Práctica 1

IBERO
Ciudad de México • Tijuana

Predicción de respuesta dinámica de sistemas lineales e invariables en tiempo (LTI)

Integrantes del equipo

César Mauricio Arellano Velásquez
Raúl González Portillo

Profesor

César Arturo Ángeles Ruiz

Materia

Taller de Desarrollo de Aplicaciones

Introducción:

En matemática y computación, el **método de Euler**, llamado así en honor a Leonhard Euler, es un procedimiento de integración numérica para resolver ecuaciones diferenciales ordinarias (EDO) a partir de un valor inicial dado. El método de Euler es el más simple de los métodos numéricos para resolver un problema de valor inicial, y el más simple de los Métodos de Runge-Kutta. El método de Euler es nombrado por Leonhard Euler, quien lo trató en su libro *Institutionum calculi integralis* (publicado en 1768-1770).

El método de Euler es un método de primer orden, lo que significa que el error local es proporcional al cuadrado del tamaño del paso, y el error global es proporcional al tamaño del paso. El método de Euler regularmente sirve como base para construir métodos más complejos.

Consiste en dividir los intervalos que va de x0 a xf en n subintervalos de ancho h; o sea:

$$h = \frac{x_f - x_0}{n}$$

de manera que se obtiene un conjunto discreto de n + 1 puntos: x0, x1, x2, …, xn del intervalo de interés [x0, xf]. Para cualquiera de estos puntos se cumple que:

$$x_i = x_0 + ih \qquad 0 \leq i \leq n$$

La condición inicial $y(x_0) = y_0$, representa el punto $P_0 = (x_0, y_0)$ por donde pasa la curva solución de la ecuación del planteamiento inicial, la cual se denotará como $F(x) = y$ Ya teniendo el punto P_0 se puede evaluar la primera derivada de $F(x)$ en ese punto; por lo tanto:

$$F'(x) = \left.\frac{dy}{dx}\right|_{P_0} = f(x_0, y_0)$$

Se resuelve para y_1

$$y_1 = y_0 + (x_1 - x_0) f(x_0, y_0) = y_0 + h f(x_0, y_0)$$

Es evidente que la ordenada y_1 calculada de esta manera no es igual a $F(x_1)$ pues existe un pequeño error. Sin embargo, el valor y_1 sirve para que se aproxime $F'(x)$ en el punto $P = (x_1, y_1)$ y repetir el procedimiento anterior a fin de generar la sucesión de aproximaciones siguiente:

$$
\begin{aligned}
y_1 &= y_0 + hf(x_0, y_0) \\
y_2 &= y_1 + hf(x_1, y_1) \\
&\vdots \\
y_{i+1} &= y_i + hf(x_i, y_i) \\
&\vdots \\
y_n &= y_{n-1} + hf(x_{n-1}, y_{n-1})
\end{aligned}
$$

Objetivos:

- Comprender de manera básica el funcionamiento de algoritmos de predicción a través de métodos matemáticos para tener una mejor toma de decisiones.
- Determinar una correcta complejidad de tiempo y espacio para optimizar procesos y codificación.
- Mejorar habilidades de investigación para resolución de problemas.
- Comprender el método de Euler para obtener la curva solución para predecir el comportamiento de una función.

Análisis

Pseudocódigo:

#define return = retorna el valor de alguna operación o variable.
#define fopen = Abre el archivo (de texto o binario) que se le indica y junto al modo de ejecución.
#define fclose = Cierra el archivo que se le indique y que previamente esté abierto.
#define fprintf = Imprime en el archivo especificado.
#define wt = Abre el archivo en modo de escritura.
#define sscanf -> Formatear texto de una cadena.
#define free -> Liberar Memoria.

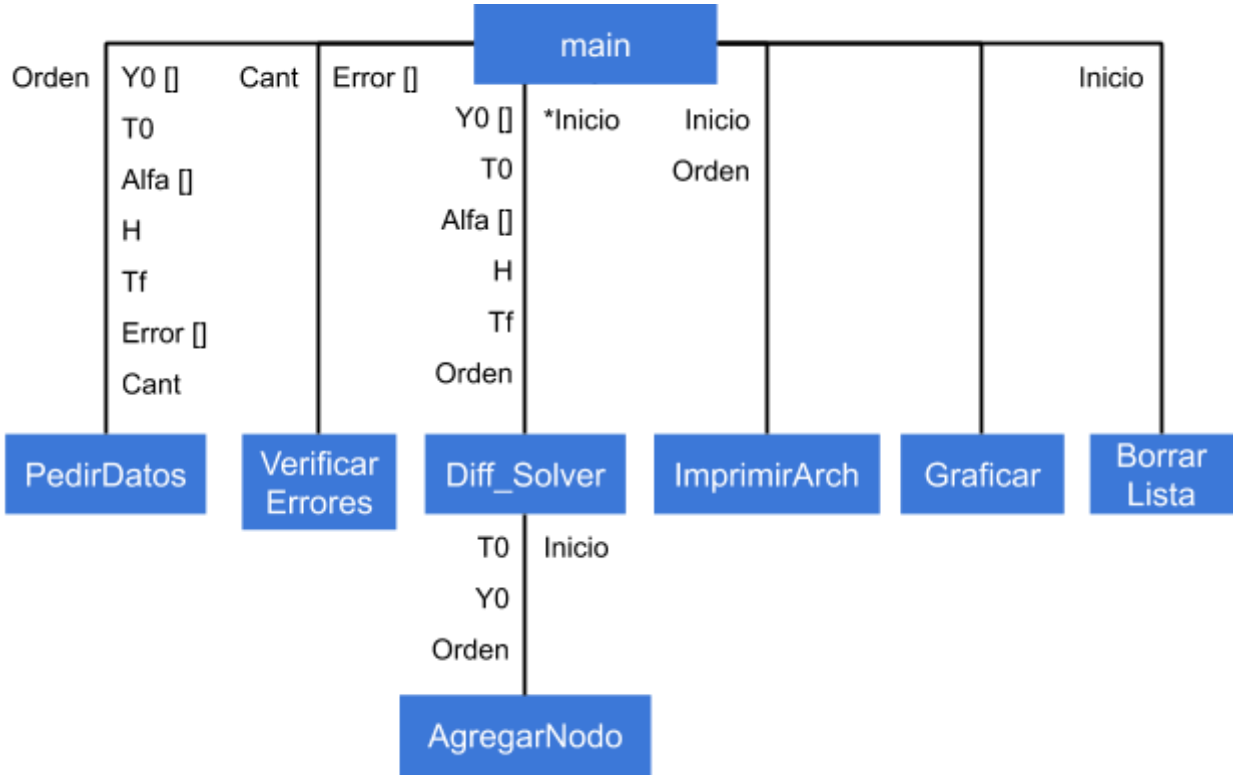
```
PedirDatos( | T0, Y0, Alfa, H, Tf, Orden, Error, Cant);
VerificarErrores(Error, Cant | );
Funcion(T0, Y0 | );
DiffSolver(T0, Y0, Alfa, H, Tf, int Orden | ↑ ↑Inicio);
AgregarNodo(T0, Y0, Orden | ↑ ↑Inicio);
ImprimirArch(↑Inicio, Orden | );
Graficar(|);
BorrarLista(↑Inicio | );
```

```
Principal (argc, ↑argv | )
{
  Error [0] = argc + 1;
  VerificarErrores(Error, Cant | );
  sscanf (argv[1], Orden);
  ↑Inicio = NULL;
  PedirDatos( | T0, Y0, Alfa, H, Tf, Orden, Error, Cant);
  VerificarErrores(Error, Cant | );
  DiffSolver(T0, Y0, Alfa, H, Tf, int Orden | ↑ ↑Inicio);
  ImprimirArch(Inicio,Orden);
  Graficar();
  BorrarLista( ↑Inicio | );
  return 0;
}

PedirDatos( | T0, Y0, Alfa, H, Tf, Orden, Error, Cant)
{
  j = 3;
  Imprimir("Introduzca los siguientes datos:”);
  Imprimir("T0: ");
  Error[0] = Leer(T0);
  Imprimir("Avance (h):");
  Error[1] = Leer(H);
  Imprimir("Tf:");
  Error[2] = Leer(Tf);
  desde i = 0; hasta i < Orden; i++
      {
      Imprimir("y^(T0):);
      Error[j] = Leer(Y0[i]);
      j++;
      Imprimir("α:");
      Error[j] = Leer(Alfa[i]);
      j++;
      }
  Cant = j-1;
}

VerificarErrores(Error, Cant | )
{
   desde i = 0; hasta i <= Orden; i++
      {
      Si(Error[i] = 0)
    {
      Imprimir("Error en uno o más datos ingresados");
      exit(1);
    }
      Si(Error[i] = 2 OR Error[i] >= 4)
    {
      Imprimir ("Ingresó una cantidad de argumentos incorrecta");
      exit (2);
    }
      }
```

```
}

Funcion(T0, Y0 | );
{
  return T0 - (3*Y0);
}

DiffSolver(T0, Y0, Alfa, H, Tf, int Orden | ↑ ↑Inicio)
{
  AgregarNodo(T0, Y0, Orden | ↑ ↑Inicio);
  desde T0 += H; Hasta T0 <= Tf; T0 += H
      {
      Acum=0;
      j = Orden - 1;
      desde i = 0; Hasta i < Orden; i++
       {
      Acum += -Y0[i]*Alfa[j];
      Si(Orden <> i+1)
      U[i] = Y0[i+1] * H;
      Si no
      {
      U[i] = Acum + H*(Funcion(T0-H, Y0[i - 1]));
      AgregarNodo(T0, U, Orden | ↑ ↑Inicio);
      }
      Y0[i] = U[i];
      j--;
       }
      }
}

AgregarNodo(T0, Y0, Orden | ↑ ↑Inicio);
{
  new(Temp);
  desde i = 0; Hasta i < Orden; i++
       ↑Temp.Y[i] = Y0[i];
   ↑Temp.T = T0;
   ↑Temp.Sig = NULL;
  Si ( ↑Inicio <> NULL)
      {
      Temp2 =  ↑Inicio;
      mientras ( ↑Temp2.Sig <> NULL)
            Temp2 =  ↑Temp2.Sig;
       ↑Temp2.Sig = Temp;
      }
  Si no
       ↑Inicio = Temp;
}

ImprimirArch( ↑Inicio, Orden | );
```

```
{
  Archivo = fopen("LTI.txt","wt");
  Temp = Inicio;
  mientras (Temp <> NULL)
      {
      fprintf(Archivo, ↑Temp.T);
      Desde i=0; Hasta i<(Orden-1);i++
            fprintf(Archivo, ↑Temp.Y[i]);
      fprintf(Archivo, ↑Temp.Y[i]);
      Temp =  ↑Temp.Sig;
      }
  fclose(Archivo);
}

Graficar()
{
   ↑AbrirGnuPlot[] = {"set title \"Método de Predición / Euler\"",
               "set ylabel \"----Y--->\"",
               "set xlabel \"----T--->\"",
               "plot \"LTI.txt\" with lines"
  };
  FILE *VentanaGnuPlot = popen ("gnuplot -persist", "w");
  Desde i=0; Hasta i<4; i++
      fprintf(VentanaGnuPlot, AbrirGnuPlot[i]);
}
BorrarLista( ↑Inicio | );
{
  mientras (Inicio <> NULL)
      {
      Temp = Inicio;
      Inicio =  ↑Inicio.Sig;
      free (Temp);
      }
}
```

Diagrama IPO



Entradas Procesos y Salidas

Entradas

Nombre	Descripción
Orden	Indica el de orden de la ecuación diferencial.
Y0[ ]	Guarda las evaluaciones de las distintas funciones Y en T0 (Y(T0), Y'(T0), etc.)
T0	Guarda el valor inicial T0
h	Guarda el valor del step / incremento que dará la función por cada iteración
Tf	Indica el Tiempo donde terminará la gráfica
Alfa[ ]	Contiene los diferentes coeficientes de la ecuación. Depende del orden.

Procesos

Nombre	Descripción
PedirDatos	Solicita los datos necesarios para realizar la predicción
VerificarErrores	Verifica si hay errores en los datos de entrada.
Diff_Solver	Realiza una predicción en base a los datos del usuario utilizando el método de Euler y la guarda en dos arreglos los cálculos de Y para posteriormente guardarlos en una lista dinámica.
AgregarNodo	Esta función crea espacio en memoria para guardar los datos de T y Y y lo agrega al final de la lista dinámica que empieza por la variable *Inicio.
ImprimirArch	Imprime los datos de los arreglos TipoCoordenada ->Y[ ] y TipoCoordenada ->T[ ] en un formato interpretable por GNUPlot
Graficar	Inicializa GNUPlot con los títulos de la gráfica y los ejes y le envía el archivo creado para graficarlo.
BorrarLista	Libera la memoria creada por la lista dinámica.

Salidas

Nombre	Descripción
Cant	Una variable que guarda la cantidad de valores que necesitamos verificar en busca de errores de lectura.
U[ ]	Un arreglo que guarda los resultados a graficar en el eje Y
*Inicio	Un apuntador que guarda la posición del primer elemento de la lista dinámica que guarda los resultados generados por Diff_Solver.
Error[ ]	A través del retorno de scanf, este guardará los errores que haya al momento de la lectura de datos.

Código

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

typedef struct def_Coordenada
{
  double Y[10],Alfa[10],T;
  struct def_Coordenada *Sig;
} TipoCoordenada;

typedef enum defErrores
  {
   NoHayError, Lectura, Argumento
  } Errores;

void PedirDatos(double T0[], double Y0[], double *Alfa, double *H, double *Tf,
int Orden, Errores Error[],int *Cant);
void VerificarErrores(Errores Error[],int Cant);
double Funcion(double T0, double Y0);
void DiffSolver(TipoCoordenada **Inicio, double T0, double Y0[], double Alfa[],
double H , double Tf, int Orden);
void AgregarNodo(TipoCoordenada **Inicio, double T0, double Y0[],int Orden);
void ImprimirArch(TipoCoordenada *Inicio, int Orden);
void Graficar();
void Cargando(char Mensaje[]);
void BorrarLista(TipoCoordenada *Inicio);

int main (int argc, char *argv[])
{
  int Orden,Cant;
  double Y0[100],Alfa[100],T0,H,Tf;
  Errores Error[100];
  Error [0] = argc + 1;
  VerificarErrores(Error, 0);
  sscanf (argv[1], "%d", &Orden);
  TipoCoordenada *Inicio = NULL;
  PedirDatos(&T0,Y0,Alfa,&H,&Tf,Orden,Error,&Cant);
  VerificarErrores(Error,Cant);
  Cargando("Resolviendo ecuación");
  DiffSolver(&Inicio,T0,Y0,Alfa,H,Tf,Orden);
  Cargando("Imprimiendo datos en archivo");
  ImprimirArch(Inicio,Orden);
  Cargando("Graficando con GNUPlot");
  Graficar();
  BorrarLista (Inicio);
  return 0;
}
```

```
void PedirDatos(double T0[], double Y0[], double *Alfa, double *H, double *Tf,
int Orden, Errores Error[],int *Cant)
{
  int i, j = 3;
  printf("Introduzca los siguientes datos:\n\n");
  printf("T0: \n");
  Error[0] = scanf(" %lf",T0);
  printf("Avance (h): \n");
  Error[1] = scanf(" %lf",H);
  printf("Tf: \n");
  Error[2] = scanf(" %lf",Tf);
  for (i = 0; i < Orden; i++)
      {
      printf("y^[%d](T0) = Y^[%d]0: \n", i, i);
      Error[j] = scanf(" %lf", &Y0[i]);
      j++;
      printf("α[%d]:\n", i);
      Error[j] = scanf(" %lf", &Alfa[i]);
      j++;
      }
  *Cant = j-1;
}

void VerificarErrores(Errores Error[],int Cant)
{
  for(int i=0; i<=Cant;i++)
      {
      if(Error[i] ==0)
    {
      printf("Error en uno o más datos ingresados\n");
      exit(1);
    }
      if(Error[i] == 2 || Error[i] >= 4)
    {
      printf ("Ingresó una cantidad de argumentos incorrecta\n");
      exit (2);
    }
      }
}

double Funcion(double T0, double Y0)
{
  return T0 - (3*Y0);
}

void DiffSolver(TipoCoordenada **Inicio, double T0, double Y0[], double Alfa[],
double H , double Tf, int Orden)
{
  double U[100],Acum;
  int i,j;
  AgregarNodo(Inicio,T0,Y0,Orden);
  for(T0 += H; T0 <= Tf; T0 += H)
      {
```

```
      Acum=0;
      j = Orden - 1;
      for(i = 0; i < Orden; i++)
       {
      Acum += -Y0[i]*Alfa[j];
      if(Orden != i+1)
      U[i] = Y0[i+1] * H;
      else
      {
      U[i] = Acum + H*(Funcion(T0-H, Y0[i - 1]));
      AgregarNodo(Inicio,T0,U,Orden);
      }
      Y0[i] = U[i];
      j--;
       }
      }
}

void AgregarNodo(TipoCoordenada **Inicio, double T0, double Y0[],int Orden)
{
  TipoCoordenada *Temp, *Temp2;
  Temp = (TipoCoordenada *) malloc (sizeof (TipoCoordenada));
  for (int i=0;i<Orden;i++)
      Temp -> Y[i] = Y0[i];
  Temp -> T = T0;
  Temp -> Sig = NULL;
  if (*Inicio != NULL)
      {
      Temp2 = *Inicio;
      while (Temp2 -> Sig != NULL)
    Temp2 = Temp2 -> Sig;
      Temp2 -> Sig = Temp;
      }
  else
      *Inicio = Temp;
}

void ImprimirArch(TipoCoordenada *Inicio, int Orden)
{
  int i;
  FILE *Archivo;
  Archivo = fopen("LTI.txt","wt");
  TipoCoordenada *Temp;
  Temp = Inicio;
  while (Temp != NULL)
      {
      fprintf(Archivo,"%f ",Temp -> T);
      for(i=0;i<(Orden-1);i++)
    fprintf(Archivo,"%f ", Temp -> Y[i]);
      fprintf(Archivo,"%f\n",Temp -> Y[i]);
      Temp = Temp -> Sig;
      }
  fclose(Archivo);
```

```
}

void Graficar()
{
  int i;
  char *AbrirGnuPlot[] = {"set title \"Método de Predición / Euler\"",
              "set ylabel \"----Y--->\"",
              "set xlabel \"----T--->\"",
              "plot \"LTI.txt\" with lines"
  };
  FILE *VentanaGnuPlot = popen ("gnuplot -persist", "w");
  for (i=0; i<4; i++)
      fprintf(VentanaGnuPlot, "%s \n", AbrirGnuPlot[i]);
}

void Cargando (char Mensaje[])
{
  system ("clear");
  puts (Mensaje);
  printf ("\n");
  system ("sleep 0.15");
  system ("clear");
  puts (Mensaje);
  printf (".\n");
  system ("sleep 0.15");
  system ("clear");
  puts (Mensaje);
  printf ("..\n");
  system ("sleep 0.15");
  system ("clear");
  puts (Mensaje);
  printf ("...\n");
  system ("sleep 0.15");
}

void BorrarLista (TipoCoordenada *Inicio)
{
  TipoCoordenada *Temp;
  while (Inicio != NULL)
      {
      Temp = Inicio;
      Inicio = Inicio -> Sig;
      free (Temp);
      }
}
```

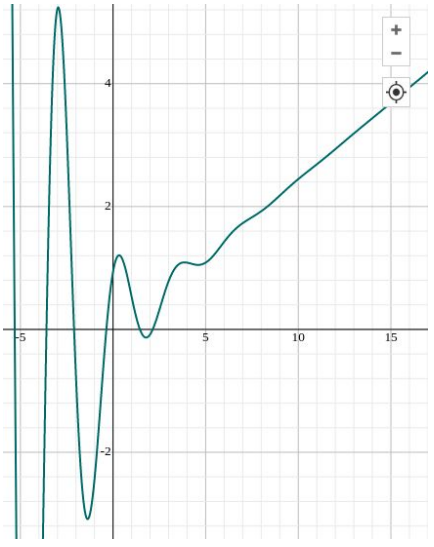
Ejecución del programa

```
cesar@cesar: ~/Cursos/TallerDeApps/Prácticas/Práctica1/Orden Su...
Graficando con GNUPlot
...
cesar@cesar:~/Cursos/TallerDeApps/Prácticas/Práctica1/Orden Superior Entrega Fin
al$ ./Practica1.exe 2
Introduzca los siguientes datos:

T0:
-3
Avance (h):
0.1
Tf:
1000
y^[0](T0) = Y^[0]0:
5
a[0]:
1
y^[1](T0) = Y^[1]0:
2
a[1]:
4
```

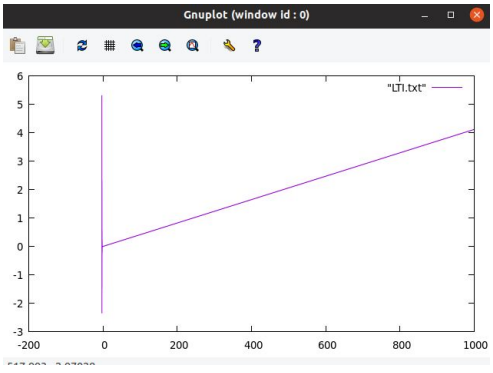
Demostración a través de gráficas.

Función: $y'' + y' + y = x - 3y$

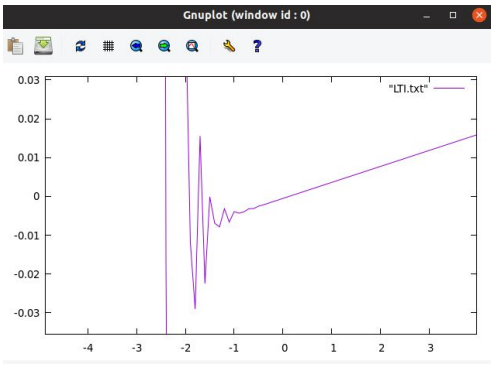
Solución real



Escala Normal



Escala Ampliada



Conclusión:

En esta práctica aprendimos cómo funciona el método de Euler para realizar predicciones de respuestas dinámicas de sistemas lineales. Para implementarlo en un programa hicimos uso de conocimientos previos de C como Operadores, Ciclos, Argumentos, tanto en la función principal como en el resto de funciones, Arreglos, Listas Dinámicas y Manipulación de Archivos. El mayor reto para nosotros fue el entender el método de Euler en un principio, ya que la solución ante la problemática era un poco abrumadora, por el hecho de no tener un conocimiento previo del tema, esto hace que nosotros como programadores nos veamos forzados a preguntar e investigar más allá de la información proporcionada, a inferir y basarnos en nuestra lógica de tal modo que se encuentre la forma de adaptar aquellos requerimientos a un programa.

Recomendaciones:

Al hacer un programa de complejidad elevada, el procurar la modularidad del mismo es uno de los aspectos más importantes. Esto permite hacer modificaciones mayores al programa con menor esfuerzo además de que permite identificar problemas de una manera más rápida.
Para entender el problema, si este lo permite (en nuestro caso, el método de Euler se podía resolver en diferentes órdenes, los primeros más fáciles de entender que los posteriores) puede ayudar bastante el resolver el problema de forma ascendente, es decir, empezando por los casos más fáciles, ya que esto permite tener un mayor entendimiento del problema antes de empezar a trabajar en los aspectos más complejos del mismo.

Nota: Para ejecutar el programa, es necesario indicar el orden de la ecuación diferencial.

Ejemplo.
```
$./Practica1.exe 2
```