Actividad realizada por Alan Acero, Johan López y Nicolás Duque

Se utilizarán las siguientes librerías:

```
md"""
Actividad realizada por Alan Acero, Johan López y Nicolás Duque

Se utilizarán las siguientes librerías:
"""
```

```
using Plots, Dates, LinearAlgebra, Optim, DifferentialEquations;
```

Actividad Ajuste de Datos

En esta actividad de ajuste de datos, vamos a analizar la ocupación de las camas UCI en Bogotá durante el inicio de 2022, relacionándola también con el uso por el COVID-19. El objetivo de este trabajo es hacer uso de esa información para explorar y aprender acerca de distintos enfoques para el ajuste de datos. Tomaremos, entonces, diversas funciones con varias variables para el ajuste de los datos.

El ajuste de datos, también llamado ajuste de curvas, hace referencia a, dado un conjunto de datos, buscar alguna correlación entre ellos; es decir, intentar encontrar una relación que describa de manera cercana cómo se comportan los datos en relación entre sí.

Para este tipo de problemas, lo que se hace es usar un algoritmo de optimización y elegir las variables que nos den un mínimo residuo al usar este algoritmo. El algoritmo que usaremos en cada caso será implementado por la librería *Optim* [1], y nuestra función a minimizar será la de residuo, que en nuestro caso será por mínimos cuadrados (con la norma euclidiana o la norma base 2).

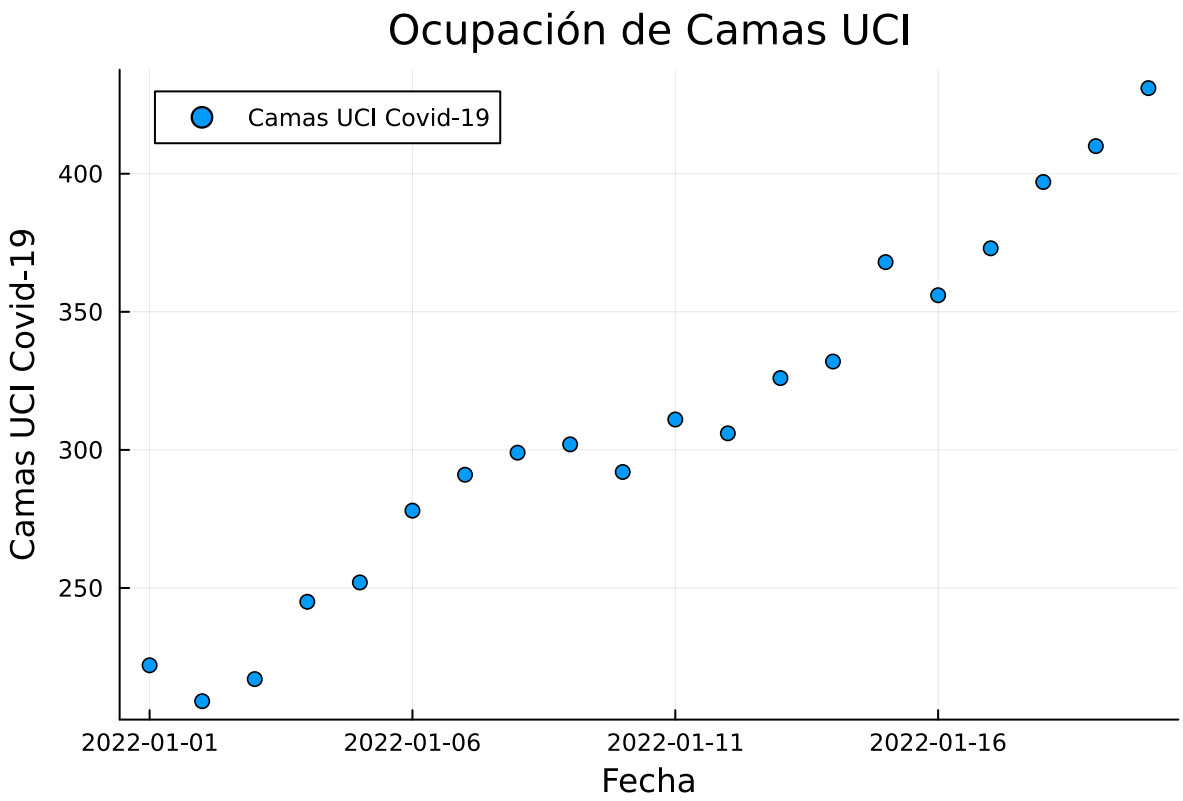
Este trabajo se basa en el cuaderno "Análisis numérico - Ajuste de curvas" del Laboratorio Virtual de Matemáticas de la Universidad Nacional de Colombia, Sede Bogotá [2].

```
20×2 Matrix{Any}:
 2022-01-01  222
 2022-01-02  209
 2022-01-03  217
 2022-01-04  245
 2022-01-05  252
 2022-01-06  278
 2022-01-07  291
 ⋮
 2022-01-15  368
 2022-01-16  356
 2022-01-17  373
 2022-01-18  397
 2022-01-19  410
 2022-01-20  431
```

```julia
begin
    df = DateFormat("d/m/y");

    Datos = [
        Date("1/01/2022", df)   222
        Date("2/01/2022", df)   209
        Date("3/01/2022", df)   217
        Date("4/01/2022", df)   245
        Date("5/01/2022", df)   252
        Date("6/01/2022", df)   278
        Date("7/01/2022", df)   291
        Date("8/01/2022", df)   299
        Date("9/01/2022", df)   302
        Date("10/01/2022", df)  292
        Date("11/01/2022", df)  311
        Date("12/01/2022", df)  306
        Date("13/01/2022", df)  326
        Date("14/01/2022", df)  332
        Date("15/01/2022", df)  368
        Date("16/01/2022", df)  356
        Date("17/01/2022", df)  373
        Date("18/01/2022", df)  397
        Date("19/01/2022", df)  410
        Date("20/01/2022", df)  431
    ];

    fechas = Datos[:,1];
    dias = collect(1:size(fechas, 1));
    camas = Datos[:,2]
    arrAux = fill(1, size(dias));
    Datos
    # Dates.value(fechas[1])
end
```




```
scatter(fechas,camas,ls=:dash,label="Camas UCI Covid-19",lw=4, xlabel =
"Fecha",yaxis="Camas UCI Covid-19", title="Ocupación de Camas UCI")
```

Para ilustrar el procedimiento, ajustaremos los datos a los siguientes modelos:

1. Un modelo polinomial de grado uno.
2. Un modelo polinomial cúbico.
3. Un modelo de redes neuronales artificiales.
4. Algunos modelos no lineales.
5. Un modelo de ecuaciones diferenciales.

Modelos no basados en Ecuaciones Diferenciales

Modelo Lineal

Empezaremos con el modelo lineal, que asumiremos tiene la siguiente forma:

$$V(t) \approx a + bt$$

Y entonces buscamos encontrar los parámetros $a, b \in \mathbb{R}$ tal que minimicen nuestro residuo. Para calcular el residuo, creamos la siguiente función, donde nuestra entrada será *tuplaC*, una tupla que representa nuestro valor inicial para la optimización, *vDatos*, que será el vector de datos de las camas, y *tiempo*, que corresponderá a nuestros días.

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

```
function residuoLineal(tuplaC, vDatos, tiempo)
    a,b = tuplaC;
    arrAux = fill(1, size(tiempo));
    vModelo = a * arrAux + b * tiempo;
    res=vDatos-vModelo
    nRes=norm(res)

    return nRes
end
```

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

```
rL(tuplaC) = residuoLineal(tuplaC, camas, dias)
```

```
oL =  * Status: success

      * Candidate solution
         Final objective value:     5.577769e+01

      * Found with
         Algorithm:     L-BFGS

      * Convergence measures
         |x - x'|               = 3.57e-05 ≰ 0.0e+00
         |x - x'|/|x'|          = 1.79e-07 ≰ 0.0e+00
         |f(x) - f(x')|         = 1.25e-09 ≰ 0.0e+00
         |f(x) - f(x')|/|f(x')| = 2.24e-11 ≰ 0.0e+00
         |g(x)|                 = 2.20e-10 ≤ 1.0e-08

      * Work counters
         Seconds run:   0  (vs limit Inf)
         Iterations:    9
         f(x) calls:    26
         ∇f(x) calls:   26
```

```
1 oL =Optim.optimize(rL, [3.0, 5.0], LBFGS())
```

▸ [199.068, 10.6459]

```
1 oL.minimizer
```

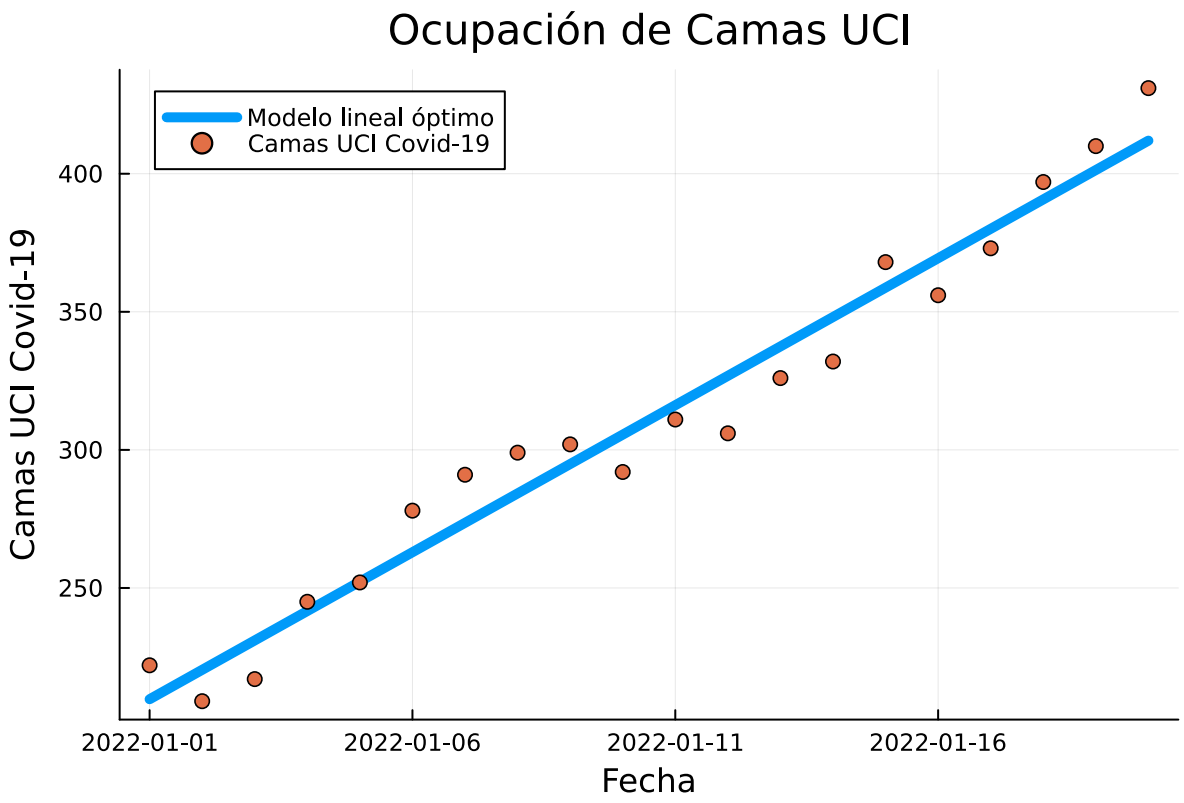
55.777693819479026

```
1 oL.minimum
```

Nuestra función que minimiza el residuo bajo estas condiciones es ent

$$V(t) \approx 199.068 + 10.646t$$

con un residuo aproximado de 55.778




```
begin
    vModeloL = oL.minimizer[1] * arrAux + oL.minimizer[2] * dias;
    plot(fechas, vModeloL, lw=5, label="Modelo lineal óptimo");
    scatter!(fechas, camas, ls=:dash,label="Camas UCI Covid-19",lw=4, xlabel =
    "Fecha",yaxis="Camas UCI Covid-19", title="Ocupación de Camas UCI")

end
```

Modelo Cúbico

Para el modelo cúbico, buscaremos los parámetros $a, b, c, d \in \mathbb{R}$ que ajusten el siguiente modelo:

$$V(t) \approx a + bt + ct^2 + dt^3$$

A continuación, programamos la función de residuo utilizando la norma euclidiana.

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

```
function residuoCubico(tuplaC, vDatos, tiempo)
    a,b,c,d = tuplaC;
    arrAux = fill(1, size(tiempo));
    vModelo = a * arrAux + b * tiempo + c * tiempo .^ 2 + d * tiempo .^ 3;
    res=vDatos-vModelo
    nRes=norm(res)

    return nRes
end
```

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

```
rCub(tuplaC) = residuoCubico(tuplaC, camas, dias)
```

```
oCub =  * Status: success (objective increased between iterations)

        * Candidate solution
           Final objective value:     4.501428e+01

        * Found with
           Algorithm:     L-BFGS

        * Convergence measures
           |x - x'|               = 5.34e-10 ≰ 0.0e+00
           |x - x'|/|x'|          = 2.88e-12 ≰ 0.0e+00
           |f(x) - f(x')|         = 2.13e-14 ≰ 0.0e+00
           |f(x) - f(x')|/|f(x')| = 4.74e-16 ≰ 0.0e+00
           |g(x)|                 = 1.76e-09 ≤ 1.0e-08

        * Work counters
           Seconds run:   0  (vs limit Inf)
           Iterations:    22
           f(x) calls:    78
           ∇f(x) calls:   78
```

```
oCub =Optim.optimize(rCub, [0.1, 0.1, 0.1, 0.1], LBFGS())
```

```
▸ [185.333, 19.8622, -1.24469, 0.0433457]
```

```
oCub.minimizer
```

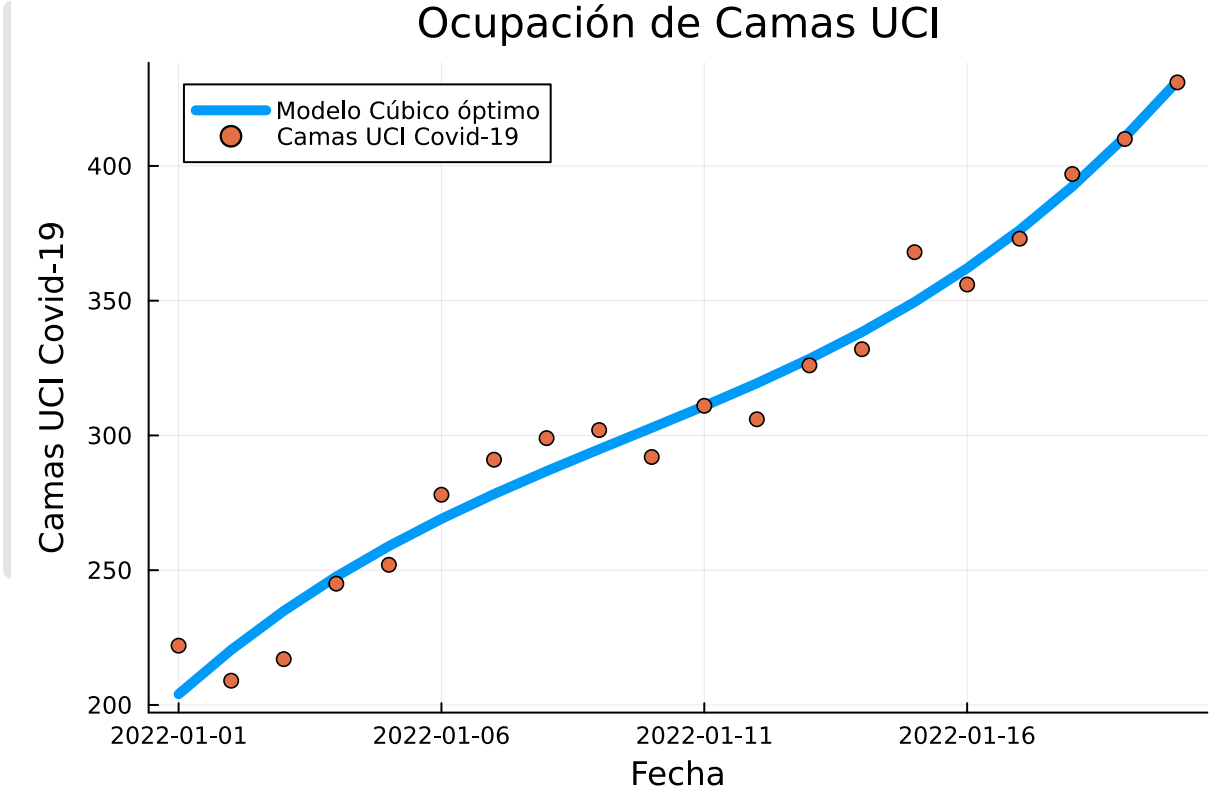
```
45.01428140986388
```

```
oCub.minimum
```

Entonces, nuestra función optima segun el algoritmo usado seria:

$$V(t) \approx 185.333 + 19.862t + -1.245t^2 + 0.043t^3$$

con un residuo aproximado de 45.014




```
begin
    vModeloCub = oCub.minimizer[1] * arrAux + oCub.minimizer[2] * dias +
    oCub.minimizer[3] * dias .^ 2 + oCub.minimizer[4] * dias .^ 3;
    plot(fechas, vModeloCub, lw=5, label="Modelo Cúbico óptimo");
    scatter!(fechas, camas, ls=:dash,label="Camas UCI Covid-19",lw=4, xlabel =
    "Fecha",yaxis="Camas UCI Covid-19", title="Ocupación de Camas UCI")

end
```

Modelo de Redes Neuronales Artificiales

Ahora continuamos con el modelo de redes neuronales artificiales. Aquí buscaremos valores a, b, c, d, f, $g \in \mathbb{R}$ tal que nos aproxime la función:

$$V(t) \approx a\frac{1}{1+e^{bt+c}} + d\frac{1}{1+e^{ft+g}}$$

Entonces, proseguimos construyendo la función a minimizar.

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

```
function modeloRNA(tuplaC, vDatos, tiempo)
    a,b,c,d,f,g = tuplaC;
    arrAux = fill(1, size(tiempo));
    vModelo=a *( arrAux./(arrAux+exp.( b*tiempo + c*arrAux )))+ d *( arrAux./
    (arrAux+exp.( f*tiempo + g*arrAux )))
    res=vDatos-vModelo
    nRes=norm(res)

    return nRes

end
```

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

```
rRNA(tuplaC) = modeloRNA(tuplaC, camas, dias)
```

```
oRNA =  * Status: failure (reached maximum number of iterations)

        * Candidate solution
           Final objective value:     5.306245e+01

        * Found with
           Algorithm:     L-BFGS

        * Convergence measures
           |x - x'|               = 1.48e+02 ≰ 0.0e+00
           |x - x'|/|x'|          = 1.75e-03 ≰ 0.0e+00
           |f(x) - f(x')|         = 6.01e-06 ≰ 0.0e+00
           |f(x) - f(x')|/|f(x')| = 1.13e-07 ≰ 0.0e+00
           |g(x)|                 = 4.57e+00 ≰ 1.0e-08

        * Work counters
           Seconds run:   0  (vs limit Inf)
           Iterations:    1000
           f(x) calls:    2555
           ∇f(x) calls:   2555
```

```
oRNA=Optim.optimize(rRNA, [.1,.001,.001,.001,.001,.001], LBFGS())
```

```
▸ [-94.0642, -24740.6, -2125.72, 84746.2, -0.0265342, 5.62914]
```

```
oRNA.minimizer
```

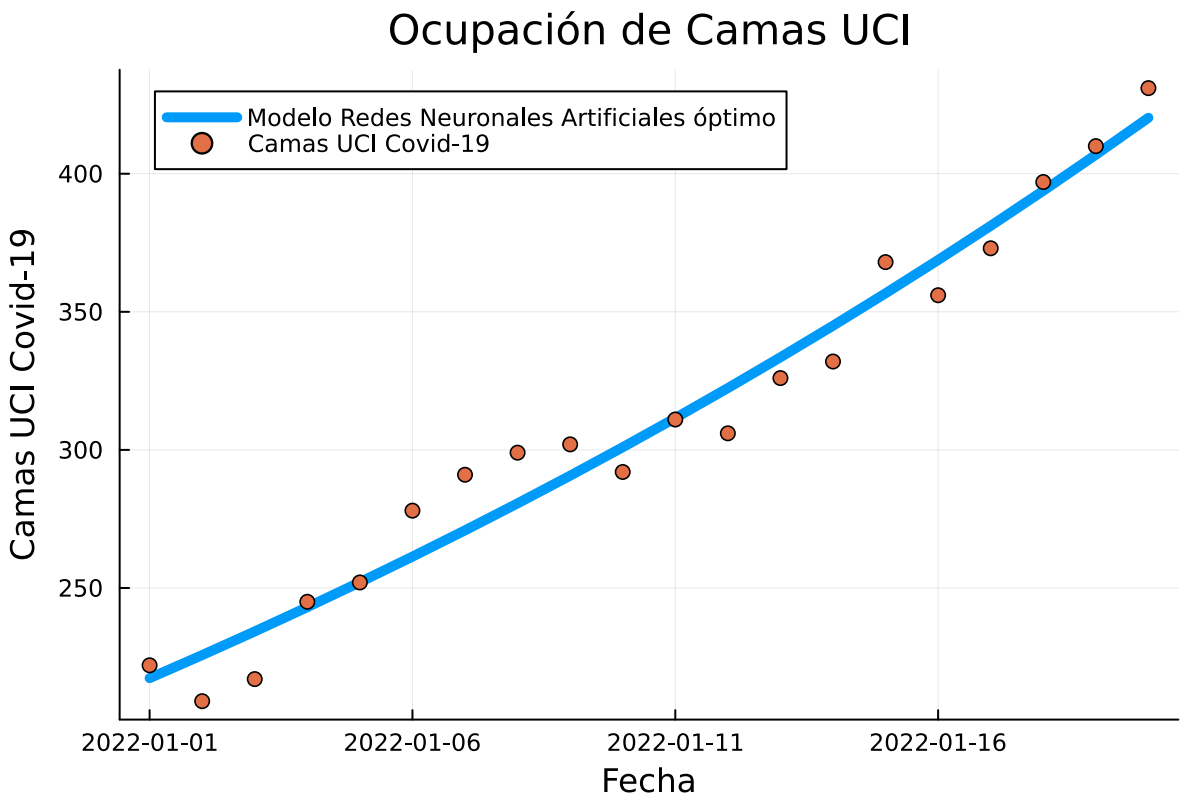
```
53.06245026851765
```

```
oRNA.minimum
```

La función que minimiza el residuo con las Redes neuronales artificiales seria aproximadamente la siguiente

$$V(t) \approx -94.064\frac{1}{1+e^{-24740.6t+-2125.72}} + 84746.2\frac{1}{1+e^{-0.027t+5.629}}$$

Y tiene un residuo aproximado de 53.062



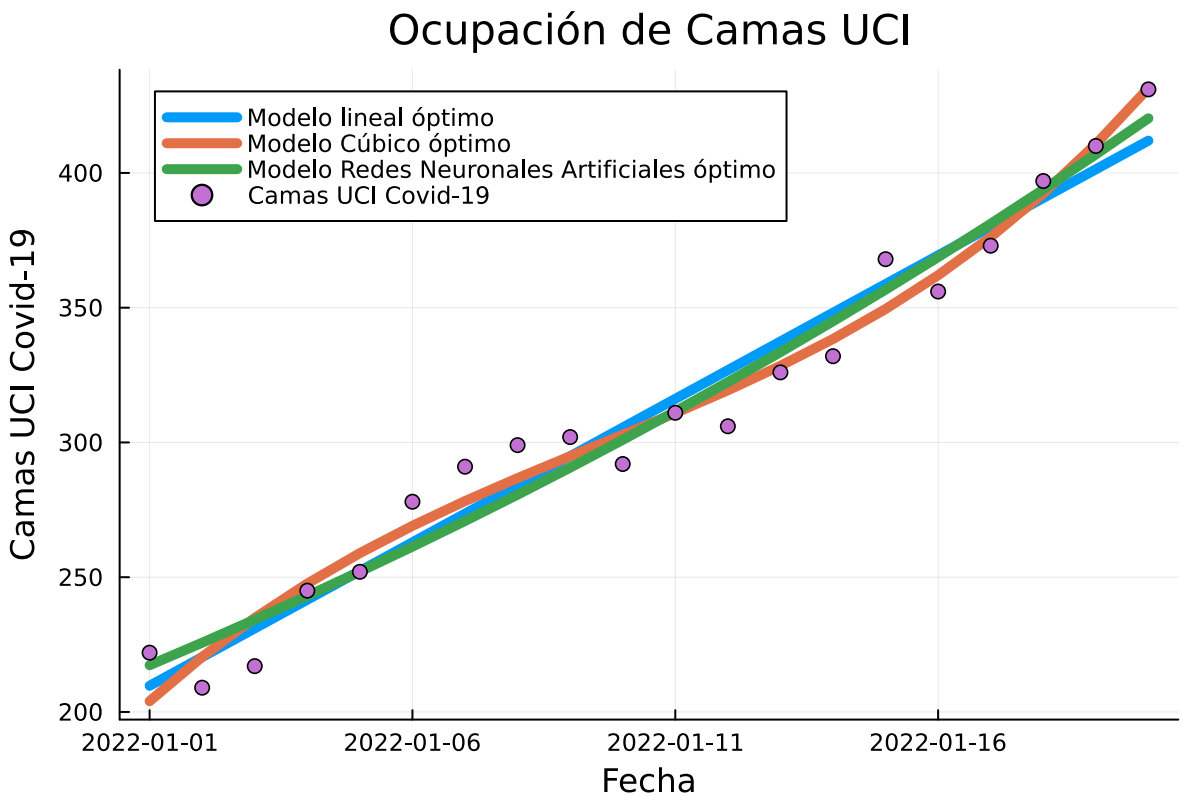

```
begin
    vModeloRNA = oRNA.minimizer[1] *
    ( arrAux./(arrAux+exp.( oRNA.minimizer[2]*dias + oRNA.minimizer[3]*arrAux ))) +
    oRNA.minimizer[4] *( arrAux./(arrAux+exp.( oRNA.minimizer[5]*dias +
    oRNA.minimizer[6]*arrAux )))

    plot(fechas, vModeloRNA, lw=5, label="Modelo Redes Neuronales Artificiales
    óptimo");
    scatter!(fechas, camas, ls=:dash,label="Camas UCI Covid-19",lw=4, xlabel =
    "Fecha",yaxis="Camas UCI Covid-19", title="Ocupación de Camas UCI")

end
```

Análisis de los Modelos Presentados

Dentro de lo revisado entre estos tres modelos, habíamos supuesto que el que quedaría con menor residuo sería el de redes neuronales, pero como se puede ver, en realidad fue el cúbico, lo que también da a entender que el modo en el que se desarrollan los datos y su contexto pueden darnos pistas sobre qué tipo de modelos usar. A continuación se presenta una tabla con los tres modelos.



Otros modelados

Con el fin de analizar modelos no lineales y tener una idea más amplia del comportamiento de cada modealmiento. Analizaremos la aproximación que nos genera cada una y las ilustraremos. De esta manera, tengamos en cuenta los siguientes modelamientos:

1. Hiperbólico Ajustado

Sean $a, b \in \mathbb{R}$ los parámetros a ser estimados y optimizados, así, asumamos nuestro modelo de la siguiente forma:

$$V(t) \approx \frac{a}{t} + b$$

Para lograrlo seguiremos usando la metodología presentada en los métodos lineales.

```
function residuoNoLinealOne(tuplaC, vDatos, tiempo)
    a,b = tuplaC
    arrAux = fill(1, size(tiempo))
    vModelo = a*(arrAux./(tiempo))+b*arrAux
    res = vDatos-vModelo
    nRes = norm(res)

    return nRes
end;

```

A partir de nuestra función residuo, podremos aproximar los mejores valores que puedan tener a y b para obtener un gran ajuste.

Sin embargo, es importante destacar que este modelo nos genera un desajuste significativo. Por ejemplo, si $a = 5$ y $b = 10$, esto es $V(t) \approx \frac{5}{t} + 10$, este es aproximadamente 1371.037.

```
1371.0371118430592
```

```
residuoNoLinealOne([5,10], camas, dias)
```

De la misma forma, definimos una función que depende exclusivamente de la variable de decisión de la optimización.

```
rNLO(tuplaC) = residuoNoLinealOne(tuplaC,camas,dias);
```

```
oNLO =  * Status: success

        * Candidate solution
           Final objective value:     2.074795e+02

        * Found with
           Algorithm:     L-BFGS

        * Convergence measures
           |x - x'|               = 6.91e-03 ≰ 0.0e+00
           |x - x'|/|x'|          = 2.00e-05 ≰ 0.0e+00
           |f(x) - f(x')|         = 6.69e-07 ≰ 0.0e+00
           |f(x) - f(x')|/|f(x')| = 3.22e-09 ≰ 0.0e+00
           |g(x)|                 = 4.01e-10 ≤ 1.0e-08

        * Work counters
           Seconds run:   0  (vs limit Inf)
           Iterations:    7
           f(x) calls:    21
           ∇f(x) calls:   21
```

```
oNLO = Optim.optimize(rNLO, [3.0, 5.0], LBFGS())
```

Finalmente, optimizando los parámetros:

```
▸ [-193.225, 345.609]
```

```
oNLO.minimizer
```

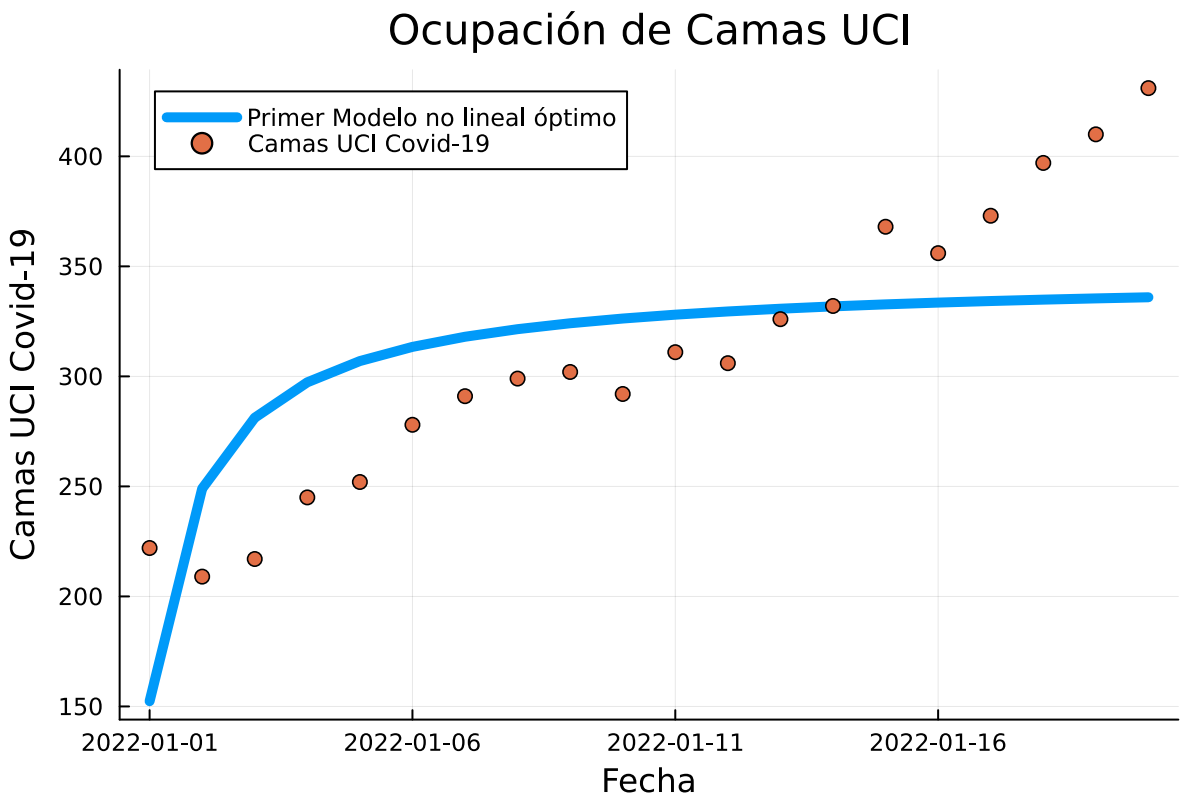
```
207.47951054782357
```

```
oNLO.minimum
```

Lo que nos permite saber que el mejor modelo sería:

$$V(t) \approx \frac{-193.225}{t} + 345.609$$

Al igual, visualizar nuestra función aproximada con los datos obtenidos:



Notemos que los valores que fueron asignados a los parámetros a y b no contienen ningún valor cualitativo, por lo que no nos permiten predecir ni estimar.

2. Hiperbólico con desplazamiento

Sean $d, c \in \mathbb{R}$ los parámetros a ser optimizados en nuestro modelo:

$$V(t) \approx \frac{d}{t+c}$$

Ahora, formemos nuestra función residuo que nos permitirá estimar los mejores valores para los parámetros c y d.

```
function residuoNoLinealTwo(tuplaC, vDatos, tiempo)
    d,c = tuplaC
    arrAux = fill(1, size(dias))
    vModelo = d*(arrAux./(tiempo+(arrAux*c)))
    res = vDatos-vModelo
    nRes = norm(res)

    return nRes
end;
```

Luego, procedemos con la mismas ideas anteriores:

```
md"""Luego, procedemos con la mismas ideas anteriores:"""
```

```
rNLT(tuplaC) = residuoNoLinealTwo(tuplaC,camas,dias);
```

```
oNLT =  * Status: failure (line search failed)

        * Candidate solution
           Final objective value:     2.808747e+02

        * Found with
           Algorithm:     L-BFGS

        * Convergence measures
           |x - x'|               = 0.00e+00 ≤ 0.0e+00
           |x - x'|/|x'|          = 0.00e+00 ≤ 0.0e+00
           |f(x) - f(x')|         = 3.94e+01 ≰ 0.0e+00
           |f(x) - f(x')|/|f(x')| = 1.40e-01 ≰ 0.0e+00
           |g(x)|                 = 1.85e-33 ≤ 1.0e-08

        * Work counters
           Seconds run:   0  (vs limit Inf)
           Iterations:    5
           f(x) calls:    69
           ∇f(x) calls:   69
```

```
1 oNLT = Optim.optimize(rNLT, [.01, 100.], LBFGS())
```

▸ [60340.8, 184.948]

```
1 oNLT.minimizer
```

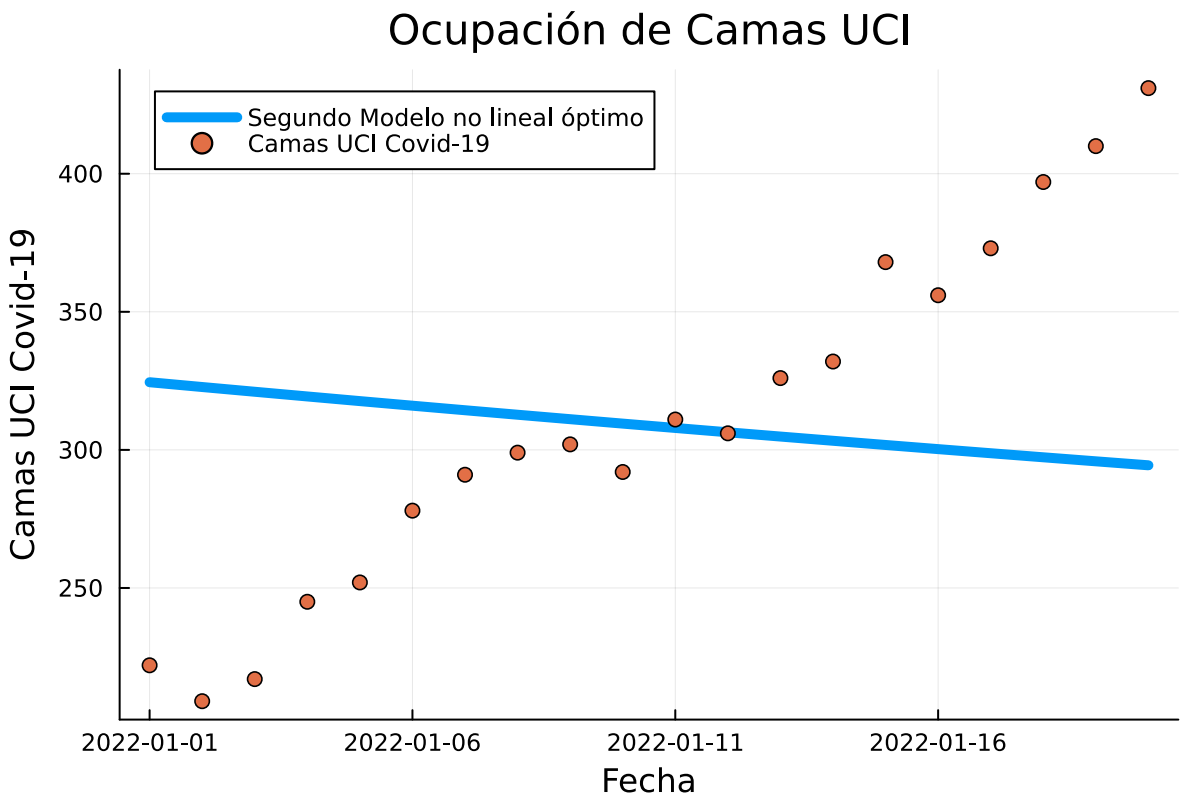
280.8747068529022

```
1 oNLT.minimum
```

De igual manera que nuestro método hiperbólico ajustado, encontramos que este método tiene un desajuste grande, hasta más que el anterior. Por lo tanto, tenemos

$$V(t) \approx \frac{60340.8}{t + 184.948}$$

con su respectiva gráfica:



3. Exponencial

Asumamos que nuestro método tiene siguiente forma:

$$V(t) \approx ae^{bt}$$

con $a, b \in \mathbb{R}$ los parámetros que deben ser optimizados.

Un aspecto interesante de este enfoque es su relación con las ecuaciones diferenciales. En particular, cuando resolvemos una ecuación diferencial lineal homogénea de segundo orden, es común suponer que la solución adopta esta forma exponencial.

Creamos nuestra función residuo:

```
function residuoNoLinealThree(tuplaC, vDatos, tiempo)
    a,b = tuplaC
    arrAux = fill(1, size(dias))
    vModelo = a*(exp.(b*tiempo))
    res = vDatos-vModelo
    nRes = norm(res)

    return nRes
end;
```

Ahora, formamos una función que solo dependa de los parámetros a optimizar y nos ayudamos de la libreria *Optim*:

```
rNLTh(tuplaC) = residuoNoLinealThree(tuplaC,camas,dias);
```

```
oNLTh =  * Status: success (objective increased between iterations)

         * Candidate solution
            Final objective value:     5.332285e+01

         * Found with
            Algorithm:     BFGS

         * Convergence measures
            |x - x'|               = 6.41e-08 ≰ 0.0e+00
            |x - x'|/|x'|          = 3.02e-10 ≰ 0.0e+00
            |f(x) - f(x')|         = 1.14e-13 ≰ 0.0e+00
            |f(x) - f(x')|/|f(x')| = 2.13e-15 ≰ 0.0e+00
            |g(x)|                 = 7.04e-09 ≤ 1.0e-08

         * Work counters
            Seconds run:   0  (vs limit Inf)
            Iterations:    22
            f(x) calls:    87
            ∇f(x) calls:   87
```

```
oNLTh = Optim.optimize(rNLTh, [.1, .01], BFGS())
```

```
▸ [212.45, 0.0343887]
```

```
oNLTh.minimizer
```

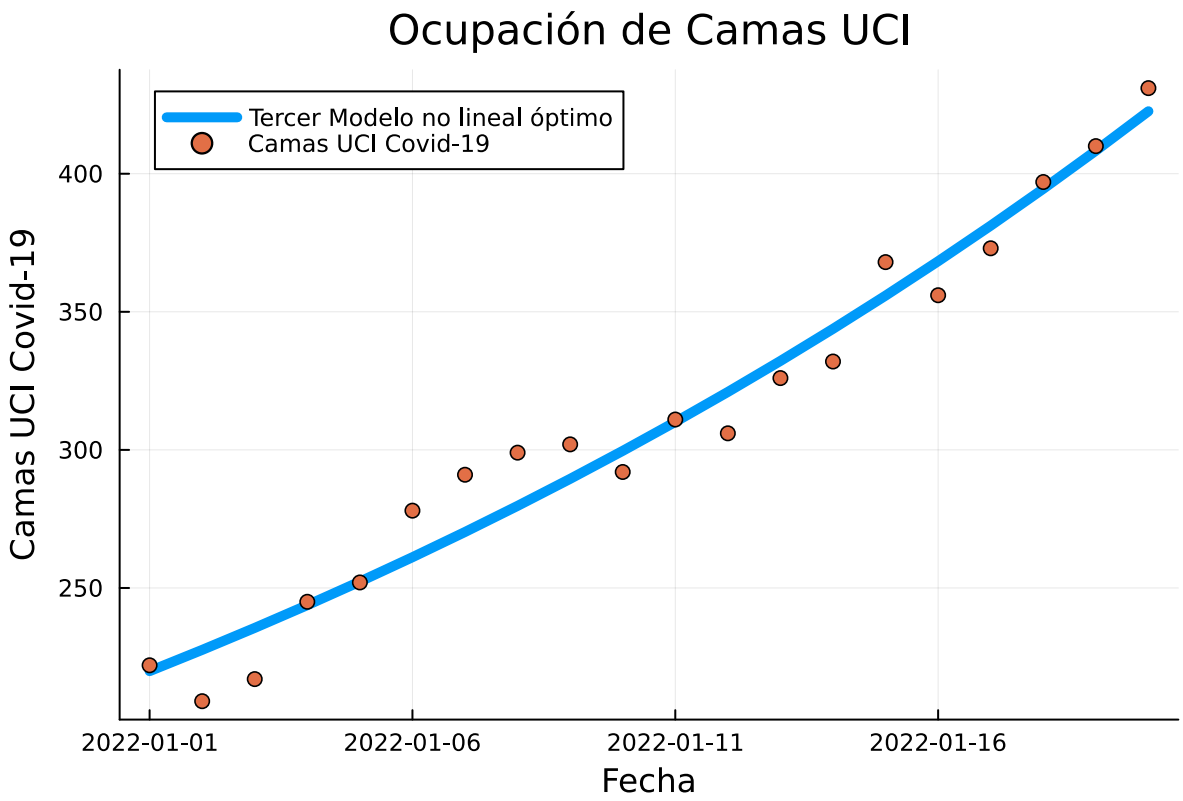
```
53.32285308928369
```

```
oNLTh.minimum
```

De modo similar, obtenemos los valores de nuestros parametros con un residuo muchísimo menor a los anteriores métodos y con una ilustración más cercana a los datos obtenidos. Esto es

$$V(t) \approx 212.45e^{0.034}$$

con su respectiva gráfica.



Es importante destacar cómo el mejoramiento en la aproximación de nuestro método exponencial no solo optimiza el ajuste, sino que también aporta un significado cualitativo a los parámetros a y b. Esto permite obtener estimaciones y predicciones más precisas, ofreciendo herramientas útiles para profesionales de la salud. En este contexto:

1. Donde a representa el valor inicial del estudio.
2. Donde b describe la tasa de crecimiento relativo, indicando la rapidez con la que cambia la variable en función del tiempo.

Estos parámetros no solo mejoran el ajuste del modelo, sino que también contribuyen a una interpretación más intuitiva y práctica en aplicaciones del mundo real.

4. Logístico

En este modelo asumimos:

$$V(t) \approx \frac{a}{1 + be^{cx}}$$

con $a, b \in \mathbb{R}$ los parámetros a ser optmizados.

Definimos una función para calcular la magnitud del residuo, que mide el grado de desajuste entre el modelo y los datos observados. Este cálculo se realiza utilizando la norma euclidiana, aplicada a la diferencia (o residuo) entre los valores predichos por el modelo y los valores reales de los datos. Así:

```
function residuoNoLinealFour(tuplaC, vDatos, tiempo)
    a,b,c = tuplaC
    arrAux = fill(1, size(dias))
    vModelo = (a*arrAux)./(arrAux+b*exp.(c*tiempo))
    res = vDatos-vModelo
    nRes = norm(res)

    return nRes
end;
```

Así, tomamos una función que tome como valor principal un vector con los parametros para luego ser optimizado.

```
rNLF(tuplaC) = residuoNoLinealFour(tuplaC,camas,dias);
```

```
oNLF =  * Status: success

        * Candidate solution
           Final objective value:     5.329519e+01

        * Found with
           Algorithm:     BFGS

        * Convergence measures
           |x - x'|               = 0.00e+00 ≤ 0.0e+00
           |x - x'|/|x'|          = 0.00e+00 ≤ 0.0e+00
           |f(x) - f(x')|         = 0.00e+00 ≤ 0.0e+00
           |f(x) - f(x')|/|f(x')| = 0.00e+00 ≤ 0.0e+00
           |g(x)|                 = 1.88e-07 ≰ 1.0e-08

        * Work counters
           Seconds run:   0  (vs limit Inf)
           Iterations:    112
           f(x) calls:    365
           ∇f(x) calls:   365
```

```
oNLF = Optim.optimize(rNLF, [.01,.01,.01], BFGS())
```

```
▸ [4389.64, 19.7466, -0.0370714]
```

```
oNLF.minimizer
```

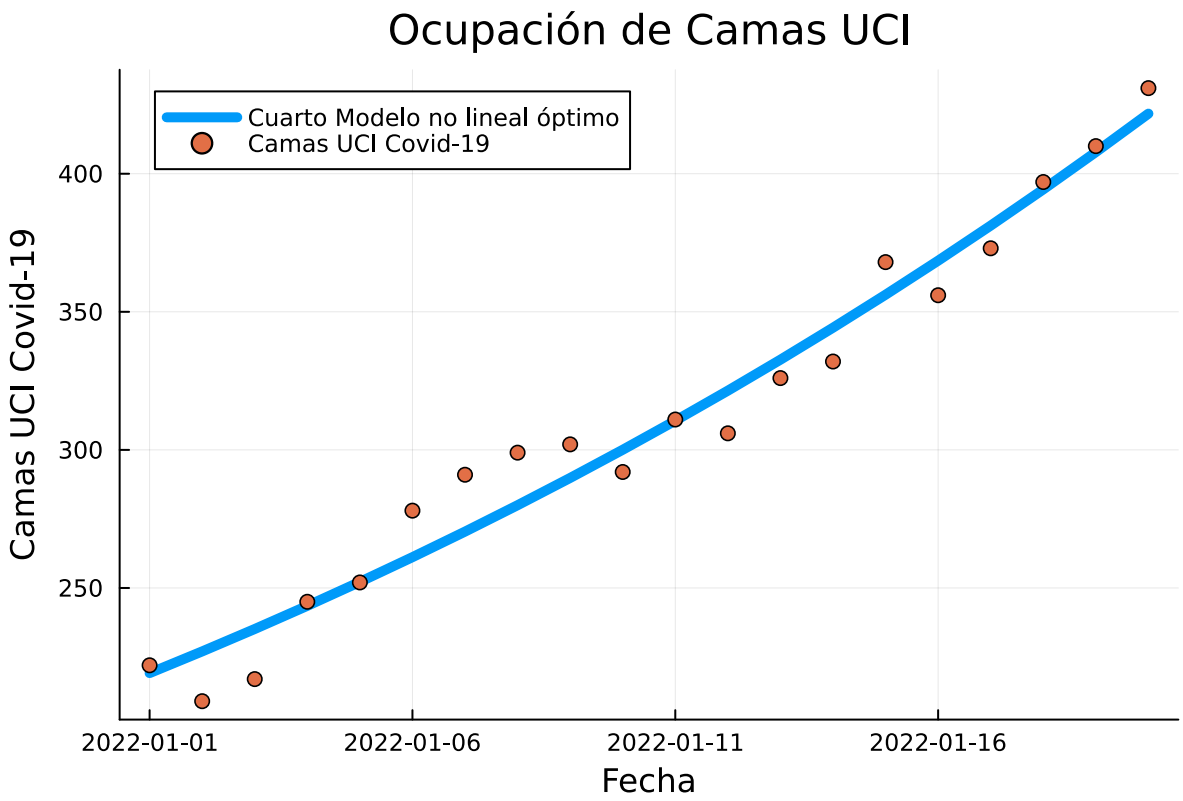
```
53.295193930789836
```

```
oNLF.minimum
```

De tal manera, obtenemos una función óptima que se acerca más a los datos deseados.

$$V(t) \approx \frac{4389.64}{1 + 19.746e^{-0.0370714}}$$

Veamos su gráfica:



Al igual que en nuestro modelo anterior, podemos observar que el desajuste es significativamente bajo, y nuestra aproximación es considerablemente mejor en comparación con los modelos previos. Esto se debe tanto a que el residuo es menor como a que cada parámetro tiene un significado cualitativo dentro del modelo. En particular:

1. El parámetro a representa la capacidad límite o valor asintótico superior, que corresponde al máximo valor que el modelo puede alcanzar. Es conocido también como la capacidad de carga.
2. El parámetro b corresponde al valor inicial, proporcionando una referencia para evaluar qué tan lejos o cerca estamos del límite a.
3. El parámetro c indica la tasa de cambio, que describe la velocidad de crecimiento del modelo.

Al igual que en el modelo exponencial, estas ideas intuitivas enriquecen el modelo, ya que no solo permiten realizar mejores aproximaciones, sino que también facilitan una interpretación más clara y comprensible para los usuarios, como en aplicaciones prácticas y académicas.

Modelos basados en Ecuaciones Diferenciales

Con el fin de analizar algunos modelos basados en EDO, analizaremos la aproximación que nos genera dos de estos modelos y loss ilustraremos. De esta manera, tendremos en cuenta los siguientes modelamientos:

Modelo de von Bertalanffy

Asumamos que nuestro método tiene siguiente forma:

$$V' = aV^{\frac{2}{3}} - bV$$

Donde V corresponde al valor de los datos y $a, b \in \mathbb{R}$ los parámetros a ser optmizados. Definimos una función para calcular la magnitud del residuo, el cual se realiza utilizando la norma euclidiana, aplicada a la diferenciaentre los valores predichos por el modelo y los valores reales de los datos. Así:

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

```
modeloVB(vDatos, tupla, tiempo) = tupla[1] * (vDatos ^ (2/3)) - tupla[2] * vDatos
```

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

```
function residuoVB(tupla,vDatos,tiempo)
    dominioTiempo = (0,30);
    V0 = 222;
    EDO = ODEProblem(modeloVB, V0, dominioTiempo, tupla);
    Sol = solve(EDO);
    vModelo = [Sol(t) for t in tiempo];
    res = vDatos - vModelo;
    nRes = norm(res);
    return nRes;
end
```

Así, tomamos una función que tome como valor principal un vector con los parametros para luego ser optimizado.

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

```
rVB(tupla) = residuoVB(tupla, camas, dias)
```

```
oVB =  * Status: success

       * Candidate solution
          Final objective value:     5.602550e+01

       * Found with
          Algorithm:     Nelder-Mead

       * Convergence measures
          √(Σ(yᵢ-ȳ)²)/n ≤ 1.0e-08

       * Work counters
          Seconds run:   0  (vs limit Inf)
          Iterations:    56
          f(x) calls:    110
```

```
1 oVB = Optim.optimize(rVB, [.01,.01], NelderMead())
```

```
Float64[
    1:  -0.334623
    2:  -0.0824633
]
```

```
1 oVB.minimizer
```

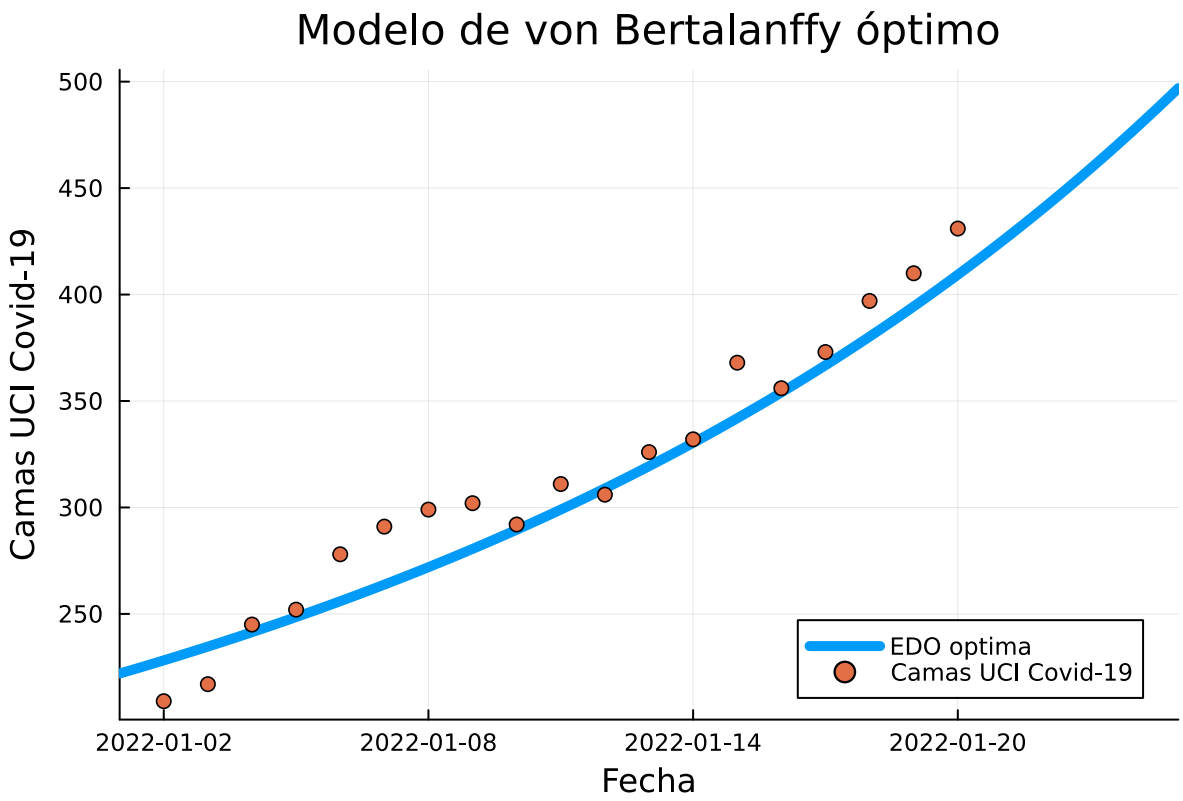
```
56.025497541122576
```

```
1 oVB.minimum
```

De tal manera, obtenemos una función óptima que se acerca más a los datos deseados.

$$V' = (-0.334623)V^{\frac{2}{3}} - (-0.0824633)V$$

Veamos su gráfica:



Modelo de Crecimiento Logístico

Asumamos que nuestro método tiene siguiente forma:

$$V' = aV\left(1 - \frac{V}{b}\right)$$

Donde V corresponde al valor de los datos y $a, b \in \mathbb{R}$ los parámetros a ser optmizados. Definimos una función para calcular la magnitud del residuo, el cual se realiza utilizando la norma euclidiana, aplicada a la diferenciaentre los valores predichos por el modelo y los valores reales de los datos. Así:

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

```
1 modeloCL(vDatos, tupla, tiempo) = tupla[1] * vDatos * (1 - (vDatos / tupla[2]))
```

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

```
function residuoCL(tupla,vDatos,tiempo)
    dominioTiempo = (0,30);
    V0 = 222;
    EDO = ODEProblem(modeloCL, V0, dominioTiempo, tupla);
    Sol = solve(EDO);
    vModelo = [Sol(t) for t in tiempo];
    res = vDatos - vModelo;
    nRes = norm(res);
    return nRes;
end
```

Así, tomamos una función que tome como valor principal un vector con los parametros para luego ser optimizado.

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

```
rCL(tupla) = residuoCL(tupla, camas, dias)
```

```
oCL =  * Status: success

       * Candidate solution
          Final objective value:     5.585818e+01

       * Found with
          Algorithm:     Nelder-Mead

       * Convergence measures
          √(Σ(yᵢ-ȳ)²)/n ≤ 1.0e-08

       * Work counters
          Seconds run:   0  (vs limit Inf)
          Iterations:    258
          f(x) calls:    483
```

```
1 oCL = Optim.optimize(rCL, [.01,.01], NelderMead())
```

```
Float64[
    1:  0.0150148
    2:  -262.515
]
```

```
1 oCL.minimizer
```

```
55.85817905999721
```

```
1 oCL.minimum
```

De tal manera, obtenemos una función óptima que se acerca más a los datos deseados.

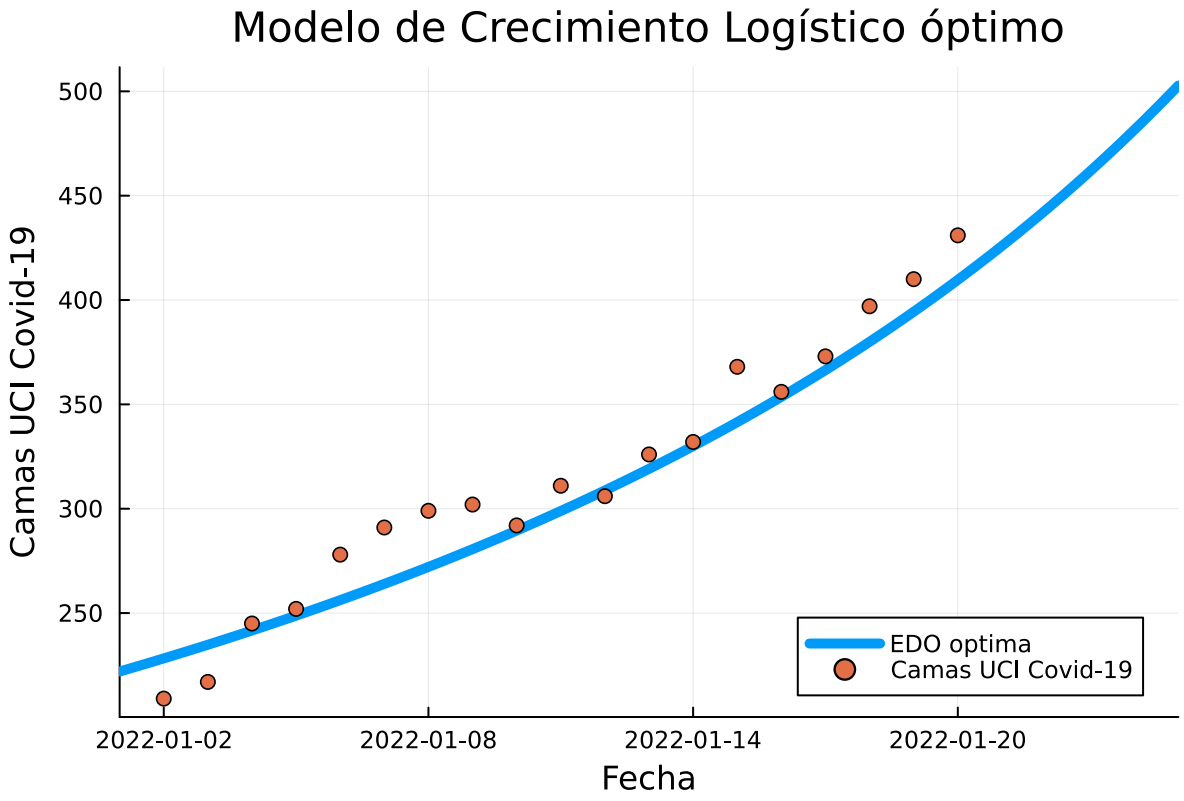
$$V' = (0.0150148)V\left(1 - \frac{V}{(-262.515)}\right)$$

Veamos su gráfica:

```
md"""
De tal manera, obtenemos una función óptima que se acerca más a los datos deseados.

$V' = (0.0150148)V\left(1 - \frac{V}{(-262.515)}\right)$

Veamos su gráfica:
"""
```



Es importante observar que los dos modelos no presentan diferencias notables a simple vista, si bien su funcionamiento y los parámetros optimizados difieren.

Bibliografía

[1] "Optim.jl", Sciml.ai, 2024. https://docs.sciml.ai/Optimization/stable/optimization_packages/optim/ Accedido 20 de Noviembre de 2024

[2] J. Galvis, F. Gómez y Y. Trujillo, "Ajuste de curvas", Laboratorio de matemáticas, Material de acompañamiento para los cursos del Departamento de Matemáticas y Ciencias de la Computación de la Universidad Nacional de Colombia, sede Bogotá, 2022. https://labmatecc.github.io/Notebooks/AnalisisNumerico/AjusteDeCurvas/ Accedido 21 de Noviembre de 2024