**SALIDA de DATOS , SCRIPTS, FUNCIONES y CONTROL de FLUJO**

**H. Volcado de datos formateados usando fprintf( )**

1. Volcar el valor del número pi con 5 decimales: fprintf('%.5f\n',pi);

Volcar el número e (e=exp(1)) con 2 decimales en una línea y 8 en la siguiente:

e=exp(1); fprintf('%.2f\n%.8f\n',e,e);

1. Sea la variable x = 174. Usar fprintf( ) para volcar una línea mostrando el valor de x como:

* un entero (%d),
* un entero reservando 4 columnas (%4d)
* un número real con 2 decimales (%.2f)
* un número real en notación científica (%e)

**>> fprintf('%d %4d %.2f %e\n',x,x,x,x,x)**

1. Sea la matriz A = [2 -15 3 127; -97 32 0 3]. Usar fprintf para visualizar sus contenidos con un solo comando:
2. En cuatro líneas, cada una mostrando una columna de A como números enteros.

**fprintf('%d %d\n',A)**

1. En dos líneas, cada una conteniendo una fila de la matriz como números enteros.

**fprintf('%d %d %d %d\n',A')**

1. Idem antes, pero reservando 4 columnas de espacio para cada número para verlos alineados. **fprintf('%4d %4d %4d %4d\n',A')**
2. Al hacer >>x=0.1 MATLAB por defecto crea una variable tipo double (doble precisión). A partir de x cread una variable xs convirtiendo x a precisión simple: xs=single(x);
3. Con **whos x xs**, ver el tamaño de ambas variables dentro de MATLAB. ¿Cuántos bytes ocupa una variable tipo double?¿Y single?
4. Con un solo comando fprintf volcar ambas variables en dos líneas distintas usando 15 decimales (formato %.15f). ¿Coinciden ambos valores? ¿Cuál es incorrecto?
5. Calcular el error relativo del valor en precisión simple (xs), comparándolo con el valor en precisión doble (x):

1. Repetir el apartado b) con 20 decimales. ¿Es correcto ahora alguno de los valores?

Escribid un script llamado ejer4.m con todos los comandos usados en este ejemplo.

Es **MUY RECOMENDABLE inicializar vuestros SCRIPTS con el comando clear;**

Un script puede ejecutarse desde el editor () o tecleando ejer4 (+ Enter) desde la ventana de comandos. Adjuntar código del script y respuestas a las preguntas.

1. Dado el vector x = (1:0.2:2), calcular sus logaritmos en un nuevo vector y=log(x) y usar un solo comando fprintf para generar la siguiente salida:

log(1.00) = 0.000

log(1.20) = 0.182

...

Pista: crear una matriz A cuya primera fila sea el vector x y la segunda el vector y.

Usar a continuación el comando fprintf('log(%.2f) = %.3f\n',A);

¿En qué orden recorre MATLAB una matriz al ir volcando todos sus componentes?

**I. Funciones**

1. Escribir una función que reciba como argumentos de entrada un número x y un entero k y devuelva el valor de cos(k·x):

**function p = fun1(x,k)**

**p= cos(k\*x);**

**return**

1. Escribir una función que reciba como argumentos de entrada un número x y un entero n y devuelva el valor de la n-ésima potencia de x (x^n). 2º argumento puede ser opcional. Si se omite el valor por defecto es n=2 (devolviendo el cuadrado de x).

**function p = potencia (x,n)**

**if nargin==1, n=2; end**

**p= x^n;**

**return**

1. Escribir una función llamada (cosn.m) que reciba dos argumentos x y n y devuelva el valor de la función (**la función coseno en MATLAB es cos() y la potencia es el operador ^).** Partir del siguiente template, guardándolo en un fichero cosn.m

**function res = cosn(x,n)**

**return**

Una vez completada, guardarla y probad la función desde la ventana de comandos haciendo **>> cosn(0.3,4)**. El resultado debería ser igual a hacer cos(0.3)^4.

Modificar la función para que siga funcionando si x es un vector en vez de un único número. **Pista: recordar el uso de los operadores punto a punto.**

Usar la función creada para hacer una gráfica superponiendo para n=5 y n=2 en el intervalo [-2,2]. Usar un espaciado "fino" (intervalo pequeño) para las x's para simular el aspecto "continuo" que esperamos en nuestras gráficas de funciones.

Adjuntar el código de vuestra función definitiva y la gráfica obtenida.

**J. BUCLES de CONTROL**

1. Sea la matriz 5x7 de números aleatorios entre 0 y 1 generada por **A=rand(5,7)**.
   1. Usar bucles anidados asignando a cada elemento de A el valor 0.2 si el valor original era menor/igual a 0.4 y 0.7 si era mayor que 0.4:

**for k=1:5, for j=1:7,**

**if A(k,j)<=0.4, A(k,j)=0.2; else A(k,j)=0.7; end**

**end; end**

* 1. Repetir usando indexado lógico: **A(A<=0.4)=0.2; A(A>0.4)=0.7;**

1. Utilizar la relación: para n=0,1,…

para estimar el valor del sin(x) en x=0.5, sumando los primeros 20 términos de la serie con los siguientes métodos:

1. Un script utilizando los comandos **.*\* ./* .^**

**N=30; n=[0:N-1];**

**x=0.5; s = ((-1).^n) .\* (x.^(2\*n+1)) ./ factorial(2\*n+1);**

**dif=sum(s)-sin(x), % Comprobación**

1. Un script utilizando bucles **for … end**.

**s = x; temp=x;**

**for n=1:29, temp = -temp\*x\*x/((2\*n+1)\*(2\*n)); s = s + temp; end**

**dif=s-sin(x)**

1. Un script con un bucle **while() … end**, con la condición de parar si el término a sumar es menor que 1e-18 (). Indicar el número de términos sumados.

**s = x; temp=x; n=1;**

**while(abs(temp)>1e-18),**

**temp = -temp\*x\*x/((2\*n+1)\*(2\*n)); s = s + temp; n=n+1;**

**end**

**dif=s-sin(x), n**

1. Escribir un script que calcule la suma de los inversos de los impares hasta 999:

1. Implementarlo con un bucle for end
2. Repetir sin emplear bucles (utilizando comandos punto a punto)
3. Usando tic, toc medir el tiempo en ambos casos.

Modificar el script anterior (en su versión sin bucles), convirtiéndolo en una función que reciba como argumento el índice máximo a sumar (999 en el ejemplo anterior).

1. Este algoritmo calcula una aproximación numérica (bastante rápida) al número :
   1. Inicializar las siguientes variables:
   2. Actualizar las 4 variables anteriores de la siguiente forma:
   3. Calcular la aproximación numérica de como:
   4. Si |a-b|>1e-15 (10-15) repetir pasos 2 y 3. En caso contrario terminar bucle.

1. Implementar el algoritmo anterior con un script usando un bucle **while**. En cada paso calcular el error absoluto de la estimación . Adjuntar script.

En MATLAB el valor "correcto" de  está almacenado en la variable pi.

1. En cada paso del bucle, mostrad (**fprintf**) el número de iteración (entero %d), el valor aproximado (15 decimales %.15f) y su error (%.1e). Cada línea os debe salir algo como esto: Iter 1: 3.1405792505221686 -> Error = 1.0e-03

Adjuntad vuestro volcado.

1. Con los resultados obtenidos completar la siguiente tabla

|  |  |  |
| --- | --- | --- |
| Iter | Error de la aproximación | Decimales OK |
|  |  |  |
|  |  |  |
|  |  |  |

1. Cambiad el while por un bucle for, corriendo 10 iteraciones sin aplicar ninguna condición de salida. ¿Mejora la estimación de ? ¿Por qué?
2. Usar la serie: para aproximar el valor de e (en MATLAB e=exp(1) ) sumando 51 términos de la sucesión de las 3 formas siguientes:
3. A partir de un vector inicial n=(0:50), utilizando los comandos punto a punto de MATLAB: **.*\* , ./ ,*** etc. y el comando sum para sumar los elementos de un vector. **Nota: en MATLAB n! = factorial(n).**
4. Mediante un bucle for … end con 50 iteraciones.

Bonus: tratar de evitar operaciones innecesarias (no llamar a la función factorial)

1. Repetir el bucle anterior pero añadiendo un comando break, de forma que el bucle termine si el término a sumar es menor que 10-18 (1e-18 en MATLAB).

¿Cuántos términos se han sumado? ¿Hay diferencias entre ambas soluciones?

Adjuntar vuestro código para las 3 opciones. Responder a las preguntas.

**K. ALGORITMOS VARIOS**

1. Escribir un script que cree un vector F con los primeros 20 números de Fibonnacci:

a) Crear un vector F de 20 elementos e inicializar sus dos primeras entradas con 1's.

b) Usando un bucle, rellenar las casillas desde k=3 hasta 20 de acuerdo a la regla de

qué cada número de Fibonacci es la suma de los 2 anteriores:

Una vez obtenido el vector F, hallar un vector C (de tamaño 19) con los cocientes

entre los sucesivos números de Fibonacci: F(2)/F(1), F(3)/F(2), …

Intentad hacerlo con un solo comando MATLAB (usando el operador ./)

Adjuntad vuestro código.

Hacer un plot del vector C usando una línea discontinua y marcando los datos con

puntos rojos aislados (modificador 'ro:' en el plot). Adjuntad gráfica resultante.

Observaréis que la sucesión C tiende a un límite s, que se puede demostrar que es:

s= (razón áurea).

Para verificarlo calculad la diferencia entre los términos de la sucesión C y el valor de s y haced

un plot del **valor absoluto de la diferencia** entre el vector C y el límite s. Veréis que

dicha diferencia tiende a cero. Adjuntad gráfica.

Observando la gráfica, ¿podéis decir cuál es el primer término de C que está a menos

de 10-6 del límite s?

Repetir la gráfica pero ahora usando un eje log en el eje Y (semilogy).

Adjuntad la nueva gráfica. ¿Podéis ahora responder mejor a la pregunta anterior?

1. Cread un vector x con valores entre ¼ y 4 a saltos de 10-5. Usando la función single() convertirlo a un nuevo vector xs en precisión simple. Calculad el error (xs-x) y haced un gráfico de dicha diferencia en función de x. Adjuntad la gráfica y justificarla.

Usando el cursor de datos de MATLAB () determinar la máxima diferencia en cada zona. Indicarlas ¿Qué relación existe entre ellas?

Seleccionar el zoom de Matlab () y escoger (click derecho) Zoom horizontal. Adjuntad la imagen ampliada donde se aprecie el comportamiento del error

1. Escribid una función mult\_rus.m que reciba dos argumentos a y b (números enteros) y devuelva su multiplicación.

**function P=mult\_rusa(a,b) % Template de la funcion**

**% Recibe dos enteros a y b y devuelve su producto en P**

**return**

Para la multiplicación, usaremos un viejo algoritmo que se enseñaba en Rusia, muy distinto del que aprendíamos por aquí:

El algoritmo es el siguiente:

1. Inicializar P = 0 (la variable donde guardaremos el producto)
2. Mientras que b no valga 0 (condición b~=0 en MATLAB), repetir los pasos 3 y 4
3. Si b es impar, sumar a P el valor de a. Usar  **if (condicion), comando/s; end**

para que el comando indicado (P=P+a) solo se ejecute si se cumple la condición de que b sea impar. Para comprobar si b es impar podéis usar la función mod (help mod). El módulo 2 de un número impar siempre será 1.

1. Al margen de si se cumple o no la condición anterior sobre b, multiplicar a por 2 y dividir b por 2 usando división entera sin decimales. Para esto podéis hacer **b=floor(b/2).** Guardar los resultados en a y b (machacando los viejos valores)

Comprobad el resultado de la función para a=1234, b=5678, comparándolo con el

producto a\*b.

% ADJUNTAR EL CODIGO de VUESTA FUNCION.

El siguiente código mide el tiempo necesario (tic arranca un reloj y toc nos da el tiempo transcurrido) para llevar a cabo 100000 multiplicaciones usando la función mult\_rus:

**>> tic, for k=1:100000, mult\_rusa(1234567890,2); end; toc**

**>> tic, for k=1:100000, mult\_rusa(2,1234567890); end; toc**

¿Qué tiempos obtenéis? ¿Por qué mult\_rusa(a,b) y mult\_rusa(b,a) no tardan lo mismo?

¿Cómo podríais optimizar la función para evitar este problema?

1. La función **rand** (llamada sin argumentos) genera un número aleatorio entre 0 y 1. Escribir un script (lo más sencillo posible) utilizando un **while**, para contar cuántas llamadas a la función **rand** se necesitan para obtener un número mayor que 0.999.

Adjuntar el código.

Ejecutar el código varias veces. ¿Se obtiene el mismo resultado? ¿Por qué?.

**ALGORITMICA NUMÉRICA Ejercicios Laboratorio 2 Respuestas de Clase**

**Apellidos, Nombre:**

**Apellidos, Nombre:**

**Apellidos, Nombre:**

Adjuntar las respuestas a los siguientes problemas, incorporando el código necesario, gráficas, comentarios pedidos, etc.

**H.4**

**I.3**

**Función cosn(x)**

function res=cosn(x,n)

res=cos(x).^n;

return

**código para representar la gráfica**

v=(-2:0.01:2);

x=cosn(v,5);

y=cosn(v,2);

plot(x);hold on; plot(y,'r');hold off;



**J.4**

**a.)**

**a=1;**

**b=1/sqrt(2);**

**x=1;**

**while abs(a-b)>1e-15**

**y=a;**

**a=(a+b)/2;**

**b=sqrt(b\*y);**

**t=t-x\*(y-a)²;**

**x=2\*x;**

**pinumber=(a+b)^2/(4\*t);**

**e= abs(pi-pinumber)**

**end**

**disp(pinumber)**

**b.)**

**a=1;**

**b=1/sqrt(2);**

**x=1;**

**iteration=1;**

**while abs(a-b)>1e-15**

**y=a;**

**a=(a+b)/2;**

**b=sqrt(b\*y);**

**t=t-x\*(y-a)²;**

**x=2\*x;**

**pinumber=(a+b)^2/(4\*t);**

**e= abs(pi-pinumber)**

**fprintf(‘Iteración: %d. %.15f %.1e \n’ , iteration,pinumber,e)**

**iteration = iteration + 1;**

**end**

**c.)**

|  |  |  |
| --- | --- | --- |
| Iteración | Error de la aproximación | Decimales OK |
| 1 | 0.0010 | 3 |
| 2 | 7.4e-09 | 8 |
| 3 | 8.9e-16 | 15 |
| 4 | 8.9e-16 | 15 |

**d.)**

**a=1;**

**b=1/sqrt(2);**

**x=1;**

**for i = 1:10**

**y=a;**

**a=(a+b)/2;**

**b=sqrt(b\*y);**

**t=t-x\*(y-a)²;**

**x=2\*x;**

**pinumber=(a+b)^2/(4\*t);**

**e= abs(pi-pinumber)**

**fprintf(‘Iteración: %d. %.15f %.1e \n’ , iteration,pinumber,e)**

**iteration = iteration + 1;**

**end**

**disp(pinumber)**

**La aproximación se mantiene exactamente igual por lo que la aproximación no es más precisa. Esto es porque a partir de la tercera iteración los valores de la aproximación son el mismo.**

**K.1**

Si os da tiempo:

**J.5**

**K.2**

**K.3**