entregable

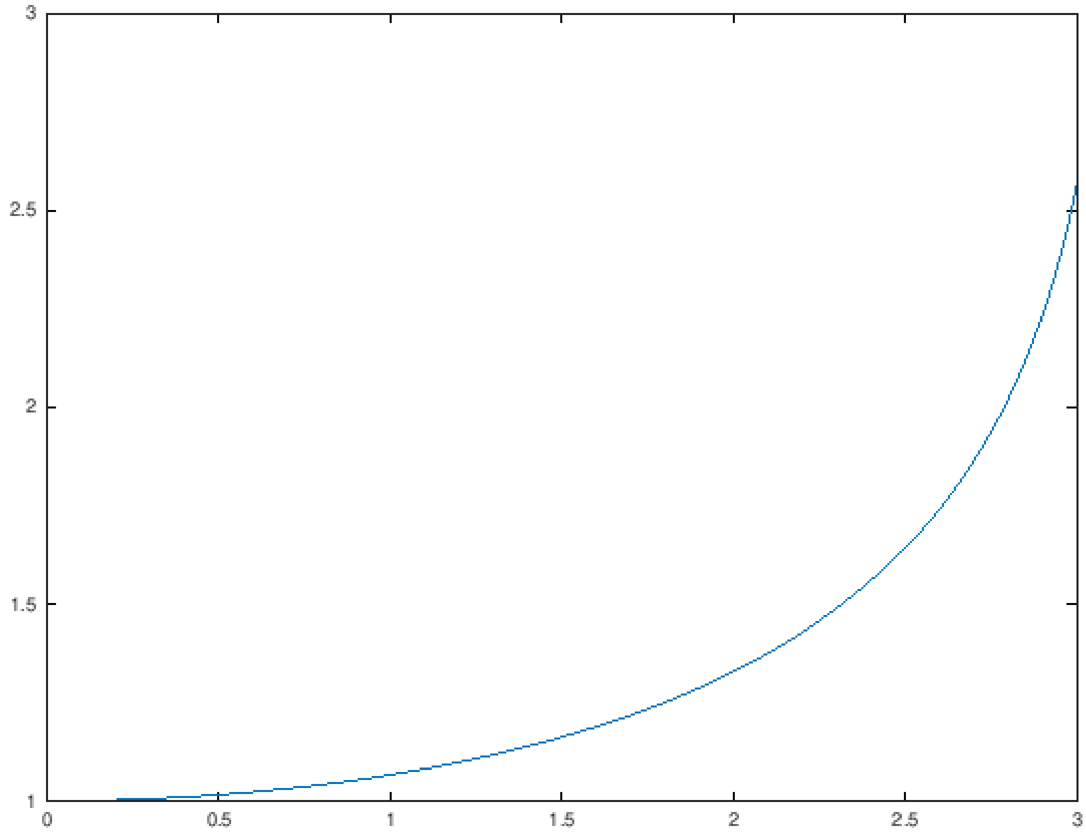
May 21, 2023

1 Ejercicio 1:

[121]:
```
clear all
addpath('./Biblioteca')
graphics_toolkit ("gnuplot"); %% Comando solo para jupyter notebooks
output_precision(16)
fmt='%5u\t %+17.17f\t %+17.17f\n';
```

[122]:
```
g = 9.807;
T0 = (2*pi)/sqrt(g);

a = 0;
b = 3;

f = @(x,alpha) 1./sqrt(1-((sin(alpha/2)).^2.*(sin(x)).^2));
```

[123]:
```
abcisas = linspace(0,3,200);
figure (1)
for k=1:length(abcisas)
    fz = @(x) f(x,abcisas(k));
    y(k) = 2*(1/pi)*simpson(fz, 0, pi/2, 20);
endfor
plot(abcisas, y)
```



[124]:
```
function [alpha_1, ga1, numpasos] = secante_mod(x0, x1, tol, maxit)
    numpasos = 1;
    imprime = 1;
    fmt='%5u\t %+17.17f\t %+17.17f\n';

    f = @(x,alpha) 1./sqrt(1-((sin(alpha/2)).^2.*(sin(x)).^2));
    g = 9.807;

    alpha_0 = x0;
    g0 = @(x) f(x, alpha_0);
    ga0 = simpson(g0, 0, pi/2, 100) * (2/pi) -1.5; %% Restamos 1.5 porque es␣
 ↪donde queremos hallar la raíz.

    alpha_1 = x1;
    g1 = @(x) f(x, alpha_1);
    ga1 = simpson(g1, 0, pi/2, 100) * (2/pi) -1.5;

    if (imprime == 1)
      printf(fmt,numpasos,alpha_1,ga1)
    endif
```

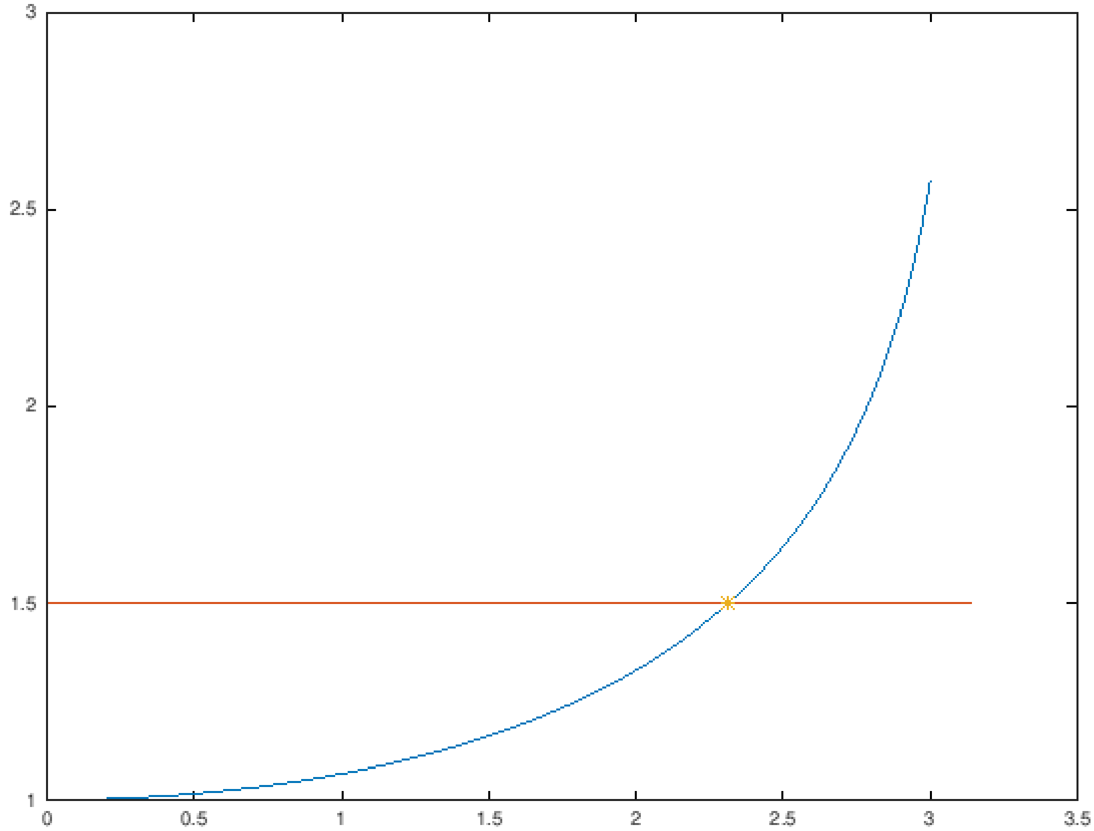
```
    while numpasos <= maxit
        numpasos += 1;
        if (abs(ga1 - ga0))<eps
            disp('Pendiente = 0');
            return;
        endif

        d = ga1 * (alpha_1 - alpha_0)/(ga1 - ga0);

        alpha_0 = alpha_1;
        ga0 = ga1;

        alpha_1 = alpha_1 - d;
        g1 = @(x) f(x, alpha_1);
        ga1 = simpson(g1, 0, pi/2, 100) * (2/pi)-1.5;

        if (imprime == 1)
            printf(fmt,numpasos,alpha_1,ga1)
        endif

        if and((abs(d)<=tol*(1+abs(alpha_1))), abs(ga1)<=tol)
            return;
        endif

    endwhile
    disp('No hay convergencia.');
    return;
endfunction
```

[125]:

```
figure(2)

x0 = 3;
x1 = 2.5;

[alpha, ga, numpasos] = secante_mod(x0, x1, 1E-14, 20);

plot(abcisas, y, [0, pi], [1.5, 1.5])
hold on
plot(alpha,ga+1.5,'*')

printf('La aproximación obtenida en %3u pasos es: %15.15e\n',numpasos,alpha)

%% Una de las ventajas de usar el método de la secante es, que al ser convexa␣
↪la función el método converge a la raíz muy rápidamente.
```

```
1    +2.50000000000000000    +0.14298239844576588
2    +2.42298294665177849    +0.07798822291468976
```

```
    3    +2.33056815600824230    +0.01008845547592063
    4    +2.31683729159011920    +0.00079951874313866
    5    +2.31565544658701539    +0.00000884684192415
    6    +2.31564222290231747    +0.00000000784016096
    7    +2.31564221117295954    +0.00000000000007749
    8    +2.31564221117284363    -0.00000000000000022
    9    +2.31564221117284408    -0.00000000000000022
La aproximación obtenida en   9 pasos es: 2.315642211172844e+00
```

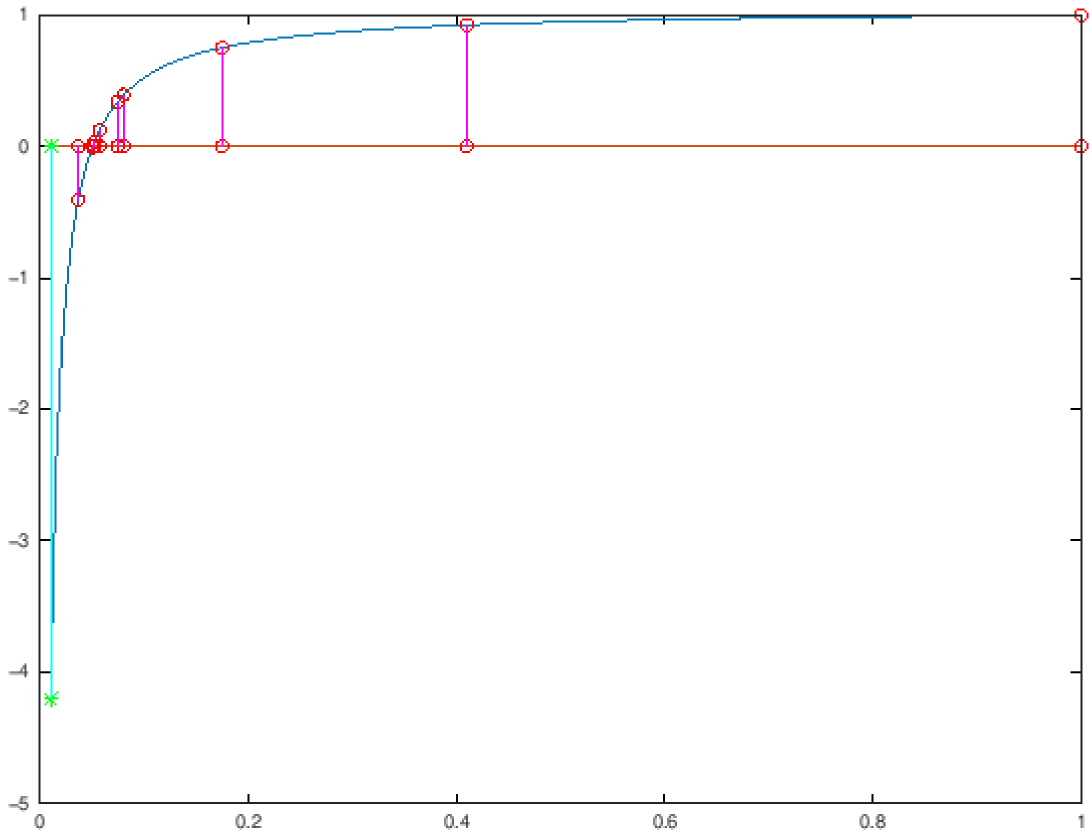


2 Ejercicio 2:

[126]:
```
maxit = 50;
imprime = 0;
tol = 10^(-12);

f = @(x) (20.*x - 1)./(19.*x);

x0 = 0.01;
x1 = 1;
```

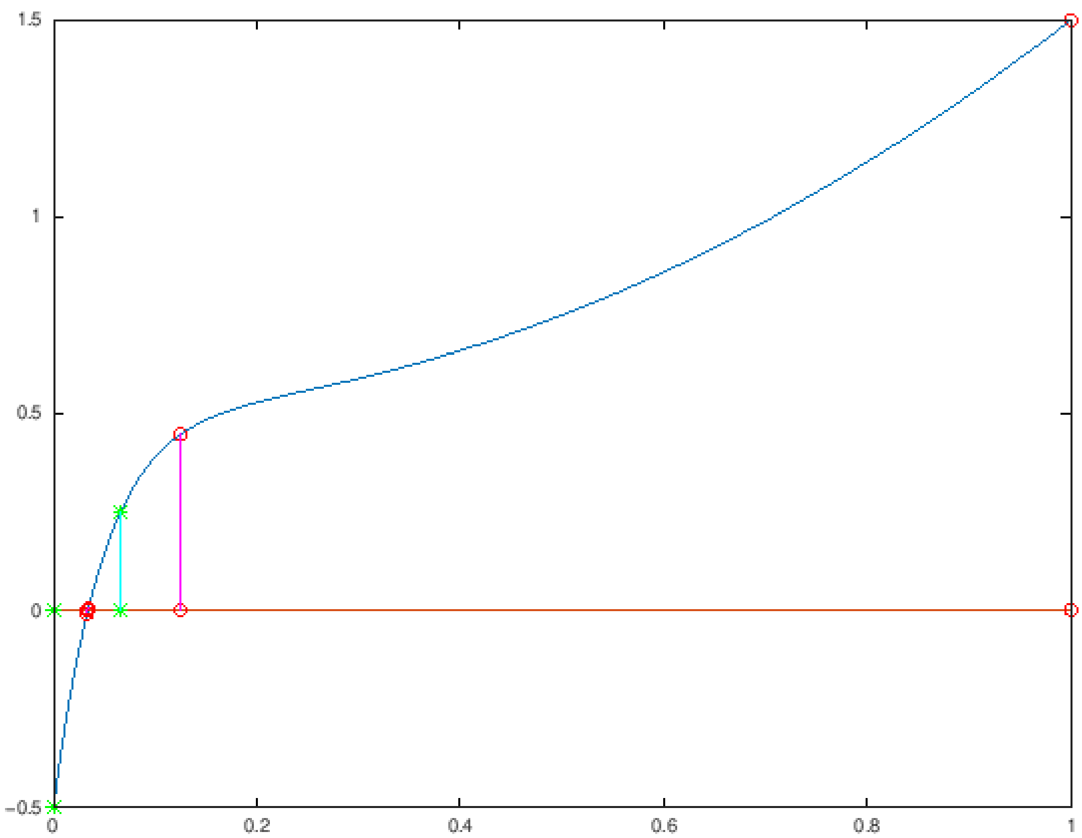
```
[x, fx, npasos] = zeroshybrid(f, x0, x1, tol, maxit, imprime);
[x,fx,npasos]= secante(f,x0,x1,tol,maxit,imprime)

%% En este caso podemos observar que directamente el método de la secante
normal directamente no converge, pues llega un caso en el que
%% la pendientes es tan pequeña (menor que eps cuando para) que el ordenador
empieza a cometer errores de redondeo pues no tiene precisión suficiente para
%% realizar los calculos. Con el nuevo método nos ahorramos estos problemas.
```

```
…

fx = -1.168655815394902e-16
x = 5.000000000000000e-02
npasos = 17
…END

Secante: pendiente nula
x = -9.306821065433922e+39
fx = 1.052631578947368
npasos = 10
```



[127]:
```
f = @(x) x.^2 - (x-1).^(20) + 1/2;

x0 = 0;
x1 = 1;

[x, fx, npasos] = zeroshybrid(f, x0, x1, tol, maxit, imprime);
[x,fx,npasos]= secante(f,x0,x1,tol,maxit,imprime);
[x,fx,npasos]= biseccion(f,x0,x1,tol,imprime);

%% Este caso es más sorprendente si cabe. El método de la bisección necesita de
 ↪41 pasos mientras que el método de la secante ni siquiera converge.
%% Por su parte el método mixto consigue llegar a la aproximación en solo 10
 ↪pasos.
```

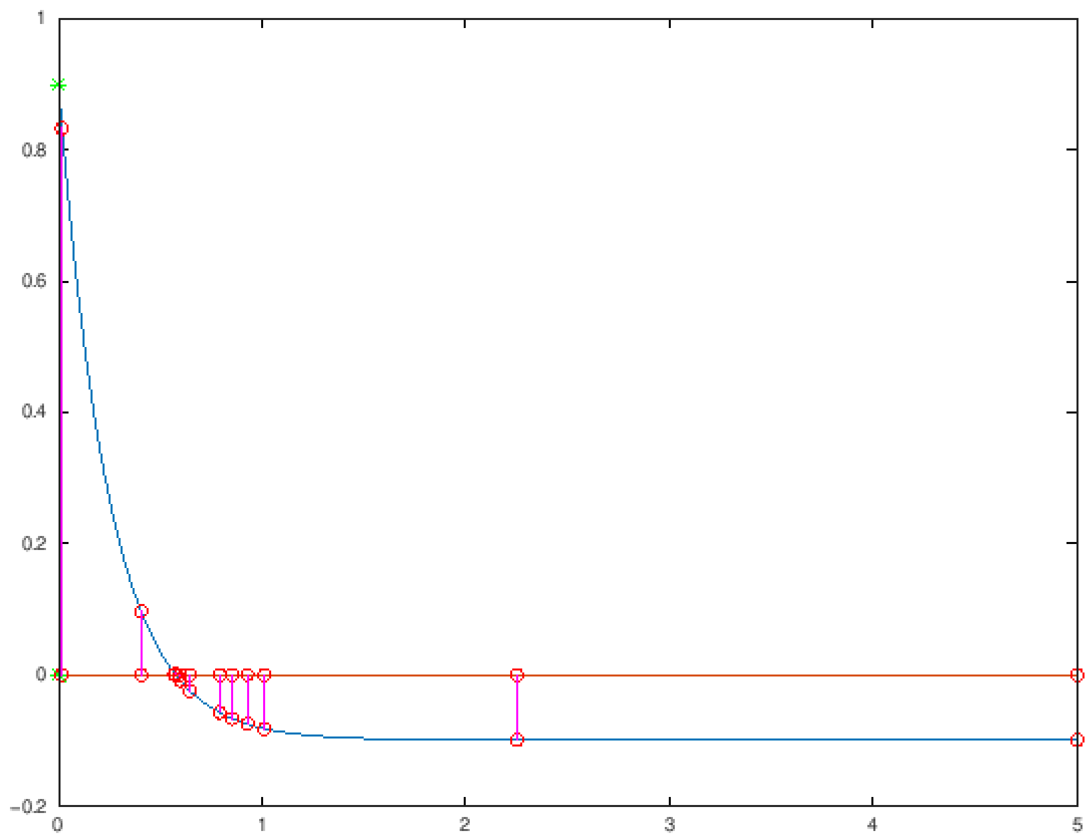
```
…

fx = 6.661338147750939e-16
x = 3.395244275091246e-02
npasos = 10
…END

Secante: No hay convergencia
```



[128]:

```
f = @(x) e.^(-4.*x) - 1/10;

x0 = 0;
x1 = 5;

[x, fx, npasos] = zeroshybrid(f, x0, x1, tol, maxit, imprime);
[x,fx,npasos]= secante(f,x0,x1,tol,maxit,imprime);
[x,fx,npasos]= regulaFalsi (f, x0, x1,tol,200,imprime)

%% Nuevamente, el método de la secante no es capaz de converger numéricamente␣
 ↪al ser la pendiente de f por la derecha de la raíz tan pequeña de manera
%% constante y estar tan pegada al eje de abcisas. Después de la primera␣
 ↪iteración, los siguientes dos puntos están tan pegados y la diferencia de sus
%% imagenes por f tan pequeña que el siguiente punto de la iteración es tan␣
 ↪negativo que el ordenador no puede computar su imagen por f y directamente
%% le asigna el valor +inf. Por otro lado el método regula falsi sí converge␣
 ↪pero de manera muy lenta por estar tan pegada la función al cero a la
%% derecha de la raíz, necesitando de casi 100 pasos. Por su parte el método␣
 ↪híbrido aprovecha la bisección (método que ya de por si sería mucho más
%% rápidp que cualquiera de los otros dos ya mencionados) para situarse en un␣
 ↪intervalo lo suficientemente pequeño donde el método de la secante puede
%% actuar sin problemas, alcanzando así la aproximación en solo 18 pasos.
```

```
…

fx = -1.387778780781446e-17
x = 0.575646273248511
npasos = 18
…END

Secante: No hay convergencia
x = 0.575646273248512
fx = -1.526556658859590e-16
npasos = 200
```


1
0.8
0.6
0.4
0.2
0
−0.2
0
1
2
3
4
5