Introducción a Wolfram Mathematica

I. Objetivos

1. Explorar Wolfram Mathematica y su interfaz.
2. Ejecutar comandos básicos para familiarizarse con la sintaxis de cada uno de ellos.
3. Aprender a realizar gráficos y ajustes de datos.

II. Problema

Muchos problemas de carácter científico requieren el uso de herramientas computacionales para fácilitar el trabajo de análisis de datos, visualizar algún fenómeno o trabajar con expresiones númericas y simbólicas. Wolfram Mathematica es una herramienta versátil y robusta utilizada en una amplia variedad de campos científicos por el apoyo que ofrece para diferentes necesidades. Realizar modelos y ajustes, simulaciones, análisis de datos, resolución de ecuaciones, cálculo simbólico, aproximaciones númericas, gráficas, etc.

III. Introducción

Wolfram Mathematica es un programa de computación orientado a la resolución de problemas de carácter científico. Al igual que otros lenguajes de programación, nos da la libertad de definir nuestras própias funciones conforme a nuestra necesidad y el problema que deseemos resolver, asi como la visualización de gráficos y modelado de los mismos. Una de las áreas de resolución que nos ofrece y en la que nos vamos a centrar es en su apartado de cálculo multivariable y cálculo vectorial, el cual orientaremos a la solución de problemas de electrostática, tanto de forma numérica, simbólica y gráfica. No dejando de lado el apartado de laboratorio, se trabajaran ajustes de conjuntos de datos experimentales recogidos durante las prácticas de laboratorio, centrandonos en cómo realizar dichos ajustes y cómo gráficarlos.

Esta práctia cubrira lo básico, siendo esto la declaraciónn de variables, operaciones básicas, comandos para trabajar con expresiones algebraicas, derivación, integración, ajustes líneales y no líneales, creación de funciones, creación de Plots y comandos útiles para limpiar variables de forma local y global.

Mathematica cuenta con una colección de comandos con fácilidad de uso débido a su interfaz intuitiva y amigable al usuario, lo cual vuelve más eficiente nuestro trabajo. Mathematica también permite exportar o guardar nuestro trabajo en forma de un archivo notebook con extensión .nb por defecto, aunque permite exportar en otros formatos útiles como; .pdf, .html, .txt, .tex, entre otros.

IV. Definiendo variables

Para definir una variable o una cantidad escalar en mathematica, tenemos que asignarle un nombre y la cantidad que deseemos guardar en esa variable utilizando el operador de asignación.

- Mathematica cuenta con dos operadores de asignación, estos son $=$ y $:=$, el primero es un operador de asignación directa, recomendado para guardar valores fijos o constantes. El segundo operador es recomendable utilizarlo con expresiones dinámicas o dependientes de otros valores como las funciones, vectores o campos.
- Para definir una cantidad fijas o constantes, almacenamos un valor en una variable y esta será reconocida por el programa de forma global en nuestro notebook, por lo que al momento de volver a llamar a nuestra variable, mathematica nos devolvera directamente el valor almacenado en ella.

Se debe ser cuidadoso con las variables definidas ya que no podemos guardar más de un valor fijo en una misma variable, si esto sucede nuestro código dara respuestas erroneas o desplegara una advertencia de error.

Ejemplo:

```
numero = 3;
a = 2;
```

Listing 1: Definición de variables fijas

- Una buena práctica al momento de utilizar mathematica, es añadir ; al final de nuestros comandos, esto permite suprimir la salida de la celda, de forma que no se muestre el valor almacenado en la variable.

Hacer esto nos ahorrara mucho tiempo cuando compilemos código más extenso débido a que mathematica ahorrara recursos y sobre todo tiempo al no imprimir la salida de cada una de las celdas del notebook.

Ejemplo:

```
variable = 7
cantidad = 3
7
3
```

Listing 2: Deiniendo variables sin utilizar "; "

V. Limpiar variables

- Un comando sumamente útil cuando utilizamos variables es el comando **Clear[variable]**. Este comando permite limpiar una variable o eliminar el valor que almacena para poder reutilizarla asignandole otro valor.

Ejemplo:

```
v = 5
5
Clear[v]
v = 3
3
```

Listing 3: Limpiando variables para reutilizarlas

Cuando trabajamos con otras personas y tenemos que unir un trabajo, es complicado tener un control de las variables que utilizan los demás miembros del grupo. Lo recomandable en estos casos es limpiar las variables que nosotros utilizaremos antes de trabajar con ellas.

- El comando **Clear[variable1,variable2, ..., variableN]** permite limpiar todas las variables que utilizaremos y asi no tendremos problemas al juntar un trabajo.
- Otro comando útil de correr es **ClearAll["Global‘*"]** al inicio de nuestros notebook para que mathematica limpie cualquier valor residual.

VI. Operaciones básicas

Para realizar operaciones aritméticas básicas en Wolfram Mathematica, necesitamos utilizar los operadores correspondientes de suma, resta, multiplicación y división, respetando la jerarquía de operaciones y la sintaxis de mathematica.

- Cuando escribimos un comando, mathematica lo lee de izquierda a derecha por lo que necesitamos ser cuidadosos al momento de operar expresiones, es de buena práctica separar nuestras expresiones con paréntesis ( ) para no tener problemas con respuestas erroneas.

Ejemplo:

```
numero1 = 4;
numero2 = 5;

numero1 + numero2
9

numero2 - numero1
1
```

Listing 4: Suma y resta

- El operador designado para la multiplicación es * este se coloca entre las variables para multiplicarlas. Otra forma de hacerlo es dejar un espacio entre las variables y mathematica lo interpretara como una multiplicación, pero esto no es recomendable al trabajar con expresiones grandes.

Ejemplo:

```
x = 5;
y = 10;

x*y
50

x y
50
```

Listing 5: Multiplicación

- El operador designado para la división es la barra / , esta se coloca entre las expresiones que deseemos dividir, tenemos que ser cuidadosos con la sintaxis de mathematica débido a que si no agrupamos correctamente entre paréntesis las expresiones podemos obtener un resultado erroneo.

Ejemplo:

```
numerador = 15;
denominador = 3;

numerador / denominador
5
```

Listing 6: División

En el siguientes ejemplo podemos observar que el resultado es diferente si agrupamos y si no agrupamos las expresiones o variables utilizando paréntesis. En el primer comando, mathematica divide la variable *cantidad1* entre *cantidad2* y luego multiplica ese resultado por *cantidad3*. En el segundo comando, mathematica divide la variable *cantidad1* entre el resultado de multiplicar la variable *cantidad2* por la variable *cantidad3*.

```
cantidad1 = 4;
cantidad2 = 2;
cantidad3 = 8;

cantidad1 / cantidad2*cantidad3
16

cantidad1 / (cantidad2*cantidad3)
1/4
```

Listing 7: Importancia de agrupar términos

VII. De fracciones a decimales y viceversa

Al momento de trabajar expresiones con valores fijos, puede que obtengamos las respues de forma fraccionaria y no de forma decimal, dependiendo de lo que necesitemos, podemos utilizar comandos de mathematica para convertir un resultado a su equivalente en cantidad decimal.

- El comando **N[cantidad fraccionaria]** o la notación **(cantidad fraccionaria)** //**N** permite transformar una cantidad fraccionaria a su equivalente en número decimal.

- La doble barra // , operador postfijo, permite utilizar funciones de mathematica de forma más compacta, casi cualquier función se puede denotar de esta forma como las funciones trigonométricas, de simplificación, manipulación algebraica, entre otras.

Ejemplo:

```
N[1/7]
0.142857

(1/7) //N
0.142857

N[variable/cantidad]
2.33333

(variable/cantidad) //N
2.33333
```

Listing 8: Convirtiendo fracciones a decimale

- El comando **Rationalize[cantidad decimal]** o en notación compacta **(cantidad decimal)** //**Rationalize**, permite transformar una cantidad decimal a fracción.

Ejemplo:

```
Rationalize[2.5]
5/2

(2.5)//Rationalize
5/2
```

Listing 9: Convirtiendo decimales a fracciones

VIII. Potencias

- Para elevar a una potencia una cantidad o expresión, podemos utilizar la función **Power[variable, potencia]** que recibe la variable que deseamos elevar y la potencia a la que lo haremos.
- Otra forma de hacerlo es utilizando el operador ∧ es importante agrupar el término que será la potencia y el que será base para evitar errores.
- Mathematica cuenta con una función para calcular la raíz cuadrada **Sqrt[variable]** para interpretar raíces tendremos que elevar a potencias fraccionarias.

Ejemplo:

```
b = 2;

b^2
4
Power[b, 2]
4

4^(1/2)
2
Sqrt[4]
2
Power[4, 1/2]
2
```

Listing 10: Potencia de un número

IX. Álgebra

Mathematica cuenta con una gran colección de comandos específicamente para trabajar con expresiones algebraicas o polinómicas. Estos comandos nos permiten simplificar, factorizar, expandir, etc.

- El definir una expresión polinómica es similar a cuando definimos una variable fija, le asignamos un nombre y una expresión.

  Ejemplo:

```
expresion1 = x^2 - 2 x + 1;
expresion2  = 6*x^2 - 28 + 32 x;
```

Listing 11: Definiendo expresiones polinómicas

- El comando **Simplify[expresion]**, simplifica una expresión de forma directa y básica.

  Ejemplo:

```
Simplify[expresion1]
(-1 + x)^2
```

Listing 12: Simplificar una expresión

- **FullSimplify[expresion]** es un comando más robusto, este utiliza todos los métodos algebraicos posibles para simplificar una expresión. Útil cuando contamos con expresiones muy grandes.

  Ejemplo:

```
FullSimplify[expresion2*expresion1 - expresion1^2]
(-1 + x)^2 (-29 + x (34 + 5 x))
```

Listing 13: Simplificar expresiones grandes

- El comando **Factor[expresion]** permite factorizar la expresión no simplificarla, simplemente saca un factor en común.

  Ejemplo:

```
Factor[expresion2]
2 (-14 + 16 x + 3 x^2)
```

Listing 14: Factorizar una expresión

- El comando **Expand[expresion]** permite expandir expresiones que están elevadas a una potencia, maneja potencias enteras y fraccionarias (raíces).

  Ejemplo:

```
Expand[(x - 2)^7]
-128 + 448 x - 672 x^2 + 560 x^3 - 280 x^4 + 84 x^5 - 14 x^6 + x^7
```

Listing 15: Expandir una expresión

- El comando **ExpandAll[expresion]** es un comando más robusto, útil cuando trabajamos con expresiones grandes ya que expande todas las partes de una expresión.

  Ejemplo:

```
ExpandAll[(expresion1 - expresion2)^3 *(expresion1 + expresion2)^2]

17779581 - 102045258 x + 215818551 x^2 - 184113960 x^3 +
 15777770 x^4 + 50371044 x^5 - 3650442 x^6 - 7974376 x^7 -
 1879335 x^8 - 177450 x^9 - 6125 x^10
```

Listing 16: Expandiendo expresiones grandes

- La notación compacta que vimos anteriormente, también nos permite evaluar expresiones utilizando la sintaxis **/.x→ numero**.

  Ejemplo:

```
expresion1 /.x -> 1
0

expresion2 /.x -> 1
10
```

Listing 17: Evaluar una expresión

- El comando **Apart[expresion]**, nos permite separar una expresión polinómica racional en sus fracciones parciales.

Ejemplo:

```
Apart[1/((x + 1) (x + 2))]

1/(1 + x) - 1/(2 + x)
```

Listing 18: Separando una expresión en fracciones parciales

- El comando **Together[expresion]** permite sumar expresiones polinómicas racionales y convertirlas en una sola expresión con denominador en común.

Ejemplo:

```
Together[1/(x + 1) + 1/(x + 2)]

(3 + 2 x)/((1 + x) (2 + x))
```

Listing 19: Suma de polinomios racionales

- Los comandos **PolynomialGCD[expresion1, expresion2]** y **PolynomialLCM[expresion1, expresion2]** permiten encontrar el máximo común divisor y el mínimos común múltiplo respectivamente, entre dos expresiones polinómicas.

Ejemplo:

```
PolynomialGCD[expresion1, expresion2]
1

PolynomialLCM[expresion1, expresion2]
(1 - 2 x + x^2) (-28 + 32 x + 6 x^2)
```

Listing 20: Máximo común divisor y mínimo común múltiplo

- El comando **PolynomialMod[polinomio1,polinomio2]** permite dividir, 2 polinomios y retornar como respuesta el modulo o residuo de esa división.

Ejemplo:

```
PolynomialMod[expresion1, expresion2]

17/3 - (22 x)/3
```

Listing 21: Modulo de una división

X. Funciones trigonométricas

Mathematica cuenta con sus propias funciones trigonométricas, por lo que no hay necesidad de definirlas. Mathematica por default trabaja en radianes y no con grados aunque cuenta con un comando para hacer la conversión si no queremos hacerlo de forma manual.

Ejemplo:

```
Sin[Pi/2] (*La funcion  Pi nos permite utilizar el valor de la constante pi*)
1

Cos[Pi]
-1

Tan[Pi/4]
1
```

Listing 22: Funciones trigonométricas

```
ArcSin[1]
Pi/2

ArcCos[1]
0

ArcTan[1]
Pi/4
```

Listing 23: Funciones trigonométricas inverssas

XI. Solución de ecuaciones y sistemas de ecuaciones

- Mathematica nos permite por medio de su comando **Solve[ecuacion, variable]** resolver una ecuación con respecto a la variable, un sistema de ecuaciones en una variable, para múltiples variables.
- Si necesitamos la solución numérica, usamos el comando **NSolve[ecuacion, variable]**

Ejemplos:

```
Solve[x^2 - 2 == 0, x] (*Ecuacion simple*)
{{x -> -Sqrt[2]}, {x -> Sqrt[2]}}

Solve[{2 x + 3 == 7, 4 x - 5 == 3}, x] (*Sistema de ecuaciones con variable x
    *)
{{x -> 2}}

Solve[{x + y == 2, x - y == 0}, {x, y}] (*Sistema de ecuaciones con 2
    variables*)
{{x -> 1, y -> 1}}

NSolve[x^2 - 2 == 0, x]
{{x -> -1.41421}, {x -> 1.41421}}
```

Listing 24: Resolución de ecuaciones

XII. Cálculo

En mathematica podemos trabajar las operaciones básicas de cálculo diferencial; integrales, derivadas, límites, etc. Además utilizando la notación para evaluar expresiones aprendida anteriormente **/.x→ numero**, podemos evaluar las expresiones resultantes.

Ejemplos:

```
D[x^3, x]
3 x^2
(*El comando D[polinomio,variable de derivacion] permite derivar el polinomio con
    respecto a la variable escogida*)

D[x^3, {x,2}]
6 x
(*El comando D[polinomio,{variable de derivacion, orden de derivada}] permite
    derivar el polinomio el numero de veces que queramos*)

D[x^2 *y^3 + y^2 *x^3, x, y]
6 x^2 y + 6 x y^2
(*Deriva con respecto a ambas variables, primero con respecto a x luego con
    respecto a y*)

D[x^3, x] /.x -> 1
3

D[x^3, {x, 2}] /.x -> 1
6

D[x^2 *y^3 + y^2 *x^3, x, y] /. {x -> 1, y -> 1}
12
```

Listing 25: Derivación y evaluación

```
Integrate[x^2, x]
x^3/3
(*El comando Integrate[polinomio,variable de integracion] permite integrar la
    expresion con respecto a la variable x*)

Integrate[x^2, x] /.x -> 1
1/3
(*Permite evaluar la respuesta final*)

Integrate[x^2, {x, 0, 1}]
1/3
(*Permite hacer una integral definida en un intervalo*)

Integrate[x^2 + y^2, {x, 0, 1}, {y, 0, 1}]
2/3
(*Podemos realizar integrales dobles o triples definidas en sus respectivos
    intervalos*)
```

Listing 26: Integración y evaluación

XIII. Definir funciones y graficarlas

Mathematica al igual que distintos lenguajes de programación, permite definir funciones que hagan lo que necesitemos y posteriormente gráficarlas. Además de gráficar, podemos manipular nuestros gráficos, cambiar los colores de la curva, agregar etiquetas a cada curva, nombrar los ejes, nombrar nuestro gráfico, cambiar el espesor de las líneas, etc.

- Para crear una función escalar tenemos que respetar la sintaxis. Colocamos el nombre que queramos a la función y entre parentesis colocamos las variables de la función seguido de un guión bajo al momento de definirla, cuando deseemos llamar a nuestra función posteriormente, podemos omitir los guiones bajos.

Ejemplo:

```
f[x_] = x^2;
g[x_] = Sin[x] + Cos[x];
```

Listing 27: Definiendo funciones

Para plotear o visualizar una función contamos con distintos comandos, ya sea plotear funciones de una variable (Gráfico en 2D), plotear funciones de 2 variables (Gráfico en 3D), plotear una curva paramétrizada o visualizar un conjunto de datos.

- Para realizar un plot sencillo utilizamos el comando ***Plot[función[variable],variable, mínimo, máximo]** donde el mínimo y máximo definen el intervalo donde está definida nuestra función.

Ejemplo:

```
  Plot[f[x], {x, -1, 1}]
  (*{x,-1,1} es el intervalo de la funcion*)
```

Listing 28: Graficado funciones

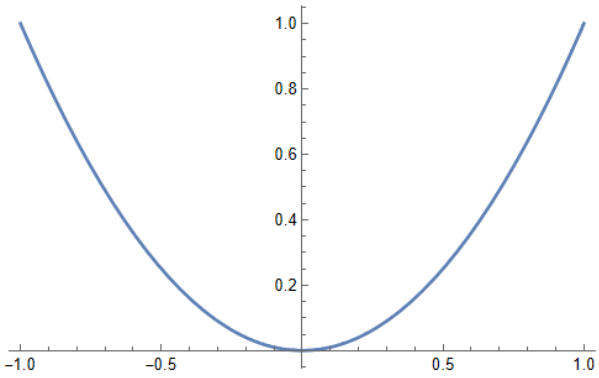

Figura 1: Plot de la función f[x]

XIV. Personalización de gráficos

- Para personalizar nuestros gráficos contamos con el comando **PlotStyle[ ]** donde podemos asignarle un color diferente al gráfico y volverlo más delgado o grueso. Los colores disponibles: **Red, Blue, Green, Yellow, Orange, Purple, Black, White, Gray, Pink, Brown**. Para el espesor: **Thick (Grueso), Thin (Delgado)** y el comando ***Thickness[tamaño]** donde ajustamos manualmente el espesor.

- **PlotStyle**→ $\{color, espesor\}$

Ejemplo:

```
Plot[f[x], {x, -1, 1}, PlotStyle -> {Red, Thick}]
```

Listing 29: Cambiar el color y grosor de una gráfica

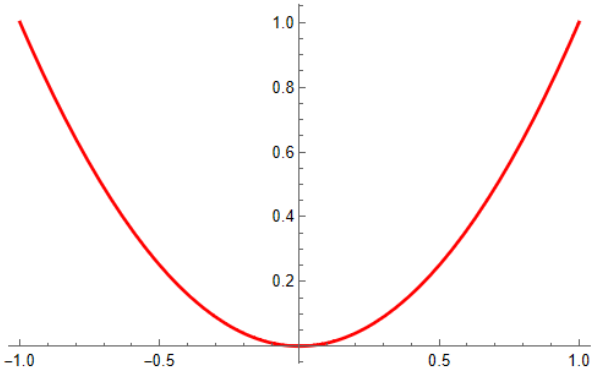

Figura 2: Plot de la función f[x] personalizada

Otro conjunto de comandos útiles son los asignados para nombrar los ejes, las curvas y el gráfico entero. Esto ayuda a la lectura el gráfico para saber diferenciar cada curva o saber que cantidades estamos comparando leyendo el nombre de los ejes.

- **AxesLabel** → **{"nombre del eje x", "nombre del eje y"}**

- **PlotLegend** →**{"nombre de la curva mostrada en el gráfico"}**

- **PlotLabel** → **"Nombre del gráfico"**
  Estos comandos se deben escribir tal y como están, con llaves donde hay llaves y con comillas donde hay comillas.

  Ejemplo:

```
Plot[g[x],{x, -1, 1}, AxesLabel->{"Eje_x","Eje_y"}, PlotLegends->{"Cos(x)+Sin(
    x)"}, PlotLabel->"Grafico_de_la_funcion_g(x)", PlotStyle->{Orange, Thick}]
```

Listing 30: Nombrar elementos del gráfico

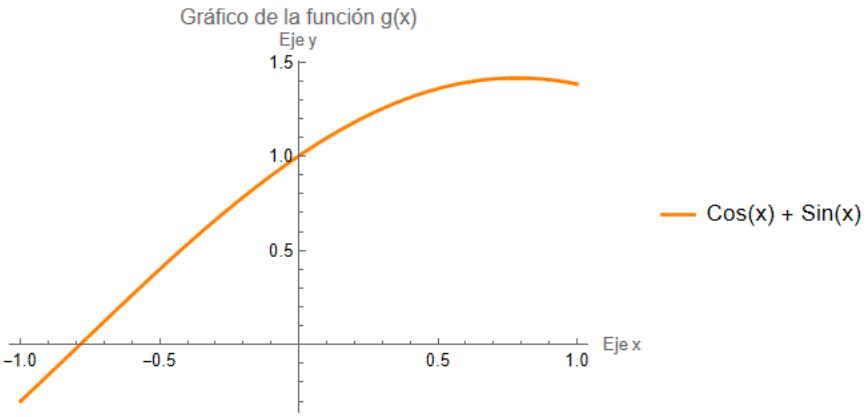

Figura 3: Etiquetas para un plot

Al igual que con las variables, es posible almacenar un gráfico. Esto es conveniente cuando contamos con varios gráficos y queremos mostrarlos todos en uno solo, definirlos con un nombre hace posible simplificar la notación a utilizar y permite evitar errores de sintaxis.

- El comando **Show[ ]** es el responsable de mostrar varios gráficos en un mismo plot. La condición que debe cumplir es que todas las funciones tengan un intervalo donde esten definidas en común. Por defecto Mathematica toma el intervalo del primer plot que enviemos, aunque podemos arreglar esto con el comando **PlotRange → All**
- **Show[grafico1, grafico2, ..., PlotRange → All]**

Ejemplo:

```
grafico1 = Plot[f[x],{x, -1, 1}, PlotStyle->Red, PlotLegends->{"f(x)=x^2"}];

grafico2 = Plot[g[x],{x, -5, 5},PlotStyle->Blue,PlotLegends->"g(x)=Cos(x)+Sin(
    x)"}, AxesLabel->{"Eje_x", "Eje_y"}];

Show[grafico2, grafico1, PlotRange->All]
```

Listing 31: Mostrar más de un plot a la vez

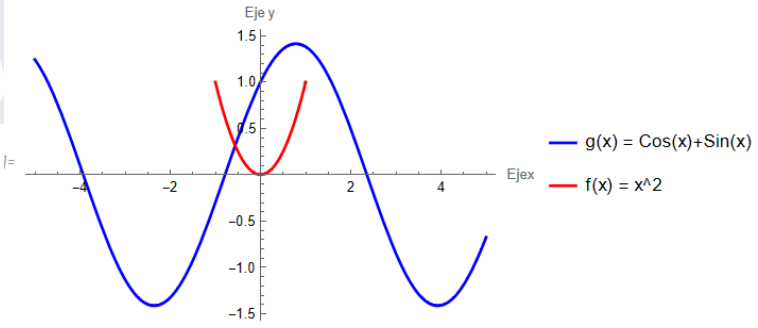

Figura 4: Rango total de múltiples gráficos

XV. Manipulación de gráficos

- El comando **Manipulate[ ]** es una función de Mathematica utilizada para crear gráficos interactivos por medio de la manipulación de los parámetros de la curva para ver como afectan a la función, la sintaxis básica necesita un gráfico, la variable que se controla y los valores máximos y mínimos que puede tomar esta variable:
- **Manipulate[Plot del gráfico con el parámetro, {parámetro, mínimo, máximo}]**

Ejemplo:

```
F[x_] := Sin[x];

Manipulate[Plot[F[a*x], {x, -15, 15}], {a, -1, 1}]
```

Listing 32: Manipulación de gráficos

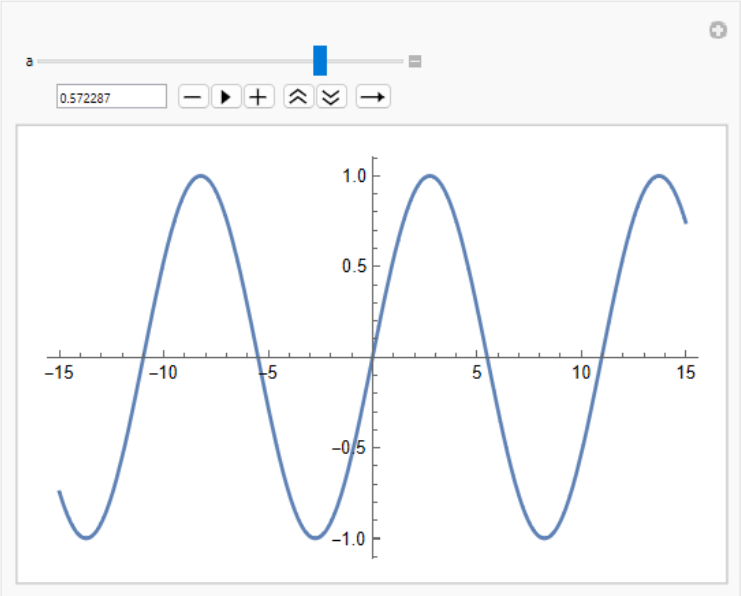

Figura 5: Manipulación del Sin[x]

- En ocasiones necesitamos comparar gráficos en lugar de colocarlos todos en un solo plot, para esto Mathematica cuenta con la función **GraphicsGrid[ ]**, en lugar de colocar varios plot en un solo plano coordenado, permite organizar múltiples gráficos en una cuadrícula.
- **GraphicsGrid[grafico1, grafico2, grafico3, grafico4]**, los elemenos entre llaves se organizan como filas y cada conjunto es una columna.

Ejemplo:

```
GraphicsGrid[{{grafico1, grafico2}}] (*Se muestra el grafico1 a la par del
    grafico2 en una fila*)
```

Listing 33: Organización de gráficos

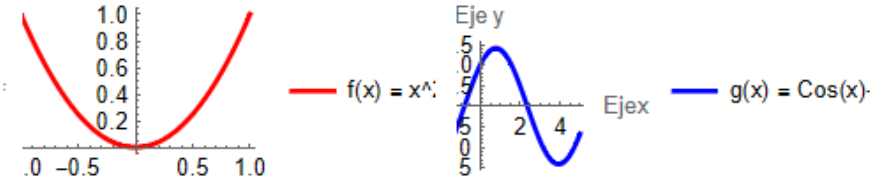

Figura 6: Un plot al lado de otro

```
GraphicsGrid[{{grafico1}, {grafico2}}] (*Se muestra el grafico1 por encima del
    grafico2 como en una columna*)
```

Listing 34: Organización de gráficos

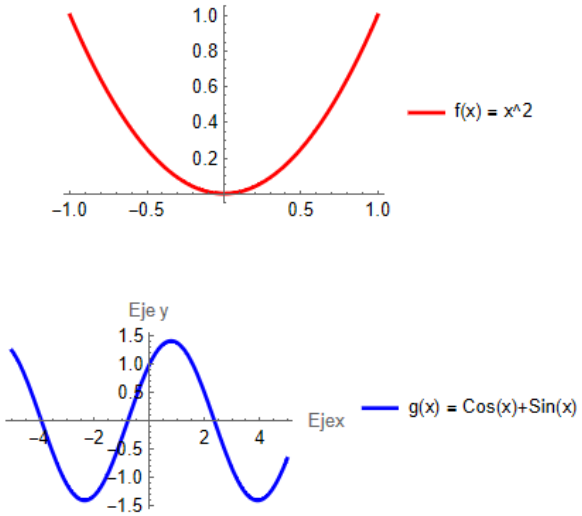

Figura 7: Un plot por encima de otro

- El comando **ImageSize[ ]** nos permite modificar el tamaño de nuestro gráfico a determinado número de pixeles deseado. Otra forma es utilizando directamente **"Small", "Medium", "Large"**.

  ImageSize → número de pixeles deseado
  ImageSize → Small
  ImageSize → Medium
  ImageSize → Large

Ejemplo:

```
Plot[f[x], {x, -1, 1}, ImageSize -> 200]
```

Listing 35: Cambiar el tamaño de un gráfico

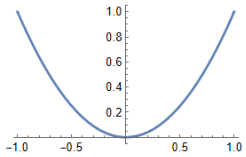

Figura 8: Plot más pequeño

XVI. Ajustes líneales y No líneales

Mathematica cuenta con la opción de realizar ajustes de datos de forma líneal y no líneal, esto dependiendo del conjunto de datos que tengamos y la ecuación que los relaciona. Cuando desconocemos esto podemos recurrir a Mathematica para encontrar la función que mejor se ajuste a los datos.

- Para realizar ajustes líneales podemos utilizar el comando **Fit[ ]** este comando es el más básico para realizar ajustes de polinomios y funciones, proporcionando la expresión simbólica del ajuste pero no proporciona información extra sobre el mismo.
- Los comandos **LinearModelFit[ ]** y **NonlinearModelFit[ ]** utilizados para modelos líneales y no líneales respectivamente, devuelven un objeto de ajuste con datos estadísticos más completos como el error estándar, el coeficiente de correlación, etc.

Ejemplo:

```
datos = {{1, 2}, {2, 3}, {2, 4}, {3, 4}, {4, 4}, {4, 6}, {4, 6}, {5, 5}, {6,
    7}};
(*Definimos de esta forma un conjunto de datos {x,y}*)

ajuste1 = Fit[datos, {1, x}, x];
ajuste1
1.52747 + 0.879121 x

ajuste3 = LinearModelFit[datos, {x}, x];
ajuste3["ParameterError"]
{0.676682, 0.180137}
(*Permite observar las incertudembres de los parametros del ajuste*)
```

Listing 36: Ajustes líneales

- Para visualizar maestros datos para el ajuste, utilizamos el comando **ListPlot[ ]** que nos permite visualizar los datos experimentales.

Ejemplo:

```
graficoDatos = ListPlot[datos]; (*Este plot muestra los datos como puntos que
    apareceran sobre la recta*)

curvaDatos = Show[graficoDatos, Plot[ajuste1, {x, 0, 7}, PlotLegends->{"Fit[]"
    }, PlotStyle->Blue]];

curvaDatos3 = Show[graficoDatos, Plot[ajuste3[x], {x, 0, 7}, PlotLegends ->
 {"LinearModelFit[]"}, PlotStyle->Green]];

GraphicsGrid[{{curvaDatos}, {curvaDatos3}}] (*Colocamos un grafico encima del
    otro*)
```

Listing 37: Visualización del ajuste líneal

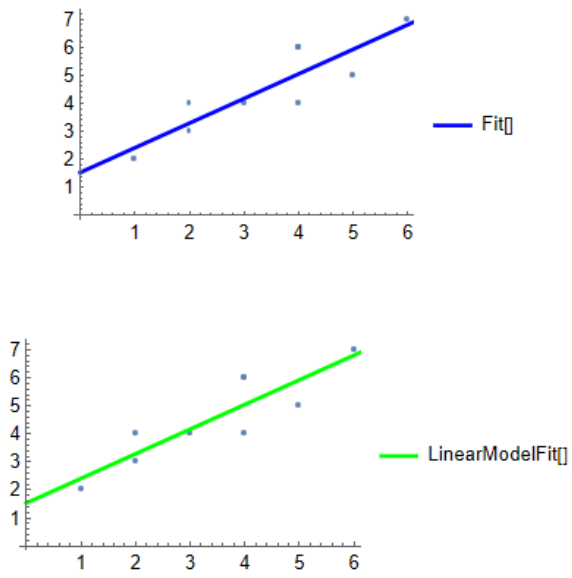

Figura 9: Ajuste líneal y datos

Para realizar ajustes no líneales, usamos un procedimiento análogo.
Ejemplo:

```
datos2 = {{15, 1}, {18, 3}, {23, 8}, {25, 15}, {28, 30}, {109, 57}};

ajusteNo1 = Fit[datos2, {1, x, x^2}, x];
ajusteNo1
-39.4234 + 2.71369 x - 0.0167765 x^2

ajusteNo3 = NonlinearModelFit[datos2, A*x^2 + B*x + C , {A, B, C}, x];
ajusteNo3["ParameterErros"]
{0.00637826, 0.826248, 15.0577}
```

Listing 38: Ajuste NO líneal