

Resumen MATLAB

1. CARACTERÍSTICAS BÁSICAS:

- MATLAB conoce la suma (+), resta (-), multiplicación (*), división (/ o \) y potenciación (^).
- MATLAB evalúa una expresión de izquierda a derecha dando precedencia a la potencia sobre la multiplicación y la división y a éstas sobre la suma y la resta.
- Un punto y coma (;) al final de una orden de MATLAB suprime la impresión de resultados.
- Si una orden es demasiado larga, escriba tres puntos (...) seguido por **Enter** para continuar la orden de MATLAB en la línea siguiente.
- Por defecto, MATLAB almacena resultados en la variable `ans`.
- Sólo se recuerdan los 19 primeros caracteres de un nombre de variable.
- Los nombres de variables deben comenzar con una letra. Después de esa letra, es válida cualquier ordenación de letras, números y signos de subrayado.
- MATLAB es sensible a las mayúsculas, `fruta`, `Fruta`, `FrUTA` y `FRUTA` son variables diferentes.
- Con la orden `who` se puede obtener una lista de las variables que MATLAB tiene en memoria. Y con `whos` además proporciona información sobre el tamaño, tipo, etc. de las variables.
- MATLAB tiene una serie de variables especiales: `ans` (resultados), `pi`, `eps` (número en coma flotante más pequeño), `inf` (1/0), `NaN` (0/0), `i` y `j` ($\sqrt{-1}$), `realmin` (El número real positivo más pequeño), `realmax` (el número real positivo más grande).
- MATLAB es capaz de guardar y recuperar datos en archivos mediante las ordenes `save` y `load`.
- Mediante la orden `format` se pueden visualizar los números con diferentes formatos.

2. CARACTERÍSTICAS CIENTÍFICAS:

- MATLAB tiene muchas funciones matemáticas comunes.
- Los números complejos no requieren un tratamiento especial en MATLAB. Se deben utilizar en forma rectangular como si fueran números reales.
- MATLAB sólo opera en radianes.
- Las funciones `real`, `imag`, `abs`, y `angle` son útiles para la conversión entre las formas polar y rectangular.
- El valor por defecto de `i` y `j` es $\sqrt{-1}$. Si les reasigna un nuevo valor en una orden de MATLAB, no las puede utilizar ya para crear valores complejos. Sin embargo puede reasignarlos o limpiarlos.
- Añadiendo `i` o `j` al final de un número, se le dice a MATLAB que haga el número la parte imaginaria de un número complejo, p. e., `x=3i` produce $0 + 3i$.

3. CARACTERÍSTICAS DE AYUDA EN LÍNEA:

- La orden `help` tiene muchas funciones matemáticas comunes. Tecleando sólo `help` se obtiene una lista de temas de ayuda
- La orden `lookfor` proporciona ayuda buscando en todas las primeras líneas de las ayudas a temas de MATLAB.

4. OPERACIONES CON MATRICES:

- Matrices arbitrarias se introducen como: `variable=[(una lista de números separados por espacios o comas)]`, por ejemplo, `x=[4 1 sqrt(2) sin(.5) 23.2]`.
- Las funciones de matrices aplican las funciones a los elementos individuales de las matrices.
- Las matrices que contienen elementos de datos espaciados linealmente se pueden formar usando la notación de dos puntos `primero:incremento:último` o `primero:último`. En esta última forma, se supone que el incremento es uno.
- Las matrices que contienen elementos de datos espaciados linealmente se pueden formar usando la función `linspace(primero, último, número_de_puntos)` de MATLAB.
- Las matrices que contienen elementos de datos espaciados logarítmicamente se pueden formar usando la función `logspace(primer_exp, último_exp, número_de_puntos)` de MATLAB.
- Los valores almacenados en una matriz de datos se pueden acceder utilizando subíndices, por ejemplo, `x(2:6)` son los elementos de `x` que van del segundo al sexto.
- Las operaciones matemáticas básicas entre escalares y matrices aplican la operación a todos los elementos en la matriz, por ejemplo, `x-2` resta dos de todos los elementos en `x`.
- Las operaciones matemáticas básicas entre matrices son válidas si las matrices tienen el mismo tamaño. De nuevo, el resultado se obtiene por la aplicación elemento a elemento de la operación. La multiplicación y la división se simbolizan respectivamente por `.*` y `./` o `.\`.
- La potencia de una matriz tiene múltiples definiciones dependiendo de la forma de la expresión. Por ejemplo, si `a` y `b` son matrices, `a^(0.5)` calcula la raíz cuadrada de los elementos de `a`, `2.^b` eleva 2 a las potencias dadas en `b`, y `a.^b` eleva cada elemento en `a` al correspondiente elemento en `b`. Como con la multiplicación y la división, es necesaria la notación del punto.
- Se pueden construir matrices a partir de subconjuntos de otras matrices. Esto es, si `x` y `y` son matrices orientadas a filas, `z=[x y]` produce una matriz compuesta de `x` seguido por `y`, mientras que `z=[x(1:5) -x(6:10) 5.6 2 10 y(5:23)]` produce una matriz compuesta por las submatrices `x` e `y`, y algunos valores adicionales.
- Las matrices en MATLAB pueden tener cualquier orientación rectangular. Las matrices que contienen una fila se suelen llamar vector fila. Las matrices que contienen una sola columna se suelen llamar vector columna.
- Los vectores fila se pueden convertir a vectores columna y viceversa usando la operación traspuesta representada por punto-apóstrofe (`'`), por ejemplo, si `x` es una fila, `x'` es una columna.
- Punto-apóstrofe denota traspuesta. Un apóstrofe simple representa traspuesta compleja conjugada.

5. GRÁFICAS SIMPLES:

- La función `plot` de MATLAB escoge automáticamente los ejes, marca los puntos individuales y dibuja líneas rectas entre ellos. Permite hacer gráficas de múltiples conjuntos de datos sobre los mismos ejes, usar diferentes tipos de líneas y colores. Además, puede colocar etiquetas sobre los ejes, un título de cabecera, dibujar una rejilla, etc.
- Con la función `plot3` se genera una gráfica 3-D.

6. GRÁFICAS 2-D:

- La orden `plot` crea una gráfica de vectores o columnas de matrices. La forma de la orden es `plot(x1, y1, s1, x2, y2, s2, ...)` donde (x_n, y_n) son conjuntos de arrays de datos y s_n son cadenas opcionales que especifican color, marcas y/o estilos de línea.
- `grid` coloca una rejilla en las marcas de gráfica.
- Se pueden añadir títulos y etiquetas en los ejes usando `title`, `xlabel`, y `ylabel`.
- La orden `text(x, y, S)` añade la cadena de caracteres S a la gráfica actual en las coordenadas (x, y) .
- `gtext` permite colocar texto interactivamente en la gráfica utilizando el ratón
- `axis([xmin xmax ymin ymax])` escala la gráfica actual a los valores dados como argumentos, `axis('string')`, donde 'string' es una cadena de una serie de opciones específicas, cambia los límites de los ejes o la apariencia de las gráficas de muchas formas.
- La orden `print` imprime la gráfica de la ventana de figura actual en la impresora, el portapapeles o un archivo.
- La orientación de las gráficas impresas se puede cambiar con la orden `orient`. Las tres orientaciones son `portrait`, `landscape` y `tall`.
- Puede añadir gráficas a la gráfica actual fijando `hold on`. Haciendo `hold off` permite que la próxima orden `plot` borre la ventana de figura antes de hacer una nueva gráfica.
- Se pueden generar múltiples ventanas de figura mediante la orden `figure`. `figure(n)` escoge la ventana de figura n para que sea la ventana de figura activa.
- Una ventana de figura se puede subdividir y cualquier subdivisión hacerse activa con la orden `subplot`.
- Si fija `zoom on`, la ventana de figura activa se puede expandir interactivamente usando el ratón.

7. GRÁFICAS 3-D:

- `plot3` es la versión 3-D de la orden `plot` y se utiliza de la misma forma.
- `zlabel` se usa para etiquetar el eje z en gráficos 3-D.
- La orden `axis([xmin xmax ymin ymax zmin zmax])` fija límites para los tres ejes. `axis('ij')` mueve el origen de una gráfica 3-D y cambia el eje y para aumentar de izquierda a derecha en la vista por defecto.

- La forma 3-D de la función texto, `text(x,y,z,S)`, coloca la cadena de caracteres `S` en la posición (x,y,z) en la gráfica actual.
- Una rejilla rectangular de puntos igualmente espaciados en el plano xy puede generarse fácilmente usando `meshgrid`. `[X,Y]=meshgrid(x,y)`, donde `x` e `y` son vectores, genera una matriz `X`, cuyas filas son copias del vector `x` y una matriz `Y`, cuyas columnas son copias del vector `y`. Las coordenadas (X_{ij}, Y_{ij}) representan una rejilla rectangular en el plano xy .
- `mesh(Z)`, donde `Z` es una matriz, representa una malla en tres dimensiones con intersecciones en los puntos (i,j,Z) . `Mesh(X,Y,Z)` representa una malla con intersecciones en los puntos (X_{ij}, Y_{ij}, Z_{ij}) . Un argumento de color es opcional.
- `surf` es la versión superficie de `mesh` y usa los mismos argumentos. `surf` dibuja una gráfica de malla y rellena los parches (agujeros entre los segmentos de línea) con colores apropiados.
- Las gráficas de contorno se pueden generar con `contour`. Esta función usa la forma general `contour(X,Y,Z,n,S)` donde `X`, e `Y` son vectores opcionales especificando los ejes x e y , `Z` es la matriz de datos, `n` es un número opcional de líneas de contorno a dibujar y `S` es una cadena de caracteres que especifica color y/o tipo de línea como en la orden `plot`. `contour3` dibuja contornos en el espacio 3-D.
- Se pueden generar gráficos de pseudocolor usando `pcolor`. `pcolor(C)` genera una gráfica de “tablero de damas” de la matriz `C`, donde los valores de los elementos de `C` determinan el color de cada celda de la gráfica. `pcolor(X,Y,C)`, donde `X` e `Y` son vectores o matrices, dibuja una gráfica de pseudocolor sobre la rejilla definida por `C` e `Y`. `Pcolor` es en realidad una función `surf` con un punto de vista fijado directamente desde arriba.
- El ángulo de visión de una gráfica 3-D se puede fijar con la orden `view`. El formato usual es `view(a,e)`, donde `a` es el azimut (ángulo en grados desde el eje y), y `e` es el ángulo de elevación (en grados por encima del plano xy). Una forma alternativa es `view([x y z])`, donde (x,y,z) es la coordenada cartesiana del observador. `View(2)` fija la visión por defecto 2-D (`view(0,90)`) y `view(3)` fija la visión por defecto (`view(-37.5,30)`).
- La eliminación de líneas escondidas se puede controlar con `hidden off` y `hidden on`.
- Tres tipos de sombreado están disponibles usando `shading flat`, `shading interp` y `shading faceted`.
- Un mapa de color es una matriz de 3 columnas cuyas filas contienen los valores RGB de un color en la forma $[r \ g \ b]$.
- MATLAB suministra 10 funciones para crear mapas de color diferentes.
- `colormap(M)` instala la matriz `M` como el mapa de color actual. `M` debe tener tres columnas y contiene sólo valores entre 0 y 1.
- Funciones que toman un argumento de color, normalmente aceptan un carácter representando uno de los colores en la tabla gráfica de color y estilo de línea, un único valor RGB en la forma $[r \ g \ b]$ o un vector columna o matriz cuyos elementos se escalan y utilizan como índices en el mapa de color actual.
- Los mapas de color se pueden visualizar viendo los elementos de la matriz del mapa, usando `pcolor` para visualizar barras de color, o usando la función `colorbar` para añadir una escala de color a una gráfica existente.
- `brithten` ajusta el brillo del mapa de color actual o crea un nuevo mapa de color a partir de uno existente.

- Los mapas de color se pueden crear a partir de cualquier operación de array; además se pueden instalar y usar mientras consten de tres columnas y contengan sólo valores entre 0 y 1.

8. ARCHIVOS SCRIPT:

- La orden `echo on` le dice a MATLAB que visualice o efectúe un eco de las órdenes en la ventana de ordenes. Se deshabilita mediante la orden `echo off`.
- MATLAB proporciona una serie de órdenes de gestión de archivos: `what`, listado de archivos-M en el directorio actual, `dir` y `ls`, listan todos los archivos en el directorio o carpeta actual., `type test`, Visualiza el archivo-M `test.m` en la ventana de órdenes, `delete test` suprime el archivo `test.m`, `cd path` y `chdir path`, cambian al directorio o carpeta dada por `path`, `cd`, `chdir` y `pwd`, muestran el directorio o carpeta de trabajo actual y `which test`, que muestra el camino al directorio de `test.m`.

9. TEXTO:

- Las cadenas de caracteres se manejan como vectores fila. Una cadena de caracteres es simplemente texto rodeado por comillas simple
- Las cadenas de caracteres pueden tener múltiples filas, pero cada fila debe tener el mismo número de caracteres (el mismo número de columnas).
- Son posibles las operaciones matemáticas sobre cadenas. Sin embargo, una vez realizada una operación matemática sobre una cadena, esta ya no se visualiza como una cadena, se ve como una matriz de números en el estandar ASCII.
- Para ver la representación ASCII de una cadena se toma su valor absoluto o se le suma 0.
- Para ver la cadena de caracteres se puede utilizar la función `setstr`.

10. OPERACIONES RELACIONALES Y LÓGICAS.

- MATLAB considera que cualquier número distinto de cero es verdadero y falso si es cero. La salida de todas las expresiones relacionales produce un uno si es verdadero y cero si es falso.
- Los operadores relacionales en MATLAB incluyen: `<` menor que, `<=` menor o igual que, `>` mayor que, `>=` mayor o igual que, `==` igual a, `~=` distinto de.
- Las comparaciones pueden hacerse comparando dos matrices del mismo tamaño o una matriz y un escalar.
- Los operadores lógicos proporcionan un medio para combinar o negar expresiones relacionales. MATLAB incluye: `&` AND, `|` OR, `~` NOT.
- MATLAB proporciona una serie de funciones relacionales y lógicas adicionales: `xor(x,y)` OR exclusivo, `any(x)` Devuelve uno si algún elemento en un vector `x` es no nulo y uno para cada columna en una matriz que tiene elementos no nulos, `all(x)` Devuelve uno si todos los elementos en un vector `x` son no nulos. `isnan(x)` devuelve unos en magnitudes no numéricas en `x`, `isinf(x)` devuelve unos en magnitudes infinitas, `finite(x)` devuelve unos en valores finitos de `x`.

11. ÁLGEBRA LINEAL Y MATRICES:

- Con la función `det` se obtiene el determinante de una matriz.

- La función `inv` calcula la inversa de una matriz. Si $\det(A)=0$ o está próximo a cero, se da un mensaje de aviso y los resultados pueden no tener sentido.

12. MANIPULACIÓN MATRICIAL:

- Los elementos matriciales se direccionan en el formato fila, columna: `A(filas, columnas)`.
- Los valores internos de una matriz se acceden identificando los subíndices de los elementos deseados.
- Utilizar el símbolo dos puntos como la designación de filas o columnas implica, respectivamente, todas las filas o columnas; por ejemplo, `A(2, :)` representa todas las columnas en la fila 2.
- Colocar datos fuera del rango actual de una matriz rellena con ceros las zonas no especificadas para mantener una forma de matriz rectangular.
- Fijar las filas o columnas de una matriz igual a la matriz vacía `[]` elimina estas filas y columnas.
- Usar sólo los dos puntos, por ejemplo, `A(:)`, reagrupa una matriz en un vector columna, tomando todas las columnas a un tiempo.
- Vectores lógicos 0-1 pueden utilizarse también para direccionar partes de un vector. En este caso, los vectores lógicos 0-1 deben tener el mismo tamaño que el vector que direcciona. Los elementos falsos (0) se eliminan, los elementos verdaderos (1) se retienen.
- La función `find` devuelve los subíndices o índices donde una expresión relacional es verdadera.
- La función `size` devuelve el número de filas y columnas de una matriz. La función `length` devuelve la longitud de un vector o la máxima dimensión de una matriz.
- `flipud(A)` intercambia una matriz de arriba abajo.
- `fliplr(A)` intercambia una matriz de izquierda a derecha.
- `rot90(A)` gira una matriz en dirección contraria a las agujas del reloj.
- `reshape(A,m,n)` devuelve una matriz $m \times n$ cuyos elementos se toman por columnas de A, A debe contener $m \times n$ elementos.
- `diag(v)` crea una matriz diagonal, con el vector v sobre la diagonal.
- `diag(A)` extrae la diagonal de la matriz A como un vector columna.

13. MATRICES ESPECIALES:

- `zeros(3)` produce una matriz 3x3 de ceros.
- `ones(2,4)` genera una matriz 2x4 de unos.
- `rand(3,1)` crea una matriz de números aleatorios distribuidos uniformemente entre cero y uno.

- `randn(2)` genera una matriz 2x2 de números aleatorios distribuidos normalmente con media cero y varianza uno.
- `eye(3)` crea una matriz identidad 3x3.

14. CONTROL DE FLUJO:

- `for x = array`
 órdenes
`end`
- `while expresión`
 órdenes
`end`
- `if expresión`
 órdenes
`end`
- `if expresión`
 órdenes evaluadas si exp verdadera
`else`
 órdenes evaluadas si exp falsa
`end`
- `if expresión1`
 órdenes evaluadas si exp1 verdadera
`elseif expresión2`
 órdenes evaluadas si exp2 verdadera
 .
 .
 .
`else`
 órdenes evaluadas si ninguna otra expresión
 es verdadera
`end`
- `break` termina la ejecución de bucles `for` y
 bucles `while`.

15. FUNCIONES EN ARCHIVOS-M:

- El nombre de la función y el nombre del archivo deben ser idénticos.
- Las líneas de comentarios hasta la primera línea de no comentario en un archivo-M de función son el texto de ayuda que se devuelve cuando se solicita con `help`.
- La primera línea de ayuda, conocida como la línea H1, es la línea que examina la orden `lookfor`.
- Todas las variables dentro de una función se aíslan del espacio de trabajo de MATLAB. Las únicas conexiones entre las variables dentro de una función y el espacio de trabajo de MATLAB son las variables de entrada y salida. Si una función cambia el valor de cualquier variable de entrada, los cambios aparecen sólo dentro de la función y no afectan a las variables en el espacio de trabajo de MATLAB.

- Cuando una función tiene más de una variable de salida, éstas se encierran entre corchetes, p. e. `[V,D] = eig (A)`. Esto es una sintaxis algo confusa ya que `[V,D]` en el lado derecho de un signo igual es la composición de una matriz de `V` y `D`.
- La primera vez que MATLAB ejecuta un archivo-M de función, abre el correspondiente archivo de texto y compila las órdenes en una representación interna que acelera su ejecución para todas las llamadas subsiguientes a la función.
- MATLAB busca los archivos-M de función de la misma forma que lo hace con los archivos-M script. Por ejemplo, si escribe `cow` en la línea de orden, MATLAB primero considera que `cow` es una variable. Si no lo es, entonces considera que es una función propia. Si no lo es, a continuación prueba si `cow.m` está en el directorio o carpeta actual. Si no está allí, examina todos los directorios o carpetas en el camino de búsqueda de MATLAB para ver si localiza a `cow.m`.
- El número de variables de entrada pasadas a una función está disponible en la variable `nargin`. El número de variables de salida solicitadas cuando una función se llama está disponible dentro de la función en la variable `nargout`.

16. ANÁLISIS DE DATOS:

- `corrcoef(x)` Coeficientes de correlación.
- `cov(x)` Matriz de covarianza.
- `cumprod(x)` Producto acumulativo de columnas.
- `cumsum(x)` Suma acumulativa de columnas.
- `diff(x)` Calcula las diferencias entre elementos.
- `hist(x)` Histograma o diagrama de barras.
- `mean(x)` Valor medio de columnas.
- `median(x)` Valor de la mediana de las columnas.
- `prod(x)` Producto de elemntos en columnas.
- `rand(x)` Números aleatorios distribuidos uniformemente.
- `randn(x)` Números aleatorios distribuidos normalmente.
- `sort(x)` Ordenar columnas en orden ascendente.
- `std(x)` Desviación estándar de columnas.
- `sum(x)` Suma de elementos en cada columna.

17. POLINOMIOS:

- Los polinomios se representan en MATLAB por vectores filas de sus coeficientes en orden descendente.

- `roots(p)` Calcula las raíces del polinomio.
- `poly(r)` Encuentra el polinomio asociado con las raíces.
- `conv(a, b)` Multiplica los dos polinomios a y b.
- `deconv(b, c)` Divide el polinomio b entre c.
- MATLAB no tiene incorporada una función para sumar polinomios. Sin embargo, es fácil construir un archivo-M de función que lo haga.
- `polyder(p)` Calcula la derivada del polinomio p.
- `polyval(p, x)` Evalúa el polinomio p en todos los valores de x.
- `residue(x, d)` Calcula el desarrollo en fracciones simples.
- `polyder(n, d)` Calcula la derivada del cociente de n a d donde n y d son polinomios.

18. AJUSTE DE CURVAS E INTERPOLACIÓN:

- La interpolación es el proceso de estimar valores intermedios de una función dados un conjunto de puntos que la describen.
- El ajuste de curvas o regresión es el proceso de encontrar la curva que “mejor se ajuste” a un conjunto de puntos. El “mejor ajuste” generalmente no pasa a través de todos los puntos.
- El ajuste de curvas por mínimos cuadrados a un polinomio encuentra los coeficientes del polinomio que minimizan la suma del error al cuadrado en los puntos, donde el error se define como la diferencia entre los puntos conocidos y el polinomio evaluado en esos puntos.
- La función `polyfit(x, y, n)` devuelve los coeficientes del polinomio de orden n-ésimo que minimiza el error de mínimos cuadrados dados los datos en x e $y = f(x)$.
- La función `polyval(p, x)` evalúa el polinomio especificado por p en el punto dado en x.
- Al menos se requieren $n+1$ puntos para el ajuste de un polinomio de orden n-ésimo.
- El orden del polinomio escogido para el ajuste de curvas debe permanecer pequeño.
- La interpolación es el proceso de estimar valores de una función fuera de aquellos dados por un conjunto de puntos.
- La interpolación no es generalmente útil para estimar valores de una función entre aquellos fuera de aquellos dados por un conjunto de puntos.
- Las variables independientes utilizadas para la interpolación deben ser monótonas.
- La interpolación unidimensional estima $y=f(x)$ para valores de x dado el conocimiento de pares de datos $\{x_i, y_i\}$ que describen $y=f(x)$.
- La interpolación bi-dimensional estima $z=f(x, y)$ para valores de x e y dado el conocimiento de tripletes de datos $\{x_i, y_i, z_i\}$ que describen $z=f(x, y)$.

- Las funciones `interp1`, e `interp2` permiten hacer interpolaciones en una y dos dimensiones respectivamente.

19. ANÁLISIS NUMÉRICO:

- `fplot(nombre_de_función, [inferior, superior])` representa una función entre los valores `inferior` y `superior`. Esta función es útil para representar funciones que no son fácilmente representables usando la técnica estándar de evaluar sobre un rango espaciado linealmente.
- `fmin(nombre_de_función, [inferior, superior])` encuentra el mínimo de una función unidimensional entre los valores `inferior` y `superior`.
- `fzero(nombre_de_función, estimación)` encuentra un cero de una función cerca de la estimación inicial.
- `trapz(x, y)` aproxima el área bajo la función $y=f(x)$ donde los datos son `x` e `y`, utilizando el método trapezoidal.
- `quad(nombre_de_función, inferior, superior)` y `quad8(nombre_de_función, inferior, superior)` calcula el área bajo una función de `inferior` a `superior` usando el método de cuadratura.
- `diff(y)./diff(x)` aproxima la derivada de $y=f(x)$ usando diferencias finitas.
- `ode23(nombre_de_función, t0, tf, y0)` y `ode45(nombre_de_función, t0, tf, y0)` integra un conjunto de ecuaciones diferenciales descritas en la función `nombre_de_función` desde un instante inicial `t0`, a un instante final `tf`, comenzando con la condición inicial `y0`.
- La diferenciación basada en el ajuste de curvas a menudo conduce a mejores estimaciones de la derivada.
- Todas las funciones de análisis numérico aproximan a la solución realmente buscada. La única forma de obtener la solución exacta es resolver el problema a mano, analíticamente.
- La minimización y la determinación de ceros no siempre converge a soluciones viables. Como consecuencia de ello, es siempre bueno comprobar los resultados.
- El *toolbox de matemática simbólica* ofrece muchas capacidades para realizar la localización de ceros, integración, diferenciación y la resolución de ecuaciones diferenciales ordinarias.