# Portafolio

## Física Computacional II

**Joaquín Ignacio Parra Sánchez**
Ciencias Físicas
Departamento de Física
Facultad de Ciencias Físicas y Matemáticas
UNIVERSIDAD DE CONCEPCIÓN

Agosto-Diciembre 2025
Concepción, Chile

**Profesor Guía:** Dr. Roberto E. Navarro

# Índice general

# Introducción

Un portafolio es una herramienta que recopila de manera organizada evidencias del trabajo y aprendizaje del/la estudiante. Su propósito es mostrar el proceso de construcción del conocimiento del/la estudiante, permitiendo valorar su progreso, reflexión crítica y desarrollo de competencias. Este portafolio, en particular, recopila evidencias de aprendizaje de la asignatura de **Física Computacional II** (510240), dictada en el segundo semestre de 2025 en el departamento de Física de la **Universidad de Concepción**.

Esta asignatura se enfoca en la resolución de problemas en Física usando métodos numéricos y el lenguaje de programación `python`. Al finalizar este portafolio, se espera que los y las estudiantes logren:

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

La evaluación consiste en este portafolio, **alojado en un repositorio** de GitHub en la organización fiscomp2-UdeC2025 especialmente habilitada para esta asignatura.

1. El documento debe ser escrito completamente en LaTeX, y debe contener: (a) La identificación del/la estudiante en la portada y sección de información personal; (b) las evidencias de aprendizaje, que corresponden a la resolución de problemas asignados en las tareas del curso, y; (c) una sección de conclusiones final donde el/la estudiante reflexione sobre su proceso de aprendizaje.
2. Cada tarea, compuesta por 1 a 4 problemas, será publicada con plazo aproximado de dos semanas para su entrega en GitHub. Estas tareas deben ser resueltas en grupos de 3 estudiantes (2 en casos debidamente justificados). Los grupos no son fijos y podrán reorganizarse en cada nueva tarea.
3. La solución de cada problema debe ser presentada en un capítulo independiente del portafolio (un problema = un capítulo). Cada capítulo debe contener: (a) los **autores** de la tarea; (b) un **resumen** breve de los objetivos; (c) un **desarrollo** de la solución explicado de forma clara, completa y concisa; y (d) una sección de **agradecimientos** donde se indique cuál fue el **aporte de cada integrante del grupo** y, si corresponde, una declaración sobre el uso de herramientas de IA o de ayuda externa al grupo.
4. Aunque las tareas se resuelvan en grupos, **cada estudiante es responsable de mantener su propio portafolio**, asegurándose de la integridad del mismo (por ejemplo: entregar las soluciones antes del plazo límite, comprobar que el documento compila correctamente y sin errores, y redactar mensajes de *commits* de forma clara, descriptiva y adecuada).
5. La evaluación se realizará sobre la versión de uno de los integrantes del grupo, elegida aleatoriamente tras el cierre del plazo de cada tarea. La calificación y retroalimentación obtenidas serán válidas para todo el grupo. Si un integrante del grupo no ha subido sus respuestas al momento del cierre, su calificación individual será la mínima, sin afectar la del resto de su grupo. Luego, se procederá a evaluar el portafolio de otro integrante para determinar la calificación del grupo.

6. Cada estudiante podrá editar su respuesta original para incluir esta retroalimentación en su propio portafolio, de manera independiente del resto del grupo. Esto generará una nueva evidencia, cuyo progreso puede ser seguido con el historial de git/github. **No está permitido eliminar ni alterar este historial**, ya que es la única forma de evidenciar el proceso de aprendizaje.
7. El portafolio completo será evaluado al final del semestre, de forma independiente de las tareas, lo que constituirá la nota de portafolio. Cada estudiante tendrá plazo hasta el **miércoles 10 de diciembre de 2025** para incorporar en el documento final las mejoras sugeridas en la retroalimentación de las tareas parciales. En esta revisión se evaluará: (a) que cada problema haya sido resuelto de forma completa y correcta; (b) que el/la estudiante haya incorporado las correcciones sugeridas en la retroalimentación; y (c) la coherencia y calidad del capítulo de conclusiones.
8. Se considerará causal de reprobación inmediata de la asignatura, con sanciones académicas correspondientes, cualquiera de las siguientes conductas:

   *a*) Plagio total o parcial del portafolio, ya sea entre estudiantes del mismo curso o a partir de fuentes externas. En caso de detectarse copia sustancial entre portafolios, ambos estudiantes involucrados serán sancionados, salvo que existan evidencias claras que identifiquen al autor original.
   *b*) Presentar un portafolio no auténtico, elaborado por otra persona o generado de manera automática mediante inteligencia artificial u otros medios sin participación real o sin declaración explícita del estudiante.
   *c*) Entrega del portafolio por medios distintos a la plataforma oficial definida para el curso.
   *d*) Alteración o falsificación de evidencias, incluyendo manipulación de resultados, datos o retroalimentación docente, con el fin de obtener ventaja académica indebida.

## Criterios de evaluación

Como criterios de evaluación general, se considerará:

1. Coherencia de las evidencias con los resultados de aprendizaje del curso y la autoevaluación al final del documento.
2. Redacción y competencias comunicativas, incluyendo ortografía, gramática, sintaxis, presentación de figuras, uso de LaTeX y explicación clara de la parte relevante de scripts de `python`.
3. Presentación: Claridad, limpieza y orden del documento.
4. Incorporación de retroalimentación recibida en entregas parciales y evidencia de progreso a lo largo del curso.

Este portafolio, además de ser una herramienta de evaluación, es una oportunidad para que el/la estudiante reflexione sobre su proceso de aprendizaje, consolide sus conocimientos y desarrolle habilidades clave en el ámbito de la computación aplicada a la física. Se espera que este documento sirva como un registro tangible de su progreso y de las competencias adquiridas en la asignatura, las cuales podrán ser de gran utilidad en futuros desafíos académicos y profesionales.

# Información personal y académica

## Datos personales

| | |
|---|---|
| **Nombre completo** | Joaquín Ignacio Parra Sánchez |
| **Matrícula** | 2024429400 |
| **Fecha de Nacimiento** | 30 de Junio del 2005 |
| **Nacionalidad** | Chileno |
| **E-Mail institucional** | jparra2024@udec.cl |

## Breve biografía académica

Mi nombre es Joaquín Ignacio Parra Sánchez, estudiante de la carrera de Ciencias Físicas en la Universidad de Concepción, Chile. Ingresé a la carrera el año 2024 y me encuentro cursando el cuarto semestre. Previo a mi ingreso a la UdeC, estudié en el Colegio Seminario Padre Albero Hurtado de Chillán. Durante la enseñanza media participé activamente en clubes de debate de mi institución, y en olimpiadas de física y matemática, destacando haber obtenido primer lugar en las olimpiadas de física UdeC, medalla de oro y plata en las olimpiadas de matemática de la UBB y UFRO, respectivamente en 2023. Espero especializarme en algún área dentro del margen de gravitación y altas energías (aunque ya esté trillado).

## Visión general e interés sobre la asignatura

Comencé mirando a Física Computacional II como un ramo ajeno pero intrigante, pues al igual que la mayoría de personas, no tuve un acercamiento previo a la programación durante mi adolescencia. En el futuro inmediato, espero que este curso me entregue las herramientas básicas para abordar problemas que no posean soluciones analíticas, o que sean muy complicadas (como el péndulo doble). Espero desarrollar competencias sólidas en programación y sobre todo, tener una base lo suficientemente sólida para poder abordar otros problemas computacionales (por ejemplo, alguna EDP). Considero que este curso me entregará herramientas fundamentales para abordar problemas en los cursos de Mecánica Clásica I y Electrodinámica I, donde muchas veces los cálculos se vuelven demasiado complejos como para intentar conseguir la solución analítica.

## Resultados esperados de este portafolio

Al finalizar este portafolio, espero haber desarrollado un manejo adecuado de herramientas matemáticas de nivel intermedio para aplicarlas al modelado y resolución de problemas de física donde no es posible obtener soluciones analíticas simples. La organización del portafolio me permitirá presentar de manera clara y ordenada la evidencia de mis aprendizajes, así como llevar un seguimiento de mi progreso a lo largo del semestre.

# Evidencias de aprendizaje

# Capítulo 1

# El Zen de Python

Joaquín Parra, Ignacio Falcón, Alvaro Osses

**Fecha de la actividad:** 8 de Octubre de 2025

En este capitulo, analizaremos el paquete `this`, el cual imprime el Zen de Python codificado con un cifrado particular. Para ello, discutiremos como encontrar este módulo en la computadora, y luego analizaremos directamente el código para comprender su funcionamiento y como este logra imprimir el Zen de Python.

## 1.1. El Zen de Python

El archivo `this` fue encontrado utilizando el comando `find` de bash, en particular, se utiliza el comando `$ find -name this.py` que retorna una lista de ubicaciones de archivos con ese nombre (posiblemente debido a que esta computadora ha tenido varias instalaciones de `python` distintas). Tomando solo una, se utiliza el comando cd para ir al directorio en el que se encuentra, específicamente, se ejecuta `$ cd anaconda3/lib/python3.13` y luego, se abre el archivo utilizando el editor de texto nvim, `$ nvim this.py`. Notar que la dirección del archivo es `/home/$USER$/anaconda3/lib/python3.13/this.py`. Cabe añadir que el metodo anteriormente descrito es valido únicamente para computadoras con sistema operativo Linux, en particular, el proceso fue realizado en un sistema con Arch Linux 6.16.5.

Tenemos que el archivo contiene el siguiente codigo:

```
s = """Gur Mra bs Clguba, ol Gvz Crgref

Ornhgvshy vf orggre guna htyl.
[...17 líneas...]
Anzrfcnprf ner bar ubaxvat terng vqrn -- yrg'f qb zber bs gubfr!"""

d = {}
for c in (65, 97):
    for i in range(26):
        d[chr(i+c)] = chr((i+13) % 26 + c)

print("".join([d.get(c, c) for c in s]))
```

Primero, notar que 65 y 97 representan los códigos ASCII de los caracteres "A" y "a" respectivamente (Wikipedia contributors, 2025a), 26 representa la cantidad de caracteres en el abecedario

inglés. Además, tenemos que la función de este programa es sumarle 13 al código ASCII de todos los caracteres (tanto mayúsculas como minúsculas).

Nótese que el primer ciclo for `for c in (65, 97)` utiliza una tupla como iterable, es decir, este bucle tendrá dos ciclos, uno donde `c = 65` y otro donde `c = 97`. Recordando que 65 y 97 son los códigos ASCII de "A" y "a" respectivamente, podemos deducir que el propósito de este primer `for` es decodificar primero los caracteres mayúsculos y luego los minúsculos.

El segundo ciclo tiene precisamente ese propósito, notando que `d` es una variable de tipo diccionario, esta guardará categorías y elementos asociados a estas. Teniendo esto en cuenta, y recordando que `chr(n)` retorna el caracter asociado al código ASCII `n`, el segundo `for` asociará un caracter al que se encuentre a 13 espacios de este, por ejemplo, en el primer ciclo (`c = 65`, `i = 0`) tendremos `d[chr(0 + 65)] = chr((0 + 13) % 26 + 65)` donde `13%26 + 65 = 13 + 65 = 78`, es decir `d[chr(65)] = chr(78)` esto es, a la categoría en `d` dada por `chr(65) = "A"` le asocia el objeto `chr(78) = "N"`. Dicho de otro modo, al caracter "A" le asocia el caracter a 13 espacios de este, en este caso "N". Esto lo repite para todos los caracteres del abecedario en mayúsculas y minúsculas (para ambos ciclos del `for` anterior).

Cabe añadir que el operador `%` (modulo) tiene la función de "contar" cuanto se pasa la suma `i + 13` de 26, por ejemplo, si `i + 13 = 27` entonces `(i+13)%26 = 27%26 = 1`, en caso de que la suma sea menor que 26 simplemente se retorna el valor de la suma (`(i + 13)%26 = i + 13` si `i+13 < 26`) de esta manera la codificación se mantiene dentro de los caracteres del abecedario, pues si pasan del ultimo caracter en este subconjunto la lista se devuelve al primer caracter, "A" o "a" dependiendo del caso. Por ejemplo, para `i = 12`, `c = 65` tendremos `d[chr(12 + 65)] = chr((12+13) % 26 + 65)`, notando que `12 + 65 = 77` y `(12+13) % 26 = 25%26 = 25`, con lo cual `(12+13) % 26 + 65 = 25 + 65 = 90`, esto es `d[chr(77)] = chr(90)`, o bien `d["M"] = "Z"`. En cambio, para `i = 13`, `c = 65` `d[chr(13 + 65)] = chr((13+13) % 26 + 65)`, donde tenemos que `(13+13) % 26 = 26 % 26 = 0`, por lo tanto `d[chr(78)] = chr(65)`, esto es `d["N"] = "A"`

En la última línea, se define un string vacío `""`. La función `get(c, d)` retorna el valor, o valores, asociado a la categoría `c`, en caso de que este no exista retorna `d`. Luego, la función `join` concatena ese caracter al string vacío. Es decir, en esta última línea, se utiliza un ciclo `for` para iterar por cada caracter del string `s` y concatena su caracter asociado por el diccionario `d` al string vacío; en caso de que este caracter no exista como categoría en `d`, como es el caso de los puntos, comas y guiones, simplemente retorna el mismo caracter. Dicho de otro modo, el ciclo `[d.get(c, c) for c in s]` tiene la función de "decodificar" `s`. Por ejemplo, para el caso particular en que `s = "Gur Mra bs Clguba"` el código iterará por `s` y decodificará cada caracter para luego concatenarlo al string vacio `""` mediante `join`. Es decir, tomará el primer caracter de `s` (`"G"`), lo pasará por el alogritmo ya descrito y lo convertirá en `"T"`, luego lo concatena al string vacio, de manera que este pase a ser `"T"`. Este proceso se repite para cada caracter en `s` resultando finalmente en que el string vacio se convierta en la version descifrada del `s`, esto es `"The Zen of Python"`. Aplicando este proceso al string `s` completo, resulta finalmente en el Zen de Python decodificado. Notar que el tipo de cifrado que realiza el módulo se denomina ROT13, pues a cada letra del abecedario le asigna una letra de un abecedario "rotado", en este caso por 13 espacios. (Wikipedia contributors (2025b))

## Conclusiones

Así, hemos descifrado el funcionamiento del modulo `this.py`. El trabajo invertido en este capitulo es un recordatorio de la importancia de ser claro y explícito a la hora de programar, pues así será mucho más fácil entender el código a simple vista y no será necesario analizarlo linea por linea, como fue el caso con este modulo.

## Agradecimientos

Este capitulo fue principalmente escrito por Alvaro Osses Nieto, con contribuciones de Joaquín Parra Sanchez e Ignacio Falcón Painemal para el desciframiento del código en cuestión, y búsqueda de referencias pertinentes al capitulo.

Cabe añadir que durante la escritura de este capitulo no se utilizaron herramientas de inteligencia artificial.

# Capítulo 2

# Análisis de Funciones en Numpy y Resolución de problemas

Ignacio Falcón, Joaquín Parra, Alvaro Osses

**Fecha de la actividad:** 8 de Octubre del 2025

Para este capítulo, se investigó y trabajó con una variedad de funciones pertenecientes a la libreria *Numpy* del lenguaje de Programación Python con el objetivo de evaluar y comprobar las capacidades que pueden brindar a la hora de trabajar en matemáticas. Luego, se analizaron las funciones estudiadas con el objetivo de demostrar su utilidad frente a objetivos concretos.

## 2.1. Funciones de Numpy

Para esta sección, se tenía como problema:

*Consultar y buscar información acerca de las siguientes funciones de* `numpy`, `arange`, `array`, `linspace`, `geomspace`, `ones`, `zeros`, `random.random`, `random.normal`, `random.randint`, *además, se debe listar cada función según su nombre, definición y ejemplo.*

Para esto, se confeccionó una tabla compuesta por un conjunto de múltiples funciones pertenecientes a la librería *Numpy* junto a su debida explicación, a su vez, se listó ejemplos del uso de cada una de las funciones mencionadas.

Esta tabla se puede encontrar en el Cuadro (2.1)

## 2.2. Visualización de una distribución normal en un conjunto de 200 datos aleatorios

Este problema, consistía en generar una cantidad concreta (mayor a 200) de números aleatorios que cumplan con una distribución normal. Luego, en una misma figura, generar un gráfico de $a_n$ vs $n$ y un histograma de $a_n$. Para esto, se abordó el uso de la función `numpy.random.normal` para generar los números en cuestión. Luego, haciendo uso de la libreria *MatplotLib.pyplot*, se generaron dos gráficos, cada uno correspondiente a los valores $(n, a_n)$ y al histograma de $a_n$.

```
dist_normal = np.random.normal(0,0.5,N) # Se generan N cantidad de números aleatorios
                                        # según una distribución normal centrada en 0
                                        # y con desviación estándar de 0.5

plt.scatter(np.arange(0,201), dist_normal) #Crea el gráfico de puntos en el espacio
```

| Función | Explicación | Ejemplos |
|---|---|---|
| arange | Función que genera un arreglo númerico en un intervalo a saltos, debe indicarse en orden, el inicio y término del intervalo (no lo toma) y el salto entre valores. | `numpy.arange(0,10,2)`<br>`# array([0. 2. 4. 6. 8.])` |
| array | Función que convierte una lista específica de datos en un arreglo numérico de Numpy, se le debe indicar dentro del paréntesis la lista a trabajar. | `numpy.array([1,2,3])`<br>`#array([1. 2. 3.])` |
| linspace | Función que genera un arreglo en un intervalo para una cantidad de puntos a una distancia igual, se debe indicar en orden donde comienza el intervalo, donde termina el intervalo y la dimensión de datos del arreglo. | `numpy.linspace(0,10,6)`<br>`#array([0. 2. 4. 6. 8. 10.])` |
| geomspace | Función que genera un arreglo en un intervalo a una proporción geométrica entre valores, se le debe indicar en orden donde comienza el intervalo, donde termina y la dimensión de datos que tendrá el arreglo. | `numpy.geomspace(1,1000,4)`<br>`# array([1. 10. 100. 1000.])` |
| ones | Función que genera un arreglo de una cantidad específica de coeficientes, los cuales todos son de valor 1, esta cantidad debe ser agregada dentro del paréntesis de la función. | `numpy.ones(4)`<br>`#array([1. 1. 1. 1.])` |
| zeros | Función que genera un arreglo de una cantidad de específica de coeficientes, los cuales son todos de valor 0, esta cantidad debe ser agregada dentro del paréntesis de la función. | `numpy.zeros(4)`<br>`# array([0. 0. 0. 0.])` |
| random.random | Función que genera un arreglo de una cantidad específica de valores aleatorios que se encuentran entre 0 y 1, este arreglo puede ser de la dimensión que se escoja. | `numpy.random.random(4)`<br>`#array([0.12771608, 0.65220371`<br>`# , 0.99625441, 0.90112646])` |
| random.normal | Función que genera un arreglo de coeficientes que cumplen una distribución normal en torno a un valor promedio (primer valor), una desviación estándar concreta(Segundo valor) y la dimensión de elementos | `numpy.random.normal(10,2,5)`<br>`# array([ 8.879 12.321`<br>`# 9.768 11.045 7.533])` |
| random.randint | Función que genera un arreglo de valores enteros, en un intervalo definido, debe indicarse en orden el inicio del intervalo, el término el intervalo (valor que no toma) y la dimensión (cantidad) de elementos. | `numpy.random.randint(1,10, 3)`<br>`# array([3. 7. 9.])` |

Cuadro 2.1: Tabla de funciones de Numpy como: Nombre - Descripción - Ejemplo.

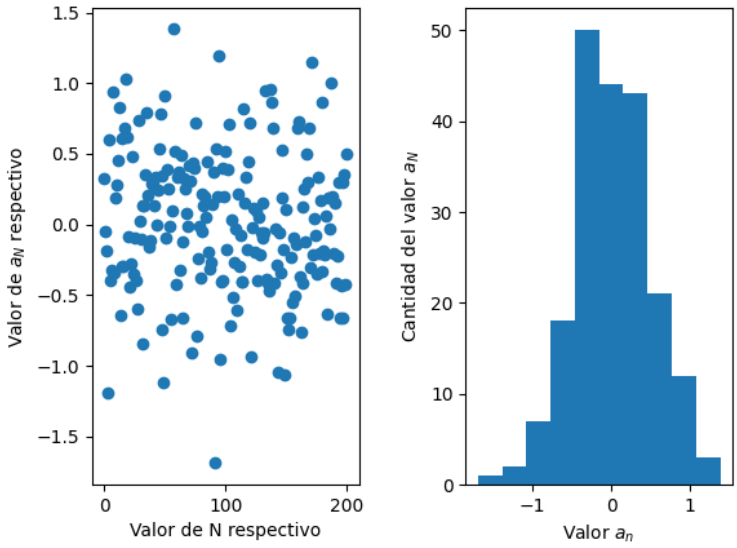


Figura 2.1: Gráfico e Histograma de datos

```
6
7 plt.hist(dist_normal) # Se genera un histograma
```

Esto, al ejecutarse, genera la Figura 2.1, la que compone al gráfico $a_n$ y al histograma de datos $a_n$.

## 2.3. Generación de matriz de números complejos e impresión de valores según su posición

El problema consistía en definir inicialmente una matriz 3x5 de números complejos, para luego imprimir la primera fila, la última columna y el elemento de la segunda fila y primera columna.

Para este problema, se generó una matriz concreta haciendo uso de la función array, a la cual se le introdujo 3 listas de 5 valores cada una, todos con números complejos, luego se procedió a imprimir lo requerido. A continuación, el código utilizado.

```
matriz = np.random.random((3,5)) + 1j*np.random.random((3,5)) # Crea una matriz de 3x5
                        # con números complejos


print(matriz[0]) # Imprime la Primera fila de la matriz
print(matriz[:,-1]) # Imprime la última columna de la matriz
print(matriz[1,0]) # Imprime el coeficiente 2x1 de la matriz
```

## 2.4. Diferencia entre elementos adyacentes de dos series de datos numéricos

Para el tercer problema, se requería generar una serie enésima de números enteros aleatorios, en este caso, considerando hasta $N = 10$, luego, se pedía evaluar una nueva serie de diferencias entre elementos

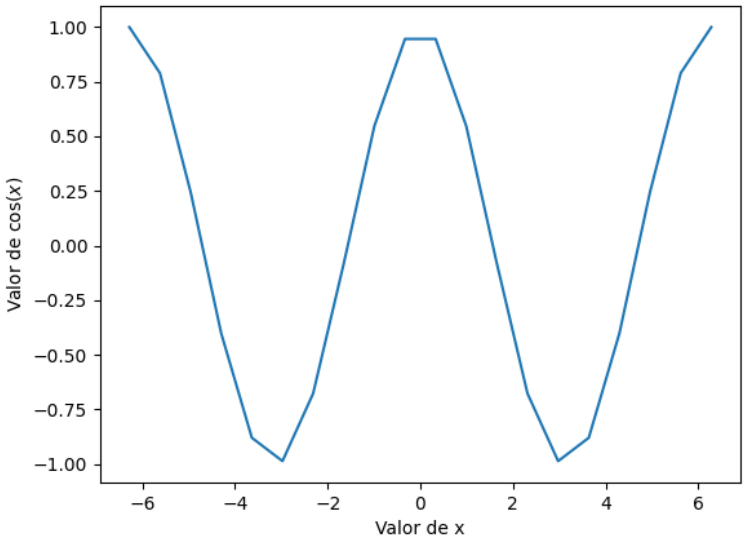


Figura 2.2: Gráfico de cos(x) en función del Arreglo X

adyacentes de la serie generada. Esto debía trabajarse tanto con ciclos *for*, como con operaciones vectoriales de *Numpy*.

Para esto, se abordó inicialmente haciendo uso de un ciclo for para generar la diferencia entre valores dentro de arreglos generados por una lista $x_i$, luego, se hizo uso de la función *diff* de Numpy, con la cuál se puede encontrar la diferencia de valor entre posiciones seguidas de un arreglo. A continuación se presenta el código usado.

```
serie = np.random.randint(1,100,n+1) # Crea un array de N+1 números aleatorios enteros
                                     # del 1 al 100

diferencia = np.zeros(n) # Generamos un arreglo de N espacios con ceros
for i in range(0,len(serie) - 1):
    diferencia[i] = serie[i+1] - serie[i] # Para cada posición del arreglo, evalua
    # la diferencia entre el espacio siguiente y el actual.

print(serie[1:] - serie[:-1],"Es la serie hecha con la propiedad vectorizada de arrays.")
# Imprime la nueva serie usando propiedades vectorizadas
```

## 2.5. Representación de la función coseno en un gráfico a puntos equidistantes

Para el último problema, se pedía definir una variable $x$ como un arreglo de 20 elementos en orden creciente en el intervalo $-2\pi \leq x \leq 2\pi$, luego, se requería graficar $\cos x$ en función del arreglo e interpretar su resultado.

Para esto, se generó una variable x con 20 elementos aleatorios de misma distancia usando la función *linspace*, de Numpy, en un rango entre $[-2\pi, 2\pi]$. Luego, haciendo uso de la librería *MatplotLib.pyplot*, se graficó $\cos x$ en función del arreglo x generado anteriormente, como se puede notar en la Figura 2.2.

Es importante señalar que el gráfico no es suave, en concreto, luce del estilo poligonal con una cantidad N de aristas, esto es debido a que este está generado en base a una cantidad discreta de

valores, en donde `plot` genera rectas para cada intervalo entre puntos, esto, a diferencia de la función $\cos x$ para un intervalo, como lo sería para $x = [-2\pi, 2\pi]$ donde esta, al poseer una cantidad infinita de puntos, es de gráfica suave.

A continuación, se presenta el código usado.

```
plt.figure() # Genera una nueva figura vacía

x = np.linspace(-2*np.pi, 2*np.pi, 20) # Genera un arreglo de 20 valores entre -2pi
                                       # y 2pi de forma creciente

plt.plot(x, np.cos(x)) # Grafica la función cos(x) con x el arreglo
                       # anteriormente trabajado

plt.ylabel(r'Valor de $\cos(x)$')
plt.xlabel(r'Valor de x')

plt.show()
```

## Conclusiones

Para el presente caítulo, se puede concluir el aprendizaje obtenido a través del análisis de funciones en Numpy, una útil librería de `Python` que permite realizar diferentes acciones matématicas, principalmente a través del uso de *Arrays* o Arreglos de números.

Gracias a esta librería, a su vez, se pueden destacar una variedad de funciones, como se muestra para la primera sección de este capítulo, donde se abordaron funciones elementales como `numpy.linspace` o `numpy.random.normal`.

A su vez, se usaron las funciones analizadas en una serie de contextos a modo de problema para las posteriores secciones, donde se evaluaron respectivos usos, tales como la generación de números y aplicaciones en productos de diferente índole, y posibles limitaciones que presentan estos objetos a la hora de trabajar con arreglos de tipo numérico (para contextos de números reales o complejos), de esta forma, se pudo comprender de mejor manera los usos posibles y aplicaciones que tienen estas en el contexto de estudio para la asignatura o próximos proyectos.

Debido a esto, se espera que para próximos capítulos, se puedan utilizar las propiedad pertenecientes a las funciones analizadas con mayor profundidad en el ámbito del cálculo numérico.

## Agradecimientos

Para este capítulo, se requirió ayuda de múltiples fuentes de documentación, especialmente para la librería de `Numpy`, por lo cual se agradece la propia documentación que está posee. Notar la mayoritaria creación del capítulo por Ignacio Falcón, además, notar el trabajo de Alvaro Osses en consejos y correcciones para el código de los problemas y a Joaquín Parra a la hora de desarrollar la tabla inicial, especialmente debido a los múltiples errores que se obtenían al momento de proceder a compilar el documento en cuestión.

Finalmente, se agradece al Dr. Roberto Navarro por las correcciones hechas para este capítulo, las cuales se analizaron en búsqueda de la mejor solución posible.

# Capítulo 3

# Números de Catalán

Joaquín Parra, Ignacio Falcón, Alvaro Osses

**Fecha de la actividad:** 8 de Octubre de 2025

En este capítulo vamos a familiarizarnos a calcular los primeros números de Catalán mediante el uso de distintas herramientas de `python` con el fin de observar la precisión de los distintos métodos. También, exploraremos otras formas de expresar esta secuencia de números y aproximaciones útiles de estos. Los códigos tanto de funciones como de gráficos pueden encontrarse en $src/catalan.

## 3.1. Cómputo y Precisión

El $n$-ésimo número de Catalán, para $n \in \mathbb{N}_0$ se define como:

$$C_n = \frac{(2n)!}{(n+1)! \cdot n!}. \tag{3.1}$$

Para trabajar con los números de Catalán, se definieron dos funciones factoriales: una con comandos nativos de `python` y otra con `numpy`, la cual permite ajustar la precisión de los números de forma manual. En base a esto, podemos definir otras 2 funciones que nos retornen los números de Catalán:

```python
def catalan(N):
    lista = []
    q = factorial(2*(N+1))
    for i in range(N+1):
        lista.append(q[2*i] / (q[i+1] * q[i]))
    return lista
```

```python
def catalan_numpy(N, dtype=np.int64):
    q = factorial_numpy(2*N, dtype=dtype)
    return q[::2] / ( q[1:N+2] * q[:N+1] )
```

El cuadro 3.1 muestra los primeros 15 números de Catalán. La notación lleva un `[N]` pues recordemos que las funciones nos devuelven listas/arreglos, y extraemos el N-ésimo elemento que se corresponde con el N+1-ésimo número de Catalán. Por conveniencia, se ha decidido trabajar la función `catalan_numpy` con alta precisión de `np.float128` para realizar una comparación en igualdad de condiciones cuando N comienza a crecer.
Gracias a la alta precisión escogida, podemos ver que para los primeros N=15 números de Catalán,

| N | Nativo | np.int12 | np.int32 | np.int64 | np.float128 |
|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 5 | 5 | 5 | 5 | 5 |
| 4 | 14 | -8.75555556 | 14 | 14 | 14 |
| 5 | 42 | 1.16564417 | 42 | 42 | 42 |
| 6 | 132 | -0.04210526 | 132 | 132 | 132 |
| 7 | 429 | -0.71428571 | 6.2936256 | 429 | 429 |
| 8 | 1430 | -2 | 1.14759881 | 1430 | 1430 |
| 9 | 4862 | 0 | 0.51752581 | 4862 | 4862 |
| 10 | 16796 | nan | 1.05997819 | 16796 | 16796 |
| 11 | 58786 | nan | -2.45869297 | -65.41040340 | 58786 |
| 12 | 208012 | nan | 0.64969271 | -2.62683037 | 208012 |
| 13 | 742900 | nan | -1.10638298 | -0.198536786 | 742900 |
| 14 | 2674440 | nan | 4.49315068 | 97.0002887 | 2674440 |

Cuadro 3.1: Primeros 15 números de Catalán a distinta precisión.

ambas funciones se comportan bien: obtenemos los mismos resultados. Sin embargo, existen discrepancias importantes cuando variamos el parámetro `dtype`. La figura 3.1 muestra los retornos de la función `catalan_numpy` cuando ajustamos la precisión a `np.int16`, `np.int32` y `np.int64`.

Notamos que hasta N=4 no se aprecian anomalías. Después observamos discrepancias en N=5, N=7 y N=11 para las precisiones de (`np.int`) 16, 32 y 64 bits, respectivamente, y si observamos el código, vemos que se produce un Overflow. La explicación de este comportamiento errático está en como se relaciona la precisión utilizada con la definición de los números de Catalán. Recordemos que cuando ajustamos la precisión a entero de 16 bits, podemos almacenar a lo más el entero positivo $2^{15} - 1 = 32767$ (un bit se guarda para el signo). Para cuando calculamos el quinto número de Catalán, `catalan_numpy(4)` debe calcular el factorial $(2 \cdot 4)! = 40320$, sobrepasando el límite impuesto por la precisión, generando un Overflow. Esto también explica la desaparición de algunos datos (retorna `nan` o valores errados). Mismo argumento se aplica para las precisiones de 32 y 64 bits, y nos permite comparar de forma segura con precisión de 128 bits, pues en N=15 seguimos en un régimen ordenes de magnitud menor a $2^{127} - 1$.

## 3.2. Relación de recurrencia para los números de Catalán

Los números de Catalán también pueden ser expresados con la siguiente relación de recurrencia:

$$C_0 = 1, \qquad C_{n+1} = \frac{4n+2}{n+2} C_n. \tag{3.2}$$

En efecto, sea $C_0 = 1$, luego usando la definición original (3.1):

$$\begin{aligned} \frac{C_{n+1}}{C_n} &= \frac{(2n+2)! \cdot n!}{(n+2)! \cdot (2n)!} \\ &= \frac{(2n+2)(2n+1)(2n)! \cdot n!}{(n+2)(n+1)n! \cdot (2n)!} \\ &= \frac{4n+2}{n+2}, \end{aligned}$$

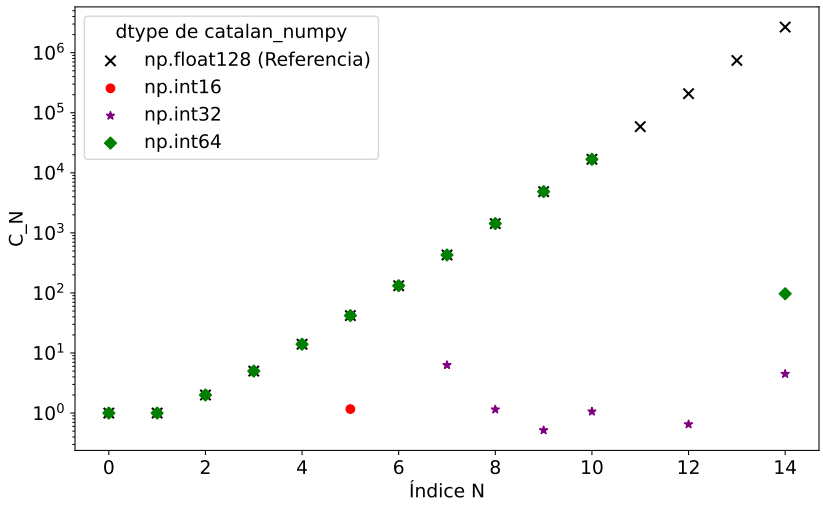


Figura 3.1: `catalan_numpy(N)` a distintas precisiones

despejando el cociente entre $C_{n+1}$ y $C_n$ llegamos a la expresión (3.2):

$$C_{n+1} = \frac{4n+2}{n+2} C_n.$$

¿Cómo se comporta esta expresión alternativa cuando $n$ crece y `dtype` cambia? ¿Existen diferencias con las definiciones anteriores? Primero definimos esta nueva expresión en `python` para realizar las comparaciones:

```python
def cat_rec(N, dtype=np.int32):
    C = np.zeros(N+1, dtype=dtype)
    C[0] = 1.0
    for i in range(N)
        C[i+1] = (4*i +2) / (i+2) * C[i]
    return C
```

Con la función ya definida, podemos hacer las comparaciones. El cuadro 3.2 muestra los resultados obtenidos al calcular los primeros 15 números de Catalán utilizando con las funciones mencionadas y variando sus precisiones. Tomando de referencia el cuadro 3.1, vemos que la función `catalan_numpy` (a la cual se comenzará a referir como **original**) comienza a fallar para N=7 a precisión entero de 32 bits. Esto es debido a que internamente la función original utiliza la ecuación (3.1) para determinar los números de Catalán, y para N=7 debe calcularse $(2 \cdot 7)! = 14!$. Este número excede el máximo de la precisión de un entero de 32 bits ($2^{31} - 1$), lo que produce un Overflow y los números comienzan a cambiar de signo, o ya no representan un número de la secuencia. En cambio, la función `cat_rec` (a la cual se referirá como **recurrencia**) funciona sin ningún problema, dada su definición sencilla sin factoriales, es mucho más fácil de computar y no excedemos el límite de la precisión de un entero a 32 bits.

A precisión entero de 16 bits, ambas funciones experimentan un Overflow, con la diferencia que la función de recurrencia no comienza a correr desde N=11 en adelante, pues `python` identifica de antemano que se excederá el límite de la precisión de 16 bits. Para N<11, la función de recurrencia funciona sin ningún problema, mientras que la función original deja de entregar números de la secuencia para N>4, pues internamente se calcula $8! > 2^{15} - 1$ y entramos en Overflow, solo que el código sigue corriendo,

| N | catalan_numpy(N)[N](32) | cat_rec(N)[N](32) | catalan_numpy(N)[N](16) | cat_rec(N)[N](16) |
|---|---|---|---|---|
| 0 | 1. | 1 | 1. | 1 |
| 1 | 1. | 1 | 1. | 1 |
| 2 | 2. | 2 | 2. | 2 |
| 3 | 5. | 5 | 5. | 5 |
| 4 | 14. | 14 | -8.75555556 | 14 |
| 5 | 42. | 42 | 1.16564417 | 42 |
| 6 | 132. | 132 | -0.04210526 | 132 |
| 7 | 6.2936256 | 429 | -0.71428571 | 429 |
| 8 | 1.14759881 | 1430 | -2. | 1430 |
| 9 | 0.51752581 | 4862 | -0 | 4862 |
| 10 | 1.05997819 | 16796 | nan | 16796 |
| 11 | -2.45869297 | 58786 | nan | Overflow |
| 12 | 0.64969271 | 208012 | nan | Overflow |
| 13 | -1.10638298 | 742900 | nan | Overflow |
| 14 | 4.49315068 | 2674440 | nan | Overflow |

Cuadro 3.2: catalan_numpy vs cat_rec en cálculo de los primeros 15 números de Catalán a precisión de (np.int) 16 y 32 bits.

lo que resulta en los valores incorrectos y los nan (not a number, posiblemente generado por forzar a que el número sea un entero).

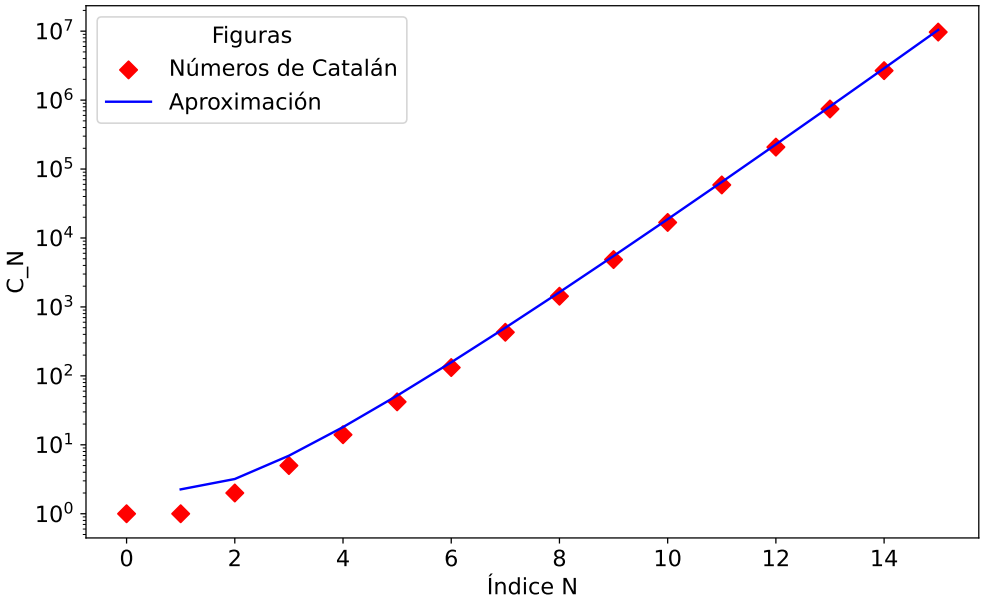


Figura 3.2: Comparación entre la aproximación y los primeros 16 ($C_{15}$) números de Catalán.

## 3.3. Aproximación

Existe una relación que permite estimar en $n$-ésimo número de Catalán (al menos para $0 < n \leq 15$), la cual es:

$$C_n \approx \frac{4^n}{n^{3/2}\sqrt{\pi}}. \tag{3.3}$$

Si bien de primeras puede parecer muy rebuscada, obtenemos muy buenas aproximaciones en el rango de valores mencionado: la figura 3.2 muestra que esta relación aproxima bastante bien el comportamiento y el valor de los números de Catalán a medida que $n$ crece. Dado que el cálculo de números de Catalán se vuelve más pesado de computar rápidamente cuando avanzamos en la secuencia, para estudios del comportamiento de esta puede ser muy útil una aproximación que no sea tan pesada de calcular.

# Conclusiones

Hemos presentado los números de Catalán y hemos evidenciado las dificultades computacionales de trabajar con ellos. Hemos estudiado formas alternativas y más eficientes de trabajar con ellos, para finalmente conocer una aproximación útil y liviana para el computador. Durante este capítulo, se familiarizó un concepto ajeno a la física y se presentó de forma interactiva, explicando paso a paso todo el procedimiento para lograr un acercamiento efectivo. Si bien de momento no hemos estudiado una aplicación directa en nuestra área de estudio, siempre es bueno expandir conocimientos y realizar trabajos en áreas afines para mejorar nuestras habilidades, en este caso, de programación.

## Agradecimientos

Este capítulo fue escrito principalmente por Joaquín Parra, con revisiones tanto de código como de contenido de parte de sus compañeros de trabajo Alvaro Osses e Ignacio Falcón, quienes también ayudaron a demostrar la relación de recurrencia descrita. Es necesario también agradecer al docente responsable del curso, Dr. Roberto Navarro, pues sus clases sirvieron de apoyo para este trabajo.

# Capítulo 4

# Derivada no equidistante de tres puntos y nodos de Gauss-Chebyshev

Ignacio Falcón, Joaquín Parra, Alvaro Osses

**Fecha de la actividad:** 7 de Noviembre del 2025

En este capítulo, se analizará y demostrará una fórmula de tres puntos no necesariamente equidistantes para realizar una derivada numérica sobre una función escalar haciendo uso de series de Taylor, para luego, desarrollar y discutir el error de truncamiento generado a través de la presente fórmula.

Seguidamente, se usarán nodos de Gauss-Chebyshev para evaluar la derivada numérica demostrada frente a la derivada analítica de la función.

## 4.1. Formulación de la derivada de tres puntos junto a su orden de error

Para iniciar, es importante dar conocimiento a las herramientas utilizadas durante esta demostración, por ello, se presenta a continuación la serie de Taylor.

Se tiene que, para una función continua e infinitamente derivable en un punto $x_0 = x + h$, el valor de la función en un punto a una distancia $h$ próxima a este por la derecha en la recta, será de:

$$f(x+h) = f(x) + f'(x) \cdot h + \frac{1}{2!} f''(x) \cdot h^2 + ... \tag{4.1}$$

En base a esto, se puede definir derivadas numéricas haciendo uso de una cantidad concreta de puntos. Para el presente desarrollo, se harán uso de tres puntos o nodos de una distancia no equidistante unos con otros.

Estos puntos, corresponderán a ser $x_{k-1}$, $x_k$ y $x_{k+1}$, en donde a su vez, se define la distancia entre puntos como $h_{k+1} = x_{k+1} - x_k$ y $h_k = x_k - x_{k-1}$. Además, se denotará también a la función evaluada en el punto $x_{k-1}$, $x_k$ y $x_{k+1}$ como $f_{k-1}$, $f_k$ y $f_{k+1}$, correspondientemente.

En base a esto y, haciendo uso de la serie de Taylor correspondiente, se tiene que:

$$f_{k+1} = f_k + \frac{df_k}{dx} \cdot h_{k+1} + \frac{1}{2!} \frac{d^2 f_k}{dx^2} \cdot h_{k+1}^2 + \frac{1}{3!} \frac{d^3 f(\xi_1)}{dx^3} \cdot h_{k+1}^3 \,. \tag{4.2}$$

En donde se puede notar que esta serie considera a los puntos $x_k$ y $x_{k+1}$, además, se considera al punto $(\xi_1, f(\xi_1))$ tal que $x_k < \xi_1 < x_{k+1}$, esto basado en el teorema del Valor Medio, a modo de realizar una consideración del resto que conlleva la serie infinita. Por otro lado, se puede considerar

a los puntos $x_{k-1}$ y $x_k$ para realizar una serie de Taylor, donde, similarmente, se puede encontrar al valor $\xi_2$, en este caso estando dentro del intervalo $x_{k-1} < \xi_2 < x_k$, entonces:

$$f_{k-1} = f_k - \frac{df_k}{dx} \cdot h_k + \frac{1}{2!}\frac{d^2 f_k}{dx^2} \cdot h_k^2 - \frac{1}{3!}\frac{d^3 f(\xi_2)}{dx^3} \cdot h_k^3 \tag{4.3}$$

Notar que en este caso, los términos impares son negativos, esto es debido al hecho que la serie está considerando el valor $f_{k-1} = f(x_k - h_k)$, por lo que su expansión considera el término $-h_k$ dentro del cálculo, por lo que este genera términos negativos en derivadas impares (ya que el exponente del término $h_k$ es impar).

Ahora, considerando (4.3) y (4.2) se puede obtener una expresión para $h_k f_{k+1} - h_{k+1} f_{k-1}$ y por tanto, luego despejar $\frac{df_k}{dx}$, obteniendo entonces:

$$\frac{df_k}{dx} = \frac{f_k - f_{k-1}}{h_{k+1} + h_k}\frac{h_{k+1}}{h_k} + \frac{f_{k+1} - f_k}{h_{k+1} + h_k}\frac{h_k}{h_{k+1}} - \frac{1}{3!}h_k h_{k+1}\frac{f'''(\xi_1)h_{k+1} + f'''(\xi_2)h_k}{h_k + h_{k+1}} \tag{4.4}$$

Ahora bien, considerando el término que contiene a $f'''(\xi_1)$ y $f'''(\xi_2)$, se puede hacer uso del teorema del valor medio para asegurar que existe un valor $\xi$ tal que exista $f(\xi)$ dentro de la recta que une los puntos $(\xi_1, f(\xi_1))$ y $(\xi_2, f(\xi_2))$, por lo cual, se puede considerar que:

$$\frac{df_k}{dx} = \frac{f_k - f_{k-1}}{h_{k+1} + h_k}\frac{h_{k+1}}{h_k} + \frac{f_{k+1} - f_k}{h_{k+1} + h_k}\frac{h_k}{h_{k+1}} - \frac{1}{3!}\frac{d^3 f(\xi)}{dx^3} \cdot h_k \cdot h_{k+1} \tag{4.5}$$

Por ello, notar que al truncar la expresión anterior en función de los primeros dos términos iniciales, y de esta forma, truncando así el error $f(\xi)$, se tiene que:

$$\frac{df_k}{dx} = \frac{f_k - f_{k-1}}{h_{k+1} + h_k}\frac{h_{k+1}}{h_k} + \frac{f_{k+1} - f_k}{h_{k+1} + h_k}\frac{h_k}{h_{k+1}} \tag{4.6}$$

Donde, se puede notar que debido al resto existente del truncamiento, el orden de menor grado $h_k$ o $h_{k+1}$ es de:

$$E_k = -\frac{1}{3!}\frac{d^3 f(\xi)}{dx^3} \cdot h_k \cdot h_{k+1} \tag{4.7}$$

De orden $O(h_k h_{k+1})$.

Esto es debido a que, a pesar que las distancias $h_k \neq h_{k+1}$, estos valores de distancia se consideran pequeños para efectos de una derivada aproximada correcta, por lo que $h_k, h_{k+1} << 1$, en consecuencia, se tiene entonces que $h_k \cdot h_{k+1} > h_k^2 \cdot h_{k+1}$ o bien $h_k \cdot h_{k+1} > h_k \cdot h_{k+1}^2$, por lo que en consecuencia, el orden con mayor relevancia (o bien, mayor valor) es de $O(h_k h_{k+1})$ Esto además, es producto del efecto obtenido al sumar las expansiones en serie de Taylor sobre los términos con derivada de segundo grado, los cuales se cancelan, cancelando de esta forma términos con posible error de orden $O(h)$.

## 4.2. Análisis de la derivada de tres puntos con nodos Gauss-Chebyshev

Para continuar con el análisis de la derivada de tres puntos formulada en la sección anterior, se comparará su resultado numérico frente a la derivada algebraica.

Para esto, se hará uso de los **Nodos de Gauss-Chebyshev**.

En concreto, se definen como un conjunto de puntos específicos que son de utilidad en el análisis numérico para obtener aproximaciones precisas para funciones a la hora de realizar análisis de derivación o integración (Mena Gutiérrez (2019)).

Estos provienen de los llamados *Polinomios de Chebyshev* o polinomios de menor máximo absoluto, en donde corresponden a ser ceros de estos, por lo que comparten la propiedad de minimizar el error de aproximación a la hora de interpolar datos (Merino (0607)).

Ahora, el cálculo de estos puntos procede la siguiente ecuación (Departamento de Física Aplicada, UPV/EHU (2025)):

$$x_k = \cos\left(\frac{2k-1}{2N}\pi\right) \tag{4.8}$$

En donde $k = 1, 2, 3, 4...$ corresponde al nodo a encontrar, mientras que $N$ es la cantidad de nodos de Gauss-Chebyshev ($x_k$) a considerar.

Ahora, para hacer uso de estos, se compararán las derivadas numérica (haciendo uso de la derivada de tres puntos en (**??**) y la derivada analítica de la función $f(x) = \cos(x\pi)$ calculada algebraicamente, la cual corresponde a ser:

$$f'(x) = -\pi \sin(x\pi) \tag{4.9}$$

Esto, se hará a través del uso del lenguaje de programación *Python*, haciendo uso de la librería interna de *Numpy*.

A continuación, el código utilizado:

```python
N=  10

def dx_numerica(x,f):
    f = f(x)
    y = np.zeros(len(x))
    for i in range(1,len(x)-1):
        h_k = x[i] - x[i-1]
        h_k_1 = x[i+1] - x[i]
        df_1 = h_k_1/(h_k + h_k_1) * (f[i] - f[i-1])/h_k
        df_2 = h_k/(h_k + h_k_1) * (f[i+1] - f[i])/h_k_1
        y[i] = df_1 + df_2

    return x[1:-1], y[1:-1]

def cos(a):
    return np.cos(np.pi*a)

def gauss_chebyshev(N):
    x = np.zeros(N)
    print(N)
    for i in range(N):
        x[i] = np.cos((2*(i+1) -1)/(2*(N)) * np.pi)
    return x

def dx_analitica(x):
    return -np.sin(np.pi*x)*np.pi

# Obtención de los valores f(x_k) de forma analítica
x_prueba = np.linspace(-1,1,100)
y = dx_analitica(x_prueba)

# Obtención de los valores f(x_k) de forma numérica
x = gauss_chebyshev(N)
x_gen, y_gen = dx_numerica(x,cos)
```

A partir del anterior código presentado, se puede analizar gráficamente los datos entregados para un conjunto de 10 nodos de Gauss-Chebyshev, los cuales son visibles en la Figura 4.1.

Gracias a esto, es posible visibilizar el hecho de que el conjunto de rectas generadas por los puntos de la derivada numérica asemeja a la derivada analítica de la función $f(x) = cos(\pi x)$ (4.9) para los

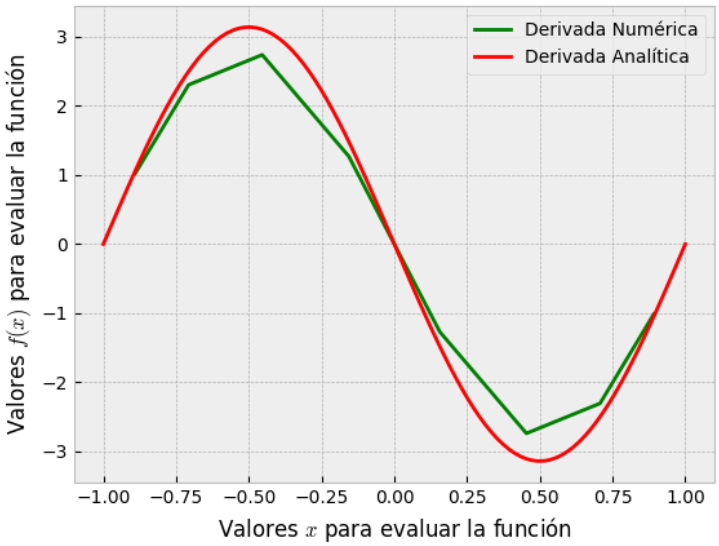


Figura 4.1: Comparación entre derivada analítica y numérica para 10 valores $x_k$. Para esto, se hace uso de la función $f(x) = \cos(\pi x)$ y el método de derivación numérica para tres puntos no equidistantes. Notar que debido a esto, no existe la derivada en los extremos de los conjuntos de puntos $x_i$ puesto que no se puede calcular su respectiva derivada debido a la falta de un punto anterior

valores $x_k$ asignados previamente de los nodos de Gauss-Chebyshev (4.8), esto muestra que la derivada de tres puntos realiza una correcta ejecución de la derivada numérica.

Notar también, que la distancia entre nodos es de menor valor cerca de los extremos del intervalo, esto es debido a la propiedad que poseen los ceros de polinomios de Gauss-Chebyshev, lo que permite evitar efectos indeseados a la hora de analizar e interpolar valores, tales como el fenómeno de Runge[1], que genera oscilaciones de error cerca de los bordes de los intervalos de trabajo.

Finalmente, hay que dar cuenta del hecho que a diferencia de la derivada analítica, la derivada numérica posee una diferencia circunstancial en la amplitud que alcanza (Notar de Figura 4.1), esto es debido al error que se genera debido a la distancia que existe entre los valores $x_k$ (donde para menor diferencia entre cada valor), esto a través de una mayor cantidad de puntos, la amplitud alcanzada tiende a ser igual a la vista en la derivada analítica.

## Conclusiones

Para concluir este capítulo, se demostró la derivada de tres puntos no equidistantes, en donde se mostró que el orden de esta era de $O(h_k h_{k+1})$, lo cuál, para valores de $h_k \approx h_{k+1}$ es sustancialmente mas útil que derivadas del tipo adelantada y retrasada, las cuales poseen orden del tipo $O(h)$[2].

Como siguiente punto, se hizo uso de la derivada numérica anteriormente vista para obtener la derivada de una función $f(x)$, donde se puede observar que esta cumple de forma satisfactoria con los puntos generados para la función a través de los nodos de Gauss-Chebyshev.

A través de este capítulo, se pudo comprender de forma más interactiva la relación entre puntos de una función con la derivada numérica que estos pueden generar, esto, a través del uso de una derivada a puntos no equidistantes.A su vez, se conoció sobre los polinomios de Gauss-Chebyshev y los ceros

[1] Efecto de error generado para puntos equidistantes, consultar referencia (Merino (0607))
[2] Derivadas de este estilo obtenidas a través del uso de la Serie de Taylor (4.1)

de estos, también llamados nodos de Gauss-Chebyshev, elementos que se esperan que en un futuro se puedan volver a utilizar debido a las propiedades que estas poseen para análisis numérico y proyectos de este índole.

## Agradecimientos

Notar la autoría principal de este capitulo por Ignacio Falcón, complementado por comentarios de Alvaro Osses en ámbitos de código para el desarrollo de la derivada y Joaquín Parra en correcciones de cohesión y redacción.

Para el código de las referencias de este capítulo dentro de `referencias.bib`, se hizo uso de Gemini para darles el respectivo formato.

# Capítulo 5

# Ecuaciones de Lotka-Volterra: análisis de la dinámica cazador-presa

Joaquín Parra, Ignacio Falcón, Alvaro Osses

**Fecha de la actividad:** 7 de Octubre de 2025

El **modelo depredador-presa de Lotka-Volterra** representa una interacción entre dos especies cuando una de ellas (el depredador) se alimenta de la segunda (la presa). Si llamamos $D(t)$ a la función que modela la densidad de depredadores y $P(t)$ a la que modela la densidad de presas, entonces el modelo establece que estas funciones se relacionan según el sistema de ecuaciones diferenciales no lineales:

$$\begin{aligned} P'(t) &= \alpha P - \beta PD, \\ D'(t) &= -\gamma D + \delta PD, \\ P(0) &= P_0, \\ D(0) &= D_0. \end{aligned} \tag{5.1}$$

Donde $\alpha, \beta, \gamma, \delta$ son constantes positivas que regulan las tasas de crecimiento, mortalidad e interacción del sistema.
Dada la no linealidad de las ecuaciones, obtener soluciones explicitas puede ser difícil, por lo que abordaremos este modelo mediante el uso de código en `python`.

## 5.1. Reducción de variables

Consideremos el siguiente precedente:

**Teorema** (Teorema Π de Buckingham)**.** *Sean $q_1, q_2, \ldots, q_n$ $n$ variables involucradas en un problema particular, tal que exista una relación funcional de la forma:*

$$f(q_1, q_2, \ldots, q_n) = 0,$$

*entonces las $n$ variables siempre se pueden combinar en productos para formar exactamente $(n - r)$ variables independientes adimensionales, donde $r$ es el rango de la matriz dimensional (matriz donde cada columna representa una variable y cada fila una dimensión). Cada parámetro adimensional es llamado $\Pi_i$, y la relación funcional puede ser reescrita como:*

$$\phi(\Pi_1, \Pi_2, \ldots, \Pi_{n-r}) = 0.$$

| Dimensión | $P$ | $D$ | $t$ | $\alpha$ | $\beta$ | $\delta$ | $\gamma$ | $P_0$ | $D_0$ |
|---|---|---|---|---|---|---|---|---|---|
| $[P]$ | 1 | 0 | 0 | 0 | 0 | -1 | 0 | 1 | 0 |
| $[D]$ | 0 | 1 | 0 | 0 | -1 | 0 | 0 | 0 | 1 |
| $[T]$ | 0 | 0 | 1 | -1 | -1 | -1 | -1 | 0 | 0 |

Cuadro 5.1: Exponentes de las dimensiones de cada variable del problema.

*Los $\Pi_i$ no son únicos, y se construyen fijando $n-r$ variables con distintas dimensiones, formando una base adimensional o $\Pi$ grupo. También se puede construir una nueva base $\Pi_i'$ a partir de una productos de una base $\Pi_i$ (P. Kundu, 2008).*

Haciendo uso del teorema $\Pi$ reduciremos el número de variables de nuestro sistema. Analizando las ecuaciones (5.1), identificamos 3 dimensiones: la densidad de presas [P], la densidad de depredadores [D] y el tiempo [T]. Puesto que las dimensiones a ambos lados de la ecuación deben ser las mismas, las dimensiones de $\alpha, \beta, \gamma$ y $\delta$ se deducen rápido y se exponen en la tabla 5.1.

Vemos que se corresponde con una matriz de tamaño $3 \times 9$, por lo que la matriz dimensional debe ser a lo más de rango 3. En efecto, identificamos la submatriz principal:

$$A[\{P, D, t\}] = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

con $\det(A[\{P, D, t\}]) = 1$, y así la matriz dimensional es de rango 3, por lo que el teorema $\Pi$ asegura que podemos reducir el problema a $9 - 3 = 6$ variables adimensionales. En particular, utilizaremos la base adimensional dada por las constantes del problema:

$$\begin{gathered} \tau = \alpha t, \quad p = \tfrac{\delta}{\gamma} P, \quad d = \tfrac{\beta}{\alpha} D, \quad \mu = \tfrac{\gamma}{\alpha}, \\ p_0 = \tfrac{\delta}{\gamma} P_0, \quad d_0 = \tfrac{\beta}{\alpha} D_0. \end{gathered}$$

Esta selección particular resulta conveniente pues las densidades de población $P, D$ se normalizan con los parámetros que rigen la dinámica del sistema: la tasa de crecimiento de presas $\alpha$, la tasa de mortalidad de los depredadores $\gamma$ y los parámetros $\beta$, $\delta$ asociados a la efectividad de los depredadores en los encuentros con las presas (Anisiu, 2014). El tiempo $t$ se normaliza con la tasa de crecimiento de las presas $\alpha$ y además se tiene el parámetro $\mu$ que es el cociente entre las tasas de mortandad/crecimiento de depredadores y presas respectivamente, lo cual ofrece un parámetro de control al sistema. Estas elecciones permiten reescribir el sistema (5.1) en función de las nuevas variables $\{\tau, p, d, \mu, p_0, d_0\}$ como el sistema de ecuaciones diferenciales (5.2).

$$\begin{aligned} \alpha \frac{\gamma}{\delta} p'(\tau) &= \alpha \frac{\gamma}{\delta} p - \beta \frac{\gamma}{\delta} \frac{\alpha}{\beta} pd, \\ \alpha \frac{\alpha}{\beta} d'(\tau) &= -\gamma \frac{\alpha}{\beta} d + \delta \frac{\gamma}{\delta} \frac{\alpha}{\beta} pd, \\ \frac{\gamma}{\delta} p(0) &= \frac{\gamma}{\delta} p_0, \\ \frac{\alpha}{\beta} d(0) &= \frac{\alpha}{\beta} d_0. \end{aligned} \tag{5.2}$$

Donde se ha hecho uso de $(d/dt) = (d/d\tau) \cdot (d\tau/dt) = \alpha(d/d\tau)$. Es fácil ver que podemos simplificar el sistema al dividir por algunos parámetros a ambos lados e identificar elementos de la base adimensional, obteniendo así el sistema de ecuaciones diferenciales (5.3), donde los únicos parámetros libres son $\mu, p_0$ y $d_0$.

$$\begin{aligned} p'(\tau) &= p(1-d), \\ d'(\tau) &= d\mu(p-1), \\ p(0) &= p_0, \\ d(0) &= d_0. \end{aligned} \tag{5.3}$$

La elección de la base adimensional $\{\tau, p, d, \mu, p_0, d_0\}$ permite que la dinámica de sistemas escalados[1] que tengan la misma tasa de depredación $\beta$ y tasa de eficiencia de depredadores $\delta$ quede representada por el sistema (5.3). Basta con escoger un sistema con parámetros libres arbitrarios (escalados), por ejemplo, $\{2\alpha, 2\gamma, 2P_0, 2D_0\}$ y notar que $\tau = 2\alpha t$, $d = \beta/(2\alpha)D$, $p = \delta/(2\gamma)P$ y $\mu = (2\gamma)/(2\alpha)$. Luego, al replantear el sistema (5.1) tenemos el sistema (5.4).

$$\begin{aligned} 2\alpha\frac{2\gamma}{\delta}p'(\tau) &= 2\alpha\frac{2\gamma}{\delta}p - \beta\frac{2\gamma}{\delta}\frac{2\alpha}{\beta}pd, \\ 2\alpha\frac{2\alpha}{\beta}d'(\tau) &= -2\gamma\frac{2\alpha}{\beta}d + \delta\frac{2\gamma}{\delta}\frac{2\alpha}{\beta}pd, \\ \frac{2\gamma}{\delta}p(0) &= \frac{2\gamma}{\delta}p_0, \\ \frac{2\alpha}{\beta}d(0) &= \frac{2\alpha}{\beta}d_0. \end{aligned} \tag{5.4}$$

Donde se ha utilizado el hecho que $(d/dt) = (d/d\tau)\cdot(d\tau/dt) = 2\alpha(d/d\tau)$. Al simplificar a ambos lados, vemos que recuperamos las ecuaciones (5.3), por lo que en efecto, la dinámica de sistemas escalados en los que se fija los parámetros asociados a la efectividad de los depredadores en los encuentros $(\beta, \delta)$ queda determinada por el sistema (5.3).

## 5.2. Implementación del método leap-frog para obtener soluciones numéricas

El **método del salto de la rana o leap-frog** es un método numérico para resolver sistemas de ecuaciones diferenciales, usualmente en mecánica. Una aplicación clásica es determinar las ecuaciones de movimiento de un cuerpo acelerado, que se obtiene de resolver el sistema de ecuaciones diferenciales (5.5).

$$\begin{aligned} \begin{pmatrix} \dot{x} \\ \dot{v} \end{pmatrix} &= \begin{pmatrix} v \\ a \end{pmatrix}, \\ x(0) &= x_0, \\ v(0) &= v_0. \end{aligned} \tag{5.5}$$

Donde $x = x(t)$, $v = v(t)$ y $a = a(t)$. El método consiste en utilizar un esquema como el (5.6) en el que destaca el uso de "pasos intermedios" para resolver numéricamente el sistema (nosotros lo implementaremos en `python`).

$$\begin{aligned} v(t+h/2) &= v(t) + \frac{h}{2}a(t), \\ x(t+h) &= x(t) + hv(t+h/2), \\ v(t+h) &= v(t+h/2) + \frac{h}{2}a(t+h). \end{aligned} \tag{5.6}$$

[1] Aquellos sistemas en los que los parámetros libres se relacionan mediante una transformación lineal dada por una constante fija.

Las ecuaciones no admiten el método de forma directa: al momento de plantear el esquema, nos encontramos con:

$$\begin{aligned} d(\tau + h/2) &= d(\tau) + \frac{h}{2} d'(\tau), \\ p(\tau + h) &= p(\tau) + h p'(\tau + h/2), \\ d(\tau + h) &= d(\tau + h/2) + \frac{h}{2} d'(t + h). \end{aligned} \tag{5.7}$$

Bajo la consideración que al implementar este método en `python`, surge un problema a primera vista: al ser pasos discretos, no se pueden determinar los pasos intermedios a menos que se tenga una fórmula con argumentos enteros. Para nuestro caso, no poseemos una fórmula para los términos $p'(\tau + h/2)$ que contienen a $p(\tau + h/2)$, por lo que no podemos aplicar de manera directa el método leap-frog. En cambio, realizaremos el siguiente cambio de variables:

$$x = \ln p, \quad y = \ln d,$$

Con lo que nuestro modelo Lotka-Volterra adimensional adopta la forma del sistema (5.8).

$$\begin{aligned} x'(\tau) &= \frac{p'}{p}, \\ y'(\tau) &= \frac{d'}{d}, \\ x(0) &= \ln p_0, \\ y(0) &= \ln d_0. \end{aligned} \tag{5.8}$$

Donde reemplazando los términos $p, d, p', d'$ obtenemos el sistema (5.9).

$$\begin{aligned} x'(\tau) &= 1 - e^y, \\ y'(\tau) &= \mu(e^x - 1), \\ x(0) &= \ln p_0, \\ y(0) &= \ln d_0. \end{aligned} \tag{5.9}$$

Bajo este planteamiento, podemos aplicar un esquema de salto de rana. Notemos que podemos calcular el paso medio mediante una derivada centrada, como se muestra en las ecuaciones (5.10).

$$\begin{aligned} y'(\tau) &= \frac{y(\tau + h/2) - y(\tau - h/2)}{h} + \mathcal{O}(h^2), \\ y(\tau + h/2) &= y(\tau - h/2) + h y'(\tau) + \mathcal{O}(h^3), \\ y(\tau + h/2) &= y(\tau) + \frac{h}{2} y'(\tau) + \mathcal{O}(h^3). \end{aligned} \tag{5.10}$$

Donde se ha utilizado la aproximación $y(t) \approx y(t - h/2) + (h/2)y'(t)$ dada por el método de Euler. Luego, podemos calcular el paso entero de $x$ con una derivada centrada, como se muestra en las ecuaciones (5.11).

$$\begin{aligned} x'(\tau + h/2) &= \frac{x(\tau + h) - x(\tau)}{2 \cdot h/2} + \mathcal{O}(h^2), \\ x(\tau + h) &= x(\tau) + h x'(\tau + h/2) + \mathcal{O}(h^3). \end{aligned} \tag{5.11}$$

Y del método de Euler, podemos aproximar el siguiente paso entero de $y$ como en la ecuación (5.12).

$$y(\tau + h) = y(\tau + h/2) + \frac{h}{2} y'(\tau + h/2). \tag{5.12}$$

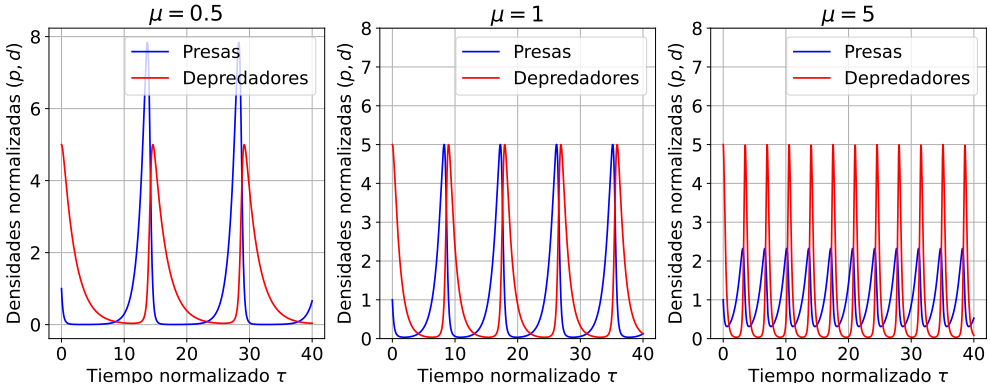


Figura 5.1: Soluciones numéricas al sistema adimensional de Lotka-Volterra a distintos $\mu$, con $p_0 = 1, d_0 = 5$.

Con lo que tenemos listo el esquema de salto de rana. Utilizando las relaciones descritas en (5.9), el esquema de salto de rana que utilizaremos en nuestro código es el sistema (5.13).

$$\begin{aligned} y(\tau + h/2) &= y(\tau) + \frac{h}{2}\mu(1 - e^{x(\tau)}), \\ x(\tau + h) &= x(\tau) + h(e^{y(\tau+h/2)} - 1), \\ y(\tau + h) &= y(\tau + h/2) + \frac{h}{2}\mu(1 - e^{x(\tau+h)}). \end{aligned} \tag{5.13}$$

Con estas ecuaciones, y el siguiente código:

```
def _condiciones_iniciales(t, x0, y0):
    x0 = np.asarray(x0)
    y0 = np.asarray(y0)
    x = np.zeros((len(t),) + x0.shape)
    y= np.zeros((len(t),) + y0.shape)
    x[0] = x0
    y[0] = y0
    return x,y

def dy(a):
    return (np.exp(a)-1)

def leapfrog(dy, x0, y0, t, mu):
    dt = np.diff(t)
    x, y = _condiciones_iniciales(t, x0, y0)
    dy0 = dy(x0)
    for n in range(t.size-1):
        ymedio = y[n] + 0.5 * mu * dt[n] * dy0
        x[n+1] = x[n] + dt[n]*(1- np.exp(ymedio) )
        dy0 = dy(x[n+1])
        y[n+1] = ymedio + 0.5 * mu * dt[n] * dy0
    return  np.exp(x), np.exp(y)
```

podemos obtener la figura 5.1. En ella, observamos como varía la dinámica del sistema al cambiar el parámetro de control $\mu = \gamma/\alpha$ (el cociente entre las tasas de mortandad y nacimiento de depredadores/presas). Vemos que en los 3 casos, los resultados son acordes a la interpretación de $\mu$ como parámetro de control: cuando es menor a 1, es decir, $\alpha > \gamma$ (nacen más presas de lo que mueren

depredadores) tenemos que la densidad de presas supera a la de depredadores al correr el tiempo, y se cumplen pocos ciclos en un determinado período de tiempo. Cuando $\mu = 1$, tenemos un equilibrio entre el nacimiento de presas y la muerte de depredadores, y ambas poblaciones alcanzan el mismo máximo de densidad. Aumenta la cantidad de ciclos en el período de tiempo. Finalmente, con $\mu = 5$, la tasa de mortandad de depredadores es mayor a la tasa de nacimiento de las presas, lo que provoca la gran frecuencia de oscilaciones (ciclos más cortos) pues los ciclos se deben completar en el tiempo de vida de los depredadores, y la amplitud de las oscilaciones de la densidad de depredadores supera a la de las presas. En ningún gráfico la densidad de depredadores sobrepasa las 5 unidades. Esto se analizará más adelante.

## 5.3. Determinación de una invariante y análisis de estabilidad

Es conveniente identificar invariantes del sistema (magnitudes constantes en el tiempo) y analizar su comportamiento en el espacio de fases del mismo. Generalmente, esta invariante suele ser la energía mecánica del sistema. Sin embargo, no estamos trabajando en un contexto mecánico: estamos analizando la dinámica de poblaciones, por lo que algún ajuste deberá ser hecho para sostener este análisis. El sistema entrega soluciones estables en el tiempo, periódicas, lo que puede ser indicio de la existencia de una invariante en el tiempo. En mecánica, el Hamiltoniano $H$ de un sistema actúa como una función potencial en el espacio de fases[2] si este no depende explícitamente del tiempo, y si se aplica a soluciones, deberíamos esperar ver trayectorias cerradas donde $H = cte$.

Primero debemos chequear primero si el sistema de ecuaciones diferenciales respeta una estructura coherente con las ecuaciones de Hamilton (Gonzáles-Santos, 2019). Nótese que solamente hablamos de estructura: en la medida que esto se cumpla, podremos utilizar el método sin preocuparnos si trabajamos en un problema de mecánica o no. Nuestro sistema consta solo de 2 ecuaciones, con lo que las ecuaciones de Hamilton adoptan la forma del sistema (5.14) (Goldstein, 1994).

$$\begin{aligned} p' &= \frac{\partial H}{\partial d}, \\ d' &= -\frac{\partial H}{\partial p}. \end{aligned} \tag{5.14}$$

Donde $H = H(p, d)$ es el Hamiltoniano del sistema, que es una función a determinar. Podemos notar que el sistema Lotka-Volterra adimensional tal y como estaba planteado en (5.3) no admite la existencia de un Hamiltoniano: basta con tomar las derivadas cruzadas y observar que:

$$\begin{aligned} \frac{\partial^2 H}{\partial p \partial d} &= 1 - d, \\ \frac{\partial^2 H}{\partial d \partial p} &= -\mu d. \end{aligned}$$

Y dado que por teorema de Schwarz (Stewart, 2016) las derivadas mixtas deben ser iguales, el sistema no admite un Hamiltoniano (al menos no de forma directa).
Sin embargo, nos damos cuenta que bajo el cambio de variables (5.9), el sistema admite un Hamiltoniano, pues satisface el teorema de Schwarz:

$$\frac{\partial^2 H}{\partial x \partial y} = \frac{\partial^2 H}{\partial y \partial x} = 0.$$

[2] Comentario del Dr. Juan Crisóstomo.

En efecto, de las ecuaciones (5.14) deducimos que:

$$\begin{aligned} H(x,y) &= \int x' \, dy + C(x), \\ C(x) &= -\int y' \, dx + C. \end{aligned} \tag{5.15}$$

Desarrollando, obtenemos:

$$\begin{aligned} H(x,y) &= y - e^y + C(x), \\ C(x) &= \mu(x - e^x) + C, \end{aligned} \tag{5.16}$$

lo que define al Hamiltoniano del sistema como la ecuación (5.17):

$$H(x,y) = y - e^y + \mu(x - e^x), \tag{5.17}$$

o como la ecuación (5.18) con las variables adimensionales:

$$H(p,d) = \ln d - d + \mu(\ln p - p), \tag{5.18}$$

donde se ha fijado $C = 0$ sin pérdida de generalidad. Vemos que en efecto el Hamiltoniano depende implícitamente de $\tau$ y que describe una invariante temporal. En nuestro problema, si bien $p, d$ no representan coordenadas canónicas, admiten una estructura Hamiltoniana con un corchete de Poisson generalizado (Nutku and Y., 1990). En este contexto, la evolución temporal de una función $f(p,d)$ viene dada por:

$$f' = \{f, H\}, \tag{5.19}$$

donde $\{\cdot,\cdot\}$ denota un corchete de Poisson. En particular, para el Hamiltoniano se tiene:

$$\frac{dH}{d\tau} = \{H, H\}. \tag{5.20}$$

Dado que todo corchete de Poisson es antisimétrico (Goldstein, 1994), se cumple:

$$\{H, H\} = -\{H, H\}, \tag{5.21}$$

lo que implica a su vez que:

$$\frac{dH}{d\tau} = 0. \tag{5.22}$$

Por lo tanto, $H(p,d)$ constituye una invariante temporal del sistema. En la figura 5.2 se muestran 100 curvas de nivel para distintos valores de $\mu$. Notar que para todos los casos, tenemos valores estrictamente negativos, los cuales parecen aumentar a medida que nos acercamos al punto $(1,1)$, por lo que decimos que este punto actúa como un máximo local de $H(p,d)$.

Para el análisis de puntos críticos, podemos partir enunciando que nuestro sistema de ecuaciones diferenciales (5.3) es un sistema **autónomo** pues no tiene dependencia explícita del tiempo. Para encontrar puntos críticos basta con estudiar donde $p', q'$ se anulan al mismo tiempo, y podemos utilizar el criterio de estabilidad para sistemas autónomos planos (Zill, 2018) para estudiar la naturaleza de estos puntos. Es fácil ver que los 2 puntos críticos del sistema (5.3) son $(0,0)$ y $(1,1)$. El criterio utiliza los Jacobianos del Hamiltoniano en los puntos para determinar su naturaleza. Estos se calculan en las ecuaciones (5.23) y (5.24). Observamos de inmediato que los autovalores asociados al punto $(0,0)$ son $\lambda_1 = 1, \lambda_2 = -\mu$. Como $\mu$ es definido positivo, tenemos un autovalor con parte real negativa, y de acuerdo al criterio ya mencionado, $(0,0)$ representa un punto de equilibrio inestable del sistema, lo que se aprecia en las hipérbolas que hacen las curvas de nivel cerca del $(0,0)$ en la figura 5.3.

$$J(0,0) = \begin{pmatrix} 1 & 0 \\ 0 & -\mu \end{pmatrix}, \tag{5.23}$$

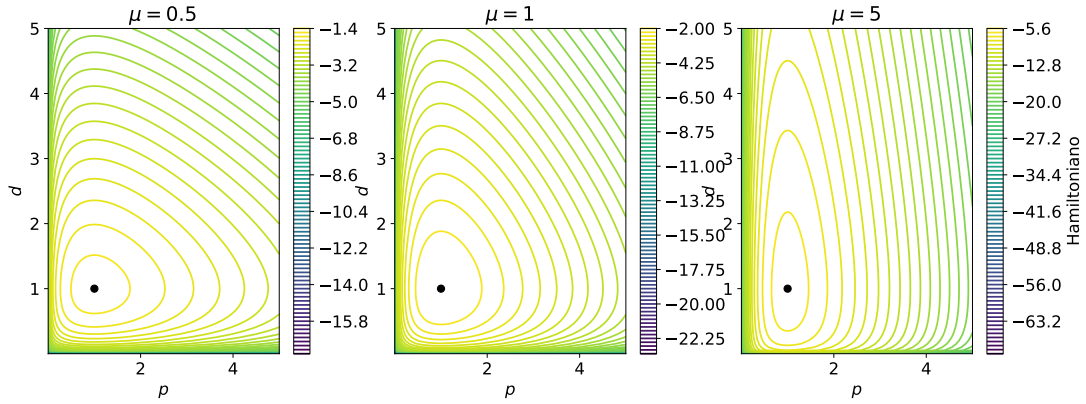


Figura 5.2: 100 curvas de nivel para $H$ a distintos $\mu$. El punto negro es el punto de equilibrio estable $(1, 1)$.

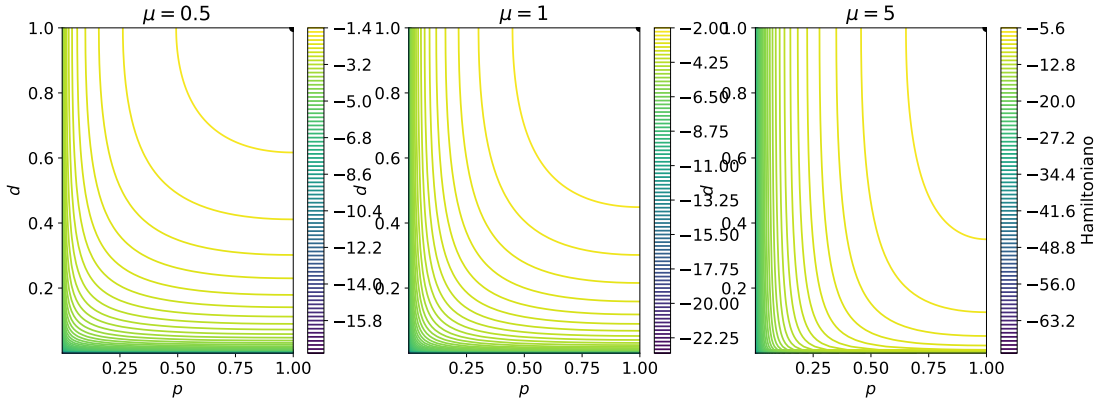


Figura 5.3: Zoom a cada espacio de fase cerca del $(0, 0)$.

$$J(1,1) = \begin{pmatrix} 0 & -1 \\ \mu & 0 \end{pmatrix}. \tag{5.24}$$

Sin embargo, los autovalores para $(1, 1)$ son $\lambda_{1,2} = \pm i\sqrt{\mu}$, y el criterio no es concluyente, solo se clasifica al punto como un centro, por lo que podemos esperar que se formen órbitas/trayectorias cerradas alrededor de este punto . Para determinar su naturaleza, analizaremos este punto mediante funciones de Liapunov.

El criterio consiste en determinar una función $V(x, y)$ (de Liapunov) a valores reales con derivadas parciales continuas en una vecindad de un punto crítico tal que: sí $(x_0, y_0)$ es un punto crítico aislado de nuestro sistema, $V(x_0, y_0) = 0$ y $V(x, y) > 0$ sobre el dominio en el que se trabaja, se tienen los casos:

1. Sí $\frac{\partial V}{\partial x}F + \frac{\partial V}{\partial y}G \leq 0$ en el dominio , entonces $(x_0, y_0)$ es estable.
2. Sí $\frac{\partial V}{\partial x}F + \frac{\partial V}{\partial y}G < 0$ en el dominio, entonces $(x_0, y_0)$ es asintóticamente estable.
3. Sí $\frac{\partial V}{\partial x}F + \frac{\partial V}{\partial y}G > 0$ en el dominio, entonces $(x_0, y_0)$ es inestable (Osses, 2005).

Donde $F, G$ serían en este caso $p', d'$. Podemos construir nuestra función $V$ a partir del Hamiltoniano $H$ ya descrito. En particular, podemos ver que $V(p, d) = -H(p, d) + H(1, 1)$ satisface las hipótesis

necesarias para aplicar los criterios. Utilizando (5.22), vemos que la derivada temporal de $V$ también es nula. Con esto, podemos asegurar que $(1, 1)$ es un punto de equilibrio estable. Esto se chequea al observar que las curvas de nivel parecieran estar "centradas" alrededor de este punto, como se observa en la figura 5.2, pues $(1, 1)$ describe un máximo para el Hamiltoniano, y como su derivada temporal es 0, sumado a que su Jacobiano toma valores netamente imaginarios, es de esperar que actúe como un centro de las curvas de nivel.

Finalizando, podemos analizar el rol que juegan las condiciones iniciales en todo esto. La figura 5.4 muestra variaciones de condiciones iniciales a distintos $\mu$. Vemos que difieren bastante de la figura 5.1, debido fundamentalmente a que se varían las condiciones iniciales de la figura anterior. Sin embargo, podemos darnos cuenta que cuando una de las condiciones iniciales es 1, induce que la otra magnitud se mantenga estable, es decir, su amplitud máxima es la misma que su condición inicial. Ya lo veíamos en 5.1, pero queda expuesto en esta nueva figura. Esto tiene una explicación simple: al mirar las ecuaciones (5.3), vemos que las derivadas de cada función se anulan cuando una de las coordenadas es 1 (se anulan ambas en el punto de equilibrio). Luego, si una condición inicial de $p$ ó $d$ es 1, induce que la otra función tenga un punto crítico en $\tau = 0$. La naturaleza de este punto se determina rápidamente con el criterio de la segunda derivada:

$$\begin{aligned} p''_{d_0=1}(0) &= \mu p_0(1 - p_0), \\ d''_{p_0=1}(0) &= \mu d(1 - d_0). \end{aligned} \tag{5.25}$$

Esto se deduce de las ecuaciones (5.3) y asumiendo que la otra condición inicial es 1. Analizando, vemos que la naturaleza de cada función en $\tau = 0$ cuando la otra posee condición inicial 1, queda determinada por su propia condición inicial: si es mayor a 1, entonces la condición inicial determina un máximo (pues la segunda derivada sería negativa), caso contrario, un mínimo. En la figura 5.4, solo se trabajó con condiciones mayores a 1, por lo que solo se definen máximos. Finalmente, los sistemas cazador-presa de Lotka-Volterra suelen ser periódicos, por lo que estos puntos definen máximos absolutos para nuestras funciones.

En el espacio de fases, las condiciones iniciales determinan las trayectorias, pues $H(p, d) = C$ para el sistema, y una condición inicial impone $H(p_0, d_0) = C_0$ diferente para cada par $(p_0, d_0)$. Observamos que en la figura 5.5 las condiciones iniciales impuestas son exclusivas de ciertas curvas de nivel, en la cual se resalta la evolución de un estado inicial, lo que rompe con la mirada global de los espacios de fases: nos centramos en casos particulares, cada uno con su propias amplitudes y frecuencias, pero respetando todos la misma dinámica. Un análisis rápido a las figuras 5.2 y 5.5 muestra que bajo mismas condiciones iniciales, en el espacio de fases, un mayor valor de $\mu$ genera curvas de nivel más delgadas en el eje $p$ y más amplias en el eje $d$, y caso contrario cuando disminuye $\mu$. Esto se corresponde con las gráficas 5.4, pues al aumentar el $\mu$, tiende a crecer la cantidad de depredadores y al disminuir, aumenta la de presas.

## Conclusiones

Hemos realizado un análisis completo a la dinámica cazador-presa bajo el modelo de Lotka-Volterra, estudiando alternativas de resolver las ecuaciones, la implementación de métodos numéricos para resolverlo y análisis sobre la estabilidad y propiedades de las ecuaciones. La capacidad de poder trabajar un problema de dinámica de poblaciones utilizando herramientas de la mecánica evidencia el alcance de los análisis que podemos realizar con nuestros conocimientos en física, siempre respaldado por un buen código coherente con nuestro problema.

## Agradecimientos

Este capítulo fue escrito principalmente por Joaquín Parra, con revisiones tanto de código como

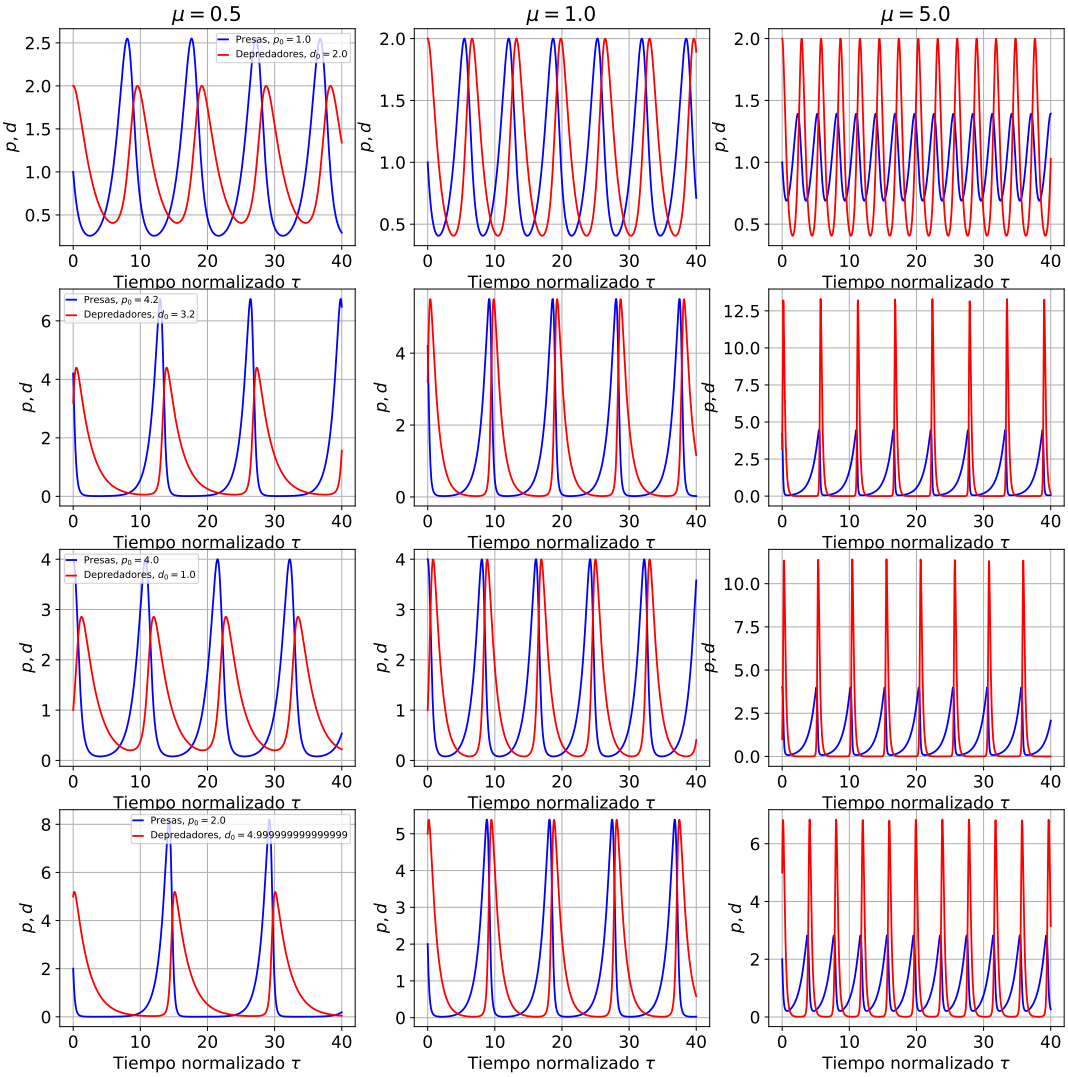


Figura 5.4: Soluciones numéricas al sistema adimensional de Lotka-Volterra a diferentes condiciones iniciales y distintos $\mu$. Figuras en la misma fila comparten condiciones iniciales.

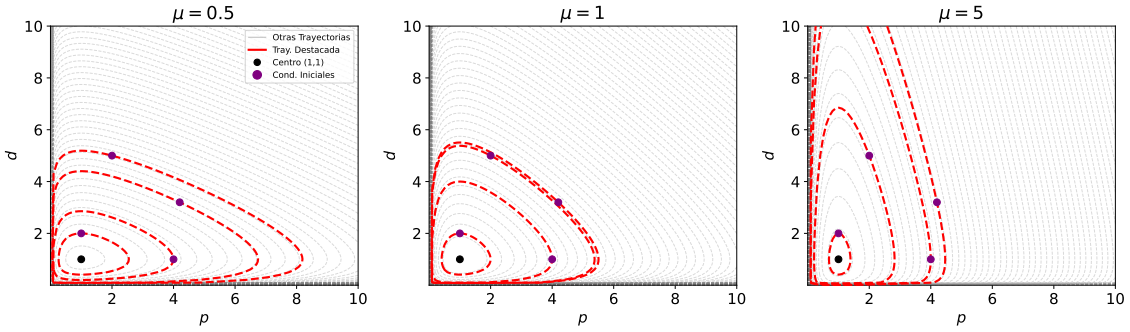


Figura 5.5: Espacio de fases para las distintas condiciones iniciales de la figura 5.4.

de contenido de parte de sus compañeros de trabajo Alvaro Osses e Ignacio Falcón, quienes también contribuyeron a buscar referencias pertinentes al capítulo y a ajustar el desarrollo del Teorema Pi. Es necesario mencionar los comentarios del profesor del curso, Dr. Roberto Navarro, quien entregó retroalimentación crucial para este problema, cuyo repositorio de códigos para el curso sirvió de apoyo para este trabajo, y comentarios del profesor Dr. Juan Crisóstomo útiles para abordar parte de la interpretación mecánica del problema. Finalmente, en materia de código, se agradece al compañero de curso Israel Bravo por entregar guía en ciertos desarrollos del código, y se ha utilizado Gemini (IA de Google) para realizar el último gráfico del capítulo.

# Capítulo 6

# Método de Shu-Osher aplicado a problema de cinemática

Joaquín Parra, Ignacio Falcón, Alvaro Osses

**Fecha de la actividad:** 9 de Noviembre de 2025

En este capítulo analizaremos y plantearmos un método para resolver ecuaciones diferenciales numericamente basado en el método de Runge-Kutta útil para situaciones donde existe roce. Para ello, implementaremos un método de Runge-Kutta genérico en `python`, para así instanciarlo con componenetes de una matriz de Runge-Kutta convenientes, junto con los respectivos pesos y nodos temporales. Luego aplicaremos este método a la resolución de un problema de cinemática.

Para ello, tendremos en cuenta que un método de Runge-Kutta genérico se $s$ etapas se escribe de la siguiente manera:

$$K_i = hf\left(t_n + c_i h, y_n + \sum_{j=1}^{i-1} a_{ij} K_j\right), \qquad y_{n+1} = y_n + \sum_{i=1}^{s} b_i K_i \tag{6.1}$$

Donde $h > 0$ es el paso de tiempo, de manera que $t_n = t_0 + nh$, $f$ es la función que define al problema $y'(t) = f(t, y)$, y los coeficientes $a_{ij}$, $b_i$, $c_i$ son los componentes de la matriz de Runge-Kutta, los pesos y los nodos temporales, respectivamente. Estos se definen a través de una tabla de Butcher, como la del cuadro 6.1

| | | | | |
|---|---|---|---|---|
| $c_1$ | $0$ | | | |
| $c_2$ | $a_{21}$ | $0$ | | |
| $\vdots$ | $\vdots$ | $\ddots$ | $\ddots$ | |
| $c_s$ | $a_{s1}$ | $\cdots$ | $a_{s,s-1}$ | $0$ |
| | $b_1$ | $b_2$ | $\cdots$ | $b_s$ |

Cuadro 6.1: Tabla de Butcher para un método de Runge-Kutta genérico

## 6.1. Método de Shu-Osher

El método de Shu y Osher es un esquema de Runge-Kutta de tercer orden, útil para resolver sistemas que presenten roce (Shu (1988)). Podemos implementar este método utilizando su tabla de Butcher (cuadro 6.2).

Implementamos el método explícitamente de la siguiente manera:

```
def shu_osher_exp(f, x0, tmax, dt, t0 = 0, **kwargs):
    x0 = np.asarray(x0)#Considera el caso en que x0 es un escalar, y lo convierte en arreglo

    t = np.arange(t0, tmax, dt)#Arreglo de tiempo
    x = np.zeros((*t.shape, *x0.shape))#Se desempaquetan los iterables
    x[0] = x0

    #Se aplica explicitamente el método de Shu-Osher.
    for i in range(t.size - 1):
        K1 = dt*f(t[i], x[i], **kwargs)
        K2 = dt*f(t[i] + 1*dt, x[i] + 1*K1)
        K3 = dt*f(t[i] + 0.5*dt, x[i] + 0.25*K1 + 0.25*K2)
        x[i+1] = x[i] + ((1/6)*K1 + (1/6)*K2 + (2/3)*K3)
    return t, x
```

Donde definimos explícitamente las expresiones para cada $K$, obtenidas desarrollando la ecuación 6.1, teniendo en cuenta que `x0` y `dt` representan las condiciones inciales y el valor de $h$, respectivamente.

Por otro lado, podemos implementar un método general que tome una tabla de Butcher cualquiera y genere el método asociado a esta, de manera que podamos instanciar este codigo con la tabla del cuadro 6.2 para obtener el método de Shu y Osher. Para ello, consideramos el siguiente código:

```
def RK_gen(f, x0, tmax, dt, butch, t0 = 0, **kwargs):
    x0 = np.asarray(x0) #Considera el caso en que x0 es un escalar, y lo convierte en arreglo

    t = np.arange(t0, tmax, dt) #Arreglo de tiempo
    x = np.zeros((*t.shape, *x0.shape)) #Se desempaquetan los iterables
    x[0] = x0  #Definimos la condición inical

    shape = np.shape(butch)

    #Obtenemos los pesos y nodos temporales para aplicar el metodo de RK
    a = butch[0:shape[0]-1, 1:]
    b = butch[shape[0] - 1, 1:]
    c = butch[:, 0]

    #Aplicamos la forma general para el metodo de RK
    for i in range(t.size - 1):
        K = np.zeros((np.size(b), *x0.shape)) #Creamos un arreglo para almacenar los K
        sum = 0
        for k in range(0, np.size(b)):
            K[k] = dt*f(t[i] + c[k]*dt, x[i] + sum, **kwargs)
            for j in range(1, k):
              sum = sum + a[k-1, j-1]*K[j+1]

        sum2 = 0

        for p in range(1, np.size(b)+1):
            sum2 = sum2 + b[p-1]*K[p-1]

```

| | | | |
|---|---|---|---|
| 0 | | | |
| 1 | 1 | | |
| 1/2 | 1/4 | 1/4 | |
| | 1/6 | 1/6 | 2/3 |

Cuadro 6.2: Tabla de Butcher asociada al método de Shu y Osher

```
        x[i+1] = x[i] + sum2
    return t, x
```

Donde implementamos explícitamente las fórmulas (6.1), teniendo en cuenta que `butch` es la tabla de Butcher a utilizar entregada como un arreglo de `numpy`.

Teniendo esta función, podemos instanciarla con el cuadro 6.2 para obtener el método de Shu y Osher. Este proceso es equivalente a implementar el método explicitamente, por lo que esta será la función utilizada para tratar ecuaciones diferenciales a lo largo de este capítulo.

## 6.2. Movimiento de proyectil

Considerando las ecuaciones

$$\vec{v}' = \vec{g} - \mu||\vec{v}||\vec{v}, \quad \vec{r}(0) = \vec{r}_0, \quad \vec{v}(0) = \vec{v}_0 \tag{6.2}$$

Primero, notamos que como $||\vec{v}||\vec{v}$ tiene unidades de $[\mathrm{m}^2/\mathrm{s}^2]$ y $\vec{g}$ tiene unidades de $[\mathrm{m}/\mathrm{s}^2]$, es necesario que $\mu$ tenga unidades de $[1/\mathrm{m}]$ para que estos terminos puedan sumarse y la ecuación tenga sentido. Luego, teniendo en cuenta que $\vec{v}' = \frac{d\vec{v}}{dt}$, tenemos que para cada componente vectorial, el problema consiste de 7 variables, las cuales, como muestra el cuadro 6.3, están en terminos de 2 unidades fundamentales, longitud ($L$) y tiempo ($T$). Luego, de acuerdo al teorema $\pi$ de Buckingham, es posible normalizar el sistema de manera que se eliminen tantas variables como hay unidades fundamentales, es decir, es posbile eliminar 2 variables.

| Variable | $t$ | $v$ | $v_0$ | $g$ | $\mu$ | $r$ | $r_0$ |
|---|---|---|---|---|---|---|---|
| Unidad | $T$ | $LT^{-1}$ | $LT^{-1}$ | $LT^{-2}$ | $L^{-1}$ | $L$ | $L$ |

Cuadro 6.3: Variables del problema con sus respectivas unidades

Ahora, si normalizamos el sistema con respecto a las constantes $v_0 = ||\vec{v}_0||$ y $g = ||\vec{g}||$, notando que las cantidades $v_0/g$ y $v_0^2/g$ tienen unidades de tiempo y espacio, respectivamente, podemos crear el $\Pi$-grupo:

$$\begin{gathered}\Pi_1 = t \cdot \left(\frac{g}{v_0}\right) = \tau, \quad \Pi_2 = v \cdot \left(\frac{1}{v_0}\right) = \nu, \quad \Pi_3 = v_0 \cdot \left(\frac{1}{v_0}\right) = 1, \quad \Pi_4 = g \cdot \left(\frac{1}{g}\right) = 1, \\ \Pi_5 = \mu \cdot \left(\frac{g}{v_0^2}\right) = \tilde{\mu}, \quad \Pi_6 = r \cdot \left(\frac{g}{v_0^2}\right) = \rho, \quad \Pi_7 = r_0 \cdot \left(\frac{g}{v_0^2}\right) = \rho_0\end{gathered} \tag{6.3}$$

Donde eliminamos las constantes $v_0$ y $g$, de manera que el problema original puede reescribirse como:

$$\frac{d\vec{\nu}}{d\tau} = \frac{1}{g}\vec{g} - \tilde{\mu}||\vec{\nu}||\vec{\nu}, \quad \vec{\rho}(0) = \vec{\rho}_0, \quad \vec{\nu}(0) = \frac{1}{v_0}\vec{v}_0 = \vec{\nu}_0 \tag{6.4}$$

Donde $\vec{\nu}_0$ es un vector unitario en la dirección de la velocidad inicial y $\frac{1}{g}\vec{g} = \hat{g}$ es el vector unitario en dirección de la aceleración de gravedad, en este caso $\hat{g} = -\hat{y}$.

Ahora, consideramos el problema planteado en la ecuación (6.4) teniendo en cuenta que $\vec{\rho}(\tau) = x(\tau)\hat{x} + y(\tau)\hat{y}$, $\hat{g} = -\hat{y}$ y notando además las condiciones iniciales:

$$\vec{\rho}(0) = \vec{0}, \quad \vec{\nu}(0) = (\cos(\alpha)\hat{x} + \sin(\alpha)\hat{y}), \tag{6.5}$$

y el hecho de que $\frac{d\vec{\rho}}{d\tau} = \vec{\nu}$.

Tenemos el siguiente sistema de ecuaciones diferenciales normalizado:

$$\frac{d\nu_x}{d\tau} = -\tilde{\mu}||\vec{\nu}||\nu_x, \quad \frac{d\nu_y}{d\tau} = -1-\tilde{\mu}||\vec{\nu}||\nu_y, \quad \frac{d\rho_x}{d\tau} = \nu_x, \quad \frac{d\rho_y}{d\tau} = \nu_y, \\ \vec{\rho}(0) = \vec{0}, \quad \vec{\nu}(0) = \cos(\alpha)\hat{x} + \sin(\alpha)\hat{y}. \tag{6.6}$$

Ahora, podemos resolver este sistema utilizando el método de Shu y Osher planteado anteriormente, para ello, consideramos el siguiente código, donde resolvemos el sistema para distintos valores de $\mu$ y $\alpha$. Notar que esto es un sistema de 4 ecuaciones diferenciales, una para cada componente del vector de posición y velocidad.

```
#Usamos esta tabla de Butcher para utilizar el método de Shu-Osher
butcher = np.array([[0, 0, 0, 0],
                    [1, 1, 0, 0],
                    [0.5, 0.25, 0.25, 0],
                    [0, 1/6, 1/6, 2/3]])

#Definimos el sistema a resolver
def fullvec(t, x, mu):
    posx, posy, velx, vely = x
    mag = np.hypot(velx, vely)
    return np.array([velx, vely, -mu*mag*velx, -1-mu*mag*vely])

def caso(mu, alpha): #Definimos una solución genérica para iterar con distintos valores de
                     #mu y alpha.
    pos0x = 0
    pos0y = 0
    vel0x = np.cos(alpha)
    vel0y = np.sin(alpha)

    t, X = RK_gen(fullvec, x0 = [pos0x, pos0y,vel0x, vel0y], tmax = 2, dt = 0.001,
    butch = butcher, mu = mu)

    posx, posy, velx, vely = X[:, 0], X[:, 1], X[:, 2], X[:, 3]

    posytrue = posy[posy > 0]
    posxtrue = posx[:posytrue.size]

    return posxtrue, posytrue


for mu in np.linspace(0, 1, 6): #Graficamos para distintos valores de mu
   x, y = caso(mu, np.pi/4)

for alpha in np.linspace(0, np.pi/2, 6): #Graficamos para distintos valores de alpha
  x, y = caso(0.6, alpha)

for alpha in np.linspace(0, np.pi/2, 7): #Graficamos para distintos valores de alpha en
                                         #condiciones ideales
  x, y = caso(0, alpha)
```

Notando que la función `fullvec(t, x)` representa el sistema normalizado de la ecuación (6.6). De este modo, generamos los gráficos de las figuras 6.1, 6.2 y 6.3. Notar además que al ejecutar el código anterior, se le hizo una pequeña modificación a la función `RK_gen` para que esta deje de integrar la función una vez que la posición vertical sea 0, es decir, cuando el proyectil vuelve a tocar el suelo.

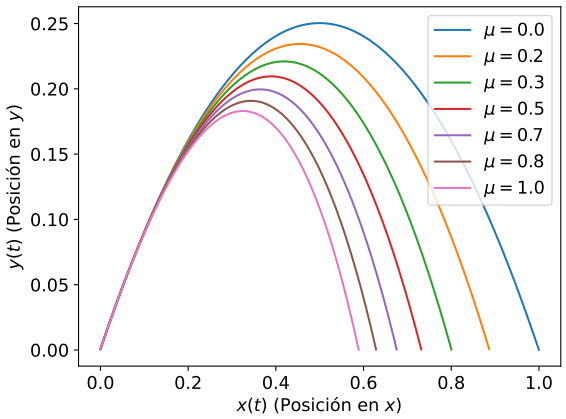


Figura 6.1: Comparación de trayectorias en $x(t)$ e $y(t)$ para distintos valores de $\mu$ con $\alpha = \pi/4$.

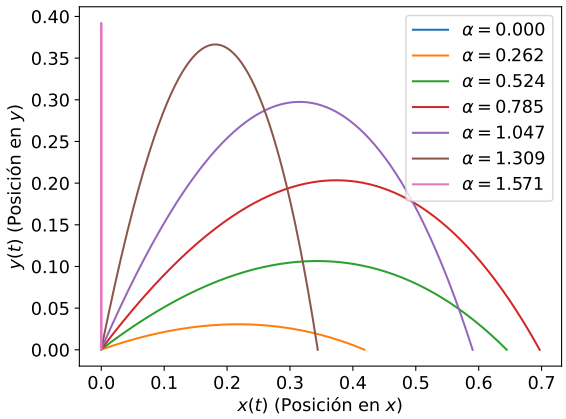


Figura 6.2: Comparación de trayectorias en $x(t)$ e $y(t)$ para distintos valores de $\alpha$ con $\mu = 0.6$.

```
1
2  def RK_gen(f, x0, tmax, dt, butch, t0 = 0, **kwargs):
3  [...37 líneas...]
4
5        for p in range(1, np.size(b)+1):
6            sum2 = sum2 + b[p-1]*K[p-1]
7
8        x[i+1] = x[i] + sum2
9
10       if x[i+1][1]  <= 0:
11           return t,x
12
13   return t, x
```

De la figura 6.1, se puede observar que la distancia en $x$ que recorre el proyectil es mayor a medida que disminuye el valor de $\mu$, siendo este máximo cuando $\mu = 0$, es decir, cuando el movimiento no presenta roce. Por otro lado, de la figura 6.2, se puede observar que la distancia en $x$ es máxima cuando $\alpha = 0.785 \approx \pi/4$, mientras que esta es nula cuando $\alpha = 1.571 \approx \pi/2$, pues aquí el objeto se lanza completamente hacia arriba. Finalmente, cuando $\alpha = 0$, el objeto nunca cambia su posición vertical, por lo que su trayectoria no puede observarse en la figura.

Finalmente, observamos que en la solución exacta, cuando $\mu = 0$, el desplazamiento horizontal es máximo, tal como muestra la figura 6.1. Es más, observando la figura 6.3, vemos que la configuración $\mu = 0$ y $\alpha = \pi/4$ produce una trayectoria que maximiza simultaneamente el desplazamiento vertical y horizontal. Es más, para cualquier valor de $\mu \neq 0$, el desplazamiento horizontal se maximiza con $\alpha = \frac{\pi}{4}$.

## Conclusiones

Para concluir, hemos visto que es posible, y sumamente conveniente, crear un método de Runge-Kutta genérico que pueda instanciarse con cualquier tabla de Butcher. Además, hemos aplicado un método conveniente a un problema de física de variables normalizadas, pudiendo así observar directamente los efectos del roce con aire en el movimiento de un proyectil.

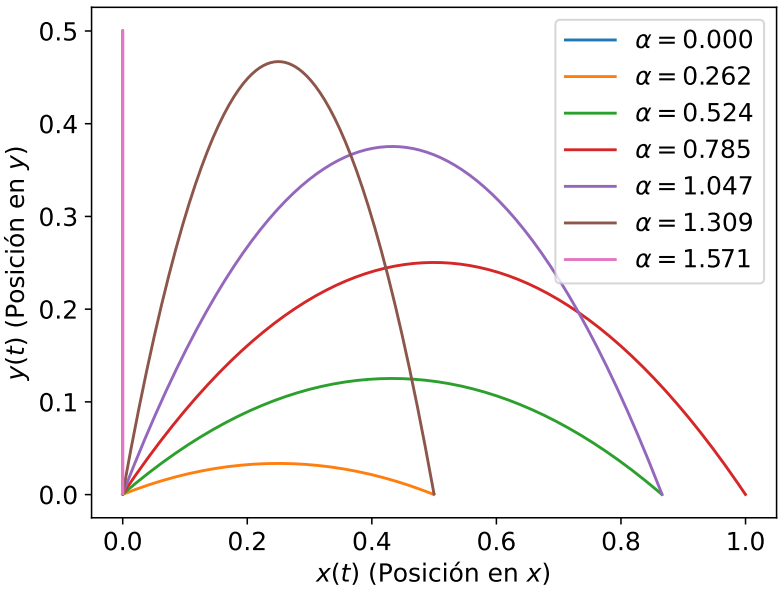


Figura 6.3: Trayectoria ideal con $\mu = 0$ y distintos valores de $\alpha$.

## Agradecimientos

Para finalizar, este capítulo fue escrito principalmente por Alvaro Osses, con contribuciones de Joaquín Parra en la aplicación del Teorema-II de Buckingham, e Ignacio Falcón en la redacción.
Para la creación de este capítulo no se utilizaron herramientas de inteligencia artificial.

# Capítulo 7

# Método de la Secante para encontrar ceros de una función

Ignacio Falcón, Joaquín Parra, Alvaro Osses

**Fecha de la actividad:** 26 de Noviembre del 2025

Para el presente capítulo, se analizarán uno de muchos métodos numéricos para calcular los ceros (valores $x = c$, tal que $f(c) = 0$) de una función $f(x)$, con el nombre de **Método de la secante**, además, se hará uso de una función polinómica con el objetivo de demostrar su uso.

Luego de ello, se analizará el error generado por el método anteriormente mencionado, junto a una comparación entre los diferentes métodos conocidos.

## 7.1. Método de la secante

Para comenzar, se debe tener presente el funcionamiento de este método. Este se basa en estimar los valores "ceros" de una función a partir del punto donde una recta secante (recta que atraviesa dos puntos en de una curva) atraviesa el eje horizontal (Eje X o $y = 0$).

Para hacer uso, se deben conocer la función a utilizar ($f(x)$), y dos valores pertenecientes a la variable independiente de la función $x_{n-1}$ y $x_n$ tal que $f(x_{n-1}) \neq 0 \neq f(x_n)$.

A partir de ello, se define el método de la secante como:

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n) \tag{7.1}$$

Es importante notar el hecho de que este método no asegura que el valor cero se encuentre en el intervalo $[x_{n-1}, x_n]$, es por esto que se debe asegurar de alguna forma que este se encuentre dentro del intervalo a trabajar a través de métodos como el del *Teorema de los ceros de Bolzano*.

Para realizar su uso, vamos a considerar una función polinómica definida como:

$$f(x) = x^3 - 5x \quad , \quad -3 \leq x \leq 3 \tag{7.2}$$

Ahora, para realizar la búsqueda de los ceros de la función a través de métodos numéricos, se procederá a utilizar `Python` en `Numpy`, a través de los siguientes pasos.

- Se considera en dividir el intervalo de trabajo ($-3 \leq x \leq 3$) en 10 puntos equidistantes, para esto se hace uso de la función `numpy.linspace()`.

- A continuación, se hace uso del teorema de valor intermedio, en particular, el teorema de los ceros de Bolzano, el cuál se enuncia como:

  *Para una función continua en un intervalo cerrado* $[a, b]$*, tal que* $f(a) \cdot f(b) < 0$*, entonces debe existir* $c \in [a, b]$ *tal que* $f(c) = 0$

  Esto se hará uso considerando y analizando los límites de cada intervalo antes generado, haciendo uso de un ciclo `if` que evalua el valor del producto mencionado por el teorema.

- En caso de que se cumpla el teorema, se considerará los valores extremos del intervalo como puntos $x_{n-1}$ y $x_n$ para introducir en el método de la secante (7.1) hasta que se haya de cumplir la condición de convergencia de:

$$|x_{n+1} - x_n| < 10^{-5} \tag{7.3}$$

Finalmente, para cada raíz se entrega la aproximación numérica y la cantidad de iteraciones que se realizaron hasta superar la tolerancia (7.3).

A continuación, se muestra el código utilizado, el cuál sigue las instrucciones antes mencionadas junto a la función a trabajar (7.2).

```
tolerancia = 1e-5

f = lambda x: x**3 -5*x # Se define la función a utilizar

    # Se define el método de la secante
def met_secante(f,a,b,tolerancia):
    zeros = []
    int = np.linspace(a,b,10)
    print(int)
    for i in range(int.size-1):

        if f(int[i])*f(int[i+1]) <= 0:
            a,b = int[i:i+2]
            fa,fb = f(a), f(b)
            m = 0
            while (abs(fb*(b-a)/(fb-fa)) >=tolerancia):
                bold=b
                b = b - fb*(b-a)/(fb-fa)
                fa=fb
                a=bold
                fb=f(b)
                m +=1
            print("Existe un cero",b,"con iteraciones de",m,"veces")
            zeros.append(b)
    return zeros

met_secante(f,-3,3,tolerancia)
```

Ahora, para encontrar los ceros de la función (7.2), es necesario enunciar el *Teorema Fundamental del Álgebra*:

*El Teorema Fundamental del Álgebra establece que todo polinomio de grado mayor que cero, tiene al menos una raíz compleja. En concreto, para la extensión del cuerpo de los números complejos, un polinomio de grado N siempre tendrá N raíces o valores tal que este hagan cero al polinomio.*

Por esto, considerando que la función es de grado tres, se puede afirmar que existen tres soluciones $c$ tal que $f(c) = 0$ o también, que existen tres ceros para la función, ahora, es necesario analizar si estos son de naturaleza real o compleja, puesto que se desea trabajar en el cuerpo de los números reales.

Para esto, usando métodos algebráicos, se pueden obtener estos valores a partir de (7.4):

| Número del cero | Cero analítico | Cero numérico | N° Iteración | Error Absoluto |
|---|---|---|---|---|
| 1 | -2.236067977 | -2.2360679871216194 | 5 | 0.00000001 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 2.236067977 | 2.23606776287524 | 4 | 0.00000021 |

Cuadro 7.1: Valores entre los cálculos de ceros de forma analítica y numéricamente, esto, aproximando los valores respectivos de $-\sqrt{5}$ y $\sqrt{5}$.

$$0 = x(x^2 - 5) \Rightarrow x = 0, x = \sqrt{5}, x = -\sqrt{5} \tag{7.4}$$

Por tanto, se pueden expresar los cálculos realizados [1] como la tabla de datos donde se puede notar que estos ceros pertenecen al conjunto de los números reales (7.1):

En donde se puede observar que los datos poseen un error absoluto de menor orden en comparación a los valores de los ceros. También, es notable la diferencia que existe en el número de iteraciones realizadas para cada valor.

Esto, se debe al hecho de cómo realiza el Método de la Secante el cálculo de los ceros, puesto que en este caso, al separar el intervalo $[-3, 3]$ con diez puntos equiespaciados, se puede notar que el primer cero se encuentra más cercano al inicio del segundo intervalo $[-2.3333, -1.666]$, por lo que se requiere de más cantidad de iteraciones del algoritmo para que el valor b se acerce al cero con la suficiente tolerancia. Esto, a diferencia del último cero, el cual se encuentra en el intervalo $[1.666, 2.333]$, donde se puede notar que su cero se encuentra considerablemente más cerca del extremo final del intervalo y, en consecuencia, este requiere de una menor cantidad de iteraciones para que el valor "b" se acerque al cero con la tolerancia requerida.

Finalmente, notar que para el cero en $x = 0$, los extremos del intervalo que contienen a este poseen el mismo valor absoluto y en consecuencia, en la primera iteración el método de la secante genera que el valor "b" sea cero y en consecuencia, se alcanza la tolerancia automáticamente.

A su vez, es posible visualizar el gráfico y su relación los ceros de este, los cuales intersectan al eje X ($y = 0$), tal como se puede observar en 7.1):

## 7.2. Análisis de error para el método de la secante

### 7.2.1. Demostración de la proporción del error

A continuación, se procederá a analizar el error presente en el método de la secante a través de métodos algebráicos en función de sus términos componentes.

Por ello, se procederá a definir el error existente entre el valor $x_n$ y el valor del cero en la función, denotado $x^*$.

$$\varepsilon_n = x_n - x^* \tag{7.5}$$

Usando esto, se busca demostrar que:

$$\varepsilon_{n+1} \propto \varepsilon_n \varepsilon_{n-1} \tag{7.6}$$

Para comenzar, usando el la serie de Taylor centrada en el cero ($x^*$), tenemos que para $f(x_n)$ y $f(x_{n-1})$

$$f(x_n) = f(x^*) + f'(x^*) \cdot \varepsilon_n + \frac{1}{2} f''(x^*)\varepsilon_n^2 + \frac{1}{3!} f'''(\xi_1)\varepsilon_n^3 \tag{7.7}$$

[1] Para realizar el cálculo de la aproximación a $\sqrt{5}$ se hizo uso de la calculadora científica de Desmos Desmos, Inc. (2025)

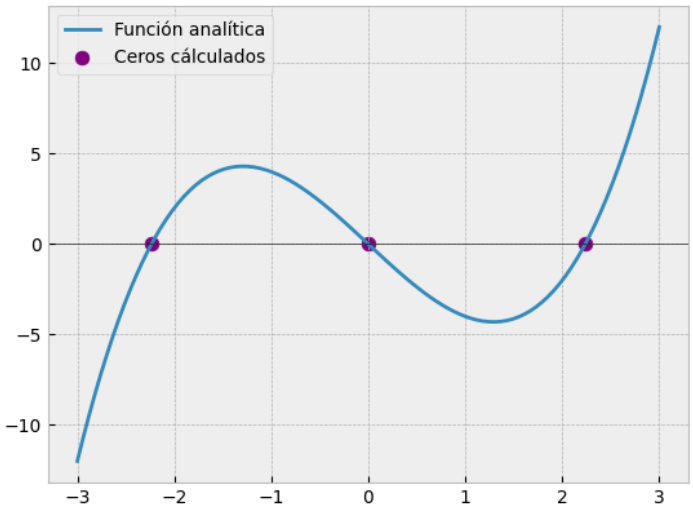


Figura 7.1: Ceros cálculados en la función analítica $f(x)$

$$f(x_{n-1}) = f(x^*) + f'(x^*) \cdot \varepsilon_{n-1} + \frac{1}{2} f''(x^*)\varepsilon_{n-1}^2 + \frac{1}{3!} f'''(\xi_1)\varepsilon_{n-1}^3 \tag{7.8}$$

Donde $\xi_1$ y $\xi_2$ son valores que consideran el residuo de la serie infinita de Taylor.

A partir de esto, se puede expresar del método de la secante (7.1), la expresión con los términos $x_n$ y $x_{n-1}$ de la siguiente forma:

$$x^* + \varepsilon_{n+1} = x^* + \varepsilon_n - \frac{(x^* + \varepsilon_n) - (x^* + \varepsilon_{n-1})}{f(x_n) - f(x_{n-1})} f(x_n) \tag{7.9}$$

Ahora, restando a cada lado de la igualdad el término $x^*$ y a vez, distribuyendo términos dentro de la fracción, se tiene que:

$$\varepsilon_{n+1} = \varepsilon_n - \frac{\varepsilon_n - \varepsilon_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n) \tag{7.10}$$

Ahora, distribuyendo términos para el segundo lado de la ecuación, se tiene que:

$$\varepsilon_{n+1} = \frac{\varepsilon_{n-1} f(x_n) - \varepsilon_n f(x_{n-1})}{f(x_n) - f(x_{n-1})} \tag{7.11}$$

Ahora, notar que la expresión del numerador y el denominador poseen los términos $f(x_n)$ y $f(x_{n-1})$, al reemplazar estos por su serie de Taylor, se tiene que:

$$\varepsilon_{n+1} = \frac{\varepsilon_{n-1}(\varepsilon_n f'(x^*) + \frac{\varepsilon_n^2}{2} f''(x^*) + \frac{\varepsilon_n^3}{6} f'''(\xi_1)) - \varepsilon_n (f'(x^*)\varepsilon_{n-1} + \frac{\varepsilon_{n-1}^2}{2} f(x^*) + \frac{\varepsilon_{n-1}^3}{3!} f'''(\xi_2))}{\left[\varepsilon_n f'(x^*) + \frac{\varepsilon_n^2}{2} f''(x^*) + \frac{\varepsilon_n^3}{6} f'''(\xi_1)\right] - \left[\varepsilon_{n-1} f'(x^*) + \frac{\varepsilon_{n-1}^2}{2} f''(x^*) + \frac{\varepsilon_{n-1}^3}{6} f'''(\xi_2))\right]} \tag{7.12}$$

$$\varepsilon_{n+1} = \frac{\frac{f''(x^*)}{2}(\varepsilon_{n-1}\varepsilon_n^2 - \varepsilon_n \varepsilon_{n-1}^2) + \frac{\varepsilon_{n-1}\varepsilon_n}{6}(\varepsilon_n^2 f'''(\xi_1) - \varepsilon_{n-1}^2 f'''(\xi_2))}{(\varepsilon_n - \varepsilon_{n-1})\left[f'(x^*) + \frac{f''(x^*)}{2}(\varepsilon_n + \varepsilon_{n-1})\right] + \frac{\varepsilon_n^3}{6} f'''(\xi_1) - \frac{\varepsilon_{n-1}^3}{6} f'''(\xi_2))} \tag{7.13}$$

Ahora bien, considerando un $\varepsilon_n$ y $\varepsilon_{n-1}$ lo suficientemente pequeños en caso de que este se encuentre convergiendo (ya que estos corresponden a la distancia entre puntos $x_n$ y $x_{n-1}$), se puede notar que es posible truncar los términos del error $f'''(\xi_1)$ y $f'''(\xi_2)$, quedando entonces:

$$\varepsilon_{n+1} = \frac{\frac{f''(x^*)}{2}\varepsilon_{n-1}\varepsilon_n(\varepsilon_n - \varepsilon_{n-1})}{(\varepsilon_n - \varepsilon_{n-1})\left[f'(x^*) + \frac{f''(x^*)}{2}(\varepsilon_n + \varepsilon_{n-1})\right]} \tag{7.14}$$

Finalmente, se puede notar que el término $(\varepsilon_n + \varepsilon_{n-1})\frac{f'''(x^*))}{2}$ es, en comparación a $f'(x^*)$, suficientemente pequeño como para poder ser truncado. Esto debido al hecho que como se encuentra el algoritmo convergiendo, se tiene entonces que los términos $\varepsilon_{n+1}$ y $\varepsilon_n$ tienden a un valor cercano a cero, y por tanto, como $f'(x^*)$ es una constante, el término del denominador acaba siendo solo considerable para $f'(x^*)$, esto en consecuencia, y cancelando factores semejantes, se tiene que:

$$\varepsilon_{n+1} = \frac{f''(x^*)}{2f'(x^*)}\varepsilon_{n-1}\varepsilon_n \tag{7.15}$$

Ahora, se puede notar que, debido a que $x^*$ es un valor fijo (puesto que corresponde al valor del cero de la función), esto implica que sus derivadas son constantes y, en consecuencia, el factor $\frac{f''(x^*)}{2f'(x^*)}$ también es constante, por tanto, queda demostrada la afirmación inicial (7.6).

### 7.2.2. Factor de proporcionalidad y comparación con otros métodos

A continuación, usando como suposición que para cierto $n > N$, se cumple la hipótesis de $\varepsilon_{n+1} = K\varepsilon_n^p$, con $K$ una constante, entonces se debe demostrar que se cumple también $p = (1 + \sqrt{5})/2$.

Para iniciar, considerar que debido a que el término de error o diferencia $\varepsilon_n = x^* - x_n$ proviene del ciclo de iteraciones convergentes y, por criterio de convergencia general, entonces, se debe cumplir la misma afirmación para la iteración anterior a esta, por tanto, se debe cumplir para un término $n = n - 1$, y por ello, debe existir:

$$\varepsilon_n = K\varepsilon_{n-1}^p \tag{7.16}$$

Ahora, de esta ecuación se puede obtener una expresión para $\varepsilon_{n-1}$, inyectando esta dentro de (7.15), se tiene que:

$$\varepsilon_{n-1} = K^{-1/p} \cdot \varepsilon_n^{1/p} \quad \Rightarrow \quad \varepsilon_{n+1} = \frac{f''(x^*)}{2f'(x^*)}\varepsilon_n \cdot K^{-1/p}\varepsilon_n^{1/p} \tag{7.17}$$

En consecuencia, por hipótesis, se tiene que:

$$K\varepsilon_n^p = \frac{f''(x^*)}{2f'(x^*)}K^{-1/p} \cdot \varepsilon_n^{1+1/p} \tag{7.18}$$

Por tanto, como los valores $\varepsilon_n$ deben ser iguales y se tiene que tanto $K$ como $\frac{f''(x^*)}{2f'(x^*)}$ son constantes, entonces los exponentes de los términos $\varepsilon_n$ deben ser equivalentes, por tanto:

$$p = 1 + 1/p \Rightarrow p = (1 \pm \sqrt{5})/2 \tag{7.19}$$

Finalmente, como se sabe que esta proporción debe ser siempre positiva, puesto que el error $\varepsilon_{n+1}$ debe disminuir en proporción al error $\varepsilon_n$ para que el método funcione y a su vez, no diverja, en consecuencia, $p = (1 + \sqrt{5})/2$.

Ahora, esto implica que el método de la secante posee una convergencia del tipo *Superlineal*, debido a que su tasa/ratio de convergencia se encuentra entre 1 y 2, en concreto, este siendo el número áureo ($\phi = (1 + \sqrt{5})/2$)) (Wikipedia (2025)).

| Método | Convergencia | Ratio de Convergencia | Utilidad |
|---|---|---|---|
| Bisección | Lineal | 1 | Permite calcular de forma básica, los ceros de una función a través de la división constante del intervalo a analizar |
| Secante | Superlineal | $(1+\sqrt{5})/2$ | Permite obtener ceros para casos donde la derivada sea difícil o imposible de calcular |
| Newton-Raphson | Cuadrática | 2 | Permite calcular los ceros de una función con menor cántidad de iteraciones haciendo uso del valor de la derivada en el punto |

Cuadro 7.2: Comparación entre diferentes métodos para encontrar ceros de forma analítica para una función $f(x)$, esto, junto a el orden de convergencia para cada método (U. Politecnica de Cataluña (2012))

Comparando este con otros métodos numéricos como el de la bisección o Newton-Raphson, se pueden notar diferencias en funcionalidad y eficiencia (U. Politecnica de Cataluña (2012)), como se describe en (7.2).

En base a esto, se puede notar que la tasa de convergencia para el método de Newton-Raphson es mayor a la secante, lo que naturalmente permite una mejor eficiencia a la hora de encontrar ceros de una función, esto, a diferencia del método de la Bisección, el cual necesita de mayor cantidad de iteraciones para realizar la misma tarea (Math for College (sf)).

Sin embargo, notar que para funciones tales que posean un doble cero, el método de Newton-Raphson posee problemas, puesto que el método podría no converger al hacer uso de su derivada sobre los puntos de interés (Math for College (sf)), además, al haber un doble cero, el error $\varepsilon_{n+1}$ posee un orden del tipo lineal.

Para este caso, el método de la secante sufre también de problemas, en concreto, hay que analizar como cambia el error $\varepsilon_n$ frente a esto:

Para ello, se puede obtener el valor del error $\varepsilon_{n+1}$ referenciado en (7.10), esto a través de la fórmula (7.11).

Al usar la expansión en serie de Taylor con los ceros en la función y primera derivada $f(x^*) = f'(x^*) = 0$, se tiene que:

$$\varepsilon_{n+1} = \frac{\varepsilon_{n-1}\left[\frac{1}{2}f''(x^*)\varepsilon_n^2 + \frac{1}{3!}f'''(\xi_1)\varepsilon_n^3\right] - \varepsilon_n\left[\frac{1}{2}f''(x^*)\varepsilon_{n-1}^2 + \frac{1}{3}f'''(\xi_2)\varepsilon_{n-1}^3\right]}{\frac{1}{2}f''(x^*)\varepsilon_n^2 + \frac{1}{3!}f'''(\xi_1)\varepsilon_n^3 - \frac{1}{2}f''(x^*)\varepsilon_{n-1}^2 - \frac{1}{3}f'''(\xi_2)\varepsilon_{n-1}^3} \tag{7.20}$$

Reordenando los elementos dentro del numerador y denominador, se tiene que:

$$\varepsilon_{n+1} = \frac{\frac{1}{2}f''(x^*)\varepsilon_n\varepsilon_{n-1}(\varepsilon_n - \varepsilon_{n-1}) + \frac{1}{3!}f'''(\xi_1)\varepsilon_n^3 - \frac{1}{3}f'''(\xi_2)\varepsilon_{n-1}^3}{\frac{1}{2}f''(x^*)(\varepsilon_n + \varepsilon_{n-1})(\varepsilon_n - \varepsilon_{n-1})} \tag{7.21}$$

Considerando a los términos $f(\xi_1)$ y $f(\xi_2)$ como términos de mínimo tamaño debido al grado del error $\varepsilon$, se pueden truncar, por lo que reordenando la ecuación quedaría como:

$$\varepsilon_{n+1} = \frac{\frac{1}{2}f(x^*)\varepsilon_{n-1}\varepsilon_n(\varepsilon_n - \varepsilon_{n-1})}{\frac{1}{2}f(x^*)(\varepsilon_n - \varepsilon_{n-1})(\varepsilon_n + \varepsilon_{n-1})} \tag{7.22}$$

Ahora, factorizando términos semejantes, tenemos que:

$$\varepsilon_{n+1} = \frac{\varepsilon_n \varepsilon_{n-1}}{\varepsilon_n + \varepsilon_{n-1}} \tag{7.23}$$

Ahora, factorizando el término $\varepsilon_{n-1}$, se tiene que:

$$\varepsilon_{n+1} = \frac{\varepsilon_n}{1 + \frac{\varepsilon_n}{\varepsilon_{n-1}}} \tag{7.24}$$

Ahora considerando que el método es convergente, debe entonces ocurrir que $\varepsilon_n < \varepsilon_{n-1}$ (error disminuye conforme se itera el algoritmo) y en consecuencia, $\frac{\varepsilon_n}{\varepsilon_{n-1}} < 1$ (Solís A. Cordero A. (2012)), finalmente, considerar que este término debe ser superior a 0 puesto que de lo contrario, el método no convergería (el error enésimo sería superior al error anterior al enésimo).

Por tanto, se tiene que $0 < \frac{\varepsilon_n}{\varepsilon_{n-1}} < 1$ y por tanto está acotada. Denominandole como K, se tiene entonces que:

$$\varepsilon_{n+1} = \frac{1}{1+K}\varepsilon_n \tag{7.25}$$

Por lo que se puede concluir, que al haber una función de doble cero, o de multiplicidad 2, el método de la secante trabajaría con un orden de convergencia del tipo lineal.

## Conclusiones

Para finalizar, notar que se hizo un análisis del método numérico de la secante para funciones, en concreto, se hizo de la función $f(x) = x^3 - 5x$, esto con el objetivo de obtener sus ceros. Esto se logró de manera efectiva, notando una mínima cantidad de iteraciones para cada punto.

A su vez, este capítulo dió la oportunidad de aprender y entender sobre las diferencias conceptuales y funcionales que existen entre los métodos abordados en este curso, tales como el método de Newton-Raphson o el método de la bisección, a la vez que se pudo comparar entre las utilidades que existen y su relación con la tasa/ratio de convergencia para cada uno ($p$).

## Agradecimientos

Señalar el agradecimiento al profesor Roberto Navarro por sus comentarios para el código del método de la secante y la ayuda en la capacidad de mejorar este para reducir el número de iteraciones internas.

También, notar la autoría mayoritaria del capítulo por Ignacio Falcón, complementado por Joaquín Parra en correcciones en el código y eficiencia del método y a Álvaro Osses por sus comentarios para redacción y ortografía dentro del contenido de este capítulo, principalmente en la demostración del error $\varepsilon_{n+1}$.

Finalmente, mencionar que para este capítulo, se hizo uso de la inteligencia artificial de Gémini para poder encontrar errores de LaTeX dentro del código de ecuaciones.

# Capítulo 8

# Método implícito de Euler

Joaquín Parra, Ignacio Falcón, Alvaro Osses

**Fecha de la actividad:** 26 de Noviembre de 2025

En el presente capítulo implementaremos el método implícito de Euler para resolver problemas del tipo $dx/dt = f(x)$ de condición incial $x(0) = x_0$, utilizando además el método de Newton-Raphson. Posteriormente, aplicaremos esto a la resolución de una ecuación diferencial simple.

## 8.1. Implementación del método

El método de Euler implícito busca resolver el problema

$$\frac{dx}{dt} = f(x), \quad x(0) = x_0 \tag{8.1}$$

utilizando un esquema adelantado de la forma

$$x_{n+1} = x_n + hf(x_{n+1}) \tag{8.2}$$

De este modo, podemos interpretar el método como una búsqueda de ceros de la función

$$g(x) = x_n + hf(x) - x \tag{8.3}$$

considerando que $g(x_{n+1}) = 0$.

Para implementar este esquema, utilizaremos el método de Newton-Raphson para búsqueda de ceros, el cual aproxima la raíz de una función $g(x)$ a través del siguiente esquema:

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)} \tag{8.4}$$

Para poder utilizar este método con la función $g(x)$ definida en (8.3), necesitamos conocer su derivada, para esto, la derivamos analíticamente:

$$\frac{dg}{dx}(x) = h\frac{df}{dx}(x) - 1 \tag{8.5}$$

Ahora, para implementar este método, se escribió el siguiente código:

```python
def Newton(f, df, seed, tol = 1e-5, **kwargs):
    stop = False

    x = seed

    while not stop:
        aux = f(x, **kwargs)
        err = aux/df(x) # Se aplica le esquema de Newton-Raphson
        x = x - err

        stop = abs(err) < tol and abs(aux) < tol # Se checea si el valor encontrado es
                                                 # aceptable, en cuyo caso se detiene el ciclo

    return x

def euler_exp(f, x0, tmax, dt, t0 = 0, **kwargs):
    x0 = np.asarray(x0) #considera casos en que x0 es un escalar, no un arreglo
                        #(convierte escalar a array)
    t = np.arange(t0, tmax, dt) #arreglo de tiempos
    x = np.zeros((*t.shape, *x0.shape)) #desempaquetado de iterables
    x[0] = x0

    for i in range(t.size - 1):
        x[i+1] = x[i] + dt * f(t[i], x[i], **kwargs)
    return t, x

def euler_imp(f, df, x0, tmax, dt, t0 = 0, **kwargs):
    x0 = np.asarray(x0) # Considera casos en que x0 es un escalar, no un arreglo
    t = np.arange(t0, tmax, dt) # Crea arreglo de tiempos
    x = np.zeros((*t.shape, *x0.shape)) # Desempaqueta iterables
    x[0] = x0
    for i in range(t.size-1):
        g = lambda y: x[i] + dt*f(t[i], y, **kwargs) - y
        dg = lambda y: dt*df(t[i], y, **kwargs) - 1
        x[i+1] = Newton(g, dg, seed = x[i]) # Utiliza Newton-Raphson para encontrar el
                                            # siguiente paso
    return t, x
```

donde definimos el método de Newton-Raphson, y los métodos de Euler implícito y explícito, tenemos además que `f` y `df` representan la función $f$ y su derivada, `x0` representa la condición incial del problema y `seed` representa el valor semilla que necesita el método de Newton-Raphson.

## 8.2. Implementación a un problema real

Habiendo planteado el método, podemos aplicarlo al problema de decaimiento exponencial

$$x'(t) = -5x, \qquad x(0) = 1, \tag{8.6}$$

con $0 \leq x \leq 2$ y $h = 0.1$. Para que el problema sea compatible con el método, debemos encontrar la derivada de $x'(t)$, para ello derivamos analíticamente la expresión (8.6). obteniendo la expresión:

$$f'(x) = x''(t) = -5 \tag{8.7}$$

Así, aplicamos el método a través del siguiente código

```python
fun = lambda t, x: -5*x
```

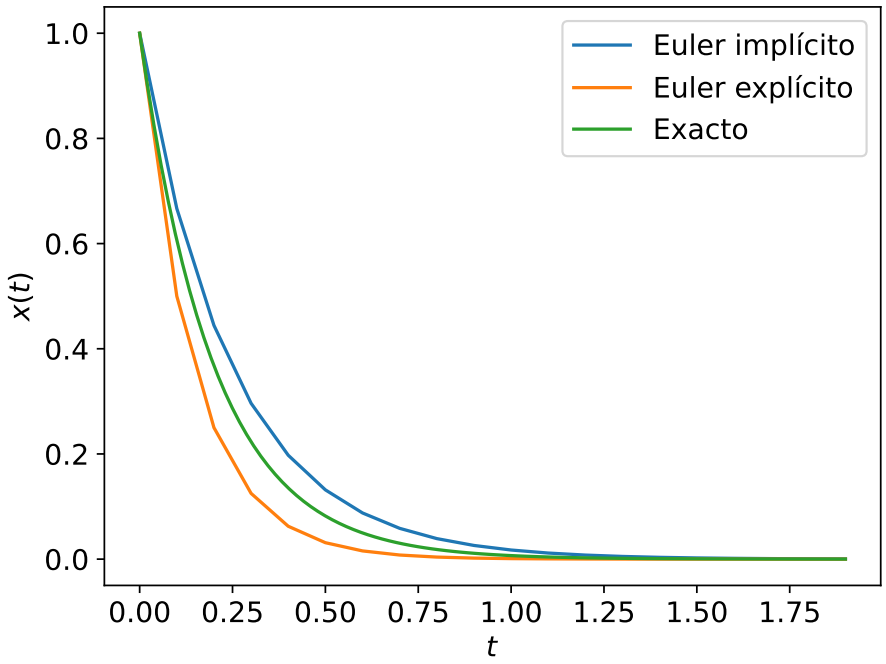


Figura 8.1: Comparación de método de Euler implícito y explícito con la solución exacta.

```
dfun = lambda t, x: -5

t, x = euler_imp(fun, dfun, x0 = 1, tmax = 2, dt=0.1)
teuler, xeuler = euler(fun, x0 = 1, tmax = 2, dt = 0.1)
```

y generamos el gráfico de la figura 8.1 donde comparamos el resultado del método implícito y explícito con la solución exacta. De aquí se puede observar que para valores pequeños, el método implícito se acerca más a la solución exacta que el método explícito.

En la figura 8.2 se aprecia de mejor manera como el método implícito se acerca más a la solución exacta que el explícito para valores de $t < 0.5$. Pasado ese punto el error del método implícito es ligeramente mayor al del explícito. Esto último se debe a que el método implícito de euler es numéricamente más estable que el método explícito para valores de $dt$ más grandes (Butcher (2003)). Ahora bien, una desventaja del método es que este realizará una busqueda de ceros a cada paso del proceso, lo cual significa un mayor consumo de recursos que el método explícito. Esto puede llegar a causar problemas para funciones donde la busqueda de ceros no converja rápidamente.

## Conclusiones

Para finalizar, podemos decir que el método implícito de Euler es un esquema de resolución de ecuaciones diferenciales que resulta sumamente útil, pues este reduce una ecuación diferencial a un problema de busqueda de ceros, un problema mucho más simple, y este mostrará resultados numéricamente estables y cercanos a la solución exacta con valores de $dt$ mayores que los requeridos por su contraparte explícita. Ahora bien, es evidente que la principal desventaja de este método, o al menos de la implementación utilizada en este capítulo, es que se debe conocer la derivada de la función asociada al problema. Esto puede resultar trabajoso si se está lidiando con una función muy rebuscada, pero

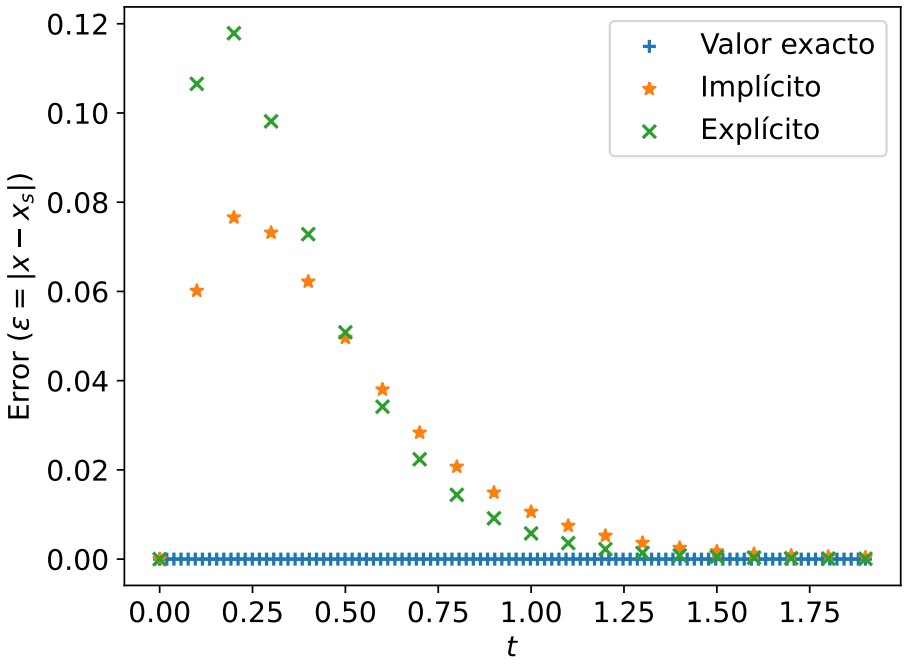


Figura 8.2: Comparación de errores de cada método.

puede solucionarse implementando métodos numéricos para derivar esta función, o bien utilizando otro método de busqueda de ceros que no requiera de la derivada.

## Agradecimientos

Este capítulo fue escrito principalmente por Alvaro Osses con contribuciones por Joaquín Parra e Ignacio Falcón. Se extienden agradecimientos al profesor Roberto Navarro por responder dudas respecto al método implementado.

No se utilizaron herramientas de inteligencia artificial para la creación de este capítulo.

# Capítulo 9

# Implementación de métodos numéricos en análisis de estabilidad de un péndulo invertido conectado a un resorte

Joaquín Parra, Ignacio Falcón, Alvaro Osses

**Fecha de la actividad:** 26 de Octubre de 2025

En este capítulo haremos uso del método de la bisección (implementado via `python`) para un problema particular en el contexto de la mecánica: un péndulo invertido de largo $\ell$ con una masa $m$ en su extremo con un resorte conectado a la barra del péndulo a una altura $d < \ell$ como se muestra en la figura 9.1. Determinaremos la energía potencial del sistema y observaremos como la configuración de las constantes afecta a la estabilidad del mismo.

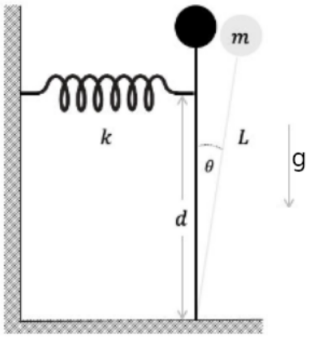


Figura 9.1: Problema a analizar.

## 9.1. Determinación de la energía potencial del sistema

La energía potencial $U$ de una partícula o de un sistema de partículas surge al describir cómo ciertas configuraciones del sistema están asociadas a interacciones que condicionan su dinámica. En muchos sistemas físicos, las partículas no se mueven libremente, sino que están sometidas a fuerzas conservativas y a ligaduras que restringen su movimiento. Cuando las fuerzas son conservativas, pueden derivarse de un potencial $U$, el cual cuantifica su influencia sobre la evolución del sistema.
En la figura 9.1, podemos identificar tres fuerzas actuando sobre la masa $m$: identificamos la fuerza peso $\vec{W}$ ejercida sobre la masa, una fuerza de ligadura[1] $\vec{f}$ o tensión de la barra que la conecta a tierra y la fuerza elástica $\vec{F}_e$ del resorte al que está conectada. Recordemos que la tensión no es una fuerza conservativa, por lo que no aporta a la energía potencial del sistema. Entonces, las únicas fuerzas que aportan términos a la energía potencial de la masa son la peso y la elástica. Considerando un sistema cartesiano usual con origen en la base del cable, las energías potenciales quedan definidas como:

$$U_g(y) = mgy, \qquad U_e(x) = \frac{1}{2}kx^2. \tag{9.1}$$

Donde se tiene una altura arbitrariamente pequeña para que que $g$ se considere constante y obtener así la expresión de arriba, $x$ siendo la deformación del resorte a altura $d$ e $y$ siendo la altura de la masa. Puesto que el movimiento está restringido por una barra (de masa despreciable y no deformable), nuestra masa se moverá sobre un arco de circunferencia, por lo que puede ser apropiado realizar un cambio de coordenadas cartesianas a polares. Consideremos:

$$x = d\sin(\theta), \qquad y = \ell\cos(\theta), \tag{9.2}$$

con $\ell$ el largo de la barra y $\theta$ el ángulo que forma el cable con la normal (en radianes). Implementando el cambio de coordenadas en (9.1), obtenemos:

$$U_g(\theta) = mg\ell\cos(\theta), \qquad U_e(\theta) = \frac{1}{2}kd^2\sin^2(\theta). \tag{9.3}$$

Notemos como con el cambio de coordenadas redujimos los grados de libertad del sistema de dos a uno, y con el uso del teorema $\Pi$ podemos normalizar el sistema. En particular, vemos que dividiendo por el $\Pi$ grupo $\Pi_1 = mg\ell$, las ecuaciones (9.3) quedan normalizadas. Tenemos entonces que la energía potencial normalizada del sistema, definida como la suma de la energías potenciales gravitatoria y elástica (normalizadas) es:

$$u(\theta) = \cos(\theta) + \beta\sin^2(\theta), \tag{9.4}$$

con $\beta = kd^2/2mg\ell$.
Queda ahora identificar los puntos de equilibrio del sistema, que se deducen de la ecuación:

$$\vec{\nabla}u = \vec{0}, \tag{9.5}$$

o en nuestro caso, de forma más simplificada:

$$\frac{du}{d\theta} = 0. \tag{9.6}$$

## 9.2. Derivada del potencial y búsqueda de ceros

En la figura 9.2 podemos ver soluciones numéricas para la derivada de (9.4), haciendo uso tanto de derivadas centradas, adelantadas y retrasadas, variando el parámetro $\beta$. Podemos ver que para valores $\beta > 0.5$ aparece un segundo cero, un cero no trivial $\theta_{nt}$.

Identificaremos estos ceros mediante el método de la bisección, tal como se muestra a continuación.

[1] Comentarios sobre mecánica clásica cortesía de estudiante de ingeniería civil aeroespacial.

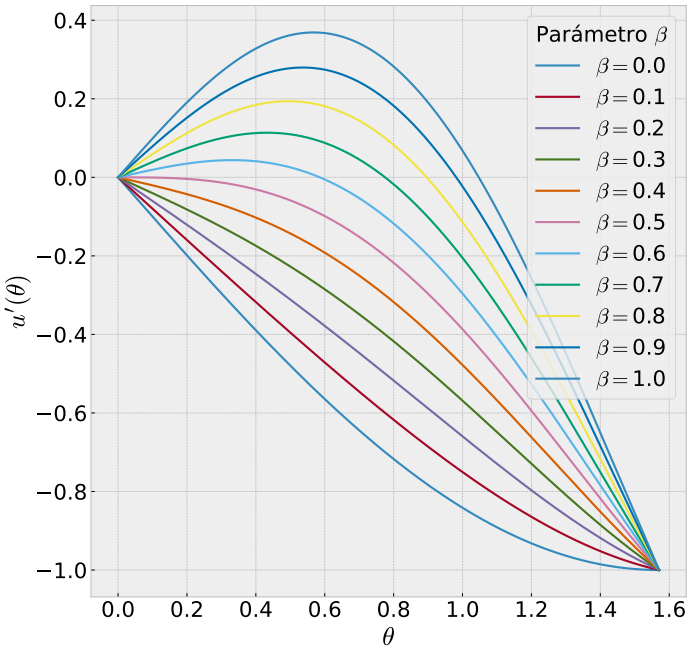


Figura 9.2: $u'(\theta)$ a distintos $\beta$ para $0 \leq \theta \leq \pi/2$.

| $\beta$ | $\theta_{nt}$ |
|---|---|
| 0.6 | 0.58568 |
| 0.7 | 0.77519 |
| 0.8 | 0.89566 |
| 0.9 | 0.98176 |
| 1.0 | 1.04719 |

Cuadro 9.1: Valores de $\theta_{nt}$ para cada $\beta$, truncados al quinto decimal.

```python
def bis(f, a, b, tol=1e-10, iter=100, **kwargs):
    iter_count = 0

    while iter_count < iter:
        c = 0.5 * (a + b)

        if abs(f(c, **kwargs)) < tol or abs(b - a) < tol:
            return c

        if f(a, **kwargs) * f(c, **kwargs) < 0:
            b = c
        else:
            a = c
        iter_count += 1

def u(theta, beta):
      return np.cos(theta) + beta * (np.sin(theta))**2

def du_cen(f, theta, h=1e-5, **kwargs):
    return (f(theta + h, **kwargs) - f(theta - h, **kwargs)) / (2*h)

def u_prime(theta, beta):
    return du_cen(u, theta, beta=beta)

def cero(beta):      #retorna el cero no trivial para un cierto beta
   if beta <= 0.5:
      return "No hay cero no trivial"
   else:
      return bis(u_prime, 0.5, 1.1, beta=beta)
```

Este código se utilizará ahora para encontrar los ceros no triviales para distintos betas, para lo cual necesitaremos trabajar con función del potencial $u'(\theta)$, la cual se trabajará mediante un esquema de derivada centrada al igual que en la figura 9.2. Implementando el código, obtenemos los ceros del cuadro 9.1. En la figura 9.3 podemos ver la curva descrita por los ceros no triviales en función de $\beta$ como variable continua (de pasos muy cortos) , y superpuestos están los ceros ya conocidos.

## 9.3. Tipos de equilibrio

Finalmente, queda estudiar el tipo de equilibrio que se tiene en el cero trivial y en los no triviales. Estos se pueden determinar fácilmente con el criterio de la segunda derivada, por el cual tenemos que si $\theta$ es un punto de equilibrio, si $u''(\theta) > 0$, entonces estamos frente a un punto de equilibrio estable, y si $u''(\theta) < 0$, entonces estamos frente a un punto de equilibrio inestable. Mediante el uso de derivadas de segundo orden adelantadas y centradas, podemos estimar estos valores y determinar el tipo de equilibrio. En la figura 9.4 podemos ver que para todos los ceros no triviales utilizados, la segunda

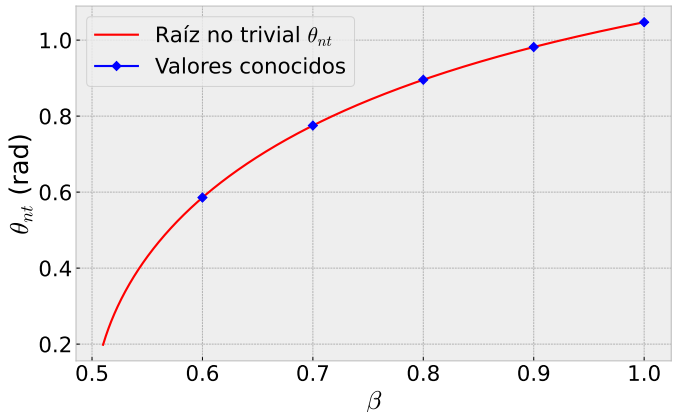


Figura 9.3: $\theta_{nt}(\beta)$ para $0.5 < \beta \leq 1$.

derivada toma valores negativos, por lo cual estos representan puntos de equilibrio inestables, lo cual era de esperarse dada la configuración del sistema. Sin embargo, vemos que para valores de $\beta$ menores a 0.5, $u''(0)$ adopta valores negativos, por lo que representa también equilibrio inestable en estos casos. Dado que $\beta = kd^2/(2mgl)$, podemos interpretar estos valores como casos en los cuales la fuerza peso le gana a la fuerza elástica, por lo que perturbaciones que aumenten ligeramente el valor de $\theta$ hacen que el sistema se aleje del punto de equilibrio, pues la fuerza elástica no es capaz de contrarrestar al peso. Finalmente, para $\beta$ mayores a 0.5, vemos que se adoptan valores positivos, por lo que en estas configuraciones donde la fuerza elástica vence a la peso, al perturbar el sistema y aumentar ligeramente $\theta$, este busca volver a la posición $\theta = 0$, por lo que se clasifica como punto de equilibrio estable.

## Conclusiones

En este breve capítulo hicimos uso de el método de la bisección para estudiar un problema poco convencional en mecánica, encontrando puntos de equilibrio y clasificándolos con el uso de métodos numéricos, a la vez que aprendimos criterios al momento de realizar análisis de equilibrio en sistemas mecánicos. Pudimos extraer información sin tener que calcular (necesariamente) derivadas, y comprobamos resultados con modelos analíticos. Podemos concluir que lo anterior se logró de manera efectiva gracias a la comparaciones realizadas, las cuales dan cuenta de la precisión de los cálculos realizados.

## Agradecimientos

Este capítulo fue principalmente escrito por Joaquín Parra, con revisiones tanto de código como de contenido de parte de sus compañeros de trabajo Alvaro Osses e Ignacio Falcón, quienes aportaron también escribir el código de la bisección y con el análisis físico de la primera sección. Se agradece al profesor responsable del curso, Dr. Roberto Navarro por entregar las nociones necesarias para realizar este trabajo y por sus comentarios en clases, los cuales siempre son de utilidad al momento de implementar código. También se agradece a Jesús Coronel, estudiante de segundo año de ingeniería civil aeroespacial por breves comentarios sobre el concepto de ligaduras, los cuales se ocuparon al inicio del capítulo. Finalmente, se menciona que se utilizó ayuda de ChatGPT y Gemini a la hora de realizar los gráficos del capítulo.

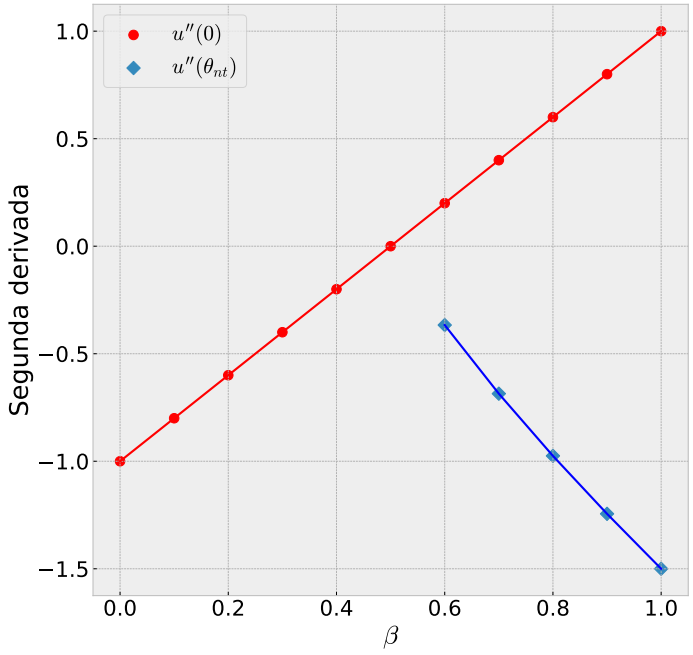


Figura 9.4: Segunda derivada del potencial versus $\beta$ evaluada en los puntos de equilibrio.

# Capítulo 10

# Polinomios de Legendre e Interpolación para un intervalo de datos

Ignacio Falcón, Joaquín Parra, Alvaro Osses

**Fecha de la actividad:** 10 de Diciembre del 2025

Para el presente capítulo, se estudiarán los polinomios de Legendre basados en la fórmula de Laplace, con la cual se obtendrán valores para un rango $|x| \leq 1$ a través del uso de la Regla de Simpson 1/3. Luego de esto, se hará uso de la **Interpolación de Lagrange** para llenar una cantidad determinada de puntos en base a nodos $x_i$.

## 10.1. Fórmulas para los Polinomios de Legendre

Dentro del análisis real y la física, la necesidad de un conjunto de "bases" para un espacio vectorial, como es el de $\mathbb{R}[x]$ ha sido de vital importancia, especialmente en la resolución de problemas con simetrías esféricas en ecuaciones diferenciales.

Para esto, se puede hacer uso de los *Polinomios de Legendre*, los cuales responden a ser soluciones de la *ecuación diferencial de Legendre* (Wikimedia Foundation, Inc. (2025b)), la cual posee la fórmula explícita de:

$$\frac{d}{dx}\left[(1-x^2)\frac{d}{dx}P_n(x)\right] + n(n+1)P_n(x) = 0 \tag{10.1}$$

Estos a su vez, poseen la particularidad ser ortogonales para valores $|x| \leq 1$ lo que significa que responden al producto punto o escalar definido (Facultad de Ciencias Astronómicas y Geofísicas, UNLP (2025)) como:

$$\int_{-1}^{1} P_m(x)P_n(x)dx = \frac{2}{2n+1}\delta_{mn} \tag{10.2}$$

Donde $\delta_{mn}$ denota a la *Delta de Kronecker*, la cual devuelve 1 si $m = n$ (Los polinomios $P_n(x)$ y $P_m(x)$ son iguales) y 0 si $m \neq n$ (Los poliniomios son diferentes).

En base a esto, se puede formar una definición explícita para los Polinomios de Legendre haciendo uso de la **Fórmula de Rodrigues** :

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}[(x^2-1)] \tag{10.3}$$

O haciendo uso de la **Fórmula de Laplace** como:

$$P_n(x) = \frac{1}{2\pi} Re \int_0^{2\pi} d\phi (x + i\sqrt{1-x^2}\cos\phi)^n \tag{10.4}$$

Para efectos de este capítulo, se hará uso de la Fórmula de Laplace.

## 10.2. Análisis numérico de los Polinomios de Legendre para un intervalo definido

Para esta sección, se estudiará el uso de cálculos numéricos de los polinomios de Legendre haciendo uso de la ecuación (10.4) dentro del intervalo $-1 \leq x \leq 1$ con el fin de representarlos gráficamente a través del módulo `matplotlib` de `Python`.

Para esto, se hará uso de la *Regla de Simpson 1/3 Compuesta* , la cual está definida como:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 4\sum_{i=1,i\text{ impar}}^{n-1} f(x_i) + 2\sum_{i=2,i\text{ par}}^{n-2} f(x_i) + f(x_n)\right] \tag{10.5}$$

donde $h$ es la distancia entre los valores $x_i$, y $n$ siendo la cantidad de valores $x_i$ con los que trabajar.

Finalmente, para efectos de la gráfica, se harán uso de $N = 13$ valores para realizar la integral, con la que finalmente, se desean obtener los primeros seis polinomios de Legendre $(P_1(x), P_2(x), ..., P_6(x))$ en el intervalo de $-1 \leq x \leq 1$.

El código presentado a continuación explicíta la idea concebida anteriormente:

```
def integrado(x, n, phi):
    return (x + np.sqrt( 1 - x**2)*np.cos(phi)*1j )**n

def Metodo_Simpson_1_3(f,n,x,c=13, **kwargs):
    sum_impar, sum_par = 0,0
    phi = np.linspace(0,2*np.pi, c)
    h = phi[1] - phi[0]

    for i in range(2,c-2,2):
        sum_par += f(x, n,phi[i], **kwargs)

    for i in range(1,c-1,2):
        sum_impar += f(x,n,phi[i], **kwargs)

    return h/3 * ( f(x,n,phi[0], **kwargs) + f(x,n,phi[-1], **kwargs) + 4 * sum_impar  +  2 * sum_par )

def Legendre_n(x,n,g,f, **kwargs):
    return 1/(2*np.pi) * g(f,n,x, **kwargs).real

def Polinomios_Legendre(n, w):
    t = np.linspace(-1,1,w)
    for i in range(1,n+1):
        y = (Legendre_n(x =np.linspace(-1,1,w),n=i,g=Metodo_Simpson_1_3, f=integrado))
        plt.plot(t,y,label=f'N ={i}')
    plt.legend()
    plt.show()
```

Finalmente, al ejecutar la función `Polinomios_Legendre(6, 100)` se obtiene el presente gráfico (10.1).

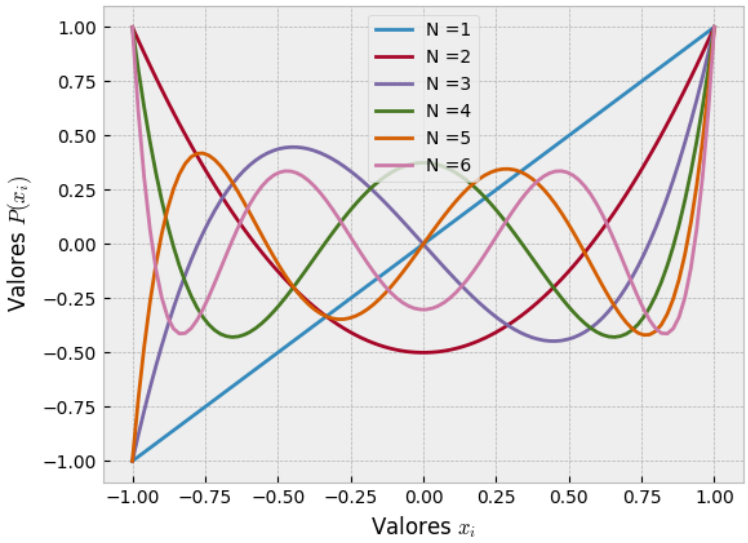


Figura 10.1: Gráfico de valores $(x_i, P_n(x_i))$ para grados de $n = 1, 2, ..., 6$

## 10.3. Interpolación de Lagrange y coeficientes del polinomio

### 10.3.1. Interpolación de Lagrange para Polinomios

A continuación, se hará uso de la interpolación de Lagrange en base a una cantidad determinada de nodos para generar una curva que asemeje a la del polinomio de Legendre a tratar, con el objetivo de encontrar los coeficientes de este a través de un ajuste de datos (Encyclopedia of Mathematics (2025)).

Para comenzar, se debe definir la interpolación de Lagrange, la cual es un método numérico que tiene como objetivo interpolar un intervalo de datos entre diferentes puntos $(x_i, y_i)$ a través de un *Polinomio de Lagrange*, que tiene la forma:

$$P(o) = \sum_{i=0}^{n} y_i \cdot L_i(o) \tag{10.6}$$

Donde $L_i(o)$ se conoce como el *Polinomio base de Lagrange* (Wikimedia Foundation, Inc. (2025a)), el cual se define como:

$$L_i(o) = \prod_{j=0, j\neq i}^{n} \frac{o - x_j}{x_i - x_j} \tag{10.7}$$

Ahora, al hacer uso de esto, se considerarán como nodos $(x_i, y_i)$ puntos de los Polinomios de Legendre, para comparar los coeficientes generados por la interpolación y los coeficientes reales para cada grado.

Por lo cual, se tiene que para un Polinomio de Legendre con grado $n$, el código que grafica esta función interpolada sería:

```
def Interpol_Lagrange(int, x, y):
    k = len(x) - 1
    sum = 0
    for i in range(k + 1):
```

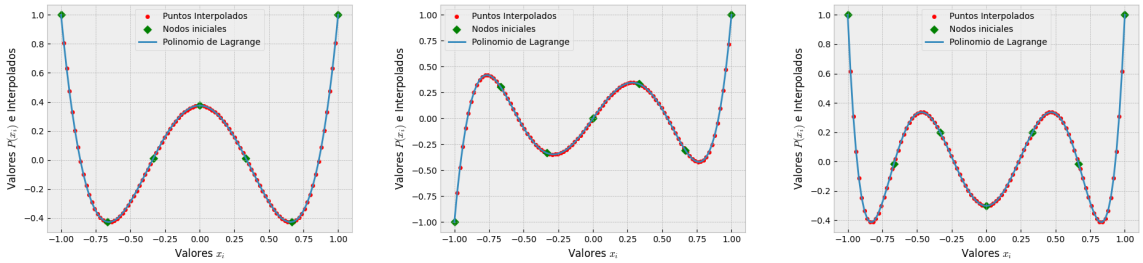


Figura 10.2: Gráficos de comparación entre Polinomios de Legendre para grados $n = 4, 5, 6$. Para cada gráfico, en rojo, los puntos generados por la interpolación de Lagrange, en verde los nodos generados por la Fórmula de Laplace para los Polinomios de Legendre, y en azul la curva del Polinomio descrito a través de la fórmula de Laplace.

```
        prod = y[i]
        for j in range(k + 1):
            if i != j:
                prod = prod*(int - x[j])/(x[i] - x[j])
        sum = sum + prod
    return sum

def Interpolar_Legendre(n,m, w):
    data= np.linspace(-1,1,w)
    data_m = np.linspace(-1,1,m)
    s = Legendre_n(x =data_m,n=n,g=Metodo_Simpson_1_3, f=integrado)
    l = (Legendre_n(x =data,n=n,g=Metodo_Simpson_1_3, f=integrado))
```

Este código, junto a las funciones mencionadas anteriormente, permite graficar los polinomios interpolados tal como se puede observar en (10.2)

### 10.3.2. Comparación de Coeficientes del Polinomio

A partir de esto, se pueden utilizar los puntos $(x_k, y_k)$ generados por la interpolación de Lagrange para obtener los coeficientes aproximados del Polinomio de Legendre sobre el cual se generaron los nodos.

Esto se realizará a través de una interpolación con el método de Vandermonde, donde se obtendrán los valores de los coeficientes a través del cálculo de la inversa de la matriz de Vandermonde para los nodos $(x_i, y_i)$ (Academia Lab, 2025) .

En base a esto, se puede definir el código a utilizar como:

```
def Coeficientes_Interpolados(n):
    w = n+1
    x = np.linspace(-1,1,w)
    y = Legendre_n(x =x,n=n,g=Metodo_Simpson_1_3, f=integrado)
    matriz_vander = np.vander(x)
    Coeficientes = np.linalg.solve(matriz_vander, y)
    legendre_coef = np.array(scp.special.legendre(n))
```

Luego de esto, y usando los coeficientes obtenidos, se pueden comparar haciendo uso del cálculo del error absoluto entre cada coeficiente real y obtenido, donde el error se define como:

$$\varepsilon_{\text{abs}} = |A_{\text{calculado}} - A_{\text{polinomio}}| \tag{10.8}$$

Obteniendo de esta forma la información de (10.1):

| Polinomio | Coeficiente Exacto | Coeficiente Interpolado | Error Absoluto |
|---|---|---|---|
| $P_1(x)$ | | | |
| Coef. $x^1$ | 1 | 1 | 0 |
| Coef. $x^0$ | 0 | 0 | 0 |
| $P_2(x)$ | | | |
| Coef. $x^2$ | 1.5 | 1.5 | 0 |
| Coef. $x^1$ | 0 | 0 | 0 |
| Coef. $x^0$ | -0.5 | -0.5 | 0 |
| $P_3(x)$ | | | |
| Coef. $x^3$ | 2.5 | 2.5 | 4.44089210e-16 |
| Coef. $x^2$ | 0 | -2.49800181e-16 | 2.49800181e-16 |
| Coef. $x^1$ | -1.5 | -1.5 | 2.22044605e-16 |
| Coef. $x^0$ | 0 | 2.22044605e-16 | 2.22044605e-16 |
| $P_4(x)$ | | | |
| Coef. $x^4$ | 4.375 | 4.375 | 0 |
| Coef. $x^3$ | 4.85722573e-16 | 0 | 4.85722573e-16 |
| Coef. $x^2$ | -3.75 | -3.75 | 8.88178420e-16 |
| Coef. $x^1$ | 2.42861287e-16 | 0 | 2.42861287e-16 |
| Coef. $x^0$ | 3.75e-1 | 3.75e-1 | 5.55111512e-17 |
| $P_5(x)$ | | | |
| Coef. $x^5$ | 7.875 | 7.875 | 1.77635684e-15 |
| Coef. $x^4$ | 0 | -1.44560290e-15 | 1.44560290e-15 |
| Coef. $x^3$ | -8.75 | -8.75 | 1.77635684e-15 |
| Coef. $x^2$ | -4.37150316e-16 | 1.61907524e-15 | 2.05622556e-15 |
| Coef. $x^1$ | 1.875 | 1.875 | 2.22044605e-16 |
| Coef. $x^0$ | 0 | 2.22044605e-16 | 2.22044605e-16 |
| $P_6(x)$ | | | |
| Coef. $x^6$ | 1.44375000e1 | 1.44270833e1 | 1.04166667e-2 |
| Coef. $x^5$ | 0 | 6.18255447e-15 | 6.18255447e-15 |
| Coef. $x^4$ | -1.96875000e1 | -1.96562500e1 | 3.12500000e-2 |
| Coef. $x^3$ | 1.60288449e-15 | -9.59232693e-15 | 1.11952114e-14 |
| Coef. $x^2$ | 6.5625 | 6.53125 | 3.125e-2 |
| Coef. $x^1$ | 0 | 3.40977246e-15 | 3.40977246e-15 |
| Coef. $x^0$ | -3.125e-1 | -3.02083333e-1 | 1.04166667e-2 |

Cuadro 10.1: Comparación de Coeficientes Exactos vs. Interpolados para $P_n(x)$ con su respectivo error absoluto. Notar que el **Coeficiente Exacto**, proviene del término obtenido por `numpy.special.legendre`, mientras que **Coeficiente Interpolado**, proviene del código utilizado al realizar la interpolación. Finalmente, el error absoluto fue calculado haciendo uso del código anteriormente mencionado.

Dentro de esta, se puede notar el hecho que una variedad de coeficientes poseen términos de orden $\approx -14$, lo que indica valores extremadamente pequeños a diferencia de los que se encuentran en otros grados. Esto se debe a errores de redondeo dentro del sistema de la computadora, la cual se debe al hecho de que la precisión de la máquina no es capaz de computar con números tan pequeños.

Finalmente, es importante destacar el hecho de que para trabajar con esta construcción de los Polinomios de Legendre de orden $n$, es necesario contar con una cantidad $n+1$ de nodos $x_i$ con los que poder actuar.

Esto es debido al hecho de que al realizar la matriz de Vandermonde o para utilizar los polinomios interpoladores de Lagrange, con el objetivo de encontrar los coeficientes de un polinomio de grado $n$, estos generan un conjunto de ecuaciones con $n+1$ variables, donde $(x_i, y_i)$ son variables conocidas (nodos). Por tanto, para un polinomio, la matriz que reúne las ecuaciones estaría condicionada como:

$$\begin{pmatrix} x_1^n & x_1^{n-1} & ...x_1 & 1 \\ x_2^n & x_2^{n-1} & ...x_2 & 1 \\ x_3^n & x_3^{n-1} & ...x_3 & 1 \\ & ... & & \\ x_m^n & x_m^{n-1} & ...x_m & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ ... \\ c_m \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ ... \\ y_m \end{pmatrix} \tag{10.9}$$

Donde se puede notar que la única forma que esta matriz tenga inversa y por ende solución (suponiendo nodos diferentes entre si) es que sea cuadrada, y por ende, que $m = n+1$.

## Conclusión

Para concluir, hay que hacer mención del conocimiento adquirido sobre los Polinomios de Legendre, su definición y cómo implementarla en el ámbito numérico, esto a través de una de sus definiciones haciendo uso de la Fórmula de Laplace, la cual tuvo por desafió diseñar la integral apropiada, como fue en este caso haciendo uso del Método de Simpson 1/3.

Por otro lado, se pudo constatar y comprender de forma más práctica la utilidad que posee la Interpolación y como esta permite obtener coeficientes de un polinomio concreto en base a esta, permitiendo obtener datos del desarrollo de funciones en cuestión en base a datos o puntos finitos, esto, a través de métodos como el uso de la matriz de Vandermonde, método con el cual obtener coeficientes de un polinomio interpolado.

## Agradecimientos

Hacer los agradecimientos correspondientes a Amaro Díaz y Fernanda Mella por sus comentarios y conocimientos para poder aplicar el método de Interpolación de Lagrange.

A su vez, notar la mayoritaria contribución del capitulo por Ignacio Falcón, además, notar el trabajo de Álvaro Osses por el código dentro de la interpolación de Lagrange, a Joaquín Parra por correcciones de cohesión, redacción y ortografía, y a Benjamín Fuentes por la explicación del uso de la matriz de Vandermonde durante la búsqueda de coeficientes.

Finalmente, mencionar que para este informe, se hizo uso de inteligencia artificial con el código de LaTex para confeccionar tablas y referencias.

# Capítulo 11

# Aplicación de ajuste por mínimos cuadrados a la evolución de los niveles de $CO_2$

Joaquín Parra, Ignacio Falcón, Alvaro Osses

**Fecha de la actividad:** 10 de Diciembre, 2025

El dióxido de carbono es una molécula fundamental para la vida, siendo parte esencial de la fotosíntesis que realizan las plantas. Sin embargo, su presencia en la atmósfera es problemática dado que es uno de los tantos gases asociados al efecto invernadero. En este capítulo, nos familiarizaremos con los niveles de dióxido de carbono en la atmósfera observando la curva de Keeling y realizaremos un ajuste por método de mínimos cuadrados con la finalidad de obtener un modelo con el cual extrapolar datos.

## 11.1. Curva de Keeling

La curva de Keeling es un gráfico que muestra la concentración de dióxido de carbono en la atmósfera desde 1958, medida desde una estación en Mauna Loa, Hawái, manejado por la Oficina Nacional de Administración Oceánica y Atmósferica (NOAA por sus siglas en inglés). En la figura 11.1 se pueden apreciar los datos recolectados desde 1958. Realizaremos un ajuste sobre los últimos 5 años (Noviembre de 2020 a Noviembre de 2025).

## 11.2. Ajuste sobre los últimos 5 años

Ahora, procederemos a realizar un ajuste por mínimos cuadrados sobre los datos de los últimos años, y testearemos el modelo con los datos de años pasados. El modelo a ajustar es:

$$f(\tau) = p_0 + \sum_{n=1}^{3} \left[ p_n \tau^n + c_n \cos(2\pi n \tau) + s_n \sin(2\pi n \tau) \right], \tag{11.1}$$

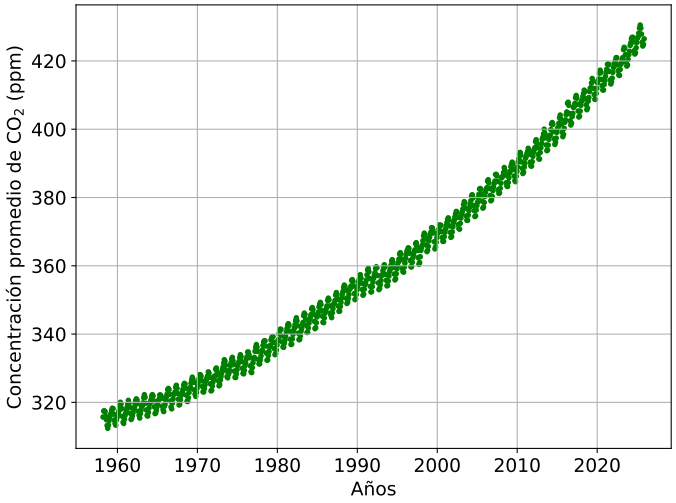


Figura 11.1: Curva de Keeling hecha a partir de los datos del NOAA.

con $\tau = 2(t - \langle t \rangle)/(t_{max} - t_{min})$, con $t$ siendo un arreglo de `numpy` con los datos de los últimos 5 años y $\langle t \rangle = (t_{max} + t_{min})/2$. Se utiliza un tiempo normalizado, pues en las escalas en las que trabaja la curva de Keeling, las potencias de $t$ pueden hacerse muy grandes y podrían traer errores numéricos o de precisión. Notar que con esta normalización $\tau \in [-1, 1]$. Si bien el modelo en sí es no lineal pues posee exponentes de $\tau$ distintos de 1, además de la aparición de funciones trigonométricas, el modelo es lineal **sobre** los parámetros a ajustar $p, c, s$. Con esto en mente, se utilizó el siguiente código para obtener los 10 coeficientes del ajuste:

```
years, values = np.genfromtxt(
    "datos.txt",
    comments="#",
    usecols=(2, 3),
    unpack=True
)

def fil_norm(t, y, t_min, t_max):
    mask = (t >= t_min) & (t <= t_max)
    t_sel = t[mask]
    y_sel = y[mask]

    t_centro = 0.5 * (t_sel.min() + t_sel.max())
    ancho = t_sel.max() - t_sel.min()
    tau = 2 * (t_sel - t_centro) / ancho

    return t_sel, y_sel, tau

t_sel, y_sel, tau = fil_norm(years, values, tmin, tmax)

grado = 3
m = len(tau)
n = 1 + 3 * grado

B = np.empty((m, n))
```

| Parámetro | Valor |
|---|---|
| $p_0$ | 4.20975313e+02 |
| $p_1$ | 8.39005705e+00 |
| $c_1$ | -2.36688097e-01 |
| $s_1$ | -2.03957847e-01 |
| $p_2$ | 7.81697099e-01 |
| $c_2$ | -8.73250486e-02 |
| $s_2$ | -9.84883860e-02 |
| $p_3$ | -3.01850065e+00 |
| $c_3$ | -8.44165599e-02 |
| $s_3$ | 1.37553593e-02 |

Cuadro 11.1: Valores obtenidos por el ajuste.

```
B[:, 0] = 1.0

for j in range(1, grado + 1):
    idx = 1 + 3 * (j - 1)
    B[:, idx]     = tau ** j
    B[:, idx + 1] = np.cos(2 * np.pi * j * tau)
    B[:, idx + 2] = np.sin(2 * np.pi * j * tau)

BT = B.transpose()
coef_fit = np.linalg.inv(BT @ B) @ (BT @ y_sel)

values_fit = B @ coef_fit
mse_value = np.mean((y_sel - values_fit) ** 2)
```

De aquí, reportamos los valores de los coeficientes obtenidos por el ajuste mediante mínimos cuadrados en el cuadro 11.1. Con estos parámetros, nuestro ajuste mediante método de mínimos cuadrados sobre el período comprendido por los últimos 5 años se aprecia en la figura 11.2 junto a los residuos del modelo, los cuales no parecen seguir un patrón ordenado a primera vista. Además, se reporta un error cuadrático medio MSE $\approx 4.6$, o de forma equivalente, un RMSE $= \sqrt{MSE} \approx 2.1$. Este último parámetro se suele comparar con la desviación estándar de los datos, de la cual se reporta un valor de $\sigma = 3.9$, obteniendo un cociente entre ambas de aproximadamente 0.54. El que el RMSE sea menor a la desviación estándar es un buen indicador que el modelo refleja la naturaleza de los datos. Esto, junto a los residuos, dan cuenta de que el ajuste si bien refleja la tendencia de los datos, tiene ciertos problemas prediciendo los valores.

## 11.3. Extrapolación

Finalmente, queda hacer la extrapolación. El modelo pensado para mirar 5 años al pasado se extrapolará a 55 años al pasado, es decir, desde 1970. Existe una sutileza al hacer esto, y es que este nuevo set de años no puede normalizarse de manera idéntica a la forma anterior, pues quedaríamos en el mismo intervalo $[-1, 1]$ y el modelo se comportaría exactamente igual, pero los datos serían distintos. Para arreglar esto, realizaremos una normalización coherente con la anterior: $\tau' = 2(t_{new} - \langle t_{old} \rangle)/(t_{old,max} - t_{old,min})$, donde $t_{new}$ es un arreglo con los datos a normalizar (en este caso, hasta 1970) y $t_{old}$ es el arreglo utilizado anteriormente con los datos de los últimos 5 años. Sobre este nuevo set de valores, se aplica el mismo modelo obtenido en el apartado anterior, como se aprecia en la figura 11.3. Se aprecia de inmediato que el ajuste ya no funciona bien fuera del rango de los 5 años (intervalo $[-1, 1]$), incluso se puede ver en la figura 11.4 como inmediatamente al salir de este intervalo el ajuste ya no predice la tendencia de los datos.

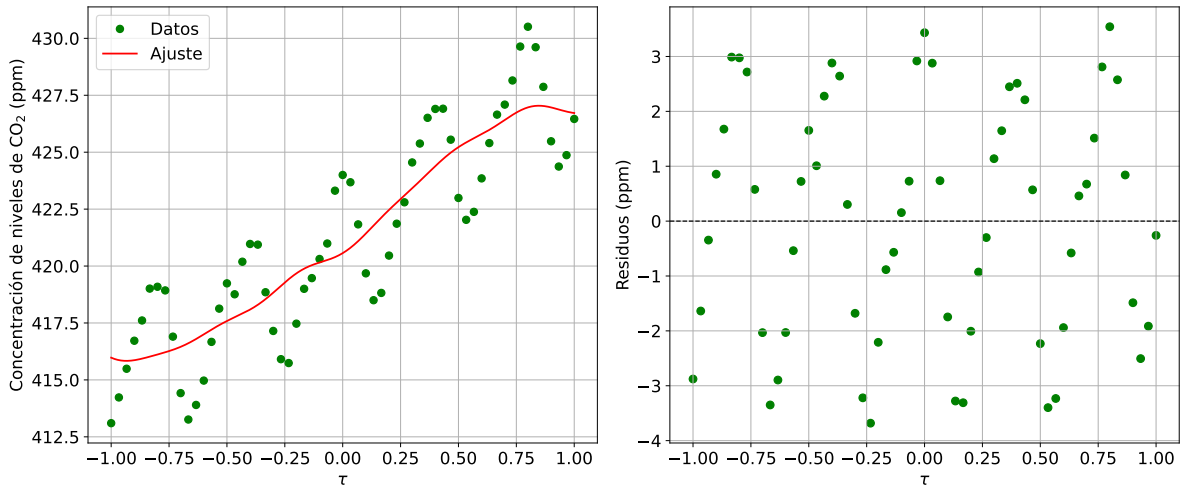


Figura 11.2: Gráfica del ajuste por método de mínimos cuadrados versus datos, junto a sus residuos.

El hecho de que el modelo no resista una extrapolación y que no se ajuste adecuadamente a los datos, puede ser atribuido a que el modelo propuesto no refleja la naturaleza del fenómeno descrito. Al no identificar una tendencia en los residuos, es difícil aventurarse a proponer correcciones, por lo que únicamente nos podemos quedar con que el modelo propuesto, si bien refleja hasta cierto punto la tendencia de los datos en el período de los últimos cinco, falla al predecir sus valores (mirar las magnitudes de los residuos) en este intervalo y no refleja la naturaleza de los datos al salir de esta ventana de tiempo.

## Conclusiones

En este capítulo hicimos uso de un ajuste mediante método de mínimos cuadrados a un modelo propuesto para estudiar un fenómeno contingente como lo son las emisiones de carbono. Este modelo, si bien describe relativamente bien los datos en determinado intervalo de tiempo, en general, no es un buen modelo, pues al salirnos de este intervalo falla rotundamente al predecir la tendencia de las emisiones. Si bien no conocemos las causas de esta discrepancia, aprendimos durante el análisis a realizar ajustes por método de mínimos cuadrados y a comprender la importancia de realizar extrapolaciones/predicciones como forma de testear modelos.

## Agradecimientos

Este capítulo fue escrito por Joaquín Parra, con revisiones tanto de código como de contenido de parte de sus compañeros de trabajo Alvaro Osses e Ignacio Falcón, quienes también ayudaron a la hora de hacer los gráficos del capítulo. Se agradece al profesor guía del curso, Dr. Roberto Navarro por presentar los temas en clases y cuyo código sirvió de base para elaborar los nuestros. Se ha utilizado ayuda de Gemini y ChatGPT para realizar algunos gráficos, así como ayuda al momento de refinar código y presentar conceptos relevantes para usar el error cuadrático medio. Finalmente, se agradece al compañero Israel Bravo por entregar guía y comentarios sobre como implementar operaciones matriciales para resolver por método de mínimos cuadrados.

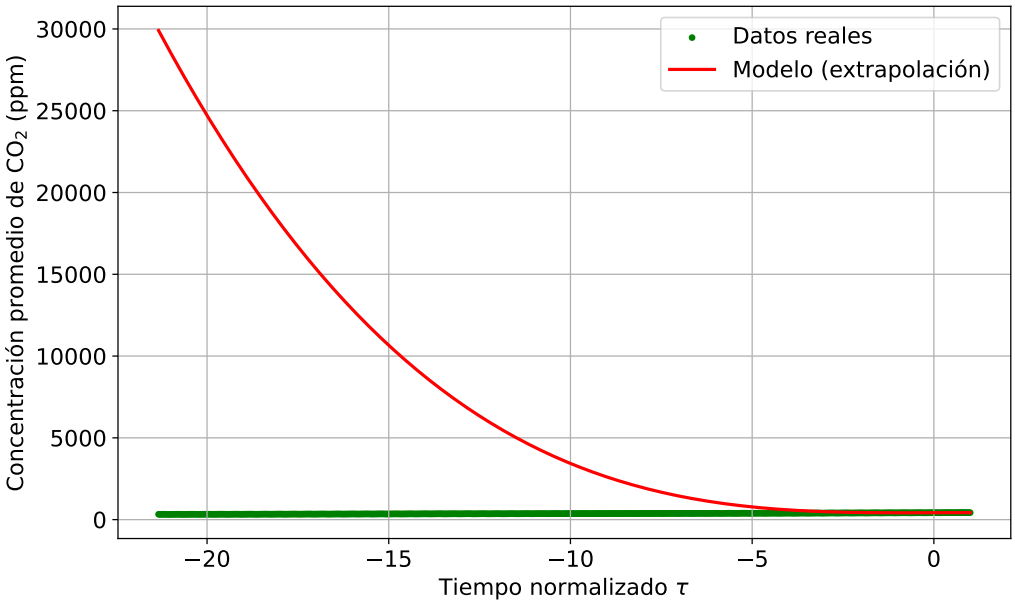


Figura 11.3: Extrapolación del modelo hasta 1970.

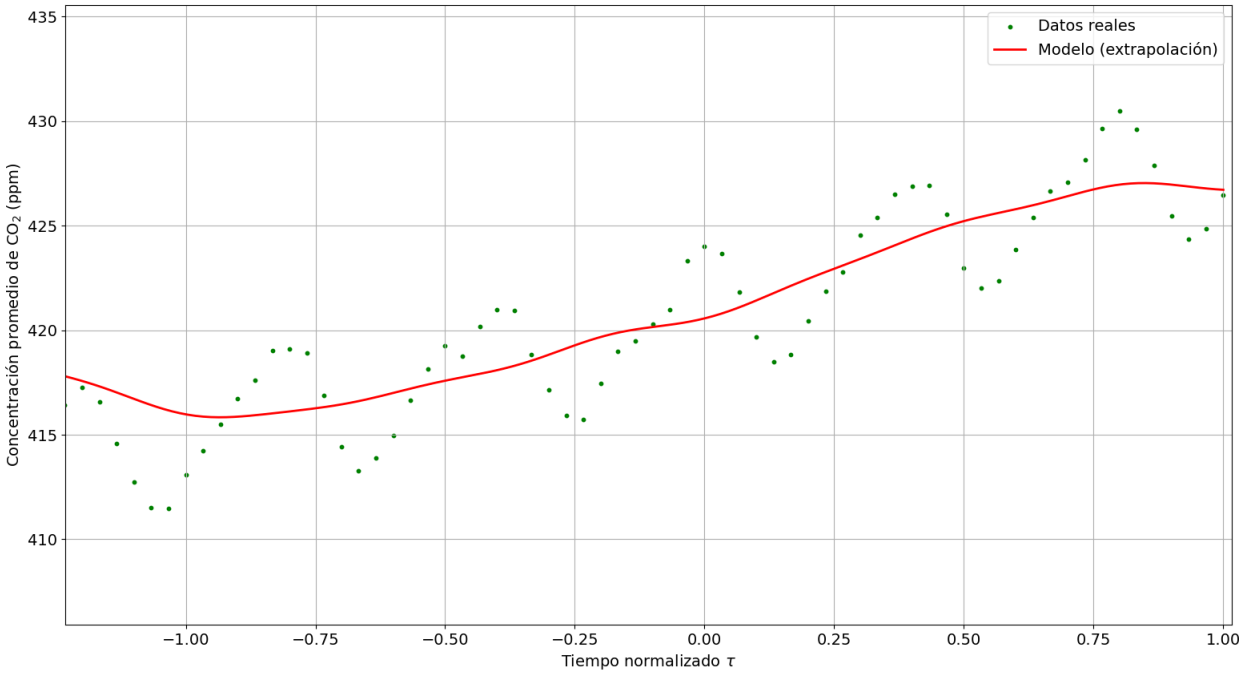


Figura 11.4: Zoom a la figura 11.3. Se aprecia que en entre -1 y 1 se mantiene la forma de la figura 11.2.

# Conclusiones

**Fecha de presentación:** Jueves 12 de diciembre de 2025

El objetivo de este portafolio siempre fue el de llevar registro de los aprendizajes adquiridos durante este semestre y ser una forma autónoma de poder mirar hacia atrás en el tiempo y reflexionar sobre los contenidos de esta asignatura. Durante los once capítulos de este portafolio, quedan expresados (en su mayoría) los contenidos del curso impartidos durante el segundo semestre de 2025, desarrollados por el autor y sus compañeros de trabajo. En el primer capítulo (1), nos familiarizamos con `python` descifrando un mensaje encriptado del mismo, haciendo uso de herramientas nativas. Luego, en el segundo capítulo (2), en la misma linea nos familiarizamos con `numpy`, una biblioteca fundamental para la computación científica de la cual utilizamos sus funciones más conocidas. Cerrando la linea de conocer `python`, en el tercer capítulo (3) aplicamos lo aprendido en los capítulos anteriores para el estudio de los números de Catalán, realizando cómputos y estimaciones de estos.
Comenzando otra línea temática, en el cuarto capítulo (4) estudiamos una derivada de tres puntos no equidistantes, demostrando el error asociado a su aproximación y el orden del mismo, como también su aplicación con nodos de Gauss-Chebyshev. Luego, en el capítulo cinco (5), estudiamos la dinámica de poblaciones mediante el modelo de Lotka-Volterra, haciendo uso de métodos numéricos para ecuaciones diferenciales (en particular, leap-frog) para estudiar el comportamiento de cazadores y presas a la vez que aprendíamos como normalizar ecuaciones mediante el Teorema Pi de Buckingham. Finalmente en el capítulo seis (6), implementamos otros método numérico para ecuaciones diferenciales, el famoso método Runge-Kutta para solucionar un problema de cinemática donde también se aplicó el Teorema Pi.
Abriendo otra linea temática, en el capítulo siete (7) aprendimos como implementar en `python` el método de la secante para hallar ceros, como también analizamos el error asociado a este método. Continuando, en el capítulo ocho (8) estudiamos el método implícito de Euler junto al método de Newton-Raphson para hallar soluciones numéricas a cierto tipo de ecuaciones diferenciales. Finalmente, en el capítulo nueve (9), implementamos el método de la bisección para estudiar la estabilidad de un péndulo invertido conectado a un resorte.
Cerrando con los contenidos del curso, en el capítulo diez (10) se implementaron métodos de integración, como la regla de Simpson $1/3$ compuesta, junto al método de interpolación de Lagrange para realizar un análisis y aproximación de los conocidos polinomios de Legrende. Finalmente, en el capítulo once (11) implementamos un método de mínimos cuadrados para ajustar un modelo que busca modelar datos reales de concentración promedio de dióxido de carbono en la atmósfera.
En relación al trabajo personal, si bien se cometieron errores señalados por el docente en las retroalimentaciones, considero que fui constante durante la creación de los capítulos y gracias a esas retroalimentaciones pude adaptarme a una forma más profesional y estandarizada de redactar en ciencias. Reconozco que en algunas ocasiones (a excepción de una que se comenta más abajo), se dejó la elaboración de los capítulos para un par de día antes de la fecha de entrega de las tareas dado que en la mayoría de ocasiones existían evaluaciones cercanas en otras asignaturas y se decidió aplazar el portafolio al ser más práctico, lo cual pudo haber afectado a la redacción y profundidad de las respuestas entregadas. En un futuro, me gustaría poder organizar mejor mis tiempos y corroborar mis resultados.

Mis expectativas sobre la asignatura fueron sobrepasadas, pues si bien sabía que por fin iba a introducirme al mundo de los métodos numéricos, pensé que el enfoque iba a ser mucho más práctico, quizás algo más cercano a la asignatura de Cálculo Numérico que toman en ingeniería. Para mi sorpresa, esta asignatura tuvo un fuerte componente teórico, que si bien a ratos fue difícil, hizo la asignatura mucho más agradable para mi. Percibo que cierro esta asignatura con una amplio espectro de métodos para afrontar los distintos problemas que vengan durante la carrera, al menos hasta que llegue Física Computacional 3. Lo que más destaco es el capítulo acerca de la dinámica de poblaciones, el cual fue escrito mayoritariamente por mi. Ese capítulo marcó un antes y un después para mi en esta asignatura, pues no solamente disfrute estudiando el método de salto de rana, si no que una cadena involuntaria de clicks durante la investigación para realizar el capítulo me llevó a realizar el desarrollo extra que posee ese capítulo. Recuerdo que cuando encontré la función $H(p, d)$ y noté que era igual a la función propuesta $C(p, d)$ de la tarea, fue el momento más eufórico del semestre. Fue un capítulo que de verdad disfruté mucho hacer (también se dio que no habían otras evaluaciones por aquellas fechas) y me motivó a estudiar y darle más tiempo a esta asignatura, independiente de como retornasen las futuras notas.

Finalmente, agradecer al docente del curso, Dr. Roberto Navarro y los ayudantes del curso por llevar la asignatura de una forma que personalmente se me hizo entretenida y pedagógica. Si bien a ratos el curso se hizo denso, las notas no fueron las mejores y me costó muchas veces aprender a implementar los contenidos en `python`, salgo de este curso con un buen sabor de boca y seguro que lo que aprendí será fundamental para el resto de mi carrera como científico.

# Bibliografía


Academia Lab,, "Matriz de Vandermonde," https://academia-lab.com/enciclopedia/matriz-de-vandermonde/ (2025), enciclopedia en línea. Consultado el 18 de diciembre de 2025.

Anisiu, M.-C., Didactica Mathematica **32**, 9 (2014).

Butcher, J. C., *Numerical Methods for Ordinary Differential Equations*, 2nd ed. (Wiley, 2003) p. 64.

Departamento de Física Aplicada, UPV/EHU,, "Polinomios de chebyshev," http://www.sc.ehu.es/sbweb/fisica3/especial/chebyshev/chebyshev.html (2025), [Online; accessed 3-November-2025].

Desmos, Inc.,, "Desmos Gráfica de Funciones (Online Graphing Calculator)," https://www.desmos.com (2025), recurso de software en línea. Revisado el 21 de noviembre de 2025.

Encyclopedia of Mathematics,, "Lagrange interpolation formula," https://encyclopediaofmath.org/index.php?title=Lagrange_interpolation_formula (2025), artículo en línea. Consultado el 18 de diciembre de 2025.

Facultad de Ciencias Astronómicas y Geofísicas, UNLP,, "Apuntes Teóricos: Polinomios de Legendre (Ortogonalidad)," http://mat-avanzadas.fcaglp.unlp.edu.ar/public_html/Mat-Esp-II/pdfs/apuntes-teoricos/Legendre.pdfs (2025), documento en línea. Consultado el 18 de diciembre de 2025.

Goldstein, H., *MECÁNICA CLÁSICA*, segunda (2da) edición ed. (EDITORIAL REVERTÉ S.A., 1994) p. 444.

Gonzáles-Santos, G., Jou. Cie. Ing. **11**, 7 (2019).

Math for College,, "Secant method for solving a nonlinear equation," https://mathforcollege.com/nm/mws/gen/03nle/mws_gen_nle_txt_secant.pdf (s.f.), recuperado el 18 de noviembre de 2025.

Mena Gutiérrez, D. J., *Cuadratura Gauss-Chebyshev*, Ph.D. thesis, Universidad Nacional Autónoma de Nicaragua (UNAN) (2019), [Online; accessed 3-November-2025].

Merino, M., "Tema 3: Interpolación polinomial — análisis numérico i," http://www.uhu.es/manuel.merino/files/Analisis_NumericoI/Teoria/Tema3.pdf (2006/07), [Online; accessed 3-November-2025].

Nutku, and Y.,, Physics Letters A **145**, 27 (1990).

Osses, A., "Ecuaciones diferenciales ordinarias ma26a," https://www.dim.uchile.cl/~operedo/apuntes/apunteEDO.pdf (2005).

P. Kundu, I. C., *Fluid Mechanics*, 4th ed. (Elsevier/Academic Press, 2008) pp. 284–287.

Shu, C.-W. . S. O., Journak of Computational Physics **77.2**, 439 (1988).

Solís A. Cordero A.„ "Condiciones de Convergencia," https://www.redalyc.org/journal/6079/607964424002/html/ (2012).

Stewart, J., *Cálculo: Trascendentes tempranas*, 8th ed., Vol. 2 (Cengage Learning, Boston, 2016).

U. Politecnica de Cataluña„ "Ceros de Funciones," https://portal.camins.upc.edu/materials_guia/250133/2012/CerosFunciones.pdf (2012).

Wikimedia Foundation, Inc.„ "Lagrange polynomial," https://en.wikipedia.org/wiki/Lagrange_polynomial (2025a), artículo de Wikipedia. Consultado el 18 de diciembre de 2025.

Wikimedia Foundation, Inc.„ "Polinomios de Legendre," https://es.wikipedia.org/wiki/Polinomios_de_Legendre (2025b), artículo de Wikipedia. Consultado el 18 de diciembre de 2025.

Wikipedia„ "Método de la secante," https://es.wikipedia.org/wiki/M%C3%A9todo_de_la_secante (2025), wikipedia, La enciclopedia libre. Revisado el 18 de noviembre de 2025.

Wikipedia contributors„ "Ascii — Wikipedia, the free encyclopedia," (2025a), [Online; accessed 4-November-2025].

Wikipedia contributors„ "Rot13 — Wikipedia, the free encyclopedia," (2025b), [Online; accessed 4-November-2025].

Zill, D., *Ecuaciones Diferenciales con problemas de valores de frontera*, novena (9a) edición ed. (Cengage Learning, 2018) pp. 408–409.