 1: Método das diferenças de Newton para o polinômio P. Os números em negrito são os coeficientes c_i. Os números $A(n), \cdots$ são funções de n, não necessariamente 0.

Tendo todos os coeficientes, podemos calcular $P(k)$, k natural e maior que n:

$$
\begin{aligned}
P(k) &= -1 - 1(k-1) + \frac{1}{n!}(k-1)\cdots(k-n) \\
&= -k + \frac{(k-1)!}{n!(k-1-n)!} \\
&= \binom{k-1}{n} - k
\end{aligned}
$$

8. Vamos montar os polinômios interpoladores p_{k-2} e p_{k-1} de graus $k-2$ e $k-1$, respectivamente, através do método das diferenças de Newton. Depois vamos encontrar o coeficiente c_{k-1}, que corresponde a $\Delta[x_1, \cdots, x_k]$. Ao final, vamos calcular o limite pedido.

   Seja o polinômio interpolador p_{k-2} de f com os pontos $\{x_1, \cdots, x_{k-1}\}$ através dos métodos das diferenças de Newton:

$$p_{k-2}(x) = c_0 + \cdots + c_{k-2}(x - x_1)\cdots(x - x_{k-2})$$

   Uma das principais características do método de Newton é que podemos "reutilizar" o polinômio ao adicionar mais pontos. Se quisermos interpolar $\{x_1, \cdots, x_k\}$, basta fazermos

$$p_{k-1}(x) = c_0 + \cdots + c_{k-2}(x - x_1)\cdots(x - x_{k-2}) + c_{k-1}(x - x_1)\cdots(x - x_{k-1})$$

   Observe que c_{k-1} corresponde a $\Delta[x_1, \cdots, x_k]$, que é o que queremos. Segue:

$$\begin{aligned} p_{k-1}(x) - p_{k-2}(x) &= c_{k-1}(x - x_1)\cdots(x - x_{k-1}) \\ (f(x) - p_{k-2}(x)) - (f(x) - p_{k-1}(x)) &= c_{k-1}(x - x_1)\cdots(x - x_{k-1}) \end{aligned} \quad (1)$$

   Seja $A = [\min\{x_1, \cdots, x_k\}, \max\{x_1, \cdots, x_k\}]$. Por conveniência, assumindo que $\{x_1, \cdots, x_k\}$ são diferentes e sabendo que $f(x) = \sin(cx + d)$ é de classe C^∞ no intervalo A, sabemos, por teorema, que, para cada $x \in A$, existem α_x, β_x tais que

$$\begin{aligned} f(x) - p_{k-2}(x) &= \frac{f^{(k-1)}(\alpha_x)}{(k-1)!}(x - x_1)\cdots(x - x_{k-1}) \\ f(x) - p_{k-1}(x) &= \frac{f^{(k)}(\beta_x)}{k!}(x - x_1)\cdots(x - x_k) \end{aligned}$$

   Então, se avaliarmos essas funções em x_k e substituirmos na Equação 1, obtemos

$$\begin{aligned} \frac{f^{(k-1)}(\alpha_{x_k})}{(k-1)!}(x_k - x_1)\cdots(x_k - x_{k-1}) &= c_{k-1}(x_k - x_1)\cdots(x_k - x_{k-1}) \\ \frac{f^{(k-1)}(\alpha_{x_k})}{(k-1)!} &= c_{k-1} \end{aligned}$$

   Segue portanto:

$$
\begin{aligned}
|c_{k-1}| &= \left| \frac{f^{(k-1)}(\alpha_{x_k})}{(k-1)!} \right| \\
&= \frac{|c^{k-1}\sin(c\alpha_{x_k}+d)|}{(k-1)!} \text{ ou } \frac{|c^{k-1}\cos(c\alpha_{x_k}+d)|}{(k-1)!} \\
&\leq \frac{|c|^{k-1}}{(k-1)!}
\end{aligned}
$$

Basta agora mostrar que isso tende a zero quando k tende ao infinito. Ou seja, vamos mostrar que, se $a \geq 0$,

$$
\lim_{k\to\infty} \frac{a^k}{k!} = 0
$$

Seja a sequência $y_k = \dfrac{a^k}{k!}$. Considere o termo y_{2a+1}. Então:

$$
y_{2a+1} = \frac{a^{2a+1}}{(2a+1)!} = \frac{a^{2a}}{(2a)!}\frac{a}{2a+1} < \frac{a^{2a}}{(2a)!}\frac{1}{2} = \frac{1}{2}y_{2a}
$$

Concluímos que, para $k \in \mathbb{N}^*$, $y_{2a+k} < \left(\dfrac{1}{2}\right)^k y_{2a}$. Logo:

$$
\lim_{k\to\infty} \frac{a^k}{k!} = \lim_{k\to\infty} y_k = \lim_{k\to\infty} y_{2a+k} \leq \lim_{k\to\infty} \left(\frac{1}{2}\right)^k y_{2a} = 0
$$

Portanto, $c_{k-1} \to 0$, ou seja, $\Delta[x_1, \cdots, x_k] \to 0$

12. (a) Sejam os pontos $(x_0, y_0), \cdots, (x_n, y_n)$, enumerados a partir do zero por conveniência. Ou seja, $n+1$ pontos. Os pontos guia são $(\hat{x}_0, \hat{y}_0)$, $(\hat{x}_n, \hat{y}_n)$ Vamos asssumir intervalos de comprimento $h_i = 1/n$, isso é, teremos $t_0 < \cdots < t_n$, com $t_i = \dfrac{i}{n}$ (a conveniência). Portanto, vamos interpolar os $n+1$ pontos com a curva $\gamma(t) = (x(t), y(t))$, $t \in [0, 1]$. Vamos interpolar cada intervalo $[t_i, t_{i+1}]$, para $i = 0, \cdots, n-1$, através de um polinômio de grau 3. Vamos listar as condições que queremos para nossa curva γ:

- Passar pelos pontos: $\gamma(t_i) = (x(t_i), y(t_i)) = (x_i, y_i)$, $i = 0, \cdots, n$
- Ancoragem 1: $\gamma'(0) = (x'(0), y'(0)) = k_0(\hat{x}_0 - x_0, \hat{y}_0 - y_0)$
- Ancoragem 2: $\gamma'(1) = (x'(1), y'(1)) = k_1(\hat{x}_n - x_n, \hat{y}_n - y_n)$
- Continuidade de γ em t_i, $i = 1, \cdots, n-1$
- Continuidade de γ' em t_i, $i = 1, \cdots, n-1$
- Continuidade de γ'' em t_i, $i = 1, \cdots, n-1$

Observe que nada impede de aplicarmos as condições de γ separadamente para cada componente x e y. Consideremos x. Então, aplicando todas essas condições separadamente para x:

- Passar pelos pontos: $x(t_i) = x_i$, $i = 0, \cdots, n$
- Ancoragem 1: $x'(0) = k_0(\hat{x}_0 - x_0)$
- Ancoragem 2: $x'(1) = k_1(\hat{x}_n - x_n)$
- Continuidade de x em t_i, $i = 1, \cdots, n-1$
- Continuidade de x' em t_i, $i = 1, \cdots, n-1$
- Continuidade de x'' em t_i, $i = 1, \cdots, n-1$

Ora, estamos pedindo uma interpolação por uma *spline* cúbica ancorada para x, afinal os valores de t_i são crescentes. Isso já foi estudado. Então sabemos que, se $t \in [t_i, t_{i+1}]$,

$$
\begin{aligned}
x(t) &= \frac{M_i}{6h_i}(t_{i+1} - t)^3 + \frac{M_{i+1}}{6h_i}(t - t_i)^3 + c_1 t + c_2 \\
&= \frac{nM_i}{6}(t_{i+1} - t)^3 + \frac{nM_{i+1}}{6}(t - t_i)^3 + c_1 t + c_2
\end{aligned}
$$

sendo

$$
\begin{aligned}
c_1 &= \frac{x_{i+1} - x_i}{h_i} - \frac{h_i}{6}(M_{i+1} - M_i) \\
&= n(x_{i+1} - x_i) - \frac{1}{6n}(M_{i+1} - M_i)
\end{aligned}
$$

$$\begin{aligned} c_2 &= x_i - \frac{M_i h_i^2}{6} - c_1 t_i \\ &= x_i - \frac{M_i}{6n^2} - c_1 t_i \end{aligned}$$

Os valores de M_i, por sua vez, são encontrados através das equações

$$\frac{1}{6n}M_{i-1} + \frac{2}{3n}M_i + \frac{1}{6n}M_{i+1} = n(x_{i+1} - 2x_i + x_{i-1})$$

para $i = 1, \cdots, n-1$. Essas equações são encontradas aplicando as condições dadas acima, com exceção das ancoragens. Como isso foi feito em sala, é conveniente omitir essa dedução. Porém, essas equações são em quantidade $n-1$, e temos $n+1$ variáveis M_i para descobrir: $M_0, \cdots, M_n$.

Para resolver isso, podemos aplicar as condições de ancoragem em $t_0 = 0$ e $t_n = 1$. Basta resolver $x'(0) = k_0(\hat{x}_0 - x_0)$. Vejamos. Para $t \in [0, 1/n]$, vale que:

$$x'(t) = -\frac{M_0}{2h_0}(t_1 - t)^2 + \frac{M_1}{2h_0}(t - t_0)^2 + \frac{x_1 - x_0}{h_0} - \frac{h_0}{6}(M_1 - M_0)$$

Então:

$$\begin{aligned} x'(0) &= k_0(\hat{x}_0 - x_0) \\ -\frac{M_0}{3n} + n(x_1 - x_0) - \frac{M_1}{6n} &= k_0(\hat{x}_0 - x_0) \\ \frac{1}{3n}M_0 + \frac{1}{6n}M_1 &= -k_0(\hat{x}_0 - x_0) + n(x_1 - x_0) \end{aligned}$$

De forma análoga:

$$\frac{1}{6n}M_{n-1} + \frac{1}{3n}M_n = k_1(\hat{x}_n - x_n) - n(x_n - x_{n-1})$$

Encontramos as duas equações que estavam faltando. Sendo assim, podemos montar o seguinte sistema de equações que deve ser resolvido:

$$Ax = b$$

sendo

$$A = \begin{bmatrix} \frac{1}{3n} & \frac{1}{6n} & & & & & \\ \frac{1}{6n} & \frac{2}{3n} & \frac{1}{6n} & & & & \\ & \frac{1}{6n} & \frac{2}{3n} & \frac{1}{6n} & & & \\ & & & & \ddots & & & \\ & & & & & \frac{1}{6n} & \frac{2}{3n} & \frac{1}{6n} \\ & & & & & & \frac{1}{6n} & \frac{1}{3n} \end{bmatrix}$$

$$x = \begin{bmatrix} M_0 \\ \vdots \\ M_n \end{bmatrix}$$

$$b = \begin{bmatrix} -k_0(\hat{x}_0 - x_0) + n(x_1 - x_0) \\ n(x_2 - 2x_1 + x_0) \\ \vdots \\ n(x_n - 2x_{n-1} + x_{n-2}) \\ k_1(\hat{x}_n - x_n) - n(x_n - x_{n-1}) \end{bmatrix}$$

Ou seja, conseguindo todos os M_i, a partir de um algoritmo de resolução de sistemas lineares (observe que o sistema é diagonalmente dominante, perfeito para vários métodos estudados em aula), calculamos c_1 e c_2 e podemos expressar $x(t)$.

O processo para y é totalmente análogo.

Ao final, temos $\gamma(t) = (x(t), y(t))$.

(b) O programa foi implementado em Matlab e pode ser conferido no Apêndice A. A utilização do programa se dá da seguinte forma: vamos montar k *splines* cúblicas paramétricas, cada *spline* emendada à anterior. Cada *spline* será composta de $n+3$ pontos, sendo $n+1$ os pontos de interpolação $(x_0, y_0), \cdots, (x_n, y_n)$ e 2 pontos de ancoragem $(\hat{x}_0, \hat{y}_0), (\hat{x}_n, \hat{y}_n)$. Iniciamos a função com o comando

interactive_spline(n, k, k_0, k_1)

sendo k_0, k_1 relacionados à ancoragem, já vistos anteriormente. Sendo assim, desenha-se os pontos, na ordem

$$(x_0, y_0), (\hat{x}_0, \hat{y}_0), (x_1, y_1), \cdots, (x_n, y_n), (\hat{x}_n, \hat{y}_n)$$

um total de k vezes.

O programa lê os pontos selecionados e constrói, para cada intervalo $[t_i, t_{i+1}]$, o polinômio interpolador, tanto para x quanto para y. O sistema de equações lineares é resolvido utilizando o método de Seidel (já implementado em exercícios anteriores deste curso), afinal a matriz é diagonalmente dominante. Os dois polinômios resultantes em cada pequeno intervalo são calculados e exibidos graficamente para vários pontos desse pequeno intervalo, dando a ideia de continuidade. As Figuras 1, 2 e 3 mostram exemplos do resultado da utilização do programa.

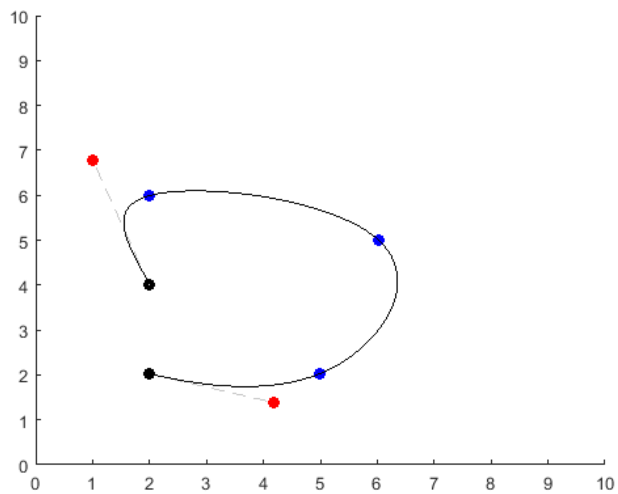

Figura 1: Interpolação por *spline* cúbica paramétrica para 5 pontos de interpolação e 2 pontos de ancoragem.

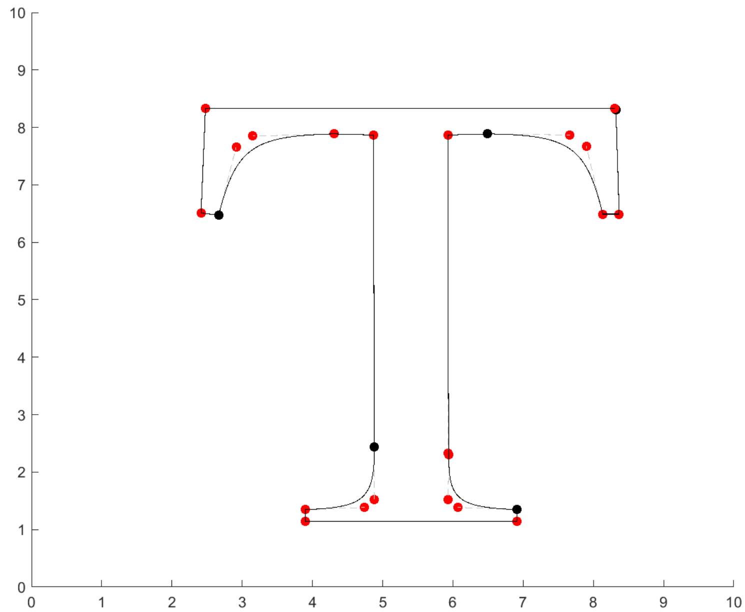

Figura 2: Interpolação por várias *splines* cúbicas paramétricas para 2 pontos de interpolação e 2 pontos de ancoragem.

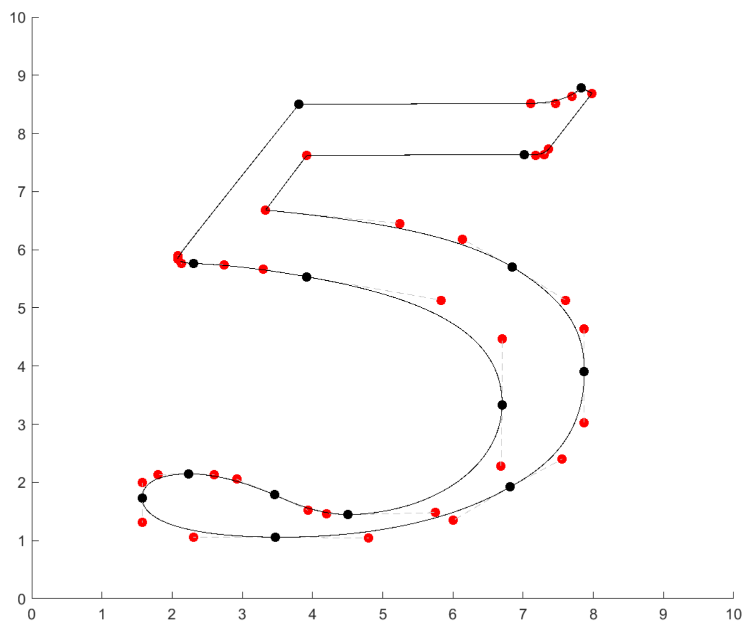

Figura 3: Interpolação por várias *splines* cúbicas paramétricas para 2 pontos de interpolação e 2 pontos de ancoragem.

A Programa computacional para *splines* interativos

Implementação do algoritmo em Matlab para construção de *splines* cúbicos paramétricos interativos. O código também pode ser acessado neste link (pasta em um repositório do GitHub). As funções nessa pasta possuem os mesmos nomes daquelas apresentadas abaixo.

```
function interactive_spline(n, k, k_0, k_1)
    clf;
    axis([0, 10, 0, 10]);
    hold on;
    [x_click, y_click] = ginput(1);
    x = [x_click];
    y = [y_click];
    plot(x, y, 'o', ...
        'MarkerEdgeColor', 'k', ...
        'MarkerFaceColor', 'k');
    for j = 1:k
        for i = 1:n+2
            [x_click, y_click] = ginput(1);
            x = [x; x_click];
            y = [y; y_click];
            if i == 1 || i == n+2
                plot(x_click, y_click, 'o', ...
                'MarkerEdgeColor', 'r', ...
                'MarkerFaceColor', 'r');
                plot(x(end-1:end), y(end-1:end), '--', ...
                    'Color', [200, 200, 200]/255);
            else
                if i == n+1
                    plot(x_click, y_click, 'o', ...
                    'MarkerEdgeColor', 'k', ...
                    'MarkerFaceColor', 'k');
                else
                    plot(x_click, y_click, 'o', ...
                    'MarkerEdgeColor', 'b', ...
                    'MarkerFaceColor', 'b');
                end
            end
        end
        parametric_cubic_spline(x, y, k_0, k_1);
        x = [x(end-1)];
        y = [y(end-1)];
    end
    hold off;
```

```
end

function parametric_cubic_spline(x, y, k_0, k_1)
    x_t = anchored_cubic_spline(x, k_0, k_1);
    y_t = anchored_cubic_spline(y, k_0, k_1);
    plot(x_t, y_t, '-k');
end

function x_t = anchored_cubic_spline(x, k_0, k_1)
    x_hat_0 = x(2);
    x_hat_n = x(end);
    x(end) = [];
    x(2) = [];
    [n, ~] = size(x);
    n = n-1;
    h = 1/n;
    A = zeros(n+1, n+1);
    b = zeros(n+1, 1);
    for i=0:n
        if i == 0
            j = i+1;
            A(j, j) = 1/(3*n);
            A(j, j+1) = 1/(6*n);
            b(j) = -k_0*(x_hat_0 - x(j)) + n*(x(j+1) - x(j));
        else
            if i == n
                j = i+1;
                A(j, j-1) = 1/(6*n);
                A(j, j) = 1/(3*n);
                b(j) = k_1*(x_hat_n - x(j)) - n*(x(j) - x(j-1));
            else
                j = i+1;
                A(j, j-1) = 1/(6*n);
                A(j, j) = 2/(3*n);
                A(j, j+1) = 1/(6*n);
                b(j) = n*(x(j+1) - 2*x(j) + x(j-1));
            end
        end

    end
    M = seidel(A, b, 10^-6);
    x_t = [];
    for i=0:(n-1)
        p = interpolator(M(i+1), M(i+2), i/n, (i+1)/n, x(i+1), x(i+2), n);
        x_t = [x_t, p(i/n:0.01:(i+1)/n)];
    end
```

```
end

function x = seidel(A, b, eps)
    [m, ~] = size(A);
    x = zeros(m, 1);
    err = 1;
    x_old = x;
    while err > eps
        for i=1:m
            x(i) = A(i, 1:(i-1))*x(1:(i-1)) + A(i, (i+1):m)*x((i+1):m);
            x(i) = -x(i);
            x(i) = x(i) + b(i);
            x(i) = x(i)/A(i, i);
        end
        err = norm(x - x_old, 'inf')/norm(x_old, 'inf');
        x_old = x;
    end
end

function x_t = interpolator(M_i, M_ip, t_i, t_ip, x_i, x_ip, n)
    c_1 = (x_ip-x_i)*n - (M_ip - M_i)/(6*n);
    c_2 = x_i - M_i/(6*n^2) - c_1*t_i;
    syms x_t(t);
    x_t(t) = n*M_i/6*(t_ip-t)^3 + n*M_ip/6*(t-t_i)^3 + c_1*t + c_2;
end
```