Métodos Numéricos con Python

Marcos Prunello

2024-01-01

Tabla de contenidos

Prefacio

El presente documento es la guía de estudio para la asignatura Métodos Numéricos de la Licenciatura en Estadística (Universidad Nacional de Rosario). Se ha utilizado como fuente para la creación de este material a la bibliografía mencionada en el programa de la asignatura. La asignatura se complementa con variados materiales (prácticas, ejemplos, proyectos) disponibles en el aula virtual del curso de acceso privado.

Estos apuntes no están libres de contener errores. Sugerencias para corregirlos o para expresar de manera más adecuada las ideas volcadas son siempre bienvenidas[1].

Primera publicación: enero 2024.

[1] En general, no se cuenta con derechos para las imágenes empleadas (a menos que sean de creación propia). Ante cualquier problema, contactar al autor.

1 Conceptos básicos de análisis numérico

1.1. Errores de truncamiento, de redondeo y aritmética computacional

- Como ya hemos dicho, un método numérico propone un algoritmo para resolver de forma aproximada un problema que no se puede resolver mediante métodos analíticos.
- Esto hace que inevitablemente haya errores en las soluciones obtenidas.

Definición: se llama **error de truncamiento** a la diferencia entre el valor aproximado propuesto por el método y la solución exacta del problema.

- Este tipo de error ocurre cuando un proceso que requiere un número infinito de pasos debe ser detenido en una cantidad finita de iteraciones.
- Por ejemplo, podemos recordar el desarrollo en serie de Taylor de la función $f(x) = e^{x^2}$ alrededor del cero:

$$e^{x^2} = 1 + x^2 + \frac{x^4}{2!} + \frac{x^6}{3!} + ... + \frac{x^{2n}}{n!} + ...$$

- Si nos quedamos sólo con los primeros 4 términos, estamos aproximando una suma que tiene una infinita cantidad de sumandos sólo con los primeros 4, de manera que dicha aproximación presentará un error de truncamiento.
- Mientras que el valor exacto de $f(0.5) = e^{0.5^2}$ es 1.2840, la aproximación de Taylor con 4 términos da 1.2839. Esta diferencia es un *error de truncamiento*.
- Sin embargo, al realizar cálculos con una máquina se presenta otro tipo de errores que generalmente ignoramos.
- Esto pasa porque la aritmética realizada con una calculadora o computadora es diferente a la que hacemos "mentalmente".
- Ejemplo: sabemos que $(\sqrt{3})^2 = 3$. Pero... ¿qué pasa si corremos en Python lo siguiente?

```
from math import sqrt
sqrt(3)**2 == 3
#> False
```

- En nuestro mundo matemático tradicional, los números pueden tener una infinita cantidad de dígitos. Por eso podemos operar con números como $\sqrt{3}$, que es irracional.
- ¿Pero una computadora? ¿Puede trabajar con infinitos dígitos?

- En el mundo computacional cada número tiene una cantidad fija y finita de dígitos. Sólo los números racionales (y no todos) pueden ser representados de forma exacta por la computadora.
- Entonces, la máquina trabaja con una representación aproximada de $\sqrt{3}$ que no es exactamente igual a ese valor. Sin embargo, esa aproximación a $\sqrt{3}$ que hace la compu explica el resultado de `sqrt(3)**2 == 3`.
- En muchos casos esa aproximación es aceptable y no le damos importancia a la diferencia que tiene con respecto al valor exacto. En otros casos, esto puede generar problemas.

Definición: se llama **error de redondeo** al error que se produce cuando se utiliza una computadora para realizar cálculos con números reales, debido a que la aritmética realizada en una máquina incluye números con una cantidad finita de dígitos, dando como resultado cálculos realizados con representaciones aproximadas de los números reales.

- El error de redondeo está ligado fundamentalmente al tipo de precisión que se emplee (determinado por el procesador y software usados). Sin embargo, el efecto final de los errores de redondeo depende también del algoritmo propuesto para aplicar un método numérico y de la forma de programarlo.
- Existen operaciones que son especialmente sensibles a los errores de redondeo o también puede ser que un algoritmo haga que los mismos se amplifiquen.
- Si bien estamos acostumbrados a realizar cálculos con el sistema decimal, las computadoras operan con el sistema binario. Por eso, vamos a empezar comentando qué es un sistema de numeración y en qué se diferencia el decimal del binario.

1.2. Sistemas de numeración

Definición: un **sistema de numeración** es un conjunto de símbolos y reglas que permiten construir todos los números válidos.

- **Conjunto de símbolos**: en el sistema decimal usamos son los dígitos 0, 1, 2, …, 9; mientras que en el sistema binario se usan solo el 0 y el 1. En el sistema octal los símbolos son 0, 1, …, 7; en el hexadecimal son 0, 1, …, 9, A, B, C, D, E, F; y en el romano, I, V, X, L, C, D, M.
- **Conjunto de reglas**: indican qué operaciones son válidas. Por ejemplo, el sistema decimal es **posicional**, porque el valor de un dígito depende tanto del símbolo como de su posición en el número: 350 y 530 tienen los mismos dígitos pero representan magnitudes diferentes, mientras que el 5 en el primer caso aporta "50" (se posiciona en la decena), el 5 en el segundo caso aporta "500" (se posiciona el la centena). En cambio, el sistema romano es **no posicional**: los dígitos tienen el valor del símbolo utilizado y no depende de la posición que ocupa, sino que se va sumando o restando su valor (MMXXII = 1000 + 1000 + 10 + 10 + 1 + 1 = 2022).
- En un sistema de numeración posicional, se le llama **base** al número que define el orden de magnitud en que se ve incrementada cada una de las cifras sucesivas que componen

el número (y también es la cantidad de símbolos presentes en dicho sistema).

1.2.1. El sistema decimal

Definición: el **sistema de numeración decimal** es un sistema de numeración posicional cuya base es igual a 10. Los dígitos que se utilizan son 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9.

- En el sistema decimal cada dígito es multiplicado por una potencia de 10, con el exponente determinado por la posición de la dígito con respecto al punto decimal.
- Por ejemplo, el número decimal 1563 se puede escribir en forma desarrollada utilizando potencias con base 10 así:

$$\begin{aligned} 1563 &= 1000 + 500 + 60 + 3 \\ &= (1 \times 10^3) + (5 \times 10^2) + (6 \times 10^1) + (3 \times 10^0) \end{aligned}$$

- Entonces interpretamos que el 1 "aporta" mil unidades porque por su posición hay que considerarlo multiplicado por la tercera potencia de 10. Lo mismo con los otros dígitos.
- Los números con parte fraccionaria se pueden expresar así:

$$\begin{aligned} 16.302 &= 10 + 6 + 0.3 + 0.00 + 0.002 \\ &= (1 \times 10^1) + (6 \times 10^0) + (3 \times 10^{-1}) + (0 \times 10^{-2}) + (2 \times 10^{-3}) \end{aligned}$$

- Cuando usamos el sistema decimal, escribimos el "punto decimal" después del dígito que va multiplicado por 10^0. Por ejemplo, 4.13 o 2874.1.
- Sin embargo, esa forma de escribir los números no es práctica para trabajar con magnitudes muy grandes o muy pequeñas, porque ocupan muchos dígitos y porque es probable que muchos de ellos no provean información "exacta".
- En esos casos, se recurre a su representación en **notación científica**.

Definición: la **notación científica** de un número real r está compuesta por:

$$r = c \times b^e$$

- c: el coeficiente, formado por un número real (negativo o positvo).
- b: la base (10 en el sistema decimal).
- e: el exponente u "orden de magnitud", que eleva la base a una potencia.
- Por ejemplo:
  - El número -2.3×10^3 es -2300. También puede escribirse $-2.3E3$ (aquí E no tiene nada que ver con la constante matemática $e = 2.718282...$).
  - El número $0.01E-7$ es 0.000000001.
  - El número $34E5$ es 3400000.
- Se considera que el número de dígitos en el coeficiente es la cantidad de **cifras o dígitos significativos**. Esta expresión se usa para describir vagamente el número de dígitos

decimales que parecen ser exactos, es decir, en los que se puede "confiar", que aportan información.[1].

- Por ejemplo:
  - La masa de un protón es igual a 0.00000000000000000000000000167 kg. En notación científica estándar es igual a $1.67E-27$. Los dígitos 1, 6 y 7 son los que "importan", tiene 3 cifras significativas.
  - La circunferencia de la Tierra en el Ecuador es $40\,091\,000\,m$. Si en notación científica aparece como $4.0091E7$, entendemos que presenta 5 cifras significativas y que el valor exacto tal vez ronda entre $40\,090\,500\,m$ y $40\,091\,500\,m$.

1.2.2. El sistema binario

Definición: el **sistema de numeración binario** es un sistema de numeración posicional cuya base es igual a 2. Los dígitos que se utilizan son el 0 y el 1.

- En el sistema binario el número 1563 se escribe como 11000011011. Para no hacer lío, se suele usar un subíndice para señalar el sistema elegido para representar a un número: $1563_{(10)} = 11000011011_{(2)}$.
- ¿Y cómo podemos corroborar que esto es así?

$$\begin{aligned} 11000011011_{(2)} &= (1\times 2^{10}) + (1\times 2^9) + (0\times 2^8) + (0\times 2^7) + (0\times 2^6) + \\ &\quad (0\times 2^5) + (1\times 2^4) + (1\times 2^3) + (0\times 2^2) + (1\times 2^1) + (1\times 2^0) \\ &= (1\times 2^{10}) + (1\times 2^9) + (1\times 2^4) + (1\times 2^3) + (1\times 2^1) + (1\times 2^0) \\ &= 1024 + 512 + 16 + 8 + 2 + 1 \\ &= 1563_{(10)} \end{aligned}$$

- Los números binarios también admiten partes fraccionarias. Por ejemplo:

$$\begin{aligned} 11.0101_{(2)} &= (1\times 2^1) + (1\times 2^0) + (0\times 2^{-1}) + (1\times 2^{-2}) + (0\times 2^{-3}) + (1\times 2^{-4}) \\ &= 2 + 1 + \frac{1}{4} + \frac{1}{16} \\ &= 3.3125_{(10)} \end{aligned}$$

1.2.3. Conversión de decimal a binario

- En los ejemplos de arriba se vio cómo convertir de binario a decimal.
- Ahora vamos a mencionar cómo se hace la conversión al revés.

Conversión de decimales enteros a binario

[1] Hay algunas reglas para determinar cuántas cifras significativas tienen los números que no estén expresados en un formato normalizado de punto flotante decimal, pero no nos vamos a preocupar por eso.

- A la parte entera hay que dividirla sucesivamente por la base 2, hasta obtener un cociente igual a cero.
- El conjunto de los restos de las sucesivas divisiones, ordenados desde el último hasta el primero, constituyen el número en formato binario.
- Ejemplo:

$$\begin{aligned} 123 &= 61 \times 2 + 1 \\ 61 &= 30 \times 2 + 1 \\ 30 &= 15 \times 2 + 0 \\ 15 &= 7 \times 2 + 1 \\ 7 &= 3 \times 2 + 1 \\ 3 &= 1 \times 2 + 1 \\ 1 &= 0 \times 2 + 1 \end{aligned}$$

- De manera que $123_{(10)} = 1111011_{(2)}$.

Conversión de números decimales fraccionarios a binario

- A la parte fraccionaria hay que multiplicarla sucesivamente por 2, hasta que la misma se haga 0 o se alcance un número deseado de dígitos.
- El conjunto de los dígitos delante de la coma forman el número binario.
- Ejemplo:

$$\begin{aligned} 0.3125 \times 2 &= 0.625 \\ 0.625 \times 2 &= 1.25 \\ 0.25 \times 2 &= 0.5 \\ 0.5 \times 2 &= 1.0 \end{aligned}$$

- De manera que $0.3125_{(10)} = 0.0101_{(2)}$.
- Combinando ambos ejemplos: $123.3125_{(10)} = 1111011.0101_{(2)}$.

1.2.4. ¿Por qué nos interesa el sistema binario?

- Porque es el sistema de representación numérica que utilizan las computadoras.
- Este sistema es natural para las computadoras ya que su memoria consiste de un enorme número de dispositivos de registro electrónico, en los que cada elemento sólo tiene los estados de "encendido" y "apagado".
- Estos elementos constituyen la unidad mínima de información, sólo pueden tomar dos valores posibles, `0` o `1`, y reciben el nombre de **bit** (*bi*nary digi*t*).
- Aunque nosotros no nos damos cuenta, toda operación numérica que le indicamos a la computadora en sistema decimal, es traducida y procesada internamente en binario.
- Por lo tanto es muy importante entender cómo opera la computadora, para entender qué sucede con las operaciones que queremos que realice. Por ejemplo…

Ejercicio: Escribir un programa para realizar las siguientes operaciones, empleando estructuras iterativas para las sumatorias:

a) $10000 - \sum_{i=1}^{100000} 0.1$
b) $10000 - \sum_{i=1}^{80000} 0.125$

¿Cuál es el resultado exacto en estos cálculos? ¿Qué resultados arrojó la computadora? ¿Por qué?

Respuesta:

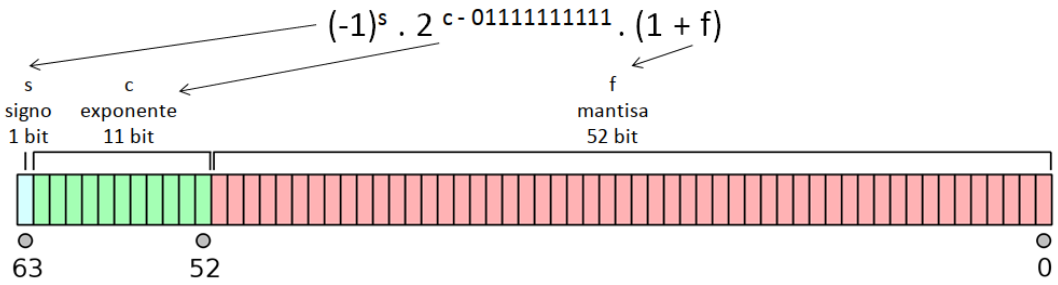
- Pensemos en el número decimal periódico $1/3 = 0.\overline{3}$.
- Para aproximarlo, sólo podemos usar una cantidad finita de cifras, por ejemplo, 0.333 o 0.33333. Estas aproximaciones guardan cierto error, que depende de la cantidad de cifras empleadas.
- Con los números binarios ocurre exactamente lo mismo.
- $0.1_{(10)} = 0.0001100110011..._{(2)} = 0.0\overline{0011}_{(2)}$ (verificación opcional). Es decir, la representación de 0,1 en binario es periódica, la computadora necesariamente debe redondear o truncar para almacenar y operar.
- Por esta razón, sumar 100 mil veces $0,1$ no da exactmente 10000.
- Por el contrario, 0.125 en binario no es periódico, la computadora lo puede representar exactamente y no se produjo error.

Actividad opcional: verificar $0.1_{(10)} = 0.0\overline{0011}_{(2)}$ y encontrar la representación binaria de $0.125_{(10)}$.

1.3. Números de máquina binarios

- Ya sabemos que la computadora emplea el sistema binario. Ahora bien, ¿cómo se organiza para almacenar los números?
- Utiliza un sistema conocido como **representación de punto (o coma) flotante** (en inglés, *floating point*).
- Existe un protocolo que es usado por todas las computadoras actuales y que establece las reglas para este tipo de representación.
- Se lo conoce como **IEEE-754** ya que fue publicado por el *Institute for Electrical and Electronic Engineers* en 1985 y actualizado en 2008.
- Este estándar define dos tipos de formatos: el de *precisión simple* (en el cual cada número ocupa 32 bit de memoria) y el de *precisión doble* (un número ocupa 64 bit).
- El formato de doble precisión en 64 bit, empleado actualmente en casi todas las computadoras, establece que todo número real es representado por la computadora con una aproximación binaria del tipo:

$$(-1)^s \times 2^{c-01111111111} \times (1+f)$$



- El primer bit es indicador de **signo**, s: vale 0 si el número es positivo o 1 si es negativo.
- Le sigue un **exponente** o **característica** c que ocupa 11 bits dando lugar a $2^{11} = 2048$ valores distintos, entre el 0 y 2047. Sin embargo, a c se le resta $01111111111_{(2)} = 1023_{(10)}$ para tener exponentes negativos y positivos entre -1023 y 1024, lo cual produce una mejora en la representación de números con magnitud pequeña.
- Finalmente, la representación termina con una **fracción** binaria de 52 bits que se llama **mantisa**, f.
- Ejemplo: el siguiente conjunto de 64 bits representa al número decimal 74.5:

$$0100000001010010101000000000000000000000000000000000000000000000$$

0	1	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Verificación opcional del ejemplo:

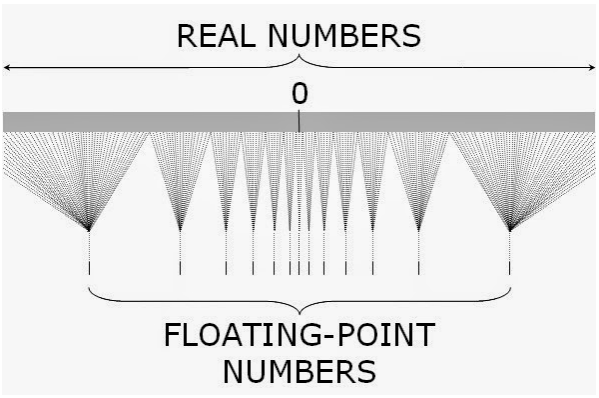
- Primero el exponente: $c = 10000000101_{(2)} = 2^{10} + 2^2 + 2^0 = 1029_{(10)}, \quad 1029 - 1023 = 6.$
- Luego la mantisa (recordar que es fraccionaria):

$$\begin{aligned} 0010101_{(2)} &= 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} + 0 \times 2^{-6} + 1 \times 2^{-7} \\ &= 1 \times 2^{-3} + 1 \times 2^{-5} + 1 \times 2^{-7} \\ &= \frac{1}{8} + \frac{1}{32} + \frac{1}{128} \\ &= 0.1640625_{(10)} \end{aligned}$$

- Juntando todo:

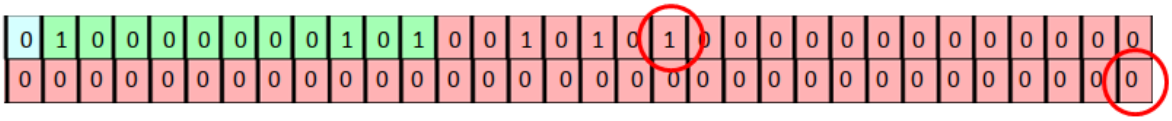
$$(-1)^0 \times 2^{1029-1023} \times (1 + 0.1640625) = (-1) \times 64 \times 1.1640625 = 74.5_{(10)}$$

- En este link o en este otro se puede encontrar una calculadora que convierte números entre sus representaciones en decimal y en coma flotante.
- A través de este sistema, las computadoras sólo pueden representar un subconjunto de los números racionales y no pueden representar números irracionales como π o $\sqrt{3}$ dado que tienen infinitos decimales no periódicos.

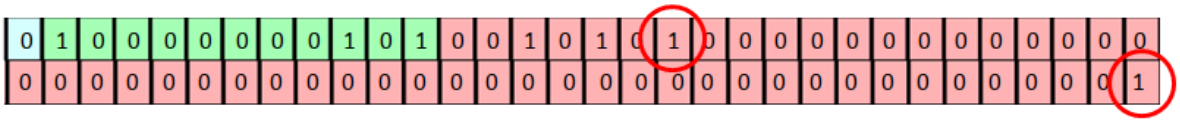


- Esto hace que en la representación surjan los errores de redondeo mencionados en ejemplos anteriores.
- Por ejemplo, este era el número de máquina del ejemplo:

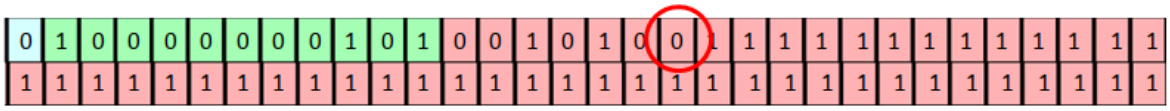
$$0100000001010010101000000000000000000000000000000000000000000000$$



- Este es el número más grande que le sigue. En decimal, es 74.500000000000014210854715:



- Este es el número más chico que le sigue. En decimal, es 74.499999999999985789145284:



- Esto significa el número de máquina

$$0100000001010010101000000000000000000000000000000000000000000000$$

no solo representa al 74.5, sino aproximadamente a la mitad de los números que están entre 74.499999999999985789145284 y 74.500000000000014210854715.

- El número positivo normalizado más pequeño que se puede representar tiene $s = 0$, $c = 1$ y $f = 0$ y es equivalente a :

$$2^{-1022} \approx 0.22251 \times 10^{-307}$$

- Los números que se presentan en los cálculos que tienen una magnitud menor que esa resultan en un **subdesbordamiento** (*underflow*) y, en general, se configuran en cero.
- El número positivo normalizado más grande que se puede representar tiene $s = 0$, $c = 2046$ y $f = 1 - 2^{-52}$ y es equivalente a :

$$2^{1023} \times (2 - 2^{-52}) \approx 0.17977 \times 10^{309}$$

- Los números superiores resultan en **desbordamiento** (*overflow*) y, comúnmente, causan que los cálculos se detengan.

1.4. Poda, redondeo y medida del error

- En la sección anterior quedó en claro que la computadora solo puede trabajar con una aproximación finita de cualquier número que nos interese.
- Muchos "problemas" pueden generarse por esta situación.
- Para examinar estos problemas y medir los errores de redondeo, utilizaremos números decimales, ya que nos resultan más familiares que los binarios.
- Vamos a considerar que para representar a los números estamos restringidos a usar el siguiente **formato normalizado de punto flotante decimal**:

$$\pm 0.d_1 d_2 \cdots d_k \times 10^n, \quad 1 \leq d_1 \leq 9 \quad y \quad 0 \leq d_i \leq 9 \quad i = 2, \cdots, k$$

- Cualquier real y puede ser expresado en un formato normalizado como ese, pero claro, usando cualquier cantidad de dígitos (a veces infinitos):

$$y = \pm 0.d_1 d_2 \cdots d_k d_{k+1} d_{k+2} ... \times 10^n$$

- Cuando un número se informa de esta manera, generalmente se considera que la cantidad de dígitos que están en la mantisa (los d_i) son los **dígitos o cifras significativas** del número.
- Para emular la aritmética finita que manejan las computadoras, hay que restringir la representación de y a nuestro sistema que sólo permite k dígitos en la mantisa. A esto le decimos *forma de punto flotante de y* y se denota $fl(y)$.
- Existen dos formas de "quedarnos" sólo con k dígitos:

El **método de corte o poda** consiste en simplemente cortar los dígitos $d_{k+1} d_{k+2}$..., produciendo:

$$fl(y) = \pm 0.d_1 d_2 \cdots d_k \times 10^n$$

- Por ejemplo, el número π tiene una expansión decimal infinita de la forma $\pi = 3.14159265....$ Escrito en forma normalizada es: $\pi = 0.314159265... \times 10^1$. Si

tenemos que representarlo con $k = 5$ dígitos usando poda, el formato de punto flotante de π es:

$$fl(\pi) = 0.31415 \times 10^1$$

El **método de redondeo** consiste en sumarle 1 a d_k si $d_{k+1} \geq 5$ (*redondear hacia arriba*) o cortarlo reteniendo los primeros k dígitos si $d_{k+1} \leq 5$ (*redondear hacia abajo*)[2]. Esto hace que los dígitos puedan quedar distintos, entonces:

$$fl(y) = \pm 0.\delta_1\delta_2 \cdots \delta_k \times 10^n$$

Si se redondea hacia abajo, $\delta_i = d_i \, \forall i$, pero si se redondea hacia arriba pueden cambiar los dígitos e incluso el exponente.

- En el ejemplo anterior, como el sexto dígito de la expansión decimal de π es un 9, el formato de punto flotante con redondeo de cinco dígitos es:

$$fl(\pi) = 0.31416 \times 10^1 = 3.1416$$

- Tener que aproximar a π con un formato de precisión finita introduce error.

Definición: se llama **error de redondeo** al error que resulta de reemplazar un número por su forma de punto flotante (independientemente de si se usa el método de redondeo o de poda)

- Vamos a definir tres formas de medir errores de aproximación.

Definición: sea p^* una aproximación a p.

- **Error real**: $E = p - p^*$
- **Error absoluto**: $EA = |p - p^*|$
- **Error relativo**: $ER = \frac{|p-p^*|}{|p|}$, siempre que $p \neq 0$.
- El error real y el absoluto se miden en la misma unidad de la variable que se trata de aproximar, mientras que el error relativo se puede interpretar como un porcentaje y es independiente de las unidades de medida.
- En general, el error relativo es una mejor medición de precisión que el error absoluto porque considera el tamaño del número que se va a aproximar (ver ejemplo 2 de la página 14 del libro de Burden).
- A veces no se puede encontrar un valor preciso para el error verdadero en una aproximación, pero se puede encontrar una cota para el error, lo cual proporciona una idea de cuál es "el peor error posible".
- Por ejemplo, se puede demostrar que si representamos a un real y en el formato de punto flotante decimal de k dígitos visto antes con poda, el error relativo de la aproximación queda acotado por:

[2] Hay otros métodos de redondeo más sofisticados.

$$ER = \left| \frac{y - fl(y)}{y} \right| \leq 10^{-k+1}$$

y si se usa redondeo:

$$ER = \left| \frac{y - fl(y)}{y} \right| \leq 0.5 \times 10^{-k+1}$$

- Estas cotas para el error relativo son independientes del número que se va a representar y esto se debe a que la cantidad de números de máquina decimales que se pueden representar en cada intervalo $[10^n, 10^{n+1}]$ es la misma para todo n. Es decir, este formato admite la representación de la misma cantidad de números dentro de cada uno de estos intervalos: $[0.1, 1]$, $[1, 10]$, $[10, 100]$, etc.

1.5. Aritmética de dígitos finitos

- Ya vimos que tenemos el problema de que la representación de los números no es exacta.
- A esto se le suma el inconveniente de que la aritmética que se efectúa en una computadora tampoco es exacta.
- La mecánica real de las operaciones aritméticas que realiza la computadora manipulando los bits es compleja, por eso vamos a seguir ejemplificando estas cuestiones con el sistema decimal, operando bajo un formato de punto flotante restringido a k dígitos.
- Si queremos sumar dos números reales x e y, primero tenemos que buscar su representación de punto flotante, $fl(x)$ y $fl(y)$, luego hacemos la suma entre ellas $fl(x) + fl(y)$ y a este resultado lo expresamos en punto flotante. Por lo tanto, la suma entre x e y es representada por:

$$fl(fl(x) + fl(y))$$

- Hacemos lo mismo con otras operaciones.

Ejemplo: utilizar el corte de cinco dígitos para calcular $x+y$, $x-y$, $x \times y$ y x/y para $x = 5/7$ e $y = 1/3$

$$fl(x) = 0.71428 \times 10^0 \qquad fl(y) = 0.33333 \times 10^0$$

- *Suma*. La representación de $x + y$ en punto flotante es: 0.10476×10^1:

$$fl(0.71428 \times 10^0 + 0.33333 \times 10^0) = fl(1.04761) = 0.10476 \times 10^1$$

Siendo el valor verdadero $5/7 + 1/3 = 22/21$, tenemos:

$$EA = \left| \frac{22}{21} - 0.10476 \times 10^1 \right| = 0.000019048 = 0.190 \times 10^{-4}$$

$$ER = \left| \frac{0.19048 \times 10^{-4}}{22/21} \right| = 0.000018182 = 0.181 \times 10^{-4}$$

- Realizar los cálculos para las otras operaciones.
- El ejemplo anterior ilustra que los errores son inherentes a la aritmética finita que realizan las computadoras.
- Particularmente, hay algunos tipos de operaciones conocidos por ser particularmente problemáticos:
- **Sustracción de números casi iguales**:
  - Cuando se restan números similares el resultado tiene menos cifras significativas que los valores originales (**pérdida de cifras significativas** o **cancelación catastrófica**).
  - Por ejemplo: sean $p = 0.54617 \times 10^0$ y $q = 0.54601 \times 10^0$. Si usamos una aritmética de 5 dígitos para aproximar $p - q$ nos queda:

$$fl(0.54617 \times 10^0 - 0.54601 \times 10^0) = fl(0.00016 \times 10^0) = 0.16 \times 10^{-3}$$

  - Mientras que p y q tenían 5 cifras significativas cada uno, la aproximación para la resta solo tiene 2.
  - Esto puede producir una reducción en la precisión final de la respuesta calculada.
- **Adición de un número grande y uno pequeño**:
  - Puede hacer que el pequeño desaparezca.
  - Por ejemplo: sean $p = 0.96581 \times 10^5$ y $q = 0.37712 \times 10^0$. Se debe sumarlo usando una aritmética de 5 dígitos:

$$\begin{aligned} fl(0.96581 \times 10^5 + 0.37712 \times 10^0) &= fl(96581 \times 10^0 + 0.37712 \times 10^0) \\ &= fl(96581.37712 \times 10^0) \\ &= 0.96581 \times 10^5 \\ &= p \end{aligned}$$

  - En ciertos casos esto no ocasiona un problema ya que, si tenemos un número de gran magnitud probablemente podamos considerar al más pequeño despreciable.
  - Sin embargo debe tenerse mucho cuidado con el orden de las operaciones. Por ejemplo, si sumamos una gran cantidad de números pequeños entre ellos (que juntos tienen un peso considerable) y luego se lo sumamos a un número grande, todo funcionará correctamente. Pero si partimos del número grande y le vamos sumando uno por uno los números pequeños, en cada paso el número pequeño será considerado despreciable y llegaremos a un resultado erróneo.

- **División por cantidades pequeñas**
  - Un error mínimo en el dividendo se traduce en uno mucho mayor en el resultado, de modo que la falta de precisión podría ocasionar un error por desbordamiento o pérdida de cifras significativas.
  - Esto se da porque los números de punto flotante están más concentrados cerca del cero entonces al dividir por un número más grande es más probable conseguir una mejor aproximación.
- Estas operaciones "delicadas", la noción que el orden de las operaciones influye en la precisión y las consideraciones que hay que hacer para operar dentro del formato de punto flotante soportado por la máquina (por ejemplo, evaluar en todo tiempo si se va a producir un desbordamiento o subdesbordamiento), hacen que implementar algoritmos sea una actividad no trivial, que hay que dejar en manos de expertos.
- A la hora de escribir programas para aplicar métodos numéricos, en general saltearemos esa parte. Los programas que escribiremos funcionarán bien dentro del contexto de este curso y nos van a servir para entender el funcionamiento de cada método.
- Sin embargo, para otros contextos será mejor si hacemos uso de software desarrollado por especialistas.
- A modo ilustrativo, en la página 29 del libro de Burden se puede ver un algoritmo "casero" para calcular una distancia euclidea (algo así podemos llegar a implementar nosotros), mientras que en las páginas 30 y 31 se presenta un algoritmo más "profesional", que resuelve el mismo problema teniendo más cuidados.

1.6. Estabilidad de los algoritmos

- Hemos visto que la aritmética con dígitos finitos puede introducir errores. Si un algoritmo propone realizar cálculos sucesivos, estos errores pueden empezar a acumularse.

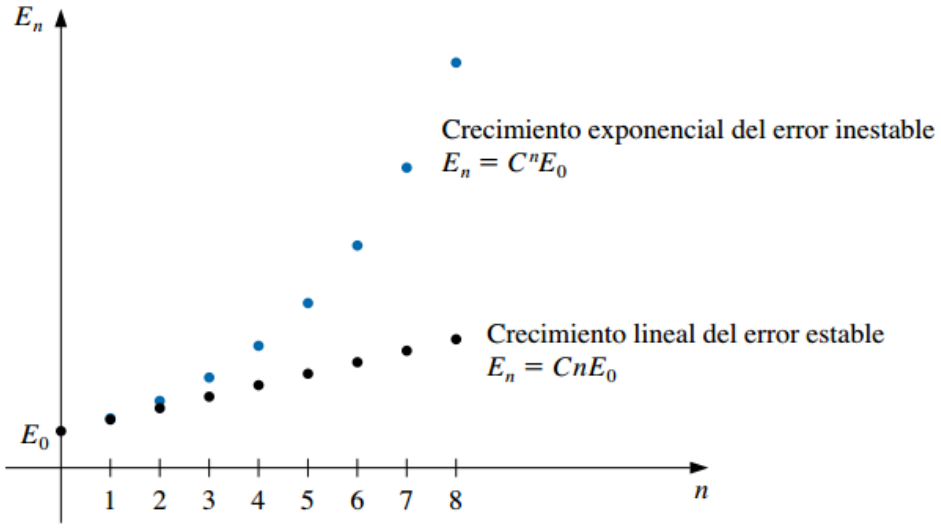
Definición: el **error propagado** es el error que se tiene al final de una cadena de operaciones sucesivas por la existencia de diferentes errores en los pasos intermedios.

- Por ejemplo, si tenemos dos valores exactos p y q con valores aproximados p^* y q^* cuyos errores reales son E_p y E_q de modo que $p^* = p - E_p$ y $q^* = q - E_q$, al realizar la suma entre los valores aproximados encontramos que el error propagado es $-E_p - E_q$:

$$p^* + q^* = (p - E_p) + (q - E_q) = (p + q) + (-E_p - E_q)$$

- Si bien es normal que en una cadena los errores iniciales se propaguen, es deseable que un error pequeño en el comienzo produzca errores pequeños en el resultado final.
- Un algoritmo con esta cualidad se llama **estable** (el error se puede acotar). En caso contrario se dice **inestable**.
- Supongamos que $E_0 > 0$ representa un error inicial y que E_n representa la magnitud del error después de n operaciones:

  - Si $E_n \approx C \times n \times E_0$, para alguna constante C el crecimiento del error es **lineal** (el algoritmo es **estable**)
  - Si $E_n \approx C^n \times E_0$, para alguna constante $C > 1$ el crecimiento del error es **exponencial** y no se puede acotar (el algoritmo es **inestable**).
- Normalmente el crecimiento lineal del error es inevitable y cuando C y E_0 son pequeñas, en general, los resultados son aceptables.
- El crecimiento exponencial del error debería evitarse porque el término C^n puede volverse grande incluso para valores relativamente pequeños de n, conduciendo a imprecisiones inaceptables.



2 Resolución de ecuaciones en una variable

2.1. Introducción

- En esta unidad consideraremos uno de los problemas más básicos de la aproximación numérica: **el problema de la búsqueda de la raíz**, es decir, encontrar una **raíz** o **solución** para una ecuación de la forma $f(x) = 0$, para una función f dada.
- Una raíz de esta ecuación también recibe el nombre de **cero de la función** f.
- Generalmente se clasifica a las ecuaciones como **lineales** o **no lineales**
  - Una **ecuación lineal** es una igualdad que involucra una o más variables elevadas a la primera potencia y no contiene productos entre las variables (involucra solamente sumas y restas de las variables). Por ejemplo: $3x + 2 = 8$.

    Para este tipo de ecuaciones es posible hallar analíticamente una expresión para su solución, aunque esto puede resultar en un proceso complejo.
  - En una **ecuación no lineal** las incógnitas están elevadas a potencias distintas de 1, aparecen en denominadores o exponentes o están afectadas por funciones no lineales (como el logaritmo o las trigonométricas).
- A las ecuaciones no lineales se las suele clasificar como:
  - **Ecuaciones algebraicas**: involucran un polinomio igualado a cero:

    $$P_n(x) = a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n = 0$$

    donde $a_0 \neq 0, n \in \mathbb{N}$ y $a_0, ..., a_n$ son constantes.
    Por ejemplo: $x^3 - x^2 + 5x - 8 = 2x^5$.
    Sabemos que si, por ejemplo, $n = 2$, la solución de $ax^2 + bx + c = 0$ está dada por la resolvente:

    $$x_{1,2} = \frac{b \pm \sqrt{b^2 - 4ac}}{2a}$$

    Sin embargo, la solución analítica para este tipo de ecuaciones existe sólo para $n \leq 4$.
  - **Ecuaciones trascendentes**: incluyen a los otros tipos de ecuaciones no lineales, como por ejemplo:

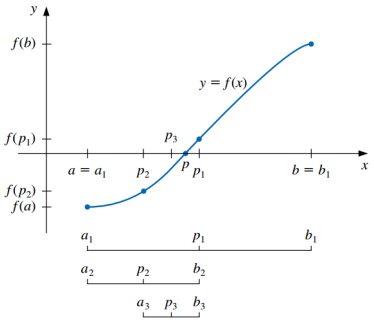
$$x^3 - ln(x) + \frac{3}{x} = 2$$
$$tg(x + 45) = 1 + sen(2x)$$
$$xe^x = 1$$
$$5^x = 9^{x+1}3^x$$

En general, tampoco es posible hallar de manera analítica una solución exacta para estas ecuaciones.

- Estudiaremos distintos métodos para encontrar las soluciones aproximadas a ecuaciones de una variable, ya sean estas lineales o no lineales.
- Todos los métodos que desarrollaremos tienen en común el empleo de una técnica fundamental para el análisis numérico: la **iteración**.
- Los **métodos iterativos** repiten un proceso hasta obtener un resultado para el problema.
- Aplicados a la búsqueda de raíces, en general estos métodos requieren de dos pasos generales:
  1. Determinar un valor aproximado de la raíz que se busca.
  2. Mejorar la solución hasta lograr un grado de precisión preestablecido.

2.2. El método de la bisección o búsqueda binaria

- Se basa en el **teorema de Bolzano**[1], que dice que si f es una función continua definida dentro del intervalo $[a, b]$ con $f(a)$ y $f(b)$ de signos opuestos, entonces existe un número p en (a, b) con $f(p) = 0$ (es decir, p es la solución de la ecuación $f(x) = 0$).
- El método realiza repetidamente una reducción a la mitad (o *bisección*) de subintervalos de $[a, b]$, localizando en cada paso la mitad que contiene a p:



- Para comenzar, sea $a_1 = a$, $b_1 = b$ y $p_1 = \frac{a_1+b_1}{2}$ el punto medio de $[a, b]$.
- Si $f(p_1) = 0$, entonces $p = p_1$ y terminamos (ya encontramos la raíz).

[1] El teorema de Bolzano es un caso particular del **teorema de los valores intermedios**.

- Si $f(p_1) \neq 0$:
  - Si $f(a_1)$ y $f(p_1)$ tienen el mismo signo, $p \in (p_1, b_1)$. Se define el nuevo subintervalo como $a_2 = p_1$ y $b_2 = b_1$.
  - Si $f(a_1)$ y $f(p_1)$ tienen signos opuestos, $p \in (a_1, p_1)$. Se define el nuevo subintervalo como $a_2 = a_1$ y $b_2 = p_1$.
- Se vuelve a aplicar el proceso al intervalo $[a_2, b_2]$ y así sucesivamente hasta verificar algún criterio de parada.
- Por ejemplo, podemos seleccionar una tolerancia $\epsilon > 0$ y detener el proceso siguiendo alguna de estas opciones:

  a. Cuando la semiamplitud del intervalo sea muy pequeña:

  $$\frac{b-a}{2} < \epsilon$$

  b. Cuando el valor de la función evaluado en $f(p_n)$ sea muy pequeño (esto implica que p_n está próximo a la raíz):

  $$|f(p_n)| < \epsilon$$

  c. Cuando la diferencia absoluta o relativa entre dos aproximaciones sucesivas sea muy pequeña:

  $$|p_n - p_{n-1}| < \epsilon$$

  $$\frac{|p_n - p_{n-1}|}{|p_n|} < \epsilon, \quad p_N \neq 0$$

- Estas última serán empleadas en muchos métodos iterativos que estudiaremos, optando generalmente por la que se basa en la diferencia relativa.
- En los métodos iterativos es importante establecer un límite superior sobre el número de iteraciones, para eliminar la posibilidad de entrar en un ciclo infinito (puede ocurrir si la sucesión diverge o si el programa está codificado incorrectamente). El método de la bisección no diverge pero aún así es recomendable establecer esta cota superior para la cantidad de iteraciones.
- **Desventajas**:
  - Convergencia lenta (n puede volverse bastante grande antes de que $|p - p_n|$ sea suficientemente pequeño).
  - Se podría descartar inadvertidamente una buena aproximación intermedia.
- **Ventajas**:
  - Conceptualmente claro.
  - Siempre converge a una solución.

- Por las características anteriores, con frecuencia se utiliza este método como punto de partida para otros métodos más eficientes.

2.3. El método del punto fijo o de las aproximaciones sucesivas

2.3.1. Punto fijo

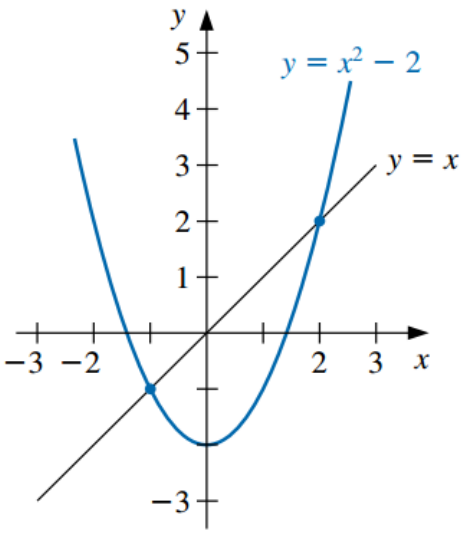
- Un *punto fijo* para una función es un número en el que el valor de la función no cambia cuando se aplica la función.

Definición: el número p es un **punto fijo** para una función dada g si $g(p) = p$.

- Ejemplos:
  - $g(x) = x^2 - 3x + 4$: 2 es un punto fijo de g porque $g(2) = 2$.
  - $g(x) = x^2$: 0 y 1 son puntos fijos de g porque $g(0) = 0$ y $g(1) = 1$.
- El problema de encontrar la raíz p de una ecuación $f(x) = 0$ puede ser planteado de forma equivalente como la búsqueda del punto fijo de alguna función $g(x)$.
- Antes de ver cómo es eso, tenemos que saber cuándo una función tiene un punto fijo y cómo aproximarlo.

2.3.1.1. Interpretación gráfica

- Dado que un punto fijo es el valor de x que satisface $x = g(x)$, un punto fijo para g ocurre precisamente cuando la gráfica de $y = g(x)$ interseca la gráfica de $y = x$ (recta identidad).
- Por ejemplo, vamos a encontrar los puntos fijos de la función $g(x) = x^2 - 2$. Si graficamos esta curva junto con la recta identidad, encontraremos los puntos fijos de g allí donde ambas se cruzan:



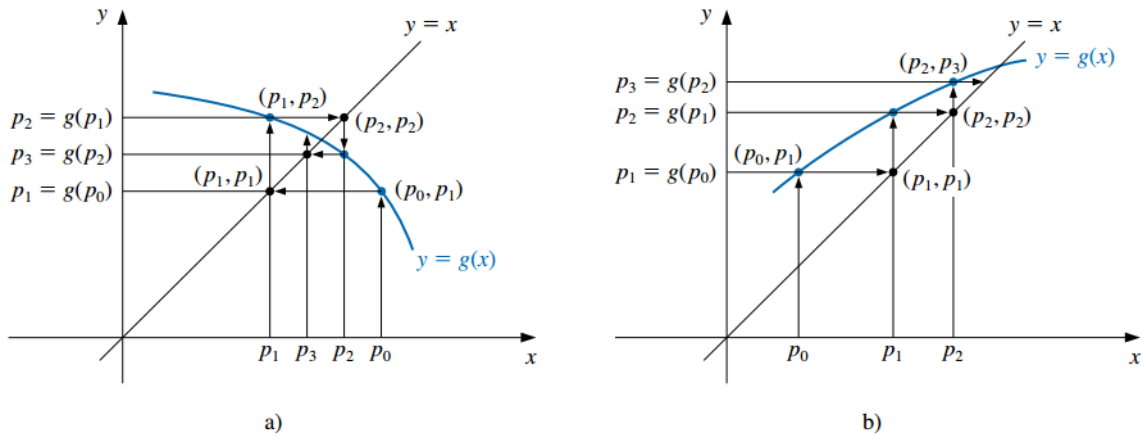
- En la figura podemos observar que los puntos fijos son -1 y 2. De hecho: $g(-1) = 1-2 = -1$ y $g(2) = 4-2 = 2$.

2.3.1.2. Cómo encontrar un punto fijo

- Para aproximar el punto fijo de una función g, elegimos una aproximación inicial p_0 y generamos la sucesión $\{p_n\}_{n=0}^{\infty}$ al permitir $p_n = g(p_{n-1})$ para cada $n \geq 1$:

$$
\begin{gathered}
p_0 \\
p_1 = g(p_0) \\
p_2 = g(p_1) \\
\vdots \\
p_n = g(p_{n-1}) \\
\vdots
\end{gathered}
$$

- Si g es continua y la sucesión converge a un número p, entonces éste es el punto fijo de g. Demostración:
  - La sucesión converge a $p \implies p = \lim_{n\to\infty} p_n$.
  - Por otro lado, $\lim_{n\to\infty} p_n = \lim_{n\to\infty} g(p_{n-1}) = g(\lim_{n\to\infty} p_{n-1}) = g(p)$.
  - De los dos ítems anteriores, resulta que $p = g(p)$, con lo cual p es un punto fijo de g.
- Esta técnica se conoce como **iteración de punto fijo** o **iteración funcional**.
- Ejemplos:



2.3.1.3. Teorema de punto fijo

- No todas las funciones tienen un punto fijo y aunque lo tengan no siempre la sucesión anterior nos conduce al mismo.
- El siguiente teorema proporciona condiciones suficientes para garantizar la existencia y unicidad de un punto fijo y para que la sucesión converja al mismo.

Teorema de punto fijo:

I. Si g es continua en $[a, b]$ y $g(x) \in [a, b]$ para todo $x \in [a, b]$, entonces g tiene por lo menos un punto fjo en $[a, b]$.

II. Si, además, $g'(x)$ existe en (a, b) y existe una constante $0 < k < 1$ con

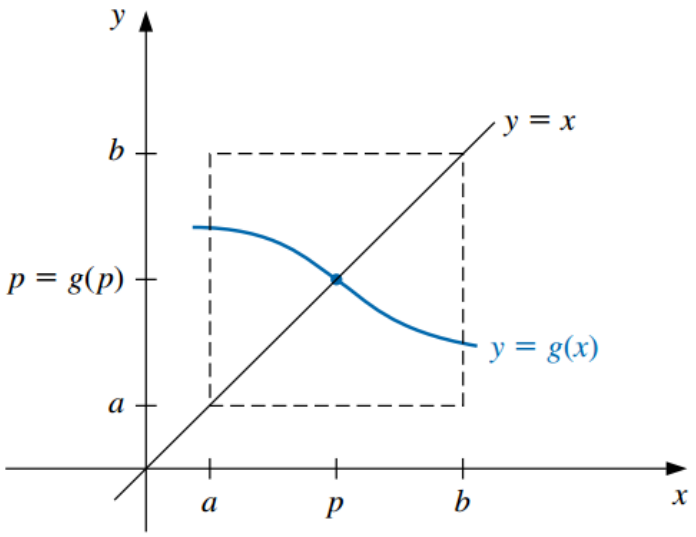
$$|g'(x)| \leq k \quad \forall x \in (a, b),$$

entonces existe exactamente un punto fijo p en $[a, b]$ y para cualquier número p_0 en $[a, b]$, la sucesión definida por:

$$p_n = g(p_{n-1}), \qquad n \geq 1$$

converge al único punto p en $[a, b]$.

- La siguiente imagen ejemplifica la primera condición establecida por el teorema:



- Estas condiciones son suficientes pero no necesarias (la función puede tener un único punto fijo aunque no se cumplan).

2.3.2. Uso de la iteración de punto fijo para resolver ecuaciones

- Sea la ecuación a resolver:

$$f(x) = 0$$

- Siempre es posible reexpresarla en la forma:

$$x = g(x)$$

  con alguna función g.
- Esto se logra despejando alguna x o, por ejemplo, sumando x a cada miembro de la ecuación:

$$\begin{aligned} 0 &= f(x) \\ x + 0 &= x + f(x) \\ x &= \underbrace{x + f(x)}_{g(x)} \end{aligned}$$

- Llamemos con p a la solución de la ecuación, es decir, al valor que satisface $f(x) = 0$.
- Si p satisface $f(x) = 0$, entonces también satisface $x = g(x)$ (puesto que es la misma ecuación escrita de otra forma).
- Entonces, la raíz buscada es el punto fijo de g.

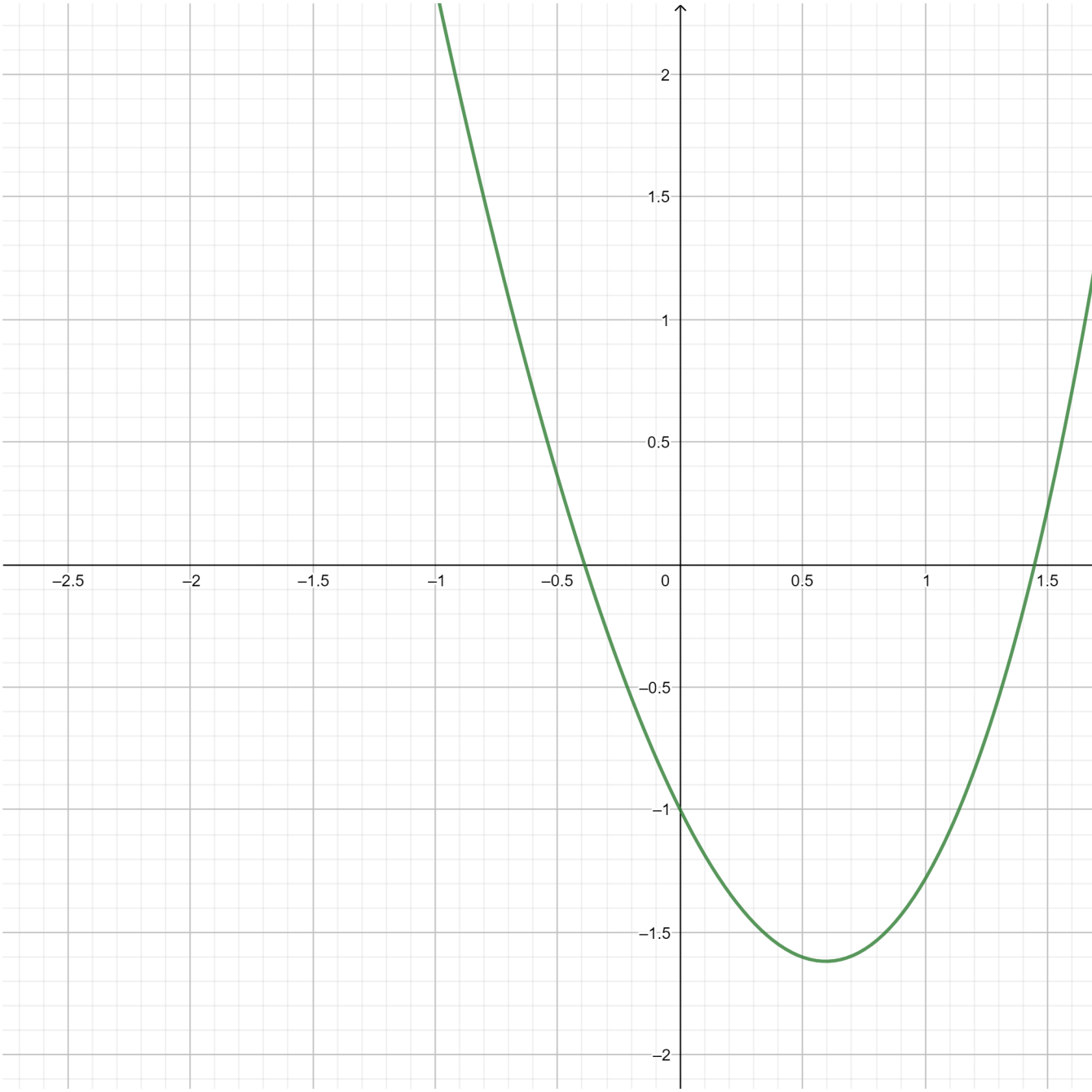
- Así, el **método de iteración de punto fijo o de aproximaciones sucesivas** para resolver $f(x) = 0$ consiste en:
  1. Expresar la ecuación en la forma $x = g(x)$.
  2. Elegir un valor inicial adecuado p_0.
  3. Realizar el siguiente cálculo iterativo:

$$\begin{gathered} p_0 \\ p_1 = g(p_0) \\ p_2 = g(p_1) \\ \vdots \\ p_n = g(p_{n-1}) \\ \vdots \end{gathered}$$

- Si a medida que n crece los p_n se aproximan a un valor fijo, se dice que el método converge y la iteración se detiene cuando la diferencia entre dos valores consecutivos p_{n-1} y p_n sea tan pequeña como se desee, a juzgar por los criterios de parada mencionados anteriormente.
- El valor p_n será una raíz aproximada de $f(x)$.

2.3.3. Ejemplo

- Hallar las raíces de la ecuación no lineal: $f(x) = x^2 - 3x + e^x - 2 = 0$
- Graficamos y vemos que las raíces están cercanas a -0.4 y 1.4 (podés hacer estos gráficos rápidamente con Geogebra.



Paso 1: postular $g(x)$

- Reescribimos $f(x) = 0$ como $x = g(x)$
- Por ejemplo, despejando la x del segundo término:

$$f(x) = x^2 - 3x + e^x - 2 = 0$$

$$\implies x = \underbrace{\frac{x^2 + e^x - 2}{3}}_{g(x)}$$

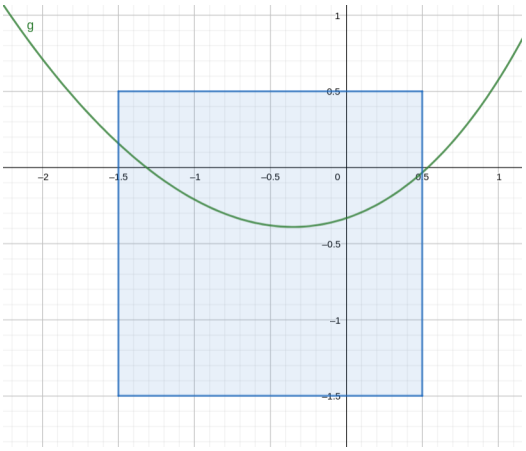
$$\implies g(x) = \frac{x^2 + e^x - 2}{3}$$

Paso 2: verificar si $g(x)$ cumple las condiciones del teorema

- Vamos a concentrarnos en la raíz negativa, para ver si las condiciones del teorema se verifican en una vecindad de la misma.
- Necesitamos calcular la derivada de g:

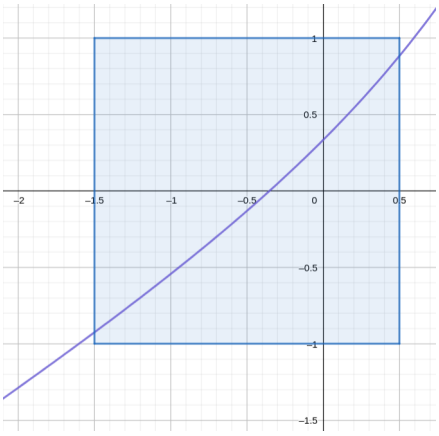
$$g'(x) = \frac{1}{3}(2x + e^x)$$

- La forma más fácil de hacer la verificación es usando una gráfica. Hay que tomar un intervalo $[a, b]$ que contenga a la raíz y graficar g y g' para poder observar el cumplimiento o no de las condiciones.
- Tomemos arbitrariamente el intervalo $[-1.5, 0.5]$. En la siguiente figura podemos ver que g es continua allí y que $g(x) \in [a, b]$ para todo $x \in [a, b]$ (la curva "sale por los costados" del cuadrado delimitado por el intervalo de interés).



- La siguiente figura muestra la derivada, confirmando que está acotada por 1 en valor absoluto:

$$g'(x) = \frac{1}{3}(2x + e^x)$$



- Lo anterior implica que hay una raíz dentro del intervalo $[-1.5, 0.5]$ y que empezando la sucesión con cualquier valor dentro del mismo vamos a llegar a la misma.
- Otra forma es demostrar analíticamente, por ejemplo, que la derivada está acotada por el valor 1 en valor absoluto en intervalo analizado. Como esto puede ser "complejo", en la práctica a veces miramos sencillamente que tanto $|g'(a)|$ como $|g'(b)|$ sean menores que 1 (pero hay que tener cuidado, que se cumpla en los extremos de los intervalos no quiere decir que se cumpla en todo el intervalo).
- Si no se cumplen las condiciones, podemos probar igualmente si el método converge (aunque no hay garantías de eso) o probar con otra expresión para $g(x)$ que sí cumpla con las condiciones.

Paso 3: elegir un punto inicial y realizar las iteraciones

- Imaginemos que en búsqueda de la raíz negativa estamos considerando el intervalo $[-1.5, 0.5]$ que como sabemos verifica las condiciones del teorema.
- Debemos tomar cualquier punto p_0 dentro de este intervalo para iniciar el proceso iterativo.
- La fórmula de recurrencia es:

$$p_n = g(p_{n-1}) = \frac{p_{n-1}^2 + e^{p_{n-1}} - 2}{3}, \qquad n = 1, 2, ...$$

que en este caso imlica:

$$p_n = \frac{p_{n-1}^2 + e^{p_{n-1}} - 2}{3}, \qquad n = 1, 2, ...$$

- Tomando $p_0 = -1.5$, el proceso converge al valor -0.390271 que consideraremos como la aproximación para la raíz buscada.

Iteracion	x
0	-1.5000000
1	0.1577101
2	-0.2681003
3	-0.3877637
4	-0.3903555
5	-0.3902688
6	-0.3902718
7	-0.3902717

- Verificar estos resultados (pueden hacerlo rápidamente en una planilla de Excel).
- Para saber cuándo detenernos, podemos fijar una toleracia $\epsilon = 1E-6$ (por ejemplo) y parar el proceso cuando la diferencia relativa entre dos aproximaciones sucesivas sea menor:

$$\frac{|p_N - p_{N-1}|}{|p_N|} < \epsilon = 1E-6$$

::: {.cell} ::: {.cell-output-display}

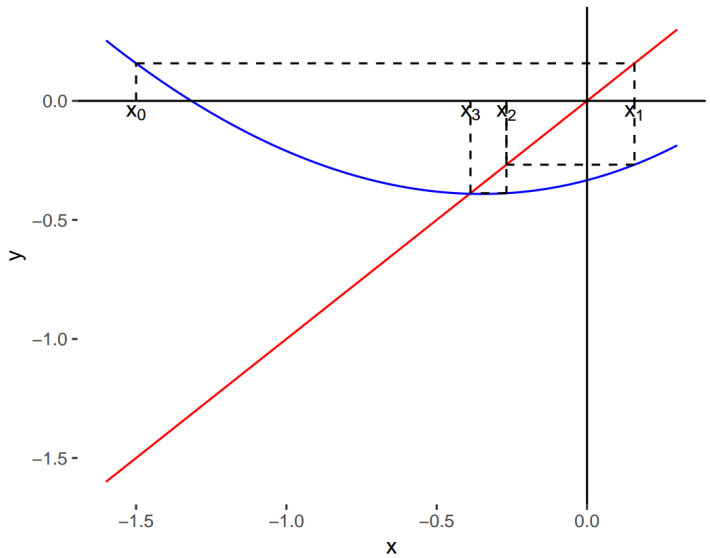
Iteracion	x	Error
0	-1.5000000	NA
1	0.1577101	1.105140e+00
2	-0.2681003	2.699957e+00
3	-0.3877637	4.463383e-01
4	-0.3903555	6.684034e-03
5	-0.3902688	2.222629e-04
6	-0.3902718	7.690434e-06
7	-0.3902717	2.657452e-07

::: :::

Paso 4: representar gráficamente

- Como hemos mencionado, el punto fijo de g se encuentra en el lugar donde la recta identidad interseca a g.
- Una gráfica de ambas nos permite visualizar el proceso iterativo de forma gráfica (la curva azul es $g(x)$ y la roja es la recta identidad):

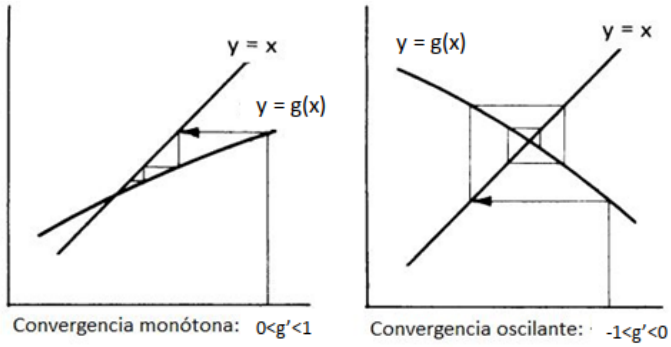
```
knitr::include_graphics("Plots/Un2/puntofijo7.png")
```



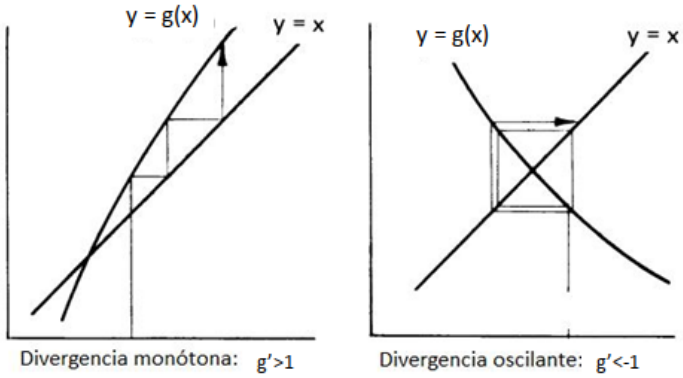
2.3.4. Casos convergentes y divergentes

- Las siguientes figuras presentan algunos ejemplos de convergencia y divergencia del proceso:

```
knitr::include_graphics("Plots/Un2/puntofijo8.png")
```




```
knitr::include_graphics("Plots/Un2/puntofijo9.png")
```



2.4. El método de Newton-Raphson

- El método de Newton (o de Newton-Raphson) es uno de los métodos numéricos más poderosos y reconocidos para resolver un problema de encontrar la raíz.
- Este método propone tomar una aproximación inicial p_0 para la raíz de la ecuación $f(x) = 0$ y generar la sucesión $\{p_n\}_{n=0}^{\infty}$ mediante:

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})} \qquad n \geq 1$$

- Si se cumplen ciertas condiciones generales que estudiaremos más adelante, esta sucesión converge al verdadero valor buscado, la raíz p.

- Para detener las iteraciones se emplea alguno de los criterios de parada mencionados cuando vimos el método de la bisección.
- Antes de ver cuáles son esas condiciones, vamos a ver de dónde surge esta fórmula de recurrencia y cómo la podemos interpretar geométricamente.

2.4.1. Deducción de la fórmula de recurrencia

- Supongamos que f es continua en un intervalo $[a, b]$ y tiene derivadas primera y segunda continuas en el mismo intervalo.
- Tomemos $p_0 \in [a, b]$ como un valor que se aproxima para p y consideremos el polinomio de Taylor de grado 1 para aproximar $f(x)$ alrededor de p_0:

$$f(x) = f(p_0) + (x - p_0)f'(p_0) + \underbrace{\frac{(x - p_0)^2}{2!}f''(\xi)}_{\text{resto, } \xi \text{ real entre } x \text{ y } p_0}$$

- Ahora, evaluemos el polinomio de Taylor en el valor verdadero p:

$$\text{Por ser } p \text{ la raíz de } f: \quad f(p) = 0$$

$$\text{Por el desarrollo de Taylor:} \quad f(p) = f(p_0) + (p - p_0)f'(p_0) + \underbrace{\frac{(p - p_0)^2}{2!}f''(\xi)}_{\text{resto, } \xi \text{ real entre } p \text{ y } p_0}$$

$$\text{Entonces:} \quad 0 = f(p_0) + (p - p_0)f'(p_0) + \frac{(p - p_0)^2}{2!}f''(\xi)$$

- Si p_0 es una aproximación adecuada, $|p - p_0|$ debe ser pequeño y entonces el término relacionado a $(p - p_0)^2$, mucho más pequeño y puede ser descartado, de modo que:

$$0 \approx f(p_0) + (p - p_0)f'(p_0)$$

- Al despejar p tenemos:

$$p \approx p_0 - \frac{f(p_0)}{f'(p_0)} \equiv p_1$$

- Llamamos al valor anterior p_1 y repetimos el procedimiento planteando el desarrollo en serie de Taylor de f alrededor de p_1, encontrando que:

$$p \approx p_1 - \frac{f(p_1)}{f'(p_1)} \equiv p_2$$

- Si seguimos repitiendo esto, damos lugar a una sucesión que debe acercarnos cada vez más al verdadero valor de p:

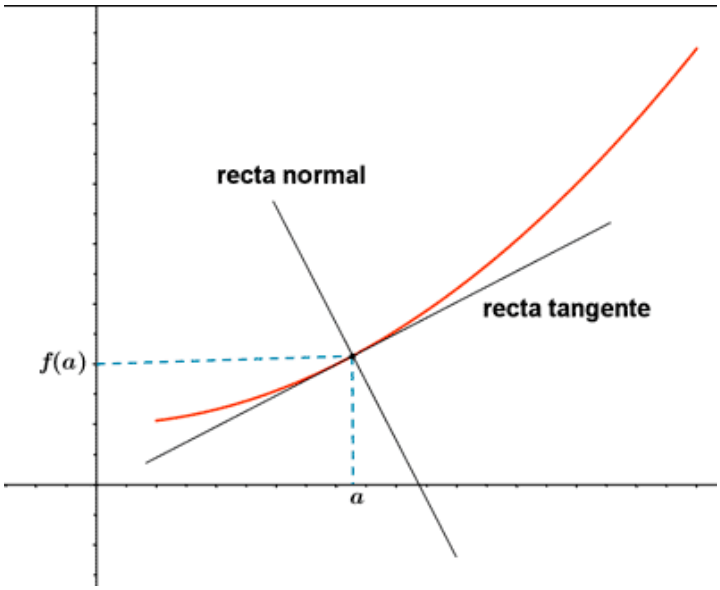
$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})} \qquad n \geq 1$$

2.4.2. Interpretación geométrica

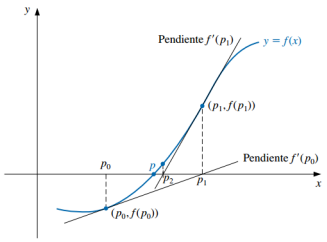
- Recordemos la definición de recta tangente:

Definición: una recta se dice que es **tangente** a una función f en un punto a cuando pasa por ese punto y su pendiente es $f'(a)$. La ecuación de la recta tangente a la gráfica de la función en el punto $(a, f(a))$ es:

$$y = f(a) + f'(a)(x - a)$$



- La fórmula de recurrencia presentada anteriormente equivale a encontrar el próximo valor p_n como el punto en el que el eje de las abscisas interseca a la recta tangente a la gráfica de f en $(p_{n-1}, f(p_{n-1}))$.
- Es decir, al empezar con la aproximación inicial p_0, la aproximación p_1 es la intersección con el eje x de la recta tangente a la gráfica de f en $(p_0, f(p_0))$.
- La aproximación p_2 es la intersección con el eje x de la recta tangente a la gráfica de f en $(p_1, f(p_1)$ y así sucesivamente:



- ¿De dónde sale que la fórmula de recurrencia equivale a esto de avanzar en la sucesión de acuerdo a las rectas tangentes? Hay que prestarle atención a las pendientes.
- Por ejemplo, por definición de recta tangente, sabemos que la pendiente de la tangente a f en p_0 es igual a:

$$m = f'(p_0)$$

- Pero también sabemos que la pendiente se puede definir como el siguiente cociente, donde (x_0, y_0) y (x_1, y_1) son dos puntos cualesquiera pertenecientes a la recta:

$$m = \frac{y_1 - y_0}{x_1 - x_0}$$

- Podemos tomar los puntos $(p_0, f(p_0))$ y $(p_1, 0)$ (el punto donde la tangente interseca al eje x) y encontrar una expresión para la pendiente de la tangente:

$$m = \frac{y_1 - y_0}{x_1 - x_0} = \frac{0 - f(p_0)}{p_1 - p_0} = -\frac{f(p_0)}{p_1 - p_0}$$

- Igualando las dos expresiones equivalentes vistas para la pendiente m:

$$f'(p_0) = -\frac{f(p_0)}{p_1 - p_0} \implies p_1 = p_0 - \frac{f(p_0)}{f'(p_0)}$$

- Si repetimos este pensamiento con la recta tangente a f en el punto p_1, vamos a encontrar que:

$$p_2 = p_1 - \frac{f(p_1)}{f'(p_1)}$$

- Esto constituye una derivación geométrica del método de Newton-Raphson desde el punto de vista de las rectas tangentes a f en los puntos p_n.

2.4.3. Convergencia

- La derivación del método de Newton por medio de la serie de Taylor señala la importancia de una aproximación inicial precisa.
- La suposición crucial es que el término relacionado con $(p - p_0)^2$ es, en comparación con $|p - p_0|$, tan pequeño que se puede eliminar.

- Claramente esto será falso a menos que p_0 sea una buena aproximación para p. Si no lo es, no existen razones para sospechar que el método convergerá en la raíz (aunque en algunos casos incluso malas aproximaciones iniciales producen convergencia).
- El siguiente teorema establece cuáles son las condiciones para el método converja, que básicamente se resumen en el hecho de que p_0 tiene que estar suficientemente cerca de p.

Teorema: convergencia del método de Newton-Raphson:
Sea f continua en un intervalo $[a, b]$ con derivadas primera y segunda continuas en el mismo intervalo. Si $p \in (a, b)$ es tal que $f(p) = 0$ y $f'(p) \neq 0$, entonces existe $\delta > 0$ tal que el método de Newton-Raphson genera una sucesión $\{p_n\}_{n=0}^{\infty}$ que converge a p para cualquier aproximación inicial $p_0 \in [p - \delta, p + \delta]$.

- El teorema se demuestra considerando que la sucesión propuesta por el método es una iteración de punto fijo. Repetimos la fórmula de recurrencia:

$$p_n = \underbrace{p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}}_{g(p_{n-1})} \qquad n \geq 1$$

- Vemos que se trata de una iteración de punto fijo $p_n = g(p_{n-1})$, en la que la función g es:

$$g(x) = x - \frac{f(x)}{f'(x)}$$

- Por lo tanto, el método converge cuando se cumplen las condiciones del Teorema del punto fijo:
  I. g es continua en $[a, b]$ y $g(x) \in [a, b]$ para todo $x \in [a, b]$.
  II. $g'(x)$ existe en (a, b) y existe una constante $0 < k < 1$ con tal que $|g'(x)| \leq k < 1$.
- Analicemos solamente la condición acerca de que la derivada de g tiene que estar acotada:

$$|g'(x)| \leq k < 1$$

- Tomamos la derivada:

$$g'(x) = 1 - \frac{[f'(x)]^2 - f(x)f''(x)}{[f'(x)]^2} = 1 - 1 + \frac{f(x)f''(x)}{[f'(x)]^2} = \frac{f(x)f''(x)}{[f'(x)]^2}$$

- Es decir, el método convergerá si:

$$|g'(x)| = \frac{|f(x)f''(x)|}{[f'(x)]^2} \leq k < 1$$

- Por hipótesis, sabemos que $f(p) = 0$; luego $g'(p) = 0$. Como $g'(x)$ es continua y $g'(p) = 0$, siempre podemos encontrar un $\delta > 0$ tal que $|g'(x)| < 1$ se cumpla en el intervalo $[p - \delta, p + \delta]$.

- Por consiguiente, que p_0 se encuentre dentro de $[p-\delta, p+\delta]$ es una condición suficiente para que la sucesión $\{x_n\}_{n=0}^{\infty}$ converja a la única raíz de $f(x)=0$ en dicho intervalo.
- La condición anterior tiene las siguientes implicancias. Para facilitar la convergencia:
  - p_0 tiene que estar suficientemente cerca de p.
  - $f''(x)$ no debe ser muy grande y $f'(x)$ no debe ser muy chica en ese intervalo.
- Si bien el teorema sirve para asegurar la convergencia, no dice cómo determinar δ, así que en la práctica se selecciona una aproximación inicial y se generan aproximaciones sucesivas con el método de Newton. Puede que éstos converjan rápidamente a la raíz o será claro que la convergencia es poco probable.
- *Observación*: el método no puede continuar si $f'(p_{n-1})=0$ para alguna n.

2.4.4. Ejemplo

- Retomemos el ejemplo anterior en cual buscábamos las raíces de la ecuación no lineal: $f(x)=x^2-3x+e^x-2=0$
- Mediante el método del punto fijo reformulamos la ecuación anterior como $x=g(x)$ con:

$$g(x)=\frac{x^2+e^x-2}{3}$$

- De esa forma pudimos hallar la raíz negativa.
- Sin embargo, el método no sirve para hallar la raíz positiva (verificar que no se cumplen las condiciones del teorema en vecindades de la raíz).
- Si bien podríamos probar con otra expresión para $g(x)$, ¿podrá el método de Newton-Raphson sernos útil en este caso?
- Verificar.

2.5. Variantes del método de Newton-Raphson

2.5.1. Método de la secante

- El método de Newton es una técnica en extremo poderosa, pero tiene una debilidad importante: la necesidad de conocer el valor de la derivada de f en cada aproximación.
- $f'(x)$ puede ser más difícil y necesitar más operaciones aritméticas para ser calculada que $f(x)$.
- Para evitar el problema de la evaluación de la derivada, el método de la secante presenta una ligera variación.
- Por definición de derivada:

$$f'(p_{n-1})=\lim_{x\to p_{n-1}}\frac{f(x)-f(p_{n-1})}{x-p_{n-1}}$$

- Si p_{n-2} está cerca de p_{n-1}:

$$f'(p_{n-1}) \approx \frac{f(p_{n-2}) - f(p_{n-1})}{p_{n-2} - p_{n-1}} = \frac{f(p_{n-1}) - f(p_{n-2})}{p_{n-1} - p_{n-2}}$$

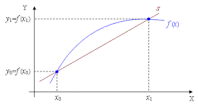
- Usando esta aproximación en la fórmula de Newton obtenemos:

$$p_n = p_{n-1} - \frac{f(p_{n-1})(p_{n-1} - p_{n-2})}{f(p_{n-1}) - f(p_{n-2})} \qquad n \geq 2$$

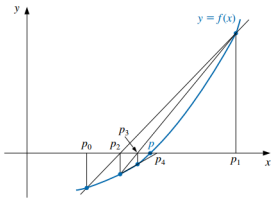
- Notar que se necesitan dos aproximaciones iniciales.
- Este método también goza de una interpretación geométrica que es la que le da su nombre.

Definición: una recta **secante** es una recta que corta a una curva f en dos o más puntos. Conforme estos puntos se acercan y su distancia se reduce a cero, la recta secante pasa a ser la *recta tangente*.
La ecuación de la recta secante a f que pasa por los puntos $(x_1, f(x_1))$ y $(x_2, f(x_2))$ es (fórmula de la recta que pasa por dos puntos):

$$y = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1)$$



- Empezando con dos aproximaciones iniciales p_0 y p_1, la aproximación p_2 es la intersección en x de la recta que une los puntos $(p_0, f(p_0))$ y $(p_1, f(p_1))$.
- La aproximación p_3 es la intersección en x de la recta que une los puntos $(p_1, f(p_1))$ y $(p_2, f(p_2))$ y así sucesivamente.



- *Observación*: sólo se necesita una evaluación de la función por cada paso para el método de la secante después de haber determinado p_2. En contraste, cada paso del método de Newton requiere una evaluación tanto de la función como de su derivada.

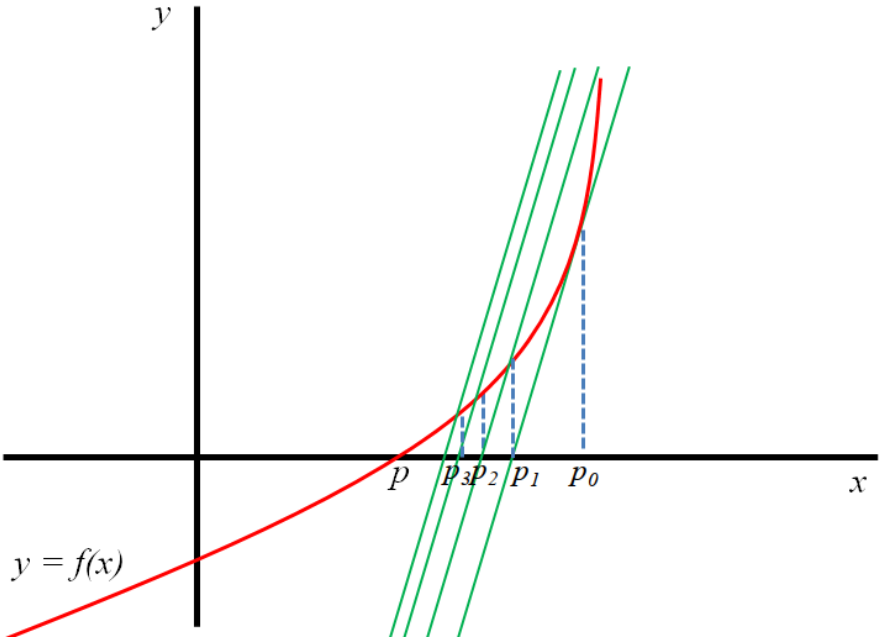
2.5.2. Método de von Mises

- En el método de Newton-Raphson, el denominador $f'(p_{n-1})$ hace que geométricamente se pase de una aproximación a la siguiente por la tangente de la curva $y = f(x)$ en el

punto correspondiente a la aproximación presente p_{n-1}.

- Esto puede producir problemas cuando se esté en puntos alejados de raíces y cerca de puntos donde el valor de $f'(x)$ sea cercano a 0 (tangentes cercanas a la horizontal).
- Para resolver este problema, von Mises sugirió sustituir $f'(p_{n-1})$ en el denominador por $f'(p_0)$.
- Es decir, obtener las aproximaciones de la sucesión por medio de rectas que son todas paralelas a la primera tangente.
- La fórmula de recurrencia resultante es:

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_0)} \qquad n \geq 1$$



- Si la derivada requiere muchos cálculos, este método posee la ventaja de que la misma sólo debe ser evaluada una vez.

2.5.3. Método de Newton-Raphson de 2º Orden

- Otra modificación al método de Newton-Raphson se deriva a partir de la utilización de un término más en el desarrollo por serie de Taylor de la función $f(x)$.
- Dada la existencia de las correspondientes derivadas, la fórmula de recurrencia resultante es:

$$p_n = p_{n-1} + \frac{f(p_{n-1})f'(p_{n-1})}{0.5f(p_{n-1})f''(p_{n-1}) - [f'(p_{n-1})]^2} \qquad n \geq 1$$

- El método de Newton-Raphson de 2º orden llega más rápidamente a la raíz que el de primer orden, pero requiere de más cálculos y la desventaja de especificar también la derivada segunda.
- Ejercicio propuesto: derivar la ecuación de recurrencia de este método de forma análoga a la derivación de la fórmula para el método de Newton-Raphson.

3 Resolución de sistemas de ecuaciones lineales

3.1. Introducción

- **Objetivo de la unidad**: examinar los aspectos numéricos que se presentan al resolver sistemas de ecuaciones lineales de la forma:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

- El anterior es un sistema de n ecuaciones con n incógnitas: decimos que es un sistema de orden $n \times n$, en el cual los coeficientes a_{ij} y los términos independientes b_i son reales fijos.
- Recordemos que los sistemas de ecuaciones lineales se puede representar matricialmente: $\mathbf{Ax} = \mathbf{b}$, de dimensión $n \times n$, $n \times 1$ y $n \times 1$, respectivamente.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

- Llamamos **matriz ampliada** o **aumentada** a:

$$\big[\mathbf{A}, \quad \mathbf{b}\big] = \left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right]$$

3.1.1. Repaso de lo que ya sabemos

- Un sistema de ecuaciones lineales se clasifica en:
  - **Compatible determinado**: tiene una única solución
  - **Compatible indeterminado**: tiene infinitas soluciones

  - **Incompatible**: no existe solución

- Además, las siguientes condiciones son equivalentes:
  - El sistema $\mathbf{Ax} = \mathbf{b}$ tiene solución única (es compatible determinado).
  - La matriz $\mathbf{A}$ es invertible (existe $\mathbf{A^{-1}}$).
  - La matriz $\mathbf{A}$ es no singular ($\det \mathbf{A} \neq 0$).
  - El sistema $\mathbf{Ax} = \mathbf{0}$ tiene como única solución $\mathbf{x} = \mathbf{0}$.
- Dos sistemas de orden $n \times n$ son **equivalentes** si tienen el mismo conjunto de soluciones.
- Existen ciertas transformaciones sobre las ecuaciones de un sistema que no cambian el conjunto de soluciones (producen un sistema equivalente). Si llamamos con E_i a cada una de las ecuaciones del sistema:
  - **Intercambio**. El orden de las ecuaciones puede cambiarse: $E_i \leftrightarrow E_j$
  - **Escalado**. Multiplicar una ecuación por una constante no nula: $\lambda E_i \rightarrow E_i$
  - **Sustitución**: una ecuación puede ser reemplazada por una combinación lineal de las ecuaciones del sistema (*teorema fundamental de la equivalencia*). Por ejemplo: $E_i + \lambda E_j \rightarrow E_i$
- Mediante una secuencia de estas operaciones, un sistema lineal se transforma en uno nuevo más fácil de resolver y con las mismas soluciones.
- Realizar estas transformaciones sobre las ecuaciones es equivalente a realizar las mismas operaciones sobre las filas de la matriz aumentada.

3.1.2. Notación

- En esta sección presentamos la notación que utilizaremos para expresar algoritmos con operaciones matriciales (facilitando su programación).
- Dada una matriz $\mathbf{Z}$ de dimensión $n \times m$, anotamos:
  - $z_{ij} = \mathbf{Z}[i, j]$: elemento en la fila i y columna j de la matriz $\mathbf{Z}$
  - $\mathbf{Z}[i,]$: vector fila de dimensión $1 \times m$ constituido por la i-ésima fila de la matriz $\mathbf{Z}$
  - $\mathbf{Z}[, j]$: vector columna $n \times 1$ constituido por la j-ésima columna de la matriz $\mathbf{Z}$
  - $\mathbf{Z}[i, k : l]$: matriz de dimensión $1 \times (l - k + 1)$ constituida con los elementos $z_{i,k}, z_{i,k+1}, \cdots, z_{i,l}$ de la matriz $\mathbf{Z}$, $l \geq k$.
  - $\mathbf{Z}[c : d, k : l]$: matriz de dimensión $(d-c+1) \times (l-k+1)$ constituida por la submatriz que contiene las filas de $\mathbf{Z}$ desde la c hasta d y las columnas de $\mathbf{Z}$ desde la k hasta la l, $d \geq c$, $l \geq k$.
- Dado un vector $\mathbf{Z}$ de largo n, anotamos:
  - $\mathbf{Z}[i]$: elemento i-ésimo del vector $\mathbf{Z}$
  - $\mathbf{Z}[k : l]$: vector de largo $(l - k + 1)$ constituido con los elementos $z_k, z_{k+1}, \cdots, z_l$ del vector $\mathbf{Z}$, $l \geq k$.

3.1.3. Métodos de resolución de sistemas de ecuaciones

- **Métodos exactos**: permiten obtener la solución del sistema de manera directa.
  - Método de sustitución hacia atrás o hacia adelante
  - Método de Eliminación de Gauss
  - Método de Gauss-Jordan
- **Métodos aproximados**: utilizan algoritmos iterativos que calculan las solución del sistema por aproximaciones sucesivas.
  - Método de Jacobi
  - Método de Gauss-Seidel
- En muchas ocasiones los métodos aproximados permiten obtener un grado de exactitud superior al que se puede obtener empleando los denominados métodos exactos, debido fundamentalmente a los errores de redondeo que se producen en el proceso.

3.2. Métodos exactos

- En esta sección repasaremos los métodos que ya conocen y que han aplicado "a mano" muchas veces para resolver sistemas de ecuaciones, con la particularidad de que pensaremos cómo expresar sus algoritmos para poder programarlos en la computadora.
- Entre los aspectos que tendremos que considerar, se encuentra la necesidad de aplicar estrategias de pivoteo (para evitar un *pivote* igual a cero cuando se resuelve el sistema) y la posibilidad de realizar reordenamientos de la matriz de coeficientes para disminuir los errores de redondeo producidos en los cálculos computacionales.

3.2.1. Sistemas con matriz de coeficientes diagonal

- Si la matriz de coeficientes es de esta forma:

$$\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

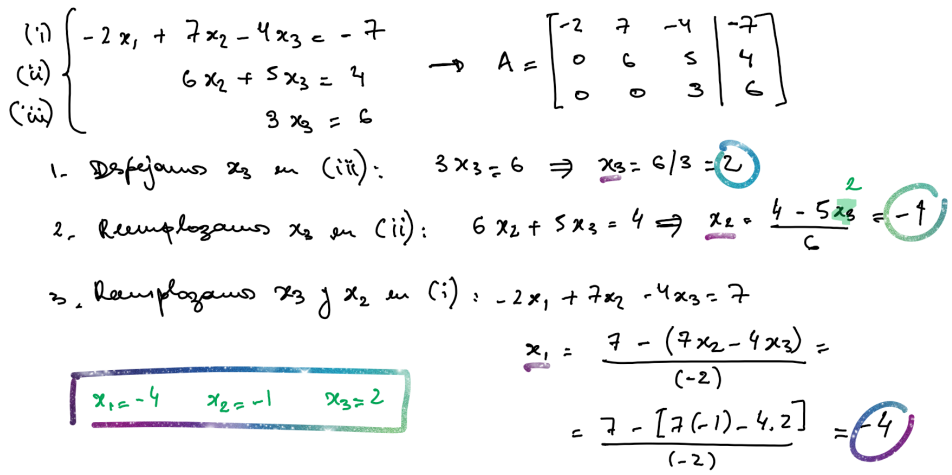
  el sistema se reduce a n ecuaciones simples y la solución sencillamente es:

$$\mathbf{x} = \begin{bmatrix} b_1/a_{11} \\ b_2/a_{22} \\ \vdots \\ b_n/a_{nn} \end{bmatrix}$$

3.2.2. Sistemas con matriz de coeficientes triangular

- Veamos el siguiente ejemplo y su resolución, que seguramente comprendemos sin problema:

```
knitr::include_graphics("Plots/U3/pizarra1.png")
```



- Hemos encontrado el valor de la última incógnita y fuimos haciendo reemplazos en las ecuaciones desde abajo hacia arriba para encontrar las restantes.
- Esto se conoce como **sustitución regresiva o hacia atrás**.
- Para poder formalizar este procedimiento que nos resulta natural y así estar en condiciones para implementarlo computacionalmente, tenemos que encontrar una fórmula que exprese sin ambigüedad cómo hacer todos esos cálculos.
- De manera genera, vamos a considerar que la matriz $\mathbf{A}$ es triangular superior:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & 0 & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

- La solución de x_n es inmediata y a partir de ella se encuentran las restantes en orden inverso $x_{n-1}, \cdots, x_1$, aplicando el algoritmo de **sustitución regresiva**:

$$x_n = \frac{b_n}{a_{nn}} \text{ y } x_k = \frac{b_k - \sum_{j=k+1}^{n} a_{kj} x_j}{a_{kk}} \quad k = n-1, n-2, \cdots, 1$$

- Empleando la notación matricial vista, la suma en el algoritmo puede reescribirse como un producto matricial:

$$\mathbf{x}[n] = \frac{\mathbf{b}[n]}{\mathbf{A}[n,n]} \qquad \mathbf{x}[k] = \frac{\mathbf{b}[k] - \mathbf{A}[k,(k+1):n] \times \mathbf{x}[(k+1):n]}{\mathbf{A}[k,k]} \qquad k = n-1, n-2, \cdots, 1$$

- Iniciando el vector solución con ceros, $\mathbf{x} = [0\ 0 \cdots 0]$, la expresión anterior se simplifica aún más:

$$\mathbf{x}[n] = \frac{\mathbf{b}[n]}{\mathbf{A}[n,n]} \qquad \mathbf{x}[k] = \frac{\mathbf{b}[k] - \mathbf{A}[k,] \times \mathbf{x}}{\mathbf{A}[k,k]} \qquad k = n-1, n-2, \cdots, 1$$

- Esto nos permite escribir un algoritmo para la implementación de este método:

```
knitr::include_graphics("Plots/U3/alg1.png")
```

Algoritmo 1 Sustitución regresiva para matrices triangulares superiores invertibles

Entrada: A: matriz triangular superior invertible nxn; b: matriz nx1
Salida: x: solución del sistema lineal A x = b

```
n ← número de filas de A
x ← vector numérico iniciado con ceros de largo n
x[n] ← b[n] / A[n, n]
para k desde n-1 hasta 1 cada -1 hacer
  x[k] ← (b[k] - A[k, ] * x) / A[k, k]
fin para
devolver x
```

Ejercicio:
Operar "a mano" de forma matricial siguiendo los pasos del algoritmo para corroborar su funcionamiento con el sistema de ecuaciones del ejemplo.

- Si la matriz $\mathbf{A}$ es triangular inferior, la obtención de la solución es análoga al caso anterior y el algoritmo recibe el nombre de **sustitución hacia adelante o progresiva**.

3.2.3. Eliminación gaussiana

- Es un método para resolver un sistema lineal general $\mathbf{Ax} = \mathbf{b}$ de n ecuaciones con n incógnitas.
- El objetivo es construir un sistema equivalente donde la matriz de coeficientes sea triangular superior para obtener las soluciones con el algoritmo de sustitución regresiva.
- El método consiste en ir eliminando incógnitas en las ecuaciones de manera sucesiva, aplicando las operaciones entre ecuaciones que producen sistemas equivalentes (repasadas en la intro).
- Ejemplo: resolver el siguiente sistema

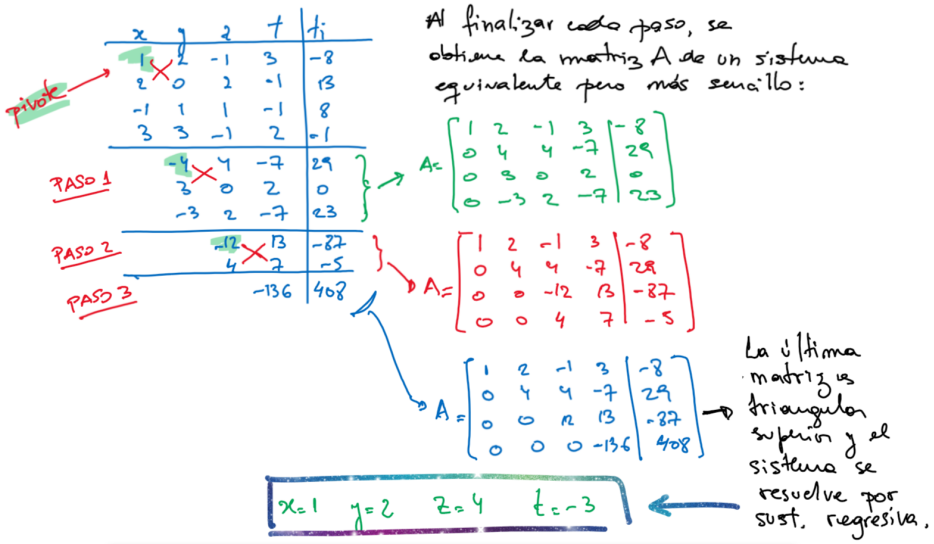
$$\begin{cases} x + 2y - z + 3t = -8 \\ 2x + 2z - t = 13 \\ -x + y + z - t = 8 \\ 3x + 3y - z + 2t = -1 \end{cases} \qquad \mathbf{A} = \begin{bmatrix} 1 & 2 & -1 & 3 \\ 2 & 0 & 2 & -1 \\ -1 & 1 & 1 & -1 \\ 3 & 3 & -1 & 2 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -8 \\ 13 \\ 8 \\ -1 \end{bmatrix}$$

- Matriz aumentada:

$$[\mathbf{A} \quad \mathbf{b}] = \left[\begin{array}{cccc|c} 1 & 2 & -1 & 3 & -8 \\ 2 & 0 & 2 & -1 & 13 \\ -1 & 1 & 1 & -1 & 8 \\ 3 & 3 & -1 & 2 & -1 \end{array}\right]$$

"A mano"

- Primero recordemos cómo resolvemos este tipo de sistemas "a mano". Seguramente nos resulte familiar la aplicación del método de Gauss, representada en la siguiente imagen:



- Como podemos ver, aunque no lo pensemos, al hacer "este por este menos este por este" estamos pasando sucesivamente de un sistema a otro equivalente, con una matriz de coeficientes más sencilla (van apareciendo ceros, es decir, se van elminando incógnitas) hasta llegar a tener una matriz triangular superior, como en el caso anterior.
- Si bien este procedimiento puede ser largo, estamos acostumbrados y no nos presenta ninguna dificultad.
- El desafío surge si tenemos que pensar en una forma de expresarlo formalmente y de hallar fórmulas que puedan ser programadas, de modo que la computadora pueda resolver el sistema por nosotros.
- A continuación vamos a repetir el proceso paso por paso para poder deducir un algoritmo.

Primer paso

- En el primer paso eliminamos la incógnita x en las ecuaciones 2, 3 y 4, buscando que queden ceros en toda la primera columna excepto en el elemento diagonal.
- Observando con detenimiento, esto se logra realizando reemplazos en las ecuaciones de esta forma:

$$E_r - m_{r1} \times E_1 \to E_r \qquad r = 2, 3, 4$$

- Los valores m_{r1} se llaman **multiplicadores** y se definen como:

$$m_{r1} = \frac{a_{r1}}{a_{11}}, \quad r = 2, 3, 4$$

- Esta elección de multiplicadores hace que se generen ceros en la primera columna desde la fila 2 hasta la última.
- En este ejemplo, nos quedan:

$$m_{21} = \frac{a_{21}}{a_{11}} = 2 \qquad m_{31} = \frac{a_{31}}{a_{11}} = -1 \qquad m_{41} = \frac{a_{41}}{a_{11}} = 3$$

- Los reemplazos a realizar entonces son:

$$\begin{aligned} E_2 - 2 \times E_1 &\to E_2 \\ E_3 - (-1) \times E_1 &\to E_3 \\ E_4 - 3 \times E_1 &\to E_4 \end{aligned}$$

- El elemento $a_{11} = 1$ que está en el denominador de todos los multiplicadores se le dice **pivote** y a la fila 1 que se usa en los reemplazos se le dice **fila pivote**.
- El resultado de los reemplazos anteriores es:

$$\begin{array}{r} \text{pivote} \to \\ m_{21} = 2 \\ m_{31} = -1 \\ m_{41} = 3 \end{array} \left[\begin{array}{cccc|c} 1 & 2 & -1 & 3 & -8 \\ 2 & 0 & 2 & -1 & 13 \\ -1 & 1 & 1 & -1 & 8 \\ 3 & 3 & -1 & 2 & -1 \end{array}\right] \Longrightarrow \left[\begin{array}{cccc|c} 1 & 2 & -1 & 3 & -8 \\ 0 & -4 & 4 & -7 & 29 \\ 0 & 3 & 0 & 2 & 0 \\ 0 & -3 & 2 & -7 & 23 \end{array}\right]$$

Segundo paso

- En el segundo paso, eliminamos la incógnita y en las ecuaciones 3 y 4.
- La fila pivote pasa a ser la segunda y el pivote es $a_{22} = -4$.
- Los multiplicadores son $m_{r2} = a_{r2}/a_{22}$, $r = 3, 4$:

$$m_{32} = \frac{a_{32}}{a_{22}} = -3/4 \qquad m_{42} = \frac{a_{42}}{a_{22}} = 3/4$$

dando lugar a los reemplazos:

$$\begin{aligned} E_3 - (-3/4) \times E_2 &\to E_3 \\ E_4 - 3/4 \times E_2 &\to E_4 \end{aligned}$$

- El resultado es:

$$\begin{array}{r} \\ \text{pivote} \to \\ m_{32} = -3/4 \\ m_{42} = 3/4 \end{array} \left[\begin{array}{cccc|c} 1 & 2 & -1 & 3 & -8 \\ 0 & -4 & 4 & -7 & 29 \\ 0 & 3 & 0 & 2 & 0 \\ 0 & -3 & 2 & -7 & 23 \end{array}\right] \Longrightarrow \left[\begin{array}{cccc|c} 1 & 2 & -1 & 3 & -8 \\ 0 & -4 & 4 & -7 & 29 \\ 0 & 0 & 3 & -13/4 & 87/4 \\ 0 & 0 & -1 & -7/4 & 5/4 \end{array}\right]$$

Tercer paso

- Finalmente, eliminamos la incógnita z en la última ecuación.
- La fila pivote es la 3º y el pivote es $a_{33} = 3$.
- El multiplicador es $m_{43} = a_{43}/a_{33} = -1/3$.
- El reemplazo a realizar es:

$$E_4 - (-1/3) \times E_3 \to E_4$$

- El resultado es:

$$\begin{array}{r} \\ \text{pivote} \to \\ m_{43} = -1/3 \end{array} \left[\begin{array}{cccc|c} 1 & 2 & -1 & 3 & -8 \\ 0 & -4 & 4 & -7 & 29 \\ 0 & 0 & 3 & -13/4 & 87/4 \\ 0 & 0 & -1 & -7/4 & 5/4 \end{array}\right] \Longrightarrow \left[\begin{array}{cccc|c} 1 & 2 & -1 & 3 & -8 \\ 0 & -4 & 4 & -7 & 29 \\ 0 & 0 & 3 & -13/4 & 87/4 \\ 0 & 0 & 0 & -17/6 & 17/2 \end{array}\right]$$

- Hemos llegado a un sistema equivalente cuya matriz de coeficientes es triangular superior, en el que aplicamos el algoritmo de sustitución regresiva:

$$\begin{cases} x + 2y - z - 3t = -8 \\ -4y + 4z - 7t = 29 \\ 3z - 13/4t = 87/4 \\ -17/6t = 17/2 \end{cases} \Longrightarrow \begin{cases} x = 1 \\ y = 2 \\ z = 4 \\ t = -3 \end{cases}$$

- Esta matriz triangular es algo diferente a la que obtuvimos cuando hicimos las cuentas con el "cuadrito" de Gauss a mano, pero es equivalente (si prestamos atención, las filas que son distintas en realidad son los mismos valores multiplicados todos por una constante).
- La ventaja de haber hecho los cálculos en términos de multiplicadores y reemplazos de filas, es que ahora tenemos un sistema para crear un algoritmo y poder programarlo.
- Para cada fila q desde 1 hasta $n-1$ y luego para cada fila r desde $q+1$ hasta n, hay que hacer los reemplazos:

$$m_{rq} = \frac{a_{rq}}{a_{qq}} \qquad E_r - m_{rq} \times E_q \to E_r \qquad q = 1, ..., n-1 \qquad r = q+1, ..., n$$

- El algoritmo resulta ser:

```
knitr::include_graphics("Plots/U3/alg2.png")
```

Algoritmo 2 Eliminación Gaussiana para matrices invertibles

Entrada: A: matriz invertible nxn; b: matriz nx1
Salida: x: solución del sistema lineal A x = b

```
n ← número de filas de A
Aum ← A || b  (Concatenación horizontal, matriz aumentada)

Los siguientes pasos triangularizan a la matriz A
para q desde 1 hasta n-1 hacer
  para r desde q+1 hasta n hacer
    mrq ← Aum[r, q] / Aum[q, q]
    Aum[r, ] ← Aum[r, ] - mrq * Aum[q, ]
  fin para
fin para

Una vez triangularizada la matriz, aplicar sustitución regresiva.
Notar que la última columna de Aum es la de términos independientes
x ← Resultado del algoritmo 1
devolver x
```

3.2.4. Eliminación gaussiana con estrategias de pivoteo

3.2.4.1. Pivoteo trivial

- En el algoritmo anterior es necesario que los pivotes $a_{qq} \neq 0 \ \forall q$.
- Si en uno de los pasos encontramos un $a_{qq} = 0$, debemos intercambiar la q-ésima fila por una cualquiera de las siguientes, por ejemplo la fila k, en la que $a_{kq} \neq 0, k > q$.
- Esta estrategia para hallar un pivote no nulo se llama **pivoteo trivial**.
- **Ejemplo:** verificar "a mano" que con el siguiente sistema, si seguimos el algoritmo anterior, incurrimos en este problema. En cambio, si seguimos el algoritmo con pivoteo trivial llegamos a la solución.

$$\begin{cases} x - 2y + z = -4 \\ -2x + 4y - 3z = 3 \\ x - 3y - 4z = -1 \end{cases}$$

3.2.4.1.1. Estrategias de pivoteo para reducir los errores de redondeo

- Como ya sabemos, dado que las computadoras usan una aritmética cuya precisión está fijada de antemano, es posible que cada vez que se realice una operación aritmética se introduzca un pequeño error de redondeo.

- En la resolución de ecuaciones por eliminación gaussiana, un adecuado reordenamiento de las filas en el momento indicado puede reducir notablemente los errores cometidos.
- Por ejemplo, se puede mostrar cómo buscar en cada paso un multiplicador de menor magnitud (es decir, un pivote de mayor magnitud) mejora la precisión de los resultados.
- Por eso, existen estrategias de pivoteo que no solamente hacen intercambio de filas cuando se tiene un pivote nulo, si no cuando se detecta que un reordenamiento puede reducir el error de redondeo.

Pivoteo parcial

- Para reducir la propagación de los errores de redondeo, antes de comenzar una nueva ronda de reemplazos con el pivote a_{qq} se evalúa si debajo en la misma columna hay algún elemento con mayor valor absoluto y en ese caso se intercambian las respectivas filas.
- Es decir, se busca si existe r tal que $|a_{rq}| > |a_{qq}|, \quad r > q$ para luego intercambiar las filas q y r.
- Este proceso suele conservar las magnitudes relativas de los elementos de la matriz triangular superior en el mismo orden de magnitud que las de los coeficientes de la matriz original.
- Ver ejemplos 1 y 2 de las páginas 280 y 281.

Pivoteo parcial escalado

- Reduce aún más los efectos de la propagación de los errores.
- Se elige el elemento de la columna q-ésima, en o por debajo de la diagonal principal, que tiene mayor tamaño relativo con respecto al resto de los elementos de su fila.
- Antes de definir el multiplicador y hacer las operaciones correspondientes, en cada paso se debe:
  1. Buscar el máximo valor absoluto en cada fila:

$$s_r = max\{|a_{rq}|, |a_{r,q+1}|, \cdots, |a_{rn}|\} \quad r = q, q+1, \cdots, n$$

  2. Elegir como fila pivote a la que tenga el mayor valor de $\frac{|a_{rq}|}{s_r}$, $r = q, q+1, \cdots, n$.
  3. Intercambiar la fila q con la fila hallada en el punto anterior.
- Ver los ejemplos bajo el título “Ilustración” de las páginas 283 y 284.

3.2.4.2. Algoritmos

- En el siguiente algoritmo se detalla la sistematización necesaria para implementar las tres estrategias de pivoteo mencionadas.
- No se pedirá su programación, se puede usar la función provista.
- Se sugiere la lectura tanto del algoritmo como de la función para distinguir cómo son aplicadas las estrategias.

Algoritmo 3 Eliminación Gaussiana con estrategias de pivoteo

Entrada: A: matriz invertible nxn; b: matriz nx1; estrategia: "trivial", "parcial" o "escalado"
Salida: x: solución del sistema lineal A x = b

```
n ← número de filas de A
Aum ← A || b  (Concatenación horizontal, matriz aumentada)

para q desde 1 hasta n-1 hacer

  si estrategia = "trivial" entonces

    Pivoteo trivial: Fijarse que el pivote no sea 0 y si lo es buscar otra fila en la que no sea 0 para
    intercambiar
    si Aum[q, q] = 0 entonces
      para r desde q + 1 hasta n hacer
        si Aum[r, q] ≠ 0 entonces
          temp ← Aum[q, ]
          Aum[q, ] ← Aum[r, ]
          Aum[r, ] ← temp
        fin si
      fin para
    fin si

  si no, si estrategia = "parcial" entonces

    Pivoteo parcial: intercambiar por la fila que tenga pivote con mayor valor absoluto
    Crear vector con los posibles pivotes en valor absoluto:
    candidatos ← abs(Aum[q:n, q])
    Calcular r, el número de fila que usaremos en el intercambio, r ≥ q:
    r ← q - 1 + (posición de max(candidatos))
    Intercambiar filas:
    temp ← Aum[q, ]
    Aum[q, ] ← Aum[r, ]
    Aum[r, ] ← temp

  si no, si estrategia = "escalado" entonces

    Pivoteo parcial escalonado: intercambiar por la fila que tenga pivote con mayor tamaño relativo a los
    elementos de su fila
    Crear vector columna con el maximo de cada fila en valor absoluto:
    sr ← vector columna con el máximo valor absoluto de cada fila de Aum[q:n, q:n]
    Posibles pivotes divididos por el maximo de su fila:
    sr2 ← abs(Aum[q:n, q]) / sr
    Fila que usaremos en el intercambio porque su pivote tiene mayor tamaño relativo:
    r ← q - 1 + (posición de max(sr2))
    Intercambiar filas:
    temp ← Aum[q, ]
    Aum[q, ] ← Aum[r, ]
    Aum[r, ] ← temp

  fin si
```

```
Si después de buscar en todas las filas sigue siendo 0 es porque no había un pivote distinto de 0 y no se
puede resolver
si Aum[q, q] = 0 entonces
  imprimir "A es singular. No hay solución o no es única".
  devolver (Finalizar sin devolver resultado)
fin si

Finalizado el pivoteo, realizar reemplazos para triangularizar a la matriz
para r desde q+1 hasta n hacer
  mrq ← Aum[r, q] / Aum[q, q]
  Aum[r, ] ← Aum[r, ] - mrq * Aum[q, ]
fin para
fin para

Una vez triangularizada la matriz, aplicar sustitución regresiva.
Notar que la última columna de Aum es la de términos independientes
x ← Resultado del algoritmo 1
devolver x
```

3.2.5. Método de eliminación de Gauss-Jordan

- Ya vimos que el método de Gauss transforma la matriz de coeficientes **A** en una matriz triangular superior, haciendo estos reemplazos:

$$m_{rq} = \frac{a_{rq}}{a_{qq}} \qquad E_r - m_{rq} \times E_q \to E_r \qquad q = 1, ..., n-1 \qquad r = q+1, ..., n$$

- El método de Gauss-Jordan transforma **A** hasta obtener la matriz identidad.
- Los multiplicadores se definen de la misma forma, pero en cada paso q se sustituyen todas las filas, no sólo las filas posteriores a la fila q.
- Para cada fila q desde 1 hasta $n-1$ y luego para cada fila r desde 1 hasta n, hay que hacer los reemplazos:

$$\text{Si } r \neq q: \; m_{rq} = \frac{a_{rq}}{a_{qq}} \qquad E_r - m_{rq} \times E_q \to E_r$$

$$\text{Si } r = q: \; E_r / a_{rr} \to E_r$$

- Trabajando de esta forma, además de resolver el sistema se puede hallar fácilmente la inversa de **A**.
- Para esto hay que concatenar a la derecha de la matriz aumentada una matriz identidad de orden n.
- Cuando en la submatriz izquierda se llega a la matriz identidad, en el centro habrá quedado el vector solución y a su derecha la matriz inversa.
- Retomemos el último ejemplo para resolver el sistema mediante Gauss-Jordan y, de paso, obtener $\mathbf{A}^{-1}$.

$$\left[\begin{array}{cccc|c} 1 & 2 & -1 & 3 & -8 \\ 2 & 0 & 2 & -1 & 13 \\ -1 & 1 & 1 & -1 & 8 \\ 3 & 3 & -1 & 2 & -1 \end{array}\right]$$

- Agregamos la matriz identidad a su derecha:

$$\left[\begin{array}{cccc|c|cccc} 1 & 2 & -1 & 3 & -8 & 1 & 0 & 0 & 0 \\ 2 & 0 & 2 & -1 & 13 & 0 & 1 & 0 & 0 \\ -1 & 1 & 1 & -1 & 8 & 0 & 0 & 1 & 0 \\ 3 & 3 & -1 & 2 & -1 & 0 & 0 & 0 & 1 \end{array}\right]$$

- Definimos los multiplicadores y aplicamos las sustituciones:

Paso 1

$$\begin{array}{c} E_2 - 2E_1 \to E_2 \\ E_3 + E_1 \to E_3 \\ E_4 - 3E_1 \to E_4 \\ E_1/1 \to E_1 \end{array} \Longrightarrow \left[\begin{array}{cccc|c|cccc} 1 & 2 & -1 & 3 & -8 & 1 & 0 & 0 & 0 \\ 0 & -4 & 4 & -7 & 29 & -2 & 1 & 0 & 0 \\ 0 & 3 & 0 & 2 & 0 & 1 & 0 & 1 & 0 \\ 0 & -3 & 2 & -7 & 23 & -3 & 0 & 0 & 1 \end{array}\right]$$

Paso 2

$$\begin{array}{c} E_1 + 1/2E_2 \to E_1 \\ E_3 + 3/4E_2 \to E_3 \\ E_4 - 3/4E_2 \to E_4 \\ E_2/(-4) \to E_2 \end{array} \Longrightarrow \left[\begin{array}{cccc|c|cccc} 1 & 0 & 1 & -1/2 & 13/2 & 0 & 1/2 & 0 & 0 \\ 0 & 1 & -1 & 7/4 & -29/4 & 1/2 & -1/4 & 0 & 0 \\ 0 & 0 & 3 & -13/4 & 87/4 & -1/2 & 3/4 & 1 & 0 \\ 0 & 0 & -1 & -7/4 & 5/4 & -3/2 & -3/4 & 0 & 1 \end{array}\right]$$

Paso 3

$$\begin{array}{c} E_1 - 1/3E_3 \to E_1 \\ E_2 + 1/3E_3 \to E_2 \\ E_4 + 1/3E_3 \to E_4 \\ E_3/3 \to E_3 \end{array} \Longrightarrow \left[\begin{array}{cccc|c|cccc} 1 & 0 & 0 & 7/12 & -3/4 & 1/6 & 1/4 & -1/3 & 0 \\ 0 & 1 & 0 & 2/3 & 0 & 1/3 & 0 & 1/3 & 0 \\ 0 & 0 & 1 & -13/12 & 29/4 & -1/6 & 1/4 & 1/3 & 0 \\ 0 & 0 & 0 & -17/6 & 17/2 & -5/3 & -1/2 & 1/3 & 1 \end{array}\right]$$

Paso 4

$$\begin{array}{c} E_1 + 7/34E_4 \to E_1 \\ E_2 + 4/17E_4 \to E_2 \\ E_3 - 13/34E_4 \to E_3 \\ E_4/(-17/6) \to E_4 \end{array} \Longrightarrow \left[\begin{array}{cccc|c|cccc} 1 & 0 & 0 & 0 & 1 & -3/17 & 5/34 & -9/34 & 7/34 \\ 0 & 1 & 0 & 0 & 2 & -2/17 & -1/17 & 7/17 & 4/17 \\ 0 & 0 & 1 & 0 & 4 & 8/17 & 15/34 & 7/34 & -13/34 \\ 0 & 0 & 0 & 1 & -3 & 10/17 & 3/17 & -2/17 & -6/17 \end{array}\right]$$

- En el la columna central tenemos la solución del sistema: $x = 1$, $y = 2$, $z = 4$ y $t = -3$ y en la submatriz de la derecha, la inversa de $\mathbf{A}$:

$$\mathbf{A}^{-1} = \begin{bmatrix} -3/17 & 5/34 & -9/34 & 7/34 \\ -2/17 & -1/17 & 7/17 & 4/17 \\ 8/17 & 15/34 & 7/34 & -13/34 \\ 10/17 & 3/17 & -2/17 & -6/17 \end{bmatrix}$$

- ¿Por qué este procedimiento nos devuelve la inversa de la matriz de coeficientes?
- Recordar que la inversa de $\mathbf{A}$ es aquella matriz $\mathbf{A}^{-1}$ que verifica $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$
- Entonces si queremos hallar la matriz $\mathbf{A}^{-1}$:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & -1 & 3 \\ 2 & 0 & 2 & -1 \\ -1 & 1 & 1 & -1 \\ 3 & 3 & -1 & 2 \end{bmatrix} \qquad \mathbf{A}^{-1} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

podemos plantear el siguiente producto matricial:

$$\begin{bmatrix} 1 & 2 & -1 & 3 \\ 2 & 0 & 2 & -1 \\ -1 & 1 & 1 & -1 \\ 3 & 3 & -1 & 2 \end{bmatrix} \times \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

que da lugar a cuatro sistemas de ecuaciones con la misma matriz de coeficientes, pero distintos vectores de términos independientes, cada uno de los cuales es una columna de la matriz identidad:

$$\begin{bmatrix} c_{11} + 2c_{21} - c_{31} + 3c_{41} & c_{12} + 2c_{22} - c_{32} + 3c_{42} & c_{13} + 2c_{23} - c_{33} + 3c_{43} & c_{14} + 2c_{24} - c_{34} + 3c_{44} \\ 2c_{11} + 2c_{31} - c_{41} & 2c_{12} + 2c_{32} - c_{42} & 2c_{13} + 2c_{33} - c_{43} & 2c_{14} + 2c_{34} - c_{44} \\ -c_{11} + c_{21} + c_{31} - c_{41} & -c_{12} + c_{22} + c_{32} - c_{42} & -c_{13} + c_{23} + c_{33} - c_{43} & -c_{14} + c_{24} + c_{34} - c_{44} \\ 3c_{11} + 3c_{21} - c_{31} + 2c_{41} & 3c_{12} + 3c_{22} - c_{32} + 2c_{42} & 3c_{13} + 3c_{23} - c_{33} + 2c_{43} & 3c_{14} + 3c_{24} - c_{34} + 2c_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{cases} c_{11} + 2c_{21} - c_{31} + 3c_{41} = 1 \\ 2c_{11} + 2c_{31} - c_{41} = 0 \\ -c_{11} + c_{21} + c_{31} - c_{41} = 0 \\ 3c_{11} + 3c_{21} - c_{31} + 2c_{41} = 0 \end{cases} \quad \begin{cases} c_{12} + 2c_{22} - c_{32} + 3c_{42} = 0 \\ 2c_{12} + 2c_{32} - c_{42} = 1 \\ -c_{12} + c_{22} + c_{32} - c_{42} = 0 \\ 3c_{12} + 3c_{22} - c_{32} + 2c_{42} = 0 \end{cases} \quad \begin{cases} c_{13} + 2c_{23} - c_{33} + 3c_{43} = 0 \\ 2c_{13} + 2c_{33} - c_{43} = 0 \\ -c_{13} + c_{23} + c_{33} - c_{43} = 1 \\ 3c_{13} + 3c_{23} - c_{33} + 2c_{43} = 0 \end{cases} \quad \begin{cases} c_{14} + 2c_{24} - c_{34} + 3c_{44} = 0 \\ 2c_{14} + 2c_{34} - c_{44} = 0 \\ -c_{14} + c_{24} + c_{34} - c_{44} = 0 \\ 3c_{14} + 3c_{24} - c_{34} + 2c_{44} = 1 \end{cases}$$

- Al agregar la matriz identidad en la matriz aumentada, lo que estamos haciendo es resolver simultáneamente 5 sistemas de ecuaciones: el original y los 4 que nos permiten encontrar las columnas de la matriz inversa de $\mathbf{A}$.
- A continuación podemos ver el algoritmo de Gauss-Jordan. Se provee también la función de Python que lo implementa:

Algoritmo 4 Eliminación de Gauss-Jordan para matrices invertibles con pivoteo trivial

Entrada: A: matriz invertible nxn; b: matriz nx1
Salida: x: solución del sistema lineal A x = b, inv: inversa de A

```
n ← número de filas de A
I ← matriz identidad de dimensión nxn
Aum ← A || b || I  (Concatenación horizontal, matriz aumentada)

para q desde 1 hasta n hacer

  Pivoteo trivial: Fijarse que el pivote no sea 0 y si lo es buscar otra fila en la que no sea 0 para intercambiar

  si Aum[q, q] = 0 entonces
    para r desde q + 1 hasta n hacer
      si Aum[r, q] ≠ 0 entonces
        temp ← Aum[q, ]
        Aum[q, ] ← Aum[r, ]
        Aum[r, ] ← temp
      fin si
    fin para
    Si después de buscar en todas las filas sigue siendo 0 es porque no había un pivote distinto de 0 y no
    se puede resolver
    si Aum[q, q] = 0 entonces
      imprimir "A es singular. No hay solución o no es única".
      devolver  (Finalizar sin devolver resultado)
    fin si
  fin si

  Realizar reemplazos para llegar a la matriz identidad
  para r desde 1 hasta n hacer
    si r = q entonces
      Aum[q, ] ← Aum[q, ] / Aum[q, q]
    si no
      mrq ← Aum[r, q] / Aum[q, q]
      Aum[r, ] ← Aum[r, ] - mrq * Aum[q, ]
    fin si
  fin para
fin para

La última columna de Aum es la solución
imprimir "La inversa de A es " Aum[, (n+2):(2*n+1)]
x ← Aum[, n+1]
inv ← Aum[, (n+2):(2*n+1)]
devolver x, inv
```

3.2.6. Factorización LU

Definición: En álgebra lineal la **factorización de una matriz** es la descomposición de la misma como producto de dos o más matrices que toman una forma especificada.

- Las factorizaciones de las matrices se utilizan para facilitar el cálculo de diversos elemen-

tos como determinantes e inversas y para optimizar algoritmos.

- La **factorización LU** se obtiene cuando se expresa a una matriz cuadrada $\mathbf{A}$ como el producto entre una matriz triangular inferior $\mathbf{L}$ y una matriz triangular superior $\mathbf{U}$, es decir, $\mathbf{A} = \mathbf{LU}$.
- Si bien existe más de una forma de encontrar la descomposición LU de una matriz $\mathbf{A}$, la misma se puede obtener fácilmente aprovechando los cálculos que se realizan con el método de eliminación gaussiana para resolver un sistema de la forma $\mathbf{Ax} = \mathbf{b}$.
- Recordemos el ejemplo visto en la sección sobre eliminación gaussiana.

$$\begin{cases} x + 2y - z + 3t = -8 \\ 2x + 2z - t = 13 \\ -x + y + z - t = 8 \\ 3x + 3y - z + 2t = -1 \end{cases} \qquad \mathbf{A} = \begin{bmatrix} 1 & 2 & -1 & 3 \\ 2 & 0 & 2 & -1 \\ -1 & 1 & 1 & -1 \\ 3 & 3 & -1 & 2 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -8 \\ 13 \\ 8 \\ -1 \end{bmatrix}$$

- La factorización LU de la matriz $\mathbf{A}$ se obtiene al considerar como $\mathbf{U}$ a la matriz triangular superior que obtuvimos al finalizar la eliminación gaussiana y al crear $\mathbf{L}$ con los multiplicadores acomodados como se indica a continuación:

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ m_{21} & 1 & 0 & 0 \\ m_{31} & m_{32} & 1 & 0 \\ m_{41} & m_{42} & m_{43} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -1 & -3/4 & 1 & 0 \\ 3 & 3/4 & -1/3 & 1 \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} 1 & 2 & -1 & 3 \\ 0 & -4 & 4 & -7 \\ 0 & 0 & 3 & -13/4 \\ 0 & 0 & 0 & -17/6 \end{bmatrix}$$

- Podemos verificar estos cálculos definiendo estas matrices y encontrando que su producto es igual a $\mathbf{A}$.

3.2.6.1. Usos de la factorización LU

- Una vez encontrada la factorización, esta se puede emplear para facilitar el cálculo de determinantes e inversas y también para resolver otros sistemas del tipo $\mathbf{Ax} = \mathbf{b}$, con la misma $\mathbf{A}$ pero cualquier $\mathbf{b}$.

a. Uso de LU para resolver sistemas de ecuaciones lineales

- Sea $\mathbf{Ax} = \mathbf{b}$ un sistema lineal que debe resolverse para $\mathbf{x}$ y supongamos que contamos con la factorización LU de $\mathbf{A}$ de modo que $\mathbf{A}$: $\mathbf{A} = \mathbf{LU}$.
- La solución $\mathbf{x}$ se encuentra con estos dos pasos:
  1. Resolver el sistema $\mathbf{Ly} = \mathbf{b}$, donde $\mathbf{y}$ es el vector de incógnitas. Siendo $\mathbf{L}$ triangular inferior, este sistema se resuelve fácilmente mediante sustitución hacia adelante.
  2. Resolver el sistema $\mathbf{Ux} = \mathbf{y}$, donde $\mathbf{y}$ es el vector obtenido en el punto anterior. Siendo $\mathbf{U}$ triangular superior, este sistema se resuelve fácilmente mediante sustitución hacia atrás.

- Como se puede ver, los dos pasos consisten resolver sistemas de ecuaciones con matrices de coeficientes triangulares, lo cual es sencillo y no requiere de la implementación de eliminación gaussiana (de todos modos, se necesita aplicar eliminación gaussiana o algún otro proceso para obtener la factorización LU en primer lugar).
- Sin embargo, este procedimiento se puede aplicar para resolver el sistema múltiples veces para diferentes **b**. Aplicar eliminación gaussiana una vez para obtener la factorización LU y luego emplearla en la solución de cada sistema es más eficiente que aplicar siempre eliminación gaussiana para cada caso.
- Se dice que las matrices **L** y **U** representan en sí mismas el proceso de eliminación gaussiana.
- *Ejemplo*: resolver el sistema $\mathbf{Ax} = \mathbf{b}$ con $\mathbf{b}^t = [-8 \quad 13 \quad 8 \quad -1]$ a partir de la descomposición LU dada por:

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -1 & -3/4 & 1 & 0 \\ 3 & 3/4 & -1/3 & 1 \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} 1 & 2 & -1 & 3 \\ 0 & -4 & 4 & -7 \\ 0 & 0 & 3 & -13/4 \\ 0 & 0 & 0 & -17/6 \end{bmatrix}$$

b. Uso de LU para calcular la inversa de A

- La inversa de **A** se puede obtener como $\mathbf{A}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}$.
- O también se pueden aplicar los dos pasos anteriores para resolver n veces el sistema $\mathbf{Ax} = \mathbf{b}$, haciendo que **b** sea igual a cada una de las columnas de la matriz identidad en cada oportunidad (cada vez x nos dará una columna de $\mathbf{A}^{-1}$).
- Estas formas no son más eficientes que el método tradicional de hallar la inversa de una matriz.

c. Uso de LU para calcular el determinante de A

- $det(\mathbf{A}) = det(\mathbf{L})det(\mathbf{U})$.
- El determinante de una matriz triangular se obtiene fácilmente como el producto de los elementos diagonales.

3.2.6.2. Otros detalles

- La factorización LU puede llegar a fallar en algunos casos. Por ejemplo, si la obtenemos a través de la eliminación gaussiana, no podríamos completarla cuando algún pivote es cero.
- Así como en la eliminación gaussiana aplicamos pivoteo para evitar ese problema, en el proceso de factorización LU intercambiar el orden de las filas de la matriz también es la solución.
- De hecho, se demuestra que toda matriz cuadrada admite una factorización LU si previamente se reordenan convenientemente sus filas.
- El reordenamiento de filas se representa matemáticamente a través de una matriz **P** de unos y ceros que premultiplicada a **A** produce el efecto de intercambiar filas. Esta

matriz se llama **matriz de permutación**. Luego, toda matriz cuadrada admite una factorización LU del tipo $\mathbf{PA} = \mathbf{LU}$ (también llamada **factorización PLU**).

- La forma vista para obtener de la factorización LU siguiendo los pasos de la eliminación gaussiana recibe el nombre de **método de Doolittle** y como resultado la matriz $\mathbf{L}$ tiene unos en su diagonal.
- Otro método que resulta en que la matriz $\mathbf{U}$ sea la que tenga unos en la diagonal se llama **método de Crout**.
- La **factorización de Cholesky** es un caso particular de la factorización LU que se aplica a matrices semidefinidas positivas y que resulta en que $\mathbf{U} = \mathbf{L}^t$, es decir, $\mathbf{A} = \mathbf{L}\mathbf{L}^t$.

3.3. Métodos Aproximados o Iterativos

- En esta sección describimos los métodos iterativos de **Jacobi** y de **Gauss-Seidel** para resolver sistemas de ecuaciones lineales.
- Una técnica iterativa para resolver el sistema lineal $\mathbf{Ax} = \mathbf{b}$ inicia con una aproximación $\mathbf{x}^{(0)}$ para la solución $\mathbf{x}$ y genera una sucesión de vectores $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$ que convergen a $\mathbf{x}$.
- Las técnicas iterativas casi nunca se usan para resolver sistemas lineales de dimensiones pequeñas ya que el tiempo requerido para conseguir una precisión suficiente excede el requerido para las técnicas directas, como la eliminación gaussiana.
- Para grandes sistemas con un alto porcentaje de entradas 0, sin embargo, estas técnicas son eficientes en términos tanto de almacenamiento como de cálculo computacional.

3.3.1. Método de Jacobi

- Consideremos el sistema:

$$\begin{cases} 4x - y + z = 7 \\ 4x - 8y + z = -21 \\ -2x + y + 5z = 15 \end{cases} \implies \begin{cases} x = (7 + y - z)/4 \\ y = (21 + 4x + z)/8 \\ z = (15 + 2x - y)/5 \end{cases}$$

- Despejar una incógnita en cada ecuación provee expresiones que sugieren la idea de un proceso iterativo.
- Dado un vector de valores iniciales $\mathbf{x}^{(0)} = (x^{(0)}, y^{(0)}, z^{(0)})$, operar con la siguiente fórmula de recurrencia hasta la convergencia:

$$\begin{cases} x^{(k+1)} = (7 + y^{(k)} - z^{(k)})/4 \\ y^{(k+1)} = (21 + 4x^{(k)} + z^{(k)})/8 \\ z^{(k+1)} = (15 + 2x^{(k)} - y^{(k)})/5 \end{cases}$$

- Por ejemplo, tomando el valor inicial $(1, 2, 2)$, el proceso converge hacia la solución exacta del sistema $(2, 4, 3)$.

- Usando 4 posiciones decimales con redondeo luego de la coma:

k	$x^{(k)}$	$y^{(k)}$	$z^{(k)}$
0	1	2	2
1	1.7500	3.3750	3.0000
2	1.8438	3.8750	3.0250
3	1.9625	3.9250	2.9625
4	1.9906	3.9766	3.0000
5	1.9942	3.9953	3.0009
6	1.9986	3.9972	2.9986
7	1.9997	3.9991	3.0000
8	1.9998	3.9999	3.0001
9	2.0000	3.9999	2.9999
10	2.0000	4.0000	3.0000

- *Observación*: no siempre este método converge. Es sensible al ordenamiento de las ecuaciones dentro del sistema.
- Ejemplo: tomamos el mismo sistema de antes pero intercambiamos las filas 1 y 3:

$$\begin{cases} 4x - y + z = 7 \\ 4x - 8y + z = -21 \\ -2x + y + 5z = 15 \end{cases} \implies \begin{cases} -2x + y + 5z = 15 \\ 4x - 8y + z = -21 \\ 4x - y + z = 7 \end{cases}$$

$$\implies \begin{cases} x = (-15 + y + 5z)/2 \\ y = (21 + 4x + z)/8 \\ z = 7 - 4x + y \end{cases} \implies \begin{cases} x^{(k+1)} = (-15 + y^{(k)} + 5z^{(k)})/2 \\ y^{(k+1)} = (21 + 4x^{(k)} + z^{(k)})/8 \\ z^{(k+1)} = 7 - 4x^{(k)} + y^{(k)} \end{cases}$$

- Tomando el mismo valor inicial $(1, 2, 2)$, esta vez el proceso diverge:

k	$x^{(k)}$	$y^{(k)}$	$z^{(k)}$
0	1	2	2
1	-1.5000	3.3750	5.0000
2	6.6875	2.5000	16.3750
3	34.6875	8.0156	-17.2500
4	-46.6172	17.8125	-123.7344
5	-307.9298	-36.1504	211.2813
6	502.6281	-124.9297	1202.5688
7	2936.4572	404.2602	-2128.4421
8	-5126.4752	1204.7983	-11334.5686
9	-27741.5224	-3977.4337	21717.6991

k	$x^{(k)}$	$y^{(k)}$	$z^{(k)}$
10	52298.0309	-11153.4238	106995.6559

- Vamos a desarrollar fórmulas para la expresión general de este método:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{j1}x_1 + a_{j2}x_2 + \cdots + a_{jn}x_n = b_j \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \implies \begin{cases} x_1 = \frac{b_1 - a_{12}x_2 - \cdots - a_{1n}x_n}{a_{11}} \\ x_2 = \frac{b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n}{a_{22}} \\ \vdots \\ x_j = \frac{b_j - a_{j1}x_1 \cdots - a_{j(j-1)}x_{j-1} - a_{j(j+1)}x_{j+1} - \cdots - a_{jn}x_n}{a_{jj}} \\ \vdots \\ x_n = \frac{b_n - a_{n1}x_1 - \cdots - a_{n(n-1)}x_{n-1}}{a_{nn}} \end{cases}$$

- A partir de estas expresiones, se plantea el proceso iterativo que arranca con valores iniciales $(x_1^{(0)}, ..., x_n^{(0)})$:

$$\begin{cases} x_1^{(k+1)} = \frac{b_1 - a_{12}x_2^{(k)} - \cdots - a_{1n}x_n^{(k)}}{a_{11}} \\ x_2^{(k+1)} = \frac{b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)} - \cdots - a_{2n}x_n^{(k)}}{a_{22}} \\ \vdots \\ x_j^{(k+1)} = \frac{b_j - a_{j1}x_1^{(k)} \cdots - a_{j(j-1)}x_{j-1}^{(k)} - a_{j(j+1)}x_{j+1}^{(k)} - \cdots - a_{jn}x_n^{(k)}}{a_{jj}} \\ \vdots \\ x_n^{(k+1)} = \frac{b_n - a_{n1}x_1^{(k)} - \cdots - a_{n(n-1)}x_{n-1}^{(k)}}{a_{nn}} \end{cases}$$

$$\implies x_j^{(k+1)} = \frac{1}{a_{jj}}\Big[b_j - \sum_{\substack{i=1 \\ i \neq j}}^{n} a_{ji}x_j^{(k)}\Big] \qquad j = 1, 2, \cdots n \qquad k = 0, 1, 2, \cdots$$

- Lo anterior se puede expresar de forma matricial para simplificar la tarea de programación o para tener una expresión más cómoda a fines de estudiar propiedades teóricas del método.
- Si descomponemos a la matriz $\mathbf{A}$ como $\mathbf{A} = \mathbf{D} + \mathbf{R}$, donde $\mathbf{D}$ es la matriz diagonal formada con la diagonal de $\mathbf{A}$ y $\mathbf{R}$ es igual a $\mathbf{A}$ excepto en la diagonal donde posee todos ceros, tenemos:

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ (\mathbf{D}+\mathbf{R})\mathbf{x} &= \mathbf{b} \\ \mathbf{D}\mathbf{x} &= \mathbf{b}-\mathbf{R}\mathbf{x} \\ \mathbf{x} &= \mathbf{D}^{-1}(\mathbf{b}-\mathbf{R}\mathbf{x}) \end{aligned}$$

- Esto da lugar a la siguiente fórmula de recurrencia, que es equivalente a la vista anteriormente:

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}(\mathbf{b}-\mathbf{R}\mathbf{x}^{(k)})$$

- Requisito: ningún elemento en la diagonal de $\mathbf{A}$ es cero.

3.3.2. Método de Gauss-Seidel

- Toma la misma idea que Jacobi, pero con una pequeña modificación para acelerar la convergencia.
- La diferencia está en que apenas calcula un nuevo valor de las incógnitas, Gauss-Seidel lo usa inmediatamente en el cálculo de las restantes dentro del mismo paso iterativo, en lugar esperar a la próxima ronda.
- Retomando el ejemplo anterior:

$$\begin{cases} 4x-y+z=7 \\ 4x-8y+z=-21 \\ -2x+y+5z=15 \end{cases} \implies \begin{cases} x=(7+y-z)/4 \\ y=(21+4x+z)/8 \\ z=(15+2x-y)/5 \end{cases} \overset{Jacobi}{\implies} \begin{cases} x^{(k+1)}=(7+y^{(k)}-z^{(k)})/4 \\ y^{(k+1)}=(21+4x^{(k)}+z^{(k)})/8 \\ z^{(k+1)}=(15+2x^{(k)}-y^{(k)})/5 \end{cases}$$

- El proceso iterativo de Gauss-Seidel es:

$$\begin{cases} x^{(k+1)}=(7+y^{(k)}-z^{(k)})/4 \\ y^{(k+1)}=(21+4x^{(k+1)}+z^{(k)})/8 \\ z^{(k+1)}=(15+2x^{(k+1)}-y^{(k+1)})/5 \end{cases}$$

- Tomando otra vez el valor inicial $(1,2,2)$ y operando con 4 posiciones decimales con redondeo luego de la coma:

Jacobi				**Gauss-Seidel**			
k	$x^{(k)}$	$y^{(k)}$	$z^{(k)}$	k	$x^{(k)}$	$y^{(k)}$	$z^{(k)}$

Jacobi				**Gauss-Seidel**			
0	1	2	2	0	1	2	2
1	1.7500	3.3750	3.0000	1	1.7500	3.7500	2.9500
2	1.8438	3.8750	3.0250	2	1.9500	3.9688	2.9862
3	1.9625	3.9250	2.9625	3	1.9957	3.9961	2.9991
4	1.9906	3.9766	3.0000	4	1.9993	3.9995	2.9998
5	1.9942	3.9953	3.0009	5	1.9999	3.9999	3.0000
6	1.9986	3.9972	2.9986	6	2.0000	4.0000	3.0000
7	1.9997	3.9991	3.0000				
8	1.9998	3.9999	3.0001				
9	2.0000	3.9999	2.9999				
10	2.0000	4.0000	3.0000				

- En general, este método converge más rápidamente que el de Jacobi, pero no siempre es así.
- Existen sistemas lineales para los que el método de Jacobi converge y el de Gauss-Seidel no.
- La expresión general para el método es:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{j1}x_1 + a_{j2}x_2 + \cdots + a_{jn}x_n = b_j \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \Longrightarrow \begin{cases} x_1 = \frac{b_1 - a_{12}x_2 - \cdots - a_{1n}x_n}{a_{11}} \\ x_2 = \frac{b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n}{a_{22}} \\ \vdots \\ x_j = \frac{b_j - a_{j1}x_1 \cdots - a_{j(j-1)}x_{j-1} - a_{j(j+1)}x_{j+1} - \cdots - a_{jn}x_n}{a_{jj}} \\ \vdots \\ x_n = \frac{b_n - a_{n1}x_1 - \cdots - a_{n(n-1)}x_{n-1}}{a_{nn}} \end{cases}$$

- A partir de estas expresiones, se plantea el proceso iterativo que arranca con valores iniciales $(x_1^{(0)}, ..., x_n^{(0)})$:

$$\begin{cases} x_1^{(k+1)} = \frac{b_1 - a_{12}x_2^{(k)} - \cdots - a_{1n}x_n^{(k)}}{a_{11}} \\ x_2^{(k+1)} = \frac{b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - \cdots - a_{2n}x_n^{(k)}}{a_{22}} \\ \vdots \\ x_j^{(k+1)} = \frac{b_j - a_{j1}x_1^{(k+1)} \cdots - a_{j(j-1)}x_{j-1}^{(k+1)} - a_{j(j+1)}x_{j+1}^{(k)} - \cdots - a_{jn}x_n^{(k)}}{a_{jj}} \\ \vdots \\ x_n^{(k+1)} = \frac{b_n - a_{n1}x_1^{(k+1)} - \cdots - a_{n(n-1)}x_{n-1}^{(k+1)}}{a_{nn}} \end{cases}$$

$$\implies x_j^{(k+1)} = \frac{1}{a_{jj}}\Big[b_j - \sum_{i<j} a_{ji}x_i^{(k+1)} - \sum_{i>j} a_{ji}x_i^{(k)}\Big] \qquad j = 1, ..., n \quad k = 0, 1, 2, ...$$

- Para encontrar una expresión matricial, descomponemos a la matriz $\mathbf{A}$ como $\mathbf{A} = \mathbf{L} + \mathbf{U}$, donde $\mathbf{L}$ es una matriz triangular inferior (incluyendo la diagonal de $\mathbf{A}$) y $\mathbf{U}$ es una matriz triangular superior (con ceros en la diagonal):

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ (\mathbf{L} + \mathbf{U})\mathbf{x} &= \mathbf{b} \\ \mathbf{Lx} &= \mathbf{b} - \mathbf{Ux} \\ \mathbf{x} &= \mathbf{L}^{-1}(\mathbf{b} - \mathbf{Ux}) \end{aligned}$$

- Esto da lugar a la siguiente fórmula de recurrencia, que es equivalente a la vista anteriormente:

$$\mathbf{x}^{(k+1)} = \mathbf{L}^{-1}(\mathbf{b} - \mathbf{U}\mathbf{x}^{(k)})$$

- Requisito: ningún elemento en la diagonal de $\mathbf{A}$ es cero.

3.3.3. Convergencia

Condiciones para la convergencia

- Para establecer una condición de convergencia de estos métodos, necesitamos la siguiente definición:

Definición:

- Se dice que una matriz $\mathbf{A}$ de orden $n \times n$ es **diagonal dominante** cuando cada elemento diagonal es mayor o igual a la suma del resto de los elementos de su fila en valor absoluto:

$$|a_{kk}| \geq \sum_{\substack{j=1 \\ j \neq k}}^{n} |a_{kj}| \quad \forall\, k = 1, 2, \cdots, n$$

- Se dice que una matriz $\mathbf{A}$ de orden $n \times n$ es **estrictamente diagonal dominante** cuando cada elemento diagonal es mayor a la suma del resto de los elementos de su fila en valor absoluto:

$$|a_{kk}| > \sum_{\substack{j=1 \\ j \neq k}}^{n} |a_{kj}| \quad \forall\, k = 1, 2, \cdots, n$$

- Los sistemas de ecuaciones cuya matriz de coeficiente es **estrictamente diagonal dominante** poseen algunas ventajas.
- Una matriz **estrictamente diagonal dominante** es no singular (el sistema es compatible determinado).
- Además, la eliminación gaussiana se puede realizar sin intercambios de fila o columna y los cálculos serán estables respecto al crecimiento de los errores de redondeo.
- Y en particular, para estas matrices está asegurada la convergencia de los métodos iterativos, como lo indica el siguiente teorema:

Teorema: si la matriz $\mathbf{A}$ es estrictamente diagonal dominante, entonces el sistema lineal $\mathbf{Ax} = \mathbf{b}$ tiene solución única y los procesos iterativos de Jacobi y de Gauss-Seidel convergen hacia la misma cualquiera sea el vector de partida $\mathbf{x}^{(0)}$.

- Es una condición suficiente pero no necesaria.
- En la práctica, ante un sistema buscamos reordenar las columnas y filas para intentar obtener una matriz con estas características.
- Si no es posible, igualmente buscamos colocar en la diagonal los elementos de mayor valor absoluto, puesto que esto puede favorecer la convergencia.

3.3.3.1. Criterios para detener el proceso iterativo

- Para establecer si los métodos iterativos están convergiendo y así detener el proceso, necesitamos una forma para medir la distancia entre vectores columna n-dimensionales (es decir, entre dos vectores $\mathbf{x}^{(k)}$ consecutivos en la iteración).
- Para definir la distancia en $\mathbb{R}^n$ usamos la noción de **norma**, que es la generalización del valor absoluto en $\mathbb{R}$.

Definición: Una **norma vectorial** en $\mathbb{R}^n$ es una función $||\cdot||$ de $\mathbb{R}^n$ a $\mathbb{R}$, con las siguientes propiedades:

1. $||\mathbf{x}|| \geq 0 \quad \forall\, \mathbf{x} \in \mathbb{R}^n$
2. $||\mathbf{x}|| = 0 \Leftrightarrow \mathbf{x} = \mathbf{0}$
3. $||\alpha\mathbf{x}|| = |\alpha|||\mathbf{x}|| \quad \forall\, \alpha \in \mathbb{R}, \mathbf{x} \in \mathbb{R}^n$
4. $||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}|| \quad \forall\, \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$

- Las siguientes son tres normas ampliamente utilizadas:
  a. **Norma** l_1: $||\mathbf{x}||_1 = \sum_{j=1}^{n} |x_j|$
  b. **Norma** l_2 **o euclideana**: $||\mathbf{x}||_2 = \sqrt{\mathbf{x}^t\mathbf{x}} = \sqrt{\sum_{j=1}^{n} x_j^2}$
  c. **Norma** l_∞: $||\mathbf{x}||_\infty = \text{máx}_{1\leq j\leq n} |x_j|$
- La norma de un vector proporciona una medida para la distancia entre un vector arbitrario y el vector cero, de la misma forma en la que el valor absoluto de un nùmero real describe su distancia desde 0.
- De igual forma, la **distancia entre dos vectores** (que necesitamos para juzgar la convergencia del proceso) está definida como la norma de la diferencia de los vectores, al igual que la distancia entre dos números reales es el valor absoluto de la diferencia.

- Luego, para "comparar" dos vectores sucesivos que aproximan a la solución del sistema, podemos usar las siguientes definiciones de distancias:

  a. **Distancia l_1 o de Manhattan**:

$$||\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}||_1 = \sum_{j=1}^{n} |x_j^{(k+1)} - x_j^{(k)}|$$

  b. **Distancia l_2 o euclídea**:

$$||\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}||_2 = \sqrt{(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)})^t(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k+1)})} = \sqrt{\sum_{j=1}^{n}(x_j^{(k+1)} - x_j^{(k)})^2}$$

  c. **Distancia l_∞ o máxima diferencia**:

$$||\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}||_\infty = \max_j |x_j^{(k+1)} - x_j^{(k)}|$$

- El proceso iterativo se detiene cuando la distancia elegida entre dos vectores solución consecutivos sea menor a algún valor tan pequeño como se desee, ϵ.
- Lo más común es emplear la norma l_∞.
- Siendo semejante a la definición de error relativo, también es usual iterar hasta que:

$$\frac{||\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}||}{||\mathbf{x}^{(k+1)}||} < \epsilon$$

- Por último, se debe establecer un número máximo de iteraciones, para detener el proceso cuando el mismo no converge.

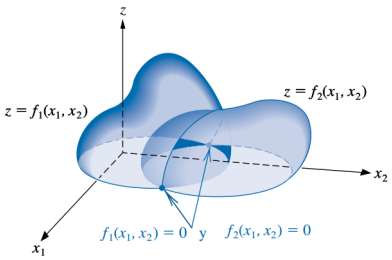
4 Resolución de sistemas de ecuaciones no lineales y optimización

4.1. Introducción

- Un sistema de ecuaciones no lineales $n \times n$ tiene la forma:

$$\begin{cases} f_1(x_1, x_2, \cdots, x_n) = 0 \\ f_2(x_1, x_2, \cdots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \cdots, x_n) = 0 \end{cases}$$

  donde cada función f_i se puede pensar como un mapeo de un vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$ del espacio n-dimensional $\mathbb{R}^n$ en la recta real $\mathbb{R}$.
- En la siguiente figura se muestra una representación geométrica de un sistema no lineal cuando $n = 2$.



- Este sistema de n ecuaciones no lineales en n variables también se puede representar al definir una función vectorial $\mathbf{F}$ de mapeo de $\mathbb{R}^n$ a $\mathbb{R}^n$ (es decir, un *campo vectorial*, así vamos nombrando cosas de Análisis y Álgebra):

$$\mathbf{F}(x_1, x_2, \cdots, x_n) = \begin{pmatrix} f_1(x_1, x_2, \cdots, x_n) \\ f_2(x_1, x_2, \cdots, x_n) \\ \vdots \\ f_n(x_1, x_2, \cdots, x_n) \end{pmatrix}$$

- Si se utiliza notación vectorial con $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$ y $\mathbf{0} = (0, 0, \cdots, 0)^T$, entonces el sistema asume la forma:

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}$$

- Las funciones $f_1, f_2, \cdots, f_n$ reciben el nombre de **funciones coordenadas** de $\mathbf{F}$.
- Un acercamiento para la resolución de sistemas de ecuaciones no lineales es pensar si los métodos vistos en la Unidad 2 para resolver ecuaciones no lineales se pueden adaptar y generalizar.
- Con eso en mente es posible generalizar, por ejemplo, el **método del punto fijo** y el **método de Newton-Raphson**, como veremos en la sección siguiente.
- Hay algo muy interesante y es que la búsqueda de la solución de un sistema de ecuaciones no lineales puede ser formulado como un **problema de optimización**, es decir, un problema en el que se quiere hallar el máximo o el mínimo de función.
- Por esta razón, en muchos libros se presenta de manera conjunta el estudio de métodos para resolver sistemas de ecuaciones no lineales y para resolver problemas de optimización, y ellos son el objeto de estudio de esta unidad.

4.2. Sistemas de ecuaciones no lineales

4.2.1. Método de los puntos fijos

- En la Unidad 2 vimos un método para resolver una ecuación del tipo $f(x) = 0$, al transformar primero la ecuación en la forma de punto fijo $x = g(x)$, tomar un valor inicial x_0 y luego iterar haciendo: $x_k = g(x_{k-1})$.
- Vamos a ver un proceso similar para las funciones de $\mathbb{R}^n$ a $\mathbb{R}^n$.

Definición: una función $\mathbf{G}$ desde $D \subset \mathbb{R}^n$ a $\mathbb{R}^n$ tiene un punto fijo en $\mathbf{p} \subset D$ si $\mathbf{G}(\mathbf{p}) = \mathbf{p}$.

- Para resolver el sistema de n ecuaciones no lineales $\mathbf{F}(\mathbf{x}) = \mathbf{0}$:

$$\begin{cases} f_1(x_1, x_2, \cdots, x_n) = 0 \\ f_2(x_1, x_2, \cdots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \cdots, x_n) = 0 \end{cases}$$

  primero debemos reescribir cada ecuación bajo la forma:

$$\begin{cases} x_1 = g_1(x_1, x_2, \cdots, x_n) \\ x_2 = g_2(x_1, x_2, \cdots, x_n) \\ \vdots \\ x_n = g_n(x_1, x_2, \cdots, x_n) \end{cases}$$

- Llamamos con $\mathbf{G}$ a la función de mapeo $\mathbb{R}^n$ en $\mathbb{R}^n$ que reune a todas las funciones g_i:

$$\mathbf{G}(\mathbf{x}) = \begin{pmatrix} g_1(x_1, x_2, \cdots, x_n) \\ g_2(x_1, x_2, \cdots, x_n) \\ \vdots \\ g_n(x_1, x_2, \cdots, x_n) \end{pmatrix}$$

- Elegimos un vector de valores iniciales $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \cdots, x_n^{(0)})^T$ y efectuamos el proceso iterativo

$$\mathbf{x}^{(k)} = \mathbf{G}(\mathbf{x}^{(k-1)}) \qquad k \geq 1$$

- Si converge, encontraremos el punto fijo de $\mathbf{G}$ que no es más que la solución de $\mathbf{F}(\mathbf{x}) = \mathbf{0}$.
- El Teorema 10.6 del libro (página 479) establece cuáles son las condiciones para garantizar la existencia y unicidad del punto fijo, además de asegurar la convergencia del método. Estas condiciones son generalizaciones de las que vimos para el teorema del punto fijo en la Unidad 2.
- Sin embargo, no siempre es posible o fácil encontrar una representación de las funciones en una forma que requiere el método y que además cumpla con las condiciones para la convergencia.
- Otra opción es adaptar el método de Newton-Raphson, lo cual resulta en una de las técnicas más potentes y ampliamente usadas para resolver sistemas no lineales y de optimización.

4.2.2. Método de Newton (o de Newton-Raphson) para sistemas de ecuaciones

4.2.2.1. Formalización

- Recordemos que para resolver una ecuación no lineal del tipo $f(x) = 0$, a partir del desarrollo de Taylor deducimos el siguiente proceso iterativo para resolverlo:

$$x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})} = x_{k-1} - [f'(x_{k-1})]^{-1} f(x_{k-1}) \qquad k \geq 1$$

  el cual converge siempre que se tome un buen valor inicial x_0.
- Para resolver un sistema no lineal $n \times n$, se extiende esta idea al proponer el siguiente procedimiento de iteración:

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - [\mathbf{J}(\mathbf{x}^{(k-1)})]^{-1} \mathbf{F}(\mathbf{x}^{(k-1)}) \qquad k \geq 1$$

- La matriz $\mathbf{J}(\mathbf{p})$ se llama **matriz jacobiana**. El elemento en la posición (i, j) es la derivada parcial de f_i con respecto a x_j:

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{bmatrix}$$

- Esto recibe el nombre de **Método de Newton para sistemas no lineales**.
- Se demuestra que converge a la verdadera solución $\mathbf{p}$ siempre que se tome un buen vector inicial $\mathbf{x}^{(0)}$ (lo mismo que pasaba con el método de Newton-Raphson) y que exista $[\mathbf{J}(\mathbf{p})]^{-1}$.
- Su convergencia es *cuadrática*, lo cual significa que es rápido.

4.2.2.2. Métodos cuasi-Newton

- La gran desventaja del método de Newton es tener que calcular e invertir la matriz $\mathbf{J}(\mathbf{x})$ en cada paso, lo cual implica realizar muchos cálculos, además de que la evaluación exacta de las derivadas parciales $\frac{\partial f_i}{\partial x_j}(\mathbf{x})$ puede no ser práctica o sencilla.
- Existen numerosas propuestas que persiguen el objetivo de reemplazar de alguna forma la matriz jacobiana con una matriz de aproximación que pueda ser actualizada fácilmente en cada iteración.
- El conjunto de estos algoritmos se conocen como **métodos cuasi-Newton**.
- Los **métodos cuasi-Newton** tienen una convergencia más lenta, pero resultan aceptables porque reducen la cantidad de cálculos a realizar.
- Por ejemplo, habíamos visto que el **método de la secante** era una opción para reemplazar el cálculo de la derivada en el método de Newton-Raphson para resolver una ecuación no lineal. Esa idea se puede extender para sistemas de ecuaciones no lineales, resultando en un método conocido como **método de Broyden** (desarrollado en la sección 10.3 del libro, no lo estudiaremos, pero lo mencionamos por tener amplia difusión y aparecer en numerosas aplicaciones).

4.3. Optimización

Definición: la **optimización matemática** se encarga de resolver problemas en los que se debe encontrar el *mejor* elemento de acuerdo a algún criterio entre un conjunto de alternativas disponibles.

- Los problemas de optimización aparecen todo el tiempo en disciplinas como ciencias de la computación, ingeniería y, en lo que nos interesa a nosotros, Estadística.
- Uno de los problemas de optimización más generales es el de encontrar el valor de $\mathbf{x}$ que minimice o maximice una función dada $f(\mathbf{x})$, sin estar sujeto a ninguna restricción sobre $\mathbf{x}$.

- En Análisis Matemático ya han resuelto problemas de optimización, aunque tal vez los hayan presentados como **problemas de máximos y mínimos**.
- La **optimización matemática** es también llamada **programación matemática**, pero acá el término *programación* no hace referencia a programar una computadora, sino que históricamente se le decía así a este tipo de problemas. En la actualidad, se sigue usando esa palabra para darle nombre al campo de la **programación lineal**, que engloba a los problemas de optimización donde la función a optimizar y las restricciones que se deben verificar están representadas por relaciones lineales.
- La resolución de sistemas de ecuaciones (lineales o no) tiene una estrecha relación con los problemas de optimización.
- Esto es así porque el problema de encontrar la solución de un sistema de ecuaciones puede ser reformulado como un problema en el que se necesita encontrar el mínimo de una función multivariada en particular.
- El sistema definido por:

$$\begin{cases} f_1(x_1, x_2, \cdots, x_n) = 0 \\ f_2(x_1, x_2, \cdots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \cdots, x_n) = 0 \end{cases}$$

  tiene una solución en $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$ precisamente cuando la función g definida por:

$$g(x_1, x_2, \cdots, x_n) = \sum_{i=1}^{n} [f_i(x_1, x_2, \cdots, x_n)]^2$$

  tiene el valor mínimo 0.
- Por esta razón ahora plantearemos de forma más general al método de Newton y sus derivados como métodos de optimización, ya que es así como suelen aparecer en la literatura y, en particular, cuando se recurre a ellos en estadística y *machine learning*.
- En esas aplicaciones, la función que se desea minimizar es una *función de pérdida* o *de costo* y las incógnitas son los valores de los parámetros que producen el valor mínimo. Es decir, g puede ser, por ejemplo, la *función mínimo cuadrática* (en un modelo de regresión que se estima por mínimos cuadradados) o el opuesto de la log-verosimilitud (en un modelo que se estima por máxima verosimilitud) y en lugar de usar la notación de $x_1, x_2, \cdots, x_n$, buscaríamos valores para los parámetros $\beta_1, \beta_2, \cdots, \beta_p$ o $\theta_1, \theta_2, \cdots, \theta_p$.
- Si el problema original se trata de resolver un sistema de ecuaciones lineales, ya vimos que lo podemos convertir sencillamente en un problema de optimización. Al crear la función g sumando al cuadrado todas las ecuaciones del sistema, hallamos la solución del mismo cuando buscamos el vector que minimiza g.

4.3.1. Método de Newton para problemas de optimización

- Recordemos el método de Newton para sistemas de ecuaciones no lineales:

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - [\mathbf{J}(\mathbf{x}^{(k-1)})]^{-1}\mathbf{F}(\mathbf{x}^{(k-1)}) \qquad k \geq 1$$

- Queremos adaptarlo para resolver problemas de optimización.
- **Objetivo**: encontrar el vector $\mathbf{x}$ que minimiza la función $g(\mathbf{x})$.
- Sabemos que para encontrar un extremo debemos derivar la función con respecto a cada variable, igualar a cero y resolver el sistema de ecuaciones resultante:

$$\begin{cases} \frac{\partial g}{\partial x_1}(\mathbf{x}) = 0 \\ \frac{\partial g}{\partial x_2}(\mathbf{x}) = 0 \\ \vdots \\ \frac{\partial g}{\partial x_n}(\mathbf{x}) = 0 \end{cases}$$

- Haciendo uso de la definición de *gradiente*, podemos simplificar la escritura del sistema anterior:

Definición: Para $g : \mathbb{R}^n \to \mathbb{R}$ el gradiente de g en $\mathbf{x} = (x_1, \cdots, x_n)^T$ se denota $\nabla g(\mathbf{x})$ y se define como:

$$\nabla g(\mathbf{x}) = \Big(\frac{\partial g}{\partial x_1}(\mathbf{x}), \frac{\partial g}{\partial x_2}(\mathbf{x}), \cdots, \frac{\partial g}{\partial x_n}(\mathbf{x})\Big)^T$$

- El gradiente de una función multivariable es análogo a la derivada de una función de una sola variable. Recordemos que la derivada de una función mide la rapidez con la que cambia el valor de dicha función, según cambie el valor de su variable independiente.
- El gradiente es una generalización de esta idea para funciones de más de una variable. De hecho, el término *gradiente* proviene de la palabra latina *gradi* que significa "caminar". En este sentido, el gradiente de una superficie indica la dirección hacia la cual habría que caminar para ir "hacia arriba" lo más rápido posible. Por el contrario, el opuesto del gradiente $\nabla g(\mathbf{x})$ indica la dirección hacia la cual se puede "bajar" lo más rápido posible.
- Podemos interpretar que un gradiente mide cuánto cambia el *output* de una función cuando sus *inputs* cambian un poquito.
- Volviendo al sistema que tenemos que resolver, lo podemos escribir así:

$$\nabla g(\mathbf{x}) = \mathbf{0}$$

- Y el método de Newton visto antes, ahora queda así:

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - [\mathbf{H}(\mathbf{x}^{(k-1)})]^{-1}\nabla g(\mathbf{x}^{(k-1)}) \qquad k \geq 1$$

- Por un lado, el lugar del vector de funciones $\mathbf{F}$ es ocupado por el gradiente ∇g.

- Por otro lado, también tuvimos que reemplazar la matriz jacobiana **J** que contenía las derivadas parciales de **F** por una matriz con las derivadas parciales del gradiente ∇g, es decir, por una matriz que contiene las derivadas parciales segundas de g.
- Esta matriz se simboliza con **H** y se llama **matriz hessiana**.

Definición: Para $g : \mathbb{R}^n \to \mathbb{R}$ cuyas segundas derivadas parciales existen y son continuas sobre el dominio de la función, la **matriz hessiana** de g denotada por $\mathbf{H}(\mathbf{x})$ es una matriz cuadrada $n \times n$ con elementos:

$$h_{ij} = \frac{\partial^2 g}{\partial x_i \partial x_j}$$

es decir:

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 g}{\partial^2 x_1}(\mathbf{x}) & \frac{\partial^2 g}{\partial x_1 \partial x_2}(\mathbf{x}) & \cdots & \frac{\partial^2 g}{\partial x_1 \partial x_n}(\mathbf{x}) \\ \frac{\partial^2 g}{\partial x_2 \partial x_1}(\mathbf{x}) & \frac{\partial^2 g}{\partial^2 x_2}(\mathbf{x}) & \cdots & \frac{\partial^2 g}{\partial x_2 \partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 g}{\partial x_n \partial x_1}(\mathbf{x}) & \frac{\partial^2 g}{\partial x_n \partial x_2}(\mathbf{x}) & \cdots & \frac{\partial^2 g}{\partial^2 x_n}(\mathbf{x}) \end{bmatrix}$$

- En otras palabras, la matriz hessiana es la matriz jacobiana del vector gradiente.
- Si g es una función convexa[1], **H** es semidefinida positiva[2] y el método de Newton nos permite encontrar un mínimo (absoluto o local).
- Si g es una función cóncava, **H** es semidefinida positiva y el método de Newton nos permite encontrar un mínimo (absoluto o local).
- No nos interesa ahora recordar estas cuestiones, pero sólo vamos a mencionar que en la mayoría de las aplicaciones de este método para el análisis de datos se trabaja con funciones convexas donde el objetivo es encontrar un mínimo.

4.3.2. Técnicas del gradiente descendiente

- Miremos de nuevo la fórmula de Newton:

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - [\mathbf{H}(\mathbf{x}^{(k-1)})]^{-1} \nabla g(\mathbf{x}^{(k-1)})$$

- Recordamos que la ventaja de este método es su velocidad de convergencia una vez que se conoce una aproximación inicial suficientemente exacta.
- Pero sus desventajas incluyen la necesidad una aproximación inicial precisa para garantizar la convergencia y de tener que recalcular $[\mathbf{H}(\mathbf{x})]^{-1}$ en cada paso.
- ¿Con qué se podría reemplazar $[\mathbf{H}(\mathbf{x})]^{-1}$?

[1] No es necesario, pero si querés recordar la diferencia entre funciones convexas y cóncavas, podés visitar este enlace.

[2] No es necesario, pero si querés recordar qué son las matrices definidas o semidefinidas positivas o negativas podés visitar este enlace.

- Siendo que para actualizar $\mathbf{x}$ en cada paso se le resta $[\mathbf{H}(\mathbf{x})]^{-1}\nabla g(\mathbf{x})$, podemos pensar que los elementos de la matriz $[\mathbf{H}(\mathbf{x})]^{-1}$ no son más que un conjunto de *coeficientes* o *pesos* que regulan "cuánto" hay que restarle a $\mathbf{x}^{(k-1)}$ para generar el siguiente vector $\mathbf{x}^{(k)}$.
- Entonces una propuesta es reemplazarlos por otro conjunto de pesos que sean más fáciles de obtener, aunque tal vez no lleguen a la convergencia tan rápido como los que fueron deducidos gracias al desarrollo en serie de Taylor.
- ¿Cuál sería la forma más fácil de obtener pesos para este proceso iterativo? Algunas ideas:
  a. ¡No usar nada! Es decir: $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - \nabla g(\mathbf{x}^{(k-1)})$
  b. Elegirlos "a mano" y setearlos como si fuesen *hiper-parámetros* a configurar en el método.
  c. En lugar de usar una matriz de pesos, usar sólo un número real.
- Vamos a ir por la última idea y vamos a llamar α a ese único escalar que utilizaremos como peso, lo que da lugar a una técnica se conoce como **método del gradiente descendiente** o **descenso de gradiente** (conocido en inglés como *gradient descent*).
- Aunque persigue la misma idea, no es considerado un método cuasi-Newton, porque no usa derivadas segundas (no es de segundo orden).
- Este método es ampliamente utilizado para poder ajustar modelos, desde casos sencillos como los modelos de regresión lineal, hasta otros más sofisticados que suelen englobarse bajo el campo del *machine learning* como las redes neuronales. En todos los casos, para ajustar el modelo es necesario minimizar alguna función de pérdida.
- El proceso iterativo resulta ser igual a:

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - \alpha\nabla g(\mathbf{x}^{(k-1)}) \qquad k \geq 1$$

- El opuesto del gradiente $-\nabla g(\mathbf{x})$ nos indica la dirección hacia la cual hay que ir para "bajar" lo más rápido posible por la superficie de g cuando estamos parados en $\mathbf{x}$, pero la constante α es la que determina cuánto vamos a "avanzar" en esa dirección. Por eso, recibe el nombre de **tasa de aprendizaje**.

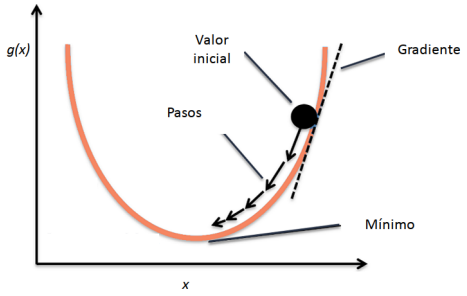


Figure 4.1: Representación del método del gradiente descendiente si g fuese univarida. Notar que es una idea semejante a la que estudiamos con Newton-Rahpson para resolver ecuaciones no lineales.

- Si α es muy pequeña, este procedimiento tardará mucho en encontrar la solución adecuada, pero si es muy grande puede que no se llegue al mínimo porque el algoritmo podría ir y venir por las "laderas" de la superficie.

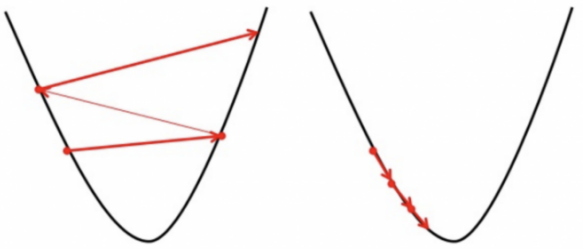


Figure 4.2: Tasa de aprendizaje grande (izquierda) y tasa de aprendizaje pequeña (derecha)

- El **método del descenso más rápido** (*steepest descent*) es un caso especial de gradiente descendiente en el cual la tasa de aprendizaje α se elige en cada paso de forma que se minimice el valor de la función objetivo g en el próximo vector de la sucesión:

$$\alpha \quad / \quad g(\mathbf{x}^{(k-1)} - \alpha \nabla g(\mathbf{x}^{(k-1)})) \quad \text{sea mínimo}$$

- Este método está presentado en la sección 10.4 del libro (pero no lo vamos a estudiar).

Analogía

- En muchos textos se explica la idea general de la técnica del gradiente descendiente con la siguiente analogía.
- Imaginemos que una persona está perdida en una montaña y está tratando de descender (es decir, está buscando el mínimo global de esa superficie). Hay mucha niebla y por lo tanto la visibilidad es muy baja. No se ve a lo lejos y la persona tiene que decidir hacia dónde ir mirando sólo a su alrededor, en su posición actual.
- Lo que puede hacer es usar el método del gradiente descendiente: mirar la pendiente alrededor de dónde está y avanzar hacia la dirección con la mayor pendiente hacia abajo. Repitiendo este procedimiento a cada rato, eventualmente llegará a la base de la montaña (o a un valle intermedio…).
- También podemos suponer que no es fácil determinar a simple vista hacia qué dirección desde donde está hay una pendiente más empinada y para definirlo necesita usar algún instrumento sofisticado que capte la inclinación del piso. Claramente, la persona no puede medir a cada rato porque si no va a perder mucho tiempo usando ese instrumento y avanzará de forma muy lenta. Tampoco puede "recalcular" su dirección poco frecuentemente porque podría caminar mucho en una dirección equivocada. Tiene que darse cuenta la frecuencia adecuada para hacer las mediciones si quiere descender antes de que anochezca.
- En esta analogía, la persona es el algoritmo, la inclinación del terreno representa la pendiente de la superficie de la función a minimizar en el punto donde está parado, el instrumento para medir esa inclinación es la diferenciación, la dirección que elige para avanzar es la que determina el opuesto del gradiente y la frecuencia con la que hace las mediciones es la tasa de aprendizaje.

Mínimos locales

- Cuando usamos el gradiente descendiente nos arriesgamos a caer en un mínimo local.
- Para evitarlo, se desarrollaron varias estrategias.
- Una muy popular por su simplicidad es la de empezar el gradiente descendiente en distintos puntos al azar y elegir la mejor solución.
- Sin embargo, se ha comprobado que, en la práctica, el riesgo de caer en un mínimo local es muy bajo, al menos en los problemas de ajuste de modelos con muchas variables.

Otras versiones

- La versión del gradiente descendiente que hemos visto es la más básica.

- Hay varias adaptaciones que se han ido realizando a lo largo de los años: gradiente descendiente estocástico, momentum, AdaGrad, RMSProp, Adam, batch, mini-batch, etc.
- Algunas de estas mejoras hacen que el elegir el ratio de aprendizaje adecuado no sea tan relevante ya que lo van adaptando sobre la marcha.
- En general, todas estas técnicas del gradiente descendiente resuelven un problema de minimización con una convergencia más lenta (lineal) que la de Newton pero con la ventaja de que normalmente también convergen con aproximaciones iniciales pobres y en ocasiones se lo usa para encontrar aproximaciones iniciales suficientemente exactas para las técnicas con base en Newton.

4.3.3. Fisher Scoring

- En Estadística los problemas de optimización suelen aparecer en el ajuste de modelos. Por ejemplo, mediante el enfoque máximo verosímil, los estimadores de los parámetros de un modelo son aquellos valores que maximizan la log-verosimilitud de la muestra ($logL$) y se emplea el método de Newton para obtenerlos.
- Como dijimos antes, la desventaja del método de Newton es tener que calcular e invertir en cada paso una matriz que involucra derivadas, es decir, la matriz hessiana en la formulación que estamos discutiendo.
- En el contexto de los Modelos Lineales Generalizados (MLG, una familia muy amplia de modelos que incluye a los modelos lineales que ya conocen y que tiene su propia asignatura en la carrera), la matriz hessiana resulta ser igual al opuesto de la *matriz de información observada* (es decir, $\mathbf{H} = -\mathbf{I}$, pero no importa si no recordamos ahora estos conceptos de Inferencia).
- Un pequeño cambio que simplifica los cálculos es reemplazarla por su esperanza, que es la *matriz de información de Fisher*, $\mathcal{I}$ (es decir se reemplaza $\mathbf{H}^{-1}$ por $-\mathcal{I}^{-1}$).
- A esta modificación del método de Newton se la conoce como **Fisher Scoring** y si bien ahora no lo vamos a usar, en MLG les van a preguntar si lo conocen (¡y esperamos que se acuerden que sí!).
- Otras características que ahora puede que no entendamos pero que si volvemos a leer esto en el futuro tendrán más sentido, incluyen:
  - La fórmula recursiva de Fisher Scoring se puede expresar como las ecuaciones normales de una regresión ponderada, por lo tanto a este procedimiento de ajuste también puede ser visto como un caso de *Mínimos Cuadrados Iterativamente Ponderados*.
  - Cuando en los MLG se usa algo que se llama *enlace canónico*, Fisher Scoring coincide exactamente con el método original de Newton.

5 Valores y vectores propios

5.1. Introducción

- Una matriz $m \times n$ se puede considerar como una función que utiliza multiplicación de matrices para transformar vectores columna n-dimensionales en vectores columna m-dimensional.
- Por eso, toda matriz $\mathbf{A}$ de dimensión $m \times n$ puede ser pensada como una transformación lineal de $\mathbb{R}^n$ a $\mathbb{R}^m$:

$$T : \mathbb{R}^n \to \mathbb{R}^m \quad | \quad T(\mathbf{x}) = \mathbf{A}\mathbf{x}$$

- Nos va a interesar de manera particular los casos donde esta función está definida por una matriz $\mathbf{A}$ cuadrada (de dimensión $n \times n$), con lo cual la transformación $\mathbf{A}\mathbf{x}$ toma un vector en $\mathbb{R}^n$ y devuelve otro en $\mathbb{R}^n$.
- Ahora bien, en general no es muy intuitivo saber qué tipo de cambios va a sufrir un vector $\mathbf{x}$ si lo premultiplicamos por $\mathbf{A}$.
- Pero hay ciertos vectores que se modifican de una manera muy sencilla: lo único que hace la matriz $\mathbf{A}$ es "estirarlos" o "comprimirlos". Es decir, puede cambiar su módulo o sentido, pero no su dirección.
- Expresado matemáticamente, para algunos vectores, la transformación $\mathbf{A}\mathbf{x}$ da por resultado el mismo vector $\mathbf{x}$, multiplicado por una constante no nula λ: $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$.
- Estos vectores que "cambian poco" cuando se los transforma mediante la matriz $\mathbf{A}$ reciben el nombre de *autovectores*, *vectores propios* o *eigenvectores* de matriz $\mathbf{A}$.

Definición: Un **autovector** de una matriz $\mathbf{A}$ es cualquier vector $\mathbf{x}$ para el que sólo cambia su escala cuando se lo multiplica con $\mathbf{A}$, es decir: $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$, para algún número λ real o complejo, que recibe el nombre de **autovalor**. En otras palabras:

$$\mathbf{x} \text{ es un autovector y } \lambda \text{ es un autovalor de } \mathbf{A} \iff \mathbf{A}\mathbf{x} = \lambda\mathbf{x}, \quad \mathbf{x} \neq \mathbf{0}, \quad \lambda \in \mathbb{C}$$

- Los autovalores y autovectores son muy importantes en muchas disciplinas, ya que los objetos que se estudian suelen ser representados con vectores y las operaciones que se hacen sobre ellos, con matrices.
- Entonces si una matriz $\mathbf{A}$ describe algún tipo de sistema u operación, los autovectores son aquellos vectores que, cuando pasan por el sistema, se modifican en una forma muy sencilla.
- Por ejemplo, si la matriz $\mathbf{A}$ representa transformaciones en $\mathbb{R}^2$, en principio $\mathbf{A}$ podría estirar y rotar a los vectores. Sin embargo, a sus autovectores lo único que puede hacerles

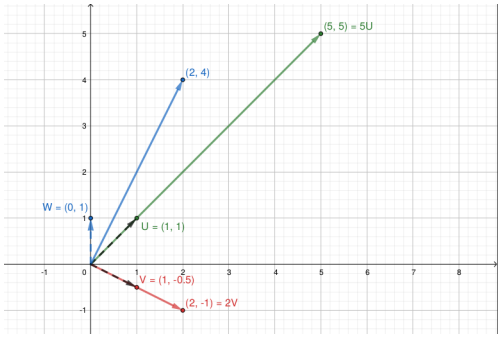
es estirarlos, no rotarlos.

- Veamos un caso concreto:

$$\mathbf{A} = \begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix} \qquad \mathbf{u} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \mathbf{v} = \begin{bmatrix} 1 \\ -0.5 \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- En este gráfico podemos ver los vectores antes de transformarlos (premultiplicarlos) mediante $\mathbf{A}$:

- Y en este gráfico podemos ver como quedan luego de la transformación:



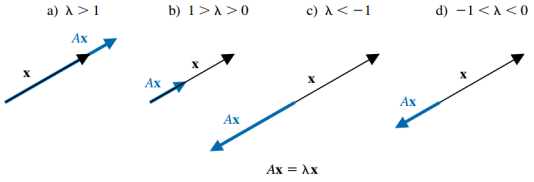
- $\mathbf{u}$ y $\mathbf{v}$ no cambiaron su dirección, sólo su norma: son **autovectores** de $\mathbf{A}$, asociados a los autovalores 5 y 2.
- En cambio, la matriz $\mathbf{A}$ modificó la dirección de $\mathbf{w}$, entonces $\mathbf{w}$ no es un autovector de $\mathbf{A}$.
- Haciendo los cálculos:

$$\mathbf{Au} = \begin{bmatrix} 5 \\ 5 \end{bmatrix} = 5\mathbf{u} \qquad \mathbf{Av} = \begin{bmatrix} 2 \\ -1 \end{bmatrix} = 2\mathbf{v} \qquad \mathbf{Aw} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

- De forma general, si $\mathbf{x}$ es un autovector asociado con el autovalor real λ, entonces $\mathbf{Ax} = \lambda\mathbf{x}$, por lo que la matriz $\mathbf{A}$ transforma al vector $\mathbf{x}$ en un múltiplo escalar de sí mismo, con las siguientes opciones:

  a) Si $\lambda > 1$, entonces $\mathbf{A}$ tiene el efecto de expandir $\mathbf{x}$ en un factor de λ.
  b) Si $0 < \lambda < 1$, entonces $\mathbf{A}$ comprime $\mathbf{x}$ en un factor de λ.

c) Si $\lambda < 0$, los efectos son similares, pero el sentido de $\mathbf{Ax}$ se invierte.



5.1.1. Propiedades

- Se debe observar que si $\mathbf{x}$ es un autovector asociado con el autovalor λ y α es cualquier constante diferente de cero, entonces $\alpha\mathbf{x}$ también es un autovector asociado con el mismo autovalor ya que:

$$\mathbf{A}(\alpha\mathbf{x}) = \alpha(\mathbf{Ax}) = \alpha(\lambda\mathbf{x}) = \lambda(\alpha\mathbf{x})$$

- En el ejemplo anterior vimos que $\mathbf{u} = (1,1)^T$ es un autovector de $\mathbf{u}$ asociado al autovalor $\lambda = 5$. Pero también lo es, por ejemplo, $\mathbf{z} = 2\mathbf{u} = (2,2)^T$, ya que $\mathbf{Az} = (10,10)^T = 5(2,2)^T = 5\mathbf{z}$.
- Si bien hay infinitos autovectores asociados a un autovalor, para todos los autovalores y usando cualquier norma vectorial $||.||$, siempre existe un autovector de norma 1, el cual puede ser hallado a partir de cualquier autovector $\mathbf{x}$ como $\alpha\mathbf{x}$, con $\alpha = ||\mathbf{x}||^{-1}$.
- Dada una matriz $\mathbf{A}$ cuadradada de orden n:
  - $\mathbf{A}$ tiene n autovalores, $\lambda_1, \lambda_2, \cdots, \lambda_n$, los cuales no necesariamente son todos distintos. Si lo son, los autovectores forman un conjunto linealmente independiente.
  - $tr(A) = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i$.
  - $\det(A) = \prod_{i=1}^{n} \lambda_i$.
  - Los autovalores de $\mathbf{A}^k$ son $\lambda_1^k, \lambda_2^k, \cdots, \lambda_n^k$.
  - Si $\mathbf{A}$ es real y simétrica todos sus autovalores son reales y los autovectores correspondientes a distintos autovalores son ortogonales.
  - Si $\mathbf{A}$ es triangular los valores propios son los elementos diagonales.
  - Los autovalores de una matriz y su transpuesta son los mismos.
  - Si $\mathbf{A}$ tiene inversa, los autovalores de $\mathbf{A}^{-1}$ son $1/\lambda_1, 1/\lambda_2, \cdots, 1/\lambda_n$.
  - Los autovalores de $\alpha\mathbf{A}$ son $\alpha\lambda_1, \alpha\lambda_2, \cdots, \alpha\lambda_n$, $\alpha \in \mathbb{R}$.
  - Dos matrices cuadradas $\mathbf{A}$ y $\mathbf{B}$ son *semejantes* o *similares* si existe una matriz invertible $\mathbf{Q}$ tal que $\mathbf{B} = \mathbf{Q}^{-1}\mathbf{AQ}$. Las matrices semejantes tienen los mismos autovalores.

5.1.2. Obtención de autovalores y autovectores

- Como estudiarán en Álgebra Lineal, para hallar autovalores y autovectores se deben seguir los siguientes dos pasos:

1. Se determinan los autovalores encontrando las soluciones de la ecuación algebraica de grado n: $det(\mathbf{A} - \lambda\mathbf{I}) = 0$ (la incógnita es λ).
2. Para cada autovalor λ, se determina un autovector al resolver el sistema lineal $n \times n$: $(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}$.

- Estos pasos son el resultado de las siguientes consideraciones:

a. A partir de la definición tenemos: $\mathbf{Ax} = \lambda\mathbf{x} \implies \mathbf{Ax} - \lambda\mathbf{x} = \mathbf{0} \implies (\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}$.
b. Esto es un sistema de ecuaciones lineales con matriz de coeficientes $\mathbf{A} - \lambda\mathbf{I}$, vector de incógnitas $\mathbf{x}$ (el autovector) y vector de términos independientes $\mathbf{0}$. Es decir, es un **sistema homogéneo**.
c. Un sistema homogéneo es siempre compatible, ya que al menos tiene la solución trivial $\mathbf{x} = (0, \cdots, 0)^T$. Esta solución no nos interesa, puesto que buscamos autovectores y los mismos deben ser no nulos.
d. Como sabemos, para que el sistema tenga otra solución además de la trivial, se tiene que tratar de un sistema indeterminado, con infinitas soluciones, ya que los sistemas compatibles o bien tienen una sola solución o infinitas. Esto tiene sentido, porque cada autovalor λ tiene asociados infinitos autovectores. Entonces, para hallar autovectores necesitamos que el sistema $(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}$ sea compatible indeterminado.
e. Para que un sistema sea indeterminado, su matriz de coeficientes debe tener determinante igual a 0, es decir: $det(\mathbf{A} - \lambda\mathbf{I}) = 0$.
f. Por eso sabemos que los autovalores de $\mathbf{A}$ tienen que ser aquellos valores λ que satisfagan la igualdad anterior, que es una ecuación algebraica en λ de grado n. $det(\mathbf{A} - \lambda\mathbf{I})$ recibe el nombre de **polinomio característico** de $\mathbf{A}$.

- **Ejemplo**:

$$\mathbf{A} = \begin{bmatrix} 5 & -2 & 0 \\ -2 & 3 & -1 \\ 0 & -1 & 1 \end{bmatrix} \implies$$

$$det(\mathbf{A} - \lambda\mathbf{I}) = \begin{vmatrix} 5-\lambda & -2 & 0 \\ -2 & 3-\lambda & -1 \\ 0 & -1 & 1-\lambda \end{vmatrix} = \cdots = -\lambda^3 + 9\lambda^2 - 18\lambda + 6 = 0$$

- Como pueden verificar ustedes (opcionalmente aplicando los métodos de la Unidad 2), las soluciones de la ecuación característica son $\lambda_1 = 6.2899, \lambda_2 = 2.2943$ y $\lambda_3 = 0.4158$, los cuales son los autovalores de $\mathbf{A}$.
- Para hallar un autovector asociado a $\lambda_1 = 6.2899$, resolvemos el sistema de ecuaciones $(\mathbf{A} - 6.2899\,\mathbf{I})\mathbf{x} = \mathbf{0}$:

$$(\mathbf{A} - 6.2899\,\mathbf{I})\mathbf{x} = \mathbf{0} \implies \begin{bmatrix} -1.2899 & -2 & 0 \\ -2 & -3.2899 & -1 \\ 0 & -1 & -5.2899 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\implies \begin{cases} -1.2899x_1 - 2x_2 & = 0 \\ -2x_1 - 3.2899x_2 - x_3 & = 0 \\ -x_2 - 5.2899x_3 & = 0 \end{cases} \implies \begin{cases} x_1 = 8.2018x_3 \\ x_2 = -5.2899x_2 \\ x_3 \in \mathbb{R} \end{cases}$$

- Como se puede ver la solución de este sistema homogéneo no es única, representando los infinitos autovectores asociados a $\lambda_1 = 6.2899$. Por ejemplo, si elegimos $x_3 = 1$, obtenemos el autovector:

$$\mathbf{x}_1 = \begin{bmatrix} 8.2018 \\ -5.2899 \\ 1 \end{bmatrix}$$

- En general, se acostumbra a informar el autovector de norma 1 (que sí es único).
- De la misma forma se procede con los restantes autovalores λ_2 y λ_3.
- Hallar la ecuación característica ya es demasiado trabajoso para $n = 3$, y mucho más será para mayor n. Ni hablar de resolver el sistema para encontrar los autovectores.
- Por eso en esta unidad veremos métodos que directamente nos dan como resultados los autovectores y autovalores de una matriz.
- Por supuesto, Python trae una función para esto. Podemos usarla para chequear los resultados, pero no porque sea fácil emplearla nos libraremos de estudiar los algoritmos encargados de producir nuestros queridos autovectores y autovalores:

```
import numpy as np
from scipy.linalg import eig

A = np.array([[ 5, -2, 0],
              [-2, 3, -1],
              [ 0, -1, 1]])

autovalores, autovectores = eig(A)
print("Autovalores:")
```

```
Autovalores:
```

```
print(autovalores)
```

```
[6.2899+0.j 2.2943+0.j 0.4158+0.j]
```

```
print("\nAutovectores:")
```

```
Autovectores:
```

```
print(autovectores)
```

```
[[ 0.836   0.5049  0.2149]
 [-0.5392  0.6831  0.4927]
 [ 0.1019 -0.5277  0.8433]]
```

5.2. El Método de Potencia

- El **método de potencia** (también conocido como *de las potencias* o *de aproximaciones sucesivas*) es una técnica iterativa que se usa para determinar el autovalor dominante de una matriz y un autovector asociado.

Definición: sean $\lambda_1, \lambda_2, \cdots, \lambda_n$ los autovalores de una matriz $n \times n$, $\mathbf{A}$. λ_1 es llamado **autovalor dominante** de $\mathbf{A}$ si:

$$|\lambda_1| > |\lambda_i|, \quad i = 2, \cdots, n$$

Los autovectores correspondientes a λ_1 se llaman **autovectores dominantes** de $\mathbf{A}$.

- En primer lugar, se debe tomar un vector inicial $\mathbf{x}^{(0)}$ con norma $||\mathbf{x}^{(0)}||_\infty = 1$.
- Por ejemplo, para $n = 3$ puede ser $\mathbf{x}^{(0)} = (1, 1, 1)^T$ o $\mathbf{x}^{(0)} = (1, 0, 0)^T$, entre otros.
- Luego, para cada $k = 1, 2, \cdots$ se da lugar al siguiente proceso iterativo:
  1. Calcular $\mathbf{y}^{(k)} = \mathbf{A}\mathbf{x}^{(k-1)}$.
  2. Determinar $\mu^{(k)}$ como la coordenada de mayor valor absoluto en $\mathbf{y}^{(k)}$.
     Es decir, tomar $\mu^{(k)} / |\mu^{(k)}| = ||\mathbf{y}^{(k)}||_\infty$. Si hay varias coordenadas que cumplen con esta característica, tomar la primera.
  3. Calcular: $\mathbf{x}^{(k)} = \frac{\mathbf{y}^{(k)}}{\mu^{(k)}}$
- De esta forma, la sucesión $\{\mu^{(k)}\}_{k=0}^{\infty}$ converge al autovalor dominante de $\mathbf{A}$, mientras que la sucesión $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$ converge a un autovector asociado de norma $L_\infty = 1$.
- La deducción y justificación de este método puede leerse opcionalmente en las páginas 432-433 del libro.
- Retomando el ejemplo de la sección anterior, vamos a aplicar este proceso con:

$$\mathbf{A} = \begin{bmatrix} 5 & -2 & 0 \\ -2 & 3 & -1 \\ 0 & -1 & 1 \end{bmatrix} \qquad \mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

k	$\mathbf{x}^{(k)}$	$\mathbf{y}^{(k)} = \mathbf{A}\mathbf{x}^{(k)}$	$\mu^{(k)}$	$\mathbf{x}^{(k+1)} = \mathbf{y}^{(k)}/\mu^{(k)}$	Error (L_2)
0	$[1\ 1\ 1]^T$	$[3\ 0\ 0]^T$	3	$[1\ 0\ 0]^T$	1.4142
1	$[1\ 0\ 0]^T$	$[5\ -2\ 0]^T$	5	$[1\ -0.4\ 0]^T$	0.4
2	$[1\ -0.4\ 0]^T$	$[5.8\ -3.2\ 0.4]^T$	5.8	$[1\ -0.5517\ 0.0690]^T$	0.1667
3	$[1\ -0.5517\ 0.0690]^T$	$[6.1034\ -3.7241\ 0.6207]^T$	6.1034	$[1\ -0.6102\ 0.1017]^T$	0.0690
4	$[1\ -0.6102\ 0.1017]^T$	$[6.2203\ -3.9322\ 0.7119]^T$	6.2203	$[1\ -0.6322\ 0.1144]^T$	0.0254
...	...	...	...	...	...
16	$[1\ -0.644972\ 0.1219239]^T$	$[6.2899\ -4.0568\ 0.7669]^T$	6.2899	$[1\ -0.644972\ 0.1219241]^T$	3.956E-7

5.2.1. Convergencia

- Para que la convergencia esté garantizada, se deben cumplir las siguientes condiciones:
  a. Los autovalores de $\mathbf{A}$, $\lambda_1, \lambda_2, \cdots, \lambda_n$ están asociados a un conjunto de autovectores linealmente independientes[1].
  b. $\mathbf{A}$ tiene un autovalor dominante, es decir, se verifica: $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n| \geq 0$.
- Si se cumplen estas condiciones, en general el método converge con cualquier vector inicial $\mathbf{x}^{(0)}$[2].
- Si no se cumplen estas condiciones, el método puede converger o fallar.
- Como en la práctica no podemos verificar el cumplimiento de las mismas, sencillamente corremos el método y observamos el resultado.
- Para detener el proceso, podemos usar los mismos criterios vistos en la Unidad 3.
- Con el método así presentado, la convergencia será más rápida cuanto mayor sea el valor absoluto del autovalor dominante $|\lambda_1|$ comparado con el que le sigue, $|\lambda_2|$.
- También es más rápida cuando se aplica en matrices simétricas que en matrices asimétricas.

5.2.2. Otras características

- La división por la coordenada de mayor valor absoluto, $\mu^{(k)}$, produce como resultado en cada paso un vector de norma $L_\infty = 1$. Si no se incluyera esta normalización, el proceso iterativo resultaría igual a:

[1] Esta condición es equivalente a decir que la matriz $\mathbf{A}$ es "diagonalizable".
[2] En realidad $\mathbf{x}^{(0)}$ debe verificar una condición teórica que se puede leer al final de la página 433, pero en la práctica no lo podemos verificar y el método sencillamente funciona.

$$\begin{aligned}
\mathbf{x}^{(1)} &= \mathbf{A}\mathbf{x}^{(0)} \\
\mathbf{x}^{(2)} &= \mathbf{A}\mathbf{x}^{(1)} = \mathbf{A}^2\mathbf{x}^{(0)} \\
\mathbf{x}^{(3)} &= \mathbf{A}\mathbf{x}^{(2)} = \mathbf{A}^3\mathbf{x}^{(0)} \\
&\vdots \\
\mathbf{x}^{(k)} &= \mathbf{A}\mathbf{x}^{(k-1)} = \mathbf{A}^k\mathbf{x}^{(0)} \\
&\vdots
\end{aligned}$$

- Esta sucesión también converge a un autovector dominante, pero no normalizado y no nos entrega el autovalor correspondiente, el cual puede ser calculado mediante el **cociente de Rayleigh** luego de detener el proceso: si $\mathbf{x}$ es un autovector de $\mathbf{A}$, entonces su correspondiente autovalor es:

$$\lambda = \frac{(\mathbf{A}\mathbf{x})^t\mathbf{x}}{\mathbf{x}^t\mathbf{x}}$$

- Sin embargo, las sucesivas potencias de $\mathbf{A}$ tienden a terminar en errores de desbordamiento o subdesbordamiento. Por eso resulta necesaria la introducción de la constante normalizadora, como se indicó inicialmente.

5.2.3. Variantes para acelerar la convergencia

- Se han desarrollado modificaciones del método de potencia que logran una convergencia más rápida y que son importantes en problemas con matrices de gran dimensión.
- En el caso de matrices generales, se pueden aplicar el *método de potencia trasladada* o el *procedimiento de Aitkens*.
- Para matrices simétricas, se puede mejorar significativamente la convergencia con algunas modificaciones en los cálculos, en lo que se conoce como *método de potencia simétrica*.
- No nos detendremos en estas variantes.

5.2.4. Variantes para hallar el autovalor más pequeño

- *Recordatorio*: los autovalores de $\mathbf{A}^{-1}$ son los recíprocos de los de $\mathbf{A}$.
- Si aplicamos el método a $\mathbf{A}^{-1}$, obtenemos su autovalor dominante.
- Y, por la observación anterior, si tomamos el recíproco del autovalor así hallado, obtenemos el autovalor de $\mathbf{A}$ de menor valor absoluto.

5.2.5. Variantes para hallar otros autovalores

Método de potencia inversa

- Es una modificación que se usa para encontrar el autovalor de $\mathbf{A}$ que está más cerca de un número específico que hay que establecer de antemano, q.

- Esto se utiliza en aplicaciones donde q es una aproximación a algún autovalor que se tiene disponible y que se desea mejorar.
- Tampoco profundizaremos en este método, pero se lo puede consultar en las páginas 439-440.

Técnicas de deflación

- Las técnicas de deflación permiten obtener los otros autovalores de la matriz, luego de haber obtenido el dominante con el método de potencia.
- Consisten en formar una nueva matriz $\mathbf{A}_2$ cuyos autovalores sean iguales a los de la matriz original $\mathbf{A}$, excepto por el autovalor dominante de $\mathbf{A}$, que es reemplazado por un autovalor igual a cero en $\mathbf{A}_2$.
- Entre estos algoritmos encontramos a la **deflación de Wielandt** y la **deflación de Hotelling**.
    - La **deflación de Wielandt** se puede utilizar de manera general para cualquier tipo de matriz. Si bien no reviste de demasiada complejidad, involucra numerosos cálculos y no nos detendremos en ello, pero puede ser consultada en la página 443 del libro.
    - La **deflación de Hotelling** se aplica para matrices simétricas. Una vez hallada una aproximación para el autovalor dominante λ_1 con un autovector asociado $\mathbf{x}_1$, se debe calcular la siguiente matriz:

      $$\mathbf{A}_2 = \mathbf{A} - \lambda_1 \mathbf{u}_1 \mathbf{u}_1^T$$

      donde $\mathbf{u}_1 = \mathbf{x}_1 / ||\mathbf{x}_1||_2$ (es decir, $\mathbf{u}_1$ es el autovector asociado a λ_1 de norma euclidiana igual a 1).

      Los autovalores de $\mathbf{A}_2$ son $\{0, \lambda_2, \cdots, \lambda_n\}$, de modo que al aplicar nuevamente el método de potencia sobre $\mathbf{A}_2$ para hallar su autovalor dominante, encontraremos el segundo autovalor de $\mathbf{A}$, λ_2.

      Repitiendo este proceso se pueden encontrar los restantes autovalores (por ejemplo, $\mathbf{A}_3 = \mathbf{A}_2 - \lambda_2 \mathbf{u}_2 \mathbf{u}_2^T$).
- No obstante, se debe tener en cuenta que las técnicas de deflación en general no se aplican para calcular todos los autovalores de una matriz, sino sólo algunos, ya que presentan un grave inconveniente ligado al deterioro de las aproximaciones de los autovalores restantes.
- Dado que el valor obtenido en la primera etapa es una aproximación del verdadero autovalor λ_1, los autovalores de $\mathbf{A}_2$ no son exactamente los restantes autovalores de $\mathbf{A}$ sino una aproximación a los mismos.
- Al aplicar el método otra vez, se obtiene una aproximación al autovalor dominante de $\mathbf{A}_2$, que es a su vez aproximado pero no igual al verdadero valor λ_2 que buscamos.
- Entonces, tras un cierto número de etapas de deflación, la acumulación de errores de redondeo y de truncamiento pueden deteriorar notablemente la aproximación.
- Por esta razón, si es necesario encontrar todos los autovalores de una matriz, es conveniente emplear otras técnicas, como la del algoritmo QR que veremos en la siguiente

sección.

5.2.6. Importancia del método

- Si hay otras técnicas que hallan todos los autovalores, ¿por qué nos preocupamos por el método de potencia que nos da sólo uno?
  - Porque hallar todos los autovalores en matriz de gran dimensión es computacionalmente costoso.
  - Porque es utilizado en muchas aplicaciones donde sólo se necesita obtener el autovalor dominante.
  - Porque es eficiente cuando la matriz es dispersa (matriz de gran dimensión con la gran mayoría de sus entradas iguales a cero).
- De hecho, Google utiliza el método de potencia en su algoritmo *PageRank* para buscar rankear los resultados de búsquedas de páginas web, desarrollado en Stanford University en 1996 por Larry Page y Sergey Brin. El exito de este algoritmo derivó en la creación de esta mega empresa que empezó siendo sólo un motor de búsqueda (pueden buscar en Wikipedia o leer el artículo The $25.000.000.000 eigenvector: the linear algebra behind Google).
- Twitter también lo usa para generar las recomendaciones acerca de a quién seguir.

5.3. El algoritmo QR

- En esta sección consideramos el **algoritmo QR**, una técnica que se utiliza para determinar en forma sistemática todos los autovalores de una matriz cuadrada.
- Primero vamos a ver de qué se trata la **factorización QR** y luego veremos el **algoritmo QR** para hallar los autovalores.

5.3.1. Factorización QR

- Ya hemos mencionado un tipo especial de factorización de matrices, la **LU**.
- Ahora vamos a ver otra factorización, que también tiene numerosas aplicaciones:
  - Resolver sistemas de ecuaciones lineales.
  - Calcular determinantes e inversas.
  - Encontrar otras factorizacones (como la de Cholesky y la de Schur).
  - Otras.

Teorema::

- Toda matriz real cuadrada no singular $\mathbf{A}$ de dimensión $n \times n$ puede factorizarse en la forma $\mathbf{A} = \mathbf{QR}$, donde $\mathbf{Q}$ es una matriz ortogonal $n \times n$ y $\mathbf{R}$ es una matriz triangular superior $n \times n$. La factorización es única si se pide que los elementos diagonales de $\mathbf{R}$ sean positivos.

- Toda matriz real rectangular $\mathbf{A}$ de dimensión $m \times n$ $(m > n)$, puede factorizarse en la forma $\mathbf{A} = \mathbf{QR}$, donde $\mathbf{Q}$ es una matriz ortogonal $m \times m$ y $\mathbf{R}$ es una matriz triangular superior $m \times n$, en la cual sus últimas $m - n$ filas son todos ceros. Dado que las últimas filas son nulas, las últimas columnas de $\mathbf{Q}$ no aportan al producto $\mathbf{QR}$ y por lo tanto otras definiciones y algunos algoritmos presentan a $\mathbf{Q}$ como una matriz $m \times n$ con columnas ortonormales y $\mathbf{R}$ como una matriz triangular $n \times n$.
- Recordamos que una matriz ortogonal es una matriz cuadrada cuya matriz inversa coincide con su matriz traspuesta: $\mathbf{Q}^T = \mathbf{Q}^{-1}$. Sus columnas son vectores ortogonales de norma 1.
- Para obtener $\mathbf{Q}$ se puede aplicar el proceso de Gram-Schmidt a las columnas de $\mathbf{A}$ (las columnas de $\mathbf{Q}$ son las de $\mathbf{A}$ luego de la ortonormalización).
- Una vez obtenida $\mathbf{Q}$, $\mathbf{R}$ se puede obtener como $\mathbf{R} = \mathbf{Q}^T\mathbf{A}$.
- Hay otros métodos que también permiten hacer esto, pero en este curso no nos vamos a preocupar por el cálculo de la factorización y directamente emplearemos la función de R con la que se obtiene.
- Ejemplo con una matriz cuadrada. Sea:

$$\mathbf{A} = \begin{bmatrix} 1 & -2 & 1 \\ -1 & 3 & 2 \\ 1 & -1 & -4 \end{bmatrix}$$

  Su factorización QR es:

$$\mathbf{Q} = \begin{bmatrix} -\frac{1}{\sqrt{3}} & 0 & -\frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} \\ -\frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} \end{bmatrix} \qquad \mathbf{R} = \begin{bmatrix} -\sqrt{3} & 2\sqrt{3} & \frac{5\sqrt{3}}{3} \\ 0 & -\sqrt{2} & \sqrt{2} \\ 0 & 0 & -\frac{\sqrt{96}}{3} \end{bmatrix}$$

  de modo que se verifica: $\mathbf{A} = \mathbf{QR}$.
- Lo comprobamos en Python.
- **Ejemplo con una matriz cuadrada.**

```
from scipy.linalg import qr

A = np.array([[1, -2, 1],
              [-1, 3, 2],
              [1, -1, -4]])

Q, R = qr(A)

print("Matriz Q (ortogonal):")
```

```
Matriz Q (ortogonal):
```

```
print(Q)
```

```
[[-0.5774  0.     -0.8165]
 [ 0.5774 -0.7071 -0.4082]
 [-0.5774 -0.7071  0.4082]]
```

```
print("\nMatriz R (triangular superior):")
```

```
Matriz R (triangular superior):
```

```
print(R)
```

```
[[-1.7321  3.4641  2.8868]
 [ 0.     -1.4142  1.4142]
 [ 0.      0.     -3.266 ]]
```

```
# Verificamos que Q es ortogonal
Q_traspuesta = np.transpose(Q)
Q_inversa = np.linalg.inv(Q)
print("\n¿Q es ortogonal? (t(Q) == inv(Q)):", np.allclose(Q_traspuesta, Q_inversa))
```

```
¿Q es ortogonal? (t(Q) == inv(Q)): True
```

```
# Verificamos A = QR
resultado = np.dot(Q, R)
print(resultado)
```

```
[[ 1. -2.  1.]
 [-1.  3.  2.]
 [ 1. -1. -4.]]
```

```
print("\n¿A = QR?", np.allclose(A, resultado))
```

```
¿A = QR? True
```

- **Ejemplo con una matriz rectangular:**

```
A = np.array([[1, -2],
              [-1, 3],
              [1, -1]])

# Forma 1
Q, R = qr(A)
```

```
print("Matriz Q (ortogonal):")
```

```
Matriz Q (ortogonal):
```

```
print(Q)
```

```
[[-0.5774  0.     -0.8165]
 [ 0.5774 -0.7071 -0.4082]
 [-0.5774 -0.7071  0.4082]]
```

```
print("\nMatriz R (triangular superior):")
```

```

Matriz R (triangular superior):
```

```
print(R)
```

```
[[-1.7321  3.4641]
 [ 0.     -1.4142]
 [ 0.      0.    ]]
```

```
# Verificamos que Q es ortogonal
Q_traspuesta = np.transpose(Q)
Q_inversa = np.linalg.inv(Q)
print("\n¿Q es ortogonal? (t(Q) == inv(Q)):", np.allclose(Q_traspuesta, Q_inversa))
```

```

¿Q es ortogonal? (t(Q) == inv(Q)): True
```

```
# Verificamos A = QR
resultado = np.dot(Q, R)
print(resultado)
```

```
[[ 1. -2.]
 [-1.  3.]
 [ 1. -1.]]
```

```
print("\n¿A = QR?", np.allclose(A, resultado))
```

```

¿A = QR? True
```

```
# Forma 2 (omite filas nulas de R)
Q, R = qr(A, mode="economic")
print("Matriz Q (ortogonal):")
```

```
Matriz Q (ortogonal):
```

```
print(Q)
```

```
[[-0.5774  0.    ]
 [ 0.5774 -0.7071]
 [-0.5774 -0.7071]]
```

```
print("\nMatriz R (triangular superior):")
```

```
Matriz R (triangular superior):
```

```
print(R)
```

```
[[-1.7321  3.4641]
 [ 0.     -1.4142]]
```

5.3.2. El algoritmo QR

- Ahora estamos en condiciones de usar la factorización QR para obtener todos los autovalores de una matriz cuadrada $n \times n$, $\mathbf{A}$.
- Es un algoritmo tan sencillo, que sorprende que sea tan efectivo.
- Primero se toma $\mathbf{A} = \mathbf{A}^{(0)}$ como matriz inicial.
- Luego, para cada $k = 1, 2, \cdots$:
  1. Realizar la factorización QR de $\mathbf{A}^{(k-1)}$ para obtener $\mathbf{Q}^{(k-1)}$ y $\mathbf{R}^{(k-1)}$ (es decir: $\mathbf{A}^{(k-1)} = \mathbf{Q}^{(k-1)}\mathbf{R}^{(k-1)}$).
  2. Calcular la siguiente matriz del proceso iterativo como: $\mathbf{A}^{(k)} = \mathbf{R}^{(k-1)}\mathbf{Q}^{(k-1)}$
- La sucesión $\mathbf{A}^{(k)}$ converge a una matriz triangular cuyos elementos diagonales son los autovalores de $\mathbf{A}$.
- La idea detrás de este método es la siguiente: las sucesivas matrices $\mathbf{A}^{(k)}$ son semejantes (revisar sección de propiedades) y, por lo tanto, tienen los mismos autovalores. Además, estas operaciones van transformando de a poco a las matrices $\mathbf{A}^{(k)}$ en triangulares superiores y sabemos que en tales matrices los autovalores son los elementos diagonales (repasar las propiedades enunciadas al inicio del apunte).
- Para darnos cuenta de que las matrices $\mathbf{A}^{(k)}$ son semejantes debemos notar:

$$\mathbf{A}^{(k)} = \mathbf{R}^{(k-1)}\mathbf{Q}^{(k-1)} = \underbrace{{\mathbf{Q}^{(k-1)}}^{-1}\mathbf{Q}^{(k-1)}}_{\mathbf{I}}\mathbf{R}^{(k-1)}\mathbf{Q}^{(k-1)} = {\mathbf{Q}^{(k-1)}}^{-1}\mathbf{A}^{(k-1)}\mathbf{Q}^{(k-1)}, \implies A^{(k)} \text{ y } A^{(k-1)} \text{ son semejante}$$

- ¿Y los autovectores?
  - Si la matriz es simétrica, los autovectores son las columnas de $\prod_{k=0}\mathbf{Q}^{(k)}$.

  - Si la matriz no es simétrica, esta forma presentada del algoritmo, que es la más sencilla posible y por eso a veces es llamado "el algoritmo QR puro" no entrega los autovectores, pero hay otras variantes que sí lo hacen.

- El proceso iterativo debe detenerse cuando se haya llegado a una matriz triangular superior (las entradas del triángulo inferior sin la diagonal deberían ser cero o muy cercanas). Para implementar un criterio más sencillo, podemos detenernos cuando la distancia entre los vectores formados por los elementos diagonales de la matriz sea tan pequeña como se desee.
- Ejemplo:

$$\mathbf{A} = \begin{bmatrix} 1 & -2 & 1 \\ -1 & 3 & 2 \\ 1 & -1 & -4 \end{bmatrix}$$

Llamamos a esta matriz con $\mathbf{A}^{(0)}$ y ya vimos que su factorización QR es:

$$\mathbf{Q}^{(0)} = \begin{bmatrix} -\frac{1}{\sqrt{3}} & 0 & -\frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} \\ -\frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} \end{bmatrix} \qquad \mathbf{R}^{(0)} = \begin{bmatrix} -\sqrt{3} & 2\sqrt{3} & \frac{5\sqrt{3}}{3} \\ 0 & -\sqrt{2} & \sqrt{2} \\ 0 & 0 & -\frac{\sqrt{96}}{3} \end{bmatrix}$$

Con lo cual, la siguiente matriz de la sucesión es:

$$\mathbf{A}^{(1)} = \mathbf{R}^{(0)}\mathbf{Q}^{(0)} = \begin{bmatrix} -\frac{4}{3} & -\frac{11}{6}\sqrt{6} & -\frac{5}{6}\sqrt{2} \\ -\frac{2}{3}\sqrt{6} & 0 & \frac{2}{3}\sqrt{3} \\ \frac{4}{3}\sqrt{2} & \frac{4}{3}\sqrt{3} & -\frac{4}{3} \end{bmatrix}$$

Verificamos en Python este y los siguientes pasos:

```
A = np.array([[1, -2, 1],
              [-1, 3, 2],
              [1, -1, -4]])

# Iteración 1
Q0, R0 = qr(A)
A1 = np.dot(R0, Q0)
print(A1)
```

```
[[ 1.3333 -4.4907  1.1785]
 [-1.633  -0.      1.1547]
 [ 1.8856  2.3094 -1.3333]]
```

```
# Iteración 2
Q1, R1 = qr(A1)
```

```
A2 = np.dot(R1, Q1)
print(A2)
```

```
[[ 1.      2.8772  0.2349]
 [ 4.0856 -1.2914  3.1605]
 [-0.3759  0.3028  0.2914]]
```

```
# Iteración 3
Q2, R2 = qr(A2)
A3 = np.dot(R2, Q2)
print(A3)
```

```
[[ 0.1495 -4.4805 -2.7609]
 [-3.0529 -0.7539 -0.1445]
 [ 0.0542  0.0489  0.6044]]
```

- Si seguimos iterando vamos a ver que la matriz converge y en su diagonal tendremos a los autovalores.
- Usando la función provista que implementa este algoritmo vemos el resultado:

```
rtdo = algoritmo_qr(A)
[print(keys, "\n", value) for keys, value in rtdo.items()]
```

```
convergencia
 True
iteraciones
 115
autovalores
 [-4.      3.4142  0.5858]
pasos
        i                                          autovalores_i       error_i
0      0                                            [1, 3, -4]           NaN
1      1  [1.3333333333333333, -1.300816253288535e-15, -...  4.027682e+00
2      2  [0.999999999999999, -1.2913907284768211, 0.291...  2.102030e+00
3      3  [0.14953271028037157, -0.7539198862276028, 0.6...  1.053630e+00
4      4  [-0.3840000000000031, -0.19584605115074222, 0....  7.724674e-01
..   ...                                                ...           ...
111  111  [-3.999999941718327, 3.4142135040914274, 0.585...  1.789869e-07
112  112  [-4.000000049746487, 3.4142136121195867, 0.585...  1.527749e-07
113  113  [-3.9999999575386904, 3.4142135199117907, 0.58...  1.304015e-07
114  114  [-4.000000036242971, 3.4142135986160715, 0.585...  1.113047e-07
115  115  [-3.9999999690646675, 3.4142135314377686, 0.58...  9.500447e-08

[116 rows x 3 columns]
```

```
[None, None, None, None]
```

- Lo comparamos con el resultado de la función `eig()` de Python:

```
autovalores, autovectores = eig(A)
print("Autovalores:")
```

```
Autovalores:
```

```
print(autovalores)
```

```
[-4.    +0.j  0.5858+0.j  3.4142+0.j]
```

```
print("\nAutovectores:")
```

```
Autovectores:
```

```
print(autovectores)
```

```
[[ 0.3015  0.9511 -0.6715]
 [ 0.3015  0.2711  0.7169]
 [-0.9045  0.1483 -0.1873]]
```

- Al algoritmo QR "puro" definido en esta sección también se lo conoce como "impráctico" porque tiene algunas desventajas:
  - La factorización QR en cada paso es costosa computacionalmente.
  - La convergencia de las entradas subdiagonales a cero es lineal (convergencia lenta).
- Por eso se han propuesto algunas modificaciones que mejoran notablemente el desempeño del método:
  - En primer lugar, se debe transformar a la matriz original $\mathbf{A}$ en otra similar (mismos autovalores) pero que sea tridiagonal (se logra con el método de Householder) o que sea una matriz de Hessenberg.
  - Luego, en el proceso iteratvio, se debe usar un procedimiento de deflación cada vez que un elemento subdiagonal se hace 0 para disminuir la cantidad de cálculos.
  - Y también se debe implementar una estrategia de cambios de filas y columnas (*shifted QR*) que acelera la convergencia.
- En este curso, no veremos estas variantes (están en el libro, que de hecho no presenta la forma simple que vimos acá).

5.4. Descomposición en valores singulares (DVS)

```
from matplotlib.image import imread
import matplotlib.pyplot as plt
```

- Una matriz rectangular $\mathbf{A}$ no puede tener un autovalor porque $\mathbf{Ax}$ y $\mathbf{x}$ son vectores de diferentes tamaños.
- Sin embargo, existen números que desempeñan un rol análogo al de los autovalores para las matrices no cuadradas.
- Se trata de los **valores singulares** de una matriz.
- La **Descomposición en Valores Singulares** (DVS, también llamada *SVD*, por las siglas de *Singular Value Decomposition*) es una factorización para matrices rectangulares que tiene numerosas aplicaciones, por ejemplo en compresión de imágenes y análisis de señales.
- En Estadística tiene gran importancia para tareas relacionadas con la reducción de dimensionalidad de grandes conjuntos de datos (tiene una vinculación directa con el Análisis de Componentes Principales, técnica que estudiarán en Análisis de Datos Multivariados). También se la puede utilizar para realizar ajustes por Mínimos Cuadrados.

Teorema de Descomposición en Valores Singulares: una matriz rectangular $\mathbf{A}$ de dimensión $m \times n$ puede ser factorizada como:

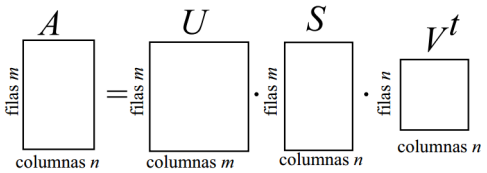
$$\mathbf{A} = \mathbf{USV}^T$$

donde:

- $\mathbf{U}$ es una matriz ortogonal $m \times m$
- $\mathbf{S}$ es una matriz diagonal $m \times n$ con elementos σ_i ($\mathbf{s}_{ij} = 0 \, \forall i \neq j$).
- $\mathbf{V}$ es una matriz ortogonal $n \times n$

Además:

- Los elementos diagonales de $\mathbf{S}$, σ_i, son llamados **valores singulares** de $\mathbf{A}$. Son tales que $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_k \geq 0$, con $k = min\{m, n\}$ y son iguales a las raíces cuadradas positivas de los autovalores no nulos de $\mathbf{A}^T\mathbf{A}$.
- Las columnas de $\mathbf{V}$ son los autovectores ortonormales de $\mathbf{A}^T\mathbf{A}$ y se llaman *vectores singulares derechos* porque $\mathbf{AV} = \mathbf{US}$.
- Las columnas de $\mathbf{U}$ son los autovectores ortonormales de $\mathbf{AA}^T$ y se llaman *vectores singulares izquierdos* porque $\mathbf{U}^T\mathbf{A} = \mathbf{SV}^T$.



- Esas tres últimas observaciones proporcionan una forma de obtener la DVS.

5.4.1. Ejemplo

- Vamos a buscar la DVS de la siguiente matriz haciendo los cálculos en Python:

$$\mathbf{A} = \begin{bmatrix} 4 & 2 & 0 \\ 1 & 5 & 6 \end{bmatrix}$$

```
A = np.array([[4,2,0],
              [1,5,6]])
```

- Para generar la matriz diagonal $\mathbf{S}$, buscamos los valores singulares que son las raíces positivas de los autovalores no nulos de $\mathbf{A}^T\mathbf{A}$.

```
ATA = A.T @ A
print(ATA)
```

```
[[17 13  6]
 [13 29 30]
 [ 6 30 36]]
```

```
autovalores, autovectores = np.linalg.eig(ATA)

val_sing = np.sqrt(autovalores)
print(val_sing)
```

```
[8.1387 3.97   0.    ]
```

```
# Ponemos a los valores singulares en la matriz S
S = np.zeros((A.shape[0], A.shape[1]))
for i in range(min(A.shape)):
    S[i, i] = val_sing[i]
print(S)
```

```
[[8.1387 0.     0.    ]
 [0.     3.97   0.    ]]
```

- Las columnas de $\mathbf{V}$ son los autovectores ortonormales de $\mathbf{A}^T\mathbf{A}$:

```
V = autovectores
print(V.T)
```

```
[[-0.26   -0.6592 -0.7056]
 [-0.8913 -0.1172  0.438 ]
 [ 0.3714 -0.7428  0.5571]]
```

- Las columnas de $\mathbf{U}$ son los autovectores ortonormales de $\mathbf{A}\mathbf{A}^T$:

```
AAT = A @ A.T
autovalores, autovectores = np.linalg.eig(AAT)
print(autovalores)
```

```
[15.7611 66.2389]
```

```
# Los necesitamos ordenados de mayor a menor
indices = np.argsort(autovalores)[::-1]
autovalores = autovalores[indices]
print(autovalores)
```

```
[66.2389 15.7611]
```

```
U = autovectores[:, indices] # reordenamos de la misma forma los autovectores
print(U)
```

```
[[-0.2898 -0.9571]
 [-0.9571  0.2898]]
```

- Con los resultados obtenidos, podemos verificar que $\mathbf{A} = \mathbf{USV}^T$:

```
print(A)
```

```
[[4 2 0]
 [1 5 6]]
```

```
print(U @ S @ V.T)
```

```
[[ 4.  2. -0.]
 [ 1.  5.  6.]]
```

- Esta no es la forma más eficiente ni robusta de obtener la descomposición.
- Es sensible al ordenamiento de los autovalores y a los signos de los autovectores de $\mathbf{A}^T\mathbf{A}$: y de $\mathbf{AA}^T$, que se obtienen de forma independiente entre sí.
- Por eso, en debemos utilizar programas creados específicametne para este fin.
- Python tiene una función que se encarga de aplicar esto: `np.linalg.svd()`.
- Debemos notar que devuelve directamente la transpuesta de V.

```
U, val_sing, VT = np.linalg.svd(A, full_matrices=True)

print("Matriz U:")
```

```
Matriz U:
```

```
print(U)
```

```
[[ 0.2898  0.9571]
```

```
 [ 0.9571 -0.2898]]
```

```
print("\nValores singulares:")
```

```
Valores singulares:
```

```
print(val_sing)
```

```
[8.1387 3.97  ]
```

```
S = np.zeros((A.shape[0], A.shape[1]))
for i in range(min(A.shape)):
    S[i, i] = val_sing[i]
print("\nMatriz S (valores singulares):")
```

```
Matriz S (valores singulares):
```

```
print(S)
```

```
[[8.1387 0.     0.    ]
 [0.     3.97   0.    ]]
```

```
print("\nMatriz VT (transpuesta de V):")
```

```
Matriz VT (transpuesta de V):
```

```
print(VT)
```

```
[[ 0.26    0.6592  0.7056]
 [ 0.8913  0.1172 -0.438 ]
 [ 0.3714 -0.7428  0.5571]]
```

```
# Comprobamos que se reconstruye la matriz A
U @ S @ VT
```

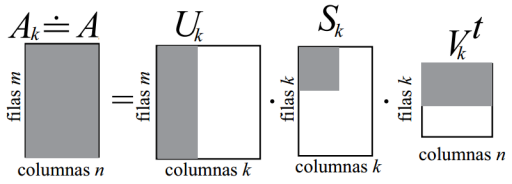
```
array([[ 4.,  2., -0.],
       [ 1.,  5.,  6.]])
```

5.4.2. Aplicaciones

- La razón de la importancia de la DVS en muchas aplicaciones es que nos permite captar las características más importantes de una matriz $m \times n$ (en muchos casos, con m mucho mayor que n) usando una matriz que, a menudo, es de tamaño significativamente más

pequeño.

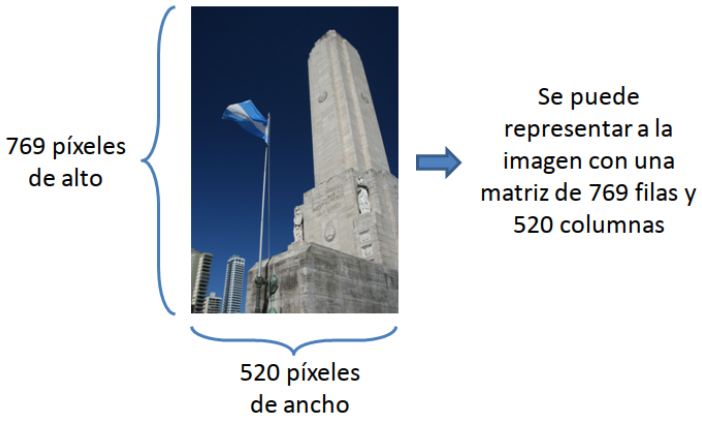
- El hecho de que los valores singulares están en la diagonal de $\mathbf{S}$ en orden decreciente implica que al hacer el producto $\mathbf{USV}^T$ para reconstruir a $\mathbf{A}$, quienes aportan la mayor parte de la información son las primeras columnas de cada una de estas matrices.
- Entonces para reconstruir $\mathbf{A}$ de manera exacta necesitamos estas tres matrices completas, pero para construir una muy buena aproximación a $\mathbf{A}$ nos alcanza con hacer el mismo producto usando sólo sus primeras k columnas:



- ¡Esto es un resultado impresionante! Significa que a un gran conjunto de datos lo podemos almacenar con mucho menos espacio mediante esas matrices reducidas, con muy poca pérdida de información.
- No hay una forma anticipada de saber con cuántos valores singulares (k) alcanza para tener una buena aproximación, eso depende de cada caso[3].
- La matriz $\mathbf{A}$ de dimensión $m \times n$ requiere mn registros para su almacenamiento.
- Sin embargo, la matriz $\mathbf{A}_k$, que aproxima a $\mathbf{A}$ y también es dimensión $m \times n$, sólo requiere de $k(m+n+1)$ registros para su almacenamiento (mk para $\mathbf{U}_k$, k para $\mathbf{S}_k$ y nk para $\mathbf{V}_k$).
- Hacer las cuentas para ver cuánto se gana de "espacio" si $m = 100$, $n = 10$ y $k = 4$…
- Esto se conoce como **compresión de datos** y de aquí que la DVS está tan relacionada con el Análisis de Componentes Principales, una técnica de reducción de la dimensionalidad.
- Para ponernos un poco más rigurosos, vale comentar que la matriz $\mathbf{A}_k = \mathbf{U}_k\mathbf{S}_k\mathbf{V}_k^T$ es de rango k y se demuestra que es la mejor aproximación mediante una matriz de rango $k < n$ de la matriz de datos $\mathbf{A}$ (posiblemente de rango n), en el sentido que es la que minimiza el error cuadrático de la predicción[4].
- Para finalizar vamos a ver un ejemplo de DVS aplicado al procesamiento de imágenes.
- ¿Qué tienen que ver las imágenes con nuestros conocimientos de matrices? Toda imagen digital se representa en la computadora como una matriz de píxeles, es decir, como un gran conjunto de puntitos ordenados en forma de matriz con filas y columnas, cada uno de un color en particular, que visualizados juntos dan lugar a la figura. Por lo tanto, una imagen se puede representar por una matriz donde cada celda tiene información acerca del color del píxel correspondiente:

[3] Para cuando vean Análisis de Componentes Principales, esto equivale a tener que elegir el número de componentes a utilizar.

[4] Para más detalles se puede consultar el libro sobre Análisis Multivariado de Peña, página 168.



- Existen códigos para representar a los distintos colores, por ejemplo, en el sistema hexadecimal, el código para el rojo es *FF0000*. Entonces, en la matriz que representa a una imagen digital está el valor *FF0000* por cada píxel rojo que la misma tenga. En ese caso, la matriz es de tipo caracter. Hay otros tipos de representación de colores que usan arreglos tridimensionales numéricos para indicar *cuánto* de rojo, de azúl y de verde tiene un píxel, ya que combinando esos tres se pueden formar el resto de los colores.
- Cuando se trabaja con imágenes en escala de grises, la cuestión es más sencilla. En cada celda de la matriz hay un número que varía entre 0 y 1. Una celda con un valor de 0 indica un píxel negro, mientras que una celda con un valor de 1 indica un pixel blanco. Es decir, un valor cercano a 0 es un gris bien oscuro, mientras que un valor cercano a 1 es un gris bien clarito. Por ejemplo:

Imagen original
de 3x2 píxeles

0	**0.6**
0.2	**0.8**
0.4	**1**

Matriz que la
representa en la compu

- Con el siguiente código leemos la imagen del Monumento Nacional a la Bandera mostrada anteriormente y la convertimos en escala de grises:
  1. Cargar una imagen desde el archivo "monu.png" utilizando la función imread y representarla con la matriz `A`.
  2. Convertirla a escala de grises. Para esto, se calcular la media a lo largo del último eje de la matriz A utilizando la función `np.mean` de NumPy con el argumento -1. Esto es equivalente a tomar el promedio a través de los canales de color en una imagen.

Cuando se trabaja con imágenes, el último eje suele representar los canales de color (por ejemplo, Rojo, Verde, Azul en una imagen RGB). Al tomar el promedio a lo largo del último eje, se obtiene una imagen en escala de grises en la que cada píxel representa el valor promedio de los canales de color en el píxel correspondiente. Por lo tanto, `A` queda como la matriz de valores entre 0 y 1 que que representa a la imagen en escala de grises.

```
# Lectura de la imagen
A = imread("Plots/U5/monu.png")

# Convertir a grises
A = np.mean(A, -1)

# Explorarla un poco
np.max(A)
```

```
0.97647065
```

```
np.min(A)
```

```
0.023529412
```

```
A
```

```
array([[0.1085, 0.1085, 0.1085, ..., 0.1203, 0.1203, 0.119 ],
       [0.1124, 0.1124, 0.1085, ..., 0.1203, 0.1203, 0.119 ],
       [0.1163, 0.1085, 0.1085, ..., 0.1203, 0.1203, 0.119 ],
       ...,
       [0.102 , 0.1137, 0.1242, ..., 0.4314, 0.4235, 0.4052],
       [0.1033, 0.1124, 0.132 , ..., 0.4235, 0.4157, 0.4   ],
       [0.1072, 0.1111, 0.1373, ..., 0.4549, 0.4353, 0.4196]], dtype=float32)
```

```
# Graficarla
plt.imshow(A, cmap='gray')
plt.axis('off')
```

```
(-0.5, 519.5, 768.5, -0.5)
```

```
plt.show()
```

- Siendo la imagen de dimensión $m = 769 \times n = 560$, se requiere de $769 \times 520 = 399880$ valores para su registro.
- ¿Será posible aplicarle una DVS para poder almacenarla con muchos menos valores, pero elegidos de forma tal que los mismos sirvan para reconstruir una buena aproximación de la imagen? Probemos…

```
# Aplicar DVS
U, val_sing, VT = np.linalg.svd(A, full_matrices = False)
S = np.diag(val_sing)

# Reconstruir la imagen de manera exacta (salvo errores de redondeo)
A2 = U @ S @ VT
plt.imshow(A2, cmap='gray')
plt.axis('off')
```

```
(-0.5, 519.5, 768.5, -0.5)
```

```
plt.show()
```

- Ahora vamos qué sucede si empleamos $k = 20$ valores singulares. En lugar de necesitar 399880 valores para almacenar la imagen, esto nos permitirá emplear sólo $k(m+n+1) = 20(520 + 769 + 1) = 25800$ (un 6.45 % del original).

```
# Reconstruir la imagen usando menos información
k = 20
img = U[:, :k] @ S[:k, :k] @ VT[:k, :]
plt.imshow(img, cmap='gray')
plt.axis('off')
```

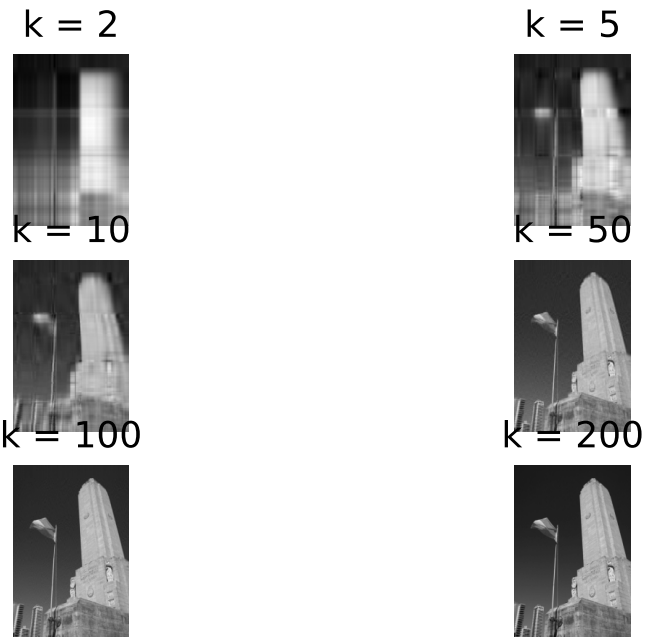
```
(-0.5, 519.5, 768.5, -0.5)
```

```
plt.show()
```

- Vemos que con una cantidad de registros que representa tan sólo un 6.45 % de la cantidad original, se logra reconstruir una imagen que conserva todos los rasgos principales.
- A continuación se presenta el resultado empleando distintos valores de k. Calcular en cada caso cuántos registros se necesitan.

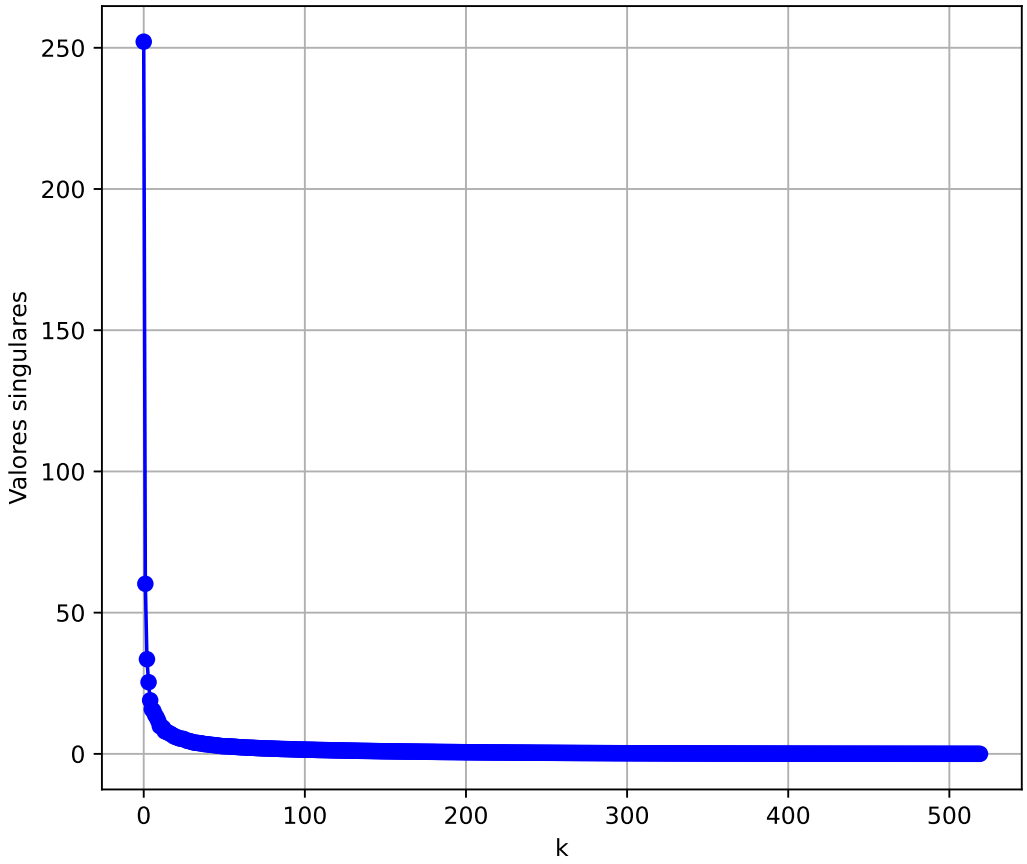
```
valores_k = [2, 5, 10, 50, 100, 200]

for i in range(len(valores_k)):

    # Establecer k
    k = valores_k[i]

    # Calcular la aproximación de la imagen
    img = U[:, :k] @ S[:k, :k] @ VT[:k, :]

    # Mostrar la aproximación de la imagen en escala de grises
    plt.subplot(3, 2, i+1)
    plt.imshow(img, cmap='gray')
    plt.axis('off')
    plt.title(f'k = {k}')
```

```
plt.show()
```



- Para determinar un buen valor de k es útil graficar los valores singulares en orden decreciente. Se puede elegir el valor para el cual se forma una especie de "codo", a partir del cual los valores singulares son similares entre sí y tienen un valor pequeño con respecto a los primeros. En el caso de la imagen del monumento, parece que $k = 20$ estaría bien.

```
# Trazar los valores singulares en función de k
plt.plot(range(520) , val_sing, marker='o', color='b')
plt.xlabel("k")
plt.ylabel("Valores singulares")
plt.grid(True)
plt.show()
```



- En otras aplicaciones, aplicar una DVS a una imagen puede ser necesario para poder borrar "ruido". Por ejemplo, si se trata de una fotografía que tal vez se tomó a gran distancia, como una imagen satelital, es probable que la misma incluya ruido, es decir, datos que no representan verdaderamente la imagen, sino el deterioro de ésta mediante partículas atmosféricas, la calidad de las lentes, procesos de reproducción, etc. Los datos de ruido se incorporan en los datos de la matriz $\mathbf{A}$, pero con suerte este ruido es mucho menos significativo que la verdadera imagen. Se espera que los valores singulares más grandes representen a la verdadera imagen y que los más pequeños, los más cercanos a cero, sean las contribuciones del ruido. Al realizar la DVS que solamente retiene esos valores singulares por encima de cierto umbral, podríamos ser capaces de eliminar la mayor parte del ruido y, en realidad, obtener una imagen que no sólo sea de menor tamaño sino también una representación más clara de la superficie.

5.4.3. No está de más saber que...

- Ya quedó claro que DVS será importante a la hora de estudiar Análisis de Datos Multivariados.
- En esta sección vamos a mencionar un par de detalles adicionales que son un nexo entre lo que han estudiado de Álgebra Lineal, estamos aplicando ahora mediante Métodos Numéricos y verán su utilidad en Análisis de Datos Multivariados:

  1. Un poquito más arriba mencionamos al rango de una matriz. Recordamos que el rango de una matriz es el número máximo de vectores fila o columna que linealmente independientes. Si $\mathbf{A}_{m\times n}$, $rg(\mathbf{A}) \leq min(m,n)$. Si $rg(\mathbf{A}) = min(m,n)$, se dice que la matriz es de rango completo. En Estadística, el rango de una matriz de datos nos indica la dimensión real necesaria para representar el conjunto de datos, o el número real de variables distintas que disponemos. Analizar el rango de una matriz de datos es la clave para reducir el número de variables sin pérdida de información.
  2. También ha aparecido en la DVS la matriz $\mathbf{A}^T A$. Pasó por ahí casi desapercibida, pero esta matriz es fundamental en Estadística. Si $\mathbf{A}$ es nuestra matriz de datos, que generalmente denotamos con $\mathbf{X}$, entonces $\mathbf{X}^T\mathbf{X}$ es proporcional a la **matriz de variancias y covariancias**. Su **determinante** es una medida global de la independencia entre las variables. A mayor determinante, mayor independencia. Para que la DVS pueda hacer una buena compresión con pocos valores singulares k, se necesita que las variables estén correlacionadas.
  3. La **traza** de una matriz es una medida global de su tamaño que se obtiene sumando sus elementos diagonales. Por ejemplo, la traza de una matriz de variancias y covariancias es la suma de todas las variancias de las variables. Entonces, la suma de los elementos diagonales es una medida de variabilidad que, a diferencia del determinante, no tiene en cuenta las relaciones entre las variables.

6 Aproximación polinomial - Parte 1: interpolación

6.1. Introducción

- Una de las clases más útiles y conocidas de funciones que mapean el conjunto de números reales en sí mismo son los **polinomios algebraicos**:

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$$

  donde n es un entero positivo y $a_0, ..., a_n$ son constantes reales.
- Una razón de su importancia es que se aproximan de manera uniforme a las funciones continuas.
- Es decir, dada una función definida y continua sobre un intervalo cerrado y acotado, existe un polinomio que está tan "cerca" de la función dada como se desee. **Ver teorema 3.1 (Weierstrass) y Figura 3.1 (página 78)**.
- Por esta razón, empleamos polinomios para aproximar el valor de una función $f(x)$ en un intervalo de interés (es decir, para realizar **interpolación**) y también para aproximar derivadas e integrales.
- Englobamos a las técnicas que permiten cumplir con esos objetivos bajo el nombre de **métodos de aproximación polinomial**.
- **Observación**: los polinomios de Taylor son esenciales en muchos aspectos del análisis numérico porque permiten aproximar el valor de una función alrededor de un punto específico. Sin embargo, la aproximación polinomial no se basa en el uso de polinomios de Taylor. Leer páginas 78 y 79.
- En este apunte vamos a tratar el tema de la interpolación.
- Preparativos para los ejemplos en Python:

```
# Algunas librerías y funciones necesarias
from unidad6_funciones import *
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import numpy.polynomial.polynomial as npol
```

6.2. Polinomios de interpolación de Lagrange

6.2.1. Fórmula

- Vamos a considerar el problema en el que contamos con los valores que toma una función de interés f en los $n+1$ puntos $x_0, x_1, ..., x_n$ (es decir, conocemos $f(x_0), ..., f(x_n)$) y queremos aproximar el valor de $f(x)$ para otros valores x.

Definición: la **interpolación** es el uso de polinomios que coinciden con una función f en puntos determinados dentro de un intervalo para aproximar el valor de f en otros puntos dentro del mismo intervalo.

- A los valores x_i para los cuales tenemos $f(x_i)$ les decimos **nodos** y a estos pares de valores los presentamos en una tabla como la siguiente:

i	x_i	$f(x_i)$
0	x_0	$f(x_0)$
1	x_1	$f(x_1)$
$\vdots$	$\vdots$	$\vdots$
n	x_n	$f(x_n)$

- Por ejemplo:

i	x_i	$f(x_i)$
0	-1	0.5403
1	0	1.0000
2	2	-0.4162
3	2.5	-0.8011

- Pensemos en que estamos estudiando una función f para la cual conocemos los valores $f(x_0)$ y $f(x_1)$ y necesitamos aproximar cuánto vale $f(x)$ para algún x entre x_0 y x_1.
- ¿Qué se nos ocurre hacer?
- Podemos trazar una recta que una los puntos $(x_0, f(x_0))$ y $(x_1, f(x_1))$ y utilizar como aproximación el valor de esta recta para el x de interés.
- Eso equivale a utilizar un polinomio de grado 1 (una recta) para hacer la interpolación.
- De nuestros conocimientos de geometría sabemos que la ecuación de la recta que pasa por dos puntos $(x_0, f(x_0))$ y $(x_1, f(x_1))$ es:

$$P_1(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

- Esto se puede reescribir como:

$$P_1(x) = \underbrace{\frac{x - x_1}{x_0 - x_1}}_{L_0(x)} f(x_0) + \underbrace{\frac{x - x_0}{x_1 - x_0}}_{L_1(x)} f(x_1) = L_0(x)f(x_0) + L_1(x)f(x_1)$$

- Podemos comprobar que dicha recta para por $(x_0, f(x_0))$ y $(x_1, f(x_1))$ (sustituir x y verificar).
- **Importante**: $P(x)$ es el único polinomio de grado 1 que pasa por dichos puntos. Podríamos reescribirlo de muchas formas, pero es un polinomio único (en este caso, es la única recta).
- Esta expresión se puede extender para obtener polinomios de grados superiores que pasen por más puntos.
- Por ejemplo, el polinomio que pasa por los puntos $(x_0, f(x_0))$, $(x_1, f(x_1))$ y $(x_2, f(x_2))$ es:

$$P_2(x) = \underbrace{\frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}}_{L_0(x)} f(x_0) + \underbrace{\frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}}_{L_1(x)} f(x_1) + \underbrace{\frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}}_{L_2(x)} f(x_2)$$

- Se puede ver fácilmente que este polinomio pasa exactamente por los tres puntos dados.
- De la misma forma, el polinomio de grado 3 que pasa por cuatro puntos $(x_0, f(x_0))$, $(x_1, f(x_1))$, $(x_2, f(x_2))$ y $(x_3, f(x_3))$ es:

$$\begin{aligned} P_3(x) = {} & \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)} f(x_0) + \frac{(x - x_0)(x - x_2)(x - x_3)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)} f(x_1) \\ & + \frac{(x - x_0)(x - x_1)(x - x_3)}{(x_2 - x_0)(x_2 - x_1)(x_2 - x_3)} f(x_2) + \frac{(x - x_0)(x - x_1)(x - x_2)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)} f(x_3) \end{aligned}$$

- Se puede ver fácilmente que este polinomio pasa exactamente por los cuatro puntos dados.
- Generalizando la idea anterior se obtiene la fórmula de interpolación de Lagrange.

Teorema: Si $x_0, x_1, ..., x_n$ son $n + 1$ números distintos y f es una función cuyos valores están determinados en estos números, entonces existe un único polinomio $P(x)$ de grado a lo sumo n con

$$f(x_k) = P(x_k) \qquad \forall\, k = 0, 1, ..., n$$

Este polinomio recibe el nombre de **enésimo polinomio de interpolación de Lagrange** y está determinado por:

$$P_n(x) = L_{n,0}(x)f(x_0) + ... + L_{n,n}(x)f(x_n) = \sum_{k=0}^{n} f(x_k)L_{n,k}(x),$$

donde para cada $k = 0, 1, ..., n$:

$$L_{n,k}(x) = \frac{(x-x_0)(x-x_1)...(x-x_{k-1})(x-x_{k+1})...(x-x_n)}{(x_k-x_0)(x_k-x_1)...(x_k-x_{k-1})(x_k-x_{k+1})...(x_k-x_n)} = \prod_{\substack{i=0 \\ i\neq k}}^{n} \frac{x-x_i}{x_k-x_i}$$

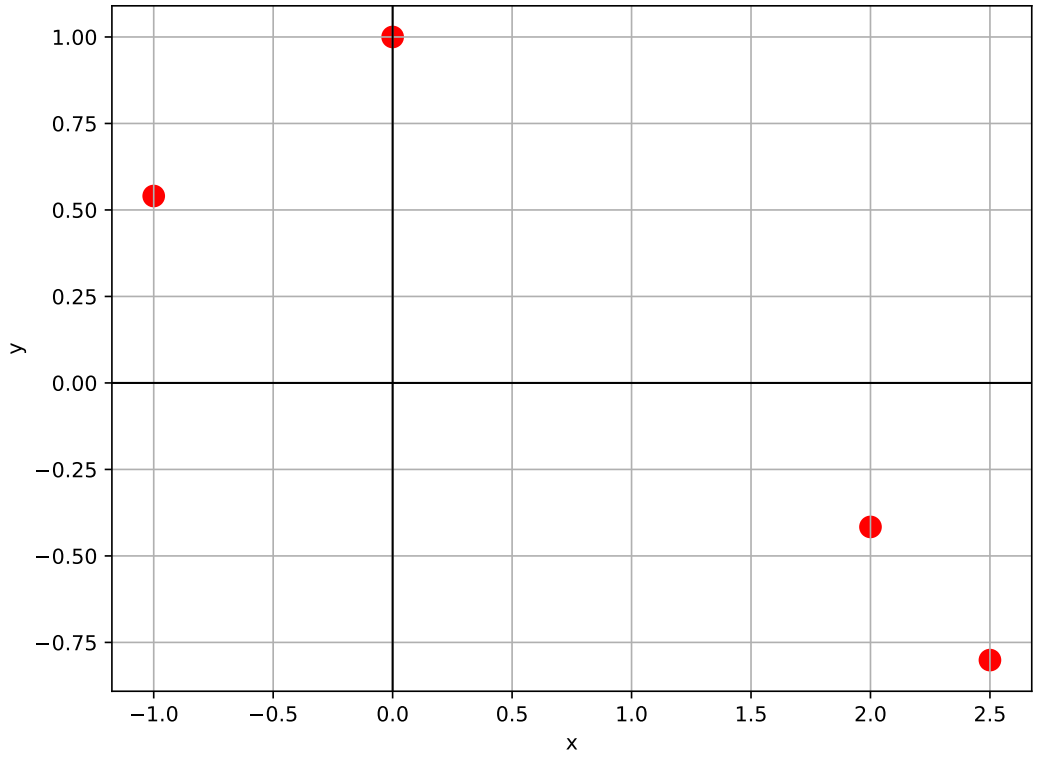
- Cuando no hay confusión acerca del valor de n escribimos directamente $L_k(x)$ en lugar de $L_{n,k}(x)$.
- La función $L_{n,k}(x)$ hace que el polinomio pase por los puntos dados debido a que $L_{n,k}(x_i) = 0$ cuando $i \neq k$ y $L_{n,k}(x_k) = 1$ cuando $i = k$.

6.2.2. Ejemplo

- Emplear el método de Lagrange para interpolar el valor de la función f en $x = 2.25$, con el polinomio interpolante de grado 3 que pasa por los cuatro puntos tabulados:

i	x_i	$f(x_i)$
0	-1	0.5403
1	0	1.0000
2	2	-0.4162
3	2.5	-0.8011

```
# Crear un DataFrame con los datos
datos = pd.DataFrame({'x': [-1, 0, 2, 2.5], 'y': [0.5403, 1, -0.4162, -0.8011]})

# Crear el gráfico
plt.figure(figsize=(8, 6))
plt.grid(True)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.scatter(datos['x'], datos['y'], color='red', s=100)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
y
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
x

$$
\begin{aligned}
P_3(x) &= \sum_{k=0}^{3} f(x_k)\Big(\prod_{\substack{i=0\\ i\neq k}}^{3} \frac{x-x_i}{x_k-x_i}\Big) \\
&= f(x_0)\frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} \\
&\quad + f(x_1)\frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} \\
&\quad + f(x_2)\frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} \\
&\quad + f(x_3)\frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} \\
\implies P_3(2.25) &= 0.5403 \times \frac{(2.25-0)(2.25-2)(2.25-2.5)}{(-1-0)(-1-2)(-1-2.5)} \\
&\quad + 1 \times \frac{(2.25+1)(2.25-2)(2.25-2.5)}{(0+1)(0-2)(0-2.5)} \\
&\quad - 0.4162 \times \frac{(2.25+1)(2.25-0)(2.25-2.5)}{(2+1)(2-0)(2-2.5)} \\
&\quad - 0.8011 \times \frac{(2.25+1)(2.25-0)(2.25-2)}{(2.5+1)(2.5-0)(2.5-2)} = \\
&= -0.6217561 \\
\therefore f(2.25) &\approx -0.6217561
\end{aligned}
$$

- La fórmula fue programada en la función provista `lagrange()`:

```
x = np.array([-1, 0, 2, 2.5])
fx = np.array([0.5403, 1, -0.4162, -0.8011])
lagrange(x, fx, 2.25)
```

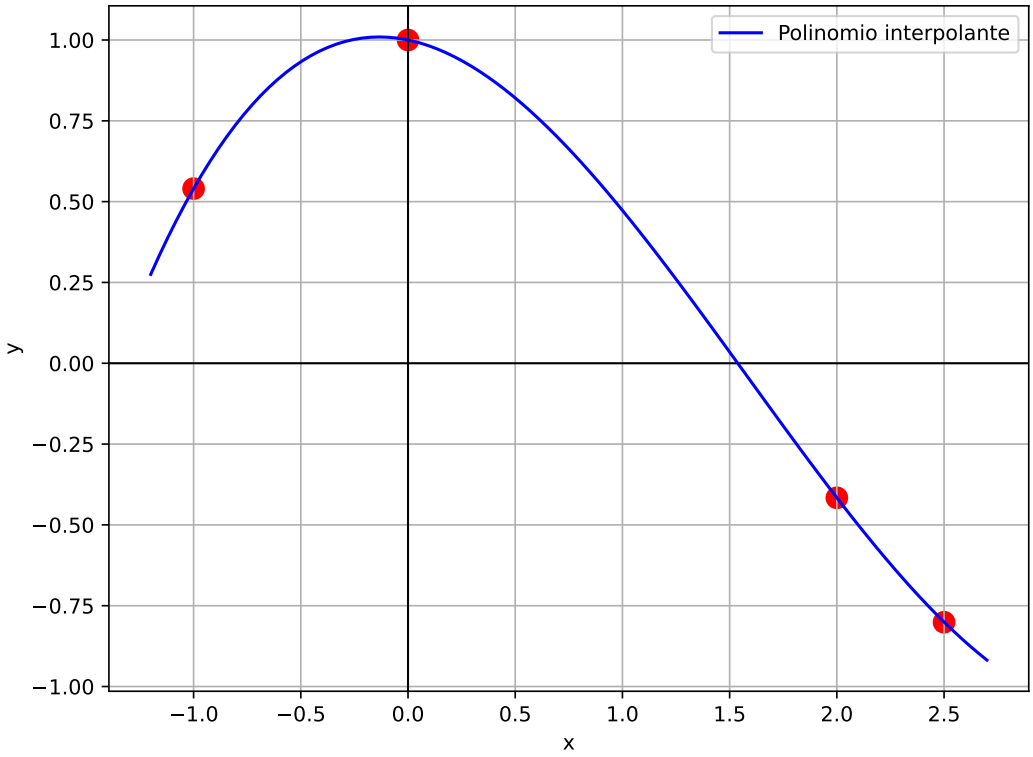
```
-0.6217560714285715
```

- Si en la fórmula anterior en lugar de reemplazar x por un valor particular (en este caso, 2.25) operamos y re acomodamos los términos, podemos hallar la expresión del polinomio interpolante:

$$P_3(x) = 0.1042x^3 - 0.4934x^2 - 0.1379x + 1$$

- Podemos graficar este polinomio para ver lo que ha logrado este método:

```
# Definir la función p3 como una función lambda
p3 = lambda x: 0.1042 * x**3 - 0.4934 * x**2 - 0.1379 * x + 1

# Crear un rango de valores para x
rango_x = np.linspace(-1.2, 2.7, 100)

# Calcular los valores correspondientes de p3
y = p3(rango_x)

# Crear el gráfico
plt.figure(figsize=(8, 6))
plt.grid(True)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.scatter(datos['x'], datos['y'], color='red', s=100)
plt.xlabel('x')
plt.ylabel('y')
plt.plot(rango_x, y, label="Polinomio interpolante", color='blue')
plt.legend()
plt.show()
```



- En Python podemos usar la función `polyfit()` de la librería `numpy.polynomial.polynomial` para obtener la expresión del polinomio interpolante de Lagrange y realizar interpolaciones, pero como siempre escribimmos nuestras propias funciones para entender lo que hacen los métodos y repasar programación:

```
# Ajustar un polinomio a los datos
x = np.array([-1, 0, 2, 2.5])
fx = np.array([0.5403, 1, -0.4162, -0.8011])

# Coeficientes del polinomio (menor a mayor grado)
coefs = npol.polyfit(x, fx, deg = len(x) - 1)
print(coefs)
```

```
[ 1.         -0.1379019  -0.49343429  0.10416762]
```

```
# Convertir esos coeficientes en una función polinómica
poli = npol.Polynomial(coefs)
# Mostrar la expresion del polinomio
```

```
print(poli)
```

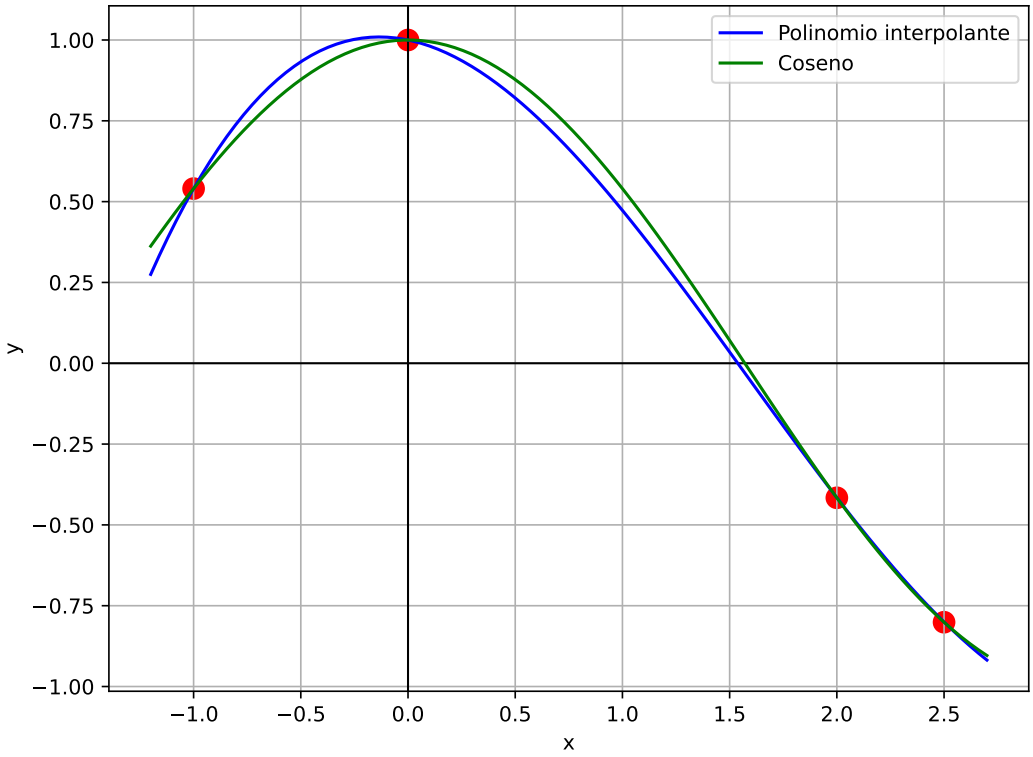
```
1.0 - 0.1379019·x - 0.49343429·x² + 0.10416762·x³
```

```
# Evaluar el polinomio interpolante en el punto 2.25
poli(2.25)
```

```
-0.6217560714285701
```

- La verdadera función que generó los valores tabulados es $cos(x)$. Podemos compararla con el polinomio interpolante:

```
y_cos = np.cos(rango_x)

plt.figure(figsize=(8, 6))
plt.grid(True)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.scatter(datos['x'], datos['y'], color='red', s=100)
plt.xlabel('x')
plt.ylabel('y')
plt.plot(rango_x, y, label="Polinomio interpolante", color='blue')
plt.plot(rango_x, y_cos, label="Coseno", color='green')
plt.legend()
plt.show()
```

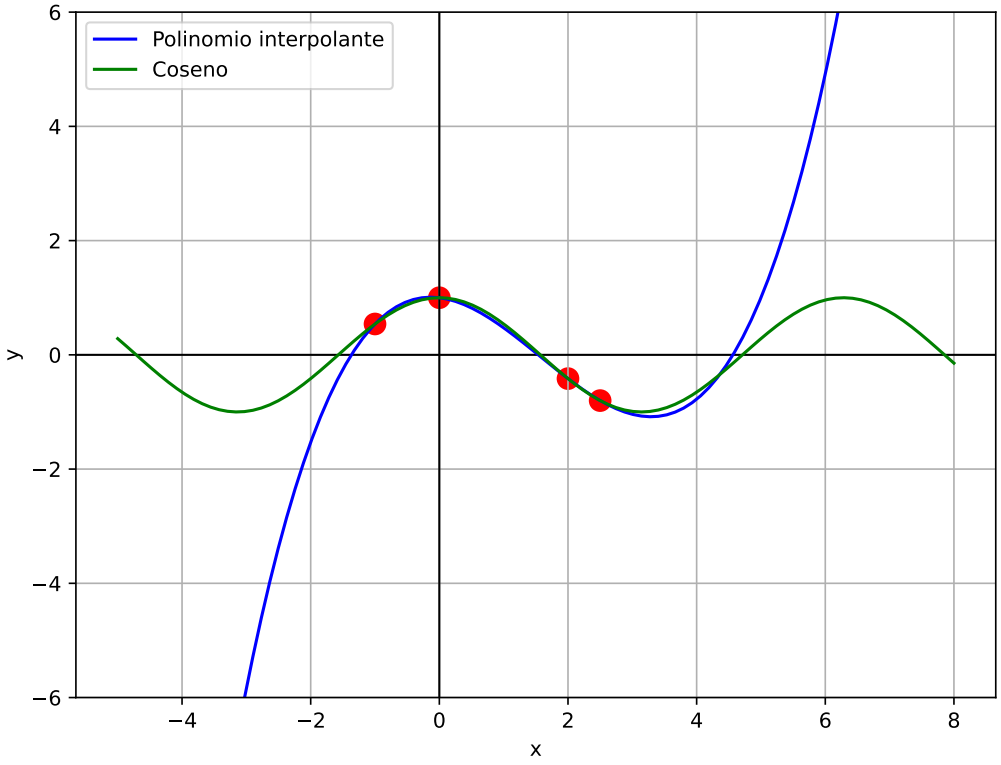


- Ojo, si nos alejamos del rango estudiado, el polinomio no tiene por qué aproximar bien (cuidado con la extrapolación)...

```
# Función verdadera (coseno) y polinimo interpolante fuera del rango de puntos
# disponibles
rango_x = np.linspace(-5, 8, 100)
y_cos = np.cos(rango_x)
y = p3(rango_x)

plt.figure(figsize=(8, 6))
plt.grid(True)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.scatter(datos['x'], datos['y'], color='red', s=100)
plt.plot(rango_x, y, label="Polinomio interpolante", color='blue')
plt.plot(rango_x, y_cos, label="Coseno", color='green')
plt.ylim(-6, 6)
```

```
(-6.0, 6.0)
```

```
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



6.2.3. Error de aproximación

- En el ejemplo, sabiendo el verdadero valor $cos(2.25) = -0.6281736$, podemos calcular el error absoluto:

$$|-0.6281736 + 0.6217561| = 0.0064175$$

y el error relativo:

$$\frac{|-0.6281736 + 0.6217561|}{|-0.6281736|} = 1.021612\,\%$$

- En una situación práctica donde no tenemos el verdadero $f(x)$, vale el siguiente resultado.
- Se puede demostrar (ver opcionalemente Teorema 3.3) que el error en la aproximación con el polinomio de Lagrange está dado por:

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)(x - x_1)...(x - x_n)$$

  donde ξ es algún número dentro del intervalo en el que se encuentran los nodos x_i.
- Para saber cuál es la cota superior del error se busca acotar la expresión anterior para cualquier ξ dentro del rango estudiado (ver opcionalmente ejemplos 3 y 4).
- En el ejemplo nuestro, siendo $f(x) = cos(x)$, $n = 3$ y $f^{(4)}(x) = cos(x)$:

$$f(x) - P_n(x) = \frac{cos(\xi)}{4!}(x + 1)x(x - 2)(x - 2.5) \qquad -1 \leq \xi \leq 2.5$$

- Sabiendo que $|cos(\xi)| \leq 1$, encontramos una cota superior para el error de aproximación:

$$|f(x) - P_n(x)| \leq \left|\frac{1}{4!}(x + 1)x(x - 2)(x - 2.5)\right|$$

- Para el punto analizado $x = 2.25$ esa cota superior nos da 0.01904297 (y efectivamente el error absoluto, que lo pudimos calcular, era menor que esto).
- Este resultado es muy importante porque permite evaluar el desempeño de la aproximación pero muchas veces es impracticable porque no se conoce cuál es la derivada $f^{(n+1)}$, por lo tanto tiene utilidad teórica pero no práctica.
- En esos casos, sólo es posible evaluar la precisión al comparar las aproximaciones obtenidas con polinomios de distinto grado (ver ejemplo en la sección "Ilustración" en páginas 86 y 87).

6.2.4. Ventajas y desventajas

- **Ventajas**:
  - Una ventaja de este método de Lagrange es que provee de una forma sencilla una expresión explícita para el polinomio de interpolación.
  - Además, no requiere que los puntos x_i estén ordenados ni sean equiespaciados.
- **Desventaja**:
  - Dado que el término de error difícilmente puede ser construído, generalmente se necesitan considerar varios polinomios de interpolación de distinto grado para comparar las aproximaciones logradas.
  - Sin embargo, con el método de Lagrange no hay relación entre la construcción de un polinomio de un grado n y otro de grado $n+1$; cada polinomio debe construirse individualmente realizando todos los cálculos otra vez, lo cual resulta laborioso y poco eficiente.

- El **Método de Neville** (no lo estudiaremos) soluciona este inconveniente, reformulando los cálculos para poder obtener aproximaciones usando cálculos previos. Una aproximación de grado $n+1$ se logra tomando la de grado n y sumándole otro término.
- Sin embargo, este método no permite obtener una expresión explícita del polinomio.
- Para poder generar sucesivamente los polinomios interpolantes se puede recurrir a los métodos de **diferencias de Newton** que se presentan en la siguiente sección.

6.3. Polinomios de interpolación de Newton

- A pesar de que el polinomio que $P_n(x)$ que concuerda con la función f en $x_0, ..., x_n$ es único, existen diferentes representaciones algebraicas que son útiles en distintas situaciones y la fórmula de Lagrange es sólo una de ellas.
- Para poder generer polinomios interpolantes con otra expresión que admita cálculos recursivos (es decir, aprovechando los cálculos hechos para polinomios de menor grado), vamos a necesitar obtener una **tabla de diferencias divididas** y una **tabla de diferencias ordinarias**.

6.3.1. Diferencias divididas

- Presentar formalmente a las **diferencias divididas** es más difícil que calcularlas.

Definición:

- La *cero-ésima diferencia dividida* de la función f respecto a x_i es: $f[x_i] = f(x_i)$.
- La *primera diferencia dividida* de f respecto a x_i y x_{i+1} es:

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$

- La *segunda diferencia dividida* de f respecto a x_i, x_{i+1} y x_{i+2} es:

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}$$

- De manera general, después de haber calculado las $k-1$ diferencias divididas:

$$f[x_i, ..., x_{i+k-1}] \quad \text{y} \quad f[x_{i+1}, ..., x_{i+k}],$$

la k-ésima diferencia dividida relativa a $x_i, ..., x_{i+k}$ es:

$$f[x_i, ..., x_{i+k}] = \frac{f[x_{i+1}, ..., x_{i+k}] - f[x_i, ..., x_{i+k-1}]}{x_{i+k} - x_i}$$

- Lo anterior es más claro si lo presentamos en una **tabla de diferencias divididas**.
- Por ejemplo, con 6 puntos:

x	$f(x)$	Primeras diferencias divididas	Segundas diferencias divididas	Terceras diferencias divididas
x_0	$f[x_0]$			
		$f[x_0,x_1]=\frac{f[x_1]-f[x_0]}{x_1-x_0}$		
x_1	$f[x_1]$		$f[x_0,x_1,x_2]=\frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$	
		$f[x_1,x_2]=\frac{f[x_2]-f[x_1]}{x_2-x_1}$		$f[x_0,x_1,x_2,x_3]=\frac{f[x_1,x_2,x_3]-f[x_0,x_1,x_2]}{x_3-x_0}$
x_2	$f[x_2]$		$f[x_1,x_2,x_3]=\frac{f[x_2,x_3]-f[x_1,x_2]}{x_3-x_1}$	
		$f[x_2,x_3]=\frac{f[x_3]-f[x_2]}{x_3-x_2}$		$f[x_1,x_2,x_3,x_4]=\frac{f[x_2,x_3,x_4]-f[x_1,x_2,x_3]}{x_4-x_1}$
x_3	$f[x_3]$		$f[x_2,x_3,x_4]=\frac{f[x_3,x_4]-f[x_2,x_3]}{x_4-x_2}$	
		$f[x_3,x_4]=\frac{f[x_4]-f[x_3]}{x_4-x_3}$		$f[x_2,x_3,x_4,x_5]=\frac{f[x_3,x_4,x_5]-f[x_2,x_3,x_4]}{x_5-x_2}$
x_4	$f[x_4]$		$f[x_3,x_4,x_5]=\frac{f[x_4,x_5]-f[x_3,x_4]}{x_5-x_3}$	
		$f[x_4,x_5]=\frac{f[x_5]-f[x_4]}{x_5-x_4}$		
x_5	$f[x_5]$			

- La tabla anterior podría llegar hasta las quintas diferencias divididas.
- **En general, si contamos con $n+1$ puntos, podemos calcular hasta las diferencias divididas de orden n.**

6.3.2. Fórmula general de Newton para la interpolación con diferencias divididas

- Para poder generar polinomios interpolantes de distinto orden recursivamente Newton propone emplear la siguiente representación para $P_n(x)$:

$$P_n(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + ... + a_n(x-x_0)...(x-x_{n-1}) \qquad (6.1)$$

- De hecho, antes repasamos la fórmula de la recta (polinomio interpolante de grado 1) que pasa por dos puntos y estaba escrita de esa forma:

$$P_1(x) = \underbrace{f(x_0)}_{a_0} + \underbrace{\frac{f(x_1)-f(x_0)}{x_1-x_0}}_{a_1}(x-x_0)$$

- Si miramos bien, tenemos que notar que a_0 y a_1 coinciden con la definición de **diferencias divididas** para x_0 de orden 0 y 1, respectivamente.
- De manera general, se puede probar que las constantes a_k necesarias para expresar $P_n(x)$ de la forma deseada (6.1) son las diferencias divididas $a_k = f[x_0,...,x_k]$, haciendo que el polinomio quede así:

$$P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0,...,x_k](x-x_0)...(x-x_{k-1}) \qquad (6.2)$$

- Es decir, se usan las diferencias divididas que están en la diagonal de la tabla.

- **Ejemplo**. Retomamos el ejemplo anterior. A continuación se presentan los puntos tabulados junto con las diferencias divididas. Los valores en negrita son los que se utilizan en la fórmula. Verificar los cálculos a mano.

i	x_i	$f(x_i)$	Primeras diferencias	Segundas diferencias	Terceras diferencias
0	-1	**0.5403**			
			0.4597		
1	0	1.0000		**-0.3893**	
			-0.7081		**0.1042**
2	2	-0.4162		-0.0247	
			-0.7698		
3	2.5	-0.8011			

- En Python vamos a usar la función provista `diferencias()` para obtener este resultado:

```
diferencias(fx, x)
```

```
array([[ 0.5403    ,  0.4597    , -0.38926667,  0.10416762],
       [ 1.        , -0.7081    , -0.02468   ,         nan],
       [-0.4162    , -0.7698    ,         nan,         nan],
       [-0.8011    ,         nan,         nan,         nan]])
```

- Vamos emplear el método de las diferencias divididas de Newton para interpolar el valor de la función f en $x = 2.25$, con el polinomio interpolante de grado 3 que pasa por los cuatro puntos tabulados:

$$P_3(x) = f[x_0] + \sum_{k=1}^{3} f[x_0, ..., x_k](x - x_0)...(x - x_{k-1})$$

$$= 0.5403$$

$$+ 0.4597(x + 1)$$

$$- 0.3893(x + 1)x$$

$$+ 0.1042(x + 1)x(x - 2)$$

$$\implies P_3(2.25) = -0.6217561$$

- En Python lo vamos a aplicar en la función `newton_general()` (provista en el archivo de funciones):

```
newton_general(x, fx, valor = 2.25)
```

```
-0.6217560714285713
```

- Como el polinomio de grado 3 que pasa por 4 puntos es único, este resultado coincide con el del método de Lagrange.
- Sin embargo, tiene una ventaja importante. No siempre es necesario ni conveniente ajustar el polinomio de mayor orden posible, pero probar con distintos grados es muy laborioso en el método de Lagrange.
- En cambio este método lo vuelve más sencillo, porque pasamos de una aproximación de un grado menor a otra de un grado superior solamente sumando un término más en la cuenta. Se aprovechan los cálculos anteriores. Entonces:

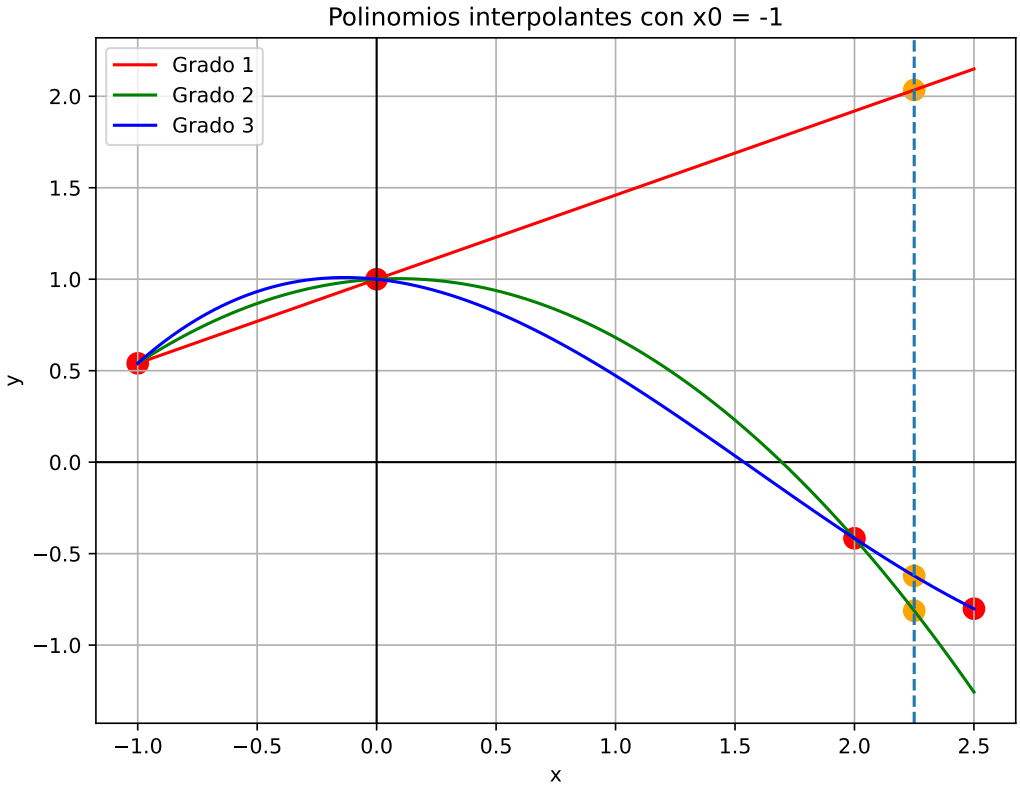
Grado	$P_n(2.25) =$
1	$f[x_0] + f[x_0, x_1](x - x_0) = 2.0343$
2	$2.0343 + f[x_0, x_1, x_2](x - x_0)(x - x_1) = -0.8124$
3	$-0.8124 + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_3) = -0.6218$

- En Python:

```
for grado in range(1, 4):
  print(newton_general(x, fx, valor = 2.25, g = grado))
```

```
2.034325
-0.8121874999999998
-0.6217560714285713
```

- La primera aproximación es una interpolación lineal y pasa por los primeros dos puntos; la segunda es cuadrática y pasa por los primeros tres puntos y la última es cúbica y pasa por todos los puntos:



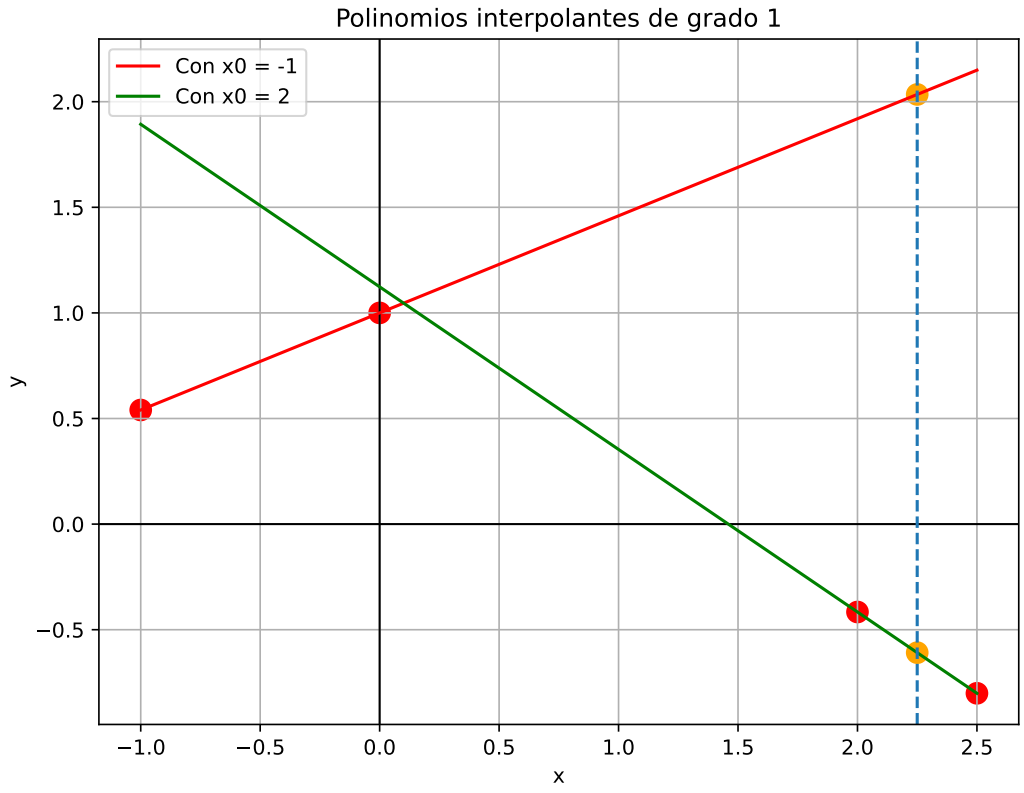
- Es claro que la aproximación lineal anterior no es buena para $x = 2.25$.
- Sin embargo, una recta que pase por los últimos dos puntos podría tener un desempeño similar al del polinomio de grado 3:

```
newton_general(x[2:], fx[2:], 2.25)
```

```
-0.60865
```

```
# preparar curvas para graficar
p1b = npol.Polynomial(npol.polyfit(x[2:], fx[2:], deg = 1))
y_p1b = p1b(rango_x)

# Crear el gráfico
plt.figure(figsize=(8, 6))
plt.grid(True)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.scatter(datos['x'], datos['y'], color='red', s=100)
```

```
plt.xlabel('x')
plt.ylabel('y')
plt.plot(rango_x, y_p1, label="Con x0 = -1", color='red')
plt.plot(rango_x, y_p1b, label="Con x0 = 2", color='green')
plt.legend()
plt.title("Polinomios interpolantes de grado 1")
plt.axvline(x=2.25, linestyle='--')
plt.scatter([2.25, 2.25], [p1(2.25), p1b(2.25)], color = "orange", s = 100)
plt.show()
```



- La expresión anterior facilita la obtención de polinomios interpolantes de diferentes grados, puesto que los mismos sólo se diferencian en términos que se van agregando en la parte de la sumatoria.
- Sin embargo, la expresión de la fórmula no deja de ser algo compleja e involucra muchos cálculos.
- En el caso de que los nodos x_i están ordenados de menor a mayor y tengan igual espaciado entre ellos, la fórmula se simplifica muchísimo y recibe el nombre de

- Pero para conocer esta fórmula necesitamos definir las **diferencias ordinarias**.

6.3.3. Diferencias ordinarias

- Las **diferencias ordinarias** se asemejan a las divididas, pero no hacen la división por las restas entre valores de x.

Definición: las **diferencias ordinarias** se expresan como:

$$\Delta^j f(x_k) = \Delta^{j-1} f(x_{k+1}) - \Delta^{j-1} f(x_k) \qquad j = 1, ..., n \qquad k = 0, ..., j-1 \qquad \Delta^0 f(x_k) = f(x_k)$$

- Es decir que las diferencias de orden 0 son los mismos valores $f(x_i)$, las diferencias de orden 1 son $f(x_{k+1}) - f(x_k)$, las diferencias de orden 2 son restas con las de orden 1, y así sucesivamente.
- Para simplificar la notación, cuando no haya ambigüedad vamos a escribir: $\Delta^j f(x_k) = \Delta^j_k$.
- La **tabla de diferencias ordinarias** es:

x	$f(x)$	Primeras diferencias	Segundas diferencias	Terceras diferencias	Cuartas diferencias	Quintas diferencias
x_0	$f(x_0)$					
		$\Delta^1_0 = f(x_1) - f(x_0)$				
x_1	$f(x_1)$		$\Delta^2_0 = \Delta^1_1 - \Delta^1_0$			
		$\Delta^1_1 = f(x_2) - f(x_1)$		$\Delta^3_0 = \Delta^2_1 - \Delta^2_0$		
x_2	$f(x_2)$		$\Delta^2_1 = \Delta^1_2 - \Delta^1_1$		$\Delta^4_0 = \Delta^3_1 - \Delta^3_0$	
		$\Delta^1_2 = f(x_3) - f(x_2)$		$\Delta^3_1 = \Delta^2_2 - \Delta^2_1$		$\Delta^5_0 = \Delta^4_1 - \Delta^4_0$
x_3	$f(x_3)$		$\Delta^2_2 = \Delta^1_3 - \Delta^1_2$		$\Delta^4_1 = \Delta^3_2 - \Delta^3_1$	
		$\Delta^1_3 = f(x_4) - f(x_3)$		$\Delta^3_2 = \Delta^2_3 - \Delta^2_2$		
x_4	$f(x_4)$		$\Delta^2_3 = \Delta^1_4 - \Delta^1_3$			
		$\Delta^1_4 = f(x_5) - f(x_4)$				
x_5	$f(x_5)$					

- **En general, si contamos con $n+1$ puntos, podemos calcular hasta las diferencias ordinarias de orden n.**

6.3.4. Fórmula de interpolación de Newton con diferencias hacia adelante

- Veamos cómo la fórmula general de Newton se simplifica para el caso particular en el que los nodos x_i están ordenados de menor a mayor y son equiespaciados.
- Llamamos con h al espaciado uniforme: $h = x_{i+1} - x_i$, para cada $i = 0, 1, ..., n-1$.

- Cualquier valor x puede ser expresado como $x = x_0 + sh$ y en particular los nodos se pueden escribir como $x_i = x_0 + ih$.
- Entonces nos queda: $x - x_i = (s - i)h$.
- Haciendo los reemplazos correspondientes y después de varios pasos algebraicos, (6.2) nos queda de una forma mucho más compacta:

$$\begin{aligned} P_n(x) &= f(x_0) + s\Delta_0^1 + \frac{s(s-1)}{2!}\Delta_0^2 + \frac{s(s-1)(s-2)}{3!}\Delta_0^3 + ... \\ &= \sum_{k=0}^{n} \binom{s}{k} \Delta_0^k \end{aligned}$$

  donde: $\binom{s}{k} = \frac{s(s-1)(s-2)...(s-k+1)}{k!}$.
- Esta es la **fórmula de interpolación de Newton con diferencias hacia adelante**, porque utiliza la primera diagonal desde arriba a la izquierda hacia abajo a la derecha de la **tabla de diferencias ordinarias**.
- Hay que observar que la fórmula empieza con el valor de $f(x_0)$ y luego continúa empleando las diferencias Δ_0^k.

Ejemplo.

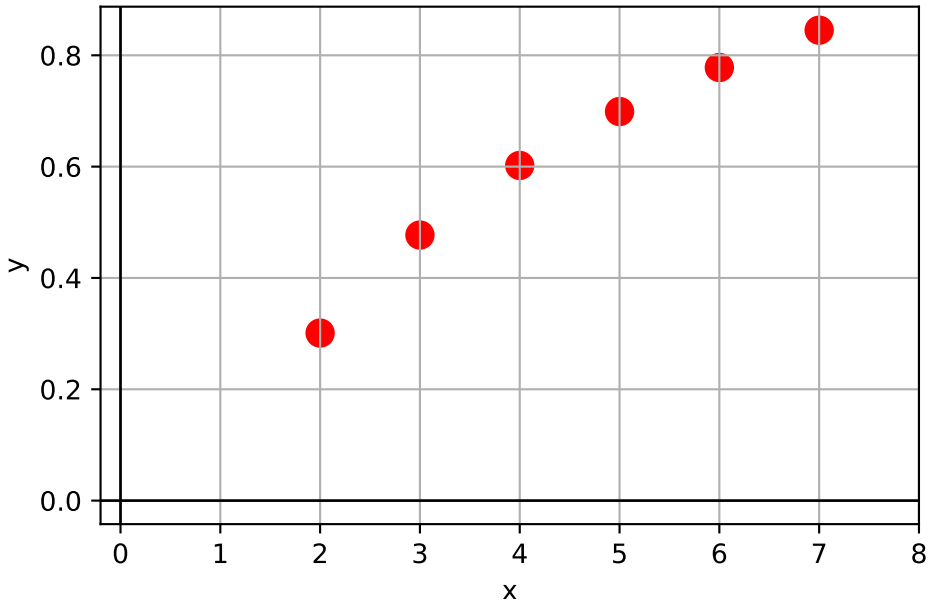
- Sea $f(x)$ una función desconocida de la cual se tienen los valores tabulados $(x_i, f(x_i))$ que se presentan a continuación, junto con una representación gráfica de los mismos:

i	x_i	$f(x_i)$
0	2	0.3010
1	3	0.4771
2	4	0.6021
3	5	0.6990
4	6	0.7781
5	7	0.8451

```
# Crear un DataFrame con los datos
datos = pd.DataFrame({'x': range(2, 8), 'y': [0.3010, 0.4771, 0.6021, 0.6990, 0.7781, 0.8451]

# Crear el gráfico
plt.figure()
plt.scatter(datos['x'], datos['y'], color='red', s=100)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-0.2, 8)
```

```
(-0.2, 8.0)
```

```
plt.grid(True)
plt.show()
```



- La correspondiente tabla de diferencias ordinarias es (verificar los cálculos a mano):

i	x_i	$f(x_i)$	Δ_i^1	Δ_i^2	Δ_i^3	Δ_i^4	Δ_i^5
0	2	**0.3010**					
			0.1761				
1	3	0.4771		**-0.0511**			
			0.1250		**0.0230**		
2	4	0.6021		-0.0281		**-0.0127**	
			0.0969		0.0103		**0.0081**
3	5	0.6990		-0.0178		-0.0046	
			0.0791		0.0057		
4	6	0.7781		-0.0121			
			0.0670				
5	7	0.8451					

- Los valores en negrita son los que utiliza la fórmula.
- También usamos la función `diferencias()` para obtenerla, que devuelve las diferencias ordinarias si no se provee un vector de x_i:

```
fx = np.array([.3010, .4771, .6021, .6990, .7781, .8451])
diferencias(fx)
```

```
array([[ 0.301 ,  0.1761, -0.0511,  0.023 , -0.0127,  0.0081],
       [ 0.4771,  0.125 , -0.0281,  0.0103, -0.0046,     nan],
       [ 0.6021,  0.0969, -0.0178,  0.0057,     nan,     nan],
       [ 0.699 ,  0.0791, -0.0121,     nan,     nan,     nan],
       [ 0.7781,  0.067 ,     nan,     nan,     nan,     nan],
       [ 0.8451,     nan,     nan,     nan,     nan,     nan]])
```

- Vamos a aproximar $f(2.3)$ con un polinomio de grado 5 que pase por todos los puntos provistos:
  - $h = 1$ (espaciado)
  - $x = 2.3$
  - $x_0 = 2$
  - $s = \frac{x - x_0}{h} = \frac{2.3-2}{1} = 0.3$

$$P_5(x) = \sum_{k=0}^{5} \binom{s}{k} \Delta_0^k$$

$$= f(x_0) + s\Delta_0^1 + \frac{s(s-1)}{2!}\Delta_0^2 + \frac{s(s-1)(s-2)}{3!}\Delta_0^3 + \ldots$$

$$= 0.3010$$

$$+ 0.3 \times 0.1761$$

$$+ \frac{0.3(0.3-1)}{2!}(-0.0511)$$

$$+ \frac{0.3(0.3-1)(0.3-2)}{3!}0.0230$$

$$+ \frac{0.3(0.3-1)(0.3-2)(0.3-3)}{4!}(-0.0127)$$

$$+ \frac{0.3(0.3-1)(0.3-2)(0.3-3)(0.3-4)}{5!}0.0081$$

$$\implies P_5(2.3) = 0.3613$$

- En Python vamos a usar la función `newton_adelante()` (provista):

```
newton_adelante(fx, s = 0.3)
```

```
0.36131479777500003
```

- Si en la fórmula anterior reemplazamos s por su defición $(x-2)/1$ y con mucha paciencia operamos en términos de x, podemos encontrar una expresión explícita para el polinomio.
- Como vimos antes, en Python lo hacemos con:

```
x = np.arange(2, 8)
fx = np.array([.3010, .4771, .6021, .6990, .7781, .8451])

# Coeficientes del polinomio (menor a mayor grado)
coefs_g5 = npol.polyfit(x, fx, deg = len(x) - 1)
print(coefs)
```

```
[ 1.         -0.1379019  -0.49343429  0.10416762]
```

```
# Convertir esos coeficientes en una función polinómica
poli_g5 = npol.Polynomial(coefs_g5)
# Mostrar la expresion del polinomio
print(poli_g5)
```

```
-0.4086 + 0.55547833·x - 0.13677083·x² + 0.02170417·x³ - 0.00187917·x +
(6.75e-05)·x
```

```
# Evaluar el polinomio interpolante en el punto 2.3
poli_g5(2.3)
```

```
0.3613147977750001
```

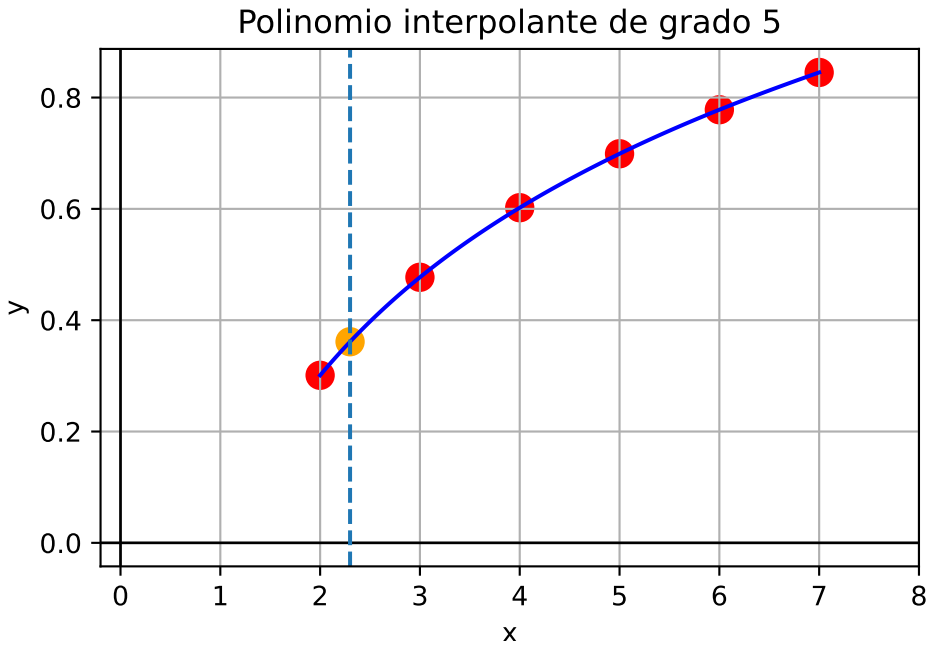
- Luego, lo podemos graficar:

```
# Crear un rango de valores para x
rango_x = np.linspace(2, 7, 100)

# Calcular los valores correspondientes
y_g5 = poli_g5(rango_x)

# Crear el gráfico
plt.figure()
plt.scatter(datos['x'], datos['y'], color='red', s=100)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.xlabel('x')
plt.ylabel('y')
```

```
plt.xlim(-0.2, 8)
```

```
(-0.2, 8.0)
```

```
plt.grid(True)
plt.plot(rango_x, y_g5, color='blue')
plt.title("Polinomio interpolante de grado 5")
plt.scatter(2.3, poli_g5(2.3), color = "orange", s = 100)
plt.axvline(x=2.3, linestyle='--')
plt.show()
```



- Como el polinomio de grado 5 que pasa por 6 puntos es único, la expresión hallada coincide a la que se hubiese obtenido con la fórmula general de Newton o con la de Lagrange.
- ¿Es necesario haber empleado un polinomio de grado 5?
- Viendo el gráfico podemos pensar que una función cuadrática puede proveer un buen ajuste.
- Eso tiene la ventaja de ahorrar cálculos y evita acumular errores por redondeo.
- Por eso, podríamos haber propuesto inicialmente un polinomio interpolante de grado 2, que pasa por los primeros 3 puntos, usando hasta la diferencia de segundo orden:
  - $h = 1$ (espaciado)
  - $x = 2.3$
  - $x_0 = 2$

- $s = \frac{x - x_0}{h} = \frac{2.3 - 2}{1} = 0.3$

$$P_2(x) = \sum_{k=0}^{2} \binom{s}{k} \Delta_0^k$$

$$= f(x_0) + s\Delta_0^1 + \frac{s(s-1)}{2!}\Delta_0^2$$

$$= 0.3010 + 0.3 \times 0.1761 + \frac{0.3(0.3-1)}{2!}(-0.0511)$$

$$\implies P_2(2.3) = 0.3592$$

```
# Polinomio interpolante de grado 2 que pasa por los primeros 3 ptos
poli_g2 = npol.Polynomial(npol.polyfit(x[:3], fx[:3], deg = 2))

# Mostrar la expresion del polinomio
print(poli_g2)
```

```
-0.2045 + 0.30385·x - 0.02555·x²
```

```
# Evaluar el polinomio interpolante en el punto 2.3
poli_g2(2.3)
```

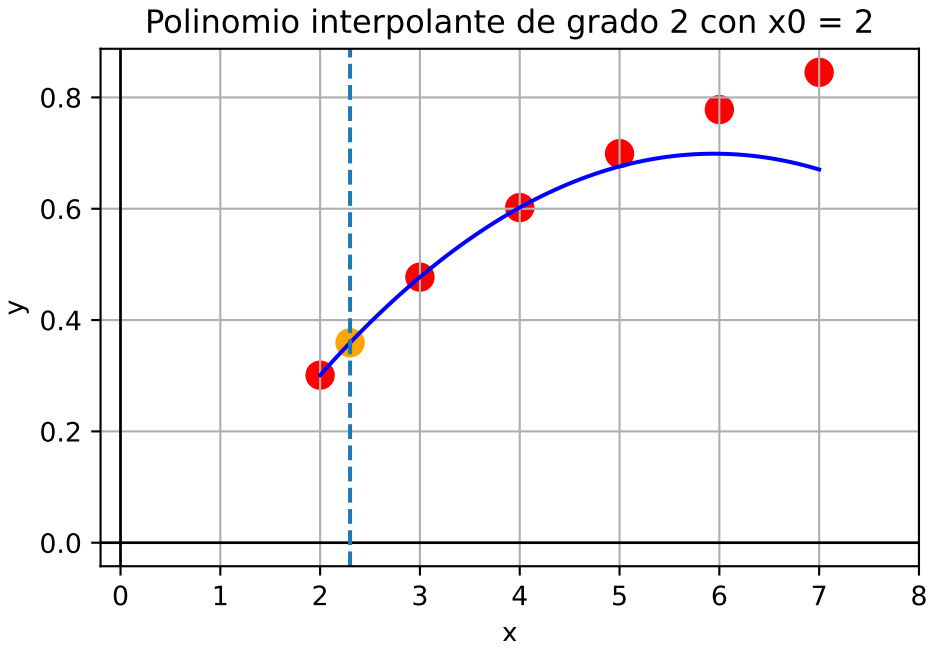
```
0.35919549999999956
```

```
# Graficarlo
y_g2 = poli_g2(rango_x)
plt.figure()
plt.scatter(datos['x'], datos['y'], color='red', s=100)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-0.2, 8)
```

```
(-0.2, 8.0)
```

```
plt.grid(True)
plt.plot(rango_x, y_g2, color='blue')
plt.title("Polinomio interpolante de grado 2 con x0 = 2")
plt.scatter(2.3, poli_g2(2.3), color = "orange", s = 100)
plt.axvline(x=2.3, linestyle='--')
```

```
plt.show()
```



- **Observación**: es claro que no podemos esperar que este polinomio provea aproximaciones razonables para x cercano al final del rango.
- Lo bueno de este método, a diferencia del del Lagrange y al igual que la fórmula general, es que permite empezar por una interpolación de grado 1 e ir generado las aproximaciones de grado superior al agregar de a uno los términos siguientes, aprovechando los cálculos anteriores:

Grado	$P_n(2.3) =$
1	$f(x_0) + s\Delta_0^1 = 0.3538$
2	$0.3538 + \frac{s(s-1)}{2!}\Delta_0^2 = 0.3592$
3	$0.3592 + \frac{s(s-1)(s-2)}{3!}\Delta_0^3 = 0.3606$
4	$0.3606 + \frac{s(s-1)(s-2)(s-3)}{4!}\Delta_0^4 = 0.3611$
5	$0.3611 + \frac{s(s-1)(s-2)(s-3)(s-4)}{5!}\Delta_0^5 = 0.3613$

- En Python:

```
for grado in range(1, 6):
  print(newton_adelante(fx, s = 0.3, g = grado))
```

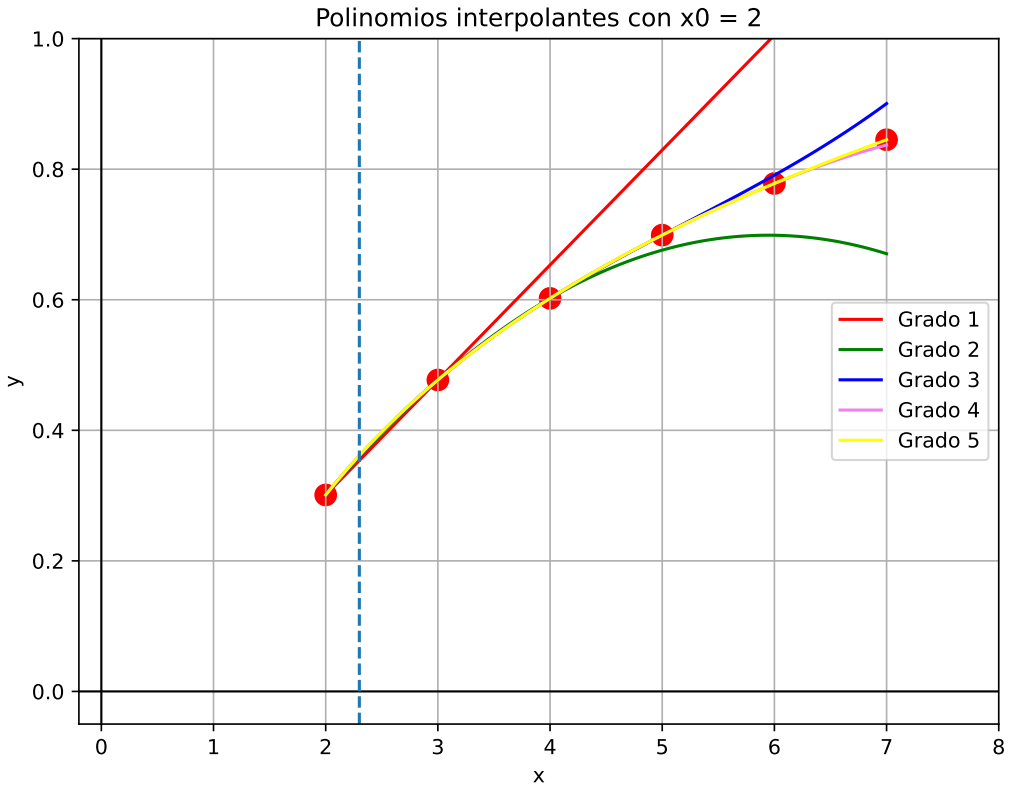
```
0.35383
0.3591955
```

```
0.360564
0.36107406375
0.36131479777500003
```

- La primera aproximación es una interpolación lineal y pasa por los primeros dos puntos; la segunda es cuadrática y pasa por los primeros tres puntos, etc.

```
(-0.2, 8.0)
```

```
(-0.05, 1.0)
```



- La función que generó la tabla de este ejemplo es el logaritmo en base 10, es decir, que el valor exacto es $f(2.3) = log(2.3) = 0.3617$.
- Con esta información, podemos calcular el Error Relativo que resulta de hacer las aproximaciones anteriores:

Grado	$P_n(2.3) =$	ER
1	0.3538	0.0218
2	0.3592	0.0070

Grado	$P_n(2.3) =$	ER
3	0.3606	0.0032
4	0.3611	0.0018
5	0.3613	0.0012

6.3.5. Otras fórmulas de interpolación de Newton

- Así como dijimos que cualquier valor x puede ser expresado como $x = x_0 + sh$, también podemos decir que cualquier valor x puede ser expresado como $x = x_n + sh$ con s negativo y los nodos se pueden escribir como $x_i = x_n - (n-i)h$.
- Teniendo en cuenta esto y haciendo varios reemplazos y pasos, el polinomio (6.2) queda así:

$$\begin{aligned} P_n(x) &= f(x_n) - (-s)\Delta^1_{n-1} + \frac{-s(-s-1)}{2!}\Delta^2_{n-2} - \frac{-s(-s-1)(-s-2)}{3!}\Delta^3_{n-3} + ... \\ &= \sum_{k=0}^{n}(-1)^k\binom{-s}{k}\Delta^k_{n-k} \end{aligned}$$

- Esta es la **fórmula de interpolación de Newton con diferencias hacia atrás**, porque utiliza la última diagonal desde abajo a la izquierda hacia arriba a la derecha de la **tabla de diferencias ordinarias**.
- Hay que observar que la fórmula empieza con el valor de $f(x_n)$ y luego continúa empleando la última diferencia de cada orden.
- No realizaremos ejemplos con esta forma de interpolación.
- Sin embargo, al tener dos fórmulas de diferencias ordinarias para hacer interpolaciones, cabe preguntarse: ¿en qué se diferencian? ¿Cuál usar?
- Ambas fórmulas son expresiones diferentes del **mismo polinomio de grado** n, por lo tanto, es lo mismo usar una u otra.
- No obstante, sus resultados pueden diferir por causa de los **errores de redondeo** propios de la aritmética finita con la que se realizan los tantísimos cálculos que hay que hacer y que son distintos en una y otra fórmula.
- Si tenemos que interpolar para un x cercano al comienzo de la tabla (es decir, más bien cerca de x_0), conviene utilizar la fórmula hacia adelante. La misma empieza usando $f(x_0)$, y estando x cerca de x_0, es de esperar que $f(x_0)$ esté cerca de $f(x)$, por lo cual es un buen punto de partida.
- En cambio, si tenemos que interpolar para un x cercano al final de la tabla, por la misma razón conviene utilizar la fórmula hacia atrás.
- *Atención*: si en lugar de usar las fórmulas completas, empleamos menos términos para interpolar con polinomios de menor grado (es decir, que pasen por algunos pero no todos

los x_i), entonces los polinomios que resulten de una u otra fórmula ya no serán equivalentes (porque la fórmula hacia adelante usará los primeros puntos tabulados, mientras que la fórmula hacia atrás usará los últimos).

- ¿Y si el valor de x está más bien cerca de mitad de tabla?...
- Las fórmulas de diferencias hacia adelante y hacia atrás de Newton no son adecuadas para aproximar $f(x)$ cuando x se encuentra cerca del centro de la tabla.
- Existen varias fórmulas de diferencias divididas para este caso, cada una más ventajosa que otras para distintas situaciones.
- Estos métodos reciben el nombre de **fórmulas de diferencias centradas**.
- Uno de ellos se conoce como **fórmula de Stirling**.
- La fórmula considera que x_0 es el nodo central y entonces hay que "cambiarle el nombre" a los valores de la tabla de diferencias ordinarias (pero son los mismos):

x	$f(x)$	Primeras diferencias	Segundas diferencias	Terceras diferencias	Cuartas diferencias
x_{-2}	$f(x_{-2})$	Δ^1_{-2}			
x_{-1}	$f(x_{-1})$	Δ^1_{-1}	Δ^2_{-2}	Δ^3_{-2}	
x_0	$f(x_0)$	Δ^1_0	Δ^2_{-1}	Δ^3_{-1}	Δ^4_{-2}
x_1	$f(x_1)$	Δ^1_1	Δ^2_0		
x_2	$f(x_2)$				

- El polinomio de Stirling usa las diferencias que están pintadas, pero no consideraremos su fórmula.
- En resumen, la siguiente tabla de diferencias ordinarias tiene indicadas cuáles son las que se usan en un polinomio de interpolación con las fórmulas de diferencias hacia adelante, hacia atrás y centradas.
- Si se construye un polinomio de grado n, los tres polinomios son el mismo.
- Dependiendo de dónde se encuentre el punto x a interpolar, conviene usar una u otra para disminuir errores de redondeo.

x	$f(x)$	Primeras diferencias	Segundas diferencias	Terceras diferencias	Cuartas diferencias
x_0	$f(x_0)$	Δ^1_0			
x_1	$f(x_1)$	Δ^1_1	Δ^2_0	Δ^3_0	
x_2	$f(x_2)$	Δ^1_2	Δ^2_1	Δ^3_1	Δ^4_0
x_3	$f(x_3)$	Δ^1_3	Δ^2_2		
x_4	$f(x_4)$				

Hacia adelante
Hacia atrás
Centradas

6.4. Observaciones finales

- Un polinomio de grado n ajustado a $n+1$ puntos es único.
- El polinomio de interpolación se puede expresar en varias formas distintas, pero todas son equivalentes por el punto anterior.
- Si tenemos $n+1$ puntos podemos calcular n columnas de diferencias hacia adelante.
- Si la función $f(x)$ que dio lugar a la tabla es un polinomio de orden q, entonces la columna para la diferencia de orden q es constante y las siguientes columnas son todas nulas.
- Por lo tanto, si en el proceso de obtención de las diferencias sucesivas de una función, las diferencias de orden q se vuelven constantes (o aproximadamente constantes), sabemos que los datos provienen exactamente (o muy aproximadamente) de un polinomio de orden q.
- Errores de redondeo podrían hacer que a pesar de que los datos provengan de un polinomio, no encontremos diferencias constantes.
- Si una función se aproxima mediante un polinomio de interpolación, no hay garantía de que dicho polinomio converja a la función exacta al aumentar el número de datos. En general, la interpolación mediante un polinomio de orden grande debe evitarse o utilizarse con precauciones extremas.
- Eso se debe a que los polinomios de orden superior pueden oscilar erráticamente; es decir, una fluctuación menor sobre una pequeña parte del intervalo puede inducir fluctuaciones grandes sobre todo el rango (ver ejemplo en la figura 3.14).
- Aunque no existe un criterio para determinar el orden óptimo del polinomio de interpolación, generalmente se recomienda utilizar uno con orden relativamente bajo en un pequeño rango de x.

7 Aproximación polinomial - Parte 2: Derivación e integración

7.1. Diferenciación numérica

- Nos enfrentamos al problema de aproximar el valor de la derivada de una función $f(x)$.
- Por definición, la derivada de la función f en x_0 es:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

- Esta fórmula proporciona una forma obvia de generar una aproximación para $f'(x_0)$: simplemente hay tomar h pequeño y calcular:

$$\frac{f(x_0 + h) - f(x_0)}{h}$$

- Esta idea sencilla tiene el problema de enfrentarse a errores de redondeo (ya que sabemos que las divisiones por números pequeños son operaciones "delicadas").
- Sin embargo, es un buen punto de partida y coincide con la idea de derivar la expresión del polinomio de interpolación de Lagrange que pasa por los puntos $(x_0, f(x_0))$ y $(x_1, f(x_1))$, considerando $x_1 = x_0 + h$.
- Recordemos la interpolación lineal de Lagrange:

$$P_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1)$$

$$= \frac{x - x_0 - h}{-h} f(x_0) + \frac{x - x_0}{h} f(x_0 + h)$$

- Ya estudiamos que disponemos de una fórmula para el error de truncamiento (o aproximación) al utilizar polinomios interpolantes. Cuando el polinomio es de grado $n = 1$, dicha fórmula queda como:

$$\frac{(x - x_0)(x - x_0 - h)}{2} f''(\xi) \qquad \xi \text{ entre } x_0 \text{ y } x_1 = x_0 + h$$

- Entonces podemos escribir:

$$f(x) = P_1(x) + Error$$

$$= \frac{x - x_0 - h}{-h} f(x_0) + \frac{x - x_0}{h} f(x_0 + h) + \frac{(x - x_0)(x - x_0 - h)}{2} f''(\xi)$$

- Si derivamos tenemos:

$$f'(x) = -\frac{1}{h} f(x_0) + \frac{1}{h} f(x_0 + h) + D_x\Big[\frac{(x - x_0)(x - x_0 - h)}{2} f''(\xi)\Big]$$

$$= \frac{f(x_0 + h) - f(x_0)}{h} + D_x\Big[\frac{(x - x_0)(x - x_0 - h)}{2} f''(\xi)\Big]$$

- Borrando el término relacionado con ξ obtenemos la aproximación propuesta inicialmente:

$$f'(x) \approx \frac{f(x_0 + h) - f(x_0)}{h}$$

7.1.1. Fórmula general

- Esta idea se puede generalizar para obtener las **fórmulas generales de la aproximación a la derivada**:

Definición: si $x_0, x_1, ..., x_n$ puntos distintos en algún intervalo $[a, b]$ y f tiene derivadas continuas hasta de orden $n + 1$ en dicho intervalo, sabemos por la interpolación de Lagrange que:

$$f(x) = \sum_{k=0}^{n} f(x_k) L_k(x) + \frac{(x - x_0)...(x - x_n)}{(n + 1)!} f^{(n+1)}(\xi) \qquad L_k(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{x - x_i}{x_k - x_i}$$

para algún $\xi \in [a, b]$.
Al derivar esta ecuación obtenemos:

$$f'(x) = \sum_{k=0}^{n} f(x_k) L_k'(x) + D_x\Big[\frac{(x - x_0)...(x - x_n)}{(n + 1)!} f^{(n+1)}(\xi)\Big]$$

En general no se tiene conocimiento sobre $D_x(f^{(n+1)}(\xi))$ por lo cual no se puede acotar el error de truncamiento, pero si x es uno de los nodos x_j, entonces se demuestra que la fórmula queda igual a:

$$f'(x_j) = \underbrace{\sum_{k=0}^{n} f(x_k)L'_k(x)}_{\text{Aproximación}} + \underbrace{\frac{f^{(n+1)}(\xi)}{(n+1)!}\prod_{\substack{k=0\\k\neq j}}^{n}(x_j - x_k)}_{\text{Error}}$$

y es llamada **fórmula de $n+1$ puntos para aproximar $f'(x_j)$**.
De esta forma, se puede acotar el error si a la derivada la calculamos en algún nodo del polinomio interpolante x_j, pero nada impide utilizar la fórmula en otro caso, si estamos dispuestos a trabajar sin una cota para el error.

- En general, el uso de más puntos de evaluación en la ecuación anterior produce mayor precisión, pero el número de evaluaciones funcionales y el crecimiento del error de redondeo disuaden un poco de esto.
- Las fórmulas más comunes son las de tres y cinco puntos de evaluación. Veremos sólo las de 3.

7.1.2. Fórmulas de tres puntos

- Cuando se consideran tres puntos de evaluación, a partir del polinommio interpolante de grado de 2 de Lagrange, la fórmula anterior queda igual a:

$$\begin{aligned} f'(x_j) = f(x_0)&\frac{2x_j - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)} \\ &+ f(x_1)\frac{2x_j - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)} \\ &+ f(x_2)\frac{2x_j - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)} \\ &+ \frac{f^{(3)}(\xi)}{6}\prod_{\substack{k=0\\k\neq j}}^{2}(x_j - x_k) \end{aligned}$$

- Esto se simplifica mucho si los nodos están igualmente espaciados, de modo que $x_1 = x_0 + h$ y $x_2 = x_0 + 2h$.
- Implementando dichos reemplazos y aplicando cambios de variables, a partir de lo anterior se deducen dos fórmulas de tres puntos para aproximar $f'(x_0)$:

Fórmula del extremo de tres puntos:

$$f'(x_0) = \frac{-3f(x_0) + 4f(x_0 + h) - f(x_0 + 2h)}{2h} + \frac{h^2}{3}f^{(3)}(\xi), \qquad \xi \text{ entre } x_0 \text{ y } x_0 + 2h$$

Fórmula del punto medio de tres puntos:

$$f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h} - \frac{h^2}{6} f^{(3)}(\xi), \qquad \xi \text{ entre } x_0 - h \text{ y } x_0 + h$$

- La fórmula del extremo se usa para aproximar la derivada cuando x_0 es uno de los extremos del intervalo (h puede ser negativo), mientras que la otra cuando x_0 es el punto medio.
- La fórmula del punto medio presenta un error que es la mitad del de la fórmula del extremo y además requiere que f se evalúe solamente en dos puntos.

7.1.3. Inestabilidad del error de redondeo

- Para reducir el error de truncamiento necesitamos reducir h.
- Pero se demuestra que conforme h se reduce, el error de redondeo crece.
- En la práctica, entonces, casi nunca es ventajoso dejar que h sea demasiado pequeña, porque en este caso, el error de redondeo dominará los cálculos.
- La diferenciación numérica es **inestable** ya que los valores de h necesarios para reducir el error de truncamiento causan que el error de redonde crezca.

7.2. Integración numérica

- A menudo surge la necesidad de evaluar la integral definida de una función que no tiene una antiderivada o cuya antiderivada no es fácil de obtener (por ejemplo, para calcular probabilidades bajo la distribución normal).
- En estos casos se puede aproximar el valor de la integral mediante los **métodos de cuadratura**.

Definición: los **métodos de cuadratura** se utilizan para aproximar la integral definida $\int_a^b f(x)dx$ mediante una suma $\sum_{k=0}^{n} a_k f(x_k)$.

- La idea básica es seleccionar un conjunto de nodos $x_0, x_1, ..., x_n$ en el intervalo $[a, b]$ e integrar el polinomio interpolante de Lagrange que pase por dichos nodos.
- Recordemos la expresión del polinomio de Lagrange junto con su término de error:

$$f(x) = P_n(x) + Error$$

$$= \sum_{k=0}^{n} f(x_k) L_k + \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{k=0}^{n} (x - x_k) \qquad \xi \in (a, b) \qquad L_k = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{x - x_i}{x_k - x_i}$$

- Integrando nos queda:

$$\int_a^b f(x)dx = \underbrace{\sum_{k=0}^{n} \Big(\overbrace{\int_a^b L_k dx}^{a_k} \Big) f(x_k)}_{\text{Fórmula de cuadratura}} + \underbrace{\frac{1}{(n+1)!} \int_a^b f^{(n+1)}(\xi) \prod_{k=0}^{n} (x - x_k) dx}_{\text{Error de aproximación}}$$

- $\int_a^b L_k dx$ es fácil de hallar porque se trata de un polinomio de grado n que depende de los nodos observados x_i.
- La fórmula anterior es general para cualquier cantidad de nodos n, sean equidistantes o no.
- Cuando se trabaja con nodos igualmente espaciados, esta expresión da lugar a conjunto de fórmulas conocidas como **Fórmulas de integración de Newton-Cotes**.

7.2.1. Fórmulas comunes y cerradas de Newton-Cotes

- La denominación de "cerradas" hace referencia a que los extremos del intervalo $[a, b]$ se incluyen como nodos (están las fórmulas "abiertas" pero no las veremos).
- La denominación de "comunes" distingue a estas fórmulas de las "compuestas" que veremos en la próxima sección.
- Vamos a ver tres fórmulas, que resultan de simplificar la expresión anterior para los siguientes casos particulares:
  1. Cuando se consideran sólo dos nodos en el intervalo $[a, b]$ (*regla trapezoidal*).
  2. Cuando se consideran tres nodos equidistantes en el intervalo $[a, b]$ (*regla de Simpson*).
  3. Cuando se consideran cuatro nodos equidistantes en el intervalo $[a, b]$ (*regla de tres octavos de Simpson*).

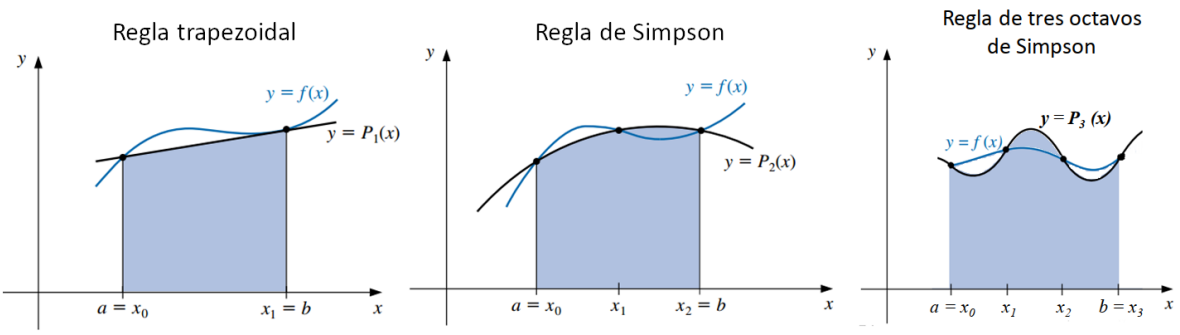


Figure 7.1: El área sombreada es el resultado de la aproximación.

7.2.1.1. Regla trapezoidal

- Se consideran sólo dos nodos en el intervalo $[a, b]$.

- En este caso se tiene $a = x_0$ y $b = x_1$ y llamamos $h = x_1 - x_0$.
- $P_n(x)$ es la recta que pasa por los puntos $(x_0, f(x_0))$ y $(x_1, f(x_1))$.
- La fórmula es:

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{2}[f(x_0) + f(x_1)] - \underbrace{\frac{h^3}{12}f''(\xi)}_{\text{Error}}$$

- Recibe el nombre de regla trapezoidal porque cuando f es una función con valores positivos, $\int_a^b f(x)dx$ se aproxima mediante el área de un trapecio.

7.2.1.2. Regla de Simpson

- Resulta de la integración sobre $[a, b]$ del segundo polinomio de Lagrange con nodos igualmente espaciados $x_0 = a$, $x_1 = x_0 + h$ y $x2 = x_0 + 2h = b$, es decir: $h = (b - a)/2$.

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] - \underbrace{\frac{h^5}{90}f^{(4)}(\xi)}_{\text{Error}}$$

- El término de error queda en función de la cuarta derivada de f, por lo que da resultados exactos cuando $f(x)$ es un polinomio de grado 3 o menos.

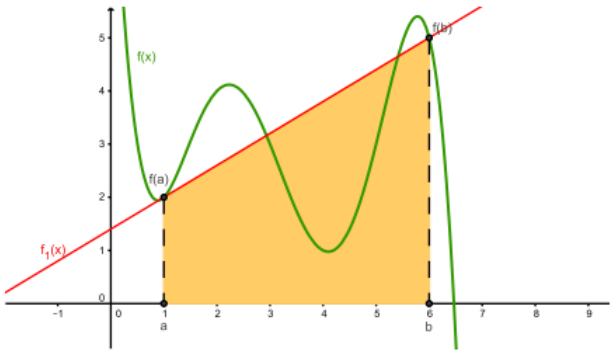
7.2.1.3. Regla de tres octavos Simpson

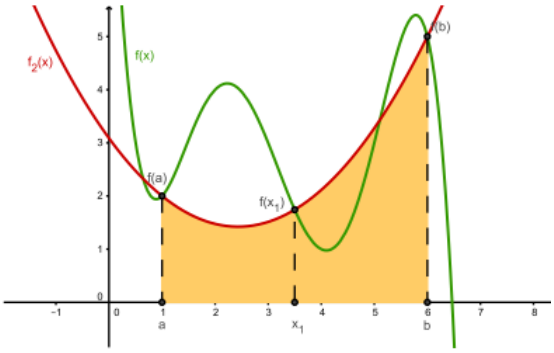
- Resulta de considerar cuatro nodos igualmente espaciados $x_0 = a$, $x_1 = x_0 + h$, $x2 = x_0 + 2h$ y $x_3 = x_0 + 3h = b$ e integrar el polinomio de Lagrange de grado 3 que pasa por ellos
- En este caso: $h = (b - a)/3$.

$$\int_{x_0}^{x_3} f(x)dx = \frac{3h}{8}[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)] - \underbrace{\frac{3h^5}{80}f^{(4)}(\xi)}_{\text{Error}}$$

7.2.2. Fórmulas compuestas de Newton-Cotes

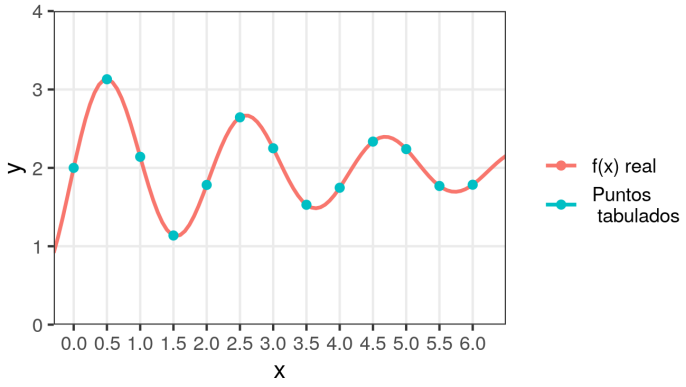
- En general, el uso de las fórmula de Newton-Cotes es inapropiado sobre largos intervalos de integración, porque se pueden dar situaciones como estas:



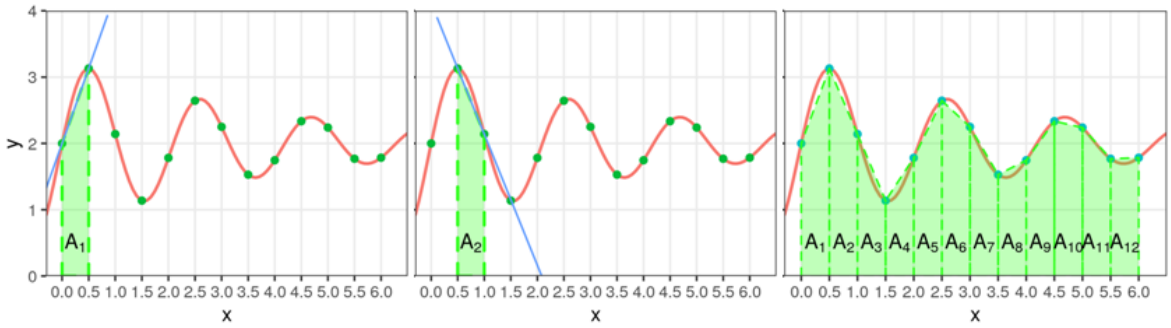


- Se podrían usar fórmulas de grado superior a las vistas, dividiendo al intervalo en más de 4 puntos e integrando el polinomio interpolante que pase por ellos, pero esto implica la necesidad de obtener coeficientes de cálculo cada vez más tedioso.
- Como alternativa se presentan las **fórmulas de integración compuestas**, que consisten en emplear las fórmulas de bajo orden de la sección anterior sucesivamente a lo largo del intervalo de interés.
- **Ejemplo**. Se tienen los siguientes valores tabulados de $f(x)$ y se desea hallar su integral entre 0 y 6. La curva roja es la verdadera función $f(x)$ que originó la tabla, la cual suponemos desconocida o difícil de integrar. En este caso tenemos $h = 0.5$.

x	$f(x)$
0.0	2.00
0.5	3.13
1.0	2.14
1.5	1.14
2.0	1.78
2.5	2.64
3.0	2.25
3.5	1.53
4.0	1.75
4.5	2.34
5.0	2.24
5.5	1.77
6.0	1.78



7.2.2.1. Regla trapecial compuesta



- Se aplica la regla trapecial entre cada par consecutivo de nodos x_i y se suman los resultados.
- En el ejemplo, para el primer intervalo entre x_0 y x_1:

$$\int_{x_0}^{x_1} f(x)dx \approx \frac{h}{2}[f(x_0)+f(x_1)] \implies \int_{0}^{0.5} f(x)dx \approx \frac{0.5}{2}(3.13+2) = 1.2825 = A_1$$

- Geométricamente, esto equivale al área A_1 del trapecio formado por la recta de interpolación y el eje de las abscisas, entre x_0 y x_1.
- De manera semejante, se puede emplear la interpolación lineal para obtener una aproximación de la integral entre x_1 y x_2:

$$\int_{x_1}^{x_2} f(x)dx \approx \frac{h}{2}(f(x_1)+f(x_2)) = 1.3175 = A_2$$

- Y sucesivamente para todos los intervalos entre x_{i-1} y x_i:

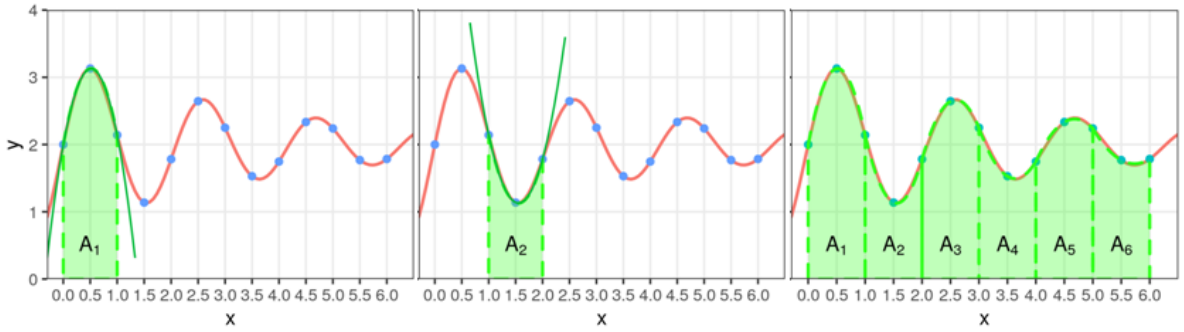
$$\int_{x_{i-1}}^{x_i} f(x)dx \approx \frac{h}{2}(f(x_{i-1})+f(x_i)) = A_i \quad i = 1, \cdots, n$$

- La suma de las áreas de los trapecios A_i resulta ser la aproximación para la integral entre x_0 y x_n:

$$\int_{x_0}^{x_n} f(x)dx \approx \sum_{i=1}^{n} A_i = \sum_{i=1}^{n} \frac{h}{2}(f(x_{i-1}) + f(x_i)) = \frac{h}{2}\Big[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)\Big]$$

- El error está dado por: $-\frac{b-a}{12}h^2 f''(\mu)$, con $\mu \in (a, b)$.
- Cuanto menor sea el ancho de los intervalos h y más se acerque $f(x)$ a una recta dentro de dichos intervalos, mejor será la aproximación así obtenida.
- En el ejemplo el resultado es $\int_0^6 f(x)dx \approx 12.3000$.
- El valor exacto es: $\int_0^6 f(x)dx = 12.2935$, con lo cual el error relativo de la aproximación con la fórmula trapecial fue: $0.05\,\%$.

7.2.2.2. Regla de Simpson compuesta



- Se aplica la regla de Simpson tomando de a tres nodos x_i y se suman los resultados. Es decir, se usan integrales de polinomios de grado 2.
- Requiere que la cantidad $n + 1$ de nodos sea impar.
- En el ejemplo y para el primer tramo, obtenemos el área encerrada entre el eje de las abscisas, x_0 y x_2 y el polinomio integrador que pasa por $(x_0, f(x_0))$, $(x_1, f(x_1))$ y $(x_2, f(x_2))$:

$$\int_{x_0}^{x_2} f(x)dx \approx \frac{h}{3}(f(x_0) + 4f(x_1) + f(x_2)) = 2.7766 = A_1$$

- De manera semejante, se puede emplear la interpolación cuadrática para obtener una aproximación de la integral entre x_2 y x_4:

$$\int_{x_2}^{x_4} f(x)dx \approx \frac{h}{3}(f(x_2) + 4f(x_3) + f(x_4)) = 1.4133 = A_2$$

- Y sucesivamente para todos los tramos:

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx \approx \frac{h}{3}(f(x_{i-1}) + 4f(x_i) + f(x_{i+1})) \quad i = 1, 3, 5, \cdots, n-1$$

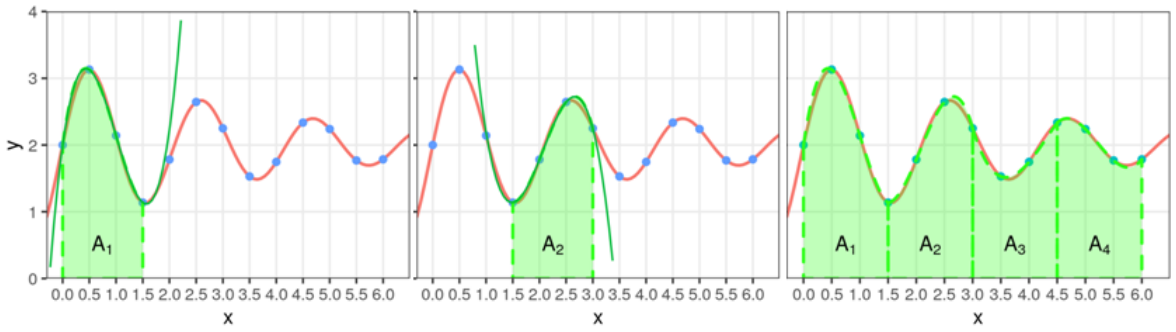
- La suma de estas áreas resulta ser la aproximación para la integral entre x_0 y x_n:

$$\int_{x_0}^{x_n} f(x)dx \approx \sum_{\substack{i=1 \\ i \ impar}}^{n-1} \frac{h}{3}\Big[f(x_{i-1}) + 4f(x_i) + f(x_{i+1})\Big]$$

$$= \frac{h}{3}\Big[f(a) + 2\underbrace{\sum_{i=1}^{n/2-1} f(x_{2i})}_{\substack{\text{suma con nodos} \\ \text{de subíndice par} \\ \text{entre 2 y n-2}}} + 4\underbrace{\sum_{i=1}^{n/2} f(x_{2i-1})}_{\substack{\text{suma con nodos} \\ \text{de subíndice impar} \\ \text{entre 1 y n-1}}} + f(b)\Big]$$

- El error está dado por: $-\frac{b-a}{180}h^4 f^{(4)}(\mu)$, con $\mu \in (a, b)$.
- En el ejemplo el resultado es 12.3833.
- El valor exacto es: $\int_0^6 f(x)dx = 12.2935$, con lo cual el error relativo de la aproximación con la fórmula compuesta de Simpson fue: 7.3 %.

7.2.2.3. Regla de tres octavos de Simpson compuesta



- Se aplica la regla de Simpson tomando de a cuatro nodos x_i y se suman los resultados. Es decir, se usan integrales de polinomios de grado 3.
- Es necesario que n sea múltiplo de 3, siendo n la cantidad de nodos menos uno (o bien, la cantidad de intervalos equiespaciados entre nodos).
- En el primer tramo con los 4 primeros valores de x, se obtiene:

$$\int_{x_0}^{x_3} f(x)dx \approx \frac{3h}{8}\Big(f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\Big) = 3.5531 = A_1$$

- Geométricamente, esto equivale al área encerrada entre el eje de las abscisas, x_0 y x_3 y el polinomio integrador que pasa por $(x_0, f(x_0))$, $(x_1, f(x_1))$, $(x_2, f(x_2))$ y $(x_3, f(x_3))$:

- De manera semejante, se puede emplear la interpolación cúbica para obtener una aproximación de la integral entre x_3 y x_6:

$$\int_{x_3}^{x_6} f(x)dx \approx \frac{3h}{8}\Big(f(x_3) + 3f(x_4) + 3f(x_5) + f(x_6)\Big) = 3.1219 = A_2$$

- Y sucesivamente para todos los intervalos:

$$\int_{x_i}^{x_{i+3}} f(x)dx \approx \frac{3h}{8}\Big(f(x_i) + 3f(x_{i+1}) + 3f(x_{i+2}) + f(x_{i+3})\Big) \quad i = 0, 3, 6, \cdots, n-3$$

- De modo que la suma de estas áreas resulta ser la aproximación para la integral entre x_0 y x_n:

$$\int_{x_0}^{x_n} f(x)dx \approx \frac{3h}{8}\sum_{i=1}^{n/3}\Big(f(x_{3i-3}) + 3f(x_{3i-2}) + 3f(x_{3i-1}) + f(x_{3i})\Big)$$

- El error está dado por: $-\frac{b-a}{80}h^4 f^{(4)}(\mu)$, con $\mu \in (a, b)$.
- En el ejemplo el resultado es 12.4088.
- El valor exacto es: $\int_0^6 f(x)dx = 12.2935$, con lo cual el error relativo de la aproximación con la fórmula compuesta de tres octavos de Simpson fue: 9.4 %.

7.2.2.4. Estabilidad del error de redondeo

- Una propiedad importante compartida por todas las técnicas de integración compuesta es su estabilidad respecto al error de redondeo.
- Es decir, sorprendentemente, el error de redondeo no depende del número de cálculos realizados al aplicar estas fórmulas (demostración en página 156).
- Esto no es verdad para los procedimientos de diferenciación numérica.

7.2.3. Métodos de cuadratura adaptable o integración adaptativa

- Las reglas compuestas de cuadratura vistas necesitan nodos equiespaciados.
- Generalmente, se usa un incremento pequeño h de manera uniforme en todo el intervalo de integración para garantizar una precisión global.
- Este proceso no tienen en cuenta el hecho de que en algunas porciones de la curva puedan aparecer oscilaciones más pronunciadas que en otras y, en consecuencia, requieran incrementos más pequeños para conseguir la misma precisión.
- Sería interesante disponer de un método que vaya ajustando el incremento de manera que sea menor en aquellas porciones de la curva en la que aparezcan oscilaciones más pronunciadas.
- Los métodos de cuadratura adaptable o integración adaptativa se encargan de implementar esto, pero no los estudiaremos.

7.2.4. Ejemplos en Python

- Importamos módulos necesarios:

```
from unidad6_funciones import *
import numpy as np
```

- Vamos a usar la función `newton_cotes()` para aplicar estas tres fórmulas de integración:

```
fx = np.array([2, 3.13, 2.14, 1.14, 1.78, 2.64, 2.25, 1.53, 1.75, 2.34, 2.24, 1.77, 1.78])

# Fórmula trapecial
newton_cotes(fx, h = 0.5, formula = "trapecio")
```

```
12.299999999999999
```

```
# Fórmula de Simpson de 1/3
newton_cotes(fx, h = 0.5, formula = "simpson")
```

```
12.383333333333333
```

```
# Fórmula de Simpson de 3/8
newton_cotes(fx, h = 0.5, formula = "tres_octavos")
```

```
12.408750000000001
```

```
# Proveyendo distintos valores de fx se pueden obtener otras aproximaciones
# Por ejemplo, fórmula del trapecio común entre x=0 y x=6 (ojo, el intervalo es
# muy ancho para esto)
newton_cotes(fx = np.array([2, 1.78]), h = 6, formula = "trapecio")
```

```
11.34
```