Examen Final de Modelado (PRACTICA)

Ejercicio #1

Primero obtenemos los datos de las funciones para encontrar los parámetros de nuestras funciones S(t) y U(t)

Insertamos los valores de las funciones evaluadas en dos puntos y esos dos puntos. El programa realiza el despeje para las constantes de las funciones.

```
% datoCmin = 5000;
% datosCmax  = 5500;
% tcmin =0.5;
% tcmax =1;
% divt1 = tcmax-tcmin;
% datosCdiv = datosCmax/datoCmin;
% b = log(datosCdiv)/divt1
% a = datosCmax/exp(b*tcmax)
b = 0.26
```

```
b = 0.2600
```

```
a = 1000
```

```
a = 1000
```

```
%Ojo ya que la exponencial de S(t) tiene argumento negativo el datos con
nombre max siempre será menor
datosSmin = 3500000;
datosSmax = 3200000;
tsmin = 0;
tsmax = 1;
divt2 = tsmax-tsmin;
datosSdiv = datosSmax/datosSmin;
d = -log(datosSdiv)/divt2
```

```
d = 0.0896
```

```
c = datosSmax/exp(-d*tsmax)
```

```
c = 3500000
```

Creamos las funciones con las constantes anteriormente obtenidas.

```
syms t
funS = @(t) c*exp(-d*t);
funC = @(t) a*exp(b*t);

K = funS(0)
```

```
K = 3500000
```

Linea auxiliar debido a que la función G de costos incluye una integral, por ello debemos obtener el resultado simbólico de la integral y convertirlo a función del tiempo

```
funCintegrada = matlabFunction(int(funC,0,t))
```

```
funCintegrada = function_handle with value:
    @(t)exp(t.*(1.3e+1./5.0e+1)).*3.846153846153846e+3-3.846153846153846e+3
```

Creamos la función G(t) y la graficamos en un intervalo limitado

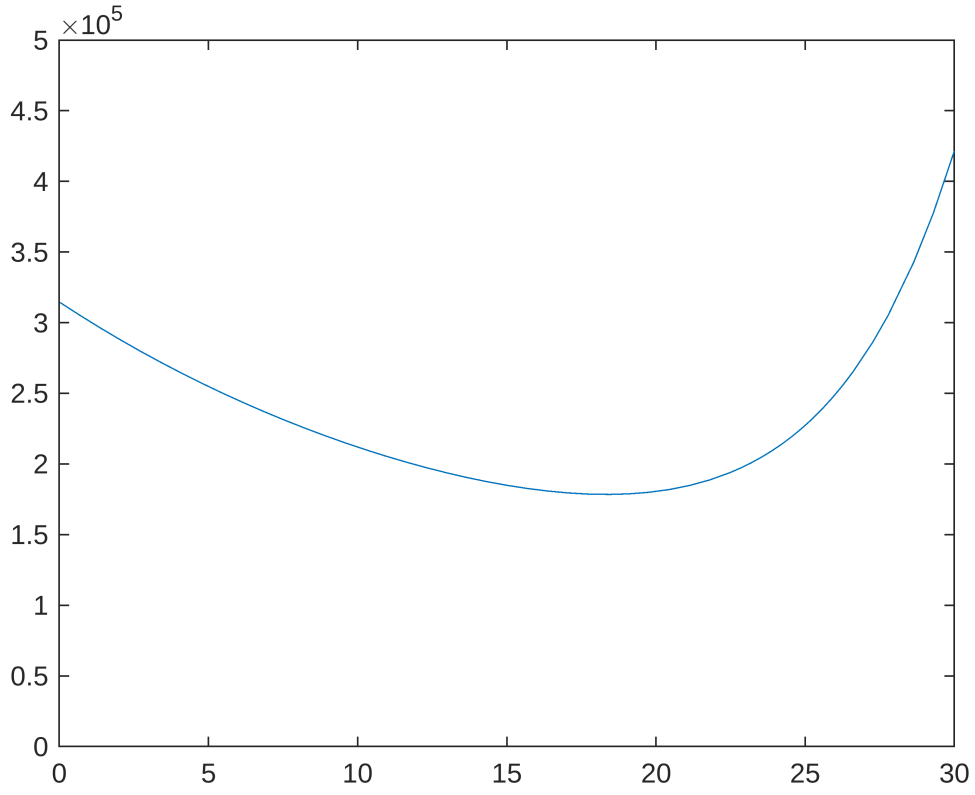
```
G = @(t) K/t + 1/t*funCintegrada(t) -funS(t)/t
```

```
G = function_handle with value:
    @(t)K/t+1/t*funCintegrada(t)-funS(t)/t
```

```
fplot(G, [0,30])
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize
your function to return an output with the same size and shape as the input arguments.
```

```
axis([0 30 0 5e5])
```



Observamos que el mínimo sucede alrrededor del cinco, debemos dar esto como parámetros a nuestro método Newton Rhapson junto con la función G para encontrar el mínimo.

La función genera el tiempo óptimo así como el valor de la función objetivo en ese punto.

```
[tOpt, fObj] = newtonRaphsonMinimo(G, 20)
```

```
tOpt = 18.3428
fObj = 1.7843e+05
```

Ejercicio #1 pt. 2

```
% datoCmin = 5000;
% datosCmax  = 5500;
% tcmin =0.5;
% tcmax =1;
% divt1 = tcmax-tcmin;
% datosCdiv = datosCmax/datoCmin;
% b = log(datosCdiv)/divt1
% a = datosCmax/exp(b*tcmax)
b = 0.26
```

```
b = 0.2600
```

```
a = 1000
```

```
a = 1000
```

```
%Ojo ya que la exponencial de S(t) tiene argumento negativo el datos con
nombre max siempre será menor
datosSmin = 3500000;
datosSmax = 3200000;
tsmin = 0;
tsmax = 1;
divt2 = tsmax-tsmin;
datosSdiv = datosSmax/datosSmin;
d = -log(datosSdiv)/divt2
```

```
d = 0.0896
```

```
c = datosSmax/exp(-d*tsmax)
```

```
c = 3500000
```

Creamos las funciones con las constantes anteriormente obtenidas.

```
syms t
funS = @(t) c*exp(-d*t);
funC = @(t) a*exp(b*t);

g = 3500000
```

```
g = 3500000
```

```
h = 0.041964
```

```
h = 0.0420
```

```
funK = @(t) g*exp(h*t);
```

Linea auxiliar debido a que la función G de costos incluye una integral, por ello debemos obtener el resultado simbólico de la integral y convertirlo a función del tiempo

```
funCintegrada = matlabFunction(int(funC,0,t))
```

```
funCintegrada = function_handle with value:
    @(t)exp(t.*(1.3e+1./5.0e+1)).*3.846153846153846e+3-3.846153846153846e+3
```

Creamos la función G(t) y la graficamos en un intervalo limitado

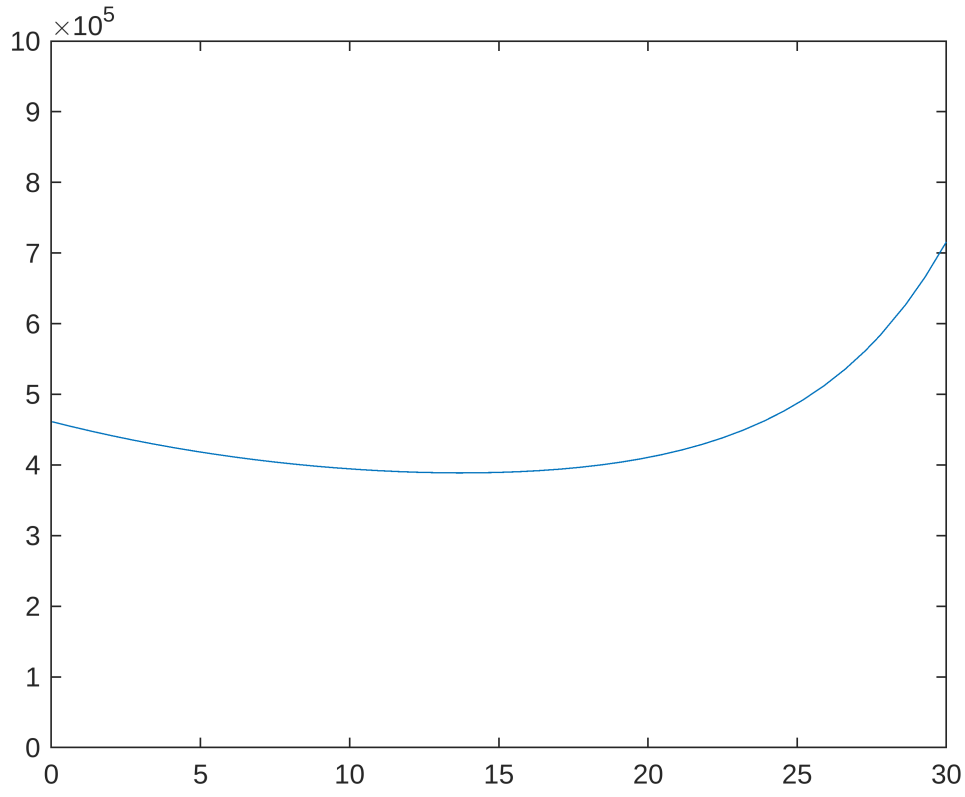
```
G = @(t) funK(t)/t + 1/t*funCintegrada(t) -funS(t)/t
```

```
G = function_handle with value:
    @(t)funK(t)/t+1/t*funCintegrada(t)-funS(t)/t
```

```
fplot(G, [0,30])
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize
your function to return an output with the same size and shape as the input arguments.
```

```
axis([0 30 0 1e6])
```



Observamos que el mínimo sucede alrrededor del cinco, debemos dar esto como parámetros a nuestro método Newton Rhapson junto con la función G para encontrar el mínimo.

La función genera el tiempo óptimo así como el valor de la función objetivo en ese punto.

```
[tOpt, fObj] = newtonRaphsonMinimo(G, 16)
```

```
tOpt = 13.7519
fObj = 3.8873e+05
```

Pregunta #2

Calculos de G y t para la función 1. También se despliega la gráfica cambiando las alhpas.

```
alpha = 0.1
```

```
alpha = 0.1000
```

```
G1 = @(t)  (t+alpha*(1-t))./(t-t.^2/2);
fplot(G1)
hold on
[tOpt, fObj] = newtonRaphsonMinimo(G1, 1)
```

```
tOpt = 0.3732
fObj = 1.4359
```

```
alpha = 0.2
```

```
alpha = 0.2000
```

```
G1 = @(t)  (t+alpha*(1-t))./(t-t.^2/2);
fplot(G1)
[tOpt, fObj] = newtonRaphsonMinimo(G1, 1)
```

```
tOpt = 0.5000
fObj = 1.6000
```

```
alpha = 0.3
```

```
alpha = 0.3000
```

```
G1 = @(t)  (t+alpha*(1-t))./(t-t.^2/2);
fplot(G1)
[tOpt, fObj] = newtonRaphsonMinimo(G1, 1)
```

```
tOpt = 0.5916
fObj = 1.7141
```

```
alpha = 0.4
```

```
alpha = 0.4000
```

```
G1 = @(t)  (t+alpha*(1-t))./(t-t.^2/2);
fplot(G1)
[tOpt, fObj] = newtonRaphsonMinimo(G1, 1)
```

```
tOpt = 0.6667
fObj = 1.8000
```

```
alpha = 0.5
```

```
alpha = 0.5000
```

```
G1 = @(t)  (t+alpha*(1-t))./(t-t.^2/2);
fplot(G1)
[tOpt, fObj] = newtonRaphsonMinimo(G1, 1)
```

```
tOpt = 0.7321
fObj = 1.8660
```

```
alpha = 0.6
```

```
alpha = 0.6000
```

```
G1 = @(t)  (t+alpha*(1-t))./(t-t.^2/2);
fplot(G1)
[tOpt, fObj] = newtonRaphsonMinimo(G1, 1)
```

```
tOpt = 0.7913
fObj = 1.9165
```

```
alpha = 0.7
```

```
alpha = 0.7000
```

```
G1 = @(t)  (t+alpha*(1-t))./(t-t.^2/2);
fplot(G1)
[tOpt, fObj] = newtonRaphsonMinimo(G1, 1)
```

```
tOpt = 0.8465
fObj = 1.9539
```

```
alpha = 0.8
```

```
alpha = 0.8000
```

```
G1 = @(t)  (t+alpha*(1-t))./(t-t.^2/2);
fplot(G1)
[tOpt, fObj] = newtonRaphsonMinimo(G1, 1)
```

```
tOpt = 0.8990
fObj = 1.9798
```

```
alpha = 0.9
```

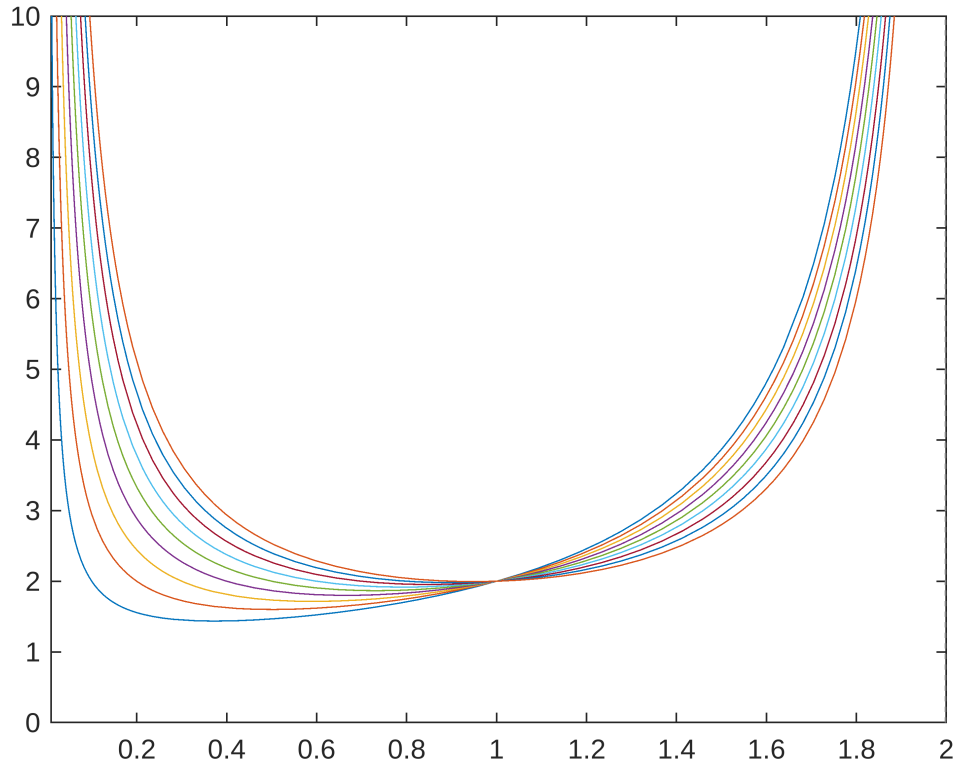
```
alpha = 0.9000
```

```
G1 = @(t)  (t+alpha*(1-t))./(t-t.^2/2);
fplot(G1)
axis([0.01 2 0 10])

[tOpt, fObj] = newtonRaphsonMinimo(G1, 1)
```

```
tOpt = 0.9499
fObj = 1.9950
```

```
%legend()
hold off
```



Pregunta #2

Calculos de G y t para la función 2 También se despliega la gráfica.

```
fun = @(t) 3*(2*t-t.^2+alpha*(1-2*t+t.^2))./(t*(t.^2-3*t+3));
alpha = 0.1
```

```
alpha = 0.1000
```

```
G1 = @(t) 3*(2*t-t.^2+alpha*(1-2*t+t.^2))./(t*(t.^2-3*t+3));
fplot(G1)
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize
your function to return an output with the same size and shape as the input arguments.
```

```
hold on
[tOpt, fObj] = newtonRaphsonMinimo(G1, 0.3)
```

```
tOpt = 0.2948
fObj = 2.5524
```

```
alpha = 0.2
```

```
alpha = 0.2000
```

```
G1 = @(t) 3*(2*t-t.^2+alpha*(1-2*t+t.^2))./(t*(t.^2-3*t+3));
fplot(G1)
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize
your function to return an output with the same size and shape as the input arguments.
```

```
[tOpt, fObj] = newtonRaphsonMinimo(G1, 0.3)
```

```
tOpt = 0.4126
fObj = 2.7240
```

```
alpha = 0.3
```

```
alpha = 0.3000
```

```
G1 = @(t) 3*(2*t-t.^2+alpha*(1-2*t+t.^2))./(t*(t.^2-3*t+3));
fplot(G1)
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize
your function to return an output with the same size and shape as the input arguments.
```

```
[tOpt, fObj] = newtonRaphsonMinimo(G1, 0.3)
```

```
tOpt = 0.5050
fObj = 2.8285
```

```
alpha = 0.4
```

```
alpha = 0.4000
```

```
G1 = @(t) 3*(2*t-t.^2+alpha*(1-2*t+t.^2))./(t*(t.^2-3*t+3));
fplot(G1)
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize
your function to return an output with the same size and shape as the input arguments.
```

```
[tOpt, fObj] = newtonRaphsonMinimo(G1, 0.3)
```

```
tOpt = 0.5858
fObj = 2.8971
```

```
alpha = 0.5
```

```
alpha = 0.5000
```

```
G1 = @(t) 3*(2*t-t.^2+alpha*(1-2*t+t.^2))./(t*(t.^2-3*t+3));
fplot(G1)
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize
your function to return an output with the same size and shape as the input arguments.
```

```
[tOpt, fObj] = newtonRaphsonMinimo(G1, 0.3)
```

```
tOpt = 0.6601
fObj = 2.9422
```

```
alpha = 0.6
```

```
alpha = 0.6000
```

```
G1 = @(t) 3*(2*t-t.^2+alpha*(1-2*t+t.^2))./(t*(t.^2-3*t+3));
fplot(G1)
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize
your function to return an output with the same size and shape as the input arguments.
```

```
[tOpt, fObj] = newtonRaphsonMinimo(G1, 0.3)
```

```
tOpt = 0.7307
fObj = 2.9710
```

```
alpha = 0.7
```

```
alpha = 0.7000
```

```
G1 = @(t) 3*(2*t-t.^2+alpha*(1-2*t+t.^2))./(t*(t.^2-3*t+3));
fplot(G1)
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize
your function to return an output with the same size and shape as the input arguments.
```

```
[tOpt, fObj] = newtonRaphsonMinimo(G1, 0.3)
```

```
tOpt = 0.7992
fObj = 2.9879
```

```
alpha = 0.8
```

```
alpha = 0.8000
```

```
G1 = @(t) 3*(2*t-t.^2+alpha*(1-2*t+t.^2))./(t*(t.^2-3*t+3));
fplot(G1)
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize
your function to return an output with the same size and shape as the input arguments.
```

```
[tOpt, fObj] = newtonRaphsonMinimo(G1, 0.3)
```

```
tOpt = 0.8665
fObj = 2.9964
```

```
alpha = 0.9
```

```
alpha = 0.9000
```

```
G1 = @(t) 3*(2*t-t.^2+alpha*(1-2*t+t.^2))./(t*(t.^2-3*t+3));
fplot(G1)
```

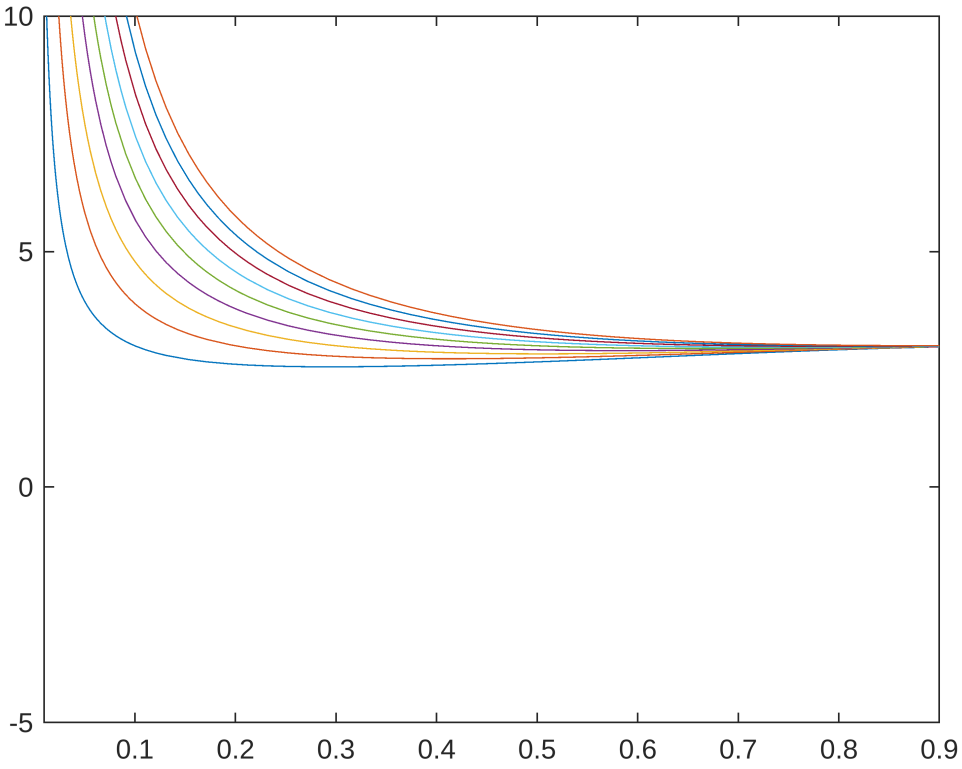
```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize
your function to return an output with the same size and shape as the input arguments.
```

```
axis([0.01 0.9 -5 10])

[tOpt, fObj] = newtonRaphsonMinimo(G1, 0.3)
```

```
tOpt = 0.9333
fObj = 2.9996
```

```
%legend()
hold off
```



Nota: No me dio el tiempo de graficar la función 3 y de crear una tabla para comparar los tiempos, pero es más rápido y evidente en el Excel.

Problema #4

Enfoque tradicional

```
syms t G
a = 0;
b = 4;

C1 = 1000;
C2 = 10;
%exp con alpha = -0.01

f = @(t) 1/2-t/8;
F = matlabFunction(int(f, 0, t))
```

```
F = function_handle with value:
    @(t)t.*(t-8.0).*(-1.0./1.6e+1)
```

```
R = @(t) 1- F(t)
```

```
R = function_handle with value:
    @(t)1-F(t)
```

```
interR = matlabFunction(int(R, 0, t))
```

```
interR = function_handle with value:
    @(t)(t.*(t.*-1.2e+1+t.^2+4.8e+1))./4.8e+1
```

```
G1 = @(t) (C1*F(t)+C2*R(t))/interR(t)
```

```
G1 = function_handle with value:
    @(t)(C1*F(t)+C2*R(t))/interR(t)
```

```
fplot(G1, [0.01, 3])
```

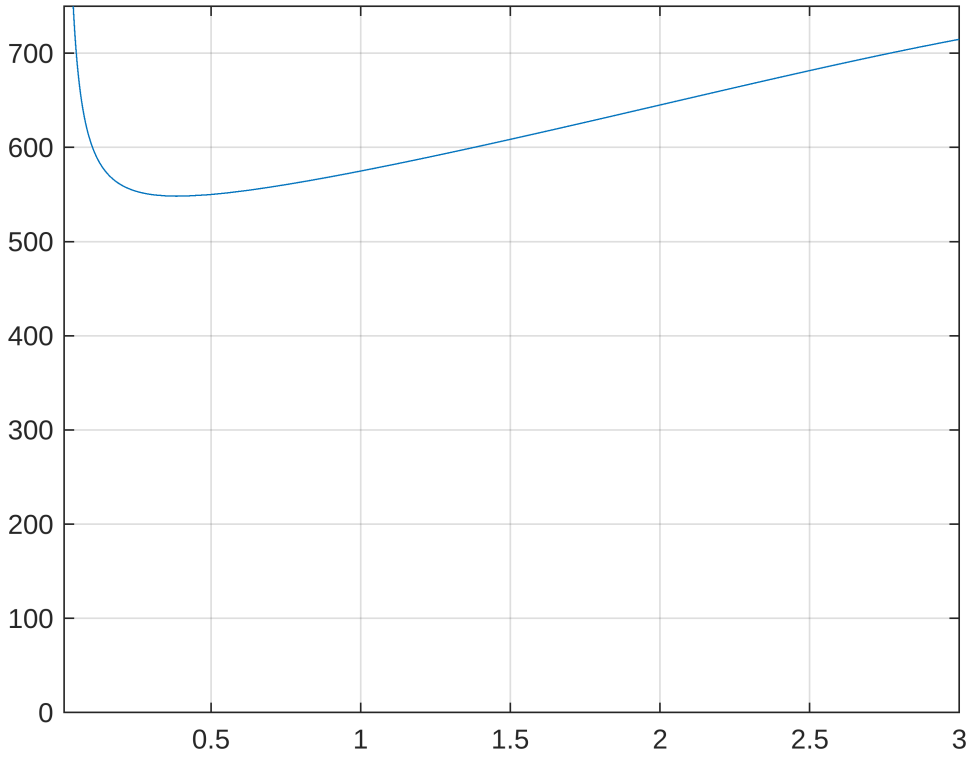
```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize
your function to return an output with the same size and shape as the input arguments.
```

```
axis([0.01 3 0 750])
grid on
[tOpt, fObj] = newtonRaphsonMinimo(G1, 0.5)
```

```
tOpt = 0.3883
fObj = 548.2240
```

Enfoque nuevo encontrando una G manteniendo tiempos óptimos

```
hold off
```




```
U =  G*interR(t)-C1*F(t)-C2*R(t)
```

U =

$$\frac{495\,t\,(t-8)}{8}+\frac{G\,t\,(t^2-12\,t+48)}{48}-10$$

```
Udf = diff(U,t)
```

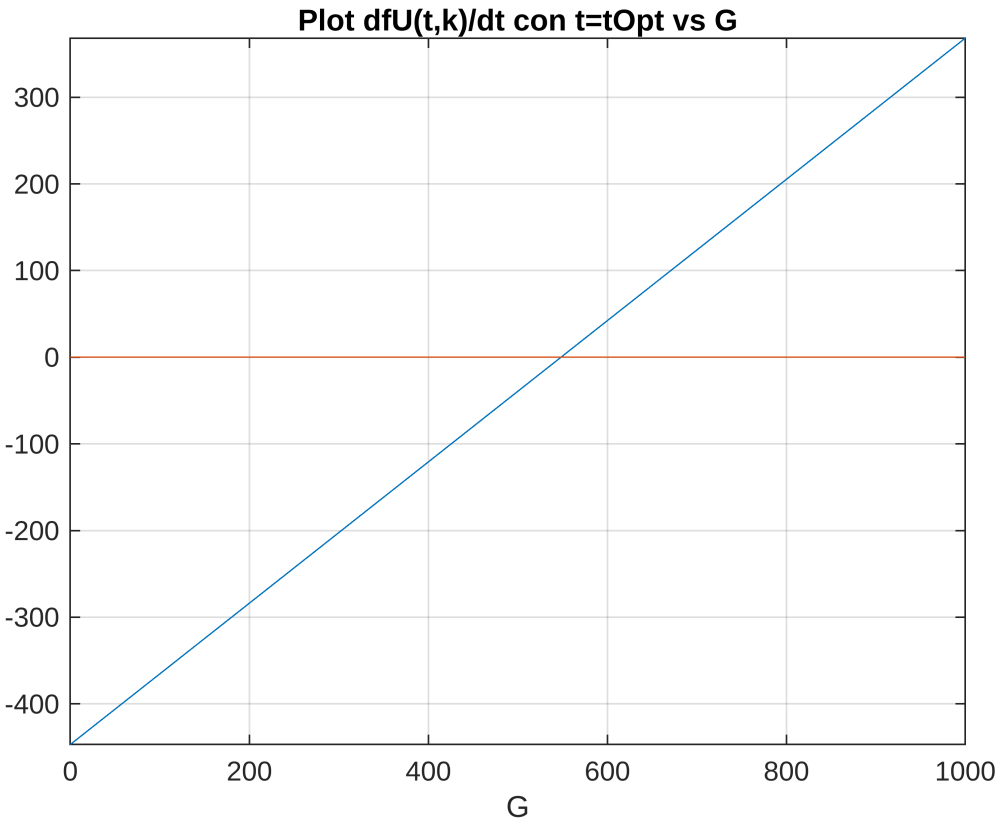
Udf =

$$\frac{495\,t}{4}+\frac{G\,(t^2-12\,t+48)}{48}+\frac{G\,t\,(2\,t-12)}{48}-495$$

```
Udf = matlabFunction(Udf, 'Vars', {t,G});
Udf = @(G) Udf(tOpt,G);
fplot(Udf, [0,1000]);hold on;fplot(@(t) 0, [0,1000]);title("Plot dfU(t,k)/dt
con t=tOpt vs G");xlabel("G");hold off;grid on
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize
your function to return an output with the same size and shape as the input arguments.
```

```
hold off
```



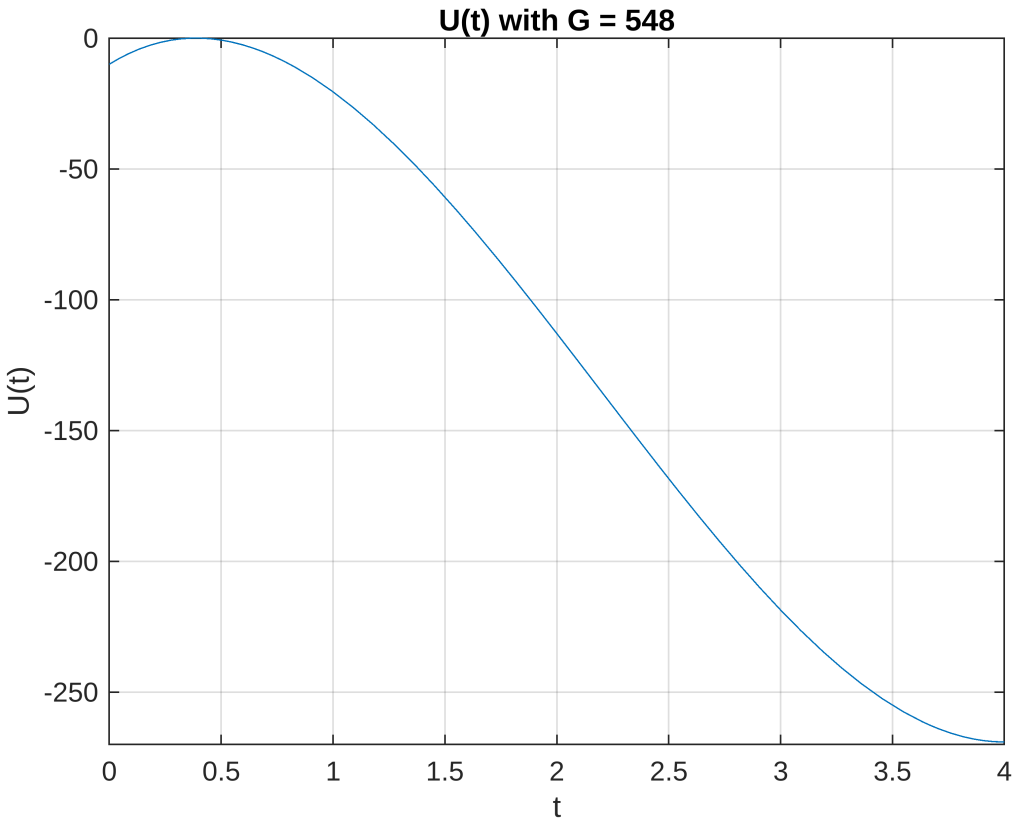

```
root = fzero(Udf, 2)
```

```
root = 548.2240
```

```
U = @(t) root*interR(t)-C1*F(t)-C2*R(t)
```

```
U = function_handle with value:
    @(t)root*interR(t)-C1*F(t)-C2*R(t)
```

```
fplot(U, [a,b]);title("U(t) with G = 548");xlabel("t");ylabel("U(t)");grid on
```




```
[tOpt2, fObj2] = newtonRaphsonMinimo(U, .5)
```

```
tOpt2 = 0.3883
fObj2 = 5.3291e-15
```

Haciendo variaciones en la G

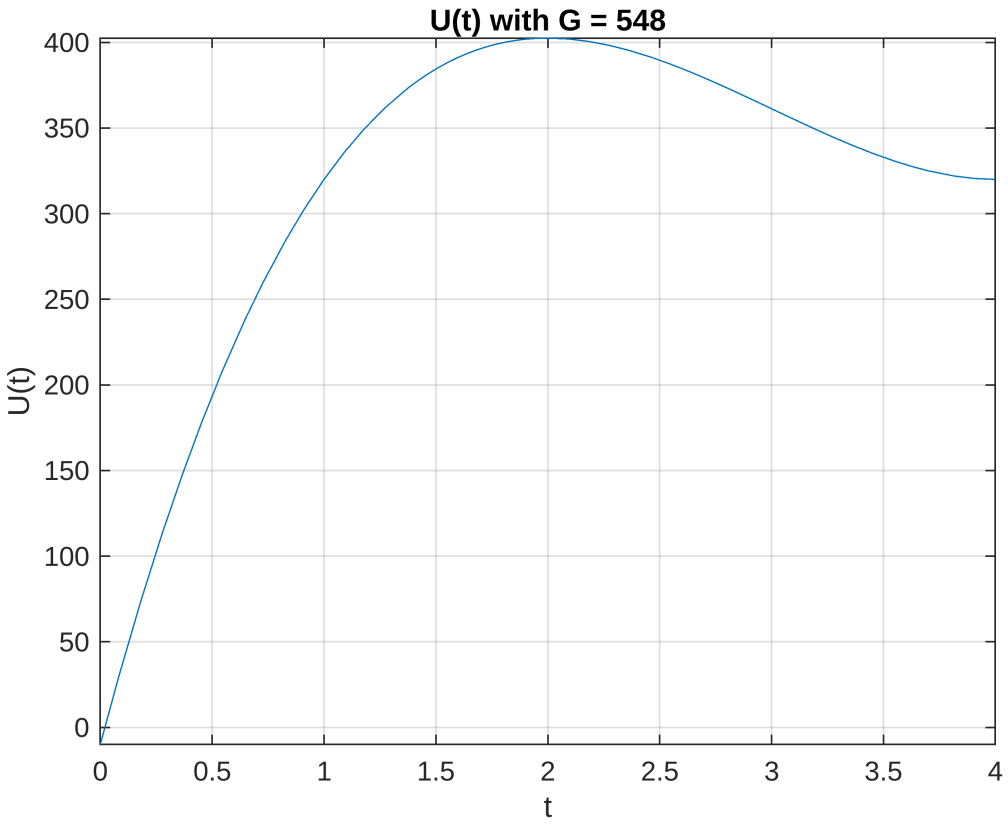
```
Gc = 990
```

```
Gc = 990
```

```
U = @(t) Gc*interR(t)-C1*F(t)-C2*R(t)
```

```
U = function_handle with value:
    @(t)Gc*interR(t)-C1*F(t)-C2*R(t)
```

```
fplot(U, [a,b]);title("U(t) with G = 548");xlabel("t");ylabel("U(t)");grid on
```




```
[tOpt2, fObj2] = newtonRaphsonMinimo(U, .5)
```

```
tOpt2 = 2
fObj2 = 402.5000
```

Buscamos los valores Esperados de la función.

```
h = matlabFunction(int(t*U, 0, t))
```

```
h = function_handle with value:
    @(t)(t.^2.*(t.*5.28e+3-t.^2.*1.485e+3+t.^3.*1.32e+2-1.6e+2))./3.2e+1
```

```
E = int(t*U, 0, tOpt2)
```

```
E =
```

$$\frac{1379}{2}$$

```
EUOpt = h(tOpt2)
```

```
EUOpt = 689.5000
```

Minimización: Newton-Rhapson

Esta función se manda a llamar para minimizar. Solo es encesario dar la función y un punto cercano que no cruce por asíntotas.

```
function [x, fx, i, min] = newtonRaphsonMinimo(f,x)
    %Parámetros iniciales
    max = 52;
    tol = sqrt(eps);
    i = 0;
    cond =  true;
    %Declaramos las funciones anónimas
    f = f;
    fs = sym(f);
    dfs = diff(fs);
    dfs2 = diff(dfs);
    df = matlabFunction(dfs);
    df2 = matlabFunction(dfs2);
    if nargin(df)<1
        error('Function has not second reivate')
    end
    if nargin(df2) < 1
        df2 = @(x) df2();
    end
    while cond
        xp = x;
        x = xp-df(xp)/df2(xp);
        cond = abs((x-xp)/x) > tol && i < max;
        i = i+1;
    end
    min = df2(x) > 0;
    fx = f(x);
end
```