Homework 5, Derivación Numérica

Camilo Andres Huertas Archila[1]

[1]20212107049

---

1 Introducción:

Se realizó 3 métodos de derivación númerica (**Forward, Backward y diferencias centradas**), de datos de tiempo y posición de una caída libre de 4s y 80m de altura (datos los cuales fueron generados computacionalmente), esto para calcular el valor de la **velocidad** y la **aceleración** en cada instante.
También se buscó determinar cual de los 3 métodos era más preciso.

1.1 Expresiones matemáticas usadas:

1.1.1 Método Forward:

$$v_i = \frac{x_{i+1} - x_i}{t_{i+1} - t_i}$$

$$a_i = \frac{v_{i+1} - v_i}{t_{i+1} - t_i}$$

1.1.2 Método Backward:

$$v_i = \frac{x_i - x_{i-1}}{t_i - t_{i-1}}$$

$$a_i = \frac{v_i - v_{i-1}}{t_i - t_{i-1}}$$

1.1.3 Diferencias centradas:

$$v_i = \frac{x_{i+1} - x_{i-1}}{t_{i+1} - t_{i-1}}$$

$$a_i = \frac{v_{i+1} - v_i}{t_{i+1} - t_i}$$

2 Implementación en Libreoffice Calc:

En el libro haciendo uso de las funciones de Excel se pudo obtener la columna de velocidad y aceleración en cada instante de tiempo (cada método en una hoja de cálculo).
Para contestar la pregunta *¿Cual método es más aproximado?* se calculó el error medio:

- Se añadió una columna con *velocidad teórica* cuya expresión es $v = -gt$ (teniendo en cuenta que $V_i = 0$), por lo tanto conociendo los instante de tiempo y $g = 9.81$, se pudo determinar dicho valor.
- Con la expresión $\epsilon = \frac{|experimental - teorico|}{|teorico|} \times 100$, se calculó el porcentaje de error para cada fila, en cada método.

- Luego se calculó la media aritmética para la columna de **errores**.

Los resultados fueron:
5,45326 % de error medio para el metodo Forward y Backward, y un **0% de error medio** para las diferencias centradas.

3 Implementación en C++ con Gnuplot:

Aqui se invocan las librerias a usar como por ejemplo *vector* para los array dinamicos, *fstream* para la lectura y escritura de archivos, entre otros.

```
// Librerias a usar:
#include<iostream>
#include<fstream>
#include<cstdlib>
#include<vector>
#include<ostream>
#include<string>
```

Listing 1: Librerias usadas

```
// Variables
std::vector<double> tiempos, posiciones, velocidades_1, aceleraciones_1,
    velocidades_2, aceleraciones_2, velocidades_3, aceleraciones_3;
double tiempo, posicion, velocidad, aceleracion;
```

Listing 2: Variables globales

Usando la filosofía de *divide y vencerás*, se construyó diferentes donde cada una realiza una tarea específica. Por cuestión de economización es posible fusionar alguna de esas funciones con otras (principalmente aquellas que recorrar los vectores con ciclo for con los mismos intervalos); no obstante, aqui no fue necesario.

```
// Funciones
void LeerDatos();
void DerivacionForward();
void DerivacionBackward();
void DerivacionCentrada();
void GuardarDatos();
void GraficarDatos(const std::string& nombreImagen, const std::string&
    nombreTitulo, const std::string& nombreColunmaX, const std::string&
    nombreColumnaY, const std::string& usingConfig, const std::string&
    nombreTrayectoria);
```

Listing 3: Funciones

```
int main(void){
    LeerDatos();
    DerivacionForward();
    DerivacionBackward();
    DerivacionCentrada();
    GuardarDatos();
    GraficarDatos("VvsT_Forward.jpg" , "Metodo Forward / V vs T" , "Tiempo (s)" ,
        "Velocidad (m/s)" , "1:3" , "V");
    GraficarDatos("AvsT_Forward.jpg" , "Metodo Forward / A vs T" , "Tiempo (s)" ,
        "Aceleracion (m/s**2)" , "1:6" , "A");
    GraficarDatos("VvsT_Backward.jpg" , "Metodo Backward / V vs T" , "Tiempo (s)"
        , "Velocidad (m/s)" , "1:4" , "V");
    GraficarDatos("AvsT_Backward.jpg" , "Metodo Backward / A vs T" , "Tiempo (s)"
        , "Aceleracion (m/s**2)" , "1:7" , "A");
```

```
    GraficarDatos("VvsT_Centradas.jpg" , "Diferencias centradas / V vs T" , "
        Tiempo (s)" , "Velocidad (m/s)" , "1:5" , "V");
    GraficarDatos("AvsT_Centradas.jpg" , "Diferencias centradas / A vs T" , "
        Tiempo (s)" , "Aceleracion (m/s**2)" , "1:8" , "A");

    std :: cout << "Graficas generadas y guardadas en archivos .jpg" << std ::endl
        ;

    return 0;
}
```

Listing 4: Función principal

Se lee el archivo DatosCaida.dat con 2 columnas (tiempo y posición) y se almacenan en un vectores.

```
void LeerDatos(){
    std :: ifstream archivo("DatosCaida.dat");
    if (!archivo){
        std :: cerr << "Error al abrir el archivo" << std :: endl;
        exit(0);
    }
        while (archivo >> tiempo >> posicion){
        tiempos.push_back(tiempo);
        posiciones.push_back(posicion);
    }
    archivo.close();
}
```

Listing 5: Función para leer datos en .dat

Esta función hace el cálculo de derivación numérica por método Forward. Es importante notar que al calcular la velocidad de la ultima fila, no existe el tiempo ni la posición "i+1" para ese caso, por ello el ciclo for se detiene una fila antes y agrega un 0 (por cuestiones de lectura y gráfico de datos).
Para el caso de la aceleración, lo mismo, se detiene 2 filas antes y luego se llenan los faltantes con 0.

```
void DerivacionForward(){
    velocidades_1.clear();
    aceleraciones_1.clear();
    for (int i = 0 ; i < (tiempos.size() - 1);i++){
        velocidad = 0;
        velocidad = ((posiciones[i+1]-posiciones[i])/(tiempos[i+1]-tiempos[i]));
        velocidades_1.push_back(velocidad);
    }
    velocidades_1.push_back(0);
    for (int i = 0 ;  i < (tiempos.size() - 2); i++){
        aceleracion = 0;
        aceleracion = ((velocidades_1[i+1]-velocidades_1[i])/(tiempos[i+1]-tiempos
            [i]));
        aceleraciones_1.push_back(aceleracion);
    }
    aceleraciones_1.push_back(0);
    aceleraciones_1.push_back(0);
}
```

Listing 6: funcion para Derivación Forward

Misma correción, solo que para el método Backward la velocidad en la fila 1 se llena con cero y el ciclo for inicia desde 1.
Para la aceleración, fila 1 y 2 = 0 y el ciclo for inicia desde 2.

```
void DerivacionBackward(){
```

```
    velocidades_2.clear();
    aceleraciones_2.clear();
    velocidades_2.push_back(0);
    for (int i = 1 ; i < tiempos.size();i++){
        velocidad = 0;
        velocidad = ((posiciones[i]-posiciones[i-1])/(tiempos[i]-tiempos[i-1]));
        velocidades_2.push_back(velocidad);
    }
    aceleraciones_2.push_back(0);
    aceleraciones_2.push_back(0);
    for (int i = 2 ;  i < tiempos.size(); i++){
        aceleracion = 0;
        aceleracion = ((velocidades_2[i]-velocidades_2[i-1])/(tiempos[i]-tiempos[i
            -1]));
        aceleraciones_2.push_back(aceleracion);
    }
}
```

Listing 7: Función para Derivación Backward

En este caso la corrección para la velocidad es añadir 0 en la primera y ultima fila, además el ciclo for inicia una fila después y acaba una fila antes.
Para la aceleración se llena con 0 la primera, segunda, penúltima y ultima fila, y el ciclo for inicia 2 filas después y acaba 2 filas antes.
Todo esto es necesario para que no haga una búsqueda inexistente en los vectores (se salga del rango y de error.)

```
void DerivacionCentrada(){
    velocidades_3.clear();
    aceleraciones_3.clear();
    velocidades_3.push_back(0);
    for (int i = 1 ; i < (tiempos.size() - 1);i++){
        velocidad = 0;
        velocidad = ((posiciones[i+1]-posiciones[i-1])/(tiempos[i+1]-tiempos[i-1])
            );
        velocidades_3.push_back(velocidad);
    }
    velocidades_3.push_back(0);
    aceleraciones_3.push_back(0);
    aceleraciones_3.push_back(0);
    for (int i = 2 ;  i < (tiempos.size() - 2); i++){
        aceleracion = 0;
        aceleracion = ((velocidades_3[i+1]-velocidades_3[i-1])/(tiempos[i+1]-
            tiempos[i-1]));
        aceleraciones_3.push_back(aceleracion);
    }
    aceleraciones_3.push_back(0);
    aceleraciones_3.push_back(0);
}
```

Listing 8: Función para Derivación centrada

Se guardan los datos en un nuevo .dat (teniendo en cuenta que en clase aun no hemos visto la opción de agregar datos a un .dat ya existente).

```
void GuardarDatos(){
    std::ofstream archivo("datos_calculados.dat");
    if (archivo.is_open()){
        archivo.precision(5);
        for (int i=0; i < tiempos.size() ; i++){
```

```
            archivo << tiempos[i] << " " << posiciones[i] << " " << velocidades_1[
                i] << " " << velocidades_2[i] << " " << velocidades_3[i] << " " <<
                aceleraciones_1[i] << " " << aceleraciones_2[i] << " " <<
                aceleraciones_3[i] << std :: endl ;
        }
    }
}
```

Listing 9: Función para guardar los archivos en un nuevo .dat

Obtenemos las graficas usando Gnuplot desde C++.

```
void GraficarDatos(const std::string& nombreImagen, const std::string&
    nombreTitulo, const std::string& nombreColunmaX, const std::string&
    nombreColumnaY, const std::string& usingConfig, const std::string&
    nombreTrayectoria){
    std :: string comando = "gnuplot -e \"set terminal jpeg; "
        "set output '" + nombreImagen + "'; "
        "set title '" + nombreTitulo + "'; "
        "set xlabel '" + nombreColunmaX + "'; "
        "set ylabel '" + nombreColumnaY + "'; "
        "set grid; "
        "plot 'datos_calculados.dat' using " + usingConfig + " with linespoints
            title '" + nombreTrayectoria + "'\"";
    system(comando.c_str());
}
```

Listing 10: Función para graficar los datos

4 Gráficas obtenidas:

NOTA: Es de notar que las gráficas hechas en Gnuplot contiene datos en el eje y = 0 ya sea al inicio o al final, esto es debido a la propia implementación en el código c++ (por una cuestión de tamaños de vectores).
En Libreoffice/excel, aunque toda la columna de aceleracion sea igual a -9.81, se grafica de forma caotica, sin explicación aparente.

4.1 Con método Forward:

4.1.1 Con Libreoffice:

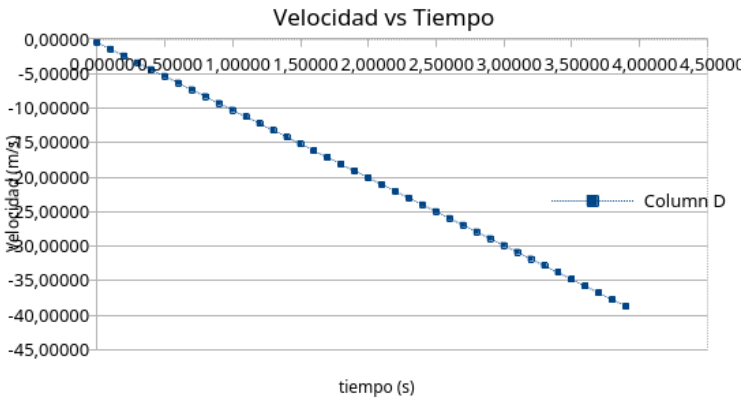

Figure 1: V vs T , Método Forward

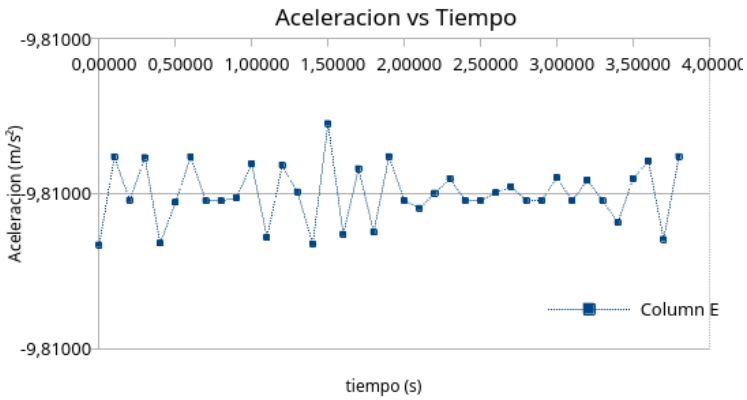

Figure 2: A vs T , Método Forward

4.1.2 Con C++ y Gnuplot:

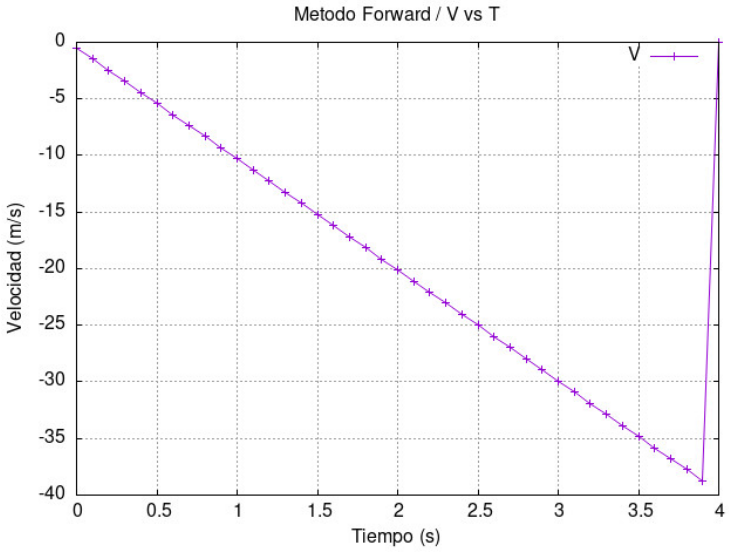

Figure 3: V vs T , Método Forward

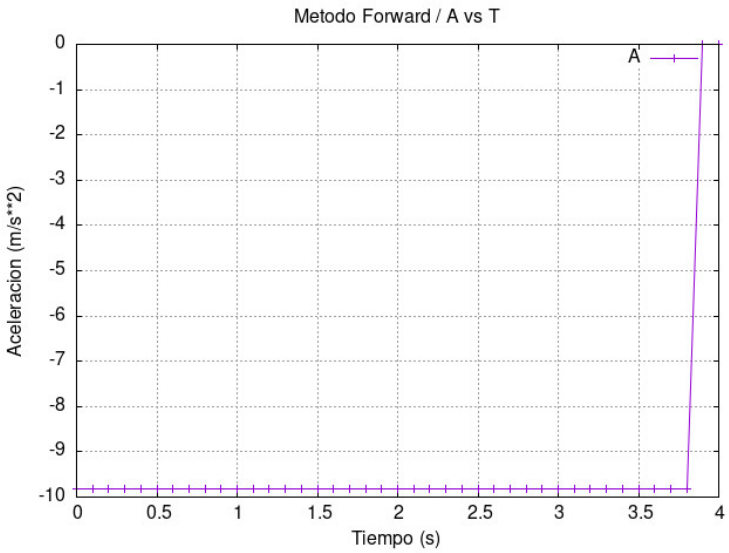

Figure 4: A vs T , Método Forward

4.2 Con método Backward:

4.2.1 Con Libreoffice:

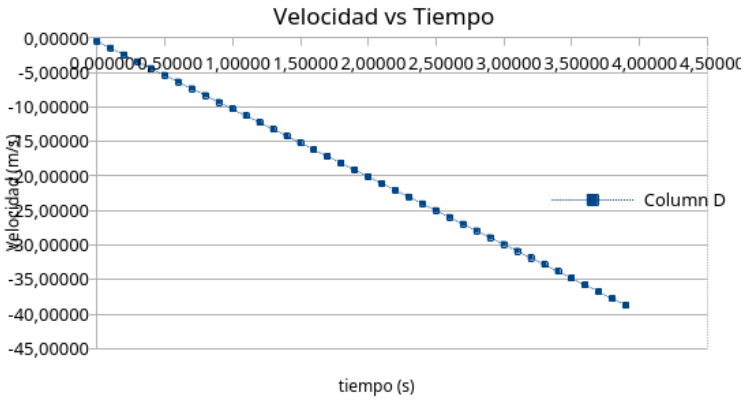

Figure 5: V vs T , Método Backward

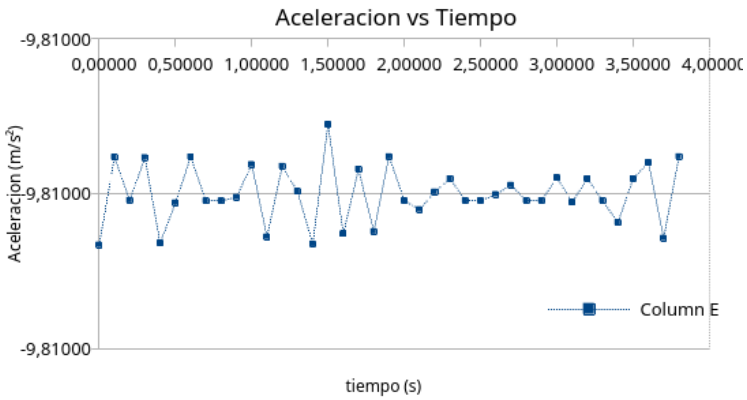

Figure 6: A vs T , Método Backward

4.2.2 Con C++ y Gnuplot:

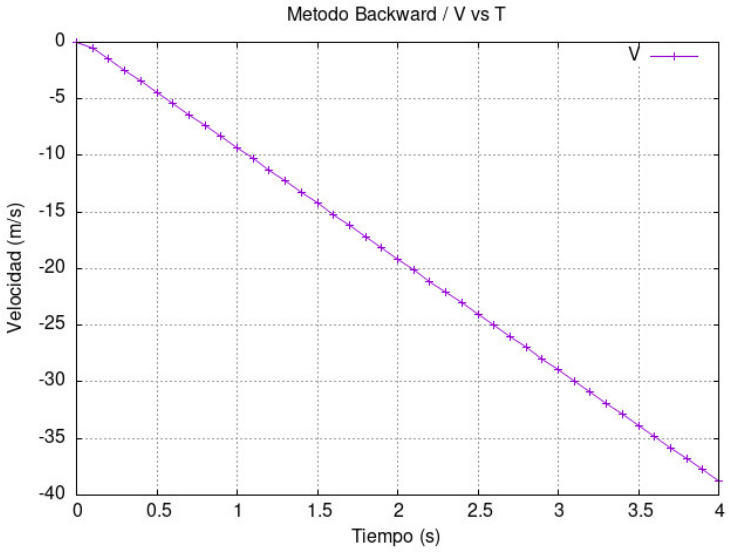

Figure 7: V vs T , Método Backward

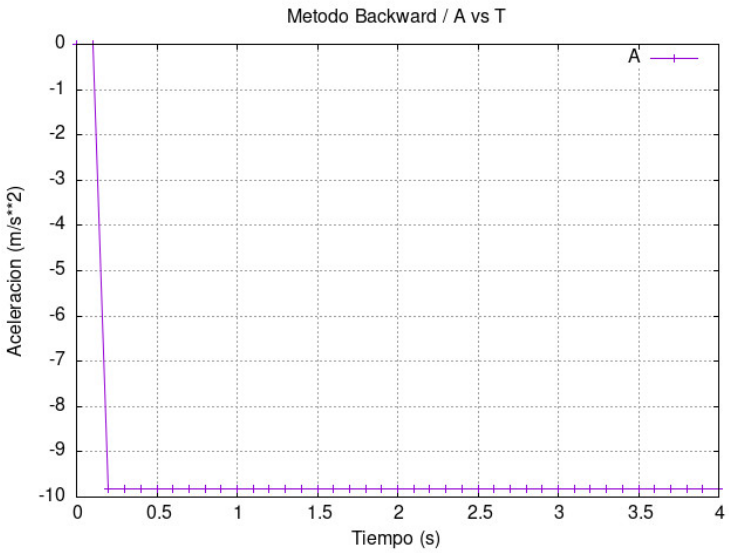

Figure 8: A vs T , Método Backward

4.3 Con Diferencias Centradas:

4.3.1 Con Libreoffice:

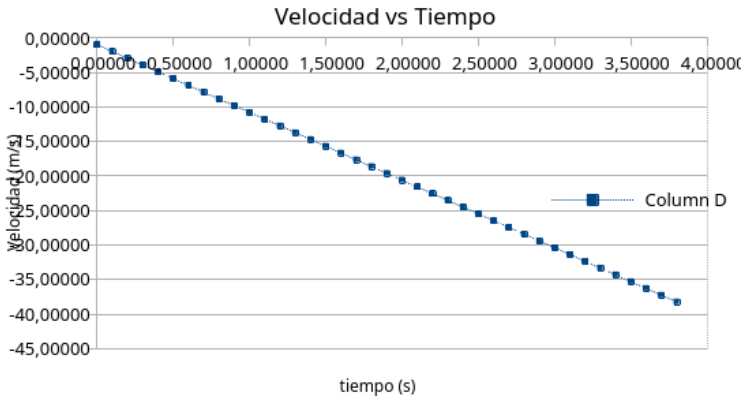

Figure 9: V vs T , Diferencias centradas

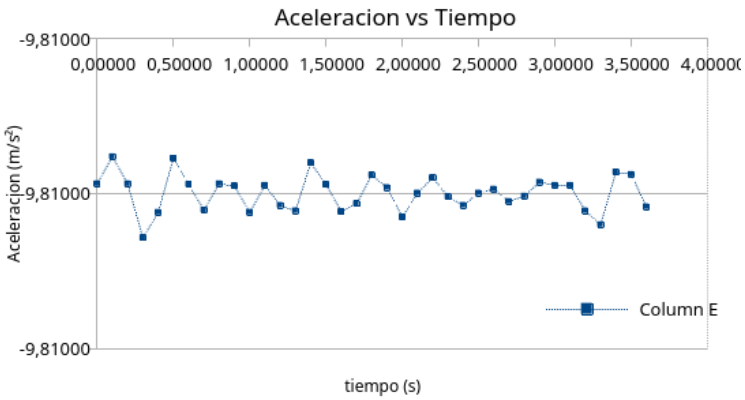

Figure 10: A vs T , Diferencias centradas

4.3.2 Con C++ y Gnuplot:

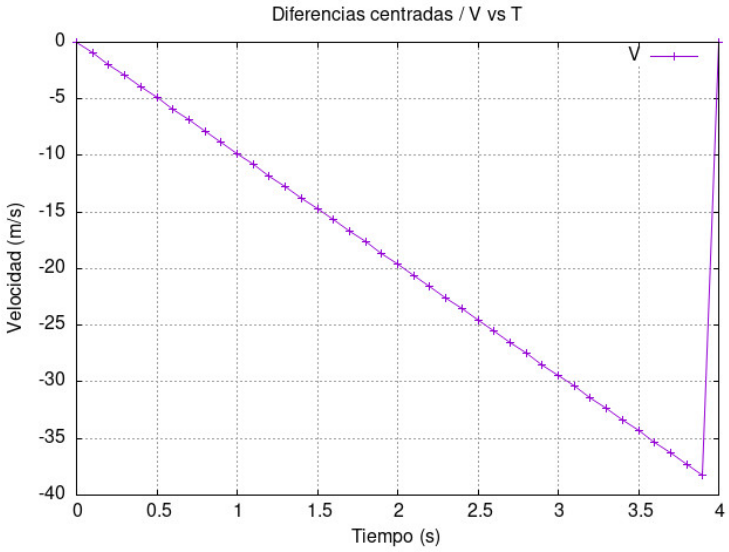

Figure 11: V vs T , Diferencias centradas

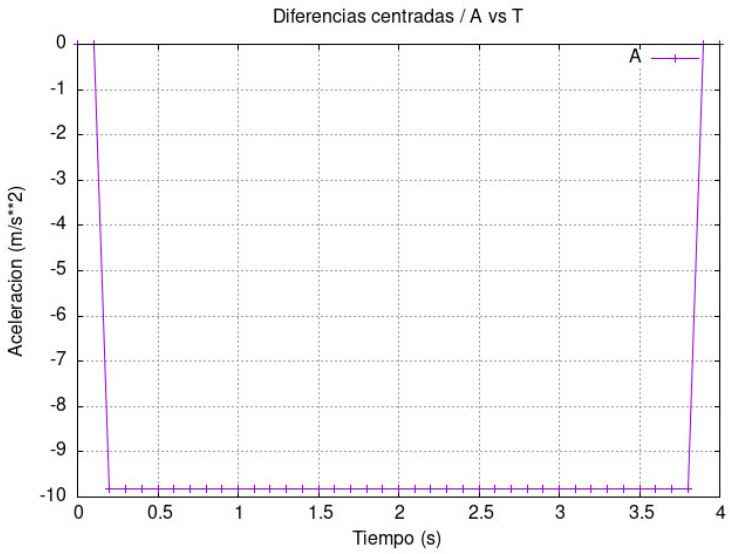

Figure 12: A vs T , Diferencias centradas

5 Conclusiones

- Aunque tomó más tiempo realizar la implementación en c++ y generar las graficas en Gnuplot, ahora se tiene un script de derivación numerica para todo conjunto de datos experimentales (que tengan tiempo y posición), ello supone un ahorro de tiempo para futuras ocasiones.
- En este caso el método de diferencias centradas obtuvo un 0% de error medio.