UNIVERSIDAD DE COSTA RICA

Fundamentos de MATLAB

Mario De León Urbina

Escuela de Matemática

13 de agosto de 2023

0 Contenidos

UNIVERSIDAD DE COSTA RICA

1 Contenidos

1 Introducción

1 Introducción

En MathWorks encontramos una definición de qué es el Análisis Numérico:

1 Introducción

En MathWorks encontramos una definición de qué es el Análisis Numérico:

1 Introducción

En MathWorks encontramos una definición de qué es el Análisis Numérico:

- El **Análisis Numérico** es una rama de las matemáticas que resuelve problemas continuos mediante la aproximación numérica.

1 Introducción

En MathWorks encontramos una definición de qué es el Análisis Numérico:

- El **Análisis Numérico** es una rama de las matemáticas que resuelve problemas continuos mediante la aproximación numérica.
  - Implica diseñar métodos que den soluciones numéricas aproximadas pero precisas, lo cual es útil en los casos en que la solución exacta es imposible o prohibitivamente costosa de calcular.

En MathWorks encontramos una definición de qué es el Análisis Numérico:

- El **Análisis Numérico** es una rama de las matemáticas que resuelve problemas continuos mediante la aproximación numérica.
  - Implica diseñar métodos que den soluciones numéricas aproximadas pero precisas, lo cual es útil en los casos en que la solución exacta es imposible o prohibitivamente costosa de calcular.
  - El análisis numérico también implica caracterizar la convergencia, la precisión, la estabilidad y la complejidad computacional de estos métodos.

1 Introducción

Entonces es importante utilizar un software que permita realizar algoritmos para problemas matemáticos formulados desde distintas áreas del conocimiento. En nuestro caso utilizaremos `MATLAB`:

1 Introducción

Entonces es importante utilizar un software que permita realizar algoritmos para problemas matemáticos formulados desde distintas áreas del conocimiento. En nuestro caso utilizaremos `MATLAB`:

Entonces es importante utilizar un software que permita realizar algoritmos para problemas matemáticos formulados desde distintas áreas del conocimiento. En nuestro caso utilizaremos `MATLAB`:

`MATLAB` se usa ampliamente para el análisis numérico aplicado en ingeniería, finanzas computacionales y biología computacional. Proporciona una gama de métodos numéricos para:

Entonces es importante utilizar un software que permita realizar algoritmos para problemas matemáticos formulados desde distintas áreas del conocimiento. En nuestro caso utilizaremos `MATLAB`:

`MATLAB` se usa ampliamente para el análisis numérico aplicado en ingeniería, finanzas computacionales y biología computacional. Proporciona una gama de métodos numéricos para:

- Interpolación, extrapolación y regresión

Entonces es importante utilizar un software que permita realizar algoritmos para problemas matemáticos formulados desde distintas áreas del conocimiento. En nuestro caso utilizaremos `MATLAB`:

`MATLAB` se usa ampliamente para el análisis numérico aplicado en ingeniería, finanzas computacionales y biología computacional. Proporciona una gama de métodos numéricos para:

- Interpolación, extrapolación y regresión
- Diferenciación e integración

Entonces es importante utilizar un software que permita realizar algoritmos para problemas matemáticos formulados desde distintas áreas del conocimiento. En nuestro caso utilizaremos `MATLAB`:

`MATLAB` se usa ampliamente para el análisis numérico aplicado en ingeniería, finanzas computacionales y biología computacional. Proporciona una gama de métodos numéricos para:

- Interpolación, extrapolación y regresión
- Diferenciación e integración
- Sistemas lineales de ecuaciones

Entonces es importante utilizar un software que permita realizar algoritmos para problemas matemáticos formulados desde distintas áreas del conocimiento. En nuestro caso utilizaremos `MATLAB`:

`MATLAB` se usa ampliamente para el análisis numérico aplicado en ingeniería, finanzas computacionales y biología computacional. Proporciona una gama de métodos numéricos para:

- Interpolación, extrapolación y regresión
- Diferenciación e integración
- Sistemas lineales de ecuaciones
- Valores propios y valores singulares

Entonces es importante utilizar un software que permita realizar algoritmos para problemas matemáticos formulados desde distintas áreas del conocimiento. En nuestro caso utilizaremos `MATLAB`:

`MATLAB` se usa ampliamente para el análisis numérico aplicado en ingeniería, finanzas computacionales y biología computacional. Proporciona una gama de métodos numéricos para:

- Interpolación, extrapolación y regresión
- Diferenciación e integración
- Sistemas lineales de ecuaciones
- Valores propios y valores singulares
- Ecuaciones diferenciales ordinarias (EDO)

Entonces es importante utilizar un software que permita realizar algoritmos para problemas matemáticos formulados desde distintas áreas del conocimiento. En nuestro caso utilizaremos `MATLAB`:

`MATLAB` se usa ampliamente para el análisis numérico aplicado en ingeniería, finanzas computacionales y biología computacional. Proporciona una gama de métodos numéricos para:

- Interpolación, extrapolación y regresión
- Diferenciación e integración
- Sistemas lineales de ecuaciones
- Valores propios y valores singulares
- Ecuaciones diferenciales ordinarias (EDO)
- Ecuaciones diferenciales parciales (EDP)

2 Contenidos

2 Comandos generales

- Help:

`help`
Muestra el texto de ayuda de un comando en la consola o buscador.

2 Comandos generales

- Help:

> `help`
> Muestra el texto de ayuda de un comando en la consola o buscador.

- Coma y semicolon:

> "," y ";" separan comandos. Semicolon (;) suprime la impresión del output.

2 Comandos generales

- Información

`whos`
Da información sobre las variables guardadas en la memoria.

- Información

  `whos`
  Da información sobre las variables guardadas en la memoria.

- Medición del tiempo de ejecución:

  `tic` *línea* `toc`
  Ejecuta la *línea* de código y mide el tiempo de ejecución requerido.

2 Comandos generales

- Comentarios:

> %
> Sirve para escribir % comentarios para líneas de código.

- Comentarios:

> %
> Sirve para escribir % comentarios para líneas de código.

Observación:

- Comentarios:

> %
> Sirve para escribir % comentarios para líneas de código.

Observación:

En nuestro caso en lo que resta del curso utilizaremos más el entorno único llamado **Live Editor**, para integrar scripts y funciones. La descripción puede consultarse en

- Comentarios:

> %
> Sirve para escribir % comentarios para líneas de código.

Observación:

En nuestro caso en lo que resta del curso utilizaremos más el entorno único llamado **Live Editor**, para integrar scripts y funciones. La descripción puede consultarse en

https://la.mathworks.com/help/matlab/matlab_prog/what-is-a-live-script-or-function.html

2 Presentación de la información

- Los resultados numéricos son mostrados en notación fija decimal con 4 cifras decimales si su valor absoluto está comprendido entre 10^{-3} y 10^{3}. En caso contrario son mostrados en notación científica.

- Los resultados numéricos son mostrados en notación fija decimal con 4 cifras decimales si su valor absoluto está comprendido entre 10^{-3} y 10^{3}. En caso contrario son mostrados en notación científica.
- Se puede modificar este comportamiento mediante el comando `format`

- Los resultados numéricos son mostrados en notación fija decimal con 4 cifras decimales si su valor absoluto está comprendido entre 10^{-3} y 10^{3}. En caso contrario son mostrados en notación científica.
- Se puede modificar este comportamiento mediante el comando `format`

- Los resultados numéricos son mostrados en notación fija decimal con 4 cifras decimales si su valor absoluto está comprendido entre 10^{-3} y 10^{3}. En caso contrario son mostrados en notación científica.
- Se puede modificar este comportamiento mediante el comando `format`

`format`
Cambia el formato de salida a su valor por defecto, `short`

- Los resultados numéricos son mostrados en notación fija decimal con 4 cifras decimales si su valor absoluto está comprendido entre 10^{-3} y 10^{3}. En caso contrario son mostrados en notación científica.
- Se puede modificar este comportamiento mediante el comando `format`

`format`
Cambia el formato de salida a su valor por defecto, `short`

`format long`
muestra 15 dígitos

- Los resultados numéricos son mostrados en notación fija decimal con 4 cifras decimales si su valor absoluto está comprendido entre 10^{-3} y 10^{3}. En caso contrario son mostrados en notación científica.
- Se puede modificar este comportamiento mediante el comando `format`

`format`
Cambia el formato de salida a su valor por defecto, `short`

`format long`
muestra 15 dígitos

`format rat`
muestra los números como cociente de enteros.

2 Matrices

`MATLAB` identifica escalares con matrices de 1×1, y vectores columna (fila) de dimensión m con matrices $m \times 1\,(1 \times m)$.

`MATLAB` identifica escalares con matrices de 1×1, y vectores columna (fila) de dimensión m con matrices $m \times 1\,(1 \times m)$.

- Asignación:

> `A=` *expresión*
> Asigna a la variable `A` el valor de la expresión.

`MATLAB` identifica escalares con matrices de 1×1, y vectores columna (fila) de dimensión m con matrices $m \times 1$ $(1 \times m)$.

- Asignación:

> `A=` *expresión*
> Asigna a la variable `A` el valor de la expresión.

- Operaciones:

> `+` , `-`, `*` son las operaciones de matrices de adición, resta y multiplicación. `^` es potencia de matrices cuadradas.

`MATLAB` identifica escalares con matrices de 1×1, y vectores columna (fila) de dimensión m con matrices $m \times 1$ $(1 \times m)$.

- Asignación:

> `A=` *expresión*
> Asigna a la variable `A` el valor de la expresión.

- Operaciones:

> `+` , `-`, `*` son las operaciones de matrices de adición, resta y multiplicación. `^` es potencia de matrices cuadradas.
>
> `A/B` calcula la solución X del sistema lineal $A = XB$ y `A\B` calcula la solución X de $AX = B$.

2 Matrices

- Operaciones componente a componente:

> `.+`, `.-`, `.*`, `./`, `.^` son las operaciones componente a componente.

- Operaciones componente a componente:

> `.+`, `.-`, `.*`, `./`, `.^` son las operaciones componente a componente.

- Matriz adjunta (transpuesta):

> `A'` es la matriz adjunta de `A`.

2 Matrices

- Input para matriz:

```
A = [ 1 2 3;
4 5 6 ];
```

O también `A = [ 1 2 3; 4 5 6 ];` asigna la siguiente matriz a la variable `A`:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

- Input para matriz:

```
A = [ 1 2 3;
4 5 6 ];
```

O también `A = [ 1 2 3; 4 5 6 ];` asigna la siguiente matriz a la variable `A`:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

- Matriz identidad:

`eye(m)` es la matriz identidad de dimensión m.

2 Matrices

- Matriz nula :

> `zeros(m,n)`, `zeros(m)`
> es la matriz de entradas nulas de dimensión $m \times n$ y $m \times m$ respectivamente.

- Matriz nula :

> `zeros(m,n)`, `zeros(m)`
> es la matriz de entradas nulas de dimensión $m \times n$ y $m \times m$ respectivamente.

- Matriz de unos:

> `ones(m,n)`, `ones(m)`
> es la matriz de entradas iguales a 1.

2 Matrices

- Matrices aleatorias:

`rand(m,n)`, `rand(m)`, `randn(m,n)`, `randn(m)`
generan matrices con entradas aleatorias.

- Matrices aleatorias:

`rand(m,n)`, `rand(m)`, `randn(m,n)`, `randn(m)`
generan matrices con entradas aleatorias.

- Vectores indexados:

```
j:k, j:s:k
```

- Matrices aleatorias:

`rand(m,n)`, `rand(m)`, `randn(m,n)`, `randn(m)`
generan matrices con entradas aleatorias.

- Vectores indexados:

`j:k`, `j:s:k`

son los vectores fila
`[j, j+1, ..., k-1,k]` y `[j,j+s, ..., j+m*s]` con

- Matrices aleatorias:

`rand(m,n)`, `rand(m)`, `randn(m,n)`, `randn(m)`
generan matrices con entradas aleatorias.

- Vectores indexados:

`j:k`, `j:s:k`

son los vectores fila
`[j, j+1, ..., k-1,k]` y `[j,j+s, ..., j+m*s]` con

$$m = \left[\!\left[\frac{k-j}{s} \right]\!\right]$$

2 Matrices

- Vectores regularmente espaciados:

`linspace(a,b,n)`
Si a y b números reales, n entero, genera un vector fila de longitud n cuyo primer elemento es a y último elemento b, y los elementos están regularmente espaciados.

- Vectores regularmente espaciados:

`linspace(a,b,n)`
Si a y b números reales, n entero, genera un vector fila de longitud n cuyo primer elemento es a y último elemento b, y los elementos están regularmente espaciados.

- Componentes:

`A(j,k)`
es la entrada A_{jk} de la matriz `A`.

2 Matrices

- Bloques de matrices (submatrices):

- Bloques de matrices (submatrices):

`A(j1:j2, k1:k2)`
es la submatriz de `A` en la cual las fila va de $j1 \leq j2$ y las columnas van de $k1 \leq k2$.

- Bloques de matrices (submatrices):

`A(j1:j2, k1:k2)`
es la submatriz de `A` en la cual las fila va de $j1 \leq j2$ y las columnas van de $k1 \leq k2$.

`A(j1:j2, :)`, `A(:, k1:k2)`
son las submatrices construidas desde las filas $j1$ a $j2$ y de las columnas $k1$ a $k2$ respectivamente.

- Bloques de matrices (submatrices):

`A(j1:j2, k1:k2)`
es la submatriz de `A` en la cual las fila va de $j1 \leq j2$ y las columnas van de $k1 \leq k2$.

`A(j1:j2, :), A(:, k1:k2)`
son las submatrices construidas desde las filas $j1$ a $j2$ y de las columnas $k1$ a $k2$ respectivamente.

`A(j,:), A(:,k)`
son el vector fila $j-$ésima y el vector columna $k-$ésima de la matriz `A`.

2 Matrices

- Matrices triangulares:

`tril(A)`, `triu(A)`
extrae la parte triangular inferior y triangular superior de la matriz `A`.

- Matrices triangulares:

`tril(A)`, `triu(A)`
extrae la parte triangular inferior y triangular superior de la matriz `A`.

- Diagonales de/en una matriz:

`diag(A,k)`
extrae como vector columna la diagonal $k-$ésima de la matriz `A`.

- Matrices triangulares:

`tril(A)`, `triu(A)`
extrae la parte triangular inferior y triangular superior de la matriz `A`.

- Diagonales de/en una matriz:

`diag(A,k)`
extrae como vector columna la diagonal $k-$ésima de la matriz `A`.

`diag(v,k)`, `diag(v)`
si v es un vector entonces se genera una matriz de ceros en donde la diagonal k es el vector v.

2 Matrices

- Dimensiones de una matriz:

- Dimensiones de una matriz:

`size(A,1)`
da el número de filas

- Dimensiones de una matriz:

`size(A,1)`
da el número de filas

`size(A,2)`
da el número de columnas.

- Dimensiones de una matriz:

`size(A,1)`
da el número de filas

`size(A,2)`
da el número de columnas.

`size(A)`
da un vector `v = [m, n]` donde m es el número de filas y n el número de columnas de `A`.

- Dimensiones de una matriz:

`size(A,1)`
da el número de filas

`size(A,2)`
da el número de columnas.

`size(A)`
da un vector `v = [m, n]` donde m es el número de filas y n el número de columnas de `A`.

`length(v)`
da la dimensión de un vector columna o fila `v`.

3 Contenidos

3 M-funciones

```
function [o_1, o_2, ..., o_n] = myfunc(i_1, i_2, ...,
i_m)
```

```
function [o_1, o_2, ..., o_n] = myfunc(i_1, i_2, ...,
i_m)
```

donde `i_1, i_2, ..., i_m` representan las variables de inputs y `o_1, o_2, ..., o_n` representan las variables de outputs.

`function [o_1, o_2, ..., o_n] = myfunc(i_1, i_2, ..., i_m)`

donde `i_1, i_2, ..., i_m` representan las variables de inputs y `o_1, o_2, ..., o_n` representan las variables de outputs.

La función puede ser guardada en un archivo `myfunc.m` en el directorio del proyecto.

`function [o_1, o_2, ..., o_n] = myfunc(i_1, i_2, ..., i_m)`

donde `i_1, i_2, ..., i_m` representan las variables de inputs y `o_1, o_2, ..., o_n` representan las variables de outputs.

La función puede ser guardada en un archivo `myfunc.m` en el directorio del proyecto. Se puede llamar en la línea de comando (o en la creación de otras funciones) como:

`function [o_1, o_2, ..., o_n] = myfunc(i_1, i_2, ..., i_m)`

donde `i_1, i_2, ..., i_m` representan las variables de inputs y `o_1, o_2, ..., o_n` representan las variables de outputs.

La función puede ser guardada en un archivo `myfunc.m` en el directorio del proyecto. Se puede llamar en la línea de comando (o en la creación de otras funciones) como:

`[o_1, o_2, ..., o_n] = myfunc(i_1, i_2, ..., i_m)`

`function [o_1, o_2, ..., o_n] = myfunc(i_1, i_2, ..., i_m)`

donde `i_1, i_2, ..., i_m` representan las variables de inputs y `o_1, o_2, ..., o_n` representan las variables de outputs.

La función puede ser guardada en un archivo `myfunc.m` en el directorio del proyecto. Se puede llamar en la línea de comando (o en la creación de otras funciones) como:

`[o_1, o_2, ..., o_n] = myfunc(i_1, i_2, ..., i_m)`

Las variables de salida (outputs) podrían ignorarse reemplazándolas por $\sim$, por ejemplo:
`[~ , o_2] = myfunc(i_1, i_2, ..., i_m)`

3 Funciones

- Función anónima

- Función anónima

`f = @(i_1, i_2, ..., i_m)` *expresión*

define una función que asigna el valor de la expresión con los inputs `i_1, i_2, ..., i_m`.

- Función anónima

`f = @(i_1, i_2, ..., i_m)` *expresión*

define una función que asigna el valor de la expresión con los inputs `i_1, i_2, ..., i_m`.

> Al llamarla se ejecuta como

- Función anónima

`f = @(i_1, i_2, ..., i_m)` *expresión*

define una función que asigna el valor de la expresión con los inputs `i_1, i_2, ..., i_m`.

> Al llamarla se ejecuta como

```
o_1 = f(i_1, i_2, ..., i_m)
```

3 Funciones matemáticas

Función		Función	
`sqrt(x)`	raíz cuadrada	`sin(x)`	seno (radianes)
`abs(x)`	módulo	`cos(x)`	coseno (radianes)
`conj(z)`	conjugado	`tan(x)`	tangente (radianes)
`real(z)`	parte real	`cotg(x)`	cotangente (radianes)
`imag(z)`	parte imaginaria	`asin(x)`	arcoseno
`exp(x)`	exponencial	`acos(x)`	arcocoseno
`log(x)`	logaritmo natural	`atan(x)`	arcotangente
`log10(x)`	logaritmo decimal	`cosh(x)`	cos hiperbólico
`rat(x)`	aprox. racional	`sinh(x)`	seno hiperbólico
`fix(x)`	redondeo hacia 0	`tanh(x)`	tang. hiperbólico
`ceil(x)`	redondeo hacia $+\infty$	`acosh(x)`	arccos hiperbólico
`floor(x)`	redondeo hacia $-\infty$	`asinh(x)`	arcsen hiperbólico
`round(x)`	redondeo al entero más próx.	`atanh(x)`	arctan hiperbólico

3 Funciones matriciales

Función	
`sum(v)`	suma de las entradas de v vector
`sum(A)`	suma de las entradas de A matriz
`sum(A,1)`	suma de los elementos por columnas
`sum(A,2)`	suma de los elementos por filas
`prod(v)`	producto de las entradas de v vector
`prod(A)`	producto de las entradas de A
`max(v)`	máximo elemento de v
`min(v)`	mínimo elemento de v
`norm(v, p)`	norma $\|v\|_p$ vectorial/matricial
`norm(v, Inf)`	norma $\|v\|_\infty$ vectorial/matricial
`eig(A)`	espectro de A

3 Funciones lógicas

- Comparaciones:

```
A == B, A ~= B, A<=B, A >= B, A <  B, A > B
```

comparaciones componente a componente de matrices;
1 es para ‘verdadero’ y 0 para ‘falso’.

- Comparaciones:

`A == B, A ~= B, A<=B, A >= B, A < B, A > B`

comparaciones componente a componente de matrices;
1 es para 'verdadero' y 0 para 'falso'.

- Operaciones lógicas booleanas:

`a && b, a || b`
son los operadores lógicos 'y' y 'o', respectivamente.

$\sim$ *expresión*, `not` (*expresión*)
ambas retornan la negación lógica de la expresión.

4 Contenidos

4 Graficación

4 Graficación

Se muestran algunas funciones elementales de dibujo de MATLAB, cuyo uso permite realizar gráficas de funciones.

Se muestran algunas funciones elementales de dibujo de `MATLAB`, cuyo uso permite realizar gráficas de funciones.

- Plot:

```
plot(x, y)
```

se puede utilizar junto con `linspace`.

Se muestran algunas funciones elementales de dibujo de `MATLAB`, cuyo uso permite realizar gráficas de funciones.

- Plot:

```
plot(x, y)
```

se puede utilizar junto con `linspace`.

Por ejemplo, al graficar $y = \dfrac{x^2 + 2}{x + 5}$ en el intervalo $[-2, 3]$:

Se muestran algunas funciones elementales de dibujo de `MATLAB`, cuyo uso permite realizar gráficas de funciones.

- Plot:

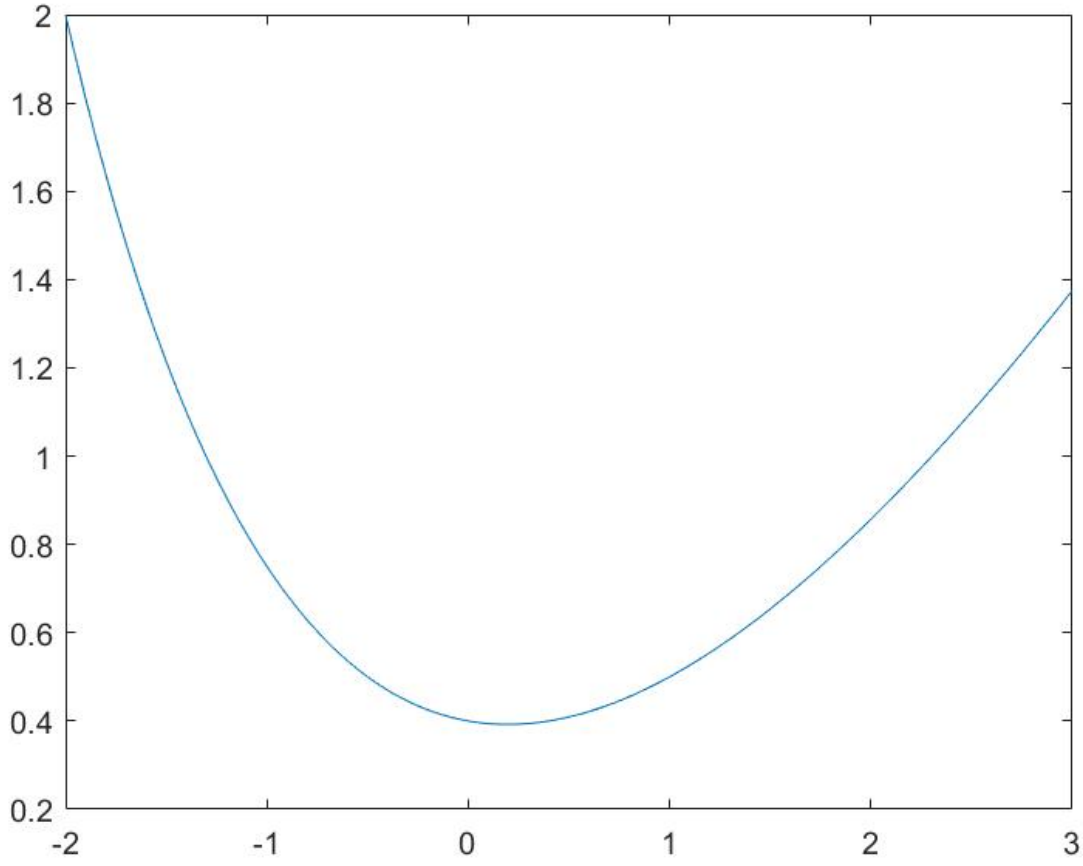
`plot(x, y)`

se puede utilizar junto con `linspace`.

Por ejemplo, al graficar $y = \dfrac{x^2+2}{x+5}$ en el intervalo $[-2, 3]$:

```
x = linspace(-2, 3);
y = (x.^2+2)./(x+5);
plot(x,y)
```

4 Graficación



4 Graficación

```
plot(x1, y1, x2, y2, x3, y3)
```

```
plot(x1, y1, x2, y2, x3, y3)
```

Se pueden dibujar dos o más curvas, proporcionando varios pares de vectores (abscisas y ordenadas) para cada una de las funciones en cuestión.

```
plot(x1, y1, x2, y2, x3, y3)
```

Se pueden dibujar dos o más curvas, proporcionando varios pares de vectores (abscisas y ordenadas) para cada una de las funciones en cuestión.

Por ejemplo, al graficar $y = e^{-x} + x^2 - x, \quad y = e^x + \sin(x^2)$:

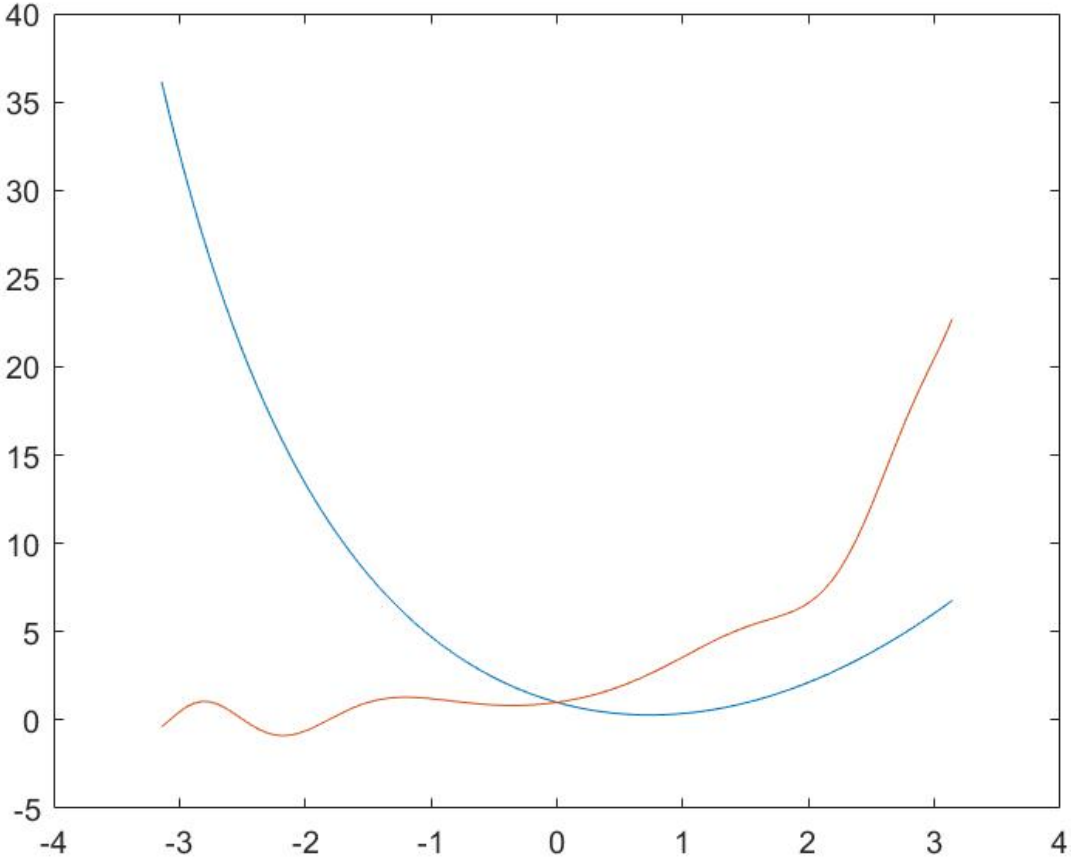
```
plot(x1, y1, x2, y2, x3, y3)
```

Se pueden dibujar dos o más curvas, proporcionando varios pares de vectores (abscisas y ordenadas) para cada una de las funciones en cuestión.

Por ejemplo, al graficar $y = e^{-x} + x^2 - x, \quad y = e^x + \sin(x^2)$:

```
x = linspace(-pi, pi);
y1 = exp(-x) + x.^2 -x;
y2 = exp(x) + sin(x.^2);
plot(x, y1, x, y2)
```

4 Graficación



4 Graficación

4 Graficación

En la graficación anterior no se distingue cuál gráfica pertenece a cada criterio. Por esto se puede agregar información. Algunas funciones son:

En la graficación anterior no se distingue cuál gráfica pertenece a cada criterio. Por esto se puede agregar información. Algunas funciones son:

- `title('título')`: añade título a la gráfica.

En la graficación anterior no se distingue cuál gráfica pertenece a cada criterio. Por esto se puede agregar información. Algunas funciones son:

- `title('título')`: añade título a la gráfica.
- `xlabel('eje_x')`: añade una etiqueta al eje x.

En la graficación anterior no se distingue cuál gráfica pertenece a cada criterio. Por esto se puede agregar información. Algunas funciones son:

- `title('título')`: añade título a la gráfica.
- `xlabel('eje_x')`: añade una etiqueta al eje x.
- `ylabel('eje_x')`: añade una etiqueta al eje y.

En la graficación anterior no se distingue cuál gráfica pertenece a cada criterio. Por esto se puede agregar información. Algunas funciones son:

- `title('título')`: añade título a la gráfica.
- `xlabel('eje_x')`: añade una etiqueta al eje x.
- `ylabel('eje_x')`: añade una etiqueta al eje y.
- `legend('f', 'g', ...)`: define etiquetas para las gráficas de las funciones involucradas.

En la graficación anterior no se distingue cuál gráfica pertenece a cada criterio. Por esto se puede agregar información. Algunas funciones son:

- `title('título')`: añade título a la gráfica.
- `xlabel('eje_x')`: añade una etiqueta al eje x.
- `ylabel('eje_x')`: añade una etiqueta al eje y.
- `legend('f', 'g', ...)`: define etiquetas para las gráficas de las funciones involucradas.
- `grid`: muestra una cuadrícula.

En la graficación anterior no se distingue cuál gráfica pertenece a cada criterio. Por esto se puede agregar información. Algunas funciones son:

- `title('título')`: añade título a la gráfica.
- `xlabel('eje_x')`: añade una etiqueta al eje x.
- `ylabel('eje_x')`: añade una etiqueta al eje y.
- `legend('f', 'g', ...)`: define etiquetas para las gráficas de las funciones involucradas.
- `grid`: muestra una cuadrícula.
- `text(x, y, 'texto')`: agrega texto en el lugar definido por el punto (x, y).

4 Graficación

4 Graficación

Al usar (`' '`) se puede utilizar el lenguaje LaTeX con ayuda de `'interpreter'`, y así digitar de forma matemática fórmulas y expresiones.

Al usar (`' '`) se puede utilizar el lenguaje LaTeX con ayuda de `'interpreter'`, y así digitar de forma matemática fórmulas y expresiones.

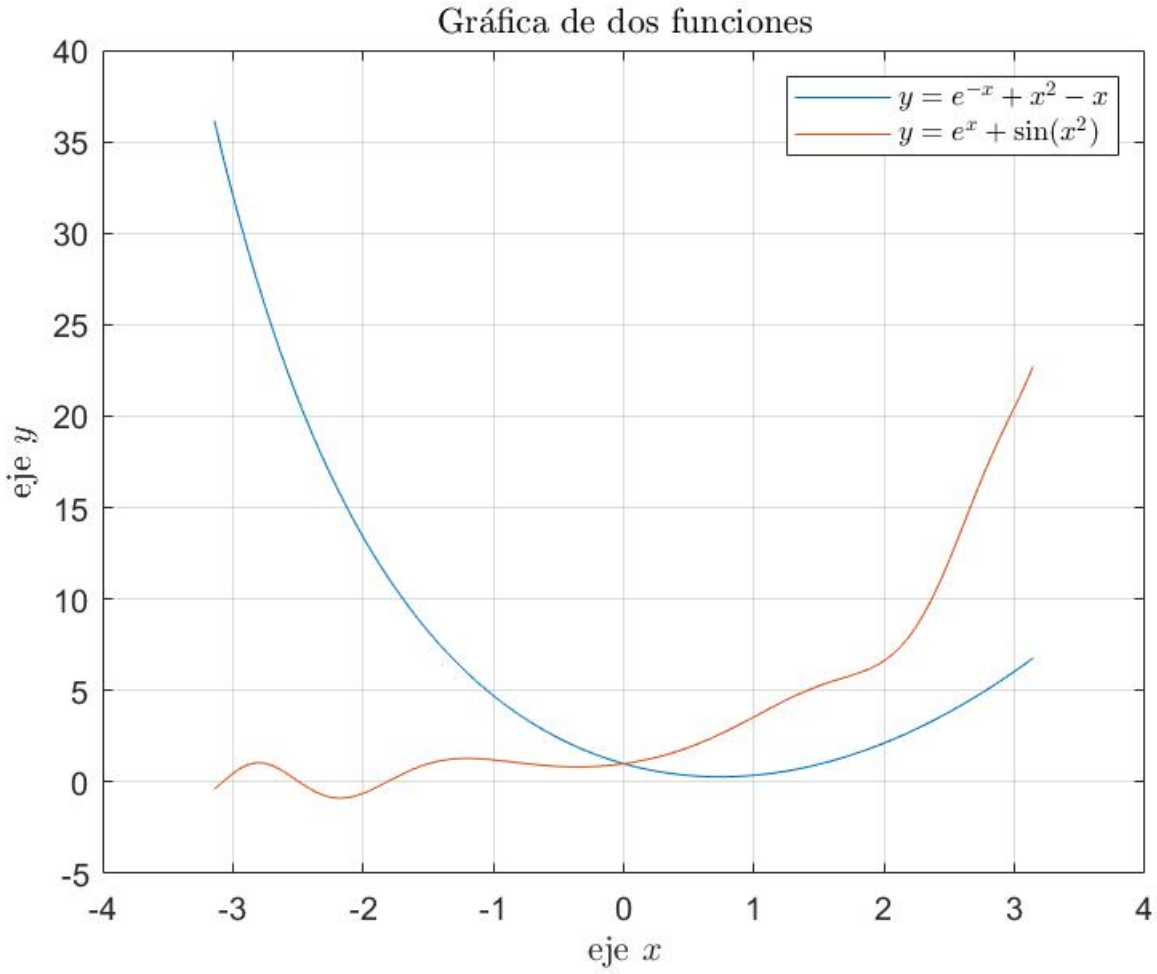
Por ejemplo, al graficar $y = e^{-x} + x^2 - x, \quad y = e^x + \sin(x^2)$:

```
x = linspace(-pi, pi);
y1 = exp(-x) + x.^2 -x;
y2 = exp(x) + sin(x.^2);
plot(x, y1, x, y2), grid

% Informacion de la grafica

title('Gr\''afica de dos funciones', 'interpreter', ...
'latex')
xlabel('eje $x$', 'interpreter', 'latex')
ylabel('eje $y$', 'interpreter', 'latex')
l = legend('$y=e^{-x}+x^2-x$', '$y=e^x+\sin(x^2)$');
set(l, 'interpreter', 'latex')
```

4 Graficación



¡Muchas Gracias!