TECNOLÓGICO NACIONAL DE MÉXICO

I.T.E.

ENSENADA, B.C.

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Proyecto Final

Maestro: Dr. Cecil Vidriales

Métodos Numéricos

Descripción:

Los métodos numéricos son técnicas matemáticas que permiten resolver problemas que son difíciles o imposibles de resolver de manera analítica. Estas técnicas son esenciales en campos como la ingeniería, la física, la economía y las ciencias computacionales. Proveen soluciones aproximadas a ecuaciones diferenciales, integrales y otros problemas matemáticos complejos.

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

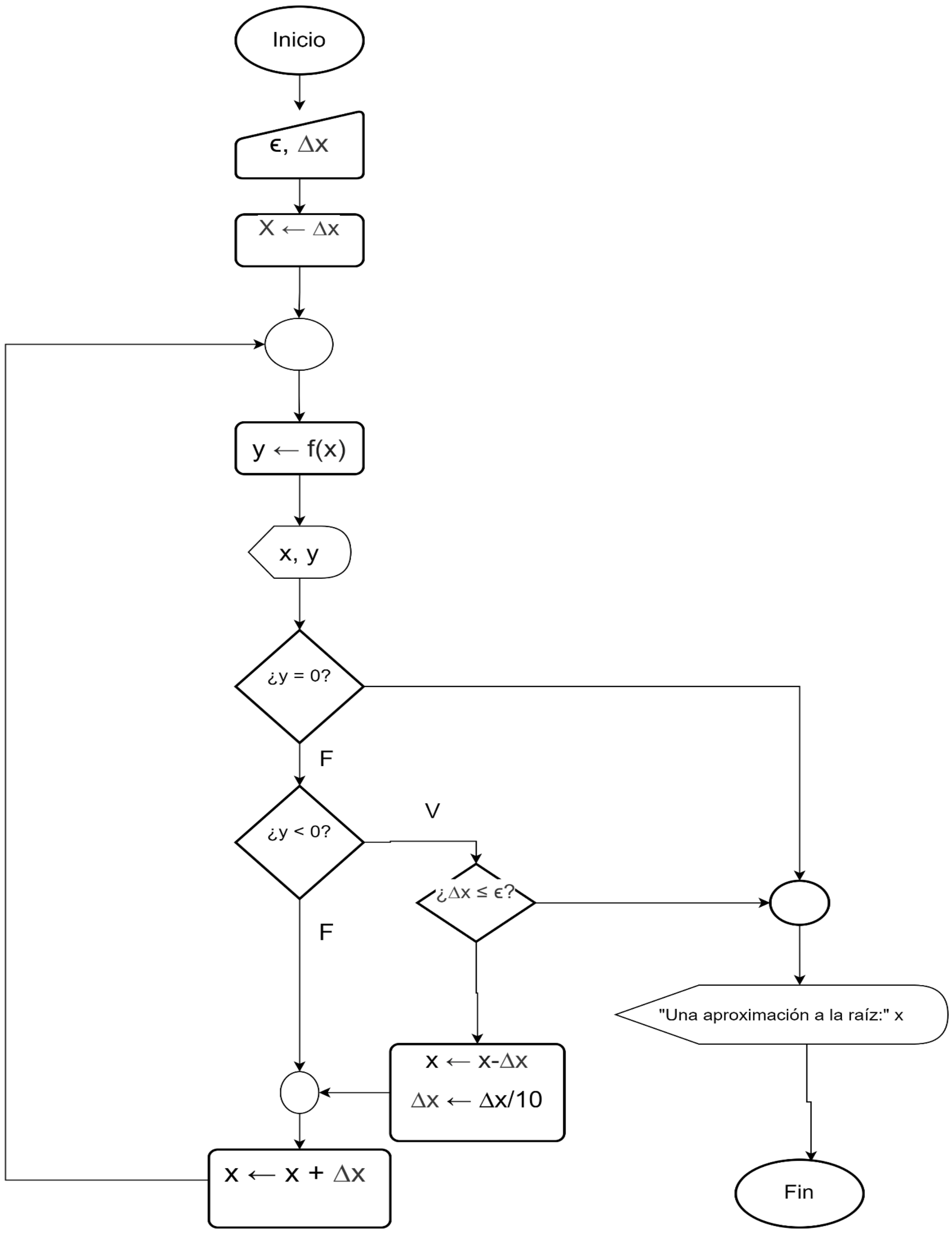
Método Tanteos

Maestro: Dr. Cecil Vidriales

Método de Tanteos

Descripción: El Método de Tanteos es un método iterativo de búsqueda que aproxima la raíz de una ecuación probando diferentes valores.

Utilidad: Es útil para obtener una solución rápida cuando se conoce el rango en el que se encuentra la raíz.



pseudocódigo del Método de Tanteos

1. Inicio
2. Entrada:
    - Función f(x) para la cual queremos encontrar la raíz
    - Tolerancia ε para la aproximación de la raíz
    - Incremento Δx para ajustar los valores iniciales
    - Número máximo de iteraciones NMI
3. Proceso:
    - k = 0: Inicializar el contador de iteraciones
    - x0 = 0: Establecer el primer valor inicial como 0
    - x1 = Δx: Establecer el segundo valor inicial como el incremento Δx
    - Mientras (k < NMI):
        - f0 = f(x0)
        - f1 = f(x1)
        - Si |f0| < ε o |x1 - x0| < ε entonces:
            - Salida: x0 es una aproximación de la raíz
        - Si no:
            - Si f0 * f1 < 0 entonces:
                - Δx = |x1 - x0| / 2: Reducir el incremento a la mitad
                - x1 = x0 + Δx: Actualizar el segundo valor inicial
            - Si no:
                - x0 = x1: Actualizar el primer valor inicial como el segundo valor inicial
                - x1 = x1 + Δx: Actualizar el segundo valor inicial sumando el incremento
        - k = k + 1: Incrementar el contador de iteraciones
    - Fin Mientras
4. Salida:
    - Si (|f0| < ε o |x1 - x0| < ε) entonces:
        - Imprimir "El método de tanteos encontró una aproximación de la raíz después de iteraciones."
        - Imprimir "Aproximación de la raíz: x = valor"
    - Si no, entonces:
        - Imprimir "El método de tanteos no converge a una solución dentro del número máximo de iteraciones."
5. Fin

Código:

```
double delta, e, x, y;
    printf("Ingrese el valor del salto: ");
    scanf("%lf", &delta);
    printf("Ingrese el valor del error: ");
    scanf("%lf", &e);

    x = delta;
    for(int i = 0; i < 10000; i++){
```

```
    y = fun(x);
    if(y == 0){
        printf("%lf es un aproximado a la raiz", x);
        return;
    }else{
        if(y < 0){
            if(delta <= e){
                printf("%lf es un aproximado a la raiz", x);
                return;
            }else{
                x = x - delta;
                delta = delta / 10;
            }
        }
        x = x + delta;
    }
}
```

Prueba de escritorio
Ingrese el valor del salto: .1
Ingrese el valor del error: 0.001
Iteracion 1:
x = 0.100000
y = -1.990000
delta antes de la actualizacion: 0.100000
x despues de restar delta: 0.000000
delta despues de dividir por 10: 0.010000
x despues de sumar delta: 0.010000
Iteracion 2:
x = 0.010000
y = -1.999900
delta antes de la actualizacion: 0.010000
x despues de restar delta: 0.000000
delta despues de dividir por 10: 0.001000
x despues de sumar delta: 0.001000
Iteracion 3:
x = 0.001000
y = -1.999999
0.001000 es un aproximado a la raiz

```
Raices de ecuaciones
Elige el metodo a usar
1.Metodo de tanteos
2.Metodo de punto fijo
3.Metodo de Newton-Raphson
4.Metodo de Newton 2do orden
5.Metodo de Biseccion
6.Metodo de Falsa Posicion(cerrado)
7.Metodo Secante (Hibrido)
8.Volver
1
Metodo de tanteos
Ingrese el valor del salto: 1
Ingrese el valor del error: 1
Se encontro una raiz aproximada: 2.000000
El tiempo de ejecucion fue: 0.000000 segundos
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

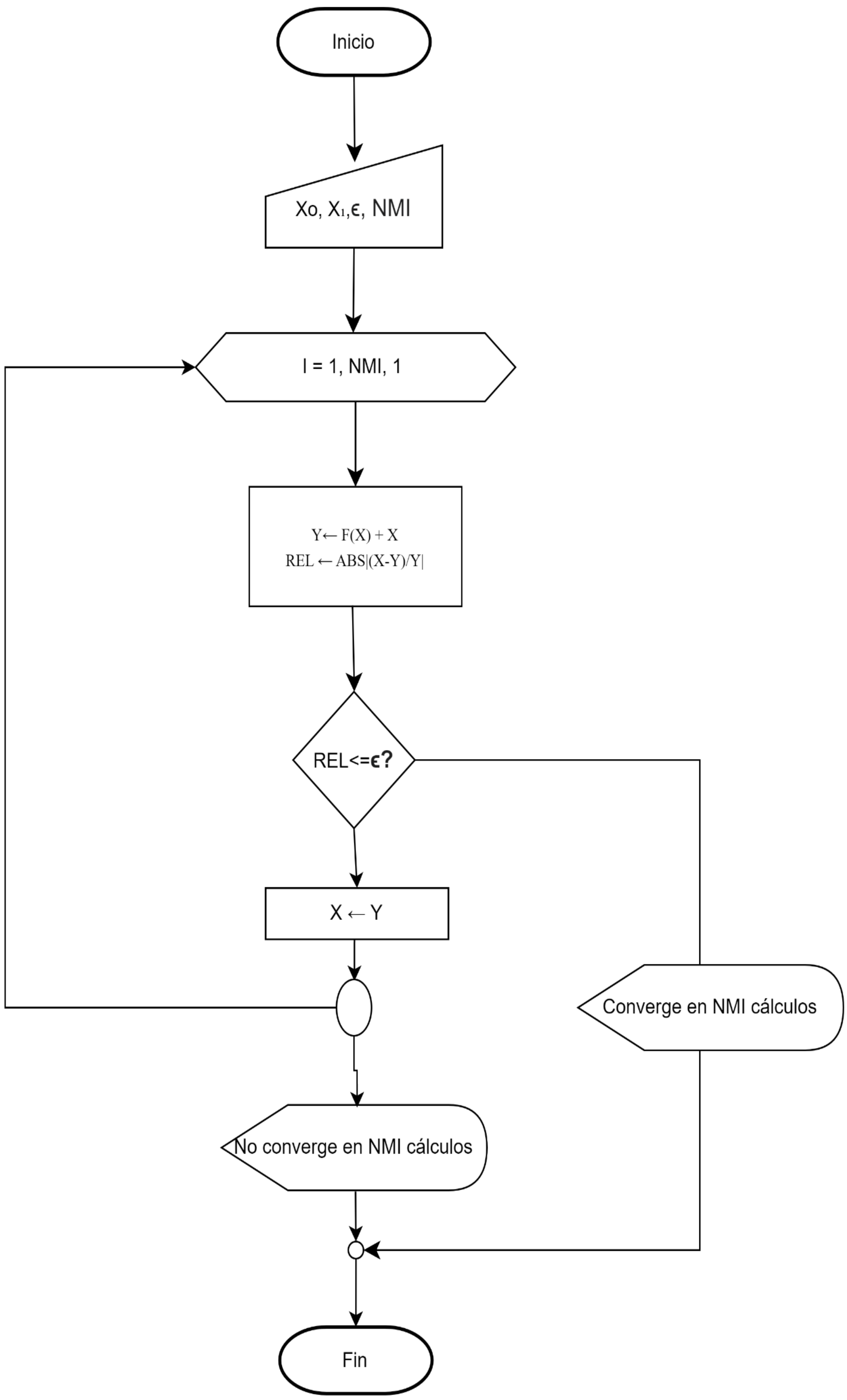
Método de Punto Fijo

Maestro: Dr. Cecil Vidriales

Método de Punto Fijo

Descripción: El Método de Punto Fijo transforma una ecuación en una función iterativa que converge hacia una solución.

Utilidad: Se utiliza para resolver ecuaciones no lineales, pero requiere una buena elección de la función iterativa para converger.



Pseudocódigo del Método del Punto Fijo

```
Inicio
    Leer X0, X1, ε, NMI
    I ← 1
    Repetir desde I = 1 hasta NMI con incremento de 1
        Y ← F(X0) + X0
        REL ← ABS(X0 - Y) / Y
        Si REL ≤ ε Entonces
            Imprimir "Converge en ", I, " cálculos"
            Terminar
        FinSi
        X0 ← Y
    FinRepetir
    Imprimir "No converge en NMI cálculos"
Fin
```

Código:

```
double nmi, e,x,y,rel;

printf("Ingresa el valor de X:");

scanf("%lf", &x);

printf("Ingresa el valor del error:");

scanf("%lf", &e );

printf("Ingresa el numero de iteraciones:");

scanf("%lf", &nmi );

for(int i=0;i<=nmi;i++){

    printf("entro %lf \n", x);

    y=fun(x) + x;

    rel = (x-y)/y;

    if(rel<0)

        rel*=-1;

    printf("y=  %lf \n rel= %lf", y, rel);

    if(rel<=e){

        printf("si converge en %lf iteraciones, y= %lf", nmi, y);

        return;

    }else{

        x=y;

    }

}

printf("No converge en %lf iteraciones", nmi);

return;
```

Prueba de escritorio:

Ingresa el valor de X:0.5

Ingresa el valor del error:0.001

Ingresa el numero de iteraciones:10

i = 0,10

y ← 〖0.500000〗^2- 0.5 + (0.500000)= 0.250000

rel ←|(0.500000 - 0.250000)/0.250000|= 1.000000

rel ≤ e? Falso

x ←y

i = 1,10

y ← 〖0.250000〗^2- 0.5 + (0.250000)= -0.187500

rel ←|(0.250000 - -0.187500)/(-0.187500)|= 2.333333

rel ≤ e? Falso

x ←y

i = 2,10

y ←- 〖0.187500〗^2- 0.5 + (-0.187500)= -0.652344

rel ←|(-0.187500 - -0.652344)/(-0.652344)|= 0.712575

rel ≤ e? Falso

x ←y

i = 3,10

y ←- 〖0.652344〗^2- 0.5 + (-0.652344)= -0.726791

rel ←|(-0.652344 - -0.726791)/(-0.726791)|= 0.102433

rel ≤ e? Falso

x ←y

i = 4,10

y ←- 〖0.726791〗 ^2- 0.5 + (-0.726791)= -0.698566

rel ←|(-0.726791 - -0.698566)/(-0.698566)|= 0.040405

rel ≤ e? Falso

x ←y

i = 5,10

y ←- 〖0.698566〗 ^2- 0.5 + (-0.698566)= -0.710572

rel ←|(-0.698566 - -0.710572)/(-0.710572)|= 0.016896

rel ≤ e? Falso

x ←y

i = 6,10

y ←- 〖0.710572〗 ^2- 0.5 + (-0.710572)= -0.705660

rel ←|(-0.710572 - -0.705660)/(-0.705660)|= 0.006961

rel ≤ e? Falso

x ←y

i = 7,10

y ←- 〖0.705660〗 ^2- 0.5 + (-0.705660)= -0.707704

rel ←|(-0.705660 - -0.707704)/(-0.707704)|= 0.002889

rel ≤ e? Falso

x ←y

i = 8,10

y ←- 〖0.707704〗 ^2- 0.5 + (-0.707704)= -0.706859

rel ←|(-0.707704 - -0.706859)/(-0.706859)|= 0.001196

rel ≤ e? Falso

x ←y

i = 9,10

y ←- 〖0.706859〗^2- 0.5 + (-0.706859)= -0.707209

rel ←|(-0.706859 - -0.707209)/(-0.707209)|= 0.000495

rel ≤ e? Verdadero

si converge en 10 iteraciones,y= -0.707209

Ejecución:

```
Metodo de punto fijo
Ingresa el valor de X:0.5
Ingresa el valor del error:0.001
Ingresa el numero de iteraciones:10
entro 0.500000
y= 0.250000
rel= 1.000000
entro 0.250000
y= -0.187500
rel= 2.333333
entro -0.187500
y= -0.652344
rel= 0.712575
entro -0.652344
y= -0.726791
rel= 0.102433
entro -0.726791
y= -0.698566
rel= 0.040405
entro -0.698566
y= -0.710572
rel= 0.016896
entro -0.710572
y= -0.705660
rel= 0.006961
entro -0.705660
y= -0.707704
rel= 0.002889
entro -0.707704
y= -0.706859
rel= 0.001196
entro -0.706859
y= -0.707209
rel= 0.000495
Tiempo de ejecucion: 0.003000 segundos
Si converge en 10.000000 iteraciones, y= -0.707209
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

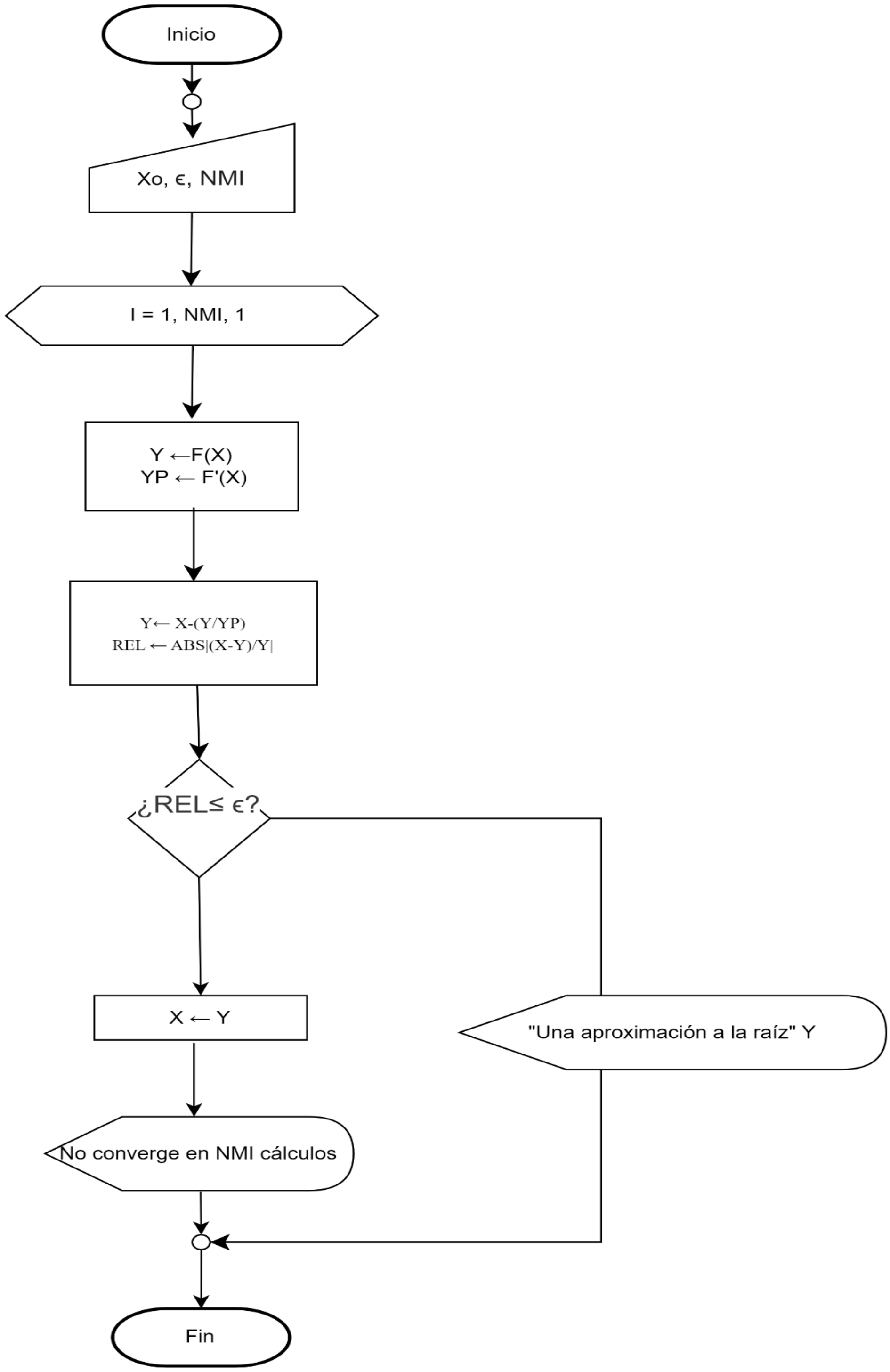
Método de Newton-Raphson

Maestro: Dr. Cecil Vidriales

Método de Newton-Raphson

Descripción: El Método de Newton-Raphson es un método iterativo que usa la derivada de la función para encontrar raíces.

Utilidad: Es muy eficiente y converge rápidamente si se está cerca de la raíz, adecuado para problemas donde se requiere alta precisión.



Pseudocódigo del Método de Newton-Raphson

Inicio

Entrada:

- Función f(x) para la cual queremos encontrar la raíz
- Valor inicial de tanteo x0
- Tolerancia ε para la aproximación de la raíz
- Número máximo de iteraciones NMI

Proceso: Para i de 1 a NMI con paso 1:

- y = f(x): Calcular el valor de f(x)
- yp = f'(x): Calcular el valor de la derivada de f(x)
- x = x - (y / yp): Actualizar x usando la fórmula de Newton-Raphson
- REL = abs((x - y) / y): Calcular el error relativo
- Si REL <= ε, detenerse y salir del bucle.

Salida:

- Si el algoritmo converge (es decir, REL <= ε), imprimir el valor aproximado de la raíz.
- Si el algoritmo no converge dentro del número máximo de iteraciones, imprimir un mensaje indicando que no se encontró una solución.

Fin

Código:

```
int nmi;
double y, yp, rel, x, e;
printf("Ingresa el valor de X: ");
scanf("%lf", &x);
printf("Ingresa el valor del error: ");
scanf("%lf", &e);
printf("Ingresa el numero de iteraciones: ");
scanf("%d", &nmi);
for(int i=0;i<=nmi;i++){
	//printf("x= %lf\n y= %lf\n rel=%lf\n", x, y, rel);
	y=fun(x);
	yp=fun2(x);
	y=x-(y/yp);
	rel = (y-x)/y;
	if(rel<0)
		rel*=-1;
	if(rel<=e){
		printf("Una aproximacion a la raiz %lf",y);
		return;
	}else{
		x=y;
	}
}
printf("No convcergue en %d calculos", nmi);
return;
```

Prueba de escritorio:

Ingresa el valor de X: 0

Ingresa el valor del error: 0.001

Ingresa el numero de iteraciones: 10

Iteracion 0:

Calculando f(x):

f(x)= -3.000000

Calculando f^' (x):

f^' (x)= 12.000000

Calculando el nuevo x:

x_new= x -f(x)/(f^' (x))= 0.000000 - (-3.000000/12.000000)= 0.250000

Calculando la relacion de convergencia:

rel = |(x_new- x)/x_new |= |(0.250000 - 0.000000)/0.250000|= 1.000000

Verificando si rel ≤ e:

Falso

Actualizando x: x ←x_new

Iteracion 1:

Calculando f(x):

f(x)= -0.531250

Calculando f^' (x):

f^' (x)= 7.875000

Calculando el nuevo x:

x_new= x -f(x)/(f^' (x))= 0.250000 - (-0.531250/7.875000)= 0.317460

Calculando la relacion de convergencia:

rel = |(x_new- x)/x_new |= |(0.317460 - 0.250000)/0.317460|= 0.212500

Verificando si rel ≤ e:

Falso

Actualizando x: x ←x_new

Iteracion 2:

Calculando f(x):

f(x)= -0.033518

Calculando f^' (x):

f^' (x)= 6.890401

Calculando el nuevo x:

x_new= x -f(x)/(f^' (x))= 0.317460 - (-0.033518/6.890401)= 0.322325

Calculando la relacion de convergencia:

rel = |(x_new- x)/x_new |= |(0.322325 - 0.317460)/0.322325|= 0.015092

Verificando si rel ≤ e:

Falso

Actualizando x: x ←x_new

Iteracion 3:

Calculando f(x):

f(x)= -0.000168

Calculando f^' (x):

f^' (x)= 6.821514

Calculando el nuevo x:

x_new= x -f(x)/(f^' (x))= 0.322325 - (-0.000168/6.821514)= 0.322349

Calculando la relacion de convergencia:

rel = |(x_new- x)/x_new |= |(0.322349 - 0.322325)/0.322349|= 0.000076

Verificando si rel ≤ e:

Verdadero

Una aproximacion a la raiz es 0.322349

Ejecución:

```
Raices de ecuaciones
Elige el metodo a usar
1.Metodo de tanteos
2.Metodo de punto fijo
3.Metodo de Newton-Raphson
4.Metodo de Newton 2do orden
5.Metodo de Biseccion
6.Metodo de Falsa Posicion(cerrado)
7.Metodo Secante (Hibrido)
8.Volver
3
Metodo de Newton-Raphson
Ingresa el valor de X: 0
Ingresa el valor del error: 0.001
Ingresa el numero de iteraciones: 10
Tiempo de ejecucion: 0.000000 segundos
Una aproximacion a la raiz 0.322349
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

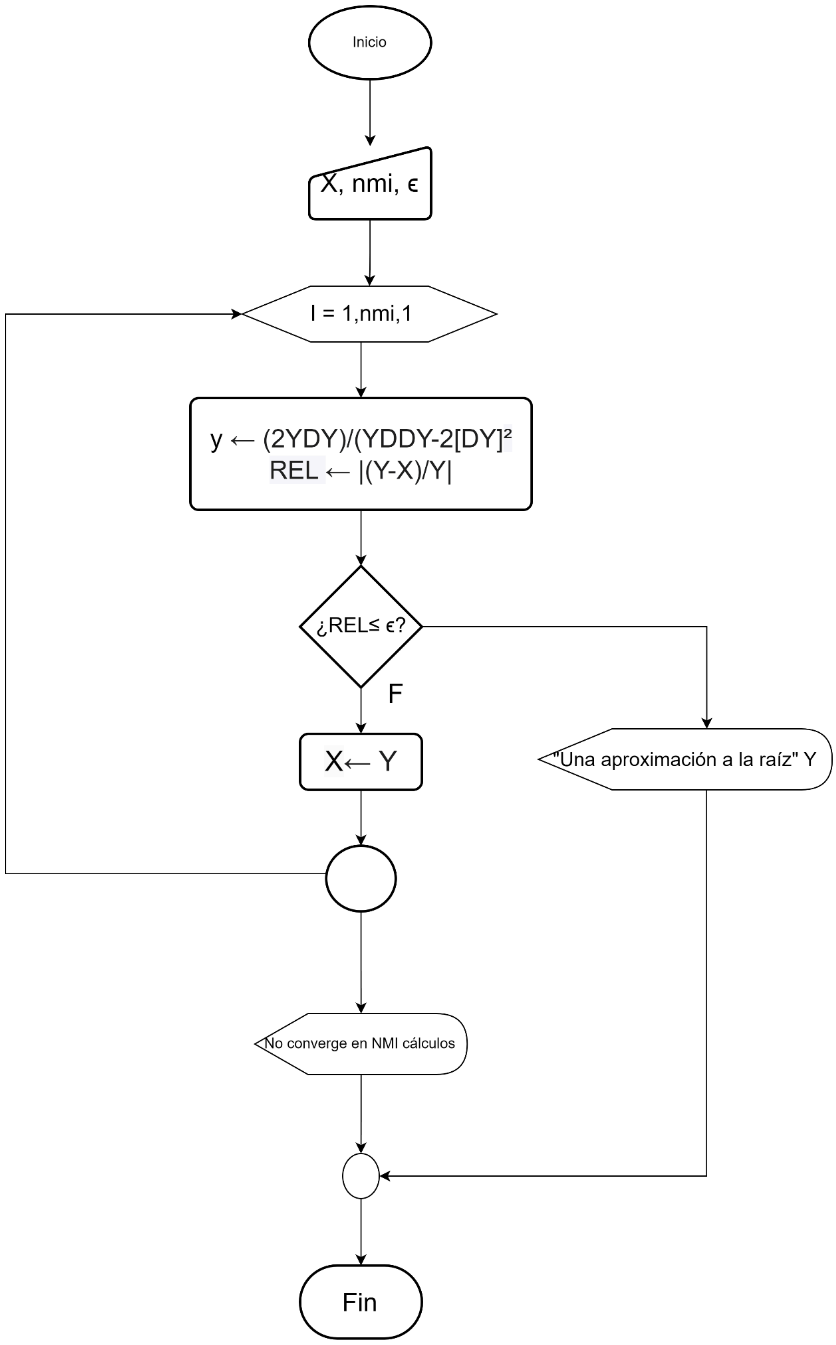
Método de Newton segundo orden

Maestro: Dr. Cecil Vidriales

Método de Newton de Segundo Orden

Descripción: Una extensión del Método de Newton-Raphson que utiliza derivadas de segundo orden para mejorar la convergencia.

Utilidad: Ofrece mayor precisión y una convergencia más rápida en la solución de ecuaciones no lineales.



Pseudocodigo de newton segundo orden

Inicio

Entrada:

- Función f(x) para la cual queremos encontrar la raíz
- Derivada f'(x) de la función f(x)
- Valor inicial de tanteo x0
- Tolerancia ε para la aproximación de la raíz
- Número máximo de iteraciones NMI

Proceso: Para i de 1 a NMI con paso 1:

- y = f(x): Calcular el valor de f(x)
- yp = f'(x): Calcular el valor de la derivada de f(x)
- ypp = f''(x): Calcular el valor de la segunda derivada de f(x)
- x = x - (y * yp) / (yp^2 - y * ypp): Actualizar x usando la fórmula de Newton-Raphson de segundo orden
- REL = abs((x - y) / y): Calcular el error relativo
- Si REL <= ε, detenerse y salir del bucle.

Salida:

- Si el algoritmo converge (es decir, REL <= ε), imprimir el valor aproximado de la raíz.
- Si el algoritmo no converge dentro del número máximo de iteraciones, imprimir un mensaje indicando que no se encontró una solución.

Fin

Código:

```
int delta, nmi;

double e,x,y,yp,rel;

printf("Ingresa el valor del X: ");

scanf("%lf", &x);

printf("Ingresa el valor del error: ");

scanf("%lf", &e);

printf("Ingresa el numero de iteraciones: ");

scanf("%d", &nmi);


for(int i=0;i<=nmi;i++){

    y= x - ( (fun(x)*fun2(x)) / ((fun2(x)*fun2(x) )-(fun(x)*fun3(x))  ));

    rel=(y-x)/y;

    printf("x= %lf\n y= %lf\n rel=%lf\n", x, y, rel);

    if(rel<0)

        rel*=-1;

    if(rel<=e){

        printf("Una aproximacion a la raiz %lf",y);

        return;

    }else{

        x=y;

    }

}

printf("No convergue en %d iteraciones", nmi);

return;
```

Prueba de escritorio

Ingresa el valor de X: 0

Ingresa el valor del error: 0.001

Ingresa el numero de iteraciones: 10

Iteracion 0:

x = 0.000000

f(x)= -3.000000

f^' (x)= 12.000000

f^'' (x)= -18.000000

Operacion de y = x - ((f(x)* f^' (x))/((f^' (x)* f^' (x))- (f(x)* f^'' (x)))):

y = 0.400000

rel = 1.000000

Iteracion 1:

x = 0.400000

f(x)= 0.488000

f^' (x)= 5.760000

f^'' (x)= -13.200000

Operacion de y = x - ((f(x)* f^' (x))/((f^' (x)* f^' (x))- (f(x)* f^'' (x)))):

y = 0.329053

rel = -0.215611

Iteracion 2:

x = 0.329053

f(x)= 0.045407

f^' (x)= 6.726707

f^'' (x)= -14.051369

Operacion de y = x - ((f(x)* f^' (x))/((f^' (x)* f^' (x))- (f(x)* f^'' (x)))):

y = 0.322396

rel = -0.020647

Iteracion 3:

x = 0.322396

f(x)= 0.000319

f^' (x)= 6.820505

f^'' (x)= -14.131246

Operacion de y = x - ((f(x)* f^' (x))/((f^' (x)* f^' (x))- (f(x)* f^'' (x)))):

y = 0.322349

rel = -0.000145

Una aproximacion a la raiz: 0.322349

Ejecución:

```
Raices de ecuaciones
Elige el metodo a usar
1.Metodo de tanteos
2.Metodo de punto fijo
3.Metodo de Newton-Raphson
4.Metodo de Newton 2do orden
5.Metodo de Biseccion
6.Metodo de Falsa Posicion(cerrado)
7.Metodo Secante (Hibrido)
8.Volver
4
Metodo de Newton 2do orden
Ingresa el valor del X: 0
Ingresa el valor del error: 0.001
Ingresa el numero de iteraciones: 10
x= 0.000000
 y= 0.400000
 rel=1.000000
x= 0.400000
 y= 0.329053
 rel=-0.215611
x= 0.329053
 y= 0.322396
 rel=-0.020647
x= 0.322396
 y= 0.322349
 rel=-0.000145
Tiempo de ejecucion: 0.000000 segundos
Una aproximacion a la raiz 0.322349
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

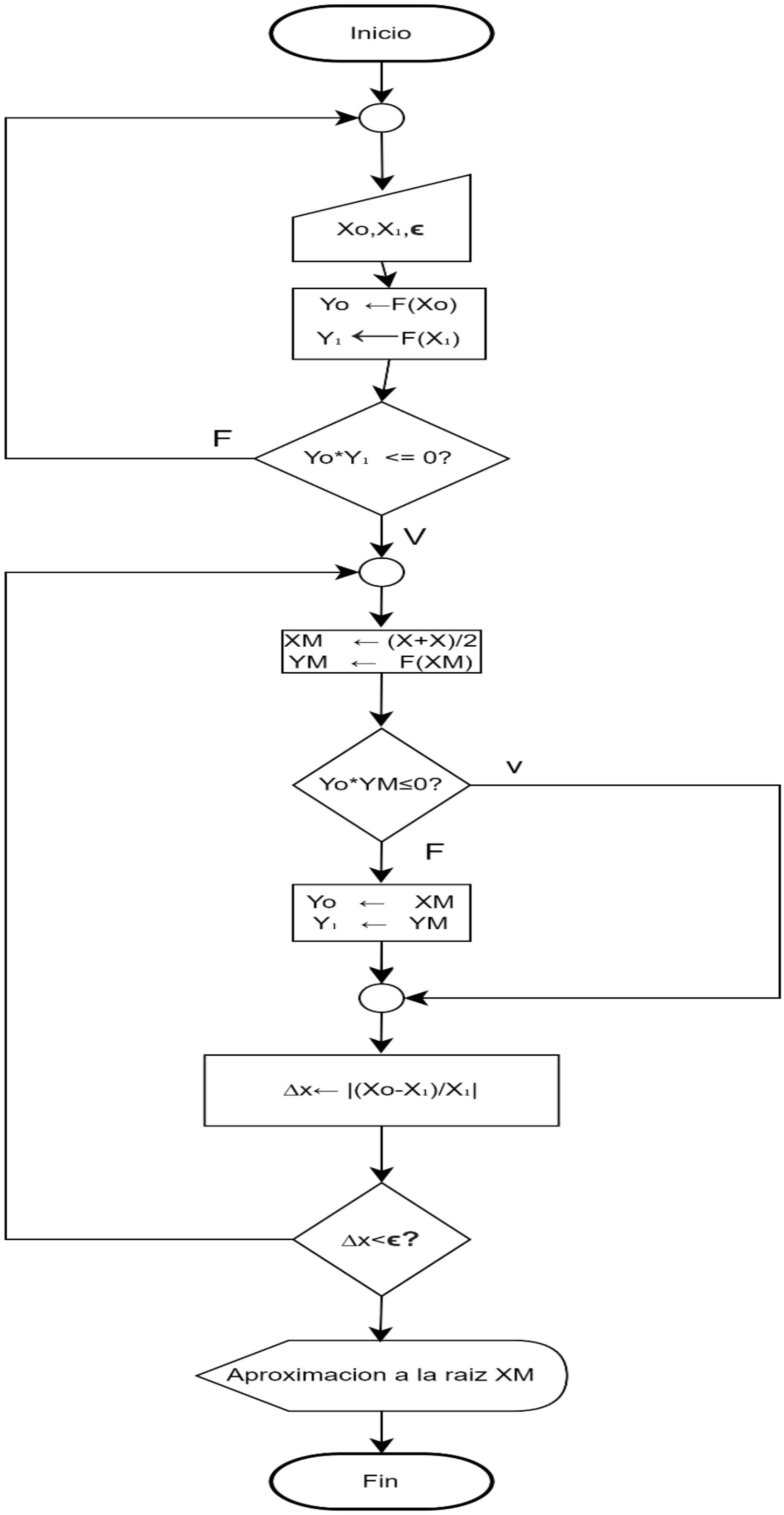
Método de Bisección

Maestro: Dr. Cecil Vidriales

Método de Bisección

Descripción: El Método de Bisección es un método iterativo que divide el intervalo de búsqueda en dos y selecciona el subintervalo donde cambia el signo de la función.

Utilidad: Es robusto y garantiza la convergencia, aunque es más lento que otros métodos.



Pseudocódigo de bisección

```
Inicio

Leer x0

Leer x1

Leer e

y0 = fun(x0)

y1 = fun(x1)

Si y0 * y1 >= 0:

    Imprimir "Para que funcione, el producto de la multiplicación de x0 y x1 debe ser mayor a 0, cambiar los valores"

    Llamar a la función bisección()

Sino:

    i = 0

    Repetir:

        xm = (x0 + x1) / 2

        ym = fun(xm)

        Si y0 * ym < 0:

            x1 = xm

        Sino:

            x0 = xm

            y0 = ym

        delta = (x1 - x0) / x0

        Si delta < 0:

            delta *= -1

    Mientras delta > e

    Imprimir "Un aproximado a la raíz es: ", xm

Fin
```

Código:

```
double x0,x1,e,y0,y1,xm,ym,delta;
	int i=0;
	printf("Ingresa el valor de X0: ");
	scanf("%lf", &x0);
	printf("Ingresa el valor de X1: ");
	scanf("%lf", &x1);
	printf("Ingresa el valor del error: ");
	scanf("%lf", &e);
	y0=fun(x0);
	y1=fun(x1);
	if(y0*y1>=0){
		printf("Para que funcione, el producto de la multiplicacion de x0 y x1 debe ser mayor a 0, cambiar los valores\n");
		biseccion();
	}else{
		do{
			xm = (x0+x1)/2;
			ym=fun(xm);
			if(y0*ym<0){
				x1=xm;
			}else{
				x0=xm;
				y0=ym;
			}
			delta= (x1-x0)/x0;
			if(delta<0)
```

```
                    delta*=-1;
        }while(!delta<=e);
        printf("Un aproximado a la raiz es: %lf", xm);
}
return;
```

Prueba de escritorio

Ingresa el valor de X0: -1

Ingresa el valor de X1: 2

Ingresa el valor del error: 0.0001

Iteracion 0:

x0 = -1.000000

x1 = 2.000000

xm =(x0 + x1)/2= 0.500000

ym = fun(xm)= 1.000000

Cambiando x1 = xm

delta =(x1 - x0)/x0= 1.500000

Iteracion 1:

x0 = -1.000000

x1 = 0.500000

xm =(x0 + x1)/2= -0.250000

ym = fun(xm)= -6.593750

Cambiando x0 = xm

delta =(x1 - x0)/x0= 3.000000

Iteracion 2:

x0 = -0.250000

x1 = 0.500000

xm =(x0 + x1)/2= 0.125000

ym = fun(xm)= -1.636719

Cambiando x0 = xm

delta =(x1 - x0)/x0= 3.000000

Iteracion 3:

x0 = 0.125000

x1 = 0.500000

xm =(x0 + x1)/2= 0.312500

ym = fun(xm)= -0.067871

Cambiando x0 = xm

delta =(x1 - x0)/x0= 0.600000

Iteracion 4:

x0 = 0.312500

x1 = 0.500000

xm =(x0 + x1)/2= 0.406250

ym = fun(xm)= 0.523743

Cambiando x1 = xm

delta =(x1 - x0)/x0= 0.300000

Iteracion 5:

x0 = 0.312500

x1 = 0.406250

xm =(x0 + x1)/2= 0.359375

ym = fun(xm)= 0.242973

Cambiando x1 = xm

delta =(x1 - x0)/x0= 0.150000

Iteracion 6:

x0 = 0.312500

x1 = 0.359375

xm =(x0 + x1)/2= 0.335938

ym = fun(xm)= 0.091388

Cambiando x1 = xm

delta =(x1 - x0)/x0= 0.075000

Iteracion 7:

x0 = 0.312500

x1 = 0.335938

xm =(x0 + x1)/2= 0.324219

ym = fun(xm)= 0.012727

Cambiando x1 = xm

delta =(x1 - x0)/x0= 0.037500

Iteracion 8:

x0 = 0.312500

x1 = 0.324219

xm =(x0 + x1)/2= 0.318359

ym = fun(xm)= -0.027329

Cambiando x0 = xm

delta =(x1 - x0)/x0= 0.018405

Iteracion 9:

x0 = 0.318359

x1 = 0.324219

xm =(x0 + x1)/2= 0.321289

ym = fun(xm)= -0.007240

Cambiando x0 = xm

delta =(x1 - x0)/x0= 0.009119

Iteracion 10:

x0 = 0.321289

x1 = 0.324219

xm =(x0 + x1)/2= 0.322754

ym = fun(xm)= 0.002759

Cambiando x1 = xm

delta =(x1 - x0)/x0= 0.004559

Iteracion 11:

x0 = 0.321289

x1 = 0.322754

xm =(x0 + x1)/2= 0.322021

ym = fun(xm)= -0.002237

Cambiando x0 = xm

delta =(x1 - x0)/x0= 0.002274

Iteracion 12:

x0 = 0.322021

x1 = 0.322754

ym = fun(xm)= 0.000262

Cambiando x1 = xm

delta =(x1 - x0)/x0= 0.001137

Iteracion 13:

x0 = 0.322021

x1 = 0.322388

xm =(x0 + x1)/2= 0.322205

ym = fun(xm)= -0.000987

Cambiando x0 = xm

delta =(x1 - x0)/x0= 0.000568

Iteracion 14:

x0 = 0.322205

x1 = 0.322388

xm =(x0 + x1)/2= 0.322296

ym = fun(xm)= -0.000363

Cambiando x0 = xm

delta =(x1 - x0)/x0= 0.000284

Iteracion 15:

x0 = 0.322296

x1 = 0.322388

xm =(x0 + x1)/2= 0.322342

ym = fun(xm)= -0.000050

Cambiando x0 = xm

delta =(x1 - x0)/x0= 0.000142

Iteracion 16:

x0 = 0.322342

x1 = 0.322388

xm =(x0 + x1)/2= 0.322365

ym = fun(xm)= 0.000106

Cambiando x1 = xm

delta =(x1 - x0)/x0= 0.000071

Un aproximado a la raiz es: 0.322365

Ejecución:

```
x0 = 0.322021
x1 = 0.322388
xm = (x0 + x1)/2 = 0.322205
ym = fun(xm) = -0.000987
Cambiando x0 = xm
delta = (x1 - x0)/x0 = 0.000568

Iteracion 14:
x0 = 0.322205
x1 = 0.322388
xm = (x0 + x1)/2 = 0.322296
ym = fun(xm) = -0.000363
Cambiando x0 = xm
delta = (x1 - x0)/x0 = 0.000284

Iteracion 15:
x0 = 0.322296
x1 = 0.322388
xm = (x0 + x1)/2 = 0.322342
ym = fun(xm) = -0.000050
Cambiando x0 = xm
delta = (x1 - x0)/x0 = 0.000142

Iteracion 16:
x0 = 0.322342
x1 = 0.322388
xm = (x0 + x1)/2 = 0.322365
ym = fun(xm) = 0.000106
Cambiando x1 = xm
delta = (x1 - x0)/x0 = 0.000071
Tiempo de ejecucion: 0.013000 segundos

Un aproximado a la raiz es: 0.322365
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

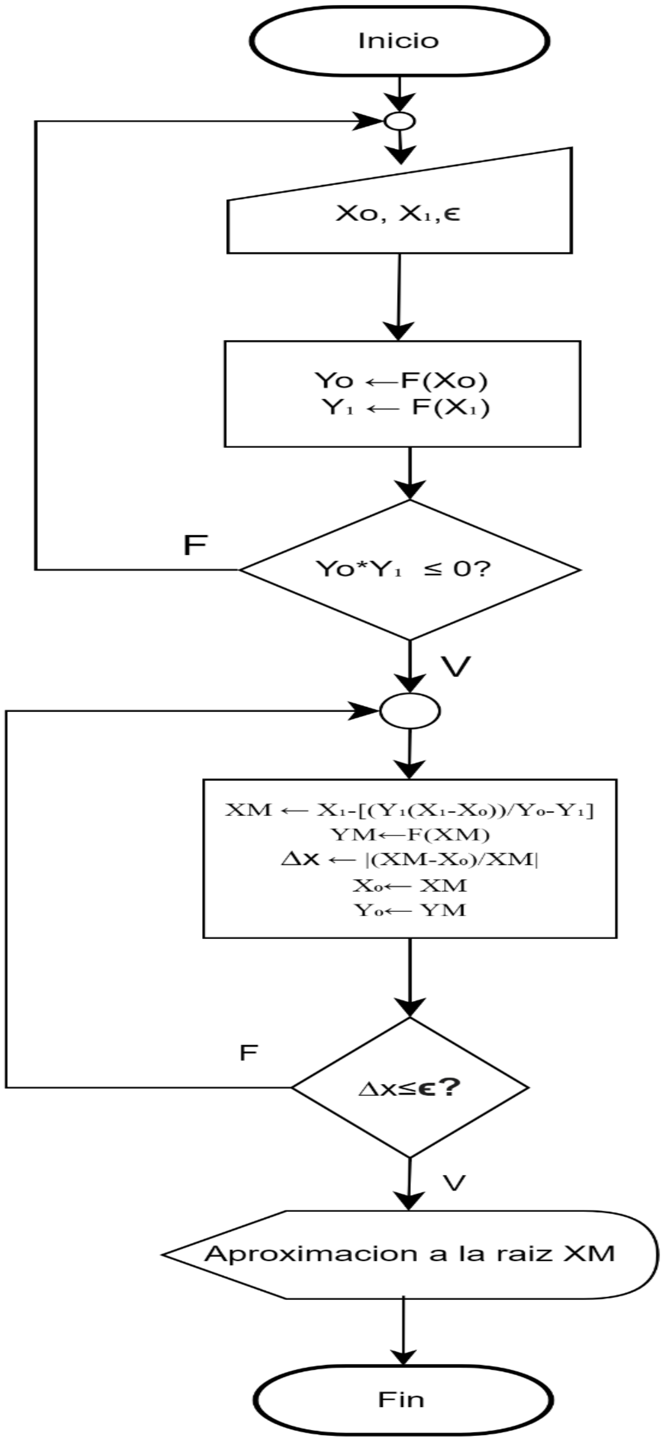
Método de Falsa Posición

Maestro: Dr. Cecil Vidriales

Método de Falsa Posición

Descripción: El Método de Falsa Posición combina elementos del método de bisección y la interpolación lineal para encontrar raíces.

Utilidad: Ofrece una convergencia más rápida que el Método de Bisección en algunos casos, útil para encontrar raíces con mayor eficiencia.



Pseudocódigo método de falsa posición

```
Inicio
   Leer X0
   Leer X1
   Leer e (valor del error)

   Y0 ← fun(X0)
   Y1 ← fun(X1)

   Si Y0 * Y1 ≥ 0 Entonces
      Imprimir "Para que funcione, el producto de la multiplicación de X0 y X1 debe ser
mayor a 0, cambiar los valores"
      Llamar a falsa_pos() (recursivamente o reiniciar el proceso)
   Sino
      Repetir
         Y0 ← fun(X0)
         Y1 ← fun(X1)
         XM ← X1 - ((Y1 * (X1 - X0)) / (Y1 - Y0))
         YM ← fun(XM)
         Δ ← (XM - X0) / XM

         Si Δ < 0 Entonces
            Δ ← -Δ
         FinSi

         Si Y0 * YM < 0 Entonces
            X1 ← XM
            Y1 ← YM
         Sino
            X0 ← XM
            Y0 ← YM
         FinSi

      Hasta que Δ < e

      Imprimir "Un aproximado a la raíz es: ", XM
   FinSi

Fin
```

Código:

```
        double x0,x1,e,y0,y1,xm,ym,delta;
        printf("Ingresa el valor de X0: ");
        scanf("%lf", &x0);
        printf("Ingresa el valor de X1: ");
        scanf("%lf", &x1);
        printf("Ingresa el valor del error:" );
        scanf("%lf", &e);
        y0=fun(x0);
        y1=fun(x1);
        if(y0*y1>=0){
                printf("Para que funcione, el producto de la multiplicacion de x0 y x1 debe ser
mayor a 0, cambiar los valores\n");
                falsa_pos();
        }else{
                do{
                        y0=fun(x0);
                        y1=fun(x1);
                        xm=x1-((y1*(x1-x0))/(y1-y0));
                        ym=fun(xm);
                        delta=(xm-x0)/xm;
                        if(delta<0)
                                delta*=-1;
                        if (y0 * ym < 0) {
           x1 = xm;
           y1 = ym;
        } else {
           x0 = xm;
           y0 = ym;
        }
                }while(delta>=e);
                printf("Un aproximado a la raiz es: %lf", xm);
        }
        return;
```

Prueba de escritorio
Ingresa el valor de X0: .1
Ingresa el valor de X1: .6
Ingresa el valor del error: 0.01

Iteracion 0:
x0 = 0.100000
x1 = 0.600000
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.387805
ym = fun(xm)= 0.416771
delta =(xm - x0)/xm= 0.742138
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 1:
x0 = 0.100000
x1 = 0.387805
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.335761
ym = fun(xm)= 0.090219
delta =(xm - x0)/xm= 0.702169
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 2:
x0 = 0.100000
x1 = 0.335761
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.325009
ym = fun(xm)= 0.018093
delta =(xm - x0)/xm= 0.692316
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 3:
x0 = 0.100000
x1 = 0.325009
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322873
ym = fun(xm)= 0.003572
delta =(xm - x0)/xm= 0.690281
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 4:
x0 = 0.100000
x1 = 0.322873
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322452
ym = fun(xm)= 0.000703
delta =(xm - x0)/xm= 0.689877
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 5:
x0 = 0.100000
x1 = 0.322452
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322370
ym = fun(xm)= 0.000138
delta =(xm - x0)/xm= 0.689797
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 6:
x0 = 0.100000
x1 = 0.322370
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322353
ym = fun(xm)= 0.000027
delta =(xm - x0)/xm= 0.689781
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 7:
x0 = 0.100000
x1 = 0.322353
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322350
ym = fun(xm)= 0.000005
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 8:
x0 = 0.100000
x1 = 0.322350
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349
ym = fun(xm)= 0.000001
delta =(xm - x0)/xm= 0.689778

Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 9:
x0 = 0.100000
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349
ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 10:
x0 = 0.100000
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349
ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 11:
x0 = 0.100000
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349
ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 12:
x0 = 0.100000
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349
ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 13:
x0 = 0.100000
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349

ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 14:
x0 = 0.100000
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349
ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 15:
x0 = 0.100000
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349
ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 16:
x0 = 0.100000
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349
ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 17:
x0 = 0.100000
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349
ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 18:
x0 = 0.100000
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349
ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 19:
x0 = 0.100000
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349
ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 20:
x0 = 0.100000
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349
ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 21:
x0 = 0.100000
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349
ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Verdadero
x1 = xm
y1=ym

Iteracion 22:
x0 = 0.100000
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349

ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.689778
Condicion (y0 * ym < 0): Falso
 x0 = xm
 y0=ym

Iteracion 23:
x0 = 0.322349
x1 = 0.322349
xm = x1 - ((y1 * (x1 - x0))/(y1 - y0))= 0.322349
ym = fun(xm)= 0.000000
delta =(xm - x0)/xm= 0.000000
Condicion (y0 * ym < 0): Falso
 x0 = xm
 y0=ym

Un aproximado a la raiz es: 0.322349

Ejecución:

```
Raices de ecuaciones
Elige el metodo a usar
1.Metodo de tanteos
2.Metodo de punto fijo
3.Metodo de Newton-Raphson
4.Metodo de Newton 2do orden
5.Metodo de Biseccion
6.Metodo de Falsa Posicion(cerrado)
7.Metodo Secante (Hibrido)
8.Volver
6
Metodo de Falsa Posicion(Cerrado)
Ingresa el valor de X0: .1
Ingresa el valor de X1: .6
Ingresa el valor del error: 0.01
Tiempo de ejecucion: 0.000000 segundos
Un aproximado a la raiz es: 0.322349
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

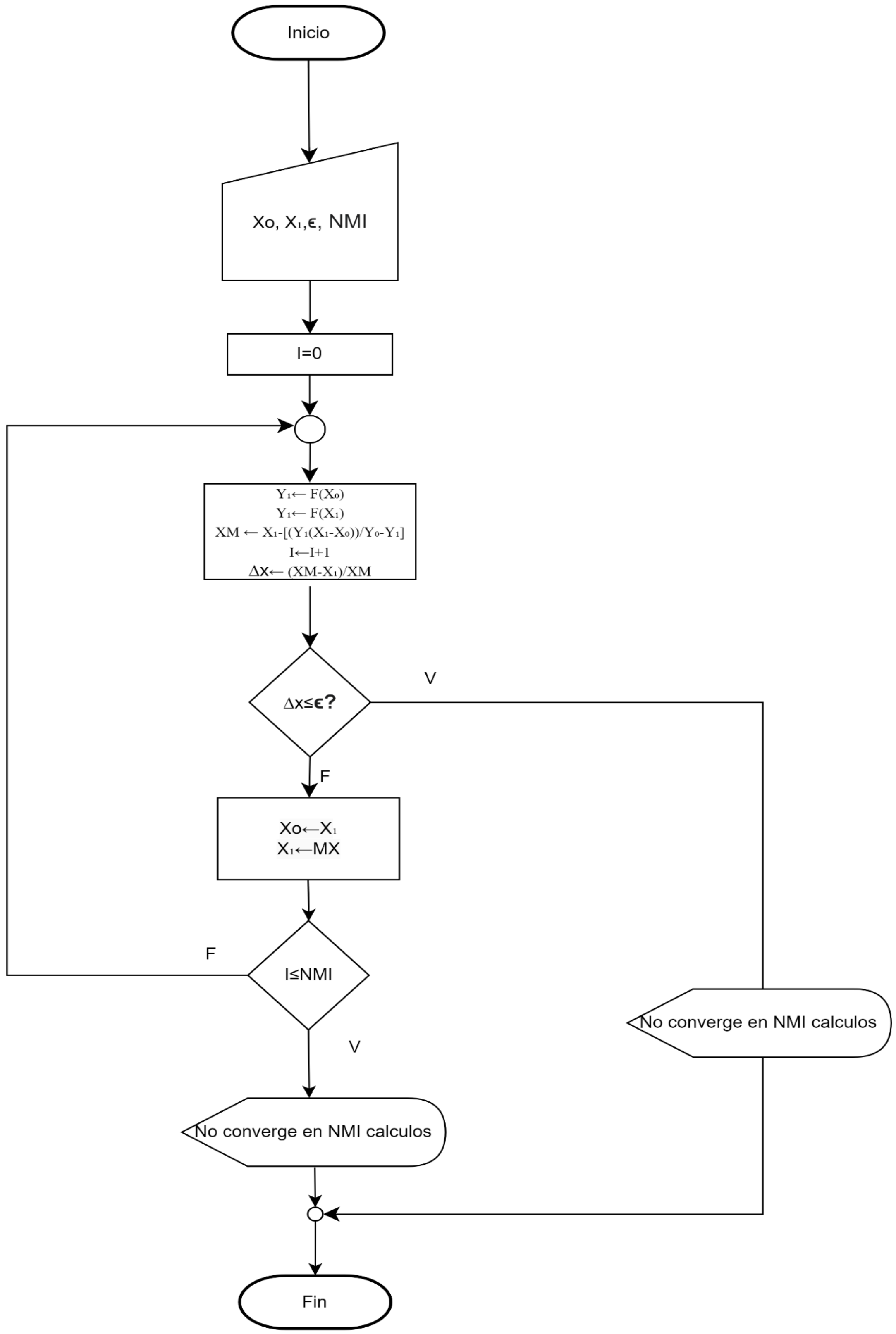
Método Secante(Híbrido)

Maestro: Dr. Cecil Vidriales

Método Secante (Híbrido)

Descripción: El Método Secante es un método iterativo que no requiere la derivada de la función, usando una secante para aproximar la raíz.

Utilidad: Es útil cuando la derivada de la función no está disponible o es costosa de calcular, combinando elementos de métodos iterativos y directos.



Pseudocódigo del método secante

```
Inicio
   Leer X0, X1, ε, NMI
   I ← 0

   Repetir
      Y0 ← F(X0)
      Y1 ← F(X1)
      XM ← X1 - (Y1 * (X1 - X0) / (Y1 - Y0))
      ΔX ← (XM - X1) / XM
      I ← I + 1

      Si ΔX < 0 Entonces
         ΔX ← -ΔX
      FinSi

      Si ΔX ≤ ε Entonces
         Imprimir "Converge en ", I, " cálculos"
         Terminar
      FinSi

      X0 ← X1
      X1 ← XM

      Si I ≥ NMI Entonces
         Imprimir "No converge en NMI cálculos"
         Terminar
      FinSi

   Hasta que ΔX ≤ ε o I ≥ NMI

Fin
```

}

Código:

```
double x0,x1,e,nmi,y0,y1,xm,delta;

printf("Ingresa el valor de X0: ");

scanf("%lf", &x0);

printf("Ingresa el valor de X1: ");

scanf("%lf", &x1);

printf("Ingresa el valor del error: ");

scanf("%lf", &e);

printf("Ingresa el numero de iteraciones: ");

scanf("%lf", &nmi);

for(int i=0;i<=nmi;i++){

	y0=fun(x0);

	y1=fun(x1);

	xm=x1-((y1*(x0-x1))/(y0-y1));

	delta=(xm-x1)/xm;

	if(delta<0)

		delta*=-1;

	if(delta<=e){

		printf("Un aproximado a la raiz es: %lf", xm);

		return;

	}else{

		x0=x1;

		x1=xm;

	}

}

printf("No converge en %lf calculos", nmi);

				return;
```

Prueba escritorio
Ingresa el valor de X0: -1
Ingresa el valor de X1: 2
Ingresa el valor del error: 0.001
Ingresa el numero de iteraciones: 100

Iteracion 0:
x0 = -1.000000
x1 = 2.000000
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 1.888889
delta =(xm - x1)/xm= 0.058824

Iteracion 1:
x0 = 2.000000
x1 = 1.888889
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 5.240000
delta =(xm - x1)/xm= 0.639525

Iteracion 2:
x0 = 1.888889
x1 = 5.240000
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 1.854048
delta =(xm - x1)/xm= 1.826248

Iteracion 3:
x0 = 5.240000
x1 = 1.854048
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 1.818041
delta =(xm - x1)/xm= 0.019806

Iteracion 4:
x0 = 1.854048
x1 = 1.818041
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 3.141093
delta =(xm - x1)/xm= 0.421208

Iteracion 5:
x0 = 1.818041
x1 = 3.141093
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 1.606201
delta =(xm - x1)/xm= 0.955604

Iteracion 6:
x0 = 3.141093

x1 = 1.606201
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 1.290736
delta =(xm - x1)/xm= 0.244407

Iteracion 7:
x0 = 1.606201
x1 = 1.290736
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 2.542607
delta =(xm - x1)/xm= 0.492357

Iteracion 8:
x0 = 1.290736
x1 = 2.542607
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= -4.229331
delta =(xm - x1)/xm= 1.601184

Iteracion 9:
x0 = 2.542607
x1 = -4.229331
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 2.502098
delta =(xm - x1)/xm= 2.690314

Iteracion 10:
x0 = -4.229331
x1 = 2.502098
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 2.465346
delta =(xm - x1)/xm= 0.014908

Iteracion 11:
x0 = 2.502098
x1 = 2.465346
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 2.035532
delta =(xm - x1)/xm= 0.211156

Iteracion 12:
x0 = 2.465346
x1 = 2.035532
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 1.526290
delta =(xm - x1)/xm= 0.333646

Iteracion 13:
x0 = 2.035532
x1 = 1.526290
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 3.154842

delta =(xm - x1)/xm= 0.516207

Iteracion 14:
x0 = 1.526290
x1 = 3.154842
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 1.167014
delta =(xm - x1)/xm= 1.703345

Iteracion 15:
x0 = 3.154842
x1 = 1.167014
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 0.545176
delta =(xm - x1)/xm= 1.140618

Iteracion 16:
x0 = 1.167014
x1 = 0.545176
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= -0.463469
delta =(xm - x1)/xm= 2.176295

Iteracion 17:
x0 = 0.545176
x1 = -0.463469
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 0.444082
delta =(xm - x1)/xm= 2.043657

Iteracion 18:
x0 = -0.463469
x1 = 0.444082
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 0.386144
delta =(xm - x1)/xm= 0.150042

Iteracion 19:
x0 = 0.444082
x1 = 0.386144
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 0.313004
delta =(xm - x1)/xm= 0.233669

Iteracion 20:
x0 = 0.386144
x1 = 0.313004
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 0.322993
delta =(xm - x1)/xm= 0.030925

Iteracion 21:
x0 = 0.313004
x1 = 0.322993
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 0.322355
delta =(xm - x1)/xm= 0.001978

Iteracion 22:
x0 = 0.322993
x1 = 0.322355
xm = x1 - ((y1 * (x0 - x1))/(y0 - y1))= 0.322349
delta =(xm - x1)/xm= 0.000019
Un aproximado a la raiz es: 0.322349

Ejecución:

```
Raices de ecuaciones
Elige el metodo a usar
1.Metodo de tanteos
2.Metodo de punto fijo
3.Metodo de Newton-Raphson
4.Metodo de Newton 2do orden
5.Metodo de Biseccion
6.Metodo de Falsa Posicion(cerrado)
7.Metodo Secante (Hibrido)
8.Volver
7
Metodo Secante (Hibrido)
Ingresa el valor de X0: -1
Ingresa el valor de X1: 2
Ingresa el valor del error: 0.001
Ingresa el numero de iteraciones: 100
Tiempo de ejecucion: 0.000000 segundos
Un aproximado a la raiz es: 0.322349
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

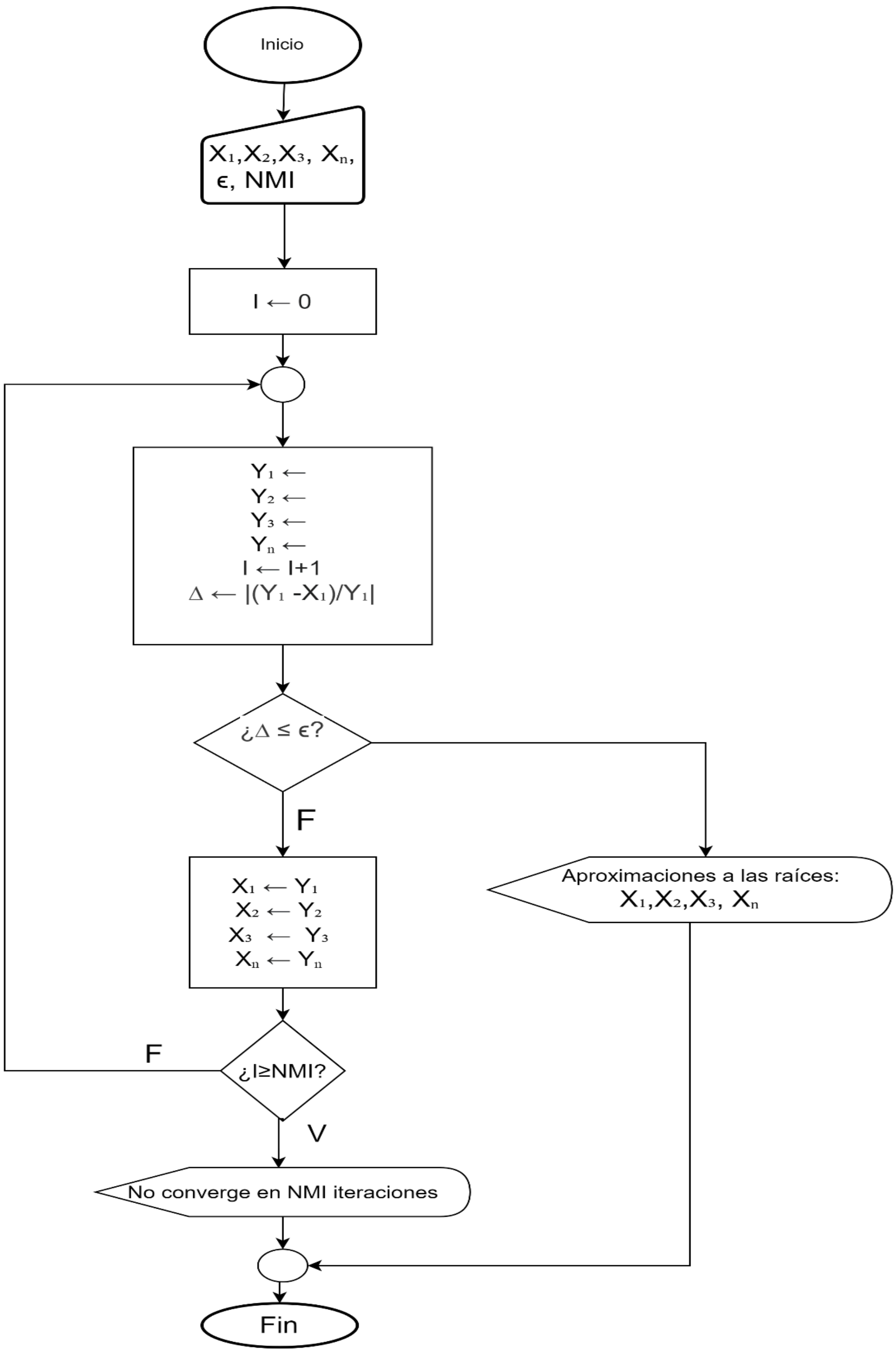
Algoritmo de Jacobi

Maestro: Dr. Cecil Vidriales

Algoritmo de Jacobi

Descripción: El Algoritmo de Jacobi es un método iterativo para resolver sistemas de ecuaciones lineales, basado en la descomposición de la matriz.

Utilidad: Es útil para sistemas de ecuaciones grandes y dispersos donde los métodos directos son impracticables.



Pseudocódigo Jacobi

1. Inicio
2. Entrada:
   - X1, X2, X3, X: Valores iniciales de las aproximaciones a las raíces
   - ε: Tolerancia de error
   - NMI: Número máximo de iteraciones
3. Proceso:
   - Calcular Y:
     Y = (X1 + X2 + X3) / 3
   - Verificar si Y es la raíz:
     Si abs(X - Y) < ε:
       Salida: X1, X2, X3, X son las aproximaciones a las raíces
       Fin del proceso
   - Actualizar las aproximaciones:
     X1 = (X1 + Y) / 2
     X2 = (X2 + Y) / 2
     X3 = (X3 + Y) / 2
   - Verificar si se ha alcanzado el número máximo de iteraciones:
     Si NMI == 0:
       Salida: El algoritmo no converge en NMI iteraciones
       Fin del proceso
     Si no:
       NMI = NMI - 1
       Ir al paso 1
4. Fin

Código:

```
    double x1=0, x2=0, x3=0, nmi, delta1, delta2, delta3, e, i=0, y1, y2, y3;

    printf("Ingresa el valor del error: ");

    scanf("%lf", &e);

    printf("Ingresa el numero de iteraciones: ");

    scanf("%lf", &nmi);

    do {

  y1 = (20 + x2 - x3) / 20;

  y2 = (11 - 2 * x1 + x3) / 10;

  y3 = (-18 - x1 - x2) / (-20);


  delta1 = fabs((y1 - x1) / y1);
```

```
        delta2 = fabs((y2 - x2) / y2);

        delta3 = fabs((y3 - x3) / y3);


        x1 = y1;

        x2 = y2;

        x3 = y3;


        i++;


        if (delta1 <= e && delta2 <= e && delta3 <= e) {

            printf("Aproximacion a las raices:\n y1 = %lf \n y2 = %lf \n y3 = %lf \n", y1, y2, y3);

            return;

        }


    } while (i < nmi);

        printf("No converge\n");

        return;
```

Prueba de escritorio

Ingresa el valor del error: 0.001

Ingresa el numero de iteraciones: 100

Iteracion 0.000000:

Operaciones:

y1 =(20 + x2 - x3)/20=(20 + 0.000000 - 0.000000)/20= 1.000000

y2 =(11 - 2 * x1 + x3)/10=(11 - 2 * 0.000000 + 0.000000)/10= 1.100000

y3 =(-18 - x1 - x2)/(-20)=(-18 - 0.000000 - 0.000000)/(-20)= 0.900000

delta1 = |(y1 - x1)/y1|= |(1.000000 - 0.000000)/1.000000|= 1.000000

delta2 = |(y2 - x2)/y2|= |(1.100000 - 0.000000)/1.100000|= 1.000000

delta3 = |(y3 - x3)/y3|= |(0.900000 - 0.000000)/0.900000|= 1.000000

Iteracion 1.000000:

Operaciones:

y1 =(20 + x2 - x3)/20=(20 + 1.100000 - 0.900000)/20= 1.010000

y2 =(11 - 2 * x1 + x3)/10=(11 - 2 * 1.000000 + 0.900000)/10= 0.990000

y3 =(-18 - x1 - x2)/(-20)=(-18 - 1.000000 - 1.100000)/(-20)= 1.005000

delta1 = |(y1 - x1)/y1|= |(1.010000 - 1.000000)/1.010000|= 0.009901

delta2 = |(y2 - x2)/y2|= |(0.990000 - 1.100000)/0.990000|= 0.111111

delta3 = |(y3 - x3)/y3|= |(1.005000 - 0.900000)/1.005000|= 0.104478

Iteracion 2.000000:

Operaciones:

y1 =(20 + x2 - x3)/20=(20 + 0.990000 - 1.005000)/20= 0.999250

y2 =(11 - 2 * x1 + x3)/10=(11 - 2 * 1.010000 + 1.005000)/10= 0.998500

y3 =(-18 - x1 - x2)/(-20)=(-18 - 1.010000 - 0.990000)/(-20)= 1.000000 delta1 = |(y1 - x1)/y1|= |(0.999250 - 1.010000)/0.999250|= 0.010758

delta2 = |(y2 - x2)/y2|= |(0.998500 - 0.990000)/0.998500|= 0.008513

delta3 = |(y3 - x3)/y3|= |(1.000000 - 1.005000)/1.000000|= 0.005000

Iteracion 3.000000:

Operaciones:

y1 =(20 + x2 - x3)/20=(20 + 0.998500 - 1.000000)/20= 0.999925

y2 =(11 - 2 * x1 + x3)/10=(11 - 2 * 0.999250 + 1.000000)/10= 1.000150

y3 =(-18 - x1 - x2)/(-20)=(-18 - 0.999250 - 0.998500)/(-20)= 0.999888

delta1 = |(y1 - x1)/y1|= |(0.999925 - 0.999250)/0.999925|= 0.000675

delta2 = |(y2 - x2)/y2|= |(1.000150 - 0.998500)/1.000150|= 0.001650

delta3 = |(y3 - x3)/y3|= |(0.999888 - 1.000000)/0.999888|= 0.000113

Iteracion 4.000000:

Operaciones:

y1 =(20 + x2 - x3)/20=(20 + 1.000150 - 0.999888)/20= 1.000013

y2 =(11 - 2 * x1 + x3)/10=(11 - 2 * 0.999925 + 0.999888)/10= 1.000004

y3 =(-18 - x1 - x2)/(-20)=(-18 - 0.999925 - 1.000150)/(-20)= 1.000004

delta1 = |(y1 - x1)/y1|= |(1.000013 - 0.999925)/1.000013|= 0.000088

delta2 = |(y2 - x2)/y2|= |(1.000004 - 1.000150)/1.000004|= 0.000146

delta3 = |(y3 - x3)/y3|= |(1.000004 - 0.999888)/1.000004|= 0.000116

Aproximacion a las raices:

y1 = 1.000013

y2 = 1.000004

y3 = 1.000004

Ejecución:

```
Menu
Elige el tema que deseas ejecutar:
Temas:
1.Raices de ecuaciones.
2.Sistemas de Ecuaciones Lineales Algebraicas
3.Derivacion e integracion numerica.
4.Ecuaciones diferenciales ordinarias.
5.Salir
2
Sistemas de Ecuaciones Lineales Algebraicas
Elige el metodo a usar
1.Algoritmo de Jacobi
2.Algoritmo de Gauss Seidel
3.Volver al Menu
1
Algoritmo de Jacobi
Ingresa el valor del error: 0.001
Ingresa el numero de iteraciones: 100
Tiempo de ejecucion: 0.000000 segundos
Aproximacion a las raices:
 y1 = 1.000013
 y2 = 1.000004
 y3 = 1.000004
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

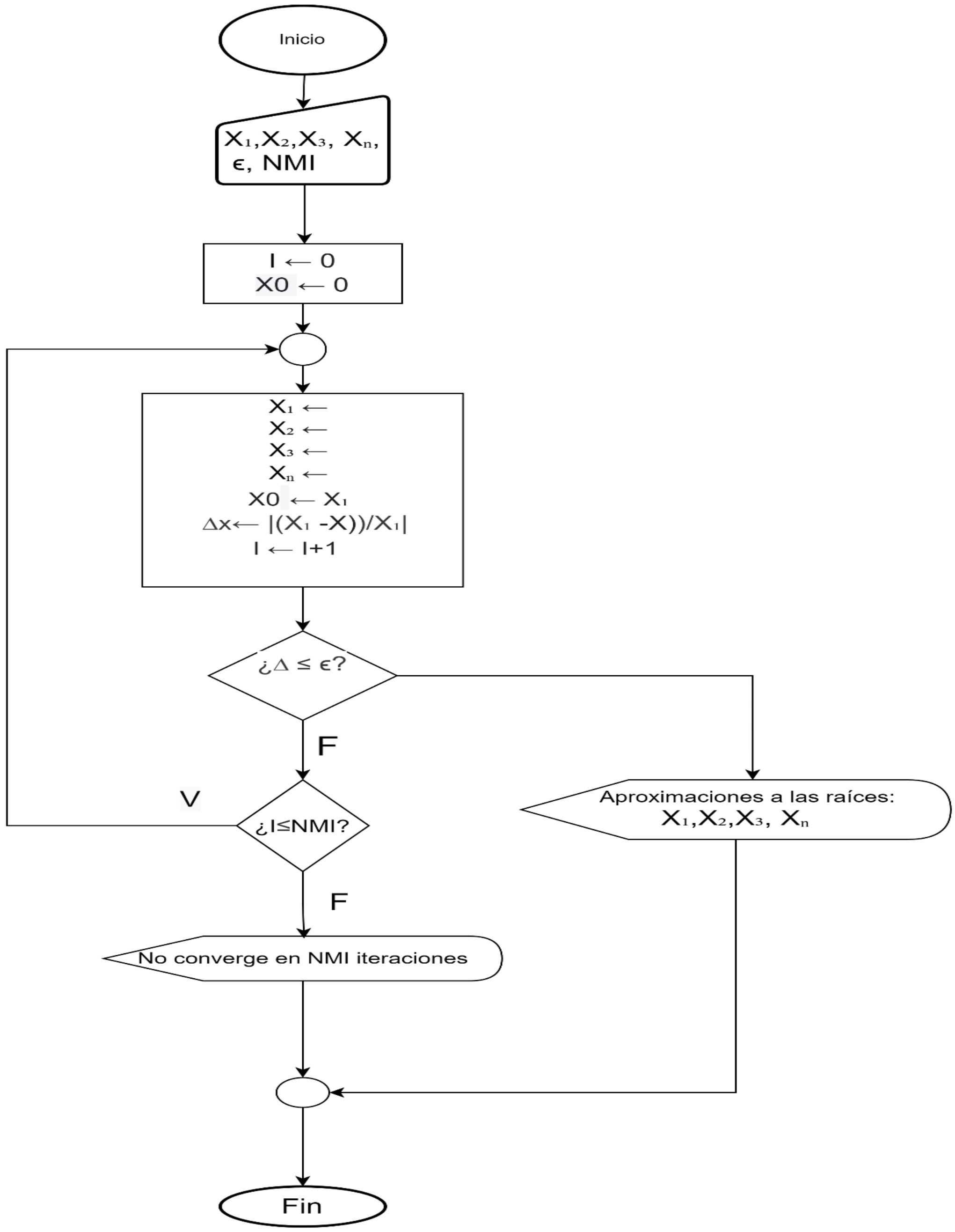
Algoritmo de Gauss-Seidel

Maestro: Dr. Cecil Vidriales

Algoritmo de Gauss-Seidel

Descripción: El Algoritmo de Gauss-Seidel es una mejora del Método de Jacobi, que usa los valores más recientes en las iteraciones.

Utilidad: Converge más rápido que el Método de Jacobi y es útil para sistemas de ecuaciones grandes y dispersos.



Pseudo código Gauss-Seidel

```
Inicio
   x1 ← 0
   x2 ← 0
   x3 ← 0
   x0 ← 0
   i ← 0

   Leer e (valor del error)
   Leer nmi (número de iteraciones)

   Repetir
      x1 ← (20 + x2 - x3) / 20
      x2 ← (11 - 2 * x1 + x3) / 10
      x3 ← (-18 - x1 - x2) / -20
      delta ← (x1 - x0) / x1

      Si delta < 0 Entonces
         delta ← -delta
      FinSi

      x0 ← x1
      i ← i + 1

      Si delta ≤ e Entonces
         Imprimir "Aproximación a las raíces:"
         Imprimir "x1 = ", x1
         Imprimir "x2 = ", x2
         Imprimir "x3 = ", x3
         Terminar
      FinSi

   Hasta que i ≥ nmi

   Imprimir "No converge"
   Terminar
Fin
```

Código:

```
	double x1=0, x2=0, x3=0, nmi, delta, e, i=0, x0=0;
	printf("Ingresa el valor del error: ");
	scanf("%lf", &e);
	printf("Ingresa el numero de iteraciones: ");
	scanf("%lf", &nmi);
	do{
		x1=(20+x2-x3)/20;
		x2=(11-2*x1+x3)/10;
		x3=(-18-x1-x2)/(-20);
		delta=(x1-x0)/x1;
		if(delta<0)
			delta*=-1;
		x0=x1;
		i++;
		if (delta<=e){
			printf("Aproximacion a las raices:\n x1 = %lf \n x2 = %lf \n x3 = %lf \n",
x1, x2, x3);
			return;
		}

	}while(i<nmi);
	printf("No converge\n");
	return;
```

Prueba de escritorio
Ingresa el valor del error: 0.001
Ingresa el numero de iteraciones: 100

Iteracion 0.000000:
x1 =(20 + x2 - x3)/20=(20 + 0.900000 - 0.995000)/20= 1.000000
x2 =(11 - 2 * x1 + x3)/10=(11 - 2 * 1.000000 + 0.995000)/10= 0.900000
x3 =(-18 - x1 - x2)/(-20)=(-18 - 1.000000 - 0.900000)/(-20)= 0.995000
delta =(x1 - x0)/x1=(1.000000 - 0.000000)/1.000000= 1.000000

Iteracion 1.000000:
x1 =(20 + x2 - x3)/20=(20 + 1.000450 - 0.999785)/20= 0.995250
x2 =(11 - 2 * x1 + x3)/10=(11 - 2 * 0.995250 + 0.999785)/10= 1.000450
x3 =(-18 - x1 - x2)/(-20)=(-18 - 0.995250 - 1.000450)/(-20)= 0.999785
delta =(x1 - x0)/x1=(0.995250 - 1.000000)/0.995250= 0.004773

Iteracion 2.000000:
x1 =(20 + x2 - x3)/20=(20 + 0.999972 - 1.000000)/20= 1.000033
x2 =(11 - 2 * x1 + x3)/10=(11 - 2 * 1.000033 + 1.000000)/10= 0.999972
x3 =(-18 - x1 - x2)/(-20)=(-18 - 1.000033 - 0.999972)/(-20)= 1.000000
delta =(x1 - x0)/x1=(1.000033 - 0.995250)/1.000033= 0.004783

Iteracion 3.000000:
x1 =(20 + x2 - x3)/20=(20 + 1.000000 - 1.000000)/20= 0.999999
x2 =(11 - 2 * x1 + x3)/10=(11 - 2 * 0.999999 + 1.000000)/10= 1.000000
x3 =(-18 - x1 - x2)/(-20)=(-18 - 0.999999 - 1.000000)/(-20)= 1.000000
delta =(x1 - x0)/x1=(0.999999 - 1.000033)/0.999999= 0.000035

Aproximación a las raices:
x1 = 0.999999
x2 = 1.000000
x3 = 1.000000

Ejecución:

```
Sistemas de Ecuaciones Lineales Algebraicas
Elige el metodo a usar
1.Algoritmo de Jacobi
2.Algoritmo de Gauss Seidel
3.Volver al Menu
2
Algoritmo de Gauss Seidel
Ingresa el valor del error: 0.001
Ingresa el numero de iteraciones: 100
Tiempo de ejecucion: 0.000000 segundos
Aproximacion a las raices:
 x1 = 0.999999
 x2 = 1.000000
 x3 = 1.000000
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Diferenciación numérica

Primer derivada baja precisión

Maestro: Dr. Cecil Vidriales

Diferenciación numérica primer derivada baja precisión

Descripción: La diferenciación numérica de primera derivada con baja precisión es una técnica que utiliza diferencias finitas hacia adelante para aproximar la primera derivada de una función en un punto dado. Se calcula la pendiente de la función entre dos puntos muy cercanos, lo que proporciona una estimación de la tasa de cambio en ese punto.

Utilidad: Esta técnica es útil cuando se necesita una estimación rápida de la pendiente de una función en un punto específico y no se dispone de una expresión analítica para la derivada. Aunque su precisión puede ser limitada, es práctica en situaciones donde la simplicidad y la rapidez son prioritarias, especialmente en contextos de cálculo numérico o en problemas donde la función es compleja y el cálculo analítico resulta difícil.

Funcion:

$$f'(x) = (f(x_{i+1}) - f(x_{i-1}))/(2 * h)$$

Código:

```
double h, x, x1, x2, res;

   printf("Ingresa el valor de X: ");

   scanf("%lf", &x); /

   printf("Ingresa el valor de h (salto): ");

   scanf("%lf", &h);

   if (h == 0) {

      printf("El valor de h no puede ser cero.\n");

      return;

   }

   x1 = x + h;

   x2 = x - h;

   res = (funDev(x1) - funDev(x2)) / (2 * h); // Cambio de funcion por funDev

   printf("El resultado de la aproximacion de la derivada es: %lf\n", res)
```

Prueba de escritorio

Ingresa el valor de X: 2

Ingresa el valor de h (salto): .2

Puntos:

x1 = x + h = 2.000000 + 0.200000 = 2.200000

x2 = x - h = 2.000000 - 0.200000 = 1.800000

El resultado de la aproximacion de la derivada es: -5.280527

Ejecución:

```
Elige el metodo a usar
1.Derivar
2.integrar
3.Volver
1
Derivadas
1.Primera derivada
2.Segunda Derivada
3.Tercera derivada
4.Cuarta derivada
5.Volver
1
Primera Derivada
Elige la presicion de la primera derivada
1.Baja Presicion
2.Alta Presicion
3.Volver
1
Primera derivada Baja Presicion
Ingresa el valor de X: 2
Ingresa el valor de h (salto): .2
Tiempo de ejecucion: 0.000000 segundos
El resultado de la aproximacion de la derivada es: -5.280527
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Diferenciación numérica

Primera derivada alta precisión

Maestro: Dr. Cecil Vidriales

Diferenciación numérica primer derivada alta precisión

Descripción: La diferenciación numérica de primer derivada con alta precisión utiliza diferencias finitas centradas para aproximar la primera derivada de una función en un punto dado.
Utilidad: Es crucial para obtener estimaciones altamente precisas de la pendiente de una función en un punto específico cuando no se dispone de una expresión analítica para la derivada.

Funcion:

$$f'(x) = (-\ f(x_{i+2}) + 8f(x_{i+1}) -\ 8f(x_{i-1}) + f(x_{i-2}))/12h$$

Código:

```
double h, x, res;


    printf("Ingresa el valor de X: ");
    scanf("%lf", &x);
    printf("Ingresa el valor de h (salto): ");
    scanf("%lf", &h);


    if (h == 0) {
        printf("El valor de h no puede ser cero.\n");
        return;
    }


    double x1 = x - (2 * h);
    double x2 = x - h;
    double x3 = x;
    double x4 = x + h;
    double x5 = x + (2 * h);

    printf("\tValores de X\n");
    printf("Xi-2= %lf\n    Xi-1= %lf\n    Xi= %lf\n  Xi+1= %lf\n   Xi+2= %lf\n\n", x1, x2, x3, x4,
x5);


    double a = funDev(x1);
    double b = funDev(x2);
    double c = funDev(x3);
    double d = funDev(x4);
    double e = funDev(x5);

    printf("\tValores f(x)\n");
```

```
printf("f(Xi-2) = %lf\n f(Xi-1)=%lf\n f(Xi)=%lf\n f(Xi+1)=%lf\n f(Xi+2)=%lf\n\n", a, b, c, d, e);


res = ((-1 * funDev(x5)) + 8 * funDev(x4) - 8 * funDev(x2) + funDev(x1)) / (12 * h);

printf("El resultado de la aproximacion de la derivada es: %lf\n", res);
```

Prueba de escritorio
Ingresa el valor de X: 2
Ingresa el valor de h (salto): .2
        Valores de X
Xi-2= 1.600000
Xi-1= 1.800000
Xi= 2.000000
Xi+1= 2.200000
Xi+2= 2.400000

        Valores f(x)
f(Xi-2) = -0.074751
f(Xi-1) = -0.736135
f(Xi) = -1.664587
f(Xi+1) = -2.848345
f(Xi+2) = -4.247388

El resultado de la aproximación de la derivada es: -5.302103

Ejecución:

```
Primera derivada Alta Presicion
Ingresa el valor de X: 2
Ingresa el valor de h (salto): .2
        Valores de X
Xi-2= 1.600000
    Xi-1= 1.800000
    Xi= 2.000000
  Xi+1= 2.200000
   Xi+2= 2.400000

        Valores f(x)
f(Xi-2) = -0.074751
 f(Xi-1)=-0.736135
 f(Xi)=-1.664587
 f(Xi+1)=-2.848345
 f(Xi+2)=-4.247388

Tiempo de ejecucion: 0.001000 segundos
El resultado de la aproximacion de la derivada es: -5.302103
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Diferenciación numérica

Segunda derivada alta precisión

Maestro: Dr. Cecil Vidriales

Diferenciación numérica segunda derivada alta precisión

Descripción: La **diferenciación numérica de segunda derivada con alta precisión** emplea diferencias finitas centradas para calcular la segunda derivada de una función en un punto dado.

Utilidad: Es esencial para obtener estimaciones altamente precisas de la concavidad de una función en un punto específico cuando no se tiene acceso a una expresión analítica para la segunda derivada.

Funcion:

$$f''(x_i) = (-f(x_{i+2}) + 16f(x_{i+1}) - 30f(x_i) + 16f(x_{i-1}) - f(x_{i-2}))/12h^2$$

Código:

```
double h, x, res;

printf("Ingresa el valor de X: ");

scanf("%lf", &x);

printf("Ingresa el valor de h (salto): ");

scanf("%lf", &h);

if (h == 0) {

    printf("El valor de h no puede ser cero.\n");

    return;

}

double x1 = x - (2 * h);

double x2 = x - h;

double x3 = x;

double x4 = x + h;

double x5 = x + (2 * h);
```

```
    res = (-funDev(x5) + 16 * funDev(x4) - 30 * funDev(x3) + 16 * funDev(x2) - funDev(x1)) / (12 * h * h);


    printf("El resultado de la aproximación de la segunda derivada es: %lf\n", res);
```

Prueba de escritorio

Ingresa el valor de X: 2

Ingresa el valor de h (salto): .2

Valores de X

Xi-2= 1.600000

Xi-1= 1.800000

Xi= 2.000000

Xi+1= 2.200000

Xi+2= 2.400000

Valores f(x)

f(Xi-2) = -0.074751

f(Xi-1) = -0.736135

f(Xi) = -1.664587

f(Xi+1) = -2.848345

f(Xi+2) = -4.247388

El resultado de la aproximacion de la segunda derivada es: -6.441509

Ejecución:

```
Derivadas
1.Primera derivada
2.Segunda Derivada
3.Tercera derivada
4.Cuarta derivada
5.Volver
2
Segunda Derivada
Ingresa el valor de X: 2
Ingresa el valor de h (salto): .2
Tiempo de ejecucion: 0.000000 segundos
El resultado de la aproximaci├│n de la segunda derivada es: -6.441509
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Diferenciación numérica

Tercera derivada alta precisión

Maestro: Dr. Cecil Vidriales

Diferenciación numérica de tercera derivada con alta precisión

Descripción: La diferenciación numérica de tercera derivada con alta precisión utiliza técnicas de diferenciación numérica para calcular la tercera derivada de una función en un punto determinado.

Utilidad: Esta técnica es fundamental para obtener estimaciones altamente precisas de la tasa de cambio de la tasa de cambio de una función en un punto específico, especialmente en contextos donde la exactitud es crítica y no se dispone de una expresión analítica para la tercera derivada.

Funcion:

$$f'''(x_i) = (-f(x_{i+3}) + 8f(x_{i+2}) - 13f(x_{i+1}) + 13f(x_{i-1}) - 8f(x_{i-2}) + f(x_{i-3}))/8h^3$$

Código:

```
double h, x, res;



    printf("Ingresa el valor de X: ");

    scanf("%lf", &x);

    printf("Ingresa el valor de h (salto): ");

    scanf("%lf", &h);

    if (h == 0) {

        printf("El valor de h no puede ser cero.\n");

        return;

    }

    double x1 = x - (3 * h);

    double x2 = x - (2 * h);

    double x3 = x - h;

    double x4 = x;

    double x5 = x + h;

    double x6 = x + (2 * h);
```

```
    double x7 = x + (3 * h);

    res = (-(funDev(x7)) + 8 * (funDev(x6)) - 13 * (funDev(x5)) + 13 * (funDev(x3)) - 8 *
(funDev(x2)) + funDev(x1)) / (8 * (h * h * h));

    printf("El resultado de la aproximacion de la tercera derivada es: %lf\n", res);
```

Prueba de escritorio

Ingresa el valor de X: 2
Ingresa el valor de h (salto): .2
Valores de X
Xi-3 = x - 3h = 2.000000 - 3*0.200000 = 1.400000
Xi-2 = x - 2h = 2.000000 - 2*0.200000 = 1.600000
Xi-1 = x - h = 2.000000 - 0.200000 = 1.800000
Xi = 2.000000
Xi+1 = x + h = 2.000000 + 0.200000 = 2.200000
Xi+2 = x + 2h = 2.000000 + 2*0.200000 = 2.400000
Xi+3 = x + 3h = 2.000000 + 3*0.200000 = 2.600000

Valores f(x)
f(Xi-3) = funDev(1.400000) = 0.333136
f(Xi-2) = funDev(1.600000) = -0.074751
f(Xi-1) = funDev(1.800000) = -0.736135
f(Xi) = funDev(2.000000) = -1.664587
f(Xi+1) = funDev(2.200000) = -2.848345
f(Xi+2) = funDev(2.400000) = -4.247388
f(Xi+3) = funDev(2.600000) = -5.792568

Calculo de la tercera derivada:
res = (-f(Xi+3) + 8*f(Xi+2) - 13*f(Xi+1) + 13*f(Xi-1) - 8*f(Xi-2) + f(Xi-3)) / (8 * h^3)
res = (-(-5.792568) + 8*-4.247388 - 13*-2.848345 + 13*-0.736135 - 8*-0.074751 + 0.333136) / (8 * (0.200000)^3)
res = (--5.792568 + -33.979102 - -37.028490 + -9.569752 - -0.598006 + 0.333136) / 0.064000
res = 3.177273
El resultado de la aproximacion de la tercera derivada es: 3.177273

Ejecución:

```
Derivadas
1.Primera derivada
2.Segunda Derivada
3.Tercera derivada
4.Cuarta derivada
5.Volver
3
Tercera Derivada
Ingresa el valor de X: 2
Ingresa el valor de h (salto): .2
Tiempo de ejecucion: 0.000000 segundos
El resultado de la aproximacion de la tercera derivada es: 3.177273
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Diferenciación numérica

Cuarta derivada alta precisión

Maestro: Dr. Cecil Vidriales

Diferenciación numérica cuarta derivada alta precisión

Descripción: La diferenciación numérica de cuarta derivada con alta precisión** emplea métodos numéricos para calcular la cuarta derivada de una función en un punto específico.

Utilidad: Es esencial para obtener estimaciones altamente precisas de la tasa de cambio de la tasa de cambio de la tasa de cambio de una función en un punto dado, particularmente cuando la exactitud es crucial y no se tiene acceso a una expresión analítica para la cuarta derivada.

Funcion:

$$f''''(x_i) = (-f(x_{i+3}) + 12f(x_{i+2}) - 39f(x_{i+1}) + 56f(x_i) - 39f(x_{i-1}) - 12\,f(x_{i-2}) - f(x_{i-3}))/6h^4$$

Código:

```
double h, x, res;


    printf("Ingresa el valor de X: ");
    scanf("%lf", &x);
    printf("Ingresa el valor de h (salto): ");
    scanf("%lf", &h);


    if (h == 0) {
        printf("El valor de h no puede ser cero.\n");
        return;
    }

    double x1 = x - (3 * h);
    double x2 = x - (2 * h);
    double x3 = x - h;
    double x4 = x;
    double x5 = x + h;
    double x6 = x + (2 * h);
    double x7 = x + (3 * h);
    res = (-(funDev(x7)) + 12 * (funDev(x6)) - 39 * (funDev(x5)) + 56 * funDev(x4) - 39 *
(funDev(x3)) + 12 * (funDev(x2)) - funDev(x1)) / (6 * (h * h * h * h));

    printf("El resultado de la aproximacion de la cuarta derivada es: %lf\n", res);
```

Prueba de escritorio
Ingresa el valor de X: 2
Ingresa el valor de h (salto): .2
Valores de X
Xi-3 = x - 3h = 2.000000 - 3*0.200000 = 1.400000
Xi-2 = x - 2h = 2.000000 - 2*0.200000 = 1.600000
Xi-1 = x - h = 2.000000 - 0.200000 = 1.800000
Xi = 2.000000
Xi+1 = x + h = 2.000000 + 0.200000 = 2.200000
Xi+2 = x + 2h = 2.000000 + 2*0.200000 = 2.400000
Xi+3 = x + 3h = 2.000000 + 3*0.200000 = 2.600000

Valores f(x)
f(Xi-3) = funDev(1.400000) = 0.333136
f(Xi-2) = funDev(1.600000) = -0.074751
f(Xi-1) = funDev(1.800000) = -0.736135
f(Xi) = funDev(2.000000) = -1.664587
f(Xi+1) = funDev(2.200000) = -2.848345
f(Xi+2) = funDev(2.400000) = -4.247388
f(Xi+3) = funDev(2.600000) = -5.792568

Calculo de la cuarta derivada:
res = (-f(Xi+3) + 12*f(Xi+2) - 39*f(Xi+1) + 56*f(Xi) - 39*f(Xi-1) + 12*f(Xi-2) - f(Xi-3)) / (6 * h^4)
res = (-(-5.792568) + 12*-4.247388 - 39*-2.848345 + 56*-1.664587 - 39*-0.736135 + 12*-0.074751 - 0.333136) / (6 * (0.200000)^4)
res = (--5.792568 + -50.968654 - -111.085471 + -93.216891 - -28.709257 + -0.897009 - 0.333136) / 0.009600
res = 17.875586
El resultado de la aproximacion de la cuarta derivada es: 17.875586

Ejecución:

```
Derivadas
1.Primera derivada
2.Segunda Derivada
3.Tercera derivada
4.Cuarta derivada
5.Volver
4
Cuarta Derivada
Ingresa el valor de X: 2
Ingresa el valor de h (salto): .2
Tiempo de ejecucion: 0.000000 segundos
El resultado de la aproximacion de la cuarta derivada es: 17.875586
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Método de Boole 5/288 Modelo Simple

Maestro: Dr. Cecil Vidriales

Método de Boole 5/288 Modelo Simple

Descripción: El Método de Boole 5/288 es una fórmula de integración de orden alto que usa coeficientes específicos para mejorar la precisión.

Utilidad: Se emplea para integrar numéricamente funciones con alta precisión en modelos simples.

Funcion:

$$(b - a)/288 * [19[f(x_0) + f(x_5)] + 75f(x_1) + 50f(x_2) + 50f(x_3) + 75f(x_4)]$$

Código:

```
double b, a, x0,x1,x2,x3, x4, x5, y, h;

printf("Ingrese el intervalo cerrado para la integral\n a: ");

scanf("%lf", &a);

printf("b: ");

scanf("%lf", &b);

h=(b-a)/5;

x0=a;

x1=x0+h;

x2=x1+h;

x3=x2+h;

x4=x3+h;

x5=x4+h;

y=((b-a)/288)*(19*(fun(x0)+fun(x5)) + 75*(fun(x1) + fun(x4)) + 50*(fun(x2)+ fun(x3)));

printf("El resultado de la integral es: %lf\n", y);
```

prueba de escritorio

fun(x)=2x^3– 9x^2+ 12x - 3

Ingrese el intervalo cerrado para la integral

a: 1

b: 3

y = ((b - a) / 288) * (19 * (fun(x0) + fun(x5)) + 75 * (fun(x1) + fun(x4)) + 50 * (fun(x2) + fun(x3)));

El resultado de la integral es: 4.000000

Ejecución:

```
Metodo Boole 5/288
Elige un modelo:
1.Modelo Simple
2. Modelo de Segmentos Multiples
3.Salir
1
Metodo Boole 5/288---Modelo Simple
Ingrese el intervalo cerrado para la integral
 a: 1
b: 3
Tiempo de ejecucion: 0.000000 segundos
El resultado de la integral es: 4.000000
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Método de Boole 5/288 Segmentos Múltiples

Maestro: Dr. Cecil Vidriales

Método de Boole 5/288 Modelo de Segmentos Múltiples

Descripción: Este método extiende el Método de Boole 5/288 a múltiples segmentos para integrar funciones en intervalos más grandes.

Utilidad: Mejora la precisión en la integración de funciones sobre intervalos extensos.

Funcion:

$h = (b - a)/n$

$$5h/288 * [19[f(x_0) + f(x_n)] + 75 \sum_{i-1}^{n-4} f(x_{5i-4}) + 50 \sum_{i=1}^{n-3} f(x_{5i-3}) + 50 \sum_{i=1}^{n-2} f(x_{5i-2}) + 75 \sum_{i=1}^{n-2} f(x_{5i-1}) + 38 \sum_{i=1}^{n-5} f(x_{5i})]$$

Código:

```
double h, x0, xn, xi, n, sum1 = 0, sum2 = 0, sum3 = 0, sum4 = 0, sum5 = 0, a, b, y;
    int i;
    printf("Ingrese el intervalo cerrado para la integral\n a: ");
    scanf("%lf", &a);
    printf("b: ");
    scanf("%lf", &b);
    printf("Ingrese el número de segmentos (múltiplo de 5): ");
    scanf("%lf", &n);
    if ((int)n % 5 != 0) {
        printf("El número de segmentos debe ser múltiplo de 5.\n");
        return;
    }
    x0 = a;
    xn = b;
    h = (b - a) / n;
    for (i = 1; i <= n - 1; i++) {
        xi = x0 + i * h;
        if (i % 5 == 1) {
```

```
        sum1 += fun(xi);
      } else if (i % 5 == 2) {
        sum2 += fun(xi);
      } else if (i % 5 == 3) {
        sum3 += fun(xi);
      } else if (i % 5 == 4) {
        sum4 += fun(xi);
      } else {
        sum5 += fun(xi);
      }
    }

    y = (5 * h / 288) * (19 * (fun(x0) + fun(xn)) + 75 * (sum1 + sum5) + 50 * (sum2 + sum4) + 38 * sum3);
    printf("El resultado de la integral es: %lf\n", y);
```

Prueba de escritorio

fun(x)=2x^3– 9x^2+ 12x - 3

Ingrese el intervalo cerrado para la integral
a: 1
b: 3
Ingrese el numero de segmentos (multiplo de 5): 10
y = (5 * h / 288) * (19 * (fun(x0) + fun(xn)) + 75 * (sum1 + sum5) + 50 * (sum2 + sum4) + 38 * sum3);
El resultado de la integral es: 3.529278

Ejecución:

```
Metodo Boole 5/288
Elige un modelo:
1.Modelo Simple
2. Modelo de Segmentos Multiples
3.Salir
2
Metodo Boole 5/288---Modelo de segmentos Multiples
Ingrese el intervalo cerrado para la integral
 a: 1
b: 3
Ingrese el numero de segmentos (multiplo de 5): 10
Tiempo de ejecucion: 0.000000 segundos
El resultado de la integral es: 3.529278
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Método de Boole 2/45 Modelo Simple

Maestro: Dr. Cecil Vidriales

Método de Boole 2/45 Modelo Simple

Descripción: Una versión simplificada del Método de Boole que utiliza menos puntos de evaluación.

Utilidad: Adecuado para la integración de funciones cuando se requiere una solución rápida y razonablemente precisa.

Funcion:

$(b - a)/90 * [7f(x_0) + 7f(x_4) + 32f(x_1) + 12f(x_2) + 32f(x_3)]$

Código:

```
double b, a, x0,x1,x2,x3, x4, y, h;

        printf("Ingrese el intervalo cerrado para la integral\n a: ");

        scanf("%lf", &a);

        printf("b: ");

        scanf("%lf", &b);

        h=(b-a)/4;

        x0=a;

        x1=x0+h;

        x2=x1+h;

        x3=x2+h;

        x4=x3+h;

        y=((b-a)/90)*(7*fun(x0) + 7*fun(x4) + 32*fun(x3) + 32*fun(x1) + 12*fun(x2));

        printf("El resultado de la integral es: %lf\n", y);
```

Prueba de escritorio

fun(x)=2x^3– 9x^2+ 12x - 3

Ingrese el intervalo cerrado para la integral

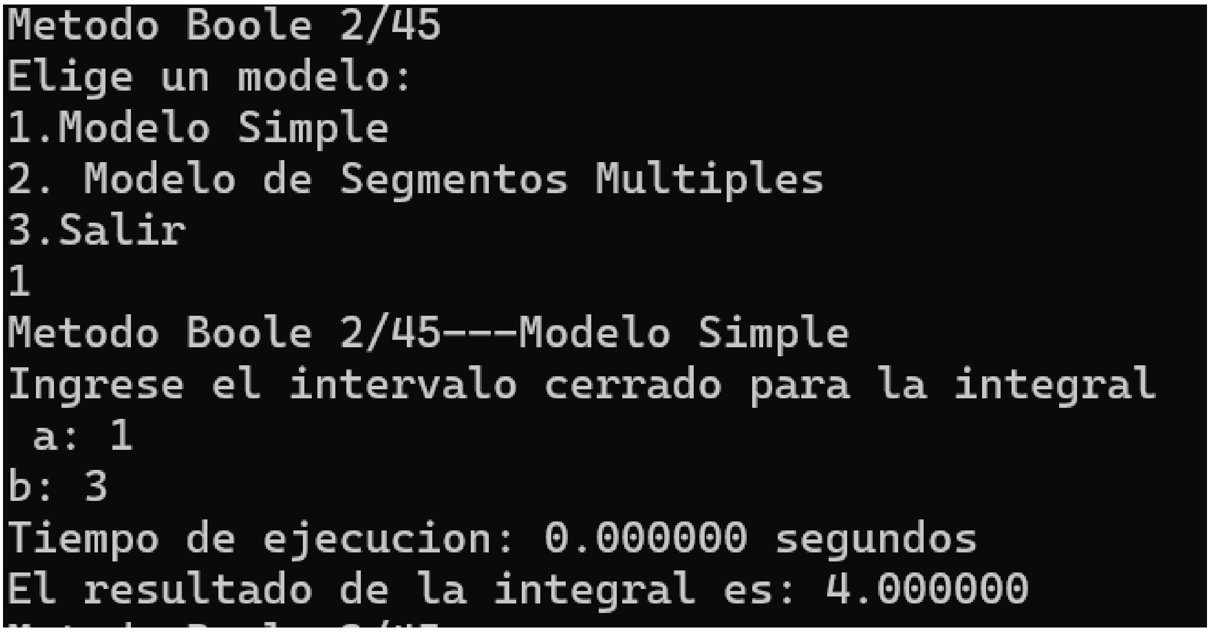
a: 1

b: 3

y = ((b - a) / 90) * (7 * fun(x0) + 7 * fun(x4) + 32 * fun(x3) + 32 * fun(x1) + 12 * fun(x2));

El resultado de la integral es: 4.000000

Ejecución:



```
Metodo Boole 2/45
Elige un modelo:
1.Modelo Simple
2. Modelo de Segmentos Multiples
3.Salir
1
Metodo Boole 2/45---Modelo Simple
Ingrese el intervalo cerrado para la integral
 a: 1
b: 3
Tiempo de ejecucion: 0.000000 segundos
El resultado de la integral es: 4.000000
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Método de Boole 2/45 Segmentos Múltiples

Maestro: Dr. Cecil Vidriales

Método de Boole 2/45 Modelo de Segmentos Múltiples

Descripción: Aplica el Método de Boole 2/45 a múltiples segmentos para mejorar la precisión en intervalos grandes.

Utilidad: Útil para la integración de funciones en intervalos extensos con una precisión razonable.

Funcion:

$h = (b - a)/n$

$2h/45 * [7f(x_0) + 7f(x_n) + 3\sum_{i=1}^{n-3} f(x_{4i-3}) + 12\sum_{i=1}^{n-2} f(x_{4i-2}) + 32\sum_{i=1}^{n-2} f(x_{4i-1}) + 14\sum_{i=1}^{n-4} f(x_{4i})]$

Código:

```
double h, x0, xn, xi, n, sum1 = 0, sum2 = 0, sum3 = 0, sum4 = 0, a, b, y;

   int i;


   printf("Ingrese el intervalo cerrado para la integral\n a: ");

   scanf("%lf", &a);

   printf("b: ");

   scanf("%lf", &b);

   printf("Ingrese el número de segmentos (múltiplo de 4): ");

   scanf("%lf", &n);


   // Verificar que n es múltiplo de 4

   if ((int)n % 4 != 0) {

       printf("El número de segmentos debe ser múltiplo de 4.\n");

       return;

   }


   x0 = a;

   xn = b;
```

```
    h = (b - a) / n;

    // Sumar las funciones evaluadas en los puntos intermedios
    for (i = 1; i <= n - 1; i++) {
        xi = x0 + i * h;
        if (i % 4 == 1) {
            sum1 += fun(xi);
        } else if (i % 4 == 2) {
            sum2 += fun(xi);
        } else if (i % 4 == 3) {
            sum3 += fun(xi);
        } else {
            sum4 += fun(xi);
        }
    }

    y = (2 * h / 45) * (7 * fun(x0) + 7 * fun(xn) + 32 * sum1 + 12 * sum2 + 32 * sum3 + 14 *
sum4);
    printf("El resultado de la integral es: %lf\n", y);
```

Prueba de escritorio

fun(x)=2x^3– 9x^2+ 12x - 3

Ingrese el intervalo cerrado para la integral
 a: 1
b: 3
Ingrese el numero de segmentos (multiplo de 4): 8
y = (2 * h / 45) * (7 * fun(x0) + 7 * fun(xn) + 32 * sum1 + 12 * sum2 + 32 * sum3 + 14 * sum4);
El resultado de la integral es: 4.000000

Ejecución:

```
Metodo Boole 2/45---Modelo de segmentos Multiples
Ingrese el intervalo cerrado para la integral
 a: 1
b: 3
Ingrese el numero de segmentos (multiplo de 4): 8
Tiempo de ejecucion: 0.000000 segundos
El resultado de la integral es: 4.000000
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Regla de Simpson ⅓ modelo simple

Maestro: Dr. Cecil Vidriales

Regla de Simpson 1/3 Modelo Simple

Descripción: La Regla de Simpson 1/3 es un método de integración que usa parábolas para aproximar el área bajo una curva.

Utilidad: Se emplea para obtener una aproximación precisa de integrales en modelos simples.

Funcion:

$$(b - a)/6 * [f(x_0) + 4f(x_1) + f(x_n)]$$

Código:

```
double b, a, x0,x1,x2, y, h;

printf("Ingrese el intervalo cerrado para la integral\n a: ");

scanf("%lf", &a);

printf("b: ");

scanf("%lf", &b);

h=(b-a)/2;

x0=a;

x1=x0+h;

x2=x1+h;

y=((b-a)/6)*(fun(x0) + 4*fun(x1) + fun(x2));

printf("el resultado de la integral es: %lf\n", y);
```

Prueba de escritorio

fun(x)=2x^3– 9x^2+ 12x - 3

Ingrese el intervalo cerrado para la integral
 a: 1
b: 3

Cálculos intermedios:
h = (b - a) / 2 = (3.000000 - 1.000000) / 2 = 1.000000
x0 = a = 1.000000
x1 = x0 + h = 1.000000 + 1.000000 = 2.000000
x2 = x1 + h = 2.000000 + 1.000000 = 3.000000
fun(x0) = fun(1.000000) = 2.000000
fun(x1) = fun(2.000000) = 1.000000
fun(x2) = fun(3.000000) = 6.000000

Formula: y = ((b - a) / 6) * (fun(x0) + 4 * fun(x1) + fun(x2))
y = ((3.000000 - 1.000000) / 6) * (2.000000 + 4 * 1.000000 + 6.000000)
y = (2.000000 / 6) * (2.000000 + 4.000000 + 6.000000)
y = 4.000000

El resultado de la integral es: 4.000000

Ejecución:

```
Regla de Simpson 1/3
Elige un modelo:
 1.Modelo Simple
2. Modelo de Segmentos Multiples
3.Salir
1
Regla de Simpson 1/3---Modelo Simple
Ingrese el intervalo cerrado para la integral
 a: 1
b: 3
Tiempo de ejecucion: 0.000000 segundos
el resultado de la integral es: 4.000000
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Regla de Simpson ⅓ Segmentos Múltiples

Maestro: Dr. Cecil Vidriales

Regla de Simpson 1/3 Segmentos Múltiples

Descripción: Extiende la Regla de Simpson 1/3 a múltiples segmentos para mejorar la precisión en intervalos más grandes.

Utilidad: Mejora la precisión de la integración en intervalos amplios.

funcion:

$$h = (b - a)/n$$

$$(h/3) * [f(x_0) + 4 \sum_{i-1}^{n-1} f(x_{2i-1}) + 2 \sum_{i=1}^{n-1} f(x_{2i}) + f(x_n)]$$

Código:

```
double h, x0, xn, xi, n, sum1 = 0, sum2 = 0, a, b, y;

    int i;



    printf("Ingrese el intervalo cerrado para la integral\n a: ");

    scanf("%lf", &a);

    printf("b: ");

    scanf("%lf", &b);

    printf("Ingrese el numero de segmentos (debe ser par): ");

    scanf("%lf", &n);



    // Verificar que n es par

    if ((int)n % 2 != 0) {

        printf("El numero de segmentos debe ser par.\n");

        return;

    }



    x0 = a;
```

```
xn = b;
h = (b - a) / n;

// Sumar las funciones evaluadas en los puntos impares
for (i = 1; i <= n - 1; i += 2) {
    sum1 += fun(x0 + i * h);
}

// Sumar las funciones evaluadas en los puntos pares
for (i = 2; i <= n - 2; i += 2) {
    sum2 += fun(x0 + i * h);
}

y = (h / 3) * (fun(x0) + fun(xn) + 4 * sum1 + 2 * sum2);
printf("El resultado de la integral es: %lf\n", y);
```

Prueba de escritorio

fun(x)=2x^3– 9x^2+ 12x - 3

Ingrese el intervalo cerrado para la integral

a: 1

b: 3

Ingrese el número de segmentos (debe ser par): 6

Cálculos intermedios:

h = (b - a) / n = (3.000000 - 1.000000) / 6.000000 = 0.333333

i = 1, xi = x0 + i * h = 1.000000 + 1 * 0.333333 = 1.333333, fun(xi) = 1.740741

i = 3, xi = x0 + i * h = 1.000000 + 3 * 0.333333 = 2.000000, fun(xi) = 1.000000

i = 5, xi = x0 + i * h = 1.000000 + 5 * 0.333333 = 2.666667, fun(xi) = 2.925926

i = 2, xi = x0 + i * h = 1.000000 + 2 * 0.333333 = 1.666667, fun(xi) = 1.259259

i = 4, xi = x0 + i * h = 1.000000 + 4 * 0.333333 = 2.333333, fun(xi) = 1.407407

Formula: y = (h / 3) * (fun(x0) + fun(xn) + 4 * sum1 + 2 * sum2)

y = (0.333333 / 3) * (2.000000 + 6.000000 + 4 * 5.666667 + 2 * 2.666667)

El resultado de la integral es: 4.000000

Ejecución:

```
Regla de Simpson 1/3
Elige un modelo:
 1.Modelo Simple
2. Modelo de Segmentos Multiples
3.Salir
2
Regla de Simpson 1/3---Modelo de segmentos Multiples
Ingrese el intervalo cerrado para la integral
 a: 1
b: 3
Ingrese el numero de segmentos (debe ser par): 6
Tiempo de ejecucion: 0.000000 segundos
El resultado de la integral es: 4.000000
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Método Simpson 3/8 modelo simple

Maestro: Dr. Cecil Vidriales

Método de Simpson 3/8 Modelo Simple

Descripción: El Método de Simpson 3/8 utiliza parábolas cúbicas para aproximar integrales, mejorando la precisión.

Utilidad: Se usa para obtener una aproximación más precisa que la Regla de Simpson 1/3 en modelos simples.

Funcion:

$(b - a)/8 * [f(x_0) + f(x_3) + 3f(x_1) + 3f(x_2)]$

Código:

```
double b, a, x0,x1,x2,x3, y, h;

	printf("Ingrese el intervalo cerrado para la integral\n a: ");

	scanf("%lf", &a);

	printf("b: ");

	scanf("%lf", &b);

	h=(b-a)/3;

	x0=a;

	x1=x0+h;

	x2=x1+h;

	x3=x2+h;

	y=((b-a)/8)*(fun(x0)+fun(x3) + 3*fun(x1) + 3*fun(x2));

	printf("El resultado de la integral es: %lf\n", y);
```

Prueba de escritorio

fun(x)=2x^3– 9x^2+ 12x - 3

Ingrese el intervalo cerrado para la integral
 a: 1
b: 3

Cálculos intermedios:
h = (b - a) / 3 = (3.000000 - 1.000000) / 3 = 0.666667
y = ((b - a) / 8) * (fun(x0) + fun(x3) + 3 * fun(x1) + 3 * fun(x2))
  = ((3.000000 - 1.000000) / 8) * (2.000000 + 6.000000 + 3 * 1.259259 + 3 * 1.407407)
El resultado de la integral es: 4.000000

Ejecución:

```
Metodo de Simpson 3/8
Elige un modelo:
1.Modelo Simple
2. Modelo de Segmentos Multiples
3.Salir
1
Metodo de Simpson 3/8---Modelo Simple
Ingrese el intervalo cerrado para la integral
 a: 1
b: 3
Tiempo de ejecucion: 0.000000 segundos
El resultado de la integral es: 4.000000
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Método Simpson $\frac{3}{8}$ Segmentos Múltiples

Maestro: Dr. Cecil Vidriales

Método de Simpson 3/8 Segmentos Múltiples

Descripción: Aplica el Método de Simpson 3/8 a múltiples segmentos, aumentando la precisión en intervalos grandes.

Utilidad: Ofrece una mejor precisión en la integración de funciones sobre intervalos amplios.

Funcion:

$$h = (b - a)/n$$

$$3h/8 * [f(x_0) + f(x_n) + 3\sum_{i-1}^{n-2} f(x_{3i-2}) + 3\sum_{i=1}^{n-1} f(x_{3i-i}) + 2\sum_{i=1}^{n-3} f(x_{3i})]$$

Código:

```
double h, x0, xn, xi, n, sum1 = 0, sum2 = 0, sum3 = 0, a, b, y;

  int i;


  printf("Ingrese el intervalo cerrado para la integral\n a: ");

  scanf("%lf", &a);

  printf("b: ");

  scanf("%lf", &b);

  printf("Ingrese el número de segmentos (múltiplo de 3): ");

  scanf("%lf", &n);


  // Verificar que n es múltiplo de 3

  if ((int)n % 3 != 0) {

    printf("El número de segmentos debe ser múltiplo de 3.\n");

    return;

  }


  x0 = a;
```

```
xn = b;
h = (b - a) / n;

// Sumar las funciones evaluadas en los puntos intermedios
for (i = 1; i <= n - 1; i++) {
    xi = x0 + i * h;
    if (i % 3 == 0) {
        sum3 += fun(xi);
    } else {
        sum1 += fun(xi);
    }
}

y = (3 * h / 8) * (fun(x0) + fun(xn) + 2 * sum3 + 3 * sum1);
printf("El resultado de la integral es: %lf\n", y);
```

Prueba de escritorio

fun(x)=2x^3– 9x^2+ 12x - 3

Ingrese el intervalo cerrado para la integral

a: 1

b: 3

Ingrese el número de segmentos (múltiplo de 3): 9

xi[1] = 1.222222, fun(xi[1]) = 1.873800

xi[2] = 1.444444, fun(xi[2]) = 1.582990

xi[3] = 1.666667, fun(xi[3]) = 1.259259

xi[4] = 1.888889, fun(xi[4]) = 1.034294

xi[5] = 2.111111, fun(xi[5]) = 1.039781

xi[6] = 2.333333, fun(xi[6]) = 1.407407

xi[7] = 2.555556, fun(xi[7]) = 2.268861

xi[8] = 2.777778, fun(xi[8]) = 3.755830

y = (3 * 0.222222 / 8) * (2.000000 + 6.000000 + 2 * 2.666667 + 3 * 11.555556)

El resultado de la integral es: 4.000000

Ejecución:

```
Metodo de Simpson 3/8
Elige un modelo:
1.Modelo Simple
2. Modelo de Segmentos Multiples
3.Salir
2
Metodo de Simpson 3/8---Modelo de segmentos Multiples
Ingrese el intervalo cerrado para la integral
 a: 1
b: 3
Ingrese el numero de segmentos (multiplo de 3): 9
Tiempo de ejecucion: 0.000000 segundos
El resultado de la integral es: 4.000000
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Regla Trapezoidal Modelo Simple

Maestro: Dr. Cecil Vidriales

Regla Trapezoidal Modelo Simple

Descripción: La Regla Trapezoidal es un método de integración que aproxima el área bajo una curva como una serie de trapezoides.

Utilidad: Se utiliza para integrar funciones de manera simple y rápida, aunque con menor precisión que otros métodos.

Funcion:

$$((b - a)/a) * [f(x_0 + f(x_i)]$$

Código:

```
double b, a, x0,x1, y;

printf("Ingrese el intervalo cerrado para la integral\n a: ");

scanf("%lf", &a);

printf("b: ");

scanf("%lf", &b);

x0=a;

x1=b;

y = ((b-a)/2)*(fun(x0)+fun(x1));

printf("el resultado de la integral es: %lf \n", y);
```

Prueba de escritorio

fun(x)=2x^3– 9x^2+ 12x - 3

Ingrese el intervalo cerrado para la integral

a: 1

b: 3

Intervalo: [1.000000, 3.000000]

f(1.000000) = 2.000000

f(3.000000) = 6.000000

El resultado de la integral es: 8.000000

Ejecución:

```
Regla trapezoidal
Elige un modelo:
1.Modelo Simple
2. Modelo de Segmentos Multiples
3.Salir
1
Regla trapezoidal---Modelo Simple
Ingrese el intervalo cerrado para la integral
 a: 1
b: 3
Tiempo de ejecucion: 0.000000 segundos
el resultado de la integral es: 8.000000
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Regla Trapezoidal

Modelo de Segmentos Múltiples

Maestro: Dr. Cecil Vidriales

Regla Trapezoidal Modelo de Segmentos Múltiples

Descripción: Extiende la Regla Trapezoidal a múltiples segmentos, mejorando la precisión en intervalos más grandes.

Utilidad: Útil para la integración de funciones sobre intervalos extensos con una precisión razonable.

Funcion:

$$h = (b - a)/n$$

$$(h/2) * [f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)]$$

Código:

```
double h, x0, xn, xi, n, sum, a,b, y;

printf("Ingrese el intervalo cerrado para la integral\n a: ");

scanf("%lf", &a);

printf("b: ");

scanf("%lf", &b);

printf("Ingrese el numero de segmentos (2,3,4,...):");

scanf("%lf", &n);

x0=a;

xi=x0;

xn=b;

h=(b-a)/n;

for(int i=1; i<=n-1; i++){

    sum+=fun(xi+ h*i);

}

y=(h/2)*(fun(x0)+fun(xn)+2*sum);

printf("el resultado de la integral es: %lf \n", y);
```

Prueba de escritorio

fun(x)=2x^3– 9x^2+ 12x - 3

Ingrese el intervalo cerrado para la integral
 a: 1
b: 3
Ingrese el numero de segmentos (2, 3, 4, ...): 6
Intervalo: [1.000000, 3.000000]
Numero de segmentos: 6.000000
h: 0.333333
f(1.333333) = 1.740741
f(1.666667) = 1.259259
f(2.000000) = 1.000000
f(2.333333) = 1.407407
f(2.666667) = 2.925926
El resultado de la integral es: 4.111111

Ejecución:

```
Regla trapezoidal---Modelo de segmentos Multiples
Ingrese el intervalo cerrado para la integral
 a: 1
b: 3
Ingrese el numero de segmentos (2,3,4,...):6
Tiempo de ejecucion: 0.000000 segundos
el resultado de la integral es: 4.111111
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

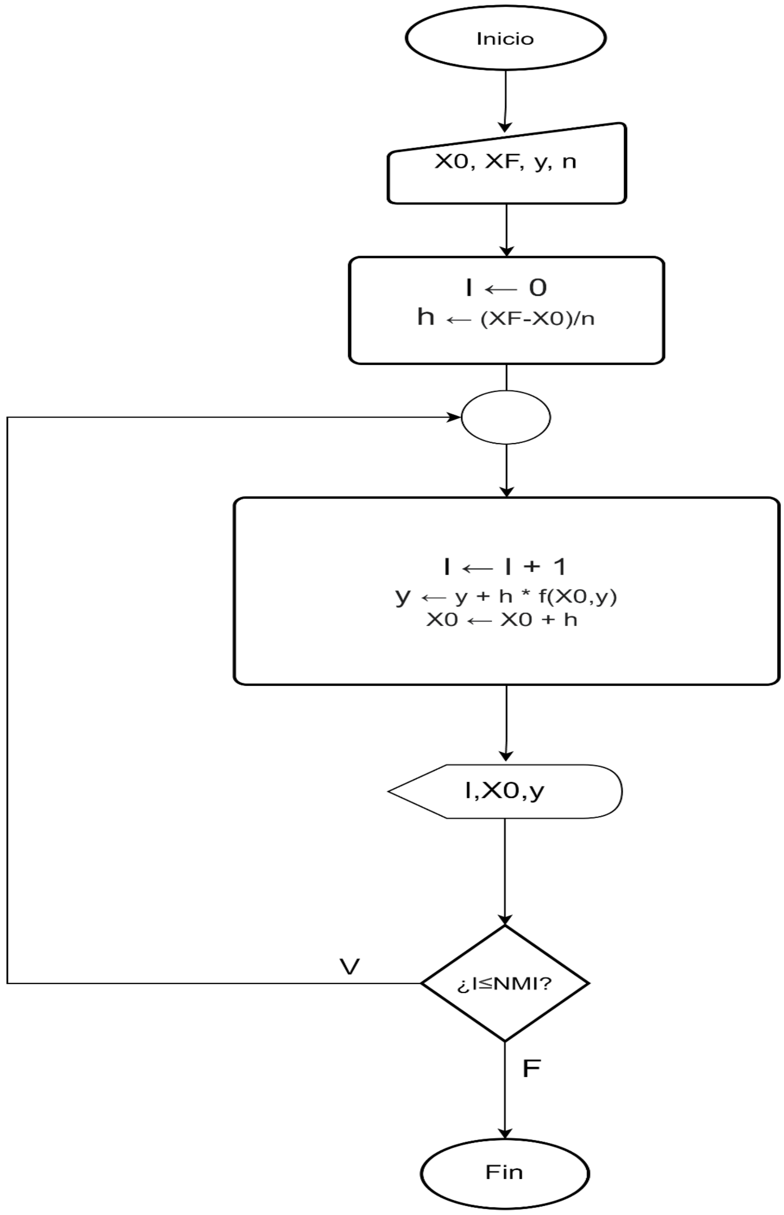
Algoritmo de Euler

Maestro: Dr. Cecil Vidriales

Algoritmo de Euler

Descripción: El Método de Euler es una técnica numérica sencilla para resolver ecuaciones diferenciales ordinarias de primer orden, usando una aproximación lineal en pequeños pasos discretos.

Utilidad: Sirve para obtener soluciones aproximadas a EDOs de manera rápida y fácil, aunque con una precisión limitada que depende del tamaño del paso.



Pseudocódigo del Algoritmo de Euler

1. Inicio
2. Inicializar Variables:
   - x = x0 (valor inicial de la variable independiente)
   - y (valor inicial de la solución, se asume que ya está dado)
   - xf (valor final de la variable independiente)
   - n (número de pasos)
   - h = (xf - x0) / n (tamaño del paso)
3. Bucle For (i = 0; i < n; i++):
   - Calcular la derivada en el punto actual: f = f(x, y)
   - Actualizar el valor de y: y = y + h * f
   - Actualizar el valor de x: x = x + h
   - Imprimir (opcional): Imprimir x, y
4. Fin del Bucle For
5. Fin

Explicación del Pseudocódigo

1. Inicio: Comienza el algoritmo.
2. Inicializar Variables:
   - x = x0: Asigna el valor inicial de la variable independiente.
   - y: Asigna el valor inicial de la solución (se asume que ya está dado o previamente calculado).
   - xf: Define el valor final de la variable independiente.
   - n: Establece el número de pasos que se van a calcular.
   - h = (xf - x0) / n: Calcula el tamaño del paso dividiendo el intervalo [x0, xf] en n partes iguales.
3. Bucle For: Se repite n veces para calcular los valores de y en cada paso.
   - Calcular la derivada: f = f(x, y): Evalúa la derivada de y con respecto a x en el punto actual.
   - Actualizar el valor de y: y = y + h * f: Calcula el nuevo valor de y usando la fórmula de Euler.
   - Actualizar el valor de x: x = x + h: Avanza al siguiente punto en el tiempo.
   - Imprimir (opcional): Se puede imprimir los valores de x y y.
4. Fin del Bucle For
5. Fin

Código:

```
double x0, xf, y, h, n;
    printf("Ingresa el valor de X0: ");
    scanf("%lf", &x0);

    printf("Ingresa el valor de XF: ");
    scanf("%lf", &xf);

    printf("Ingresa el valor de Y: ");
    scanf("%lf", &y);

    printf("Ingresa el numero de iteraciones: ");
    scanf("%lf", &n);

    h = (xf - x0) / n;

    for (int i = 0; i < n; ++i) {
        y = y + h * f(x0, y);
        x0 = x0 + h;
    }

    printf("El valor de Y en XF es: %lf\n", y);
```

Prueba de escritorio

```
f(x,y)=sin(x) + y;
Ingresa el valor de X0: 0
Ingresa el valor de XF: 4
Ingresa el valor de Y: 2
Ingresa el numero de iteraciones: 20
Iteracion 1:
  y = y + 0.200000 * f(0.000000, 1.920000);
  x0 = x0 + 0.200000;
  y = 2.400000
Iteracion 2:
  y = y + 0.200000 * f(0.200000, 2.296053);
  x0 = x0 + 0.200000;
  y = 2.919734
Iteracion 3:
  y = y + 0.200000 * f(0.400000, 2.787368);
  x0 = x0 + 0.200000;
  y = 3.581564
Iteracion 4:
  y = y + 0.200000 * f(0.600000, 3.415716);
  x0 = x0 + 0.200000;
  y = 4.410806
Iteracion 5:
  y = y + 0.200000 * f(0.800000, 4.205679);
```

```
  x0 = x0 + 0.200000;
  y = 5.436438
Iteracion 6:
  y = y + 0.200000 * f(1.000000, 5.185322);
  x0 = x0 + 0.200000;
  y = 6.692020
Iteracion 7:
  y = y + 0.200000 * f(1.200000, 6.387057);
  x0 = x0 + 0.200000;
  y = 8.216832
Iteracion 8:
  y = y + 0.200000 * f(1.400000, 7.848740);
  x0 = x0 + 0.200000;
  y = 10.057288
Iteracion 9:
  y = y + 0.200000 * f(1.600000, 9.615013);
  x0 = x0 + 0.200000;
  y = 12.268660
Iteracion 10:
  y = y + 0.200000 * f(1.800000, 11.738960);
  x0 = x0 + 0.200000;
  y = 14.917162
Iteracion 11:
  y = y + 0.200000 * f(2.000000, 14.284103);
  x0 = x0 + 0.200000;
  y = 18.082454
Iteracion 12:
  y = y + 0.200000 * f(2.200000, 17.326816);
  x0 = x0 + 0.200000;
  y = 21.860644
Iteracion 13:
  y = y + 0.200000 * f(2.400000, 20.959199);
  x0 = x0 + 0.200000;
  y = 26.367865
Iteracion 14:
  y = y + 0.200000 * f(2.600000, 25.292530);
  x0 = x0 + 0.200000;
  y = 31.744538
Iteracion 15:
  y = y + 0.200000 * f(2.800000, 30.461357);
  x0 = x0 + 0.200000;
  y = 38.160443
Iteracion 16:
  y = y + 0.200000 * f(3.000000, 36.628381);
  x0 = x0 + 0.200000;
  y = 45.820756
Iteracion 17:
  y = y + 0.200000 * f(3.200000, 43.990261);
```

```
  x0 = x0 + 0.200000;
  y = 54.973232
Iteracion 18:
  y = y + 0.200000 * f(3.400000, 52.784525);
  x0 = x0 + 0.200000;
  y = 65.916771
Iteracion 19:
  y = y + 0.200000 * f(3.600000, 63.297801);
  x0 = x0 + 0.200000;
  y = 79.011621
Iteracion 20:
  y = y + 0.200000 * f(3.800000, 75.875630);
  x0 = x0 + 0.200000;
  y = 94.691573
El valor de Y en XF es: 94.691573
```

Ejecución:

```
Algoritmo de Euler
Ingresa el valor de X0: 0
Ingresa el valor de XF: 4
Ingresa el valor de Y: 2
Ingresa el numero de iteraciones: 20
Tiempo de ejecucion: 0.000000 segundos
El valor de Y en XF es: 94.691573
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Euler-Gauss

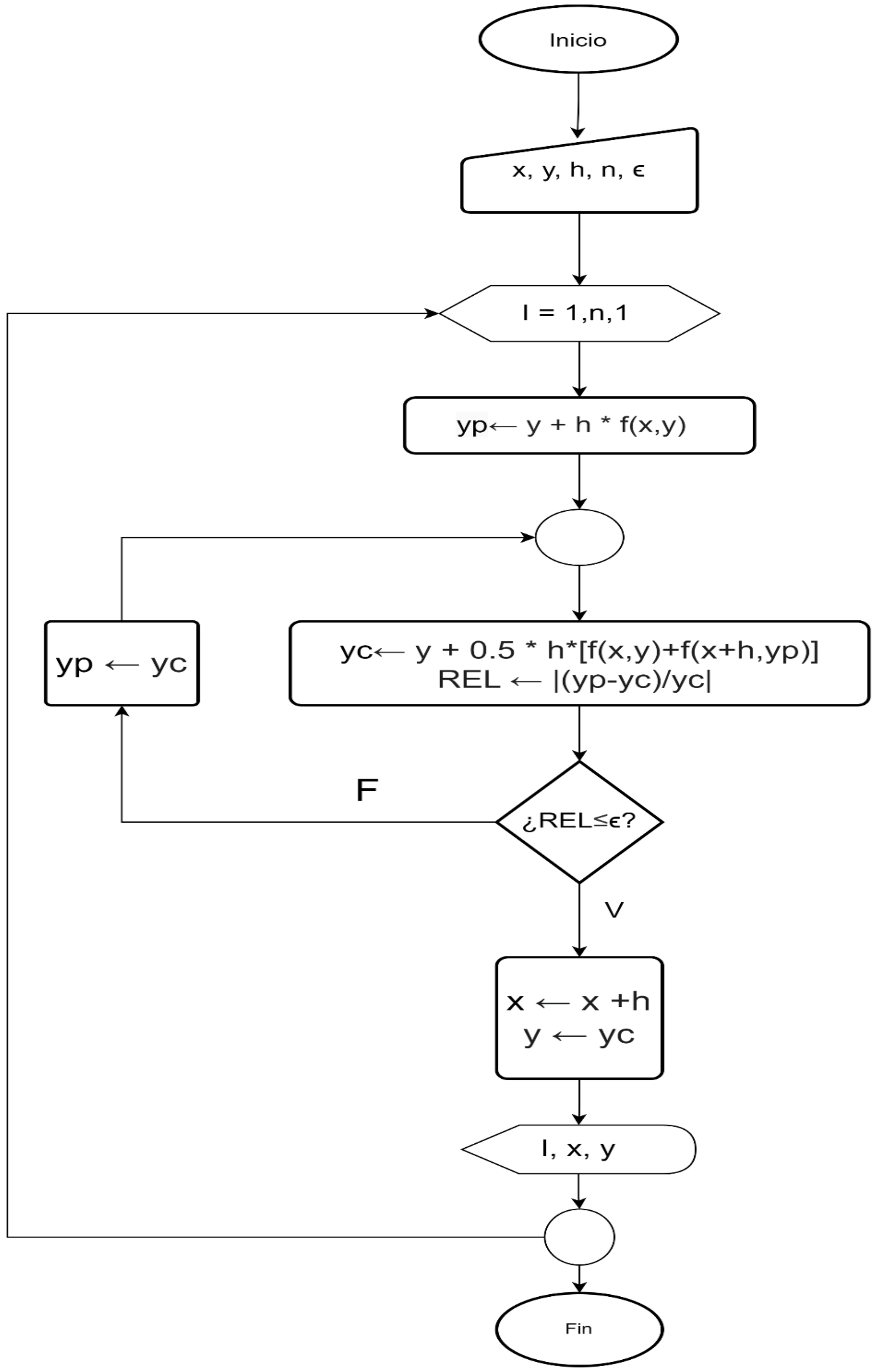
Proyecto Final

Maestro: Dr. Cecil Vidriales

Algoritmo de Euler-Gauss

Descripción: El Método de Euler-Gauss mejora el Método de Euler al promediar las pendientes al inicio y al final del intervalo de integración.

Utilidad: Proporciona una mejor precisión en la resolución de EDOs en comparación con el Método de Euler simple.



Pseudocódigo Euler-Gauss

1. Inicio: Comienza el algoritmo.
2. Inicializar Variables:
    - x = x0: Asigna el valor inicial de la variable independiente.
    - y: Asigna el valor inicial de la solución (se asume que ya está dado o previamente calculado).
    - xf: Define el valor final de la variable independiente.
    - n: Establece el número de pasos que se van a calcular.
    - h = (xf - x0) / n: Calcula el tamaño del paso dividiendo el intervalo [x0, xf] en n partes iguales.
3. Bucle For: Se repite n veces para calcular los valores de y en cada paso.
    - Calcular la derivada: k1 = f(x, y): Evalúa la derivada de y con respecto a x en el punto actual.
    - Calcular la predicción del valor de x y y en el punto medio:
        - xm = x + h / 2: Calcula el punto medio de x.
        - ym = y + h / 2 * k1: Calcula el valor de y en el punto medio usando k1.
    - Calcular la derivada en el punto medio: k2 = f(xm, ym): Evalúa la derivada de y con respecto a x en el punto medio.
    - Actualizar el valor de y usando la derivada del punto medio: y = y + h * k2: Usa k2 para actualizar y.
    - Actualizar el valor de x: x = x + h: Avanza al siguiente punto en el tiempo.
    - Imprimir (opcional): Se puede imprimir los valores de x y y.
4. Fin del Bucle For
5. Fin

Código:

```
double x, y, yp, yc, rel, h, n, a, b, e;
   printf("Ingresa el valor de X: ");
   scanf("%lf", &x);

   printf("Ingresa el valor de a: ");
   scanf("%lf", &a);

   printf("Ingresa el valor de b: ");
   scanf("%lf", &b);

   printf("Ingresa el valor de Y: ");
   scanf("%lf", &y);

   printf("Ingresa el numero de iteraciones: ");
   scanf("%lf", &n);

   printf("Ingresa el valor del error: ");
   scanf("%lf", &e);

   h = (b - a) / n;
```

```
for (int i = 1; i <= n; ++i) {
    yp = y + h * f(x, y);
    do {
        yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
        rel = (yp - yc) / yc;
        if (rel < 0)
            rel *= -1;
        if (rel > e)
            yp = yc;
    } while (rel > e);

    // Actualiza el valor de y con el valor corregido
    y = yc;
    // Incrementa x
    x = x + h;
    // Imprime los valores de x y y en cada iteración
    printf("x = %lf, y = %lf\n", x, y);
}
```

Prueba de escritorio

```
f(x,y)=sin(x) + y;
Ingresa el valor de X: 2
Ingresa el valor de a: 0
Ingresa el valor de b: 4
Ingresa el valor de Y: 0
Ingresa el numero de iteraciones: 20
Ingresa el valor del error: 0.01

Iteracion 1:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 0.000000 + 0.200000 * f(2.000000, 0.000000);
    yp = 0.181859
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 0.000000 + 0.5 * 0.200000 * (f(2.000000, 0.000000) + f(2.200000, 0.181859));
      yc = 0.189965
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 0.000000 + 0.5 * 0.200000 * (f(2.000000, 0.000000) + f(2.200000, 0.189965));
      yc = 0.190776
  Actualizando valores:
    y = 0.190776
    x = 2.200000
```

```
Iteracion 2:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 0.190776 + 0.200000 * f(2.200000, 0.190776);
    yp = 0.390630
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 0.190776 + 0.5 * 0.200000 * (f(2.200000, 0.190776) + f(2.400000, 0.390630));
      yc = 0.397313
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 0.190776 + 0.5 * 0.200000 * (f(2.200000, 0.190776) + f(2.400000, 0.397313));
      yc = 0.397981
  Actualizando valores:
    y = 0.397981
    x = 2.400000

Iteracion 3:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 0.397981 + 0.200000 * f(2.400000, 0.397981);
    yp = 0.612669
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 0.397981 + 0.5 * 0.200000 * (f(2.400000, 0.397981) + f(2.600000, 0.612669));
      yc = 0.618142
  Actualizando valores:
    y = 0.618142
    x = 2.600000

Iteracion 4:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 0.618142 + 0.200000 * f(2.600000, 0.618142);
    yp = 0.844871
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 0.618142 + 0.5 * 0.200000 * (f(2.600000, 0.618142) + f(2.800000, 0.844871));
      yc = 0.849492
  Actualizando valores:
    y = 0.849492
    x = 2.800000
```

```
Iteracion 5:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 0.849492 + 0.200000 * f(2.800000, 0.849492);
    yp = 1.086389
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 0.849492 + 0.5 * 0.200000 * (f(2.800000, 0.849492) + f(3.000000, 1.086389));
      yc = 1.090691
  Actualizando valores:
    y = 1.090691
    x = 3.000000

Iteracion 6:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 1.090691 + 0.200000 * f(3.000000, 1.090691);
    yp = 1.337054
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 1.090691 + 0.5 * 0.200000 * (f(3.000000, 1.090691) + f(3.200000, 1.337054));
      yc = 1.341740
  Actualizando valores:
    y = 1.341740
    x = 3.200000

Iteracion 7:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 1.341740 + 0.200000 * f(3.200000, 1.341740);
    yp = 1.598414
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 1.341740 + 0.5 * 0.200000 * (f(3.200000, 1.341740) + f(3.400000, 1.598414));
      yc = 1.604364
  Actualizando valores:
    y = 1.604364
    x = 3.400000

Iteracion 8:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 1.604364 + 0.200000 * f(3.400000, 1.604364);
    yp = 1.874129
  Aplicando correccion de Gauss-Seidel:
```

```
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 1.604364 + 0.5 * 0.200000 * (f(3.400000, 1.604364) + f(3.600000, 1.874129));
      yc = 1.882408
  Actualizando valores:
    y = 1.882408
    x = 3.600000

Iteracion 9:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 1.882408 + 0.200000 * f(3.600000, 1.882408);
    yp = 2.170385
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 1.882408 + 0.5 * 0.200000 * (f(3.600000, 1.882408) + f(3.800000, 2.170385));
      yc = 2.182249
  Actualizando valores:
    y = 2.182249
    x = 3.800000

Iteracion 10:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 2.182249 + 0.200000 * f(3.800000, 2.182249);
    yp = 2.496327
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 2.182249 + 0.5 * 0.200000 * (f(3.800000, 2.182249) + f(4.000000, 2.496327));
      yc = 2.513241
  Actualizando valores:
    y = 2.513241
    x = 4.000000

Iteracion 11:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 2.513241 + 0.200000 * f(4.000000, 2.513241);
    yp = 2.864528
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 2.513241 + 0.5 * 0.200000 * (f(4.000000, 2.513241) + f(4.200000, 2.864528));
      yc = 2.888180
  Actualizando valores:
    y = 2.888180
```

```
    x = 4.200000

  Iteracion 12:
    Calculando yp:
      yp = y + h * f(x, y);
      yp = 2.888180 + 0.200000 * f(4.200000, 2.888180);
      yp = 3.291500
    Aplicando correccion de Gauss-Seidel:
      Calculando yc:
        yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
        yc = 2.888180 + 0.5 * 0.200000 * (f(4.200000, 2.888180) + f(4.400000, 3.291500));
        yc = 3.323830
    Actualizando valores:
      y = 3.323830
      x = 4.400000

  Iteracion 13:
    Calculando yp:
      yp = y + h * f(x, y);
      yp = 3.323830 + 0.200000 * f(4.400000, 3.323830);
      yp = 3.798275
    Aplicando correccion de Gauss-Seidel:
      Calculando yc:
        yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
        yc = 3.323830 + 0.5 * 0.200000 * (f(4.400000, 3.323830) + f(4.600000, 3.798275));
        yc = 3.841511
    Aplicando correccion de Gauss-Seidel:
      Calculando yc:
        yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
        yc = 3.323830 + 0.5 * 0.200000 * (f(4.400000, 3.323830) + f(4.600000, 3.841511));
        yc = 3.845835
    Actualizando valores:
      y = 3.845835
      x = 4.600000

  Iteracion 14:
    Calculando yp:
      yp = y + h * f(x, y);
      yp = 3.845835 + 0.200000 * f(4.600000, 3.845835);
      yp = 4.416263
    Aplicando correccion de Gauss-Seidel:
      Calculando yc:
        yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
        yc = 3.845835 + 0.5 * 0.200000 * (f(4.600000, 3.845835) + f(4.800000, 4.416263));
        yc = 4.473059
    Aplicando correccion de Gauss-Seidel:
      Calculando yc:
        yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
```

```
      yc = 3.845835 + 0.5 * 0.200000 * (f(4.600000, 3.845835) + f(4.800000, 4.473059));
      yc = 4.478738
  Actualizando valores:
    y = 4.478738
    x = 4.800000

Iteracion 15:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 4.478738 + 0.200000 * f(4.800000, 4.478738);
    yp = 5.175253
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 4.478738 + 0.5 * 0.200000 * (f(4.800000, 4.478738) + f(5.000000, 5.175253));
      yc = 5.248629
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 4.478738 + 0.5 * 0.200000 * (f(4.800000, 4.478738) + f(5.000000, 5.248629));
      yc = 5.255966
  Actualizando valores:
    y = 5.255966
    x = 5.000000

Iteracion 16:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 5.255966 + 0.200000 * f(5.000000, 5.255966);
    yp = 6.115374
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 5.255966 + 0.5 * 0.200000 * (f(5.000000, 5.255966) + f(5.200000, 6.115374));
      yc = 6.208862
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 5.255966 + 0.5 * 0.200000 * (f(5.000000, 5.255966) + f(5.200000, 6.208862));
      yc = 6.218211
  Actualizando valores:
    y = 6.218211
    x = 5.200000

Iteracion 17:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 6.218211 + 0.200000 * f(5.200000, 6.218211);
```

```
    yp = 7.285162
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 6.218211 + 0.5 * 0.200000 * (f(5.200000, 6.218211) + f(5.400000, 7.285162));
      yc = 7.402926
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 6.218211 + 0.5 * 0.200000 * (f(5.200000, 6.218211) + f(5.400000, 7.402926));
      yc = 7.414703
  Actualizando valores:
    y = 7.414703
    x = 5.400000

Iteracion 18:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 7.414703 + 0.200000 * f(5.400000, 7.414703);
    yp = 8.743091
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 7.414703 + 0.5 * 0.200000 * (f(5.400000, 7.414703) + f(5.600000, 8.743091));
      yc = 8.890079
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 7.414703 + 0.5 * 0.200000 * (f(5.400000, 7.414703) + f(5.600000, 8.890079));
      yc = 8.904778
  Actualizando valores:
    y = 8.904778
    x = 5.600000

Iteracion 19:
  Calculando yp:
    yp = y + h * f(x, y);
    yp = 8.904778 + 0.200000 * f(5.600000, 8.904778);
    yp = 10.559480
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 8.904778 + 0.5 * 0.200000 * (f(5.600000, 8.904778) + f(5.800000, 10.559480));
      yc = 10.741617
  Aplicando correccion de Gauss-Seidel:
    Calculando yc:
      yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
      yc = 8.904778 + 0.5 * 0.200000 * (f(5.600000, 8.904778) + f(5.800000, 10.741617));
```

```
    yc = 10.759831
  Actualizando valores:
   y = 10.759831
   x = 5.800000

Iteracion 20:
  Calculando yp:
   yp = y + h * f(x, y);
   yp = 10.759831 + 0.200000 * f(5.800000, 10.759831);
   yp = 12.818876
  Aplicando correccion de Gauss-Seidel:
   Calculando yc:
    yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
    yc = 10.759831 + 0.5 * 0.200000 * (f(5.800000, 10.759831) + f(6.000000, 12.818876));
    yc = 13.043300
  Aplicando correccion de Gauss-Seidel:
   Calculando yc:
    yc = y + 0.5 * h * (f(x, y) + f(x + h, yp));
    yc = 10.759831 + 0.5 * 0.200000 * (f(5.800000, 10.759831) + f(6.000000, 13.043300));
    yc = 13.065742
  Actualizando valores:
   y = 13.065742
   x = 6.000000

Resultado final:
  El valor de Y en XF es: 13.065742
```

Ejecución:

```
Algoritmo de Euler-Gauss
Ingresa el valor de X: 2
Ingresa el valor de a: 0
Ingresa el valor de b: 4
Ingresa el valor de Y: 0
Ingresa el numero de iteraciones: 20
Ingresa el valor del error: 0.01
x = 2.200000, y = 0.190776
x = 2.400000, y = 0.397981
x = 2.600000, y = 0.618142
x = 2.800000, y = 0.849492
x = 3.000000, y = 1.090691
x = 3.200000, y = 1.341740
x = 3.400000, y = 1.604364
x = 3.600000, y = 1.882408
x = 3.800000, y = 2.182249
x = 4.000000, y = 2.513241
x = 4.200000, y = 2.888180
x = 4.400000, y = 3.323830
x = 4.600000, y = 3.845835
x = 4.800000, y = 4.478738
x = 5.000000, y = 5.255966
x = 5.200000, y = 6.218211
x = 5.400000, y = 7.414703
x = 5.600000, y = 8.904778
x = 5.800000, y = 10.759831
x = 6.000000, y = 13.065742
Tiempo de ejecucion: 0.010000 segundos
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

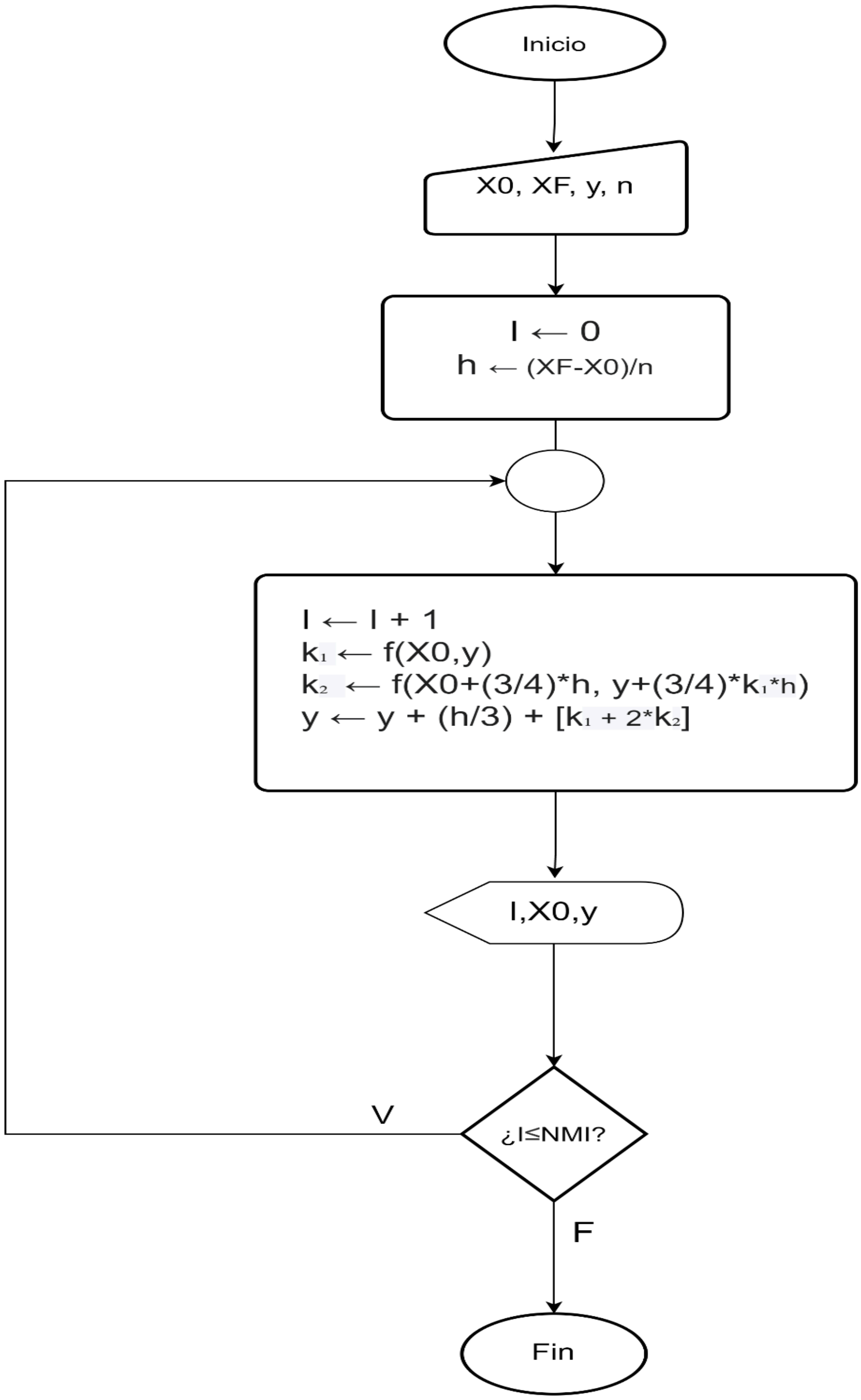
Algoritmo de Ralston

Maestro: Dr. Cecil Vidriales

Algoritmo de Ralston

Descripción: El Método de Ralston es una mejora del Método de Euler, optimizando el peso de la pendiente dentro del intervalo de integración para reducir el error.

Utilidad: Ofrece una mayor precisión sin incrementar significativamente la complejidad computacional.



Pseudocódigo del Algoritmo de Ralston

1. Inicio
2. Inicializar Variables:
   - x = x0 (valor inicial de la variable independiente)
   - y (valor inicial de la solución, se asume que ya está dado)
   - xf (valor final de la variable independiente)
   - n (número de pasos)
   - h = (xf - x0) / n (tamaño del paso)
3. Bucle For (i = 0; i < n; i++):
   - Calcular la derivada en el punto actual: k1 = f(x, y)
   - Calcular la predicción del valor de x y y en el punto ponderado (2/3 del paso):
     - x_ponderado = x + (3/4) * h
     - y_ponderado = y + (3/4) * h * k1
   - Calcular la derivada en el punto ponderado: k2 = f(x_ponderado, y_ponderado)
   - Actualizar el valor de y usando una combinación ponderada de las derivadas: y = y + h * ((1/3) * k1 + (2/3) * k2)
   - Actualizar el valor de x: x = x + h
   - Imprimir (opcional): Imprimir x, y
4. Fin del Bucle For
5. Fin

Código:

```
double x0, xf, y, h, n;
    printf("Ingresa el valor de X: ");
    scanf("%lf", &x0);

    printf("Ingresa el valor de XF: ");
    scanf("%lf", &xf);

    printf("Ingresa el valor de Y: ");
    scanf("%lf", &y);

    printf("Ingresa el numero de iteraciones: ");
    scanf("%lf", &n);

    h = (xf - x0) / n;

    for (int i = 0; i < n; ++i) {
        double k1 = f(x0, y);
        double k2 = f(x0 + (3.0 / 4.0) * h, y + (3.0 / 4.0) * k1 * h);
        y = y + (h / 4.0) * (k1 + 2 * k2);
        x0 = x0 + h;
    }

    printf("El valor de Y = %lf", y);
```

Prueba de escritorio
f(x,y)=sin(x) + y;
Ingresa el valor de X: 0
Ingresa el valor de XF: 3
Ingresa el valor de Y: 2
Ingresa el numero de iteraciones: 20
Iteracion 1:
  k1 = f(x0, y0) = f(0.150000, 2.250295) = 2.000000
  k2 = f(x0 + (3/4) * h, y0 + (3/4) * k1 * h) = f(0.150000, 2.250295) = 2.337263
Iteracion 2:
  k1 = f(x1, y1) = f(0.300000, 2.548767) = 2.399733
  k2 = f(x1 + (3/4) * h, y1 + (3/4) * k1 * h) = f(0.300000, 2.548767) = 2.779760
Iteracion 3:
  k1 = f(x2, y2) = f(0.450000, 2.900651) = 2.844287
  k2 = f(x2 + (3/4) * h, y2 + (3/4) * k1 * h) = f(0.450000, 2.900651) = 3.269650
Iteracion 4:
  k1 = f(x3, y3) = f(0.600000, 3.311428) = 3.335617
  k2 = f(x3 + (3/4) * h, y3 + (3/4) * k1 * h) = f(0.600000, 3.311428) = 3.809211
Iteracion 5:
  k1 = f(x4, y4) = f(0.750000, 3.786871) = 3.876070
  k2 = f(x4 + (3/4) * h, y4 + (3/4) * k1 * h) = f(0.750000, 3.786871) = 4.401213
Iteracion 6:
  k1 = f(x5, y5) = f(0.900000, 4.333119) = 4.468510
  k2 = f(x5 + (3/4) * h, y5 + (3/4) * k1 * h) = f(0.900000, 4.333119) = 5.049050
Iteracion 7:
  k1 = f(x6, y6) = f(1.050000, 4.956752) = 5.116446
  k2 = f(x6 + (3/4) * h, y6 + (3/4) * k1 * h) = f(1.050000, 4.956752) = 5.756878
Iteracion 8:
  k1 = f(x7, y7) = f(1.200000, 5.664891) = 5.824175
  k2 = f(x7 + (3/4) * h, y7 + (3/4) * k1 * h) = f(1.200000, 5.664891) = 6.529770
Iteracion 9:
  k1 = f(x8, y8) = f(1.350000, 6.465316) = 6.596930
  k2 = f(x8 + (3/4) * h, y8 + (3/4) * k1 * h) = f(1.350000, 6.465316) = 7.373872
Iteracion 10:
  k1 = f(x9, y9) = f(1.500000, 7.366598) = 7.441040
  k2 = f(x9 + (3/4) * h, y9 + (3/4) * k1 * h) = f(1.500000, 7.366598) = 8.296575
Iteracion 11:
  k1 = f(x10, y10) = f(1.650000, 8.378254) = 8.364093
  k2 = f(x10 + (3/4) * h, y10 + (3/4) * k1 * h) = f(1.650000, 8.378254) = 9.306690
Iteracion 12:
  k1 = f(x11, y11) = f(1.800000, 9.510918) = 9.375119
  k2 = f(x11 + (3/4) * h, y11 + (3/4) * k1 * h) = f(1.800000, 9.510918) = 10.414636
Iteracion 13:
  k1 = f(x12, y12) = f(1.950000, 10.776545) = 10.484766
  k2 = f(x12 + (3/4) * h, y12 + (3/4) * k1 * h) = f(1.950000, 10.776545) = 11.632640
Iteracion 14:
  k1 = f(x13, y13) = f(2.100000, 12.188622) = 11.705505
  k2 = f(x13 + (3/4) * h, y13 + (3/4) * k1 * h) = f(2.100000, 12.188622) = 12.974944

Iteracion 15:
  k1 = f(x14, y14) = f(2.250000, 13.762418) = 13.051832
  k2 = f(x14 + (3/4) * h, y14 + (3/4) * k1 * h) = f(2.250000, 13.762418) = 14.458030
Iteracion 16:
  k1 = f(x15, y15) = f(2.400000, 15.515251) = 14.540491
  k2 = f(x15 + (3/4) * h, y15 + (3/4) * k1 * h) = f(2.400000, 15.515251) = 16.100858
Iteracion 17:
  k1 = f(x16, y16) = f(2.550000, 17.466787) = 16.190714
  k2 = f(x16 + (3/4) * h, y16 + (3/4) * k1 * h) = f(2.550000, 17.466787) = 17.925118
Iteracion 18:
  k1 = f(x17, y17) = f(2.700000, 19.639368) = 18.024470
  k2 = f(x17 + (3/4) * h, y17 + (3/4) * k1 * h) = f(2.700000, 19.639368) = 19.955514
Iteracion 19:
  k1 = f(x18, y18) = f(2.850000, 22.058375) = 20.066748
  k2 = f(x18 + (3/4) * h, y18 + (3/4) * k1 * h) = f(2.850000, 22.058375) = 22.220061
Iteracion 20:
  k1 = f(x19, y19) = f(3.000000, 24.752626) = 22.345853
  k2 = f(x19 + (3/4) * h, y19 + (3/4) * k1 * h) = f(3.000000, 24.752626) = 24.750421
El valor de Y = 24.752626

Ejecución:

```
Algoritmo de Ralston
Ingresa el valor de X: 0
Ingresa el valor de XF: 3
Ingresa el valor de Y: 2
Ingresa el numero de iteraciones: 20
El valor de Y = 24.752626
Tiempo de ejecucion: 0.0000
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

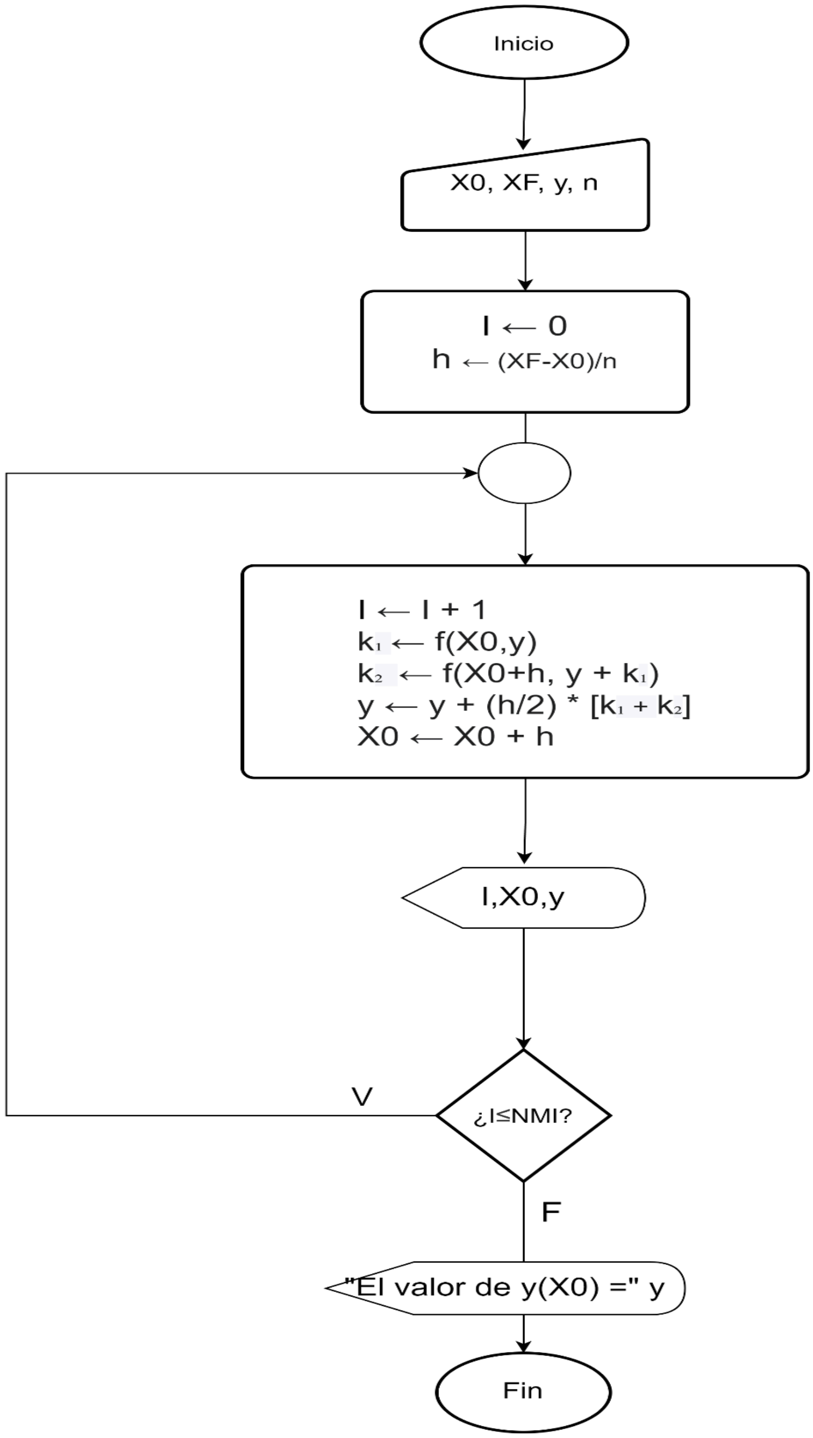
Algoritmo de Runge-Kutta(segundo orden)

Maestro: Dr. Cecil Vidriales

Algoritmo de Runge-Kutta de Segundo Orden

Descripción: Este método es una extensión del Método de Euler, que utiliza un promedio ponderado de varias pendientes evaluadas dentro del intervalo.

Utilidad: Se usa para obtener soluciones más precisas de EDOs de segundo orden, equilibrando precisión y eficiencia.



Pseudocódigo del Algoritmo de Runge-Kutta de Segundo Orden

1. Inicio
2. Inicializar Variables:
    - x = x0 (valor inicial de la variable independiente)
    - y (valor inicial de la solución, se asume que ya está dado)
    - xf (valor final de la variable independiente)
    - n (número de pasos)
    - h = (xf - x0) / n (tamaño del paso)
3. Bucle For (i = 0; i < n; i++):
    - Calcular la derivada en el punto actual: k1 = f(x, y)
    - Calcular la predicción del valor de x y y en el siguiente punto:
        - x_next = x + h
        - y_predict = y + h * k1
    - Calcular la derivada en el punto predicho: k2 = f(x_next, y_predict)
    - Actualizar el valor de y usando el promedio de las derivadas: y = y + (h / 2) * (k1 + k2)
    - Actualizar el valor de x: x = x + h
    - Imprimir (opcional): Imprimir x, y
4. Fin del Bucle For
5. Fin

Código:

```
double x0, xf, y, h, n;

   printf("Ingresa el valor de X: ");

   scanf("%lf", &x0);

   printf("Ingresa el valor de XF: ");

   scanf("%lf", &xf);

   printf("Ingresa el valor de Y: ");

   scanf("%lf", &y);

   printf("Ingresa el numero de iteraciones: ");

   scanf("%lf", &n)

   h = (xf - x0) / n;

   for (int i = 0; i < n; ++i) {

      double k1 = f(x0, y);
```

```
        double k2 = f(x0 + h, y + k1 * h); // Corrección aquí

        y = y + (h / 2) * (k1 + k2);

        x0 = x0 + h;

    }

    printf("El valor de Y = %lf", y);
```

Prueba de escritorio

f(x,y)=sin(x) + y;
Ingresa el valor de X: 0
Ingresa el valor de XF: 2
Ingresa el valor de Y: 1
Ingresa el numero de iteraciones: 20
Iteracion 1:
k1 = f(x0, y0) = f(0.000000, 1.000000) = 1.000000
k2 = f(x0 + h, y0 + k1 * h) = f(0.000000 + 0.100000, 1.000000 + 1.000000 * 0.100000) = 1.199833
y1 = y0 + (h / 2) * (k1 + k2) = 1.000000 + (0.100000 / 2) * (1.000000 + 1.199833) = 1.109992
Iteracion 2:
k1 = f(x1, y1) = f(0.100000, 1.109992) = 1.209825
k2 = f(x1 + h, y1 + k1 * h) = f(0.100000 + 0.100000, 1.109992 + 1.209825 * 0.100000) = 1.429644
y2 = y1 + (h / 2) * (k1 + k2) = 1.109992 + (0.100000 / 2) * (1.209825 + 1.429644) = 1.241965
Iteracion 3:
k1 = f(x2, y2) = f(0.200000, 1.241965) = 1.440634
k2 = f(x2 + h, y2 + k1 * h) = f(0.200000 + 0.100000, 1.241965 + 1.440634 * 0.100000) = 1.681549
y3 = y2 + (h / 2) * (k1 + k2) = 1.241965 + (0.100000 / 2) * (1.440634 + 1.681549) = 1.398074
Iteracion 4:
k1 = f(x3, y3) = f(0.300000, 1.398074) = 1.693594
k2 = f(x3 + h, y3 + k1 * h) = f(0.300000 + 0.100000, 1.398074 + 1.693594 * 0.100000) = 1.956852
y4 = y3 + (h / 2) * (k1 + k2) = 1.398074 + (0.100000 / 2) * (1.693594 + 1.956852) = 1.580597
Iteracion 5:
k1 = f(x4, y4) = f(0.400000, 1.580597) = 1.970015
k2 = f(x4 + h, y4 + k1 * h) = f(0.400000 + 0.100000, 1.580597 + 1.970015 * 0.100000) = 2.257024
y5 = y4 + (h / 2) * (k1 + k2) = 1.580597 + (0.100000 / 2) * (1.970015 + 2.257024) = 1.791949
Iteracion 6:
k1 = f(x5, y5) = f(0.500000, 1.791949) = 2.271374
k2 = f(x5 + h, y5 + k1 * h) = f(0.500000 + 0.100000, 1.791949 + 2.271374 * 0.100000) = 2.583728
y6 = y5 + (h / 2) * (k1 + k2) = 1.791949 + (0.100000 / 2) * (2.271374 + 2.583728) = 2.034704
Iteracion 7:

k1 = f(x6, y6) = f(0.600000, 2.034704) = 2.599346
k2 = f(x6 + h, y6 + k1 * h) = f(0.600000 + 0.100000, 2.034704 + 2.599346 * 0.100000) = 2.938856
y7 = y6 + (h / 2) * (k1 + k2) = 2.034704 + (0.100000 / 2) * (2.599346 + 2.938856) = 2.311614
Iteracion 8:
k1 = f(x7, y7) = f(0.700000, 2.311614) = 2.955831
k2 = f(x7 + h, y7 + k1 * h) = f(0.700000 + 0.100000, 2.311614 + 2.955831 * 0.100000) = 3.324553
y8 = y7 + (h / 2) * (k1 + k2) = 2.311614 + (0.100000 / 2) * (2.955831 + 3.324553) = 2.625633
Iteracion 9:
k1 = f(x8, y8) = f(0.800000, 2.625633) = 3.342989
k2 = f(x8 + h, y8 + k1 * h) = f(0.800000 + 0.100000, 2.625633 + 3.342989 * 0.100000) = 3.743259
y9 = y8 + (h / 2) * (k1 + k2) = 2.625633 + (0.100000 / 2) * (3.342989 + 3.743259) = 2.979945
Iteracion 10:
k1 = f(x9, y9) = f(0.900000, 2.979945) = 3.763272
k2 = f(x9 + h, y9 + k1 * h) = f(0.900000 + 0.100000, 2.979945 + 3.763272 * 0.100000) = 4.197744
y10 = y9 + (h / 2) * (k1 + k2) = 2.979945 + (0.100000 / 2) * (3.763272 + 4.197744) = 3.377996
Iteracion 11:
k1 = f(x10, y10) = f(1.000000, 3.377996) = 4.219467
k2 = f(x10 + h, y10 + k1 * h) = f(1.000000 + 0.100000, 3.377996 + 4.219467 * 0.100000) = 4.691150
y11 = y10 + (h / 2) * (k1 + k2) = 3.377996 + (0.100000 / 2) * (4.219467 + 4.691150) = 3.823527
Iteracion 12:
k1 = f(x11, y11) = f(1.100000, 3.823527) = 4.714734
k2 = f(x11 + h, y11 + k1 * h) = f(1.100000 + 0.100000, 3.823527 + 4.714734 * 0.100000) = 5.227040
y12 = y11 + (h / 2) * (k1 + k2) = 3.823527 + (0.100000 / 2) * (4.714734 + 5.227040) = 4.320616
Iteracion 13:
k1 = f(x12, y12) = f(1.200000, 4.320616) = 5.252655
k2 = f(x12 + h, y12 + k1 * h) = f(1.200000 + 0.100000, 4.320616 + 5.252655 * 0.100000) = 5.809439
y13 = y12 + (h / 2) * (k1 + k2) = 4.320616 + (0.100000 / 2) * (5.252655 + 5.809439) = 4.873720
Iteracion 14:
k1 = f(x13, y13) = f(1.300000, 4.873720) = 5.837279
k2 = f(x13 + h, y13 + k1 * h) = f(1.300000 + 0.100000, 4.873720 + 5.837279 * 0.100000) = 6.442898
y14 = y13 + (h / 2) * (k1 + k2) = 4.873720 + (0.100000 / 2) * (5.837279 + 6.442898) = 5.487729
Iteracion 15:
k1 = f(x14, y14) = f(1.400000, 5.487729) = 6.473179
k2 = f(x14 + h, y14 + k1 * h) = f(1.400000 + 0.100000, 5.487729 + 6.473179 * 0.100000) = 7.132542

y15 = y14 + (h / 2) * (k1 + k2) = 5.487729 + (0.100000 / 2) * (6.473179 + 7.132542) = 6.168015
Iteracion 16:
k1 = f(x15, y15) = f(1.500000, 6.168015) = 7.165510
k2 = f(x15 + h, y15 + k1 * h) = f(1.500000 + 0.100000, 6.168015 + 7.165510 * 0.100000) = 7.884140
y16 = y15 + (h / 2) * (k1 + k2) = 6.168015 + (0.100000 / 2) * (7.165510 + 7.884140) = 6.920498
Iteracion 17:
k1 = f(x16, y16) = f(1.600000, 6.920498) = 7.920071
k2 = f(x16 + h, y16 + k1 * h) = f(1.600000 + 0.100000, 6.920498 + 7.920071 * 0.100000) = 8.704170
y17 = y16 + (h / 2) * (k1 + k2) = 6.920498 + (0.100000 / 2) * (7.920071 + 8.704170) = 7.751710
Iteracion 18:
k1 = f(x17, y17) = f(1.700000, 7.751710) = 8.743375
k2 = f(x17 + h, y17 + k1 * h) = f(1.700000 + 0.100000, 7.751710 + 8.743375 * 0.100000) = 9.599895
y18 = y17 + (h / 2) * (k1 + k2) = 7.751710 + (0.100000 / 2) * (8.743375 + 9.599895) = 8.668873
Iteracion 19:
k1 = f(x18, y18) = f(1.800000, 8.668873) = 9.642721
k2 = f(x18 + h, y18 + k1 * h) = f(1.800000 + 0.100000, 8.668873 + 9.642721 * 0.100000) = 10.579446
y19 = y18 + (h / 2) * (k1 + k2) = 8.668873 + (0.100000 / 2) * (9.642721 + 10.579446) = 9.679982
Iteracion 20:
k1 = f(x19, y19) = f(1.900000, 9.679982) = 10.626282
k2 = f(x19 + h, y19 + k1 * h) = f(1.900000 + 0.100000, 9.679982 + 10.626282 * 0.100000) = 11.651907
y20 = y19 + (h / 2) * (k1 + k2) = 9.679982 + (0.100000 / 2) * (10.626282 + 11.651907) = 10.793891
El valor de Y = 10.793891

Ejecución:

```
Algoritmo de Runge-Kutta(2do orden
)Ingresa el valor de X: 0
Ingresa el valor de XF: 2
Ingresa el valor de Y: 1
Ingresa el numero de iteraciones: 20
El valor de Y = 10.793891
Tiempo de ejecucion: 0.000000 segundos
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

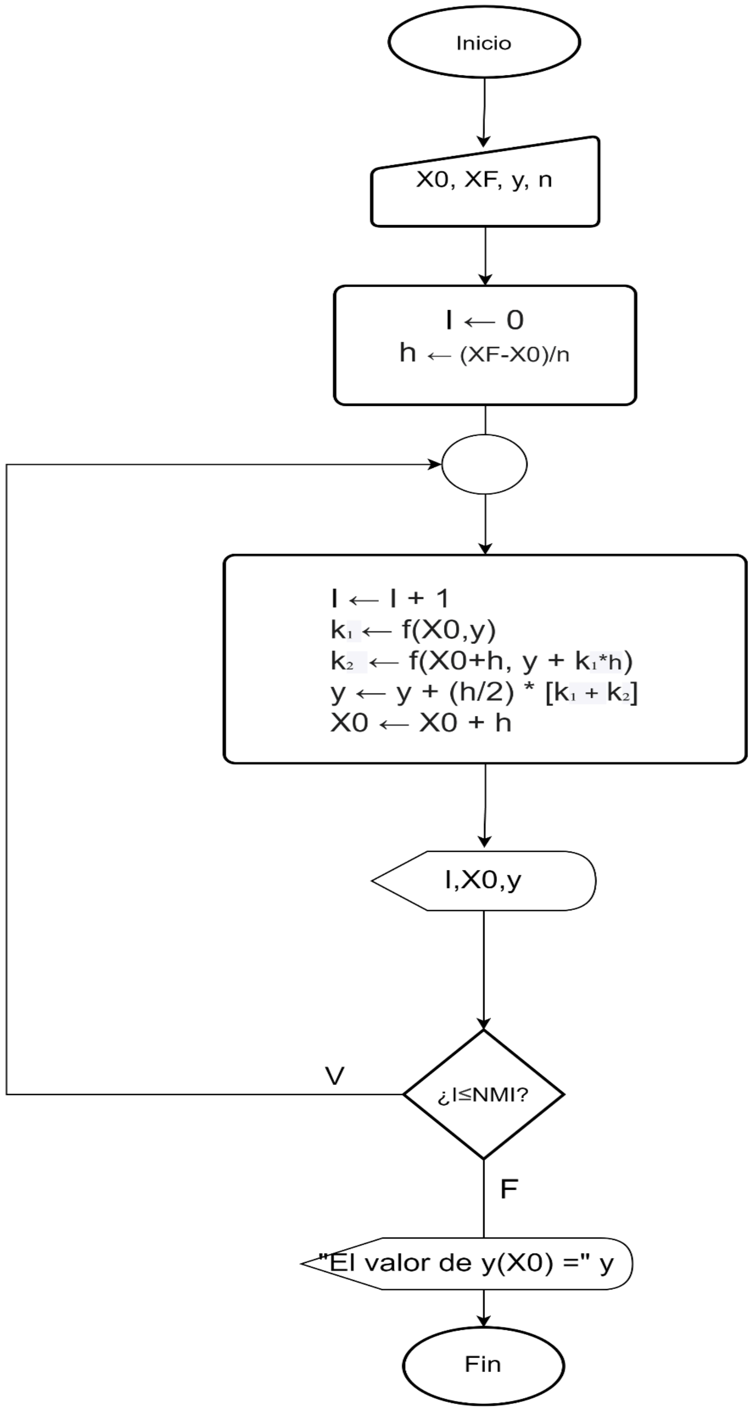
Algoritmo de Heun

Maestro: Dr. Cecil Vidriales

Algoritmo de Heun

Descripción: El Método de Heun, también conocido como el método de Euler modificado, promedia la pendiente al inicio y al final del intervalo.

Utilidad: Ofrece mejor precisión en la solución de EDOs, adecuado para problemas que requieren un balance entre precisión y complejidad.



Pseudocódigo del Algoritmo de Heun

1. Inicio
2. Inicializar Variables:
    - x = x0
    - y
    - xf
    - n
    - NMI
    - h = (xf - x0) / n
3. Bucle For (i = 0; i < n; i++):
    - k1 = f(x, y)
    - k2 = f(x + h, y + k1 * h)
    - y = y + (h / 2) * (k1 + k2)
    - x = x + h
4. Fin del Bucle For
5. Imprimir: i, x, y
6. Condición: Si i <= NMI entonces
    - Imprimir: i, x, y
7. Fin

Código:

```
double x0, xf, y, h, n;
    printf("Ingresa el valor de X: ");
    scanf("%lf", &x0);

    printf("Ingresa el valor de XF: ");
    scanf("%lf", &xf);

    printf("Ingresa el valor de Y: ");
    scanf("%lf", &y);

    printf("Ingresa el numero de iteraciones: ");
    scanf("%lf", &n);

    h = (xf - x0) / n;

    for (int i = 0; i < n; ++i) {
        double k1 = f(x0, y);
        double k2 = f(x0 + h, y + k1 * h); // Corrección aquí
        y = y + (h / 2) * (k1 + k2);
        x0 = x0 + h;
    }

    printf("El valor de Y = %lf", y);
```

Prueba de escritorio
f(x,y)=sin(x) + y;
Ingresa el valor de X: 0
Ingresa el valor de XF: 2
Ingresa el valor de Y: 1
Ingresa el numero de iteraciones: 20
Iteracion 1:
k1 = f(x0, y0) = f(0.000000, 1.000000) = 1.000000
k2 = f(x0 + h, y0 + k1 * h) = f(0.000000 + 0.100000, 1.000000 + 1.000000 * 0.100000) = 1.199833
y1 = y0 + (h / 2) * (k1 + k2) = 1.000000 + (0.100000 / 2) * (1.000000 + 1.199833) = 1.109992
x1 = x0 + h = 0.000000 + 0.100000 = 0.100000
Iteracion 2:
k1 = f(x1, y1) = f(0.100000, 1.109992) = 1.209825
k2 = f(x1 + h, y1 + k1 * h) = f(0.100000 + 0.100000, 1.109992 + 1.209825 * 0.100000) = 1.429644
y2 = y1 + (h / 2) * (k1 + k2) = 1.109992 + (0.100000 / 2) * (1.209825 + 1.429644) = 1.241965
x2 = x1 + h = 0.100000 + 0.100000 = 0.200000
Iteracion 3:
k1 = f(x2, y2) = f(0.200000, 1.241965) = 1.440634
k2 = f(x2 + h, y2 + k1 * h) = f(0.200000 + 0.100000, 1.241965 + 1.440634 * 0.100000) = 1.681549
y3 = y2 + (h / 2) * (k1 + k2) = 1.241965 + (0.100000 / 2) * (1.440634 + 1.681549) = 1.398074
x3 = x2 + h = 0.200000 + 0.100000 = 0.300000
Iteracion 4:
k1 = f(x3, y3) = f(0.300000, 1.398074) = 1.693594
k2 = f(x3 + h, y3 + k1 * h) = f(0.300000 + 0.100000, 1.398074 + 1.693594 * 0.100000) = 1.956852
y4 = y3 + (h / 2) * (k1 + k2) = 1.398074 + (0.100000 / 2) * (1.693594 + 1.956852) = 1.580597
x4 = x3 + h = 0.300000 + 0.100000 = 0.400000
Iteracion 5:
k1 = f(x4, y4) = f(0.400000, 1.580597) = 1.970015
k2 = f(x4 + h, y4 + k1 * h) = f(0.400000 + 0.100000, 1.580597 + 1.970015 * 0.100000) = 2.257024
y5 = y4 + (h / 2) * (k1 + k2) = 1.580597 + (0.100000 / 2) * (1.970015 + 2.257024) = 1.791949
x5 = x4 + h = 0.400000 + 0.100000 = 0.500000
Iteracion 6:
k1 = f(x5, y5) = f(0.500000, 1.791949) = 2.271374
k2 = f(x5 + h, y5 + k1 * h) = f(0.500000 + 0.100000, 1.791949 + 2.271374 * 0.100000) = 2.583728
y6 = y5 + (h / 2) * (k1 + k2) = 1.791949 + (0.100000 / 2) * (2.271374 + 2.583728) = 2.034704
x6 = x5 + h = 0.500000 + 0.100000 = 0.600000
Iteracion 7:
k1 = f(x6, y6) = f(0.600000, 2.034704) = 2.599346
k2 = f(x6 + h, y6 + k1 * h) = f(0.600000 + 0.100000, 2.034704 + 2.599346 * 0.100000) = 2.938856
y7 = y6 + (h / 2) * (k1 + k2) = 2.034704 + (0.100000 / 2) * (2.599346 + 2.938856) = 2.311614
x7 = x6 + h = 0.600000 + 0.100000 = 0.700000

Iteracion 8:
k1 = f(x7, y7) = f(0.700000, 2.311614) = 2.955831
k2 = f(x7 + h, y7 + k1 * h) = f(0.700000 + 0.100000, 2.311614 + 2.955831 * 0.100000) =
3.324553
y8 = y7 + (h / 2) * (k1 + k2) = 2.311614 + (0.100000 / 2) * (2.955831 + 3.324553) = 2.625633
x8 = x7 + h = 0.700000 + 0.100000 = 0.800000
Iteracion 9:
k1 = f(x8, y8) = f(0.800000, 2.625633) = 3.342989
k2 = f(x8 + h, y8 + k1 * h) = f(0.800000 + 0.100000, 2.625633 + 3.342989 * 0.100000) =
3.743259
y9 = y8 + (h / 2) * (k1 + k2) = 2.625633 + (0.100000 / 2) * (3.342989 + 3.743259) = 2.979945
x9 = x8 + h = 0.800000 + 0.100000 = 0.900000
Iteracion 10:
k1 = f(x9, y9) = f(0.900000, 2.979945) = 3.763272
k2 = f(x9 + h, y9 + k1 * h) = f(0.900000 + 0.100000, 2.979945 + 3.763272 * 0.100000) =
4.197744
y10 = y9 + (h / 2) * (k1 + k2) = 2.979945 + (0.100000 / 2) * (3.763272 + 4.197744) =
3.377996
x10 = x9 + h = 0.900000 + 0.100000 = 1.000000
Iteracion 11:
k1 = f(x10, y10) = f(1.000000, 3.377996) = 4.219467
k2 = f(x10 + h, y10 + k1 * h) = f(1.000000 + 0.100000, 3.377996 + 4.219467 * 0.100000) =
4.691150
y11 = y10 + (h / 2) * (k1 + k2) = 3.377996 + (0.100000 / 2) * (4.219467 + 4.691150) =
3.823527
x11 = x10 + h = 1.000000 + 0.100000 = 1.100000
Iteracion 12:
k1 = f(x11, y11) = f(1.100000, 3.823527) = 4.714734
k2 = f(x11 + h, y11 + k1 * h) = f(1.100000 + 0.100000, 3.823527 + 4.714734 * 0.100000) =
5.227040
y12 = y11 + (h / 2) * (k1 + k2) = 3.823527 + (0.100000 / 2) * (4.714734 + 5.227040) =
4.320616
x12 = x11 + h = 1.100000 + 0.100000 = 1.200000
Iteracion 13:
k1 = f(x12, y12) = f(1.200000, 4.320616) = 5.252655
k2 = f(x12 + h, y12 + k1 * h) = f(1.200000 + 0.100000, 4.320616 + 5.252655 * 0.100000) =
5.809439
y13 = y12 + (h / 2) * (k1 + k2) = 4.320616 + (0.100000 / 2) * (5.252655 + 5.809439) =
4.873720
x13 = x12 + h = 1.200000 + 0.100000 = 1.300000
Iteracion 14:
k1 = f(x13, y13) = f(1.300000, 4.873720) = 5.837279
k2 = f(x13 + h, y13 + k1 * h) = f(1.300000 + 0.100000, 4.873720 + 5.837279 * 0.100000) =
6.442898
y14 = y13 + (h / 2) * (k1 + k2) = 4.873720 + (0.100000 / 2) * (5.837279 + 6.442898) =
5.487729
x14 = x13 + h = 1.300000 + 0.100000 = 1.400000
Iteracion 15:

k1 = f(x14, y14) = f(1.400000, 5.487729) = 6.473179
k2 = f(x14 + h, y14 + k1 * h) = f(1.400000 + 0.100000, 5.487729 + 6.473179 * 0.100000) = 7.132542
y15 = y14 + (h / 2) * (k1 + k2) = 5.487729 + (0.100000 / 2) * (6.473179 + 7.132542) = 6.168015
x15 = x14 + h = 1.400000 + 0.100000 = 1.500000
Iteracion 16:
k1 = f(x15, y15) = f(1.500000, 6.168015) = 7.165510
k2 = f(x15 + h, y15 + k1 * h) = f(1.500000 + 0.100000, 6.168015 + 7.165510 * 0.100000) = 7.884140
y16 = y15 + (h / 2) * (k1 + k2) = 6.168015 + (0.100000 / 2) * (7.165510 + 7.884140) = 6.920498
x16 = x15 + h = 1.500000 + 0.100000 = 1.600000
Iteracion 17:
k1 = f(x16, y16) = f(1.600000, 6.920498) = 7.920071
k2 = f(x16 + h, y16 + k1 * h) = f(1.600000 + 0.100000, 6.920498 + 7.920071 * 0.100000) = 8.704170
y17 = y16 + (h / 2) * (k1 + k2) = 6.920498 + (0.100000 / 2) * (7.920071 + 8.704170) = 7.751710
x17 = x16 + h = 1.600000 + 0.100000 = 1.700000
Iteracion 18:
k1 = f(x17, y17) = f(1.700000, 7.751710) = 8.743375
k2 = f(x17 + h, y17 + k1 * h) = f(1.700000 + 0.100000, 7.751710 + 8.743375 * 0.100000) = 9.599895
y18 = y17 + (h / 2) * (k1 + k2) = 7.751710 + (0.100000 / 2) * (8.743375 + 9.599895) = 8.668873
x18 = x17 + h = 1.700000 + 0.100000 = 1.800000
Iteracion 19:
k1 = f(x18, y18) = f(1.800000, 8.668873) = 9.642721
k2 = f(x18 + h, y18 + k1 * h) = f(1.800000 + 0.100000, 8.668873 + 9.642721 * 0.100000) = 10.579446
y19 = y18 + (h / 2) * (k1 + k2) = 8.668873 + (0.100000 / 2) * (9.642721 + 10.579446) = 9.679982
x19 = x18 + h = 1.800000 + 0.100000 = 1.900000
Iteracion 20:
k1 = f(x19, y19) = f(1.900000, 9.679982) = 10.626282
k2 = f(x19 + h, y19 + k1 * h) = f(1.900000 + 0.100000, 9.679982 + 10.626282 * 0.100000) = 11.651907
y20 = y19 + (h / 2) * (k1 + k2) = 9.679982 + (0.100000 / 2) * (10.626282 + 11.651907) = 10.793891
x20 = x19 + h = 1.900000 + 0.100000 = 2.000000
El valor de Y = 10.793891

Ejecución:

```
Algoritmo de Heun
Ingresa el valor de X: 0
Ingresa el valor de XF: 2
Ingresa el valor de Y: 1
Ingresa el numero de iteraciones: 20
El valor de Y = 10.793891
Tiempo de ejecucion: 0.000000 segundos
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

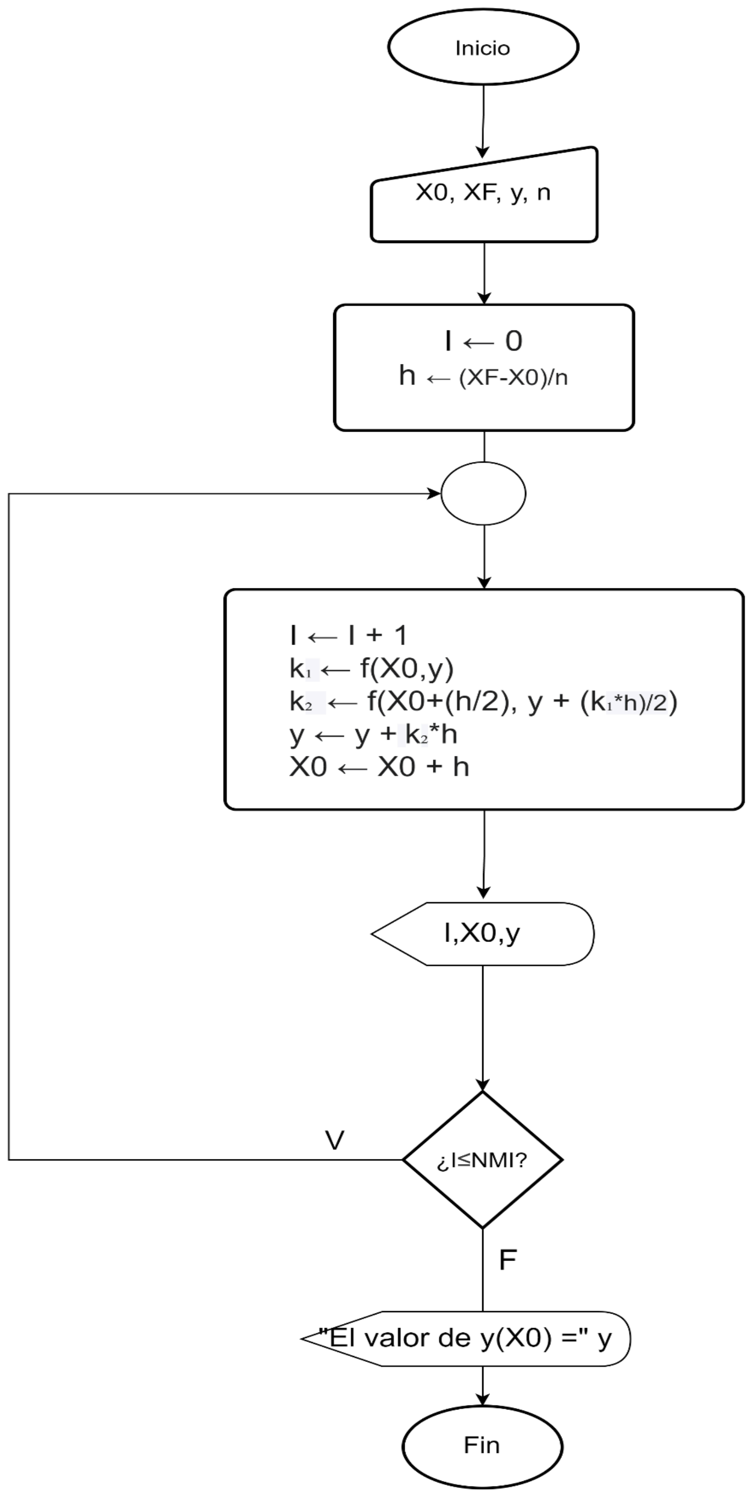
Algoritmo de Punto Medio

Maestro: Dr. Cecil Vidriales

Algoritmo de Punto Medio

Descripción: El Método de Punto Medio es un método de segundo orden que utiliza la pendiente en el punto medio del intervalo para mejorar la precisión.

Utilidad: Se emplea para mejorar la precisión en la solución de EDOs sin aumentar significativamente la carga computacional.



Pseudocódigo del Algoritmo del Punto Medio

1. Inicio
2. Inicializar Variables:
   - x = x0
   - y
   - xf
   - n
   - NMI
   - h = (xf - x0) / n
3. Bucle For (i = 0; i < n; i++):
   - k1 = f(x, y)
   - xm = x + h / 2
   - ym = y + h / 2 * k1
   - k2 = f(xm, ym)
   - y = y + h * k2
   - x = x + h
4. Fin del Bucle For
5. Imprimir: i, x, y
6. Condición: Si i <= NMI entonces
   - Imprimir: i, x, y
7. Fin

Código:

```
double x0, xf, y, h, n;

printf("Ingresa el valor de X: ");

scanf("%lf", &x0);



printf("Ingresa el valor de XF: ");

scanf("%lf", &xf);



printf("Ingresa el valor de Y: ");

scanf("%lf", &y);



printf("Ingresa el numero de iteraciones: ");

scanf("%lf", &n);
```

```
    h = (xf - x0) / n;


    for (int i = 0; i < n; ++i) {

        double k1 = f(x0, y);

        double k2 = f(x0 + h/2, y + (k1 * h)/2); // Corrección aquí

        y = y + (k2 * h);

        x0 = x0 + h;

    }


    printf("El valor de Y = %lf", y);
```

Prueba de escritorio
f(x,y)=sin(x) + y;
Ingresa el valor de X: 0
Ingresa el valor de XF: 2
Ingresa el valor de Y: 1
Ingresa el numero de iteraciones: 20
Iteracion 1:
k1 = f(x0, y0) = f(0.000000, 1.000000) = 1.000000
k2 = f(x0 + h/2, y0 + (k1 * h)/2) = f(0.000000 + 0.100000/2, 1.000000 + (1.000000 * 0.100000)/2) = 1.099979
y1 = y0 + (k2 * h) = 1.000000 + (1.099979 * 0.100000) = 1.109998
x1 = x0 + h = 0.000000 + 0.100000 = 0.100000
Iteracion 2:
k1 = f(x1, y1) = f(0.100000, 1.109998) = 1.209831
k2 = f(x1 + h/2, y1 + (k1 * h)/2) = f(0.100000 + 0.100000/2, 1.109998 + (1.209831 * 0.100000)/2) = 1.319928
y2 = y1 + (k2 * h) = 1.109998 + (1.319928 * 0.100000) = 1.241991
x2 = x1 + h = 0.100000 + 0.100000 = 0.200000
Iteracion 3:
k1 = f(x2, y2) = f(0.200000, 1.241991) = 1.440660
k2 = f(x2 + h/2, y2 + (k1 * h)/2) = f(0.200000 + 0.100000/2, 1.241991 + (1.440660 * 0.100000)/2) = 1.561428
y3 = y2 + (k2 * h) = 1.241991 + (1.561428 * 0.100000) = 1.398133
x3 = x2 + h = 0.200000 + 0.100000 = 0.300000
Iteracion 4:
k1 = f(x3, y3) = f(0.300000, 1.398133) = 1.693654
k2 = f(x3 + h/2, y3 + (k1 * h)/2) = f(0.300000 + 0.100000/2, 1.398133 + (1.693654 * 0.100000)/2) = 1.825714

y4 = y3 + (k2 * h) = 1.398133 + (1.825714 * 0.100000) = 1.580705
x4 = x3 + h = 0.300000 + 0.100000 = 0.400000
Iteracion 5:
k1 = f(x4, y4) = f(0.400000, 1.580705) = 1.970123
k2 = f(x4 + h/2, y4 + (k1 * h)/2) = f(0.400000 + 0.100000/2, 1.580705 + (1.970123 * 0.100000)/2) = 2.114177
y5 = y4 + (k2 * h) = 1.580705 + (2.114177 * 0.100000) = 1.792122
x5 = x4 + h = 0.400000 + 0.100000 = 0.500000
Iteracion 6:
k1 = f(x5, y5) = f(0.500000, 1.792122) = 2.271548
k2 = f(x5 + h/2, y5 + (k1 * h)/2) = f(0.500000 + 0.100000/2, 1.792122 + (2.271548 * 0.100000)/2) = 2.428387
y6 = y5 + (k2 * h) = 1.792122 + (2.428387 * 0.100000) = 2.034961
x6 = x5 + h = 0.500000 + 0.100000 = 0.600000
Iteracion 7:
k1 = f(x6, y6) = f(0.600000, 2.034961) = 2.599604
k2 = f(x6 + h/2, y6 + (k1 * h)/2) = f(0.600000 + 0.100000/2, 2.034961 + (2.599604 * 0.100000)/2) = 2.770128
y7 = y6 + (k2 * h) = 2.034961 + (2.770128 * 0.100000) = 2.311974
x7 = x6 + h = 0.600000 + 0.100000 = 0.700000
Iteracion 8:
k1 = f(x7, y7) = f(0.700000, 2.311974) = 2.956192
k2 = f(x7 + h/2, y7 + (k1 * h)/2) = f(0.700000 + 0.100000/2, 2.311974 + (2.956192 * 0.100000)/2) = 3.141422
y8 = y7 + (k2 * h) = 2.311974 + (3.141422 * 0.100000) = 2.626116
x8 = x7 + h = 0.700000 + 0.100000 = 0.800000
Iteracion 9:
k1 = f(x8, y8) = f(0.800000, 2.626116) = 3.343472
k2 = f(x8 + h/2, y8 + (k1 * h)/2) = f(0.800000 + 0.100000/2, 2.626116 + (3.343472 * 0.100000)/2) = 3.544570
y9 = y8 + (k2 * h) = 2.626116 + (3.544570 * 0.100000) = 2.980573
x9 = x8 + h = 0.800000 + 0.100000 = 0.900000
Iteracion 10:
k1 = f(x9, y9) = f(0.900000, 2.980573) = 3.763900
k2 = f(x9 + h/2, y9 + (k1 * h)/2) = f(0.900000 + 0.100000/2, 2.980573 + (3.763900 * 0.100000)/2) = 3.982184
y10 = y9 + (k2 * h) = 2.980573 + (3.982184 * 0.100000) = 3.378792
x10 = x9 + h = 0.900000 + 0.100000 = 1.000000
Iteracion 11:
k1 = f(x10, y10) = f(1.000000, 3.378792) = 4.220263
k2 = f(x10 + h/2, y10 + (k1 * h)/2) = f(1.000000 + 0.100000/2, 3.378792 + (4.220263 * 0.100000)/2) = 4.457228
y11 = y10 + (k2 * h) = 3.378792 + (4.457228 * 0.100000) = 3.824514
x11 = x10 + h = 1.000000 + 0.100000 = 1.100000
Iteracion 12:
k1 = f(x11, y11) = f(1.100000, 3.824514) = 4.715722
k2 = f(x11 + h/2, y11 + (k1 * h)/2) = f(1.100000 + 0.100000/2, 3.824514 + (4.715722 * 0.100000)/2) = 4.973064

y12 = y11 + (k2 * h) = 3.824514 + (4.973064 * 0.100000) = 4.321821
x12 = x11 + h = 1.100000 + 0.100000 = 1.200000
Iteracion 13:
k1 = f(x12, y12) = f(1.200000, 4.321821) = 5.253860
k2 = f(x12 + h/2, y12 + (k1 * h)/2) = f(1.200000 + 0.100000/2, 4.321821 + (5.253860 * 0.100000)/2) = 5.533498
y13 = y12 + (k2 * h) = 4.321821 + (5.533498 * 0.100000) = 4.875171
x13 = x12 + h = 1.200000 + 0.100000 = 1.300000
Iteracion 14:
k1 = f(x13, y13) = f(1.300000, 4.875171) = 5.838729
k2 = f(x13 + h/2, y13 + (k1 * h)/2) = f(1.300000 + 0.100000/2, 4.875171 + (5.838729 * 0.100000)/2) = 6.142830
y14 = y13 + (k2 * h) = 4.875171 + (6.142830 * 0.100000) = 5.489454
x14 = x13 + h = 1.300000 + 0.100000 = 1.400000
Iteracion 15:
k1 = f(x14, y14) = f(1.400000, 5.489454) = 6.474903
k2 = f(x14 + h/2, y14 + (k1 * h)/2) = f(1.400000 + 0.100000/2, 5.489454 + (6.474903 * 0.100000)/2) = 6.805912
y15 = y14 + (k2 * h) = 5.489454 + (6.805912 * 0.100000) = 6.170045
x15 = x14 + h = 1.400000 + 0.100000 = 1.500000
Iteracion 16:
k1 = f(x15, y15) = f(1.500000, 6.170045) = 7.167540
k2 = f(x15 + h/2, y15 + (k1 * h)/2) = f(1.500000 + 0.100000/2, 6.170045 + (7.167540 * 0.100000)/2) = 7.528206
y16 = y15 + (k2 * h) = 6.170045 + (7.528206 * 0.100000) = 6.922866
x16 = x15 + h = 1.500000 + 0.100000 = 1.600000
Iteracion 17:
k1 = f(x16, y16) = f(1.600000, 6.922866) = 7.922439
k2 = f(x16 + h/2, y16 + (k1 * h)/2) = f(1.600000 + 0.100000/2, 6.922866 + (7.922439 * 0.100000)/2) = 8.315852
y17 = y16 + (k2 * h) = 6.922866 + (8.315852 * 0.100000) = 7.754451
x17 = x16 + h = 1.600000 + 0.100000 = 1.700000
Iteracion 18:
k1 = f(x17, y17) = f(1.700000, 7.754451) = 8.746116
k2 = f(x17 + h/2, y17 + (k1 * h)/2) = f(1.700000 + 0.100000/2, 7.754451 + (8.746116 * 0.100000)/2) = 9.175742
y18 = y17 + (k2 * h) = 7.754451 + (9.175742 * 0.100000) = 8.672025
x18 = x17 + h = 1.700000 + 0.100000 = 1.800000
Iteracion 19:
k1 = f(x18, y18) = f(1.800000, 8.672025) = 9.645873
k2 = f(x18 + h/2, y18 + (k1 * h)/2) = f(1.800000 + 0.100000/2, 8.672025 + (9.645873 * 0.100000)/2) = 10.115594
y19 = y18 + (k2 * h) = 8.672025 + (10.115594 * 0.100000) = 9.683584
x19 = x18 + h = 1.800000 + 0.100000 = 1.900000
Iteracion 20:
k1 = f(x19, y19) = f(1.900000, 9.683584) = 10.629884
k2 = f(x19 + h/2, y19 + (k1 * h)/2) = f(1.900000 + 0.100000/2, 9.683584 + (10.629884 * 0.100000)/2) = 11.144038

y20 = y19 + (k2 * h) = 9.683584 + (11.144038 * 0.100000) = 10.797988
x20 = x19 + h = 1.900000 + 0.100000 = 2.000000
El valor de Y = 10.797988

Ejecución:

```
Algoritmo de Punto Medio
Ingresa el valor de X: 0
Ingresa el valor de XF: 2
Ingresa el valor de Y: 1
Ingresa el numero de iteraciones: 20
El valor de Y = 10.797988
Tiempo de ejecucion: 0.000000 segundos
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

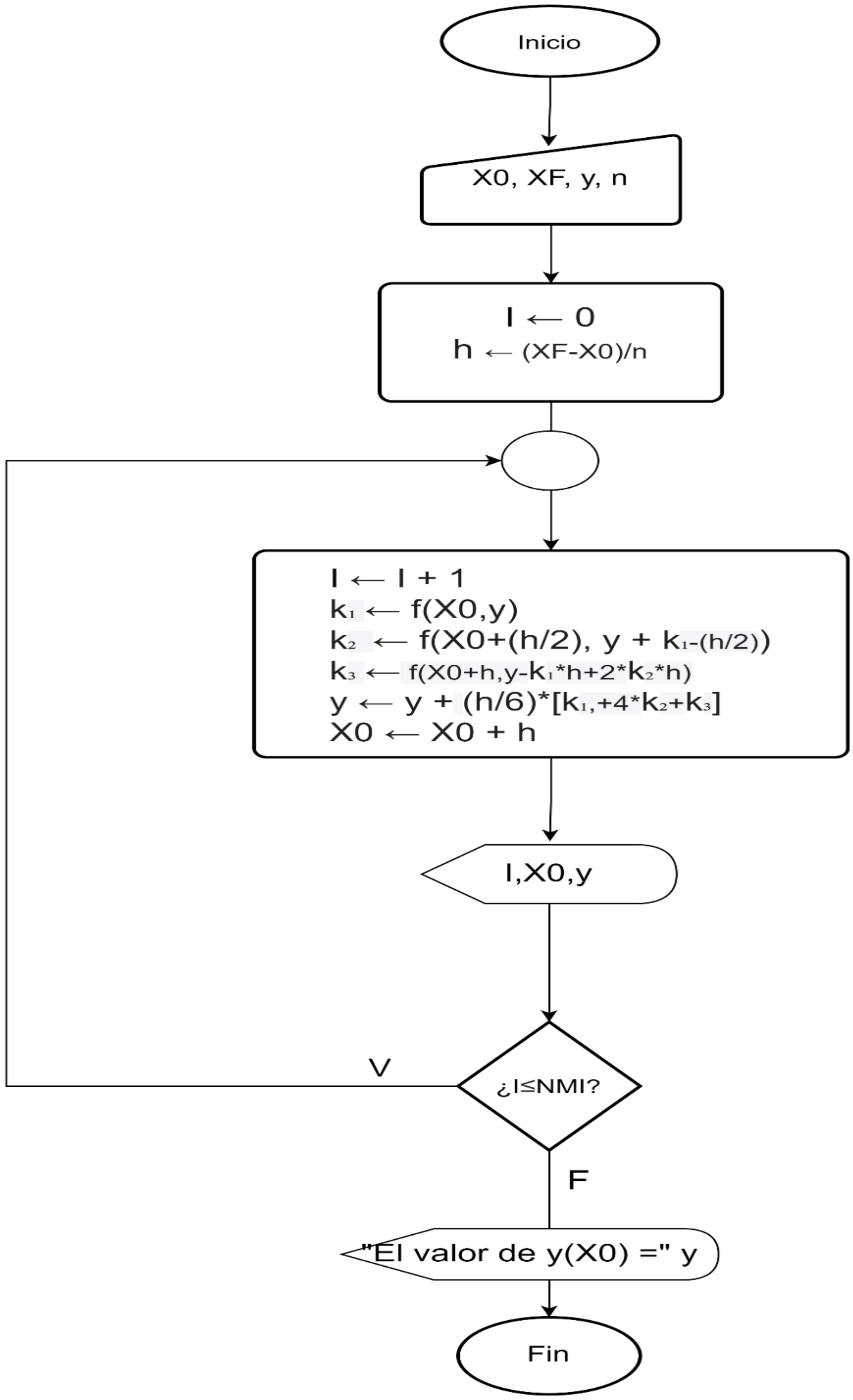
Algoritmo de Runge-Kutta(tercer orden)

Maestro: Dr. Cecil Vidriales

Algoritmo de Runge-Kutta de Tercer Orden

Descripción: Este método utiliza una combinación de pendientes en varios puntos dentro del intervalo para lograr una precisión de tercer orden.

Utilidad: Proporciona una mayor precisión en la solución de EDOs que los métodos de menor orden.



Pseudocódigo del Método de Runge-Kutta de Tercer Orden

1. Inicio
2. Inicializar Variables:
    - x = x0
    - y
    - xf
    - n
    - NMI
    - h = (xf - x0) / n
3. Bucle For (i = 0; i < n; i++):
    - k1 = f(x, y)
    - k2 = f(x + h / 3, y + h * k1 / 3)
    - k3 = f(x + 2 * h / 3, y - h * k1 / 3 + h * k2)
    - y = y + h * (k1 / 8 + 3 * k2 / 8 + 3 * k3 / 8)
    - x = x + h
4. Fin del Bucle For
5. Imprimir: i, x, y
6. Condición: Si i <= NMI entonces
    - Imprimir: i, x, y
7. Fin

Código:

```
double x0, xf, y, h, n;
printf("Ingresa el valor de X: ");
scanf("%lf", &x0);

printf("Ingresa el valor de XF: ");
scanf("%lf", &xf);

printf("Ingresa el valor de Y: ");
scanf("%lf", &y);

printf("Ingresa el numero de iteraciones: ");
scanf("%lf", &n);

h = (xf - x0) / n;

for (int i = 0; i < n; ++i) {
    double k1 = f(x0, y);
    double k2 = f(x0 + h/2, y + k1 * h/2);
    double k3 = f(x0 + h, y - k1 * h + 2 * k2 * h);
    y = y + (h / 6) * (k1 + 4 * k2 + k3);
    x0 = x0 + h;
}   printf("El valor de Y = %lf", y);
```

prueba de escritorio
f(x,y)=sin(x) + y;
Ingresa el valor de X: 0
Ingresa el valor de XF: 3
Ingresa el valor de Y: 2
Ingresa el numero de iteraciones: 20
Iteracion 1:
k1 = f(x0, y0) = f(0.000000, 2.000000) = 2.000000
k2 = f(x0 + h/2, y0 + k1 * h/2) = f(0.000000 + 0.150000/2, 2.000000 + 2.000000 * 0.150000/2) = 2.224930
k3 = f(x0 + h, y0 - k1 * h + 2 * k2 * h) = f(0.000000 + 0.150000, 2.000000 - 2.000000 * 0.150000 + 2 * 2.224930 * 0.150000) = 2.516917
y1 = y0 + (h / 6) * (k1 + 4 * k2 + k3) = 2.000000 + (0.150000 / 6) * (2.000000 + 4 * 2.224930 + 2.516917) = 2.335416
x1 = x0 + h = 0.000000 + 0.150000 = 0.150000
Iteracion 2:
k1 = f(x1, y1) = f(0.150000, 2.335416) = 2.484854
k2 = f(x1 + h/2, y1 + k1 * h/2) = f(0.150000 + 0.150000/2, 2.335416 + 2.484854 * 0.150000/2) = 2.744886
k3 = f(x1 + h, y1 - k1 * h + 2 * k2 * h) = f(0.150000 + 0.150000, 2.335416 - 2.484854 * 0.150000 + 2 * 2.744886 * 0.150000) = 3.081674
y2 = y1 + (h / 6) * (k1 + 4 * k2 + k3) = 2.335416 + (0.150000 / 6) * (2.484854 + 4 * 2.744886 + 3.081674) = 2.749068
x2 = x1 + h = 0.150000 + 0.150000 = 0.300000
Iteracion 3:
k1 = f(x2, y2) = f(0.300000, 2.749068) = 3.044588
k2 = f(x2 + h/2, y2 + k1 * h/2) = f(0.300000 + 0.150000/2, 2.749068 + 3.044588 * 0.150000/2) = 3.343684
k3 = f(x2 + h, y2 - k1 * h + 2 * k2 * h) = f(0.300000 + 0.150000, 2.749068 - 3.044588 * 0.150000 + 2 * 3.343684 * 0.150000) = 3.730450
y3 = y2 + (h / 6) * (k1 + 4 * k2 + k3) = 2.749068 + (0.150000 / 6) * (3.044588 + 4 * 3.343684 + 3.730450) = 3.252812
x3 = x2 + h = 0.300000 + 0.150000 = 0.450000
Iteracion 4:
k1 = f(x3, y3) = f(0.450000, 3.252812) = 3.687778
k2 = f(x3 + h/2, y3 + k1 * h/2) = f(0.450000 + 0.150000/2, 3.252812 + 3.687778 * 0.150000/2) = 4.030608
k3 = f(x3 + h, y3 - k1 * h + 2 * k2 * h) = f(0.450000 + 0.150000, 3.252812 - 3.687778 * 0.150000 + 2 * 4.030608 * 0.150000) = 4.473470
y4 = y3 + (h / 6) * (k1 + 4 * k2 + k3) = 3.252812 + (0.150000 / 6) * (3.687778 + 4 * 4.030608 + 4.473470) = 3.859904
x4 = x3 + h = 0.450000 + 0.150000 = 0.600000
Iteracion 5:
k1 = f(x4, y4) = f(0.600000, 3.859904) = 4.424547
k2 = f(x4 + h/2, y4 + k1 * h/2) = f(0.600000 + 0.150000/2, 3.859904 + 4.424547 * 0.150000/2) = 4.816642
k3 = f(x4 + h, y4 - k1 * h + 2 * k2 * h) = f(0.600000 + 0.150000, 3.859904 - 4.424547 * 0.150000 + 2 * 4.816642 * 0.150000) = 5.322854

y5 = y4 + (h / 6) * (k1 + 4 * k2 + k3) = 3.859904 + (0.150000 / 6) * (4.424547 + 4 * 4.816642 + 5.322854) = 4.585253
x5 = x4 + h = 0.600000 + 0.150000 = 0.750000
Iteracion 6:
k1 = f(x5, y5) = f(0.750000, 4.585253) = 5.266892
k2 = f(x5 + h/2, y5 + k1 * h/2) = f(0.750000 + 0.150000/2, 4.585253 + 5.266892 * 0.150000/2) = 5.714818
k3 = f(x5 + h, y5 - k1 * h + 2 * k2 * h) = f(0.750000 + 0.150000, 4.585253 - 5.266892 * 0.150000 + 2 * 5.714818 * 0.150000) = 6.292992
y6 = y5 + (h / 6) * (k1 + 4 * k2 + k3) = 4.585253 + (0.150000 / 6) * (5.266892 + 4 * 5.714818 + 6.292992) = 5.445732
x6 = x5 + h = 0.750000 + 0.150000 = 0.900000
Iteracion 7:
k1 = f(x6, y6) = f(0.900000, 5.445732) = 6.229059
k2 = f(x6 + h/2, y6 + k1 * h/2) = f(0.900000 + 0.150000/2, 5.445732 + 6.229059 * 0.150000/2) = 6.740614
k3 = f(x6 + h, y6 - k1 * h + 2 * k2 * h) = f(0.900000 + 0.150000, 5.445732 - 6.229059 * 0.150000 + 2 * 6.740614 * 0.150000) = 7.400981
y7 = y6 + (h / 6) * (k1 + 4 * k2 + k3) = 5.445732 + (0.150000 / 6) * (6.229059 + 4 * 6.740614 + 7.400981) = 6.460545
x7 = x6 + h = 0.900000 + 0.150000 = 1.050000
Iteracion 8:
k1 = f(x7, y7) = f(1.050000, 6.460545) = 7.327968
k2 = f(x7 + h/2, y7 + k1 * h/2) = f(1.050000 + 0.150000/2, 6.460545 + 7.327968 * 0.150000/2) = 7.912410
k3 = f(x7 + h, y7 - k1 * h + 2 * k2 * h) = f(1.050000 + 0.150000, 6.460545 - 7.327968 * 0.150000 + 2 * 7.912410 * 0.150000) = 8.667112
y8 = y7 + (h / 6) * (k1 + 4 * k2 + k3) = 6.460545 + (0.150000 / 6) * (7.327968 + 4 * 7.912410 + 8.667112) = 7.651663
x8 = x7 + h = 1.050000 + 0.150000 = 1.200000
Iteracion 9:
k1 = f(x8, y8) = f(1.200000, 7.651663) = 8.583702
k2 = f(x8 + h/2, y8 + k1 * h/2) = f(1.200000 + 0.150000/2, 7.651663 + 8.583702 * 0.150000/2) = 9.252011
k3 = f(x8 + h, y8 - k1 * h + 2 * k2 * h) = f(1.200000 + 0.150000, 7.651663 - 8.583702 * 0.150000 + 2 * 9.252011 * 0.150000) = 10.115434
y9 = y8 + (h / 6) * (k1 + 4 * k2 + k3) = 7.651663 + (0.150000 / 6) * (8.583702 + 4 * 9.252011 + 10.115434) = 9.044342
x9 = x8 + h = 1.200000 + 0.150000 = 1.350000
Iteracion 10:
k1 = f(x9, y9) = f(1.350000, 9.044342) = 10.020065
k2 = f(x9 + h/2, y9 + k1 * h/2) = f(1.350000 + 0.150000/2, 9.044342 + 10.020065 * 0.150000/2) = 10.785238
k3 = f(x9 + h, y9 - k1 * h + 2 * k2 * h) = f(1.350000 + 0.150000, 9.044342 - 10.020065 * 0.150000 + 2 * 10.785238 * 0.150000) = 11.774399
y10 = y9 + (h / 6) * (k1 + 4 * k2 + k3) = 9.044342 + (0.150000 / 6) * (10.020065 + 4 * 10.785238 + 11.774399) = 10.667727
x10 = x9 + h = 1.350000 + 0.150000 = 1.500000

Iteracion 11:
k1 = f(x10, y10) = f(1.500000, 10.667727) = 11.665222
k2 = f(x10 + h/2, y10 + k1 * h/2) = f(1.500000 + 0.150000/2, 10.667727 + 11.665222 * 0.150000/2) = 12.542610
k3 = f(x10 + h, y10 - k1 * h + 2 * k2 * h) = f(1.500000 + 0.150000, 10.667727 - 11.665222 * 0.150000 + 2 * 12.542610 * 0.150000) = 13.677592
y11 = y10 + (h / 6) * (k1 + 4 * k2 + k3) = 10.667727 + (0.150000 / 6) * (11.665222 + 4 * 12.542610 + 13.677592) = 12.555559
x11 = x10 + h = 1.500000 + 0.150000 = 1.650000
Iteracion 12:
k1 = f(x11, y11) = f(1.650000, 12.555559) = 13.552424
k2 = f(x11 + h/2, y11 + k1 * h/2) = f(1.650000 + 0.150000/2, 12.555559 + 13.552424 * 0.150000/2) = 14.560125
k3 = f(x11 + h, y11 - k1 * h + 2 * k2 * h) = f(1.650000 + 0.150000, 12.555559 - 13.552424 * 0.150000 + 2 * 14.560125 * 0.150000) = 15.864580
y12 = y11 + (h / 6) * (k1 + 4 * k2 + k3) = 12.555559 + (0.150000 / 6) * (13.552424 + 4 * 14.560125 + 15.864580) = 14.746996
x12 = x11 + h = 1.650000 + 0.150000 = 1.800000
Iteracion 13:
k1 = f(x12, y12) = f(1.800000, 14.746996) = 15.720844
k2 = f(x12 + h/2, y12 + k1 * h/2) = f(1.800000 + 0.150000/2, 14.746996 + 15.720844 * 0.150000/2) = 16.880146
k3 = f(x12 + h, y12 - k1 * h + 2 * k2 * h) = f(1.800000 + 0.150000, 14.746996 - 15.720844 * 0.150000 + 2 * 16.880146 * 0.150000) = 18.381873
y13 = y12 + (h / 6) * (k1 + 4 * k2 + k3) = 14.746996 + (0.150000 / 6) * (15.720844 + 4 * 16.880146 + 18.381873) = 17.287579
x13 = x12 + h = 1.800000 + 0.150000 = 1.950000
Iteracion 14:
k1 = f(x13, y13) = f(1.950000, 17.287579) = 18.216539
k2 = f(x13 + h/2, y13 + k1 * h/2) = f(1.950000 + 0.150000/2, 17.287579 + 18.216539 * 0.150000/2) = 19.552430
k3 = f(x13 + h, y13 - k1 * h + 2 * k2 * h) = f(1.950000 + 0.150000, 17.287579 - 18.216539 * 0.150000 + 2 * 19.552430 * 0.150000) = 21.284037
y14 = y13 + (h / 6) * (k1 + 4 * k2 + k3) = 17.287579 + (0.150000 / 6) * (18.216539 + 4 * 19.552430 + 21.284037) = 20.230336
x14 = x13 + h = 1.950000 + 0.150000 = 2.100000
Iteracion 15:
k1 = f(x14, y14) = f(2.100000, 20.230336) = 21.093546
k2 = f(x14 + h/2, y14 + k1 * h/2) = f(2.100000 + 0.150000/2, 20.230336 + 21.093546 * 0.150000/2) = 22.635307
k3 = f(x14 + h, y14 - k1 * h + 2 * k2 * h) = f(2.100000 + 0.150000, 20.230336 - 21.093546 * 0.150000 + 2 * 22.635307 * 0.150000) = 24.634970
y15 = y14 + (h / 6) * (k1 + 4 * k2 + k3) = 20.230336 + (0.150000 / 6) * (21.093546 + 4 * 22.635307 + 24.634970) = 23.637080
x15 = x14 + h = 2.100000 + 0.150000 = 2.250000
Iteracion 16:
k1 = f(x15, y15) = f(2.250000, 23.637080) = 24.415153

k2 = f(x15 + h/2, y15 + k1 * h/2) = f(2.250000 + 0.150000/2, 23.637080 + 24.415153 * 0.150000/2) = 26.197033
k3 = f(x15 + h, y15 - k1 * h + 2 * k2 * h) = f(2.250000 + 0.150000, 23.637080 - 24.415153 * 0.150000 + 2 * 26.197033 * 0.150000) = 28.509380
y16 = y15 + (h / 6) * (k1 + 4 * k2 + k3) = 23.637080 + (0.150000 / 6) * (24.415153 + 4 * 26.197033 + 28.509380) = 27.579897
x16 = x15 + h = 2.250000 + 0.150000 = 2.400000
Iteracion 17:
k1 = f(x16, y16) = f(2.400000, 27.579897) = 28.255360
k2 = f(x16 + h/2, y16 + k1 * h/2) = f(2.400000 + 0.150000/2, 27.579897 + 28.255360 * 0.150000/2) = 30.317360
k3 = f(x16 + h, y16 - k1 * h + 2 * k2 * h) = f(2.400000 + 0.150000, 27.579897 - 28.255360 * 0.150000 + 2 * 30.317360 * 0.150000) = 32.994484
y17 = y16 + (h / 6) * (k1 + 4 * k2 + k3) = 27.579897 + (0.150000 / 6) * (28.255360 + 4 * 30.317360 + 32.994484) = 32.142879
x17 = x16 + h = 2.400000 + 0.150000 = 2.550000
Iteracion 18:
k1 = f(x17, y17) = f(2.550000, 32.142879) = 32.700563
k2 = f(x17 + h/2, y17 + k1 * h/2) = f(2.550000 + 0.150000/2, 32.142879 + 32.700563 * 0.150000/2) = 35.089341
k3 = f(x17 + h, y17 - k1 * h + 2 * k2 * h) = f(2.550000 + 0.150000, 32.142879 - 32.700563 * 0.150000 + 2 * 35.089341 * 0.150000) = 38.191977
y18 = y17 + (h / 6) * (k1 + 4 * k2 + k3) = 32.142879 + (0.150000 / 6) * (32.700563 + 4 * 35.089341 + 38.191977) = 37.424126
x18 = x17 + h = 2.550000 + 0.150000 = 2.700000
Iteracion 19:
k1 = f(x18, y18) = f(2.700000, 37.424126) = 37.851506
k2 = f(x18 + h/2, y18 + k1 * h/2) = f(2.700000 + 0.150000/2, 37.424126 + 37.851506 * 0.150000/2) = 40.621426
k3 = f(x18 + h, y18 - k1 * h + 2 * k2 * h) = f(2.700000 + 0.150000, 37.424126 - 37.851506 * 0.150000 + 2 * 40.621426 * 0.150000) = 44.220306
y19 = y18 + (h / 6) * (k1 + 4 * k2 + k3) = 37.424126 + (0.150000 / 6) * (37.851506 + 4 * 40.621426 + 44.220306) = 43.538064
x19 = x18 + h = 2.700000 + 0.150000 = 2.850000
Iteracion 20:
k1 = f(x19, y19) = f(2.850000, 43.538064) = 43.825542
k2 = f(x19 + h/2, y19 + k1 * h/2) = f(2.850000 + 0.150000/2, 43.538064 + 43.825542 * 0.150000/2) = 47.039883
k3 = f(x19 + h, y19 - k1 * h + 2 * k2 * h) = f(2.850000 + 0.150000, 43.538064 - 43.825542 * 0.150000 + 2 * 47.039883 * 0.150000) = 51.217318
y20 = y19 + (h / 6) * (k1 + 4 * k2 + k3) = 43.538064 + (0.150000 / 6) * (43.825542 + 4 * 47.039883 + 51.217318) = 50.618124
x20 = x19 + h = 2.850000 + 0.150000 = 3.000000
El valor de Y = 50.618124

Ejecución:

```
Algoritmo de Runge-Kutta(3er orden)
Ingresa el valor de X: 0
Ingresa el valor de XF: 3
Ingresa el valor de Y: 2
Ingresa el numero de iteraciones: 20
El valor de Y = 50.618124
Tiempo de ejecucion: 0.000000 segundos
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

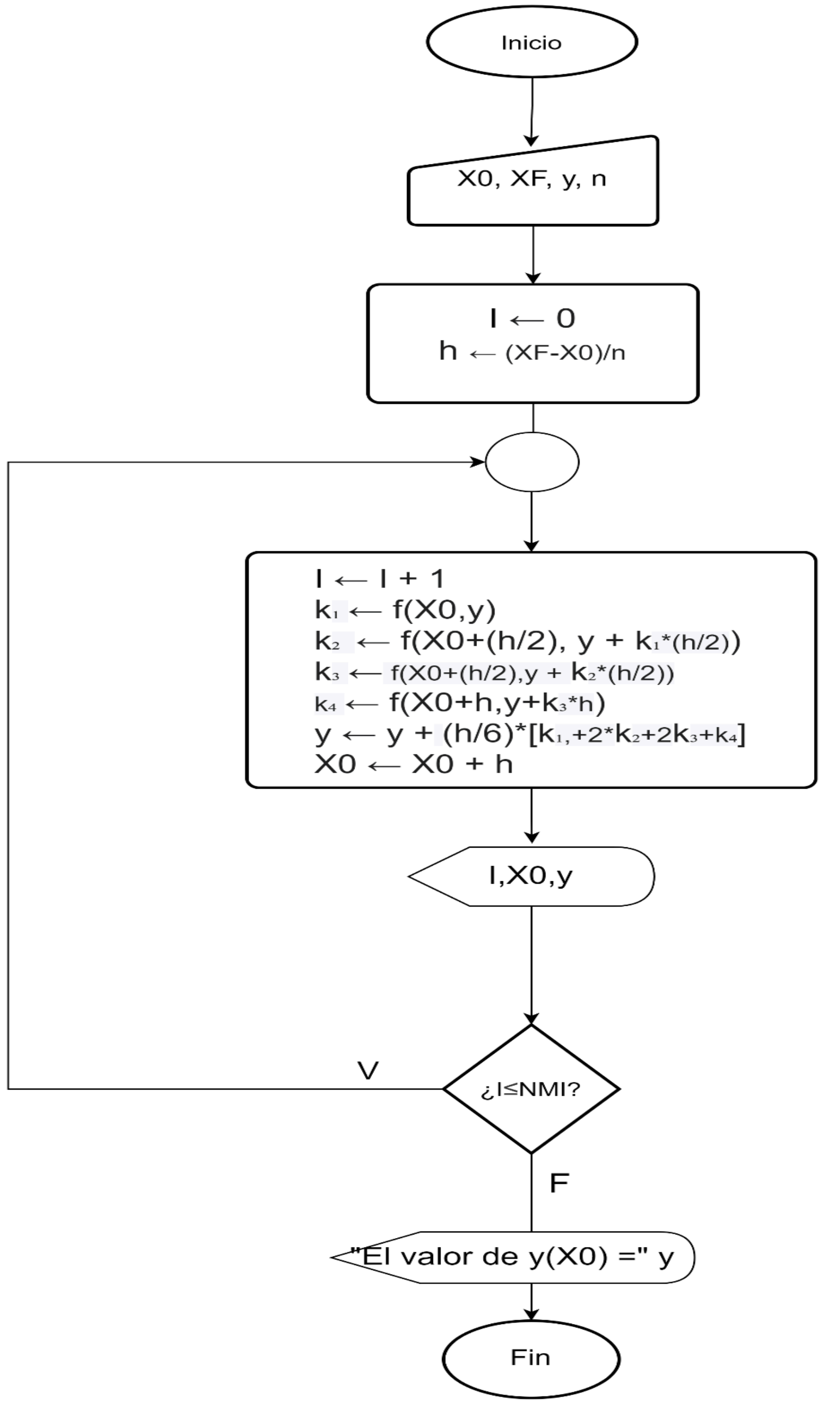
Algoritmo de Runge-Kutta(Cuarto orden)

Maestro: Dr. Cecil Vidriales

Algoritmo de Runge-Kutta de Cuarto Orden

Descripción: Uno de los métodos más populares, el Algoritmo de Runge-Kutta de Cuarto Orden (RK4), utiliza cuatro evaluaciones de la pendiente por intervalo para alcanzar una alta precisión.

Utilidad: Es ampliamente usado por su balance entre precisión y eficiencia computacional en la solución de EDOs.



Pseudocódigo del Método de Runge-Kutta de Cuarto Orden

1. Inicio
2. Inicializar Variables:
    - x = x0
    - y
    - xf
    - n
    - NMI
    - h = (xf - x0) / n
3. Bucle For (i = 0; i < n; i++):
    - k1 = f(x, y)
    - k2 = f(x + h / 2, y + h / 2 * k1)
    - k3 = f(x + h / 2, y + h / 2 * k2)
    - k4 = f(x + h, y + h * k3)
    - y = y + h / 6 * (k1 + 2 * k2 + 2 * k3 + k4)
    - x = x + h
4. Fin del Bucle For
5. Imprimir: i, x, y
6. Condición: Si i <= NMI entonces
    - Imprimir: i, x, y
7. Fin

Código:

```
double x0, xf, y, h, n;
printf("Ingresa el valor de X: ");
scanf("%lf", &x0);

printf("Ingresa el valor de XF: ");
scanf("%lf", &xf);

printf("Ingresa el valor de Y: ");
scanf("%lf", &y);

printf("Ingresa el numero de iteraciones: ");
scanf("%lf", &n);

h = (xf - x0) / n;

for (int i = 0; i < n; ++i) {
    double k1 = f(x0, y);
    double k2 = f(x0 + h/2, y + k1 * h/2);
    double k3 = f(x0 + h/2, y + k2 * h/2); // Corrección aquí
    double k4 = f(x0 + h, y + k3 * h);
    y = y + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4);
    x0 = x0 + h;
printf("El valor de Y = %lf", y);
```

Prueba de escritorio
f(x,y)=sin(x) + y;
Ingresa el valor de X: 0
Ingresa el valor de XF: 2
Ingresa el valor de Y: 1
Ingresa el numero de iteraciones: 20
Iteracion 1:
k1 = f(x0, y0) = f(0.000000, 1.000000) = 1.000000
k2 = f(x0 + h/2, y0 + k1 * h/2) = f(0.000000 + 0.100000/2, 1.000000 + 1.000000 * 0.100000/2) = 1.099979
k3 = f(x0 + h/2, y0 + k2 * h/2) = f(0.000000 + 0.100000/2, 1.000000 + 1.099979 * 0.100000/2) = 1.104978
k4 = f(x0 + h, y0 + k3 * h) = f(0.000000 + 0.100000, 1.000000 + 1.104978 * 0.100000) = 1.210331
y1 = y0 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 1.000000 + (0.100000 / 6) * (1.000000 + 2 * 1.099979 + 2 * 1.104978 + 1.210331) = 1.110337
x1 = x0 + h = 0.000000 + 0.100000 = 0.100000
Iteracion 2:
k1 = f(x1, y1) = f(0.100000, 1.110337) = 1.210171
k2 = f(x1 + h/2, y1 + k1 * h/2) = f(0.100000 + 0.100000/2, 1.110337 + 1.210171 * 0.100000/2) = 1.320284
k3 = f(x1 + h/2, y1 + k2 * h/2) = f(0.100000 + 0.100000/2, 1.110337 + 1.320284 * 0.100000/2) = 1.325790
k4 = f(x1 + h, y1 + k3 * h) = f(0.100000 + 0.100000, 1.110337 + 1.325790 * 0.100000) = 1.441586
y2 = y1 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 1.110337 + (0.100000 / 6) * (1.210171 + 2 * 1.320284 + 2 * 1.325790 + 1.441586) = 1.242736
x2 = x1 + h = 0.100000 + 0.100000 = 0.200000
Iteracion 3:
k1 = f(x2, y2) = f(0.200000, 1.242736) = 1.441405
k2 = f(x2 + h/2, y2 + k1 * h/2) = f(0.200000 + 0.100000/2, 1.242736 + 1.441405 * 0.100000/2) = 1.562210
k3 = f(x2 + h/2, y2 + k2 * h/2) = f(0.200000 + 0.100000/2, 1.242736 + 1.562210 * 0.100000/2) = 1.568250
k4 = f(x2 + h, y2 + k3 * h) = f(0.200000 + 0.100000, 1.242736 + 1.568250 * 0.100000) = 1.695081
y3 = y2 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 1.242736 + (0.100000 / 6) * (1.441405 + 2 * 1.562210 + 2 * 1.568250 + 1.695081) = 1.399359
x3 = x2 + h = 0.200000 + 0.100000 = 0.300000
Iteracion 4:
k1 = f(x3, y3) = f(0.300000, 1.399359) = 1.694879
k2 = f(x3 + h/2, y3 + k1 * h/2) = f(0.300000 + 0.100000/2, 1.399359 + 1.694879 * 0.100000/2) = 1.827001
k3 = f(x3 + h/2, y3 + k2 * h/2) = f(0.300000 + 0.100000/2, 1.399359 + 1.827001 * 0.100000/2) = 1.833607
k4 = f(x3 + h, y3 + k3 * h) = f(0.300000 + 0.100000, 1.399359 + 1.833607 * 0.100000) = 1.972138

y4 = y3 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 1.399359 + (0.100000 / 6) * (1.694879 + 2 * 1.827001 + 2 * 1.833607 + 1.972138) = 1.582497
x4 = x3 + h = 0.300000 + 0.100000 = 0.400000
Iteracion 5:
k1 = f(x4, y4) = f(0.400000, 1.582497) = 1.971915
k2 = f(x4 + h/2, y4 + k1 * h/2) = f(0.400000 + 0.100000/2, 1.582497 + 1.971915 * 0.100000/2) = 2.116058
k3 = f(x4 + h/2, y4 + k2 * h/2) = f(0.400000 + 0.100000/2, 1.582497 + 2.116058 * 0.100000/2) = 2.123265
k4 = f(x4 + h, y4 + k3 * h) = f(0.400000 + 0.100000, 1.582497 + 2.123265 * 0.100000) = 2.274249
y5 = y4 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 1.582497 + (0.100000 / 6) * (1.971915 + 2 * 2.116058 + 2 * 2.123265 + 2.274249) = 1.794577
x5 = x4 + h = 0.400000 + 0.100000 = 0.500000
Iteracion 6:
k1 = f(x5, y5) = f(0.500000, 1.794577) = 2.274002
k2 = f(x5 + h/2, y5 + k1 * h/2) = f(0.500000 + 0.100000/2, 1.794577 + 2.274002 * 0.100000/2) = 2.430964
k3 = f(x5 + h/2, y5 + k2 * h/2) = f(0.500000 + 0.100000/2, 1.794577 + 2.430964 * 0.100000/2) = 2.438812
k4 = f(x5 + h, y5 + k3 * h) = f(0.500000 + 0.100000, 1.794577 + 2.438812 * 0.100000) = 2.603100
y6 = y5 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 1.794577 + (0.100000 / 6) * (2.274002 + 2 * 2.430964 + 2 * 2.438812 + 2.603100) = 2.038188
x6 = x5 + h = 0.500000 + 0.100000 = 0.600000
Iteracion 7:
k1 = f(x6, y6) = f(0.600000, 2.038188) = 2.602830
k2 = f(x6 + h/2, y6 + k1 * h/2) = f(0.600000 + 0.100000/2, 2.038188 + 2.602830 * 0.100000/2) = 2.773515
k3 = f(x6 + h/2, y6 + k2 * h/2) = f(0.600000 + 0.100000/2, 2.038188 + 2.773515 * 0.100000/2) = 2.782050
k4 = f(x6 + h, y6 + k3 * h) = f(0.600000 + 0.100000, 2.038188 + 2.782050 * 0.100000) = 2.960610
y7 = y6 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 2.038188 + (0.100000 / 6) * (2.602830 + 2 * 2.773515 + 2 * 2.782050 + 2.960610) = 2.316097
x7 = x6 + h = 0.600000 + 0.100000 = 0.700000
Iteracion 8:
k1 = f(x7, y7) = f(0.700000, 2.316097) = 2.960315
k2 = f(x7 + h/2, y7 + k1 * h/2) = f(0.700000 + 0.100000/2, 2.316097 + 2.960315 * 0.100000/2) = 3.145752
k3 = f(x7 + h/2, y7 + k2 * h/2) = f(0.700000 + 0.100000/2, 2.316097 + 3.145752 * 0.100000/2) = 3.155023
k4 = f(x7 + h, y7 + k3 * h) = f(0.700000 + 0.100000, 2.316097 + 3.155023 * 0.100000) = 3.348956
y8 = y7 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 2.316097 + (0.100000 / 6) * (2.960315 + 2 * 3.145752 + 2 * 3.155023 + 3.348956) = 2.631277
x8 = x7 + h = 0.700000 + 0.100000 = 0.800000
Iteracion 9:

k1 = f(x8, y8) = f(0.800000, 2.631277) = 3.348634
k2 = f(x8 + h/2, y8 + k1 * h/2) = f(0.800000 + 0.100000/2, 2.631277 + 3.348634 * 0.100000/2) = 3.549990
k3 = f(x8 + h/2, y8 + k2 * h/2) = f(0.800000 + 0.100000/2, 2.631277 + 3.549990 * 0.100000/2) = 3.560057
k4 = f(x8 + h, y8 + k3 * h) = f(0.800000 + 0.100000, 2.631277 + 3.560057 * 0.100000) = 3.770610
y9 = y8 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 2.631277 + (0.100000 / 6) * (3.348634 + 2 * 3.549990 + 2 * 3.560057 + 3.770610) = 2.986933
x9 = x8 + h = 0.800000 + 0.100000 = 0.900000
Iteracion 10:
k1 = f(x9, y9) = f(0.900000, 2.986933) = 3.770260
k2 = f(x9 + h/2, y9 + k1 * h/2) = f(0.900000 + 0.100000/2, 2.986933 + 3.770260 * 0.100000/2) = 3.988862
k3 = f(x9 + h/2, y9 + k2 * h/2) = f(0.900000 + 0.100000/2, 2.986933 + 3.988862 * 0.100000/2) = 3.999792
k4 = f(x9 + h, y9 + k3 * h) = f(0.900000 + 0.100000, 2.986933 + 3.999792 * 0.100000) = 4.228383
y10 = y9 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 2.986933 + (0.100000 / 6) * (3.770260 + 2 * 3.988862 + 2 * 3.999792 + 4.228383) = 3.386532
x10 = x9 + h = 0.900000 + 0.100000 = 1.000000
Iteracion 11:
k1 = f(x10, y10) = f(1.000000, 3.386532) = 4.228003
k2 = f(x10 + h/2, y10 + k1 * h/2) = f(1.000000 + 0.100000/2, 3.386532 + 4.228003 * 0.100000/2) = 4.465356
k3 = f(x10 + h/2, y10 + k2 * h/2) = f(1.000000 + 0.100000/2, 3.386532 + 4.465356 * 0.100000/2) = 4.477223
k4 = f(x10 + h, y10 + k3 * h) = f(1.000000 + 0.100000, 3.386532 + 4.477223 * 0.100000) = 4.725462
y11 = y10 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 3.386532 + (0.100000 / 6) * (4.228003 + 2 * 4.465356 + 2 * 4.477223 + 4.725462) = 3.833843
x11 = x10 + h = 1.000000 + 0.100000 = 1.100000
Iteracion 12:
k1 = f(x11, y11) = f(1.100000, 3.833843) = 4.725050
k2 = f(x11 + h/2, y11 + k1 * h/2) = f(1.100000 + 0.100000/2, 3.833843 + 4.725050 * 0.100000/2) = 4.982859
k3 = f(x11 + h/2, y11 + k2 * h/2) = f(1.100000 + 0.100000/2, 3.833843 + 4.982859 * 0.100000/2) = 4.995749
k4 = f(x11 + h, y11 + k3 * h) = f(1.100000 + 0.100000, 3.833843 + 4.995749 * 0.100000) = 5.265457
y12 = y11 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 3.833843 + (0.100000 / 6) * (4.725050 + 2 * 4.982859 + 2 * 4.995749 + 5.265457) = 4.332971
x12 = x11 + h = 1.100000 + 0.100000 = 1.200000
Iteracion 13:
k1 = f(x12, y12) = f(1.200000, 4.332971) = 5.265010
k2 = f(x12 + h/2, y12 + k1 * h/2) = f(1.200000 + 0.100000/2, 4.332971 + 5.265010 * 0.100000/2) = 5.545206

k3 = f(x12 + h/2, y12 + k2 * h/2) = f(1.200000 + 0.100000/2, 4.332971 + 5.545206 * 0.100000/2) = 5.559216
k4 = f(x12 + h, y12 + k3 * h) = f(1.200000 + 0.100000, 4.332971 + 5.559216 * 0.100000) = 5.852451
y13 = y12 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 4.332971 + (0.100000 / 6) * (5.265010 + 2 * 5.545206 + 2 * 5.559216 + 5.852451) = 4.888410
x13 = x12 + h = 1.200000 + 0.100000 = 1.300000
Iteracion 14:
k1 = f(x13, y13) = f(1.300000, 4.888410) = 5.851968
k2 = f(x13 + h/2, y13 + k1 * h/2) = f(1.300000 + 0.100000/2, 4.888410 + 5.851968 * 0.100000/2) = 6.156732
k3 = f(x13 + h/2, y13 + k2 * h/2) = f(1.300000 + 0.100000/2, 4.888410 + 6.156732 * 0.100000/2) = 6.171970
k4 = f(x13 + h, y13 + k3 * h) = f(1.300000 + 0.100000, 4.888410 + 6.171970 * 0.100000) = 6.491056
y14 = y13 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 4.888410 + (0.100000 / 6) * (5.851968 + 2 * 6.156732 + 2 * 6.171970 + 6.491056) = 5.505084
x14 = x13 + h = 1.300000 + 0.100000 = 1.400000
Iteracion 15:
k1 = f(x14, y14) = f(1.400000, 5.505084) = 6.490533
k2 = f(x14 + h/2, y14 + k1 * h/2) = f(1.400000 + 0.100000/2, 5.505084 + 6.490533 * 0.100000/2) = 6.822323
k3 = f(x14 + h/2, y14 + k2 * h/2) = f(1.400000 + 0.100000/2, 5.505084 + 6.822323 * 0.100000/2) = 6.838913
k4 = f(x14 + h, y14 + k3 * h) = f(1.400000 + 0.100000, 5.505084 + 6.838913 * 0.100000) = 7.186470
y15 = y14 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 5.505084 + (0.100000 / 6) * (6.490533 + 2 * 6.822323 + 2 * 6.838913 + 7.186470) = 6.188408
x15 = x14 + h = 1.400000 + 0.100000 = 1.500000
Iteracion 16:
k1 = f(x15, y15) = f(1.500000, 6.188408) = 7.185903
k2 = f(x15 + h/2, y15 + k1 * h/2) = f(1.500000 + 0.100000/2, 6.188408 + 7.185903 * 0.100000/2) = 7.547487
k3 = f(x15 + h/2, y15 + k2 * h/2) = f(1.500000 + 0.100000/2, 6.188408 + 7.547487 * 0.100000/2) = 7.565566
k4 = f(x15 + h, y15 + k3 * h) = f(1.500000 + 0.100000, 6.188408 + 7.565566 * 0.100000) = 7.944538
y16 = y15 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 6.188408 + (0.100000 / 6) * (7.185903 + 2 * 7.547487 + 2 * 7.565566 + 7.944538) = 6.944351
x16 = x15 + h = 1.500000 + 0.100000 = 1.600000
Iteracion 17:
k1 = f(x16, y16) = f(1.600000, 6.944351) = 7.943924
k2 = f(x16 + h/2, y16 + k1 * h/2) = f(1.600000 + 0.100000/2, 6.944351 + 7.943924 * 0.100000/2) = 8.338412
k3 = f(x16 + h/2, y16 + k2 * h/2) = f(1.600000 + 0.100000/2, 6.944351 + 8.338412 * 0.100000/2) = 8.358136
k4 = f(x16 + h, y16 + k3 * h) = f(1.600000 + 0.100000, 6.944351 + 8.358136 * 0.100000) = 8.771829

y17 = y16 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 6.944351 + (0.100000 / 6) * (7.943924 + 2 * 8.338412 + 2 * 8.358136 + 8.771829) = 7.779498
x17 = x16 + h = 1.600000 + 0.100000 = 1.700000
Iteracion 18:
k1 = f(x17, y17) = f(1.700000, 7.779498) = 8.771163
k2 = f(x17 + h/2, y17 + k1 * h/2) = f(1.700000 + 0.100000/2, 7.779498 + 8.771163 * 0.100000/2) = 9.202042
k3 = f(x17 + h/2, y17 + k2 * h/2) = f(1.700000 + 0.100000/2, 7.779498 + 9.202042 * 0.100000/2) = 9.223586
k4 = f(x17 + h, y17 + k3 * h) = f(1.700000 + 0.100000, 7.779498 + 9.223586 * 0.100000) = 9.675704
y18 = y17 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 7.779498 + (0.100000 / 6) * (8.771163 + 2 * 9.202042 + 2 * 9.223586 + 9.675704) = 8.701133
x18 = x17 + h = 1.700000 + 0.100000 = 1.800000
Iteracion 19:
k1 = f(x18, y18) = f(1.800000, 8.701133) = 9.674981
k2 = f(x18 + h/2, y18 + k1 * h/2) = f(1.800000 + 0.100000/2, 8.701133 + 9.674981 * 0.100000/2) = 10.146158
k3 = f(x18 + h/2, y18 + k2 * h/2) = f(1.800000 + 0.100000/2, 8.701133 + 10.146158 * 0.100000/2) = 10.169717
k4 = f(x18 + h, y18 + k3 * h) = f(1.800000 + 0.100000, 8.701133 + 10.169717 * 0.100000) = 10.664405
y19 = y18 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 8.701133 + (0.100000 / 6) * (9.674981 + 2 * 10.146158 + 2 * 10.169717 + 10.664405) = 9.717319
x19 = x18 + h = 1.800000 + 0.100000 = 1.900000
Iteracion 20:
k1 = f(x19, y19) = f(1.900000, 9.717319) = 10.663619
k2 = f(x19 + h/2, y19 + k1 * h/2) = f(1.900000 + 0.100000/2, 9.717319 + 10.663619 * 0.100000/2) = 11.179460
k3 = f(x19 + h/2, y19 + k2 * h/2) = f(1.900000 + 0.100000/2, 9.717319 + 11.179460 * 0.100000/2) = 11.205252
k4 = f(x19 + h, y19 + k3 * h) = f(1.900000 + 0.100000, 9.717319 + 11.205252 * 0.100000) = 11.747142
y20 = y19 + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4) = 9.717319 + (0.100000 / 6) * (10.663619 + 2 * 11.179460 + 2 * 11.205252 + 11.747142) = 10.836989
x20 = x19 + h = 1.900000 + 0.100000 = 2.000000
El valor de Y = 10.836989

Ejecución:

```
Algoritmo de Runge-Kutta(4to orden)
Ingresa el valor de X: 0
Ingresa el valor de XF: 2
Ingresa el valor de Y: 1
Ingresa el numero de iteraciones: 20
El valor de Y = 10.836989
Tiempo de ejecucion: 0.000000 segundos
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Algoritmo de Runge-Kutta-Merson

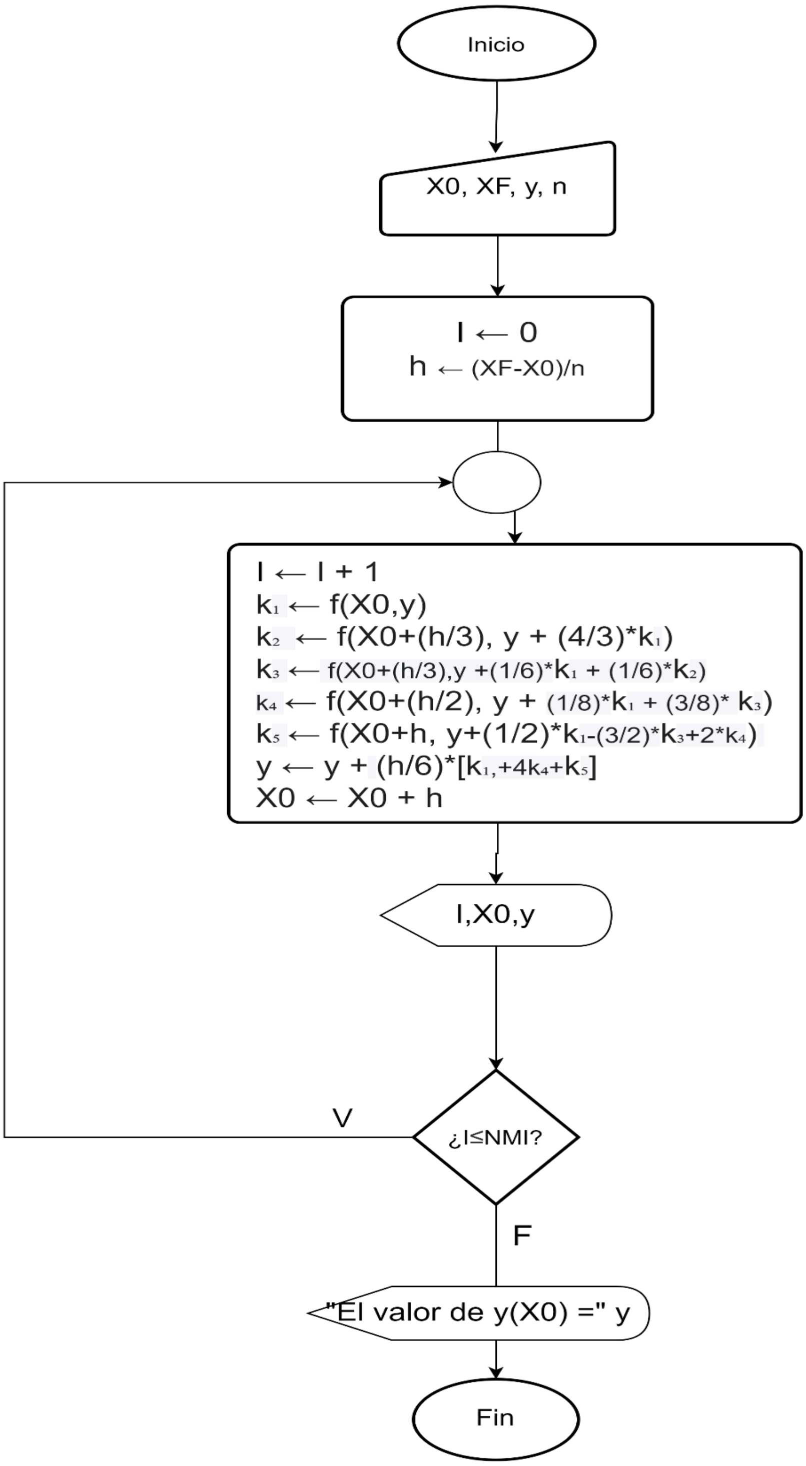
(Quinto orden)

Maestro: Dr. Cecil Vidriales

Algoritmo de Runge-Kutta-Merson de Quinto Orden

Descripción: El Método de Runge-Kutta-Merson es una variación de quinto orden que ajusta automáticamente el tamaño del paso para controlar el error.

Utilidad: Es útil para problemas que requieren alta precisión con control adaptativo del error.



Pseudocódigo del Método de Runge-Kutta-Merson de Quinto Orden

1. Inicio
2. Inicializar Variables:
   - x = x0
   - y
   - xf
   - n
   - NMI
   - h = (xf - x0) / n
3. Bucle For (i = 0; i < n; i++):
   - k1 = f(x, y)
   - k2 = f(x + h / 3, y + h / 3 * k1)
   - k3 = f(x + h / 3, y + h / 6 * k1 + h / 6 * k2)
   - k4 = f(x + h / 2, y + h / 8 * k1 + h / 2 * k3)
   - k5 = f(x + h, y - h / 2 * k2 + h * k3 + 2 * h * k4)
   - y = y + h / 6 * (k1 + 4 * k3 + k4)
   - x = x + h
4. Fin del Bucle For
5. Imprimir: i, x, y
6. Condición: Si i <= NMI entonces
   - Imprimir: i, x, y
7. Fin

Código:

```
double x0, xf, y, h, n;
printf("Ingresa el valor de X: ");
scanf("%lf", &x0);

printf("Ingresa el valor de XF: ");
scanf("%lf", &xf);

printf("Ingresa el valor de Y: ");
scanf("%lf", &y);

printf("Ingresa el numero de iteraciones: ");
scanf("%lf", &n);

h = (xf - x0) / n;

for (int i = 0; i < n; ++i) {
    double k1 = f(x0, y);
    double k2 = f(x0 + h/3, y + (h/3) * k1);
    double k3 = f(x0 + h/3, y + (h/6) * k1 + (h/6) * k2);
    double k4 = f(x0 + h/2, y + (h/8) * k1 + (3*h/8) * k3);
    double k5 = f(x0 + h, y + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2);
    y = y + (h/6) * (k1 + 4*k4 + k5);
    x0 = x0 + h;
   printf("El valor de Y = %lf", y);
```

Prueba de escritorio
f(x,y)=sin(x) + y;
Ingresa el valor de X: 0
Ingresa el valor de XF: 2
Ingresa el valor de Y: 1
Ingresa el numero de iteraciones: 20
Iteracion 1:
k1 = f(x0, y0) = f(0.000000, 1.000000) = 1.000000
k2 = f(x0 + h/3, y0 + (h/3) * k1) = f(0.000000 + 0.100000/3, 1.000000 + (0.100000/3) * 1.000000) = 1.066660
k3 = f(x0 + h/3, y0 + (h/6) * k1 + (h/6) * k2) = f(0.000000 + 0.100000/3, 1.000000 + (0.100000/6) * 1.000000 + (1.066660/6) * 1.067772) = 0.000000
k4 = f(x0 + h/2, y0 + (h/8) * k1 + (3*h/8) * k3) = f(0.000000 + 0.100000/2, 1.000000 + (0.100000/8) * 1.000000 + (3*0.100000/8) * 1.067772) = 1.102521
k5 = f(x0 + h, y0 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(0.000000 + 0.100000, 1.000000 + (0.100000/2) * 1.000000 - (3*0.100000/2) * 1.067772 + 2 * 0.100000 * 1.066660) = 1.203000
y1 = y0 + (h/6) * (k1 + 4*k4 + k5) = 1.000000 + (0.100000/6) * (1.000000 + 4*1.102521 + 1.203000) = 1.110218
x1 = x0 + h = 0.000000 + 0.100000 = 0.100000
Iteracion 2:
k1 = f(x1, y1) = f(0.100000, 1.110218) = 1.210051
k2 = f(x1 + h/3, y1 + (h/3) * k1) = f(0.100000 + 0.100000/3, 1.110218 + (0.100000/3) * 1.210051) = 1.283492
k3 = f(x1 + h/3, y1 + (h/6) * k1 + (h/6) * k2) = f(0.100000 + 0.100000/3, 1.110218 + (0.100000/6) * 1.210051 + (1.283492/6) * 1.284716) = 0.100000
k4 = f(x1 + h/2, y1 + (h/8) * k1 + (3*h/8) * k3) = f(0.100000 + 0.100000/2, 1.110218 + (0.100000/8) * 1.210051 + (3*0.100000/8) * 1.284716) = 1.322959
k5 = f(x1 + h, y1 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(0.100000 + 0.100000, 1.110218 + (0.100000/2) * 1.210051 - (3*0.100000/2) * 1.284716 + 2 * 0.100000 * 1.283492) = 1.433381
y2 = y1 + (h/6) * (k1 + 4*k4 + k5) = 1.110218 + (0.100000/6) * (1.210051 + 4*1.322959 + 1.433381) = 1.242472
x2 = x1 + h = 0.100000 + 0.100000 = 0.200000
Iteracion 3:
k1 = f(x2, y2) = f(0.200000, 1.242472) = 1.441142
k2 = f(x2 + h/3, y2 + (h/3) * k1) = f(0.200000 + 0.100000/3, 1.242472 + (0.100000/3) * 1.441142) = 1.521732
k3 = f(x2 + h/3, y2 + (h/6) * k1 + (h/6) * k2) = f(0.200000 + 0.100000/3, 1.242472 + (0.100000/6) * 1.441142 + (1.521732/6) * 1.523076) = 0.100000
k4 = f(x2 + h/2, y2 + (h/8) * k1 + (3*h/8) * k3) = f(0.200000 + 0.100000/2, 1.242472 + (0.100000/8) * 1.441142 + (3*0.100000/8) * 1.523076) = 1.565006
k5 = f(x2 + h, y2 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(0.200000 + 0.100000, 1.242472 + (0.100000/2) * 1.441142 - (3*0.100000/2) * 1.523076 + 2 * 0.100000 * 1.521732) = 1.685935
y3 = y2 + (h/6) * (k1 + 4*k4 + k5) = 1.242472 + (0.100000/6) * (1.441142 + 4*1.565006 + 1.685935) = 1.398924
x3 = x2 + h = 0.200000 + 0.100000 = 0.300000
Iteracion 4:
k1 = f(x3, y3) = f(0.300000, 1.398924) = 1.694444

k2 = f(x3 + h/3, y3 + (h/3) * k1) = f(0.300000 + 0.100000/3, 1.398924 + (0.100000/3) * 1.694444) = 1.782600
k3 = f(x3 + h/3, y3 + (h/6) * k1 + (h/6) * k2) = f(0.300000 + 0.100000/3, 1.398924 + (0.100000/6) * 1.694444 + (1.782600/6) * 1.784070) = 0.100000
k4 = f(x3 + h/2, y3 + (h/8) * k1 + (3*h/8) * k3) = f(0.300000 + 0.100000/2, 1.398924 + (0.100000/8) * 1.694444 + (3*0.100000/8) * 1.784070) = 1.829905
k5 = f(x3 + h, y3 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(0.300000 + 0.100000, 1.398924 + (0.100000/2) * 1.694444 - (3*0.100000/2) * 1.784070 + 2 * 0.100000 * 1.782600) = 1.961974
y4 = y3 + (h/6) * (k1 + 4*k4 + k5) = 1.398924 + (0.100000/6) * (1.694444 + 4*1.829905 + 1.961974) = 1.581858
x4 = x3 + h = 0.300000 + 0.100000 = 0.400000
Iteracion 5:
k1 = f(x4, y4) = f(0.400000, 1.581858) = 1.971276
k2 = f(x4 + h/3, y4 + (h/3) * k1) = f(0.400000 + 0.100000/3, 1.581858 + (0.100000/3) * 1.971276) = 2.067466
k3 = f(x4 + h/3, y4 + (h/6) * k1 + (h/6) * k2) = f(0.400000 + 0.100000/3, 1.581858 + (0.100000/6) * 1.971276 + (2.067466/6) * 2.069069) = 0.100000
k4 = f(x4 + h/2, y4 + (h/8) * k1 + (3*h/8) * k3) = f(0.400000 + 0.100000/2, 1.581858 + (0.100000/8) * 1.971276 + (3*0.100000/8) * 2.069069) = 2.119055
k5 = f(x4 + h, y4 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(0.400000 + 0.100000, 1.581858 + (0.100000/2) * 1.971276 - (3*0.100000/2) * 2.069069 + 2 * 0.100000 * 2.067466) = 2.262980
y5 = y4 + (h/6) * (k1 + 4*k4 + k5) = 1.581858 + (0.100000/6) * (1.971276 + 4*2.119055 + 2.262980) = 1.793699
x5 = x4 + h = 0.400000 + 0.100000 = 0.500000
Iteracion 6:
k1 = f(x5, y5) = f(0.500000, 1.793699) = 2.273125
k2 = f(x5 + h/3, y5 + (h/3) * k1) = f(0.500000 + 0.100000/3, 1.793699 + (0.100000/3) * 2.273125) = 2.377877
k3 = f(x5 + h/3, y5 + (h/6) * k1 + (h/6) * k2) = f(0.500000 + 0.100000/3, 1.793699 + (0.100000/6) * 2.273125 + (2.377877/6) * 2.379623) = 0.100000
k4 = f(x5 + h/2, y5 + (h/8) * k1 + (3*h/8) * k3) = f(0.500000 + 0.100000/2, 1.793699 + (0.100000/8) * 2.273125 + (3*0.100000/8) * 2.379623) = 2.434037
k5 = f(x5 + h, y5 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(0.500000 + 0.100000, 1.793699 + (0.100000/2) * 2.273125 - (3*0.100000/2) * 2.379623 + 2 * 0.100000 * 2.377877) = 2.590630
y6 = y5 + (h/6) * (k1 + 4*k4 + k5) = 1.793699 + (0.100000/6) * (2.273125 + 4*2.434037 + 2.590630) = 2.037031
x6 = x5 + h = 0.500000 + 0.100000 = 0.600000
Iteracion 7:
k1 = f(x6, y6) = f(0.600000, 2.037031) = 2.601674
k2 = f(x6 + h/3, y6 + (h/3) * k1) = f(0.600000 + 0.100000/3, 2.037031 + (0.100000/3) * 2.601674) = 2.715588
k3 = f(x6 + h/3, y6 + (h/6) * k1 + (h/6) * k2) = f(0.600000 + 0.100000/3, 2.037031 + (0.100000/6) * 2.601674 + (2.715588/6) * 2.717487) = 0.100000
k4 = f(x6 + h/2, y6 + (h/8) * k1 + (3*h/8) * k3) = f(0.600000 + 0.100000/2, 2.037031 + (0.100000/8) * 2.601674 + (3*0.100000/8) * 2.717487) = 2.776644
k5 = f(x6 + h, y6 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(0.600000 + 0.100000, 2.037031 + (0.100000/2) * 2.601674 - (3*0.100000/2) * 2.717487 + 2 * 0.100000 * 2.715588) = 2.946827

y7 = y6 + (h/6) * (k1 + 4*k4 + k5) = 2.037031 + (0.100000/6) * (2.601674 + 4*2.776644 + 2.946827) = 2.314616
x7 = x6 + h = 0.600000 + 0.100000 = 0.700000
Iteracion 8:
k1 = f(x7, y7) = f(0.700000, 2.314616) = 2.958833
k2 = f(x7 + h/3, y7 + (h/3) * k1) = f(0.700000 + 0.100000/3, 2.314616 + (0.100000/3) * 2.958833) = 3.082593
k3 = f(x7 + h/3, y7 + (h/6) * k1 + (h/6) * k2) = f(0.700000 + 0.100000/3, 2.314616 + (0.100000/6) * 2.958833 + (3.082593/6) * 3.084656) = 0.100000
k4 = f(x7 + h/2, y7 + (h/8) * k1 + (3*h/8) * k3) = f(0.700000 + 0.100000/2, 2.314616 + (0.100000/8) * 2.958833 + (3*0.100000/8) * 3.084656) = 3.148915
k5 = f(x7 + h, y7 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(0.700000 + 0.100000, 2.314616 + (0.100000/2) * 2.958833 - (3*0.100000/2) * 3.084656 + 2 * 0.100000 * 3.082593) = 3.333734
y8 = y7 + (h/6) * (k1 + 4*k4 + k5) = 2.314616 + (0.100000/6) * (2.958833 + 4*3.148915 + 3.333734) = 2.629419
x8 = x7 + h = 0.700000 + 0.100000 = 0.800000
Iteracion 9:
k1 = f(x8, y8) = f(0.800000, 2.629419) = 3.346776
k2 = f(x8 + h/3, y8 + (h/3) * k1) = f(0.800000 + 0.100000/3, 2.629419 + (0.100000/3) * 3.346776) = 3.481156
k3 = f(x8 + h/3, y8 + (h/6) * k1 + (h/6) * k2) = f(0.800000 + 0.100000/3, 2.629419 + (0.100000/6) * 3.346776 + (3.481156/6) * 3.483395) = 0.100000
k4 = f(x8 + h/2, y8 + (h/8) * k1 + (3*h/8) * k3) = f(0.800000 + 0.100000/2, 2.629419 + (0.100000/8) * 3.346776 + (3*0.100000/8) * 3.483395) = 3.553162
k5 = f(x8 + h, y8 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(0.800000 + 0.100000, 2.629419 + (0.100000/2) * 3.346776 - (3*0.100000/2) * 3.483395 + 2 * 0.100000 * 3.481156) = 3.753807
y9 = y8 + (h/6) * (k1 + 4*k4 + k5) = 2.629419 + (0.100000/6) * (3.346776 + 4*3.553162 + 3.753807) = 2.984640
x9 = x8 + h = 0.800000 + 0.100000 = 0.900000
Iteracion 10:
k1 = f(x9, y9) = f(0.900000, 2.984640) = 3.767967
k2 = f(x9 + h/3, y9 + (h/3) * k1) = f(0.900000 + 0.100000/3, 2.984640 + (0.100000/3) * 3.767967) = 3.913847
k3 = f(x9 + h/3, y9 + (h/6) * k1 + (h/6) * k2) = f(0.900000 + 0.100000/3, 2.984640 + (0.100000/6) * 3.767967 + (3.913847/6) * 3.916278) = 0.100000
k4 = f(x9 + h/2, y9 + (h/8) * k1 + (3*h/8) * k3) = f(0.900000 + 0.100000/2, 2.984640 + (0.100000/8) * 3.767967 + (3*0.100000/8) * 3.916278) = 3.992016
k5 = f(x9 + h, y9 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(0.900000 + 0.100000, 2.984640 + (0.100000/2) * 3.767967 - (3*0.100000/2) * 3.916278 + 2 * 0.100000 * 3.913847) = 4.209837
y10 = y9 + (h/6) * (k1 + 4*k4 + k5) = 2.984640 + (0.100000/6) * (3.767967 + 4*3.992016 + 4.209837) = 3.383738
x10 = x9 + h = 0.900000 + 0.100000 = 1.000000
Iteracion 11:
k1 = f(x10, y10) = f(1.000000, 3.383738) = 4.225209
k2 = f(x10 + h/3, y10 + (h/3) * k1) = f(1.000000 + 0.100000/3, 3.383738 + (0.100000/3) * 4.225209) = 4.383588
k3 = f(x10 + h/3, y10 + (h/6) * k1 + (h/6) * k2) = f(1.000000 + 0.100000/3, 3.383738 + (0.100000/6) * 4.225209 + (4.383588/6) * 4.386228) = 0.100000

k4 = f(x10 + h/2, y10 + (h/8) * k1 + (3*h/8) * k3) = f(1.000000 + 0.100000/2, 3.383738 + (0.100000/8) * 4.225209 + (3*0.100000/8) * 4.386228) = 4.468460
k5 = f(x10 + h, y10 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(1.000000 + 0.100000, 3.383738 + (0.100000/2) * 4.225209 - (3*0.100000/2) * 4.386228 + 2 * 0.100000 * 4.383588) = 4.704989
y11 = y10 + (h/6) * (k1 + 4*k4 + k5) = 3.383738 + (0.100000/6) * (4.225209 + 4*4.468460 + 4.704989) = 3.830472
x11 = x10 + h = 1.000000 + 0.100000 = 1.100000
Iteracion 12:
k1 = f(x11, y11) = f(1.100000, 3.830472) = 4.721679
k2 = f(x11 + h/3, y11 + (h/3) * k1) = f(1.100000 + 0.100000/3, 3.830472 + (0.100000/3) * 4.721679) = 4.893690
k3 = f(x11 + h/3, y11 + (h/6) * k1 + (h/6) * k2) = f(1.100000 + 0.100000/3, 3.830472 + (0.100000/6) * 4.721679 + (4.893690/6) * 4.896557) = 0.100000
k4 = f(x11 + h/2, y11 + (h/8) * k1 + (3*h/8) * k3) = f(1.100000 + 0.100000/2, 3.830472 + (0.100000/8) * 4.721679 + (3*0.100000/8) * 4.896557) = 4.985878
k5 = f(x11 + h, y11 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(1.100000 + 0.100000, 3.830472 + (0.100000/2) * 4.721679 - (3*0.100000/2) * 4.896557 + 2 * 0.100000 * 4.893690) = 5.242849
y12 = y11 + (h/6) * (k1 + 4*k4 + k5) = 3.830472 + (0.100000/6) * (4.721679 + 4*4.985878 + 5.242849) = 4.328939
x12 = x11 + h = 1.100000 + 0.100000 = 1.200000
Iteracion 13:
k1 = f(x12, y12) = f(1.200000, 4.328939) = 5.260978
k2 = f(x12 + h/3, y12 + (h/3) * k1) = f(1.200000 + 0.100000/3, 4.328939 + (0.100000/3) * 5.260978) = 5.447903
k3 = f(x12 + h/3, y12 + (h/6) * k1 + (h/6) * k2) = f(1.200000 + 0.100000/3, 4.328939 + (0.100000/6) * 5.260978 + (5.447903/6) * 5.451018) = 0.100000
k4 = f(x12 + h/2, y12 + (h/8) * k1 + (3*h/8) * k3) = f(1.200000 + 0.100000/2, 4.328939 + (0.100000/8) * 5.260978 + (3*0.100000/8) * 5.451018) = 5.548099
k5 = f(x12 + h, y12 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(1.200000 + 0.100000, 4.328939 + (0.100000/2) * 5.260978 - (3*0.100000/2) * 5.451018 + 2 * 0.100000 * 5.447903) = 5.827474
y13 = y12 + (h/6) * (k1 + 4*k4 + k5) = 4.328939 + (0.100000/6) * (5.260978 + 4*5.548099 + 5.827474) = 4.883620
x13 = x12 + h = 1.200000 + 0.100000 = 1.300000
Iteracion 14:
k1 = f(x13, y13) = f(1.300000, 4.883620) = 5.847178
k2 = f(x13 + h/3, y13 + (h/3) * k1) = f(1.300000 + 0.100000/3, 4.883620 + (0.100000/3) * 5.847178) = 6.050464
k3 = f(x13 + h/3, y13 + (h/6) * k1 + (h/6) * k2) = f(1.300000 + 0.100000/3, 4.883620 + (0.100000/6) * 5.847178 + (6.050464/6) * 6.053852) = 0.100000
k4 = f(x13 + h/2, y13 + (h/8) * k1 + (3*h/8) * k3) = f(1.300000 + 0.100000/2, 4.883620 + (0.100000/8) * 5.847178 + (3*0.100000/8) * 6.053852) = 6.159452
k5 = f(x13 + h, y13 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(1.300000 + 0.100000, 4.883620 + (0.100000/2) * 5.847178 - (3*0.100000/2) * 6.053852 + 2 * 0.100000 * 6.050464) = 6.463443

y14 = y13 + (h/6) * (k1 + 4*k4 + k5) = 4.883620 + (0.100000/6) * (5.847178 + 4*6.159452 + 6.463443) = 5.499427
x14 = x13 + h = 1.300000 + 0.100000 = 1.400000
Iteracion 15:
k1 = f(x14, y14) = f(1.400000, 5.499427) = 6.484877
k2 = f(x14 + h/3, y14 + (h/3) * k1) = f(1.400000 + 0.100000/3, 5.499427 + (0.100000/3) * 6.484877) = 6.706156
k3 = f(x14 + h/3, y14 + (h/6) * k1 + (h/6) * k2) = f(1.400000 + 0.100000/3, 5.499427 + (0.100000/6) * 6.484877 + (6.706156/6) * 6.709844) = 0.100000
k4 = f(x14 + h/2, y14 + (h/8) * k1 + (3*h/8) * k3) = f(1.400000 + 0.100000/2, 5.499427 + (0.100000/8) * 6.484877 + (3*0.100000/8) * 6.709844) = 6.824820
k5 = f(x14 + h, y14 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(1.400000 + 0.100000, 5.499427 + (0.100000/2) * 6.484877 - (3*0.100000/2) * 6.709844 + 2 * 0.100000 * 6.706156) = 7.155920
y15 = y14 + (h/6) * (k1 + 4*k4 + k5) = 5.499427 + (0.100000/6) * (6.484877 + 4*6.824820 + 7.155920) = 6.181762
x15 = x14 + h = 1.400000 + 0.100000 = 1.500000
Iteracion 16:
k1 = f(x15, y15) = f(1.500000, 6.181762) = 7.179257
k2 = f(x15 + h/3, y15 + (h/3) * k1) = f(1.500000 + 0.100000/3, 6.181762 + (0.100000/3) * 7.179257) = 7.420369
k3 = f(x15 + h/3, y15 + (h/6) * k1 + (h/6) * k2) = f(1.500000 + 0.100000/3, 6.181762 + (0.100000/6) * 7.179257 + (7.420369/6) * 7.424387) = 0.100000
k4 = f(x15 + h/2, y15 + (h/8) * k1 + (3*h/8) * k3) = f(1.500000 + 0.100000/2, 6.181762 + (0.100000/8) * 7.179257 + (3*0.100000/8) * 7.424387) = 7.549701
k5 = f(x15 + h, y15 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(1.500000 + 0.100000, 6.181762 + (0.100000/2) * 7.179257 - (3*0.100000/2) * 7.424387 + 2 * 0.100000 * 7.420369) = 7.910714
y16 = y15 + (h/6) * (k1 + 4*k4 + k5) = 6.181762 + (0.100000/6) * (7.179257 + 4*7.549701 + 7.910714) = 6.936574
x16 = x15 + h = 1.500000 + 0.100000 = 1.600000
Iteracion 17:
k1 = f(x16, y16) = f(1.600000, 6.936574) = 7.936148
k2 = f(x16 + h/3, y16 + (h/3) * k1) = f(1.600000 + 0.100000/3, 6.936574 + (0.100000/3) * 7.936148) = 8.199158
k3 = f(x16 + h/3, y16 + (h/6) * k1 + (h/6) * k2) = f(1.600000 + 0.100000/3, 6.936574 + (0.100000/6) * 7.936148 + (8.199158/6) * 8.203541) = 0.100000
k4 = f(x16 + h/2, y16 + (h/8) * k1 + (3*h/8) * k3) = f(1.600000 + 0.100000/2, 6.936574 + (0.100000/8) * 7.936148 + (3*0.100000/8) * 8.203541) = 8.340274
k5 = f(x16 + h, y16 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(1.600000 + 0.100000, 6.936574 + (0.100000/2) * 7.936148 - (3*0.100000/2) * 8.203541 + 2 * 0.100000 * 8.199158) = 8.734347
y17 = y16 + (h/6) * (k1 + 4*k4 + k5) = 6.936574 + (0.100000/6) * (7.936148 + 4*8.340274 + 8.734347) = 7.770434
x17 = x16 + h = 1.600000 + 0.100000 = 1.700000
Iteracion 18:
k1 = f(x17, y17) = f(1.700000, 7.770434) = 8.762099

k2 = f(x17 + h/3, y17 + (h/3) * k1) = f(1.700000 + 0.100000/3, 7.770434 + (0.100000/3) * 8.762099) = 9.049324
k3 = f(x17 + h/3, y17 + (h/6) * k1 + (h/6) * k2) = f(1.700000 + 0.100000/3, 7.770434 + (0.100000/6) * 8.762099 + (9.049324/6) * 9.054111) = 0.100000
k4 = f(x17 + h/2, y17 + (h/8) * k1 + (3*h/8) * k3) = f(1.700000 + 0.100000/2, 7.770434 + (0.100000/8) * 8.762099 + (3*0.100000/8) * 9.054111) = 9.203476
k5 = f(x17 + h, y17 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(1.700000 + 0.100000, 7.770434 + (0.100000/2) * 8.762099 - (3*0.100000/2) * 9.054111 + 2 * 0.100000 * 9.049324) = 9.634135
y18 = y17 + (h/6) * (k1 + 4*k4 + k5) = 7.770434 + (0.100000/6) * (8.762099 + 4*9.203476 + 9.634135) = 8.690603
x18 = x17 + h = 1.700000 + 0.100000 = 1.800000
Iteracion 19:
k1 = f(x18, y18) = f(1.800000, 8.690603) = 9.664451
k2 = f(x18 + h/3, y18 + (h/3) * k1) = f(1.800000 + 0.100000/3, 8.690603 + (0.100000/3) * 9.664451) = 9.978486
k3 = f(x18 + h/3, y18 + (h/6) * k1 + (h/6) * k2) = f(1.800000 + 0.100000/3, 8.690603 + (0.100000/6) * 9.664451 + (9.978486/6) * 9.983720) = 0.100000
k4 = f(x18 + h/2, y18 + (h/8) * k1 + (3*h/8) * k3) = f(1.800000 + 0.100000/2, 8.690603 + (0.100000/8) * 9.664451 + (3*0.100000/8) * 9.983720) = 10.147074
k5 = f(x18 + h, y18 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(1.800000 + 0.100000, 8.690603 + (0.100000/2) * 9.664451 - (3*0.100000/2) * 9.983720 + 2 * 0.100000 * 9.978486) = 10.618265
y19 = y18 + (h/6) * (k1 + 4*k4 + k5) = 8.690603 + (0.100000/6) * (9.664451 + 4*10.147074 + 10.618265) = 9.705120
x19 = x18 + h = 1.800000 + 0.100000 = 1.900000
Iteracion 20:
k1 = f(x19, y19) = f(1.900000, 9.705120) = 10.651420
k2 = f(x19 + h/3, y19 + (h/3) * k1) = f(1.900000 + 0.100000/3, 9.705120 + (0.100000/3) * 10.651420) = 10.995168
k3 = f(x19 + h/3, y19 + (h/6) * k1 + (h/6) * k2) = f(1.900000 + 0.100000/3, 9.705120 + (0.100000/6) * 10.651420 + (10.995168/6) * 11.000897) = 0.100000
k4 = f(x19 + h/2, y19 + (h/8) * k1 + (3*h/8) * k3) = f(1.900000 + 0.100000/2, 9.705120 + (0.100000/8) * 10.651420 + (3*0.100000/8) * 11.000897) = 11.179756
k5 = f(x19 + h, y19 + (h/2) * k1 - (3*h/2) * k3 + 2 * h * k2) = f(1.900000 + 0.100000, 9.705120 + (0.100000/2) * 10.651420 - (3*0.100000/2) * 11.000897 + 2 * 0.100000 * 10.995168) = 11.695888
y20 = y19 + (h/6) * (k1 + 4*k4 + k5) = 9.705120 + (0.100000/6) * (10.651420 + 4*11.179756 + 11.695888) = 10.822892
x20 = x19 + h = 1.900000 + 0.100000 = 2.000000
El valor de Y = 10.822892

Ejecución:

```
Algoritmo de Runge-Kutta-Merson(5to orden)
Ingresa el valor de X: 0
Ingresa el valor de XF: 2
Ingresa el valor de Y: 1
Ingresa el numero de iteraciones: 20
El valor de Y = 10.822892
Tiempo de ejecucion: 0.000000 segundos
```

Solorzano Galvez Gilberto Jesus

22760235

4to SA

Métodos Numéricos

Algoritmo de Runge-Kutta-Butcher

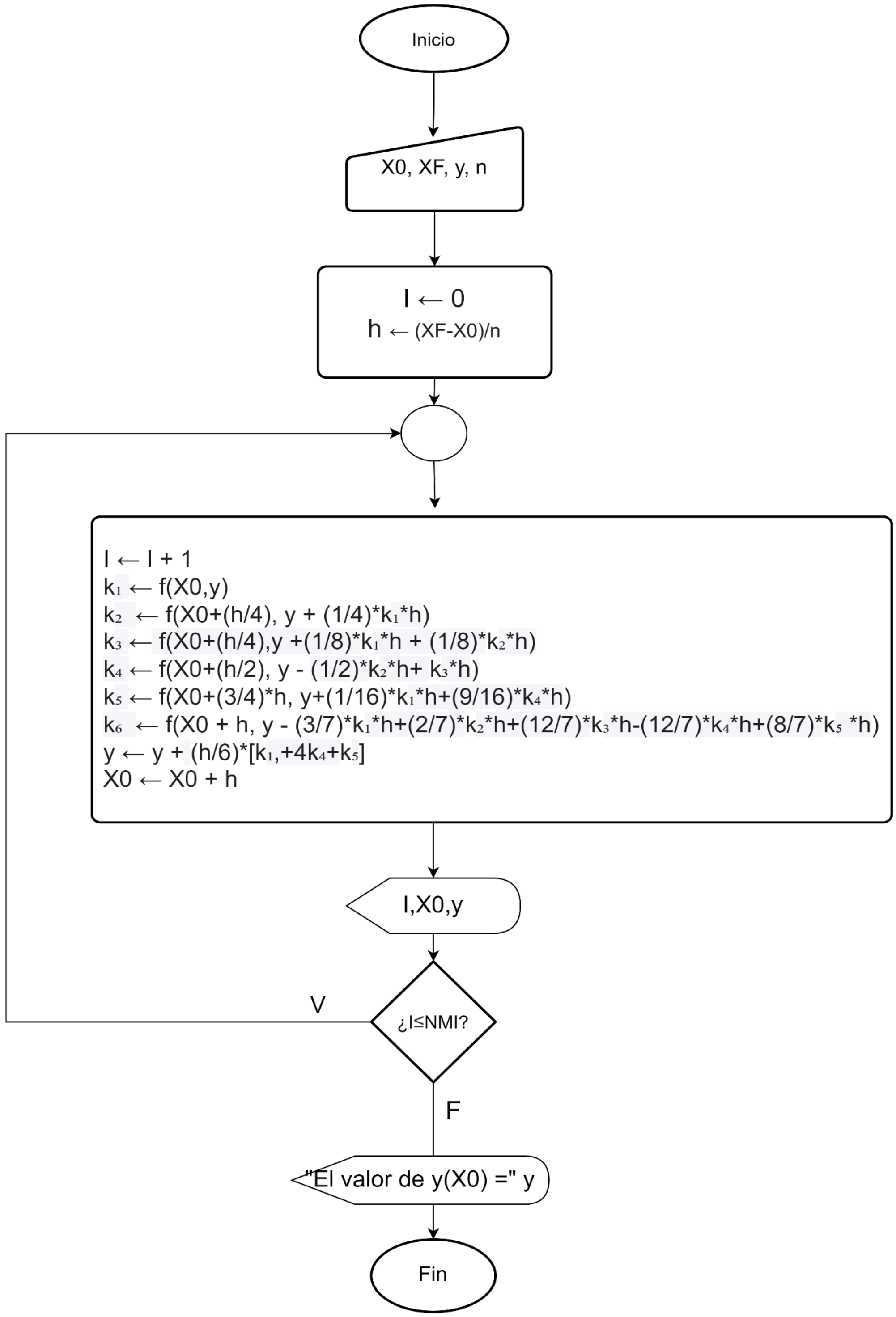
(sexto orden)

Maestro: Dr. Cecil Vidriales

Algoritmo de Runge-Kutta-Butcher de Sexto Orden

Descripción: Este método emplea una combinación más compleja de pendientes para alcanzar una precisión de sexto orden.

Utilidad: Se utiliza en problemas que requieren una precisión extremadamente alta en la solución de EDOs.



Pseudocódigo del Algoritmo de Runge-Kutta-Butcher (Sexto Orden)

1. Inicio
2. Inicializar Variables:
   - x = x0
   - y
   - xf
   - n
   -
   - NMI
   - h = (xf - x0) / n
3. Bucle For (i = 0; i < n; i++):
   - k1 = f(x, y)
   - k2 = f(x + h / 4, y + h / 4 * k1)
   - k3 = f(x + h / 4, y + h / 8 * k1 + h / 8 * k2)
   - k4 = f(x + h / 2, y - h / 2 * k2 + h * k3)
   - k5 = f(x + 3 * h / 4, y + 3 * h / 16 * k1 + 9 * h / 16 * k4)
   - k6 = f(x + h, y - 3 * h / 7 * k1 + 2 * h / 7 * k2 + 12 * h / 7 * k3 - 12 * h / 7 * k4 + 8 * h / 7 * k5)
   - y = y + h / 90 * (7 * k1 + 32 * k3 + 12 * k4 + 32 * k5 + 7 * k6)
   - x = x + h
4. Fin del Bucle For
5. Imprimir: i, x, y
6. Condición: Si i <= NMI entonces
   - Imprimir: i, x, y
7. Fin

Codigo:

```
double x0, xf, y, h, n;
   printf("Ingresa el valor de X: ");
   scanf("%lf", &x0);

   printf("Ingresa el valor de XF: ");
   scanf("%lf", &xf);

   printf("Ingresa el valor de Y: ");
   scanf("%lf", &y);

   printf("Ingresa el numero de iteraciones: ");
   scanf("%lf", &n);

   h = (xf - x0) / n;

   for (int i = 0; i < n; ++i) {
      double k1 = f(x0, y);
      double k2 = f(x0 + h/4, y + k1 * h / 4);
      double k3 = f(x0 + h/4, y + (1/8) * k1 * h + (1/8) * k2 * h);
      double k4 = f(x0 + h/2, y - (1/2) * k2 * h + k3 * h);
      double k5 = f(x0 + (3/4) * h, y + (1/16) * k1 * h + (9/16) * k4 * h);
```

```
        double k6 = f(x0 + h, y - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h +
(8/7) * k5 * h);
        y = y + (h / 90) * (7*k1 + 32*k3 +  12*k4 + 32*k5 + 7*k6);
        x0 = x0 + h;
    }

    printf("El valor de Y = %lf", y);
```

Prueba de escritorio
f(x,y)=sin(x) + y;
Ingresa el valor de X: 0
Ingresa el valor de XF: 4
Ingresa el valor de Y: 1
Ingresa el numero de iteraciones: 20
Iteracion 1:
k1 = f(x0, y0) = f(0.000000, 1.000000) = 1.000000
k2 = f(x0 + h/4, y0 + k1 * h / 4) = f(0.000000 + 0.200000/4, 1.000000 + 1.000000 * 0.200000 / 4) = 1.099979
k3 = f(x0 + h/4, y0 + (1/8) * k1 * h + (1/8) * k2 * h) = f(0.000000 + 0.200000/4, 1.000000 + (1/8) * 1.000000 * 0.200000 + (1/8) * 1.099979 * 0.200000) = 1.049979
k4 = f(x0 + h/2, y0 - (1/2) * k2 * h + k3 * h) = f(0.000000 + 0.200000/2, 1.000000 - (1/2) * 0.200000 * 1.099979 + 1.049979 * 1.309829) = 1.049979
k5 = f(x0 + (3/4) * h, y0 + (1/16) * k1 * h + (9/16) * k4 * h) = f(0.000000 + (3/4) * 0.200000, 1.000000 + (1/16) * 0.200000 * 1.000000 + (9/16) * 1.309829 * 1.000000) = 1.049979
k6 = f(x0 + h, y0 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(0.000000 + 0.200000, 1.000000 - (3/7) * 0.200000 * 1.000000 + (2/7) * 0.200000 * 1.099979 + (12/7) * 0.200000 * 1.049979 - (12/7) * 0.200000 * 1.309829 + (8/7) * 0.200000 * 1.000000) = 0.200000
y0 = y0 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 1.217209
El valor de Y despues de la iteracion 1 es: 1.217209

Iteracion 2:
k1 = f(x1, y1) = f(0.200000, 1.217209) = 1.415879
k2 = f(x1 + h/4, y1 + k1 * h / 4) = f(0.200000 + 0.200000/4, 1.217209 + 1.415879 * 0.200000 / 4) = 1.535407
k3 = f(x1 + h/4, y1 + (1/8) * k1 * h + (1/8) * k2 * h) = f(0.200000 + 0.200000/4, 1.217209 + (1/8) * 1.415879 * 0.200000 + (1/8) * 1.535407 * 0.200000) = 1.464613
k4 = f(x1 + h/2, y1 - (1/2) * k2 * h + k3 * h) = f(0.200000 + 0.200000/2, 1.217209 - (1/2) * 0.200000 * 1.535407 + 1.464613 * 1.805652) = 1.464613
k5 = f(x1 + (3/4) * h, y1 + (1/16) * k1 * h + (9/16) * k4 * h) = f(0.200000 + (3/4) * 0.200000, 1.217209 + (1/16) * 0.200000 * 1.415879 + (9/16) * 1.805652 * 1.415879) = 1.464613
k6 = f(x1 + h, y1 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(0.200000 + 0.200000, 1.217209 - (3/7) * 0.200000 * 1.415879 + (2/7) * 0.200000 * 1.535407 + (12/7) * 0.200000 * 1.464613 - (12/7) * 0.200000 * 1.805652 + (8/7) * 0.200000 * 1.415879) = 0.200000
y1 = y1 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 1.520556
El valor de Y despues de la iteracion 2 es: 1.520556

Iteracion 3:
k1 = f(x2, y2) = f(0.400000, 1.520556) = 1.909974
k2 = f(x2 + h/4, y2 + k1 * h / 4) = f(0.400000 + 0.200000/4, 1.520556 + 1.909974 * 0.200000 / 4) = 2.051020
k3 = f(x2 + h/4, y2 + (1/8) * k1 * h + (1/8) * k2 * h) = f(0.400000 + 0.200000/4, 1.520556 + (1/8) * 1.909974 * 0.200000 + (1/8) * 2.051020 * 0.200000) = 1.955521
k4 = f(x2 + h/2, y2 - (1/2) * k2 * h + k3 * h) = f(0.400000 + 0.200000/2, 1.520556 - (1/2) * 0.200000 * 2.051020 + 1.955521 * 2.391085) = 1.955521
k5 = f(x2 + (3/4) * h, y2 + (1/16) * k1 * h + (9/16) * k4 * h) = f(0.400000 + (3/4) * 0.200000, 1.520556 + (1/16) * 0.200000 * 1.909974 + (9/16) * 2.391085 * 1.909974) = 1.955521
k6 = f(x2 + h, y2 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(0.400000 + 0.200000, 1.520556 - (3/7) * 0.200000 * 1.909974 + (2/7) * 0.200000 * 2.051020 + (12/7) * 0.200000 * 1.955521 - (12/7) * 0.200000 * 2.391085 + (8/7) * 0.200000 * 1.909974) = 0.200000
y2 = y2 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 1.925932
El valor de Y despues de la iteracion 3 es: 1.925932

Iteracion 4:
k1 = f(x3, y3) = f(0.600000, 1.925932) = 2.490574
k2 = f(x3 + h/4, y3 + k1 * h / 4) = f(0.600000 + 0.200000/4, 1.925932 + 2.490574 * 0.200000 / 4) = 2.655647
k3 = f(x3 + h/4, y3 + (1/8) * k1 * h + (1/8) * k2 * h) = f(0.600000 + 0.200000/4, 1.925932 + (1/8) * 2.490574 * 0.200000 + (1/8) * 2.655647 * 0.200000) = 2.531118
k4 = f(x3 + h/2, y3 - (1/2) * k2 * h + k3 * h) = f(0.600000 + 0.200000/2, 1.925932 - (1/2) * 0.200000 * 2.655647 + 2.531118 * 3.076373) = 2.531118
k5 = f(x3 + (3/4) * h, y3 + (1/16) * k1 * h + (9/16) * k4 * h) = f(0.600000 + (3/4) * 0.200000, 1.925932 + (1/16) * 0.200000 * 2.490574 + (9/16) * 3.076373 * 2.490574) = 2.531118
k6 = f(x3 + h, y3 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(0.600000 + 0.200000, 1.925932 - (3/7) * 0.200000 * 2.490574 + (2/7) * 0.200000 * 2.655647 + (12/7) * 0.200000 * 2.531118 - (12/7) * 0.200000 * 3.076373 + (8/7) * 0.200000 * 2.490574) = 0.200000
y3 = y3 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 2.450979
El valor de Y despues de la iteracion 4 es: 2.450979

Iteracion 5:
k1 = f(x4, y4) = f(0.800000, 2.450979) = 3.168335
k2 = f(x4 + h/4, y4 + k1 * h / 4) = f(0.800000 + 0.200000/4, 2.450979 + 3.168335 * 0.200000 / 4) = 3.360676
k3 = f(x4 + h/4, y4 + (1/8) * k1 * h + (1/8) * k2 * h) = f(0.800000 + 0.200000/4, 2.450979 + (1/8) * 3.168335 * 0.200000 + (1/8) * 3.360676 * 0.200000) = 3.202259
k4 = f(x4 + h/2, y4 - (1/2) * k2 * h + k3 * h) = f(0.800000 + 0.200000/2, 2.450979 - (1/2) * 0.200000 * 3.360676 + 3.202259 * 3.874757) = 3.202259
k5 = f(x4 + (3/4) * h, y4 + (1/16) * k1 * h + (9/16) * k4 * h) = f(0.800000 + (3/4) * 0.200000, 2.450979 + (1/16) * 0.200000 * 3.168335 + (9/16) * 3.874757 * 3.168335) = 3.202259
k6 = f(x4 + h, y4 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(0.800000 + 0.200000, 2.450979 - (3/7) * 0.200000 * 3.168335 + (2/7) * 0.200000 * 3.360676 + (12/7) * 0.200000 * 3.202259 - (12/7) * 0.200000 * 3.874757 + (8/7) * 0.200000 * 3.168335) = 0.200000

y4 = y4 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 3.115592
El valor de Y despues de la iteracion 5 es: 3.115592

Iteracion 6:
k1 = f(x5, y5) = f(1.000000, 3.115592) = 3.957063
k2 = f(x5 + h/4, y5 + k1 * h / 4) = f(1.000000 + 0.200000/4, 3.115592 + 3.957063 * 0.200000 / 4) = 4.180868
k3 = f(x5 + h/4, y5 + (1/8) * k1 * h + (1/8) * k2 * h) = f(1.000000 + 0.200000/4, 3.115592 + (1/8) * 3.957063 * 0.200000 + (1/8) * 4.180868 * 0.200000) = 3.983015
k4 = f(x5 + h/2, y5 - (1/2) * k2 * h + k3 * h) = f(1.000000 + 0.200000/2, 3.115592 - (1/2) * 0.200000 * 4.180868 + 3.983015 * 4.803402) = 3.983015
k5 = f(x5 + (3/4) * h, y5 + (1/16) * k1 * h + (9/16) * k4 * h) = f(1.000000 + (3/4) * 0.200000, 3.115592 + (1/16) * 0.200000 * 3.957063 + (9/16) * 4.803402 * 3.957063) = 3.983015
k6 = f(x5 + h, y5 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(1.000000 + 0.200000, 3.115592 - (3/7) * 0.200000 * 3.957063 + (2/7) * 0.200000 * 4.180868 + (12/7) * 0.200000 * 3.983015 - (12/7) * 0.200000 * 4.803402 + (8/7) * 0.200000 * 3.957063) = 0.200000
y5 = y5 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 3.942586
El valor de Y despues de la iteracion 6 es: 3.942586

Iteracion 7:
k1 = f(x6, y6) = f(1.200000, 3.942586) = 4.874625
k2 = f(x6 + h/4, y6 + k1 * h / 4) = f(1.200000 + 0.200000/4, 3.942586 + 4.874625 * 0.200000 / 4) = 5.135302
k3 = f(x6 + h/4, y6 + (1/8) * k1 * h + (1/8) * k2 * h) = f(1.200000 + 0.200000/4, 3.942586 + (1/8) * 4.874625 * 0.200000 + (1/8) * 5.135302 * 0.200000) = 4.891571
k4 = f(x6 + h/2, y6 - (1/2) * k2 * h + k3 * h) = f(1.200000 + 0.200000/2, 3.942586 - (1/2) * 0.200000 * 5.135302 + 4.891571 * 5.884458) = 4.891571
k5 = f(x6 + (3/4) * h, y6 + (1/16) * k1 * h + (9/16) * k4 * h) = f(1.200000 + (3/4) * 0.200000, 3.942586 + (1/16) * 0.200000 * 4.874625 + (9/16) * 5.884458 * 4.874625) = 4.891571
k6 = f(x6 + h, y6 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(1.200000 + 0.200000, 3.942586 - (3/7) * 0.200000 * 4.874625 + (2/7) * 0.200000 * 5.135302 + (12/7) * 0.200000 * 4.891571 - (12/7) * 0.200000 * 5.884458 + (8/7) * 0.200000 * 4.874625) = 0.200000
y6 = y6 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 4.958552
El valor de Y despues de la iteracion 7 es: 4.958552

Iteracion 8:
k1 = f(x7, y7) = f(1.400000, 4.958552) = 5.944002
k2 = f(x7 + h/4, y7 + k1 * h / 4) = f(1.400000 + 0.200000/4, 4.958552 + 5.944002 * 0.200000 / 4) = 6.248465
k3 = f(x7 + h/4, y7 + (1/8) * k1 * h + (1/8) * k2 * h) = f(1.400000 + 0.200000/4, 4.958552 + (1/8) * 5.944002 * 0.200000 + (1/8) * 6.248465 * 0.200000) = 5.951265
k4 = f(x7 + h/2, y7 - (1/2) * k2 * h + k3 * h) = f(1.400000 + 0.200000/2, 4.958552 - (1/2) * 0.200000 * 6.248465 + 5.951265 * 7.146300) = 5.951265
k5 = f(x7 + (3/4) * h, y7 + (1/16) * k1 * h + (9/16) * k4 * h) = f(1.400000 + (3/4) * 0.200000, 4.958552 + (1/16) * 0.200000 * 5.944002 + (9/16) * 7.146300 * 5.944002) = 5.951265

k6 = f(x7 + h, y7 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(1.400000 + 0.200000, 4.958552 - (3/7) * 0.200000 * 5.944002 + (2/7) * 0.200000 * 6.248465 + (12/7) * 0.200000 * 5.951265 - (12/7) * 0.200000 * 7.146300 + (8/7) * 0.200000 * 5.944002) = 0.200000
y7 = y7 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 6.194925
El valor de Y despues de la iteracion 8 es: 6.194925

Iteracion 9:
k1 = f(x8, y8) = f(1.600000, 6.194925) = 7.194498
k2 = f(x8 + h/4, y8 + k1 * h / 4) = f(1.600000 + 0.200000/4, 6.194925 + 7.194498 * 0.200000 / 4) = 7.551515
k3 = f(x8 + h/4, y8 + (1/8) * k1 * h + (1/8) * k2 * h) = f(1.600000 + 0.200000/4, 6.194925 + (1/8) * 7.194498 * 0.200000 + (1/8) * 7.551515 * 0.200000) = 7.191790
k4 = f(x8 + h/2, y8 - (1/2) * k2 * h + k3 * h) = f(1.600000 + 0.200000/2, 6.194925 - (1/2) * 0.200000 * 7.551515 + 7.191790 * 8.624947) = 7.191790
k5 = f(x8 + (3/4) * h, y8 + (1/16) * k1 * h + (9/16) * k4 * h) = f(1.600000 + (3/4) * 0.200000, 6.194925 + (1/16) * 0.200000 * 7.194498 + (9/16) * 8.624947 * 7.194498) = 7.191790
k6 = f(x8 + h, y8 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(1.600000 + 0.200000, 6.194925 - (3/7) * 0.200000 * 7.194498 + (2/7) * 0.200000 * 7.551515 + (12/7) * 0.200000 * 7.191790 - (12/7) * 0.200000 * 8.624947 + (8/7) * 0.200000 * 7.194498) = 0.200000
y8 = y8 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 7.689301
El valor de Y despues de la iteracion 9 es: 7.689301

Iteracion 10:
k1 = f(x9, y9) = f(1.800000, 7.689301) = 8.663149
k2 = f(x9 + h/4, y9 + k1 * h / 4) = f(1.800000 + 0.200000/4, 7.689301 + 8.663149 * 0.200000 / 4) = 9.083734
k3 = f(x9 + h/4, y9 + (1/8) * k1 * h + (1/8) * k2 * h) = f(1.800000 + 0.200000/4, 7.689301 + (1/8) * 8.663149 * 0.200000 + (1/8) * 9.083734 * 0.200000) = 8.650576
k4 = f(x9 + h/2, y9 - (1/2) * k2 * h + k3 * h) = f(1.800000 + 0.200000/2, 7.689301 - (1/2) * 0.200000 * 9.083734 + 8.650576 * 10.365716) = 8.650576
k5 = f(x9 + (3/4) * h, y9 + (1/16) * k1 * h + (9/16) * k4 * h) = f(1.800000 + (3/4) * 0.200000, 7.689301 + (1/16) * 0.200000 * 8.663149 + (9/16) * 10.365716 * 8.663149) = 8.650576
k6 = f(x9 + h, y9 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(1.800000 + 0.200000, 7.689301 - (3/7) * 0.200000 * 8.663149 + (2/7) * 0.200000 * 9.083734 + (12/7) * 0.200000 * 8.650576 - (12/7) * 0.200000 * 10.365716 + (8/7) * 0.200000 * 8.663149) = 0.200000
y9 = y9 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 9.487050
El valor de Y despues de la iteracion 10 es: 9.487050

Iteracion 11:
k1 = f(x10, y10) = f(2.000000, 9.487050) = 10.396348
k2 = f(x10 + h/4, y10 + k1 * h / 4) = f(2.000000 + 0.200000/4, 9.487050 + 10.396348 * 0.200000 / 4) = 10.894230
k3 = f(x10 + h/4, y10 + (1/8) * k1 * h + (1/8) * k2 * h) = f(2.000000 + 0.200000/4, 9.487050 + (1/8) * 10.396348 * 0.200000 + (1/8) * 10.894230 * 0.200000) = 10.374413

k4 = f(x10 + h/2, y10 - (1/2) * k2 * h + k3 * h) = f(2.000000 + 0.200000/2, 9.487050 - (1/2) * 0.200000 * 10.894230 + 10.374413 * 12.425142) = 10.374413
k5 = f(x10 + (3/4) * h, y10 + (1/16) * k1 * h + (9/16) * k4 * h) = f(2.000000 + (3/4) * 0.200000, 9.487050 + (1/16) * 0.200000 * 10.396348 + (9/16) * 12.425142 * 10.396348) = 10.374413
k6 = f(x10 + h, y10 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(2.000000 + 0.200000, 9.487050 - (3/7) * 0.200000 * 10.396348 + (2/7) * 0.200000 * 10.894230 + (12/7) * 0.200000 * 10.374413 - (12/7) * 0.200000 * 12.425142 + (8/7) * 0.200000 * 10.396348) = 0.200000
y10 = y10 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 11.643258
El valor de Y despues de la iteracion 11 es: 11.643258

Iteracion 12:
k1 = f(x11, y11) = f(2.200000, 11.643258) = 12.451754
k2 = f(x11 + h/4, y11 + k1 * h / 4) = f(2.200000 + 0.200000/4, 11.643258 + 12.451754 * 0.200000 / 4) = 13.043918
k3 = f(x11 + h/4, y11 + (1/8) * k1 * h + (1/8) * k2 * h) = f(2.200000 + 0.200000/4, 11.643258 + (1/8) * 12.451754 * 0.200000 + (1/8) * 13.043918 * 0.200000) = 12.421331
k4 = f(x11 + h/2, y11 - (1/2) * k2 * h + k3 * h) = f(2.200000 + 0.200000/2, 11.643258 - (1/2) * 0.200000 * 13.043918 + 12.421331 * 14.873229) = 12.421331
k5 = f(x11 + (3/4) * h, y11 + (1/16) * k1 * h + (9/16) * k4 * h) = f(2.200000 + (3/4) * 0.200000, 11.643258 + (1/16) * 0.200000 * 12.451754 + (9/16) * 14.873229 * 12.451754) = 12.421331
k6 = f(x11 + h, y11 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(2.200000 + 0.200000, 11.643258 - (3/7) * 0.200000 * 12.451754 + (2/7) * 0.200000 * 13.043918 + (12/7) * 0.200000 * 12.421331 - (12/7) * 0.200000 * 14.873229 + (8/7) * 0.200000 * 12.451754) = 0.200000
y11 = y11 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 14.225059
El valor de Y despues de la iteracion 12 es: 14.225059

Iteracion 13:
k1 = f(x12, y12) = f(2.400000, 14.225059) = 14.900522
k2 = f(x12 + h/4, y12 + k1 * h / 4) = f(2.400000 + 0.200000/4, 14.225059 + 14.900522 * 0.200000 / 4) = 15.607850
k3 = f(x12 + h/4, y12 + (1/8) * k1 * h + (1/8) * k2 * h) = f(2.400000 + 0.200000/4, 14.225059 + (1/8) * 14.900522 * 0.200000 + (1/8) * 15.607850 * 0.200000) = 14.862824
k4 = f(x12 + h/2, y12 - (1/2) * k2 * h + k3 * h) = f(2.400000 + 0.200000/2, 14.225059 - (1/2) * 0.200000 * 15.607850 + 14.862824 * 17.796096) = 14.862824
k5 = f(x12 + (3/4) * h, y12 + (1/16) * k1 * h + (9/16) * k4 * h) = f(2.400000 + (3/4) * 0.200000, 14.225059 + (1/16) * 0.200000 * 14.900522 + (9/16) * 17.796096 * 14.900522) = 14.862824
k6 = f(x12 + h, y12 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(2.400000 + 0.200000, 14.225059 - (3/7) * 0.200000 * 14.900522 + (2/7) * 0.200000 * 15.607850 + (12/7) * 0.200000 * 14.862824 - (12/7) * 0.200000 * 17.796096 + (8/7) * 0.200000 * 14.900522) = 0.200000
y12 = y12 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 17.314441
El valor de Y despues de la iteracion 13 es: 17.314441

Iteracion 14:
k1 = f(x13, y13) = f(2.600000, 17.314441) = 17.829942

k2 = f(x13 + h/4, y13 + k1 * h / 4) = f(2.600000 + 0.200000/4, 17.314441 + 17.829942 * 0.200000 / 4) = 18.677969
k3 = f(x13 + h/4, y13 + (1/8) * k1 * h + (1/8) * k2 * h) = f(2.600000 + 0.200000/4, 17.314441 + (1/8) * 17.829942 * 0.200000 + (1/8) * 18.677969 * 0.200000) = 17.786471
k4 = f(x13 + h/2, y13 - (1/2) * k2 * h + k3 * h) = f(2.600000 + 0.200000/2, 17.314441 - (1/2) * 0.200000 * 18.677969 + 17.786471 * 21.299115) = 17.786471
k5 = f(x13 + (3/4) * h, y13 + (1/16) * k1 * h + (9/16) * k4 * h) = f(2.600000 + (3/4) * 0.200000, 17.314441 + (1/16) * 0.200000 * 17.829942 + (9/16) * 21.299115 * 17.829942) = 17.786471
k6 = f(x13 + h, y13 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(2.600000 + 0.200000, 17.314441 - (3/7) * 0.200000 * 17.829942 + (2/7) * 0.200000 * 18.677969 + (12/7) * 0.200000 * 17.786471 - (12/7) * 0.200000 * 21.299115 + (8/7) * 0.200000 * 17.829942) = 0.200000
y13 = y13 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 21.011584
El valor de Y despues de la iteracion 14 es: 21.011584

Iteracion 15:
k1 = f(x14, y14) = f(2.800000, 21.011584) = 21.346572
k2 = f(x14 + h/4, y14 + k1 * h / 4) = f(2.800000 + 0.200000/4, 21.011584 + 21.346572 * 0.200000 / 4) = 22.366391
k3 = f(x14 + h/4, y14 + (1/8) * k1 * h + (1/8) * k2 * h) = f(2.800000 + 0.200000/4, 21.011584 + (1/8) * 21.346572 * 0.200000 + (1/8) * 22.366391 * 0.200000) = 21.299062
k4 = f(x14 + h/2, y14 - (1/2) * k2 * h + k3 * h) = f(2.800000 + 0.200000/2, 21.011584 - (1/2) * 0.200000 * 22.366391 + 21.299062 * 25.510646) = 21.299062
k5 = f(x14 + (3/4) * h, y14 + (1/16) * k1 * h + (9/16) * k4 * h) = f(2.800000 + (3/4) * 0.200000, 21.011584 + (1/16) * 0.200000 * 21.346572 + (9/16) * 25.510646 * 21.346572) = 21.299062
k6 = f(x14 + h, y14 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(2.800000 + 0.200000, 21.011584 - (3/7) * 0.200000 * 21.346572 + (2/7) * 0.200000 * 22.366391 + (12/7) * 0.200000 * 21.299062 - (12/7) * 0.200000 * 25.510646 + (8/7) * 0.200000 * 21.346572) = 0.200000
y14 = y14 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 25.438855
El valor de Y despues de la iteracion 15 es: 25.438855

Iteracion 16:
k1 = f(x15, y15) = f(3.000000, 25.438855) = 25.579975
k2 = f(x15 + h/4, y15 + k1 * h / 4) = f(3.000000 + 0.200000/4, 25.438855 + 25.579975 * 0.200000 / 4) = 26.809319
k3 = f(x15 + h/4, y15 + (1/8) * k1 * h + (1/8) * k2 * h) = f(3.000000 + 0.200000/4, 25.438855 + (1/8) * 25.579975 * 0.200000 + (1/8) * 26.809319 * 0.200000) = 25.530320
k4 = f(x15 + h/2, y15 - (1/2) * k2 * h + k3 * h) = f(3.000000 + 0.200000/2, 25.438855 - (1/2) * 0.200000 * 26.809319 + 25.530320 * 30.586500) = 25.530320
k5 = f(x15 + (3/4) * h, y15 + (1/16) * k1 * h + (9/16) * k4 * h) = f(3.000000 + (3/4) * 0.200000, 25.438855 + (1/16) * 0.200000 * 25.579975 + (9/16) * 30.586500 * 25.579975) = 25.530320
k6 = f(x15 + h, y15 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(3.000000 + 0.200000, 25.438855 - (3/7) * 0.200000 * 25.579975 + (2/7) * 0.200000 * 26.809319 + (12/7) * 0.200000 * 25.530320 - (12/7) * 0.200000 * 30.586500 + (8/7) * 0.200000 * 25.579975) = 0.200000
y15 = y15 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 30.745575
El valor de Y despues de la iteracion 16 es: 30.745575

Iteracion 17:
k1 = f(x16, y16) = f(3.200000, 30.745575) = 30.687201
k2 = f(x16 + h/4, y16 + k1 * h / 4) = f(3.200000 + 0.200000/4, 30.745575 + 30.687201 * 0.200000 / 4) = 32.171740
k3 = f(x16 + h/4, y16 + (1/8) * k1 * h + (1/8) * k2 * h) = f(3.200000 + 0.200000/4, 30.745575 + (1/8) * 30.687201 * 0.200000 + (1/8) * 32.171740 * 0.200000) = 30.637380
k4 = f(x16 + h/2, y16 - (1/2) * k2 * h + k3 * h) = f(3.200000 + 0.200000/2, 30.745575 - (1/2) * 0.200000 * 32.171740 + 30.637380 * 36.715305) = 30.637380
k5 = f(x16 + (3/4) * h, y16 + (1/16) * k1 * h + (9/16) * k4 * h) = f(3.200000 + (3/4) * 0.200000, 30.745575 + (1/16) * 0.200000 * 30.687201 + (9/16) * 36.715305 * 30.687201) = 30.637380
k6 = f(x16 + h, y16 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(3.200000 + 0.200000, 30.745575 - (3/7) * 0.200000 * 30.687201 + (2/7) * 0.200000 * 32.171740 + (12/7) * 0.200000 * 30.637380 - (12/7) * 0.200000 * 36.715305 + (8/7) * 0.200000 * 30.687201) = 0.200000
y16 = y16 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 37.113717
El valor de Y despues de la iteracion 17 es: 37.113717

Iteracion 18:
k1 = f(x17, y17) = f(3.400000, 37.113717) = 36.858176
k2 = f(x17 + h/4, y17 + k1 * h / 4) = f(3.400000 + 0.200000/4, 37.113717 + 36.858176 * 0.200000 / 4) = 38.653084
k3 = f(x17 + h/4, y17 + (1/8) * k1 * h + (1/8) * k2 * h) = f(3.400000 + 0.200000/4, 37.113717 + (1/8) * 36.858176 * 0.200000 + (1/8) * 38.653084 * 0.200000) = 36.810175
k4 = f(x17 + h/2, y17 - (1/2) * k2 * h + k3 * h) = f(3.400000 + 0.200000/2, 37.113717 - (1/2) * 0.200000 * 38.653084 + 36.810175 * 44.124969) = 36.810175
k5 = f(x17 + (3/4) * h, y17 + (1/16) * k1 * h + (9/16) * k4 * h) = f(3.400000 + (3/4) * 0.200000, 37.113717 + (1/16) * 0.200000 * 36.858176 + (9/16) * 44.124969 * 36.858176) = 36.810175
k6 = f(x17 + h, y17 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(3.400000 + 0.200000, 37.113717 - (3/7) * 0.200000 * 36.858176 + (2/7) * 0.200000 * 38.653084 + (12/7) * 0.200000 * 36.810175 - (12/7) * 0.200000 * 44.124969 + (8/7) * 0.200000 * 36.858176) = 0.200000
y17 = y17 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 44.764724
El valor de Y despues de la iteracion 18 es: 44.764724

Iteracion 19:
k1 = f(x18, y18) = f(3.600000, 44.764724) = 44.322204
k2 = f(x18 + h/4, y18 + k1 * h / 4) = f(3.600000 + 0.200000/4, 44.764724 + 44.322204 * 0.200000 / 4) = 46.494048
k3 = f(x18 + h/4, y18 + (1/8) * k1 * h + (1/8) * k2 * h) = f(3.600000 + 0.200000/4, 44.764724 + (1/8) * 44.322204 * 0.200000 + (1/8) * 46.494048 * 0.200000) = 44.277937
k4 = f(x18 + h/2, y18 - (1/2) * k2 * h + k3 * h) = f(3.600000 + 0.200000/2, 44.764724 - (1/2) * 0.200000 * 46.494048 + 44.277937 * 53.090475) = 44.277937
k5 = f(x18 + (3/4) * h, y18 + (1/16) * k1 * h + (9/16) * k4 * h) = f(3.600000 + (3/4) * 0.200000, 44.764724 + (1/16) * 0.200000 * 44.322204 + (9/16) * 53.090475 * 44.322204) = 44.277937
k6 = f(x18 + h, y18 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(3.600000 + 0.200000, 44.764724 - (3/7) * 0.200000 * 44.322204 + (2/7) * 0.200000 *

46.494048 + (12/7) * 0.200000 * 44.277937 - (12/7) * 0.200000 * 53.090475 + (8/7) * 0.200000 * 44.322204) = 0.200000
y18 = y18 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 53.967678
El valor de Y despues de la iteracion 19 es: 53.967678

Iteracion 20:
k1 = f(x19, y19) = f(3.800000, 53.967678) = 53.355820
k2 = f(x19 + h/4, y19 + k1 * h / 4) = f(3.800000 + 0.200000/4, 53.967678 + 53.355820 * 0.200000 / 4) = 55.984844
k3 = f(x19 + h/4, y19 + (1/8) * k1 * h + (1/8) * k2 * h) = f(3.800000 + 0.200000/4, 53.967678 + (1/8) * 53.355820 * 0.200000 + (1/8) * 55.984844 * 0.200000) = 53.317053
k4 = f(x19 + h/2, y19 - (1/2) * k2 * h + k3 * h) = f(3.800000 + 0.200000/2, 53.967678 - (1/2) * 0.200000 * 55.984844 + 53.317053 * 63.943322) = 53.317053
k5 = f(x19 + (3/4) * h, y19 + (1/16) * k1 * h + (9/16) * k4 * h) = f(3.800000 + (3/4) * 0.200000, 53.967678 + (1/16) * 0.200000 * 53.355820 + (9/16) * 63.943322 * 53.355820) = 53.317053
k6 = f(x19 + h, y19 - (3/7) * k1 * h + (2/7) * k2 * h + (12/7) * k3 * h - (12/7) * k4 * h + (8/7) * k5 * h) = f(3.800000 + 0.200000, 53.967678 - (3/7) * 0.200000 * 53.355820 + (2/7) * 0.200000 * 55.984844 + (12/7) * 0.200000 * 53.317053 - (12/7) * 0.200000 * 63.943322 + (8/7) * 0.200000 * 53.355820) = 0.200000
y19 = y19 + (h / 90) * (7*k1 + 32*k3 + 12*k4 + 32*k5 + 7*k6) = 65.049100
El valor de Y despues de la iteracion 20 es: 65.049100

El valor de Y final es: 65.049100

Ejecución:

```
Algoritmo de Runge-Kutta-Butcher(6to orden)
Ingresa el valor de X: 0
Ingresa el valor de XF: 4
Ingresa el valor de Y: 1
Ingresa el numero de iteraciones: 20
El valor de Y = 65.049100
Tiempo de ejecucion: 0.000000 segundos
```