Taller de Desarrollo de Aplicaciones
PRÁCTICA No. 1

Alan Flores Quezada
Allan Jair Escamilla Hernández

Abstract- Este documento presenta la resolución de un programa que resolverá ecuaciones diferenciales ordinarias (EDO) de N orden mediante el uso del método numérico de Euler.

I. INTRODUCCIÓN

A lo largo del desarrollo de este documento vamos presentar una solución computacional para aproximar la resolución de Ecuaciones Diferenciales Ordinarias a través del método de Euler. Además, se le dará solución a ecuaciones de orden n dado por el usuario, esto por el hecho de que el mundo real varios de los procesos de ciencias e ingenierías no pueden ser modelados mediante ecuaciones lineales, pero tomando en cuenta la variación en el tiempo, se puede aproximar los modelos acerca de lo que realmente sucede y anticipar datos .

II. ANÁLISIS

Para la resolución de este problema vamos a combinar nuestros conocimientos de programación en el lenguaje c junto a conocimientos de cálculo aplicado para resolver ecuaciones diferenciales ordinarias.

Pseudocódigo

```
Principal( | ){
  imprimir("Ingresar x0: ");
  leer(x0);
  imprimir("Ingresar y0: ");
  leer(y0);
  imprimir("Ingresar el salto
(h): ");
  leer(h);
  imprimir("Ingresar la
cantidad de pasos (n): ");
  leer(n);
  ↑Inicio = NULL;
  euler(x0, y0, h, n | Inicio);
  guardarEnArchivo(Inicio);
  len = longitud(Inicio);
  graficarSolucion(len);
}
f(x, y | result){
  result = x + y;
}

euler(x0, y0, h, n | Inicio){
  xaux = 0;
  yaux = 0;
  agregarNodo(x0, yo, Inicio);
  ant = Inicio;
  next = (*Inicio)->sig;
  Para i = 0 hasta i = n{
      yaux = ant->y +
h*f(ant->x, ant->y);
      xaux = ant->x + h;
      agregarNodo(xaux, yaux,
Inicio);
  }
```

```
}
agregarNodo(x, y, Inicio){
   ↑n = Inicio;
   ↑temp = new Nodo;
   ↑temp.x = x;
   ↑temp.y = y;
   ↑temp.sig = NULL;
  Si( ↑Inicio == NULL)
     ↑Inicio = temp;
  sino{
     ↑temp2 = Inicio;
    mientras( ↑temp2.sig !=
NULL)
       temp2 = temp2.sig;
    temp2.sig = temp;
  }
}

guardarEnArchivo(Inicio){
  temp = Inicio;
  fp =
abrirArchivo("solucion.dat",
"wt");
  mientras(temp != NULL){
    escribirEnArchivo(temp.x,
temp.y);
    temp = temp.sig;
  }
  cerrarArchivo(fp);
}

graficarSolucion(len){
  commandsForGnuplot[] = {"set
title \"Solucion de la ecuacion
diferencial\"", "plot
'solucion.dat'"};
  gnuplotPipe = abrirArchivo
("gnuplot -persistent", "wt");
  Para i=0 hasta i < len{

imprimirEnArchivo(gnuplotPipe,
"%s \n",
commandsForGnuplot[i]); //Send
commands to gnuplot one by one.
  }
}

longitud(Inicio){
  temp = Inicio;
  len = 0;
  mientras(temp != NULL){
    len++;
    temp = temp->sig;
  }
  regresa len;
}
```

III. DISEÑO

El programa recibirà los argumentos dependiendo del orden de la ecuación diferencial, esto por las condiciones iniciales necesarias para la correcta ejecuciòn del programa.

Una vez que el usuario haya ingresado los datos correspondientes, El programa se ejecutarà e imprimirá los resultados en un archivo de texto, el cual serà utilizado para graficar la solución usando gnuplot.

IV. ALCANCES Y LIMITACIONES

Es indispensable utilizar buenas prácticas de codificación y documentación

Se debe programar modularmente.

En una primer parte, el grado de la ecuación diferencial n, será máximo de orden 3, sin embargo, el código debe ser independiente a la estructura de dato utilizado para generar la ecuación diferencial.

Se deben usar archivos de texto.

Se deben dar argumentos desde la línea de comandos.
Se deben mostrar elementos para la prueba y verificación del código mediante ecuaciones diferenciales ordinarias. eg. : $y' + 2y = 2 - e{-}4t\ y(0) = 1$

V. VERSIONES PREVIAS DEL CÓDIGO

Aquí se encuentra la versión previa a la definitiva de este programa.
Código Previo

Aquí se encuentra la versiòn final del Còdigo.

VI. VERSIÓN FINAL DEL CÓDIGO

```c
/*
* @author:  Allan Jair Escamilla Hernández, Alan Flores Quezada
* @date:    11/septiembre/2019
* @file:    code.c
*/
#include <stdio.h> // Incluyendo bibliotecas
#include <stdlib.h>
#include <math.h>


// Definimos una variable enumerable para los errores
typedef enum defErrores{
  TODOBIEN, INGRESO_CADENA, INGRESO_CARCTER_INCORRECTO
}Errores;


// PROTOTIPOS DE FUNCIONES
float f(float x, float y);
void euler(float x0, float y0, int n, float h);
float* generarArreglo(int cant);
float** generarMatriz(int ancho, int alto);
void pedirValoresIniciales(float* valoresInicialesx, float* valoresInicialesy, int orden);
void valoresUT(float* ut, int orden);
void liberarMemoria(float* arr);
void liberarMemoriaMat(float** matriz, int alto);
void llenarMatriz(float** Matriz, int orden, float coeficientes[]);
void resolverEcuacion(float** Matriz, float* valoresInicialesy, float* valoresInicialesx, float* uT, float h, int n, int orden);
float* multiplicarMatrices(float** Matriz1, float* Matriz2, int orden);
float* matrizporEscalar(float* ut, float h, float x, float y, int orden);
void pedirCoeficientes(float* coeficientes, int orden);
float* sumarVectores(float* v1, float* v2, int orden);
void verificarErrores(Errores error[], int cantidad);
void plot();


// FUNCION PRINCIPAL
int main(){
  Errores error[5];
  int n, orden;
  float h, x0, y0, *valoresNuevos, *valoresInicialesy, *valoresInicialesx, *uT, **Matriz, *coeficientes;
  printf("Ingresar el orden de la EDO-> ");
  error[0] = scanf("%d", &orden);
  printf("Ingresar tamaño de salto (h)-> ");
  error[1] = scanf("%f", &h);
  printf("Ingresar numero de pasos
```

```
(n)-> ");
  error[2] = scanf("%d", &n);
  if(orden == 1){ // Orden 1
    printf("Ingresar x0-> ");
    error[3] = scanf("%f", &x0);
    printf("Ingresar y0-> ");
    error[4] = scanf("%f", &y0);
    verificarErrores(error, 5);
    printf("Espere un momento
mientras se calcula la solucion...
\n");
    euler(x0, y0, n, h);
    printf("Ejecucion terminada...
\n");
  }else{ // Orden superior
    valoresInicialesx =
generarArreglo(orden);
    valoresInicialesy =
generarArreglo(orden);
    coeficientes =
generarArreglo(orden);
    uT = generarArreglo(orden);
    Matriz = generarMatriz(orden,
orden);

pedirValoresIniciales(valoresInici
alesx, valoresInicialesy, orden);

pedirCoeficientes(coeficientes,
orden);
    valoresUT(uT, orden);
    llenarMatriz(Matriz, orden,
coeficientes);
    resolverEcuacion(Matriz,
valoresInicialesy,
valoresInicialesx, uT, h, n,
orden);

liberarMemoria(valoresInicialesx);

liberarMemoria(valoresInicialesy);
    liberarMemoria(uT);
    liberarMemoriaMat(Matriz,
orden);
  }
  plot(); // Plot del resultado
obtenido
  return 0;
}

// DESARROLLANDO LAS FUNCIONES

/*  * Función de ecuacion
diferencial
    * @param float x. Parametro x
para evaluar en la funcion
    * @param float y. Parametro y
para evaluar en la funcion
*/
float f(float x, float y){
  return cos(x);
}



/*  * Funcion que resuelve la
ecuacion diferencial por el metodo
de euler
    * @param float x0. Valor
inicial de x.
    * @param float y0. Valor
inicial de y.
    * @param int n. Numero de
pasos.
    * @param float h. Tamaño del
salto.
*/
void euler(float x0, float y0, int
n, float h){
  FILE* fp = fopen("solucion.dat",
"wt");
  float xaux, yaux, anty= y0,
antx= x0;
  fprintf(fp, "%f, %f\n", antx,
anty);
  for(int i = 0; i < n; i++){
    yaux = anty + h*f(antx, anty);
    xaux = antx + h;
    antx = xaux;
    anty = yaux;
```

```
    fprintf(fp, "%f, %f\n", xaux,
yaux);
  }
  fclose(fp);
}


//  plot
exp(x)*(-x*exp(-x)-exp(-x)+2),
"solucion.dat" with lines

/*  * Funcion que genera un
arreglo en tiempo de ejecucion
    * @param int cant. Define el
tamaño del arreglo.

*/
float* generarArreglo(int cant){
  return
(float*)malloc(cant*sizeof(float))
;
}

/*  * Funcion que genera una
matriz en tiempo de ejecucion
    * @param int ancho. Determina
el ancho de la matriz.
    * @param int alto. Determina
el alto de la matriz.

*/
float** generarMatriz(int ancho,
int alto){
  float **Matriz =
(float**)malloc(alto*sizeof(float*
)); // Columnas
    for(int i = 0; i < alto; i++)
      Matriz[i] =
(float*)malloc(ancho*sizeof(float)
); // Filas
  return Matriz;
}

/*  * Funcion que le pide al
usuario los valores iniciales para
la EDO de orden superior
    * @param float*
valoresInicialesx. Es un arreglo
que almacenará los valores
iniciales de x.
    * @param float*
valoresInicialesy. Es un arreglo
que almacenará los valores
iniciales de y.
    * @param int orden. Define el
orden de la EDO.

*/
void pedirValoresIniciales(float*
valoresInicialesx, float*
valoresInicialesy, int orden){
  Errores er[orden*2];
  int cont = 0;
  for(int i = 0; i < orden; i++){
    printf("Ingresar x%d inicial->
", i);
    er[cont] = scanf("%f",
&valoresInicialesx[i]);
    cont++;
    printf("Ingresar y%d inicial->
", i);
    er[cont] = scanf("%f",
&valoresInicialesy[i]);
    cont++;
  }
  verificarErrores(er, orden*2);
}

/*  * Funcion que llena un arreglo
con los valores que requerimos.
    * @param float* ut. Arreglo
donde se almacenaran valores
    * @param int orden. Orden del
arreglo.

*/
void valoresUT(float* ut, int
orden){
  for(int i = 0; i < orden; i++){
    if(i == orden-1)
```

```
      ut[i] = 1;
    else
      ut[i] = 0;
  }
}

/*  * Funcion que resuelve la
ecuacion diferencial dada.
    * @param float** Matriz.
Matriz para realizar operaciones.
    * @param float*
valoresInicialesy. Vector para los
valores iniciales de y.
    * @param float*
valoresInicialesx. Vector para los
valores iniciales de x.
    * @param float* Ut. Vector que
define valores para la ecuacion
vectorial que se va a resolver.
    * @param float h. Tamaño del
salto.
    * @param int n. Numero de
saltos.
    * @param int orden. Orden de
la EDO.

*/
void resolverEcuacion(float**
Matriz, float* valoresInicialesy,
float* valoresInicialesx, float*
uT, float h, int n, int orden){
  FILE* fp = fopen("solucion.dat",
"wt");
  fprintf(fp, "%f, %f\n",
valoresInicialesx[orden-1],
valoresInicialesy[orden-1]);
  for(int i = 0; i < n; i++){
    valoresInicialesy =
sumarVectores(multiplicarMatrices(
Matriz, valoresInicialesy, orden),
matrizporEscalar(uT, h,
valoresInicialesx[orden-1],
valoresInicialesy[orden-1],
orden), orden);
    for(int j = 0; j < orden;
j++){
      valoresInicialesx[j] =
valoresInicialesx[j] + h;
    }
    fprintf(fp, "%f, %f\n",
valoresInicialesx[orden-1],
valoresInicialesy[orden-1]);
  }
  fclose(fp);
}

/*  * Funcion que multiplica
matrices.
    * @param float** Matriz1.
Matriz 1 a operar
    * @param float* Matriz2.
Matriz 2 a operar
    * @param int orden. Orden de
las matrices.
*/
float* multiplicarMatrices(float**
Matriz1, float* Matriz2, int
orden){
  float contador = 0;
  float* resultado =
generarArreglo(orden);
  for(int i = 0; i < orden; i++){
    contador = 0;
    for(int j = 0; j < orden;
j++){
      contador+= Matriz1[i][j] *
Matriz2[j];
    }
    resultado[i] = contador;
  }
  return resultado;
}
/*  * Funcion que verifica que no
haya errores en las entradas del
usuario.
    * @param Errores error[].
Arreglo de los codigos de error.
    * @param int cantidad.
Cantidad de elementos del arreglo.
```

```
*/
void verificarErrores(Errores
error[], int cantidad){
  int err = 0;
  for(int i = 0; i < cantidad;
i++){
    if(error[i] == 0)
      err = 1;
  }
  if(err){
    printf("\nHa ocurrido el error
con codigo: %d\n", error[err]);
    exit(0);
  }
}

/*  * Funcion que multiplica una
matriz por un escalar.
    * @param float* ut. Matriz 1
para operar.
    * @param float h. Tamaño del
salto.
    * @param float x. Valores de
x.
    * @param float y. Valores de
y.
    * @param int orden. Orden de
la EDO.
*/
float* matrizporEscalar(float* ut,
float h, float x, float y, int
orden){
  float *resultados =
generarArreglo(orden);
  for(int i = 0; i < orden; i++){
    resultados[i] = ut[i]*h*f(x,
y);
  }
  return resultados;
}


/*  * Funcion que suma vectores.
    * @param float* v1. Arreglo 1
para operar.
    * @param float* v2. Arreglo 2
para operar.
    * @param int orden. Orden de
la EDO.
*/
float* sumarVectores(float* v1,
float* v2, int orden){
  float* resultado =
generarArreglo(orden);
  for(int i = 0; i < orden; i++){
    resultado[i] = v1[i] + v2[i];
  }
  return resultado;
}

/*  * Funcion que le pide al
usuario los coeficientes de la EDO
    * @param float* coeficientes.
Arreglo donde se almacenaran los
coeficientes.
    * @param int orden. Orden de
la EDO.
*/
void pedirCoeficientes(float*
coeficientes, int orden){
  Errores er[orden];
  for(int i = 0; i < orden; i++){
    printf("Ingresar el
coeficiente a%d-> ", i);
    er[i] = scanf("%f",
&coeficientes[i]);
  }
  verificarErrores(er, orden);
}

/*  * Funcion que llena una matriz
con valores requeridos.
    * @param float** Matriz.
Matriz que se va a rellenar.
    * @param int orden. Orden de
la EDO.
    * @param floar coeficientes[].
Arreglo de los coeficientes de la
EDO a resolver.
*/
```

```
void llenarMatriz(float** Matriz,
int orden, float coeficientes[]){
  for(int i = 0; i < orden - 1;
i++){
    for(int j = 0; j < orden;
j++){
      Matriz[i][j] = 0;
    }
  }
  for(int i = 0; i < orden; i++)
    Matriz[i][i+1] = 1;
  for(int i = 0; i < orden; i++){
    Matriz[orden-1][i] =
coeficientes[i];
  }
}

/*  * Funcion que libera la
memoria de una asignacion en
tiempo de ejecucion.
    * @param float* arr. Arreglo
que se va a liberar el espacio en
memoria.
*/
void liberarMemoria(float* arr){
  free(arr);
}


/*  * Funcion que libera la
memoria de una asignacion en
tiempo de ejecucion.
    * @param float** matriz.
Matriz que se va a liberar el
espacio en memoria.
*/
void liberarMemoriaMat(float**
matriz, int alto){
  for(int i = 0; i < alto; i++)
    free(matriz[i]);// Filas
  free(matriz); // Columnas
}

/* Función que gráfica el archivo
de texto generado
*/
void plot(){
  char * configGnuplot[] = {"set
title \"Solucion a la ecuación
diferencial\"",

"set ylabel \"Y\"",

"set xlabel \"X\"",

"plot \"solucion.dat\" with
lines",

"set autoscale",

"replot"
                                };
  FILE * ventanaGnuplot = popen
("gnuplot -persist", "w");
  for (int i=0; i < 4; i++){
    fprintf(ventanaGnuplot, "%s
\n", configGnuplot[i]);
  }
}
```

VII. CONCLUSIONES

Este programa es capaz de predecir el comportamiento de determinados fenómenos, usando como parámetros datos que conocemos de antemano, es decir que puede calcular la tendencia/el futuro de ese fenómeno usando su pasado.

Esto modelando la información que se tiene mediante el método de Euler.