

Práctica 1. “Método Predictivo en Tiempo”

Rodrigo García Díaz

Jesús Enrique Domínguez Barrios

Los avances en Inteligencia Artificial nos han permitido, a lo largo de los años, encontrar relaciones y patrones en aquello que está ahí, pero es difícil de ver a simple vista; así es como hemos logrado conocer el porqué y cómo de muchos de los fenómenos que plagan nuestro mundo. Uno de esos procesos basados en AI son los sistemas que permiten lograr una inferencia, o predicción, sobre algo que sucede en nuestro entorno. ¿Qué es aquello que, acompañadas de la observación, nos permiten conocer y moldear nuestra realidad? Exacto, las matemáticas. El método de predicción que se analizará en esta ocasión es el “Método de Euler” (método de inferencia por diferencias).

Este método nos permite conocer tiempos futuros mediante un estado actual y el conocimiento de ciertas fuerzas y eventos que actúan sobre un fenómeno y así estudiarlo con detalle. Es usado en bolsa de valores, control y para el análisis de propagación de enfermedades, etc. Gracias a estos procesos se logra facilitar la toma de decisiones sobre el futuro o influenciar comportamientos de fenómenos conocidos. La ecuación que describe este método es la siguiente:

$$y(t + \Delta t) = \Delta t f(y, t) + y(t). \text{ Método de Euler}$$

Para esta primera práctica, se quiere conseguir un sistema que, dependiendo de los valores de unas variables capaces de ser modificadas por el usuario, prediga una situación en el futuro la cual determine si, al ser jalados por una fuerza externa dentro una variante de un *bongee*, una persona amarrada a un resorte impactará o no con el suelo usando como método de predicción un “Método de Inferencia por Diferencias”, más específicamente el método de Euler. El sistema debe avisar el momento en el que la persona colgada impacta el suelo (si es que lo hace), así como generar un archivo con los tiempos y sus respectivas posiciones para, de esta forma, lograr graficar y observar el comportamiento del *bongee* bajo unas condiciones iniciales determinadas.

Especificaciones del sistema.

1. El sistema debe ser programado bajo buenas prácticas de codificación y documentación.
2. El sistema deberá estar diseñado de forma modular. Implementando archivos “.h” y “.c”.
3. El sistema deberá estar diseñado bajo la arquitectura MVC (Modelo-Vista-Controlador) la cual busca separar los datos de la lógica del sistema.
4. Se debe implementar, en la parte de la vista, un menú que presente al usuario la posibilidad de manipular las variables que éste puede alterar, como son la masa **m** de la persona, la constante **k** del resorte (*bongee*) y el Δt , que corresponde al tiempo entre una muestra y otra.
5. Con los datos *seteados* por el usuario y nuestras constantes conocidas el sistema deberá ser capaz, mediante la implementación del método de Euler, de predecir lo que pasará con ese fenómeno del salto del *bongee*.
6. Se implementará un sistema de “variables temporales” que ayudarán al manejo de los tiempos “pasados” para inferir el “futuro”.
7. Los datos, una vez terminada la predicción del sistema, deberán ser almacenados y un archivo CSV para ser graficados con ayuda de GNUplot.
8. Se pedirá el tiempo a predecir y la cantidad de muestras que se desean tomar, de esta forma se calculará Δt con la división de .

- Módulos.

- Modelo

§ El primer módulo del modelo es el que se encargará de realizar el cálculo del método de Euler mediante la implementación de una función secuencial. Recibirá todas las variables para su funcionamiento y retornará una matriz con los datos que se imprimirán luego en el archivo.

§ El tercer módulo del modelo se encargará mediante un pipe en generar los comandos correspondientes para la impresión de la gráfica de la predicción con GNUplot.

§ El tercer módulo de modelo será el encargado de recibir la matriz e imprimirla en un archivo CSV.

○ Vista.

§ El primer módulo de la vista será el menú que dará la bienvenida al usuario, explicará el funcionamiento del sistema y pedirá los valores con los que se quiera trabajar, las mandará al controlador.

§ El segundo módulo de la vista será en donde se despliegan los resultados de la predicción, como el tiempo que se predijo en el futuro, la cantidad de muestras y de si se chocó o no.

○ Controlador.

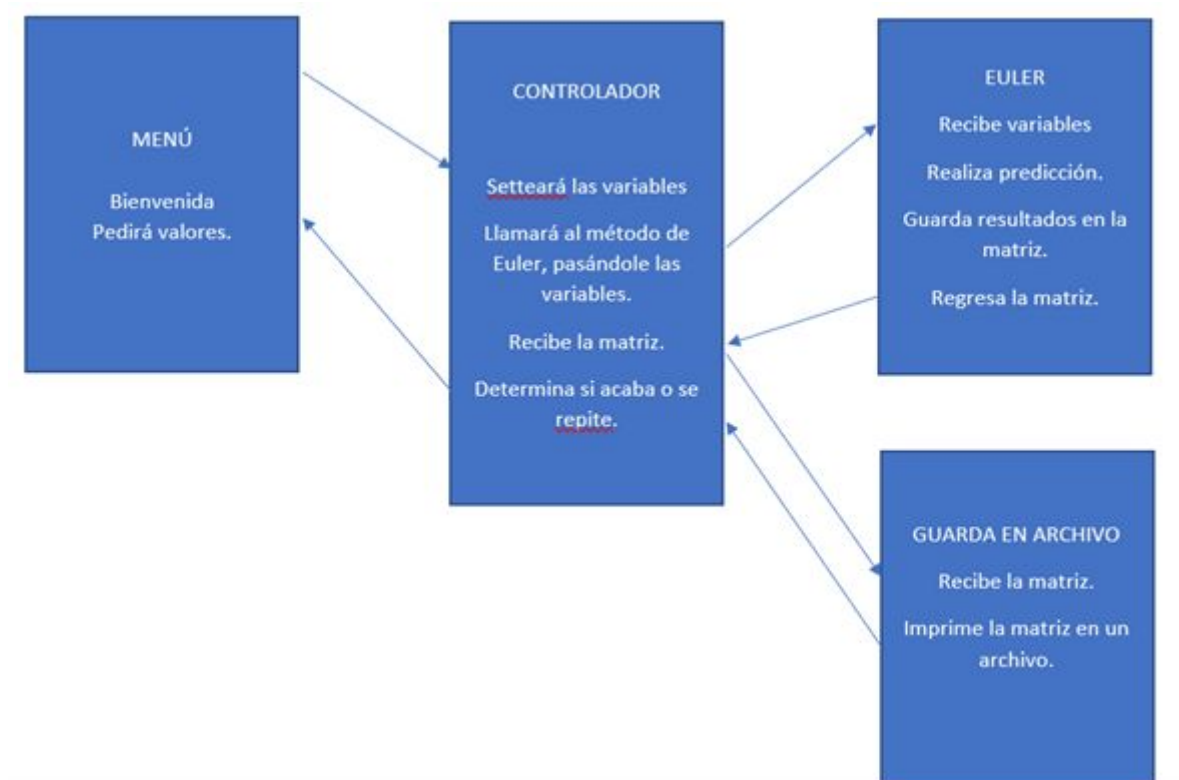
§ Un único módulo encargado de varias cosas:

- *Setteará* las variables obtenidas de la vista.
- Llamará al método de Euler para comenzar la predicción.
- Recibirá la matriz creada por el módulo del método de Euler y la enviará (llamando) a la función que la imprime en el archivo.
- Llamará a la función que imprime la gráfica con GNUplot.

Además, este controlador determinará cuando el proceso haya terminado y preguntará si se quiere repetir la predicción, en el caso de que sí, se enviará al menú principal con un estado “1” que ocasionará una iteración completa, en el caso de que no se desee repetir, se devolverá un estado “0” al menú, el sistema, después de una cordial despedida, finalizará.

El método de Euler nos permite conocer tiempos futuros partiendo de un estado actual junto a sus entradas en el modelo que se estudia. Para nuestro caso lo utilizaremos para predecir cual

será nuestra posición en intervalos de tiempo cuando se nos jala a una posición inicial de 10 metros y nos sueltan, el modelo a estudiar es un bungee.



Utilizando el método de Euler y las ecuaciones de la física necesarias que se requieren para interpretar este movimiento, nuestra ecuación queda como la siguiente:

$$x(t + 2\Delta t) = -x(t) \left[1 + \frac{k\Delta t^2}{m} \right] + 2x(t + \Delta t) + \Delta t^2 g$$

Nuestro pseudocódigo de esta parte para predecir las posiciones es el siguiente:

Modelo_Euler_Method.c

Argumentos: delta_t, index, gramos, k

Regresa un valor de tipo double.

Inicio.

Declarar un contador;

Declarar e inicializar posición 1 y posición 2 en 10;

Declarar una variable temporal;

Desde contador = 0 Hasta contador mayor que index

$$V. \text{ temporal} = (-\text{posición 1}) * (1 + (k)(\text{delta_t al cuadrado})/m) + 2 * \text{posición 2} + (\text{delta_t al cuadrado})(\text{cte. Gravedad});$$

Posición 1 = posición 2;

Posición = V. temporal;

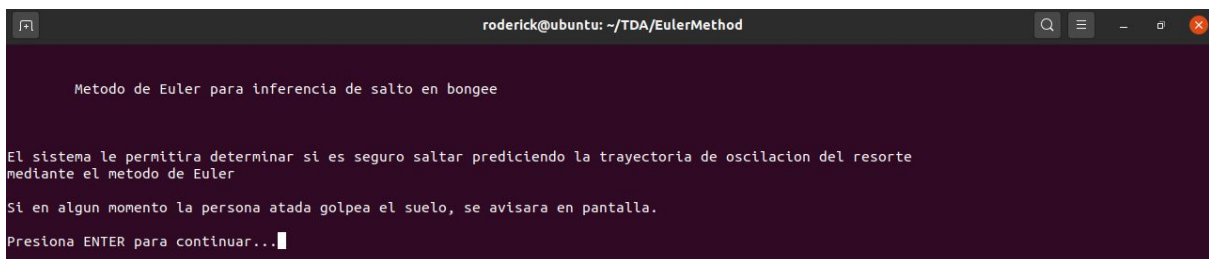
Fin Desde Hasta

Regresar el valor de la variable posición 2;

Fin.

Pantallas de funcionamiento.

Cuando se ejecuta el programa, aparecerá un mensaje de bienvenida explicando brevemente lo que hace el programa y mediante qué pasos. Cuando el usuario presiona ENTER se avanza a la siguiente pantalla.



```
roderick@ubuntu: ~/TDA/EulerMethod

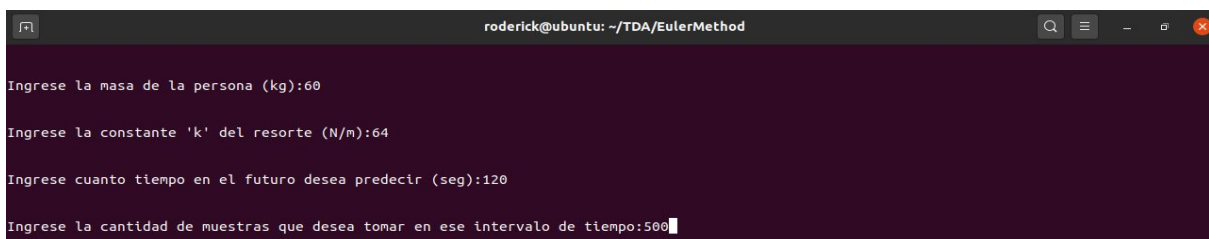
Metodo de Euler para inferencia de salto en bungee

El sistema le permitira determinar si es seguro saltar prediciendo la trayectoria de oscilacion del resorte
mediante el metodo de Euler

Si en algun momento la persona atada golpea el suelo, se avisara en pantalla.

Presiona ENTER para continuar...
```

A continuación se le pedirán al usuario los datos requeridos por el sistema. En este caso nos debe proporcionar 3 datos requeridos por las especificaciones. Masa, K del resorte y Δt , que en nuestro caso la obtenemos con la división de tiempo a predecir entre número de muestras.



```
roderick@ubuntu: ~/TDA/EulerMethod

Ingrese la masa de la persona (kg):60

Ingrese la constante 'k' del resorte (N/m):64

Ingrese cuanto tiempo en el futuro desea predecir (seg):120

Ingrese la cantidad de muestras que desea tomar en ese intervalo de tiempo:500
```

El sistema crea el archivo, te muestra los datos de entrada y responde a la pregunta si la persona chocará con el suelo y si es así en qué momento exacto lo hará.

También espera por la respuesta del usuario para hacer otra prueba con distintos datos o terminar de ejecutar el programa (opción 1 ejecuta de nuevo el programa y opción 2 termina la ejecución del programa).

```
Se ha creado el archivo!

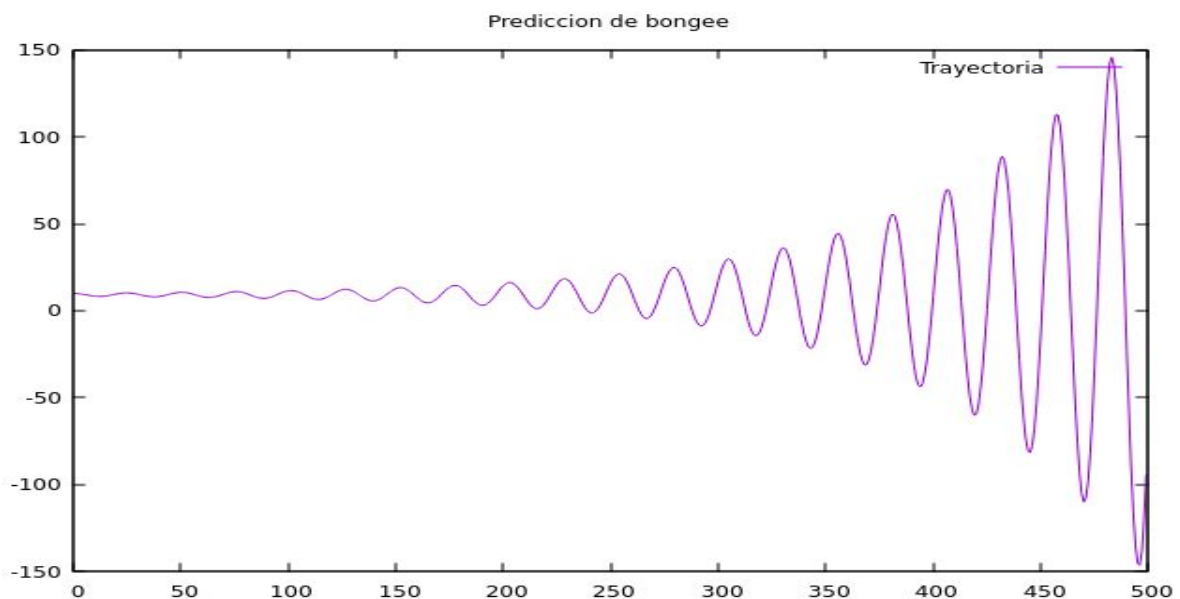
Se infirio un periodo de: 120 segundos.
Se tomaron un total de:500 muestras.
¿La persona chocara? Si
Chocara por primera vez a un tiempo de 119.76 segundos del inicio.

¿Desea predecir de nuevo con otros datos? 1-SI 0-NO
Opcion: 0

Hasta luego!!

roderick@ubuntu:~/TDA/EulerMethod$
```

Al terminar cada predicción, el programa despliega la correspondiente gráfica de la trayectoria en una ventana separada.



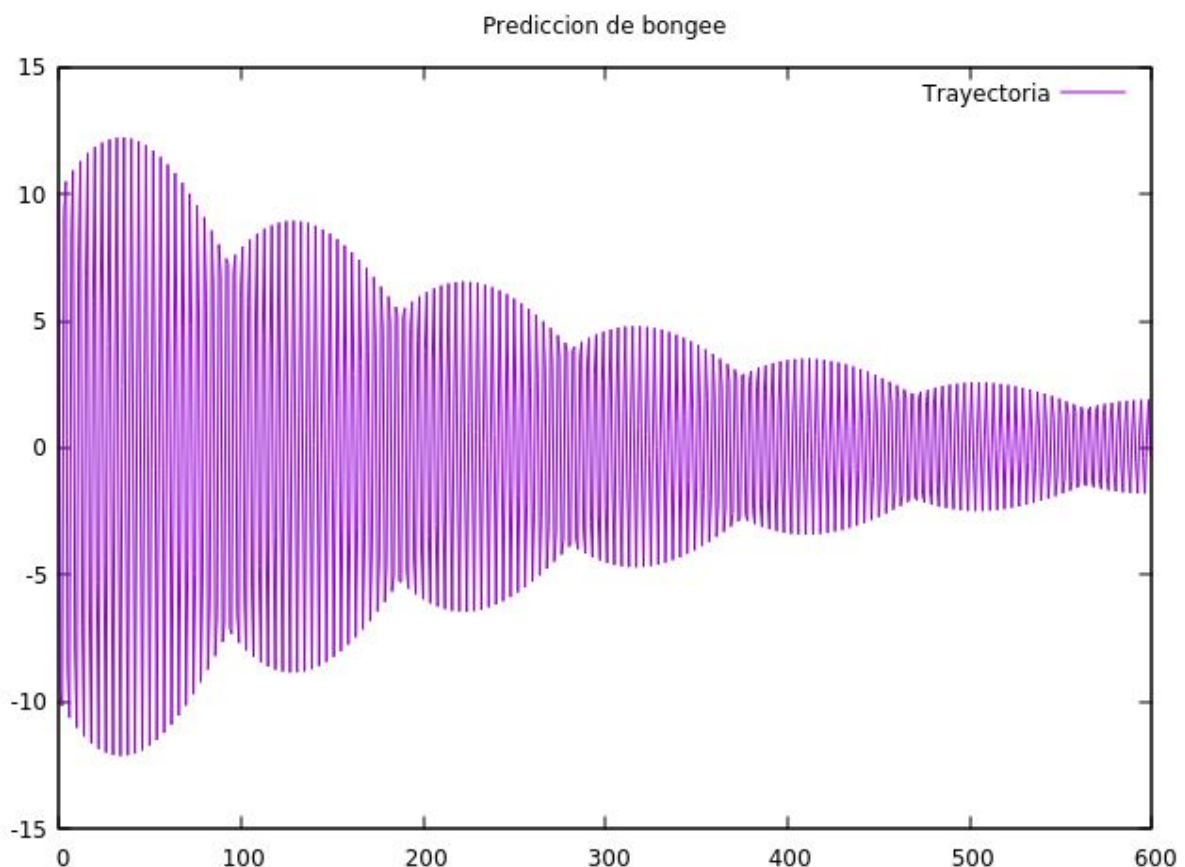
En la gráfica anterior podemos observar un comportamiento extraño y “antinatural”. Debido a que se está suponiendo que se tratan de condiciones ideales (resorte ideal, sin fuerzas externas que afecten el movimiento) es correcto pensar que la gráfica está bien ya que no hay fuerzas que están frenando el cuerpo en movimiento.

Sin embargo, en el momento en el que consideramos fuerzas externas en nuestra predicción, estamos hablando de un escenario más real y apegado a lo que naturalmente pasaría si nos exponemos a un suceso así. La forma de conocer esta segunda predicción es añadiendo el siguiente término a nuestra función de predicción:

$$\frac{-bt*\Delta t}{masa} \text{ y } -\left(2 - \frac{bt*\Delta t}{masa}\right)$$

Para así garantizar que lo que obtendremos será un resultado más real apegado a lo que pasaría si nos exponemos a un evento así.

Añadiendo estos factores anteriores, las gráficas que obtengamos comenzarán a verse así:



Siguiendo un comportamiento más real en cuanto a la oscilación en un resorte se tratara hasta así llegar a detenerse.

Conclusiones.

Como se revisaba, los algoritmos de predicción son muy útiles para un gran amplio catálogo de campos, con ellos se pueden inferir situaciones futuras que, sin estos métodos, serían casi inesperados. La forma en la que ayudan es variada, toma de decisiones, preparar acciones para tal o cual suceso posible, etc, pero también son muchas las “amenazas” que existen al conocer una predicción muy cercana a lo que en realidad pasará.

La implementación de estos algoritmos de predicción de tiempo es relativamente sencilla de implementar, pero, como ocurrió en esta ocasión, son difíciles de probar ya que hay muchos detalles que influyen y marcan la diferencia entre una buena y una mala predicción: número de muestras, tiempo a predecir, no conocer bien el modelo a seguir son solo algunas de las cuestiones que se deben tener en cuenta.

Para solucionar este problema es que se deben tener siempre en cuenta las diversas correctas prácticas de programación. Desde un análisis hasta una correcta y confiable codificación son esenciales a la hora de resolver un problema de esta o cualquier otra índole. Aunque pudiera parecer lo contrario, a la larga, mantener siempre con nosotros estas prácticas es casi seguro que no desperdiciamos tiempo valioso entendiendo cómo o se hace tal cosa en algún pedazo del código.

Código de Implementación.

```
•      bongee_prediction.h
//
// bongee_prediction.h
//
//
// Creado por Rodrigo Garcia y Jesus Enrique Domínguez el 25 de septiembre del 2020.
//

#ifndef bongee_prediction_h
#define bongee_prediction_h

/*
 * System headers required by the following declarations
 * (the implementation will import its specific dependencies):
 */

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



```
/*
 * Application specific headers required by the following declarations
 * (the implementation will import its specific dependencies):
 */

/* Constants declarations. */

/* Set EXTERN macro: */

#ifndef bongee_prediction_IMPORT
#define EXTERN
#else
#define EXTERN extern
#endif

/* Types declarations. */

/* Global variables declarations. */
#define COLUMNS 2

/* Function prototypes. */

/*
 *
 * La funcion vista_Menu proporciona al usuario un menu con la bienvenida y la forma de
operar del sistema.
 * Pedira los datos y las mandara a la siguiente funcion del proceso.
 *
 * @params
 * void

 * @returns
entero que confirma si se repetira el proceso o no
 */

EXTERN int vista_Menu(void);

/*
 *
 * La funcion vista_Resultados proporciona al usuario los resultados de su prediccion.
 * Determinara si el proceso desea repetirse.
 *
 * @params
```

```
* tiempo (double):
    tiempo en el futuro que se predijo
* muestras (double):
    muestras que se tomaron en el proceso de inferencia
* tiempo_choque (double):
    tiempo en el cual se genero el primer choque.
* chocado (int)
    variable para identificar si hubo un choque.

* @returns
    entero con la confirmacion de si se debe repetir o no
*/
```

```
EXTERN int vista_Resultados(double tiempo, double muestras, double tiempo_choque, int
chocado);
```

```
/*
*
* La funcion controlador_Proceso settea las variables y llama a los procesos dependiendo
de
* lo que se tenga que hacer ahora.
*
* @params
* masa (double):
    valor de la masa recibida del menu
* k_constante (int):
    valor de la constante k del resorte
* tiempo (double):
    cantidad de tiempo a predecir en segundos
* muestras (double):
    cantidad de muestras que se tomaran entre el tiempo dado

* @returns
    Estado para repetirse o no
*/
```

```
EXTERN int controlador_Proceso(double masa, int k_constante, double tiempo, double
muestras);
```

```
/*
*
* La funcion modelo_Euler_Method predice la posicion de la persona en el bungee
dependiendo de los valores dados
* y la regresa al controlador para ser analizada
```

```
*
* @params
*   delta_t (double):
*       variable del tiempo entre muestras dentro del periodo de tiempo dado por el usuario
*   index (int):
*       indice que calculara el metodo
*   masa (double):
*       valor de la masa de la persona proporcionado por el usuario
*   k_resorte (int):
*       constante k del resorte proporcionado por el usuario

* @returns
*   posicion obtenida de la prediccion en el index determinado
*/
```

```
EXTERN double modelo_Euler_Method(double delta_t,int index, double masa, int
k_resorte);
```

```
/*
*
* La funcion modelo_Imprime_Archivo imprime una matriz recibida en una archivo CSV
*
* @params
*   Archivo (FILE *):
*       Puntero al archivo donde se imprimira la matriz
*   columnas (size_t):
*       Columnas de la matriz
*   buffer[][columnas] (double):
*       Matriz recibida para imprimir
*   filas (size_t):
*       Filas de la matriz

* @returns
*   void
*/
```

```
EXTERN void modelo_Imprime_Archivo(FILE * Archivo, size_t columnas, double buffer[]
[columnas], size_t filas);
```

```
/*
* La funcion modelo_Nuevo_Archivo instacia un apuntador a un archivo nuevo
*
* @param
*   nombre_archivo (char *):
*       Nombre del archivo
*   modo (char *):
```

Modo en que sera abierto el archivo {r, rb, a, ab, w, wb}

```
* @return
    apuntador al archivo abierto.
*/

EXTERN FILE * modelo_Nuevo_Archivo(char *nombre_archivo, char *modo);

/*
 *
 * La funcion modelo_Grafica genera un plot del archivo CSV previamente creado.
 *
 * @params
 *     nombre_archivo (* char):
 *         Archivo el cual sera graficado por GNUplot
 *
 * @returns
 *     void
 */

EXTERN void modelo_Grafica (char * nombre_archivo);

#undef bongee_prediccion_IMPORT
#undef EXTERN

#endif /* bongee_prediction_h */
```

- prediction_method.c (Contiene el main)

```
//
// prediction_method.c
//
//
// Creado por Rodrigo Garcia el 25 de septiembre del 2020
//

#include "bongee_prediction.h"

int main(void)
{
    int ciclo = 1;
```

```
while(ciclo == 1)
{
    ciclo = vista_Menu();
}

printf("\n\nHasta luego!!\n\n");

return 0;
}
```

- vista_Menu.c

```
#include "bongee_prediction.h"
```

```
int vista_Menu(void)
{
    int k_resorte, repetir;
    double tiempo_a_calcular, muestras, masa;

    system("clear");

    printf("\n\n\tMetodo de Euler para inferencia de salto en bongee\n\n");
    printf("\n\nEl sistema le permitira determinar si es seguro saltar prediciendo la trayectoria de oscilacion del resorte\n\nmediante el metodo de Euler\n\nSi en algun momento la persona atada golpea el suelo, se avisara en pantalla.");
    printf("\n\nPresiona ENTER para continuar...");
    __fpurge(stdin);
    getchar();

    system("clear");

    printf("\n\nIngrese la masa de la persona (kg):");
    scanf("%lf", &masa);

    printf("\n\nIngrese la constante 'k' del resorte (N/m):");
    scanf("%d", &k_resorte);

    printf("\n\nIngrese cuanto tiempo en el futuro desea predecir (seg):");
    scanf("%lf", &tiempo_a_calcular);

    printf("\n\nIngrese la cantidad de muestras que desea tomar en ese intervalo de tiempo:");
    scanf("%lf", &muestras);

    repetir = controlador_Proceso(masa, k_resorte, tiempo_a_calcular, muestras);

    if(repetir == 1)
    {
```

```
    return 1;
}

else
{
    return 0;
}
}
```

- vista_Resultados.c

```
#include"bongee_prediction.h"
```

```
int vista_Resultados(double tiempo, double muestras, double tiempo_choque, int chocado)
{
    int repetir;

    printf("\n\n\t Se ha creado el archivo!\n\n");

    printf("\n\nSe infirio un periodo de: %.0lf segundos.\n",tiempo);
    printf("Se tomaron un total de:%.0lf muestras.\n",muestras);
    printf("¿La persona chocara? ");

    if(chocado == 1)
    {
        printf("Si\n");
        printf("Chocara por primera vez a un tiempo de %.2lf segundos del\n\n",tiempo_choque);
    }

    else
    {
        printf("No\n\n");
    }

    printf("¿Desea predecir de nuevo con otros datos? 1-SI 0-NO\n\n");
    printf("Opcion: ");
    scanf("%d",&repetir);

    if(repetir == 1)
    {
        return 1;
    }

    else
    {
        return 0;
    }
}
```

```
}
• controlador_Proceso.c

#include"bongee_prediction.h"

#define COLUMNAS 2
#define INDEX 0
#define POSICION 1

#define ARCHIVO "BONGEE.CSV"

int controlador_Proceso(double masa, int k_constante, double tiempo, double muestras)
{
    int muestras_entero, index, chocado=0, muestra_choque, repetir;
    double delta_t, posicion_en_index, inicio = 10;
    long int gramos;

    FILE * Archivo = NULL;

    muestras_entero = (int) muestras;

    double matriz_datos[muestras_entero][COLUMNAS];

    Archivo = modelo_Nuevo_Archivo(ARCHIVO, "w");

    delta_t = tiempo/muestras;
    gramos = masa*1000;

    system("clear");

    //printf("%d kg\t%d N/m\t%.3lf delta t\n\n", masa, k_constante, delta_t);

    for(index = 0; index < muestras_entero; index++)
    {
        posicion_en_index = modelo_Euler_Method(delta_t, index, masa, k_constante);

        //printf("\n\nPosicion en muestra %d = %.3lf", index, posicion_en_index);

        matriz_datos[index][INDEX] = index;
        matriz_datos[index][POSICION] = posicion_en_index;

        if(posicion_en_index < inicio && index != 0)
        {
            chocado = 1;
            muestra_choque = index;
        }
    }
}
```

```

    }
}

modelo_Imprime_Archivo(Archivo,COLUMNAS,matriz_datos,muestras);

fclose(Archivo);

//Llamada a la funcion que genera el plot
modelo_Grafica(ARCHIVO);

repetir = vista_Resultados(tiempo,muestras,muestra_choque*delta_t,chocado);

if(repetir == 1)
{
    return 1;
}

else
{
    return 0;
}

}

```

- modelo_Euler_Method.c

```

#include<math.h>
#include"bongee_prediction.h"

#define G 9.81

double modelo_Euler_Method(double delta_t,int index, double masa, int k_resorte)
{
    int count;
    double pos1 = 10, pos2 = 10, temp, m = 60;
    float bt = 10;

    for(count = 0;count < index;count++)
    {
        temp = (-pos1)*(((bt*delta_t)/m) + 1 + ((k_resorte*(pow(delta_t,2)))/m)) -
        (2*(-bt*delta_t)/m)*pos2 + (pow(delta_t,2))*G;

        pos1 = pos2;
        pos2 = temp;
    }
}

```



```

return pos2;

}

```

- modelo_Manejo_Archivos.c (modelo_Imprime_Archivo y modelo_Nuevo_Archivo)

```
#include "bongee_prediction.h"
```

```

FILE * modelo_Nuevo_Archivo(char * nombre_archivo, char * modo)
{
    FILE * file;

    file = fopen(nombre_archivo, modo);

    return file;
}

```

```

void modelo_Imprime_Archivo(FILE * Archivo, size_t columnas, double buffer[] [columnas],
size_t filas)
{
    int count_rows, count_columns=0;

    for(count_rows = 0; count_rows < filas; count_rows++)
    {
        fprintf(Archivo, "%.0lf ", buffer[count_rows][count_columns]);
        count_columns++;
        fprintf(Archivo, "%.3lf", buffer[count_rows][count_columns]);

        count_columns=0;

        fprintf(Archivo, "\n");
    }
}

```

- modelo_Grafica.c

```
include "bongee_prediction.h"
```

```
#define NUM_COMMANDS 2
```

```

void modelo_Grafica (char * nombre_archivo)
{
    char * commandsForGnuplot[] = {"set title \"Prediccion de bongee\\\"", "plot 'BONGEE.CSV'
using 1:2 t 'Trayectoria' with lines"};
    FILE * gnuplotPipe = NULL;
}

```

```
gnuplotPipe = popen("gnuplot -persistent","w");

//Para general el plot
for(int i = 0;i < NUM_COMMANDS;i++)
{
    fprintf(gnuplotPipe,"%s \n",commandsForGnuplot[i]);
}

pclose(gnuplotPipe);

}
```