Nome: Livia Barbosa Fonseca
DRE: 118039721

Questão 1.1-)

1. o Método da bisseção com intervalo menor que 10^{-3}

Feito na lista anterior

2. o Método de Newton com 20 passos.

Nesse exercício, vamos calcular o ln(3) pelo Método de Newton com 20 passos. Sabendo que sabemos calcular e^x para qualquer x dado e sabendo que podemos transformar ln(3) em $f(x) = e^x - 3$, vamos utilizar essa função e o código acima para nos ajudarmos a calcular o valor do ln(3) em um intervalo de 10^{-3}. Assim, vamos criar a seguinte função:

```julia
function ln3(número_de_iterações,chute) #Função que recebe um número de iterações e o chute inicial
    for i=1:número_de_iterações
        chute=chute-((exp(chute)-3)/exp(chute)) #atualizando o chute "andando" pela reta tangente
    end
    return chute #aproximação do ln3
end
```

Se passarmos o número de iterações desejadas pelo problema e um chute inicial igual a 1 temos que:

```julia
ln3(20,1)
```

```
1.0986122886681096

julia>
```

Dessa forma, obtemos que nossa aproximação de ln(3) é 1.0986122886681096

3. o Polinômio de Taylor com erro máximo de 10^{-3}

Agora vamos calcular o ln(3) via polinômio de Taylor. Vamos utilizar o polinômio de Taylor de grau 2 para isso:

$$f(x) - (f(a) + f'(a)(x - a) + \frac{f''(a)(x-a)^2}{2} \leq \frac{M(x-a)^3}{6}$$

Considerando que:

$$f(x) = \ln(x)$$

$$f'(x) = \frac{1}{x}$$

$$f''(x) = \frac{-1}{x^2}$$

E que a = e e x = 3
Colocando no Julia temos:

```
aproximação = 1+((3-exp(1))/exp(1))-((3-exp(1))^2/(2*exp(2)))
```

```
1.0982678724638968

julia> 
```

Agora temos que analisar se o erro está dentro do que foi pedido: $\frac{M(x-a)^3}{6}$

```
M = 2/exp(3) ;Valor de M
P = M* (3-exp(1))^3/6 ;Valor do erro
```

Podemos observar que o erro está dentro do esperado

```
0.00037105636225489175

julia> 
```

Assim, encontramos que nossa aproximação é de 1.0982678724638968.

4. a interpolação polinomial de grau 1 e estime o erro máximo.
Nesse exercício vamos calcular o ln(3) pelo método da interpolação polinomial de grau 1.
Para isso, vamos utilizar a seguinte função em Julia:

```
#Função que realiza a interpolação com um conjunto de dois pontos x e y
function interpolação_dois_pontos(x,y)
     #Cria a matriz V (primeira linha: todos os pontos 'x' elevados a
0,segunda linha: todos os pontos de 'x' elevados a 1)
    V=[x.^0 x.^1]
    #Calcula o sistema linear Vc = y
    c = V\y
    return c #vetor de coeficientes do polinômio
end
```

Utilizando os seguintes pontos, temos:

x	e	$e^{1.1}$

ln(x)	1	1.1

```
x=[exp(1), exp(1.1)]
y=[1, 1.1]
interpolação_dois_pontos(x,y)
```

Encontramos os coeficientes do polinômio:

```
2-element Vector{Float64}:
 0.04916680552249564
 0.34979198423164165
```

Dessa forma, vamos ter

$$f(x) = a + bx$$
$$f(x) = 0.04916680552249564 + 0.34979198423164165x$$

Como queremos o ln(3), vamos substituir x = 3:

$0.04916680552249564 + 0.34979198423164165 * 3 =$ 1.0985427582174205

Logo, temos que nossa aproximação é de 1.0985427582174205. Agora, vamos calcular o erro:

$$\frac{M}{(N+1)!}(X - X0)\dots(X - Xn)$$

Substituindo pelos nossos valores, temos:

```
(-1/(exp(1))^2)*(1/2)*(3-exp(1))*(3-exp(1.1))
```

```
7.941776548122973e-5

julia> 
```

E assim encontramos o nosso erro.

5. a interpolação polinomial de grau 2 e estime o erro máximo.
Agora vamos resolver o exercício utilizando a interpolação com três pontos. E para isso, vamos utilizar a seguinte função no Julia:

```
#Importação necessária para o funcionamento da função interpolação
using LinearAlgebra
#Função que dado um conjunto de três pontos, realiza uma interpolação e
#retorna um vetor contendo os coeficientes do polinômio interpolador
calculado
function interpolação(x,y)
```

```
#Cria a matriz V (primeira linha: todos os pontos 'x' elevados a 0,
#segunda linha: todos os pontos de 'x' elevados a 1 e terceira linha:
#todos os pontos de 'x' elevados a 2)
V=[x.^0 x.^1 x.^2]
#Calcula o sistema linear Vc = y
c=V\y
return c #Retorna o vetor dos coeficientes do polinômio
end
```

Utilizando os pontos:

x	1	e	$e^{1.1}$
ln(x)	0	1	1.1

Calculando a interpolação temos:

```
x=[1; exp(1); exp(1.1)]
y=[0,1,1.1]
interpolação(x,y)
```

```
3-element Vector{Float64}:
 -0.8968924897190071
  1.012743531929815
 -0.11585104221080783
```

Dessa forma, vamos ter:

$$f(x) = a + bx + cx^2$$

$$f(x) = -0.8968924897190071 + 1.012743531929815x - 0.11585104221080783x^2$$

Como queremos o ln(3), vamos substituir x = 3 e encontrar a aproximação:

$$-0.8968924897190071 + 1.012743531929815 * 3 - 0.11585104221080783 * 3^2 =$$

1.0986787261731674

```
-0.8968924897190071 + (1.012743531929815*3) + (-0.11585104221080783*9) #aproximação com três pontos
0.3s
1.0986787261731674
```

Agora, vamos encontrar erro:

```
#calculando o erro M/(n+1)! (x - x0)..(x-xn)
(2/(exp(1))^3)*(1/24)*(3-1)*(3-exp(1))*(3-exp(1.1)) #erro de interpolaçao de 3 pontos
```

✓ 0.1s

```
-9.738721061439817e-6
```

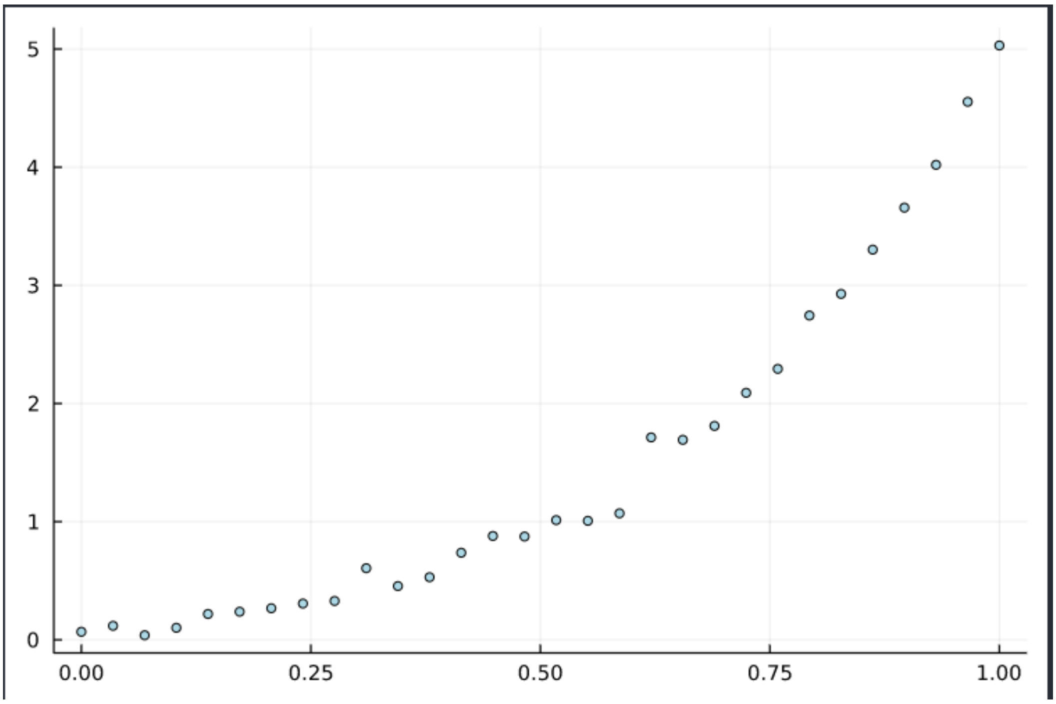
Logo, descobrimos que o erro é de -9.738721061439817e-6

Questão 1.2 -)

1.) Nesse exercício, vamos gerar aleatoriamente 30 pontos de um polinômio de grau 5. Para isso, vamos aplicar o seguinte programa no julia:

```
#Função necessária para gerar números aleatórios
Random.seed!(0)
# Número de pontos desejados
n = 30
# Gerando números aleatórios para x
x = range(0, 1, length=n)

#Definindo cada ordem da função
x1(x) = x
x2(x) = x^2
x3(x) = x^3
x4(x) = x^4
x5(x) = x^5

#Função modelada de ordem 5 com ruído
y =  x1.(x) + x2.(x) + x3.(x) + x4.(x) + x5.(x) + randn(n)*0.1

#Realizando a plotagem dos pontos gerados aleatoriamente que estão
guardados nos vetores x e y
scatter(x, y, c=:lightblue, ms=3, leg=false)
```

Podemos observar que será gerado o seguinte plot, onde encontramos todos os 30 pontos pedidos:



2.) Nesse exercício, vamos criar uma função no Julia que realiza a regressão polinomial com polinômios de grau 0 até 29.

```
#Função que recebe um conjunto de números x e y e o grau do polinômio que desejado e cria a matriz de vandermonde dele
function vandermonde(x,y,grau)
#Recebendo o tamanho do vetor x
    n,=size(y)
#Criando uma matriz zerada do tamanho desejado
    V=zeros(n,grau+1)
#Looping que vai até o número de linhas da matriz
    for i=1:n
#Looping que vai até o número de colunas desejadas na matriz (grau da matriz + 1)
        for j=1:(grau+1)
#Criando ponto a ponto da matriz (posição atual x elevado a  j-1)
            V[i,j]=x[i]^(j-1)
        end
    end
    #Retorna a matriz desejada
    return V
end

#Função que recebe o grau desejado para a regressão
function regressão(grau)
    #Criando a matriz de vandermonde
    V = vandermonde(x,y,grau)
```

```
    #Resolvendo o sistema: quando o sistema a ser resolvido não é possível de ser resolvido, o contra barra "\" do Júlia já disponibiliza a solução pelo método dos mínimos quadrados
    C = V\y

    #Criando o polinômio desejado realizando a soma entre os graus do polinômio (constante *x^k, onde k é o grau da parcela)
    f(x) = sum([C[i+1]*x^i for i in 0:grau])
    #Retorna o polinômio criado
    return f
end
```

Com essa função dada, podemos realizar a regressão polinomial com polinômios de graus 0 até 29.

3.) É possível realizar a regressão com um polinômio de grau maior do que 29 no Julia. Quando possuímos uma quantidade maior de coeficientes do que pontos, o Julia consegue realizar a regressão e informar um polinômio, que possui o quadrado dos coeficientes da função os menores possíveis. Dessa forma, teremos várias soluções para o problema visto que possuímos mais coeficientes que pontos.

Outra coisa que podemos observar é que foram realizadas regressões com graus maiores do que 29 e o erro contido nessas contas não foi ruim:

```
grau_29(x) = regressão(29)(x)
grau_50(x) = regressão(50)(x)
grau_100(x) = regressão(100)(x)
```

Utilizando a função erro_total para calcular o erro temos:

```
#Função que calcula o erro de dados os pontos x e y e o modelo desejado
function erro_total(x,y,modelo)
    #Pegando o tamanho do vetor y
    n,=size(y)
    #Iniciando uma variável com zerada
    S=0
    #Lopping que vai até a quantidade de elementos y, realizando o somatório da diferença entre y[i] e x[i] do modelo elevado ao quadrado
    for i=1:n
        S=S+(y[i]-modelo(x[i]))^2
    end
    return sqrt(S)  #Retornando o erro na ordem de y
end
```

```
erro_total(x,y,grau_29)
✓ 0.3s
8.74890691329719
```

```
erro_total(x,y,grau_50)
✓ 0.4s
0.2663720542804178
```

```
erro_total(x,y,grau_100)
✓ 0.2s
0.12832127106034172
```

Podemos observar que o erro do modelo de grau 29, ficou muito grande, mas isso acontece devido ao ruído adicionado a função. Quando retiramos o ruído dela, ela funciona como o esperado. Retirando o ruído temos:

```
#Função necessária para gerar números aleatórios
Random.seed!(0)
# Número de pontos desejados
n = 30
# Gerando números aleatórios para x
x = range(0, 1, length=n)

#Definindo cada ordem da função
x1(x) = x
x2(x) = x^2
x3(x) = x^3
x4(x) = x^4
x5(x) = x^5

#Função modelada de ordem 5 SEM ruído
y =  x1.(x) + x2.(x) + x3.(x) + x4.(x) + x5.(x) + randn(n)*0
```

```
erro_total(x,y,grau_29)
```
✓ 0.9s
```
3.623537966664621e-15
```

```
erro_total(x,y,grau_50)
```
✓ 0.4s
```
2.1300995364179145e-15
```

```
erro_total(x,y,grau_100)
```
✓ 0.6s
```
2.1877126813899928e-15
```

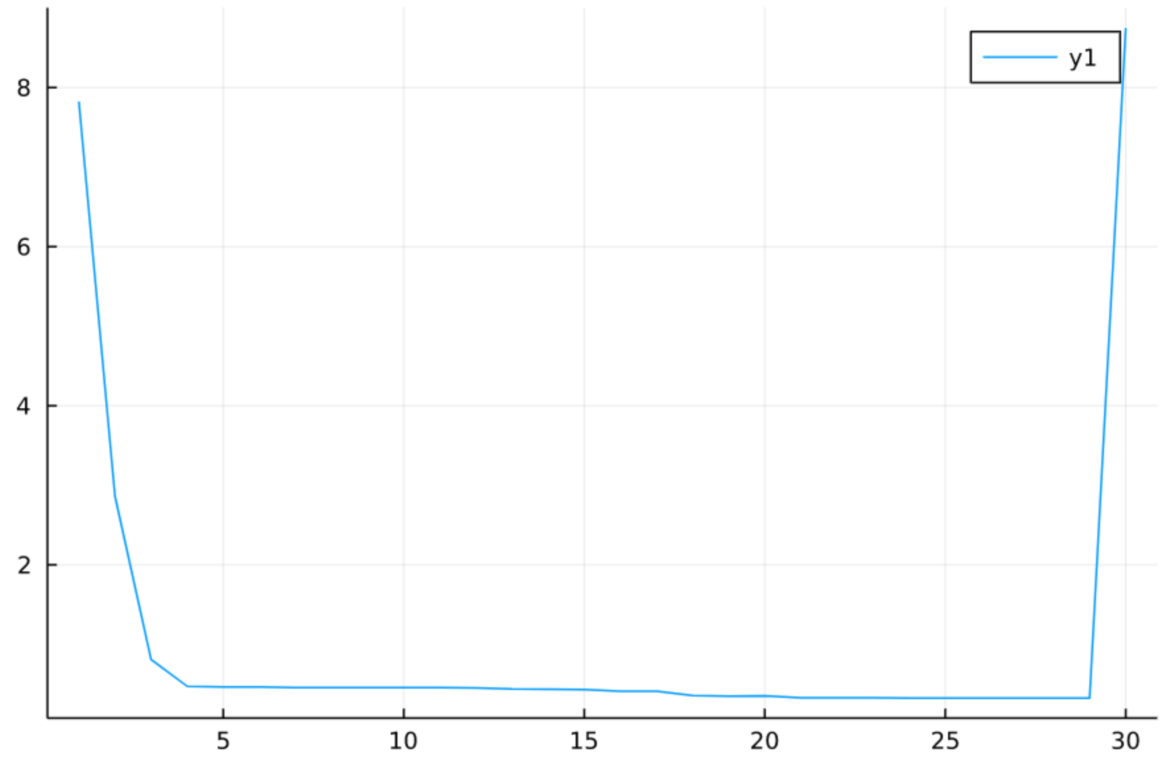
4.) Vamos realizar o plot do Erro total (eixo y) por grau (eixo x), para isso, vamos utilizar o seguinte programa:

```
using Plots
using Random
using LinearAlgebra

#Função necessária para gerar números aleatórios
Random.seed!(0)
# Número de pontos desejados
n = 30
# Gerando números aleatórios para x
x = range(0, 1, length=n)

#Definindo cada ordem da função
x1(x) = x
x2(x) = x^2
x3(x) = x^3
x4(x) = x^4
x5(x) = x^5

#Função modelada de ordem 5 COM ruído
y =  x1.(x) + x2.(x) + x3.(x) + x4.(x) + x5.(x) + randn(n)*0.1
```

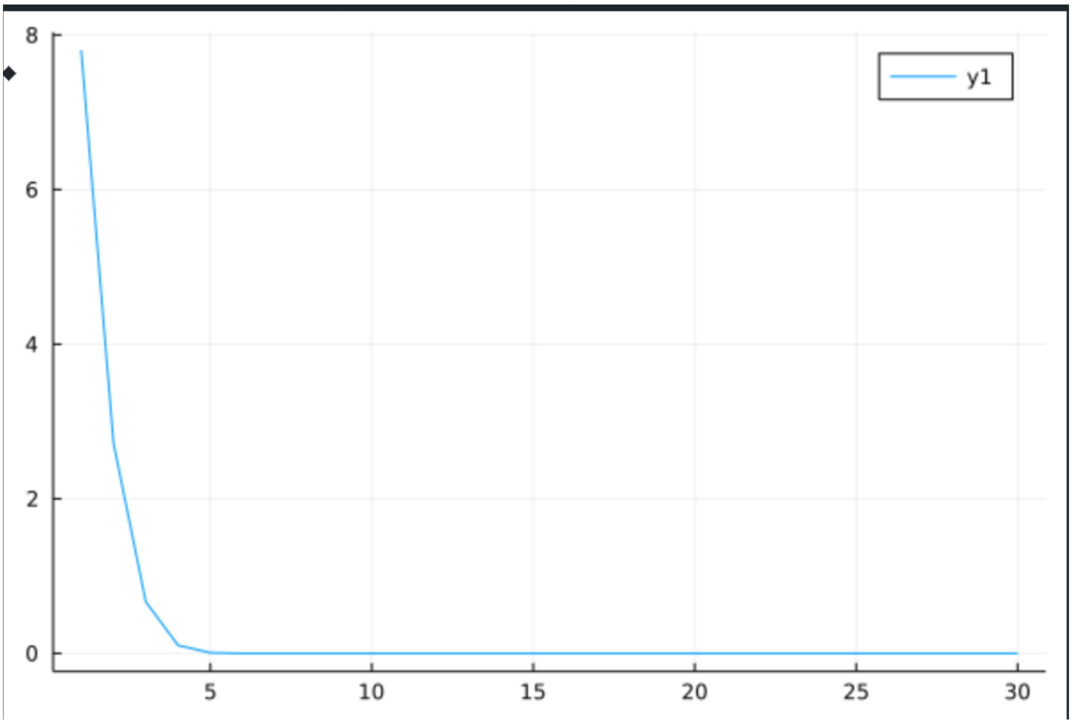
```
#Função que recebe um conjunto de números x e y e o grau do polinômio
que desejado e cria a matriz de vandermonde dele
function vandermonde(grau)
    #Recebendo o tamanho do vetor x
        n,=size(y)
    #Criando uma matriz zerada do tamanho desejado
        V=zeros(n,grau+1)
    #Looping que vai até o número de linhas da matriz
        for i=1:n
    #Looping que vai até o número de colunas desejadas na matriz (grau
da matriz + 1)
            for j=1:(grau+1)
    #Criando ponto a ponto da matriz (posição atual x elevado a  j-1)
                V[i,j]=x[i]^(j-1)
            end
        end
        #Retorna a matriz desejada
        return V
    end
#Função que recebe o grau desejado para a regressão
function regressão_polinomio(grau)
    #Criando a matriz de vandermonde
    V = vandermonde(x,y,grau)
    #Resolvendo o sistema: quando o sistema a ser resolvido não é
possível de ser resolvido, o contra barra "\" do Julia já disponibiliza
a solução pelo método dos mínimos quadrados
    C = V\y

    #Criando o polinômio desejado realizando a soma entre os graus do
polinômio (constante *x^k, onde k é o grau da parcela)
    f(x) = sum([C[i+1]*x^i for i in 0:grau])
    #Retorna o polinômio criado
    return f
end

#Função que calcula o erro de dados os pontos x e y e o modelo desejado
function erro_total(x,y,modelo)
    #Pegando o tamanho do vetor y
    n,=size(y)
    #Iniciando uma variável com zerada
    S=0
    #Lopping que vai até a quantidade de elementos y, realizando o
somatório da diferença entre y[i] e x[i] do modelo elevado ao quadrado
```

```
    for i=1:n
        S=S+(y[i]-modelo(x[i]))^2
    end
    return sqrt(S)  #Retornando o erro na ordem de y
end

#Criando um vetor que contém os coeficientes das regressões
regressão_polinomios = [regressão_polinomio(i) for i in 0:29]

#Vetor que possui os erros dos polinômios de diferentes graus
erro_y = [erro_total(x,y,regressão_polinomios[i]) for i in 1:30]

#Plotando o gráfico do erro em função do grau dos polinômios
plot(erro_y)
```

Obtemos:



Podemos observar que quando chega no grau 29 o erro fica muito alto. O erro esperado no modelo de grau 29 esperado é algo próximo de zero e não é isso que está acontecendo. Porém, quando retiramos o ruído da função, o erro se comporta como o esperado:



Questão 1.3 -)
Da pesquisa realizada podemos retirar as seguintes equações abaixo. Todas são obtidas quando subtraímos o número de escolhas do filme a pelo filme b:

1. Toy story - Rocky = 11
2. De volta pro futuro - Curtindo a vida adoidado = 3
3. Os incríveis - Duna = 7
4. Batman begins - Harry Potter 1 = 2
5. Shrek - Duna = 9
6. Harry Potter - Rocky = 7
7. Toy story - De volta para o futuro = 5
8. Os incríveis - Harry potter 1 = 5
9. Curtindo a vida adoidado - Duna = 2
10. De volta para o futuro - Duna = 2
11. Shrek - Rocky = 11
12. Os incríveis - Batman Begins = 5
13. Toy story - Batman Begins = 3
14. Os incríveis - Curtindo a vida adoidado = 7

Podemos observar que temos 9 variáveis e 14 equações. Porém, vamos acrescentar mais uma variável arbitrária e absoluta para que consigamos resolver a matriz pelo método dos mínimos quadrados.

15. Toy story = 10

Agora, vamos construir a matriz desses sistemas:

$$
\begin{bmatrix}
1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
\text{Toy story} \\
\text{Rocky} \\
\text{De volta pro futuro} \\
\text{Curtindo a vida adoidado} \\
\text{Os incríveis} \\
\text{Duna} \\
\text{Batman begins} \\
\text{Harry Potter} \\
\text{Shrek}
\end{bmatrix}
=
\begin{bmatrix}
11 \\ 3 \\ 7 \\ 2 \\ 9 \\ 7 \\ 5 \\ 5 \\ 2 \\ 2 \\ 11 \\ 5 \\ 3 \\ 7 \\ 10
\end{bmatrix}
$$

Resolvendo esse sistema pelo Julia, temos:

```
#Matriz da esquerda
A =  [ 1   -1   0   0   0   0   0   0   0;
0    0   1  -1   0   0   0   0   0;
0    0   0   0   1  -1   0   0   0;
0    0   0   0   0   0   1  -1   0;
0    0   0   0   0  -1   0   0   1;
0   -1   0   0   0   0   0   1   0;
1    0  -1   0   0   0   0   0   0;
0    0   0   0   1   0   0  -1   0;
0    0   0   1   0  -1   0   0   0;
0    0   1   0   0  -1   0   0   0;
0   -1   0   0   0   0   0   0   1;
0    0   0   0   1   0  -1   0   0;
1    0   0   0   0   0  -1   0   0;
0    0   0  -1   1   0   0   0   0;
1    0   0   0   0   0   0   0   0]

#Matriz da direita
b = [11; 3; 7; 2; 9; 7; 5; 5; 2; 2; 11; 5; 3; 7; 10]

#Resolvendo o sistema
#Quando o sistema a ser resolvido não é possível de ser resolvido, o contra barra "\" do Julia já disponibiliza a solução pelo método dos mínimos quadrados*
M = A\b
```

*Lembrando que o Método dos Mínimos Quadrados é uma técnica que procura encontrar o melhor ajuste para um conjunto de dados tentando minimizar a soma dos quadrados das diferenças entre o valor estimado e os dados observados.

Assim, obtemos:

```
9-element Vector{Float64}:
 10.000000000000002
 -0.9419496166484088
  5.314348302300112
  3.437020810514791
 10.49069003285871
  2.506024096385544
  6.627601314348305
  5.392113910186201
 10.78203723986857
```

Esses valores retornados são os valores dos coeficientes de cada filme, dessa forma vamos encontrar o que possui maior valor, visto que queremos encontrar o filme favorito da turma. O que possui maior valor é o filme **Shrek**, logo ele é o favorito da turma.

Questão 1.4-)

Para descobrir em qual dia a pessoa vai pesar 110kg, vamos utilizar a técnica de regressão para realizar os cálculos. Antes de começar, vamos pensar em um modelo que melhor se adapta a função da perda de peso. Podemos observar pela imagem que a curva de perda é parabólica, onde nas primeiras semanas a pessoa que está em dieta perde muito peso e depois de algumas semanas essa variação vai se estabilizando.

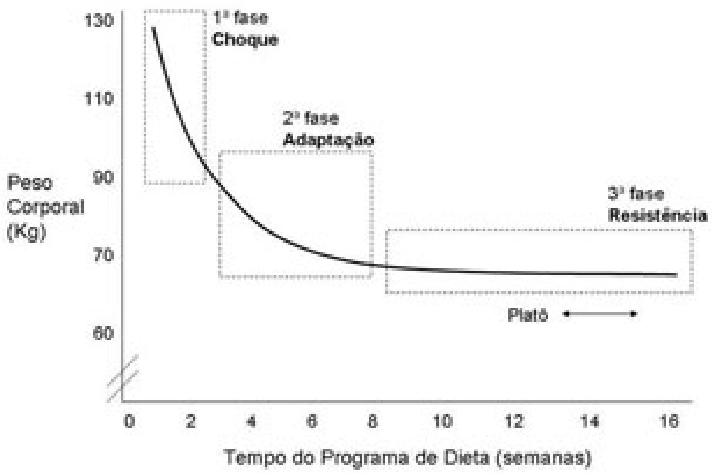


Figura 1 - Curva de perda de peso. A curva descreve um padrão parabólico em razão das adaptações fisiológicas provocadas pela restrição energética. Propõe-se a decomposição da curva em três segmentos correspondentes a três fases denominadas SAR (*Shock*/choque, *Adaptation*/adaptação e *Resistance*/resistência), porque as adaptações acontecem em sequência e a perda de peso torna-se mais difícil. Em síntese, a primeira fase denominada *Shock* é caracterizada pela significativa perda de peso e pequena diminuição no gasto energético total (GET), em razão da diminuição da termogênese induzida pela dieta (TID). A segunda fase, denominada *Adaptation* é caracterizada pela pequena perda de peso, aumento da fome, diminuição da atividade do sistema nervoso autônomo simpático e do GET, induzidos principalmente pela diminuição da concentração de leptina. A Terceira fase, denominada *Resistance* é caracterizada pelo aumento do tônus do sistema nervoso autônomo parassimpático e diminuição da termogênese sem exercício (ou adaptativa - NEAT), provocando um platô na perda de peso.

Antes de colocar os valores no Júlia, vamos transformar as datas de tal forma que o dia 26/10/2021 seja considerado o dia 1 da medição e assim sucessivamente. Lembrando que as medições não foram feitas em dias consecutivos. Dessa forma, temos:

```
using Plots
using Random
using LinearAlgebra

#Medições do peso
y=[120.6;121.6;120.8;121.4;121.1;121.1;120.4;120.3;120.8;120.6;119.6;119.8;118.7;120.5;120.1;120.2;120.7;121.7;120.7;120.7;120.3;119.4;119.1;120.2;120.7;120.1;119.7;119.2;119.4;119.5;119;118.9;118.7;118.3;118.5;118.6;118.8;118.5;118.3;117.8;118;119;118.4;116.9;117.5;117.4;117.6;118.1;117.3;117.6;117.7;117.6;117.3;118;117.8;117.5;119;117.6;116.8;116.6;116.9;116.1;116.1;115.8;115.6;116;115.4;115.5;115.3]
#Dias que foram realizadas as medições
x=[1;2;3;4;5;11;12;13;14;15;16;17;18;19;20;21;22;23;24;25;26;27;28;29;30;31;32;33;34;35;36;37;38;39;40;41;42;43;44;45;46;49;51;53;54;55;56;57;58;59;60;61;62;63;64;65;71;72;73;74;75;77;78;79;80;81;82;83;84]
```

Como estamos tratando de uma função não linear, exponencial, vamos ter que realizar a troca de variáveis a fim de conseguirmos realizar a regressão. Assim, vamos considerar a função:

$$f(x) = c1*e^{c2x}$$

Transformando ela em linear temos:

$$\ln(y) = \ln(c1*e^{c2x})$$
$$\ln(y) = \ln(c1) + \ln(e^{c2x})$$
$$\ln(y) = \ln(c1) + c2x$$

Realizando a troca de variáveis:

y_barra = ln(y)
x_barra = x
c1_barra = ln(c1)
c2_barra = c2

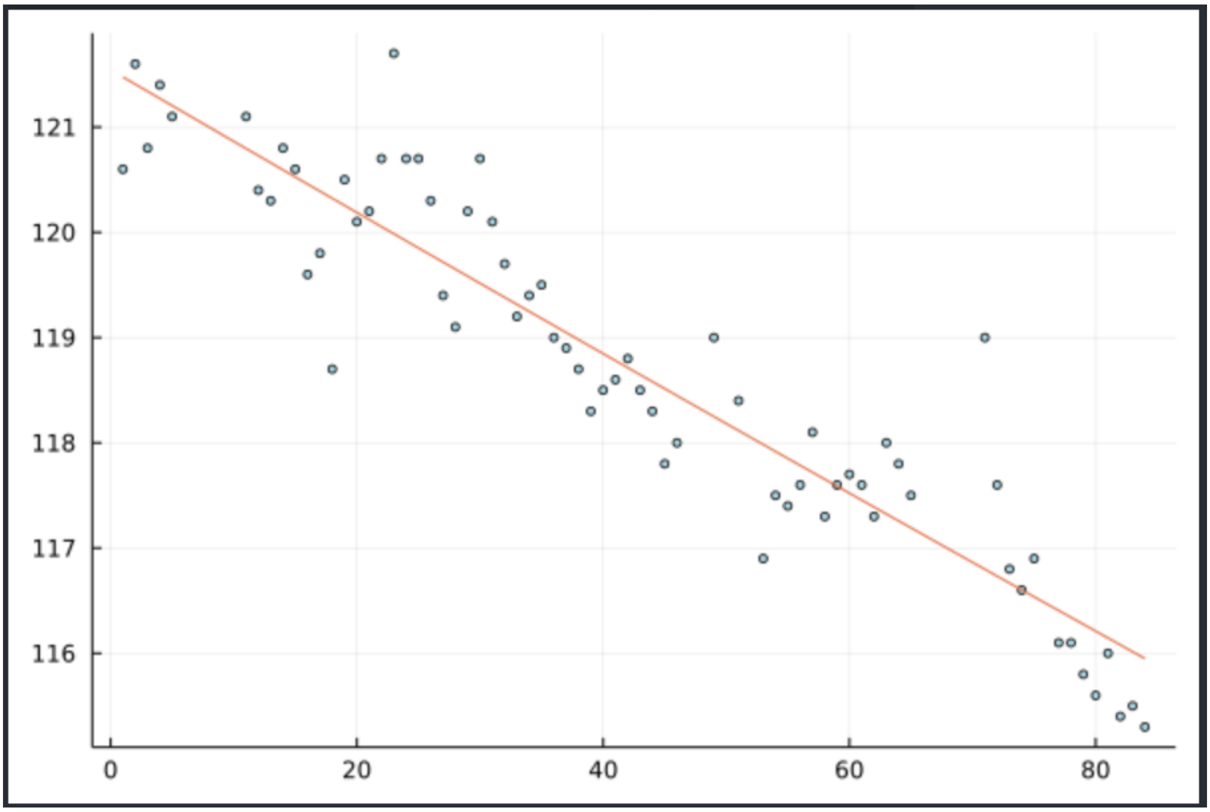
Encontramos:

y_barra = c1_barra + x_barra*c2_barra

```
#Troca de variáveis "indo para mundo linear"
x_barra=x
y_barra=log.(y)
#Função que recebe um conjunto de números x e y e o grau do polinômio
que desejado e cria a matriz de vandermonde dele
function vandermonde(x,y,grau)
#Recebendo o tamanho do vetor x
    n,=size(y)
#Criando uma matriz zerada do tamanho desejado
    V=zeros(n,grau+1)
#Looping que vai até o número de linhas da matriz
    for i=1:n
#Looping que vai até o número de colunas desejadas na matriz (grau da
matriz + 1)
        for j=1:(grau+1)
#Criando ponto a ponto da matriz (posição atual x elevado a  j-1)
            V[i,j]=x[i]^(j-1)
        end
    end
    #Retorna a matriz desejada
    return V
end

#Função que realiza a regressão dado um conjunto de pontos e o grau de
polinômio
function regressão(x,y,grau)
#Chama a função de vondermonte para criar a matriz
    V=vandermonde(x,y,grau)
#Resolvendo o sistema: quando o sistema a ser resolvido não é possível
de ser resolvido, o contra barra "\" do Julia já disponibiliza a
solução pelo método dos mínimos quadrados*
    c=V\y
```

```
    return c #Retorna os coeficientes do polinômio
end

#Realizando a regressão para encontrar os coeficientes c1 e c2 do
polinômio
c_barra=regressão(x_barra,y_barra,1)

#Criando o polinômio y_barra = c1_barra + x_barra*c2_barra passando
os coeficientes encontrados
reta_barra(x)= c_barra[1]+c_barra[2]*x

#Voltando para a função exponencial e encontrando os valores dos
coeficientes dessa função
c1=exp(c_barra[1])
c2=c_barra[2]

#Retornando o modelo exponencial
exponencial(x)=c1*exp(c2*x)

#Plotando os pontos dados
scatter(x, y, c=:lightblue, ms=3, leg=false)

#Plotando o gráfico da função exponencial
plot!(exponencial)
```

Encontramos:



Agora vamos descobrir a data que a pessoa vai pesar 110kg

```
exponencial(177)
```

```
110.05581225548903

julia>
```

Podemos observar que depois de 177 dias após a primeira medição a pessoa vai pesar 110kg. Traduzindo isso, a data estimada será 21/04/2022, ou seja, depois de 5 meses, três semanas e 5 dias.

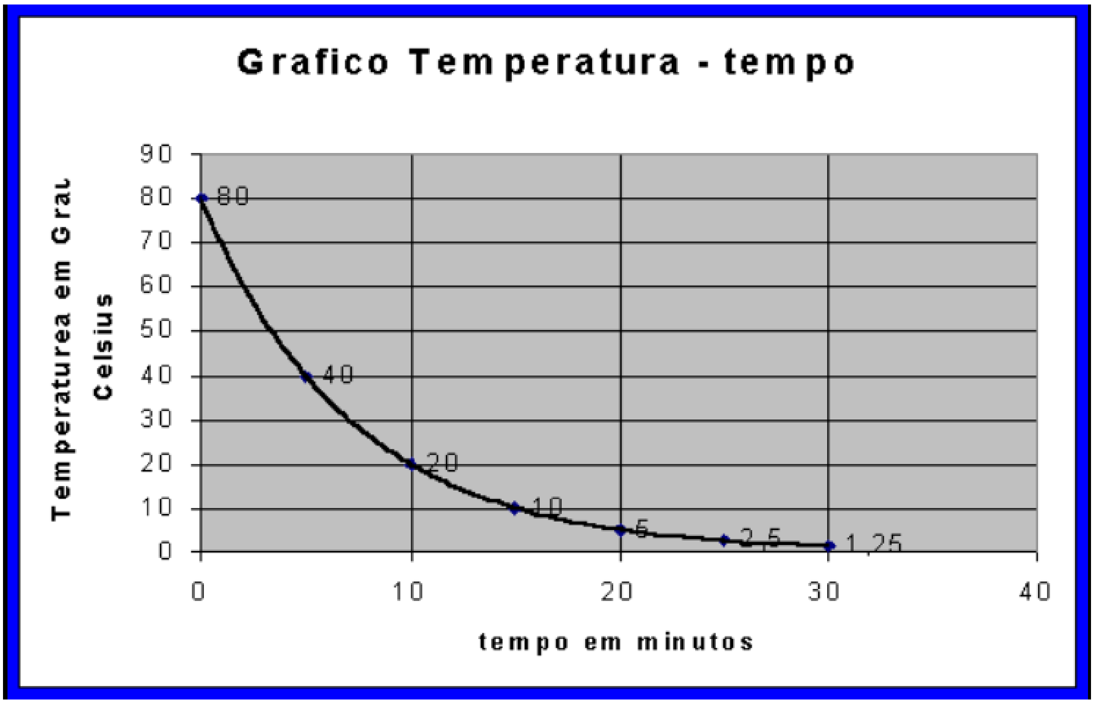
Exercício 1.5-)
Nesse exercício, vamos descobrir o horário na qual a pessoa foi morta utilizando o método da regressão com coeficientes não lineares. A questão nos informa a relação entre temperatura do corpo e hora:

Hora	**15h**	**16:30h**	**17:30h**
Temperatura	**34ºc**	**30ºc**	**25ºc**

```
using Plots
```

```
using Random
using LinearAlgebra

#Vetor de hora
x=[15;16.5;17.5]
#Vetor de temperatura
y=[34;30;25]
```

Como estamos tratando do resfriamento de um corpo temos que a função é exponencial:



Dessa forma, vamos ter que realizar a troca de variáveis a fim de conseguirmos realizar a regressão. Assim, vamos considerar a função:

$$f(x) = c1*e^{c2x}$$

Transformando ela em linear temos:

$$ln(y) = ln(c1*e^{c2x})$$
$$ln(y) = ln(c1) + ln(e^{c2x})$$
$$ln(y) = ln(c1) + c2x$$

Realizando a troca de variáveis:

y_barra = ln(y)
x_barra = x
c1_barra = ln(c1)
c2_barra = c2

Encontramos:

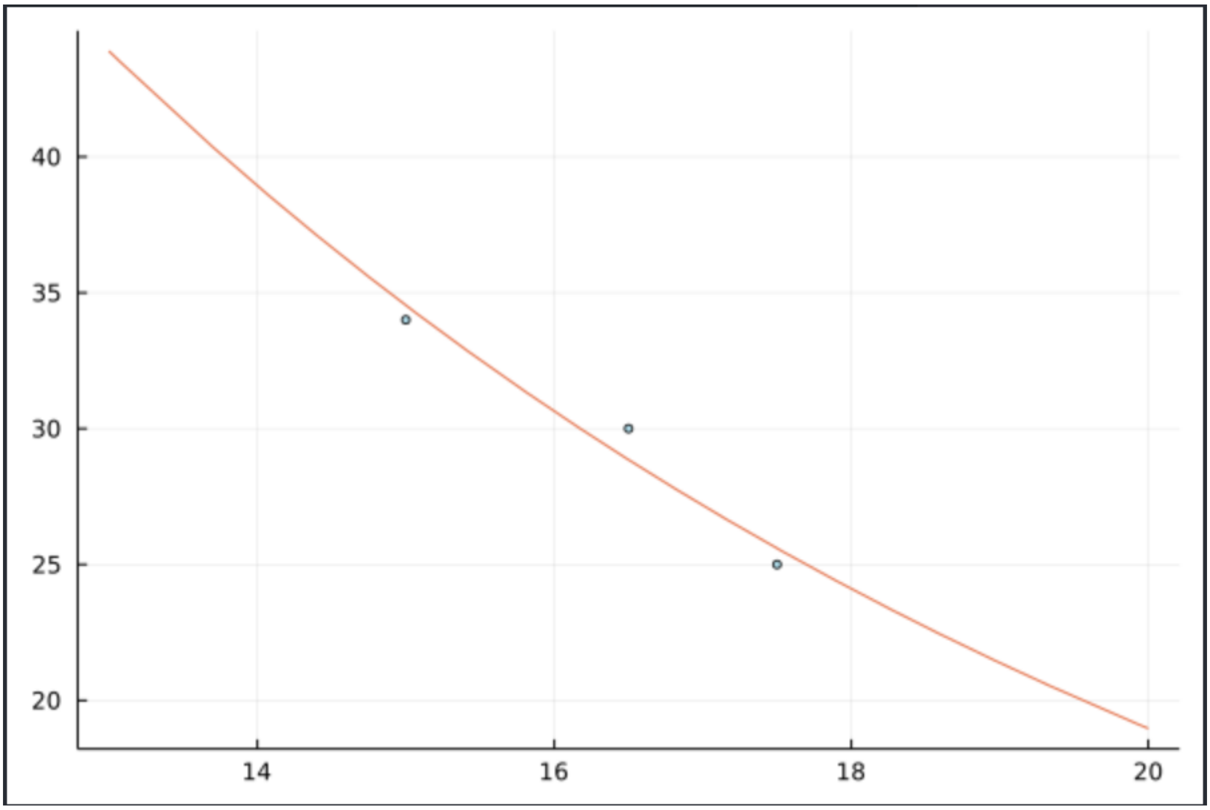
y_barra = c1_barra + x_barra*c2_barra

```
#Troca de variáveis "indo para mundo linear"
```

```
x_barra=x
y_barra=log.(y)

#Função que recebe um conjunto de números x e y e o grau do polinômio
que desejado e cria a matriz de vandermonde dele
function vandermonde(x,y,grau)
#Recebendo o tamanho do vetor x
    n,=size(y)
#Criando uma matriz zerada do tamanho desejado
    V=zeros(n,grau+1)
#Looping que vai até o número de linhas da matriz
    for i=1:n
#Looping que vai até o número de colunas desejadas na matriz (grau da
matriz + 1)
        for j=1:(grau+1)
#Criando ponto a ponto da matriz (posição atual x elevado a  j-1)
            V[i,j]=x[i]^(j-1)
        end
    end
    #Retorna a matriz desejada
    return V
end

#Função que realiza a regressão dado um conjunto de pontos e o grau de
polinômio
function regressão(x,y,grau)
#Chama a função de vondermonte para criar a matriz
    V=vandermonde(x,y,grau)
#Resolvendo o sistema: quando o sistema a ser resolvido não é possível
de ser resolvido, o contra barra "\" do Julia já disponibiliza a
solução pelo método dos mínimos quadrados*
    c=V\y
    return c #Retorna os coeficientes do polinômio
end

#Realizando a regressão para encontrar os coeficientes c1 e c2 do
polinômio
c_barra=regressão(x_barra,y_barra,1)

#Criando o polinômio y_barra = c1_barra + x_barra*c2_barra passando
os coeficientes encontrados
reta_barra(x)= c_barra[1]+c_barra[2]*x
```

```
#Voltando para a função exponencial e encontrando os valores dos
coeficientes dessa função
c1=exp(c_barra[1])
c2=c_barra[2]
#Retornando o modelo exponencial
exponencial(x)=c1*exp(c2*x)

#Plotando os pontos x e y passados
scatter(x, y, c=:lightblue, ms=3, leg=false)

#Plotando a função exponencial no intervalo de 13 até 20
plot!(exponencial,13,20)
```

Obtemos os seguintes plots:



Para encontrar a hora da morte, vamos passar diferentes valores até chegarmos em 37:

```
exponencial(14.4) #Em decimal
```

```
37.110349520261344

julia>
```

Logo, obtemos que a hora da morte é de aproximadamente **14:24h**