

# Interpolação Polinomial

Prof. Americo Cunha

Universidade do Estado do Rio de Janeiro – UERJ

[americo.cunha@uerj.br](mailto:americo.cunha@uerj.br)

[www.americocunha.org](http://www.americocunha.org)



@AmericoCunhaJr



@AmericoCunhaJr



@AmericoCunhaJr



@AmericoCunhaJr



Imagine que você tenha que lidar com uma **função complicada**:



Imagine que você tenha que lidar com uma **função complicada**:

$$f(x) = \frac{\ln x e^{\sqrt{x}}}{\ln(1 + e^x) + x^3}$$

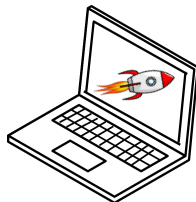
**fórmula complexa**



Imagine que você tenha que lidar com uma **função complicada**:

$$f(x) = \frac{\ln x e^{\sqrt{x}}}{\ln(1 + e^x) + x^3}$$

**fórmula complexa**

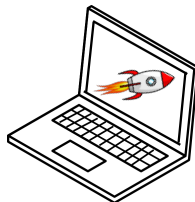


**simulador computacional  
(sem fórmula)**

Imagine que você tenha que lidar com uma **função complicada**:

$$f(x) = \frac{\ln x e^{\sqrt{x}}}{\ln(1 + e^x) + x^3}$$

**fórmula complexa**



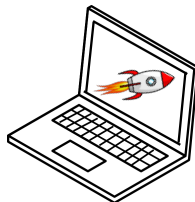
**simulador computacional  
(sem fórmula)**

**Como manipular (derivar, integrar etc)  
uma função desse tipo na prática?**

Imagine que você tenha que lidar com uma **função complicada**:

$$f(x) = \frac{\ln x e^{\sqrt{x}}}{\ln(1 + e^x) + x^3}$$

**fórmula complexa**



**simulador computacional  
(sem fórmula)**

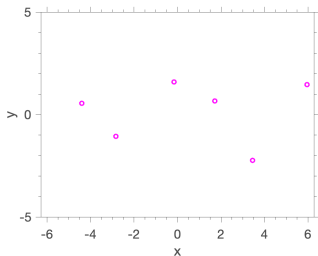
**Como manipular (derivar, integrar etc)  
uma função desse tipo na prática?**

**Através de uma função simplista que aproxima o  
comportamento de  $f(x)$ , construída a partir de alguns  
pontos onde o valor da função é conhecido!**

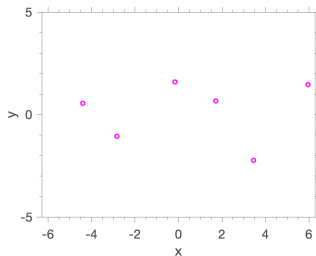


# Interpolação $\times$ Regressão

## Interpolação

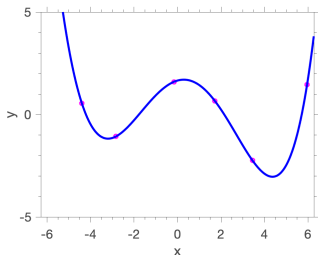


## Regressão



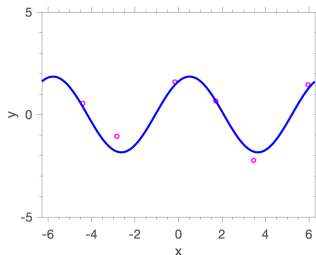
# Interpolação × Regressão

## Interpolação



A curva interpolante passa  
**por todos os pontos.**

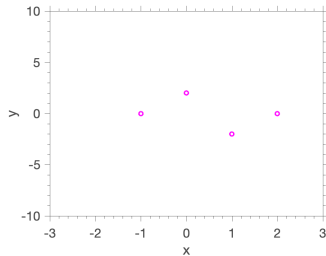
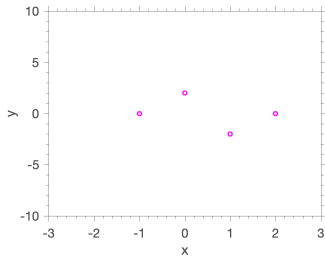
## Regressão



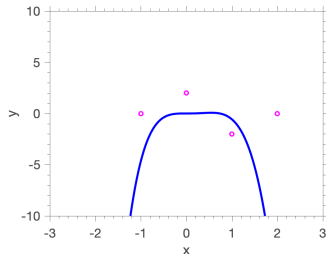
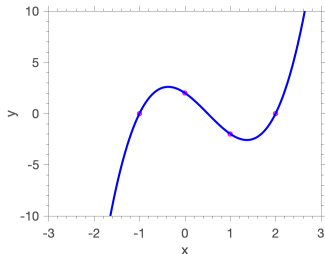
A curva regressora passa  
**“próxima” de todos os pontos.**



# Dados bem comportados ...



# Dados bem comportados ...



**... podem ser bem aproximado (em geral)  
por um processo de interpolação!**

# O problema de interpolação

Dado um conjunto com  $n + 1$  *pontos* distintos ( $x_i \neq x_j$ )

$$(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n),$$

encontre uma *função interpolante*  $p(x)$  tal

$$p(x_j) = y_j, \quad j = 0, 1, \dots, n$$

i.e., a curva  $y = p(x)$  passa pelos  $n + 1$  pontos dados.



# Processo de interpolação

1. Escolha uma forma para a **função interpolante**

$$p(x) = c_n \phi_n(x) + \cdots + c_1 \phi_1(x) + c_0 \phi_0(x)$$

- $\phi_j$  - funções linearmente independentes (predefinidas)
- $c_j$  - coeficientes (a serem determinados)

2. Encontre os **coeficientes  $c_j$**  (resolvendo um sistema linear) de modo que  **$y = p(x)$**  seja uma **curva que contenha todos os pontos da amostra**.



# Função interpolante



# Função interpolante

$$p(x) = c_n \phi_n(x) + \cdots + c_1 \phi_1(x) + c_0 \phi_0(x)$$



# Função interpolante

$$p(x) = c_n \phi_n(x) + \cdots + c_1 \phi_1(x) + c_0 \phi_0(x)$$

$$p(x_0) = c_n \phi_n(x_0) + \cdots + c_1 \phi_1(x_0) + c_0 \phi_0(x_0) = y_0$$

$$p(x_1) = c_n \phi_n(x_1) + \cdots + c_1 \phi_1(x_1) + c_0 \phi_0(x_1) = y_1$$

$$\vdots \qquad \qquad \qquad \vdots$$

$$p(x_n) = c_n \phi_n(x_n) + \cdots + c_1 \phi_1(x_n) + c_0 \phi_0(x_n) = y_n$$



# Função interpolante

$$\underbrace{\begin{bmatrix} \phi_n(x_0) & \cdots & \phi_1(x_0) & \phi_0(x_0) \\ \phi_n(x_1) & \cdots & \phi_1(x_1) & \phi_0(x_1) \\ \vdots & \ddots & \vdots & \vdots \\ \phi_n(x_{n-1}) & \cdots & \phi_1(x_{n-1}) & \phi_0(x_{n-1}) \\ \phi_n(x_n) & \cdots & \phi_1(x_n) & \phi_0(x_n) \end{bmatrix}}_A \underbrace{\begin{bmatrix} c_n \\ c_{n-1} \\ \vdots \\ c_1 \\ c_0 \end{bmatrix}}_x = \underbrace{\begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_1 \\ y_0 \end{bmatrix}}_b$$



$$A\mathbf{x} = \mathbf{b}$$





# Função interpolante

$$\underbrace{\begin{bmatrix} \phi_n(x_0) & \cdots & \phi_1(x_0) & \phi_0(x_0) \\ \phi_n(x_1) & \cdots & \phi_1(x_1) & \phi_0(x_1) \\ \vdots & \ddots & \vdots & \vdots \\ \phi_n(x_{n-1}) & \cdots & \phi_1(x_{n-1}) & \phi_0(x_{n-1}) \\ \phi_n(x_n) & \cdots & \phi_1(x_n) & \phi_0(x_n) \end{bmatrix}}_A \underbrace{\begin{bmatrix} c_n \\ c_{n-1} \\ \vdots \\ c_1 \\ c_0 \end{bmatrix}}_x = \underbrace{\begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_1 \\ y_0 \end{bmatrix}}_b$$



$$A\mathbf{x} = \mathbf{b}$$

**Esse sistema tem uma única solução!**



# Experimento computacional 1

```
clc; clear; close all
```

```
f = @(x) log(x).*exp(sqrt(x))./(log(1+exp(x))+x.^3);
```

```
xdat = [1.3 2.1 4.0 5.8 8.1]; ydat = f(xdat);
```

```
xgrid = 0.1:0.01:10; ygrid = f(xgrid);
```

```
A = [xdat.^4; xdat.^3; xdat.^2; xdat; ones(1,5)]';
```

```
xsol = A\ydat';
```

```
p1 = xsol(1)*xgrid.^4 + xsol(2)*xgrid.^3 + ...  
      xsol(3)*xgrid.^2 + xsol(4)*xgrid + xsol(5);
```

```
A = [exp(-xdat.^2); exp(-xdat); xdat.^2; xdat; ones(1,5)]';
```

```
xsol = A\ydat';
```

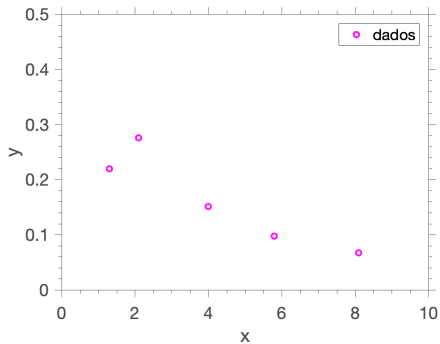
```
p2 = xsol(1)*exp(-xgrid.^2) + xsol(2)*exp(-xgrid) + ...  
      xsol(3)*xgrid.^2 + xsol(4)*xgrid + xsol(5);
```



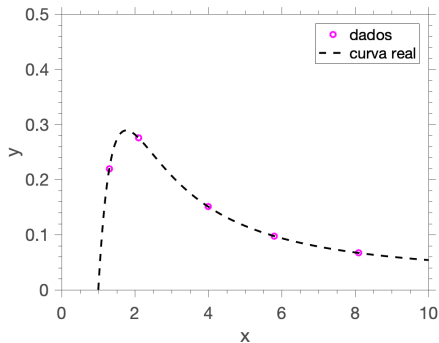
# Experimento computacional 1

```
plot(xdat,ydat,'om','LineWidth',2);  
hold on  
plot(xgrid,ygrid, '--k','LineWidth',2);  
plot(xgrid,p1, '-.r','LineWidth',2);  
plot(xgrid,p2, '-b','LineWidth',2);  
hold off  
xlabel('x'); ylabel('y');  
set(gca,'FontSize',18);  
legend('dados','f(x)','p1(x)','p2(x)')  
xlim([0 10]); ylim([0 0.5]);
```

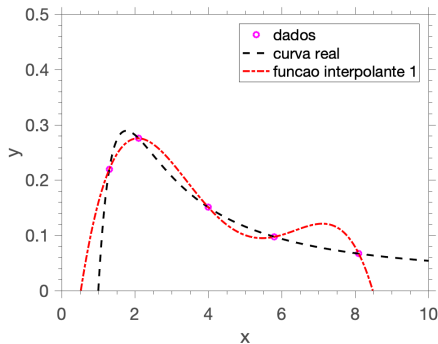




$$f(x) = \frac{\ln x e^{\sqrt{x}}}{\ln(1 + e^x) + x^3}$$

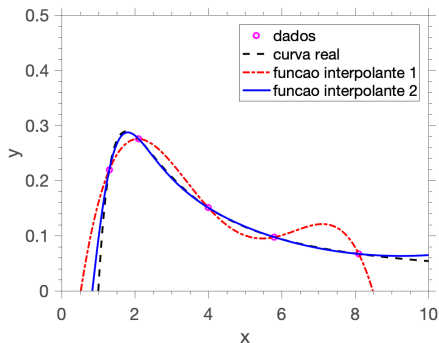


$$f(x) = \frac{\ln x e^{\sqrt{x}}}{\ln(1 + e^x) + x^3}$$



$$p_1(x) = c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0$$

$$f(x) = \frac{\ln x e^{\sqrt{x}}}{\ln(1 + e^x) + x^3}$$



$$p_1(x) = c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0$$

$$p_2(x) = c_4 e^{-x^2} + c_3 e^{-x} + c_2 x^2 + c_1 x + c_0$$



# Interpolação polinomial

A **função interpolante** é assumida como sendo

$$p(x) = c_n x^n + \cdots + c_1 x + c_0$$

i.e., as **funções base** são da forma  $\phi_j(x) = x^j$  (base monomial).





# Interpolação polinomial

A **função interpolante** é assumida como sendo

$$p(x) = c_n x^n + \cdots + c_1 x + c_0$$

i.e., as **funções base** são da forma  $\phi_j(x) = x^j$  (base monomial).

$$\begin{bmatrix} x_0^n & \cdots & x_0 & 1 \\ x_1^n & \cdots & x_1 & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_n^n & \cdots & x_n & 1 \end{bmatrix} \begin{bmatrix} c_n \\ c_{n-1} \\ \vdots \\ c_0 \end{bmatrix} = \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_0 \end{bmatrix}$$

Sistema linear de Vandermonde (solução única)



## Experimento computacional 2

```
clc; clear; close all

xdat = [1;2;4]; ydat = [1;3;3];

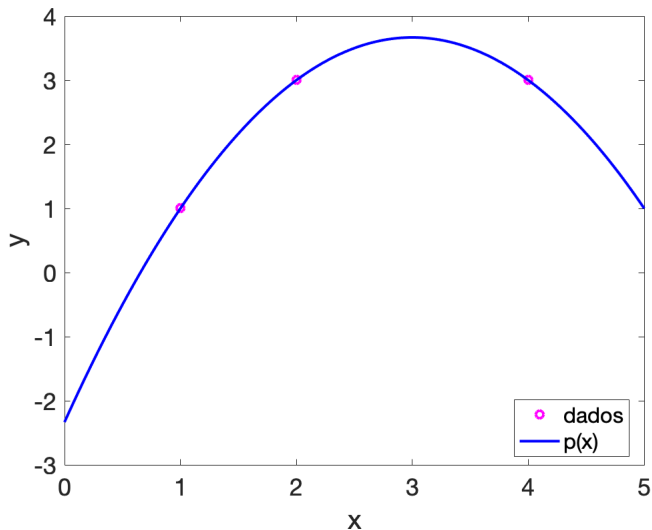
A = vander(xdat)
xsol = A\ydat

p = @(x) polyval(xsol, x);
xgrid = 0:0.01:5; ygrid = p(xgrid);

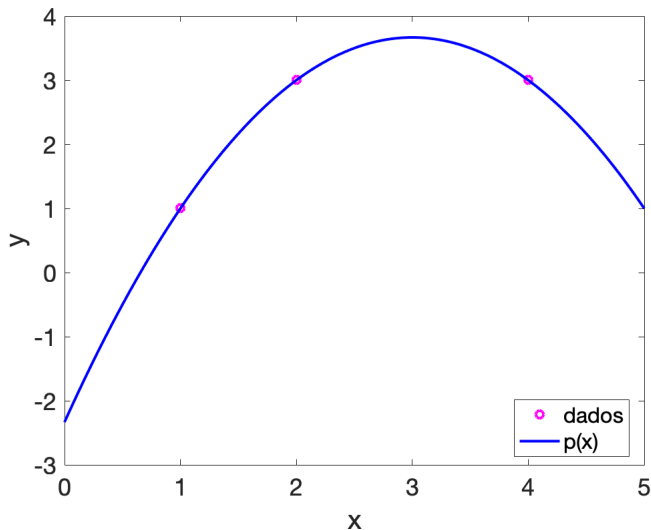
plot(xdat,ydat,'om',xgrid,ygrid,'-b','LineWidth',2)
xlabel('x'); ylabel('y');
set(gca,'FontSize',18);
legend('dados','p(x)','Location','south')
```



# Experimento computacional 2



# Experimento computacional 2



$$p(x) = -\frac{2}{3}x^2 + 4x - \frac{7}{3}$$



# Fundamentação teórica

## Teorema (Existência e unicidade do polinômio interpolante)

Para qualquer conjunto real com  $n + 1$  pontos (abscissas distintas)

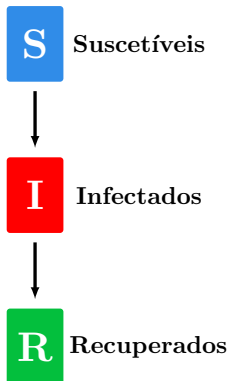
$$(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n),$$

existe um único polinômio  $p(x)$  de grau  $\leq n$  tal que

$$p(x_j) = y_j, \quad j = 0, 1, \dots, n.$$



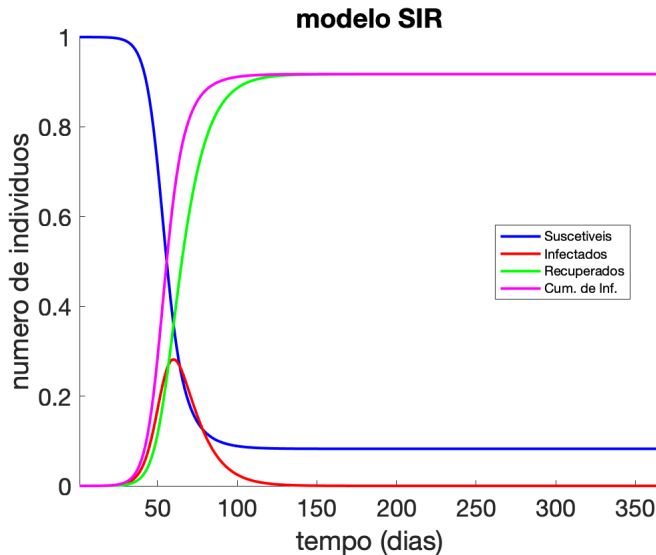
# Simulador de uma epidemia



$$\begin{aligned}\frac{dS}{dt} &= -\beta S \frac{I}{N} \\ \frac{dI}{dt} &= \beta S \frac{I}{N} - \gamma I \\ \frac{dR}{dt} &= \gamma I \\ &+ \text{condições iniciais}\end{aligned}$$

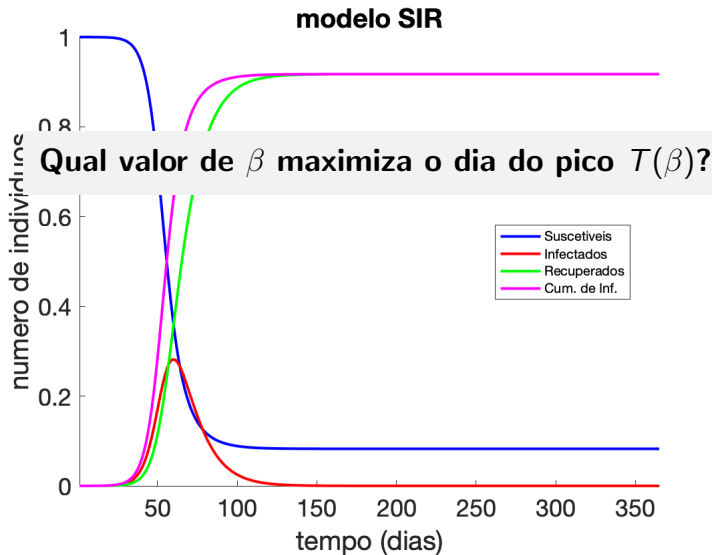
\*Simulador disponível em <http://www.EpidemicCode.org>

# Simulador de uma epidemia



\* Simulação com  $N = 1$ ,  $\beta = 1/3$ ,  $\gamma = 1/10$ ,  $R(0) = 0$ ,  $I(0) = 10^{-6}$ ,  $S(0) = N - I(0) - R(0)$

# Simulador de uma epidemia



\* Simulação com  $N = 1$ ,  $\beta = 1/3$ ,  $\gamma = 1/10$ ,  $R(0) = 0$ ,  $I(0) = 10^{-6}$ ,  $S(0) = N - I(0) - R(0)$



# Interpolando dados de um simulador

```
clc; clear; close all

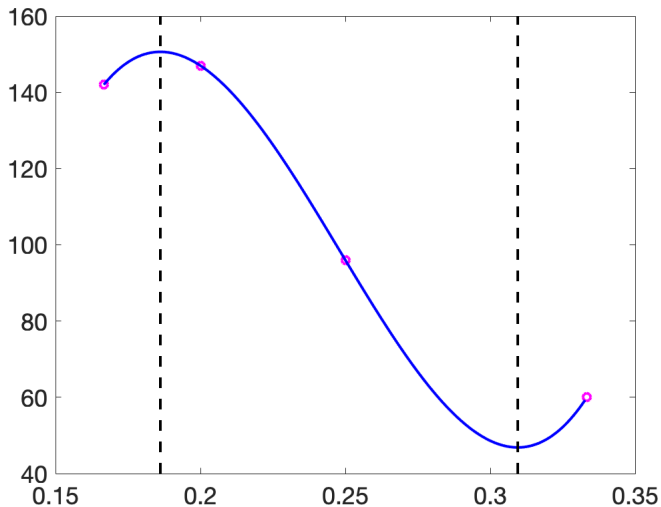
beta    = [1/3 1/4 1/5 1/6];
T_beta  = [60 96 147 142];

coef_T   = polyfit(beta,T_beta,3);
coef_dT  = polyder(coef_T);
beta_ast = roots(coef_dT)

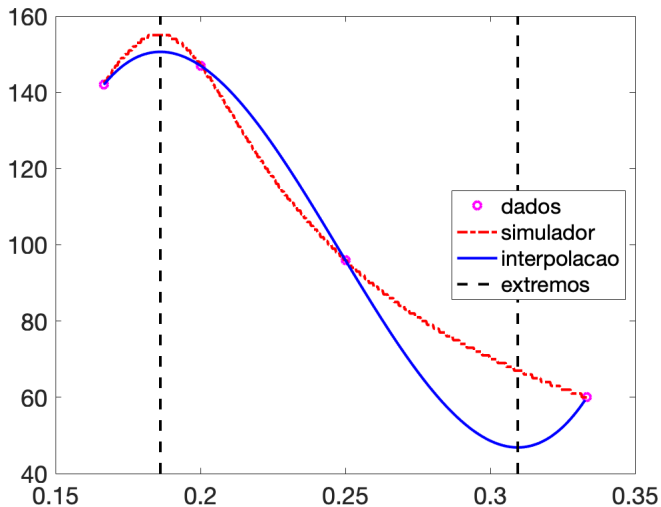
x1 = 1/6:0.001:1/3;
p = @(x) polyval(coef_T,x);
y1 = p(x1);
figure(1)
plot(beta,T_beta,'om',x1,y1,'-b')
hold on
plot([beta_ast(1) beta_ast(1)], [40 160], '-k')
plot([beta_ast(2) beta_ast(2)], [40 160], '-k')
hold off
```



# Interpolando dados de um simulador



# Interpolando dados de um simulador



# Observações sobre interpolação monomial

- Os coeficientes  $c_j$ 's não têm uma relação óbvia com os valores  $y_j$ , o que dificulta sua interpretação;
- Se adicionarmos um novo ponto aos dados, os coeficientes precisarão ser recalculados para obter o novo polinômio interpolante;
- Quando o intervalo de interpolação for muito amplo, ou  $n$  for suficientemente grande, a matriz de Vandermonde pode ser mal condicionada (induzindo erros na solução numérica do sistema);
- Se  $n$  for suficientemente grande, o custo computacional de resolver o sistema linear  $\sim 2/3 n^3$ , ou a quantidade de memória para armazenar o mesmo  $n^2 + 2n$ , podem ser questões críticas.



# Observações sobre interpolação monomial

- Os coeficientes  $c_j$ 's não têm uma relação óbvia com os valores  $y_j$ , o que dificulta sua interpretação;
- Se adicionarmos um novo ponto aos dados, os coeficientes precisarão ser recalculados para obter o novo polinômio interpolante;
- Quando o intervalo de interpolação for muito amplo, ou  $n$  for suficientemente grande, a matriz de Vandermonde pode ser mal condicionada (induzindo erros na solução numérica do sistema);
- Se  $n$  for suficientemente grande, o custo computacional de resolver o sistema linear  $\sim \frac{2}{3}n^3$ , ou a quantidade de memória para armazenar o mesmo  $n^2 + 2n$ , podem ser questões críticas.

**Todas essas questões podem ser contornadas  
utilizando outras funções base!**



# Interpolação de Lagrange

A **função interpolante** é assumida como sendo

$$p(x) = y_0 L_0^n(x) + y_1 L_1^n(x) + \cdots + y_n L_n^n(x),$$

onde os **polinômios de Lagrange** são dados por

$$L_j^n(x) = \prod_{\substack{i=0 \\ i \neq j}}^n \frac{x - x_i}{x_j - x_i}.$$

Note que  $L_j^n(x_i) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$



# Interpolação de Lagrange

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} c_n \\ c_{n-1} \\ \vdots \\ c_0 \end{bmatrix} = \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_0 \end{bmatrix}$$

Sistema linear com solução trivial ( $c_j = y_j$ )



# Interpolação de Lagrange

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} c_n \\ c_{n-1} \\ \vdots \\ c_0 \end{bmatrix} = \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_0 \end{bmatrix}$$

Sistema linear com solução trivial ( $c_j = y_j$ )

**Não é preciso resolver um sistema linear para construir o polinômio de Lagrange, basta construir as funções base.**





# Experimento computacional 3

```
clc; clear; close all

xdat = [1;2;4]; ydat = [1;3;3];

L0 = @(x) (1/3)*(x-2).*(x-4);
L1 = @(x) -(1/2)*(x-1).*(x-4);
L2 = @(x) (1/6)*(x-1).*(x-2);
p = @(x) ydat(3)*L2(x) + ydat(2)*L1(x) + ydat(1)*L0(x);
xgrid = 0:0.01:5; ygrid = p(xgrid);
```

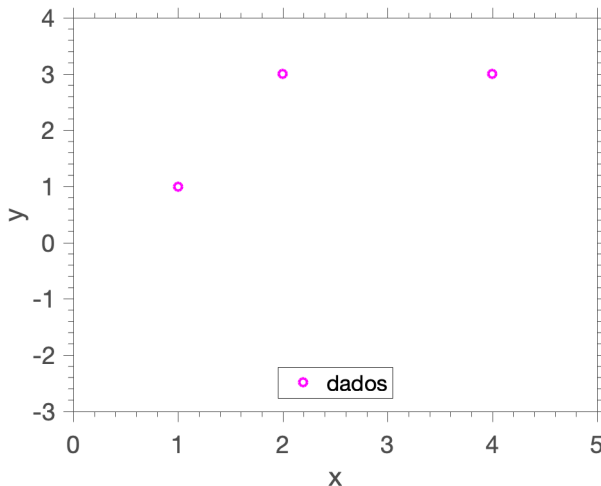


## Experimento computacional 3

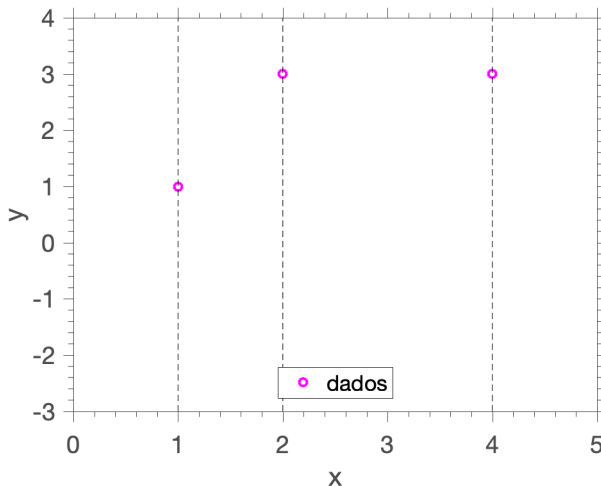
```
plot(xdat,ydat,'om',xgrid,ygrid,'-b','LineWidth',2)
hold on
plot(xgrid,L2(xgrid),'—r','LineWidth',1)
plot(xgrid,L1(xgrid),'-c','LineWidth',1)
plot(xgrid,L0(xgrid),' :g','LineWidth',1)
plot([xdat(1),xdat(1)],[-3,4],'—k','LineWidth',0.5);
plot([xdat(2),xdat(2)],[-3,4],'—k','LineWidth',0.5);
plot([xdat(3),xdat(3)],[-3,4],'—k','LineWidth',0.5);
hold off
xlabel('x'); ylabel('y');
set(gca,'FontSize',18);
legend('dados','p(x)','L2','L1','L0','Location','best')
```



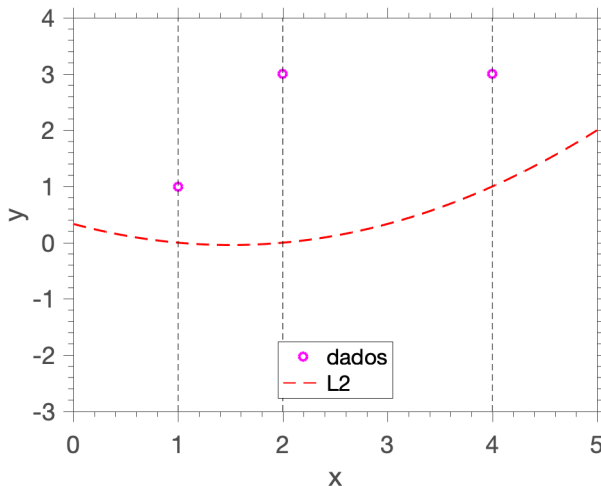
# Experimento computacional 3



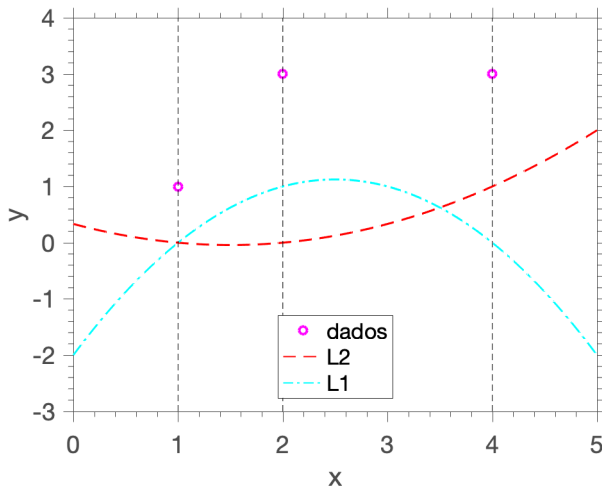
# Experimento computacional 3



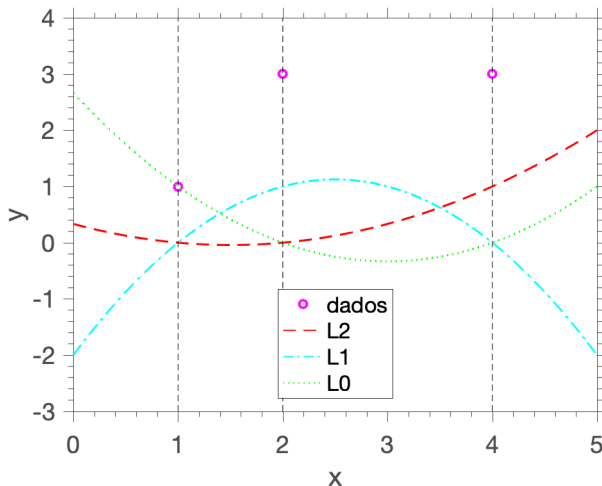
# Experimento computacional 3



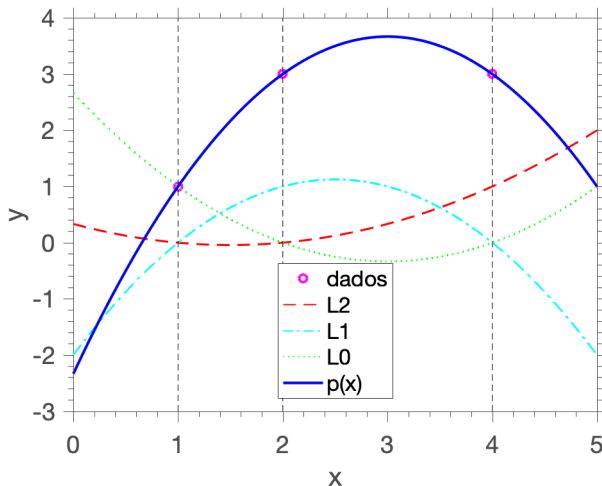
# Experimento computacional 3



# Experimento computacional 3

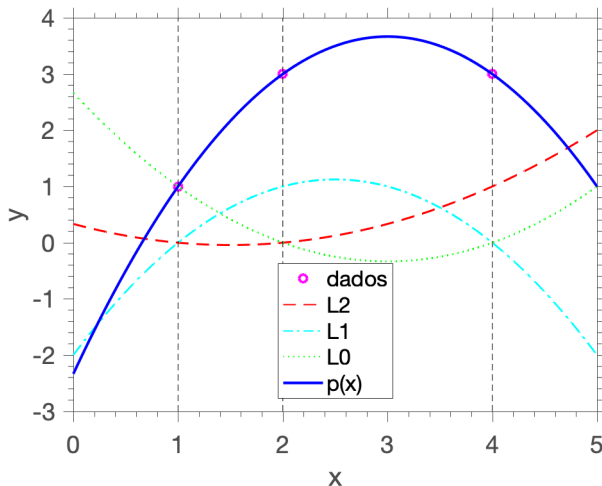


# Experimento computacional 3





# Experimento computacional 3



$$p(x) = -\frac{2}{3}x^2 + 4x - \frac{7}{3}$$



# Implementação em GNU Octave

```
function y = interplagrange(xdat,ydat,x)
    n = length(xdat);
    L = ones(n,length(x));
    for i = 1:n
        for j = 1:n
            if i ≠ j
                L(i,:) = L(i,:).*(x-xdat(j))./(xdat(i)-xdat(j));
            end
        end
    end
    y = ydat'*L;
end
```



# Experimento computacional 3 (revisado)

```
clc; clear; close all

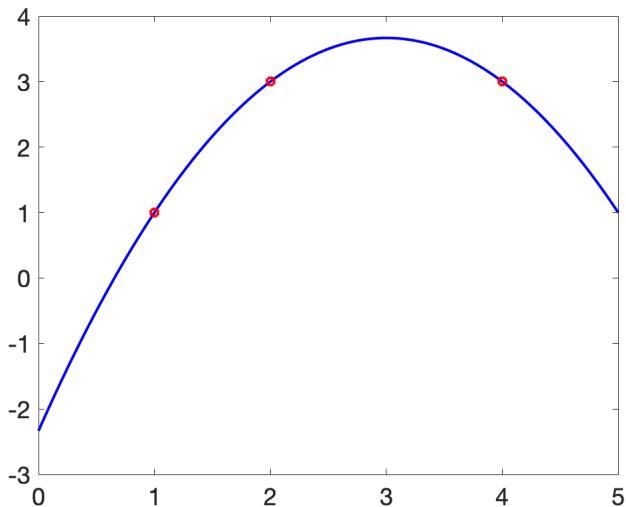
xdat = [1; 2; 4];
ydat = [1; 3; 3];

xgrid = 0:0.01:5;
ygrid = interplagrange(xdat,ydat,xgrid);

plot(xgrid,ygrid,'b',xdat,ydat,'or')
```



## Experimento computacional 3 (revisado)



# Observações sobre interpolação de Lagrange

- Os coeficientes agora têm um significado claro, eles são iguais aos valores  $y_j$ ;
- Se adicionarmos um novo ponto aos dados, os coeficientes precisarão ser recalculados para obter o novo polinômio interpolante;
- Como a matriz do sistema é a identidade, o processo de construção do polinômio é muito estável. Na prática não se resolve o sistema diagonal, existe um algoritmo bem eficiente baseado em interpolação baricêntrica.



# Interpolação de Newton

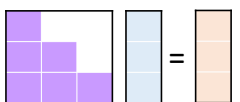
A **função interpolante** é assumida como sendo

$$p(x) = c_0 + c_1(x - x_0) + \cdots + c_n(x - x_0)(x - x_1) \cdots (x - x_{n-1}),$$

i.e., as **funções base** são da forma

$$\phi_j(x) = \prod_{i=0}^{n-1} (x - x_i).$$

Os **coeficientes** são definidos por um **sistema triangular inferior**

$$\begin{array}{ccc} A & \mathbf{x} & \mathbf{b} \\ n \times n & n \times 1 & n \times 1 \end{array}$$




# Experimento computacional 4

```
clc; clear; close all

xdat = [1;2;4]; ydat = [1;3;3];

p0 = @(x) 1*ones(size(x));
p1 = @(x) p0(x) + 2*(x-1);
p2 = @(x) p1(x) - (2/3)*(x-1).*(x-2);
xgrid = 0:0.01:5;
```



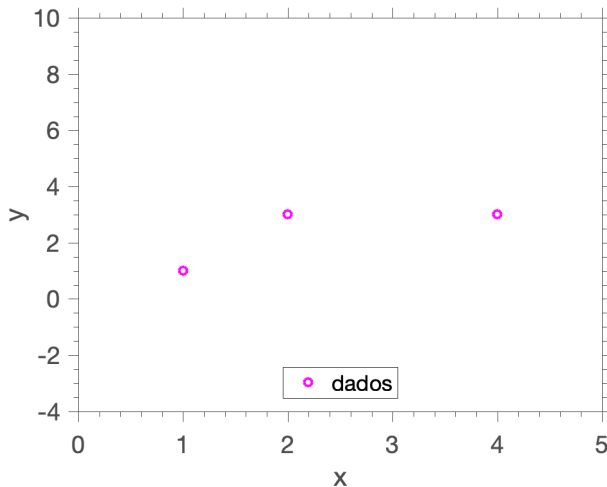
## Experimento computacional 4

```
plot(xdat,ydat,'om','LineWidth',2)
hold on
plot(xgrid,p0(xgrid),'—r','LineWidth',2)
plot(xgrid,p1(xgrid),'—c','LineWidth',2)
plot(xgrid,p2(xgrid),'—b','LineWidth',2)
plot([xdat(1),xdat(1)],[-4,10],'—k','LineWidth',0.5);
plot([xdat(2),xdat(2)],[-4,10],'—k','LineWidth',0.5);
plot([xdat(3),xdat(3)],[-4,10],'—k','LineWidth',0.5);
hold off
xlabel('x'); ylabel('y');
set(gca,'FontSize',18);
xlim([0 5]); ylim([-4 10]);
legend('dados','p_0(x)','p_1(x)','p_2(x)','Location','best')
```

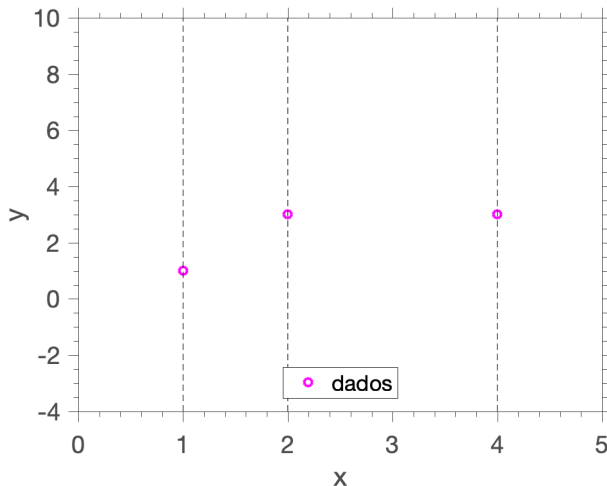




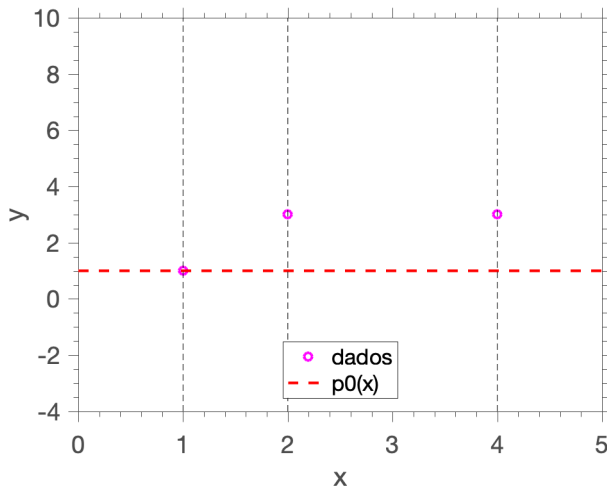
# Experimento computacional 4



# Experimento computacional 4



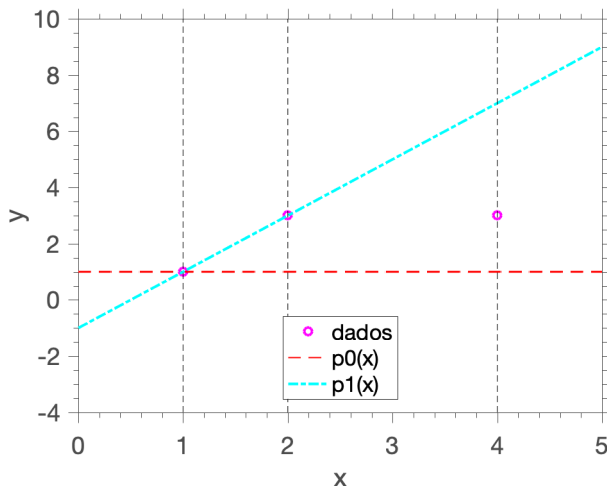
# Experimento computacional 4



$$p_0(x) = 1$$

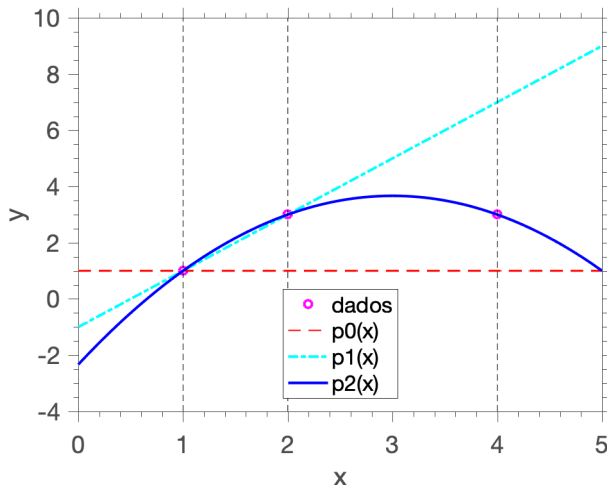


## Experimento computacional 4



$$p_1(x) = p_0(x) + c_1(x - 1) = 1 + 2(x - 1)$$

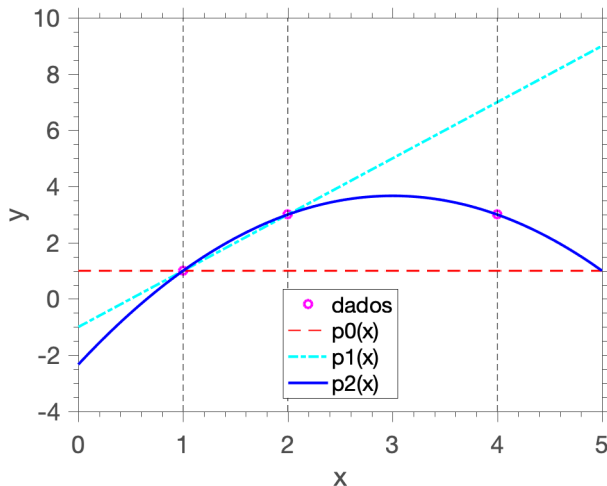
# Experimento computacional 4



$$p_2(x) = p_1(x) + c_2(x-1)(x-2) = 1 + 2(x-1) - \frac{2}{3}(x-1)(x-2)$$



# Experimento computacional 4



$$p_2(x) = -\frac{2}{3}x^2 + 4x - \frac{7}{3}$$



# Implementação em GNU Octave

```
function y = interpnewton(x,xdat,coef)
    n = length(xdat);
    y = coef(n)*ones(size(x));
    for j=n-1:-1:1
        y = y.*(x - xdat(j)) + coef(j);
    end
end
```



# Implementação em GNU Octave

```
function [coef,tab] = divdif(xdat,ydat)
    n = length(xdat)-1;
    tab = zeros(n+1,n+1);
    xdat = shiftdim(xdat);
    ydat = shiftdim(ydat);
    tab(1:n+1,1) = ydat;
    for k = 2:n+1
        num = tab(k:n+1,k-1)-tab(k-1:n,k-1);
        den = xdat(k:n+1)-xdat(1:n+1-k+1);
        tab(k:n+1,k) = num./den;
    end
    coef = diag(tab);
end
```





# Implementação em GNU Octave

```
function [coef,tab] = divdifadd(xdat,ydat,tab)
    n = length(xdat)-1;
    tab = [tab zeros(n,1); ydat(n+1) zeros(1,n)];
    for k=2:n+1
        num = tab(n+1,k-1)-tab(n,k-1);
        den = xdat(n+1)-xdat(n+1-k+1);
        tab(n+1,k) = num/den;
    end
    coef = diag(tab);
end
```



## Experimento computacional 4 (revisado)

```
clc; clear; close all

xdat1 = [1; 2; 4];    ydat1 = [1; 3; 3];
xdat2 = [1; 2; 4; 5]; ydat2 = [1; 3; 3; 2];

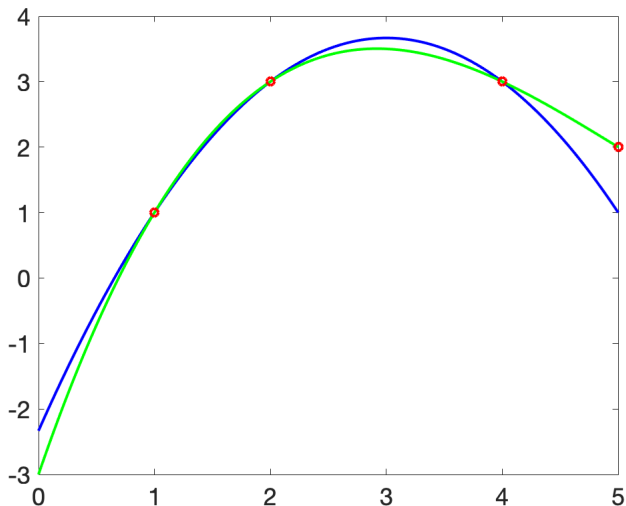
[c1,tab1] = divdif(xdat1,ydat1);
[c2,tab2] = divdifadd(xdat2,ydat2,tab1);

xgrid = 0:0.01:5;
ygrid1 = interpnewton(xgrid,xdat1,c1);
ygrid2 = interpnewton(xgrid,xdat2,c2);

figure(1)
plot(xgrid,ygrid1,'b')
hold on
plot(xgrid,ygrid2,'g',xdat2,ydat2,'or')
hold off
```



## Experimento computacional 4 (revisado)



# Custo e características do problema de interpolação

base	$\phi_j(x)$	custo de construção	custo de avaliação	características notáveis
monomial	$x^j$	$\sim \frac{2}{3} n^3$	$\sim 2 n$	simplicidade
Lagrange	$\prod_{\substack{i=0 \\ i \neq j}}^n \frac{x - x_i}{x_j - x_i}$	$\sim 2 n^2$	$\sim 5 n$	interpretabilidade e estabilidade
Newton	$\prod_{i=0}^{j-1} (x - x_i)$	$\sim \frac{3}{2} n^2$	$\sim 2 n$	adaptatividade



# Erro na interpolação polinomial

## Teorema (Erro da interpolação polinomial)

Se o polinômio  $p_n(x)$  interpola a função  $f \in C^{n+1}([a, b])$  nos pontos  $a = x_0 < x_1 < \dots < x_n = b$ , então o erro da interpolação é dado por

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^n (x - x_i),$$

onde  $\xi \in [a, b]$ .



## Como citar esse material?


A. Cunha Jr, *Interpolação*,  
Universidade do Estado do Rio de Janeiro – UERJ, 2021.



 @AmericoCunhaJr

 @AmericoCunhaJr

 @AmericoCunhaJr

 @AmericoCunhaJr

Essas notas de aula podem ser compartilhadas nos termos da licença Creative Commons BY-NC-ND 3.0, com propósitos exclusivamente educacionais.

