

# 1 Calcula los siguientes valores con dos decimales y calcula el error absoluto y el error relativo.

El error absoluto es la diferencia entre el valor real y el obtenido:

$$|p-q|$$

El error relativo es la diferencia entre el valor real y el obtenido, representado en forma de porcentaje:  $|p-q| / |p|$  suponiendo que  $p \neq 0$

Truncamos los decimales con la siguiente función, ya que

fpprintprec se limita a la hora de salida por pantalla:

```
truncamiento(x,n):=float((round(x·10^n)/10^n))$ /*Uso: redondea el número x a n deci
```

## 1.1 %e

```
p: float(%e);
q: truncamiento(p,2);
err_abs: abs(p-q);
err_rel: (err_abs/abs(p))·100$;
fpprintprec: 3$;
print(err_rel,"%")$
```

2.718281828459045  
2.72  
0.001718171540955105  
0.0632 %

## 1.2 %pi

```
p: float(%pi);
q: truncamiento(p,2);
err_abs: abs(p-q);
err_rel: (err_abs/abs(p))·100$;
fpprintprec: 3$;
print(err_rel,"%")$
```

3.14  
3.14  
0.00159  
0.0507 %

## 1.3 sqrt(123)

```
p: float(sqrt(123));
q: truncamiento(p,2);
err_abs: abs(p-q);
err_rel: (err_abs/abs(p))·100$;
fpprintprec: 3$;
print(err_rel,"%")$
```

11.1  
5.37  $10^{-4}$   
0.00484 %

## 1.4 $4.3^{8.688}$

```
p: float(4.3^8.688);
q: truncamiento(p,2);
err_abs: abs(p-q);
err_rel: (err_abs/abs(p))·100$ 
fpprintprec: 3$ 
print(err_rel,"%")$
```

3.19  $10^5$   
3.19  $10^5$   
0.00187  
 $5.88 \cdot 10^{-7}$  %  
fpprintprec: 16\$

- 2 *Usa el método de bisección para encontrar una solución de las siguientes ecuaciones con cinco decimales exactos:*

La función del método de bisección es la siguiente:

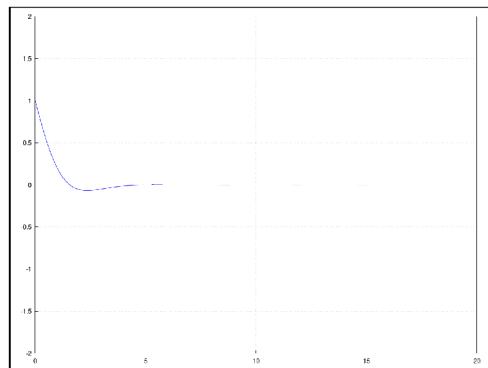
```
biseccion(expr,var,ext_inf,ext_sup):=
block(
    [a,b,c, /* extremos y punto medio */
     fa,fb,fc, /* valores de la función en dichos puntos */
     contador:0, /* número de repeticiones */
     tolx:10^(-5),tolfx:10^(-5) /* error permitido */
    ],
    numer:true,
    local(f),
    define(f(x),subst(x,var,expr)),
    a:float(min(ext_inf,ext_sup)),
    b:float(max(ext_inf,ext_sup)),
    c:(a+b)*0.5,
    fa:f(a),
    fb:f(b),
    fc:f(c),
    if abs(fc)<tolfx then return([c,1,(b-a)*0.5,fc]),
    if sign(fa)=sign(fb) then error("la función no cambia de signo en los extremos"),
    while ((b-a)/2 >tolx and abs(fc)>tolfx)
        do(
            contador:contador+1,
            c:float((a+b)/2),
            fc:f(c),
            if abs(fc)<tolfx then return(),
            if sign(fa) = sign(fc) then (a:c,fa:fc) else (b:c,fb:fc)
            ),
            [c,contador,(b-a)*0.5,f(c)]
    )$
```

## 2.1 $\exp(-x) * \cos(x)=0$ , $x$ in $[0,20]$ .

En todos los ejercicios vamos a graficar primero, para ver si los rangos que nos piden son correctos:

```
expr: exp(-x)*cos(x);
      %e^-x cos(x)

wxdraw2d(
    color= blue,
    explicit(expr, x, 0, 20),
    yrange= [-2,2],
    yaxis= true,
    xaxis= true,
    grid= true
);
```



En este caso el rango no es adecuado: en 0 y 20 la función es positiva. Vamos a tomar 3 como extremo superior, pues parece que es negativo.

**biseccion(expr,x,0,3);**

[ $1.570770263671875, 15, 9.1552734375 \cdot 10^{-5},$   
 $5.418132183104021 \cdot 10^{-6}$ ]

La solución es 1.57, y ha tomado 18 iteraciones alcanzarla. Podemos resolver la ecuación para comprobar si es cierto:

**find\_root(expr, x, 0, 3);**

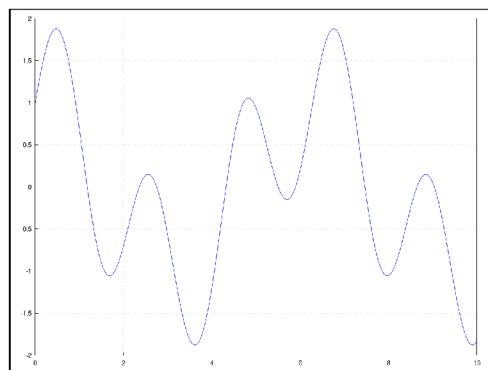
1.570796326794897

Vamos a mirar el error relativo:

**err\_rel: ( $\text{abs}(1.570796326794897 - 1.570804595947266)/\text{abs}(1.570796326794897)$ )**  
 $5.264305898868279 \cdot 10^{-4}$

## 2.2 $\cos(x)+\sin(3x)=0$ , $x$ in [0,10].

```
expr: cos(x)+sin(3·x);
      sin(3 x)+cos(x)
wxdraw2d(
    color= blue,
    explicit(expr, x, 0, 10),
    yrange= [-2,2],
    yaxis= true,
    xaxis= true,
    grid= true
);
```



En este caso el rango es adecuado, pero solo obtendríamos una solución. Tenemos que dividirlo en 9 rangos, uno por cada vez que la función hace 0. Estos serán:  
 $[0,2],[2,2.5],[2.5,3],[3,5],[5,5.5],[5.5,6],[6,8],[8,9],[9,10]$ .

Hemos creado una lista para los extremos inferiores y otra para los superiores y, con un bucle for, calculamos cada una de las bisecciones.

```
lista_x: [0,2,2.5,3,5,5.5,6,8,9,10];
lista_y: [2,2.5,3,5,5.5,6,8,9,10];
[0,2,2.5,3,5,5.5,6,8,9,10]
[2,2.5,3,5,5.5,6,8,9,10]
for i:1 thru 9 do(print(biseccion(expr, x, lista_x[i], lista_y[i])));
[1.178085327148438, 17, 7.62939453125 10-6,
4.404320934175976 10-5]
[2.356201171875, 11, 2.44140625 10-4, 9.449199954603138
10-6]
[2.748886108398438, 15, 7.62939453125 10-6,
1.142441405588279 10-5]
[4.319686889648438, 17, 7.62939453125 10-6, -
1.111996659908243 10-5]
[5.497787475585938, 15, 1.52587890625 10-5, -
4.692411216256076 10-7]
[5.890487670898438, 15, 1.52587890625 10-5,
2.212557156267003 10-6]
[7.461288452148438, 17, 7.62939453125 10-6, -
2.180303316517973 10-5]
[8.639373779296875, 15, 3.0517578125 10-5, -]
```

```

8.510945801343262 10-6 ]
[9.032073974609375, 15, 3.0517578125 10-5,
7.507335415768424 10-6 ]
done

```

## 2.3 $\log(x^2+1)-\cos(x)$ , $x$ in $[0,3]$ .

```

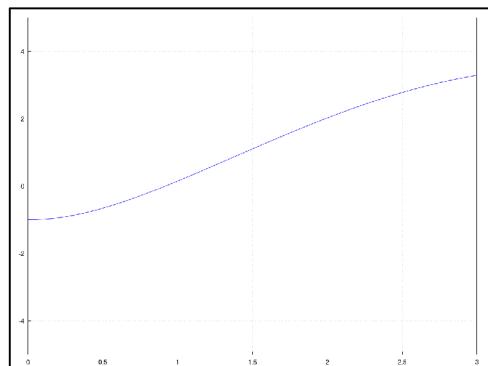
expr: log((x^2)+1)-cos(x);
      log (x2 + 1) - cos (x)

```

```

wxdraw2d(
    color= blue,
    explicit(expr, x, 0, 3),
    yrange= [-5,5],
    yaxis= true,
    xaxis= true,
    grid= true
);

```



Esta la podemos resolver directamente, ya que el rango es correcto y en él solo se corta una vez el eje x.

```

biseccion(expr, x, 0, 3);
[0.9158592224121094, 18, 1.1444091796875 10-5,
2.79708948480728 10-6 ]
find_root(expr, x, 0, 3);
0.9158576591246372

```

## 2.4 Resuelve las ecuaciones anteriores con un error absoluto inferior a una millonésima.

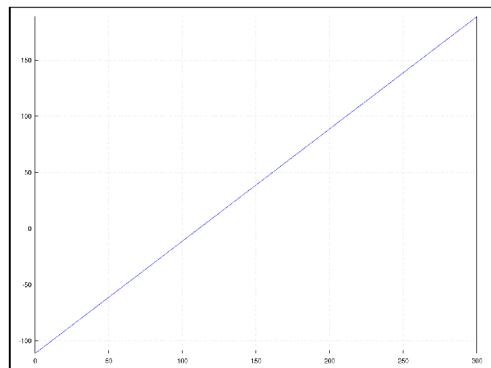
Una millonésima es  $10^{-6}$ . Para resolver este apartado, simplemente bastaría con cambiar el error permitido en la función bisección (actualmente está a  $10^{-5}$ ). No lo hago por pura pereza, es cambiar la variable y repetir los apartados anteriores.

## 2.5 Resuelve las ecuaciones anteriores con un error aproximado relativo menor al cinco por ciento.

```
biseccion_2(expr,var,ext_inf,ext_sup):=
  block(
    [a,b,c, /* extremos y punto medio */
     fa,fb,fc, /* valores de la función en dichos puntos */
     contador:0, /* número de repeticiones */
     tolx:10^(-5),tolfx:10^(-5) /* error permitido */
    ],
    local(f),
    define(f(x),subst(x,var,expr)),
    a:float(min(ext_inf,ext_sup)),
    b:float(max(ext_inf,ext_sup)),
    c:(a+b)*0.5,
    fa:f(a),
    fb:f(b),
    fc:f(c),
    if abs(fc)<tolfx then return([c,1,(b-a)*0.5,fc]),
    if sign(fa)=sign(fb) then error("la función no cambia de signo en los extremos"),
    while ((b-a)/2>tolx and abs(fc)>tolfx)
      do(
        contador:contador+1,
        c:float((a+b)/2),
        fc:f(c),
        if abs(fc)<tolfx then return(),
        if sign(fa) = sign(fc) then (a:c,fa:fc) else (b:c,fb:fc)
      ),
    [c,contador,(b-a)*0.5,f(c)]
  )$
```

### 3 Calcula $x=\sqrt{12345}$ con cinco decimales exactos usando el método de bisección.

```
wxdraw2d(
  color= blue,
  explicit(x-sqrt(12345), x, 0, 300),
  yaxis= true,
  xaxis= true,
  grid= true
);
```



```
biseccion(x-sqrt(12345),x,110,120);
```

```
[111.1080551147461, 18, 3.814697265625 10-5,  
111.1080551147461 - √12345]
```

```
find_root(x-sqrt(12345), x, 110, 120);
```

```
111.1080555135405
```

Los 5 primeros decimales coinciden.

- 4 *El método de trisección para el cálculo de ceros de funciones difiere del método de bisección en que los intervalos se dividen en tres trozos de igual longitud en vez de hacerlo en dos. Escribe un programa que desarrolle el método de trisección y que tenga como entradas: la función, el intervalo y el error permitido.*

```
triseccion(expr,var,ext_inf,ext_sup,err):=
block(
    [a,b,c,d, /* extremos y punto medio */
     fa,fb,fc,fd, /* valores de la función en dichos puntos */
     contador:0, /* número de repeticiones */
     tolx:err,tolfx:err /* error permitido */
    ],
    numer:true,
    local(f),
    define(f(x),subst(x,var,expr)),
    a:float(min(ext_inf,ext_sup)),
    b:float(max(ext_inf,ext_sup)),
    c:float((a+b)/3),
    d:float((a+b)·2/3),
    fa:f(a), fb:f(b), fc:f(c), fd:f(d),

    if abs(fc)<tolfx then return([c,1,(b-a)/3,fc]),
    if abs(fd)<tolfx then return([d,1,(b-a)·2/3,fd]),
    if sign(fa)=sign(fb) then error("la función no cambia de signo en los extremos"),

    while (contador<1500 and (b-a)/3 >tolx and abs(fc)>tolfx and (b-a)·2/3 >tolx and
          do(
            contador:contador+1,
            c:float((a+b)/3),
            d:float((a+b)·2/3),
            fc:f(c), fd:f(d),

            if abs(fc)<tolfx then return(c),
            if abs(fd)<tolfx then return(d),

            if (sign(fa) = sign(fc) or sign(fa) = sign(fd)) then (
                if sign(fa) = sign(fd) then (a:d,fa:fd) else (a:c,fa:fc)
            ),
            if (sign(fb) = sign(fc) or sign(fb) = sign(fd)) then (
                if sign(fb) = sign(fc) then (b:c,fb:fc) else (b:d,fb:fd)
            )
        ),
        [c,d,contador]
    )$)

triseccion(2·x+3,x,-10,10,10^-2);
[-1.481481481481482,-2.962962962962963,1500]
triseccion(2·x+3,x,-10,10,10^1);
[-1.481481481481482,-2.962962962962963,4]
biseccion(2·x+3,x,-10,10);
```

$$[-1.499996185302734, 20, 1.9073486328125 \cdot 10^{-5}, \\ 7.62939453125 \cdot 10^{-6}]$$

Le he tenido que poner un límite al contador porque el método de trisección tiende a oscilar y llega un punto en el cual no se acercará más al resultado real. También sirve poner un error permitido bastante alto ( $10^{-1}$ ).

- 5** *En el método de bisección siempre se elige el punto medio del intervalo como pivote para dividir el intervalo en dos partes iguales. Escribe un programa que elija como pivote un número aleatorio del intervalo. Compara los resultados que se obtienen para la función  $f(x)=x^3-5$  en el intervalo  $[-10, 10]$ .*

La función random(x) devuelve un número entre 0 y x-1. Para obtener un número dentro de un intervalo, haremos lo siguiente:  
extr\_inferior + random(extr\_superior-extr\_inferior)

```

biseccion_rand(expr,var,ext_inf,ext_sup):=
block(
    [a,b,c, /* extremos y punto medio */
     fa,fb,fc, /* valores de la función en dichos puntos */
     contador:0, /* número de repeticiones */
     tolx:10^(-4),tolfx:10^(-4) /* error permitido */
    ],
    numer:true,
    local(f),
    define(f(x),subst(x,var,expr)),
    a:float(min(ext_inf,ext_sup)),
    b:float(max(ext_inf,ext_sup)),
    c:a+random(b-a),
    fa:f(a),
    fb:f(b),
    fc:f(c),
    /*print("fa:",fa,"fb:",fb),*/
    if abs(fc)<tolfx then return([c,1]),
    if sign(fa)=sign(fb) then error("la función no cambia de signo en los extremos"),
    while ((b-a)/2 >tolx and abs(fc)>tolfx and contador<1000)
        do(
            contador:contador+1,
            c:a+random(b-a),
            fc:f(c),
            /*print("c,fc:",c,fc),*/
            if abs(fc)<tolfx then return([c,contador]),
            if sign(fa) = sign(fc) then (a:c,fa:fc) else (b:c,fb:fc)*/,
            print("a,fa:",a,fa,"b,fb:",b,fb),
            print("_____")*/
            ),
    [c,contador]
)$

biseccion_rand(3·x-5,x,-10,10);
[1.666762657352971,37]

biseccion(3·x-5,x,-10,10);
[1.666660308837891,20,9.5367431640625 10-6,-
 1.9073486328125 10-5 ]

```

## 6 Escribe una función para el método de regula-falsi análoga a la del método de bisección

```
regulafalsi(expr,var,ini,fin,err):=block(
[j:0, x0:ini, x1:fin, x2, fx, fx0, fx1, tol:10^(-10)],
numer: true,
define(f(x),subst(x,var,expr)),
fx0:f(x0),
fx1:f(x1),

while abs(x1-x0)/2 > err and j < 30 do(
j:j+1,
x2:(fx1·x0-fx0·x1)/(fx1-fx0),
if f(x2)=0 then return(x2),
if fx0·f(x2)<0 then x1:x2 else x0:x2
),
[x2,j]
)$

regulafalsi(2·x+3,x,-10,10,10^-4);
[-1.499830271829101, 14]

biseccion(2·x+3,x,-10,10);
[-1.499996185302734, 20, 1.9073486328125 10^-5,
7.62939453125 10^-6]
```