

2º Exercício Programa - Interpolação por splines cúbicos

Antonio Elias Fabris & André Chalom

2 de novembro de 2009

1 Introdução

Este exercício-programa consiste na implementação de algoritmos de interpolação por polinômios por partes. Escolhemos uma classe de funções interpoladoras amplamente utilizadas nas aplicações práticas, os splines cúbicos.

Como vimos nas notas de aulas, os splines cúbicos são funções $f \in C^2[a, b]$.

A estimativa de erros é crucial neste tipo de problema e necessariamente deve ser implementada. Vamos utilizar uma estimativa de erros prática para tentar descobrir o expoente de decaimento do erro.

Usamos para teste o problema da interpolação em pontos equiespaçados da função do exemplo de Rünge no intervalo $[-1, 1]$. Relembramos que essa era a situação em que a interpolação polinomial produzia resultados desastrosos conforme o grau dos polinômios de interpolação aumentava (veja teorema de Rünge nas notas de aulas).

Agora, mantemos constante a ordem dos polinômios em cada parte e aumentamos o número de pontos da partição τ (ou equivalentemente aumentamos a quantidade de polinômios de mesma ordem) para estudar a qualidade da aproximação dos polinômios por trechos. Tal qualidade de aproximação será avaliada através da análise dos erros máximo de interpolação conforme aumenta-se o número de nós da partição τ .

Para construir o spline cúbico interpolador, precisamos escolher a condição de contorno para ambas as extremidades. Duas escolhas possíveis quando não temos informações sobre as derivadas da função no extremo são os splines naturais (onde se impõe $f''(\tau_0) = f''(\tau_{n+1}) = 0$) e os splines not-a-knot, que produz resultados mais precisos próximos às extremidades. Neste EP, você precisará escolher uma dessas condições para implementar. Você só precisa implementar *uma* das condições.

Finalmente, o programa pode ainda produzir gráficos que ajudem a visualizar em conjunto com as estimativas de erros e análises feitas anteriormente a visualização deste método de interpolação e estimulem a discussão crítica desta técnica.

Não exigiremos tal análise neste EP. Entretanto, as tentativas serão estimuladas e bem-vindas. Tal comparação deve procurar responder questões sobre

quando em aplicações práticas deveremos usar este método ou procurar outro. Por exemplo, sabe-se que os splines cúbicos são amplamente utilizados na indústria da gestão de risco para interpolação das ETTJ (estruturas a termo das taxas de juros). Será que funções mais simples (como funções $C^1[a, b]$) não resolveriam o problema de modo tão eficiente quanto os splines cúbicos? Noutras palavras, onde usamos na interpolação das ETTJ o fato da função que interpola ser $C^2[a, b]$? Este é apenas um exemplo de análise e comparação crítica dos métodos. Outros exemplos de aplicações devem ser procurados e questionados quanto ao seu correto uso.

Você receberá um arquivo fonte em C contendo uma função principal e os protótipos das funções pedidas. Complete o programa e entregue o código fonte (extensão .c) e um relatório (extensão .doc, .pdf, etc.) dentro de um arquivo compactado (extensão .zip).

2 Tarefa

1. **Implementar** os procedimentos GaussTridiag, GeraSpline e ValorEmX. Cada um deles é discutido em detalhes na próxima seção.
2. **Erros de entrada:** A função main é responsável por detectar alguns erros de entrada de dados e enviar mensagens apropriadas. Por exemplo, deve se garantir que o número de pontos tabelados é no mínimo 4. Se o sistema montado for de resolução impossível, isso deve ser detectado pela função GaussTridiag.
3. **Teste do Programa:** Interpolar a função de Rünge

$$G(x) = \frac{1}{1 + 25x^2}, \quad x \in [-1, 1] \quad (1)$$

Usando os pontos

$$\tau_i = -1 + \frac{2i}{N}, \quad i = 0, \dots, N \quad (2)$$

4. **Determinar** o erro de interpolação para um ponto fixado $x \notin \tau_i$, e o coeficiente de decaimento do erro α . Construa a seguinte tabela para o erro de interpolação:

N	Erro	Expoente
4		-
8		
16		
...		

5. **Relatório:** escreva um relatório contendo uma breve introdução teórica, uma descrição das funções usadas (não o código fonte), os resultados que

você encontrou e uma discussão a respeito. Lembre-se de explicar qual condição de contorno você utilizou, porquê tomou essa decisão e quais foram as consequências.

6. BÔNUS: Escrever um procedimento que produza o gráfico conjunto da função G com a interpolação produzida.

3 Funções

As seguintes funções devem ser implementadas obrigatoriamente mantendo o protótipo abaixo. As variáveis usadas internamente por cada função podem ser decididas por você, assim como a necessidade de escrever outras funções auxiliares.

Note que os trechos de código já escritos dentro das funções são *sugestões*, de forma que eles podem ser alterados. Você também pode alterar a função `main`, desde que não comprometa a funcionalidade já existente.

1. `void GeraSpline(double tau[], double y[], double m[], int n)`

Entrada:	tau	Vetor ordenado de modo crescente representando as abcissas da tabela dada.
	y	Armazena os valores da função tabela em tau[i]
	n	Quantidade de polinômios ("tamanho menos um do vetor tau")
Saída:	m	Armazena os valores das inclinações que determinam o spline cúbico

A função `GeraSpline` deve montar o sistema linear tridiagonal A e resolvê-lo usando a função `GaussTridiag`. Perceba que o sistema será diferente dependendo da sua escolha para a condição de contorno.

Os dados a serem interpolados (tabela de dados) junto com as inclinações $S(N+1)$, saída de `GeraSpline`, definem o spline. Posteriormente, deve-se chamar `ValorEmX` para calcular o spline num ponto arbitrário X . De acordo com a fórmula (12) das notas de aulas temos que, para $x \in [\tau_i, \tau_{i+1}]$:

$$p_i(x) = C(1, i) + C(2, i)(x - \tau_i) + C(3, i)(x - \tau_i)^2 + C(4, i)(x - \tau_i)^3 \quad (3)$$

$$p'_i(x) = C(2, i) + 2C(3, i)(x - \tau_i) + 3C(4, i)(x - \tau_i)^2 \quad (4)$$

Fatorando, obtemos expressões mais econômicas, que são aquelas que deverão ser implementadas:

$$p_i(x) = C(1, i) + (x - \tau_i)(C(2, i) + (x - \tau_i)(C(3, i) + C(4, i)(x - \tau_i))) \quad (5)$$

$$p'_i(x) = C(2, i) + (x - \tau_i)(2 * C(3, i) + 3C(4, i) * (x - \tau_i)) \quad (6)$$

A imposição de que o spline F seja de classe C^2 — fórmula (13) das notas de aula — resulta no sistema linear com as $(N + 1)$ incógnitas

m_0, \dots, m_{N+1} e com $(N-1)$ equações dadas pela fórmula 23, das notas de aula:

$$m_{i-1}\Delta\tau_i + 2m_i(\Delta\tau_{i-1} + \Delta\tau_i) + m_{i+1}\Delta\tau_{i+1} = b_i \quad (7)$$

com b_i dado pela seguinte fórmula, escrita com a notação de diferenças divididas:

$$b_i = 3(\Delta\tau_i[\tau_{i-1}, \tau_i]g + \Delta\tau_{i-1}[\tau_i, \tau_{i+1}]g) \quad (8)$$

As condições de contorno devem ser acrescentadas de acordo com a escolha entre o spline natural e o not-a-knot. Feito isso, obtemos um sistema linear com $(N+1)$ equações e $(N+1)$ incógnitas cuja matriz A é tridiagonal estritamente dominante e, portanto, tem solução única que pode ser obtida pelo Método de Eliminação de Gauss sem pivotamento. Obviamente, deveremos explorar o fato da matriz A ser uma matriz de banda para incluir as simplificações correspondentes ao Método de Eliminação de Gauss sem pivotamento já implementado no Exercício Programa 1.

2. *double* ValorEmX(*double* x, *double* m[], *double* tau[], *double* y[], *int* n)

Entrada: x Uma abscissa pertencente ao intervalo $[\tau_0, \tau_{N+1}]$, onde será calculado o valor do spline
tau Vetor ordenado de modo crescente representando as abscissas da tabela dada.
y Armazena os valores da função tabela em tau[i]
n Quantidade de polinômios ("tamanho menos um do vetor tau")
Saída: valor Valor do spline calculado em x.

Determine em que intervalo $[\tau_i, \tau_{i+1}]$ se encontra o ponto x. Então calcule o valor do spline no ponto x utilizando a expressão (5).

3. *void* GaussTridiag(*double* A[3][MAX], *int* n, *double* b[], *double* sol[], *int* *IFLAG)

Entrada: A Matriz tridiagonal "colapsada"
n Ordem do sistema linear.
b Vetor do lado direito
Saída: sol Vetor solução
IFLAG Inteiro indicando sucesso (1) ou falha (0)

Vamos utilizar a seguinte estrutura AC para a matriz tridiagonal original A: a. a sobrediagonal de A é armazenada nas entradas $[0][1..n]$ b. a diagonal de A nas entradas $[1][0..n]$ c. a subdiagonal de A nas entradas $[2][0..n-1]$

Então a matriz original $A(N,N)$ é armazenada verticalmente na matriz AC(3,N) Por exemplo, uma matriz AC de ordem 5 seria representada na seguinte forma:

0	A_{01}	A_{12}	A_{23}	A_{34}
A_{00}	A_{11}	A_{22}	A_{33}	A_{44}
A_{10}	A_{21}	A_{32}	A_{43}	0

O programa deve ser implementado utilizando somente a matriz AC , *sem* a utilização da matriz original A .