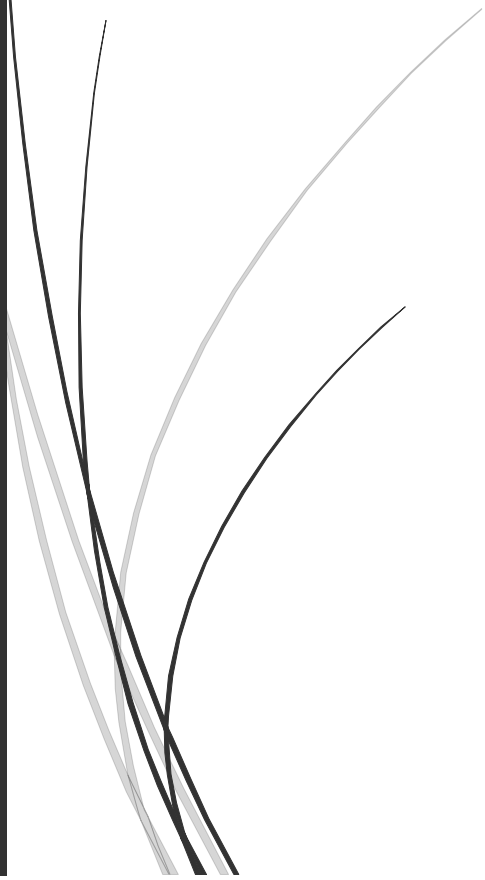
6 de noviembre de 2016

Homework 3

Ingeniería de redes y servicios



Iago Martínez Colmenero
Juan Francisco García Gómez

Homework 3

ÍNDICE

1 INTRODUCCIÓN

Debido a la pequeña longitud de los primeros apartados hemos decidido agruparlos en este apartado introductorio bajo el nombre (en referencia a *Homework 1*) *shortProblems.m*.

1.1 SISTEMA LINEAR DE ECUACIONES

Atendiendo al manual interno de Matlab (1):

> *X = A\B is the solution to the equation A*X = B.*

Por tanto, una vez definimos las matrices, aplicamos el operador y obtenemos el resultado, que imprimimos utilizando **fprintf**.

1.2 INTEGRACIÓN NUMÉRICA

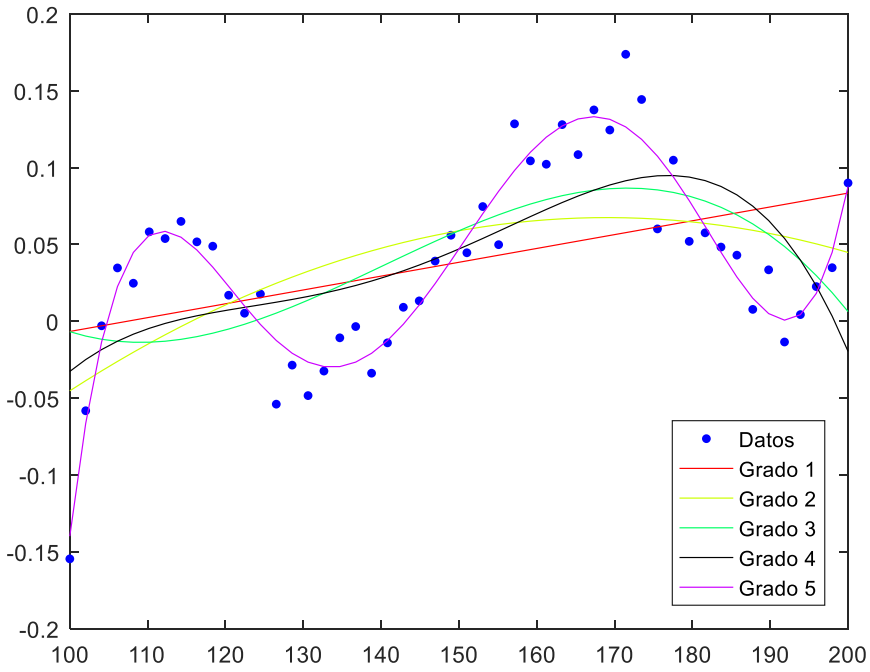
La diferencia de los métodos de integración propuestos son que **trapz** realiza una aproximación por método trapezoidal y **quad** aproxima con una cuadratura adaptativa de Simpsons. Cabe destacar que MatLab nos aconseja el uso de la función **integral** en favor de **quad** debido a una futura supresión de ésta.

1.3 CÁLCULO DE INVERSA

Para el cálculo de la inversa nos valemos de la función **inv**, no obstante, podría realizarse igualmente elevando la matriz a *-1*. Al operar la inversa con la matriz original observamos como conseguimos la matriz identidad independientemente del orden.

2 POLINOMIO DE AJUSTE

Una vez cargados y representados los datos, utilizamos **polyfit** para la obtención de los coeficientes necesarios del polinomio de grado n que mejor se ajustan a la función. A continuación, usamos **polyval** para generar los valores de un polinomio (con los coeficientes obtenidos) sobre el vector de datos y poderlo así representar.

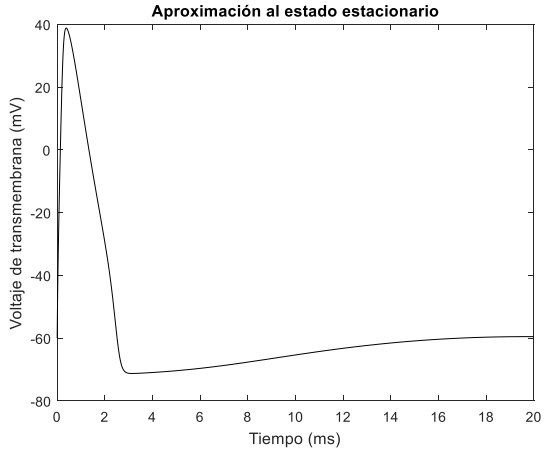


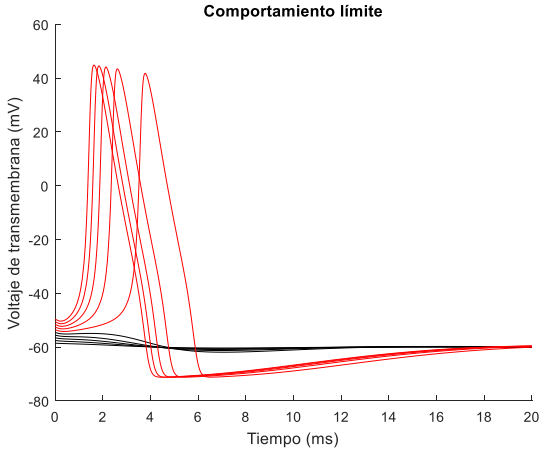
3 MODELO NEURONAL HODKIN-HUXLEY

Uno de los principales problemas en este apartado ha sido comprender el enunciado de la tarea.

Una vez hecho esto lo primero que hacemos es la función **ODE**. En ella, pasamos los datos de m, n, h, el voltaje y, finalmente, el tiempo de ejecución. Éste último se requiere para poder invocar correctamente la función **ode45**. Puesto que realmente no hacemos uso de dicho parámetro lo nombramos como ~.

Una vez hemos implementado la función **ODE** la usamos para llamar a **ode45**, pasando como parámetros el intervalo de tiempo de ejecución y los valores de inicialización de m, n, h, y v. Primero, ploteamos la última columna de los valores devueltos para poder observar el comportamiento en un estado estacionario. Tras esto y para acabar sacamos la gráfica con todas las ejecuciones.





4 OPCIONAL: CONJUNTO DE JULIA

El conjunto de Julia es una familia de conjuntos fractales que se obtienen al ser iterados por la siguiente función (2):

$$z_0 = z$$

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

4.1 ESCAPEVELOCITY.M

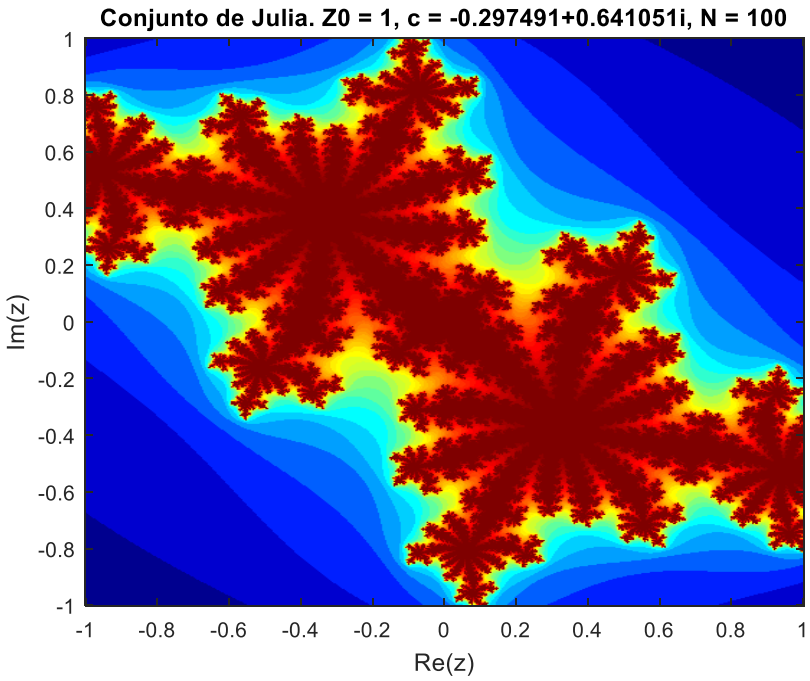
En esta función, mediante un bucle, evaluamos cuando el módulo de z_n tiende a infinito. Una vez observamos esta tendencia devolvemos el n.

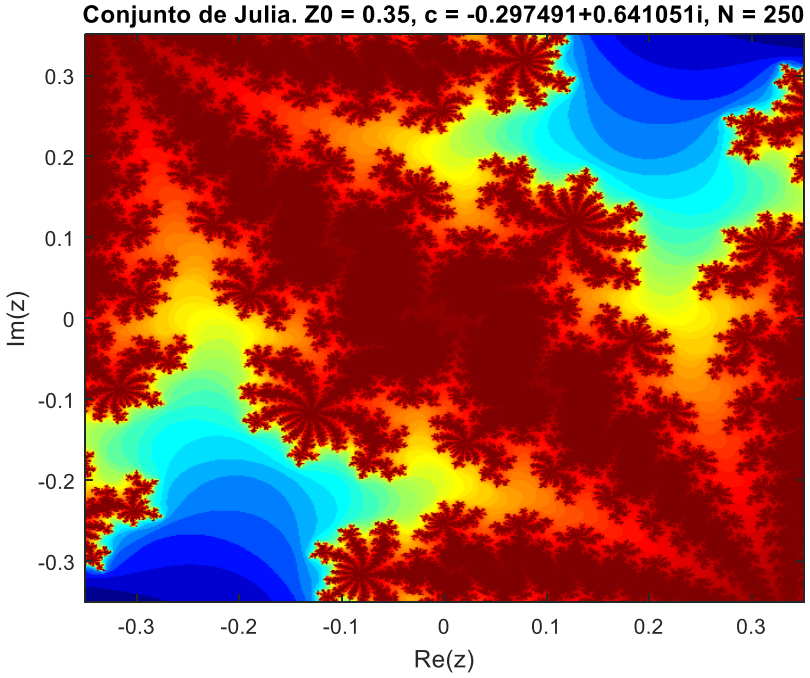
4.2 JULIA.M

Julia.m nos permite generar una matriz con los valores retornados de *escapeVelocity.m*. El eje vertical de esta matriz se corresponde a la parte imaginaria mientras que el horizontal a la parte real.

4.3 JULIATEST.M

Finalmente, comprobamos el funcionamiento con los mismos parámetros aportados en el ejemplo. Utilizamos **imagesc** para la representación y un **colormap *jet*** para conseguir la mayor similitud posible con los ejemplos.





La mayor dificultad que nos ha supuesto este apartado ha sido, por un lado, la correcta comprensión del conjunto matemático, así como de lo que nos pedían, y, por otro lado, el paso correcto de los ejes x e y a **imagesc**.

5 REFERENCIAS

1. **Solve systems of linear equations Ax = B for x - MATLAB. [En línea] https://es.mathworks.com/help/releases/R2016b/matlab/ref/mldivide.html.**

2. Conjunto de Julia. *Wikipedia.* [En línea] https://es.wikipedia.org/wiki/Conjunto_de_Julia.