Entrega 1

Ejercicio 1. Un número real se representa mediante una secuencia semi-infinita de dígitos. Un ordenador es una máquina de tamaño finito, por lo que su capacidad de representación es limitada. Por lo tanto, en un ordenador, los números reales se pueden representar únicamente de manera aproximada.
La capacidad de representación depende del entorno en el que se estén realizando los cálculos.

El objetivo de este ejercicio es determinar, realizando una búsqueda sistemática, el valor positivo más grande y el valor positivo (distinto de cero) más pequeño que se puede representar.

a. En Excel (sin utilizar Visual Basic)

b. En Matlab

Describir en detalle el procedimiento de búsqueda empleado.

Nota: Algunos procedimientos de búsqueda son más eficientes (es decir, encuentran la solución más rápido que otros). Se trata de definir el procedimiento más eficiente que garantice encontrar la solución.

```
clear
N=0;
%Da a escoger entre buscar mximo y mnimo
prompt= 'Insertar 1 si quiere valor positivo ms grande, 0 si ms pequeo representable ';
%Inicializa la variable de bsqueda
inflim=1;
choose=input(prompt);
if choose==1 %En este bucle delimito inferiormente el nmero ms alto representable
    while inflim ~= Inf
        N=N+1;
        inflim=10^(N+1);
    end
    n1=10^N;
%En estos dos bucles ajusto, descomponiendo el nmero en base decimal e
%iterando para diferentes i,j para encontrar el mximo para cada posicion
%N es la precisin a la que quiero obtener el nmero mximo
    for j=0:N
        for i=1:10 %recorre todos los valores de esta posicin
            n2=n1+i*10^(N-j);
            if n2~=Inf %si n2 es distinto de inf entonces salta a la siguiente iteraccin
                continue;
            else
                n1=n1+(i-1)*10^(N-j);%Hasta aqu llega si me paso, as que escojo el anterior
            end
            break;
        end
    end
    maximo=vpa(n1,10)
elseif choose==0 %Busco el minimo, en este caso no necesito afinar la busqueda como antes
    while inflim ~= 0
        N=N+1;
        inflim=10^-(N+1); %
    end
        n1=10^-N;
        minimo=vpa(n1,10)
else disp('Error');
end
```

Para explicar los resultados necesito utilizar el número $eps = 2^{-52}$, que es el error relativo de coma flotante, que es la mínima distancia que MatLab puede reconocer entre dos números Búsqueda del valor positivo más grande:

- MatLab: el código se dividirá en dos partes. En la primera parte, se busca el exponente de la potencia de base 10 que alcanza el límite sin superarlo, obteniendo una cota inferior del valor. Cuando el valor calculado supera el máximo representable MatLab muestra *Inf*, por lo tanto se ha utilizado un bucle tipo *while* que se ejecute hasta que el valor calculado sea mayor que el representable, y regrese el límite inferior, *n1*. En la segunda parte la idea es la misma, pero esta vez realizamos la descomposición en base 10 del supuesto valor límite, y aplicando un bucle observamos, para cada potencia (de valor exponencial, *N*, descendente), cuál es el valor límite que es capaz de almacenar para cada exponente.

  ```
  >> ejercicio_1
  Insertar 1 si quiere valor positivo más grande, 0 si más pequeño representable 1

  n1 =

  1.7977e+308
  ```

- Excel: En Excel la primera parte se realiza igual que en MatLab, con las diferencias derivadas de calcular el valor de la exponencial en las celdas en vez de utilizando bucles. En la segunda parte se crea una rejilla que, partiendo de 10^N, donde N es el exponente en base 10 límite, para cada fila se recorre una columna con rango $[1, 9]$. En estas celdas se calculan múltiplos del valor inicial, uno para cada celda e índice, por lo que se llegará a una celda en la que el valor sea mayor el máximo permitido y muestre *Inf*. De esta fila se escoge el último valor válido, y se toma como inicial para la siguiente fila. Este valor será una posición (10^{-1}) más preciso al más grande que puede mostrar Excel. Iterando se observa que la precisión para definir un valor de una celda en Excel es de 15 dígitos, por lo que el mayor valor posible de esta fila será el positivo más grande representable.

  ```
  Valor máximo=1,7977E+308
  ```

Estos límites se deben a la forma en que se almacenan los números en el programa. Siguiendo los estándares, se almacenan en formato de doble precisión, utilizando 64 bits. El primer bit se utiliza para definir el signo del número, 11 se utilizan para expresar el exponente (en base 2) y los 52 restantes para definir la mantisa. Por lo tanto, como ambos programas utilizan el formato double para almacenar, el valor máximo representable es $1.99999 * 2^{1023}$, ya que 1023 es el máximo exponente representable y 1.999999... la mayor mantisa obtenible utilizando este formato Búsqueda del valor positivo más pequeño:

- MatLab: Hallar este número es más simple que antes, ya que MatLab en cuanto llegas a un número positivo menor del mínimo admitido devuelve a uno similar.

  ```
  >> ejercicio_1
  Insertar 1 si quiere valor positivo más grande, 0 si más pequeño representable 0

  minimo =

  9.881312917e-324
  ```

  Este número puede parecer erróneo, pues sabemos que

  ```
  >> realmin

  ans =

  2.2251e-308
  ```

  pero esta diferencia se debe al error de coma flotante, ya que hemos obtenido el mínimo mediante fórmulas, operando vemos que:

  ```
  >> realmin-minimo/(eps*2)

  ans =

  0.0
  ```

- Excel: al igual que en Matlab, el mínimo positivo es más fácil de obtener. Se obtiene el valor (ya que se ha llegado a él mediante fórmulas):

  `Valor mínimo=2,2251E-308`

Se puede observar que tanto MatLab como Excel tienen los mismos valores máximo y mínimo (positivo) representable, ya que este valor está limitado por el formato de representación del número (doble), y no por el software interno.

Ejercicio 2. En matemáticas, la expresión $(1.0 + a_1) - 1.0$ es distinta de cero para cualquier valor real de a_1 distinto de cero. Sin embargo en Matlab o en Excel (y en todos los cálculos numéricos con ordenador)esto no es cierto para valores de a_1 suficientemente pequeños.

Determinar, realizando una búsqueda sistemática, el valor positivo más pequeño de a_1 tal que $(1+a_1)$ sea distinto de cero.

a. En Excel (sin utilizar Visual Basic,sólo fórmulas de Exel)

b. En Matlab

¿Son los valores de a1 encontrados en Excel y en Matlab iguales?

¿Son los dos entornos igualmente fiables?

Repetir el ejercicio para encontrar el valor positivo más pequeño a_{1024} tal que $(1024.0+a_{1024})$-1024.0 se distinto de cero.

Repetir el ejercicio para encontrar el valor positivo más pequeño a_{1024} tal que $(0.0625+a_{0.0625})$-0.0625 se distinto de cero.

¿Hay alguna relación entre los valores encontrados para $a_{0.0625}$,a_{1024} ? Propón alguna explicación para la anomalía.

En este caso utilizamos un algoritmo similar al ejercicio 1, pero recorro el bucle multiplicativo en orden inverso puesto que el objetivo es encontrar el valor mínimo y he utilizado la condición de que la diferencia sea distinta de cero. En Excel he aplicado un procedimiento similar a antes. Debido a la forma de operar que tiene este programa hay que aplicar un algoritmo más simple donde solo es necesario calcular una iteración una vez encontrada la potencia en la que aparece.

```
N=0;
valor=1;%este es el valor que meter en la funcion para calcular la
    diferencia
inflim=1;%valor inicial
while inflim ~= 0 %aqu acoto por arriba la potencia N
    N=N+1;
    inflim=(valor+10^-(N+1))-valor;
end
n1=10^-(N+1);
%En estos dos bucles ajusto, descomponiendo el nmero en base decimal e
%iterando para diferentes i,j para encontrar el mximo para cada posicion
%N es la precisin a la que quiero obtener el nmero mximo
for j=0:320
    for i=9:-1:0 %recorre todos los valores de esta posicin
        n2=n1+i*10^-(N+j);%ajusto
        dif=(valor+n2)-valor;
        if dif~=0 %si dif es distinto de 0 entonces salta a la siguiente
            iteracin
            continue;
        else
            n1=n1+i*10^-(N+j);%Hasta aqu llega si me paso, as que escojo
                el anterior
        end
        break;
    end
end
limite=n1
```

Los resultados obtenidos son:

- Para a_1

  ```
  >> ejercicio_2

  limite =

  1.1102e-16
  ```

  Este número está relacionado con el error de coma flotante, la mínima distancia que Matlab puede reconocer entre dos puntos. Este límite varía en función de la operación realizada, como se puede ver en estos ejemplos. En este caso, $a_1 = \frac{eps}{2}$

- Para a_{1024}

  ```
  limite =

  1.1369e-13
  ```

  Ahora la operación tiene valores más grandes, por lo tanto el valor de a_{1024} va a tener menos cifras significativas, exactamente $a_{1024} = a_1 * 1024 = eps \cdot 512$

- Para $a_{0.0625}$

  ```
  limite =

  6.9389e-18
  ```

  Es menor que 1, por lo tanto incluye más cifras significativas en los decimales, $a_{0.0625} = \frac{a_1}{16} = \frac{eps}{32}$ (16 es el inverso de 0,0625)

Ejercicio 3. Escribir una función Matlab que calcule el factorial de un número entero n

$$n! = 1 \cdot 2 \cdot 3 \cdot K \cdot n = \prod_{i=1}^{n} i$$

```
function [f]=miFactorial(n)
%miFactorial: calcula el factorial de un nmero entero n
%
% SINTAXIS: f = miFactorial(n)
%
% n : Entero no negativo
% f : Valor del factorial de n
% EJEMPLO:
% miFactorial(5)
%
if n==0
    f=1;
elseif n>0 && mod(n,1)==0
    for i=1:n
        vector(1,i)=i;
    end
    f=prod(vector)
else disp('Nmero no vlido, introduzca un entero mayor o igual a 0')
end
```

En este ejercicio se calcula el factorial de un número n positivo creando un vector de dimensión $1xn$ donde en cada elemento se almacena la posición en la que se encuentra. Posteriormente se realiza el producto de todos los elementos del vector, obteniendo el valor de n.
En Excel se obtienen los valores:

```
a1=1,77636E-15
a1024=1,81899E-12
a0.0625=01,38777E-17
```

Se puede observar que en Excel la precisión es menor, ya que para su valor correspondiente a_i su valor es siempre mayor que en MatLab. Se puede observar que la diferencia en el valor de a es de alrededor de 10^{-1}, por lo tanto el entorno de Excel es menos fiable que el de MatLab. Este mayor error puede deberse a la precisión que tienen ambos programas a la hora de almacenar los datos.

4. Utilizando la función del factorial definida en el ejercicio anterior, se puede escribir una función para calcular un número combinatorio

$$\binom{m}{n} = \frac{m!}{n!\,(m-n)!}$$

Consideremos la función, almacenada en el fichero miCombinatorio.m
Comprueba que funciona correctamente para calcular, por ejemplo 120

$$\binom{10}{3} = \frac{10 \cdot 9 \cdot 8}{3 \cdot 2 \cdot 1} = 120$$

miCombinatorio (10, 3)

Intenta calcular con dicha función los números combinatorios $\binom{400}{200}$, $\binom{400}{399}$, $\binom{400}{1}$.
Propón una modificación del código que permita calcular estos números combinatorios.
Compara tu solución con la que propone Matlab. Puedes verla editando el fichero nchoosek.m desde la interfaz de línea de instrucciones de Matlab.

```
function f = miCombinatorio(m,n)
%miCombinatorio: calcula el factorial de un nmero entero n
%
% SINTAXIS: f = miCombinatorio(m,n)
%
% m,n : Enteros no negativos con m >= n
% f : Valor del nmero combinatorio m sobre n
% EJEMPLO:
% miCombinatorio(10,3)

pascalt(1,1)=1;%Valor de la primera fila
pascalt(2,1:2)=[1,1];%Valor de segunda fila
%Usar la propiedad de que el valor de un nmero combinatorio aparece en el
    tringulo de Pascal
if m<n || m<0 || n<0
    disp('Error, m tiene que ser mayor o igual a n, y m,n mayor o igual a 0
        ');
    f=0
else
    for columna=3:m+1
    pascalt(columna,1)=1;%primer valor siempre es 1
    for fila=2:columna-1
        pascalt(columna,fila)=pascalt(columna-1,fila-1)+pascalt(columna-1,
            fila);
        %Cada elemento es la suma de los dos elementos encima suyo (encima
        %e izquierda)
    end
    pascalt(columna,columna)=1;%el ltimo elemento de cada fila es siempre 1
end
f=pascalt(m+1,n+1) %el combinatorio buscado est ah porque he comenzado
                   %definiendo el combinatorio (0,0)

```

```
30 end
```

Si empleamos la función proporcionada por el ejercicio, se comprueba que esta funciona perfectamente para $\binom{10}{3}$, pero no para los siguientes números combinatorios que se nos piden.
En estos casos, el resultado que se obtiene es NaN, abreviatura de Not a Number, que es como Matlab representa a los resultados que no están definidos como números. Esto se debe a que no puede almacenar valores mayores que el hallado en el apartado 1, y al calcular la división de factoriales estamos calculando los factoriales por separado, y si cualquiera de estos factoriales es mayor que el valor máximo aceptado MatLab no es capaz de almacenarlos y muestra un error.
Este problema puede subsanarse de dos formas. Una de ellas sería realizar el paso:

$$\frac{m!}{n! \cdot (m-n)!} = \frac{prod(m-n+1:m)}{prod(1:m-n)}$$

y posteriormente descomponer el cociente en pequeñas divisiones elemento a elemento de los vectores.
Otra opción es calcular el triángulo de Pascal, donde cada elemento del triángulo representa un número combinatorio diferente. La primera fila solo tiene un valor, 1; la segunda dos valores , ambos 1; y los valores de los elementos de las filas siguientes se construyen a partir de la suma de los dos elementos contiguos de la fila superior, como se muestra en el siguiente esquema

$$\begin{array}{ccccccc} 1 & & & & & & \binom{0}{0} \\ 1 & 1 & & & & & \binom{1}{0} \quad \binom{1}{1} \\ 1 & 2 & 1 & & \rightarrow & & \binom{2}{0} \quad \binom{2}{1} \quad \binom{2}{2} \\ 1 & 3 & 3 & 1 & & & \binom{3}{0} \quad \binom{3}{1} \quad \binom{3}{2} \quad \binom{3}{3} \\ 1 & 4 & 6 & 4 & 1 & & \binom{4}{0} \quad \binom{4}{1} \quad \binom{4}{2} \quad \binom{4}{3} \quad \binom{4}{4} \end{array}$$

Como se puede observar, aplicando correctamente el algoritmo se puede obtener cualquier número combinatorio a partir solo de sumas, por lo que no estaremos limitados por productos de números sino por sumas, permitiendo alcanzar una combinación con un número de elementos m mayor.
Los resultados obtenidos empleando esta función, para el cálculo de los anteriores números combinatorios son los siguientes:

$$\binom{400}{1} = 400 \quad ; \quad \binom{400}{200} = 1.0295e+119 \quad ; \quad \binom{400}{399} = 400$$

Que son los mismos resultados que si empleamos la función *nchoosek(v,k)*.

Ejercicio 5. Se conoce el valor de la serie $S = \sum_{i=1}^{\infty} \frac{1}{n^4} = \frac{\pi^4}{90}$

Vamos a intentar estimar su valor de manera aproximada con la ayuda del ordenador. Para ello truncamos la serie para un valor de N grande, pero finito

$$S = \sum_{i=1}^{\infty} \frac{1}{n^4};\ err_{abs}^N = |S_N - S|;\ err_{rel}^N = |\frac{S_N - S}{S}|$$

$$S = lim_{N\to\infty} S_N \equiv S_\infty$$

Dos implementaciones distintas para calcular S_N

IMPLEMENTACIÓN 1: Suma hacia delante

$$S_{\vec{N}} = \left(\left(\left(1 + \frac{1}{2^4}\right) + \frac{1}{3^4}\right) + ... + \frac{1}{N^4}\right)$$

IMPLEMENTACIÓN 2: Suma hacia atrás

$$S_{\vec{N}} = \left(\left(\left(\frac{1}{N^4} + \frac{1}{(N-1)^4}\right) + \frac{1}{(N-2)^4}\right) + ... + \frac{1}{2^4}\right) + 1$$

Utilizando un razonamiento matemático basado en el signo de los términos que estamos despreciando, ¿debería disminuir el error a medida que N se hace mayor? ¿Son los resultados par $S_{\vec{N}}$ y $S_{\vec{N}}$ iguales? ¿Disminuye en la práctica?

Ambas implementaciones descomponen el sumatorio en sus términos, pero la primera comienza el bucle

	1 sumaA	2 err_abs_A	3 err_rel_SA	4 sumaD	5 err_abs_D	6 err_rel_D
1	1	0	0	1	0	0
2	1.0625	0.0198	0.0183	1.0625	0.0198	0.0183
3	1.0748	0.0075	0.0069	1.0748	0.0075	0.0069
4	1.0788	0.0036	0.0033	1.0788	0.0036	0.0033
5	1.0804	0.0020	0.0018	1.0804	0.0020	0.0018
6	1.0811	0.0012	0.0011	1.0811	0.0012	0.0011
7	1.0815	7.8321e-04	7.2363e-04	1.0815	7.8321e-04	7.2363e-04
8	1.0818	5.3907e-04	4.9806e-04	1.0818	5.3907e-04	4.9806e-04
9	1.0819	3.8665e-04	3.5724e-04	1.0819	3.8665e-04	3.5724e-04
10	1.0820	2.8665e-04	2.6485e-04	1.0820	2.8665e-04	2.6485e-04
11	1.0821	2.1835e-04	2.0174e-04	1.0821	2.1835e-04	2.0174e-04
12	1.0822	1.7012e-04	1.5718e-04	1.0822	1.7012e-04	1.5718e-04
13	1.0822	1.3511e-04	1.2483e-04	1.0822	1.3511e-04	1.2483e-04
14	1.0822	1.0908e-04	1.0078e-04	1.0822	1.0908e-04	1.0078e-04
15	1.0822	8.9327e-05	8.2533e-05	1.0822	8.9327e-05	8.2533e-05
16	1.0822	7.4068e-05	6.8434e-05	1.0822	7.4068e-05	6.8434e-05
17	1.0823	6.2095e-05	5.7372e-05	1.0823	6.2095e-05	5.7372e-05

sumando desde el valor 1 del sumatorio y la segunda comienza desde el valor N. Evidentemente al añadir más términos al sumatorio se reducirá el error, ya que hay una menor variación respecto al sumatorio inicial S_N. En la implementación de MatLab sí aumenta la precisión según aumenta N, aunque evidentemente cuanto más avanza menor es la variación de error entre N y $N+1$, ya que cada vez es menor el nuevo término que se añade.

La suma de S_N hacia delante y hacia atrás son en principio iguales, pero cuando comienza a aparecer el error de medida son ligera diferentes (del orde de un múltiplo pequeño de *eps*).
Los resultados muestran que el N óptimo para la suma hacia delante es 9742 y para la suma hacia detrás es 8091. Esto significa que la suma hacia delante es más exacta, ya que tarda más en converger al punto en el que dos valores consecutivos son iguales para la precisión que aplica MatLab al cálculo. Esto se debe a que cuando se calcula la suma hacia atrás se comienza a arrastrar el error antes en la suma, ya que en esta suma es el primer término el que comienza a mostrar error en la precisión y arrastrarlo. Sin embargo, para la suma hacia delante los primeros N términos (aprox 8185) no muestran error de precisión, y por tanto comienza a arrastrarse mucho más tarde que en la suma hacia atrás, tardando más tiempo en converger. El N óptimo se ha conseguido aplicando un algoritmo de control similar a los utilizados en ejercicios anteriores.

```
N=10000;
%crea la tabla a partir de las dos funciones definidas anteriormente
[s11,s12,s13]=sumaDelante(N);
[s21,s22,s23]=sumaAtras(N);
array=[s11,s12,s13,s21,s22,s23];
tabla=array2table(array,'VariableNames', {'sumaA','err_abs_A','err_rel_SA',
    'sumaD','err_abs_D','err_rel_D'})
sum=0;


function [S_N,err_abs,err_rel]=sumaDelante(N)
S=pi^4/90
S_N(1,1)=1%valor inicial
for i=2:N
```

```
14      suma=0;
15      for j=1:i%realiza la suma desde 1 hasta i
16          suma=suma+1/j^4;
17      end
18      S_N(i,1)=suma;%calcula suma y errores absolutos y relativos
19      err_abs(i,1)=abs(S_N(i,1)-S);
20      err_rel(i,1)=abs((S_N(i,1)-S)/S);
21      %Para obtener el N ptimo, cuando dos seguidos sean iguales para
22      if S_N(i,1)==S_N(i-1,1)
23          break;
24      else continue;
25      end
26  end
27  end
28
29  function [S_N,err_abs,err_rel]=sumaAtras(N)
30  S=pi^4/90;
31  S_N(1,1)=1;%valor inicial
32  for i=2:N
33      suma=0;
34      for j=i:-1:1
35            suma=suma+1/j^4;
36      end
37      S_N(i,1)=suma;%realiza la suma desde 1 hasta i
38      err_abs(i,1)=abs(S_N(i,1)-S);
39      err_rel(i,1)=abs((S_N(i,1)-S)/S);
40      %Para obtener el N ptimo, cuando dos seguidos sean iguales para
41      if S_N(i,1)==S_N(i-1,1)
42          break;
43      else continue;
44      end
45  end
46
47  end
```

Ejercicio 6. Explicad qué se está calculando en la última instrucción Matlab

(i) ¿Varía el valor calculado cuando modificamos los valores de mu y sigma?

(ii) ¿Por qué? Explicadlo con una demostración matemática.

(iii) ¿Cuál es el mínimo valor de α para que la cantidad calculada numéricamente sea lo más próxima a 1.0 posible?

```
1  mu = 1.35;
2  sigma = 0.25;
3  alpha = 2.0;
4  f = @(x)normpdf(x,mu,sigma);
5
6  % se construye la funcin de densidad de una variable aleatoria normal de
       media 1,35 y desviacin tpica 0,25, llamndola f
7
8  x_inf = mu-alpha*sigma;
9  x_sup = mu+alpha*sigma;
10
11 % se establecen unos valores (que van a ser los lmites de integracin)
       alrededor de la media: 1,35 ± 0,25*2.
12
13 TOL_ABS = 1.0e-6;
14
```

```
15 % Se define tambin un error admitido para el clculo de la integral de
       0,000001
16 quadl(f,x_inf,x_sup,TOL_ABS)
17
18 %Se calcula la integral con la funcin quadl de la funcin de densidad entre
       0,85 y 1,85.
```

Lo que estamos calculando (con el parámetro alfa en 2) es la probabilidad de que una realización de la variable aleatoria se encuentre en un entorno de dos desviaciones típicas respecto de la media. Dicha probabilidad es de aproximadamente 0,95 para una variable que siga una distribución normal.
La función normpdf devuelve el valor de la función de densidad de una variable que sigue una distribución norma de parámetros mu y sigma, evaluada en el punto x. Con el comando @(x) definimos una función anónima asociada a dicha función de densidad. La función quadl nos permite calcular el valor de la integral definida de la función f entre los limites de integración xinf y sup, con un error admitido determinado. El método utilizado para el cálculo es la cuadratura recursiva adaptativa de Lobatto.
(i) El valor calculado es, para cualquier μ, σ:

```
>> ejercicio_6

ans =

0.9545
```

Por lo tanto no varía con los parámetros de la normal, solo con *alpha*.

(ii) Para entenderlo es necesario recordar que el área debajo de una curva (definida por la función de distribución) en un intervalo es el valor de la integral en ese intervalo. Al ser una función de densidad, sabemos que está normalizada:

$$\int_{-\alpha}^{\alpha} \frac{1}{\sqrt{2\pi}} \exp -\frac{(x-\mu)^2}{2\sigma^2} dx$$

Sea $X \sim N(\mu, \sigma^2)$ la variable aleatoria, la integral en el intervalo $[x_{inf}, x_{sup}]$ se puede definir como:

$$\int_{x_{inf}}^{x_{sup}} f(x)dx = P[X > x_{inf}] - P[X > x_{sup}] = P[x_{inf} < X < x_{sup}]$$

Sustituyendo los límites por su definición y despejando,

$$P[\mu - \alpha \cdot \sigma < X < \mu + \alpha \cdot \sigma] = P[-\alpha \cdot \sigma < X - \mu < \alpha \cdot \sigma] = P\left[-\alpha < \frac{X-\mu}{\sigma} < \alpha \right]$$

Recordando la estandarización de la función de distribución normal,

$$P\left[-\alpha < \frac{X-\mu}{\sigma} < \alpha \right] = P(-\alpha < Z < \alpha) \qquad ; \qquad Z \sim N(0,1)$$

Esta probabilidad se puede escribir de la forma

$$P(-\alpha < Z < \alpha) = \int_{-\alpha}^{\alpha} \frac{1}{\sqrt{2\pi}} \exp -\frac{z^2}{2} dz$$

Demostrando, mediante una transformación, que la integral no depende de los parámetros μ, σ y solo depende de α.
(iii) *quadl* realiza una estimación de la integral entre dos puntos. Este método es una versión de la cuadratura gaussiana, por lo tanto puede devolver diferentes resultados en función del nivel de tolerancia al que se ajuste el método.
En este caso, el nivel de tolerancia escogido es $TOL_{abs} = 10^{-16}$. El código consiste en un bucle *while* donde se comprueba, para $\Delta\alpha = 1$ cual es valor anterior máximo de α que mantiene la integral diferente de 1.
Después mediante una implementación similar a la del ejercicio 1, descomponiendo en potencias de base 10, aproximo en cada iteración de la variable j un dígito el valor de α mínimo para el cual la función devuelvo 1. El valor final es:

```
>> ejercicio_6_3

ans =

8.30000002113
```

```
clear
mu = 1;
sigma = 1;
alpha=1;
f = @(x)normpdf(x,mu,sigma);
%
x_inf = mu-alpha*sigma;
x_sup = mu+alpha*sigma;
%
TOL_ABS = 1.0e-16;
f1=0;
f2=1;
while f1 ~= f2
    alpha=alpha+1;
    x_inf = mu-alpha*sigma;
    x_sup = mu+alpha*sigma;
    f1=quadl(f,x_inf,x_sup,TOL_ABS);
    f2=quadl(f,x_inf-alpha*sigma,x_sup+alpha*sigma,TOL_ABS);
end
alpha=alpha-2;
    %N es la precisin a la que quiero obtener el nmero mximo
    for j=1:20
        for i=1:10 %recorre todos los valores de esta posicin
            test=alpha+i*10^-j;
            if quadl(f,mu-test*sigma,mu+test*sigma,TOL_ABS)~=1
                continue;
            else
                alpha=alpha+(i-1)*10^-j;
            end
            break;
        end
    end
    vpa(alpha,16)
```