Tarefa 3 - Cálculo Numérico

Aluno: Luiz Rodrigo Lacé Rodrigues
DRE: 1118049873

Exercício 1.1:

Queremos encontrar a $\sqrt{10}$, entretanto como o enunciado pede que façamos essa aproximação utilizando o método da bisseção, com um erro máximo de 10^{-8}, então vamos buscar um erro no domínio. Para isso vamos buscar então um x, tal que x = $\sqrt{10}$, logo x^2 - 10 = 0. Ou seja, queremos achar um x que zere a função f(x) = x^2 - 10.

Para isso vamos implementar algumas funções na linguagem Julia.

```
#Exercicio 1.1
# dado a e b, tal que f(a) e f(b) tem sinais opostos, retorna x_final, tal
# que |x_final - x_c| <= tamanho_final_do_intervalor/2 com f(x_c) = 0
function bissecao(f, a, b, erro)

    if e_raiz(f,a)
        return a
    end

    if e_raiz(f,b)
        return b
    end

    if !(tem_sinais_opostos(f,a,b))
        return "Os sinais da função não são opostos nesse intervalo"
    end

    tamanho_final_do_intervalo = 2*erro

    iteracoes = floor(log2((b-a)/tamanho_final_do_intervalo))+1

    for i=1:iteracoes
            m=media(a,b)

            if e_raiz(f,m)
                return m
            end

            if tem_sinais_opostos(f,a,m)
                b=m
            else
                a=m
            end
    end

    print("iterações:",iteracoes)
    x_final = media(a,b)
    return x_final

end
```

Temos a nossa função principal, chamada bissecao, nela vamos passar a função desejada, o intervalo que vai de a até b, e o erro no domínio.

Primeiramente vamos verificar se o ponto a ou b já são as raízes da função passada, com um outro método chamado “e_raiz”, que está modularizado, da seguinte forma:

```
In [214]: function e_raiz(f,a)
              return f(a)==0
          end
```

Depois partimos para a verificação do sinal da função nos sinais passados (tem_sinais_opostos), caso a função tenha o mesmo sinal em ambos os pontos, então ficamos impossibilitados de prosseguir com o método da bisseção. Mais uma vez essa função está modularizada da seguinte forma:

```
In [403]: function tem_sinais_opostos(f,a,b)
              return f(a)/f(b) < 0
          end
```

Quando a função tiver sinais trocados, então o resultado da divisão será negativo e um true será retornado.

Na nossa função bissecao, vamos fazer um if verificando se o resultado retornado dos sinais opostos é FALSE, por isso a negação(!), caso não tenhamos sinais opostos, um aviso é retornado.

Seguimos então para o tamanho final do intervalo, que no método da bisseção é 2 vezes o tamanho do erro máximo.

O tamanho final do intervalo será necessário para que possamos ter um destino final em relação ao intervalo inicial que foi passado.

O número de iterações necessário é então dado por:

$\frac{b-a}{2^k}$ = tam (tamanho final do intervalo), sendo k o número de iterações

$$2^k = \frac{b-a}{tam}$$

$$k = log_2\left(\frac{b-a}{tam}\right)$$

Como chegamos em um número quebrado, então utilizamos o método floor, para pegarmos apenas a parte inteira e somamos 1, visto que queremos o número de iterações suficiente que contemple o tamanho final do intervalo. Ficamos então com:

k = floor($log_2\left(\frac{b-a}{tam}\right)$)+1

Que no código é referente a linha:

iteracoes = floor(log2((b-a)/tamanho_final_do_intervalo))+1

Chegamos finalmente no mais importante da função, as iterações que vão diminuir o intervalo inicial e nos aproximar da raiz da função passada.

Como antes nós já verificamos que o intervalo inicial passado tem sinais opostos na função, então agora já vamos pegar o ponto entre esses dois pontos, com o método média, que também está modularizado da seguinte forma:

```
function media(x,y)
    return (x+y)/2
end
```

Então verificamos novamente se a média dos pontos é a nossa raiz.

Se não for, então passamos a verificar se o esse ponto médio tem sinal oposto na função com o ponto a. Caso tenha, então nosso b agora será esse ponto médio. Caso contrário, quem nosso a que será o ponto médio agora.

Esse processo se repete, diminuindo o tamanho do intervalo que tenha relação com o erro passado como parâmetro.

No final printamos o número de iterações e também retornamos a média dos dois pontos finais, que será a aproximação da raiz da função.

Passando então a função g(x) = x^2 - 10, no intervalo [0,20], com um erro de 0.00000001, chegamos em:

```
In [65]: #exercicio 1.1
         g(x) = x^2-10
         erro = 0.00000001
         bissecao(g, 0, 20, erro)

         iterações:30.0

Out[65]: 3.162277666851878
```

Exercício 1.2:

Nesse exercício queremos implementar uma função que recebe um polinômio de grau 5 e sua derivada e retorna uma raiz do polinômio no intervalo [-100, 100], caso o polinômio tenha sinais trocados, caso contrário ele vai retornar um aviso. O enunciado também pede que o tamanho final do intervalo para o método da bisseção seja de 0.01. No final da aproximação pelo método da bisseção, vamos aproximar mais ainda com o Método de Newton.

Para isso vamos utilizar uma função chamada bissecao_newton, que é muito parecida com a função do exercício anterior, apenas com alguns ajustes. Nela, passamos a função, o início e o fim do intervalo e também a derivada da função.

Temos a parte da bisseção, seguida do método de newton (tudo na mesma função)

```
function bissecao_newton(f, a, b, derivada)

    #inicio da interpolação
    if e_raiz(f,a)
        return a
    end

    if e_raiz(f,b)
        return b
    end

    if !(tem_sinais_opostos(f,a,b))
        return "Os sinais da função não são opostos nesse intervalo"
    end

    tamanho_final_do_intervalo = 10^-2

    iteracoes = floor(log2((b-a)/tamanho_final_do_intervalo))+1

    for i=1:iteracoes
            m=media(a,b)

            if e_raiz(f,m)
                return m
            end

            if tem_sinais_opostos(f,a,m)
                b=m
            else
                a=m
            end
    end

    x_semi_final = media(a,b)
    #fim da interpolação
```

```
    print("x_semi_final: ",x_semi_final)

    #inicio do metodo de newton
    x_final = x_semi_final - f(x_semi_final)/derivada(x_semi_final)

    return x_final
    #fim do metodo de newton
end
```

Como agora estamos com o raciocínio no tamanho final do intervalo, já o passamos dentro do código o tamanho que foi pedido no enunciado

tamanho_final_do_intervalo = 10^-2

Depois disso, o código segue exatamente igual ao que foi explicado no exercício anterior.

Após o fim da interpolação, o código está printando a variável “x_semi_final”, essa variável é referente a aproximação feita com a interpolação, e que será usada no método de newton como o “chute” para a raiz da função, para que seja retornado um valor ainda mais proximo a raiz.

Relembrando o método de newton:

Matematicamente temos:

f’(chute) = $\frac{\Delta y}{\Delta x}$ = $\frac{f(chute) - 0}{chute - x(\ que\ é\ nossa\ aproximação)}$

Generalizando, temos: f’(x1) = $\frac{f(x1) - 0}{x1 - x2}$

Manipulando, temos que x2 = x1 - $\frac{f(x1)}{f'(x1)}$

Na nossa função nós temos isso codado da seguinte forma:

x_final = x_semi_final - f(x_semi_final)/derivada(x_semi_final)

Ou seja, a variavel x_final será a aproximação final (x2), x_semi_final será nosso x1 (o “chute”), temos também a parcela da subtração, onde f(x_semi_final) é a função no ponto passado, dividido pela derivada dessa função no mesmo ponto(por isso passamos a derivada da função nos parâmetros de bissecao_newton).

E finalmente retornamos x_final no final de bissecao_newton.

Após isso tudo, vamos testar esse método, passando os seguintes parâmetros:

f(x)= x^(5)+x^(4)+(x^3)+4
f''(x)= 5*x^4 + 4*(x^3)+3*(x^2), que será chamado de derivada(x)

Temos então o seguinte resultado:

```
In [79]: #exercicio 1.2
         f(x)= x^(5)+x^(4)+(x^3)+4
         derivada(x)= 5*x^4 + 4*(x^3)+3*(x^2)
         bissecao_newton(f, -100, 100, derivada)

         x_semi_final: -1.3824462890625

Out[79]: -1.3795443138755696
```

Podemos ver que o número retornado em relação ao x_semi_final (aproximação somente pela bisseção) é mais preciso.

OBS 1.3: Expliquei algumas questões e outras apenas coloquei o print do código visto que o professor liberou fazer apenas uma questão para a correção do exercício, mas achei o resultado para todas.

Exercicio 1.3: Sabendo calcular e^x e suas propriedades,queremos aproximar ln(3) por:

Exercício 1.3.1: Método da bisseção com intervalo menor que 10^-3

Para isso vamos utilizar uma função parecida com a que a gente já tem utilizado nos exercícios anteriores.

Como vamos aproximar ln(3) pelo método da bisseção e temos a liberdade de utilizar o método exp(x) para as exponenciais, então vamos tentar encontrar um valor tal que e^x = 3, visto que temos a propriedade ln(e^x) = x.

O método busca raízes para uma função, como queremos e^x = 3, então nossa função será f(x) = e^x - 3 e assim vamos buscar um x tal que f(x) = 0, e esse x será nossa aproximação de ln(3)

A nossa função tem a seguinte cara:

```
#Exercicio 1.3
# dado a e b, tal que f(a) e f(b) tem sinais opostos, retorna x_final, tal
# que |x_final - x_c| <= tamanho_final_do_intervalor/2 com f(x_c) = 0
function bissecao131(f, a, b, tamanho_final_do_intervalo)

    if e_raiz(f,a)
        return a
    end

    if e_raiz(f,b)
        return b
    end

    if !(tem_sinais_opostos(f,a,b))
        return "Os sinais da função não são opostos nesse intervalo"
    end

    iteracoes = floor(log2((b-a)/tamanho_final_do_intervalo))+1

    for i=1:iteracoes
            m=media(a,b)

            if e_raiz(f,m)
                return m
            end

            if tem_sinais_opostos(f,a,m)
                b=m
            else
                a=m
            end
    end

    print("iterações:",iteracoes)
    x_final = media(a,b)
    return x_final

end
```

Ela é muito parecida com a das duas questões anteriores. Entretanto, dessa vez podemos passar direto o tamanho final do intervalo no parâmetro do método, que será utilizado para as iterações da bisseção.

Vamos utilizar os pontos iniciais a= 1 e b= 1.1, pois temos a liberdade de trabalhar com qualquer valor de exp(x). O enunciado também pede que encontremos o valor para um intervalo menor que 10^3.

Como temos que o número de iterações do método da bisseção é dado por:

$$k = \text{floor}(log_2(\frac{b-a}{tam}))+1$$

Então quando passarmos exatamente 10^-3, teremos o seguinte:

$$k = \text{floor}(log_2(\frac{1.1-1}{0.001}))+1 = 7$$

Teremos 7 iterações, e calculando o tamanho final do intervalo a partir das iterações, temos que:

tam = $\frac{b-a}{2^k}$ = $\frac{1.1-1}{2^7}$ = 0.00078125

Ou seja, após o ajuste das iterações, o tamanho fica menor que **10^-3**

Passando isso tudo para os parâmetros da função, chegamos em:

```
In [118]: #Exercicio 1.3.1
          f(x)=exp(x)-3
          bissecao13(f,1,1.1,0.001)

          iterações:7.0

Out[118]: 1.0988281250000003
```

Exercício 1.3.2: o Método de Newton com 20 passos.

Como já foi falado, o método de newton é dado por:

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

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

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

```
#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
```

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:

```
In [84]: #Exercicio 1.3.3

         #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[84]: 1.0982678724638968
```

```
In [83]: #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[83]: 0.00037105636225489186
```

Exercício 1.3.4:

```
In [142]: #exercicio 1.3.4
          function interpolação1(x,y)
              #criar a matriz V
              V=[x.^0 x.^1]
              c=inv(V)*y
              return c #vetor de coeficientes
          end

Out[142]: interpolação1 (generic function with 1 method)
```

```
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
```

```
In [143]: #exercicio 1.3.4
          0.049166805522496304 + 0.3497919842316417  * 3

Out[143]: 1.0985427582174214
```

In [146]:
```
function fatorial(n :: Integer)
    if n < 0
        error("Fatorial somente para n>= 0")
    elseif n==0
        return 1
    else
        return n * fatorial(n-1)
    end
end
```
Out[146]: fatorial (generic function with 1 method)

In [147]:
```
#exercicio 1.3.4
#ERRO

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

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

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

((3-exp(1)) * (3-exp(1.1)))/ fatorial(2)
```
Out[147]: -0.0005868223243925245

In [148]:
```
#exercicio 1.3.4

#M=

-1/(exp(1))^2
```
Out[148]: -0.1353352832366127

In [149]:
```
#exercicio 1.3.4
#Teto do erro

-0.1353352832366127   *  ((3-exp(1)) * (3-exp(1.1)))/ fatorial(2)
```
Out[149]: 7.941776548122973e-5

Exercício 1.3.5:

```
In [152]: #exercicio 1.3.5
function interpolação2(x,y)
    #criar a matriz V
    V=[x.^0 x.^1 x.^2]
    c=inv(V)*y
    #c=V\y #resolver o sistema linear Vc=y
    return c #vetor de coeficientes
end

Out[152]: interpolação2 (generic function with 1 method)
```

```
In [153]: #exercicio 1.3.5
x=[exp(1.1); exp(1.2); exp(1.3)]
y=[1.1, 1.2, 1.3]
interpolação2(x,y)

Out[153]: 3-element Array{Float64,1}:
 -0.3024987510902051
  0.6028903305324462
 -0.04528346645540715
```

```
In [189]: #exercicio 1.3.5
-0.3024987510902051   + 0.6028903305324462 * 3 + -0.04528346645540715 * 9

Out[189]: 1.0986210424084693
```

```
In [200]: #exercicio 1.3.5
# Calculando o erro

# f'''(x) = 2/x^3

# |2/exp(1.1)^3| * ((3-exp(1.1)) (3-e^1.2) (3-e^1.3)) / 3!

(2/exp(1.1)^3) * (((3-exp(1.1))*(3-exp(1.2))*(3-exp(1.3))) / fatorial(3))

Out[200]: -1.0973774030084486e-5
```

Exercício 1.4:

Queremos encontrar o valor de cos(40) utilizando interpolação.

A interpolação nos auxilia a encontrar determinados valores visto que com ela conseguimos uma função polinomial que se aproxima do comportamento de uma outra função desconhecida. Isso é possível através de valores tabelados (x,f(x))

Para cos(40), vamos utilizar os valores tabelados que conhecemos da função cos(x), que foram informados no enunciado.

$\cos(30) = \frac{\sqrt{3}}{2}$

$\cos(45) = \frac{\sqrt{2}}{2}$

$\cos(60) = \frac{1}{2}$

Como temos 3 pontos conhecidos, então nosso polinômio tem grau 2, isso é:

f(x) = a + b*(x) + c*(x^2)

Note que nós sabemos os valores de x e f(x), visto que eles são tabelados, o que precisamos agora é encontrar os coeficientes a, b e c, dessa forma podemos montar o polinômio e substituir o valor x= 40 para que encontremos o resultado.

Para isso temos a seguinte função interpolação2(x,y):

```
In [171]: #exercicio 1.4
          function interpolação2(x,y)
              #criar a matriz V
              V=[x.^0 x.^1 x.^2]
              
              c=V\y #resolver o sistema linear Vc=y
              return c #vetor de coeficientes
          end
```

Nela, vamos passar como parâmetro, x e y, que são os valores tabelados que nós temos, em forma de vetores.

Dentro dela temos o vetor V, nele vamos calcular as potências do polinômio, que serão os pontos x, passados no parâmetro.

O que o método ".^" faz é elevar todos os valores do vetor pelo número desejado.

Vamos então resolver o sistema linear (Vc=y) através de c=V/y, como nós temos as potências, e os valores de y, basta que façamos a divisão de V/y para encontrarmos o vetor c que nos retorna os coeficientes.

Vamos encontrar os coeficientes agora através da função:

```
In [170]: #exercicio 1.4
          x=  [30; 45; 60]
          y = [ (((3)^(1/2))/2) , (((2)^(1/2))/2) , 1/2]
          interpolação2(x,y)

Out[170]: 3-element Array{Float64,1}:
            1.039298173214251
           -0.00256321507508331
           -0.00010708479686368122
```

Nossos coeficiente então são:
a = 1.039298173214251
b = -0.00256321507508331
c = -0.00010708479686368122

Substituindo isso no polinomio, vamos ter
f(x)= 1.0392981732142514 -0.0025632150750832805*x -0.00010708479686368129*x^2

Agora basta substituir x por 40 para encontrarmos uma aproximação para cos(40):

Fazendo isso no júlia, nós temos:

```
In [162]: #exercicio 1.4
          f(x)= 1.0392981732142514 -0.0025632150750832805*x -0.00010708479686368129*x^2
          f(40)

Out[162]: 0.7654338952290302
```

Exercício 1.5:

Queremos descobrir o horário que um assassinato ocorreu. Sabendo que a temperatura normal de um corpo é de 37ºC, e que tempos a medição da temperatura para as seguintes horários:

f(34) = 15
f(30) = 30
f(25) = 17.5

Vamos então buscar f(37) e para isso vamos utilizar interpolação.

Como temos 3 pontos, então nosso polinômio terá a seguinte cara:

f(x) = a+b*x+c*x^2

Vamos utilizar a mesma função interpolação2, que foi explicada e utilizada para o exercício anterior, passando em x os valores das temperaturas, e em y os horários.

Dessa forma, chegamos em:

```
In [373]: #Exercicio 1.5
          y= [15; 16.5; 17.5]
          x= [34; 30; 25]
          interpolação2(x,y)

Out[373]: 3-element Array{Float64,1}:
            7.916666666666664
            0.8694444444444445
           -0.019444444444444445
```

Agora com os coeficientes descobertos, temos então o polinômio:

f(x)= 7.916666666666664 + 0.8694444444444445*x -0.019444444444444445*x^2

Agora basta achar f(37)

```
In [370]: #Exercicio 1.5
          f(x)= 7.916666666666664 + 0.8694444444444445*x -0.019444444444444445*x^2

Out[370]: f (generic function with 2 methods)
```

```
In [202]: f(37)

Out[202]: 13.466666666666665
```

Chegamos então em: 13.466666666666665, o que daria aproximadamente 13:24.

Entretanto temos um problema, quando fazemos a interpolação utilizando a temperatura em y e os horários em x, chegamos em:

```
In [374]: x= [15; 16.5; 17.5]
          y= [34; 30; 25]
          interpolação2(x,y)

Out[374]: 3-element Array{Float64,1}:
           -157.00000000000006
             26.733333333333334
             -0.9333333333333336
```

Com esses coeficientes, chegamos no polinomio:

g(x)= -157.00000000000006 + 26.733333333333334*x -0.9333333333333336*x^2

Se passarmos o horario que achamos anteriormente, 13.466666666666665, deberíamos encontrar 37, mas o que achamos na realidade é:

```
In [375]: #Exercicio 1.5
          g(x)=  -157.00000000000006 + 26.733333333333334*x -0.9333333333333336*x^2

Out[375]: g (generic function with 1 method)
```

```
In [377]: g(13.466666666666665)

Out[377]: 33.74785185185189
```

Achamos aproximadamente 33.7, que é bem distante de 37.

Isso ocorre porque na realidade o resfriamento ocorre de forma exponencial e a nossa função é polinomial, assim podemos dizer que o método da interpolação para esse caso não nos ajuda o que torna o horario do assassinato é inconclusivo.

Exercício 1.6:

Neste exercício temos os seguintes valores tabelados:

A(x, y)	x0 = 1	x1 = 3
y0 = 2	800m	600m
y1 = 4	400m	500m

Isso é:

A(1,2) = 800
A(1,4) = 400
A(3,2) = 600
A(3,4) = 500

Com esses pontos, o que queremos é criar um polinômio através de uma interpolação bilinear ("Lagrange em 2D") e assim nos aproximarmos da maior altura no maior pico e no menor vale da montanha no intervalo dado, ou seja, os máximos e mínimos do polinômio encontrado.

A interpolação bilinear será dada pelo seguinte polinômio:

$$P(x,y) = \frac{(x1-x)\,(y1-y)}{(x1-x0)\,(y1-y0)}A(x0,y0) + \frac{(x-x0)\,(y1-y)}{(x1-x0)\,(y1-y0)}A(x1,y0) + \frac{(x1-x)(y-y0)}{(x1-x0)(y1-y0)}A(x0,y1) + \frac{(x-x0)(y-y0)}{(x1-x0)(y1-y0)}A(x1,y1)$$

Substituindo os valores que nós temos:

$$P(x,y) = \frac{(3-x)\,(4-y)}{(3-1)\,(4-2)}A(1,2) + \frac{(x-1)\,(4-y)}{(3-1)\,(4-2)}A(3,2) + \frac{(1-x)(y-2)}{(3-1)(4-2)}A(1,4) + \frac{(x-1)(y-2)}{(3-1)(4-2)}A(3,4)$$

Como nós temos os valores de $A(1,2)$, $A(3,2)$, $A(1,4)$,$A(3,4)$, substituindo isso no polinômio, e finalizando a conta, chegamos em:

P(x,y)= -275* y -250*x + 75*x*y + 1450

Para encontrarmos os pontos de máximo e minimo, primeiro precisamos encontrar um ponto crítico nessa função, isso é a derivada em relação a x igual a zero e a derivada em relação a y igual a zero.

Temos então:

Px = -250 + 75*y
Py = -275 +75*x

igualando a zero, temos

x = 275/75 = 3.66666666667
y = 250/75 = 3.33333333333

Temos os pontos críticos, mas para sabermos se é máximo ou mínimo, precisamos calcular o determinante D, isso é

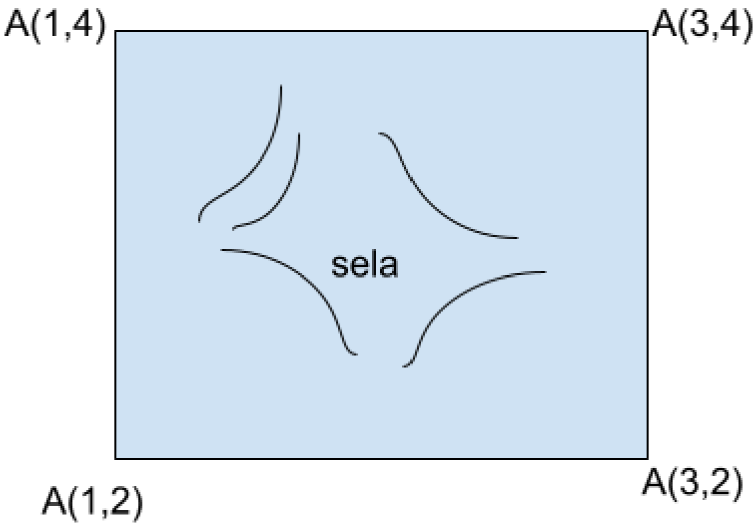
$$D(x,y) = \begin{vmatrix} fxx & fxy \\ fyx & fyy \end{vmatrix} = (fxx)(fyy) - (fxy)^2$$

Substituindo, temos que

D(x,y) = (0)(0) - (75)^2
D(x,y) = - 5625

Descobrimos então que o nosso determinante D é NEGATIVO, logo o que temos é um ponto de sela em P(x,y)

O que temos então é a seguinte situação do plano:



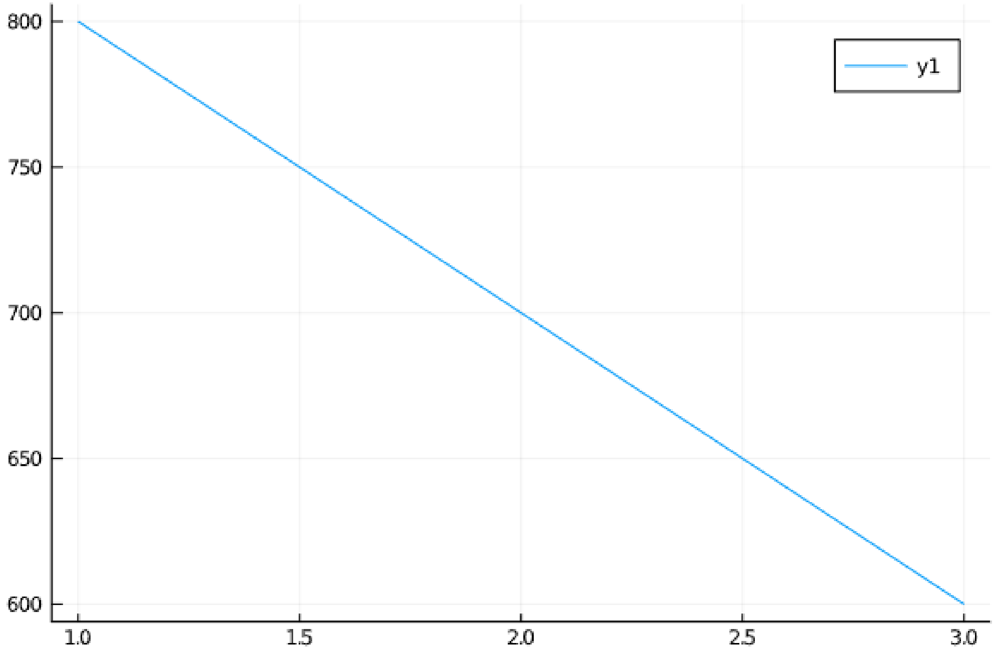
Como descobrimos que nosso ponto crítico é uma sela, então os máximos e mínimos estão nos extremos do plano.

Podemos ver isso quando “prendemos” a função nos pontos x e variamos y e quando “prendemos” o valor de y e variamos x, da seguinte forma:

Variando x no intervalo [1,3] para y = 2

In [470]:
```
fx2(x)=-275* 2 -250*x + 75*x*2 + 1450
plot(fx2,1,3)
```

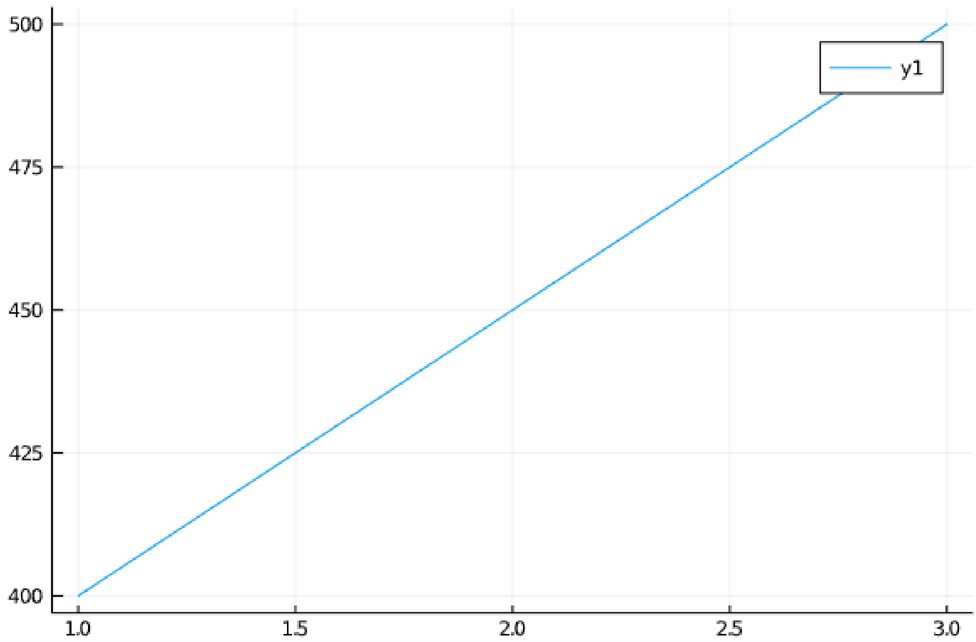
Out[470]:



Variando x no intervalo [1,3] com y = 4

In [472]:
```
fx4(x)=-275* 4 -250*x + 75*x*4 + 1450
plot(fx4,1,3)
```

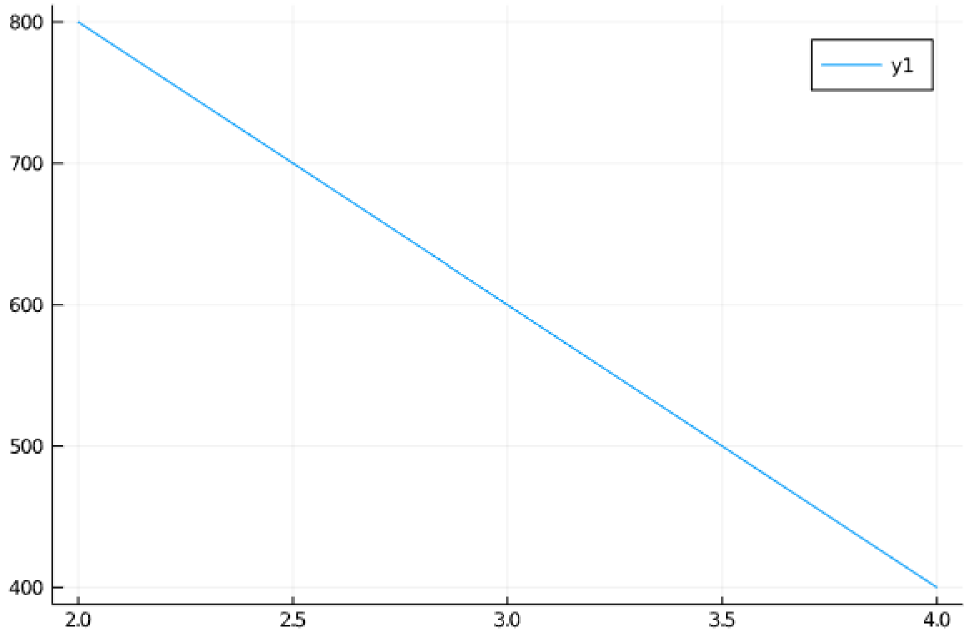
Out[472]:



Variando y no intervalo [2,4] com x = 1

In [475]:
```
f1y(y)= -275* y -250*1 + 75*1*y + 1450
plot(f1y,2,4)
```

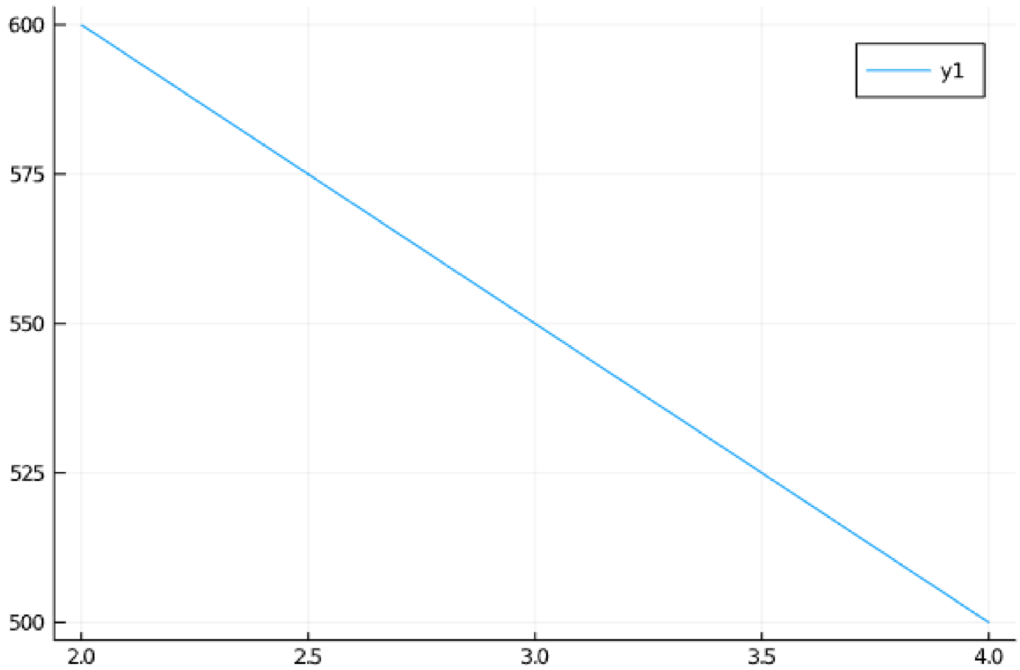
Out[475]:



Variando y no intervalo [2,4] com x = 3

In [477]:
```
f3y(y)= -275* y -250*3 + 75*3*y + 1450
plot(f2y,2,4)
```

Out[477]:



Logo o nosso pico mais alto é de 800m no ponto A(1,2) e o vale mais baixo é de 400m no ponto A(1,4)

Exercício 1.7:

Nessa questão queremos implementar uma função que receba 5 pontos e desenhe os polinômios de grau 3 por partes, e sem bicos.

Como são 5 pontos, e queremos que as funções plotadas não tenham ”bicos” e sejam polinômios de grau 3 logo teremos a seguinte esquemática:

Chamaremos nossos polinômios de P3 e Q3 (pois são de grau 3)

P3(x0) = a0 + a1*x0 + a2*(x0^2) + a3(x0^3) = y0
P3(x1) = a0 + a1*x1 + a2*(x1^2) + a3(x1^3) = y1
P3(x2) = a0 + a1*x2 + a2*(x2^2) + a3(x2^3) = y2

Q3(x2) = a0 + b1*x2 + b2*(x2^2) + b3(x2^3) = y2
Q3(x3) = a0 + b1*x3 + b2*(x3^2) + b3(x3^3) = y3
Q3(x4) = a0 + b1*x4 + b2*(x4^2) + b3(x4^3) = y4

Dessa forma temos P3(x2) = Q3(x2) = y2, para que possamos ter a “liga” entre os dois polinômios.

Para que não hajam bicos também, precisamos que as derivadas naquele ponto sejam as mesmas, o que nos garante mais uma equação:

P3’(x2) = Q’(x2) =

a1* + 2* a2*(x2) + 3* a3(x2^2) = b1 + 2* b2*(x2) + 3*b3*(x2^2)

a1* + 2* a2*(x2) + 3* a3(x2^2) -   b1 - 2* b2*(x2) - 3*b3*(x2^2) = 0

Agora também vamos garantir que a curvatura entre elas seja igual, para que isso seja possível, vamos garantir que a segunda derivada também seja igual:

P3’’(x2) = Q’’(x2)

2* a2 + 6* a3* (x2) = 2* b2* + 6* b3*(x2)

2* a2 + 6* a3* (x2) - 2* b2* + 6* b3*(x2) = 0

Para fazermos então a interpolação , baseado nessas 8 equações, teremos que resolver o seguinte sistema linear:

Chamando a matriz principal de V, os coeficiente de c e o resultado de y

temos a seguinte multiplicação de matrizes: Vc = y

$$
\begin{vmatrix}
1 & 1*(x0) & 1*(x0^2) & 1*(x0^3) & 0 & 0 & 0 & 0 \\
1 & 1*(x1) & 1*(x1^2) & 1*(x1^3) & 0 & 0 & 0 & 0 \\
1 & 1*(x2) & 1*(x2^2) & 1*(x2^3) & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1*(x2) & 1*(x2^2) & 1*(x2^3) \\
0 & 0 & 0 & 0 & 1 & 1*(x3) & 1*(x3^2) & 1*(x3^3) \\
0 & 0 & 0 & 0 & 1 & 1*(x4) & 1*(x4^2) & 1*(x4^3) \\
0 & 1 & 2*(x2) & 3*(x2^2) & 0 & -1 & -2*(x2) & -3*(x3^2) \\
0 & 0 & 1 & 6*(x2) & 0 & 0 & -1 & -6*(x2)
\end{vmatrix}
\;\mathrm{x}\;
\begin{vmatrix} a0 \\ a1 \\ a2 \\ a3 \\ b0 \\ b1 \\ b2 \\ b3 \end{vmatrix}
= \begin{vmatrix} y0 & y1 & y2 & y3 & y3 & y\,4 & 0 & 0 \end{vmatrix}
$$

Então para encontrarmos os coeficientes, basta fazer c = V \y

Implementando isso no Julia, nós temos a seguinte função:

```julia
#Exercicio 1.7
#x = [x0,x1,x2,x3,x4]
#y = [y0,y1,y2,y3,y4,y5,y6,y7]
function interpolaPQ_e_plota(xs,ys)
    #criar a matriz
    px=[xs[1],xs[2],xs[3]] #pontos x0 até x2 da função p(x)
    qx=[xs[3],xs[4],xs[5]] #pontos x2 até x4 da função q(x)

    p3=[px.^0 px.^1 px.^2 px.^3 px*0 px*0 px*0 px*0] #calculando as potencias de p3(x0) até p3(x2)
    q3=[qx*0 qx*0 qx*0 qx*0 qx.^0 qx.^1 qx.^2 qx.^3] #calculando as potencias de q3(x2) até q3(x4)

    D1 = [0 1 2*xs[3]  3*((px[3])^2) 0 -1 -2*xs[3] -3*(xs[3]^2)] #p'(x2)= q'(x2) -> p'(x2)-q'(x2) = 0
                                                                #derivada de p(x2) igual a derivada que q(x2)

    D2 = [0 0 1 6*xs[3] 0 0 -1 (-6)*xs[3]] #p''(x2)= q''(x2) -> p''(x2)-q''(x2) = 0
                                           #derivada de p(x2) igual a derivada que q(x2)

    V=[p3;q3;D1;D2] #unindo tudo em uma matriz só

    y=[ys[1] ,ys[2],ys[3],ys[3],ys[4],ys[5],0,0] #os pontos de y passados como parametro,
                                                 #repetindo o ponto ys[3] devido a junção das equaçoes no ponto x2

    c=V\y #resolver o sistema linear Vc=y

    p(x) = c[1] + c[2]*x + c[3]*x^2 + c[4]*x^3 #P3(x)
    q(x) = c[5] + c[6]*x + c[7]*x^2 + c[8]*x^3 #Q3(x)

    plot(p,xs[1],xs[3],lw=3,c=:yellow,lab="p(x)")  #plotando a equação p(x)
    plot!(q,xs[3],xs[4],lw=3,c=:black,lab="q(x)")  #plotando a equação q(x)

end
```

Na função interpolaPQ_e_plota, nós recebemos os pontos x e y como parâmetro, pegamos os pontos de x respectivos para cada pos polinômios p(x) e q(x), depois para cada um deles, nós fazemos suas respectivas potencias.

Depois disso nós montamos as linhas da matriz V referentes a igualdade da primeira e segunda derivada no ponto x2, que são os vetores D1 e D2. Depois unimos tudo em um só vetor chamado V, que será nossa matriz.

Depois disso pegamos os pontos y passados em ys no parâmetro e colocamos em um vetor também chamado de y de forma que fique semelhante a | y0 y1 y2 y3 y3 y 4 0 0 |

Como já temos V e y, agora basta as matrizes para que cheguemos em um vetor de coeficientes c.

Depois disso nós montamos os polinômios p(x) e q(x) de forma que conseguimos substituir os coeficientes achados em c no formato:

P(x) = a0 + a1*x + a2*(x^2) + a3(x^3)
Q(x) = b0 + b1*x + b2*(x^2) + b3(x^3)

Finalmente plotamos P(x) no intervalo de [x0,x2] e Q(x) no intervalo [x2,x4].

Testando a função, passando pontos aleatórios, ficamos com esse resultado:

In [518]:
```
x=[-2, 18, 6, 13, 661]
y=[-1, 57,25, 50, 07]

interpolaPQ_e_plota(x,y)
```

Out[518]:

Podemos ver que na junção das funções, não há bicos.