

Punto 2:

Descargue el archivo **Datos.txt** de la página del curso. En este encontrará un conjunto de **21** datos.

Copie estos datos y calcule el polinomio de ajuste de grado **5**:

$p(x) = c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4 + c_5x^5$ utilizando los métodos de ecuaciones normales y factorización **QR**. Compare sus resultados con los valores certificados $c_i = 1$ para $i = 0, 1, 2, 3, 4, 5$. Encuentre el residual $\|A\mathbf{c} - \mathbf{y}\|_2$ en cada caso, así como la diferencia relativa con respecto a los valores certificados. Escriba sus conclusiones.

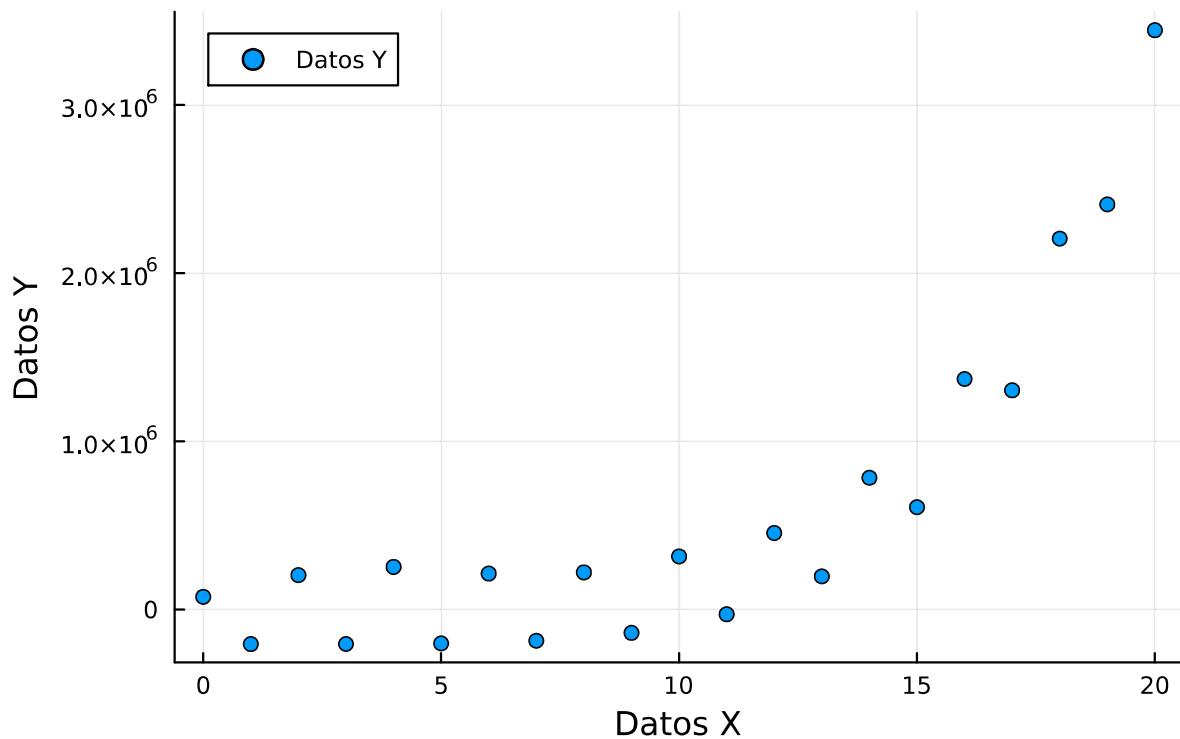
Los datos son los siguientes:

0	75901
1	-204794
2	204863
3	-204436
4	253665
5	-200894
6	214131
7	-185192
8	221249
9	-138370
10	315911
11	-27644
12	455253
13	197434
14	783995
15	608816
16	1370781
17	1303798
18	2205519
19	2408860
20	3444321

1 using LinearAlgebra, Plots, Optim

```
► [75901.0, -204794.0, 204863.0, -204436.0, 253665.0, -200894.0, 214131.0, -185192.0, 221249.  
1 begin  
2     #Datos en forma de matriz para el manejo en el código  
3 Datos=  
4 [0.    75901.  
5  1.   -204794.  
6  2.    204863.  
7  3.   -204436.  
8  4.    253665.  
9  5.   -200894.  
10 6.    214131.  
11 7.   -185192.  
12 8.    221249.  
13 9.   -138370.  
14 10.   315911.  
15 11.   -27644.  
16 12.   455253.  
17 13.   197434.  
18 14.   783995.  
19 15.   608816.  
20 16.   1370781.  
21 17.   1303798.  
22 18.   2205519.  
23 19.   2408860.  
24 20.   3444321.  
25 ]  
26  
27 DatosX = Datos[:,1] #Columna 1, los X_i  
28 DatosY = Datos[:,2] #Columna 2, los Y_i  
29 end
```

Gráfica datos



```
1 scatter(DatosX, DatosY, ls=:dash, label="Datos Y", lw=4, xlabel="Datos X",
yaxis="Datos Y", title="Gráfica datos")
2
```

Notación estándar:

A continuación se presentará la notación usada en el código para facilitar su entendimiento. Esta constará de algunos nombres para los datos y las variables como prefijos que nos dirán su utilidad, mientras que los sufijos nos dirán información acerca de cuál es el método usado para ese valor en particular.

Prefijos:

Prefijos de datos:

- **Datos**: la matriz 21×2 con los datos que requieren el ajuste.
- **DatosX** y **DatosY**: Primera y segunda columna de los datos respectivamente, donde corresponderán **DatosX_i** = x_i y **DatosY_i** = y_i como (x_i, y_i) .
- **arrAux**: un arreglo que se inicializa con unos, del mismo tamaño que **DatosX**.
- La notación fija para el sistema de ecuaciones será $\mathbf{AC} = \mathbf{B}$, donde **A** es la matriz con el sistema de ecuaciones, **C** es el vector de incógnitas, y **B** el vector de términos independientes. (El sistema de ecuaciones se planteará explícitamente más adelante).

Prefijos de funciones:

Para las funciones que serán utilizadas:

- **Res** y **NRes**: toda función que comience con **Res** será una función de residuo entre nuestro valor ideal y el dado por los datos; en caso de ser **NRes**, es la norma del residuo.
- **Dif** y **NDif**: dado que se nos da la respuesta exacta del problema, las funciones que comienzan por **Dif** mostrarán la diferencia entre el valor objetivo y el obtenido por el método en cuestión; **NDif** representará la norma de esta diferencia. Téngase en cuenta que los valores certificados son **1** en cada coeficiente; por lo tanto, la diferencia relativa será igual a la diferencia entre los valores.
- **Eval**: función de evaluación del polinomio de grado quíntico, dados los coeficientes en C_i , esto se hará evaluando la función respectiva; en caso de que sea la multiplicación de alguna matriz de ajuste con C , se usará **Evalz**.

Sufijos:

Se usarán **4** sufijos principales, cada uno acorde a un modelo para ajuste de datos diferente (para las funciones genéricas respecto a los modelos se usará el término **Modelo**); estos serán conectados con los prefijos a través del símbolo "`_`":

- **Best**: Aquella que usa los valores dados en el enunciado del ejercicio como óptimos.
- **Optim**: usa el comando *Optim* de la librería correspondiente para hallar una estimación, usa métodos ajenos al curso, pero útiles al comparar.
- **MC**: Abreviatura de Mínimos cuadrados, corresponde a tal método.
- **QR**: Usando la factorización *QR* para la solución del problema de ajuste.

Vale aclarar que algunos de estos métodos requieren sus propias derivaciones; en tal caso se planteará la notación adicional requerida al inicio de cada problema.

Sistema de ecuaciones:

Nuestro sistema de ecuaciones entonces busca solucionar un problema de interpolación, en donde buscaremos plantear $AC = B$, donde $A = A(X)$ y $B = B(Y)$; en particular tendrían la siguiente forma:

$$AC = B$$

$$\begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & x_0^4 & x_0^5 \\ 1 & x_1 & x_1^2 & x_1^3 & x_1^4 & x_1^5 \\ 1 & x_2 & x_2^2 & x_2^3 & x_2^4 & x_2^5 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{20} & x_{20}^2 & x_{20}^3 & x_{20}^4 & x_{20}^5 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{20} \end{bmatrix}$$

Ahora, como tenemos los valores, podemos reemplazar de la siguiente manera:

$$\left[\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 & 32 \\ 1 & 3 & 9 & 27 & 81 & 243 \\ 1 & 4 & 16 & 64 & 256 & 1024 \\ 1 & 5 & 25 & 125 & 625 & 3125 \\ 1 & 6 & 36 & 216 & 1296 & 7776 \\ 1 & 7 & 49 & 343 & 2401 & 16807 \\ 1 & 8 & 64 & 512 & 4096 & 32768 \\ 1 & 9 & 81 & 729 & 6561 & 59049 \\ 1 & 10 & 100 & 1000 & 10000 & 100000 \\ 1 & 11 & 121 & 1331 & 14641 & 161051 \\ 1 & 12 & 144 & 1728 & 20736 & 248832 \\ 1 & 13 & 169 & 2197 & 28561 & 371293 \\ 1 & 14 & 196 & 2744 & 38416 & 537824 \\ 1 & 15 & 225 & 3375 & 50625 & 759375 \\ 1 & 16 & 256 & 4096 & 65536 & 1048576 \\ 1 & 17 & 289 & 4913 & 83521 & 1419857 \\ 1 & 18 & 324 & 5832 & 104976 & 1889568 \\ 1 & 19 & 361 & 6859 & 130321 & 2476099 \\ 1 & 20 & 400 & 8000 & 160000 & 3200000 \end{array} \right] \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{bmatrix} = \begin{bmatrix} 75901 \\ -204794 \\ 204863 \\ -204436 \\ 253665 \\ -200894 \\ 214131 \\ -185192 \\ 221249 \\ -138370 \\ 315911 \\ -27644 \\ 455253 \\ 197434 \\ 783995 \\ 608816 \\ 1370781 \\ 1303798 \\ 2205519 \\ 2408860 \\ 3444321 \end{bmatrix}$$

Así, para nuestro modelo, definiremos las variables A y B :

```
A = 21x6 Matrix{Float64}:
 1.0  0.0  0.0  0.0  0.0  0.0
 1.0  1.0  1.0  1.0  1.0  1.0
 1.0  2.0  4.0  8.0  16.0 32.0
 1.0  3.0  9.0  27.0 81.0 243.0
 1.0  4.0  16.0 64.0 256.0 1024.0
 1.0  5.0  25.0 125.0 625.0 3125.0
 1.0  6.0  36.0 216.0 1296.0 7776.0
 :
 1.0  15.0 225.0 3375.0 50625.0 759375.0
 1.0  16.0 256.0 4096.0 65536.0 1.04858e6
 1.0  17.0 289.0 4913.0 83521.0 1.41986e6
 1.0  18.0 324.0 5832.0 104976.0 1.88957e6
 1.0  19.0 361.0 6859.0 130321.0 2.4761e6
 1.0  20.0 400.0 8000.0 160000.0 3.2e6
```

```
1 A=hcat([DatosX .^ (i-1) for i in 1:6]...)
```

```
B = ▶Float64[
1: 75901.0
2: -204794.0
3: 204863.0
4: -204436.0
5: 253665.0
6: -200894.0
7: 214131.0
8: -185192.0
9: 221249.0
10: -138370.0
11: 315911.0
12: -27644.0
13: 455253.0
14: 197434.0
15: 783995.0
16: 608816.0
17: 1.37078e6
18: 1.3038e6
19: 2.20552e6
20: 2.40886e6
21: 3.44432e6
]
```

1 B=DatosY

Funciones:

Tal como se mencionó anteriormente, se hará uso de ciertas funciones, algunas de las cuales definiremos a continuación:

Nota: **arrAux** es el arreglo de datos de valor 1 usado para el cálculo de Eval.

1 arrAux = fill(1, size(DatosX));

Eval (generic function with 1 method)

```
1 function Eval(C)
2     Eval_Modelo = C[1] * arrAux + C[2] * DatosX + C[3] * DatosX .^ 2 + C[4] * DatosX
        .^ 3 + C[5] * DatosX .^ 4 + C[6] * DatosX .^ 5;
3     return Eval_Modelo
4 end
```

Eval2 (generic function with 1 method)

```
1 function Eval2(A, C)
2     Eval2_Modelo = A * C
3     return Eval2_Modelo
4 end
```

Res (generic function with 1 method)

```
1 function Res(A, C, B)
2     Eval_Modelo = Eval(C)
3     Res_Modelo = Eval_Modelo - B
4     return Res_Modelo
5 end
```

```
NRes (generic function with 1 method)
```

```
1 function NRes(A, C, B)
2     Res_Modelo = Res(A, C, B)
3     NRes_Modelo = norm(Res_Modelo)
4     return NRes_Modelo
5 end
```

Valor objetivo.

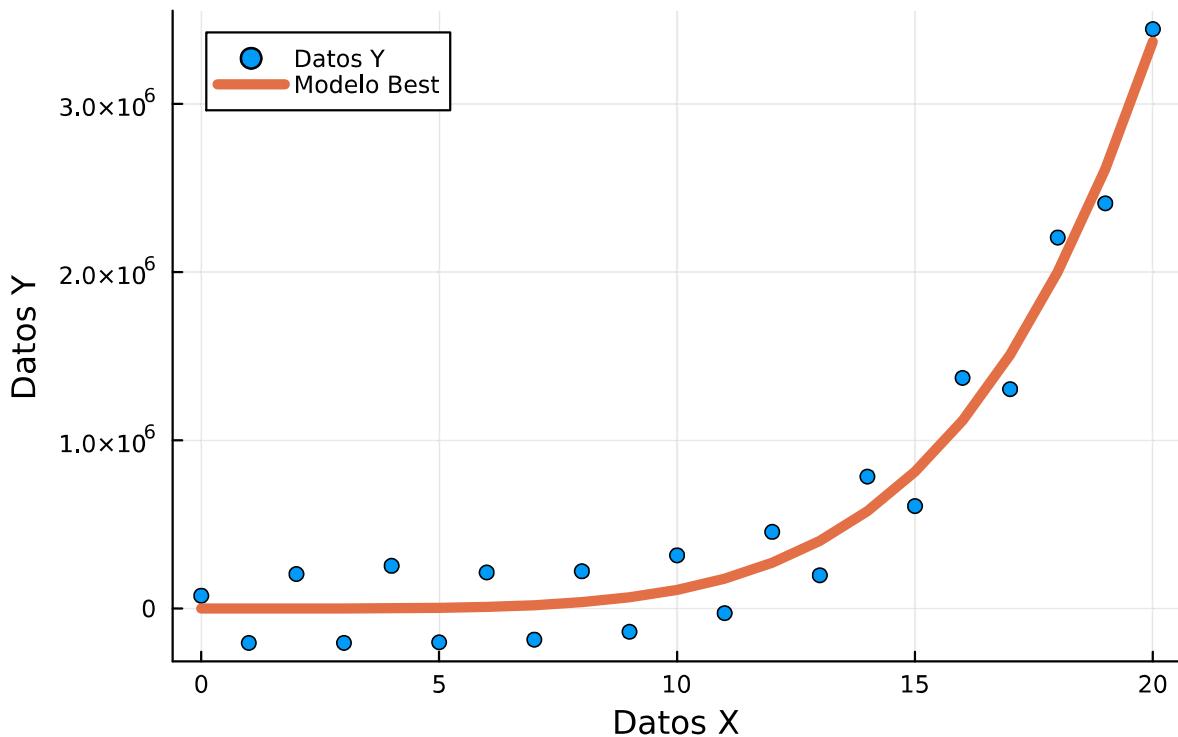
Como se mencionó anteriormente, ya tenemos solución, la cual es:

$$C = [1, 1, 1, 1, 1, 1]$$

Así que compararemos los métodos con esta solución.

```
C_Best = ▶[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
1 C_Best = [1., 1., 1., 1., 1., 1.]
```

Gráfica de datos



```
1 begin
2     Eval_Best = Eval(C_Best)
3     scatter(DatosX, DatosY, ls=:dash, label="Datos Y", lw=4, xlabel="Datos X",
4             ylabel="Datos Y", title="Gráfica de datos")
5     plot!(DatosX, Eval_Best, lw=5, label="Modelo Best")
6
7 # Este código realiza el mapeo de los datos.
```

```

Res_Best = ▶Float64[
 1: -75900.0
 2: 204800.0
 3: -204800.0
 4: 204800.0
 5: -252300.0
 6: 204800.0
 7: -204800.0
 8: 204800.0
 9: -183800.0
10: 204800.0
11: -204800.0
12: 204800.0
13: -183800.0
14: 204800.0
15: -204800.0
16: 204800.0
17: -252300.0
18: 204800.0
19: -204800.0
20: 204800.0
21: -75900.0
]

```

```
1 Res_Best=Res(A, C_Best, B)
```

```
NRes_Best = 914080.2371783344
```

```
1 NRes_Best=NRes(A, C_Best, B)
```

A continuación, construiremos las funciones que nos ayudarán a calcular las diferencias y normas relacionadas con los resultados. Estas funciones incluirán:

- **Dif_C**: diferencia de los valores de C .
- **NDif_C**: norma de esta diferencia.
- **Dif**: diferencia de los residuos.
- **NDif**: norma de la diferencia.

La función **Dif_C** también funcionará como error relativo de cada valor, ya que para cada C_i , el error relativo sería:

$$\frac{|C_i - C_{\text{Modelo}_i}|}{C_i} = |1 - \frac{C_{\text{Modelo}_i}}{1}| = |C_i - C_{\text{Modelo}_i}|$$

```
Dif_C (generic function with 1 method)
```

```
1 function Dif_C(C_Modelo)
2     Dif_Modelo=C_Modelo-C_Best
3     return Dif_Modelo
4 end
```

```
NDif_C (generic function with 1 method)
```

```
1 function NDif_C(C_Modelo)
2     Dif_Modelo = Dif_C(C_Modelo)
3     return norm(Dif_Modelo)
4 end
```

```
Dif (generic function with 1 method)
```

```
1 function Dif(Res_Modelo)
2     return Res_Modelo - Res_Best
3 end
```

```
NDif (generic function with 1 method)
```

```
1 function NDif(Res_Modelo)
2     Res_Modelo = Dif(Res_Modelo)
3     return norm(Res_Modelo)
4 end
```

Aproximación por función de minimización

Para nuestro trabajo revisaremos, además de los métodos dados, la función `optim` incorporada en la librería respectiva de Julia, usando el método numérico integrado de LBFGS.

Para esto, crearemos una función particular para la norma del residuo dada una variable de optimización C_{optim} , y luego buscaremos el argumento que minimice tal función. Esta información se guardará posteriormente en la variable `NRes_Optim_ans`:

```
NRes_Optim_function (generic function with 1 method)
```

```
1 function NRes_Optim_function(C_Optim) #Función de norma a minimizar
2     Eval_Optim=Eval(C_Optim)
3     Res_Optim=B-Eval_Optim
4     return norm(Res_Optim)
5 end
```

```
NRes_Optim_ans = * Status: success
```

```
* Candidate solution
  Final objective value: 9.140802e+05

* Found with
  Algorithm: L-BFGS

* Convergence measures
   $\left| \frac{x - x'}{|x'|} \right| = 0.00e+00 \leq 0.0e+00$ 
   $\left| \frac{f(x) - f(x')}{|f(x')|} \right| = 0.00e+00 \leq 0.0e+00$ 
   $\left| \frac{f(x) - f(x')}{|f(x')|} \right| / |f(x')| = 0.00e+00 \leq 0.0e+00$ 
   $|g(x)| = 1.44e-04 \not\leq 1.0e-08$ 

* Work counters
  Seconds run: 0 (vs limit Inf)
  Iterations: 38
  f(x) calls: 151
   $\nabla f(x)$  calls: 151
```

```
1 NRes_Optim_ans=Optim.optimize(NRes_Optim_function,[.0 ,.0 ,.0 ,.0 ,.0 ,.0], LBFGS())
```

La función `Optim` incorporada nos proporciona tanto el valor del minimizador como el del mínimo, los cuales renombraremos apropiadamente a continuación:

```
C_Optim = ▶ [0.147846, 0.566751, 1.28951, 0.955288, 1.00264, 0.999947]
```

```
1 C_Optim=NRes_Optim_ans.minimizer
```

```
NRes_Optim = 914080.2371804995
```

```
1 NRes_Optim=NRes_Optim_ans.minimum
```

Posteriormente, calcularemos el residuo y las diferencias con nuestro **Best**.

```
Res_Optim = ▶Float64[  
    1: -75900.9  
    2: 204799.0  
    3: -2.04801e5  
    4: 2.04799e5  
    5: -2.523e5  
    6: 2.048e5  
    7: -2.048e5  
    8: 2.048e5  
    9: -1.838e5  
   10: 2.048e5  
   11: -2.048e5  
   12: 2.048e5  
   13: -1.838e5  
   14: 2.048e5  
   15: -2.048e5  
   16: 2.048e5  
   17: -252300.0  
   18: 2.048e5  
   19: -2.048e5  
   20: 2.048e5  
   21: -75900.3  
]
```

```
1 Res_Optim=Res(A, C_Optim, B)
```

```
Dif_C_Optim = ▶[-0.852154, -0.433249, 0.289512, -0.0447118, 0.00263637, -5.33567e-5]
```

```
1 Dif_C_Optim=Dif_C(C_Optim)
```

```
NDif_C_Optim = 0.9998469742710429
```

```
1 NDif_C_Optim=NDif_C(C_Optim)
```

Los valores claramente no son el valor objetivo. La diferencia en norma con nuestro **Best** es de casi **1**, pero esto no nos basta para decir qué tanto se acercan; para esto son nuestras funciones **Dif** y **NDif**.

```
Dif_Optim = ▶Float64[  
    1: -0.852154  
    2: -1.03802  
    3: -0.877824  
    4: -0.552931  
    5: -0.194239  
    6: 0.111419  
    7: 0.314873  
    8: 0.398211  
    9: 0.368375  
   10: 0.250764  
   11: 0.0828258  
   12: -0.0923445  
   13: -0.232406  
   14: -0.302177  
   15: -0.280033  
   16: -0.164318  
   17: 0.0202624  
   18: 0.216231  
   19: 0.326938  
   20: 0.210163  
   21: -0.328297  
]
```

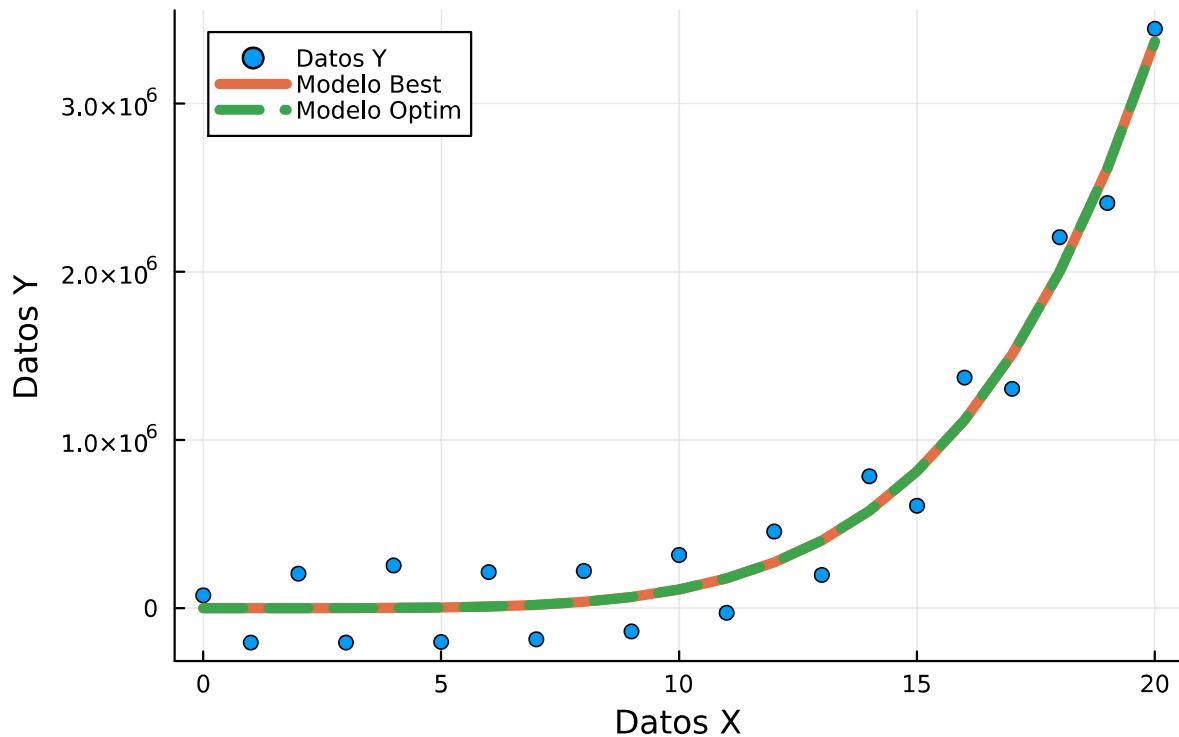
```
1 Dif_Optim=Dif(Res_Optim)
```

```
NDif_Optim = 1.9896064851831006
```

```
1 NDif_Optim=NDif(Res_Optim)
```

Tanto la diferencia como la norma de la diferencia con nuestra función original nos dan valores muy pequeños en comparación con los valores **Best**. El valor en que más se diferencia de este es con una diferencia en valor absoluto de **1.03802**, y la norma de la diferencia es de **1.9896**. Teniendo en cuenta que los valores a los que le estamos haciendo un ajuste de datos rondan los cientos de miles, esto nos da una aproximación bastante acertada (que incluso casi no se ve la diferencia con nuestro **Best**).

Grafica Optim



Aproximación por mínimos cuadrados

El método de mínimos cuadrados consiste en transformar el sistema $AC = B$ a $A^T AC = A^T B$, lo cual resulta en un sistema cuadrado para la resolución. Este enfoque se utiliza para encontrar los coeficientes que mejor ajustan un modelo lineal a los datos observados minimizando la suma de los cuadrados de las diferencias entre los valores observados y los predichos.

```
A_MC = 6x6 Matrix{Float64}:
```

21.0	210.0	2870.0	...	722666.0	1.23333e7
210.0	2870.0	44100.0		1.23333e7	2.16456e8
2870.0	44100.0	722666.0		2.16456e8	3.87729e9
44100.0	722666.0	1.23333e7		3.87729e9	7.05407e10
722666.0	1.23333e7	2.16456e8		7.05407e10	1.29916e12
1.23333e7	2.16456e8	3.87729e9	...	1.29916e12	2.41636e13

```
1 A_MC=A'A #calculo A^{T}A
```

```
B_MC = ▶ [1.31032e7, 2.29559e8, 4.10685e9, 7.46476e10, 1.3738e12, 2.55374e13]
```

```
1 B_MC=A'B
```

Resolución del sistema:

Entonces, ahora necesitamos resolver el sistema $A_{MC}C = B_{MC}$. Este es un sistema de ecuaciones mucho más simple que el que teníamos anteriormente, ya que consiste en **6** ecuaciones y **6** incógnitas. Para resolverlo, usaremos el método para la resolución de sistemas lineales.

```
C_MC = ▶ [1.0, 0.999999, 1.0, 1.0, 1.0, 1.0]
```

```
1 C_MC = A_MC\B_MC
```

```
Res_MC = ▼Float64[
```

1:	-75900.0
2:	204800.0
3:	-204800.0
4:	204800.0
5:	-252300.0
6:	204800.0
7:	-204800.0
8:	204800.0
9:	-183800.0
10:	204800.0
11:	-204800.0
12:	204800.0
13:	-183800.0
14:	204800.0
15:	-204800.0
16:	204800.0
17:	-252300.0
18:	204800.0
19:	-204800.0
20:	204800.0
21:	-75900.0

```
]
```

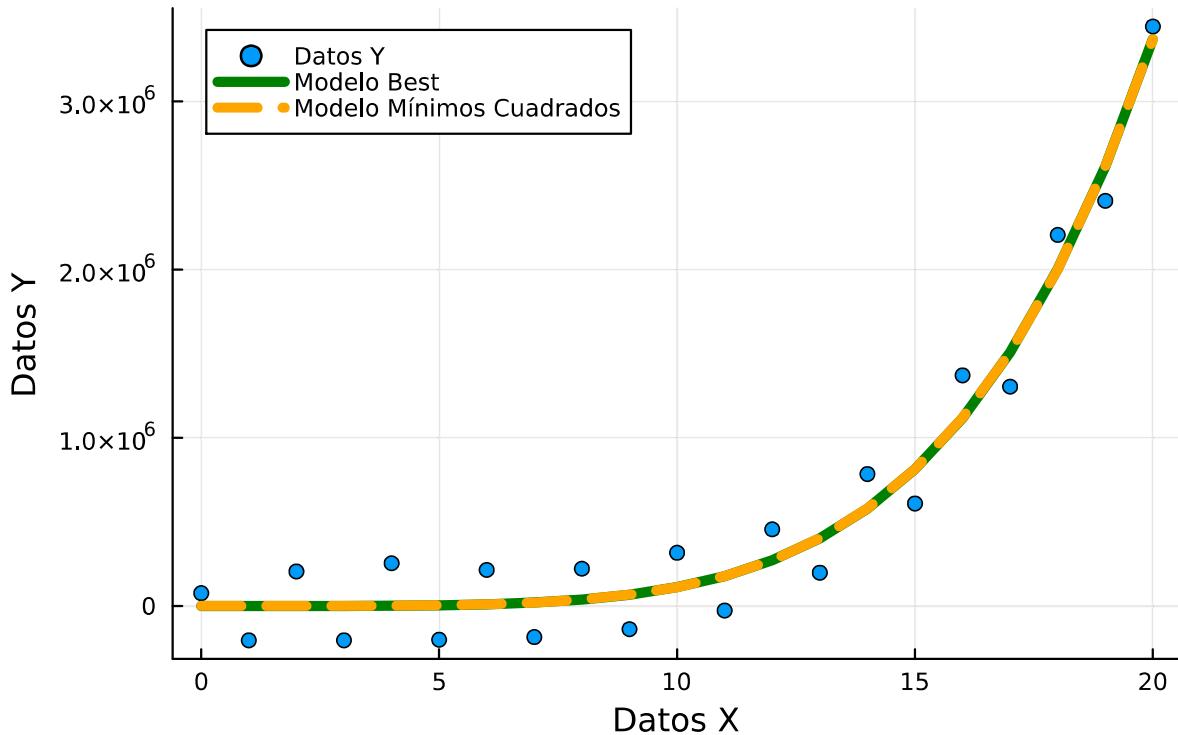
```
1 Res_MC=Res(A, C_MC, B)
```

```
NRes_MC = 914080.2371783345
```

```
1 NRes_MC=NRes(A,C_MC,B)
```

Ya hemos calculado nuestros valores residuales con los datos de ajuste, y ahora vamos a graficar nuestro nuevo modelo sobre el **Best**:

Gráfica MC



```
1 begin
2   Eval_MC = Eval(C_MC)
3   scatter(DatosX, DatosY, ls=:dash, label="Datos Y", lw=4, xlabel="Datos
  X", yaxis="Datos Y", title="Gráfica MC")
4   plot!(DatosX, Eval_Best, lw=5, label="Modelo Best", color=:green)
5   plot!(DatosX, Eval_MC, lw=5, label="Modelo Mínimos Cuadrados", linestyle=:dash,
  color=:orange)
6 end
```

Como se puede ver, la respuesta es casi la misma, con pequeñas diferencias que a continuación se pueden observar, las cuales son menores a 10^{-7} . Además, la diferencia en norma es también muy pequeña.

```
Dif_C_MC = ▶ [2.65094e-7, -5.67032e-7, 2.10268e-7, -2.82258e-8, 1.57264e-9, -3.09489e-11]
```

```
1 Dif_C_MC=Dif_C(C_MC)
```

```
NDif_C_MC = 6.609168330225955e-7
```

```
1 NDif_C_MC=NDif_C(C_MC)
```

En particular, en lo que respecta a la diferencia con los valores de ajuste de datos, contamos con los siguientes valores:

```

Dif_MC = ▶Float64[
1: 2.65092e-7
2: -1.18365e-7
3: -2.29542e-7
4: -1.85828e-7
5: -7.43021e-8
6: 4.45871e-8
7: 1.33237e-7
8: 1.73284e-7
9: 1.61759e-7
10: 1.0748e-7
11: 2.72703e-8
12: -5.77129e-8
13: -1.25729e-7
14: -1.58092e-7
15: -1.42958e-7
16: -7.91624e-8
17: 2.04891e-8
18: 1.2503e-7
19: 1.81841e-7
20: 1.13156e-7
21: -1.88593e-7
]

```

1 Dif_MC=Dif(Res_MC)

```
NDif_MC = 6.584686213081187e-7
```

1 NDif_MC=NDif(Res_MC)

Función residuo

La función residuo que buscamos minimizar es la diferencia entre los valores observados y los valores predichos por el modelo. Se define como:

$$R_{MC}(C_{MC}) = \|B - AC_{MC}\|_2^2$$

donde $R_{MC}(C_{MC})$ es el residuo en función de los coeficientes C_{MC} , siendo B el vector de valores observados y A la matriz de diseño. Al minimizar esta función, encontramos los coeficientes que ajustan mejor el modelo a los datos, minimizando así el error cuadrático. Como se puede ver, esta diferencia es el cuadrado de la norma anterior. Luego, nuestro residuo minimizado asociado al problema de Mínimos Cuadrados es:

```
NR_MC = 8.355426800000002e11
```

1 NR_MC=NRes_MC^2

Solución por método QR

El método QR es una técnica para resolver sistemas de ecuaciones lineales que se basa en la descomposición de una matriz A en el producto de una matriz ortogonal Q y una matriz triangular superior R ($A = QR$). Esta descomposición permite resolver el sistema de manera más numéricamente estable, especialmente en casos donde A es mal condicionada. Al aplicar el método QR, transformamos el problema original en uno más sencillo, lo que facilita la obtención de una solución precisa para los coeficientes del modelo.

De esta manera, pasamos de tener $RC = B$ a $RC = Q^T B$.

Para este ejercicio, usaremos la factorización de QR implementada en Julia a través del comando `qr`.

```
Q = 21x6 Matrix{Float64}:
 -0.218218 -0.360375 -0.422855 0.433298 0.405148 -0.351368
 -0.218218 -0.324337 -0.295999 0.173319 1.63758e-15 0.175684
 -0.218218 -0.2883 -0.182495 -0.0182441 -0.213236 0.314381
 -0.218218 -0.252262 -0.0823455 -0.148994 -0.284314 0.237327
 -0.218218 -0.216225 0.00445111 -0.226531 -0.257137 0.0714339
 -0.218218 -0.180187 0.0778944 -0.258458 -0.169752 -0.0963632
 -0.218218 -0.14415 0.137984 -0.252377 -0.0543542 -0.213395
 :
 -0.218218 0.180187 0.0778944 0.258458 -0.169752 0.0963632
 -0.218218 0.216225 0.00445111 0.226531 -0.257137 -0.0714339
 -0.218218 0.252262 -0.0823455 0.148994 -0.284314 -0.237327
 -0.218218 0.2883 -0.182495 0.0182441 -0.213236 -0.314381
 -0.218218 0.324337 -0.295999 -0.173319 -3.72969e-15 -0.175684
 -0.218218 0.360375 -0.422855 -0.433298 0.405148 0.351368
```

```
1 Q=Matrix(qr(A).Q)
```

```
R = 6x6 Matrix{Float64}:
 -4.58258 -45.8258 -626.285 -9623.41 -1.57699e5 -2.69135e6
 0.0 27.7489 554.977 10150.5 1.84031e5 3.35591e6
 0.0 0.0 -149.775 -4493.26 -1.0388e5 -2.19849e6
 0.0 0.0 0.0 -789.295 -31571.8 -8.84449e5
 0.0 0.0 0.0 0.0 4100.09 2.05005e5
 0.0 0.0 0.0 0.0 0.0 21011.8
```

```
1 R=qr(A).R
```

```
B_QR = ▶ [-2.85935e6, 3.55068e6, -2.30701e6, -9.1681e5, 2.09105e5, 21011.8]
```

```
1 B_QR=Q'B
```

Dadas entonces nuestras matrices R y B_{QR} , resolveremos el sistema $RC = B_{QR}$.

```
C_QR = ▶ [1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

```
1 C_QR=R\B_QR
```

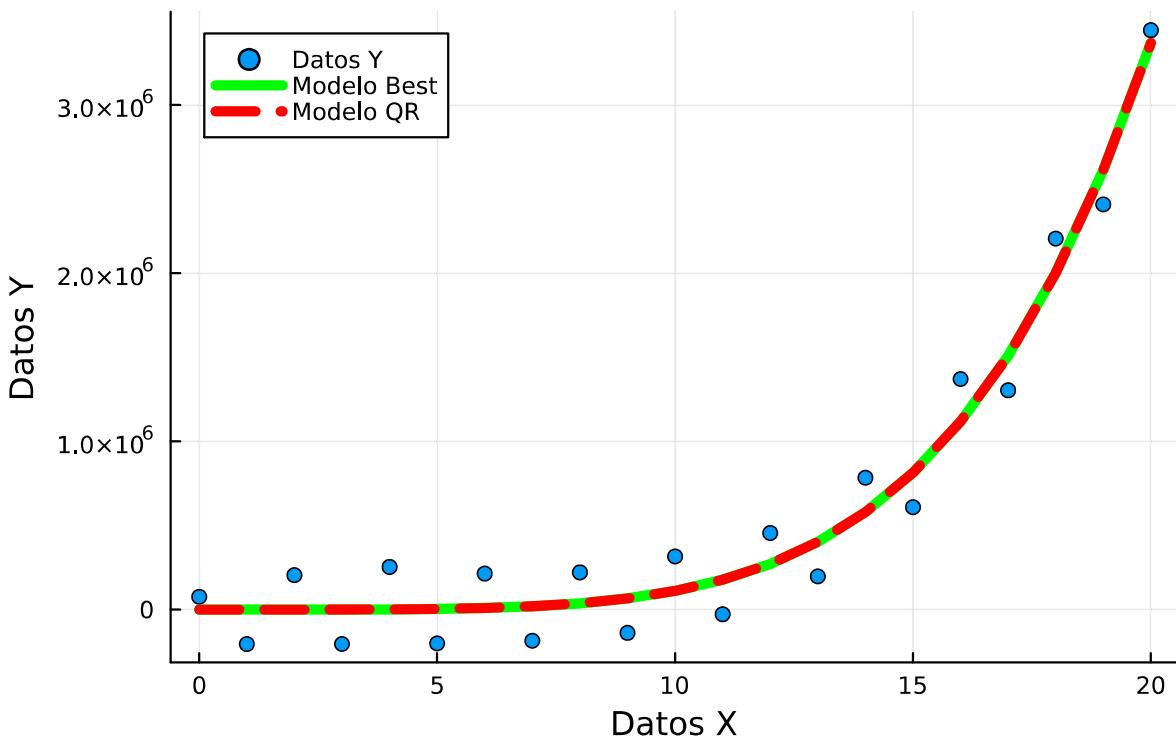
```
Res_QR = ▶Float64[  
    1: -75900.0  
    2: 204800.0  
    3: -204800.0  
    4: 204800.0  
    5: -252300.0  
    6: 204800.0  
    7: -204800.0  
    8: 204800.0  
    9: -183800.0  
   10: 204800.0  
   11: -204800.0  
   12: 204800.0  
   13: -183800.0  
   14: 204800.0  
   15: -204800.0  
   16: 204800.0  
   17: -252300.0  
   18: 204800.0  
   19: -204800.0  
   20: 204800.0  
   21: -75900.0  
]
```

```
1 Res_QR=Res(A, C_QR, B)
```

```
NRes_QR = 914080.2371783344
```

```
1 NRes_QR=NRes(A, C_QR, B)
```

Gráfica QR



```
1 begin
2     Eval_QR = Eval(C_QR)
3     scatter(DatosX, DatosY, ls=:dash, label="Datos Y", lw=4, xlabel="Datos
4     X", ylabel="Datos Y", title="Gráfica QR")
5     plot!(DatosX, Eval_Best, lw=5, label="Modelo Best", color=:lime)
6     plot!(DatosX, Eval_QR, lw=5, label="Modelo QR", linestyle=:dash, color=:red)
7 end
```

Nuestro ajuste de datos se acerca bastante, con una diferencia menor a 10^{-8} en el peor de los casos:

```
Dif_QR = ▶Float64[
    1: 5.47152e-9
    2: -3.52156e-9
    3: -5.67525e-9
    4: -4.10364e-9
    5: -9.8953e-10
    6: 2.06637e-9
    7: 4.19095e-9
    8: 4.94765e-9
    9: 4.30737e-9
    10: 2.56114e-9
    11: 2.32831e-10
    12: -2.12458e-9
    13: -3.89991e-9
    14: -4.5402e-9
    15: -3.84171e-9
    16: -1.86265e-9
    17: 1.16415e-9
    18: 4.19095e-9
    19: 5.58794e-9
    20: 3.25963e-9
    21: -6.0536e-9
]
```

```
1 Dif_QR=Dif(Res_QR)
```

```
NDif_QR = 1.7883705862852222e-8
```

```
1 NDif_QR=NDif(Res_QR)
```

Específicamente, nuestro error relativo, componente a componente, se puede ver a continuación:

```
Dif_C_QR = ► [5.4763e-9, -1.36445e-8, 5.35541e-9, -7.42992e-10, 4.23328e-11, -8.46878e-13]
```

```
1 Dif_C_QR=Dif_C(C_QR)
```

```
NDif_C_QR = 1.5665152939023555e-8
```

```
1 NDif_C_QR=NDif_C(C_QR)
```