Portafolio

Física Computacional II

Javiera Arauco; Fernanda Mella; Sebastián Reyes
Ciencias Físicas
Departamento de Física
Facultad de Ciencias Físicas y Matemáticas
UNIVERSIDAD DE CONCEPCIÓN

Agosto-Diciembre 2023
Concepción, Chile

Profesor Guía: Dr. Roberto Navarro Maldonado

Índice general

Introducción **3**

1. Presentación del/la estudiante **4**

2. Evidencias de programación **7**
2.1. Triángulo de Pascal . . . 7
2.2. Fuerza gravitacional . . . 8
2.3. Algoritmo de Shor extrapolado . . . 10
2.4. Derivada de un polinomio y su error . . . 11
2.4.1. Derivada Adelantada . . . 11
2.5. Manchas Solares en el tiempo . . . 14
2.5.1. Derivadas Discretas . . . 14
2.6. Teorema fundamental del Cálculo . . . 16
2.6.1. Derivada Centrada . . . 16
2.6.2. Método de Simpson . . . 16
2.7. Integral numérica vs analítica . . . 18
2.8. Funciones de Bessel . . . 20
2.9. Péndulo Simple . . . 22
2.9.1. Método de Euler . . . 23
2.9.2. Análisis de Energía y Espacio de Fase . . . 25
2.9.3. Método de Runge-Kutta . . . 27
2.10. Péndulo doble . . . 29
2.10.1. Solución Numérica . . . 30
2.10.2. Análisis de Energía . . . 31
2.11. Atractor de Lorenz . . . 33
2.12. Ecuaciones de Lane-Emden . . . 36
2.13. Búsqueda de ceros . . . 39
2.13.1. Método de Bisección . . . 39
2.14. Ceros de las funciones de Bessel . . . 41
2.14.1. Método de Bisección Mejorado . . . 42
2.15. Ley de Planck . . . 43

3. Conclusiones **46**

Introducción

Este portafolio deberá incluir muestras o evidencias de las actividades desarrolladas en la asignatura de Física Computacional II (510240), dictada en el segundo semestre de 2023 en el departamento de física, facultad de Ciencias Físicas y Matemáticas, de la Universidad de Concepción.

Esta asignatura se enfoca en la resolución de problemas en Física mediante lenguajes de programación especializados (en este caso `python`), contribuyendo a conocimientos generales de construcción y aplicación de algoritmos computacionales, optimización de simulaciones numéricas y organización de códigos mediante herramientas computacionales.

Esta asignatura contribuye al logro de las siguientes competencias del perfil de egreso "Aplicar lenguajes de programación avanzados tales como Fortran, C++, Python, etc., a problemas de la Física", y se espera obtener los siguientes resultados de aprendizaje:

1. Aplicar las herramientas computacionales en la resolución numérica de problemas en Física.
2. Generar programas computacionales apoyándose en algoritmos y conceptos de la física matemática y estadística.
3. Diferenciar, integrar y resolver ecuaciones diferenciales ordinarias, en forma numérica.

La evaluación se realizará a través de este portafolio. El profesor evaluará cada una de las muestras y el conjunto del portafolio a través de una pauta que se entregará oportunamente. El/la estudiante deberá auto-evaluar su portafolio y seleccionará un conjunto de evidencias de este portafolio, las cuales serán expuestas ante sus compañeros durante clases y cada dos semanas, quienes también lo evaluarán a través de una rúbrica. El documento de portafolio contendrá una sección de conclusiones donde autoevaluará su desempeño a través de una reflexión final y de sus respuestas a una serie de preguntas que serán consensuadas con el profesor.

Como criterios de evaluación, se considerará:

1. Adecuación de las muestras incluidas a los resultados del aprendizaje y a los contenidos de la asignatura.
2. Coherencia de las evidencias con las actividades realizadas en clases.
3. Competencias comunicativas: escritura (ortografía, gramática, sintaxis, estilo) y presentaciones orales (habla correctamente y con claridad; utiliza términos técnicos apropiadamente, etcétera).
4. Presentación: Claridad, limpieza y orden del documento.
5. Autoevaluación: Seriedad y reflexión de las justificaciones dadas al principio y al final del documento.

Capítulo 1

Presentación del/la estudiante

Datos personales

Nombre completo Javiera Alejandra Arauco Orias

Matrícula 2022445092

Fecha de Nacimiento 16 de febrero del 2004

Nacionalidad Chilena

Breve biografía académica

Soy Javiera Arauco, estudiante de carrera de Ciencias Físicas en la Universidad de Concepción, actualmente me enuentro en el cuarto semestre. La enseñanza básica la realicé en la Escuela Ecuador de Tomé, mientras que la enseñanza media la realicé en el Instituto de Humanidades de Concepción, lugar donde me di cuenta que lo que me impulsa en esta vida es aprender de lo que me rodea, por lo que decidí estudiar la carrera en la que estoy actualmente.

Visión general e interés sobre la asignatura

Mi visión de la asignatura es poder generar habilidades más sólidas de programación. En lo personal lo que más me interesa de la asignatura es mejorar mi manejo con los computadores, poder utilizarlos "sin miedo", puesto que en general me considero bastante torpe con ellos. También me interesa poder utilizarlos como herramientas para generar simulaciones físicas.

Resultados esperados de este portafolio

Lo que espero conseguir con este portafolio es poder tener un registro de los avances que tenga junto a mis compañeros en cuanto a conocimientos de programacón. Además, creo que el portafolio es una buena oportunidad para aprender a trabajar en equipo, habilidad que considero esencial en esta carrera. Por último, creo me ayudará a poder ser más ordenada con el registro de los ejercicios que se han realizado a lo largo del semestre.

Datos personales

Nombre completo Fernanda Mella Alvarez.

Matrícula 2022425300

Fecha de Nacimiento 28 de Junio del 2003

Nacionalidad Chilena

Breve biografía académica

Soy Fernanda Mella, estudiante de la carrera Ciencias Físicas. La educación media la realicé en el colegio Almondale Lomas de la ciudad Concepción. También desde marzo de este año he sido parte del equipo de Difusión de la facultad, en donde he hecho charlas a alumnos y profesores no sólo de las carreras sino que también de un par de áreas de física.

Visión general e interés sobre la asignatura

En un inicio la visión general que tuve de la asignatura fue que se iba a aprender los inicios de programación que vamos a necesitar a lo largo de la carrera. Y el primer interés que me generó la asignatura fue el poder realizar simulaciones de distintos fénomenos físicos, ya que desde que me enteré que se podía hacer yo quise hacerlo. Por lo que en la asignatura se ve que vamos a llegar a cumplir ese objetivo.

Resultados esperados de este portafolio

Para mí, los resultados que espero de este portafolio es poder tener resgistrado los avances y ejercicios que incluyan técnicas de programación que iremos aprendiendo clase a clase. Para que en un futuro sea más fácil acceder a estos conocimientos si es necesario. También espero fortalecer mi consistencia al actualizar el respositorio semanalmente.

Datos personales

Nombre completo Sebastián Reyes

Matrícula 2022426012

Fecha de Nacimiento 7 de julio del 2003

Nacionalidad Chileno

Breve biografía académica

Soy Sebastián Reyes, estudio la carrera de Ciencias Físicas. La mayoría de mi educación básica la hice en "Colegio Niño Jesús", en Lota, luego me fuí al "Instituto de Humanidades de Lota", en el que hice toda mi enseñanza media. Aparte de la educación formal, también exploré varias áreas, destaco mi participación en conjuntos folkóricos, de los cuales pude aprender distintas danzas chilenas y latinoamericanas.

Visión general e interés sobre la asignatura

A esta asignatura, en un principio creí que no sería mas que un "spin off"de Computación Científica del primer semestre, pero cuando nos la presentaron, me di cuenta que sería mucho más. Por fin sería la ocasión en que de verdad aprendería Python y lo usaría en aplicaciones físicas. Lo que más me llama la antención y lo que me genera interés es que me dará los conocimientos para dar un primer paso a hacer mis propias simulaciones de fenómenos físicos, me emociona puesto que, en un futuro, será mi código quien será el encargado de simular el universo.

Resultados esperados de este portafolio

Al confeccionar este portafolio espero desarrollar la habilidad o adquirir la facilidad de realizar un trabajo colaborativo a largo plazo, adaptándome a la la metodología de trabajo que requiera el proyecto. Además, espero mejorar mi constancia en la relalización de mis pendientes diarios, como lo es este portafolio.

Capítulo 2

Evidencias de programación

2.1. Triángulo de Pascal

Fecha de la actividad: 22 de septiembre de 2023

La actividad fue realizada de manera individual por Javiera Arauco y revisada por el grupo. El objetivo de esta actividad es crear un programa que imprima en pantalla el triángulo de Pascal, es decir, los coeficientes del polinomio de grado n (siendo n un número entero positivo ingresado por quien ejecute el programa) que resulta de expandir la fórmula $(1+x)^n$.

Para generar las filas del triángulo de Pascal, se utiliza el teorema del binomio de Newton:

$$\sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k, \tag{2.1}$$

Esta fórmula matemática se utiliza para expandir una expresión binomial elevada a una potencia específica. De ella, la expresión que nos brinda los coeficientes del polinomio que se forma es: $\binom{n}{k}$. Cabe señalar que:

$$\frac{n!}{k!(n-k)!} = \binom{n}{k}. \tag{2.2}$$

Como se ve en la ecuación anterior, para realizar el programa también se tuvo que trabajar con factoriales. Se sabe que el factorial de un número (entero positivo) a se calcula multiplicando todos los números enteros positivos desde 1 hasta a, es decir:

$$a! = a \times (a-1) \times (a-2) \times ... \times 2 \times 1. \tag{2.3}$$

Para lograr esto, en el programa se crea una función "fact"que devuelve el factorial del número. Para ello se utiliza un ciclo for.

```
1    def fact(n):
2        f=1
3        for i in range(n):
4            f = f*(i+1)
5        return f
```

Listing 2.1: Definición de la función factorial

Luego se crea otra función (a la que se llamó ”filita”) que entrega las filas del triángulo de Pascal.

```
    def filita(n):
        fun=1
        fila= []
        fila.append(1.0)
        for i in range(n):
            fun= (fact(n))/(fact(i+1)*fact(n-(i+1)))
            fila.append(fun)
        return fila
```

Listing 2.2: Código para generar las filas del triángulo de Pascal.

Como se ve en código, dentro de la función primero se crea un arreglo vacío "filaz se define la variable "fun"que se utilizará posteriormente, dentro del cilco "for". Lo que hace el ciclo "for"dentro de la función es entregar los números correspondientes a los coeficientes al utilizar la ecuación 2.2, y los guarda en en el arreglo ”fila”.

Para que el código fuese más versátil, se pide que la variable n sea ingresada por quien utilice el programa, siendo n la variable que nos indique hasta qué número se eleva el binomio. Finalmente, otra vez se hace uso de un ciclo "for"para que se nos entregue cada fila creada por la función ”filita”, generándose así el triángulo de Pascal.

```
    n= int(input("Ingrese un valor entero positivo: "))

    for i in range(n+1):
        print(filita(i))
```

Listing 2.3: Código para generar el triángulo de Pascal.

```
Ingrese un valor entero positivo: 6
[1.0]
[1.0, 1.0]
[1.0, 2.0, 1.0]
[1.0, 3.0, 3.0, 1.0]
[1.0, 4.0, 6.0, 4.0, 1.0]
[1.0, 5.0, 10.0, 10.0, 5.0, 1.0]
[1.0, 6.0, 15.0, 20.0, 15.0, 6.0, 1.0]
```

Figura 2.1: Triángulo de Pascal.

En esta avtividad se creó un programa en Python que imprime en la pantalla el triángulo de Pascal utilizando como base matemática el teorema del binomio de Newton. Lo que se puede concluir de haber hecho este ejercicio, es que a partir de conceptos matemáticos que fueron estudiados previamente, como el Binomio de Newton, es posible dar solución de una forma relativamente sencilla a problemas algo desafiantes de programación.

2.2. Fuerza gravitacional

Fecha de la actividad: 23 de septiembre de 2023

Esta actividad fue hecha de manera individual por Sebastián Reyes. Y revisada por el grupo.
Este ejercicio reune conceptos básicos de Python, los cuales se usan en todo este documento. Este ejercicio consta en graficar la magnitud de la fuerza gravitacional de dos masas separadas por una distancia variable r, la cual se debe encontrar entre $0{,}1 < r < 10$.

El primer paso es definir la magnitud de fuerza gravitacional(F). Por lo que se hace uso de la Ley de Gravitación Universal, la cual según el libro *Física para ciencias e ingenería con Física Moderna* [1] se define como:

$$F = G\frac{m_1 m_2}{r^2}, \tag{2.4}$$

Siendo $G = 67430 \times 10^{-11} Nm^2 kg^{-2}$, la constante de gravitación universal.

Este valor es crucial en esta simulación de la fuerza gravitacional. En el desarrollo de esta actividad se usarán valores pequeños de distancia y masas, con el objetivo de que al representar la magnitud de la fuerza gráficamente se aprerecie de mejor manera el cambio de la fuerza cuando la distancia cambia.

En este desarrollo no se definen valores fijos de las masas, en su lugar, al momento de ejecutar el código se le pide al usuario ingresar 2 valores, se recomienda ingresar $m_1 = 10[kg]$ y $m_2 = 20[kg]$. Y una vez ingresadas las masas, se tendrán las cantidades necesarias para todo el cálculo. En el enunciado se pide la magnitud de la fuerza entre las distancias $0{,}1[m]$ como mínimo y $10[m]$ como máximo.

```
#Se le pide al usuario que introduzca las masas
    m1 = float(input("Ingresa la masa 1 "))
    m2 = float(input("Ingresa la masa 2 "))
    r = np.linspace(0.1, 10, 100) #r toma 100 valores equidistantes
        entre 0.1 y 10
    F = (G * (m1 * m2) ) / (r**2) #fuerza gravitacional
```

De esta manera se calculará la fuerza gravitacional entre estas dos masas y los resultados se guardarán en la lista de valores llamada F.

Para generar el gráfico, se utiliza la librería de Python *matplotlib.pyplot*, la cual se usará también para el resto de este documento. Con el cual se podrá generar el gráfico que ilustra el decaimiento de la fuerza gravitacional mientras crece la distancia (r) entre las masas anteriormente definidas.

```
    plt.plot(r, F, marker='o')
    plt.show()
```

Estas líneas harán que se genere el gráfico, el cual se debe asemejar a la Figura 2.2.

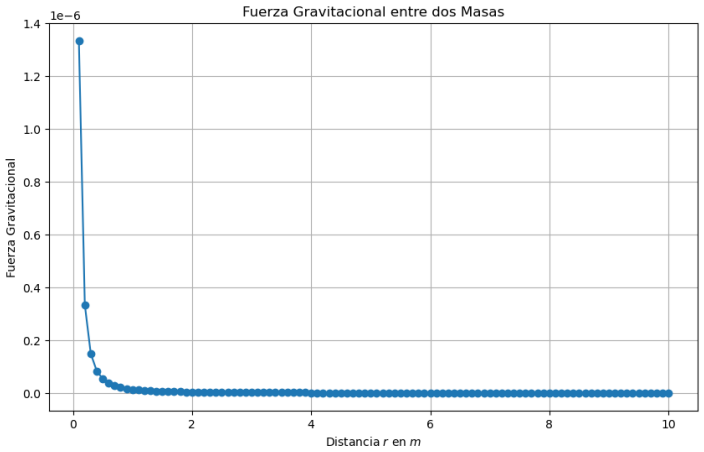

Figura 2.2: Gráfico que muestra la fuerza gravitacional de 2 masas $m_1 = 10[kg]$ y $m_2 = 20[kg]$, a lo largo de una distancia que va de 0,1$[m]$ a 10$[m]$.

En esta actividad nos pidieron graficar la fuerza gravitacional entre dos masas a través de una distancia $0{,}1 < r < 10$, se defineron las masas, las cuales deben ser suministradas al momento de correr el código, y luego el programa se encarga de graficar los valores. Esta actividad pareciera simple, pero se trabajaron las bases los procedimientos que se siguen durante todo este documento, así que es importante dominarlos y practicar con ejercicios simples, como este. En particular con este ejercicio, se puede ver como decae la fuerza gravitacional a medida de que la distancia aumenta, pero no es un decaimiento lineal, si no que adopta un decaimiento exponencial, esto se debe a que la fuerza disminuye a medida de que el cuadrado de la distancia (r^2) aumenta, esto hace que su magnitud decaiga muy rápidamente, que es lo que se puede apreciar en (2.2).

2.3. Algoritmo de Shor extrapolado

Fecha de la actividad: 10 de octubre del 2023

El problema fue basado en el Algoritmo de Shor y fue extrapolado para así descomponer números enteros entre 0 y 60 en dos factores enteros. Además fue realizado de manera individual por Fernanda Mella.

El Algoritmo de Shor es un algoritmo utilizado en computación cuántica para factorizar números enteros de gran tamaño como el producto de dos enteros menores. A partir de ello, se extrapoló para que crear un código que factorice cualquier número entero N entre 0 y 60 en dos números enteros F_1 y F_2.

Para hacerlo, se utilizó el Teorema Fundamental de la Aritmética que afirma que cada número entero mayor que 1 puede ser factorizado de manera única como un producto de números primos. Así se creará una lista de números primos para encontrar esta única factorización y a partir de ella crear una con tan solo dos factores como es descrita por el Algoritmo de Shor.

```
1  num_primos = [1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41,
       43, 47, 53, 59]
2  producto = [] #lista vacia para almacenar los factores.
3  for a in num_primos:
4      for b in num_primos:
5              for c in num_primos:
6                    for d in num_primos:
7                          for e in num_primos:
```

```
                                    if a*b*c*d*e ==N:
                                        producto = [a,b, c, d, e]  #lista
                                            con factorizacion unica
```

Luego, en la factorización única "producto.ºexistirán dos casos, que el número original N sea primo o que no lo sea. Siendo la respuesta que debe tener diferente para cada caso y descritas en 2.3. Por un lado, si N es un número primo, los dos factores serán $F_1 = N$ y $F_2 = 1$, por lo que deberá imprimir una expresión como "$N \cdot 1$". Por el otro lado, si N no es un número primo, los factores F_1 y F_2 podrán ser descritos como el producto de los factores primos dentro de "productoçomo $F_1 = a \cdot c \cdot e$ y $F_2 = b \cdot d$, por lo que deberá imprimir una expresión como $F_1 \cdot F_2$.

```
    if N in num_primos:
        print(f'{N}*1')
    else:
        F1 = producto[0]*producto[2]*producto[4]
        F2 = producto[1]*producto[3]
        print(f'{F1}*{F2}')
```

En resumen, se extrapoló el Algoritmo de Shor para un número N entre 0 y 60. En donde se aplicaron elementos básicos de Python tal que el código cumpla con el objetivo dentro del intervalo dado. También se podrá concluir que aunque originalmente el Algoritmo de Shor es de cáracter cuántico este puede ser extrapolado a computación clásica a una menor escala. Esta actividad permitió aprender sobre los algoritmos comúnmente usados en la computación cuántica y también repasar las bases del lenguaje Python.

2.4. Derivada de un polinomio y su error

Fecha de la actividad: 25 de octubre del 2023

Esta actividad fue empezada de manera individual por Sebastián Reyes y terminada en grupo.

Para realizar esta actividad es necesario importar los módulos de *numpy* y *matplotlib.pyplot*.

En la siguente actividad nos proporcionan una función polinomial, la cual debemos derivar numéricamnte con una derivada adelantada entre el intervalo $-2 < x < 2$, para luego calcular su error absoluto respecto al valor exacto $f'(0{,}5)$ para $10^{-20} < h < 0{,}1$.

Finalmente estimar el error de forma analítica, considerando tanto el error por truncamiento como el error por precisión numérica. Y para terminar, calcular el valor de h para el cual el error es mínimo.

La función a derivar es:

$$f(x) = -0{,}1x^4 - 0{,}15x^3 - 0{,}5x^2 - 0{,}25x + 1{,}2 \tag{2.5}$$

Traducida a Python queda de la siguiente manera:

```
    def pol(x):
        return (-0.1*x**4 - 0.15*x**3 - 0.5*x**2 - 0.25*x + 1.2)
```

Listing 2.4: Función polinomial a derivar.

2.4.1. Derivada Adelantada

La instrucción es calcular la derivada de esta función mediante una derivada adelantada, la cual se define analíticamnte como:

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{1}{2}f''(\xi^+)h \tag{2.6}$$

Donde ξ^+ es un valor entre x y $x+h$. Implementada en Python queda de la siguiente manera:

```
    def fp(f, x, h): #fp = f'(x)
        return ( f(h + x) - f(x) ) / h
```

Listing 2.5: Función derivada adelantada.

Ahora hay que definir un valor de h pequeño, el cual representa el "salto" infinitesimal entre cada valor de x, en este caso se elige $h = 0{,}01$, y los valores de x se ubican en $-2 < x < 2$

Con el fin de poder comparar los resultados, se define otra función que corresponde a la derivada analítica de la función:

```
    def df_analitica(x):
        return -0.4*x**3 - 0.45*x**2 - x - 0.25

    h = 0.01
    x = np.linspace(-2, 2, 100)

    df_val = [fp(pol, i, h) for i in x]
    df_analitica_val = [df_analitica(i) for i in x]
```

Listing 2.6: Valores de las derivadas.

Como se usan "for implícitos", los valores de ambas derivadas quedan almacenados en listas, al graficar ambas derivadas se puede ver su gran parecido, lo que evidencia la precisión de esta aproximación de la derivada.

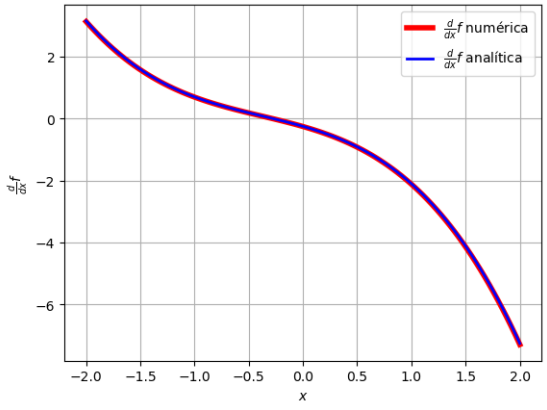

Figura 2.3: Comparación entre la derivada numérica y la derivada analítica.

Ahora, para saber con que valor de h esta presición es máxima, hay que calcular su error absoluto, para eso debemos variar el valor de h, respecto a un valor de $x = 0{,}5$. Se usará un ciclo for para graficar cada valor correspondiente a cada h en un mismo gráfico.

```
for h in np.geomspace(10e-20, 0.1, 20):
    x = 0.5
```

```
    err_abs = np.abs( df_analitica(x) - fp(pol, x, h))
    plt.loglog(h, err_abs, "o", color='blue')
```

Listing 2.7: Gráfico de valor absoluto de cada valor de h en el intervalo $10^{-20} < h < 0{,}1$.

El gráfico generado es el siguiente:

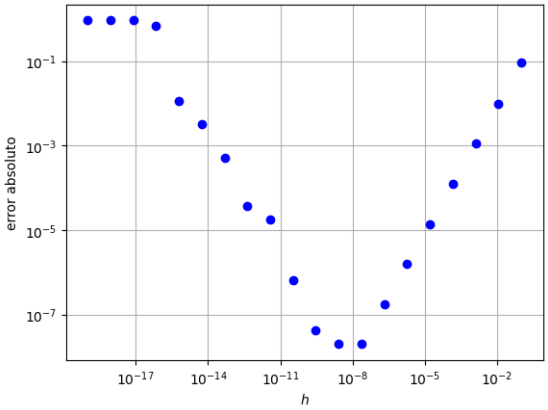

Figura 2.4: Error absoluto de la derivada numérica.

Donde se puede ver que, el error es mínimo cuando h esta cerca de 10^{-8}, por lo que se puede decir que si se quiere calcular una derivada con el mínimo error, se debe elegir $h \approx 10^{-8}$.

Ahora, para estimar el error de forma analítica, hay que considerar la forma analítica de la derivada adelantada (2.4.1), y tambien los errores de precisión numérica, lo cuales son:

$$\overline{f(x+h)} = f(x+h)(1+\epsilon_+), \qquad \overline{f(x)} = f(x)(1+\epsilon_0)$$

Reescribiendo la derivada adelantada y reemplazando los valores se tiene:

$$\begin{aligned} f'(x) - \frac{f(x+h)(1+\epsilon_+) + f(x)(1+\epsilon_0)}{h} &= f'(x) - \frac{f(x+h)+f(x)}{h} - \frac{\epsilon_+ f(x+h) + \epsilon_0 f(x)}{h} \\ &= -\frac{1}{2} f''(\xi^+)h - \frac{\epsilon_+ f(x+h) + \epsilon_0 f(x)}{h} \end{aligned}$$

Aplicamos valor absoluto en ambos lados para posteriormente usar desigualdad triangular, quedando de la siguiente forma:

$$\begin{aligned} \left| f'(x) - \frac{f(x+h)(1+\epsilon_+) + f(x)(1+\epsilon_0)}{h} \right| &= \left| -\frac{1}{2} f''(\xi^+)h - \frac{\epsilon_+ f(x+h) + \epsilon_0 f(x)}{h} \right| \\ &\leq \frac{h}{2}|f(\xi^+)| + \frac{\epsilon_+}{h}|f(x+h)| + \frac{\epsilon_0}{h}|f(x)| \end{aligned}$$

Para continuar debemos asumir que:

- Existe un valor ϵ^* tal que $\epsilon_+ < \epsilon^*$ y $\epsilon_0 < \epsilon^*$.
- $h \ll 1$ lo que hace que $\xi \simeq x$ y $|f(x+h)| \simeq |f(x)|$.

Lo que hace que el desarrollo siga como:

$$\left| f'(x) - \frac{f(x+h)(1+\epsilon_+) + f(x)(1+\epsilon_0)}{h} \right| \leq \frac{h}{2}|f(x)| + \frac{2\epsilon^*}{h}|f(x)|$$

Obteniendo así el error absoluto analítico, derivando la última expresión con respecto a h:

$$E = \frac{h}{2}|f(x)| + \frac{2\epsilon^*}{h}|f(x)| \qquad /\frac{d}{dh}$$
$$E_{abs} = \frac{|f(x)|}{2} - \frac{2\epsilon^*}{h^2}|f(x)|$$

Ahora si se quiere saber en que valor el error es mínimo, debemos igualar la expresión anterior a 0.

$$0 = \frac{|f(x)|}{2} - \frac{2\epsilon^*}{h^2}|f(x)|$$
$$\frac{|f(x)|}{2} = \frac{2\epsilon^*}{h^2}|f(x)|$$
$$h^2 = 4\epsilon^* \frac{|f(x)|}{|f(x)|} \qquad /\sqrt{}$$
$$h = \sqrt{4\epsilon^* \left|\frac{f(x)}{f(x)}\right|}$$

Obteniendo así el valor de h para el cual el error es mínimo.

En esta actividad dio una función polinomial, en donde se debio derivar numéricamente mediante una derivada adelantada, para luego calcular su error en un valor de x fijo, haciendo variar el valor de h, obteniendo su gráfica en la cual se puede ver como los puntos que corresponden a los diferentes valores de h tienen un valle, un punto mínimo, el cual se puede calcular analíticamente, tal y como se hizo anteriormente. Es curioso ver como se puede calcular una derivada con tan poco error sin siquiera saber cual es su derivada, tan sólo usando resultados que se pueden obtener mediante series de Taylor.

2.5. Manchas Solares en el tiempo

Fecha de la actividad: 23 de septiembre de 2023

Esta fue realizada por Javiera Arauco y revisada de manera grupal. Para realizar la actividad, se descargó un archivo de datos que contiene información sobre el promedio anual de manchas solares observadas en la superficie según SILSO *Sunspot Index and Long-term Solar Observations* desde el año 1770 hasta la actualidad. El objetivo será generar un programa que nos entregue el gráfico de la tasa de cambio del promedio de manchas solares como función del tiempo (en años) a partir de derivadas discretas, utilizando las herramientas de *matplotlib*, la cual es una librería completa para crear visualizaciones en Python.

2.5.1. Derivadas Discretas

Para crear este programa, primero se investigó sobre las "derivadas discretas", las cuales son una forma de calcular la razón de cambio de datos discretos en función de otro parámetro discreto (en este caso, en función del tiempo). Este tipo de derivadas se aplican a un conjunto de datos espaciados en intervalos uniformes, pero no continuos, a diferencia de las otras derivadas de cálculo que se aplican a funciones continuas. Existen diversas maneras de calcular las derivadas discretas, en este caso se hizo uso de la diferencia central: en esta técnica, se utiliza tanto el valor en el punto siguiente como el valor en el punto anterior para calcular la derivada. Su fórmula es:

$$f'(x_i) \approx \frac{f_{x_{(i+1)}} - f_{x_{(i-1)}}}{x_{(i+1)} - x_{(i-1)}}, \tag{2.7}$$

Ahora bien, para extraer los datos entregados, se importaron los módulos a utilizar: *numpy* y *matplotlib* y se utilizó una función de numpy "np.genfromtxt()". Luego, se creó el arreglo "years" extrayendo solo los datos de la primera columna de los datos entregados. También se creó el arreglo "manchas", que contiene los datos del promedio de manchas. Posterior a esto se creó un arreglo vacío llamado "derivadas" al que luego se le agregó el valor de las derivadas utilizando un ciclo "for".

```
    datos= np.genfromtxt("datos_manchas_solares.txt")
    years= datos[:,0]
    manchas= datos[:,1]

    derivada=[]
    for i in range(1, len(manchas)-1):
        df= (manchas[i+1] - manchas[i-1]) / (years[i+1] - years[i-1])
        derivada.append(df)

    plt.plot(years[1:-1], derivada)
    plt.show()
```

Listing 2.8: Código utilizado

Finalmente se generó el gráfico de la derivada con *matplotlib*, teniendo al tiempo en el eje de las abscisas y la derivada del promedio de las manchas en el eje de las ordenadas. Cabe señalar para el gráfico no se utilizó ni el primer ni el último año, ya que para estos no existe derivada discreta con diferencia central (el primer dato no posee un dato anterior, ni el último, un dato siguiente).

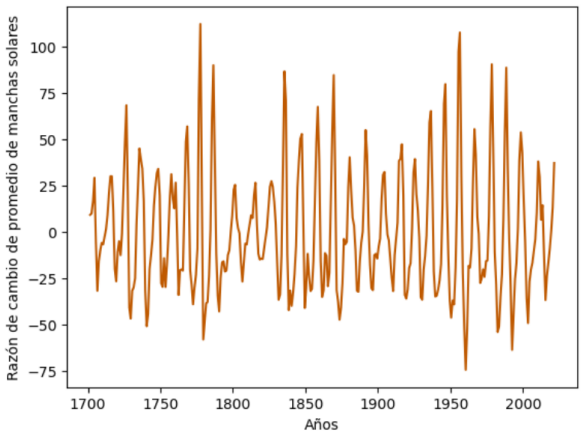

Figura 2.5: Gráfico generado.

En esta actividad se creó un programa que entregó el gráfico de la tasa promedio de cambio del promedio de manchas solares como función del tiempo, todo esto a partir de a partir de datos reales entregados por SILSO, utilizando como base matemáica el concepto de "derivadas discretas". A partir de lo anterior se puede concuir que tanto *numpy* como *matplotlib* son herramientas muy útiles para trabajar con datos externos. Por otro lado, este ejercicio fue muy bueno para trabajar con las derivadas y aprender más de ellas, pues es una herramienta matemática con la que como grupo se había trabajado. Cabe señalar que el hecho de estar trabajando ya con datos sacados de investigaciones científicas es algo que emociona y anima para seguir aprendiendo del análisis de datos utilizando métodos númericos y de programación.

2.6. Teorema fundamental del Cálculo

Fecha de la actividad: 29 de septiembre de 2023

Esta actividad fue realizada de manera individual por Fernanda Mella, además fue revisada por el grupo y por ayudantes.

En esta actividad presenta una función f y a partir de ella se debe construir su función derivada. Además se debe construir otra función que integre numéricamente la función derivada obtenida en el intervalo $0{,}06 \leq x \leq \frac{\pi}{2}$; y comparar el resultado con la función original f en el mismo intervalo.

Por lo que se comprobará el Teorema Fundamental del Cálculo para una función dada. Este teorema dirá:

$$F(x) = \int_a^x f(t)dt$$

Entonces:

$$F'(x) = f(x)$$

La función dada estará definida por:

$$F(x) = e^{-x\cos(x)} + \sin\left(\ln\left(\frac{2x}{\pi}\right)\right) - 1, \tag{2.8}$$

2.6.1. Derivada Centrada

Esta función se definió numéricamente en el código. Para luego así poder definir una función derivada, para la cual se utilizará la derivada centrada (la que presenta un error de magnitud más pequeño que el de una derivada atrasada o adelantada). Según el libro *Métodos Númericos con Python* [2] esta derivada está definida analíticamente como:

$$F'(x) \approx \frac{f(x+h) - f(x-h)}{2h}, \tag{2.9}$$

Tomando un valor de $h = 0{,}0001$. Se define la función numéricamente en el código:

```
1    def df(x): #funcion derivada centrada de f(x)
2        a = x + h
3        b = x - h
4        return (f(a) - f(b))/(2*h)
```

Listing 2.9: Definición de derivada centrada

2.6.2. Método de Simpson

Luego se definió la función integral utilizando la Regla de Simpson, la que de igual manera generalmente tiene una magnitud de error menor a otras técnicas. Esta estará definida analíticamente como:

$$\int_a^b f(x) = \frac{(b-a)}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right], \tag{2.10}$$

Por el otro lado fue definida numéricamente como:

```
def r_simpson(f, a, b): #definicion regla de simpson
    simpson = ((b-a)/6)*(f(a) + 4*f((a+b)*0.5) + f(b))
    return simpson
```

Listing 2.10: Definición Regla de Simpson

Así al tener las funciones definidas se podrá integrar la función derivada:

```
def integral(x): #uso de la regla de simpson por cada intervalo,
   generando la funcion
    lim = np.linspace(0.06, x, 100)
    integral_x = 0.0 #sera la suma con la variable "x"
    for n in range(len(lim[:-1])):
        integ += r_simpson(df, lim[n], lim[n+1])
        #regla de simpson en cada intervalo
    return integral_x
```

Listing 2.11: Integración de la función derivada

Siendo el resultado obtenido la funcion $\int_a^x f(t)dt$, esta se va a comparar con la función original $F(x)$. Y así comprobar el Teorema Fundamental del Cálculo Para poder compararlas, estas se van a graficar utilizando el módulo de Python *matplotlib*.

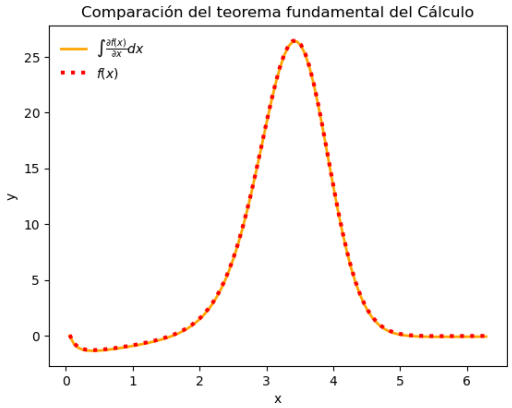

Figura 2.6: Comparación del teorema fundamental del cálculo

Así se podrá ver gráficamente como estas funciones coinciden en el intervalo. Siendo así las reglas utilizadas con errores suficientemente pequeños para que la diferencia sea despreciable, lo que permite confirmar el Teorema Fundamental del Cálculo.

En síntesis, se comprobó numéricamente el Teorema Fundamental del Cálculo graficamente. Aplicando métodos de diferenciación e integración numérica, específicamente la derivada centrada y la regla de Simpson. Además, en esta actividad se aprendío a obtener a las integales como funciones numéricas para así poder trabajar con ellas o graficarlas. Lo que es una herramienta bastante útil. Por último, lo único que no se entendió en su momento fue la redacción de la guía pues en in inicio se había hecho algo distinto a lo realmente pedido. Pero al consultar con ayudantes se pudo realizar nuevamente la actividad de manera correcta y así confirmar el teorema mencionado.

2.7. Integral numérica vs analítica

Fecha de la actividad: 21 de octubre del 2023

Esta actividad fue empezada de manera individual por Sebastián Reyes y terminada en grupo.

En este ejercicio debemos calcular integrales de manera numérica y comparar el resultado con su valor analítico.

Primera integral

La primera integral que hay que calcular y comparar con su valor analítico es:

$$\int_0^\infty \frac{1}{1+x^6} dx = \frac{\pi}{3} \tag{2.11}$$

Para poder calcular la integral de manera numérica en Python, usaremos la regla o método de Simpson, definida anteriormente de manera analítica 2.10 y de manera numérica 2.10. Se usará este método para la integración, ya que su error, en comparación a otros, es muy bajo, lo que hará de esta una muy buena aproximación de la integral.

Esta función tiene como variables (f, a, b), siendo f la funcion a integrar y a, b los límites inferior y superior de cada subintervalo, respectivamente. Para este caso particular, se elige $f(x) = 1/1 + x^6$, $a = 0$ y $b = 100$, implementado en el código queda de la siguiente forma:

```
    def funcion_int(x):
        return 1/(1 + x**6)

    limites = np.linspace(0, 100, int(1e4))
```

Listing 2.12: Primera función que se va a integrar

Se utiliza la función anterior (2.12) para evaluar los valores de f en el intervalo de integración $[0, \infty)$ y se suman cada una de las aproximaciones calculadas con el método de Simpson. Cabe señalar, el computador no es capaz de calcular infinitos valores, por lo que se aproximó el ∞ por un valor de b relativamente grande, en este caso, como ya se dijo anteriormente, $b = 100$.

```
    integ = 0
    for k in range(len(limites) - 1):
        integ += simpson(funcion_int, limites[k], limites[k+1])
```

Listing 2.13: Integración mediante la regla de Simpson. label

Si se ejecuta el programa, éste calculará la integral numérica en los límites de integración mencionados anteriormente, y su output es el siguiente:
Valor de la integral numérica $= 1.0471975511765927$

El cual es una muy buena aproximación al valor real de la integral, como se puede ver, el valor real de la integral es:
Valor de la integral analítica $= 1.0471975511965976$

Lo que confirma la precisión del método.

Segunda integral

Para esta parte, se calculan y comparán las integrales de la forma:

$$\int_0^\infty \frac{1}{1+x^{2n}}dx = \frac{\pi}{2n}\csc\frac{\pi}{2n} \tag{2.12}$$

Para $n = 1, 2, \ldots, 100$, y además se grafica cada valor para su respectivo n, en un solo gráfico.

Primero es necesario definir en python la función a integrar, de la siguente manera:

```
    def func_inte(x, n):
        return 1/(1 + (x**2 * n))
```

Listing 2.14: Segunda función a integrar

Donde sus variables son (x, n), es importante definir esta última como una variable, ya que para calcular esta integral n veces, éste valor también debe variar.

Siguendo con esta observación, también hay que agregar este parámetro a la regla de Simpson, por lo que la implementación de este método en Python será similar al anterior (2.10), solo hay que cambiar el nombre de la función y agregar el parámetro n.

```
    def simpson_dosvar(f, a, b, n):
        simpson = ((b-a)/6)*(f(a,n) + 4*f((a+b)*0.5 ,n) + f(b,n))
        return simpson
```

Listing 2.15: Regla de Simpson para dos variables

Ahora, como en la actividad piden calcular 100 integrales (una para cada valor de n), hay que usar 2 ciclos for, uno para calcular cada integral mediante la regla de Simpson, y el otro para calcular las n integrales, implementado en Python queda de la siguente forma:

```
    N_val = np.arange(1, 101) #valores de n
    int_val = [] #En esta lista se guardan lo valores de cada integral

    for n in N_val: #El encargado de calcular las n integrales
        integ = 0
        for k in range(len(limites) - 1): #El encargado de calcular
            cada integral
            integ += simpson_dosvar(func_inte, limites[k], limites[k
                +1], n)
        int_val.append(integ) #Guarda cada resultado en la lista vacia
            "int_val"
```

Listing 2.16: Cálculo de 100 integrales usando ciclos for

Con los valores de la integral numérica calculados, solo queda calcular los valores de la integral analítica, se hará usando un for implícito de la siguente forma:

```
    analit_val = [(np.pi/(2*n))*(1/np.sin(np.pi/(2*n))) for n in N_val]
```

Listing 2.17: Valores de la integral analítica

Esto creará una lista que contiene todos los valores de la integral analítica para cada valor de n.

Teniendo todo esto listo, el programa puede graficar los valores, demostrando que para $n = 1, 2, \ldots, 100$, los valores de la integral numérica con la analítica coinciden, o al menos la integral numérica se acerca bastante.

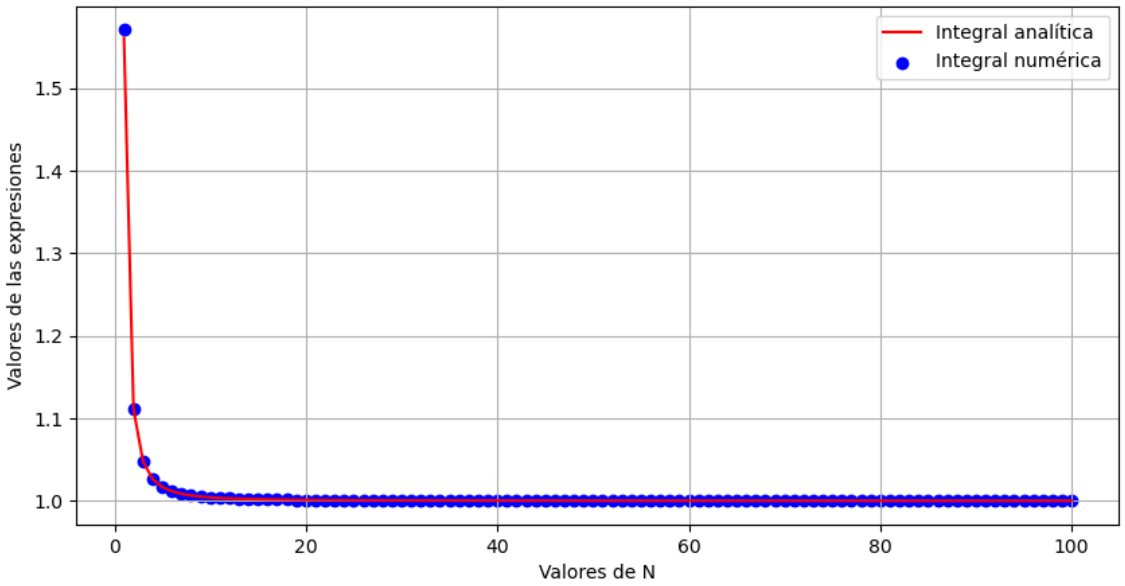

Figura 2.7: Gráfico correspondiente al cálculo de la integral numérica y la integral analítica para cada n.

En este actividad se calcularon 2 integrales y se compararon con su respectivos valores exactos, para la primera integral era $\frac{\pi}{3}$ y para la segunda $\frac{\pi}{2n} \cdot \csc\left(\frac{\pi}{2n}\right)$, la cual debe coincidir para cada valor de n en el intervalo $[1, 100]$. De la manera en que se hicieron las integrales, y los valores analíticos y exactos coincidieron, con un pequeño grado de error que puede considerarse despreciable. Esta manera de calcular integrales definidas o indefinidas es muy útil, puesto que existen integrales en las que no es posible calcular una solución analítica, pero con este método aún asi se puede saber el valor a la cual se debe llegar, y así continuar con cualquier procedimiento o proceso que se esté llevando a cabo.

2.8. Funciones de Bessel

Fecha de la actividad: 23 de septiembre de 2023

Esta actividad fue realizada de manera individual por Javiera Arauco y revisada por el grupo. El objetivo de esta actividad es crear un programa que grafique las primeras 10 funciones de Bessel.

Las funciones de Bessel son un conjunto de funciones matemáticas especiales que se utilizan mucho en la física, especialmente en el estudio de problemas con simetría cilíndrica o circular. En este ejercicio se pide trabajar con las funciones de Bessel de primera especie, denotadas como $J_n(x)$. Estas funciones se utilizan para describir fenómenos en sistemas con simetría circular o cilíndrica, y/o fenómenomenos oscilatorios. Las funciones de Bessel de n-ésimo orden se pueden obtener de la siguiente manera:

$$J_n(x) = \frac{1}{\pi} \int_0^{\pi} \cos(x \sin(\theta) - n\theta) d\theta, \tag{2.13}$$

donde n es un número entero.

Ahora bien, para graficar las funciones, primero se define la función a integrar en Python:

```
    def bessel(theta,x,n):
        return (1/np.pi)*(np.cos(x*np.sin(theta)-n*theta))
```

Listing 2.18: Definición de la funcón a integrar

Luego, se definió la regla de Simpson 2.10, la cuál es un método para calcular de forma aproximada integrales definidas. Es importante señalar que se escogió este método debido a que entrega un valor más cercano al valor real de la integral, es decir, su error es menor en comparación a otros métodos. Numéricamente se definió en 2.10.

Ahora bien, es necesario que esta regla sea aplicada a todos los intervalos en que se separará la función definida anteriormente.

```
    def integral(x,n):
        limites = np.linspace(a,b,1000)
        suma = 0
        for i in range(len(limites)-1):
            suma = suma + regla_de_simpson(bessel,limites[i],limites[i
               +1],x,n)
        return suma
```

Listing 2.19: Integración de la función con respecto a θ

Aquí se tiene que $a = 0$ y $b = \pi$ son los límites de integración y fueron definidos anteriormente en el código.

Finalmente, se grafican las funciones de Bessel como sigue:

```
    x = np.arange(0,30,0.1)

    for t in range(10):
        plt.plot(x , integral(x, n=t), label = f"$n={t}$")

    plt.xlabel("$x$")
    plt.ylabel("$J_n(x)$")
    plt.legend(loc="upper␣right", fontsize="small")
    plt.show()
```

Listing 2.20: Integración de la función con respecto a θ

De esta manera en un mismo gráfico se tienen las 10 primeras funciones de Bessel.

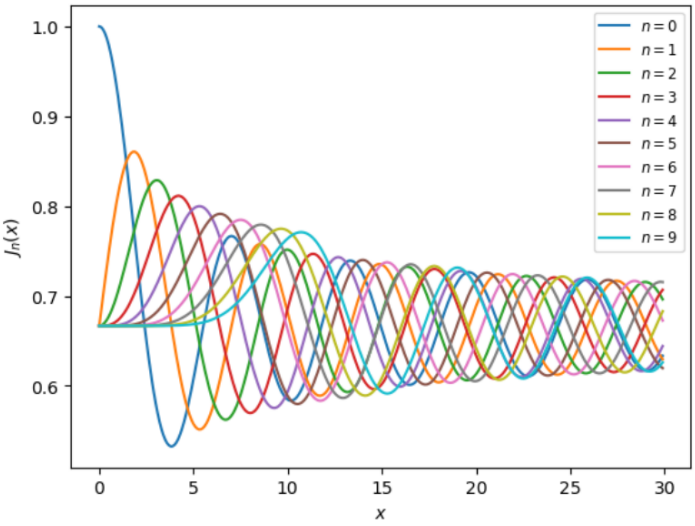

Figura 2.8: Gráfico que muestra las primeras 10 funciones de Bessel

Se creó un programa que grafica las primeras 10 funciones de Bessel de primera especie, las cuales son funciones muy utilizadas en la física, en especial para el estudio de sistemas con simetría cilíndrica o circular. Cabe señalar que se utilizó como base la regla de Simpson para dar solución a la integral. En esta actividad se logró dar solución a un problema asociado al área de la física y las matemáticas del que no se tenían conocimientos, por lo que podemos concluir que, pese a no haber tenido noción de ciertos conceptos más complejos de matemáticas, al saber aplicar de manera correcta los métodos numéricos de cuadraturas vistos previamente en clases y ayudantías, podemos de igual manera dar solución a los ejercicios planteados. Como se mencionó anteriormente, en ocación se utilizó la Regla de Simpson, herramienta que sirve para calcular integrales de forma numérica, pero con un error más bajo en comparación a otros métodos. Al comparar el resultado obtenido con el real, es posible confirmar la afirmación anterior, pues la gráfica obtenida es, a simple vista, igual a la gráfica real de las primeras 10 funciones de Bessel.

2.9. Péndulo Simple

Fecha de la actividad: 9 de octubre de 2023

Esta actividad fue realizada de manera grupal, también fue realizada por el ayudante durante la ayudantía. La actividad consiste en analizar el funcionamiento de un péndulo simple.

En este ejercicio se presenta una Ecuación Diferencial Ordinaria (EDO) que representa a la ecuación de movimiento de un péndulo. La cual se define como:

$$\frac{d^2\theta}{dt^2} + \frac{g}{l} sin(\theta) = 0, \tag{2.14}$$

Donde g es la gravedad, l es el largo de la cuerda y θ es el ángulo entre el péndulo y el eje vertical. Como es representado en 2.9.

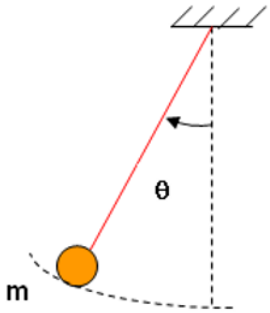

Figura 2.9: Representación gráfica de la situación.

Luego la actividad pide que si $g/l = 1$, definir $\omega = \theta'$ y conjunto a las condiciones iniciales $\theta(0) = \pi/6$ y $\theta'(0) = 0$ resolver la EDO dada.

Para enfrentar este problema se va a reducir la EDO de segundo orden a una de primer orden. Así se reducirá desde:

$$\frac{d\theta}{dt} = \omega$$
$$\frac{d\omega}{dt} = -sin(\theta),$$

utilizando una función vectorial definida como $\vec{X} = (\theta(t), \omega(t))$. Para así definir solo una EDO de primer orden dada por:

$$\vec{f}(\vec{X}, t) = \frac{d\vec{X}}{dt} \tag{2.15}$$
$$= (\omega(t), -\operatorname{sen}(\theta(t))). \tag{2.16}$$

2.9.1. Método de Euler

Se podrá utilizar el Método de Euler, el cual a partir del libro *Numerical Methods In Physics With Phyton* [3] se podrá definir como:

$$X_{i+1} \approx X_i + \Delta t \cdot f(X_i, i\Delta t) \tag{2.17}$$

Siendo esta la expresión a utilizar para resolver la EDO propuesta. Para colocarlo numéricamente en el código, se importaron los módulos *numpy* y *matplotlib.pyplot*, para poder calcular numéricamente y graficar el resultado respectivamente. Así, en un inicio se definió el intervalo de la variable independiente (t), conjunto a la función vectorial y las condiciones iniciales de $\theta(0)$ y $\omega(0)$.

```
t = dt*np.arange(0,N) # intervalos de tiempo

def f(x,t): #definicion funcion vectorial
    0, w = x
    return np.array([w, -np.sin(0)])

x[0] = np.pi/6 , 0 #condiciones iniciales
```

Luego se podrá definir el sistema de ecuaciones mostrado en (2.17) numéricamente.

```
    for i in range(N-1):
        x[i+1] = x[i] + f(x[i], i*dt)*dt

    O = x[:,0] #funcion solucion (posicion, theta)
    w = x[:,1] #velocidad angular (omega)
```

A partir de esta definición se podrá graficar a θ en función del tiempo $[t]$ utilizando el módulo *matplotlib* en 2.9.1.

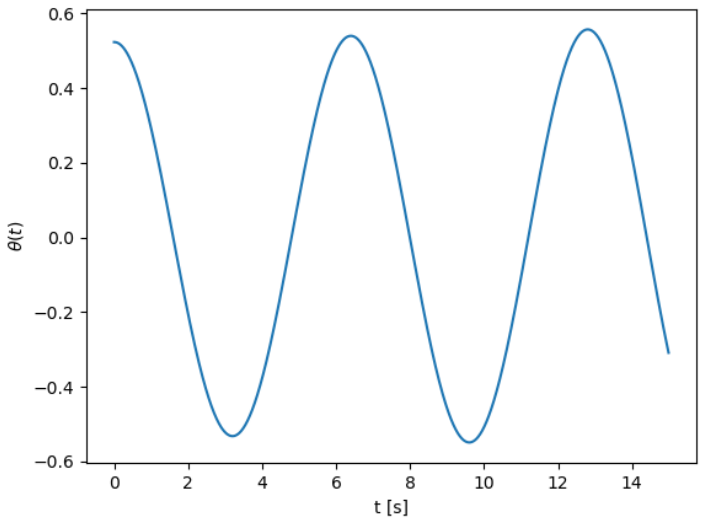

Figura 2.10: Solución $\theta(t)$ para $\theta(0) = \pi/6$.

En la segunda parte del ejercicio se pide resolver nuevamente, pero con 20 diferentes ángulos iniciales. Siendo estos pertenecientes al intervalo $[0, \pi]$.

Se utilizará el mismo Método de Euler para definir el sistema de ecuaciones a utilizar para resolver la EDO. A diferencia de la primera parte, en este apartado se deberán definir a diferentes ángulos iniciales, los que son definidos por:

```
    O_0 = np.linspace(0, np.pi, 20)
    for k in range(len(O_0)):
        N_2 = 10000 #numero maximo
        dt2 = 0.001 #intervalo entre los tiempos
        t2 = dt2*np.arange(0,N_2) #intervalo de tiempo
        x_2 = np.zeros([N_2, 2])
        x_2[0] = O_02[k], 0.0 #condiciones iniciales
```

Listing 2.21: Definición de ángulos iniciales.

Dentro del ciclo "for"mostrado, se define el sistema de ecuaciones a utilizar para la función:

```
        for i in range(N_2-1):
```

```
        x_2[i+1] = x_2[i] + f(x_2[i], i*dt2)*dt2

        O_2 = x_2[:,0] #posicion
        w_2 = x_2[:,1] #velocidad
        plt.plot(t2,O_2)
    plt.show()
```

Listing 2.22: Regla de Euler para f(x)

De manera que al tener las funciones definidas. Se podrán graficar en un solo gráfico utilizando "plt.plot(t, θ)"dentro del ciclo for, para que este comando guarde la información de cada uno de los ángulos iniciales. Y por último usar el comando "plt.show()"fuera del ciclo for, para que muestre solo un gráfico. El que será el siguiente:

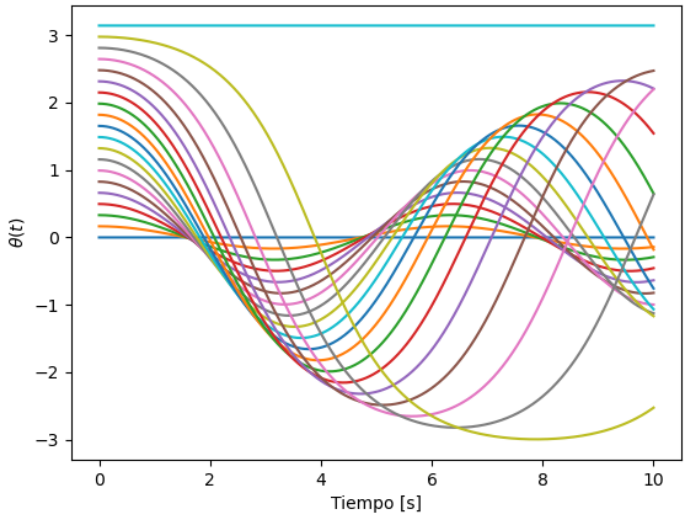

Figura 2.11: Gráfica de los 20 ángulos.

Por lo que se podrán analizar graficamente como fluctúa $\theta(t)$ para distintas condiciones iniciales.

2.9.2. Análisis de Energía y Espacio de Fase

A partir de las expresiones de $\theta(t)$ y $\omega(t)$ se podrán analizar propiedades físicas del sistema tal y como la energía. En donde se podrá obtener su expresión analítica:

$$\ddot{\theta}(t) + \frac{g}{l}\,\text{sen}(\theta(t)) = 0 \tag{2.18}$$

$$\dot{\theta}(t)\cdot\ddot{\theta}(t) + \dot{\theta}(t)\frac{g}{l}\,\text{sen}(\theta(t)) = 0 \tag{2.19}$$

$$\frac{d}{dt}\left[\frac{1}{2}\dot{\theta}(t)^2 - \frac{g}{l}\cos(\theta(t))\right] = 0 \tag{2.20}$$

$$\tag{2.21}$$

A partir de esto se definirá a la energía en este sistema como:

$$E = \frac{1}{2}\dot{\theta}(t)^2 - \frac{g}{l}\cos(\theta(t)) \tag{2.22}$$

$$E = \text{constante} \tag{2.23}$$

Por lo que esta cumplirá con el principio de conservación de energía.

Esta expresión se podrá graficar en función del tiempo para mejorar su análisis. Se definirá numéricamente en 2.23 y se podrá ver su solución graficamente en la Figura 2.12.

```
    E = [(-np.cos(O[i]) + 0.5*(w[i])**2) for i in range(len(t))]
    plt.plot(t, E_ev)
    plt.xlabel(r'Tiempo [s]')
    plt.ylabel(r'Energia [J]')
    plt.show()
```

Listing 2.23: Energía del sistema con método de Euler

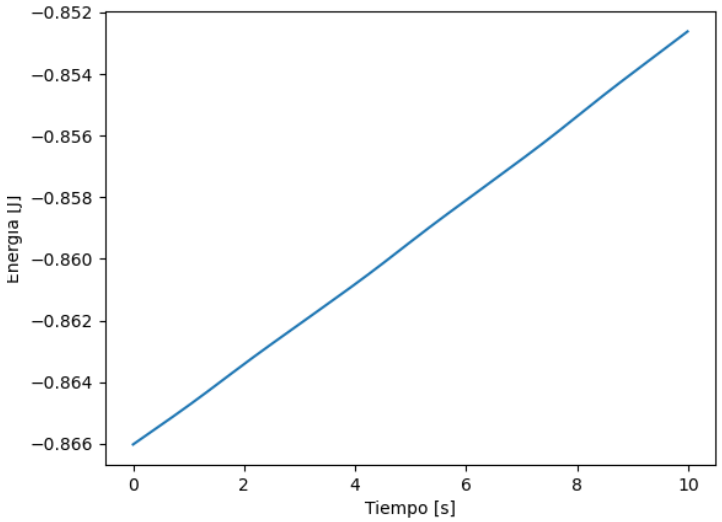

Figura 2.12: Energía vs Tiempo con el método de Euler.

A partir del análisis del gráfico se podrá decir que la energía no será conservativa, es más, esta aumentará a lo largo del tiempo lo que para esta situación es físicamente imposible. Para confirmar si la energía de esta situación es disipativa o no, se puede graficar el espacio de fase, el cual estará definido como el gráfico de $\omega(t)$ en función de $\theta(t)$. De este modo si la energía del sistema se conserva este mostrará una circunferencia cerrada, y en el caso contrario mostrará una circunferencia abierta.

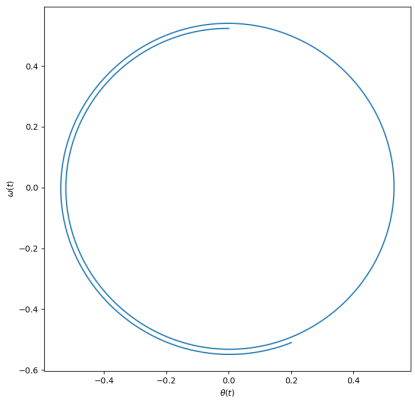

Figura 2.13: Espacio de Fase con el método de Euler.

Por lo que al ver como la circunferencia mostrada es abierta, se podrá decir que la energía del sistema será disipativa. Por lo tanto el método utilizado también será disipativo. Así aunque el método de Euler sea simple de utilizar, no será el más idóneo para el análisis de una situación física. Ante esto, para analizar de manera más acertada se deberá utilizar otro método que no sea disipativo.

2.9.3. Método de Runge-Kutta

En este caso se utilizará el Método de Runge-Kutta de Orden 4, el cual utilizará la misma función vectorial propuesta anteriormente. Este método, se definirá en *Numerical Methods In Physics With Phyton* [3] como:

$$k_1 = hf(X_n, t_n), \tag{2.24}$$

$$k_2 = hf(X_n + \frac{1}{2}k_1, t_n + \frac{h}{2}), \tag{2.25}$$

$$k_3 = hf(X_n + \frac{1}{2}k_2, t_n + \frac{h}{2}), \tag{2.26}$$

$$k_4 = hf(X_n + k_3, t_n + h). \tag{2.27}$$

$$X_{n+1} = X_n + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \tag{2.28}$$

Siendo, f la función vectorial, h el diferencia de tiempo entre un tiempo t_n y t_{n+1} y X_n la variable con el sistema de ecuaciones a resolver. Por lo que, utilizando las mismas variables definidas anteriormente se definirá el método numéricamente en 2.24. Y se podrá observar su solución graficamente como 2.9.3.

```
    def runge_kutta(f, u, t, h): #funcion Runge-Kutta general
        for n in range(len(t) - 1):
            k1 = h*f(u[n], t[n])
            k2 = h*f(u[n] + 0.5*k1, t[n] + 0.5*h)
            k3 = h*f(u[n] + 0.5*k2, t[n] + 0.5*h)
            k4 = h*f(u[n] + k3, t[n] + h)
            u[n+1] = u[n] + (1/6)*(k1 + 2*k2 + 2*k3 + k4)
        return u #sistema de ecuaciones
```

```
9       X = np.zeros([N,2])
10
11      X[0] = np.pi/6 , 0 #condiciones iniciales
12      X = runge_kutta(f,X, t, dt)
13      O_RK = X[:,0] #theta, posicion
14      w_RK = X[:,1] #omega, velocidad
```

Listing 2.24: Energía del sistema con método de Runge-Kutta

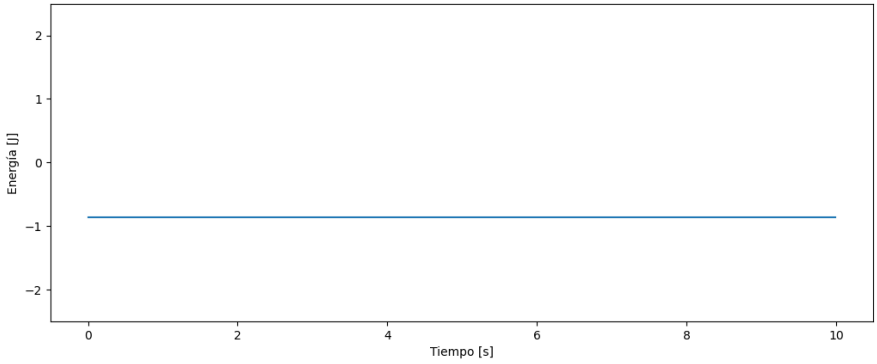

Figura 2.14: Energía a lo largo del tiempo con el método de Runge-Kutta.

También se podrá analizar al estudiar la forma de su espacio de fase en la Figura 2.15, que formará una circunferencia cerrada, lo que indica que efectivamente la solución proporcionada por este método no gana ni pierde energía. Además se podrá comparar este espacio de fase con el proporcionado por el método de Euler, y se puede ver en la separación entre las líneas lo ineficiente que fue para describir situaciones físicas. Por último también se podrá comparar las soluciones númericas obtenidas con ambos métodos en la Figura 2.16.

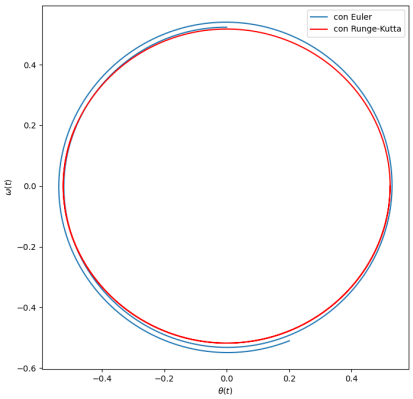

Figura 2.15: Comparación Espacios de Fase.

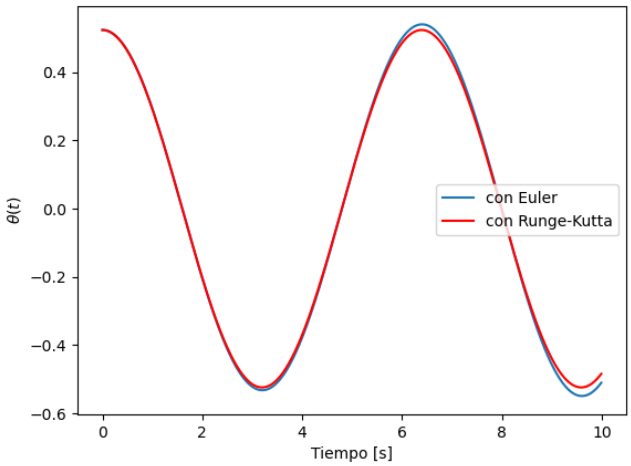

Figura 2.16: Comparación soluciones analítica.

En resumen, se realizó un análisis de un péndulo simple en donde se observó que el método utilizado originalmente, el Método de Euler, no fue acertado para el análisis de energía de este sistema físico. Por lo que se tuvo que utilizar el Método de Runge-Kutta para poder completar el análisis. Así se puede concluir que esta fue una actividad especialmente educativa para aprender ambos métodos mencionados y analizar una situación física en específico. Pero ya se menciono, el método de Euler no será certero para analizar situaciones físicas de este estilo en cambio el método Runge-Kutta será efectivo para analizar la situación del péndulo simple.

2.10. Péndulo doble

Fecha de la actividad: 22 de noviembre de 2023

El ejercicio propuesto fue realizado de manera individual por Fernanda Mella y revisado parcialmente por el grupo, ayudantes y el profesor. Esta actividad pide analizar el sistema del pédulo doble, representado en 2.10, a través de las siguientes ecuaciones diferenciales

$$(m_1+m_2)l_1^2\ddot{\theta}_1 + m_2 l_1 l_2 \ddot{\theta}_2 \cos(\theta_1-\theta_2) + m_2 l_1 l_2 \dot{\theta}_2^2 \sin(\theta_1-\theta_2) + (m_1+m_2) g l_1 \sin(\theta_1) = 0 \quad (2.29)$$

$$m_2 l_2^2 \ddot{\theta}_2 + m_2 l_1 l_2 \ddot{\theta}_1 \cos(\theta_1-\theta_2) - m_2 l_1 l_2 \dot{\theta}_1^2 \sin(\theta_1-\theta_2) + m_2 g l_2 \sin(\theta_2) = 0 \quad (2.30)$$

también llamadas Ecuaciones Diferenciales de Euler-Lagrange para los ángulos θ_1 y θ_2 respectivamente, mientras m_1 y m_2 son las masas del objeto uno y dos; y l_1 y l_2 serán los largos de las respectivas cuerdas.

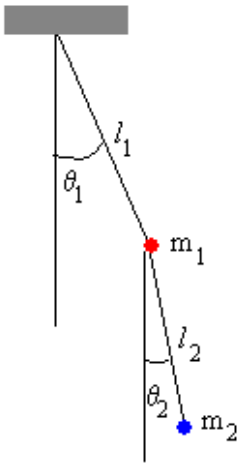

Figura 2.17: Representación Péndulo Doble.

A partir de aquí, la actividad pide reescribir el sistema anterior utilizando las siguientes sustituciones

$$\begin{aligned}
\alpha &\equiv (m_1+m_2)l_1^2, \\
\beta &\equiv m_2 l_1 l_2 \cos(\theta_1-\theta_2), \\
\gamma &\equiv m_2 l_2^2, \\
\delta &\equiv m_2 l_1 l_2 \dot{\theta}_2^2 \sin(\theta_1-\theta_2) + l_1 g (m_1+m_2)\sin(\theta_1), \\
\epsilon &\equiv -m_2 l_1 l_2 \dot{\theta}_1^2 \sin(\theta_1-\theta_2) + l_2 m_2 g \sin(\theta_2).
\end{aligned}$$

Para hacerlo, simplemente se reemplazan los términos y se llega a

$$\alpha\ddot{\theta}_1 + \beta\ddot{\theta}_2 + \delta = 0 \tag{2.31}$$

$$\gamma\ddot{\theta}_2 + \beta\ddot{\theta}_1 + \epsilon = 0. \tag{2.32}$$

Luego, se define $\omega \equiv \dot{\theta}$, debido a que piden resolver el sistema para $\dot{\omega}_1 \equiv \ddot{\theta}_1$ y para $\dot{\omega}_2 \equiv \ddot{\theta}_2$. Al hacerlo, resultaran las siguientes soluciones

$$\dot{\omega}_1 = \frac{-\beta\left(\frac{\beta\delta}{\alpha} - \epsilon\right)}{\alpha\left(\gamma - \frac{\beta^2}{\alpha}\right)} - \frac{\delta}{\alpha} \tag{2.33}$$

$$\dot{\omega}_2 = \frac{\frac{\beta\delta}{\alpha} - \epsilon}{\gamma - \frac{\beta^2}{\alpha}}. \tag{2.34}$$

Así se tendrán soluciones analíticas para $\dot{\theta}_1$, $\dot{\theta}_2$, $\dot{\omega}_1$ y $\dot{\omega}_2$. Lo que permitirá resolver el sistema 2.31 y 2.32 para $\theta_1(t)$ y $\theta_2(t)$. Específicamente en este caso, se realizará con condiciones iniciales $\theta_1(0) = \pi/2$, $\theta_2(0) = \pi/2$, $\omega_1(0) = 0$ y $\omega_2(0) = 0$. Y se utilizarán masas $m_1 = 2$, $m_2 = 1$, $l_1 = 2$, $l_2 = 5$ y la gravedad se tomará como $g = 9{,}8$.

2.10.1. Solución Numérica

Para resolver el sistema del Péndulo Doble numéricamente, se utilizo el método Runge-Kutta definido en 2.24. Para utilizarlo, se utilizaron las definiciones de $\dot{\theta}_1$, $\dot{\theta}_2$, $\dot{\omega}_1$ y $\dot{\omega}_2$ anteriormente propuestas para definir una función vectorial como:

```
    def f_vectorial(X, t, m1=m1, m2=m2, l1=l1, l2=l2):
        O1, O2, w1, w2 = X  #theta_1, theta_2, omega_1, omega_2
        a = (m1 + m2) * l1**2
        b = m2 * l1 * l2 * np.cos(O1 - O2)
        y = m2 * l2**2
        d = m2 * l1 * l2 * w2**2 * np.sin(O1 - O2) + l1 * g * (m1 + m2
            ) * np.sin(O1)
        e = -m2 * l1 * l2 * w1**2 * np.sin(O1 - O2) + l2 * m2 * g * np
            .sin(O2)
        return np.array([w1, w2, -b * ((b * d) / a - e) / (a * (y - b
            **2 / a)) - d / a, (b * d / a - e) / (y - b**2 / a)]) #
            retorna las derivadas de cada variable
```

Luego se definieron las condiciones iniciales y se aplico la función Runge Kutta definida en 2.24 para esta función.

```
    X[0] = 0.5*np.pi,0.5*np.pi, 0,0
    X = runge_kutta(f, X, t, h)
    O1 = X[:,0]
    O2 = X[:,1]
    w1 = X[:,2]
    w2 = X[:,3]
```

Así teniendo la solución esta se podrá graficar a lo largo del tiempo 2.18. En donde se podrá observar como el sistema a analizar será uno caótico, por lo que no tendrá una tendencia fija. Aún así, se comparo con gráficos con distintos valores y se determino que se parecían lo suficiente para ser considerado correcto.

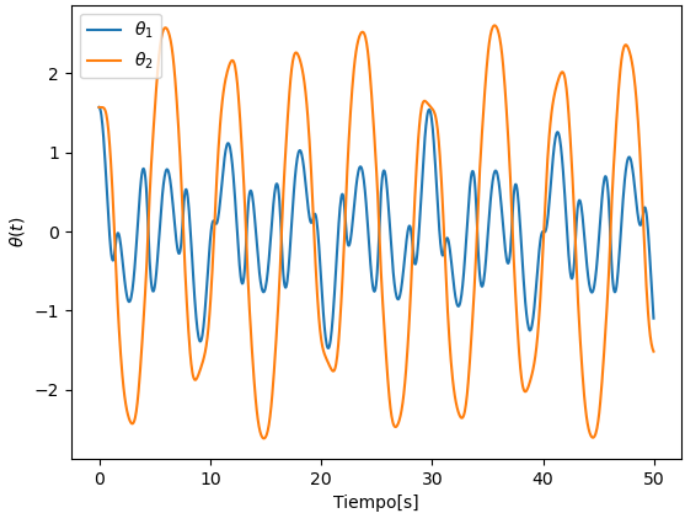

Figura 2.18: Solución númerica Péndulo Doble.

Además, esta misma solución se podrá representar mediante una animación del movimiento que va a tener el sistema en este link.

2.10.2. Análisis de Energía

Se podrá hacer un análisis más detallado de esta situación física al estudiar su energía y sus espacios de fase. Para la energía, esta se definio como la suma de la energía Cinética y Potencial de cada masa del sistema. Numéricamente se definió para este caso específico como:

```
    #Posiciones y velocidades en coordenadas cartesianas
    x1 = l1*np.sin(X[:,0])
    x2 = x1 + l2*np.sin(X[:,1])

    y1 = l1*np.cos(X[:,0])
    y2 = y1 + l2*np.cos(X[:,1])

    v1_x = l1*X[:,2]*np.cos(X[:,0])
    v2_x = v1_x + l2*X[:,3]*np.cos(X[:,1])

    v1_y = -l1*X[:,2]*np.sin(X[:,0])
    v2_y = v1_y -l2*X[:,3]*np.sin(X[:,1])

    #Energias Cineticas:
    K1 = 0.5*m1*l1**2*X[:,2]**2
    K2 = 0.5*m2*(v2_x**2 + v2_y**2)
```

```
17
18      #Energia Potencial:
19      U = m1*g*y1 + m2*g*y2
20
21      E = K1 + K2 + U
```

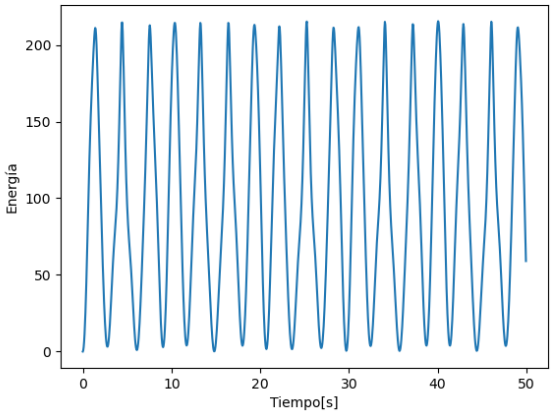

Figura 2.19: Energía Péndulo Doble con Runge Kutta.

Teniéndola definida numéricamente se puede graficar para analizarla de mejor manera en 2.10.2. Pero se observa que esta no se mantiene constante, sino que oscila. Ante esto, se realiza el mismo proceso con otros métodos numéricos para así comparar el método con una energía más constante. Específicamente se utilizó el Método del Salto de la Rana y el Método de Euler-Cromer y al tener las distintas energías, estas se compararon en 2.10.2.

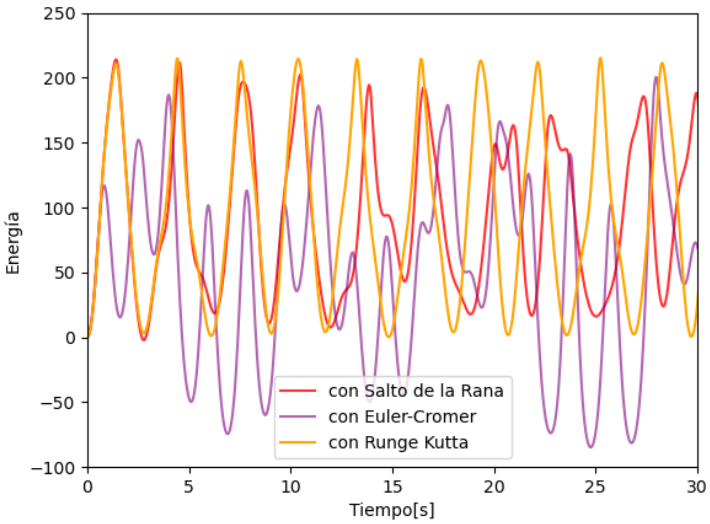

Figura 2.20: Comparación de energía con distintos métodos.

Así, se pudo observar en el gráfico que con ningún método la energía se mantiene constante, debido a que en los tres casos oscila con amplitudes altas. Sin embargo, se puede destacar que la energía obtenida con el método de Runge Kutta será la única cuya amplitud se mantiene constante. Por lo que en este caso, se priorizará la solución obtenida con ese método.

En síntesis se realizó un análisis de un péndulo doble en donde se llegó a la conclusión que la característica caótica del sistema es la que afecta al análisis de energía. Específicamente se vio como ni el método de Runge-Kutta, Euler-Cromer o Salto de Rana serán capaces de describir la solución de $\theta_1(t)$ y $\theta_2(t)$ de manera que la energía del sistema se mantenga constante. También, otro posible origen del comportamiento de la energía son las dimensiones de las variables $\dot{\omega}_1(t)$ y $\dot{\omega}_2(t)$ que podrían llegar a interferir en el análisis. Además, esta actividad fue educativa pues permitió estudiar diferentes métodos aplicados al mismo sistema físico y permitió realizar una animación de la misma situación representándola de una manera más gráfica. Por último, la principal dificultad que se tuvo fue el analísis de energía del sistema pero aún así se propusó un origen del problema.

2.11. Atractor de Lorenz

Fecha de la actividad: 09 de diciembre del 2023

Esta actividad fue hecha por el grupo.

El atractor de Lorenz es un sistema de ecuaciones diferenciales ordinarias (EDO) estudiado por primera vez por el matemático y meteorólogo Edward Lorenz en 1963. Estas ecuaciones fueron concebidas originalmente para modelar la convección atmosférica, pero ha trascendido de su aplicaión inicial, ya que han revelado propiedades intrínsecas de sistemas complejos, sobre todo un tópico bastante atrayente, el caos.

Las ecuaciones del atractor de Lorenz son:

$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = \rho x - y - xz$$
$$\frac{dz}{dt} = xy - \beta z$$

Si se quiere saber más sobre el sistema de Lorenz o más especificamente sobre el atractor de Lorenz se puede visitar las siguientes fuentes [4], [5].

En esta actividad se nos pide resolver el sistema de ecuaciones con el método de Rungue Kutta, definida anteriormente en 2.24, y traducida a un código de Python en 2.24, para luego graficar las soluciones en un gráfico 2-D y en otro 3-D.

El primer paso de esta actividad es definir la función vectorial de la siguiente forma:

```
1  def f(u, t, sigma=10, rho=28, beta=8/3):
2      x, y, z = u
3      return np.array([sigma*(y-x), rho*x - y - x*z, x*y - beta*z])
```

Listing 2.25: Ecuaciones del atractor de Lorenz en Python

Normalmente se suponen los valores de σ, ρ y β (en el código definidas como *sigma, rho*, y *beta* respectivamente) como positivos, cuando se definen los valores $\sigma = 10$, $\rho = 28$ y $\beta = 8/3$, el sistema exhibe un comportamiento caótico para estos valores y también para aquellos que se les acercan.

Para resolver este sistema de EDO, definimos una función que aplique el método de Runge Kutta, definida en 2.24, luego le proporcionamos al programa unos parámetros que son vitales para que este cálculo sea efectivo, los cuales son el tiempo máximo en que las posiciones se encontrarán, un vector vacío u de 3 columnas, las cuales guardaran las posiciones en los 3 ejes, las condiciones iniciales para cada posición en cada eje. Finalmente hacemos correr la función 2.24 para obtener las posiciones y separamos los valores de las columnas.

```
    # Tiempo final
    tmax = 30

    # Constante de paso
    h = 0.01

    # Vectores de tiempo y condiciones iniciales
    t = np.arange(0, tmax, h)
    u = np.empty([t.size ,3]) #Vector vacio

    # Condiciones iniciales
    u[:,0] = 0
    u[:,1] = 1
    u[:,2] = 1.05

    # Solucion de las ecuaciones diferenciales
    u = runge_kutta(f, u, t, h)

    x = u[:,0]
    y = u[:,1]
    z = u[:,2]
```

Listing 2.26: Cálculo de las soluciones de la EDO de Lorenz

Al graficar estos valores en un gráfico 2-D, los resultados adquieren la distribución de la Figura 2.21.

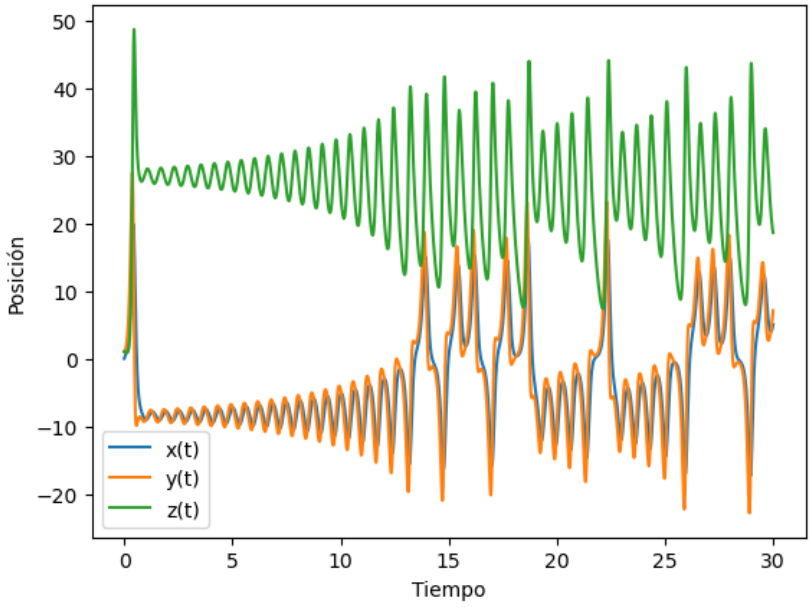

Figura 2.21: Distribución de los valores de x, y y z en el atractor de Lorenz.

Ahora, graficaremos estas soluciones en un gráfico 3-D, que se hace de una manera diferente al de los gráficos 2-D, se tiene que definir una figura vacía de tamaño a elección, la cual puede contener varios subgráficos, en nuestro caso solo va a contener 1. Luego agregamos un subgráfico tridimensional definido con el parámetro *ax*, para luego, graficar las soluciones con *ax.plot()*. Implementado en python queda de la siguente forma:

```
    #Grafica de las soluciones 3D
    fig = plt.figure(figsize=(8,8))
    ax = fig.add_subplot(111, projection='3d')
    #Graficacion de las soluciones de la EDO
    ax.plot(x, y, z, linewidth=0.5, label='Solucion 3D')
```

La gráfica en cuestion es la que se puede apreciar en la Figura 2.22.

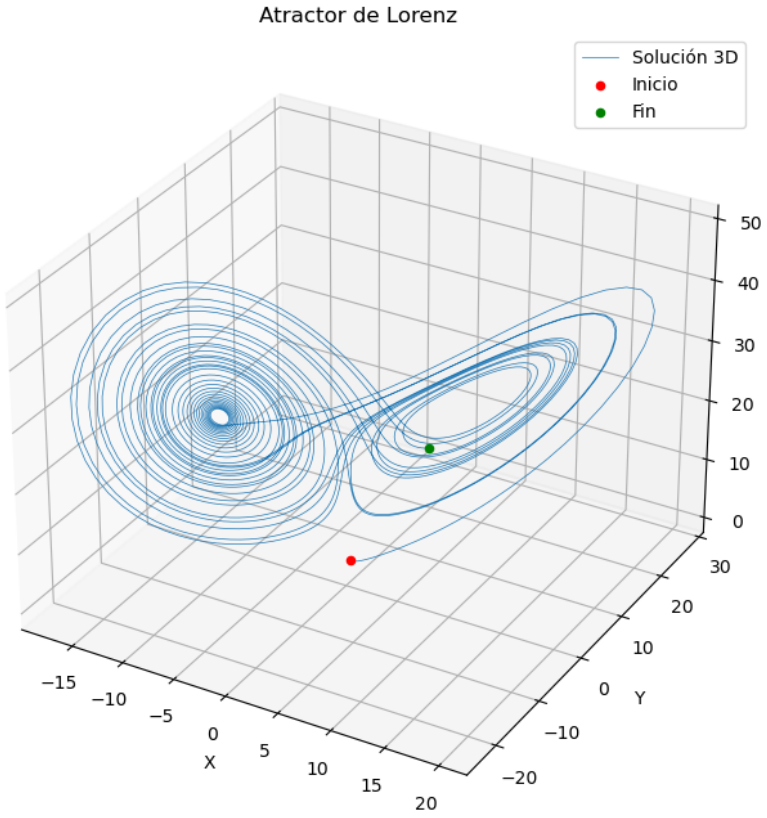

Figura 2.22: Distribución de los valores de x, y y z en el atractor de Lorenz.

Si se quiere ver como evoluciona la gráfica dependiendo de los valores de σ, ρ y β, podemos visitar *Interactive Lorenz Attractor* [6].

En esta actividad se dió a conocer el atractor de Lorenz, producto de unas ecuaciones que originalmente se concibieron para modelar la convección atmosférica, pero que presentaron resultados interesantes en el estudio sistemas complejos, en particular el caos. Se usó el método que ya definimos y usamos en una actividad anterior, también usamos un método de graficación diferente, la graficación 3-D. Es interesante como un simple sistema de EDO puede modelas el sistema más famoso para explicar el caos, y que con la mínima variación en las condiciones iniciales o con variar solo un poco los parámetros se puede llegar a resultados tan diferentes, como podemos ver en [6], si se sueltan unas partículas y dejamos que sigan el trayecto de las soluciones de la EDO, dos partículas (en este caso mariposas), que estan relativamente cerca adoptan trayectorias completamente diferentes a medida de que el tiempo avanza.

2.12. Ecuaciones de Lane-Emden

Fecha de la actividad: 20 de octubre de 2023

La actividad fue realizada de manera individual por Javiera Arauco y revisada por el grupo. El objetivo de esta actividad es crear un programa que grafique las soluciones de la ecuación de Lane-Emden.

La ecuación de Lane-Emden es una ecuación diferencial utilizada para describir la estructura de las estrellas en equilibrio electroestático, es decir, estrellas que no están experimentando cambios sig-

nificativos en su estructura y que están sujetas a fuerzas gravitatorias y presión interna. Esta ecuación es utilizada específicamente para modelar aquellas estrellas cuya temperatura interna es más o menos constante, es decir, son isotérmicas.

La ecuación de Lane-Emden se ve de la siguiente manera:

$$\frac{1}{\xi^2}\frac{d}{d\xi}\left(\xi^2\frac{d\theta}{d\xi}\right)+\theta^n=0, \tag{2.35}$$

Donde ξ es la variable de Lane-Emden, que está asociada a la densidad y a la presión al interior de la estrella, θ es una función adimensional que describe la densidad en función de ξ y n es un exponente adimensional que depende de la ecuación de estado con la que se trabaje.

Ahora bien, para graficar las soluciones de esta ecuación, con n variando entre 0 y 10, primero se resuelve la ecuación como sigue:

$$\begin{aligned}
\frac{1}{\xi^2}\frac{d}{d\xi}\left(\xi^2\frac{d\theta}{d\xi}\right) &= -\theta^n \\
\frac{1}{\xi^2}\left(2\xi\frac{d\theta}{d\xi}+\xi^2\frac{d^2\theta}{d\xi^2}\right) &= -\theta^n \\
\frac{2}{\xi}\frac{d\theta}{d\xi}+\frac{d^2\theta}{d\xi^2} &= -\theta^n \\
\frac{2}{\xi}\dot{\theta}+\ddot{\theta} &= -\theta^n
\end{aligned}$$

Luego, para generar el gráfico se resuelve numéricamente la ecuación diferencial se utiliza el método de Euler definido en (2.17). Para ello, primero se reduce la ecuación a dos ecuaciones de primer orden:

$$\begin{aligned}
\dot{\theta} &= w(\xi) \quad \textit{Se define de esta manera.} \\
\ddot{\theta} &= -\theta^n-\frac{2}{\xi}\dot{\theta}.
\end{aligned}$$

En el caso de la ecuación de Lane-Emden, para el método las variables serán:

$$\begin{aligned}
X &= \begin{pmatrix}\theta\\ \dot{\theta}\end{pmatrix} = \begin{pmatrix}\theta\\ w(\xi)\end{pmatrix} \\
f(X,t) &= \begin{pmatrix}\dot{\theta}\\ \ddot{\theta}\end{pmatrix} = \begin{pmatrix}w(\xi)\\ -\theta^n-\frac{2}{\xi}\dot{\theta}\end{pmatrix}
\end{aligned}$$

y aplicándolo, explícitamente se tiene:

$$\begin{aligned}
\theta_{n+1} &= \theta_n+\Delta\xi(w_n) \\
w_{n+1} &= w_n+\Delta\xi(-\theta^n-\frac{2}{\xi}w_n)
\end{aligned}$$

Así, en Python se define:

```
    for i in range(len(n_valores)):
        n = n_valores[i]
        theta[0]= 1
        w[0]= 0
        for j in range(len(xi)-1):
            theta[j+1] = theta[j] + w[j]*d_xi
            w[j+1] = w[j] - (theta[j]**n + (2/xi[j])*w[j])*d_xi
        plt.plot(xi, theta, label = f"$n = {n}$")
```

Listing 2.27: Método de Euler aplicado a la ecuación de Lane-Emden

Como se puede apreciar en el código, se incluyó el método de Euler dentro de un ciclo "for". Esto se hizo para que se graficaran las soluciones considerando a n como un número entero que va de 0 a 10. En el código también se pueden ver las condiciones iniciales de las variables que se propusieron en la ayudantía, $\theta_0 = 1$ y $\dot{\theta}_0 = w_0 = 0$. Además, es importante señalar que anteriormente se había definido tanto θ como w como un arreglo de ceros usando la función de numpy "np.zeros", y la variable "xi"se definió como un arreglo que iba de 0,001 a 10 con paso d_xi = 0,1.

```
d_xi = 0.1
xi = np.arange(0.001,11,d_xi)

theta = np.zeros(len(xi))
w = np.zeros(len(xi))

n_valores = np.arange(11)
```

Listing 2.28: Variables utilizadas en el código

Finalmente, se grafican las soluciones de la ecuación de Lane-Emden.

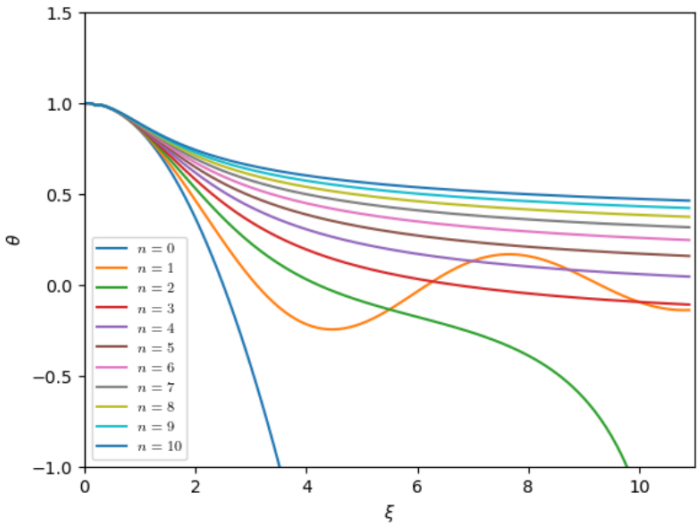

Figura 2.23: Gráfico que muestra las soluciones de la ecuación de Lane-Emden con n un número entero que va de 0 a 10

Se hizo un programa que grafica las soluciones de la ecuación diferencial de Lane-Emden, la cual es utilizada para describir la estructura de las estrellas en equilibrio electroestático. Para crear este

programa se utilizó el método de Euler. De esta actividad se puede concluir el método de Euler, pese a ser bastante sencillo y fácil de entender, es una herramienta muy útil para encontrar las soluciones de ecuaciones diferenciales ordinarias a problemas más avanzados de física. En definitiva la actividad, al plantear un problema físico interesante, del cual no se tenían conocimientos previos, anima a seguir aprendiendo de programación y métodos numéricos.

2.13. Búsqueda de ceros

Fecha de la actividad: 30 de noviembre de 2023

La actividad fue realizada de manera individual por Javiera Arauco y revisada por el grupo. El objetivo de esta actividad es crear un programa que encuentre las soluciones de 3 ecuaciones distintas.

Se pide encontrar las soluciones de las siguientes ecuaciones:

$$x^2 + x = 12 \quad -5 < x < 4 \tag{2.36}$$

$$sin(x) + 2x = 1 \quad -2 < x < 2 \tag{2.37}$$

$$tan(x) = \frac{1}{x} \quad \frac{-\pi}{2} < x < \frac{\pi}{2} \tag{2.38}$$

Ahora bien, para encontrar la solución de la primera ecuación, primero se iguala la ecuación a cero, es decir,

$$x^2 + x - 12 = 0. \tag{2.39}$$

2.13.1. Método de Bisección

De esta manera, para encontrar las raíces de la ecuación, basta con definir una función $f(x) = x^2 + x - 12$ y buscar sus ceros. Para hacerlo se deberá utilizar el método de Bisección el cual nace del Teorema que dice: "Si en un intervalo $x \in [a, b]$, se da que $f(a) \cdot f(b) < 0$, entonces existirá un $c \in [a, b]$ tal que $f(c) = 0$". De manera que este método itera esta teorema para pequeños intervalos dentro de $[a, b]$, y buscando aquellos intervalos en donde la condición $f(a) \cdot f(b) < 0$ para que así c exista y pueda ser encontrado. Este método se definirá numéricamente como:

```
1  def metodo_biseccion(f, a, b, tolerancia=0.0001):
2  cero = 0.5 * (a + b)
3      while np.abs(b - a) > tolerancia or np.abs(f(cero)) >
           tolerancia:
4      cero = 0.5 * (a + b)
5          if f(a) * f(cero) < 0:
6              b = cero
7          else:
8              a = cero
9      return cero
```

Listing 2.29: Definición del método de bisección

Aquí, f es la función, a y b son los extremos de los intervalos en donde se buscan los ceros y "tolerancia.[es] la distancia que hay entre a y b en que se termina el ciclo "while". Al ser este un número pequeño, se considera el promedio de estos como la solución. Posterior a esto se grafica la función $f(x)$. Al ver el gráfico se puede aproximar entre qué valores se encuentran los ceros de la función, es decir, se puede definir a y b.

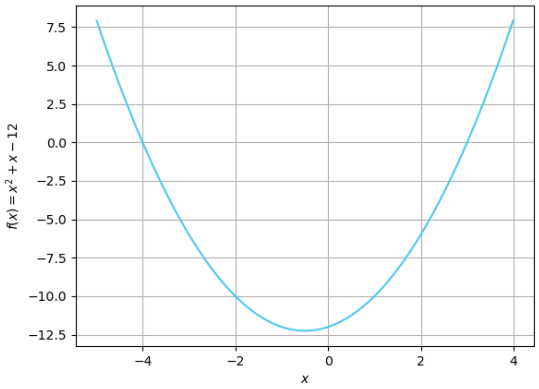

Figura 2.24: Gráfico de la función.

En el caso de la Figura 2.27, se puede ver que hay dos ceros, uno entre 2,5 y 3,5 y otro entre −4,5 y −3,5. Luego, se aplica el método de bisección para que el programa entregue las soluciones de la ecuación.

```
a1, b1 = 2, 3.5
a2, b2 = -4.5, -3

resultado1 = metodo_biseccion(f1, a1, b1)
resultado2 = metodo_biseccion(f1, a2, b2)
print(resultado1)
print(resultado2)
```

Listing 2.30: Se aplica el método de bisección

Para encontrar las soluciones de la otras ecuaciones se realiza el mismo procedimiento.

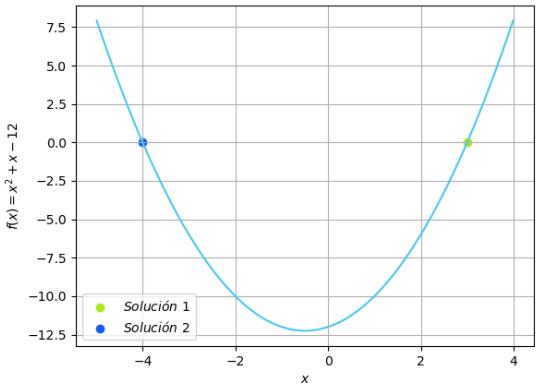

Figura 2.25: Gráfico de la primera función con sus ceros

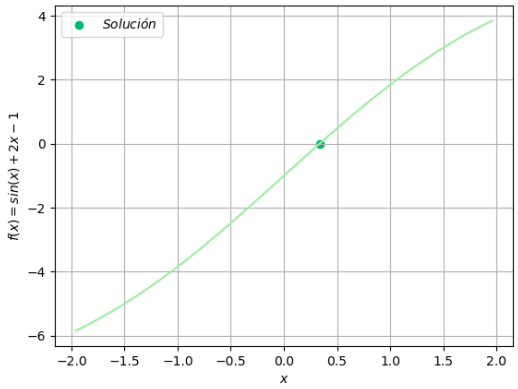

Figura 2.26: Gráfico de la segunda función con sus ceros

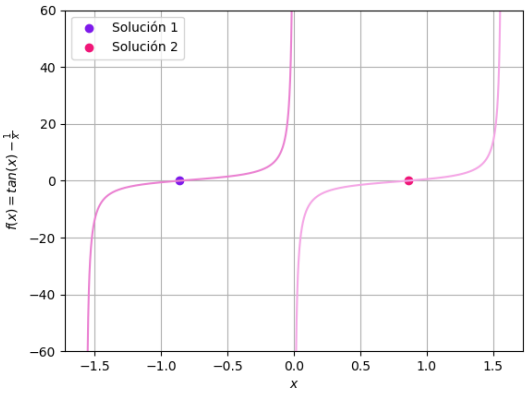

Figura 2.27: Gráfico de la tercera función con sus ceros

Se creó un programa que permite encontrar las soluciones de distintas ecuaciones. Para ello primero se definieron funciones a partir de las ecuaciones y, por medio del método de bisección, se encontraron los ceros de esas funciones, los cuales serían aproximadamente las soluciones de las ecuaciones presentadas inicialmente. Se puede concluir que el método de la bisección es efectivo, pues sí se logró encontrar las soluciones de las ecuaciones presentadas, sin embargo, no fue definido de una manera muy conveniente, pues en el caso de que se necesitara encontrar las soluciones de ecuaciones que tienen más de 3 raíces, se debería aplicar el método más de 3 veces, lo que resulta una tarea algo tediosa, por lo que se pretende optimizar este programa en un futuro.

2.14. Ceros de las funciones de Bessel

Fecha de la actividad: 04 de noviembre de 2023

Esta actividad se realizó de manera individual por Fernanda Mella y fue revisada por el profesor y el grupo. En donde se pide, graficar las funciones de Bessel definidas anteriormente en 2.13 y encontrar los ceros para n entre 0 y 10 para $x \in [0, 20]$.

2.14.1. Método de Bisección Mejorado

A partir de la definición del método de bisección propuesta en 2.29, se definirá una alternativa más eficiente a este método, la cual se definirá numéricamente como:

```
    def biseccion(f, a, b, N=100, *args, **kwargs):
        x = np.linspace(a, b, N)
        F = f(x, *args, **kwargs)
        dF = F[1:]*F[:-1]

        i = np.where(dF<0)[0]
        return x[i]
```

En donde se usan las variables *"*args"* y *"**kwargs"* para hacer a la función Bisección capaz de admitir funciones con una mayor cantidad de variables, como lo son las funciones de Bessel. Este método se aplicará para la función numérica de Bessel definida en 2.18 con n entre 0 y 10, para así encontrar los ceros de cada función generada.

```
    ceros = biseccion(integral,0,20,N=1000)
```

Los cuales se podrán observar en 2.28, por lo que se logró encontrar estos ceros de manera efectiva utilizando el método de la Bisección.

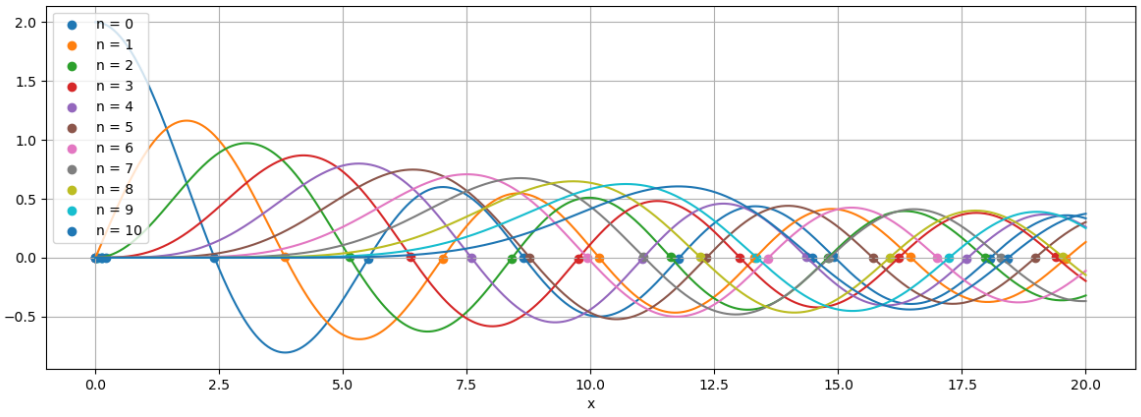

Figura 2.28: Ceros de la Función de Bessel.

En resumen, se encontraron los ceros de las funciones de Bessel para n entre 0 y 10 utilizando el Método de Bisección. Además, se puede concluir que esta fue una actividad relativamente simple, la cual permitió aprender sobre el método de la Bisección y sobre ciertas variables "*args" y "**kwargs" que permiten hacer funciones de los métodos que se utilicen más generales. Para que estas sean capaces de admitir cualquier función que se le entregue independiente de la cantidad de variables que esta tenga. Por último, lo más díficil de esta actividad no fue ningún problema con el método ni con la definición de las funciones sino que fue un percance que se tuvo al realizarla; pues al hacerla utilizando un archivo Jupyter Notebook se confundieron las funciones del método de Bisección, pero al reiniciar el Kernel se solucionó.

2.15. Ley de Planck

Fecha de la actividad: 29 de noviembre del 2023

Esta actividad fue hecha de manera individual.

La Ley de Planck, fue propuesta por el físico alemán Max Planck en 1900, la cual describe la distribución de energía de la radiación electromagnética emitida por un cuerpo negro, un objeto idealizado idealizado que absorbe toda la radiación que incide sobre él, el cual está en equilibrio termico a una temperatura determinada. Es una de las leyes más importantes de la física cuántica.

Esta ley expresada mediante la función de distribución de espectral de energía, definida como $B_\nu(T)$, que describe la cantidad de radiación emitida por unidad de área. La relación, en donde T es la temperatura del cuerpo, ν la frecuencia de la radiación electromagnética, k_B la constante de Boltzman, h la constante de Planck y la velocidad de la luz c, está ecapsulada en la siguiente ecuación:

$$B_\nu(T) = \frac{2h\nu^3}{c^2}\frac{1}{e^{h\nu/k_BT}-1} \tag{2.40}$$

Como ya se dijo antes, esta ecuación describe como la radiación electromagnética varía con la temperatura y la frecuencia, que es esencial en el entendimiento de fenómenos como la emisión térmica de objetos. Si se quiere profundizar más sobre esta ley, puede visitar las referencias [7], [8].

En esta actividad adimensionalizaremos la Ley de Planck, transformándola en una forma que depende únicamente de la razón entre energías ($x = \frac{h\nu}{k_BT}$). Además implementaremos un programa en Python para encontrar la inversa de la ley adimencionalizada y graficaremos sus soluciones. Este enfoque práctico nos permitirá practicar y profundizar métodos numéricos como el método de la bisección, definido anteriormente en 2.14.1.

Lo primero que se hará es adimencionalizar la Ley de Planck, para eso seguieremos el siguiente procedimiento:

$$\begin{aligned} B_\nu(T) &= \frac{2h\nu^3}{c^2}\frac{1}{e^{h\nu/k_BT}-1} \qquad /Reemplazamos \quad x = \frac{h\nu}{k_BT} \\ &= \frac{2h}{c^2}\left(\frac{xk_BT}{h}\right)^3\frac{1}{e^x-1} \\ &= \frac{2k_B^3T^3}{c^2h^2}\frac{x^3}{e^x-1} \\ \frac{B_\nu(T)c^2h^2}{2k_B^3T^3} &= \frac{x^3}{e^x-1} \end{aligned}$$

Definimos toda la parte izquierda como $\frac{B_\nu(T)c^2h^2}{2k_B^3T^3} = b(x)$, por lo que la Ley de Planck queda de la siguiente forma:

$$b(x) = \frac{x^3}{e^3-1} \tag{2.41}$$

Con esta versión simplificada de la ley de Planck, es posible continuar con la actividad, que es encontrar la inversa de esta función, es decir, $x = h\nu/k_BT$ como función de $0{,}1 < b(x) < 1{,}4$. Para hacer esto debemos fijar un valor fijo $b(x) = cte$ y encontrar las raíces x que cumplen con la Ley de Planck. Al hacer esto repetidamente se puede obtener la inversa en el intervalo pedido anteriormente.

Para poder convertir esto en un problema de busqueda de ceros, hay que definir una nueva función con la siguiente forma:

$$b_{cte}(x) = \frac{x^3}{e^x-1} - cte$$

implementado en Python queda de la forma:

```
    #Ley de Planck
    def bp(x):
        return x**3 / (np.exp(x) - 1)
    #Funcion que le buscaremos los ceros
    def bcte(x):
        return bp(x) - b0
```

Usando el método de la bisección definido anteriormente en 2.14.1, calculamos el valor para el cual la función *bcte(x)* se anula. Repitiendo este procedimiento para distintos valores de $b(x) = cte$, obtenemos la inversa de la ley de Planck en el intervalo anteriomente mencionado, para esto metemos la busqueda de ceros dentro de un ciclo "for"de la siguiente manera:

```
    for b0 in np.arange(0.1, 1.5, 0.1):
        cero1 = biseccion(bcte, 0, 3)
        cero2 = biseccion(bcte, 3, 10)
        plt.scatter(bp(cero1),cero1, color='red')
        plt.scatter(bp(cero2),cero2, color='red')
    plt.plot(bp(x), x)
```

Listing 2.31: Inversa de la función mediante el método de la bisección.

Primeramente en el código se busca el primer cero en el intervalo $0 < x < 3$ y luego buscamos para $3 < x < 10$, esto debido a la forma de la gráfica de la Ley de Planck, la cual tiene una forma tal que si una línea la atraviesa horizontalmente, la cortará en 2 puntos.

La gráfica de la inversa de la Ley de Planck es la que se puede ver en la figura 2.29.

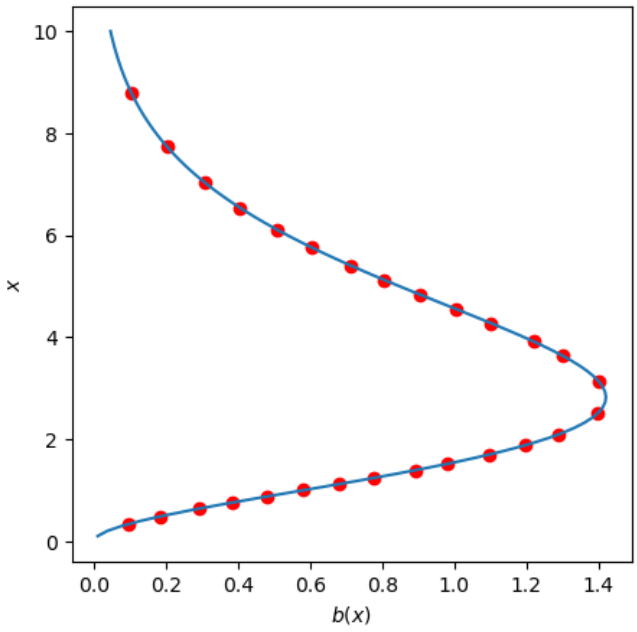

Figura 2.29: Gráfica de la inversa de la ley de Planck

En esta actividad se adimencionalizó la Ley de Planck, la cual es una de las leyes más importantes de la física cuántica. El proceso de adimensionalización es muy útil, puesto que al momento de hacer sustituciones como la que se hizo en esta actividad, eliminamos dimensiones de la ecuación, dejandola

más compacta y más fácil de entender, lo que también hace facilita la resolución y análisis de problemas que incluyan esta ecuación u otras más complejas.

Durante el desarrollo se pudo observar que un problema, como lo es encontrar la inversa de una función, con los adecuados cambios se puede reducir en una busqueda de ceros, lo que evidencia lo útiles que son estos métodos en la resolución de problemas.

Capítulo 3

Conclusiones

Fecha de presentación: 11 de diciembre del 2023

En síntesis, en este portafolio se utilizaron elementos básicos de Python para imprimir un triángulo de Pascal de n filas, reflejar la fuerza gravitacional entre dos masas y extrapolar el Algoritmo de Shor para un intervalo menor. Se usaron métodos de diferenciación e integración numérica para derivar un polinomio y encontrar su error numérica y analíticamente, calcular derivadas discretas y graficar, confirmar el Teorema Fundamental del Cálculo, comparar integrales resueltas numéricamente y analíticamente, y graficar las primeras 10 funciones de Bessel. También, se implementaron métodos de resolución de Ecuaciones Diferenciales Ordinarias para describir el movimiento de un péndulo simple y un péndulo doble, modelar el atractor de Lorenz resolviendo su sistema de ecuaciones y graficando sus soluciones, resolver la ecuación de Lane-Emnden y graficar sus soluciones. Y se utilizaron métodos de búsqueda de ceros para encontrar los ceros de las Funciones de Bessel, resolver ecuaciones y encontrar la inversa de una función para ciertos valores constantes.

Algunas actividades que se consideraron particularmente relevantes de este portafolio fueron la actividad del péndulo doble, el atractor de Lorenz y la aactividad de la ecuacción de Lane-Emden. La actividad del péndulo permitió analizar un sistema caótico y como su características afectan al análisis de energía. Para el atractor de Lorenz, de igual manera se trabajo con un sistema caótico, mostrando lo importante que son las condiciones iniciales para estos casos. Por último, la actividad de la ecuación de Lane-Emden resulto llamativa, pues permitió dar solución a una EDO utilizada para el estudio de estrellas. Siendo estas tres actividades motivo para seguir aprendiendo de prpgramación y métodos numéricos.

Así, se puede concluir que se cumplieron las visiones y los resultados esperados del curso, mejorando habilidades como el trabajo en equipo, la consistencia y el orden. También, este portafolio permitió mantener registro ordenado y conciso de las actividades, consideradas relevantes, realizadas durante el semestre con relación al curso.

Además, se puede decir que el grupo trabajo desde un inicio de manera consistente, también se logró enfrentar a los problemas dentro del grupo de buena manera, priorizando el trabajo en equipo. Además, aunque se dividió el trabajo, se logró llegar a ciertos acuerdos de forma que se pudieran conectar efectivamente las secciones del trabajo entre si. Por otro lado, con respecto a las metodologías del curso, se podrían mejorar la comunicación entre los estudiantes y el profesor por medio de la publicación de los hitos y cambios de fechas importantes a través de medios oficiales como lo es INFODA, Canvas o Teams, y así evitar cualquier duda pueda surgir al respecto. Además, con respecto a las ayudantías, se podría compartir algún ejercicio de ejemplo con su respectiva solución a través de un repositorio GitHub o por el mismo canal de ayudantías en Teams.

Por último, se puede destacar la utilidad de este portafolio en un futuro, ya que no solo deja en evidencia el trabajo realizado en este curso, sino que también muestra las habilidades de programación aprendidas en este semestre. Por lo que este portafolio servirá como base para otros ramos futuros de programación que requieran los contenidos y conceptos presentes en este documento.

Bibliografía

[1] R. A. Serway and J. Jhon W. Jewett, *Física para ciencias e ingenería con Física Moderna*, seventh ed., Vol. 2 (Elsevier/Academic Press, Amsterdam, 2003).

[2] M. A. B. y. J. A. P. Diego Arevalo, *Métodos numéricos con Python* (Editorial Politécnico Grancolombiano, 2021).

[3] A. Gezerlis, *Numerical Methods in Physics with Python* (Cambridge University Press, 2020).

[4] E. N. Lorenz, Journal of the atmospheric sciences **20**, 130 (1963).

[5] W. contributors, "Lorenz system," (2023), accessed on December 10, 2023.

[6] M. Christersson, "Interactive lorenz attractor," (2015), accessed on December 9, 2023.

[7] D. J. Griffiths, *Introduction to quantum mechanics*, 3rd ed. (Pearson Education, 2017).

[8] J. J. Sakurai and J. Napolitano, *Modern quantum mechanics*, 2nd ed. (Pearson Education, 2011).