Ejercicio 1:

1. El valor exacto de la integral es 0.828427 y lo he obtenido con los comandos:

a=2; b=3;

f=@(x) 1./sqrt(x-1);

Ianalitico=integral(f,a,b)

1. El valor aproximado con el método del punto medio es 0.828153 y lo he obtenido con los comandos:

npm=12

Ipm=puntoMedio(f,a,b,npm)

El error exacto cometido es 2.739923e-04 y lo he calculado con:

Error=abs(Ianalitico-Ipm)

1. La función implementada es:

function I= Simpson38 (f,a,b,n)

% I=simpson(f,a,b,n) obtiene la integral de f(x)

% con la fórmula de Simpson compuesta.

h = (b-a)/n;

x = a:h:b;

pesos = ones (1,n +1) ;

pesos (2:3: n) = 3; pesos (3:3: n) = 3; pesos (4:3:n -1) = 2;

I=3\*h /8\* sum ( pesos.\*f(x));

End

1. El valor aproximado con el método de Simpson 3/8 es 0.828428 y lo he obtenido con los comandos:

Ns=12

Is=Simpson38(f,a,b,Ns)

El error exacto cometido es 1.003101e-06 y lo he calculado con:

Error=abs(Ianalitico-Is)

Ejercicio 2:

1. Vamos a transformar a un problema de Valor inicial:

Con , y

Esto implementado queda:

function dY = PVI(t,y)

Y1=y(1); Y2=y(2); Y3=y(3);

dY=[Y2;Y3;...

-t.\*Y1+18+4.\*t.^2.\*Y2+t.\*exp(t.^2)-33\*t.^4];

end

1. Para resolver este apartado, utilizo los comandos:

a=1;b=2;N=20; Ya=[exp(1)+3;2\*exp(1)+9;4\*exp(1)+18];

[t,Y]=RK\_Sistemas('PVI',a,b,N,Ya)

Con esto obtengo la gráfica: 

Y los valores:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 1.25 | 1.5 | 1.75 | 2 |
| y(t) | 5.718282 | 10.134144 | 15.544367 | 20.537543 | 23.601962 |
| y'(t) | 14.436564 | 20.393349 | 21.914460 | 16.923501 | 7.323979 |

1. Para resolver este apartado, utilizo los comandos:

a=1;b=2;N=20; Ya=[exp(1)+3;2\*exp(1)+9;4\*exp(1)+18];

[t2,Y2]=AB2Sist('PVI',a,b,N,Ya)

Con esto obtengo la gráfica: 

Y los valores:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 1.25 | 1.5 | 1.75 | 2 |
| y(t) | 5.718282 | 10.335165 | 17.649862 | 29.508927 | 50.523076 |
| y’(t) | 14.436564 | 23.145531 | 36.728804 | 61.451242 | 116.018184 |

1. Lo que voy a hacer es calcular sucesiones iterativas {N,2N,4N,…} para la función de Runge-Kutta. Con esto calculo el error mediante el máximo del absoluto de la diferencia entre dos soluciones consecutivas. Sobre esto aplico log2 sobre el cociente de dos errores consecutivos viendo la secuencia [3.885025 3.942620 3.971343 3.985669 3.99283 3.992530], que obviamente converge a 4 que es el orden de Runge-Kutta. Esto lo he programado con:

[t,z1]=RK\_Sistemas('PVI',a,b,N,Ya);

[t,z2]=RK\_Sistemas('PVI',a,b,2\*N,Ya);

[t,z3]=RK\_Sistemas('PVI',a,b,4\*N,Ya);

[t,z4]=RK\_Sistemas('PVI',a,b,8\*N,Ya);

[t,z5]=RK\_Sistemas('PVI',a,b,16\*N,Ya);

[t,z6]=RK\_Sistemas('PVI',a,b,32\*N,Ya);

[t,z7]=RK\_Sistemas('PVI',a,b,64\*N,Ya);

[t,z8]=RK\_Sistemas('PVI',a,b,2\*64\*N,Ya);

E1=max(abs(z1(:,1)-z2(1:2:end,1)));

E2=max(abs(z2(:,1)-z3(1:2:end,1)));

E3=max(abs(z3(:,1)-z4(1:2:end,1)));

E4=max(abs(z4(:,1)-z5(1:2:end,1)));

E5=max(abs(z5(:,1)-z6(1:2:end,1)));

E6=max(abs(z6(:,1)-z7(1:2:end,1)));

E7=max(abs(z7(:,1)-z8(1:2:end,1)));

E=[E1 E2 E3 E4 E5 E6 E7];

orden=log2(E(1:end-1)./E(2:end))

Análogamente lo hacemos sobre Adams-Bashfoth, obteniendo la secuencia [1.812373 1.9190185 1.963408 1.982769 1.991662 1.995901], viendo que converge a 2 que es el orden de este método. Esto lo he programado con:

[t,z1]=AB2Sist('PVI',a,b,N,Ya);

[t,z2]=AB2Sist('PVI',a,b,2\*N,Ya);

[t,z3]=AB2Sist('PVI',a,b,4\*N,Ya);

[t,z4]=AB2Sist('PVI',a,b,8\*N,Ya);

[t,z5]=AB2Sist('PVI',a,b,16\*N,Ya);

[t,z6]=AB2Sist('PVI',a,b,32\*N,Ya);

[t,z7]=AB2Sist('PVI',a,b,64\*N,Ya);

[t,z8]=AB2Sist('PVI',a,b,2\*64\*N,Ya);

E1=max(abs(z1(:,1)-z2(1:2:end,1)));

E2=max(abs(z2(:,1)-z3(1:2:end,1)));

E3=max(abs(z3(:,1)-z4(1:2:end,1)));

E4=max(abs(z4(:,1)-z5(1:2:end,1)));

E5=max(abs(z5(:,1)-z6(1:2:end,1)));

E6=max(abs(z6(:,1)-z7(1:2:end,1)));

E7=max(abs(z7(:,1)-z8(1:2:end,1)));

E=[E1 E2 E3 E4 E5 E6 E7];

orden=log2(E(1:end-1)./E(2:end))

Ejercicio 3:

1. Si representamos ambos lados de la igualdad como función obtenemos:



Esto nos da el corte en x=1. 46465.

1. Para este apartado voy a adaptar el programa que ya tengo a las condiciones de parada que me piden (iter=iter1+abs(fx)). Implementando el código

[sol,iter,~,~,dif,ACOC] =TraubNL('Examen3',vpa(1.1) ,1e-10 ,50)

Obtengo la solución 1.468598, con 11 iteraciones, una diferencia entre pasos de 3.857218e-25 y un ACOC de [-0.405017, -1.487217, -0.6278722, 0.734183, 1.96662, 2.59011, 2.933893, 2.999596] viendo que converge a 3.

1. Voy a implementar el método de Ostrowski con Matlab:

function [sol ,iter, incre, f\_eval, dif , ACOC ] =Ostrowski(f,x0 ,tol , maxiter )

digits (200);

% Inicializacion de las variables

iter =1;

[fx, dfx]=feval(f,x0);

% Criterio de parada

incre = 1 + tol ;

while incre > tol && iter < maxiter

% Expresion del metodo de Newton

y=x0 -fx/ dfx ;

[fy, dfy]=feval(f,y);

x=y-(fx/(fx-2\*fy))\*(fy/dfx);

incre1 = abs(x-x0);

I(iter)= incre1;

% Actualizacion de la estimacion inicial

dif=(x-x0);

x0=x;

[fx , dfx ]= feval(f,x0);

% Incremento del contador de iteraciones

iter = iter +1;

incre=incre1+abs(fx);

end

if length(I) >2

sol =vpa(x,7);

incre1=vpa(incre1,7);

f\_eval=vpa(abs(fx),7);

dif=vpa(abs(dif),7);

ACOC = vpa(log(I(3 :end)./I(2 :end -1))./ log(I(2 :end -1)./I(1 :end -2)),7);

else

disp ('necesito mas iteraciones ')

end

end

1. Para este apartado utilizo el código que acabo de crear. Implementando el código

[sol,iter,~,~,dif,ACOC]=Ostrowski('Examen3',vpa(1.1) ,1e-10 ,50)

Obtengo la solución 1.468598, con 5 iteraciones, una diferencia entre pasos de 9.160999e-27 y un ACOC de [4.303699, 3.993669] viendo que converge a 4.

1. Vemos que ambos métodos aproximan al mismo valor 1.468598, que es prácticamente el mismo que el que hemos obtenido gráficamente en el apartado a), que era 1. 46465. Además, vemos que el método de Traub consigue esta solución con 11 interaciones, mientras que Ostrowski lo hace con 5. Esto se debe a que el primer método tiene orden de convergencia 3 y el segundo tiene orden de convergencia 4, que es lo que obtenemos por el ACOC. Así mismo, el método de mayor orden también obtiene un error entre valores consecutivos mucho menor que Traub.  
   Además, ambos son métodos No lineales sin memoria y multipunto