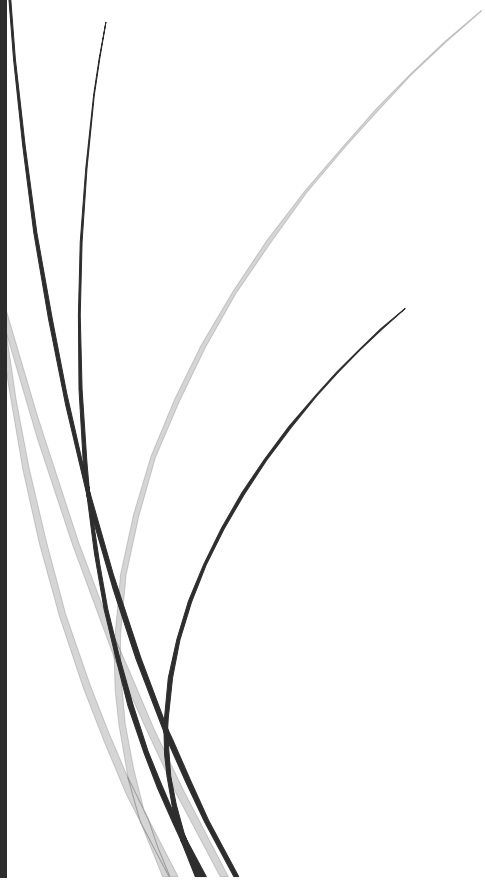
7 de octubre de 2016

Homework 1

Ingeniería de redes y servicios



Iago Martínez Colmenero
Juan Francisco García Gómez

Homework 1

ÍNDICE

1 SHORTPROBLEMS.M

Este *script* servirá para una primera toma de contacto con operaciones básicas de Matlab. Los tres primeros apartados son para definición de variables cuya declaración es similar a la de otros lenguajes como Ansi-C. Cabe destacar el uso de las funciones **exp** para trabajar con exponenciales, **logspace** para declarar un vector cuyos valores se separan logarítmicamente, **ones** para la generación de matrices de 1 (esta función esta sobrecargada como **ones(dimensión),** que da como resultado una matriz cuadrada de la dimensión especificada, y **ones(dimensión1, dimensión2, etc.)** para especificar más dimensiones como es el caso de *dMat*). Con funciones como **diag** especificamos la diagonal de una matriz que queremos, con **reshape** generamos matrices en base a la salida de operaciones que introduzcamos y **floor** y **ceil** redondean a la baja y al alza respectivamente. En cuanto a la variable *fMat*, que trabaja con 3 de las funciones mencionadas, primero generamos una matriz de numeros aleatorios con **rand**. Puesto que **rand** da valores aleatorios entre 0 y 1, lo multiplicamos por 6 para tener un rango de datos que vaya de **0·6=0** a **6·1=6**; y, finalmente lo desplazamos con una matriz de mismas dimensiones con valor de -3.

La mayor dificultad que han ofrecido los apartados del 4 al 6 ha sido interpretar las expresiones para saber cuando debiamos realizar operaciones *elemento a elemento* en una matriz, o cuando una matriz era cuadrada y podía tener sentido elevarla al cuadrado para trabajar con ella. Cuando teniamos que realizar las operaciones elemento a elemento, anteponiamos un punto "." al operador. En alguna ocasión además nos hemos valido del operador **.'** para la trasposición de matrices.

Los subapartados a y b del apartado 7 nos hacen jugar con funciones para sumar y calcular medias tales que **sum** y **mean.** Es con el subapartado c con el que más tiempo hemos pasado para dejar un código más elegante. Inicialmente probamos si funcionaba ir elemento a elemento cambiando los de la primera fila. Después, generamos la línea *eMat(1:size(eMat,1):end) = 1* en la que nos desplazamos desde el primer elemento, hasta el final en pasos del tamaño de la columna (con un 2 obtendríamos el mismo resultado, pero con el empleo de la función **size** conseguimos un resultado genérico aplicable a cualquier matriz). Investigando un poco más acerca del *indexing* llegamos a una expresión más compacta (*eMat(1,:) = 1*) en la que trabajamos directamente con la fila completa. Los subapartados restantes no suponen mayor problema. La funcion utilizada en el último, **find,** devuelve los índices de la matriz que se le pasa como parámetro que cumplen la condición impuesta. Como ejemplo dos ejecuciones:

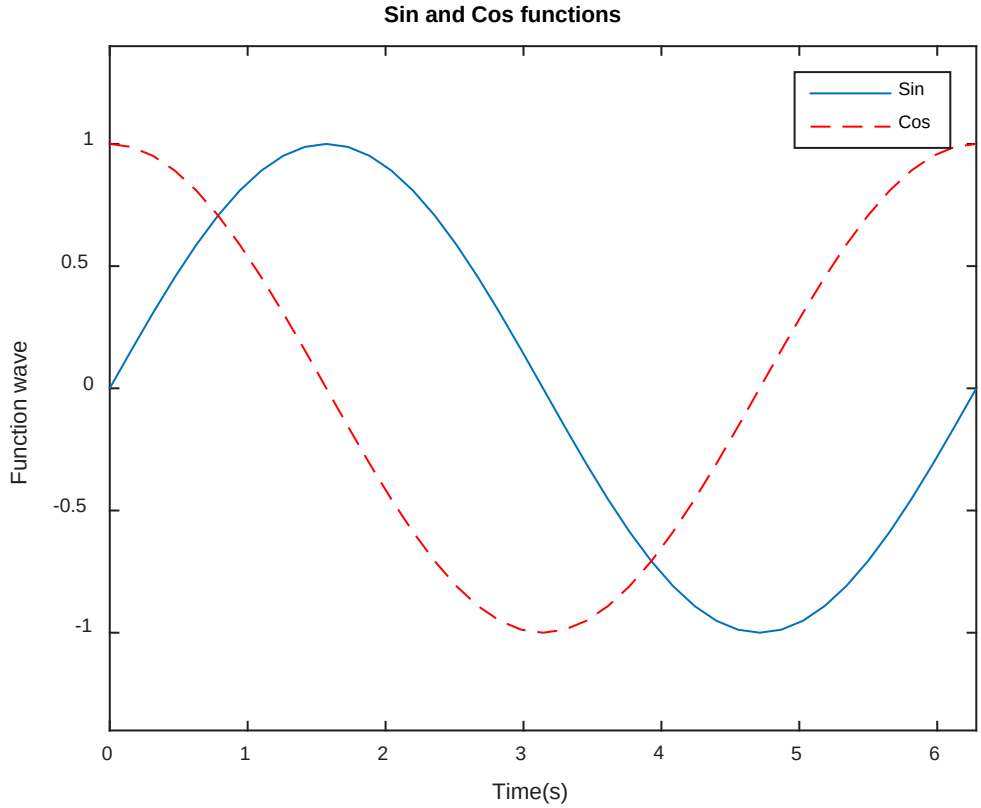
```
r =

    0.1419    0.4218    0.9157    0.7922    0.9595


r =

         0         0    0.9157    0.7922    0.9595
```

Tanto la función **find** como alguna otra de representación gráfica, no funcionaban. Al probar con versiones anteriores las mismas instrucciones descubrimos que daba fallos que creemos que podrían ser

por incompatibilidad de versiones. Por ello, dejamos de utilizar *Matlab R2016a*, sin embargo, este detalle supuso una pérdida de tiempo notable.

2 TWOLINEPLOT.M

Este script trata la representación de dos curvas en una misma figura, en este caso, un coseno y un seno. Para ello, una vez hemos definido una serie de muestras de t como para tener cierta precisión entre 0 y 2π (en nuestro caso hemos decidido tomar muestras en pasos de π/20), representamos una de las señales con la orden **plot**, por ejemplo, el seno. Para evitar que la siguiente representación no se muestre en esta figura, empleamos la orden **hold on,** que mantiene la figura a la espera de más información. Repetimos la orden **plot** para representar el coseno (esta vez con los modificadores *r* y – para que aparezca de color rojo rayada y diferenciarla fácilmente de la señal anterior). Para su mejor interpretación y lectura nos valemos de los comandos **title, xlabel** e **ylabel** que tienen como función dar nombre a la gráfica y los ejes, **legend** para dotar la gráfica de leyenda, y **xlim** e **ylim** para modificar los límites en los que se imprime la gráfica y mejorar la lectura. Finalizamos con un **hold off** para evitar interferir con otras gráficas (aunque en el caso de este script, no sería necesario).



3 SCRIPTS OPCIONALES

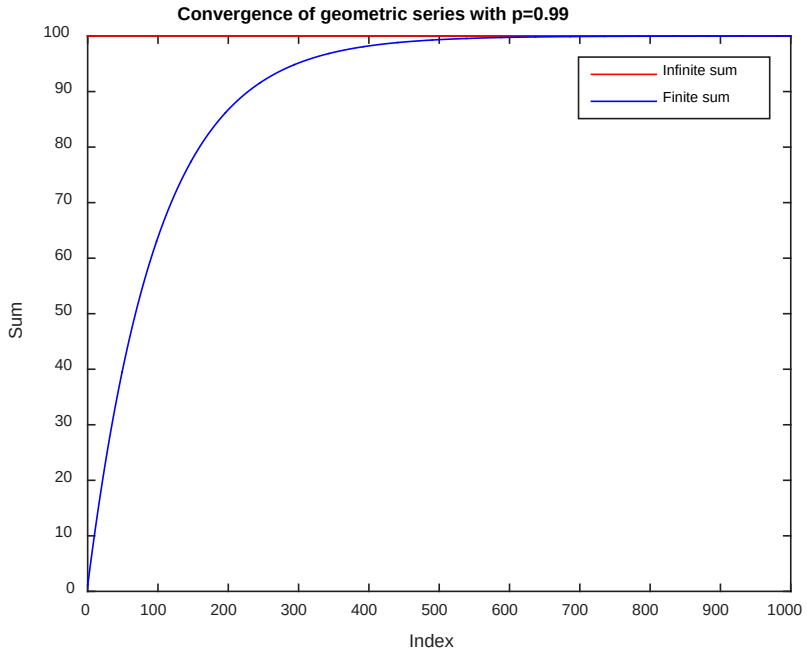
3.1 CALCULATEGRADES.M

La mayor dificultad que aporta esta práctica es la correcta compresión del enunciado en algún punto ya que puede llevar a puntos liosos. Por lo demás, cabe destacar el uso de **nanmean**, que no hace la media (de manera similar a **mean**) teniendo en cuenta los valores *NaN* como 0.

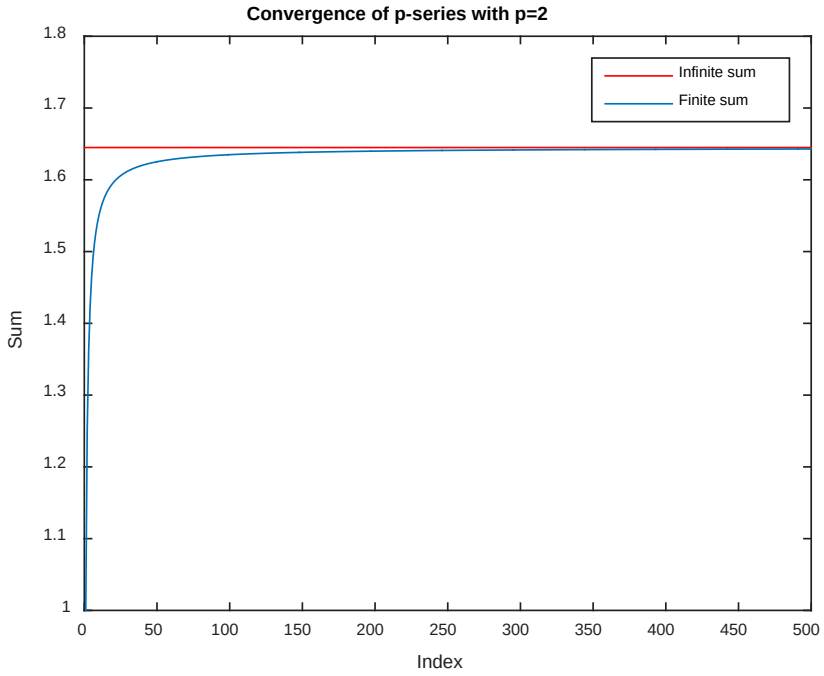
A la hora de imprimir el contenido se nos recomienda utilizar la función **disp.** Consultando el manual con la orden *help disp,* observamos que es una función para mostrar una matriz. El problema que apreciamos en esta función es que solo muestra una matriz, de manera que, si queremos concatenar una cadena de texto a una matriz, no solo tendríamos que utilizar dos líneas **disp,** sino que además introduciría un \n que no nos interesa en este caso en la impresión de datos. Atendiendo al manual, vemos otras funciones como **fprintf,** que sigue una sintaxis similar a la de lenguajes como C y consideramos que saca un resultado más adecuado.

3.2 SERIESCONVERGENCE.M

En este script trabajamos de forma similar al del script referido en el punto 2, *twoLinePlot.m*. Los apartados del *a* al *d* se basan en definiciones y operaciones básicas como las del script *shortProblems.m*. Puesto que se nos pide ver la tendencia de una suma geométrica $G = \sum_{k=0}^{\infty} p^k$ según aumentamos la muestra, utilizamos la función **sumcum** que nos permite realizar una suma acumulativa y poder así ver gráficamente como nos acercamos a una asíntota horizontal de valor G. Una vez tenemos el resultado devuelto por **sumcum**, metemos en la gráfica tanto esa función como una asíntota. Para la representación de la asíntota utilizamos *plot ([0 max(k)], G*ones(1, 2), 'r')*. En ese comando definimos los límites del eje de 0 al punto máximo del vector k y definimos dos puntos de la asíntota para formar la recta. Finalmente dotamos a la gráfica de leyenda y nombramos los ejes.



En cuanto a la suma geométrica $P = \sum_{n=1}^{\infty} \frac{1}{n^p}$ el procedimiento es similar. Puesto que las curvas interferían con la curva representada, nos valemos de **ylim** para cambiar los límites de representación del eje y, y, así poder ver mejor la representación.



3.3 THROWBALL.M

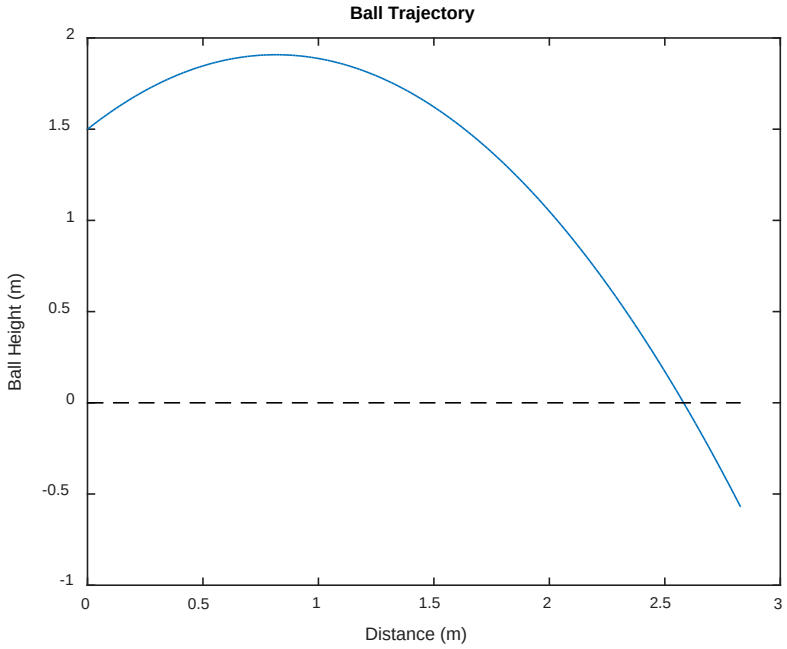
Una vez hemos definido las variables y expresado las ecuaciones necesarias para el cálculo de distancia y altura del lanzamiento parabólico de la pelota, según el enunciado, debemos buscar el punto en el que la altura empieza a ser negativa mediante la orden **find** y así obtener el punto en el que la bola impacta con el suelo. Sin atender a más que las propias instrucciones de la práctica, la distancia recorrida se obtendría con la siguiente sentencia: *x ((find (y<0, 1)))*. Sin embargo, analizando las expresiones, la distancia obtenida es correspondiente a una altura negativa (muy cercana a 0, *-2.9888e-04*) y por ello lo consideramos un resultado no real. Por tanto, vemos dos interpretaciones al respecto de este problema. Podemos coger el primer índice devuelto negativo por considerarlo suficientemente cercano a 0, siendo una distancia algo superior a la recorrida realmente, o podemos coger la anterior, con lo que tendríamos una distancia ligeramente inferior (La altura seleccionada seria de *3.1267e-04*). También podríamos aumentar el número total de muestras de t, pero, aunque nos aproximaríamos más al resultado real, incrementaríamos el tiempo de cálculo y nos enfrentaríamos al mismo problema. Finalmente, consideramos una mejor opción la semisuma de ambas, de manera que, además, obtenemos una solución genérica y trabajaríamos por tanto con una altura de *6.8929e-06.* Sabemos que al ser una función parabólica y solo hacer una aproximación lineal sería más exacto usar una estimación por derivadas, pero consideramos que el error es suficientemente pequeño para dar como aceptable el resultado, además entendemos que ese no es el objetivo del problema.

Teniendo en cuenta lo explicado anteriormente, nos queda la siguiente expresión:

```
fprintf('The ball hits the ground at a distance of %f meters\n',
((x((find(y<0, 1)))+x((find(y<0, 1))-1))/2))
```

Si utilizásemos *%d* obtendríamos el resultado en notación científica.

La gráfica se realiza de manera similar al del script *seriesConvergence.m*. Para que la asíntota fuera negra rayada el formato fue *'k-'*.



3.4 ENCRYPT.M

Tras formar la cadena de caracteres y saber su tamaño gracias a la orden **length** formamos el vector de números aleatorios de igual tamaño con la orden *key = randperm(length(original))*. Una vez que lo tenemos pasamos a codificarlo haciendo uso de *encoded = original(key),* de esta forma conseguimos rellenar las posiciones de *encoded* de manera consecutiva con la cadena original desordenada por *key.* También valoramos la opción de hacernos un vector auxiliar *vec = 1:length(original)* y codificar los caracteres del vector *encoded* de forma no consecutiva, dando saltos hacia adelante y hacia atrás, empleando la sentencia *encoded(key) = original(vec).* Descartamos este último método por considerarlo más largo y hacernos utilizar una variable más.

Posteriormente hacemos una matriz cuya primera fila es *key* y la segunda la rellenamos con números del 1 al tamaño del mensaje. Tras ello, la trasponemos para poder usar la orden **sortrows** para ordenar los valores de manera que consigamos un vector "traductor" para el mensaje codificado y almacenamos su segunda fila, nuestro vector de decodificación, que llamamos *iv*. Una vez hemos asignado el valor deseado a iv, eliminamos la variable temporal usando la orden **clear**.

Finalmente, para obtener nuestro mensaje indexamos **encoded** con el vector *iv*, y obtenemos el texto original. Tras ello, imprimimos los resultado y comprobamos que los procesos de codificado y decodificado se han realizado con éxito comparando las cadenas con la orden *strcmp(original, decoded)* que devuelve 1 en caso de que coincidan o 0 en caso de que las cadenas sean diferentes.