Guión de la Práctica 1 – Introducción a MATLAB

Procesamiento de Señales Biomédicas
Universidad de Granada

Joaquín T. Valderrama*

Índice

*Email: jvalderrama@ugr.es — Página web

1. Qué es MATLAB

MATLAB es un entorno de programación y software de alto nivel ampliamente utilizado en ingeniería y ciencias para realizar cálculos matemáticos, análisis de datos, visualización y desarrollo de algoritmos. El nombre MATLAB es un acrónimo de 'Laboratorio de matrices' (por sus siglas en inglés, *matrix laboratory*). Es especialmente útil en áreas que requieren procesamiento de datos complejos y análisis, ya que incluye herramientas para manipular matrices, realizar cálculos matemáticos avanzados y generar gráficos de alta calidad.

En el campo del **procesamiento de señales biomédicas**, como señales de electroencefalograma (EEG) y electrocardiograma (ECG), MATLAB es una herramienta de gran valor por su capacidad de análisis detallado y su flexibilidad para desarrollar soluciones personalizadas en investigación y diagnóstico biomédico.

MATLAB es un software de pago desarrollado por MathWorks (Natick, Massachussetts), con licencias específicas para distintos tipos de usuarios. La Universidad de Granada dispone de una licencia educativa, que permite a estudiantes y docentes acceder a MATLAB en los laboratorios y en actividades académicas. Sin embargo, para quienes buscan una alternativa gratuita, existe GNU Octave, un programa de libre acceso y código abierto que ofrece funciones y sintaxis similares a MATLAB. Octave es una opción práctica para realizar cálculos matemáticos, análisis de datos y procesamiento de señales, aunque puede presentar algunas limitaciones en comparación con MATLAB en términos de velocidad y ciertas funcionalidades específicas. La práctica de esta sesión puede desarrollarse tanto en MATLAB como en GNU Octave.

2. Estructura de la sesión

Guión Esta práctica está diseñada para realizarse en 2 horas de estudio y 2 horas de trabajo individual. Este guión presenta definiciones y ejemplos que te permitirán comprender y familiarizarte con herramientas importantes de procesado de señal en MATLAB. Se recomienda leer el guión de manera estructurada sin saltarse ningún campo, y ejecutar en MATLAB los comandos y las instrucciones del guión para visualizar cada unidad de aprendizaje. Pregunta al profesor cualquier concepto que no quede claro en el guión, cualquier duda que te surja, y cualquier error de programación que no consigas resolver de manera autónoma. Si ya dominas los conceptos, puedes avanzar más rápidamente sin necesidad de ejecutar en MATLAB las instrucciones y los ejemplos.

Evaluación Una vez recorrido el guión, deberás realizar de manera individual una serie de ejercicios en MATLAB. Estos ejercicios están presentados en el documento 'Ejercicios.pdf'. Esta práctica se evaluará mediante la entrega de una memoria a través de PRADO en la que se muestre la resolución de estos ejercicois. Es obligatorio generar la memoria en LaTeX. Para ello tienes disponible la carpeta 'Memoria', la cual contiene una plantilla de LaTeX que puedes modificar para generar la memoria. Los **criterios de evaluación** considerarán tanto si las respuestas son correctas y están completas, como la expresión escrita y la claridad de tus respuestas.

Objetivos de aprendizaje

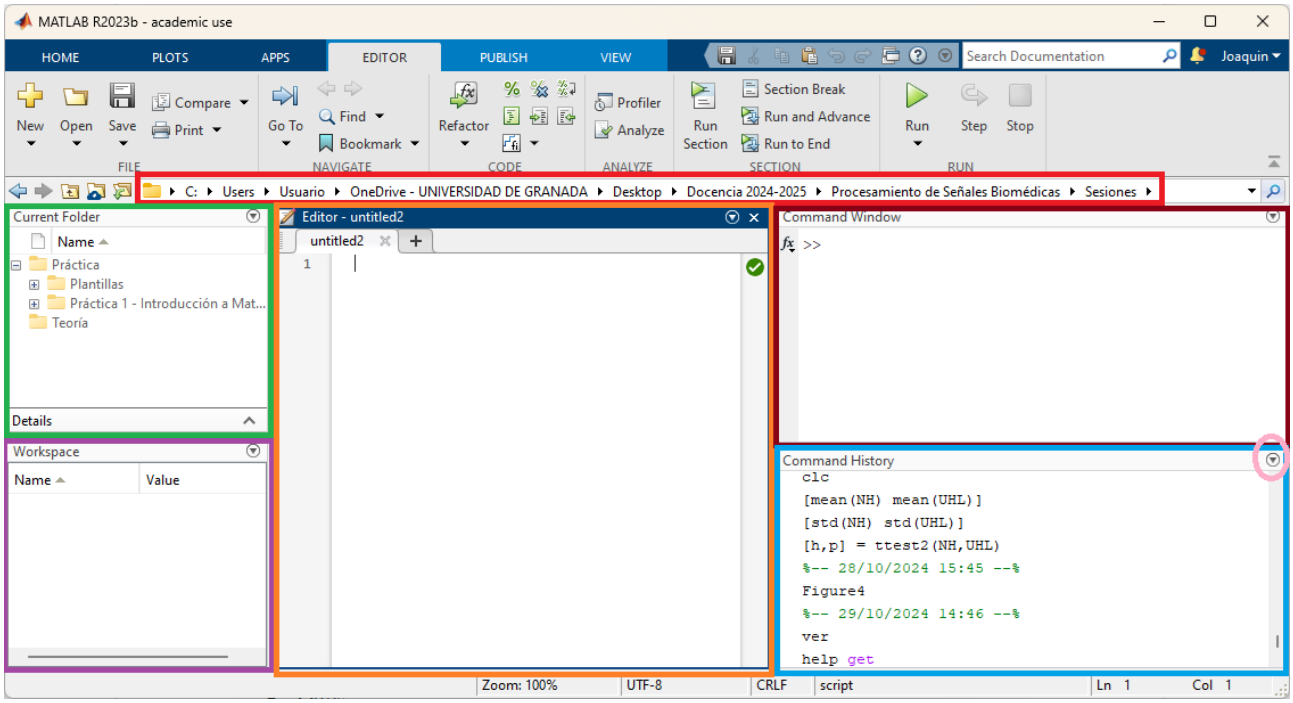
1. **Familiarizarse con el entorno de MATLAB:** Aprender a navegar y utilizar distintas secciones del entorno de MATLAB, como el editor y el workspace, para comprender cómo se organiza y gestiona el código.
2. **Manipulación y operaciones básicas:** Practicar operaciones iniciales con variables, incluyendo asignaciones, tipos de datos, y operaciones con vectores y matrices, para desarrollar una base sólida en el manejo de datos en MATLAB.
3. **Implementación de estructuras de control:** Aprender a utilizar las estructuras de control de flujo (como `if`, `for`, `while` y `switch`) para dirigir la ejecución del código, lo que permite implementar lógica condicional.
4. **Creación de gráficos personalizados:** Explorar funciones avanzadas del comando `plot` para producir visualizaciones efectivas y exportarlas en diferentes formatos.
5. **Mejorar tu dominio de LaTeX:** Realizar la memoria en LaTeX reforzará tu habilidad con esta potente herramienta.

3. Entorno de la aplicación

MATLAB puede ejecutarse como cualquier otra aplicación: haciendo doble click sobre el icono de la aplicación. La figura de abajo muestra el programa al iniciarse. En esta figura se pueden identificarse varias secciones.

- **Current Work Directory** (en rojo): es la carpeta de trabajo actual en la que MATLAB busca y guarda archivos por defecto. Aquí se pueden organizar y gestionar los archivos que se usarán en el proyecto activo.
- **Current Folder** (en verde): muestra el contenido de la carpeta actual de trabajo, permitiendo acceder rápidamente a archivos y subcarpetas. Desde esta sección, se pueden abrir, renombrar y gestionar archivos directamente.
- **Workspace** (en morado): almacena todas las variables activas durante la sesión, mostrando sus nombres, valores y tipos de datos. Permite visualizar y manipular fácilmente las variables en uso.
- **Editor** (en naranja): es la sección donde se escriben y editan scripts y funciones en MATLAB. Aquí se desarrollan programas más largos y complejos, permitiendo guardar el código y ejecutarlo según se necesite.
- **Command Window** (en marrón): es el área donde se introducen y ejecutan comandos de manera interactiva. Se utiliza para cálculos rápidos, pruebas de código y ejecución de comandos individuales.
- **Command History** (en azul): muestra un historial de todos los comandos introducidos en la Command Window, permitiendo revisar, copiar y reutilizar comandos previos en futuras sesiones o en la misma sesión actual.

La flecha hacia abajo situada en la parte superior derecha de cada sección (círculo rosa en la figura) permite cerrar, minimizar, maximizar, presentar la sección en una ventana diferente o insertar la sección en la ventana principal de la aplicación. La configuración del entorno que yo recomiendo consiste en cerrar las secciones 'Current Folder' y 'Command History', y dejar visible en una única ventana las secciones 'Workspace' en la columna de la izquierda, 'Editor' en la columna central y 'Command Window' en la columna de la derecha. Las secciones pueden reorganizarse pinchando en el cabecero de la sección y arrastrándola a la posición deseada.



4. Operaciones con variables

4.1. Primeros pasos

- Puedes ejecutar comandos introduciéndolos en la ventana de comandos (*Command window*) después del indicador de MATLAB (>>) y pulsando la tecla **Entrar**. Por ejemplo, multiplica los números 3 y 7 con el comando `3*7`. Podrás observar que se ha almacenado el resultado en una variable denominada `ans`, la cual está visible en el espacio de trabajo (*Workspace*).
- Asigna ahora el valor `3*5` a una nueva variable llamada `m` de esta forma `m=3*5`. En este caso, la variable `m` está almacenada en el espacio de trabajo, con un valor de `15`.
- Introduce ahora el valor `m=m+1` a ver qué ocurre. Observa que ahora el valor de `m` en el espacio de trabajo es de `16`.
- Crea ahora una nueva variable `y` con el valor de `m/2`.
- Si agregas un punto y coma al final de un comando, se suprime la salida, aunque el comando seguirá ejecutándose como puedes ver en el espacio de trabajo. Compara la salida de `x=6+3` con `x=6+3;`
- Puedes también recuperar comandos anteriores pulsando la tecla de la flecha hacia arriba del teclado. Ten en cuenta que para que esto funcione, la ventana activa debe ser la ventana de comandos. Pulsa la tecla hacia arriba varias veces hasta recuperar el comando `m=3*5` y edítalo para obtener el comando `m=3*x`.
- Si quieres ver el valor de una variable, puedes simplemente introducir esa variable en la ventana de comandos y pulsar 'Enter'. Visualiza el valor de la variable `y` en la ventana de comandos.
- Puedes asignar el nombre que desees a las variables de MATLAB, siempre que empiece por una letra y contenga sólo letras, números y guiones bajos. Asigna el valor `-4` a la variable `A5`.
- Utiliza el comando `clear` para eliminar todas las variables del espacio de trabajo. Este comando es muy útil cuando comiences una nueva tarea y desees comenzar con un espacio de trabajo sin ninguna variable.
- Utiliza el comando `clc` para vaciar la ventana de comandos. `clear` y `clc` son dos comandos muy utilizados al comienzo de cualquier nueva tarea.
- MATLAB tiene constantes incorporadas, como `pi` para representar el número π. Cree una variable llamada `x` con un valor de $\pi/2$.
- MATLAB contiene también una amplia variedad de funciones incorporadas, como `abs` para calcular el valor absoluto, `cos` para calcular el coseno y `sqrt` para calcular la raíz cuadrada. El apéndice de este guión presenta una lista de algunas funciones importantes incorporadas en MATLAB.
- Almacena en la variable `z` la raíz cuadrada de `-9`. Observa que la solución contiene el número imaginario `i`, que es una constante incorporada en MATLAB.
- Por defecto MATLAB sólo muestra los primeros cuatro decimales en la ventana de comandos. Puedes controlar la precisión visualizada con la función `format`. Introduce `format long` y visualiza el valor de x. Repite el proceso tras introducir el comando `format short` para volver a la visualización predeterminada.
- Crea una carpeta en el escritorio de Windows (o en el directorio que desees) con el nombre de esta práctica. Ahora copia el directorio de la carpeta y pégalo el directorio de trabajo de MATLAB. De esta manera, MATLAB operará en el directorio de la carpeta que has creado. A continuación, ejecuta el comando `save MisPrimerosPasos` en la ventana de comandos. Este comando creará un nuevo archivo denominado `MisPrimerosPasos` con extensión `.mat` en la carpeta que has creado en la que se almacenarán todas las variables del espacio de trabajo. ¿Puedes ver este nuevo archivo?

4.2. Tipos de variables

En MATLAB, los tipos de variables se utilizan para almacenar diferentes tipos de datos y se pueden clasificar en varias categorías según su contenido. A continuación, se presenta una descripción breve de los tipos de variables más comunes:

- **Numéricas**:
  - **Double**: Es el tipo de variable predeterminado en MATLAB y almacena números de punto flotante de doble precisión.
  - **Single**: Almacena números de punto flotante de precisión simple, ocupando menos memoria que el tipo double.
  - **Integer**: Se pueden especificar variables enteras de diferentes tamaños (e.g., `int8`, `uint16`), con o sin signo. Esto permite ahorrar memoria cuando no se necesitan decimales.
- **Caracteres y Cadenas**:
  - **Char**: Es un arreglo de caracteres, utilizado para almacenar texto como una secuencia de caracteres individuales.
  - **String**: Introducido en versiones más recientes, representa cadenas de texto completas como un solo objeto, lo cual facilita la manipulación de texto.
- **Lógicas (Booleanas)**:
  - **Logical**: Almacena valores lógicos (`true` o `false`). Se utiliza principalmente para operaciones condicionales y almacenamiento de resultados de comparaciones.
- **Estructuras**:
  - **Struct**: Permite almacenar datos heterogéneos en una sola variable, organizados en campos. Cada campo puede contener diferentes tipos de datos (e.g., números, cadenas, matrices).
- **Celdas**:
  - **Cell**: Es un tipo de contenedor que permite almacenar diferentes tipos de datos en cada elemento. Las celdas son útiles para almacenar matrices de diferentes tamaños y tipos en una sola estructura.
- **Matrices y Arreglos**:
  - MATLAB permite crear matrices y arreglos de diferentes dimensiones (1D, 2D, 3D, etc.), que pueden ser de cualquier tipo de datos, incluidos double, single, char, etc. Estas estructuras son fundamentales para el procesamiento de datos y cálculo numérico en MATLAB.
- **Tablas y Tiempos**:
  - **Table**: Organiza datos heterogéneos en formato tabular, permitiendo que cada columna contenga un tipo de datos distinto, muy útil para datos tipo *dataset*.
  - **Datetime** y **Duration**: Para trabajar con datos de fechas y tiempos, MATLAB ofrece estos tipos específicos que permiten manipular datos temporales con precisión.

Cada tipo de variable tiene funciones y métodos específicos que facilitan su manipulación, cálculo y visualización en MATLAB.

4.3. Operaciones con vectores y matrices

En MATLAB, los vectores y matrices se pueden crear utilizando corchetes `[]` para agrupar los elementos. Los elementos dentro de un vector o matriz se separan por espacios o comas, mientras que los saltos de línea se indican con punto y coma `;`.

- **Vector fila**: Para crear un vector fila, se colocan los elementos entre corchetes y se separan por comas o espacios: `v = [1, 2, 3]`
- También puede utilizarse la sintaxis `inicio:paso:fin` para generar vectores, siendo `inicio` el valor inicial del vector, `paso` la diferencia entre cada elemento consecutivo, y `fin` el valor máximo que el vector puede alcanzar (no incluido). Por ejemplo, el comando `1:3:15` daría como resultado el vector `[1, 4, 7, 10, 13]`
- **Vector columna**: Para crear un vector columna, se utilizan punto y coma entre cada elemento: `v = [1; 2; 3]`
- **Matriz**: Para crear una matriz, se organiza cada fila separando elementos con espacios o comas, y se usa punto y coma para indicar el cambio de fila: `A = [1, 2, 3; 4, 5, 6; 7, 8, 9]`
- **Elemento de un Vector**: Para acceder al i-ésimo elemento de un vector `v`, se usa la sintaxis `v(i)`

```
v = [10, 20, 30]
elemento = v(2)  % Devuelve 20
```

- **Elemento de una Matriz**: Para acceder al elemento en la fila `i` y columna `j` de una matriz `A`, se usa la sintaxis `A(i, j)`

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9]
elemento = A(2, 3)  % Devuelve 6
```

- **Acceso a Submatrices**: Para extraer una submatriz, se pueden especificar rangos de filas y columnas. Por ejemplo, para extraer las dos primeras filas y columnas de `A`:

```
submatriz = A(1:2, 1:2)
```

Esto devolverá:

```
[1, 2;
 4, 5]
```

- **Acceso a toda una fila o columna**: Para acceder a una fila o columna completa, se usa el símbolo `:`

```
fila1 = A(1, :)    % Primera fila: [1, 2, 3]
columna2 = A(:, 2) % Segunda columna: [2; 5; 8]
```

- **Suma y resta**: Las operaciones de suma y resta se realizan con los operadores + y −. Estas operaciones requieren que las matrices o vectores tengan las mismas dimensiones.

```
C = A + B
D = A - B
```

- **Multiplicación de matrices**: La multiplicación de matrices se realiza usando el operador `*`, que sigue las reglas de álgebra lineal (multiplicación de filas por columnas): `C = A * B` Para esta operación, el número de columnas de la primera matriz debe coincidir con el número de filas de la segunda matriz.
- **Multiplicación elemento a elemento**: Si se desea multiplicar cada elemento de una matriz con el correspondiente en otra matriz, se usa el operador `.*`: `C = A .* B` En este caso, ambas matrices deben tener el mismo tamaño.
- **Potenciación elemento a elemento**: Para elevar cada elemento de una matriz a una potencia, se usa el operador `.^`: `C = A .^ 2` Este comando eleva cada elemento de `A` al cuadrado.

4.4. Operaciones lógicas

En MATLAB, las operaciones lógicas se utilizan para realizar comparaciones entre valores o matrices. Estas operaciones devuelven valores lógicos: `true` (1) o `false` (0). Las operaciones lógicas son fundamentales para la toma de decisiones y la manipulación condicional de datos en los programas.

- **Operadores de Comparación**:
  - == Igualdad. Comprueba si dos valores o matrices son iguales elemento a elemento.

    ```
    A == B
    ```

  - = Desigualdad. Comprueba si dos valores o matrices son diferentes.

    ```
    A ~= B
    ```

  - < Menor que. Comprueba si un valor es menor que otro.
  - > Mayor que. Comprueba si un valor es mayor que otro.
  - <= Menor o igual que.
  - >= Mayor o igual que.
- **Operadores Lógicos**:
  - `&` Y lógico (AND). Devuelve `true` si ambos operandos son `true`.

    ```
    A & B
    ```

  - `|` O lógico (OR). Devuelve `true` si al menos uno de los operandos es `true`.

    ```
    A | B
    ```

  - `~` Negación lógica (NOT). Invierte el valor lógico de un operando.

    ```
    ~A
    ```

- **Operaciones Lógicas Elemento a Elemento**: Para aplicar operaciones lógicas entre matrices de forma elemento a elemento, se usa el operador punto antes de `&` y `|`, como `.&` y `.|`.

```
A .& B
A .| B
```

Estas operaciones requieren que las matrices `A` y `B` tengan el mismo tamaño.

Ejemplo

Si `A = [1, 2; 3, 4]` y `B = [2, 2; 3, 1]`, entonces:

```
A > B      % Devuelve [0, 0; 0, 1]
(A >= 2) & (B < 3)  % Devuelve [1, 1; 0, 0]
```

Este tipo de operaciones lógicas en MATLAB es útil para realizar filtrado de datos y evaluaciones condicionales dentro de matrices.

4.5. Operaciones con números complejos

En MATLAB, los números complejos se manejan de forma sencilla utilizando la letra `i` o `j` para representar la unidad imaginaria $\sqrt{-1}$. Un número complejo puede escribirse directamente en MATLAB como `a + bi` o `a + bj`, donde `a` es la parte real y `b` es la parte imaginaria.

- **Creación de números complejos**:

```
z = 3 + 4i
```

  Aquí, `z` es un número complejo con parte real 3 y parte imaginaria 4.

- **Operaciones básicas**: MATLAB permite realizar operaciones básicas de suma, resta, multiplicación y división con números complejos, de manera similar a los números reales.

```
z1 = 3 + 4i;
z2 = 1 - 2i;

suma = z1 + z2       % Suma de números complejos
resta = z1 - z2      % Resta de números complejos
producto = z1 * z2   % Multiplicación de números complejos
division = z1 / z2   % División de números complejos
```

- **Funciones para números complejos**: MATLAB proporciona varias funciones para trabajar con números complejos:
  - `real(z)`: Devuelve la parte real de `z`.
  - `imag(z)`: Devuelve la parte imaginaria de `z`.
  - `abs(z)`: Calcula el módulo o magnitud de `z`, es decir, $|z| = \sqrt{a^2 + b^2}$.
  - `angle(z)`: Calcula el argumento de `z` en radianes.
  - `conj(z)`: Devuelve el conjugado complejo de `z`, es decir, `a - bi`.

- **Ejemplo**. Si `z1 = 3 + 4i` y `z2 = 1 - 2i`, entonces:

```
real(z1)        % Devuelve 3
imag(z1)        % Devuelve 4
abs(z1)         % Devuelve 5, que es la magnitud de z1
angle(z1)       % Devuelve el argumento de z1 en radianes
conj(z1)        % Devuelve 3 - 4i, el conjugado de z1
```

5. Estructuras de control de flujo

MATLAB proporciona varias estructuras de control de flujo para dirigir la ejecución de un programa en función de condiciones o repeticiones. A continuación, se explican las estructuras básicas: `if`, `for`, `while` y `switch`. Estas estructuras de control son fundamentales para implementar lógica condicional y repetitiva en MATLAB, permitiendo crear programas más dinámicos y flexibles.

5.1. Comando *if*

Condicional `if`: La estructura `if` permite ejecutar un bloque de código si se cumple una condición. También se pueden usar `elseif` y `else` para evaluar condiciones adicionales.

```
x = 5;
if x > 0
    disp('x es positivo')
elseif x == 0
    disp('x es cero')
else
    disp('x es negativo')
end
```

5.2. Comando *for*

Bucle `for`: El bucle `for` se utiliza para repetir un bloque de código un número específico de veces. En MATLAB, se especifica el rango del índice de la siguiente manera:

```
for i = 1:5
    disp(i)
end
```

En este ejemplo, el valor de `i` toma valores desde 1 hasta 5, imprimiendo cada uno en cada iteración.

5.3. Comando *while*

Bucle `while`: El bucle `while` ejecuta un bloque de código mientras se cumpla una condición. Este tipo de bucle es útil cuando no se conoce de antemano el número de repeticiones.

```
x = 1;
while x <= 5
    disp(x)
```

```
    x = x + 1;
end
```

En este caso, el código imprime los valores de `x` desde 1 hasta 5 y se detiene cuando `x` es mayor que 5.

5.4. Comando *switch*

Estructura `switch`: La estructura `switch` permite seleccionar entre múltiples opciones basadas en el valor de una variable. Es útil cuando se desea evaluar varias condiciones de igualdad.

```
color = 'rojo';
switch color
    case 'rojo'
        disp('Alto')
    case 'amarillo'
        disp('Precaución')
    case 'verde'
        disp('Avance')
    otherwise
        disp('Color desconocido')
end
```

En este ejemplo, el bloque imprime "Alto" si el valor de `color` es "rojo", "Precaución" si es "amarillo", y "Avance" si es "verde". La opción `otherwise` se ejecuta si el valor de `color` no coincide con ninguno de los casos anteriores.

6. Scripts y funciones

6.1. Scripts

En MATLAB, un **script** es un archivo de texto que contiene una secuencia de comandos MATLAB que se ejecutan en el orden en que aparecen. Los scripts son útiles para automatizar tareas, realizar cálculos repetitivos y organizar el código para una mejor legibilidad y reutilización.

Para crear un script en MATLAB, sigue estos pasos:

- Abre el `Editor` de MATLAB desde la barra de herramientas o utiliza el comando `edit` en la línea de comandos.
- Escribe tus comandos MATLAB en el editor.
- Guarda el archivo con la extensión `.m`. Por ejemplo, `mi_script.m`.

Un ejemplo de contenido de un script podría ser:

```
% mi_script.m
% Este script calcula el área de un círculo

radio = 5;  % Definir el radio
area = pi * radio^2;  % Calcular el área
disp(['El área del círculo es: ', num2str(area)]);  % Mostrar el resultado
```

Para ejecutar un script en MATLAB, simplemente escribe el nombre del script (sin la extensión `.m`) en la línea de comandos y presiona `Enter`:

```
>> mi_script
```

Al ejecutar el script, MATLAB ejecutará todos los comandos en el archivo en el orden en que fueron escritos. Sus principales **ventajas** son:

- **Reutilización de código**: Puedes guardar scripts con funciones específicas y reutilizarlos en diferentes sesiones de trabajo.
- **Organización**: Los scripts permiten organizar y documentar el código de manera clara, facilitando su comprensión y mantenimiento.
- **Automatización de tareas**: Los scripts son ideales para ejecutar tareas repetitivas o realizar análisis de datos de manera eficiente.

6.2. Funciones

En MATLAB, una **función** es un bloque de código que realiza una tarea específica y que puede ser reutilizado en diferentes partes del programa o en otros scripts. Las funciones en MATLAB permiten organizar el código, evitando la repetición y facilitando la depuración y el mantenimiento del mismo.

Las funciones son una herramienta clave en la programación en MATLAB porque permiten dividir el código en partes más pequeñas y manejables, cada una con una tarea específica. Esto hace que el código sea más modular y que las tareas repetitivas puedan realizarse sin escribir el mismo bloque de código varias veces. Además, al definir funciones, es posible reducir el riesgo de errores, ya que los cambios realizados en una función se aplican en todas las partes del programa que la utilizan.

El uso de funciones en MATLAB ofrece diversas ventajas:

- **Reutilización de código:** Las funciones permiten usar el mismo código en múltiples lugares sin tener que escribirlo nuevamente.
- **Modularidad:** Ayudan a organizar el código en módulos independientes, lo cual facilita su lectura y mantenimiento.
- **Mantenimiento:** Al centralizar la lógica en una función, cualquier cambio en el comportamiento solo requiere modificar la función y no todas las partes donde se utiliza.
- **Depuración:** Es más sencillo identificar y corregir errores en una función que en un programa extenso.

A continuación, se muestra un ejemplo de cómo crear una función simple en MATLAB para calcular el cuadrado de un número:

```
function result = squareNumber(x)
    % squareNumber calcula el cuadrado del número de entrada x
    result = x^2;
end
```

En este ejemplo:

- La palabra clave `function` indica el inicio de la función.
- `result` es la variable de salida, que contendrá el resultado del cálculo.

- `squareNumber` es el nombre de la función.
- `x` es el argumento de entrada, es decir, el valor que será cuadrado.

Para utilizar esta función, es suficiente con llamar a `squareNumber` y pasarle un valor, por ejemplo:

```
y = squareNumber(5);
disp(y); % Salida: 25
```

De esta manera, la función `squareNumber` puede ser utilizada en cualquier parte del programa para calcular el cuadrado de un número, promoviendo la reutilización y organización del código.

7. Representación avanzada de señales

En MATLAB, el comando `plot` permite crear gráficos con un alto nivel de personalización. Esta sección cubre algunas funciones avanzadas para ajustar el tamaño de la figura, organizar subplots, modificar el estilo de las líneas y marcadores, establecer la escala de los ejes, añadir títulos y leyendas, y guardar la figura en distintos formatos.

7.1. Generar una figura de tamaño específico

Para definir el tamaño de una figura, se usa el comando `figure` con las propiedad `PaperSize` y `Units`:

```
figure('PaperSize', [ancho, alto], 'Units', 'centimeters');
```

Aquí, `ancho` y `alto` definen el tamaño de la figura en centímetros.

7.2. Crear subplots en posiciones predeterminadas

Para organizar gráficos dentro de una figura, se utiliza la función `subplot`:

```
subplot(filas, columnas, posición);
plot(x, y);
```

Por ejemplo, `subplot(2, 2, 3)` crea una cuadrícula de 2x2 y coloca el gráfico en la tercera posición.

7.3. Personalizar el estilo de líneas y marcadores

Se pueden ajustar varias propiedades del gráfico como el grosor de la línea, el color, el tipo de marcador, y su tamaño:

```
plot(x, y, 'LineWidth', 2, 'Color', [0.1, 0.5, 0.9], ...
    'Marker', 'o', 'MarkerSize', 8);
```

En este ejemplo:

- `LineWidth` establece el grosor de la línea.
- `Color` permite especificar el color de la línea con un vector RGB.
- `Marker` define el tipo de marcador (por ejemplo, `'o'` para círculos).
- `MarkerSize` fija el tamaño del marcador.

7.4. Fijar la escala de los ejes

Para ajustar los límites de los ejes, se usan las funciones `xlim` y `ylim`:

```
xlim([minX maxX]);
ylim([minY maxY]);
```

También se pueden definir escalas logarítmicas con `set(gca, 'XScale', 'log')` para el eje X, o `YScale` para el eje Y.

7.5. Añadir título y etiquetas con tamaño de fuente personalizado

Para personalizar el título y las etiquetas de los ejes, se emplean las funciones `title`, `xlabel`, y `ylabel`:

```
title('Título de la figura', 'FontSize', 14);
xlabel('Eje X', 'FontSize', 12);
ylabel('Eje Y', 'FontSize', 12);
```

Aquí, la propiedad `FontSize` permite definir el tamaño de la fuente de los textos.

7.6. Añadir una leyenda

Para identificar diferentes líneas en el gráfico, se puede añadir una leyenda con el comando `legend`:

```
plot(x, y1, 'r', x, y2, 'b');
legend('Datos 1', 'Datos 2', 'FontSize', 10, 'Location', 'northeast');
```

En este ejemplo:

- `legend` permite crear una leyenda para cada línea.
- Cada cadena en la función `legend` se asocia con las líneas en el orden de su aparición en el comando `plot`.
- `FontSize` ajusta el tamaño del texto en la leyenda, y `Location` establece su posición dentro de la figura, por ejemplo, en la esquina superior derecha (`'northeast'`).

7.7. Guardar la figura en distintos formatos y resolución

MATLAB permite exportar las figuras en diferentes formatos y con una resolución específica mediante la función `saveas` o `print`:

```
print('nombre_figura', '-dpng', '-r300');
```

En este ejemplo:

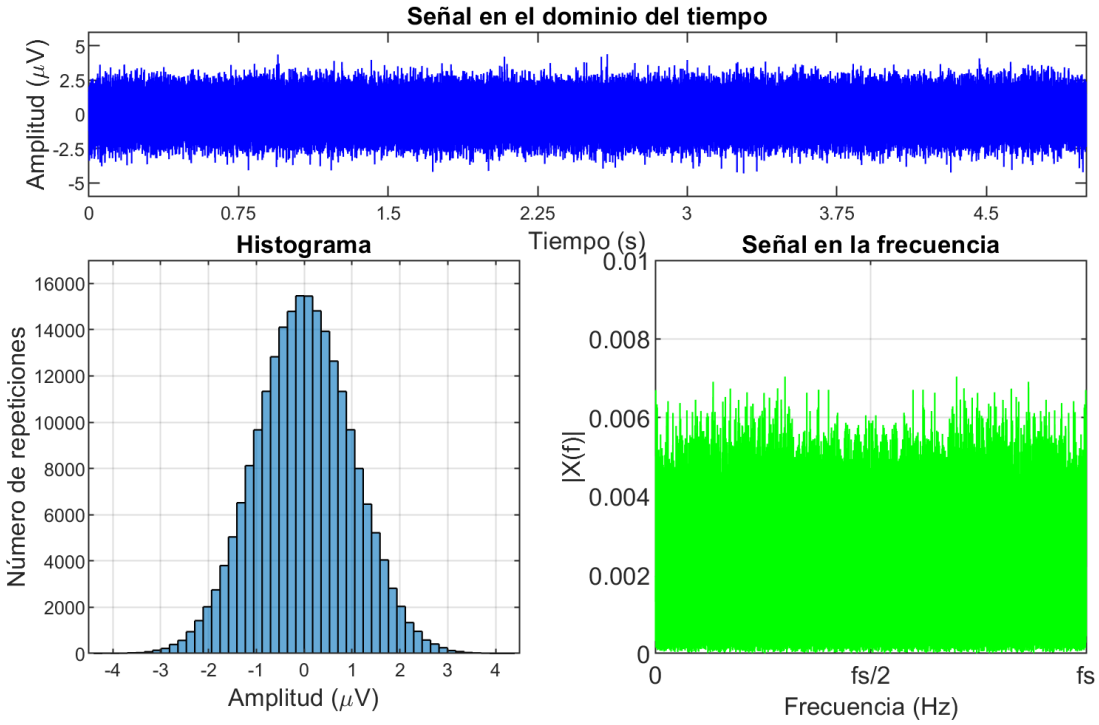
- `'nombre_figura'` es el nombre del archivo de salida.
- `-dpng` especifica el formato (también puede ser `-depsc` para EPS o `-dpdf` para PDF).
- `-r300` fija la resolución a 300 ppp (puntos por pulgada).

7.8. Ejemplo completo

A continuación se muestra un ejemplo completo de uso de estas opciones avanzadas:

```
%% Script que genera una figura utilizando funciones avanzadas
clear                                % Borra el espacio de trabajo
close all                            % Cierra todas las figuras
clc                                  % Borra la ventana de comandos

%% Datos a representar
fs = 44100;                          % Frecuencia de muestreo (Hz)
t = 0:1/fs:5-1/fs;                   % Eje de tiempo: 5 segundos de duracion
L = length(t);                       % Numero de muestras de la senal
f = 0:fs/L:fs-1/L;                   % Eje de frecuencia
x = randn(L,1);                      % Ruido blanco Gaussiano
X = abs(fft(x))/L;                   % Modulo de la Transformada de Fourier

%% Generacion de la figura
% Creamos una figura de 16 cm de ancho y 10 cm de alto
figure('PaperSize',[16,10],'Units','centimeters')
FS = 8;                              % Tamano de la fuente

% Creamos un estructura de 3 filas x 2 columnas (6 posibles posiciones), y
% representamos la senal en el tiempo en los dos subplots superiores
subplot(3,2,1:2)
plot(t,x,'-b')       % Senal en el dominio del tiempo mediante linea en azul
set(gca,'xtick',0:0.75:5,'FontSize',FS-2);    % Eje de abscisa cada 1.5 seg
set(gca,'ytick',-5:2.5:5,'FontSize',FS-2);  % Eje de ordenadas cada 2.5 uV
axis([0 5 -6 6])         % El eje x va desde 0 a 5, y el eje y de -6 a 6
title('Senal en el dominio del tiempo','FontSize',FS) % Titulo de la figura
xlabel('Tiempo (s)','FontSize',FS)       % Etiqueta del eje x
ylabel('Amplitud (\muV)','FontSize',FS) % Etiqueta del eje y

% Utilizamos la misma estructura de 3 x 2 subplots y representamos
% el histograma en los dos subplots de la izquierda (subplots 3 y 5)
subplot(3,2,[3,5])
histogram(x,50)      % Funcion de MATLAB que calcula y pinta el histograma
set(gca,'xtick',-5:5,'FontSize',FS-2)
grid on              % Activa el grid de la figura
axis([-4.5 4.5 0 17000])     % Definimos los margenes de los ejes
title('Histograma','FontSize',FS)
xlabel('Amplitud (\muV)','FontSize',FS)
ylabel('Numero de repeticiones','FontSize',FS)

% Ahora representamos el modulo de la Transformada de Fourier en los dos
% subplots de la derecha (los subplots 4 y 6)
subplot(3,2,[4,6])
plot(f,X,'-g')               % Senal representada en color verde
% Utilizamos 'xtick' y 'xticklabel' para nombrar valores del eje x
set(gca,'xtick',[0,fs/2,fs],'xticklabel',{'0','fs/2','fs'},'FontSize',FS)
grid on                      % Activa el grid de la figura
axis([0 fs 0 0.01])          % Definimos los margenes de los ejes
title('Senal en la frecuencia','FontSize',FS)
xlabel('Frecuencia (Hz)','FontSize',FS)
ylabel('|X(f)|','FontSize',FS)

%% Imprimimos la figura en formato .png con una resolucion de 300 ppi
orient tall                                % Orienta la figura para imprimir
print('-dpng','-r300','Figura_P1')        % Imprimimos la figura
close all                                  % Cerramos la figura
```

Dando como resultado la siguiente figura. En la parte superior se muestra un segmento de 5 segundos de ruido blanco Gaussiano. En el panel de abajo a la izquierda aparece el histograma de esa señal, correspondiente con una Gaussiana de media 0 y desviación estándar igual a 1. En el panel de abajo a la derecha aparece el módulo de la Transformada de Fourier de la señal, en la que se muestra que la señal presenta energía en todas las bandas de frecuencia desde 0 hasta fs/2.



Referencias

The MathWorks Inc. (**2023**). MATLAB version 23.2.0.2485118 (2023b), Natick Massachussetts: The Mathworks Inc. https://www.mathworks.com

MathWorks®

MATLAB® Basic Functions Reference

MATLAB Environment

`clc`	Clear command window
`help fun`	Display in-line help for `fun`
`doc fun`	Open documentation for `fun`
`load("filename","vars")`	Load variables from `.mat` file
`uiimport("filename")`	Open interactive import tool
`save("filename","vars")`	Save variables to file
`clear item`	Remove items from workspace
`examplescript`	Run the script file named `examplescript`
`format style`	Set output display format
`ver`	Get list of installed toolboxes
`tic, toc`	Start and stop timer
`Ctrl+C`	Abort the current calculation

Operators and Special Characters

`+, -, *, /`	Matrix math operations
`.*, ./`	Array multiplication and division (element-wise operations)
`^, .^`	Matrix and array power
`\`	Left division or linear optimization
`.', '`	Normal and complex conjugate transpose
`==, ~=, <, >, <=, >=`	Relational operators
`&&, ||, ~, xor`	Logical operations (AND, OR, NOT, XOR)
`;`	Suppress output display
`...`	Connect lines (with break)
`% Description`	Comment
`'Hello'`	Definition of a character vector
`"This is a string"`	Definition of a string
`str1 + str2`	Append strings

Special Variables and Constants

`ans`	Most recent answer
`pi`	π=3.141592654...
`i, j, 1i, 1j`	Imaginary unit
`NaN, nan`	Not a number (i.e., division by zero)
`Inf, inf`	Infinity
`eps`	Floating-point relative accuracy

Defining and Changing Array Variables

`a = 5`	Define variable a with value 5

`A = [1 2 3; 4 5 6]`
`A = [1 2 3`
`4 5 6]`
	Define A as a 2x3 matrix
	"space" separates columns
	";" or new line separates rows

`[A,B]`	Concatenate arrays horizontally
`[A;B]`	Concatenate arrays vertically
`x(4) = 7`	Change 4th element of x to 7
`A(1,3) = 5`	Change A(1,3) to 5
`x(5:10)`	Get 5th to 10th elements of x
`x(1:2:end)`	Get every 2nd element of x (1st to last)
`x(x>6)`	List elements greater than 6
`x(x==10)=1`	Change elements using condition
`A(4,:)`	Get 4th row of A
`A(:,3)`	Get 3rd column of A
`A(6, 2:5)`	Get 2nd to 5th element in 6th row of A
`A(:,[1 7])=A(:,[7 1])`	Swap the 1st and 7th column
`a:b`	[a, a+1, a+2, ..., a+n] with a+n≤b
`a:ds:b`	Create regularly spaced vector with spacing ds
`linspace(a,b,n)`	Create vector of n equally spaced values
`logspace(a,b,n)`	Create vector of n logarithmically spaced values
`zeros(m,n)`	Create m x n matrix of zeros
`ones(m,n)`	Create m x n matrix of ones
`eye(n)`	Create a n x n identity matrix
`A=diag(x)`	Create diagonal matrix from vector
`x=diag(A)`	Get diagonal elements of matrix
`meshgrid(x,y)`	Create 2D and 3D grids
`rand(m,n), randi`	Create uniformly distributed random numbers or integers
`randn(m,n)`	Create normally distributed random numbers

Complex Numbers

`i, j, 1i, 1j`	Imaginary unit
`real(z)`	Real part of complex number
`imag(z)`	Imaginary part of complex number
`angle(z)`	Phase angle in radians
`conj(z)`	Element-wise complex conjugate
`isreal(z)`	Determine whether array is real

MathWorks®

Elementary Functions

`sin(x), asin`	Sine and inverse (argument in radians)
`sind(x), asind`	Sine and inverse (argument in degrees)
`sinh(x), asinh`	Hyperbolic sine and inverse (arg. in radians)
Analogous for the other trigonometric functions: `cos`, `tan`, `csc`, `sec`, and `cot`	
`abs(x)`	Absolute value of x, complex magnitude
`exp(x)`	Exponential of x
`sqrt(x), nthroot(x,n)`	Square root, real nth root of real numbers
`log(x)`	Natural logarithm of x
`log2(x), log10`	Logarithm with base 2 and 10, respectively
`factorial(n)`	Factorial of n
`sign(x)`	Sign of x
`mod(x,d)`	Remainder after division (modulo)
`ceil(x), fix, floor`	Round toward +inf, 0, -inf
`round(x)`	Round to nearest decimal or integer

Tables

`table(var1,...,varN)`	Create table from data in variables var1, ..., varN
`readtable("file")`	Create table from file
`array2table(A)`	Convert numeric array to table
`T.var`	Extract data from variable var
`T(rows,columns)`, `T(rows,["col1","coln"])`	Create a new table with specified rows and columns from T
`T.varname=data`	Assign data to (new) column in T
`T.Properties`	Access properties of T
`categorical(A)`	Create a categorical array
`summary(T), groupsummary`	Print summary of table
`join(T1, T2)`	Join tables with common variables

Tasks (Live Editor)

Live Editor tasks are apps that can be added to a live script to interactively perform a specific set of operations. Tasks represent a series of MATLAB commands. To see the commands that the task runs, show the generated code.

Common tasks available from the Live Editor tab on the desktop toolstrip:

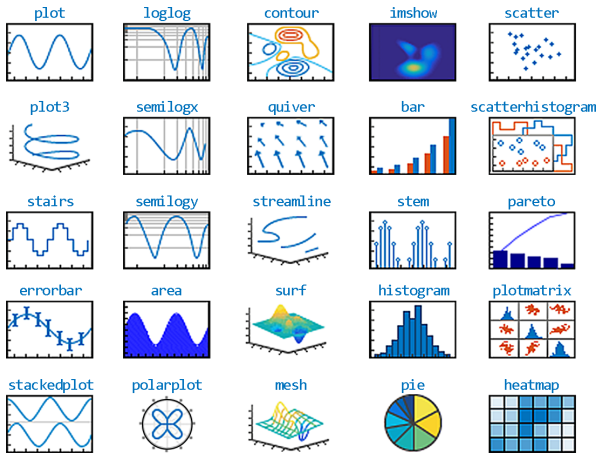
- Clean Missing Data
- Clean Outlier
- Find Change Points
- Find Local Extrema
- Remove Trends
- Smooth Data

Plotting

`plot(x,y,LineSpec)`
Line styles:
`-, --, :, -.`
Markers:
`+, o, *, ., x, s, d`
Colors:
`r, g, b, c, m, y, k, w`
	Plot y vs. x
	(`LineSpec` is optional)
	`LineSpec` is a combination of `linestyle`, `marker`, and `color` as a string.
	Example: `"-r"`
	= red solid line without markers

`title("Title")`	Add plot title
`legend("1st", "2nd")`	Add legend to axes
`x/y/zlabel("label")`	Add x/y/z axis label
`x/y/zticks(ticksvec)`	Get or set x/y/z axis ticks
`x/y/zticklabels(labels)`	Get or set x/y/z axis tick labels
`x/y/ztickangle(angle)`	Rotate x/y/z axis tick labels
`x/y/zlim`	Get or set x/y/z axis range
`axis(lim), axis style`	Set axis limits and style
`text(x,y,"txt")`	Add text
`grid on/off`	Show axis grid
`hold on/off`	Retain the current plot when adding new plots
`subplot(m,n,p)`, `tiledlayout(m,n)`	Create axes in tiled positions
`yyaxis left/right`	Create second y-axis
`figure`	Create figure window
`gcf, gca`	Get current figure, get current axis
`clf`	Clear current figure
`close all`	Close open figures

Common Plot Types



Plot Gallery: *mathworks.com/products/matlab/plot-gallery*

MathWorks®

Programming Methods

Functions

```
% Save your function in a function file or at the end
% of a script file. Function files must have the
% same name as the 1st function
function cavg = cumavg(x) %multiple args. possible
    cavg=cumsum(vec)./(1:length(vec));
end
```

Anonymous Functions

```
% defined via function handles
fun = @(x) cos(x.^2)./abs(3*x);
```

Control Structures

if, elseif Conditions

```
if n<10
    disp("n smaller 10")
elseif n<=20
    disp("n between 10 and 20")
else
    disp("n larger than 20")
```

Switch Case

```
n = input("Enter an integer: ");
switch n
    case -1
        disp("negative one")
    case {0,1,2,3} % check four cases together
        disp("integer between 0 and 3")
    otherwise
        disp("integer value outside interval [-1,3]")
end % control structures terminate with end
```

For-Loop

```
% loop a specific number of times, and keep
% track of each iteration with an incrementing
% index variable
for i = 1:3
    disp("cool");
end % control structures terminate with end
```

While-Loop

```
% loops as long as a condition remains true
n = 1;
nFactorial = 1;
while nFactorial < 1e100
    n = n + 1;
    nFactorial = nFactorial * n;
end % control structures terminate with end
```

Further programming/control commands

`break`	Terminate execution of for- or while-loop
`continue`	Pass control to the next iteration of a loop
`try, catch`	Execute statements and catch errors

Numerical Methods

`fzero(fun,x0)`	Root of nonlinear function
`fminsearch(fun,x0)`	Find minimum of function
`fminbnd(fun,x1,x2)`	Find minimum of fun in [x1, x2]
`fft(x), ifft(x)`	Fast Fourier transform and its inverse

Integration and Differentiation

`integral(f,a,b)`	Numerical integration (analogous functions for 2D and 3D)
`trapz(x,y)`	Trapezoidal numerical integration
`diff(X)`	Differences and approximate derivatives
`gradient(X)`	Numerical gradient
`curl(X,Y,Z,U,V,W)`	Curl and angular velocity
`divergence(X,..,W)`	Compute divergence of vector field
`ode45(ode,tspan,y0)`	Solve system of nonstiff ODEs
`ode15s(ode,tspan,y0)`	Solve system of stiff ODEs
`deval(sol,x)`	Evaluate solution of differential equation
`pdepe(m,pde,ic,... bc,xm,ts)`	Solve 1D partial differential equation
`pdeval(m,xmesh,... usol,xq)`	Interpolate numeric PDE solution

Interpolation and Polynomials

`interp1(x,v,xq)`	1D interpolation (analogous for 2D and 3D)
`pchip(x,v,xq)`	Piecewise cubic Hermite polynomial interpolation
`spline(x,v,xq)`	Cubic spline data interpolation
`ppval(pp,xq)`	Evaluate piecewise polynomial
`mkpp(breaks,coeffs)`	Make piecewise polynomial
`unmkpp(pp)`	Extract piecewise polynomial details
`poly(x)`	Polynomial with specified roots x
`polyeig(A0,A1,...,Ap)`	Eigenvalues for polynomial eigenvalue problem
`polyfit(x,y,d)`	Polynomial curve fitting
`residue(b,a)`	Partial fraction expansion/decomposition
`roots(p)`	Polynomial roots
`polyval(p,x)`	Evaluate poly p at points x
`polyint(p,k)`	Polynomial integration
`polyder(p)`	Polynomial differentiation

MathWorks®

Accelerating the pace of engineering and science

Matrices and Arrays

`length(A)`	Length of largest array dimension
`size(A)`	Array dimensions
`numel(A)`	Number of elements in array
`sort(A)`	Sort array elements
`sortrows(A)`	Sort rows of array or table
`flip(A)`	Flip order of elements in array
`squeeze(A)`	Remove dimensions of length 1
`reshape(A,sz)`	Reshape array
`repmat(A,n)`	Repeat copies of array
`any(A), all`	Check if any/all elements are nonzero
`nnz(A)`	Number of nonzero array elements
`find(A)`	Indices and values of nonzero elements

Linear Algebra

`rank(A)`	Rank of matrix
`trace(A)`	Sum of diagonal elements of matrix
`det(A)`	Determinant of matrix
`poly(A)`	Characteristic polynomial of matrix
`eig(A), eigs`	Eigenvalues and vectors of matrix (subset)
`inv(A), pinv`	Inverse and pseudo inverse of matrix
`norm(x)`	Norm of vector or matrix
`expm(A), logm`	Matrix exponential and logarithm
`cross(A,B)`	Cross product
`dot(A,B)`	Dot product
`kron(A,B)`	Kronecker tensor product
`null(A)`	Null space of matrix
`orth(A)`	Orthonormal basis for matrix range
`tril(A), triu`	Lower and upper triangular part of matrix
`linsolve(A,B)`	Solve linear system of the form AX=B
`lsqminnorm(A,B)`	Least-squares solution to linear equation
`qr(A), lu, chol`	Matrix decompositions
`svd(A)`	Singular value decomposition
`gsvd(A,B)`	Generalized SVD
`rref(A)`	Reduced row echelon form of matrix

Descriptive Statistics

`sum(A), prod`	Sum or product (along columns)
`max(A), min, bounds`	Largest and smallest element
`mean(A), median, mode`	Statistical operations
`std(A), var`	Standard deviation and variance
`movsum(A,n), movprod, movmax, movmin, movmean, movmedian, movstd, movvar`	Moving statistical functions n = length of moving window
`cumsum(A), cumprod, cummax, cummin`	Cumulative statistical functions
`smoothdata(A)`	Smooth noisy data
`histcounts(X)`	Calculate histogram bin counts
`corrcoef(A), cov`	Correlation coefficients, covariance
`xcorr(x,y), xcov`	Cross-correlation, cross-covariance
`normalize(A)`	Normalize data
`detrend(x)`	Remove polynomial trend
`isoutlier(A)`	Find outliers in data

Symbolic Math*

`sym x, syms x y z`	Declare symbolic variable
`eqn = y == 2*a + b`	Define a symbolic equation
`solve(eqns,vars)`	Solve symbolic expression for variable
`subs(expr,var,val)`	Substitute variable in expression
`expand(expr)`	Expand symbolic expression
`factor(expr)`	Factorize symbolic expression
`simplify(expr)`	Simplify symbolic expression
`assume(var,assumption)`	Make assumption for variable
`assumptions(z)`	Show assumptions for symbolic object
`fplot(expr), fcontour, fsurf, fmesh, fimplicit`	Plotting functions for symbolic expressions
`diff(expr,var,n)`	Differentiate symbolic expression
`dsolve(deqn,cond)`	Solve differential equation symbolically
`int(expr,var,[a, b])`	Integrate symbolic expression
`taylor(fun,var,z0)`	Taylor expansion of function

*requires Symbolic Math Toolbox

mathworks.com/help/matlab