Métodos de Computación Científica
Trabajo práctico de programación #5
14-11-2012

Victoria Martínez de la Cruz - LU. 87620

1) Para encontrar las raíces de las siguientes funciones usamos el método de Newton Raphson. Consideramos como error para el test de convergencia el épsilon machine (eps).

newton_raphson.m

```
function [ r ] = newton_raphson(f, df, xi, emax)
aux = xi;
e = 100;
i = 1;
h = 0;

fprintf('i     xi     f(xi)     df(xi)     |ep|\n')
fprintf('--------------------------------------\n')

fx = feval(f,xi);
dfx = feval(df,xi);
fprintf('%2d     %8d     %10.6f     %10.6f     %10.6f     \n', i, xi, fx, dfx, e);
i = i+1;
aux = xi;
h = xi - (fx/dfx);
xi = h;

    while e > emax
        fx = feval(f,xi);
        dfx = feval(df,xi);
        e = abs((xi-aux)/xi);
        fprintf('%2d     %8d     %10.6f     %10.6f     %10.6f     \n', i, xi, fx, dfx,
e);
        aux = xi;
        h = xi - (fx/dfx);
        xi = h;
        i = i+1;
    end
end
```

a) En este caso podemos hacer uso de la función roots(‘exp’), donde ‘exp’ es el polinomio dado representado como un vector.

```
>> c = [1 0 0 0 0 0 0 0 0 0 0 0 -2];
>> rootsC = roots(c)

rootsC =

  -1.0595
  -0.9175 + 0.5297i
  -0.9175 - 0.5297i
```

```
  -0.5297 + 0.9175i
  -0.5297 - 0.9175i
   0.0000 + 1.0595i
   0.0000 - 1.0595i
   0.5297 + 0.9175i
   0.5297 - 0.9175i
   1.0595
   0.9175 + 0.5297i
   0.9175 - 0.5297i
```

También podemos usar newton-raphson obteniendo resultados reales similares.

`roots()` también nos permite hallar raíces complejas, mientras que la implementación de Newton-Raphson utilizada no.

```
>> syms x
>> f = (x^12)-2

f =
x^12-2

>> df = diff(f,x)

df =
12*x^11


>> newton_raphson(inline('(x^12)-2'), inline('12*(x^11)'), -2, eps)
 i    xi                    f(xi)               fp(xi)              |ep|
---------------------------------------------------------------------------------
 1    -2                    4094.000000         -24576.000000       100.000000
 2    -1.833415e+000        1440.542270         -9441.675752        9.086067
 3    -1.680842e+000        506.537095          -3630.588250        9.077161
 4    -1.541323e+000        177.772337          -1399.621282        9.051920
 5    -1.414308e+000        62.051353           -543.457431         8.980688
 6    -1.300129e+000        21.325891           -215.294518         8.782117
 7    -1.201075e+000        7.012397            -90.043329          8.247156
 8    -1.123197e+000        2.031529            -43.072018          6.933606
 9    -1.076031e+000        0.409331            -26.869098          4.383319
10    -1.060797e+000        0.030417            -22.968588          1.436118
11    -1.059472e+000        0.000208            -22.655143          0.124993
12    -1.059463e+000        0.000000            -22.652984          0.000866
13    -1.059463e+000        0.000000            -22.652984          0.000000
14    -1.059463e+000        0.000000            -22.652984          0.000000
```

b)

```
>> f = atan(x)

f =
atan(x)

>> df = diff(f,x)

df =
1/(1+x^2)


>> newton_raphson(inline('atan(x)'),inline('1/(1+x^2)'), 0.5, eps)
 i    xi                    f(xi)         fp(xi)        |ep|
--------------------------------------------------------------------------
 1     5.000000e-001       0.463648      0.800000      100.000000
 2     -7.955951e-002      -0.079392     0.993710      728.460372
 3     3.353022e-004       0.000335      1.000000      23827.703039
 4     -2.513147e-011      -0.000000     1.000000      1334192472.167288
 5      0                  0.000000      1.000000       Inf
 6      0                  0.000000      1.000000       NaN
```

c)

```
>> f = sin(2*x)
f =
sin(2*x)

>> df = diff(f,x)

df =
2*cos(2*x)

>> newton_raphson(inline('sin(2*x)'), inline('2*cos(2*x)'), pi, eps)
 i    xi                    f(xi)         fp(xi)        |ep|
--------------------------------------------------------------------
 1    3.141593e+000        -0.000000     2.000000      100.000000
 2    3.141593e+000        -0.000000     2.000000      0.000000
```

d)

```
>> f = (((1.5*x)/(1+x^2)^2))-(0.65*(atan(1/x)))+((0.65*x)/(1+x^2))

f =
3/2*x/(1+x^2)^2-13/20*atan(1/x)+13/20*x/(1+x^2)
```

```
>> df = diff(f,x)

df =
3/2/(1+x^2)^2-6*x^2/(1+x^2)^3+13/20/x^2/(1+1/x^2)+13/20/(1+x^2)-13/10*x^2/(1+x^2)^2


>> newton_raphson(inline('3/2*x/(1+x^2)^2-13/20*atan(1/x)+13/20*x/(1+x^2)'),
inline('3/2/(1+x^2)^2-6*x^2/(1+x^2)^3+13/20/x^2/(1+1/x^2)+13/20/(1+x^2)-13/10*x^2/(1+x
^2)^2'), 0, eps)

 i     xi      f(xi)         fp(xi)        |ep|
------------------------------------------------------
 1     0       -1.021018     NaN           100.000000
 2     NaN      NaN          NaN           NaN
```

2) Declaramos a x y a y como variables simbólicas para utilizar la función predefinida `solve('exp1','exp2')`, donde exp1 y exp2 son las ecuaciones que conforman el sistema de ecuaciones a resolver.

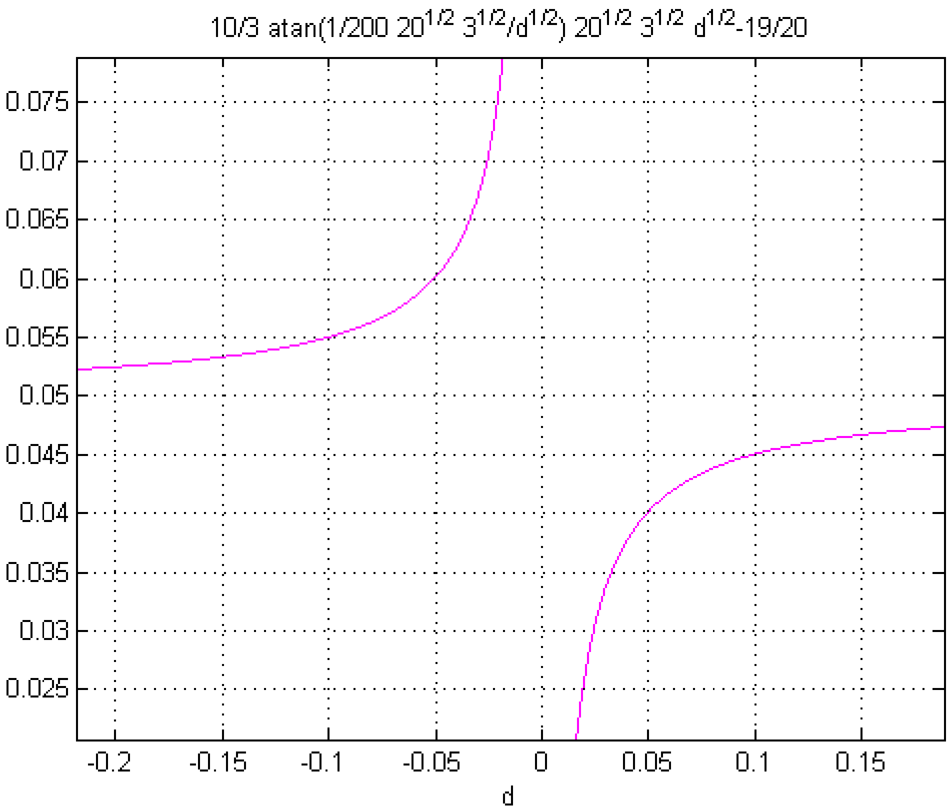
```
>> syms x, y

>> [x,y] = solve('(x^2)+(y^2)-8*x-4*y+11=0','(x^2)+(y^2)-20*x+75=0')

x =

 29/5+3/10*6^(1/2)
 29/5-3/10*6^(1/2)

y =

 7/5+9/10*6^(1/2)
 7/5-9/10*6^(1/2)
```

3)

```
>> syms d
>> A = d*d;
>> P = 4*d;
>> l = 0.1;
>> k = 240;
>> hinf = 9;
>> lambda = ((k*A)/(hinf*P))^(1/2)

lambda =
1/3*20^(1/2)*3^(1/2)*d^(1/2)

>> x = ((atan(l/lambda))/(l/lambda)) - 0.95

x =
10/3*atan(1/200*20^(1/2)*3^(1/2)/d^(1/2))*20^(1/2)*3^(1/2)*d^(1/2)-19/20
```

a)

```
>> g = ezplot(x), grid on;
>> set(g, 'Color', 'm');
```



Podemos ver que la función se aproxima al eje x (en este caso, d) en un entorno muy cercano al 0.
Estimamos que la raíz se encuentra en el intervalo (-0.05, 0) U (0, 0.05).

b)

```
>> format long
>> eps_step = 1e-5;
>> eps_abs = 1e-5;
```

<u>biseccion.m</u>

```
function [ r ] = biseccion( f, a, b, N, eps_step, eps_abs )

	% Controla que ningun punto extremo es raiz
	% y si f(a) y f(b) tienen el mismo signo, lanza una excepcion.

	if ( feval(f,a) == 0 )
		r = a;
		return;
	elseif ( feval(f,b) == 0 )
		r = b;
		return;
	elseif ( feval(f,a) * feval(f,b) > 0 )

		error( 'f(a) y f(b) no tienen signos opuestos' );
	end

	% Se itera N veces y si no es posible encontrar la raiz
	% luego de esas N iteraciones, se lanza una excepcion.

	for k = 1:N
	% Encuentra el punto medio
	c = (a + b)/2;

	% Controla si encontramos una raiz o no
	% y si debemos seguir iterando:
	%		[a, c] si f(a) y f(c) tienen signos opuestos, o
	%		[c, b] si f(c) y f(b) no tienen signos opuestos.

	if ( feval(f,c) == 0 )
		r = c;
		return;
	elseif ( feval(f,c)*feval(f,a) < 0 )
		b = c;
	else
		a = c;
	end
```

```
        % Si |b - a| < eps_step, controlar si
        %       |f(a)| < |f(b)| y |f(a)| < eps_abs y retornar 'a', o
        %       |f(b)| < eps_abs y retornar 'b'.

        if ( b - a < eps_step )
                if ( abs( feval(f,a) ) < abs( feval(f,b) ) && abs( feval(f,a) ) <
eps_abs )
                        r = a;
                        return;
                elseif ( abs( feval(f,b) ) < eps_abs )
                        r = b;
                        return;

                end
        end
end
        error( 'el metodo no converge' );
end
```

```
>> biseccionR =
biseccion(inline('10/3*atan(1/200*20^(1/2)*3^(1/2)/d^(1/2))*20^(1/2)*3^(1/2)*d^(1/2)-1
9/20'), 0, 2.0, 20, eps_step, eps_abs)

biseccionR =
    0.0091
```

c)

```
>> dx = diff(x,d)

dx =
-1/2/d/(1+3/2000/d)+5/3*atan(1/200*20^(1/2)*3^(1/2)/d^(1/2))*20^(1/2)*3^(1/2)/d^(1/2)

>> newton_raphsonR =
newton_raphson(inline('10/3*atan(1/200*20^(1/2)*3^(1/2)/d^(1/2))*20^(1/2)*3^(1/2)*d^(1
/2)-19/20'),inline('-1/2/d/(1+3/2000/d)+5/3*atan(1/200*20^(1/2)*3^(1/2)/d^(1/2))*20^(1
/2)*3^(1/2)/d^(1/2)'),1.0,eps)

 i     xi                   f(xi)        fp(xi)        |ep|
-------------------------------------------------------------------
 1     1                    -0.049500    -0.000499     100.000000
 2     -9.817914e+001       -0.050005    -0.000000     101.018546
 3     -9.640929e+005       -0.050000    -0.000000     99.989816
 4     -9.294760e+013       -0.050000    0.000000      99.999999
```

5	3.168770e+027	-0.050000	-0.000000	100.000000
6	-1.311649e+041	-0.050000	0.000000	100.000000
7	3.410807e+054	-0.050000	0.000000	100.000000
8	6.900457e+067	-0.050000	0.000000	100.000000
9	9.337970e+080	-0.050000	-0.000000	100.000000
10	-1.067847e+095	-0.050000	0.000000	100.000000
11	1.306245e+108	-0.050000	0.000000	100.000000
12	1.994418e+121	-0.050000	0.000000	100.000000
13	7.498370e+133	-0.050000	0.000000	100.000000
14	4.397109e+146	-0.050000	0.000000	100.000000
15	3.330561e+159	-0.050000	0.000000	100.000000
16	2.292733e+172	-0.050000	-0.000000	100.000000
17	-1.325293e+185	-0.050000	0.000000	100.000000
18	5.101840e+197	-0.050000	0.000000	100.000000
19	2.382802e+210	-0.050000	0.000000	100.000000
20	1.160980e+223	-0.050000	-0.000000	100.000000
21	-2.394562e+235	-0.050000	-0.000000	100.000000
22	-2.121349e+248	-0.050000	0.000000	100.000000
23	1.185415e+261	-0.050000	-0.000000	100.000000
24	-4.431485e+273	-0.050000	0.000000	100.000000
25	8.278064e+286	-0.050000	-0.000000	100.000000
26	-4.074219e+299	-0.050000	0.000000	100.000000
27	Inf	NaN	NaN	NaN

d)

<u>regula_falsi.m</u>

```
function [ r ] = regula_falsi( f, a, b, N, eps_step, eps_abs )

	% Controla que ningun punto extremo es raiz
	% y si f(a) y f(b) tienen el mismo signo, lanza una excepcion.

	if ( feval(f,a) == 0 )
		r = a;
		return;
	elseif ( feval(f,b) == 0 )
		r = b;
		return;
	elseif ( feval(f,a) * feval(f,b) > 0 )
		error( 'f(a) y f(b) no tienen signos opuestos' );
	end

	% Se itera N veces y si no es posible encontrar la raiz
	% luego de esas N iteraciones, se lanza una excepcion.
	c_old = Inf;
```

```
	for k = 1:N

	% Encuentra la posicion falsa
	c = (a*feval(f,b) + b*feval(f,a))/(feval(f,b) - feval(f,a));



	% Controla si encontramos una raiz o no
	% y si debemos seguir iterando:
	%		[a, c] si f(a) y f(c) tienen signos opuestos, o
	%		[c, b] si f(c) y f(b) no tienen signos opuestos.

	if ( feval(f,c) == 0 )
		r = c;
		return;
	elseif ( feval(f,c)*feval(f,a) < 0 )
		b = c;
	else
		a = c;
	end

	% Si |b - a| < eps_step, controlar si
	%	|f(a)| < |f(b)| y |f(a)| < eps_abs y retornar 'a', o
	%	|f(b)| < eps_abs y retornar 'b'.




	if ( abs( c - c_old ) < eps_step )
		if ( abs( feval(f,a) ) < abs( feval(f,b) ) && abs( feval(f,a) ) <
eps_abs )
			r = a;
			return;
		elseif ( abs( feval(f,b) ) < eps_abs )
			r = b;
			return;
		end
	end

	c_old = c;

	end
	error( 'el metodo no converge' );
end

>> regula_falsiR =
regula_falsi(inline('10/3*atan(1/200*20^(1/2)*3^(1/2)/d^(1/2))*20^(1/2)*3^(1/2)*d^(1/2
)-19/20'), 0, 1, 1000, eps_step, eps_abs)
```

```
regula_falsiR =
   0.0091 - 0.0000i
```