MODELAGEM DOS CASOS DE COVID-19 NO BRASIL PELO MÉTODO DE QUADRADOS MÍNIMOS

Aluno: Eric Yutaka Fukuyama, Letícia do Rocio Oliveira
ericfukuyama@alunos.utfpr.edu.br, leticia.rocio@ufpr.br

Professores: Elizabeth Karas e Lucas Pedroso

1º Semestre de 2021

1 Introdução

A pandemia global de COVID-19 vem despertando interesse crescente em várias áreas da comunidade científica, desde o desenvolvimento e produção de vacinas para a imunização da população até a modelagem da progressão do contágio pelo vírus. O assunto tem sido constantemente abordado por pesquisadores da área da matemática em artigos e trabalhos, em todos os lugares do mundo, como por exemplo em [1, 5].

Neste projeto, buscamos realizar uma previsão do contágio pelo vírus no Brasil, analisando a média móvel do número de casos durante o período de um ano e utilizando estes dados para estimar uma função que descreva o comportamento da transmissão. Para isto, nossa principal referência será [1] e apresentaremos os resultados dos testes computacionais realizados.

2 O problema

Nesta seção, vamos apresentar o problema que estamos estudando, assim como a teoria matemática e o método de Programação Não-Linear utilizado para obter os resultados. A seção tem como referências [1, 2, 3].

2.1 Método dos Quadrados Mínimos

Sejam $(x_i, y_i) \in \mathbb{R}^2$, com $i = 1, 2, ..., n$, as coordenadas de n pontos. Nosso objetivo é obter uma função $f : \mathbb{R} \to \mathbb{R}$, tal que o valor de $f(x_i)$ seja o mais próximo possível do valor de y_i. Para isto, utilizaremos o Método dos Quadrados Mínimos, que consiste em minimizar a soma dos quadrados das distâncias entre os pontos (x_i, y_i) e $(x_i, f(x_i))$, conforme ilustra a Figura 1.

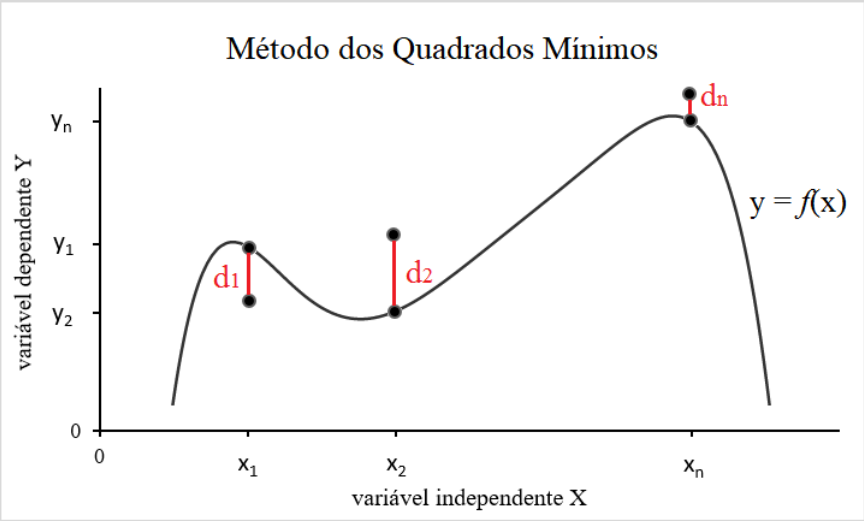

Figura 1: Método dos Quadrados Mínimos. Fonte: Autores.

Como nosso objetivo é minimizar esta soma de distâncias ao quadrado, queremos encontrar o menor valor D, tal que este valor D é chamado de Erro Absoluto da aproximação $f(x)$ e calculado como

$$D = \sqrt{\sum_{i=1}^{n}(f(x_i) - y_i))^2}.$$

No entanto, minimizar D é equivalente à minimizar a expressão

$$E = \sum_{i=1}^{n}(f(x_i) - y_i))^2, \tag{1}$$

devido a propriedades da norma.

É necessário destacar que, neste trabalho, os valores x_i representam o número de semanas. Os valores de y_i são os casos reais de contaminados por COVID-19 na semana x_i e a função $f(x_i)$ expressa o valor esperado para o número de contaminados na semana x_i. Além de ser utilizado n como a quantidade de pontos na tabela dos dados do problema.

Podemos aproximar os valores de y_i por diversos tipos de função, as mais conhecidas são: Função Linear, Função Quadrática e Função Exponencial. Neste trabalho, vamos utilizar as aproximações Linear, Quadrática e Cúbica, que serão brevemente explicadas a seguir.

Desta forma, foi escolhido o Algoritmo do Gradiente com Busca de Armijo e o Método de Newton para resolver estes problemas.

2.1.1 Modelo Linear

O modelo mais simples de aproximação é o que considera uma função linear, ou seja, uma função do tipo

$$f(x) = a_0 + a_1 x,$$

em que a_j, tal que $j = 0, 1, ..., n$, no problema em questão n é equivalente ao grau do polinômio buscado, são os coeficientes da aproximação e são as incógnitas procuradas para serem otimizadas. Ao substituirmos $f(x)$ na equação (1), obtemos a expressão

$$E(a_0, a_1) = \sum_{i=1}^{n}(a_0 + a_1 x_i - y_i))^2. \tag{2}$$

2.1.2 Modelo Quadrático

Outro modelo estudado foi o Quadrático, em que a função que é buscada é da forma

$$f(x) = a_0 + a_1 x + a_2 x^2.$$

Consequentemente, para o cálculo do erro ao quadrado, é utilizada a equação (1). Desta forma, obtêm-se

$$E(a_0, a_1, a_2) = \sum_{i=1}^{n} (a_0 + a_1 x_i + a_2 x_i^2 - y_i))^2. \tag{3}$$

2.1.3 Modelo Cúbico

Além disso, o modelo final considerado foi o modelo Cúbico. De forma análoga, foi obtida a função

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

Ademais, pode ser verificado que

$$E(a_0, a_1, a_2, a_3) = \sum_{i=1}^{n} (a_0 + a_1 x_i + a_2 x_i^2 + a_3 x_i^3 - y_i))^2. \tag{4}$$

2.2 Modelagem Matemática

Para a Modelagem dos três problemas, foi utilizado o seguinte teorema

Theorem 1 *Seja $f : \mathbb{R}^n \to \mathbb{R}$ diferenciável no ponto $x^* \in \mathbb{R}^n$. Se x^* é um minimizador local de f, então*

$$\nabla f(x^*) = 0. \tag{5}$$

A demonstração do teorema é encontrada em [2].
A partir deste Teorema, é possível desenvolver para os três casos estudados.

2.2.1 Modelo Linear

Queremos minimizar a equação (2). Deste modo, o vetor gradiente é dado por

$$\begin{bmatrix} 2\sum_{i=1}^{n} (a_0 + a_1 x_i - y_i) \\ 2\sum_{i=1}^{n} x_i (a_0 + a_1 x_i - y_i) \end{bmatrix}$$

Todavia, de (5), é possível verificar que

$$\begin{bmatrix} 2\sum_{i=1}^{n}(a_0 + a_1x_i - y_i) \\ 2\sum_{i=1}^{n} x_i(a_0 + a_1x_i - y_i) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Assim, rearranjando os termos, é possível encontrar uma multiplicação de matrizes análoga a esse sistema. Nossa motivação é encontrar o vetor de incógnitas que satisfaça a relação $Az = y$:

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \end{bmatrix}.$$

Desta forma, se busca a solução do problema.

2.2.2 Modelo Quadrático

De forma análoga, é possível fazer para o Modelo Quadrático algo semelhante ao feito para o Modelo Linear. Assim,

$$\begin{bmatrix} 2\sum_{i=1}^{n}(a_0 + a_1x_i + a_2x_i^2 - y_i) \\ 2\sum_{i=1}^{n} x_i(a_0 + a_1x_i + a_2x_i^2 - y_i) \\ 2\sum_{i=1}^{n} x_i^2(a_0 + a_1x_i + a_2x_i^2 - y_i) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Ao rearranjar os termos, temos:

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} x_i^2 y_i \end{bmatrix}$$

.

2.2.3 Modelo Cúbico

Analogamente, para o Modelo Cúbico será obtido

$$\begin{bmatrix} 2\sum_{i=1}^{n}(a_0 + a_1x_i + a_2x_i^2 + a_3x_i^3 - y_i) \\ 2\sum_{i=1}^{n} x_i(a_0 + a_1x_i + a_2x_i^2 + a_3x_i^3 - y_i) \\ 2\sum_{i=1}^{n} x_i^2(a_0 + a_1x_i + a_2x_i^2 + a_3x_i^3 - y_i) \\ 2\sum_{i=1}^{n} x_i^3(a_0 + a_1x_i + a_2x_i^2 + a_3x_i^3 - y_i) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

portanto, temos a relação

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 & \sum_{i=1}^{n} x_i^5 \\ \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 & \sum_{i=1}^{n} x_i^5 & \sum_{i=1}^{n} x_i^6 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_iy_i \\ \sum_{i=1}^{n} x_i^2y_i \\ \sum_{i=1}^{n} x_i^3y_i \end{bmatrix}$$

.

2.3 Método do Gradiente com Busca de Armijo

O primeiro algoritmo utilizado foi o Método do Gradiente. Este método consiste em usar como direção de descida o sentido oposto ao do gradiente, sendo importante ressaltar que tal algoritmo segue um movimento de zigue-zague.

No entanto, não foi utilizado o método com a busca exata, pois o custo seria mais elevado. Dessa forma, foi usada a Busca de Armijo, que, como está em [2], procura uma boa redução da função ao longo da direção, sem tentar minimizá-la. Essa busca é determinada por

$$f(\bar{x} + td) \leq f(\bar{x}) + t\nabla f(\bar{x})^T d.[2] \tag{6}$$

2.4 Método de Newton

O segundo método usado foi o Método de Newton. Este algoritmo segue a ideia de aproximar a função estudada pelo polinômio de Taylor de primeira ordem. Assim, segue uma direção dada por

$$d^k = -(\nabla^2 f(x^k))^{-1}\nabla(x^k).[2] \tag{7}$$

Além disso, é necessário ressaltar que foi usada a busca de Armijo, cuja ideia é a mesma do Método anterior.

3 Experimentos Numéricos

Nesta seção, serão apresentados os resultados que foram obtidos na resolução do problema. Os métodos foram implementados em Julia (versão 1.5.1). Foram utilizados os pacotes *LinearAlgebra* para efetuar operações com matrizes, *DelimitedFiles* para realizar a leitura dos arquivos de dados e *Plots* para gerar os gráficos das funções de aproximação. Todos códigos estão disponíveis no Apêndice I ou no GitHub `https://github.com/leticiaoliv/Projeto`.

3.1 O algoritmo implementado

O programa consiste em 5 arquivos. O documento "dados_x.csv"são o número das semanas estudadas, sendo a semana inicial a semana 1 do estudo. O arquivo "dados_y.csv"são os infectados pelo coronavirus durante uma determinada semana. Os arquivos intitulados "gradiente.jl"e "newton.jl"são os Métodos do Gradiente e do Newton, respectivamente. Por último, a função "main.jl"usa os arquivos auxiliares para resolver o problema e gerar os gráficos

Em primeiro plano, é importante destacar que os dois métodos foram definidos com o mesmo critério de parada, sendo como requisitos para o código continuar $\|\nabla f(x^k)\| > 10^{-4}$ e $k < 10000$, em que k é o número de iterações.

Além disso, no cálculo do gradiente e da função foram usados os arranjos definidos na seção Modelagem Matemática e conferidas em [1]. Ademais, o fator γ e η foram definidos como $0,5$ e $0,1$ por opção, uma vez que $\gamma \in (0;1)$ e $\eta \in (0;0,5)$.

3.2 Resultados obtidos

O código *main( )* foi rodado utilizando dados sobre COVID-19 obtidos em [4]. Em ambas as aproximações, o Método do Gradiente realizou 10000 iterações, que era um dos nossos critérios de parada. Já o Método de Newton realizou apenas uma iteração para cada uma das três aproximações, satisfazendo a condição da norma do

vetor gradiente estabelecida na seção 3.1.

Para a aproximação linear, as funções obtidas pelos métodos do Gradiente e de Newton são $f(x) = 316570.7 - 2570x$ e $f(x) = 323438.9 - 2766.3x$, respectivamente, e estão representadas graficamente abaixo.

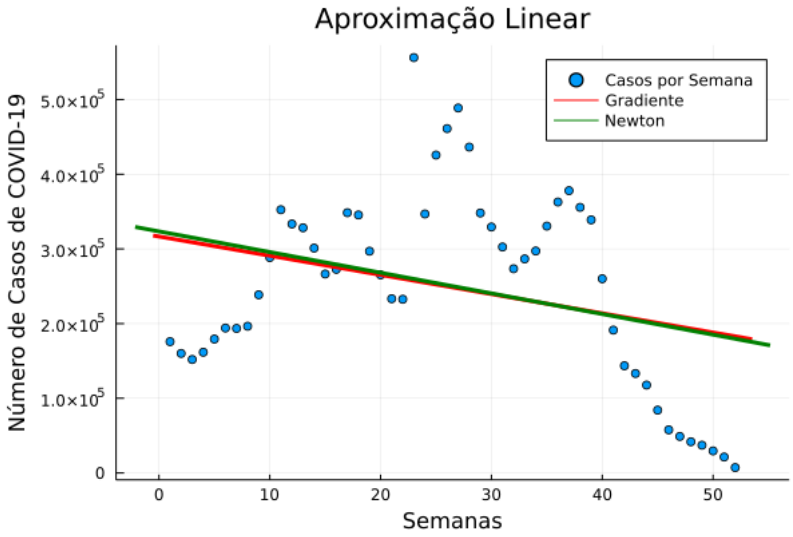

Figura 2: Aproximações Lineares. Fonte: Autores.

Já para o modelo de aproximação quadrático, os coeficientes obtidos resultaram nas funções $f(x) = 900.3 + 10975.4x - 145.6x^2$ para o Método do Gradiente e $f(x) = 79670.3 + 24319.1x - 511.1x^2$ para o Método de Newton. Abaixo, a representação gráfica das duas funções.

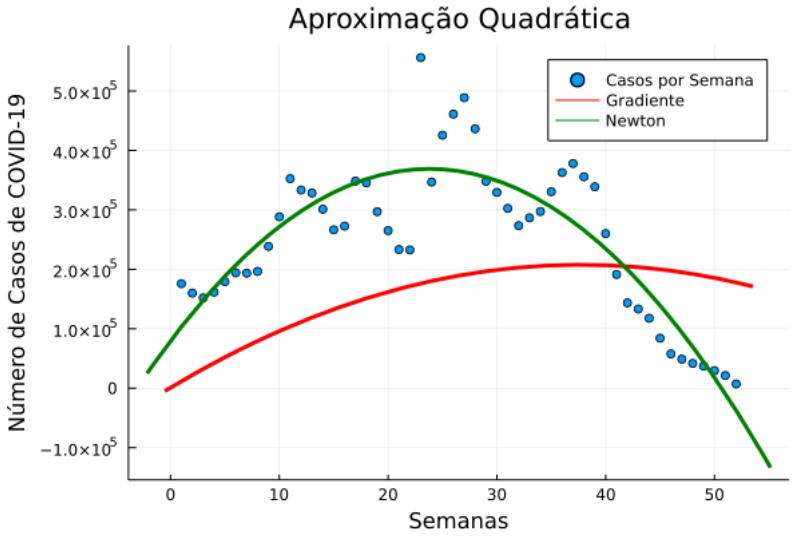

Figura 3: Aproximações Quadráticas. Fonte: Autores.

E no último caso, para a aproximação cúbica, as funções são $f(x) = 1.6 + 11.6x + 165.9x^2 - 1.7x^3$ como resultado do Método do Gradiente e $f(x) = 110426.9 + 17669.9x - 200.4x^2 - 3.9x^3$ para o Método de Newton. A Figura 4 representa graficamente cada uma das duas funções.

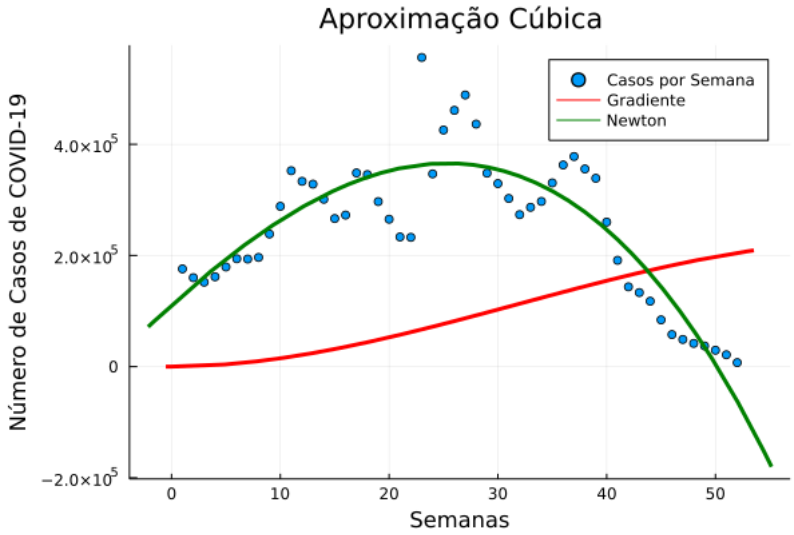

Figura 4: Aproximações Cúbicas. Fonte: Autores.

Para comparar qual das três aproximações de cada método foi a melhor, vamos calcular o Erro Relativo dos seis resultados obtidos. Utilizar o Erro Relativo para fazer esta comparação é importante devido à ordem de grandeza dos dados utilizados e este erro será calculado de seguinte forma:

$$E_{rel} = \frac{D}{\|y\|} = \frac{\sqrt{\sum_{i=1}^{n}(f(x_i) - y_i))^2}}{\sqrt{\sum_{i=1}^{n} y_i^2}},$$

com y_i sendo o vetor do número de casos de COVID-19 e $f(x_i)$ cada uma das funções de aproximação obtidas anteriormente.

A Tabela 1 apresenta os valores de Erro Relativo para cada aproximação.

Aproximação	**Erro Gradiente**	**Erro Newton**
Linear	0.428972	0.428802
Quadrática	0.576880	0.222370
Cúbica	0.817927	0.219287

Tabela 1: Erro Absoluto das Aproximações. Fonte: Autores

4 Conclusões

Os resultados obtidos para a solução com Gradiente não condizem com o esperado, sendo considerada uma solução ruim para o problema. Isso porque o programa parou na condição limite de iterações que foi posta, dessa forma, em nenhum dos três casos, foi alcançado um valor menor da norma do gradiente do que a colocada inicialmente. Assim, é visto pelos gráficos que o método do gradiente gerou uma aproximação ruim.

Por outro lado, a solução do Método de Newton para quadráticas resolve o problema em uma iteração, por causa disso, a solução torna-se ótima, pois o problema é resolvido de forma rápida e precisa. Tudo isso é confirmado, pela Tabela 1, uma vez que o método de Newton obteve erros menores quando comparados ao método do Gradiente e, apesar da Função Cúbica apresentar o menor erro, a solução quadrática também é bastante recomendável, pois a diferença do erro é pequena e exige um desempenho computacional menor.

Referências

[1] MARQUES, S.V. *Método de quadrados mínimos em época de pandemia de COVID-19.* Curitiba: [S.n.], 2020.

[2] RIBEIRO, A.A.; KARAS, E.W. *Otimização contínua: Aspectos teóricos e computacionais.* São Paulo: Cengage Learning, 2013.

[3] RUGGIERO, M.A.G.; LOPES, V.L.R. *Cálculo Numérico: Aspectos Teóricos e Computacionais.* - 2ª ed. São Paulo: Pearson Makron Books, 1996.

[4] Fundação Oswaldo Cruz - Monitora Covid-19, 2021. Disponível em: https://bigdata-covid19.icict.fiocruz.br/ Acesso em: 10 de dez. de 2021.

[5] GRILLO ARDILA, E. K.; BRAVO OCAÑA, L. E.; GUERRERO, R.; SANTAELLA-TENORIO, J. Mathematical models and the coronavirus, COVID-19. Colombia Médica, v. 51, n. 2, p. e-4277, 14 maio 2020.

Apêndice I

- **Código Método do Gradiente com Busca de Armijo**

```
function gradiente(A::Matrix, G:: Matrix, y::Array, g::Array, z::Array)
    k = 0
    t = 1
    gamma = 0.5
    eta = 0.1
    f(z) = sum((A*z - y).^2)
    grad(z) = G*z - g
    while norm(grad(z)) > 1e-4 && k < 10000
        d = -grad(z)
        while f(z + t*d) > f(z) + eta*t*grad(z)'*d
            t = gamma*t
        end
        z = z + t*d
        k = k + 1
    end
    return z
end
```

- **Código Método de Newton**

```
function newton(A::Matrix, G:: Matrix, y::Array, g::Array, z::Array)
    k = 0
    t = 1
    gamma = 0.5
    eta = 0.1
    f(z) = sum((A*z - y).^2)
    grad(z) = G*z - g
    hes = G
    while norm(grad(z)) > 1e-4 && k < 10000
        d = -inv(hes)*grad(z)
        while f(z + t*d) > f(z) + eta*t*grad(z)'*d
            t = gamma*t
        end
        z = z + t*d
        k = k + 1
    end
    return z
end
```

- **Código da Função Principal**

```
using Plots
using LinearAlgebra
using DelimitedFiles

function main()
    x = readdlm("dados_x.csv");
    y = readdlm("dados_y.csv");


    #Aproximação Linear

    A = [ones(52) x];
    G = [52 sum(x);
         sum(x) sum(x.^2)];
    g = [sum(y); sum(x.*y)];
    z = ones(2);

    gr = gradiente(A, G, y, g, z)
    nt = newton(A, G, y, g, z)
    errograd = sqrt(sum((A*gr - y).^2))/norm(y);
    erronewt = sqrt(sum((A*nt - y).^2))/norm(y);
    print(" Aproximação Linear")
    print(" Coeficientes Gradiente = ", gr,
          " Coeficientes Newton = ", nt)
    print(" Erro Absoluto Gradiente = ", sqrt(sum((A*gr - y).^2)),
          " Erro Absoluto Newton =", sqrt(sum((A*nt - y).^2)))
    print(" Erro Relativo Gradiente = ", errograd,
          " Erro Relativo Newton = ", erronewt)


    scatter(x, y, title = "Aproximação Linear",
            label = "Casos por Semana", lw = 3)
    plot!(x -> gr[1] + gr[2]*x, label = "Gradiente", c=:red, lw=3)
    plot!(x -> nt[1] + nt[2]*x, label = "Newton", c=:green, lw=3)
    xlabel!("Semanas")
    ylabel!("Número de Casos de COVID-19")
    png("AproxLin")


    #Aproximação Quadrática

    A = [ones(52) x x.^2];
```

```
G = [52 sum(x) sum(x.^2);
     sum(x) sum(x.^2) sum(x.^3);
     sum(x.^2) sum(x.^3) sum(x.^4)];
g = [sum(y); sum(x.*y); sum(x.^2 .*y)];
z = ones(3);

gr = gradiente(A, G, y, g, z)
nt = newton(A, G, y, g, z)
errograd = sqrt(sum((A*gr - y).^2))/norm(y);
erronewt = sqrt(sum((A*nt - y).^2))/norm(y);
print(" Aproximação Quadrática")
print(" Coeficientes Gradiente = ", gr,
      " Coeficientes Newton = ", nt)
print(" Erro Absoluto Gradiente = ", sqrt(sum((A*gr - y).^2)),
      " Erro Absoluto Newton =", sqrt(sum((A*nt - y).^2)))
print(" Erro Relativo Gradiente = ", errograd,
      " Erro Relativo Newton = ", erronewt)

scatter(x, y, title = "Aproximação Quadrática",
        label = "Casos por Semana", lw = 3)
plot!(x -> gr[1] + gr[2]*x + gr[3]*x.^2,
      label = "Gradiente", c=:red, lw=3)
plot!(x -> nt[1] + nt[2]*x + nt[3]*x.^2,
      label = "Newton", c=:green, lw=3)
xlabel!("Semanas")
ylabel!("Número de Casos de COVID-19")
png("AproxQuad")


#Aproximação Cúbica

A = [ones(52) x x.^2 x.^3];
G = [52 sum(x) sum(x.^2) sum(x.^3);
    sum(x) sum(x.^2) sum(x.^3) sum(x.^4);
    sum(x.^2) sum(x.^3) sum(x.^4) sum(x.^5);
    sum(x.^3) sum(x.^4) sum(x.^5) sum(x.^6)];
g = [sum(y); sum(x.*y); sum(x.^2 .*y); sum(x.^3 .*y)];
z = ones(4);

gr = gradiente(A, G, y, g, z)
nt = newton(A, G, y, g, z)
errograd = sqrt(sum((A*gr - y).^2))/norm(y);
erronewt = sqrt(sum((A*nt - y).^2))/norm(y);
```

```
    print(" Aproximação Cúbica")
    print(" Coeficientes Gradiente = ", gr,
          " Coeficientes Newton = ", nt)
    print(" Erro Absoluto Gradiente = ", sqrt(sum((A*gr - y).^2)),
          " Erro Absoluto Newton =", sqrt(sum((A*nt - y).^2)))
    print(" Erro Relativo Gradiente = ", errograd,
          " Erro Relativo Newton = ", erronewt)


    scatter(x, y, title = "Aproximação Cúbica",
            label = "Casos por Semana", lw = 3)
    plot!(x -> gr[1] + gr[2]*x + gr[3]*x.^2 + gr[4]*x.^3,
          label = "Gradiente", c=:red, lw=3)
    plot!(x -> nt[1] + nt[2]*x + nt[3]*x.^2 + nt[4]*x.^3,
          label = "Newton", c=:green, lw=3)
    xlabel!("Semanas")
    ylabel!("Número de Casos de COVID-19")
    png("AproxCub")

end
```