💻 BREVE INTRODUCCIÓN A MATLAB 💻

PABLO CASTAÑEDA, JIMENA RODRÍGUEZ LEBRIJA
Y RODRIGO ZEPEDA TELLO

Cómo prender esta cosa

¡Oh gran MATLAB, ayúdame!
El propósito de estas notas (📋) es familiarizarte con MATLAB para que (1) no te pierdas tanto y (2) cuando te pierdas sepas usar el `help`. La ayuda que MATLAB ofrece es muy útil (aunque tiene sus peculiaridades). La gente de MATLAB hizo una **documentación** apropiada que incluso contiene ejemplos (aunque no creas, esto es maravilloso en programas de computadora; la mayoría ni te dicen cómo iniciar).

Enciende tu computadora (o la que estés usando, excepto si es robada, en esa caso devuélvela). Abre MATLAB. Hay una barra de herramientas y algunas "ventanas" dentro de la principal. Sus nombres son bastante aburridos: `Current Folder`, `Details`, `Command Window`, `Workspace` y `Command History`. En estas ventanas (y otras que abrirás) es donde ocurre la acción. (Por cierto, en adelante supondremos que tienes una computadora enfrente y estás corriendo MATLAB o tienes una memoria impresionante y recuerdas a la perfección el programa.)

Identifica la `Command Window` donde (si tu MATLAB no es pirata) verás un mensaje del más allá 👻:

```
      This is a Classroom License for instructional use only.
      Research and commercial use is prohibited.
>>
```

La última línea es **la importante**, donde aparece el *prompt* designado por el símbolo `>>` (cualquier parecido al *emoji* 😒 es pura coincidencia). Es aquí donde introducirás los comandos y operaciones a ser realizadas.

En la línea del *prompt* puedes hacer muchas cosas, por ejemplo enojar a MATLAB escribiendo tonterías:

```
>> tonterias
Undefined function or variable 'tonterias'. 😡
```

Notas de Aula (Versión 2.0.2): 22 de febrero de 2017.

También puedes sumar `2 + 3` (por si no te acuerdas cuánto te da):

```
>> 2 + 3

ans =

     5
```

o bien jugar buscaminas:

```
>> xpbombs
```

hacer un rompecabezas deslizante:

```
>> fifteen
```

También puedes pedir ayuda. Para ello hay varias maneras:

1. Presionando el botón `Help` en la parte superior. Esto abre una ventana de búsqueda: `Search MATLAB Documentation`.
2. Usando el *prompt* para ver la documentación:

   ```
   >> doc
   ```

3. Escribiendo desesperadamente `help` al *prompt* esperando te comprenda:

   ```
   >> help
   ```

Estas ayudas son generales; sin embargo, podemos pedir información específica sobre funciones de MATLAB. Por ejemplo:

```
>> doc lorenz
```

abre la documentación del comando `lorenz`. Lo mismo ocurre con `help`:

```
>> help lorenz
lorenz Plot the orbit around the Lorenz chaotic attractor.
    This demo animates the integration of  the
    three coupled nonlinear differential equations
```

La función `lookfor` te ayuda a buscar todas las funciones que tienen `lorenz` en su nombre:

```
>> lookfor lorenz
lorenz    - Plot the orbit around the Lorenz chaotic attractor.
```

Por supuesto, estos comandos sólo funcionan para hallar funciones que existen en MATLAB. Si no existe lo que buscas, MATLAB te lo dirá:

```
>> lookfor love
love not found.
```

Observa que `doc` ofrece una documentación más completa que `help`. La ventaja de `help` está en que es sucinta y que, cuando alguien que no es MATLAB crea una función, automáticamente se crea un `help`.

Una vez que has visto la ayuda prueba la función:

```
>> lorenz
```

También prueba el `help` de `demo` nada más para pasar el rato:

```
>> help demo
```

Puedes consultar otros comandos básicos como `what` 😟, `clock` 🕓, `path` 🔀 o `date` 🌇. Algunos comandos de propósito general los puedes encontrar en la Tabla 1 (que aquí se llama CUADRO).

CUADRO 1. Algunos comandos de MATLAB.

`cd`	Cambia el directorio (la carpeta donde MATLAB trabaja),
`pwd`	Muestra el directorio elegido,
`clc`	Limpia la `Command Window`,
`clear`	Elimina las variables que has creado (mira el `Workspace`),
`close`	Cierra las ventanas del entorno gráfico,
`who`	Muestra las variables utilizadas.

Prueba utilizar las flechas del teclado. Enfrente del *prompt* verás que la última frase escrita reaparece cuando usas ⬆️. Esto te permite recorrer los comandos del remoto pasado (éstos además los ves en la `Command History`). Si además le dices a MATLAB que el comando que buscas comenzaba con `h`, por ejemplo, sólo buscará los que empezaban así.

Finalmente, basta decir que si seleccionas cualquier comando del *prompt* y con el *mouse* arrastras la instrucción a la barra de herramientas, entonces podrás tener un atajo para que este comando sea ejecutado. Inténtalo, pues una nueva ventana aparecerá y te guiará en la construcción de un nuevo botón.

¿Y si MATLAB me odia 😠 ?

Aunque la ayuda de MATLAB es buena, no lo es todo. Hay ocasiones en las que por más que te esfuerzas parece que MATLAB conspira en tu contra. De hecho, puedes preguntarle por qué te trata así:

```
>> why
It's your karma.
```

Si MATLAB te odia, ¡no te apures, hay más formas de pedir ayuda! El sitio StackOverflow[1] que cuenta con su versión en español es un buen lugar para realizar preguntas sobre programación en general. En particular, se pueden hacer preguntas de MATLAB de manera gratuita. ¡Fíjate bien, porque es probable que tu problema alguien más ya lo haya resuelto!

En el mismo espíritu que StackOverflow, los sitios Computational Science Stack Exchange y Mathematics Stack Exchange están diseñados para que preguntes sobre métodos numéricos (el primero) y matemáticas en general (el segundo). Cuando te atores en una tarea o algoritmo estos son buenos recursos.

La gente de MATLAB cuenta con MATLAB central un lugar donde puedes realizar preguntas de MATLAB o bien encontrar programas ya pre-elaborados por alguien más y que en la bondad de su corazón ha decidido compartir con todos los usuarios de MATLAB (es un buen sitio cuando son las 12 y no has terminado la tarea). En el mismo espíritu, Github es un sitio de intercambio de archivos (donde todo lo programable en MATLAB está programado). Páginas similares son Gitlab y Bitbucket. Estas tres últimas te permiten trabajar código de manera colaborativa por lo que son una gran opción para trabajar en equipo.

Hemos terminado esta engorrosa introducción. ¡Vamos a MATLAB!

```
>> load handel
>> sound(y, Fs)
```

El ataque de los puntos flotantes

MATLAB no sabe sumar.
Desde el *prompt* de la `Command Window` puedes llamar variables ya definidas o crear tus propias variables; `pi` y `eps` son ejemplos de variables ya definidas. La primera es un truncamiento de nuestro número trascendental favorito π mientras que la segunda es el épsilon de la máquina, llamado vulgarmente ϵ_M. En una computadora no es como en Cálculo donde puedes inventarte cualquier ϵ. En este caso, el ϵ_M ya está dado. El épsilon de máquina es la máxima precisión relativa del punto flotante; es decir, éste ϵ_M representa la diferencia máxima que puede observar la computadora para concluir que dos números son diferentes. La aritmética de punto flotante en MATLAB es de precisión doble lo que quiere decir que los cálculos numéricos son hechos con

[1] Si tienes una versión impresa de estas notas, no notarás que hay hipervínculos debajo de estas y otras palabras azules. Algunas son importantes, como el de esta nota por ejemplo, el vínculo es `http://stackoverflow.com/`, otras estarán al final del texto. (La versión en español es `http://es.stackoverflow.com/`.)

aproximadamente 16 dígitos decimales de precisión. ¡Si estás calculando algo con menos dígitos tendrás que hacer un truco como cambiar de escala! (De hecho hay concursos sobre esto.)

Para explicar mejor este concepto, vayamos a MATLAB. Aquí para concluir igualdad entre dos cosas usamos «==». Si MATLAB contesta con un `1` quiere decir que las cosas son iguales y `0` representa desigualdad. Podemos preguntar, por ejemplo, si `1` es igual a `1`:

```
>> 1 == 1

ans =

     1
```

Mientras que `1 + 2` no es igual a `1`:

```
>> 1 + 2 == 1

ans =

     0
```

Cuando usamos `eps` la computadora distingue que hay una diferencia:

```
>> 1 + eps == 1

ans =

     0
```

Pero si usamos un número más pequeño que `eps` (por ejemplo `eps/2`) ¡piensa que sigue siendo igual a 1!

```
>> 1 + eps/2 == 1

ans =

     1
```

Cuando hacemos un cálculo en MATLAB, los resultados automáticamente son desplegados en formato corto (`format short`), esto quiere decir que se muestran los primeros cuatro dígitos después del punto decimal. Con `format`

`long` se puede cambiar el formato para que muestre 16 dígitos, 15 de ellos después del punto decimal.

```
>> eps

ans =

   2.2204e-16

>> format long
>> eps

ans =

   2.220446049250313e-16
```

A primera vista esto del punto flotante puede no parecer tan aterrador como en verdad es. Por ejemplo, a nadie le causa pánico que en lugar de cero MATLAB regrese $-2.7756 \cdot 10^{-17}$ al hacer la resta:

```
>> 1 - 0.9 - 0.1

ans =

    -2.775557561562891e-17 😱
```

Sin embargo, quizá a algunos los saque de sus casillas el observar que para MATLAB no es lo mismo `0.3` que `0.1 + 0.2`.

```
>> 0.2 + 0.1 == 0.3

ans =

     0 😱
```

Prueba calcular tanto `0.1 + 0.2 - 0.3` como `0.2 - 0.3 + 0.1`.

En la vida real el punto flotante ha sido el culpable de explosiones de cohetes 🚀💥, caídas en la bolsa de Vancouver 📉💸 y hundimiento de plataformas petroleras 🏭🌊. ¡No dejes que su tamaño pequeño te engañe! Más adelante (cuando lleguemos a los `for`) veremos un ejemplo donde podemos volver arbitrariamente grande la diferencia entre lo que dice MATLAB y el verdadero valor 😱😱😱.

Subrayamos que este problema no es exclusivo de MATLAB sino de la aritmética de punto flotante y gracias a eso también tenemos trabajo los matemáticos. Empero, hay programas que le sacan la vuelta e intentan, lo más que pueden, no hacer cálculos numéricos sino usar expresiones matemáticas. El más famoso es quizá Wolfram Alpha que permite hacer estimaciones por medio de matemática simbólica, evitando (lo más posible) la aritmética de punto flotante. La versión para escritorio de la misma compañía es Mathematica (su competencia de paga es Maple que permite una conexión con MATLAB) aunque si prefieres no gastar puedes usar Sage, Yacas, Sympy o Maxima. ¡Son geniales para esa integral indefinida que no encuentras cómo resolver!

Si el Oxxo usara MATLAB así redondearía.
Como ya vimos, el hecho de que la cantidad de decimales de un número sea finito produce errores. A veces una solución consiste en redondear a un entero próximo (cuando sabemos, por ejemplo, que la solución debería ser entera). Puedes hallar tres funciones útiles para este propósito en la Tabla 2.

CUADRO 2. Funciones útiles para el redondeo.

`round(x)`	redondea `x` al entero más próximo,
`floor(x)`	devuelve el mayor entero posible menor a `x`,
`ceil(x)`	devuelve el menor entero posible mayor a `x`.

Bautismo de variables.
Así como a los bebés, a las variables de MATLAB las puedes nombrar con cualquier combinación de letras y números. MATLAB es sensible al uso de mayúsculas lo que implica que una variable llamada «`y`» es diferente a una llamada «`Y`». El signo de igual es usado para asignar valores a las variables. Puedes decirle a MATLAB que no te escriba el resultado explícitamente, este *eco* puede ser suprimido al agregar «`;`» al final del comando.

```
>> dorotea = 10;
>> Dorotea = dorotea^2

Dorotea =

   100

>> dorotea

dorotea =
```

```
    10
```

Una variable muy útil que MATLAB crea por defecto es `ans`. Ésta muestra el último resultado no asignado a una variable.

```
>> 1 + 2 + 3

ans =

     6

>> ans

ans =

     6
```

Aritmética como de costumbre.
En expresiones con más de una operación los paréntesis tienen prioridad sobre las potencias. Éstas últimas tienen prioridad sobre las multiplicaciones y las divisiones, las cuales se realizan antes de las sumas y las restas; como en la aritmética usual.

```
>> 4*(2 - 4)^2 + 10

ans =

    26

>> 1/2 + 3

ans =

   3.5000

>> 1/(2 + 3)

ans =

   0.2000
```

JUSTO CUANDO PENSABAS QUE LINEAL HABÍA ACABADO, VUELVEN LAS MATRICES

En el álgebra lineal (y la vida real), una matriz es tan sólo un rectángulo con números.[2] En computación 💻 a estas cosas (y otras más complejas como los tensores) se conocen como arreglos (`array`). En esta sección sólo crearemos matrices al ser éstas los arreglos con los que más familiaridad tenemos.

Para crear una matriz es necesario:

1. Usar corchetes para colocar, dentro de ellos, a los elementos de la matriz.
2. Separar los elementos de cada renglón por comas (o espacios en blanco; en general intenta siempre usar comas porque los programas que no son MATLAB funcionan con comas).
3. Usar punto y coma para separar renglones.

Por ejemplo la matriz:

$$A = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 5 \end{bmatrix}$$

se escribe en MATLAB como sigue:

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

A =

     2     4     6
     1     3     5
```

Así como en álgebra lineal denotamos $A_{i,j}$ la entrada de la matriz A en el renglón i y la columna j, en MATLAB usamos `A(i,j)`. Por ejemplo, para obtener la entrada $A_{2,3}$ en MATLAB basta con correr el siguiente código:

```
>> A(2, 3)

   ans =

        5
```

Para cambiar dicha entrada basta con usar un igual y poner el nuevo valor:

```
>> A(2, 3) = 100;
>> A
```

[2] Este enunciado resume el primer curso de lineal: de nada.

```
   A =

        2     4     6
        1     3   100
```

También es posible preguntar por columnas específicas de la matriz (o renglones específicos) usando «:»:

```
>> A(:, 2) % Selecciona la columna 2

ans =

     4
     3

>> A(1, :) % Selecciona el renglón 1

ans =

     2     4     6
```

Un truco útil consiste en usar un vector para seleccionar columnas o renglones específicos; por ejemplo:

```
>> cols  = [2, 3]; % Seleccionaré segunda y tercer columna
>> A(:, cols)

ans =

     4     6
     3   100

>> rengs = [1, 2]; % Seleccionaré renglón 1 y 2
>> A(rengs, :)

ans =

     2     4     6
     1     3   100
```

Podemos preguntar por el tamaño de una matriz usando `size`. Este comando regresa un vector con dos valores: el primero es el número de filas y el segundo el de columnas de una matriz:

```
>> dimensiones = size(A)

dimensiones =

     2     3

>> mayor_dim = length(A)

mayor_dim =

     3
```

Observación: En MATLAB (y en la vida real 😟) los vectores no son más que matrices en $\mathbb{R}^{1\times m}$ (si son renglón) ó $\mathbb{R}^{n\times 1}$ (si son columna). De esta forma:

```
>> vector_asombroso = [1, -1, 1, 7, 2];
>> size(vector_asombroso)

ans =

     1     5
```

Finalmente, notamos que para transponer la matriz A basta con usar el apóstrofe «`'`»:

```
>> A'

ans =

     2     1
     4     3
     6   100
```

Creemos una matriz cuadrada a partir de `A` y `A'`; para ello basta con multiplicarlas:

```
>> B = A*A';
```

Podemos calcular su determinante:

```
>> det(B)

ans =

   1.8356e+05
```

o su traza:

```
>> trace(B)

ans =

       10066
```

Hallar los valores propios (eigenvalores) es sencillo:

```
>> eig(B)

ans =

   1.0e+04 *

    0.0018
    1.0048
```

E incluso podemos estimar la inversa:

```
>> inversa = inv(B)

inversa =

    0.0545   -0.0033
   -0.0033    0.0003
```

La función ojo y otras matrices preconstruidas.
La Tabla 3 resume los comandos más importantes para construir matrices y manipularlas. Dicha tabla supone que `A` es una matriz de tamaño $n \times m$ y las variables `i, n, m` son números. Se considera además que `v` es vector renglón y `bool` una variable *booleana* (¿cóóómo? 😟), es decir toma el valor 0 si el enunciado a analizar es falso y 1 cuando es verdadero (así como al inicio de este documento).

CUADRO 3. Todos los comandos de MATLAB sobre matrices que querías saber y nunca te atreviste a preguntar resumidos en (obviamente) una matriz.

Comando	**Descripción**
`A = eye(n)`	Matriz identidad de orden n.
`A = ones(n, m)`	Matriz de $n \times m$ donde todas las entradas son 1.
`A = zeros(n, m)`	Matriz nula de tamaño $n \times m$.
`A = rand(n, m)`	Matriz de $n \times m$ donde cada entrada es un valor pseudoaleatorio (uniforme) entre 0 y 1.
`[n, m] = size(A)`	Dimensiones de la matriz `A`. El vector salida guarda en su primera entrada el número de renglones y en la segunda el de columnas.
`l = length(A)`	Máxima dimensión de la matriz `A`. Es equivalente a `max(size(A))`.
`bool = any(A)`	Analiza si existe al menos una entrada de `A` distinta de cero.
`bool = all(A)`	Determina si todas las entradas de `A` son no cero.
`v = max(A)`	Estima un vector renglón donde cada entrada es el máximo valor de cada columna de `A`. *Observación*: si `A` es un vector, entonces `max(A)` devuelve el máximo elemento.
`m = max(max(A))`	Calcula el elemento de valor máximo en la matriz.
`[m, i] = max(A)`	Devuelve el vector `m` de máximos elementos por columna de `A` y los índices `i` de los renglones donde se encuentran los susodichos.

La Tabla 3 amerita varios comentarios:

1. La función `min` es análoga a la función `max`.

2. Para especificar matrices cuadradas de tamaño n es lo mismo especificar n como número de renglones y columnas que sólo poner n, es decir, `ones(n, n) = ones(n)`; `zeros(n, n) = zeros(n)`; `rand(n, n) = rand(n)`.
3. Si `v` es un vector, entonces `diag(v)` es una matriz diagonal cuadrada con los valores de `v` en la diagonal.
4. Si `A` es una matriz cuadrada de orden n, entonces `diag(A)` es un vector renglón con los elementos de la diagonal de `A`.
5. Es posible componer las funciones previas, por ejemplo `zeros(size(A))` produce una matriz nula del mismo tamaño que la matriz original `A`.

Matlab como calculadora matricial.
La suma (`+`), la resta (`-`), el producto (`*`) y la "división" (`/`) también están definidas para matrices y vectores.

Recuerda que si A, B son matrices en $\mathbb{R}^{n\times m}$ la entrada i, j de la suma se define como:

$$(A + B)_{i,j} = A_{i,j} + B_{i,j}.$$

Para sumar en MATLAB basta con hacer:

```
>> A = [1 3; 3 5];
>> B = [1 2; 2 1];
>> A + B

ans =

     2     5
     5     6
```

Por otro lado, la entrada i, j del producto entre $A \in \mathbb{R}^{m\times p}$ y $B \in \mathbb{R}^{p\times n}$ es:

$$(AB)_{i,j} = \sum_{k=1}^{p} A_{i,k}B_{k,j}.$$

En MATLAB esto se calcula como:

```
>> A*B

ans =

     7     5
    13    11
```

Con la ayuda de un simple punto «`.`», MATLAB transforma cualquier operación en una operación que se realiza elemento a elemento. De este modo,

por ejemplo, cuando `A` y `B` tienen las mismas dimensiones, entonces `A .* B` calcula el producto entrada a entrada (producto de Hadamard) devolviendo una matriz del mismo tamaño. Es decir:

$$(A. * B)_{i,j} = A_{i,j}B_{i,j}.$$

En MATLAB esto es:

```
>>  A.*B

ans =

     1     6
     6     5
```

Esta opción del punto permite hacer operaciones elemento a elemento. Si agregas un punto enfrente de las operaciones « ^ » o «/», entonces éstas se realizarán elemento a elemento. Vamos a verlo con más detalle:

```
>> A.^2

ans =

    1     9
    9    25

>> A./3

ans =

    0.3333    1.0000
    1.0000    1.6667
```

Felizmente (para ti, no para MATLAB que se enoja cuando alguien mete la pata), MATLAB tiene modos de ayudarte cuando has cometido un error:

Ejemplo:

```
>> A = [1 2 3; 4 5 6];
>> B = [1 2 4; 1 3 9];
>> A .* B

ans =
```

```
     1     4    12
     4    15    54

>> A * B
Error using  *
Inner matrix dimensions must agree.
```

Como verás, MATLAB indica en el mensaje de error que el producto `*` está mal empleado. Esto indica que el error es por causa de la multiplicación de matrices. La siguiente línea enfatiza el error indicando que son las dimensiones internas las que deben ser iguales. Esto tiene sentido si recuerdas que "$\mathbb{R}^{n\times p} * \mathbb{R}^{p\times m} = \mathbb{R}^{n\times m}$". Observa que en el ejemplo hay un p distinto (dimensión interna) para cada una de las matrices.

Sección donde hablamos de la pseudoinversa de Moore-Penrose sólo para presumir (y también resolvemos ecuaciones lineales).
En la vida real, la división de matrices no existe; sin embargo, MATLAB proporciona las divisiones «`/`» y «`\`» como operadores de división de matrices muy particulares. Estas operaciones son muy útiles para tratar de resolver los clásicos sistemas de álgebra lineal:

$$Ax = b. \tag{1}$$

A fin de resolver (1), consideremos `A`, una matriz con los coeficientes del sistema de ecuaciones lineales asociado, `x` la solución (vector columna a estimar) y `b` el vector columna de los términos independientes. En MATLAB, la solución puede ser calculada por "división de matrices", donde `x = A\b` dice que hay que multiplicar por la inversa (si existe) de `A` a la izquierda de `b`. En principio, esto es análogo a resolver de forma analítica el sistema lineal como $x = A^{-1}b$ (`x = inv(A) * b`); numéricamente tiene sutilezas que en estas no percibiremos.

El sistema `Ax = b` puede resolverse como:

```
>> A = [1 2; 0 1];
>> b = [1; 3];
>> x1 = A \ b

x1 =

   -5
    3
```

o bien:

```
>> x2 = inv(A) * b

x2 =

   -5
    3
```

En ambos casos:

```
>> A * x1 - b

ans =

    0
    0

>> A * x2 - b

ans =

    0
    0
```

La forma analítica (Gauss-Jordan o lo que te guste hacer) requiere de más operaciones. MATLAB busca una forma de resolver el sistema lineal de manera rápida y eficiente; sin embargo es importante hacer notar que no tienes control de lo que está haciendo. A fin de ejemplificar esto, consideremos ahora un sistema `A * x = b` pero que no tenga matriz inversa. ¡Verás como MATLAB encuentra un valor de todas formas!

```
>> A = magic(8); A = A(:, 1:6)

A =

    64     2     3    61    60     6
     9    55    54    12    13    51
    17    47    46    20    21    43
    40    26    27    37    36    30
    32    34    35    29    28    38
    41    23    22    44    45    19
    49    15    14    52    53    11
     8    58    59     5     4    62
```

Observa que el rango de `A` es tan sólo 3:

```
>> rank(A)

ans =

     3
```

Por otro lado, sea el vector b puros 260:

```
>> b = 260*ones(8, 1)

b =

   260
   260
   260
   260
   260
   260
   260
   260
```

MATLAB puede hallar una solución a pesar de que A no sea invertible:

```
>> x1 = A \ b
Warning: Rank deficient, rank = 3, tol =  6.657192e-14.

x1 =

    4.0000
    5.0000
         0
         0
         0
   -1.0000
```

Este `x1` tiene en particular que es la solución al problema de optimización:

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2. \tag{2}$$

Es decir, el vector `x1` obtenido minimiza la norma euclideana de $Ax - b$ y, si el mínimo de (2) no es 0, entonces `x1` no es solución al sistema sino tan sólo

una buena aproximación. Esto de "buena aproximación" no es siempre cierto, por ejemplo considera el caso:

```
>> A = [1 0; 0 0];
>> b = [1; 1];
>> x = A \ b
Warning: Matrix is singular to working precision.

x =

     1
   Inf
```

¡Infinito es una pésima aproximación! (De hecho una mejor aproximación la podemos obtener con la pseudoinversa de Moore-Penrose 😟 de la que no hablaremos más porque la teoría necesaria se ve hasta Lineal Avanzada.)

```
>> x = pinv(A)*b

x =

     1
     0
```

Moraleja: si tu `A` es invertible entonces `x = A\b` es una solución. Si hay múltiples soluciones, `x` es una de ellas; si no hay solución, quién sabe quién es `x`... la "mejor" aproximacion.

Vectores rápidos. En MATLAB los dos puntos «`:`» producen un vector de elementos con un inicio, un tamaño de paso y un final. Algo así: `inicio:tamaño de paso:final`. Hay que notar que en caso de colocar sólo dos elementos, el incremento estándar de nuestro entero favorito, 1, será tomado.

```
>> 1:3

ans =

     1     2     3

>> x = 1:0.5:3

x =
```

```
    1.0000    1.5000    2.0000    2.5000    3.0000

>> y = sin(x) .* cos(x);
>> [x; y]

ans =

    1.0000    1.5000    2.0000    2.5000    3.0000
    0.4546    0.0706   -0.3784   -0.4795   -0.1397
```

Consulta el `help` del comando `linspace`, pues puede resultarte útil.

Estructuras para las personas obsesionadas con el control

A largo de este documento, hemos mencionado algunas operaciones y comandos importantes (quién olvidará el comando `xpbombs` para jugar en clase). Algunas han quedado implícitas y tal vez (como a, *spoiler-alert*, Nemo) ya las encontraste. Aquí profundizaremos en ellas por si escaparon a una primera lectura (si escaparon a tu cuarta lectura ya necesitas ir a dormir 😴).

Operadores lógicos y otras ayudas sobre relaciones.
En programación estos operadores son cruciales. Los operadores lógicos probablemente los conoces de teoría de conjuntos (o de lógica básica). Veamos cómo usarlos en Matlab.

Cuadro 4. Cómo Matlab ve las relaciones.

Operador relacional	**Uso**	**Como Matlab los ve**
`<`	`a < b`	`a` es poco comparado con `b`.
`>`	`a > b`	`a` es realmente mayorcito como para `b`, ¿no crees?
`<=`	`a <= b`	No sé si `b` es traga años o es igual que `a`.
`>=`	`a >= b`	Es posible que `a` sea mayor que `b`, pero creo que va a funcionar.
`==`	`a == b`	Velos, si `a` y `b` son igualitos.
`~=`	`a ~= b`	¡Ufff! pero es que `a` y `b` son bien, bien diferentes...

Hay dos tipos de operadores: lógicos y relacionales. Aunque esta es la sección de operadores lógicos hablaremos (primero) de los relacionales, ¿por qué no? Éstos son símbolos que se usan para comparar dos valores. De este modo, si

CUADRO 5. Operadores lógicos de MATLAB

Operador lógico	Uso	Descripción
`&`	`a & b`	Si `a` "y" `b` son verdaderos la afirmación es verdadera.
`|`	`a | b`	Si `a` "o" `b` son verdaderos la afirmación es verdadera.
∼	`∼a`	Esta es la "negación" de `a`; cambia falso por verdadero (y viceversa).

el resultado de la comparación es correcto la expresión se considera verdadera (recuerda MATLAB regresa un 1); en caso contrario es falsa (0 si eres MATLAB). Los operadores relacionales son `<`, `<=`, `>=`, `>`, `==`, `∼=` (son bastante intuitivos, ¿cierto?). La Tabla 4 muestra cómo MATLAB ve las relaciones en función de sus operadores 😉.

Aguas: El símbolo «=» se usa para asignar valor a una variable, pero «==» se usa como operador relacional que compara dos objetos. ¡No te confundas!

Podemos usar los operadores relacionales en MATLAB:

```
>> 10^2 >= 9.9

ans =

     1
```

Ahora sí hablemos de *operadores lógicos*. Éstos tienen un 1 en caso de tratarse de afirmaciones verdaderas y 0 si son falsas. (MATLAB sigue las tablas de verdad usuales.) Los operadores son `&` ("y" que en conjuntos corresponde a intersección), `|` ("o" que en conjuntos corresponde a unión) y ∼ ("no" que en conjuntos corresponde al complemento)[3]. Puedes hallar una descripción de estos operadores en la Tabla 5. Veamos algunos ejemplos de uso:

```
>> (1 < 2) & (3 == 4)

ans =

     0
```

[3]Colecciones de conjuntos que cumplen que si A y B están en la colección entonces $A \cap B$, $A \cup B$ y A^c están en la colección forman un álgebra (booleana). Estas cosas aparentemente inocentes son la base de gran parte de ciencias de la computación, probabilidad y teoría de la medida 😱.

Observa cómo cambia con el "o":

```
>> (1 < 2) | (3 == 4)

ans =

     1
```

También podemos usar un "*not*" para negar lo verdadero y afirmar lo falso:

```
>> (1 < 2)

ans =

     1

>> ∼(1 < 2)

ans =

     0
```

En lo anterior colocamos paréntesis extras para mejor lectura. Como dato curioso, nota que las relaciones pueden usarse como números 😟:

```
>> (1 < 2) + 9

ans =

    10
```

donde solamente `0` representa falso.

Condicionales (el hubiera sí existe).
La utilidad de los operadores lógicos radica en su poder dentro de ciclos (bucles o *loops*). Los más relevantes son el `for` y `while`. Pronto los veremos. Primero es necesario hablar de condicionales: `if` y sus variantes `if-else` o `if-elseif`.

La idea de los condicionales es ejecutar instrucciones sólo bajo ciertas condiciones; por ejemplo:

```
>> x = 3;
>> if (x == 2)
   x = 1; % Sólo se ejecuta si x era 2 al momento del if
```

```
   end
>> x

x =

     3
```

En este ejemplo, la instrucción `x = 1` se ignora pues `x` $\neq 2$; pero en el siguiente sí se realiza:

```
>> x = 2;
>> if (x == 2)
   x = 1; % Sólo se ejecuta si x era 2 al momento del if
   end
>> x

x =

     1
```

La Tabla 6 muestra el uso de los condicionales así como una breve descripción de los mismos. Las estructuras no se resumen a estas tres. Puedes hacer una combinación de ellas y tener algo muy complicado. También puedes usar un `switch`; éstos son geniales para combinaciones complejas (recuerda: `help switch` en la `Command Window`).

Ciclos (eso que en humanos llamamos obsesión).
El `while` es una manera iterativa de usar un condicional `if`. De hecho, tiene una estructura semejante: `while (cond)`, `%Programa` y `end`, mira la Tabla 6 si quieres, pero la verdad es que no tiene mucho que ver. Lo padre del `while` es que permanece ejecutando una instrucción una y otra vez hasta que la condición `cond` se vuelva falsa. ¡Ten mucho cuidado porque si te descuidas puedes trabar tu computadora! (Si eres una persona de Java quizá conozcas este comando como *do-while*, sólo que en MATLAB puede que nunca se ejecute). La forma de escribirlo en MATLAB es la siguiente:

```
while (cond)
   % Programa 1 que se ejecuta una y otra vez
   % mientras cond sea verdad
end
```

El comando `while` puede ser considerado como un condicional, pero a la vez también es un ciclo. El ciclo puro en MATLAB está dado por el comando `for`.

CUADRO 6. Una tabla más, o de cómo resumir la elección de qué pizza comprar.

Estructura	Descripción	Código

`if`	Se corren los comandos contenidos en el «`%Programa`» si `cond1` es una condición verdadera; en caso contrario, se salta todo lo que está entre el `if` y el `end` para luego proseguir.	`if (cond1)`
		`  %Programa`
		`end`

`if-else`	Si `cond1` es verdadera, entonces se corren los comandos contenidos en el «`%Programa 1`»; si es falsa se corren los comandos de «`%Programa 2`». Después de cualquier caso, va hasta el `end` y prosigue.	`if (cond1)`
		`  %Programa 1`
		`else`
		`  %Programa 2`
		`end`

`if-elseif`	Si la `cond1` es verdadera, se ejecuta «`%Programa 1`» y se va al `end`. Si la `cond1` es falsa, checa si `cond2` es verdadera. En caso de `cond2` ser verdad entonces ejecuta «`%Programa 2`»; si no es verdad, salta hasta el `end`. Es decir, si ambas condiciones son falsas, no se ejecuta ningún programa; si ambas son verdaderas, sólo el «`%Programa 1`».	`if (cond1)`
		`  %Programa 1`
		`elseif (cond2)`
		`  %Programa 2`
		`end`

Debemos recordar la manera de hacer vectores usando «`:`», pues la manera más sencilla de usarlo es la siguiente:

```
for i = inicio:incremento:final
    % Programa, donde podemos usar i como variable
end
```

Aquí para cada índice `i` se repetirá el `% Programa`. Esta `i` es en realidad una variable que toma todos los valores de la lista (y puedes usarla para hacer monerías).

Los ciclos se pueden anidar (meter unos dentro de otros como *matrioskas*). De hecho, son bastante útiles para construir matrices con estructuras bien definidas. Veamos un ejemplo donde creamos matrices de Hilbert $H = (h_{ij}) \in \mathbb{R}^{n\times n}$ donde cada elemento satisface $h_{ij} = (i + j - 1)^{-1}$.

```
n = 3;
for i = 1:n
   for j = 1:n
       H(i,j) = 1/(i + j -1);
   end
end
```

Por último vale la pena mencionar el `break`. Si estás en un ciclo a veces es necesario poner un alto para salir de él: el `break` hace eso. Por ejemplo en el siguiente código, el `break` detiene el `for` y evita que siga corriendo hasta `1000`:

```
for i = 1:1000
    sprintf('%f', i) % Muestra el valor de i por cada vuelta
    if (i > 10)
        break
    end
end
```

Nota: Otra manera de parar MATLAB es con `Ctrl+C`, pero en este caso destruyes la rutina y no se guarda nada como puede ser el caso del `break`.

RUTINAS Y LA ESCRITURA DE PROGRAMAS

Hasta ahora hemos usado MATLAB como una calculadora extremadamente estorbosa. Pero recuerda, las buenas calculadoras te permiten hacer pequeñas rutinas, guiones o funciones propias que pueden ser usadas varias veces modificando los datos de entrada. Esta es también una cualidad de MATLAB (o sea, MATLAB es una calculadora estorbosa, básicamente). En particular hay dos modos directos de actuar, uno es a través de una lista de los comandos a realizar, ésta se llama guión (`script`); la otra consiste en una serie de rutinas que son llamadas con datos específicos y devuelven resultados. Estas segundas se conocen como funciones (`function`). Mientras que en los guiones es usual cambiar su interior, las funciones son prácticamente inmutables.

Los modos en los que cada persona programa son propios (e identificables). Sin embargo, para trabajos colaborativos en general se establecen guías de

estilo de tal forma que todos los miembros de un equipo programen de maneras similares. La manera en la que programes en MATLAB depende de ti (o de tu equipo o del profe 😏). En esta sección te explicamos lo que MATLAB tiene para ofrecer.

Guiones, los *m-files*.
Las instrucciones de un programa de MATLAB pueden ser guardadas para ser utilizadas posteriormente en archivos ".m", que son conocidos como `m-files`. Estos archivos se puede crear desde MATLAB al dar clic en el ✚ del lado izquierdo y seleccionar `script`. Tus acciones ocasionarán se abra una nueva ventana del `Editor` llamada `untitled`. Guárdala bajo el nombre `fibonacho` usando el 💾. En ella podrás escribir comandos que se ejecutarán de forma secuencial. Por ejemplo:

```
% Archivo que construye los primeros 25 elementos
% de la sucesión de Fibonacci.
fibonacci    = zeros(25,1); % Vector de Fibonacci
fibonacci(2) = 1;
for i = 3:length(fibonacci)
    fibonacci(i) = fibonacci(i-1) + fibonacci(i-2);
end

% Graficación
plot(1:25, fibonacci)
```

Nota los argumentos en galante verde olivo después del %. Éstos son comentarios y MATLAB los ignora. Los comentarios son útiles para entender (cuando tu ser del futuro abra el archivo) qué estabas intentando hacer cuando escribiste todo esto. Se recomienda fuertemente comentar tu código para generar la costumbre. Verás que con el tiempo y en programas grandes se vuelve muy difícil entender qué querías hacer inicialmente. También es importante comentar para que otros (o sea el profesor) puedan entender qué pretendes hacer en cada paso.

Para correr todos los comandos de golpe hay varias formas. La usual es presionando el botón verde ▶ en el `Editor`. Otra forma es llamando al guión desde la `Command Window` mediante el nombre del `m-file`:

```
>> fibonacho
```

Observa que si hay comentarios en tu código, entonces automáticamente MATLAB crea un `help` para tu archivo:

```
>> help fibonacho
  Archivo que construye los primeros 25 elementos
  de la sucesión de Fibonacci.
```

Los `m-file` son muy importantes; tanto así que la mayoría de las funciones preestablecidas de MATLAB son `m-files`. Se puede desplegar en la pantalla el interior de estas rutinas usando `type`. Por ejemplo:

```
>> type fibonacho

% Archivo que construye los primeros 25 elementos
% de la sucesión de Fibonacci.
fibonacci    = zeros(25,1);% Vector de Fibonacci [...]
```

Para las funciones preelaboradas por MATLAB, hay diferencia entre llamarlas por su nombre y llamarlas agregando `.m`:

```
>> type max
'max' is a built-in function.
```

```
>> type max.m
%MAX    Largest component.
%   For vectors, MAX(X) is the largest element in X. For [...]
```

Nota: Como verás, hay algunas funciones de MATLAB que son conocidas como `built-in functions` (funciones internas, en español), son éstas las que no podrás ver desplegadas con el comando `type` porque son propiedad privada de Mathworks (la empresa de MATLAB).

Observa los parecidos que hay entre `type max.m` y `help max`, tú puedes construir ayudas así de bonitas. Un poco más adelante te diremos más.

Archivos con funciones.
En ocasiones un guión puede parecer poco versátil, quieres repetir una y otra vez una serie de comandos donde varías algunos pocos parámetros. Es en estos casos que tener una función puede ser útil. En su interior puedes escribir un algoritmo que realice ciertos pasos a cada cambio que desees (básicamente es como una función de la preparatoria: das valores en el dominio, calculas y obtienes resultados en el rango). Su formato es simple:

```
function [variables_de_salida] = Nombre(variables_de_entrada)
    % Programa
end
```

La primera línea declara el nombre de la función (en nuestro caso el originalísimo `Nombre`) con sus argumentos de entrada `variables_de_entrada` y de salida `variables_de_salida`.

Importantísimo El nombre de la función debe de ser igual al del `m-file`; es decir, el archivo que la contiene debe llamarse «`Nombre.m`».

Una función puede tener varios argumentos de entrada o ninguno, lo mismo ocurre con los argumentos de salida. En ambos casos, cada argumento debe ser separado por una coma.

Construyamos una función; para ello, abre un nuevo `script` y guárdalo como «`stat.m`». El siguiente código debe ser el contenido de esta función:

```
function [mean, stdev] = stat(x)
% STAT     Cálculo de la media y la desviación estándar.
%
%    Dado un vector x, [M, D] = STAT(x) devuelve la media M
%    y la desviación estándar D del vector.

    % Primero necesitamos el tamaño del vector para poder
    % obtener la media de éste y su desviación estándar
    m     = length(x);
    mean  = sum(x)/m;
    stdev = sqrt(sum(x.^2)/m - mean^2);
end
```

Esta función calcula la media (`mean`) y la desviación estándar (`stdev`) de una variable vectorial `x`. Antes de continuar observa los «`;`» al final de cada comando. Esto lo hicimos para que no se imprimiera el resultado en la pantalla cada vez que se hiciera la cuenta. Este «`;`» es particularmente útil cuando se coloca al final de cada operación dentro de un ciclo `for`, por ejemplo.

Una vez guardada la función, puedes ejecutarla desde la `Command Window` llamándola con los argumentos adecuados. En nuestro caso: un vector, digamos $x = (1, 2, \cdots, 10)$.

```
>> [ym, yd] = stat(1:10)

>> ym =

 5.5000
```

```
>> yd =

 2.8723
```

Observa que obtenemos los resultados `ym` y `yd`. Intenta llamar la misma función sin colocar la parte entre corchetes, o colocando un «`;`» al final. Tal vez también quieras buscar con `type mean`, la función original de MATLAB. Aprovechando eso, observa la diferencia entre `type stat` y `help stat`.

¡Ya falta poco para que acabe esta guía!

Un punto flotante que flota muy lejos de donde debería (como la casa del viejito de Up)

Hace mucho ¡parecen años! prometimos mostrarte un punto flotante que creciera arbitrariamente. ¡Ha llegado ese momento! ¿A poco no sientes mariposas bullir en tu estómago de la emoción?[4]

Considera la siguiente función:

$$D(x) = \begin{cases} 2x, & \text{si } 0 \leq x \leq \frac{1}{2}, \\ 2 - 2x, & \text{si } \frac{1}{2} \leq x \leq 1. \end{cases}$$

Considera $x = 0.4$ y observa que forman el siguiente curioso ciclo:

$$\begin{aligned} D(x) &= 0.8, \\ D\big(D(x)\big) &= 0.4, \\ D\Big(D\big(D(x)\big)\Big) &= 0.8, \\ D\Big(D\big(D\big(D(x)\big)\big)\Big) &= 0.4, \\ &\vdots \\ \underbrace{D\Big(D\big(\ldots\big(D(x)\big)\ldots\big)\Big)}_{N \text{ veces}} &= \begin{cases} 0.8, & \text{si } N \text{ es par}, \\ 0.4, & \text{si } N \text{ es non}. \end{cases} \end{aligned} \tag{3}$$

Finalmente, definamos $g_N(x)$ como la función que surge de aplicar N veces D a x (como en (3)) y luego multiplicar ese resultado por N. Algo así como

$$g_N(x) = N \cdot \underbrace{D\Big(D\big(\ldots\big(D(x)\big)\ldots\big)\Big)}_{N \text{ veces}} = \begin{cases} 0.8 \cdot N, & \text{si } N \text{ es par}, \\ 0.4 \cdot N, & \text{si } N \text{ es non}. \end{cases}$$

[4] Consulta a tu médico.

Ahora sí, es hora de ver cómo MATLAB mete la pata (una de cal por las que van de arena). Programemos la función D:

```
function [ y ] = D( x )
% Si x sale del 0 y 1 regresa error
y = NaN;
    if (x >= 0 && x <= 0.5)
        y = 2*x;
    elseif (x > 0.5 && x <= 1)
        y = 2 - 2*x;
    end
end
```

¡No olvides guardarla como `D.m`! Programemos ahora la función g_N:

```
function [ y ] = gN(x, N)
for i = 1:N
    % Calcula D(D(...D(x)...)) aplicando D a cada vuelta
    x = D(x);
end
y = N*x;
end
```

Observa que $g_{10000}(x)$ debería ser 4000 (0.4×10000) pero MATLAB falla por mucho:

```
>> gN(0.4, 10000)

ans =

     0
```

El 10000 no tiene nada de especial, puedes aumentar N cuanto quieras para que la diferencia entre las matemáticas reales y MATLAB será tan grande como desees:

```
>> gN(0.4, 300000)

ans =

     0
```

¿Qué ha sucedido?

Cómo ganarte su corazón con gráficas en Matlab 😏

Una aportación genial de las computadoras es su capacidad gráfica. Hay infinidad de aplicaciones desde diseño de videojuegos hasta comunicación periodística. En Internet hay muchísimos sitios con ideas y programas para el desarrollo de visualizaciones impresionantes. Nuestros favoritos son: Information is Beautiful, Flowing Data y Visual Complexity. Si hoy no tienes examen de eco ¡explóralos antes de que ya no tengas tiempo de hacerlo 🕓!

Gráficas chafas con Matlab.
Comencemos por explorar las gráficas de $\mathbb{R}$ en $\mathbb{R}$ en Matlab, es decir, figuras planas bidimensionales y aburridas 😴.

En la Tabla 7 están los comandos esenciales en el uso de gráficas 2D. Los elementos `x`, `y`, `y1`, `y2` y `y3` son vectores de las mismas dimensiones. (No importa si son columnas o renglones, pero todos deben ser iguales.)

Cuadro 7. Comandos esenciales para la creación de gráficas bidimensionales en Matlab.

Comando	Descripción
`plot(x, y)`	Genera la gráfica de `y` contra `x`.
`plot(x, y1, x, y2, x, y3)`	Genera las gráficas `y1`, `y2`, `y3` todas contra `x`, en la misma ventana.
`xlabel('texto')`	Escribe la etiqueta del eje `x`.
`ylabel('texto')`	Escribe la etiqueta del eje `y`.
`title('texto')`	Escribe el título de gráfica.

La manera usual en la que trabaja Matlab es generando una figura `figure` y colocándola en una ventana. Cada vez que se llama algún comando para graficar, Matlab borra el contenido de la ventana más reciente y la recicla para la nueva figura. Dicho de otra manera, borra la gráfica previa.

```
>> x = [0 1 2 3 4 5];
>> y = [0 1 2 3 4 5];
>> z = [1 2 1 2 1 2];
>> plot(x, y)
>> plot(x, z)
```

Dos comandos muy importantes en este sentido son `figure` y `hold on/off`. Con el primero, le dices a Matlab que abra una nueva ventana para usarla. Ésta permanecerá vacía hasta que hagas algo que la llene (por ejemplo un `plot`). El segundo comando activa/desactiva la propiedad de sobrescribir en la ventana.

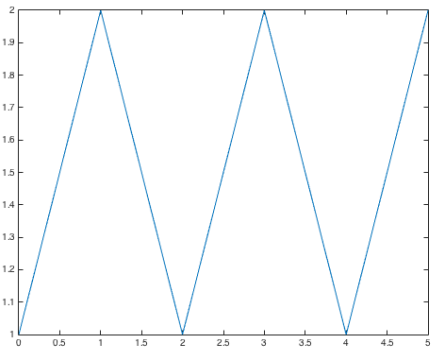

FIGURA 1. Tenemos `plot(x, z)` sin el `hold on`...

```
>> figure
>> plot(x, y)
>> hold on
>> plot(x, z)
>> hold off %Desactiva la opción de sobrescribir.
```

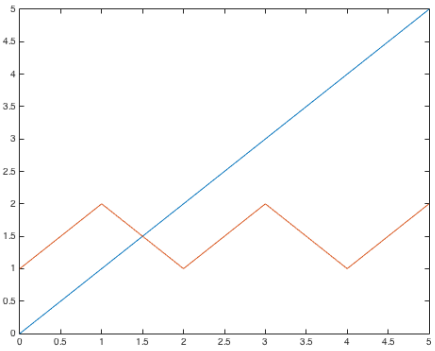

FIGURA 2. ... y claro, `plot(x, z)` con el `hold on`.

Como ya vimos, el comando `plot` genera una gráfica `x-y` en una ventana nueva. Podemos personalizar los títulos y ejes de la gráfica.

```
>> x = 0:0.1:pi;
>> y = exp(x);
>> plot(x, y)
>> xlabel('Número de gatos')
```

```
>> ylabel('Alegría')
>> title('Resumen de mi vida.')
>> grid
```

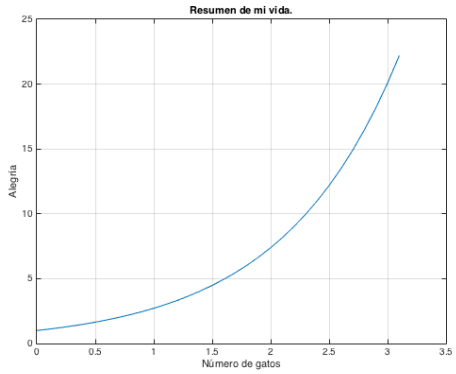

FIGURA 3. Un "`plot`" usando `xlabel, ylabel, title` y `grid`.

Esta serie de comandos crea una gráfica de `exp(x)` desde `x = 0` hasta `x = pi` con espaciado `0.1`. (Busca el comando `linspace` a ver si te gusta más.) La gráfica es lineal por pedazos (házle zoom y verás las líneas). Las etiquetas en los ejes representan una verdad absoluta e indiscutible 🐱.

Para graficar varias funciones en una misma ventana hay distintas opciones (todas resultan en lo mismo):

1. Considerar dos vectores diferentes:

```
>> x  = 0:0.1:pi;
>> y1 = sin(x);
>> y2 = sin(2*x);
>> plot(x, y1, x, y2)
```

2. Definir una matriz `Y` de dos columnas que contenga ambos vectores:

```
>> x = 0:0.1:pi;
>> Y = [sin(x)', sin(2*x)'];
>> plot(x, Y)
```

3. Usar el `hold on` y graficar cada una por su lado:

```
>> x = 0:0.1:pi;
>> y1 = sin(x);
```

```
>> y2 = sin(2*x);
>> plot(x, y1)
>> hold on
>> plot(x, y2)
>> hold off
```

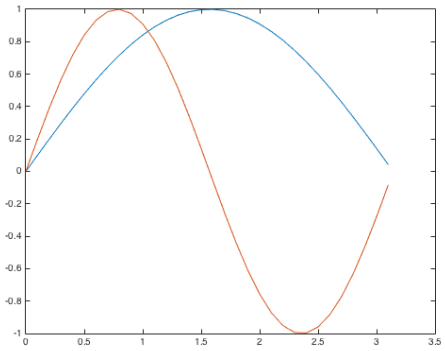

FIGURA 4. La misma gráfica con 3 recetas diferentes.

Opciones adicionales de `plot` incluyen el cambio de líneas por puntos y agregado de leyendas:

```
>> plot(x, y1, 'om', x, y2, '--g')
>> legend('Círculos', 'Rayas')
```

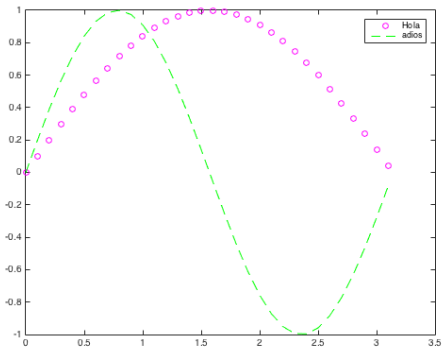

FIGURA 5. Un bonito `plot` con colores y leyenda.

En la instrucción anterior, `'om'` pone círculos de color magenta y `'--g'` líneas truncas de color verde (opciones adicionales se encuentran en la ayuda de `plot`).

Por último, y como el título de esta sección promete, te enseñamos a hacer una gráfica para ganarte el corazón de esa persona especial que conociste en el After Eco 😏.

```
>> x = [-2:.001:2];
>> y = sqrt(cos(x).*cos(200*x) + sqrt(abs(x)) - 0.7)...
     .* (4 - x.*x).^0.01;
>> plot(x, y, '--or')
```

Te dejamos el misterio de develar el contenido de esta gráfica.

Gráficas chafas con MATLAB pero en 3D.
Una de las aplicaciones más interesantes es la creación de superficies tridimensionales (como el logo de MATLAB). Para ello usaremos al comando `surf` y su hermano `mesh`. Éstos se alimentan de tres matrices: una con entradas en el eje `X`, otra con sus correspondientes valores en el eje `Y` y una última con la "elevación" de la superficie correspondiente a cada $(x_{i,j}, y_{i,j})$ en `Z`.

Primero creemos vectores con los valores `x` y `y` de interés:

```
>> x = -2:0.05:2;
>> y = x;
```

Luego obtenemos matrices con todas las posibles combinaciones de esos `x`, `y` usando `meshgrid`:

```
>> [X, Y] = meshgrid(x, y);
```

Finalmente, calculamos `Z` como una matriz en función de los valores de `x` y `y`; por ejemplo:

```
>> Z = X./exp(X.^2 + Y.^2);
```

El comando `surf` crea una superficie:

```
>> surf(X, Y, Z)
```

Mientras que `surfc` proyecta las curvas de nivel asociadas a dicha superficie:

```
>> surfc(X, Y, Z)
```

El comando `mesh` genera sólo líneas que atraviesan la superficie (sin rellenarla) y `meshc` repite el mismo proceso con curvas de nivel.

```
>> mesh(X, Y, Z)
```

Finalmente, el comando `contour` regresa las curvas de nivel y `contourf` las mismas curvas pero coloreadas de mejor manera:

```
>> contourf(X, Y, Z)
```

Usando `subplot` podemos agrupar las gráficas en una figura más grande. El primer número del `subplot` indica el número de filas en dicho gráfico; el segundo, el número de columnas y el tercero la posición de la gráfica que estamos poniendo. El código es:

```
>> subplot(3, 1, 1)
>> surfc(X, Y, Z)
>> subplot(3, 1, 2)
>> mesh(X, Y, Z)
>> subplot(3, 1, 3)
>> contourf(X, Y, Z)
```

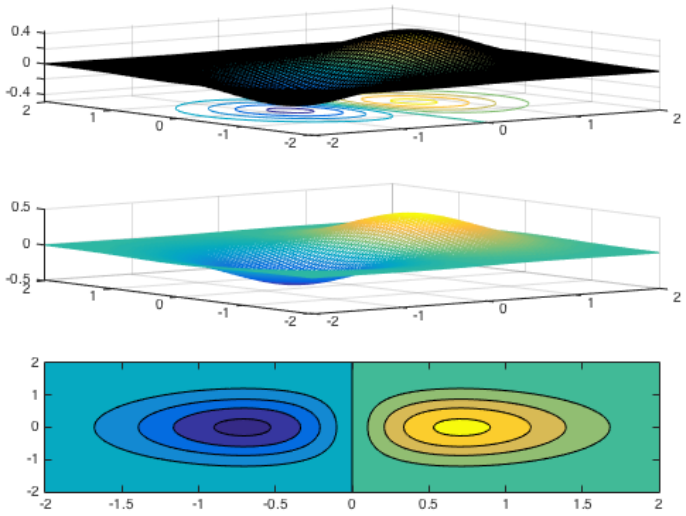

FIGURA 6. `subplot` de las figuras tridimensionales.

Esto es sólo una prueba de las gráficas que puedes crear con MATLAB. ¡Hay muchas más!

Gráficas bonitas con Plotly.
Entre la gente que hace gráficas para vivir, el consenso indica que una de las mejores opciones (actualmente) es usar D3.js (entra a su sitio, en verdad valen la pena).

Dichas gráficas (y documentos), sin embargo, requieren conocimiento de Javascript lo cual escapa de los límites de esta clase (y este documento, pues tampoco somos Wikipedia). Una opción que aterriza muchas de las gráficas de D3.js a MATLAB es Plotly. Ésta es una caja de herramientas que se agrega a MATLAB (o a otros programas) que cuenta con una versión de paga (la más nueva en su momento) y una gratuita (la que usaremos nosotros).

Para usar Plotly lo único que requieres es una llave de Api (¿una qué! 😟). Ésta es una clave única que te dan al momento de registrarte. Para ello entra aquí y regístrate. Al momento de registro te darán un nombre de usuario (digamos `AmoLosGatos`) y una "clave" que se llama *Api key* (digamos `12345`). Guarda bien ambos valores 💾 pues los necesitarás a la brevedad.

Una vez completado el registro, instala Plotly. Las instrucciones se encuentran aquí. Lo que requieres es descargar esta carpeta y descomprimirla. Una vez descomprimida corre la función `plotlysetup` (ahí incluida) con tu nombre de usuario como primer parámetro y tu clave de Api como segundo:

```
>> plotlysetup('AmoLosGatos', '12345')

Adding Plotly to MATLAB toolbox directory ...  Done
Saving Plotly to MATLAB search path via startup.m ...  Done
Saving username/api_key credentials ... Done

Welcome to Plotly!
```

Por último usa el siguiente comando para que puedas usar Plotly sin una conexión a Internet (antes te hacían pagar por esto pero ya se volvieron buenas personas):

```
>> getplotlyoffline('https://cdn.plot.ly/plotly-latest.min.js')

    Success! You can generate your first offline graph
    using the 'offline' flag of fig2plotly as follows:
    >> plot(1:10); fig2plotly(gcf, 'offline', true);
```

El comando `fig2plotly` es el más sencillo de aprender. Basta con generar una gráfica y luego poner `fig2plotly(gcf, 'offline', true)` para que Plotly automáticamente transforme tu gráfica.

```
>> x      = 0:0.1:pi;
>> seno   = sin(x);
>> coseno = cos(x);
>> plot(x, seno, x, coseno);
>> legend('seno', 'coseno');
>> fig2plotly(gcf, 'offline', true)
```

Estas gráficas son mucho mejores que las de MATLAB: puedes hacer "zoom" infinito sin que se pierda la calidad 😱; pasar el ratón sobre las gráficas da los valores estimados para cada una así como los valores en el eje `x`; finalmente, nota que si das clic sobre uno de los nombres en la leyenda, ese valor se oculta.

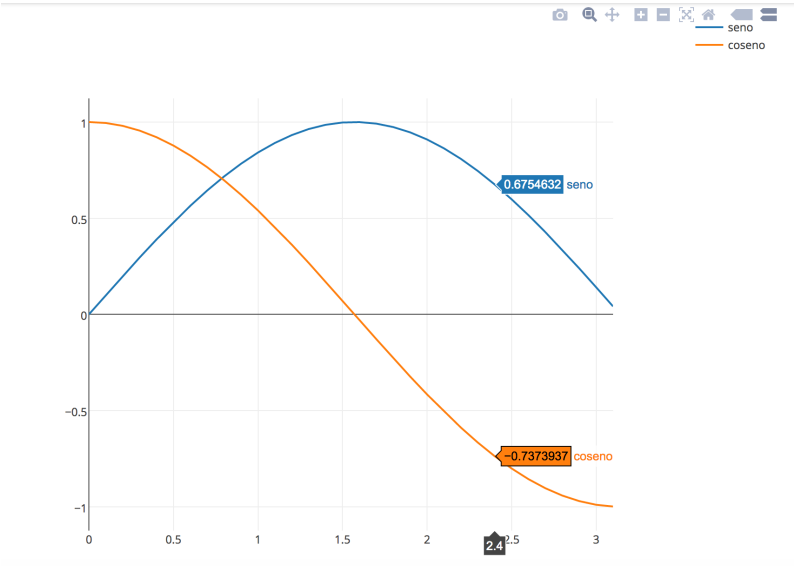

FIGURA 7. Gráfica con Plotly

Para guardar 💾 como imagen fija hay que dar click en la cámara (esquina superior derecha). Si la deseas guardar como gráfica interactiva debes agregar a `fig2plotly` una especificación de `filename` con el nombre del archivo:

```
>> contour(rand(10,10));
>> fig2plotly(gcf, 'offline', true, 'filename', 'Para_mi_sala')
```

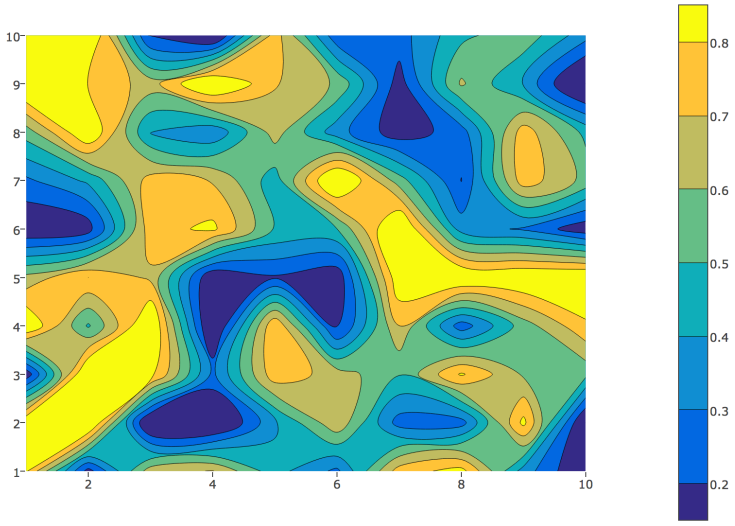

FIGURA 8. Segunda gráfica con Plotly

Hay muchos ejemplos de gráficas con Plotly. Los más interesantes (a nuestro juicio) son los Dashboards. Éstos te permiten combinar gráficas de diferentes cosas para una visualización interactiva más completa. Puedes combinar streaming de datos de la vida real con gráficas interactivas prefabricadas o bien actualizadas al momento. ¡No esperes más y crea tu propio Dashboard!

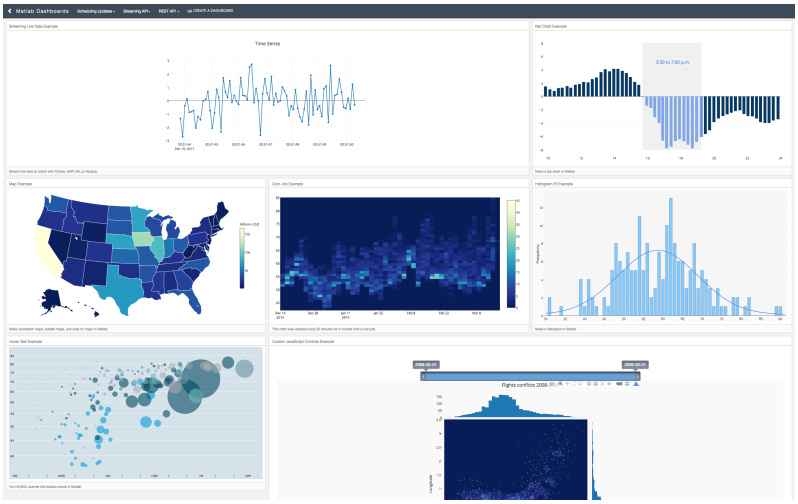

FIGURA 9. Un Dashboard de Plotly

Alternativas a Matlab

Aunque en estas notas, se desarrolla el uso de Matlab, éste no lo es todo en la vida. A nuestro juicio, la razón principal para no usar dicho programa es que cuesta (y si no eres estudiante o sigue subiendo el dólar, cuesta mucho). A continuación comentamos algunos programas gratuitos que tienen bastante en común con Matlab y por los cuales puedes sustituir Matlab confiando que dichos programas tienen casi todo (o más) de lo que Matlab contiene.

Octave: https://www.gnu.org/software/octave/
Octave es la versión gratuita de Matlab. Prácticamente todos los comandos de Matlab están en él (de hecho la meta de la gente de Octave es hacer un Matlab exactamente igual pero gratuito). La sintaxis es la misma y lo puedes correr en Internet (la excusa de "El Centro de Cómputo no abre los domingos y no tengo Matlab en mi casa" ya no te sirve). Octave es tan semejante a Matlab, que incluso puede correr tus `m-files` favoritos.

Scilab: http://www.scilab.org/
Scilab es la segunda mejor alternativa a Matlab. A diferencia de Octave (y así como el resto de la lista) no busca ser un reemplazo de Matlab sino una alternativa (aunque, ya en la vida real son muy parecidos).

R: https://www.r-project.org/
R es el Matlab de la gente de estadística. Así como Matlab tiene excelentes funciones para diseño de prototipos (si esto es lo tuyo, tu carrera es otra), R cuenta con la (actualmente) mejor librería estadística. En cuanto al resto de los problemas matemáticos que buscan resolver (ecuaciones diferenciales, simulación estocástica, optimización) son indistinguibles.
Un dato curioso es que la *Comprehensive R Archive Network* (CRAN) de México tiene una de sus sedes en el ITAM.

Python: https://www.python.org/
Python es un lenguaje de propósitos múltiples (no enfocado sólo en matemáticas como los otros de esta lista). Desde 2001 a la fecha, la librería SciPy se ha enfocado en métodos numéricos así como matemática simbólica y graficación. Python es muy utilizado en Ciencia de Datos para Aprendizaje de Máquina, Visión por Computadora y otras monerías.

Julia: http://julialang.org/
Julia es el Matlab del futuro. Aunque aún no se ha terminado la primera versión (van en la 0.5), las pruebas indican que es muchísimo más rápido que Matlab y permite el uso de múltiples procesadores de manera más eficiente (en general los cálculos son más rápidos si se hacen

así). Su escritura es similar a MATLAB y aunque no tenga aún todas las funcionalidades de éste, vale la pena no perderlo de vista.

DESPEDIDA

Como hemos dicho al inicio de este texto (y si no lo dijimos pues ya te enteraste), sólo pretendíamos dar una breve introducción al entrono de MATLAB (hacer un texto más grande sería cansado y ya tenemos sueño 😴). Esperamos que ahora puedas experimentar por tu cuenta a partir del `help` y la documentación (si no, hay otros tutoriales de MATLAB como éste que usa Comic Sans o este video que está bien aburrido. También está la academia de MATLAB).

Nunca olvides que la mejor forma de pedir ayuda de MATLAB es a través del `help`, del oráculo o gritando 😱 desesperadamente en el Centro de Cómputo hasta que alguien llegue.

Cualquier duda o sugerencia respecto a estas notas envíanos un correo electrónico (y si no llega a `spam` la tomaremos en cuenta).

Algunas referencias electrónicas

Foros y repositorios:

1.- StackOverflow: http://stackoverflow.com/
2.- Computational Science Stack Exchange: http://scicomp.stackexchange.com/
3.- Mathematics Stack Exchange: http://math.stackexchange.com/
4.- MATLAB central: https://www.mathworks.com/matlabcentral
5.- Github: https://github.com/
6.- Gitlab: https://about.gitlab.com/
7.- Bitbucket: https://bitbucket.org/

Cálculo simbólico

8.- Wolfram Alpha: https://www.wolframalpha.com/
9.- Mathematica: https://www.wolfram.com/mathematica/
10.- Maple: https://www.maplesoft.com/products/maple/
11.- Sage: https://cloud.sagemath.com/ ,
12.- Yacas: http://www.yacas.org/
13.- Sympy: http://www.sympy.org/en/index.html
14.- Maxima: http://maxima.sourceforge.net/es/

Visualización y gráficas

15.- Information is Beautiful: http://www.informationisbeautiful.net/ ,
16.- Flowing Data: https://flowingdata.com/
17.- Visual Complexity: http://www.visualcomplexity.com/vc/
18.- D3.js: https://d3js.org/
19.- Plotly: https://plot.ly/matlab/

Pablo Castañeda
Departamento Académico de Matemáticas, ITAM
Río Hondo 1, Ciudad de México 01080, México
E-mail address: pablo.castaneda@itam.mx

Rodrigo Zepeda Tello
Centro de Investigación en Salud Poblacional, INSP
7a. Cerrada de Fray Pedro de Gante 50,
Ciudad de México 14080, México
E-mail address: rodrigo.zepeda@insp.mx