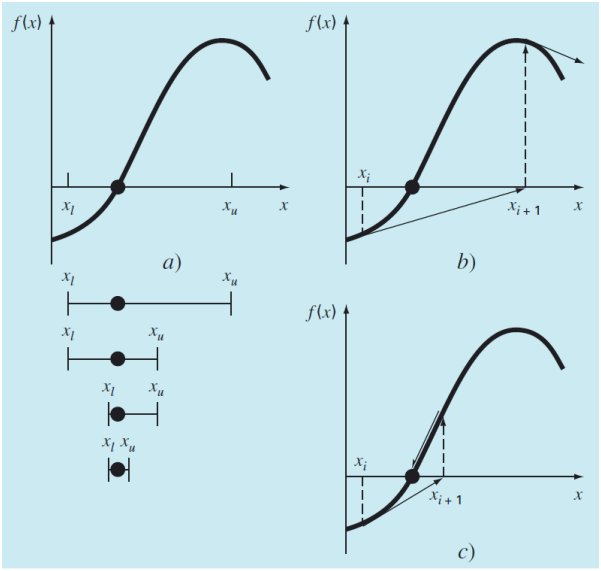
PRÁCTICA 4:
Ecuaciones no Lineales.
Métodos de Punto Fijo

4.1 Resumen de Teoría

4.1.1 Introducción

En los métodos cerrados de la práctica anterior la raíz se encuentra dentro de un intervalo dado. La aplicación repetida de estos métodos siempre genera aproximaciones cada vez más cercanas a la raíz. Se dice que tales métodos son *convergentes* porque se acercan progresivamente a la raíz a medida que se avanza en el cálculo En la figura a) vemos el método de bisección.

En contraste, los *métodos abiertos* descritos en esta práctica se basan en fórmulas que requieren únicamente de un sólo valor de inicio x o que empiecen con un par de ellos, pero que no necesariamente encierran la raíz. Éstos, algunas veces *divergen* o se alejan de la raíz verdadera a medida que se avanza en el cálculo (figura *b*). Sin embargo, cuando los métodos abiertos convergen (figura *c*), en general lo hacen mucho más rápido que los métodos cerrados.



4.1.2 Método de Iteración de Punto Fijo

La idea fundamental en los procesos de iteración del punto fijo es transformar la ecuación:

$$f(x)=0$$

en una ecuación equivalente de la forma:

$$g(x) = x$$

Es decir, el problema de buscar raíces de *f* se convierte en la búsqueda de *puntos fijos* de otra función *g*, que denotaremos como *función de iteración*. Habitualmente existen muchas maneras de transformar el problema.

La ecuación $g(x) = x$ nos proporciona una fórmula para predecir un nuevo valor de *x* en función del valor anterior de *x*. De esta manera, dado un valor inicial para la raíz x_i, la ecuación se utiliza para obtener una nueva aproximación x_{i+1}, expresada por la fórmula iterativa:

$$x_{i+1} = g(x_i)$$

Como en otros métodos iterativos, el error aproximado de este método se puede calcular usando *el error relativo entre dos iteraciones consecutivas*:

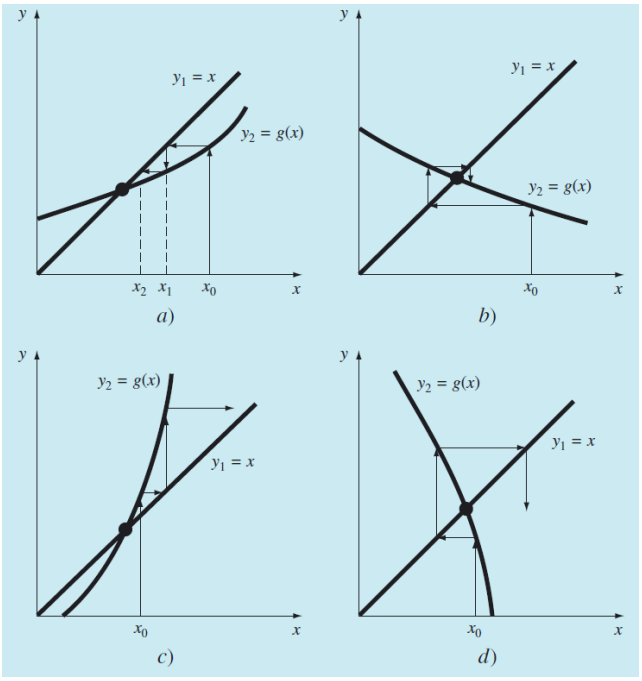
$$|\varepsilon_a| = \left|\frac{x_{i+1} - x_i}{x_{i+1}}\right|$$

Se fija una tolerancia ε_s y los cálculos se repiten hasta que:

$$|\varepsilon_a| < \varepsilon_s$$

Existen diversos teoremas que permitan garantizar la unicidad del punto fijo y la convergencia a dicho punto de la sucesión $x_{n+1} = g(x_n)$. Nosotros vamos a utilizar el *Criterio de Convergencia Global basado en la derivada primera*.

Según este criterio, si g es continua y derivable en $[a, b]$, si cumple que $g[a, b] \subset [a, b]$, y se cumple que $|g'(x)| < 1$, para todo x de $[a, b]$, la iteración de punto fijo converge a la única solución *c* de la ecuación $x=g(x)$, para cualquier elección de x_0 de $[a, b]$.



La solución en la figura *a)* es *convergente*, ya que la aproximación de *x* se acerca más a la raíz con cada iteración. Lo mismo ocurre en la figura *b)*. Sin embargo, éste no es el caso en las figuras *c)* y *d)*, donde las iteraciones divergen de la raíz.

Para la condición de mapeo $g[a, b] \subset [a, b]$, no hay que comprobar todos los puntos del intervalo. Basta con hallar los puntos donde la derivada primera se anula y comprobar que en ellos y en los extremos del intervalo se verifica. Aun así la obtención de un intervalo en el que se verifiquen las hipótesis del Teorema de Convergencia Global es, en ocasiones, compleja. Por ello hace que tenga importancia un resultado conocido con el nombre de *Teorema de Convergencia Local*.

Este resultado asegura que si c es una solución de la ecuación $x=g(x)$ con g una función con derivada continua en un intervalo (a, b) que contiene a c y que $|g'(c)| < 1$, entonces existe $\delta > 0$ de manera que si $x_0 \in [c-\delta, c+\delta]$ entonces la sucesión $x_{n+1} = g(x_n)$ converge a c.

Como vemos, este teorema es también de difícil uso práctico, pues sin conocer la solución c, debemos ser capaces de estimar el valor de su derivada y además el concepto de x_0 suficientemente próximo a la raíz es bastante relativo.

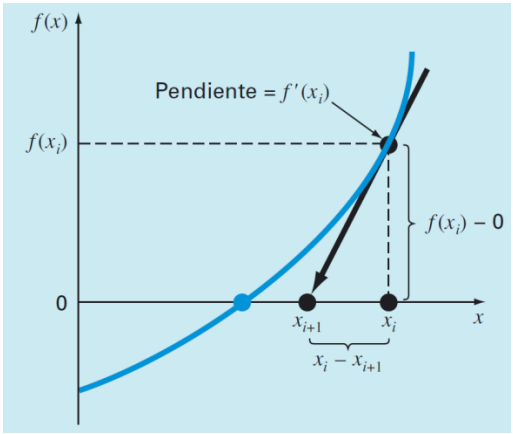
Por último indicar que el error relativo porcentual verdadero en cada iteración es proporcional al error de la iteración anterior. Esta propiedad, conocida como *convergencia lineal*, es característica de la iteración simple de punto fijo.

4.1.3 Método de Newton-Raphson

Tal vez, de las fórmulas para localizar raíces, la fórmula de Newton-Raphson sea la más ampliamente utilizada. Si el valor inicial para la raíz es x_i, entonces se puede trazar una tangente desde el punto $(x_i, f(x_i))$ de la curva. El punto donde esta tangente corta al eje x representa una aproximación mejorada de la raíz.

De la figura (sin más que recordar que la primera derivada en x es equivalente a la pendiente) se tiene:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \; para \quad i \geq 0$$



De esta manera, el Método de Newton comienza con una estimación x_0 para la raíz y a partir de ella se define usando inducción la sucesión de aproximación.

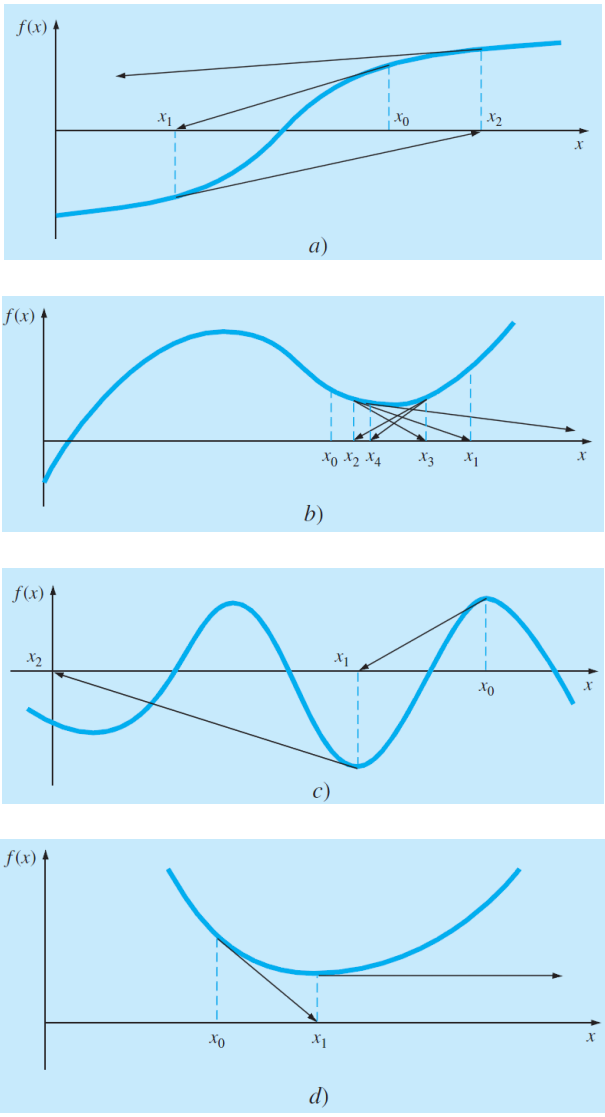
Vemos que el método de Newton-Raphson es un caso especial de la iteración del punto fijo, con:

$$g(x) = x - \frac{f(x)}{f'(x)}$$

El método de Newton-Raphson es convergente, en general, en forma cuadrática. Es decir, el error es proporcional al cuadrado del error anterior.

Aunque en general el método de Newton-Raphson es muy eficiente, hay situaciones donde se comporta de manera deficiente. Por ejemplo en el caso de raíces múltiples. Una *raíz múltiple* corresponde a un punto donde una función es tangencial al eje *x*. El problema surge porque no sólo $f(x)$, sino también $f'(x)$ se aproxima a cero en la raíz. Esto afecta a los métodos de Newton-Raphson y de la secante, los cuales contienen derivadas (o su aproximación) en el denominador de sus fórmulas respectivas. Esto provocará una división entre cero cuando la solución converge muy cerca de la raíz.

Sin embargo, también cuando se trata de raíces simples, se encuentran dificultades. Por ejemplo, la figura *a*) muestra el caso donde la raíz es cerca de un punto de inflexión, es decir con $f''(x)=0$. Las iteraciones divergen progresivamente de la raíz. En la figura *b*) se ilustra la tendencia del método de Newton-Raphson a oscilar alrededor de un mínimo o máximo local. En la figura *c*) se muestra cómo un valor inicial cercano a una raíz salta a una posición varias raíces más lejos. Esta tendencia a alejarse del área de interés se debe a que se encuentran pendientes cercanas a cero. De hecho una pendiente cero ($f'(x_i) = 0$) provoca división entre cero y el fallo del método. En la figura *d*) la solución se dispara horizontalmente y jamás toca al eje *x*.

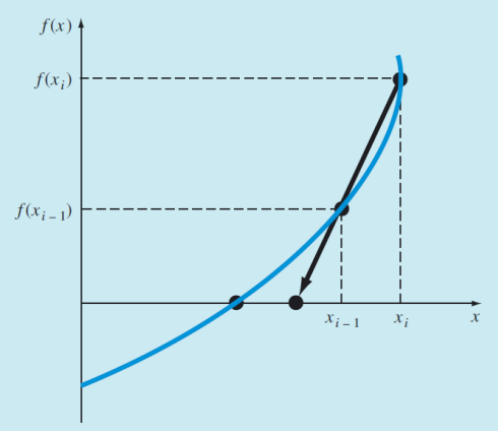


4.1.4 Método de la Secante

Un problema potencial en la implementación del método de Newton-Raphson es la evaluación de la derivada. Aunque esto no es un inconveniente para los polinomios ni para muchas otras funciones, existen algunas funciones cuyas derivadas en ocasiones resultan muy difíciles de calcular.

En dichos casos, la derivada se puede aproximar mediante una diferencia finita dividida hacia atrás:

$$f'(x_i) = \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i}$$



Esta aproximación se sustituye en la ecuación del método de Newton para obtener la siguiente ecuación iterativa:

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

La ecuación anterior es la fórmula para el *método de la secante*. Observe que el método requiere de dos valores iniciales de *x*. Sin embargo, debido a que no se necesita que *f*(*x*) cambie de signo entre los valores dados, este método no se clasifica como un método cerrado.

Este método es semejante al de Newton, aunque no tiene su convergencia cuadrática, ya que en lugar de calcular la derivada de la función, se aproxima ésta mediante una diferencia finita.

Además esta ecuación es semejante a la del método de Regula-Falsi, con la diferencia que en el método de la secante la raíz no se encuentra necesariamente entre x_{i-1} y x_i.

El algoritmo, por lo demás, es semejante al de punto fijo.

Newton y la secante son métodos de carácter local: convergen bajo ciertas condiciones si la semilla está "cerca" de la solución.

Además, se demuestra que Newton y la secante convergen linealmente si la derivada en la raíz buscada es cero: *f'*(*c*) = 0.

4.2 El Método de Iteración de Punto Fijo

Como vimos antes, con este método se trata de resolver una ecuación de la forma:

$$f(x)=0$$

convirtiéndola en una ecuación equivalente de la forma:

$$g(x) = x$$

De esta forma las raíces de *f* se convierten en los *puntos fijos* de otra función *g*, llamada *función de iteración*. En teoría hemos visto la importancia de la correcta elección de $g(x)$.

Con este método, dado un valor inicial para la raíz x_i, se obtiene una nueva aproximación x_{i+1}, dada por la fórmula iterativa:

$$x_{i+1} = g(x_i)$$

Consideremos el siguiente ejemplo: vamos a resolver la ecuación:

$$x^2 - 3x - 1 = 0$$

Las raíces de esta ecuación son fáciles de calcular con cualquiera de los métodos ya vistos. Por ejemplo, el comando **solve** nos da:

```
(%i1)    kill(all)$
(%i1)    float(solve(x^2-3*x-1));
(%o1)    [x=-0.3027756377319946,x=3.302775637731995]
```

Pero ahora vamos a pasar esta ecuación a formato de punto fijo. Quizás la elección más intuitiva sea:

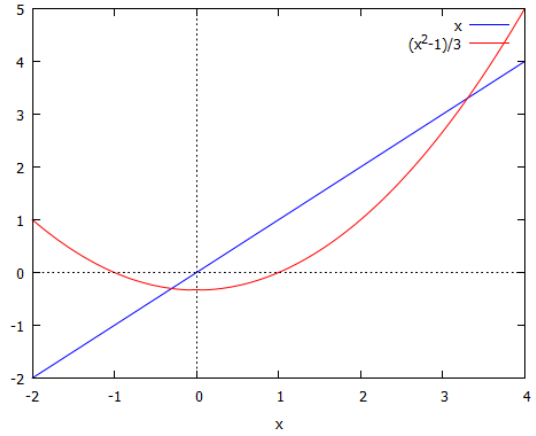
$$x = g(x) = \frac{x^2 - 1}{3}$$

En este caso $g(x)$ es continua y derivable en [-1,1]. Además g[-1, 1]⊂[-1, 1]. El valor absoluto de su derivada, $2x/3$, está acotado en [-1,1] por 2/3, es decir se cumple que: $|g'(x)|<1$, $x \in$ [-1,1]. Así que satisface las condiciones del Teorema de Convergencia Global.

Si representamos x y $g(x)$ tenemos:

```
(%i2)    g(x):=(x^2-1)/3$
(%i3)    plot2d([x,g(x)],[x,-2,4])$
```

Vamos a aplicar el método de punto fijo para buscar la raíz más cercana a 0. Partiremos para buscarla de la semilla inicial x_0=0, la cual pertenece al intervalo [-1,1].



Vamos a utilizar una notación muy práctica en Maxima para representar los sucesivos valores de *x* en las iteraciones. Consiste en **x[ ]**, lo cual equivale a un subíndice:

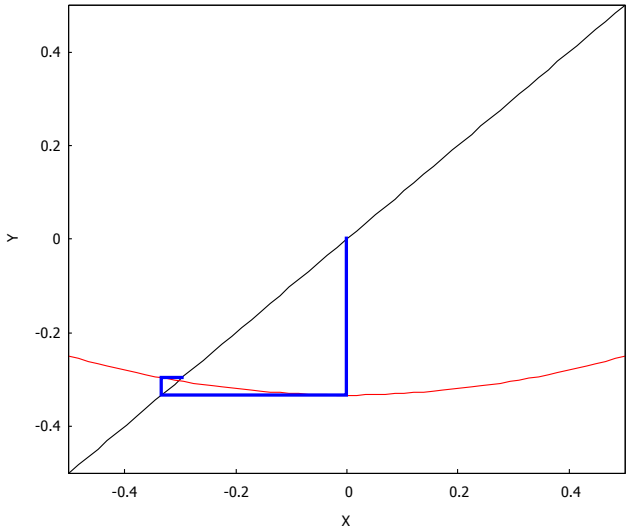
```
(%i4)    x[i];
(%o4)    x_i
```

El bucle básico que debemos programar podría ser éste, en el cual observamos la rápida convergencia:

```
(%i7)    x[0]:0$
         maxiter:10$
         for i:0 thru maxiter do (
             print(float(g(x[i]))),x[i+1]:g(x[i]));
-0.3333333333333333
  -0.2962962962962963
  -0.3040695016003658
  -0.3025139127321684
  -0.302828444201158
  -0.3027649777942354
  -0.3027777894070854
  -0.3027752034139196
  -0.3027757253992199
  -0.3027756200363254
  -0.3027756413038729
  (%o7)    done
```

Podemos visualizar las primeras iteraciones que hemos realizado. La gráfica siguiente utiliza el comando **draw2d**. En él hemos incluido la opción **points** que nos dibuja los puntos que le indicamos. En este caso el formato elegido ha sido **points([x1,x2,…],[y1,y2,…])** que nos dibuja los puntos [x1,y1],[x2,y2],…El comando **points_joined = true** nos une los puntos. El resto de las opciones, como el color, pueden consultarse en la ayuda.

```
(%i9)    load(draw)$
         draw2d(xrange=[-0.5,0.5],yrange=[-0.5,0.5],
              color=black,
               explicit(x,x,-1,1),
               color=red,
               explicit(g(x),x,-1,1),
              color=blue,
              points_joined = true,
               line_width =3,
              points([x[0],x[0],x[1],x[1],x[2]],
                  [0,g(x[0]),g(x[0]),g(x[1]),g(x[1])])
         )$
0 errors, 0 warnings
```



En formato bloque el método, incluyendo el criterio de parada basado en el error relativo entre dos iteraciones consecutivas será:

```
(%i10)   /* Función puntofijo1(g_,x0,tolr,maxiter) */
         /* Método de Punto Fijo para resolver x = g(x) */
         /* ARGUMENTOS DE ENTRADA: */
         /* g ... Función de iteración elegida */
         /* x0 .. Semilla inicial para la búsqueda */
         /* tolr .. Tolerancia relativa entre dos iteraciones */
         /* maxiter ..Número máximo de iteraciones */
         /* ARGUMENTOS DE SALIDA: */
         /* x[N] ... Valor aproximado de la raíz */;
         /* N ... Número de Iteraciones */;
         puntofijo1(g_,x0,tolr,maxiter):=block([numer],numer:true,
         x[0]:x0,
         for i:1 thru maxiter do
         (
         x[i]:g_(x[i-1]),
         if abs(x[i]-x[i-1])/abs(x[i]) < tolr then
         return((print("La aproximación buscada es
         x[",i,"]=",x[i]))),
         if i=maxiter then
         print("No lograda la aproximación deseada en ",
         maxiter," iteraciones"))
         )$
```

```
(%i11)   puntofijo1(g,0,10^-16,100);
La aproximación buscada es x[ 25 ]= -0.3027756377319946
  (%o11)   -0.3027756377319946
```

La misma obtenida en la versión **float** del comando **solve**.

El siguiente ejemplo nos advierte sobre los peligros del método de punto fijo. Vamos a hallar una raíz de la ecuación $e^x = 3x$ en el intervalo [0, 1]. Vamos a aplicar el método de iteración del punto fijo, y vamos a analizar la convergencia con dos funciones de iteración:

$$g_1(x) = \frac{e^x}{3};\ g_2(x) = \ln(3x)$$

```
(%i12)   g1(x):=exp(x)/3$
```

Aplicamos de nuevo el bloque puntofijo1 y:

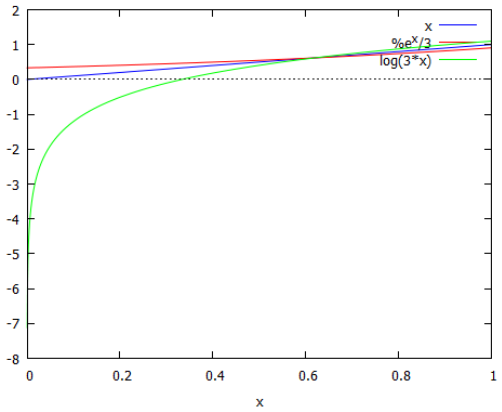
```
(%i13)   puntofijo1(g1,0.5,10^-6,100);
La aproximación buscada es x[ 25 ]= 0.6190606479535598
  (%o13)   0.6190606479535598
(%i14)   exp(0.6190606479535598)-3*(0.6190606479535598);
(%o14)   7.300111986019431 10^-7
```

Sin embargo:

```
(%i15)   g2(x):=log(3*x)$
(%i16)   puntofijo1(g2,0.5,10^-6,10);
No lograda la aproximación deseada en  10  iteraciones
  (%o16)   done
```

Se puede comprobar que la primera función de iteración $g_1(x)$ verifica las condiciones del teorema global de convergencia en [0,1] y por tanto el método converge a la única solución. Sin embargo la segunda función de iteración $g_2(x)$ no verifica las condiciones del teorema global de convergencia y enseguida calcula logaritmos de números negativos.

```
(%i17)   plot2d([x,g1(x),g2(x)],[x,0,1])$
```

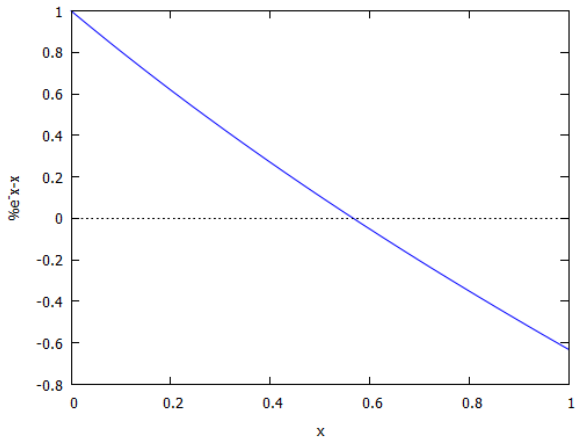


4.3 El Método de Newton-Raphson

Consideremos la ecuación: $f(x) = e^{-x} - x = 0$.

Vamos a aproximar la solución de dicha ecuación en el intervalo [0,1] por el método de Newton.

```
(%i1)    kill(all)$
(%i1)    f(x):=%e^(-x)-x$
(%i2)    plot2d(f(x),[x,0,1])$
```



Se puede demostrar que podemos aplicar el teorema de convergencia global de dicho método y por tanto que podemos comenzar a iterar por cualquier punto del intervalo [0,1], por ejemplo por el punto $x_0 = 0.5$. Comencemos escribiendo el **block** correspondiente.

```
(%i3)    /* Función newton1(f_,x0,tolr,maxiter) */
         /* Método de Newton-Raphson para resolver f(x)=0 */
         /* ARGUMENTOS DE ENTRADA: */
         /* f ... Función de la que se buscan los ceros */
         /* x0 .. Semilla inicial para la búsqueda */
         /* tolr .. Tolerancia relativa entre dos iteraciones */
         /* maxiter .. Número máximo de iteraciones */
         /* ARGUMENTOS DE SALIDA: */
         /* x[N]...Valor aproximado de la raíz */;
         /* N ... Número de Iteraciones */;

         newton1(f_,x0,tolr,maxiter):= block(
         [numer],numer:true,
         x[0]:x0,
         define(Df(x),diff(f_(x),x)),
         for i:1 thru maxiter do
         (
         x[i]:x[i-1]-f_(x[i-1])/Df(x[i-1]),
         if abs(x[i]-x[i-1])/abs(x[i]) < tolr then
         return((print("La aproximación buscada es
         x[",i,"]=",x[i]))),
         if i=maxiter then
         print("No lograda la aproximación deseada en ",
         maxiter," iteraciones"))
         )$
```

El bloque no es más que una adaptación del ya presentado para el método de punto fijo, sin más que incluir la función de iteración de este método:

$$g(x) = x - \frac{f(x)}{f'(x)}$$

En este caso usamos como criterio de salida el error relativo entre dos iteraciones consecutivas, obteniendo:

```
(%i4)    newton1(f,0.5,10^-16,100);
La aproximación buscada es x[ 6 ]= 0.5671432904097838
 (%o4)    0.5671432904097838
(%i5)    f(0.5671432904097838);
(%o5)    0.0
```

Podemos presentar otro bloque usando como criterio de salida el error en el valor de la función en la aproximación, dado que en este tipo de problemas sabemos que el valor de la función en la raíz es cero.

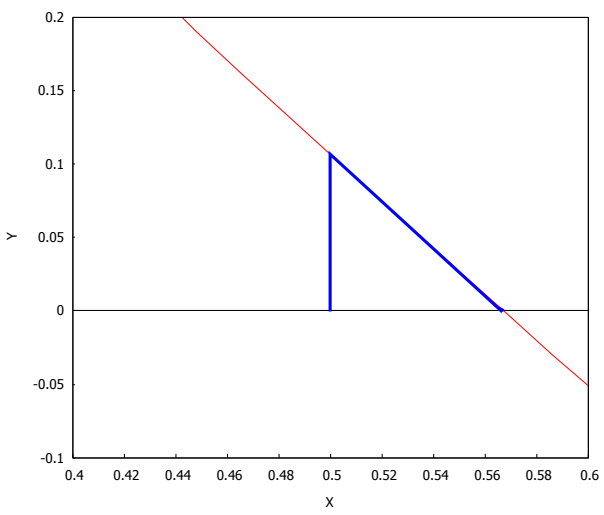
```
(%i6)    /* Función newton3(f_,x0,tol,maxiter) */
         /* Método de Newton-Raphson para resolver f(x)=0 */
         /* ARGUMENTOS DE ENTRADA: */
         /* f ... Función de la que se buscan los ceros */
         /* x0 .. Semilla inicial para la búsqueda */
         /* tol .. Tolerancia en alcanzar el valor de f(x)=0 */
         /* maxiter ..Número máximo de iteraciones */
         /* ARGUMENTOS DE SALIDA: */
         /* x[N] ... Valor aproximado de la raíz */;
         /* N ... Número de Iteraciones */;

         newton3(f_,x0,tol,maxiter):= block(
         [numer],numer:true,
         x[0]:x0,
         define(Df(x),diff(f_(x),x)),
         for i:1 thru maxiter do
         (
         x[i]:x[i-1]-f_(x[i-1])/Df(x[i-1]),
         if abs(f_(x[i])) < tol then
         return((print("La aproximación buscada es
         x[",i,"]=",x[i]))),
         if i=maxiter then
         print("No lograda la aproximación deseada en ",
         maxiter," iteraciones"))
         )$

(%i7)    newton3(f,0.5,10^-16,30)$
La aproximación buscada es x[ 5 ]= 0.5671432904097838
(%i8)    f(0.5671432904097838);
(%o8)    0.0
```

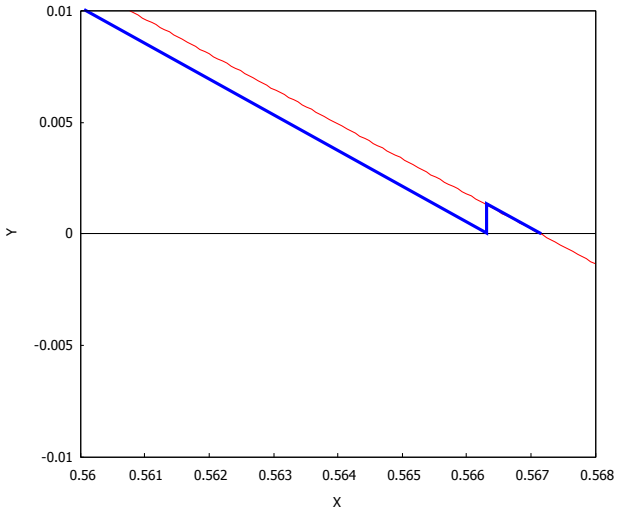
Si ponemos maxiter = 3, vemos que se detiene el programa por haber superado el número máximo de iteraciones sin conseguir la aproximación deseada.

```
(%i9)    newton3(f,0.5,10^-16,3)$
No lograda la aproximación deseada en  3  iteraciones
```

Podemos presentar una visualización del método, aunque su extraordinaria rapidez lo hace difícil. En las siguientes gráficas se presentan las dos primeras iteraciones. Obsérvese la escala de la gráfica de la derecha.

```
(%i10)   draw2d(xrange=[0.4,0.6],yrange=[-0.1,0.2],
              color=black,
               explicit(0,x,-1,1),
               color=red,
               explicit(f(x),x,-1,1),
              color=blue,
              points_joined = true,
               line_width =3,
              points([x[0],x[0],x[1],x[1],x[2],x[2]],
                  [0,f(x[0]),0,f(x[1]),0,f(x[2])])
         )$
(%i11)   draw2d(xrange=[0.56,0.568],yrange=[-0.01,0.01],
              color=black,
               explicit(0,x,0.56,0.568),
               color=red,
               explicit(f(x),x,0.56,0.568),
              color=blue,
              points_joined = true,
               line_width =3,
              points([x[0],x[0],x[1],x[1],x[2],x[2]],
                  [0,f(x[0]),0,f(x[1]),0,f(x[2])])
         )$
```





En ocasiones la convergencia del método de Newton no es tan rápida. Como comentamos suele ocurrir en el caso de presentarse raíces múltiples o puntos de inflexión. Veamos algunos ejemplos.

```
(%i13)   f1(x):=sin(x)-x$
         newton3(f1,0.5,10^-12,50)$
La aproximación buscada es x[ 20 ]= 1.492374116340444 10^-4

(%i15)   f2(x):=(x-2)^2$
         newton3(f2,1.5,10^-12,50)$
La aproximación buscada es x[ 19 ]= 1.999999046325684
(%i17)   f3(x):=(x-1)^3$
         newton3(f3,1.5,10^-16,50)$
La aproximación buscada es x[ 29 ]= 1.000003911321288
```

En el caso de la función arco tangente la elección de la semilla es fundamental:

```
(%i19)   f(x):=atan(x)$
         newton3(f,1.3917,10^-16,50)$
La aproximación buscada es x[ 15 ]= 0.0
(%i21)   f(x):=atan(x)$
         newton3(f,1.,10^-16,50)$
La aproximación buscada es x[ 5 ]= 0.0
```

4.3.1 La función newton de Maxima

Hay también una función de Maxima para el método de Newton, se trata de la siguiente:

newton(expresion,x, x0, eps)

pero requiere ejecutar previamente **load(newton1)** para cargar el paquete correspondiente.

El resultado es una raíz aproximada de **expresion**, en la variable **x**, comienza a iterar por **x0** y el proceso sigue hasta que se cumple que **|expresión| < eps**.

Para información más detallada sobre este comando teclear **? newton**.

Veamos su aplicación al ejemplo anterior.

```
(%i23)   load(newton1)$
         newton(%e^(-x)-x,x,0.5,10^-16);
(%o23)   0.5671432904097838
```

4.4 El Método de la Secante

Para finalizar presentamos una modificación del método de Newton que no precisa calcular la derivada, el conocido como método de la secante.

Recordemos que el método requiere de dos valores iniciales de *x*. El algoritmo, por lo demás, es semejante al de punto fijo.

```
(%i1)    kill(all)$
(%i1)    /* Función secante1(f_,x0,x1,tolr,maxiter) */
         /* Método de secante para resolver f(x)=0 */
         /* ARGUMENTOS DE ENTRADA: */
         /* f ... Función de la que se buscan los ceros */
         /* x0,x1 .. Semillas iniciales para la búsqueda */
         /* tolr .. Tolerancia relativa entre dos iteraciones */
         /* maxiter .. Número máximo de iteraciones */
         /* ARGUMENTOS DE SALIDA: */
         /* x[N]...Valor aproximado de la raíz */;
         /* N ... Número de Iteraciones */;
         secante1(f_,x0,x1,tolr,maxiter):= block(
         [numer],numer:true,
         x[0]:x0,
         x[1]:x1,
         for i:1 thru maxiter do
         (
         x[i+1]:x[i]-(x[i-1]-x[i])*f_(x[i])/(f_(x[i-1])-f_(x[i])),
         if abs(x[i]-x[i-1])/abs(x[i]) < tolr then
         return((print("La aproximación buscada es
         x[",i,"]=",x[i]))),
         if i=maxiter then
         print("No lograda la aproximación deseada en ",
         maxiter," iteraciones"))
         )$
(%i2)    f(x):=%e^-x-x$
(%i3)    secante1(f,0.5,0.6,10^-12,100)$
La aproximación buscada es x[ 6 ]= 0.5671432904097838
```

Pero no es necesario que las dos semillas iniciales encierren a la raíz:

```
(%i4)    secante1(f,1,1.3,10^-12,100)$
La aproximación buscada es x[ 7 ]= 0.5671432904097838
```

Si comparamos la velocidad de convergencia de Secante con Newton (previamente cargado), observamos que éste último es más rápido.

```
(%i6)    newton3(f,1.,10^-12,50)$
La aproximación buscada es x[ 4 ]= 0.5671432904097838
```

4.5 Ejercicios Propuestos

Ejercicio 1. Vamos a aplicar el algoritmo de punto fijo para buscar la raíz positiva de la ecuación:

$$x = g(x) = \sqrt{\frac{x + 3 - x^4}{2}}$$

Para ello, en primer lugar, busca la raíz mediante algún comando de Maxima y representa gráficamente el problema. Usa el bloque con $x_0 = 1$, tolerancia relativa 10^{-2} y 100 como máximo de iteraciones. ¿Qué sucede? ¿Cuánto valen los últimos valores de $x[N]$ para N=90,…,100?
Para dar una explicación, intenta encontrar un intervalo de mapeo de g y calcula el valor de la derivada en la raíz.

Ejercicio 2. Compruébese gráficamente que la ecuación $x^3 - 3x^2 - 18x + 17 = 0$ tiene sus tres soluciones en el intervalo [-4, 8] y calcúlense las mismas, aplicando el método de Newton-Raphson, con tolerancia relativa 10^{-6}.

Ejercicio 3. Constrúyase la función newton1b, modificando la función newton1, de manera que la variable de salida contenga los valores de las iteraciones del método de Newton-Raphson. A continuación, aplíquese newton1b, a la resolución de la ecuación del ejemplo anterior.

Ejercicio 4. Resuélvase nuevamente la ecuación $x^3 - 3x^2 - 18x + 17 = 0$, con la misma tolerancia de los ejemplos anteriores, aplicando el método de la secante. ¿Qué observas en la velocidad?

Ejercicio 5. Recordemos que si (x_n) es una sucesión convergente a un número c y llamamos $e_n = x_n - c$ entonces se dice que la convergencia de (x_n) es de orden p si se verifica:

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^i} = 0, \;\; para\; i = 0,1,\dots,p-1; \;\; \lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^p} = k \neq 0$$

Si $p = 1$ la convergencia se dice lineal, cuando $p = 2$ hablamos de convergencia cuadrática, y cuando $p = 3$ de convergencia cúbica. Hablando de forma simple, converger cuadráticamente significa que aproximadamente en cada iteración duplicamos el número de decimales exactos.

Resolver la ecuación $x^3 - 3x - 2 = 0$ por el método de Newton empezando por el punto 3 para calcular la raíz positiva $r = 2$ del polinomio. Verificar que el orden de convergencia es $p = 2$. Sugerencia: calcula los errores $e_n = x_n - c$ entre iteraciones consecutivas y su cociente.

Ejercicio 6. Repitamos la misma idea con la ecuación $x^3 - 3x - 2 = 0$ pero ahora para calcular la raíz doble negativa $r = -1$ del polinomio, empezando por el punto -2. Verificar que el orden de convergencia, en este caso de raíz múltiple, se convierte en lineal $p = 1$.

Ejercicio 7. Resuelve por los diversos métodos que hemos estudiado en las prácticas 3 y 4, las siguientes ecuaciones. Compara cómo se comportan y cómo de rápido convergen.

(a) $x = \cos(x)$, que tiene una raíz cerca de $x = 0.739085$.
(b) $x^{20} - 1 = 0$ que tiene una raíz en $x = 1$.
(c) $\tan(x) - 30 = 0$ que tiene una raíz cerca de $x = 1.537475$.
(d) $x^2 - 2.2x + 1.21 = 0$ que tiene una raíz en $x = 1.1$