

(#387) . BÚSQUEDA DE SOLUCIONES (I): MÉTODO DE LA BISECCIÓN

[MONOTEMA] Vamos a comenzar una serie de posts sobre la búsqueda de soluciones numéricas (también llamadas raíces o ceros) para una función. El objetivo es que los alumnos de marketing se familiaricen con técnicas matemáticas y de computación que les ayuden a resolver problemas más complejos en el futuro. Para ello vamos a emplear el software Maxima, de libre acceso.

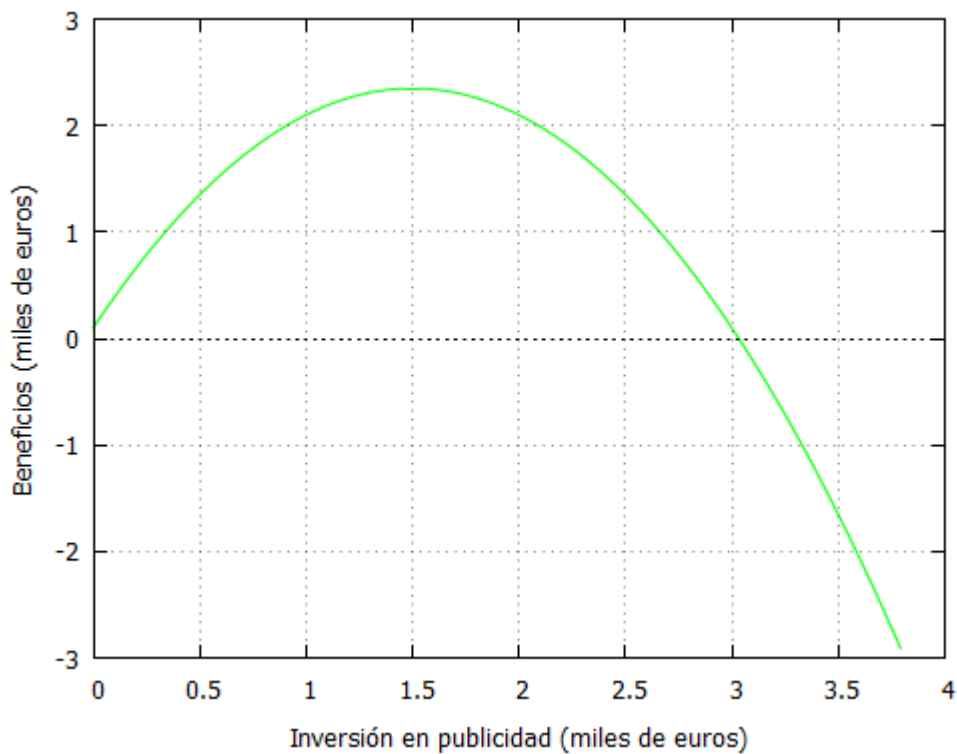
Función de partida

Imaginemos que conocemos la forma de una función que relaciona la inversión en publicidad con los beneficios. En [entradas anteriores](#) hemos explicado que esa relación es probablemente no lineal, y que puede adoptar diversas formas. Vamos a partir de la siguiente función:

$$f(x) = -x^2 + 3x + 0.1$$

Lo primero que vamos a hacer es representar esa función en Maxima:

```
funcion: -x^2+3*x+0.1;
plot2d(funcion,[x,0,4],[y,-3,3],
[xlabel, "Inversión en publicidad (miles de euros)"],
[ylabel, "Beneficios (miles de euros)"]);
```



En

esta hipotética situación, el analista de marketing puede computar el máximo y el cero de la función, de la siguiente forma:

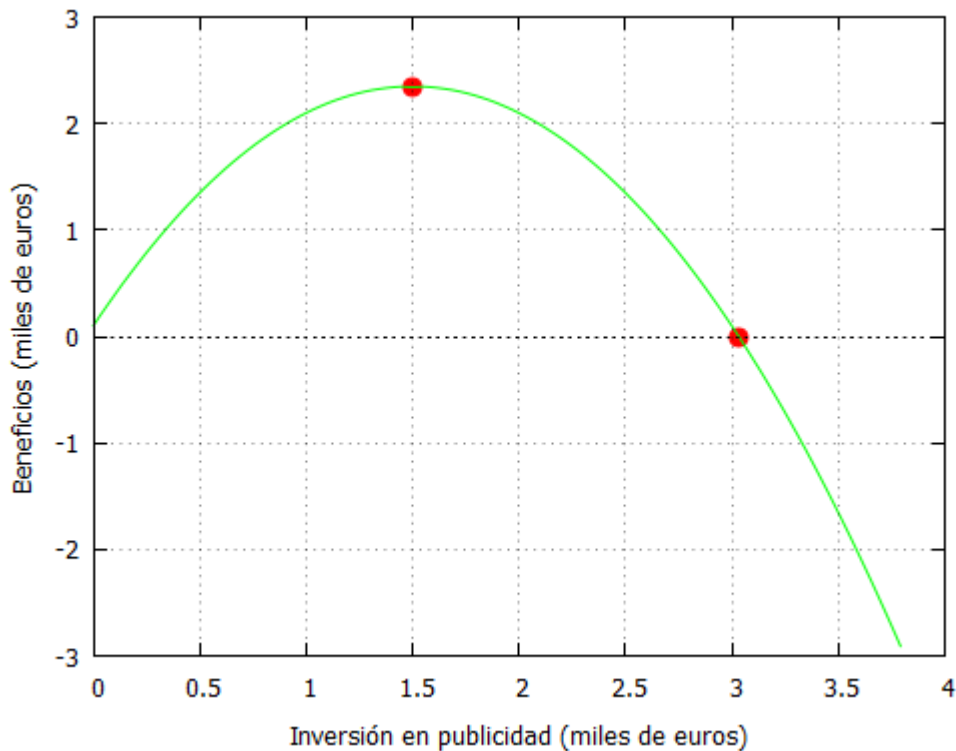
```

derivada: diff(funcion, x);
maximo:solve(derivada,x), numer;
cero: solve(funcion,x),numer;
f_maximo: ev(funcion,x=1.5), numer;
f_cero:ev(funcion,x=3.03297097167559), numer;
plot2d([[discrete, [[1.5,f_maximo], [3.03297097167559,
f_cero]]], funcion],[x,0,4],[y,-3,3],
[xlabel, "Inversión en publicidad (miles de euros)"],
[ylabel, "Beneficios (miles de euros)"],
[style, points, lines], [color, red, green], [legend,
false]);

```

De este modo tenemos que el máximo es $x_{maximo} = 1.5$, y el cero es $x_{cero} = 3.03297097167559$ cuando $x \geq 0$ (hay otra solución al ser una ecuación de segundo grado pero no nos interesa porque se refiere a una inversión en publicidad negativa, lo cual no es posible).

Podemos representar esos puntos en el gráfico anterior de la siguiente manera:



Es evidente que nos interesa conocer el valor de inversión que maximiza los beneficios, pero también nos puede resultar atractivo conocer el punto a partir del cual incrementar más la inversión produce pérdidas. Y este es el objetivo en el que nos vamos a centrar.

Para ello implementar el método de la bisección, que puede consultarse en [Burden, Faires & Burden \(2017\)](#).

Método de la bisección

(1) Objetivo: Aproximarse numéricamente a la solución p de una función $f(x)$, tal que $f(p) = 0$

(2) Condiciones: La función $f(x)$ debe ser continua en el intervalo $[a, b]$ considerado, donde $f(a)$ y $f(b)$ son de distinto signo, es decir, $f(a)f(b) < 0$.

(3) Descripción rápida: Escoger un intervalo $[a_1, b_1]$ a priori

donde $f(a_1)f(b_1) < 0$ e ir evaluando soluciones p_i en la mitad de ese intervalo y de los siguientes $[a_i, b_i]$ hasta que se llegue a un cierto número de iteraciones N_i o a un valor prefijado de error ϵ . De este modo, se trata de ir evaluando la función en los valores de medios de cada intervalo sucesivo (donde la anchura de cada intervalo es la mitad del anterior) hasta aproximarse a la solución. Por el teorema de Bolzano, sabemos que vamos a encontrar esa raíz.

(4) Convergencia: Siempre converge a una solución, aunque su velocidad puede ser lenta.

(5) Estimación del error: Hay diversas opciones, aunque una de las más recomendables es el error relativo:

$$\frac{|p_N - p_{N-1}|}{|p_N|} < \epsilon, p_N \neq 0$$

Implementación en Maxima

Vamos a programar con Maxima el algoritmo de la bisección. Para ello, empleamos un bucle con 15 iteraciones. La inspección gráfica nos permite estipular un intervalo inicial [2.5,3.5]:

```

a[0]:2.5;
b[0]:3.5;
for i:1 thru 15 do(
p[i]:a[i-1]+((b[i-1]-a[i-1])/2),
f[i]: -(p[i])^2+3*(p[i])+0.1,
g[i]: -(a[i-1])^2+3*(a[i-1])+0.1,
if g[i]*f[i]<0
then (b[i]:p[i],
a[i]:a[i-1])
else (a[i]:p[i],
b[i]:b[i-1])),
Error_abs[i]:abs(p[i+1]-p[i]),
Error_rel[i]: Error_abs[i]/abs(p[i+1]));
datos: makelist([i,"Iteración",a[i],b[i],p[i],f[i], Error_abs[i-1],
Error_rel[i-1]], i, 1,15);
matriz_resultados:apply(matrix,datos);

```

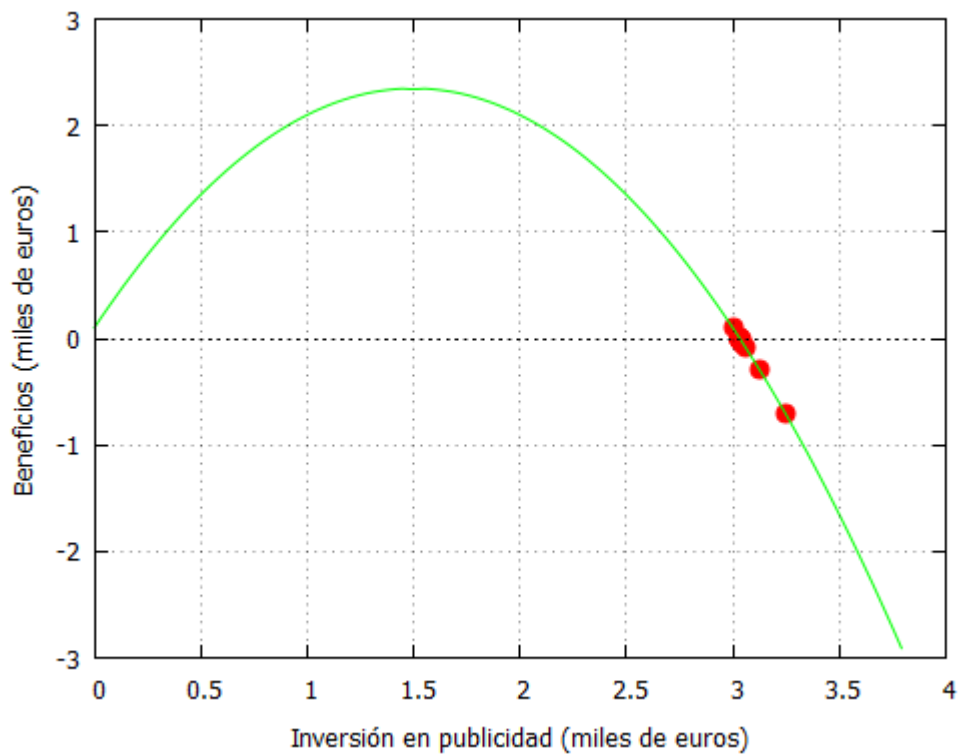
Le hemos dicho al programa que nos retorne en el output siguiente:

					<i>Error_abs₀</i>	<i>Error_rel₀</i>	
1	<i>Iteración</i>	3.0	3.5	3.0	0.1		
2	<i>Iteración</i>	3.0	3.25	3.25	-0.7125	0.25	0.07692307692307693
3	<i>Iteración</i>	3.0	3.125	3.125	-0.290625	0.125	0.04
4	<i>Iteración</i>	3.0	3.0625	3.0625	-0.09140625	0.0625	0.02040816326530612
5	<i>Iteración</i>	3.03125	3.0625	3.03125	0.005273437500000006	0.03125	0.01030927835051546
6	<i>Iteración</i>	3.03125	3.046875	3.046875	-0.04282226562499999	0.015625	0.005128205128205128
7	<i>Iteración</i>	3.03125	3.0390625	3.0390625	-0.01871337890624999	0.0078125	0.002570694087403599
8	<i>Iteración</i>	3.03125	3.03515625	3.03515625	-0.006704711914062494	0.00390625	0.001287001287001287
9	<i>Iteración</i>	3.03125	3.033203125	3.033203125	-7.118225097656194 10 ⁻⁴	0.001953125	6.43915003219575 10 ⁻⁴
10	<i>Iteración</i>	3.0322265625	3.033203125	3.0322265625	0.002281761169433599	9.765625 10 ⁻⁴	3.22061191626409 10 ⁻⁴
11	<i>Iteración</i>	3.03271484375	3.033203125	3.03271484375	7.852077484130915 10 ⁻⁴	4.8828125 10 ⁻⁴	1.610046691354049 10 ⁻⁴
12	<i>Iteración</i>	3.032958984375	3.033203125	3.032958984375	3.675222396851141 10 ⁻⁵	2.44140625 10 ⁻⁴	8.049585446349512 10 ⁻⁵
13	<i>Iteración</i>	3.032958984375	3.0330810546875	3.0330810546875	-3.375202417373602 10 ⁻⁴	1.220703125 10 ⁻⁴	4.024630740129593 10 ⁻⁵
14	<i>Iteración</i>	3.032958984375	3.03302001953125	3.03302001953125	-1.503802835941259 10 ⁻⁴	6.103515625 10 ⁻⁵	2.012355865011169 10 ⁻⁵
15	<i>Iteración</i>	3.032958984375	3.032989501953125	3.032989501953125	-5.681309849023264 10 ⁻⁵	3.0517578125 10 ⁻⁵	1.006188056547769 10 ⁻⁵

Como puede apreciarse en las dos primeras columnas de resultados tenemos los extremos de cada intervalo del algoritmo. En la tercera está el valor de p , que siempre es la mitad del valor de esos extremos. la cuarta columna evalúa la función en cada p considerado, mientras que las columnas quinta y sexta corresponden, respectivamente, al error absoluto y al relativo.

Podemos ver aquí como las iteraciones nos acercan a la solución:

```
matriz_t: transpose(matriz_resultados);
p_list: matriz_t[5];
fp_list: matriz_t[6];
funcion: -x^2+3*x+0.1;
plot2d([[discrete, p_list,fp_list],funcion],[x,0,4],[y,-3,3],
[xlabel, "Inversión en publicidad (miles de euros)",
[ylabel, "Beneficios (miles de euros)"],
[style, points, lines], [color, red, green], [legend,
false]);
```



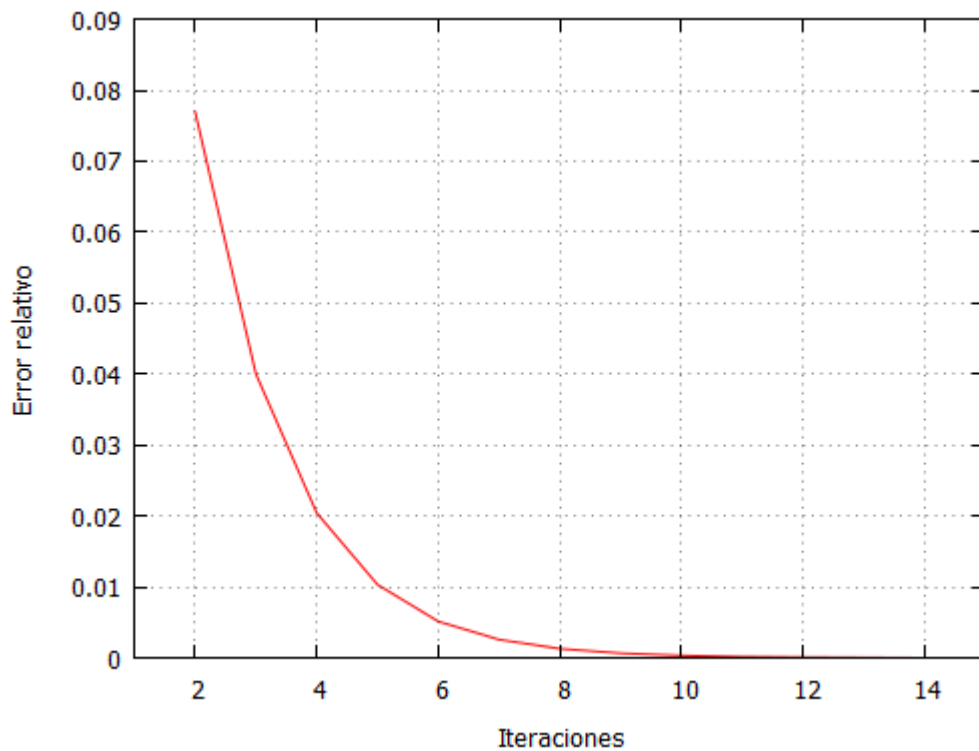
Y

también es interesante comprobar cómo disminuye el error relativo con el número de iteraciones:

```

Error_rel_list: matriz_t[8];
Iteraciones_list:matriz_t[1];
plot2d([discrete,
Iteraciones_list,Error_rel_list],[x,1,15],[y,0,0.09],
[xlabel, "Iteraciones"],
[ylabel, "Error relativo"],
[style, lines], [color, red], [legend, false]);

```



Criterio de parada

Podemos estipular que nos baste con una tolerancia de error, es decir, pedirle al programa que pare las iteraciones cuando se cumpla una condición determinada, por ejemplo, que el error relativo sea menor que una cota que fijemos. De este modo, si fijamos que el error relativo sea como máximo 0.001:

```

        a[0]:2.5;
        b[0]:3.5;
    for i:1 while Error_rel[i]>0.001
        do(
    p[i]:a[i-1]+((b[i-1]-a[i-1])/2),
    f[i]: -(p[i])^2+3*(p[i])+0.1,
    g[i]: -(a[i-1])^2+3*(a[i-1])+0.1,
        if g[i]*f[i]<0
        then (b[i]:p[i],
            a[i]:a[i-1])
        else (a[i]:p[i],
            b[i]:b[i-1]),
        Error_abs[i]:abs(p[i+1]-p[i]),
    Error_rel[i]: Error_abs[i]/abs(p[i+1]),
        print(i,p[i], f[i],Error_rel[i]));

```

Le hemos dicho al programa que fije un error relativo mayor que 0.001, por lo que parará en esa iteración, y de esta manera sabremos que en la siguiente iteración el error relativo estará por debajo de esa cota.

Alternativamente podríamos conocer de forma conservadora las iteraciones necesarias para una precisión determinada empleado la siguiente fórmula:

$$|p - p_N| \leq \frac{b - a}{2^N}$$

Conclusión

En este post hemos explicado con carácter eminentemente práctico el método de la bisección para buscar una aproximación numérica a la solución de una ecuación. Es importante señalar que los errores absoluto y relativo que hemos definido en las programaciones con Maxima se refieren siempre a las estimaciones de la solución en sus respectivas iteraciones, pero no al error absoluto y relativo tomado cuando se conoce valor real de la solución.

En próximos posts, comentaremos otros métodos diferentes.

