Métodos Numéricos para Sistemas lineares, Ajustes de curva e Interpolação Polinomial

Lenington do Carmo Rios e Washington Pagotto Batista

Curso de Engenharia de computação – Universidade Estadual de Feira de Santana (UEFS) - 44036-900 – Feira de Santana – BA– Brasil

lenintorios@gmail.com, wstroks@gmail.com

1. Metodologia e Resultados das Aplicações dos Métodos

Os tópicos a seguir irão abordar sobre a utilização de funções residentes do MATLAB R2016a para resolução de métodos numéricos para Sistemas lineares e Interpolação polinomial.

1.1 Funções residentes do MATLAB para resolução de Métodos Numéricos

O MATLAB é uma ferramenta de alto nível. Ela dispõe de operações matemáticas e de funções residentes que podem ser eventualmente utilizadas para solucionar problemas diversos. Por este motivo, esta seção tem como objetivo detalhar o funcionamento de rotinas para a resolução de Sistemas lineares, Interpolação polinomial e Ajuste de curvas.

1.1.1 Sistemas Lineares

Um método iterativo é basicamente o conjunto de soluções que vão melhorando a cada nova iteração feita. As técnicas de iteração geralmente não são utilizadas na construção de sistemas lineares de pequena dimensão, visto que essa solução por possuir maior tempo de resolução comparado com métodos diretos, exemplo a eliminação de Gauss. Quando é aplicado em sistemas lineares grandes e com uma grande quantia de zeros na matriz A, apresentam mais eficiência do que métodos diretos. A eficiência do método iterativo em análise de circuitos e na resolução de problemas aplicada com equações diferenciais.

É importante destacar que os métodos iterativos sempre apresentarão um resultado aproximado e será muito próximo ao valor exato, tendo cuidado na condição de convergência para ser encontrado a solução.

Em geral é feito a transformação do sistema linear que é $Ax = b$ em $x = Cx + g$, tendo sua fórmula geral para aproximação representada abaixo sendo k o número de iterações feitas.

$$X^{(k+1)} = Cx^{(k)} + g$$

Fórmula 1 : Aproximação

Para a solução de sistemas lineares é necessário ter ao seu dispor operações matemáticas básicas, que facilitam a resolução do problema. Visando essa problemática o MATLAB fornece funções prontas para que seja feita as operações.

Uma das operações utilizadas é a divisão, que será feita tanto pela esquerda quanto pela direita. Ambas as operações resolvem sistemas lineares escritas de forma *[a] [x] = [b]*, sendo que *[a]* nas duas operações é *nxn*. Como pode ser visto na Tabela 1, as duas operações apresentam diferenças sendo na construção de sua fórmula básica que será o inverso da outra, e por a incógnitas de [x] e constantes de [b], que serão linhas ou colunas a depender da operação utilizada.

	Fórmula	[x]	[b]
Esquerda	$x = a\backslash b$	vetor coluna (*1xn*)	vetor coluna (*1xn*)
Direita	$x = b/a$	vetor linha (*nx1*)	vetor linha (*nx1*)

Tabela 1: Diferenças da divisão a esquerda e direita.

1.1.1.1 Função mldivide e mrdivide

O comando *mldivide* é utilizado para resolução de equações de sistemas lineares na forma de `Ax=B` com divisão à esquerda, onde as matrizes `A` e `B` devem possuir a mesma quantidade de linhas [1].

Se A é uma matriz quadrada *n-por-n* e B é uma matriz de *n* **linhas**, então $x = A\backslash B$ é uma solução para a equação $Ax=B$. Ainda, se *A* for uma matriz retangular *m-por-n* e *B* uma matriz de *m* linhas, $A\backslash B$ retorna uma solução de quadrados mínimos.

Esse comando possui a seguinte sintaxe:

```
x = A\B
x = mldivide(A,B),
```

Como exemplo de sua utilização, define-se as matrizes *A* e *B* e faz a chamada da rotina. O resultado do comando *mldivide* também pode ser obtido com $x = A\backslash B$.

```
>>A = [0 2 0 1 0; 4 -1 -1 0 0; 0 0 0 3 -6; -2 0 0 0 2; 0 0 4 2 0]

>>B = [8; -1; -18; 8; 20]

>>x = mldivide(A,B)

x =

    1

    2

    3

    4

    5
```

O comando *mrdivide*, é utilizado para resolução de equações de sistemas lineares na forma de `Ax=B` com divisão à direita, onde as matrizes `A` e `B` devem possuir a mesma quantidade de colunas [2].

Se `A` é uma matriz quadrada *n-por-n* e `B` é uma matriz de *n* **colunas**, então `x = A/B` é uma solução para a equação `Ax=B`. Ainda, se `A` for uma matriz retangular *m-por-n* e `B` uma matriz de *m* colunas, `A/B` retorna uma solução de quadrados mínimos.

Esse comando possui a seguinte sintaxe:

```
x = A/B
x = mrdivide(A,B),
```

Utilizando o exemplo anterior ajustando a matriz B para o *rmdivide*:

```
>>A = [0 2 0 1 0; 4 -1 -1 0 0; 0 0 0 3 -6; -2 0 0 0 2; 0 0 4 2 0]

>>B = [8 -1 -18 8 20]

>>x = mldivide(A,B)

x =

    0.0070

    0.0598

    -0.113

    0.0281

    -0.066
```

O argumento *A* é o coeficiente de uma matriz, podendo ser um vetor, matriz completa ou esparsa. O argumento *B* é o lado direito da equação, também podendo ser representado por um vetor, matriz completa ou esparsa. Ambas matrizes devem possuir a mesma quantidade de linhas (para divisão à esquerda) ou de colunas (para divisão à direita). O argumento de saída *x* é a matriz solução da equação do sistema linear.

1.1.1.2 Função linsolve e solve

O Matlab ainda dispõe de outros dois recursos para resolução de sistemas de equações lineares, são eles os comandos *linsolve* e o comando *solve*.

O primeiro deles, *linsolve*, resolve o sistema linear $Ax = B$ usando fatoração LU com pivoteamento parcial. Para esse comando, o número de linhas da matriz *A* deve ser igual ao de *B*. Se *A* é uma matriz *n-por-m* e *B* uma matriz *n-por-k*, então *x* será *n-por-k* [3].

Este comando possui a seguinte sintaxe:

```
x = linsolve(A,B)
x = linsolve(A,B,opts),
```

Os argumentos *A* e *B* são as matrizes do sistema e *x* a matriz resultante da equação. Ainda, o argumento *opts* trata-se das propriedades da matriz *A*. Este argumento serve para facilitar no desenvolver da solução, tornando o comando linsolve mais rápido que o *mldivide* (exposto anteriormente). Caso não seja especificado as propriedades da matriz, o comando *linsolve* terá que executar todos os testes para verificar que *A* tem as propriedades específicas, o que leva a preferência da utilização do comando *mldivide*.

A Tabela 2, disponível em [3], lista todos os campos do argumento *opts* e suas propriedades de matriz correspondente. Vale ressaltar que os valores padrões para todos os campos é *false*.

Nome do Campo	**Propriedade da Matriz**
LT	Triangular inferior
UT	Triangular superior
UHESS	Hessemberg superior
SYM	Hermitiano real simétrico ou complexo
POSDEF	Positivo definitivo
RECT	Retangular geral
TRANSA	Transposto conjugado - especifica se a função resolve $Ax=B$ ou $A'x=B$.

Tabela 2: Propriedades da matriz.

Tomando como exemplo:

```
A = [0 2 0 1 0; 4 -1 -1 0 0; 0 0 0 3 -6; -2 0 0 0 2; 0 0 4 2 0]

B = [8 -1 -18 8 20]

opts.TRANSA = true;

>>x = linsolve(A,B,opts)

x =

    4.7500

   10.5000

    2.3333

   17.0000

   -1.8750

>>x = linsolve(A,B)
```

```
x =

      1

      2

      3

      4

      5
```

O comando *solve* é utilizado para solucionar equações e sistemas. Sua sintaxe é de maior complexidade comparado ao linsolve [4].

Este comando possui as seguintes sintaxes:

```
S = solve(eqn,var)

S = solve(eqn,var,Name,Value)


Y = solve(eqns,vars)

Y = solve(eqns,vars,Name,Value)


[y1,...,yN] = solve(eqns,vars)

[y1,...,yN] = solve(eqns,vars,Name,Value)

[y1,...,yN,parameters,conditions] =
solve(eqns,vars,'ReturnConditions',true)
```

Em todas elas resolvem a equação *eqn* para a variável *var*. A seguir, serão expostos alguns exemplos de aplicações com o comando *solve*.

Resolvendo uma equação simples de $sin(x) = 1$.

```
>>syms x
>>eqn = sin(x) == 1;
>>x = solve(eqn,x)

x =
    pi/2
```

Resolvendo equação com mais de uma variável armazenando as saídas em uma matriz.

```
>>syms u v
>>S = solve([2*u + v == 0, u - v == 1], [u, v])

S =
  struct with fields:
```

```
    u: [1×1 sym]
    v: [1×1 sym]
>>S.u %para acessar o valor da variável u
u =
    1/3
>>S.v %para acessar o valor da variável u
v =
   -2/3
```

Resolvendo sistema de equações atribuindo as soluções para *solv* e *solu*. Retornando uma matriz de soluções para cada variável.

```
>>syms u v
>>[solv, solu] = solve([2*u^2 + v^2 == 0, u - v == 1], [v, u])
solv =
 - (2^(1/2)*1i)/3 - 2/3
   (2^(1/2)*1i)/3 - 2/3
solu =
 1/3 - (2^(1/2)*1i)/3
 (2^(1/2)*1i)/3 + 1/3
>>solutions = [solv, solu]
solutions =
           [ - (2^(1/2)*1i)/3 - 2/3, 1/3 - (2^(1/2)*1i)/3]
           [   (2^(1/2)*1i)/3 - 2/3, (2^(1/2)*1i)/3 + 1/3]
```

1.1.2 Interpolação Polinomial

Nesta subseção serão abordados as principais rotinas do Matlab para resolução de métodos numéricos para Interpolação polinomial.

1.1.2.1 Função interp1

A função *interp1* é o meio prático para ser feito a implementação de interpolação unidimensional entre determinados pontos. Tal método retorna valores interpolados de uma função 1D (unidimensional) usando interpolação linear [5].

Este comando possui as seguintes sintaxes:

```
vq=interp1(x,v,xq)
vq=interp1(x,v,method)
vq=interp1(x,v,method,extrapolation)
vq=interp1(v,xq)
```

```
vq=interp1(v,xq,method)
vq=interp1(v,xq,method, extrapolation)
pp=interp1(x,v,method,pp)
```

Como exemplo, toma-se a função $sin(x)$ para interpolação, e o gráfico é mostrado na Figura 1.

```
>>x = 0:pi/4:2*pi
>>v = sin(x)
>>xq = 0:pi/16:2*pi
>>vq = interp1(x,v,xq)
>>plot(x,v,'o',xq,vq1,':.')
```

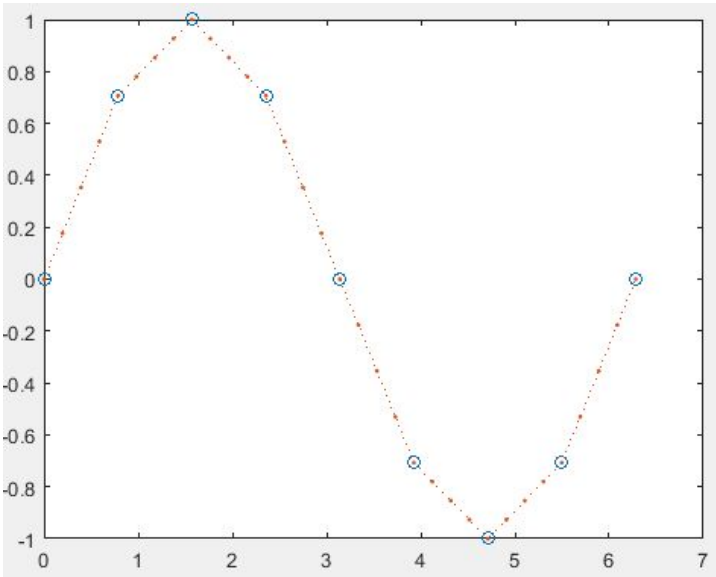

Figura 1: Comando interp1.

1.1.2.2 Função interp2

A *interp2* é a implementação bidimensional dos dados. Tal comando retorna os valores interpolados de uma função de duas variáveis utilizando interpolação linear [6].

Este comando possui as seguintes sintaxes:

```
vq=interp2(x,y,v,xq,yq)]
vq=interp2(v,xq,yq)
vq=interp2(v)
vq=interp2(v,k)
vq=interp2(__,method)
vq=interp2(___,method,extrapval)
```

A seguir, tem-se um exemplo da utilização do comando *interp2* juntamente com seu gráfico resultante (Figura 2).

```
>>[Xq,Yq] = meshgrid(-3:0.25:3)
>>Vq = interp2(X,Y,V,Xq,Yq)
>>surf(Xq,Yq,Vq)
```

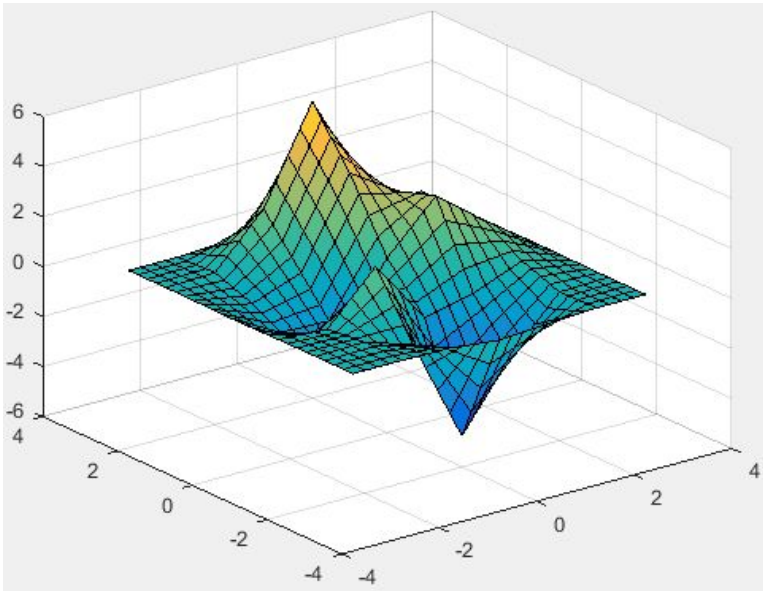

Figura 2: interp2 demonstração.

Tal método ainda pode ser utilizado para refinar uma imagem em tons de cinza. O próximo exemplo mostra como isso é possível.

```
>>load clown %carrega a imagem
>>V = single(X(1:124,75:225)) %imagem normal
>>Vq = interp2(V,5) %refina a imagem
>>imagesc(V);
>>imagesc(Vq);
>>axis image
>>axis off
```

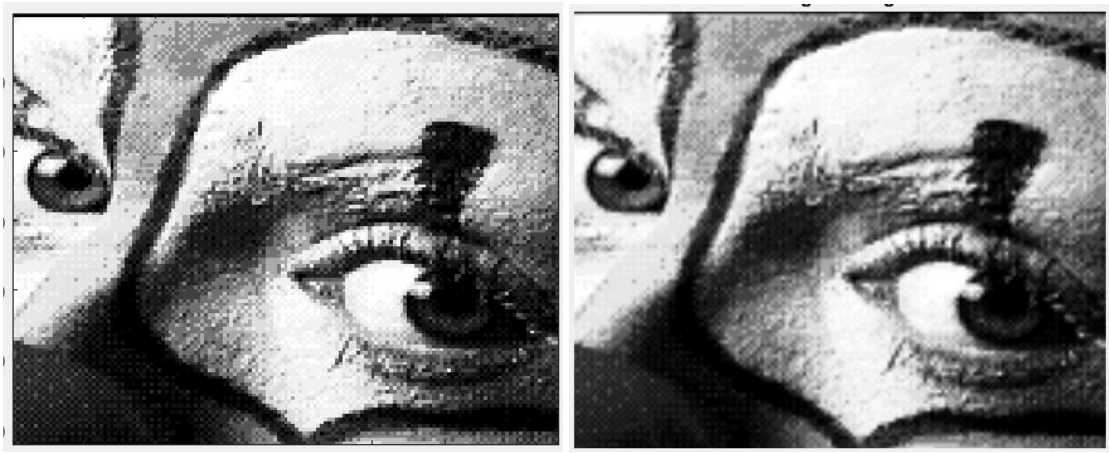

Figura 3: Imagem original versus refinada utilizando interp2.

1.1.2.3 Função interp3

A *interp3* é a implementação tridimensional. Tal comando retorna valores interpolados de uma função de três variáveis em pontos específicos utilizando interpolação linear [7].

Possui as seguintes sintaxes:

```
vq=interp3(x,y,z,v,xq,yq,zq)
vq=interp3(v,xq,yq,zq)
vq=interp3(v)
vq=interp3(v,k)
vq=interp3(__,method)
vq=interp3(___,method,extrapval)
```

O comando *interp3* pode ser utilizado para refinar imagens, assim como o *interp2*. No exemplo a seguir mostra como uma imagem pode ser definida utilizando este comando residente do Matlab.

```
>>[X,Y,Z,V] = flow(10)
>>slice(X,Y,Z,V,[6 9],2,0)
>>shading flat
>>[Xq,Yq,Zq] = meshgrid(.1:.25:10,-3:.25:3,-3:.25:3)
>>Vq = interp3(X,Y,Z,V,Xq,Yq,Zq)
>>slice(Xq,Yq,Zq,Vq,[6 9],2,0)
```

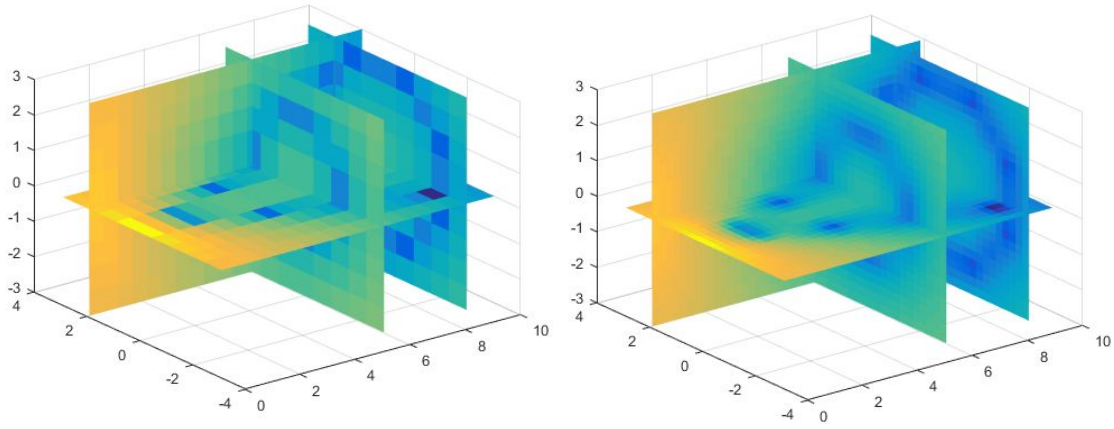

Figura 4: Gráfico normal e interpolado.

1.1.2.4 Função interpn

A função *interpn* é a função mais completa para implementação polinomial pois ela agrega tanto as funções *interp1*, *interp2*, *interp3* e *ndgrid*. Tal comando retorna valores interpolados de uma função de *n* variáveis usando interpolação linear [8].

Possui as seguintes sintaxes:

```
vq=interpn(x1,x2,...,xn,v,xq1,xq2,...,xqn)
vq=interpn(v,xq1,xq2,..xqn)
vq=interpn(v)
vq=interpn(v,k)
vq=interpn(__,method)
vq=interp3(___,method,extrapval)
```

O exemplo a seguir utiliza o comando interpn para interpolação linear e cúbica.

```
>>x = [1 2 3 4 5];
>>v = [12 16 31 10 6]
>>xq = (1:0.1:5)
>>vq = interpn(x,v,xq,'linear') %interpolação linear
>>plot(x,v,'o',xq,vq,'-')
>>vq = interpn(x,v,xq,'cubic') %interpolação cúbica
>>plot(x,v,'o',xq,vq,'-')
```

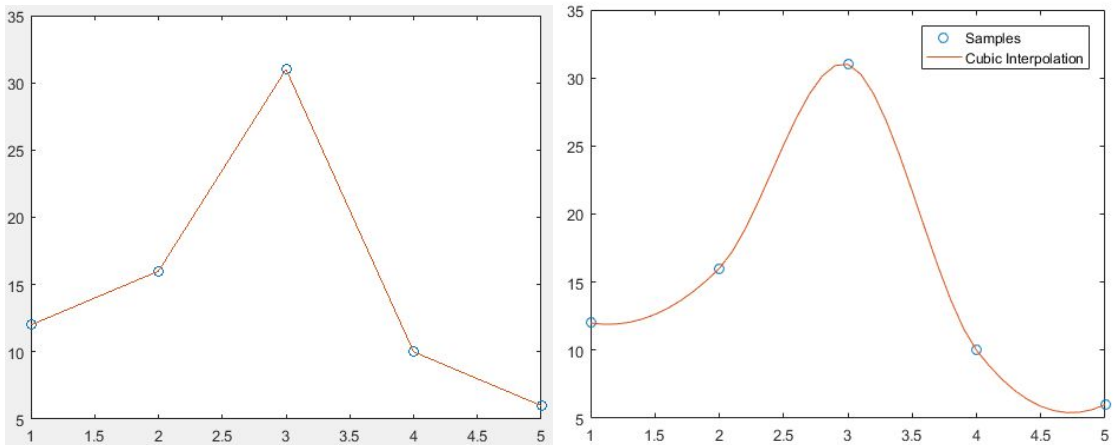

Figura 5: Gráficos interpolados linear e cúbica.

1.1.2.5 Função polyval e polyfit

O comando *polyfit* é utilizado para ajustes (utilizando o conceito de mínimos quadrados) de dados de um polinômio de grau *n* na forma $P(x) = P_1x^n + P_2x^{n-1} + ... + P_nx + P_{n-1}$. Além de fazer o retorno do ajuste, a rotina fornece de base para a encontrar a estimativa do erro utilizada pelo *polyval*. Com isso o *polyval* retornará a situação do polinômio, encontrando o erro e a onde o erro se encaixa [9][10].

O comando *polyfit* possui as seguintes sintaxes:

```
p = polyfit(x,y,n)
[p,S] = polyfit(x,y,n)
[p,S,mu] =  polyfit(x,y,n)
```

Por sua vez, *polyval* possui as seguintes sintaxes:

```
y = polyval(p,x)
[y,delta] = polyval(p,x,S)
y = polyval(p,x,[],mu)
[y,delta] = polyval(p,x,S,mu)
```

O exemplo a seguir utiliza os dois métodos para o conjunto de pontos para mostrar o quanto o polinômio diverge da função.

```
>>x = linspace(0,1,5);
>>y = 1./(1+x);
>>p = polyfit(x,y,4);
>>x1 = linspace(0,2);
>>y1 = 1./(1+x1);
>>f1 = polyval(p,x1);
>>plot(x,y,'o',x1,y1,x1,f1,'r--')
```

```
>>legend('y','y1','f1')
```

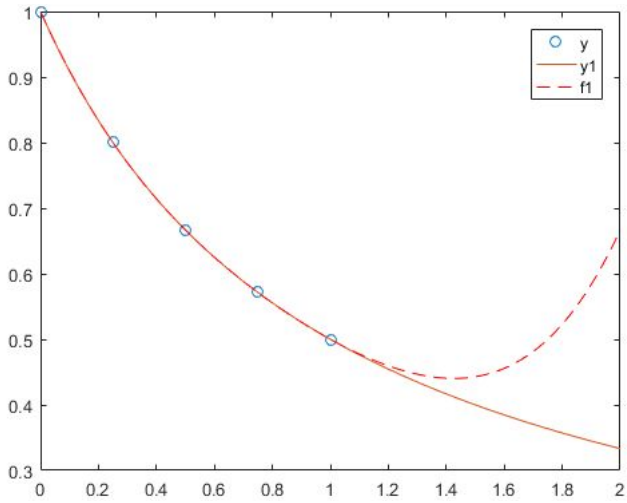

Figura 6: Ajuste de pontos usando polyfit e polyval.

1.1.2.6 Função spline

A função *spline* usa a interpolação cúbica como base. Em splines cúbica é utilizada polinômios de ordem três, determinados pelo conjunto de dados *n*, onde há um intervalo *n-1*. É importante destacar que a rotina retorna a forma polinomial da função cúbica spline [11].

Este comando possui as seguintes sintaxes:

```
yy = spline(x,Y,xx)
pp = spline(x,Y)
```

Como exemplo de sua utilização, efetuou-se a interpolação por splines da função $f(x) = sin(x)$.

```
>>x = 0:10
>>y = sin(x)
>>xx = 0:.25:10
>>yy = spline(x,y,xx)
>>plot(x,y,'o',xx,yy)
```

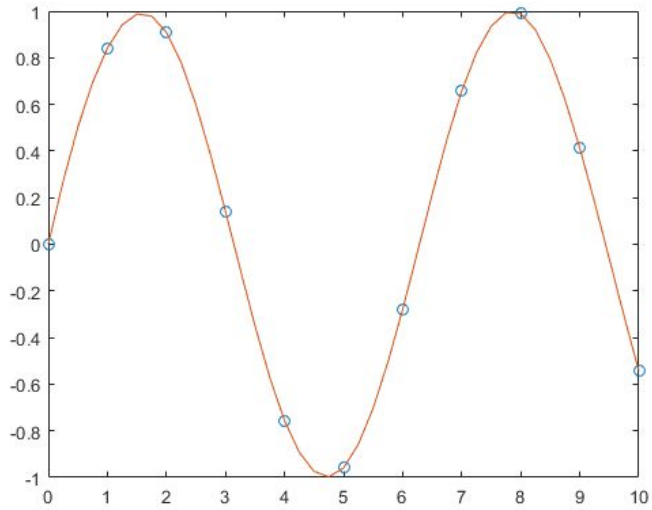

Figura 7: Interpolação por splines

1.1.2.7 Função spaps

A função *spaps* é a implementação de uma suavização em uma spline. Sendo assim a o *spaps* tenta diminuir o erro ocasionado e tenta garantir o melhor valor aproximado. Outro ponto importante é que o comando pode ser aplicado para o método de newton garantindo a convergência [13].

Tal comando possui as seguintes sintaxes:

```
sp = spaps(x,y,tol)
[sp,values] = spaps(x,y,tol)
[sp,values,rho] = spaps(x,y,tol)
[...] = spaps(x,y,tol,arg1,arg2,...)
[...] = spaps({x1,...,xr},y,tol,...)
```

Como exemplo de sua aplicação, foi efetuado uma suavização em uma *spline*.

```
>>x = -2:.2:2; y=-1:.25:1; [xx,yy] = ndgrid(x,y); rng(39);
>>z = exp(-(xx.^2+yy.^2)) + (rand(size(xx))-.5)/30;
>>sp = spaps({x,y},z,8/(60^2));  fnplt(sp), axis off
```

1.1.2.8 Função csapi

Já a função *csapi* implementa uma interpolação de spline cúbico. Um spline cúbico tem a tendência de apresentar dados com bastante ruído, com isso o comando faz um uma suavização dos dados, deixando mais coerente [14].

O comando *csapi* possui as seguintes sintaxes:

```
pp=csapi(x,y)
values = csapi(x,y,xx)
```

Como exemplo foi aplicado, efetuando uma spline cúbica.

```
>>x =.0001+[-4:.2:4]; y = -3:.2:3;
>>[yy,xx] = meshgrid(y,x); r = pi*sqrt(xx.^2+yy.^2); z = sin(r)./r;
>>bcs = csapi( {x,y}, z ); fnplt( bcs ), axis([-5 5 -5 5 -.5 1])
```

1.2 Dados Obtidos Computacionalmente

As rotinas aplicadas para a solução da proposta do trabalho, foi de total afinidade com ambos os autores. Visto que existem rotinas diferentes para cada aplicação desenvolvida. Na Tabela 3 é mostrado o que foi aplicado para a resolução e o que poderia ser usado.

Assuntos	Rotinas utilizadas	Rotinas não utilizadas
Sistema linear	*linsolve*	*solve, mrdivide e mldivide*
Interpolação Polinomial	*interpn*	*interp1,interp2, interp3,ndgrid e msgrid*
Spline	*spline*	*spaps,csapi*
Ajuste de Curvas	*polyfit e polyval*	

Tabela 3: Rotinas por assunto.

As subseções a seguir detalham a utilização das rotinas para resolução de problemas matemáticos, bem como os dados utilizados.

1.2.1 Sistema Linear

Para o sistema linear foi utilizado a rotina linsolve do Matlab para resolver a equação $Ax = B$.

Inicialmente os valores da matriz A e B foram inseridos no script para fazer a chamada do comando residente do Matlab. O seguinte código mostra os valores contidos nas respectivas matrizes.

```
>>A = [4 -2 -3 6; -6 7 6.5 -6; 1 7.5 6.25 5.5; -12 22 15.5 -1];
>>B = [12; -6.5; 16; 17];
```

A é uma matriz *4x4*, enquanto que *B* é uma matriz *1x4*. O sistema linear fica na forma como mostrado na Figura 8.

$$\begin{pmatrix} 4 & -2 & -3 & 6 \\ -6 & 7 & 6.5 & -6 \\ 1 & 7.5 & 6.25 & 5.5 \\ -12 & 22 & 15.5 & -1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 12 \\ -6 \\ 16 \\ 17 \end{pmatrix}$$

Figura 8: Equação do sistema linear.

O resultado da equação matricial que o comando *linsolve* retorna é mostrado na Figura 9.

$$x = \begin{bmatrix} 2 \\ 4 \\ -3 \\ 0.5 \end{bmatrix}$$

Figura 9: Resultado da equação matricial.

1.2.2 Interpolação Polinomial e Splines

Para os métodos de interpolação polinomial e splines foram utilizados as rotinas *interpn* e *spline*.

Inicialmente definiu-se os dados experimentais (pontos), mostrados a seguir na Tabela 4. Tais pontos foram armazenados em vetores e atribuídas em variáveis. Em seguida, aplicou-se os comandos *interpn* e *spline*.

x	1	2	3	4	5	6	7	8	9
f(*x*)	6	10	14	25	39	44	58	65	71

Tabela 4: Pontos experimentais.

Por possuir grandes números de pontos, é necessário a comparação do comportamento dos dois métodos. Isso se dá pelo fato de que a utilização de

interpolação polinomial entre os dados pode apresentar desvios significativos no seu ajuste. A Figura 10 mostra o comportamento dos dois comandos.

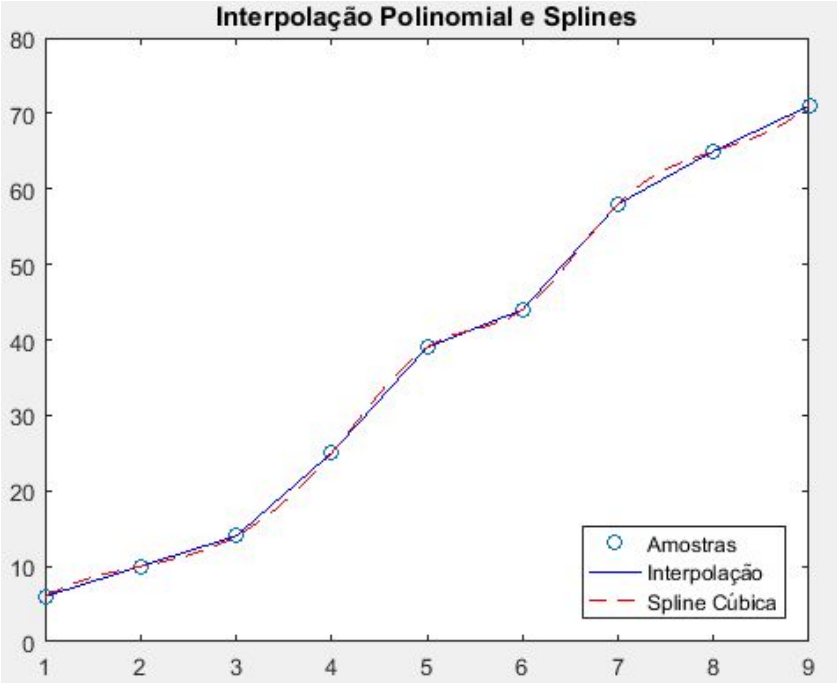

Figura 10: Rotina interpn versus spline.

Pode-se perceber através desse gráfico a diferença entre as curvas que são formadas pelos pontos. Enquanto que o interpn gera linhas retas entre cada ponto, o spline gera curvas entre os seus nós.

1.2.3 Extrapolação

Para fazer a estimativa de um valor $f(x)$ que está fora do intervalo dos pontos conhecidos, utilizou-se os comandos *polyfit* e *polyval*. Os pontos experimentais utilizados são os da Tabela 4, exposta na subseção anterior.

Tomando como base os dados de x de 1 a 8 para obter uma previsão de $x = 9$. Para isso, primeiro obteve-se o ajuste dos dados do polinômio $P_7(x) = -0.03x^7 + 0.81x^6 - 9.75x^5 + 60.95x^4 - 211.85x^3 + 407.24x^2 - 394.37x + 153$, aproximadamente, através do método *polyfit*. Com o método *polyval*, obteve-se o valor da previsão para $f(9) = -288$, um valor bastante distante, negativamente, do dado experimental para $f_{real}(9) = 71$. A Figura 10 mostra o comportamento da extrapolação do polinômio interpolador e a curva verdadeira/real.

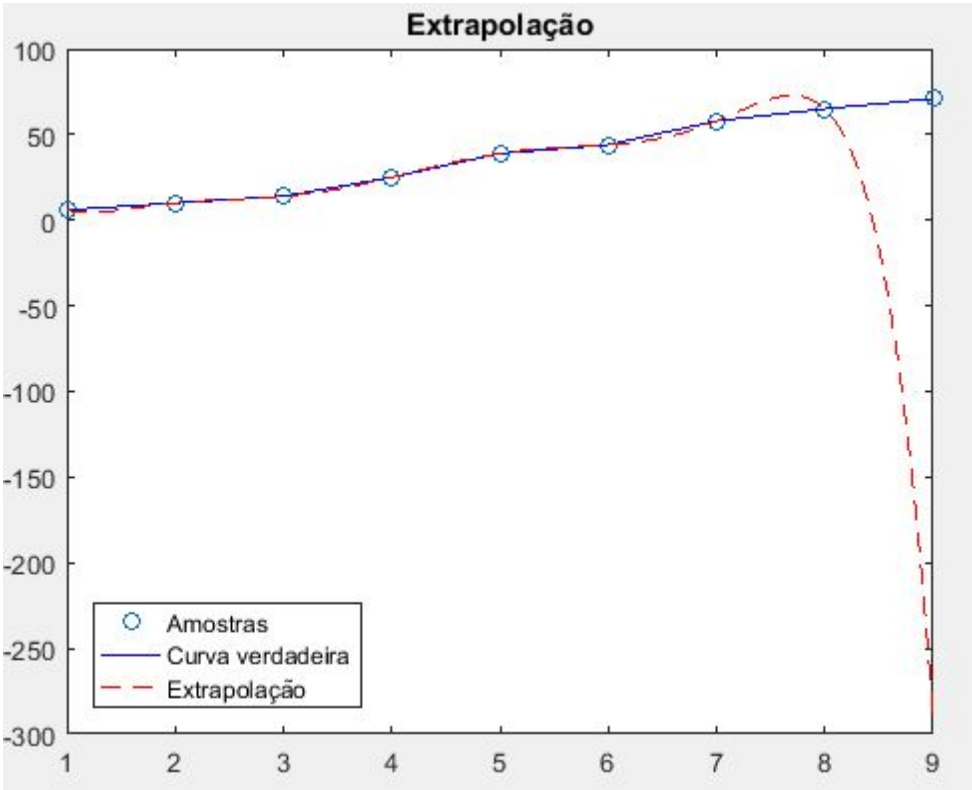

Figura 10: Extrapolação.

1.2.4 Ajustes de Curvas

Para prover um melhor ajuste de curvas do polinômio obtido anteriormente, utilizou-se novamente os comandos residentes do Matlab, o *polyfit* e o *polyval*. Para isso, utilizou-se os comandos *min* e *max* que retornam, respectivamente, o menor e o maior elemento contido no array.

Para obter a melhor reta que passa entre os pontos das amostras utilizou-se *polyfit(x,y,2)* passando por parâmetros o array dos valores de *x* e *y* e o grau do polinômio (segundo grau).

Em seguida, utilizou-se o comando *linspace*, passando por parâmetros valor mínimo e o máximo de *x*. Por fim, faz-se a chamada do método *polyval*, passando os parâmetros do retorno da rotina *polyfit* e *linspace*. O gráfico do ajuste da curva para os pontos é mostrado na Figura 11.

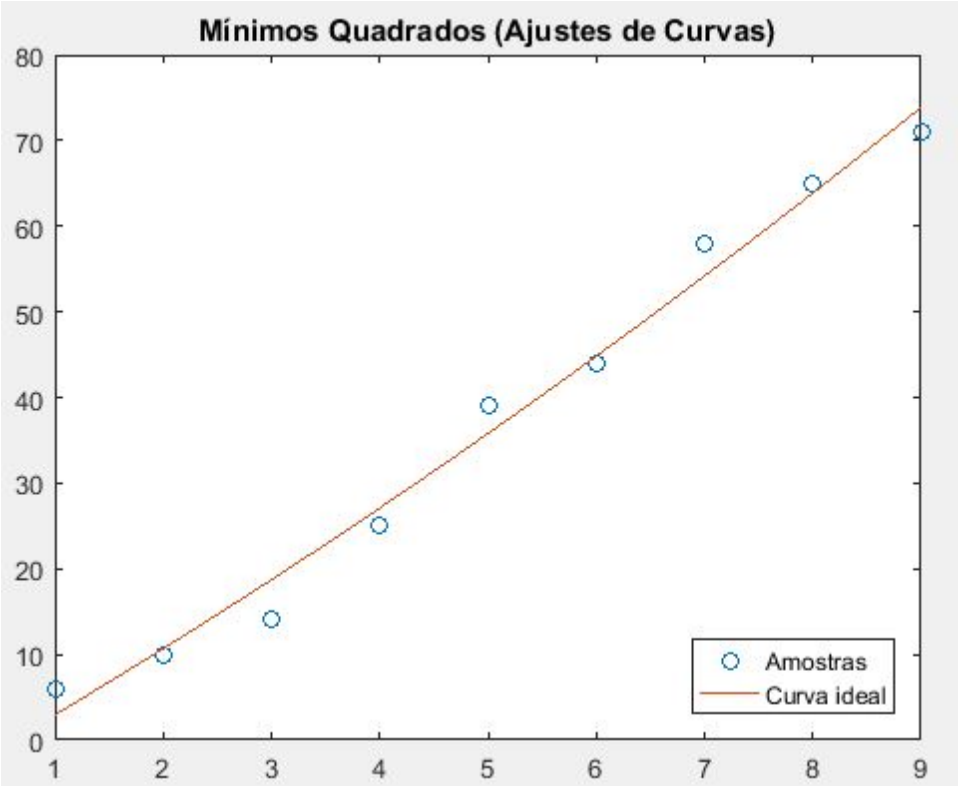

Figura 11: Ajuste do polinômio.

2.Referências

[1] MATHWORKS, Solve systems of linear equation Ax = B for x: mldivide, \ - Documentation. Disponível em: http://www.mathworks.com/help/matlab/ref/mldivide.html. Acesso em: 09 de out. 2016.

[2] MATHWORKS, Solve systems of linear equation Ax = B for x: mrdivide, / - Documentation. Disponível em: http://www.mathworks.com/help/matlab/ref/mrdivide.html. Acesso em: 09 de out. 2016.

[3] MATHWORKS, Solve systems of linear equation Ax = B for x: linsolve - Documentation. Disponível em: http://www.mathworks.com/help/matlab/ref/linsolve.html. Acesso em: 09 de out. 2016.

[4] MATHWORKS, Equations and systems solver: solve - Documentation. Disponível em: http://www.mathworks.com/help/matlab/ref/linsolve.html. Acesso em: 09 de out. 2016.

[5] MATHWORKS, interp1 - 1-D data interpolation (table lookup)- Documentation. Disponível em :

https://www.mathworks.com/help/matlab/ref/interp1html?searchHighlight=interp1. Acesso em: 09 de out. 2016.

[6] MATHWORKS, interp2 - Interpolation for 2-D gridded data in meshgrid format- Documentation. Disponível em : https://www.mathworks.com/help/matlab/ref/interp2.html?searchHighlight=interp2. Acesso em: 09 de out. 2016.

[7] MATHWORKS, interp3 - Interpolation for 3-D gridded data in meshgrid format- Documentation. Disponível em : https://www.mathworks.com/help/matlab/ref/interp3.html?searchHighlight=interp3. Acesso em: 09 de out. 2016.

[8] MATHWORKS, interpn - Interpolation for 1-D, 2-D, 3-D, and N-D gridded data in ndgrid format- Documentation. Disponível em : https://www.mathworks.com/help/matlab/ref/interpn.html?searchHighlight=interpn. Acesso em: 09 de out. 2016.

[8] MATHWORKS, ndgrid - Rectangular grid in N-D space- Documentation. Disponível em : https://www.mathworks.com/help/matlab/ref/ndgrid.html?searchHighlight=ndgrid. Acesso em: 09 de out. 2016.

[9] MATHWORKS, polyfit - Polynomial curve fitting- Documentation. Disponível em : https://www.mathworks.com/help/matlab/ref/polyfit.html?searchHighlight=polyfit. Acesso em: 09 de out. 2016.

[10] MATHWORKS, polyval - Polynomial evaluation- Documentation. Disponível em : https://www.mathworks.com/help/matlab/ref/polyval.html?searchHighlight=polyval. Acesso em: 09 de out. 2016.

[11] MATHWORKS, spline- Cubic spline data interpolation- Documentation. Disponível em : https://www.mathworks.com/help/matlab/ref/spline.html?searchHighlight=spline. Acesso em: 09 de out. 2016.

[12] GILAT, A., SUBRAMANIAM, V. Métodos Numéricos para Engenheiros e Cientistas, uma introdução com aplicações usando o MATLAB. Bookman. Porto Alegre, 2008.

[13] MATHWORKS, spaps- Smoothing spline- Documentation. Disponível em : https://www.mathworks.com/help/curvefit/spaps.html?searchHighlight=spap. Acesso em: 10 de out. 2016.

[14] MATHWORKS,csapi- Cubic spline interpolation- Documentation. Disponível em : https://www.mathworks.com/help/curvefit/csapi.html?searchHighlight=csapi. Acesso em: 10 de out. 2016.

[15] CHAPRA, S. C. Applied Numerical Methods with MATLAB for Engineers and Sientists. Third Edition. New York, NY. 2012.