Aluno: Luiz Rodrigo Lacé Rodrigues
DRE: 118049873

Exercício 1:(Revisão de vários conceito que vimos até agora) Suponha que nós sabemos calcular e^x para qualquer x, sabemos todas as propriedades de ln (tal como ln(e^c) = c) e todas as suas derivadas. Aproxime ln(3), se possível, usando 3 dos 5 itens abaixo (faça os itens que você não fez na Tarefa 3.)

Como essa questão é repetida da tarefa 3, onde nós podíamos escolher apenas um exercício e fazê-lo. Entretanto eu fiz todos, mas não expliquei todos, apenas deixei o print dos códigos do Julía, dessa forma farei a o 2,3,4 nessa tarefa 4.

Exercício 1.3.2: o Método de Newton com 20 passos.

Como queremos aproximar o ln(3), sabendo calcular e^x, então temos que ln(e^x) = x, basta então calcularmos e^x = 3. O que vamos tentar então é achar um x que zere a função f(x) = e^x - 3, esse x será nossa aproximação.

O método de newton é dado por:

$$x2 = x1 - \frac{f(x1)}{f'(x1)}$$

Para o nosso caso, temos então: $x2 = x1 - \frac{e^{x1} - 3}{e^x}$

Onde x2 é a aproximação do valor da função no ponto x1.

Para essa questão teremos o seguinte método ln3(numeroDeIteracoes, chute)

```
In [130]: #Exercicio 1.3.2
function ln3(numeroDeIteracoes, chute)#numero de iterações e chute inicial
   for i=1:numeroDeIteracoes
        chute = chute - ((exp(chute)-3)/exp(chute))
    end
    return chute
end
```

Neste método o que estamos fazendo é pegar o chute inicial e transformá-lo na própria aproximação utilizando o método de newton e isso se repete quantas vezes forem passadas no parâmetro do método. Ao final o valor da aproximação é retornado.

Como o enunciado pediu 20 passos teremos, com chute inicial igual a 1.1, a seguinte aproximação:

```
In [132]: #Exercicio 1.3.2
          ln3(20,1.1)
Out[132]: 1.0986122886681098
```

ln(3) ≅ 1.0986122886681098

Exercício 1.3.3: o Polinômio de Taylor com erro máximo de 10^−3

Para fazer a aproximação de ln(3) utilizando o Polinômio de Taylor vamos primeiro utilizar o cálculo do erro para que saibamos o grau do polinômio.

O erro do polinômio de taylor é dado por:

$M \frac{(x-a)^{n+1}}{(n+1)!}$, onde m é o módulo máximo da derivada do termo n+1 no intervalo dado.

Temos então f(x) = ln(x)

Para a = e^1

Vamos ter um erro, em taylor de ordem 1 de

(1/x^2) * (((x-a)^2)/ !)=
(1/x^2) * (((3-e)^2)/ 2!) = 0.005370451050430154

Como precisamos de um erro menor, vamos aumentar o polinômio de Taylor para o grau 2

O erro fica então
(2/x^3) * (((x-a)^2)/ 3!) =

```
In [10]: #Exercicio 1.1.3
         #Erro de taylor em ordem 2, menor que 10^-3

         #(M(x-a)^3)/6
         (2/(exp(1))^3) * ((3-exp(1))^3)/6

         #erro
Out[10]: 0.00037105636225489186
```

(2/e^2) * (((3-e)^2)/ 3!) = 0.00037105636225489186

Encontramos então um erro menor que o máximo com Taylor de Ordem 2, dessa forma, o polinômio é dado por:

f(x) - (f(a) + f'(a)(x-a) + (f''(a) *(x-a)^2)/2   <= M((x-a)^3)/6

Lembrando que nosso a = e

Vamos ter então que

ln(3) - (ln(e) + (1/e)(3-e) + ((-1/x^2)*(x-a)^2)/2   <= M((x-a)^3)/6   =

Em julia para calcularmos o termo que nos retorna a aproximação te ln(3), vamos ter:

1 + ((3-exp(1))/exp(1)) - (((3-exp(1))^2)/2)/((exp(1))^2))

Calculando no julia:

```
In [11]: #Exercicio 1.1.3

         #taylor de ordem 2
         #f(x) - (f(a) + f'(a)(x-a) + (f''(a) *(x-a)^2)/2      <= M((x-a)^3)/6
         # a = e
         1 + ((3-exp(1))/exp(1)) - (((3-exp(1))^2)/2)/((exp(1))^2)

Out[11]: 1.0982678724638968
```

Logo ln(3) ≅ 1.0982678724638968

Exercício 1.3.4: a interpolação polinomial de grau 1 e estime o erro máximo.

Com o método da interpolação estamos interessados em criar um polinômio que aproxime ao comportamento da nossa função.

Como o enunciado pede o método da interpolação com grau 1, logo teremos 2 pontos no nosso polinômio, que terá a seguinte cara

P(x) = a0 + a1* x

Sabemos calcular e^x para qualquer x, e também as suas propriedades como ln(e^x) = x
Dessa forma vamos utilizar os pontos e^(1) e e^(1.1). Pois, como nossa função f(x) = ln(x) então temos que:

P(e) = a0 + a1 * e = 1
P(e^1.1) = a0 + a1 * e^1.1 = 1.1

O que precisamos é resolver esse sistema para encontrar os coeficientes a0 e a1, com o polinômio formado, basta substituir 3 no lugar de x e vamos encontrar a aproximação para ln(3)

No julia, criei o método interpolação1(x,y), que tem a seguinte cara:

```
In [3]: #1.1.4
        function interpolação1(x,y)
            #criar a matriz V
            V=[x.^0 x.^1]
            c=V\y
            return c #vetor de coeficientes
        end
```

Nesse método, recebemos um array com os pontos de x e os resultados em y.

Dentro do método vamos criar uma matriz com os respectivos quadrados de x, vamos fazer a potência ponto a ponto de cada elemento dentro do array recebido também chamado de x.

Como estamos tentando resolver um sistema Vc = y, logo basta fazer a divisão inversa para achar os coeficientes “c” , logo c = V \ y

Logo depois retornamos os coeficientes para montarmos o polinômio.

No julia, quando passarmos os pontos de x e y que escolhemos, temos o seguinte resultado:

```
In [141]: #exercicio 1.3.4
          x=[exp(1); exp(1.1)]
          y=[1, 1.1]
          interpolação1(x,y)

Out[141]: 2-element Array{Float64,1}:
           0.049166805522496304
           0.3497919842316417
```

logo nossos coeficientes são

a0 = 0.049166805522496304
a1 = 0.3497919842316417

Logo nosso polinômio será

P(x) = 0.049166805522496304 + 0.3497919842316417 * x

Basta então calcularmos P(3) para achar a aproximação de ln(3).

Calculando isso no julia temos o seguinte resultado:

```
In [11]: 0.04916680552249564 + 0.34979198423164165  * 3

Out[11]: 1.0985427582174205
```

Temo então que Logo ln(3) ≅ 1.0985427582174205

O erro da interpolação é dado por:

(M*(x-xo)*(x-x1)) /(n+1)!

Como o módulo máximo da derivada de n+1 é:

f'n+1(x) = -1/x^2

temos que nosso erro ficará com a seguinte cara

|-1/(e^1)^2 * ((3-e^1) * (3-e^1.1)) / 2! |

Calculando isso no julia, temos:

```
In [23]: #exercicio 1.1.4
#ERRO

# o erro é dado por(M(x-xo)(x-x1))/(n+1)!
# temos então

# f'n+1(x) = -1/x^2

# |-1/(e^1)^2| * ((3-e^1) (3-e^1.1)) / 2!

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

Out[23]: 7.941776548122973e-5
```

Logo nosso erro para a aproximação de ln(3) é 7.941776548122973e-5

Exercício 1.2 (Escolhendo o polinômio correto) Usando o material da Aula 15:

1. Gere aleatoriamente 30 pontos de um polinômio de grau 5.

Gerando 30 pontos aleatórios de um polinômio de grau 5 entre 0 e 1, com ruído.

```
#execicio 1.2
#Gere aleatoriamente 30 pontos de um polinômio de grau 5.

Random.seed!(0)
n = 30
x = range(0, 1, length=n)

g0(x) = 1.0
g1(x) = x
g2(x) = x^2
g3(x) = x^3
g4(x) = x^4
g5(x) = x^5

y = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) + randn(n) * 0.1 #pontos de um polinomio de grau 5 + ruido
```

2. Faça regressão polinomial com polinômios de grau 0 até 29.

Nessa questão faremos uso dos métodos vondemonde e regressão (ambos melhor explicados na questão 1.4, pois eu fiz primeiro)

```
In [2]: function vandermonde(x,y,grau)
            n,=size(y) #pegando o tamanho do vetor y
            V=zeros(n, grau+1) #criando uma matriz de zeros
            for i=1:n #linhas
                for j=1:(grau+1) #colunas
                    V[i,j]=x[i]^(j-1) #calculando as resprectivas potencias do polinomio
                end
            end
            return V # retona a matriz de vandermonde
        end

Out[2]: vandermonde (generic function with 1 method)
```

```
In [3]: function regressão(x,y,grau)
            V=vandermonde(x,y,grau) #montando a matriz de vondermonde
            c=V\y # resolve o  sistema linear. já faz minimos quadrados
            return c #retorna os coeficientes
        end
```

Também utilizaremos os métodos gerapoli gerapoli(xs,ys,ngrau) e gerapoli_plot(x,y,grau):

```
In [4]: function gerapoli(xs,ys,ngrau)
            coeficientes = regressão(xs,ys,ngrau) #gerando os coeficientes
            f(x) = sum([coeficientes[i+1]*x^i for i in 0:ngrau]) #gerando um polinomio dependendo do grau
            return f #retornando o polinomio
        end

Out[4]: gerapoli (generic function with 1 method)
```

Esse método recebe os vetores com os pontos x e y e também o grau do polinômio.
Dentro dele, fazemos a regressão para gerar os coeficientes do polinômio aproximado e depois

vamos gerar o polinômio de forma que iteramos de 0 até o grau passado no parâmetro para “concatenarmos” cada termo do polinômio. Depois retornamos o polinômio gerado.

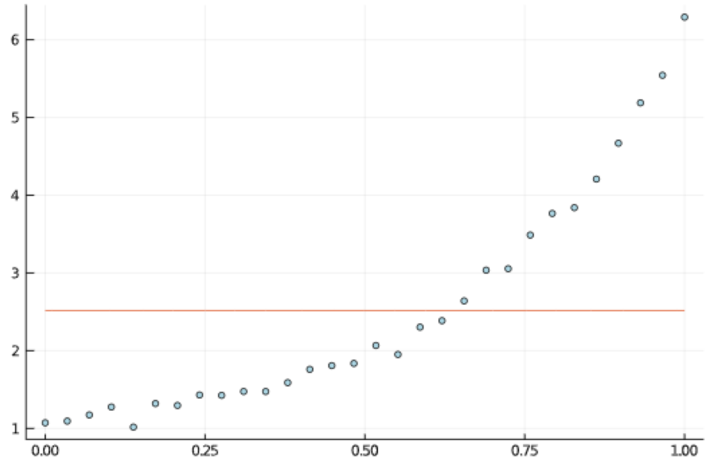
```
In [5]: function gerapoli_plot(x,y,grau)
            scatter(x, y, c=:lightblue, ms=3, leg=false) #plota os pontos
            plot!(gerapoli(x,y,grau),0,1)                #plota o polinimio
        end

Out[5]: gerapoli_plot (generic function with 1 method)
```

O método gerapoli_plot simplesmente plota os pontos x e y e depois plota o polinômio gerado a partir dos parametros passados.

Alguns exemplos:

```
In [84]: #exercicio 1.2.2
         y = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) + randn(n) * 0.1 #pontos de um polinomio de grau 5 + ruido
         y1 = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) #pontos de um polinomio de grau 5
         gerapoli_plot(x,y,0)
         #gerapoli_plot(x,y1,0)
```

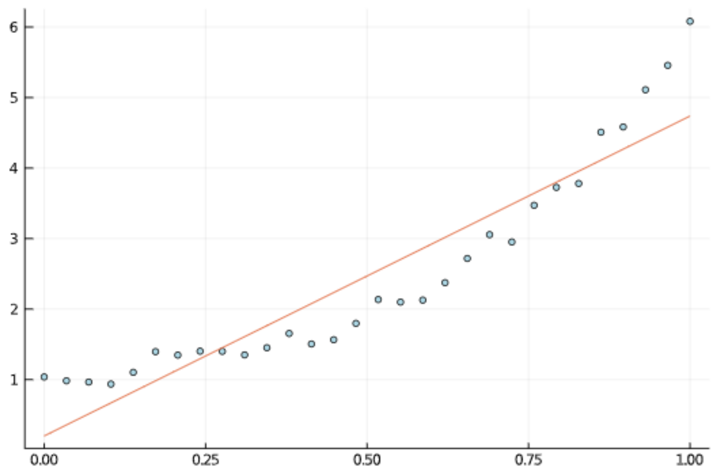
Out[84]:



Pontos de y com ruído e grau 0

```
In [85]: #exercicio 1.2.2
         y = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) + randn(n) * 0.1 #pontos de um polinomio de grau 5 + ruido
         y1 = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) #pontos de um polinomio de grau 5
         gerapoli_plot(x,y,1)
         #gerapoli_plot(x,y1,0)
```

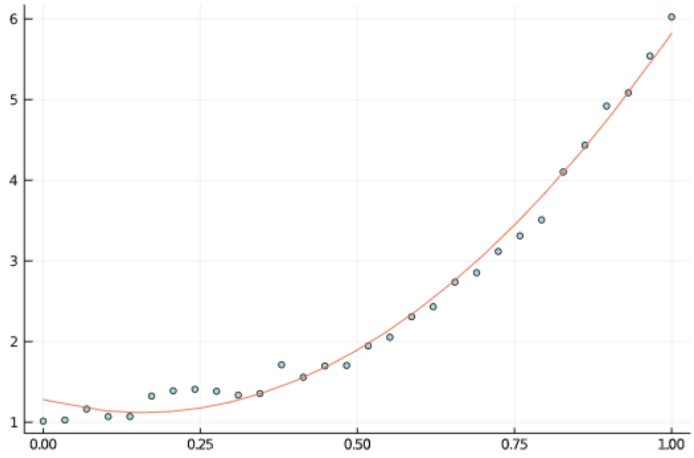
Out[85]:



Com ruído e grau 1

```
In [86]: #exercicio 1.2.2
         y = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) + randn(n) * 0.1 #pontos de um polinomio de grau 5 + ruido
         y1 = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) #pontos de um polinomio de grau 5
         gerapoli_plot(x,y,2)
         #gerapoli_plot(x,y1,0)
```

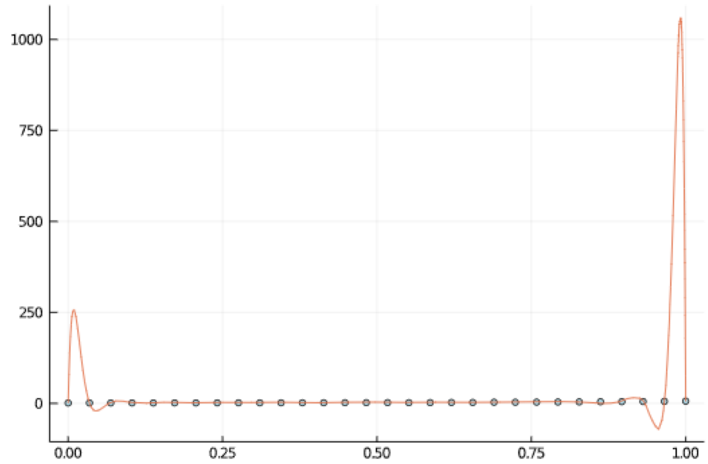
Out[86]:



Com ruído e grau 2

```
In [91]: #exercicio 1.2.2
         y = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) + randn(n) * 0.1 #pontos de um polinomio de grau 5 + ruido
         y1 = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) #pontos de um polinomio de grau 5
         gerapoli_plot(x,y,29)
         #gerapoli_plot(x,y1,0)
```

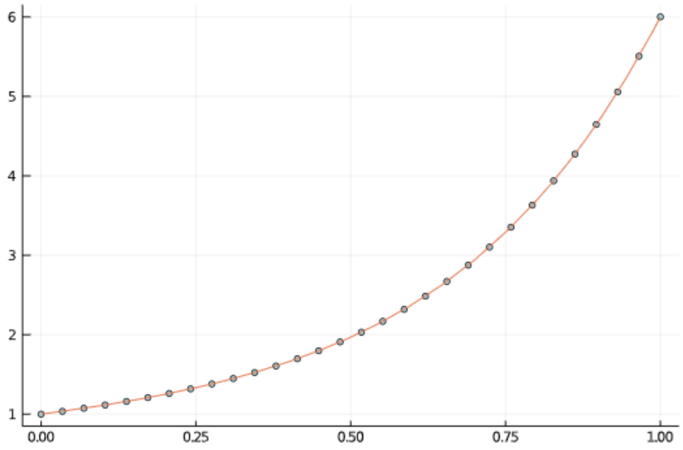
Out[91]:



Com ruído e grau 29 temos algo esquisito (devido ao ruído)

In [93]:
```
#exercicio 1.2.2
y = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) + randn(n) * 0.1 #pontos de um polinomio de grau 5 + ruido
y1 = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) #pontos de um polinomio de grau 5
#gerapoli_plot(x,y,29)
gerapoli_plot(x,y1,29)
```

Out[93]:



Com grau 29, mas dessa vez sem o ruído conseguimos fitar a função perfeitamente.

3. É possível fazer a regressão com um polinômio de grau maior que 29?
O que acontece no Julia?

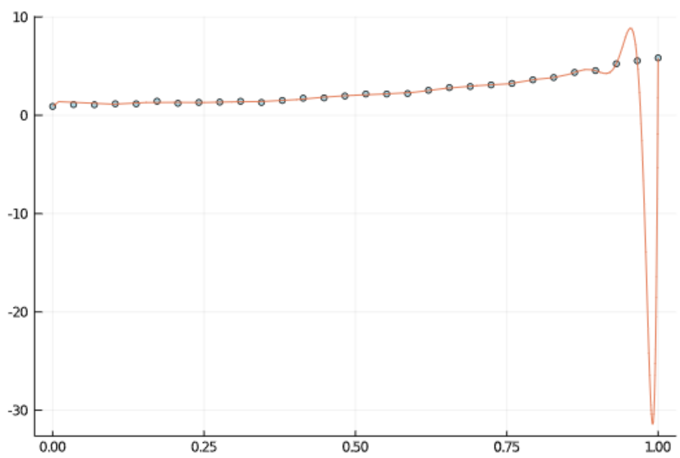
Sim, é "possível" fazer a regressão com um polinômio de grau maior do que 29 no julia. Isso é não retorna nenhum erro. Isso não deveria acontecer pois nós temos mais coeficientes do que pontos.

Isso acontece pois mesmo após o grau 29, o julia fará aproximações para minimizar o quadrado de todos os coeficientes existentes o menor possível e assim retornar uma solução o mais próxima possível do comportamento da função. Isso é uma técnica chamada SVD

Podemos visualizar os plots:

In [98]:
```
#exercicio 1.2.3
y = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) + randn(n) * 0.1 #pontos de um polinomio de grau 5 + ruido
y1 = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) #pontos de um polinomio de grau 5
gerapoli_plot(x,y,30)
#gerapoli_plot(x,y1,30)
```

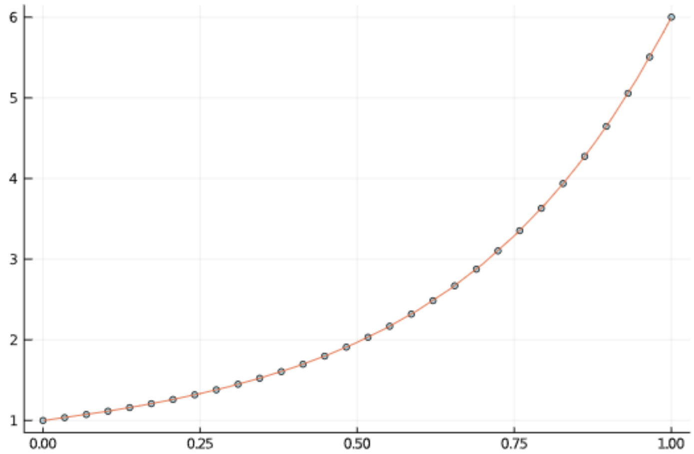
Out[98]:



Com um grau 30, mas com ruído nos pontos, podemos ver que o polinômio diverge do comportamento da função.

```
In [99]: #exercicio 1.2.3
         y = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) + randn(n) * 0.1 #pontos de um polinomio de grau 5 + ruido
         y1 = g0.(x) + g1.(x) + g2.(x) +  g3.(x) + g4.(x) +  g5.(x) #pontos de um polinomio de grau 5
         #gerapoli_plot(x,y,30)
         gerapoli_plot(x,y1,30)
```

Out[99]:



Mas sem ruído ele continua fitando a função.

4.Faça o plot do Erro total (eixo y) por grau (eixo x). O que se pode dizer desse gráfico conforme o grau aumenta? Era o que você esperava? Por quê?

Nessa questão, teremos o método erro_total, que está bem comentado abaixo.

```
In [64]: #metodo para calcular o erro
         function erro_total(x,y,modelo)
             n,=size(y) #quantidade de pontos em y
             S=0        # erro iniciado em 0
             for i=1:n #iterando pela quantidade de pontos
                 S=S+(y[i]-modelo(x[i]))^2 # Fazendo o somatório do quadrado da diferenta entre o ponto e o modelo passado
             end
             return sqrt(S) #tirando a raiz quadrada para retornar o erro sem estar ao quadrado
         end

Out[64]: erro_total (generic function with 1 method)
```

Vamos fazer uso novamente do método “gerapoli” para criar um vetor (polinomios) que recebe todos com grau de 0 a 29.

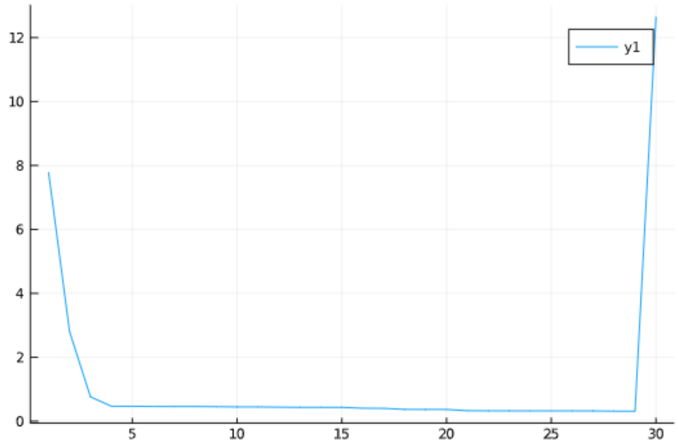
Depois criaremos um vetor com os erros, nele vamos iterar de 1 a 30 para pegar todos os polinômios do vetor polinomios, e gerar o erro de cada um deles.

Depois vamos plotar esses pontos.

No Júlia, temos o seguinte resultado:

In [56]:
```
polinomios = [gerapoli(x,y,i) for i in 0:29]          # gerando um vetor com polinomios com grau de 0 a 29
erros  =[erro_total(x,y,polinomios[i]) for i in 1:30]  # gerando um vetor com os erros de todos os polinomios criados antes
plot(erros)  #plotando o erro total por grau do polinomio
```

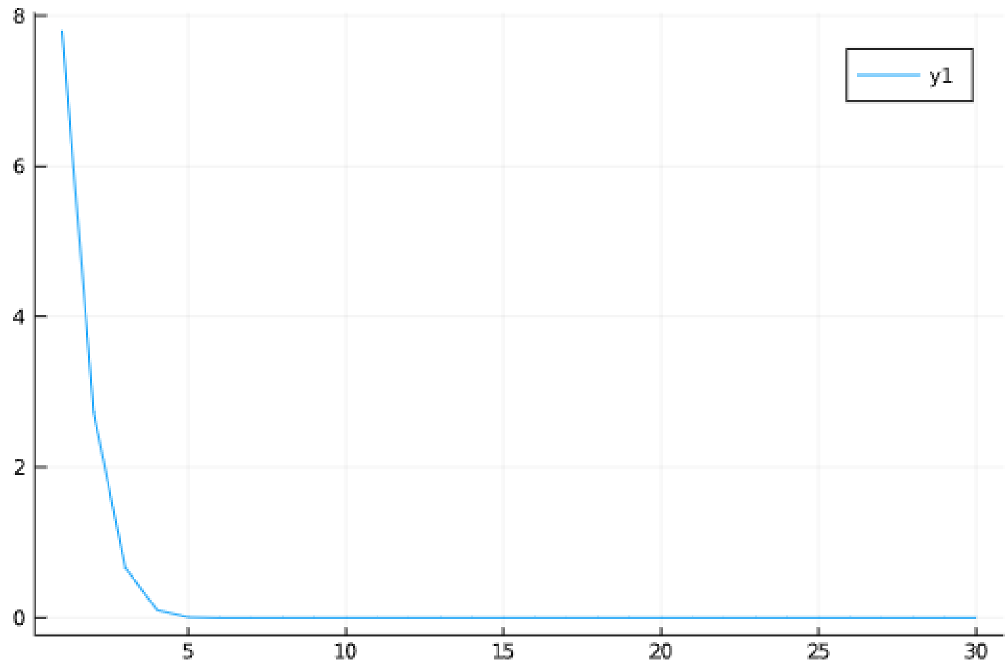
Out[56]:



Percebemos que a partir de um certo grau (cotovelo) vamos do underfit para o overfit da função. Entretanto, o que era esperado era que o erro chegasse em 0 quando o polinômio fosse de grau 29, mas o que acontece é que ele “explode”, devido ao ruído que passamos lá no início, quando geramos os pontos aleatórios de um polinômio de grau 5.

Podemos provar isso, pois quando removemos o ruído e plotamos esse erro.

Out[59]:



Agora sim estamos indo à 0.

Exercicio 1.3. O aluno Mateus Olaso fez uma pesquisa com 13 alunos da nossa turma de cálculo numérico e descobriu certas preferências quando perguntou para eles escolherem entre dois filmes:

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

Qual é o filme preferido dos 13 alunos usando mínimos quadrados e a técnica desenvolvida na aula 14?

Primeiro vamos ver de quanto cada filme ganha do outro, formando equações:
Os nomes serão abreviados de forma que segue a lista acima

1. TS - R = 11
2. DVPF - CVA = 3
3. OI - D = 7
4. BB - HP = 2
5. S - D = 9
6. HP - R = 7
7. TS - DVPF = 5
8. OI - HP = 5
9. CVA - D = 2
10. DVPF - D = 2
11. S - R = 11
12. OI - BB = 5
13. TS - BB = 3
14. OI - CVA = 7

Temos 14 equações com apenas 9 variáveis, para resolvermos esses sistema por mínimos quadrados, vamos adicionar mais uma variável, isso é, um filme com um valor aleatório.

Rocky = 5

Agora vamos tentar resolver esse sistema linear, como o julia já resolve o método por mínimos, quando necessário.

Teremos a seguinte multiplicação de matrizes

```
TS - R = 11          | 1  -  1  0  0  0  0  0  0  0  |     | TS   |       | 11 |
DVPF - CVA = 3       | 0  0  1  -1  0  0  0  0  0    |     | R    |       | 3  |
OI - D = 7           | 0  0  0  0  1  -1  0  0  0    |     | DVPF |       | 7  |
BB - HP = 2          | 0  0  0  0  0  1  -1  0       |     | CVA  |       | 2  |
S - D = 9            | 0  0  0  0  0  -1  0  0  1    |     | OI   |   =   | 9  |
HP - R = 7           | 0  -1  0  0  0  0  0  1  0    |  x  | D    |       | 7  |
TS - DVPF = 5        | 1  0  -1  0  0  0  0  0  0    |     | BB   |       | 5  |
OI - HP = 5          | 0  0  0  0  1  0  0  -1  0    |     | HP   |       | 5  |
CVA - D = 2          | 0  0  0  1  0  -1  0  0  0    |     | S    |       | 2  |
DVPF - D = 2         | 0  0  1  0  0  -1  0  0  0    |                     | 2  |
S - R = 11           | 0  -1  0  0  0  0  0  0  1    |                     | 11 |
OI - BB = 5          | 0  0  0  0  1  0  -1  0  0    |                     | 5  |
TS - BB = 3          | 1  0  0  0  0  0  -1  0  0    |                     | 3  |
OI - CVA = 7         | 0  0  0  -1  1  0  0  0  0    |                     | 7  |
R                    | 0  1  0  0  0  0  0  0  0     |                     | 5  |
```

Como temos um sistema do tipo Vc = y, basta resolvê-lo fazendo c = V\y. A divisão inversa com o símbolo \ no julia já nos retorna os coeficientes utilizando método dos mínimos quadrados, quando necessário.

Passando isso pro júlia, teremos:

In [8]:

```
#exercicio 1.3
V = [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  1 -1  0  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;
     0  1  0  0  0  0  0  0  0]

y= [ 11; 3; 7; 2; 9; 7; 5; 5; 2; 2; 11; 5; 3; 7; 5]

c = V\y
```

Out[8]:

```
9-element Array{Float64,1}:
 15.221680876979285
  4.999999999999995
 11.405602923264302
  9.65408038976857
 16.21559074299634
  9.34104750304506
 10.259439707673561
 11.607795371498165
 17.170523751522534
```

Agora que temos o valor dos coeficientes, que são os filmes, basta ver qual deles tem o maior valor e logo é o preferido da turma, que nesse caso é o último coeficiente referente ao filme **Shrek**.

Exercício 1.4

Nessa questão, temos a medição do peso de uma pessoa para determinados dias. Foi disponibilizado um arquivo com essas medições.

Queremos prever, a partir do histórico do emagrecimento, qual será o dia em que a pessoa terá 110 quilos.

Para isso, vamos utilizar o método da regressão.

Primeiramente, no eixo x teremos os dias das medições, e no eixo y, seu respectivo peso.

Antes de passarmos para o julia esses dados, o que foi feito foi transformar as datas das medições em dias corridos, para que possamos trabalhar mais facilmente com os dados

Dessa forma, a primeira medição, que foi no dia 26/10, será o dia 1, o dia 27/10 será o dia 2 e assim por diante. Vale ressaltar que para as datas "puladas", também respeitamos qual foi o dia em relação ao primeiro.

Vamos nos aproveitar dos métodos vandermonde, para criar as matrizes de vandermonde, que tem essa cara:

```
function vandermonde(x,y,grau)
    n,=size(y) #pegando o tamanho do vetor y
    V=zeros(n, grau+1) #criando uma matriz de zeros
    for i=1:n #linhas
        for j=1:(grau+1) #colunas
            V[i,j]=x[i]^(j-1) #calculando as resprectivas potencias do polinomio
        end
    end
    return V # retona a matriz de vandermonde
end
```

Esse método será mais utilizado quando chamarmos o método regressão:

```
In [15]: function regressão(x,y,grau)
             V=vandermonde(x,y,grau) #montando a matriz de vondermonde
             c=V\y # resolve o  sistema linear. já faz minimos quadrados
             return c #retorna os coeficientes
         end
```

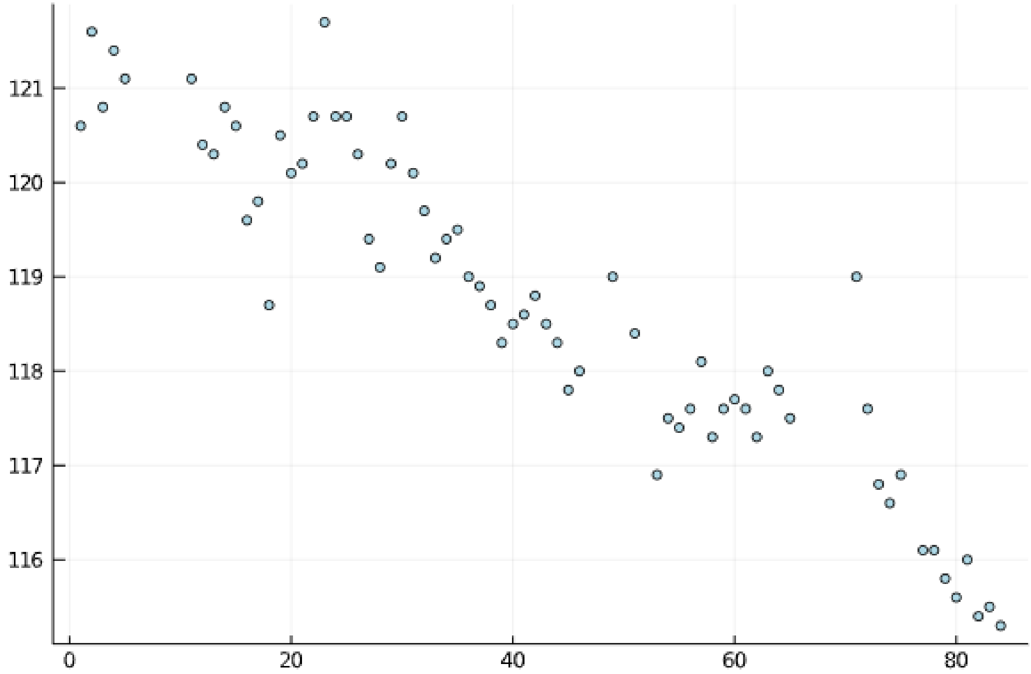
Iniciando o raciocínio, passamos os pontos x e y para um vetor

Teremos o seguinte código:

In [316]:
```
#Exercicio 1.4
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]  #datas transformadas em dias corridos

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]    #peso respectivo pra cada dia

scatter(x, y, c=:lightblue, ms=3, leg=false)
```

O plot disso terá a seguinte cara:

Out[316]:



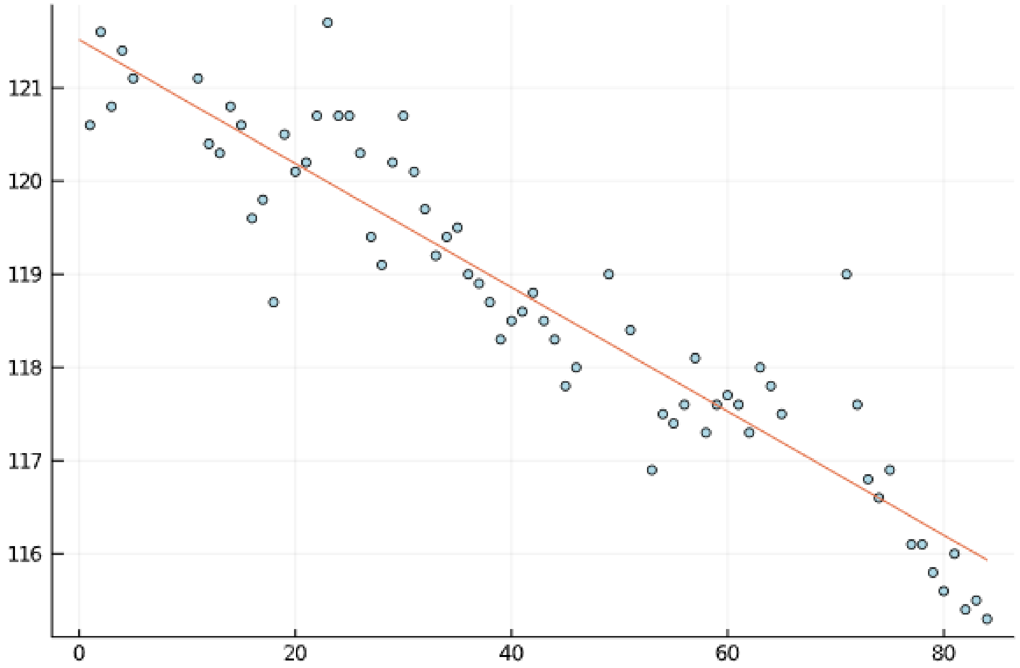
Com esses pontos parece que nossa função será uma reta.

Logo faremos uma regressão de grau 1 para que tenhamos os coeficientes (cpeso) do polinômio e assim gerar uma função aproximada.

Gerando polinômio “peso(x)” e plotando, teremos o seguinte resultado:

```
In [317]: cpeso=regressão(x,y,1) # fazendo a regressão para grau 1
          peso(x) = cpeso[1]  + cpeso[2]*x #criando um polinomio de grau 1
          scatter(x, y, c=:lightblue, ms=3, leg=false) #plotando os pontos
          plot!(peso,0,84) #plotando a função aproximada
```

Out[317]:



Visualmente, parece que a nossa função aproximada realmente satisfaz o comportamento dos pontos.

Sem considerar que o fato de emagrecer tem um comportamento exponencial, vamos procurar um dia do qual passamos como parâmetro que nos retorne o mais próximo possível de 110, que é o peso que buscamos.

Após algumas tentativas, encontrei que o melhor é o dia 173 a partir do primeiro dia da medição:

```
In [330]: peso(173)  #sem considerar um comportamento exponencial

Out[330]: 110.01778350592397
```

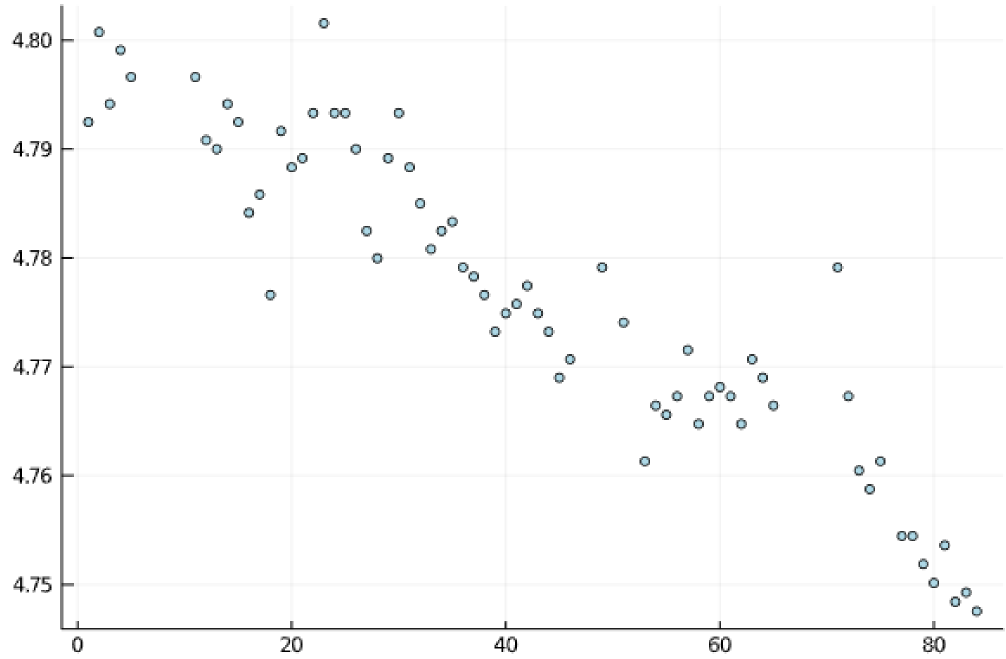
Ou seja, vai pesar 110 quilos em 17/04.

Agora, se considerarmos um comportamento exponencial, teremos que tirar o ln dos valores do eixo y.

Quanto plotamos isso, temos essa cara:

In [319]:
```
#calculos no mundo barra
y_barra=log.(y) #tirando log dos valores do eixo y
scatter(x,y_barra, c=:lightblue, ms=3, leg=false)
```

Out[319]:



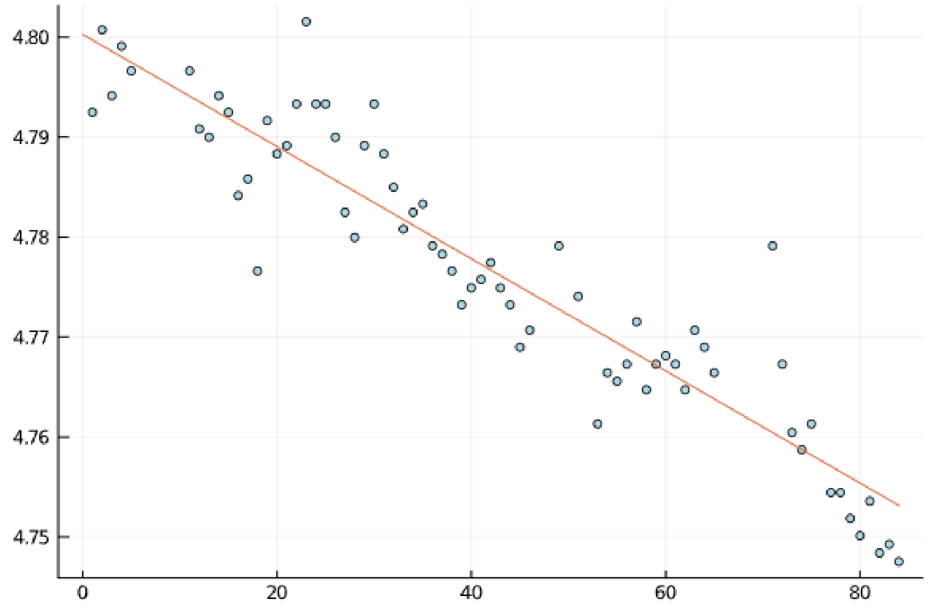
A imagem do gráfico parece bastante o que tínhamos anteriormente pois ele já parecia ter um comportamento linear, entretanto na “vida real” temos que o emagrecimento é uma exponencial, então é interessante nós avaliarmos esse caso.

Como ainda está parecendo uma reta, vamos utilizar o método da regressão novamente com grau = 1, mas dessa vez vamos passar os novos pontos de y(calculados o ln).

Calculada a regressão, e utilizando os coeficientes que dela foram retornados para criar um novo polinômio (peso_barra(x)) e em seguida plotando ele, temos o seguinte resultado:

In [320]:
```
cpeso_barra=regressão(x,y_barra,1) #fazendo a regressão com os pontos com log e grau 1
peso_barra(x) = cpeso_barra[1]  + cpeso_barra[2]*x #gerando um polinomio diferente
scatter(x,y_barra, c=:lightblue, ms=3, leg=false)
plot!(peso_barra,0,84) #plotando
```

Out[320]:



Agora que temos os coeficientes da função exponencial, podemos encontrar um polinômio agora no “mundo real”

Uma função exponencial tem a seguinte cara:

y = cpeso_barra1 * e^(cpeso_barra2*x)

Como não podemos descobrir os coeficientes dessa forma, nós fizemos todos o processo acima. E para voltamos para onde estávamos tirando o ln dos dois lados:

ln(y) = ln(cpeso_barra1) + ln (e^(cpeso_barra2*x))
ln(y) = ln(cpeso_barra1) + cpeso_barra2 * x

logo os coeficientes do nosso novo polinômio, será

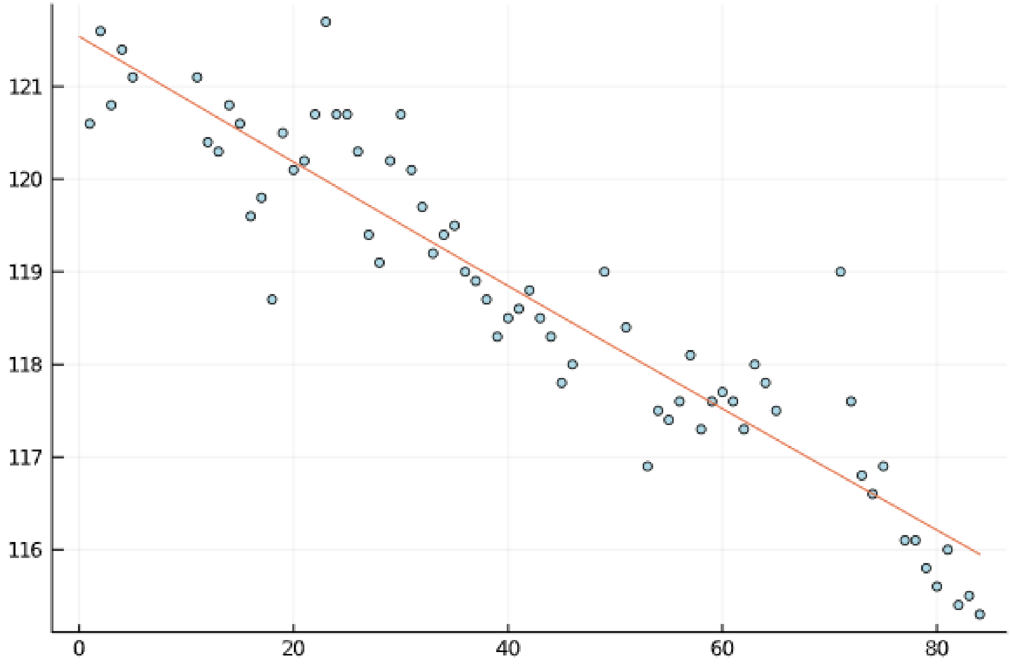
c1 = exp(ln(cpeso_barra1))
c2 = cpeso_barra2

Criando agora um novo polinômio peso_exponencial(x) = y = c1* e^(c2*x)

Calculando isso tudo e plotando, temos o seguinte resultado:

```
In [321]: #voltando para o mundo original
          c1=exp(cpeso_barra[1]) #coeficiente c1 no mundo original
          c2=cpeso_barra[2]      #coeficiente c2 no mundo original
          peso_exponencial(x)=c1*exp(c2*x) #modelo exponencial
          scatter(x, y,c=:lightblue, ms=3, leg=false)
          plot!(peso_exponencial,0,84)
```

Out[321]:



Agora vamos buscar o dia que vai pesar 110 quilos com o novo polinômio, que nesse caso será 177 dias após a primeira medição:

```
In [322]: peso_exponencial(177)  #considerando um comportamento exponencial
Out[322]: 110.05581225548862
```

Ou seja, a data que vai pesar 110 quilos será 21/04.

Exercício 1.5

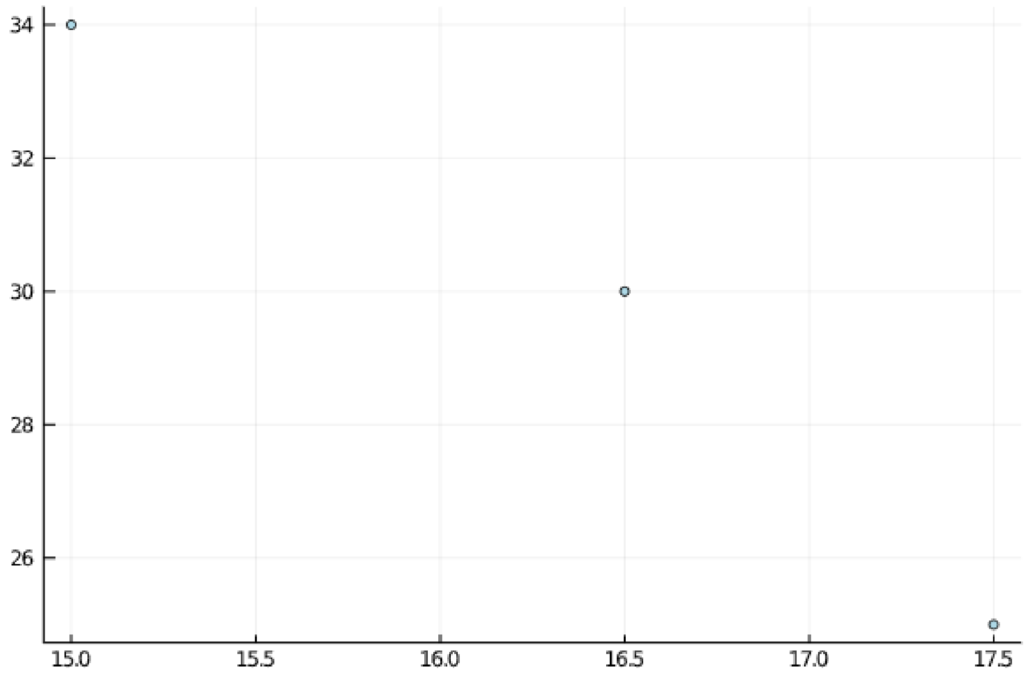
Nessa questão, temos as medições de temperatura de um corpo em relação a determinado horário e buscamos achar quando a pessoa foi morta. Dito que o corpo possui uma temperatura normal de 37ºc.

Como o comportamento do resfriamento é uma exponencial, vamos ter que fazer uma regressão de forma que a tirar o log dos pontos no eixo y primeiro para encontrar os coeficientes desta função exponencial.

Passando os respectivos horários e temperaturas para vetores e plotando esses pontos, teremos o seguinte:

In [339]:
```
#exercicio 1.5
x=[15,16.5,17.5]  #horarios das medições
y=[34,30,25]      #temperatura em dado horario

scatter(x, y,c=:lightblue, ms=3, leg=false)
```

Out[339]:



Vamos calcular o ln dos pontos em y para que possamos linearizar essa função e assim conseguirmos encontrar os coeficientes:

In [340]:
```
y_barra=log.(y) #tirando o log para linearizar a função
```

Out[340]:
```
3-element Array{Float64,1}:
 3.5263605246161616
 3.4011973816621555
 3.2188758248682006
```

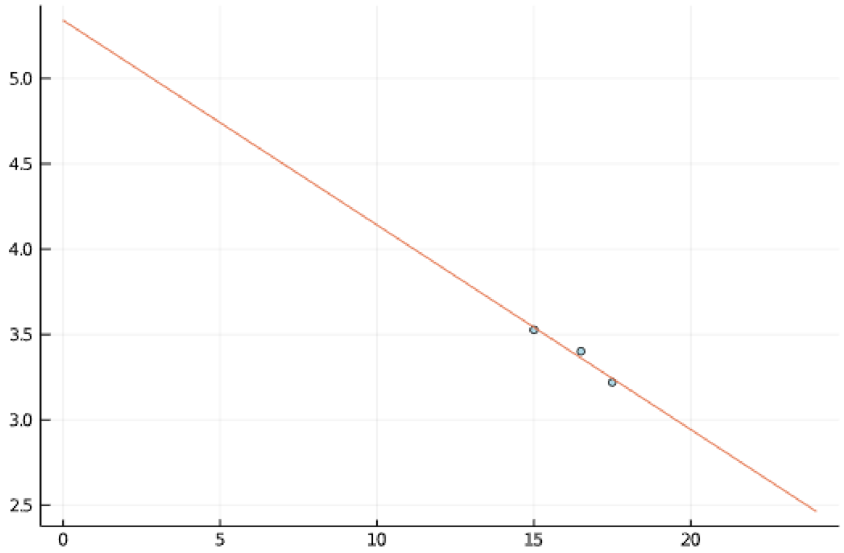
Nos utilizando da função regressão (já explicada em questões anteriores) para encontrar os coeficientes da função, passaremos o grau 1, pois como linearizar a teremos o seguinte:

```
In [341]: c_barra=regressão(x, y_barra, 1) #calculando os coeficientes do polinomio por regressão

Out[341]: 2-element Array{Float64,1}:
            5.340043630001425
           -0.11987137058893378
```

Depois basta pegar esses coeficientes e montar um polinômio agora para esse “mundo exponencial”

Montando o polinômio peso_barra(x) e o plotando, teremos o seguinte:

```
In [342]: peso_barra(x) = c_barra[1] + c_barra[2]*x #montando o polinomio com os coeficientes encontrados

Out[342]: peso_barra (generic function with 1 method)

In [343]: scatter(x, y_barra ,c=:lightblue, ms=3, leg=false)
          plot!(peso_barra,0,24)

Out[343]:
```



A função exponencial tem a seguinte cara:

y = c_barra[1] * e^(c_barra[2]*x)

Como não podemos descobrir os coeficientes dessa forma, nós fizemos todo o processo acima. E para voltamos para onde estávamos tirando o ln dos dois lados:

ln(y) = ln(c_barra[1]) + ln (e^(c_barra[2]*x))
ln(y) = ln(c_barra[1]) + c_barra[2]* x

logo os coeficientes do nosso novo polinômio, será

c1 = exp(ln(c_barra[1]))

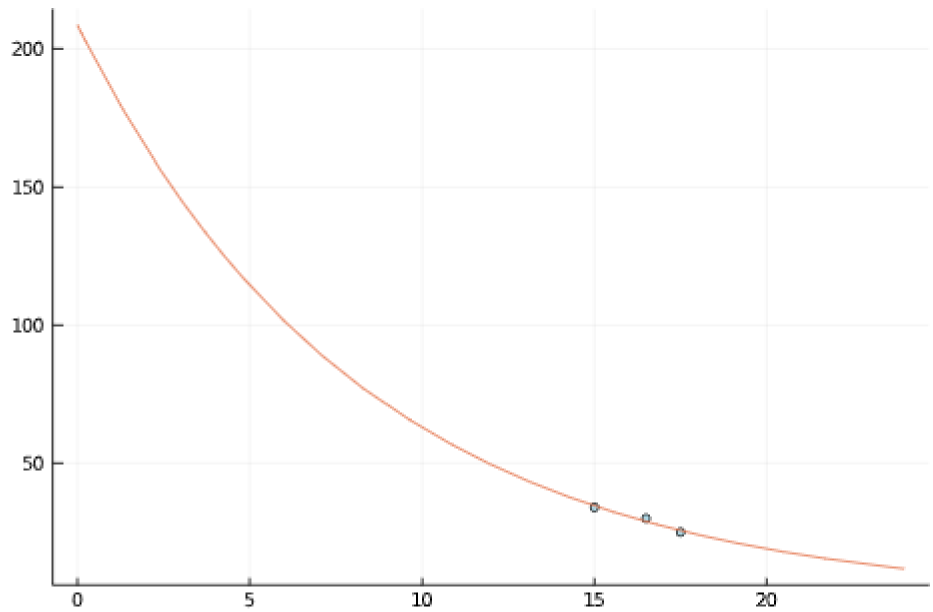
c2 = c_barra[2]

Como agora temos a nossos coeficientes no “mundo real”, po construir uma a função exponencial peso_exp(x).

Montando e plotando no julia, teremos:

```
In [344]: c1 = exp(c_barra[1])
          c2 = c_barra[2]

          peso_exp(x)= c1*exp(c2*x) #função exponencial no "mundo real"

Out[344]: peso_exp (generic function with 1 method)
```

```
In [345]: scatter(x, y ,c=:lightblue, ms=3, leg=false)
          plot!(peso_exp,0,24)

Out[345]:
```



Podemos agora passar diferentes horários como parâmetros para essa função de modo que quando retornar algo próximo de 37, será o horário que a pessoa morreu.

Após algumas tentativas achamos que 14.42 é o nosso horário.

```
In [346]: peso_exp(14.42)     #morreu em 14:25

Out[346]: 37.02148671468652
```

Isso em horas dá aproximadamente **14:25**