Se ingresa por pantalla:

>> fy=@(x,y,z)z

fy =

@(x, y, z) z

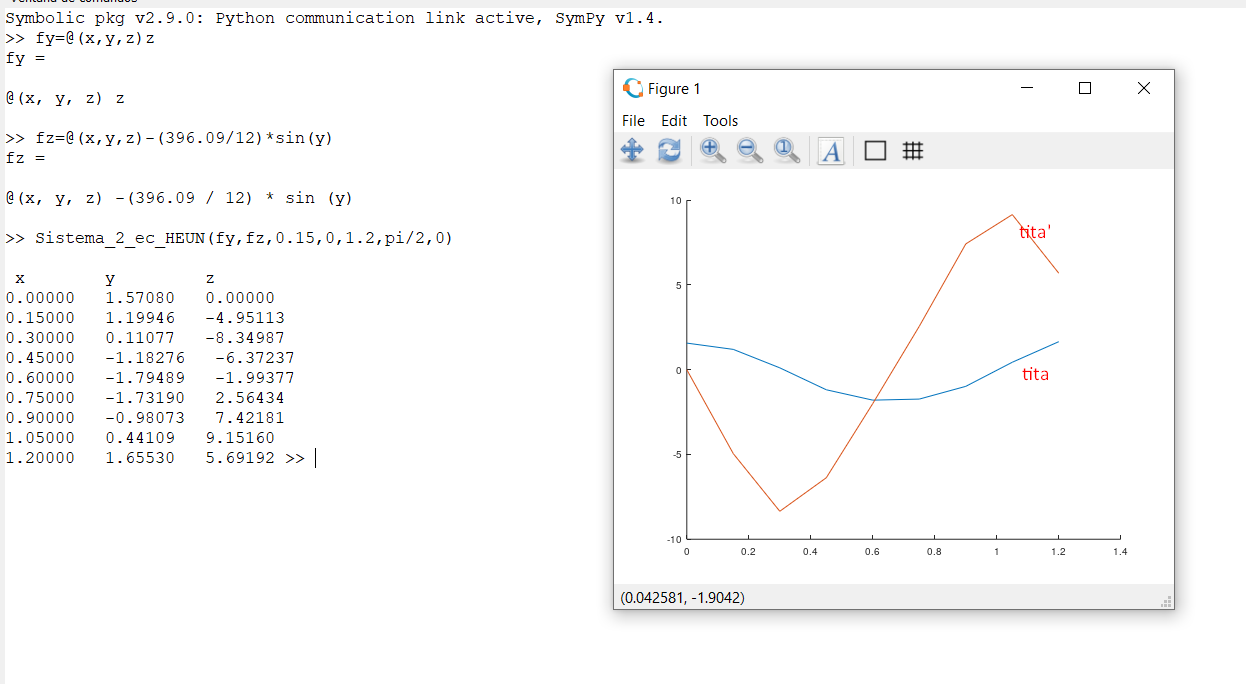
>> fz=@(x,y,z)-(396.09/12)\*sin(y)

fz =

@(x, y, z) -(396.09 / 12) \* sin (y)

>>se llama al programa ingresando los datos.

Sistema\_2\_ec\_HEUN(fy,fz,0.15,0,1.2,pi/2,0)



>> Sistema\_2\_ec\_HEUN(fy,fz,0.15,0,1.2,pi/2,0)

x y z

0.00000 1.57080 0.00000

0.15000 1.19946 -4.95113

0.30000 0.11077 -8.34987

0.45000 -1.18276 -6.37237

0.60000 -1.79489 -1.99377

0.75000 -1.73190 2.56434

0.90000 -0.98073 7.42181

1.05000 0.44109 9.15160

1.20000 1.65530 5.69192 >>

Tuve que hacer un cambio de variable

Programa:

function[]=Sistema\_2\_ec\_HEUN(fy,fz,h,x0,xn,y0,z0)% hay otra verison con la funcion solucion en forma de programa

% sistema de ecuaciones diferenciales con euler mejorado.

%fy ecuacion diferencial dejar x para el tiempo. y'=dy/dx

%fz cuacion diferencial dejar x para el tiempo. z'=dz/dx

% x0 es el valor de t(tiempo) inicial

% xn es el valor de tiempo final.

% y0 es el valor de la variable y en x=0

% z0 es el valor de la variable y en z=0

%-------------------------------------------------------------------------------

% cazador-presa

%fy = @(x,y,z)0.4\*z-0.02\*z\*y; % ecuacion diferencial dejar x para el tiempo. y'=dy/dx

%fz= @(x,y,z)0.001\*z\*y-0.03\*y; % ecuacion diferencial dejar x para el tiempo. z'=dz/dx

%-------------------------------------------------------------------------------

%Formula: y1=y0+h/2\*[f(x0,y0)+f(x1,y1\*)] where y1\*=y0+h\*f(x0,y0);

n=(xn-x0)/h;

Xvect=zeros(1,n);

Yvect=zeros(1,n);

Zvect=zeros(1,n);

fprintf('\n x y z');

Xvect(1,1)=x0;

Yvect(1,1)=y0;

Zvect(1,1)=z0;

i=1;

while x0<=xn

fprintf('\n%4.5f %4.5f %4.5f ',x0,y0,z0);%values of x and y

ky=y0+h\*fy(x0,y0,z0); % Ky seria el Yestrellita

kz=z0+h\*fz(x0,y0,z0); % Kz seria Zestrellita

x1=x0+h;

Xvect(1,i+1)=x1; %Xvect va almacenando los valores de x

y1=y0+h/2\*(fy(x0,y0,z0)+fy(x1,ky,kz)); % y1 es el valor de y con euler mejorado

z1=z0+h/2\*(fz(x0,y0,z0)+fz(x1,ky,kz)); % z1 es el valor de z con euler mejorado

Yvect(1,i+1)=y1;

Zvect(1,i+1)=z1;

x0=x1;

y0=y1;

z0=z1;

i=i+1;

end

l=size(Xvect);

l=l(1,2);

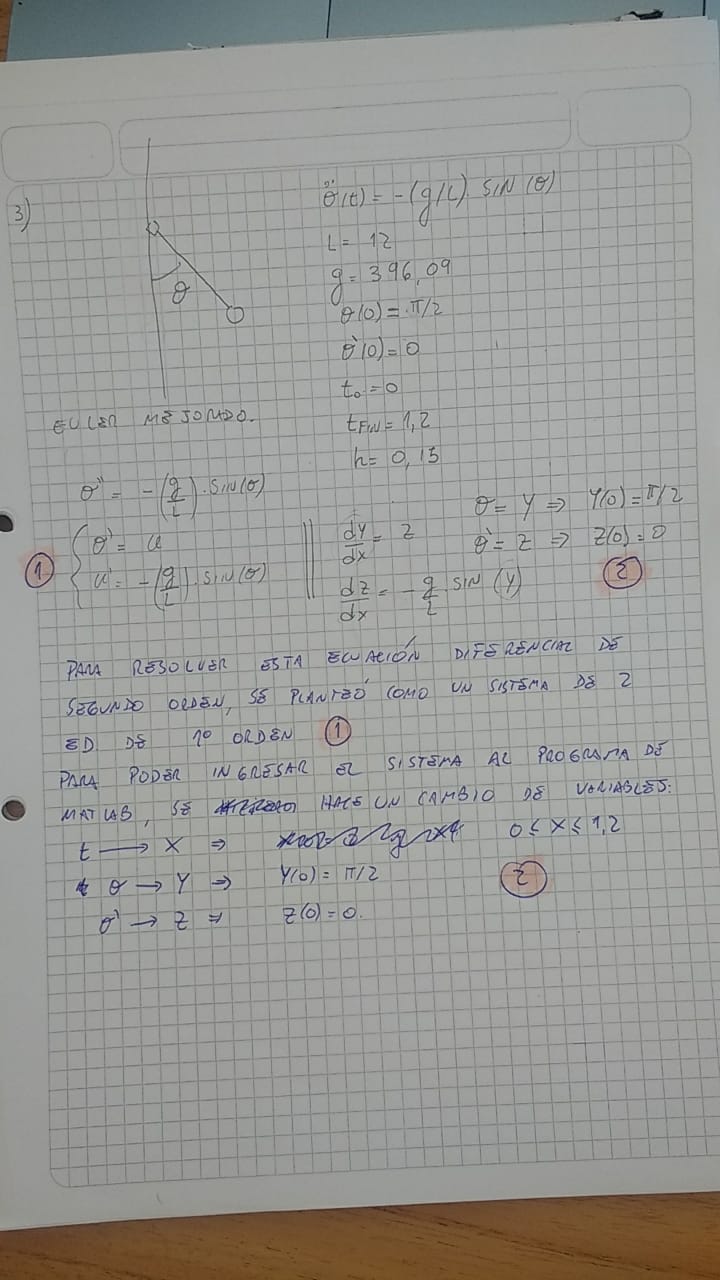
hold on

plot(Xvect(1,1:l-1), Yvect(1,1:l-1));

hold on

plot(Xvect(1,1:l-1), Zvect(1,1:l-1));

endfunction



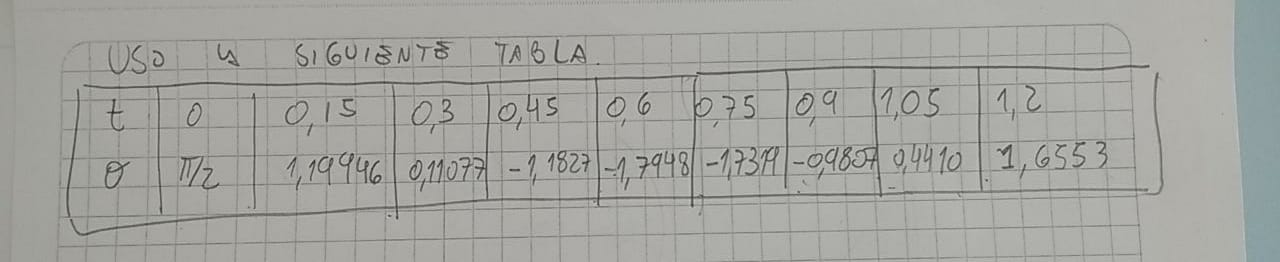
El movimiento si parece ser periódico (observar grafico)

b) se puede encontrar una mejor aproximación, seto se puede lograr usando un tamaño de paso mas chico o usando otro método (por ejemplo runge kutta de orden 4) este ultimo, devuelve mejores resultados, ya que “considera” hasta la derivada de orden 4 en la serie de Taylor (primeros 5 terminos), en lugar de la derivada segunda que se considera en el método de heun.

c) dy/dt=5

la solución a dicha ecuación diferencial es una recta, y como Euler funciona buscando la pendiente en cada punto de t (con intervalos de h). Como busca la pendiente de la recta (dy/dt), y traza una recta con dicha pendiente, aproxima a una recta con otra recta de la misma pendiente. Por ende no hay error.

d) se armo una tabla, que contiene los valores que toma t y el angulo tita encontrados cuando se resolvió la ecuación diferencial.



Para resolver la integral, se uso el siguiente programa:

%VALORES EJERCICIO 1-a

X1=[0 0.15 0.3 0.45 0.6 0.75 0.9 1.05 1.2 ]

Y1=[pi/2 1.19946 0.11077 -1.1827 -1.7948 -1.7319 -0.98073 0.44109 1.6553]

clc

pkg load symbolic

syms x

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [resultado]=Trapecios\_compuesto\_funcion(f,a,b,error,N)

%para un n determinado, ingresar error=0

syms x

F=function\_handle(f)

fiv=diff(f,2);

Fiv=function\_handle(fiv)

% las siguientes lineas calculan el maximo de la derivada segunda de f en modulo

if a!=0

abs\_max=abs(feval(Fiv,a));

posx=a;

else

abs\_max=abs(feval(Fiv,(a+0.0001)));

posx=a+0.0001;

endif

for i=(a+0.0001):0.0001:b

e1=abs(feval(Fiv,i));

if (abs\_max<e1)

abs\_max= e1;

posx=i;

endif

endfor

% se grafica la derivada segunda y su valor maximo en modulo

hold on

fplot(Fiv, [a, b])

plot(posx,abs\_max,'.r','markersize',20);

title('derivada segunda y su maximo (en valor absoluto)')

hold off

fprintf('el maximo del modulo de la derivada se encuentra en (x,y)= (%6.6g , %6.6g )\n', posx, abs\_max)

if error==0 % si error=0 significa que el usuario quiere usar una cantidad de intervalos definida por el

n=N;

h=(b-a)/n;

else

h=((error\*12)/(abs\_max\*(b-a)))^(1/2);

n=(b-a)/h;

n=round(n);

h=(b-a)/n;

endif

k=linspace(a,b,n+1);

for i=1:(n+1)

Fk(i)=feval(F,k(i)); % evaluo la funcion en n+1 puntos= n intervalos

endfor

S=0;

for i=2:(n) % sumo todos los puntos menos los correspondientes a a y b

S=S+Fk(i);

endfor

resultado=((h/2)\*(Fk(1)+Fk(n+1)+2\*S));

Et=((h^2)\*(b-a)\*abs\_max)/12;

h,n

format shortEng

Et

format long

resultado

endfunction

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function []=Trapecios\_compuesto\_puntos(X1, Y1)

%recordar que el paso (h) debe ser "constante"

n=length(X1)-1;

h=(X1(length(X1))-X1(1))/n;

S=0;

for i=2:n % sumo todos los puntos menos los correspondientes a a y b

S=S+Y1(i);

endfor

resultado=((h/2)\*(Y1(1)+Y1(n+1)+2\*S))

h,n

endfunction

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function []=Simpson\_13\_funcion(f,a,b,error,N)

%para un n determinado, ingresar error=0

% si solo se sabe el error, ingresar cualquier valor de N distinto de 0

syms x

F=function\_handle(f)

fiv=diff(f,4);

Fiv=function\_handle(fiv)

% las siguientes lineas calculan el maximo de la derivada cuarta de f en modulo

if a!=0

abs\_max=abs(feval(Fiv,a));

posx=a;

else

abs\_max=abs(feval(Fiv,(a+0.0001)));

posx=a+0.0001;

endif

for i=(a+0.0001):0.0001:b

e1=abs(feval(Fiv,i));

if (abs\_max<e1)

abs\_max= e1;

posx=i;

endif

endfor

% se grafica la derivada cuarta y su valor maximo en modulo

hold on

fplot(Fiv, [a, b])

plot(posx,abs\_max,'.r','markersize',20);

title('derivada cuarta y su maximo (en valor absoluto)')

hold off

fprintf('el maximo de la derivada se encuentra en (x,y)= (%6.6g , %6.6g )\n', posx, abs\_max)

if error==0 % si error=0 significa que el usuario quiere usar una cantidad de intervalos definida por el

n=N;

h=(b-a)/n;

else % si error es distinto de 0, se busca un h que permita obtener el error deseado por el usuario.

h=((error\*180)/(abs\_max\*(b-a)))^(1/4);

n=(b-a)/h;

n=round(n);

P\_I=(-1)^n;

if P\_I==(-1)

n=n+1;

else

n=n+2;

endif

h=(b-a)/n;

endif

k=linspace(a,b,n+1); % se genera un vector con n+1 puntos (n es la cantidad de intervalos)

for i=1:(n+1)

Fk(i)=feval(F,k(i)); % se evalua la funcion en todos los puntos

endfor

S\_Par=0;

S\_Impar=0;

for i=1:(n/2)

S\_Impar=S\_Impar+Fk(2\*i);

endfor

for i=2:(n/2)

S\_Par=S\_Par+Fk((2\*(i-1))+1);

endfor

resultado=(h/3)\*(Fk(1)+Fk(n+1)+(4\*S\_Impar)+(2\*S\_Par));

Et=((h^4)\*(b-a)\*abs\_max)/180;

format

h,n

format shortEng

Et

format long

resultado

endfunction

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function []=Simpson\_13\_puntos(X1,Y1)

syms x

n=length(X1)-1;

P\_I=(-1)^n;

if P\_I==(-1)

disp('Es necesario un numero impar de datos, dado que el numero de intervalos debe ser par.')

else

h=(X1(length(X1))-X1(1))/n;

S\_Par=0;

S\_Impar=0;

for i=1:(n/2)

S\_Impar=S\_Impar+Y1(2\*i);

endfor

for i=2:(n/2)

S\_Par=S\_Par+Y1((2\*(i-1))+1);

endfor

resultado=(h/3)\*(Y1(1)+Y1(n+1)+(4\*S\_Impar)+(2\*S\_Par))

h,n

endif

endfunction

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function []=Simpson\_38\_funcion(f,a,b,error,N)

%para un n determinado, ingresar error=0

syms x

F=function\_handle(f)

fiv=diff(f,4);

Fiv=function\_handle(fiv)

if a!=0

abs\_max=abs(feval(Fiv,a));

posx=a;

else

abs\_max=abs(feval(Fiv,(a+0.0001)));

posx=a+0.0001;

endif

for i=(a+0.0001):0.0001:b

e1=abs(feval(Fiv,i));

if (abs\_max<e1)

abs\_max= e1;

posx=i;

endif

endfor

hold on

fplot(Fiv, [a, b])

plot(posx,abs\_max,'.r','markersize',20);

title('derivada cuarta y su maximo (en valor absoluto)')

hold off

fprintf('el maximo del modulo de la derivada se encuentra en (x,y)= (%6.6g , %6.6g )\n', posx, abs\_max)

if error==0

n=N;

h=(b-a)/n;

else

h=((error\*80)/(abs\_max\*(b-a)))^(1/4);

n=(b-a)/h;

n=round(n);

while rem(n,3)!=0

n=n+1;

endwhile

h=(b-a)/n;

endif

k=linspace(a,b,n+1);

for i=1:(n+1)

Fk(i)=feval(F,k(i));

endfor

S\_1=0;

S\_2=0;

S\_3=0;

for i=1:((n-3)/3)

S\_3=S\_3+Fk((3\*i)+1);

endfor

for i=0:((n-3)/3)

S\_1=S\_1+Fk((3\*i)+2);

endfor

for i=0:((n-3)/3)

S\_2=S\_2+Fk((3\*i)+3);

endfor

resultado=((3\*h)/8)\*(Fk(1)+Fk(n+1)+(3\*S\_1)+(3\*S\_2)+(2\*S\_3))

Et=((h^4)\*(b-a)\*abs\_max)/80;

h,n

format shortEng

Et

format long

resultado

endfunction

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [resultado]=Simpson\_38\_puntos(X1,Y1)

syms x

n=length(X1)-1;

if rem(n,3)!=0

disp('La cantidad de intervalos debe ser multiplo de 3')

disp('Ingresar (3\*3m)+1 puntos, m entero')

else

h=(X1(length(X1))-X1(1))/n;

S\_1=0;

S\_2=0;

S\_3=0;

for i=1:((n-3)/3)

S\_3=S\_3+Y1((3\*i)+1);

endfor

for i=0:((n-3)/3)

S\_1=S\_1+Y1((3\*i)+2);

endfor

for i=0:((n-3)/3)

S\_2=S\_2+Y1((3\*i)+3);

endfor

resultado=((3\*h)/8)\*(Y1(1)+Y1(n+1)+(3\*S\_1)+(3\*S\_2)+(2\*S\_3))

h,n

endif

endfunction

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function []=Romberg\_funcion(f,a,b,error,N)

%N representa el numero de subintervalos

%para un N determinado, ingresar error=0

syms x

F=function\_handle(f)

i=1;

h(i)=abs(b-a);

R(i,1)=(h(i)/2)\*(feval(F,a)+feval(F,b));

i=i+1;

if error~=0

Et=error+1;

while Et>error

for k=1:(2^(i-2))

Sum\_F(k)=feval(F,(a+((((2\*k)-1)/2)\*h(i-1))));

endfor

R(i,1)=0.5\*(R(i-1,1)+h(i-1)\*(sum(Sum\_F)));

Et=abs(R(i,1)-R(i-1,1));

h(i)=abs(b-a)/(2^(i-1));

i=i+1;

endwhile

n=i-1

format long g

R

else

for i=2:N

for k=1:(2^(i-2))

Sum\_F(k)=feval(F,(a+((((2\*k)-1)/2)\*h(i-1))));

endfor

R(i,1)=0.5\*(R(i-1,1)+h(i-1)\*(sum(Sum\_F)));

Et=abs(R(i,1)-R(i-1,1));

h(i)=abs(b-a)/(2^(i-1));

n=i+1;

endfor

n=N

format long g

R

endif

endfunction

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function []=Romberg\_puntos(X1,Y1)

n=log2(2\*length(X1)-2);

i=1;

if mod(n,1)==0

a=X1(1);

b=X1(length(X1));

h(i)=abs(b-a);

R(i,1)=(h(i)/2)\*(Y1(1)+Y1(length(Y1)));

for i=2:n

for k=1:(2^(i-2))

v\_aux=a+((((2\*k)-1)/2)\*h(i-1));

Sum\_F(k)=Y1(find(X1==v\_aux));

endfor

R(i,1)=0.5\*(R(i-1,1)+h(i-1)\*(sum(Sum\_F)));

Et=abs(R(i,1)-R(i-1,1));

h(i)=abs(b-a)/(2^(i-1));

n=i+1;

endfor

n

format long g

R

else

disp('Ingresar 2^(k-1)-1 puntos, k entero')

endif

endfunction

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function R = romberg\_Richardson\_funcion(f, a, b, max\_k)

% Compute the triangular extrapolation table for Romberg integration

% using the composite trapezoid rule, starting with h=b-a.

% f: function name SYMBOLIC

% a, b: lower and upper limits of integration

% max\_k: the number of extrapolation steps (= number of columns in R, plus one.)

% max\_k=0 will do no extrapolation.

R = zeros(max\_k+1);

for j=1:max\_k+1

R(j,1) = Trapecios\_compuesto\_funcion(f,a,b,0,2^(j-1));

endfor

for k=2:max\_k+1

for j=k:max\_k+1

R(j,k) = (4^(k-1)\*R(j,k-1)-R(j-1,k-1))/(4^(k-1)-1);

endfor

endfor

endfunction

por consola se llama a la siguiente función:

Simpson\_13\_puntos(X1,Y1)

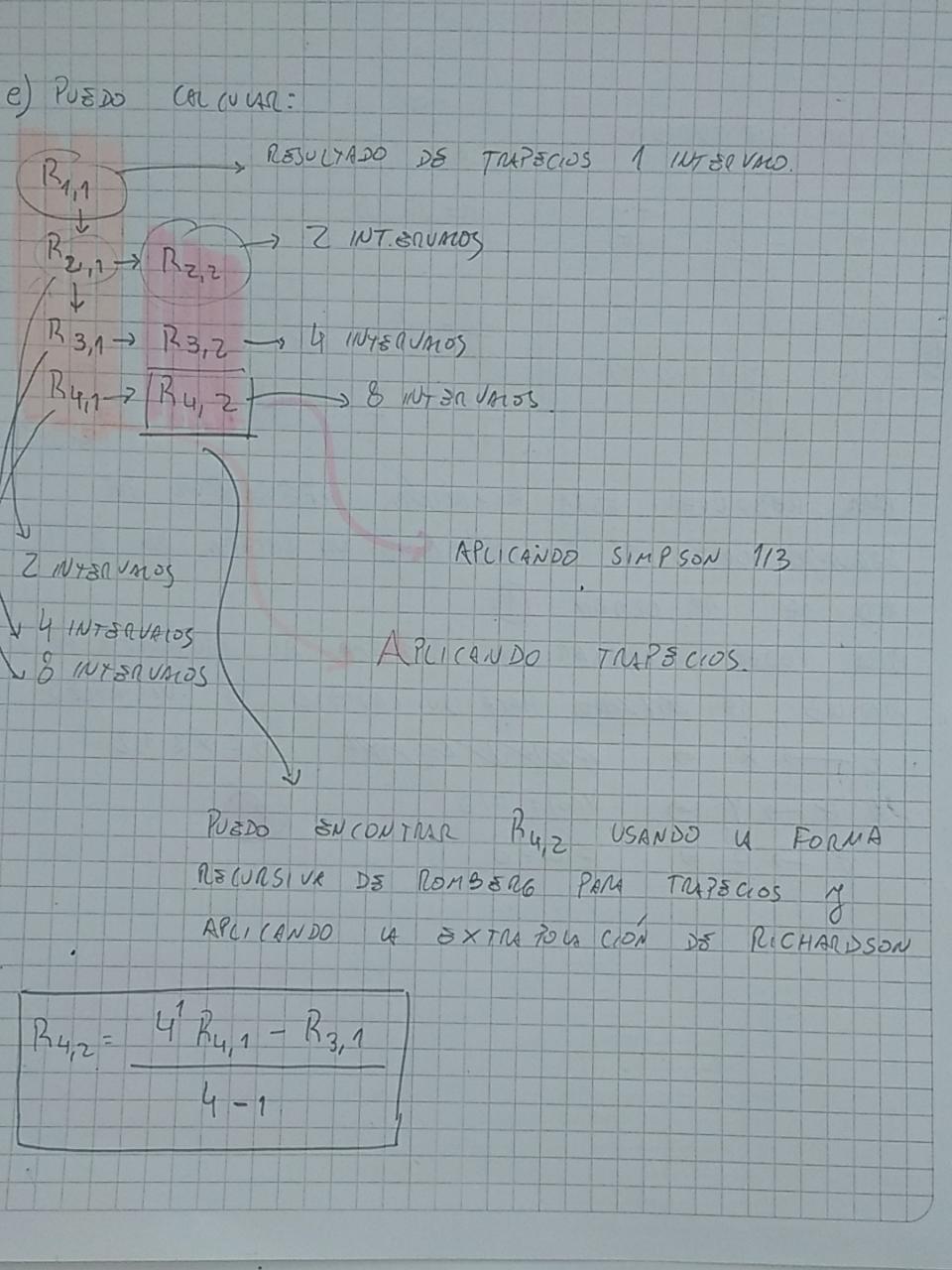
Devuelve por pantalla los siguientes valores:

Simpson\_13\_puntos(X1,Y1)

resultado = -3.599811836602551e-01

h = 1.500000000000000e-01

n = 8.000000000000000e+00

e) 

f) La ventaja de la integración adaptativa, es que en los meodos de newton-cotes, se necesitan nodos equiespaciados, se suelen usar incrementos h de manera uniforma en todo el intervalo de integración para garantizar una precisión global. El problema con lo previamente mencionado es que este procso no tiene en cuenta el hecho de que en algunas porciones de la curva puedan aparecer oscilaciones mas pronunciadas que en otras y, en consecuencia, requieran incrementos mas pequeños para conseguir la misma precisión.

Un método que aplica la integración adaptativa, va ajustando el incremento de manera que sea menor en aquellas porciones de la curva en la que aparezcan oscilaciones mas pronunciadas.

La desventaja que tiene la integración adaptativa, es que se necesita la función a integrar para ir especializándola en los ountos donde re requiera(cuando los métodos de newton cotes solo requieren el valor de la función en X0+h, X0+2\*h,…..)