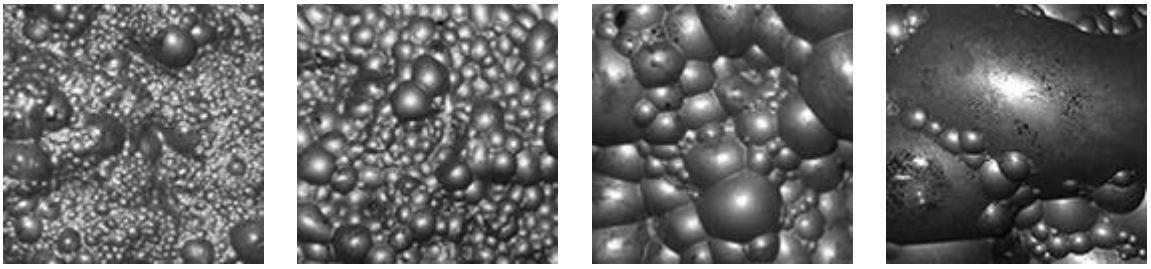
Control de Procesos – Ayudantía 00a

Repaso de Matlab

PARTE I: Matlab

Ejercicio 1. Las siguientes imágenes muestran distintos estados de un proceso de flotación para un mineral.



(a) (b) (c) (d)

Figura 1. Imágenes de un proceso de flotación de minerales.

Se adjunta un archivo `froth.mat` con las imágenes en forma de matrices de 130x130. Se pide calcular los siguientes valores para cada imagen y comparar los resultados: i) promedio, ii) mediana, iii) moda, iv) desviación estándar y v) la entropía, definida a continuación:

$$H(I) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Para una imagen I, p_i es la probabilidad de ocurrencia del valor de intensidad de un pixel, es decir, sería igual al número de pixeles con ese valor dividido en el total de pixeles de la imagen, y n es el número de niveles de intensidad de la imagen, para una imagen de 8 bits el número total es de $2^8 = 256$ niveles (en Matlab van de 0 a 255). Compare su resultado con la función `entropy()` de Matlab. ¿Cuál de los valores calculados usaría para una comparación efectiva entre las muestras?

Ejercicio 2. Solución de ecuaciones diferenciales. El balance de masa para un reactor de mezcla completa puede ser escrito como:

$$V\frac{dc}{dt} = F - Qc - kVc^2$$

Donde V = volumen ($12 m^3$), c = concentración (g/m^3), F = flujo másico alimentado (175g/min), Q = flujo de salida ($1 m^3/min$), y k = una tasa de reacción de segundo orden ($0.15 m^3/g/min$). Si $c(0) = 0$, simular la EDO hasta que la concentración alcance un nivel estable.

Ejercicio 3. Números palíndromos. Un número natural es un palíndromo si se lee igual de izquierda a derecha y de derecha a izquierda. Por ejemplo, 35853 es un palíndromo, mientras que 12345 no lo es. Escriba un programa que indique si el número ingresado es o no palíndromo.

Ejercicio 4. Graficar las siguientes ecuaciones paramétricas, experimentar con los rangos de valores para las variables.

a) $x(t) = 2\cos(t) + \sin(2t)\cos(60t),\ y(t) = \sin(2t) + \sin(60t)$

b) $x(t) = t - 1.6\cos(24t),\ y(t) = t - 1.6\sin(25t)$

c) $x(t) = 16\sin^3(t),\ y(t) = 13\cos(t) - 5\cos(2t) - 2\cos(3t) - \cos(4t)$

d) Integral seno versus integral coseno de Fresnel, S(t) vs C(t), usar las funciones `fresnels` y `fresnelc` de Matlab para evaluarlas, la curva resultante se conoce como la espiral del Cornu.

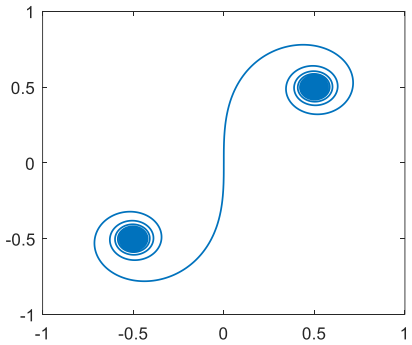
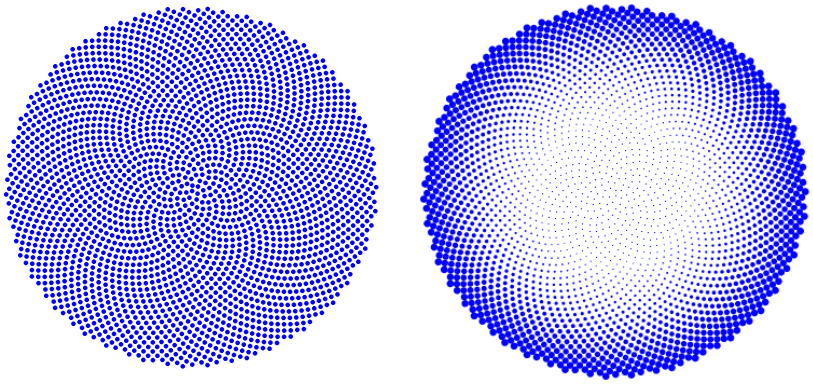


Figura 2. Espiral de Cornu.

Ejercicio 5. Misceláneo. El código entregado a continuación genera la imagen de una flor de maravilla como se muestra en la Figura 3.(a). Modificar el código para obtener la representación mostrada en la Figura 3.(b).

Código:

```
phi          = (sqrt(5)+1)/2;% Número áureo
golden_angle = 2*pi/phi;
max_angle    = 10000;
theta        = 1:golden_angle:max_angle;% ángulo
r            = sqrt(theta);% radio
[x,y]        = pol2cart(theta,r);% coordenadas polares a cartesianas
figure
plot(x,y,'.','MarkerSize',10);axis off;
```



(a) (b)

Figura 3. Flor de maravilla.

PARTE II: Señales en sistemas

Grafique las siguientes señales continuas en un rango de tiempo adecuado ($u(t)\ y\ r(t)$ son las señales escalón y rampa unitarias).

a) $f(t) = u(t) + u(t-5)r(t-5)$

b) $f(t) = u(-t) + u(t-5)r(t-5)$

c) $f(t) = \sin(2\pi 50t - \pi/6)u(t)$

d) $f(t) = \sin(2\pi 50t)^2 u(t)$

e) $f(t) = e^{-10t}u(t)$

f) $f(t) = e^{-10t}u(t-2)$

g) $f(t) = e^{-10(t-2)}u(t-2)$

h) $f(t) = e^{-(10+j10)t}u(t)$

i) $f(t) = e^{-j10t}u(t)$

j) $f(t) = e^{-(10+j10)t}r(t)$

k) $f(t) = 50 - 750\cdot\left[r(t-0.05) - r(t-0.15) - r(t-0.35) + r(t-0.45)\right]$

$\omega(t) = 2\pi f(t),\ y(t) = 1.5\cdot\sin(\omega(t)\cdot t),\ t \in [0, 0.5]$

PARTE III: Problemas opcionales (pero recomendados)

Problema 1. Animación: Movimiento Browniano. Escribir una función con la siguiente declaración: `brown2D(N)`. La función toma como único argumento el valor `N`, el cual es un entero que especifica el número de partículas o puntos a simular. Todos los puntos debieran iniciar su movimiento en el origen (0,0). Graficar todos los puntos en una figura usando como marcador '**.**', y fijar los ejes a una región rectangular con límites desde -1 a 1 en ambos ejes. Para simular el movimiento Browniano de los puntos, escribir un ciclo de 1000 iteraciones el cual calculará una nueva posición de *x* e *y* para cada punto y desplegará estas nuevas posiciones como una animación (la sentencia `drawnow` puede ser de ayuda). La nueva posición de cada punto se obtiene al sumar una variable normalmente distribuida con una desviación estándar de 0.005 a cada valor de *x* e *y* (usar `randn`; si por ejemplo tienes 10 puntos, necesitarás sumar 10 valores aleatorios distintivos a los valores de *x* y 10 valores aleatorios distintivos a los valores de *y*). Cada ves que las nuevas posiciones de todos los puntos son calculadas, graficarlos sobre la figura.

Lo que verás es una simulación del fenómeno de difusión, donde las partículas se mueven aleatoriamente lejos del centro de la figura. Por ejemplo, las siguientes figuras muestran la simulación al inicio, mitad y final para 100 puntos:

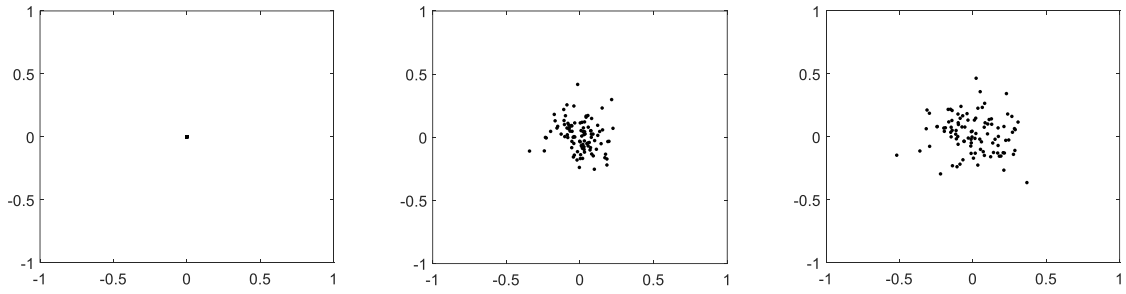


Figura 4. Simulación de movimiento Browniano.

Adicional: ¿cómo guardar la animación completa en formato gif o como un video?

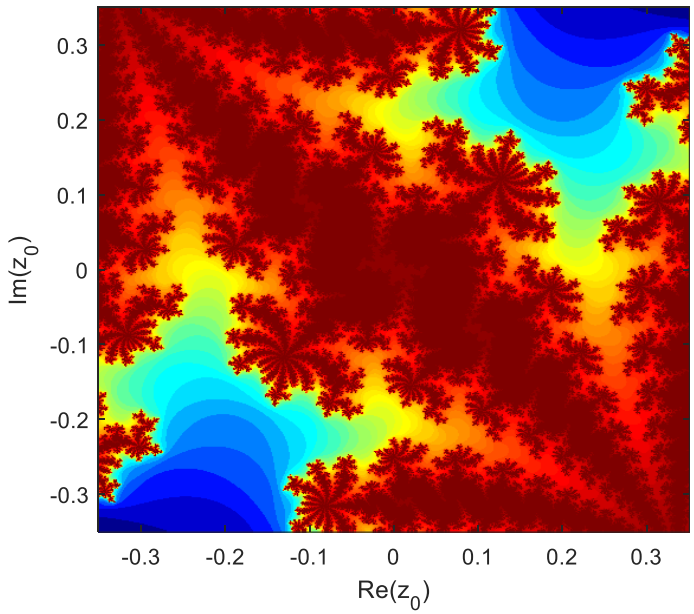
Problema 2. *Julia Sets*. En este problema generarán los Conjuntos de Julia [wiki], la siguiente descripción es una adaptación del artículo de *Wikipedia* y del curso *Introduction to Matlab* del profesor Danilo Šćepanović del MIT.

Dados dos números complejos, c y z_0, se define la siguiente recursión:

$$z_n = z_{n-1}^2 + c$$

Este es un sistema dinámico conocido como mapa cuadrático. Dada una elección específica de c y z_0, la recursión anterior generará una secuencia de números complejos z_1, z_2, z_3, ..., llamada órbita de z_0. Dependiendo de la elección exacta de c y z_0, es posible generar un gran rango de patrones de órbita, sin embargo, para un c dado, muchas elecciones de z_0 pueden generar orbitas que tienden a infinito, i.e. el módulo $|z_n|$ crece sin límite a medida que n aumenta, mientras que, para otros valores de c, ciertas elecciones de z_0 generan eventualmente órbitas que parecen caer en ciclos periódicos. Finalmente, algunos valores iniciales generan órbitas que parecen danzar alrededor del plano complejo, aparentemente de forma aleatoria, este es un ejemplo de *caos*. Dichos valores iniciales generan el conjunto de Julia del mapa, denotado como J_c.

En este problema, se pide que escriban un script en Matlab que visualice una versión ligeramente diferente de este conjunto, llamado el conjunto de Julia relleno (o el conjunto *prisionero*), denotado K_c, el cuál es el conjunto de todos los z_0 con orbitas que no tienden a infinito. La figura de abajo ilustra un conjunto de Julia para un valor particular de c. Ustedes programarán un código en Matlab que genere tales fractales.



Para el desarrollo del problema seguir los siguientes pasos:

a) Se ha mostrado que si el módulo de z_n es mayor a 2 para algún n entonces es garantía de que la órbita tenderá a infinito. El valor de n para el cual esto es cierto es llamado la "velocidad de escape" del z_0 particular. Escribir una función que retorne la velocidad de escape de un z_0 y c dados. La declaración de la función debiera ser: `n = escapeVelocity(z0,c,N)` donde `N` es la velocidad de escape máxima permitida, básicamente, si el módulo de z_n no excede 2 para

$n < N$, la función retornará N como la velocidad de escape (esto prevendrá tener ciclos infinitos). Usar la función **`abs()`** para calcular el módulo de un número complejo.

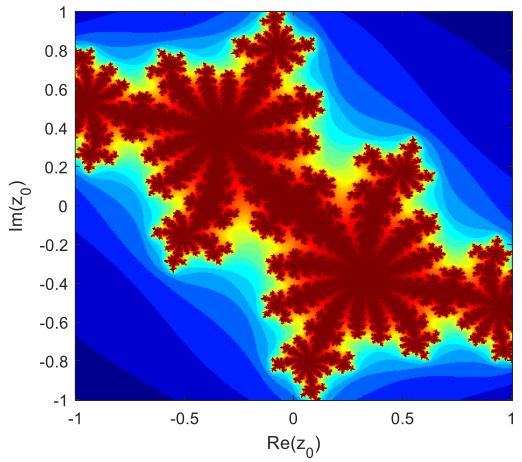
b) Para generar el conjunto K_c, escribir la siguiente función `M = julia(zMax,c,N)`. `zMax` será el máximo de las partes imaginarias y real de los números complejos z_0s para los cuales se calcularán las velocidades de escape. `c` y `N` son los mismos valores definidos anteriormente, y `M` es la matriz que contiene la velocidad de escape de varios z_0s.

   i. En esta función, partir generando una matriz de 500×500 que contendrá los números complejos con parte real entre `-zMax` y `zMax`, y parte imaginaria también entre esos valores. Llamar a esta matriz `Z`. Hacer que la parte imaginaria varíe a lo largo del eje *y* y la parte real a lo largo del eje *x* de la matriz. Esto se puede hacer fácilmente usando `linspace` y `meshgrid`, la otra opción es hacerlo dentro de un ciclo.

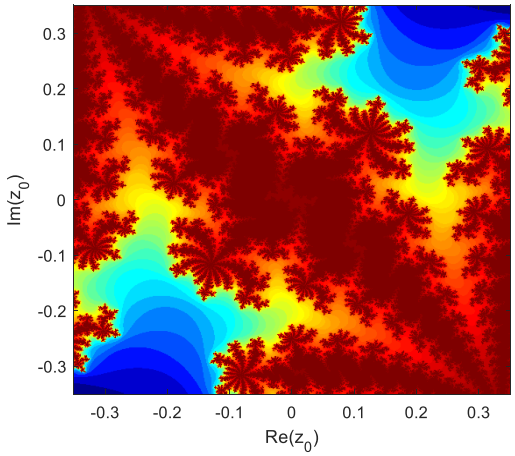
   ii. Luego, para cada elemento de `Z`, calcular la velocidad de escape (llamando a la función `escapeVelocity()` desarrollada) y guardarla en la misma posición en la matriz `M`. Al hacer esto la matriz M debiera tener el mismo tamaño que `Z` y contener las velocidades de escape entre 1 y N.

   iii. Ejecutar `julia()` con varios valores de `zMax`, `c`, y `N` para generar varios fractales. Para desplegarlos de mejor forma, usar `imagesc()` sobre `atan(0.1*M)` (experimentar con este mapeo), también pueden usar `axis xy` para que los valores de y no aparezcan invertidos en su eje. Experimentar con distintos mapas de color usando `colormap()`. Nota: dependiendo de los tamaños de la matriz inicial, la función puede tomar un tiempo en ejecutarse.

Abajo se muestran unos ejemplos de fractales creados con ciertos valores de `zMax`, `c` y `N`. ¡¡Explorar!!



`M = julia(1,-.297491+0.641051i,100)`



`M = julia(.35,-.297491+0.641051i,250)`

Otros ejemplos:

i) `M = julia(1.3,0.3+0.6i,500);`

ii) `M = julia(1.5,-.687+0.312i,500);`

Otro conjunto interesante de explorar es el Conjunto de Mandelbrot, derivado del conjunto de Julia.

Problema 3. *Pi Day*. El pasado 14 de marzo (3/14) se 'celebró' el día del número π, en relación con esto, ejecutar el script `Piday.m` para visualizar los dígitos de este número. (Este script fue propuesto por desarrolladores de la comunidad de Matlab y se dejan los créditos respectivos en el archivo). La gráfica de más abajo muestra la visualización generada. Modificar el script y realizar visualizaciones de otros números irracionales, ej. el número *e*, raíces de números primos, el número de oro, etc., experimentar también con la paleta de colores.