Universidad del Atlántico
MATUA
Facultad de Ciencias Básicas
©Programa de Matemáticas
Vol. 04 , Nº 2, (2017)
Revista De Matemáticas de la Universidad del Atlántico Páginas: 1–15

Cuencas de atracción usando MatLab®

Basins of attraction using MatLab®

Carlos E. Cadenas R.[a], César V. Luna M.[a]

ccadenas@uc.edu.ve; ccadenas45@gmail.com; cluna.m@gmail.com

[a]*Departamento de Matemáticas, Facultad Experimental de Ciencias y Tecnología. Universidad de Carabobo, Venezuela.*
Centro Multidiciplinario de Cómputo de Visualización Científica.

Resumen

Existen muchos métodos numéricos para encontrar la solución de una ecuación no lineal de la forma $f(x) = 0$, donde f es una función real o compleja, entre otros tipos de funciones. Estos métodos pueden ser clasificados tomando en cuenta el orden de convergencia y algunas medidas de eficiencia. Otro criterio que ha sido utilizado en las últimas décadas, debido a la valiosa información que brinda, buen criterio de comparación es la cuenca de atracción de estas soluciones que brinda sorprendentes figuras que destaca el comportamiento de estos métodos dado diferentes puntos iniciales. Aquí presentamos un algoritmo en MatLab® que permite la construcción de la cuenca de atracción de los puntos fijos de la función de iteración de diversos esquemas iterativos. También se presentan diversos planos dinámicos de los métodos de Newton, Halley y Jarratt para raíces simples, así como sus adaptaciones a raíces con multiplicidad conocida, con resultados que prueban el desempeño del algoritmo propuesto.

Palabras claves:
cuenca de atracción, MatLab®, métodos iterativos, ecuaciones no lineales.

Abstract

There are many numeric methods to find the solution of a nonlinear equation of the form $f(x) = 0$, where f is a real or complex function among others types of functions. Taking into account the order of convergence and some measures of efficiency these methods can be classified. Another criterion that has been used in the last decades, due to the valuable information it provides, good criterion of comparison is the basin of attraction of these solutions that provides surprising figures that highlight the behavior of these methods given different starting points. Here we present an algorithm in MatLab® that allows the construction of the basin of attraction of the fixed points of the iteration function of several iterative schemes. Dynamic planes of the Newton, Halley and Jarratt methods for simple roots are presented, as well as their adaptations to roots with known multiplicity, with results that show the performance of the proposed algorithm.

Keywords:
basin of attraction, MatLab®, iterative scheme, nonlinear equations.

1. Introducción

En la literatura se encuentra una inmensa cantidad de métodos para calcular los ceros simples o múltiples de una ecuación no lineal, por lo que es indispensable tener herramientas que permitan conocer de cierta forma que métodos tiene mejor comportamiento que otros para cierta clases de problemas. Podemos clasificar los métodos por su orden de convergencia, cantidad de datos iniciales, cantidad de funciones evaluadas en cada iteración y medidas de eficiencias. Gracias al avance tecnológico de las últimas décadas se puede disponer de herramientas computacionales que permitan representar las cuencas de atracción de dicho ceros simples o múltiples. Por medio de las cuencas de atracción se realizan comparaciones eficientes entre métodos destacando su comportamiento a partir de diversos puntos iniciales. Existen muchos software que realizan este tipo de estudio, destacando entre ellos Mathematica®. Por ejemplo, en [1] se presenta una comparación gráfica (cuenca de atracción) y numérica (medidas de eficiencia) de métodos iterativos conocidos, describiendo un programa en Mathematica® con el cual se obtiene de manera partícular las cuencas de atracción de las raíces de la función $f(z) = z^3 - 1$. En [2] se implementa y explica un algoritmo en Mathematica® para construir las cuencas de atracción de las raíces de la función $f(z) = z^3 + (r-1)z - r$ para $r = 0, 1, \frac{1}{2}$, así como de la función $f(z) = z^7 - 1$, para el caso del método de Newton. En [3] se presenta un algoritmo en Mathematica® para analizar la cuenca de atracción de un punto fijo, en particular para la función $f(z) = z^3 + 0{,}33 + 0{,}35i$. Un algoritmo en Mathematica® para generar, utilizando el método de Newton, cuencas de atracciones de diferentes ceros de polinomios es detallado en [4].

En este documento se propone un algoritmo flexible y eficiente implementado en MatLab® que permite la representación gráfica de la cuenca de atracción de los ceros simples o múltiples de cualquier método iterativo, esto con el fin de brindar una nueva alternativa para hacer la comparación gráfica de métodos iterativos para resolver ecuaciones no lineales.

El resto del artículo es organizado como sigue: la Sección 2 esta formada por conceptos preliminares; en la Sección 3 se desarrolla el algoritmo propuesto; en la Sección 4 se presentan los planos dinámicos para diversos métodos usando varias ecuaciones de prueba; en la Sección 5 se establece una discusión y conclusiones.

2. Preliminares

Con el fin de tener una mejor comprensión del texto, se presentan algunas definiciones básicas de dinámica compleja.

Sea $R : \hat{\mathbb{C}} \to \hat{\mathbb{C}}$ un mapeo racional sobre la esfera de Riemann:

2.1. *Para $z \in \hat{\mathbb{C}}$ se define órbita como el conjunto $O(z) = \{z, R(z), R^2(z), \cdots, R^n(z), \cdots\}$.*

2.2. *Un punto complejo z_0 es un punto fijo de R si $R(z_0) = z_0$.*

2.3. *z_0 es un punto períodico para R si existe $k \geq 1$ tal que $R^k(z_0) = z_0$. se denomina período de z_0 al valor $p = min\{k \geq 1 | R^k(z_0) = z_0\}$.*

2.4. *Considérese la ecuación de iteración de cualquier método iterativo de la siguiente forma:*

$$z_{n+1} = G(z_n), \quad n = 0, 1, 2, \cdots$$

la cual genera la sucesión $\{z_n\}_{n=0}^{\infty}$. Además si se denota como α a un punto fijo de G, entonces se denomina cuenca de atracción de α al conjunto de puntos $z_0 \in \mathbb{C}$ tales que la sucesion $\{z_n\}_{n=0}^{\infty}$ generada por G converge a α.

A continuación se presentan los seis métodos que se estarán utilizando a lo largo de este documento a efectos de hacer las representaciones gráficas de los planos dinámicos, destacándose el orden de convergencia r y la ecuación de iteración, tanto para el caso de raíces simple como para el caso de raíces múltiples con multiplicidad m conocida:

1. Método de Newton [5] ($r = 2$)

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, \cdots \tag{1}$$

2. Método de Halley [6] ($r = 3$)

$$x_{n+1} = x_n - \frac{f(x_n)f'(x_n)}{[f'(x_n)]^2 - \frac{1}{2}f(x_n)f''(x_n)}, \quad n = 0, 1, \cdots \tag{2}$$

3. Método de Jarratt [7] ($r = 4$)

$$\begin{aligned} y_n &= x_n - \frac{2f(x_n)}{3f'(x_n)} \\ x_{n+1} &= x_n - \left[1 - \frac{3}{2}\frac{f'(y_n) - f'(x_n)}{3f'(y_n) - f'(x_n)}\right]\frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, \cdots \end{aligned} \tag{3}$$

4. Variante del método de Newton descrito en [8] ($r = 2$) :

$$x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, \cdots \tag{4}$$

5. Una adaptación para raíces múltiples del método de Halley desarrollada en [9] ($r = 3$):

$$x_{n+1} = x_n - \frac{f(x_n)f'(x_n)}{\frac{m+1}{2m}[f'(x_n)]^2 - \frac{1}{2}f(x_n)f''(x_n)}, \quad n = 0, 1, \cdots \tag{5}$$

6. Una modificación del método de Jarrat derivado en [10] ($r = 4$), donde $p = \left(\frac{m}{m+2}\right)^m$:

$$\begin{aligned} y_n &= x_n - \frac{2m}{m+2}\frac{f(x_n)}{f'(x_n)} \quad n = 0, 1, \cdots \\ x_{n+1} &= x_n - \frac{m}{8}\left[(m^3 - 4m + 8)\frac{f(x_n)}{f'(x_n)} - (m+2)^2 p\left(2(m-1) - (m+2)p\frac{f'(x_n)}{f'(y_n)}\right)\frac{f(x_n)}{f'(y_n)}\right] \end{aligned} \tag{6}$$

3. Algoritmo en Matlab

En esta sección se describe el funcionamiento del algoritmo propuesto. Si se considera el iterado inicial $z_0 \in \mathbb{C}$ y el esquema iterativo $z_{n+1} = G(z_n)$, el cual bajo cierta condiciones converge a las raíz de una función, entonces al distinguir con un color diferente los conjuntos de puntos que convergen a cada raíz de la función, se puede generar mediante el uso del algoritmo propuesto, imágenes que muestran la complejidad de dichas regiones, lo cual permite observar el comportamiento del esquema utilizando dichos puntos iniciales, además de observar la convergencia de estos a las respectivas raíces de la función estudiada.

El procedimiento utilizado aquí para obtener las cuencas de atracción de los ceros simples o múltiples de f empleando un método numérico se resume como sigue:

- **Paso 1**: introducir la función, derivadas y parámetros según el método numérico a utilizar.
- **Paso 2**: introducir las raíces de la función dada en el **Paso 1**.
- **Paso 3**: hacer la partición de los intervalos $[a, b]$ y $[c, d]$ utilizando N y M puntos igualmente espaciados en cada uno de ellos y realizar el producto cartesiano de dichas particiones para obtener $N \times M$ puntos en el plano complejo, es decir, puntos de la forma $z_0 = u + iv$, donde $u \in [a, b]$, $v \in [c, d]$ y $i = \sqrt{-1}$.
- **Paso 4**: utilizar cada punto generado en el **Paso 3** como punto inicial, z_0, del método seleccionado, utilizando un número máximo de iteraciones de $Niter$ y una tolerancia ϵ para el error absoluto cometido al calcular las diversas aproximaciones de α.
- **Paso 5**: asignar un color diferente a cada punto inicial z_0 del **Paso 4** de acuerdo a que raíz converge. Si el método supera el valor máximo de iteraciones sin satisfacer la tolerancia se considerará como no convergencia o convergencia lenta a la raíz y será pintado de un color diferente a los puntos iniciales que cumplan con las condiciones de convergencia preestablecida para las raíces.
- **Paso 6**: graficar lo obtenido en el **Paso 5**.

Observaciones: En el **Paso 3**, los valores de $\{a, b, c, d\}$ deben ser seleccionados tomando en cuenta la ubicación de la raíces introducidas en **Paso 2**. Se debe notar que en el paso **Paso 5** se necesitan $Nr + 1$ colores, donde Nr denota la cantidad de raíces de la función.

Debido a que los autores no encontraron un mapa de colores que cumpliera satisfactoriamente los requerimientos del algoritmo (**Paso 5**) se desarrolló el Algoritmo 1, el cual permite obtener un mapa de colores personalizado que hiciera esta labor de manera amigable; el cual se explica a continuación:

1. Lineas (8–16): se valida que el número de colores no exceda de 20, ya que aquí se trabaja con una cantidad de 20 colores diferentes, pero esto se puede cambiar simplemente agregando o quitando colores en el formato RGB de acuerdo a lo que se requiera, de igual manera se valida que la degradación sea un número comprendido entre 0 y 1.
2. Lineas (17–21): se construye una matriz con un total de 20 colores divididos entres los colores primarios, secundarios y otros colores adicionales. En esta parte se puede mezclar, quitar o agregar colores a conveniencia.
3. Lineas (25–28): se crea una matriz de tal forma que en sus posiciones impares se representen los colores de la matriz RGB y en sus posiciones pares se degrade dicho color cierto porcentaje (entre 0 y 1).
4. Lineas (33–41): se obtienen los valores de interpolación lineal entre un color y sus respectiva degradación de acuerdo a un número de tonalidades predefinido. Esto nos brinda una matriz de salida que posee tantas tonalidades como se requieran entre el color y su respectiva degradación.

Algoritmo 1. Mapa de Colores

```
1 function [map] = MatrixColors(Ncolor,degra,Ntonal)
2 %in:
3 %Ncolor: Number of colors to building a colormap (Max 20 colour).
4 %degra: Grade of shade
5 %Ntonal: Number of tonalities.
```

```
%output:
%map: Represent a matrix of color
if   Ncolor <= 0 || Ncolor>20
     disp('Error, Ncolor must be less than or equal to 20')
     return;
     else
          if degra<0 || degra>1
               disp('Error, degra must be between 0 and 1')
               return;
          end
end
T=zeros(2*Ncolor,3);%Preallocating Arrays
PrimaryColor=255*eye(3)';%Red, Green, Blue.
SecundaryColor=(255*ones(3)-255*eye(3))'; %Cyan, Purple,Yellow.
OtherColors=[123 104 238;112 128 144;25 25 112;188 143 143;244 164 96;176 224 230;255 228
      181;25 200 120;65 105 225;105 92 92; 64 224 208;250 128 114;100 80 0;255 100 20];
RGB=[PrimaryColor;SecundaryColor;OtherColors];%Matrix with all color
%T(1,:) and T(2,:) are zero to represent a black color.
%T(odd,:) Represent a color of RGB.
%T(odd+1:) Represent a degradation of the above color.
for j=2:Ncolor
     T(2*j-1,:)=RGB(j-1,:);
     T(2*j,:)=degra*RGB(j-1,:);
end
%Identify every color in the Matrix T
%The colors of the RGB matrix are identify by the odd position of vector R.
%and the degradation of the each color are identify by the par position.
%For example: R(1)->Black; R(2)->0.2Black, R(3)->Red; R(4)->0.2Red, ...
R=linspace(0,Ntonal*Ncolor,size(T,1));
map=[];
%Building of the degradation of every color in RGB
for k=1:Ncolor
     U=linspace(R(2*k-1),R(2*k),Ntonal);
     M=interp1(R,T,U);
     map=[map;M]; %Interpolate all points in U.
end
map=map./255;%RGB colors in [0,1]
end
```

En la Figura 1 se aprecian 20 colores predefinidos con una degradación de 0.2 y un total de 9 tonalidades, esta puede ser obtenida utilizando los siguientes comando:

```
[map] = MatrixColors(10,0.2,9);
I=1:Ntonal*Ncolor;
imagesc(I)
colorbar
colormap(map)
```

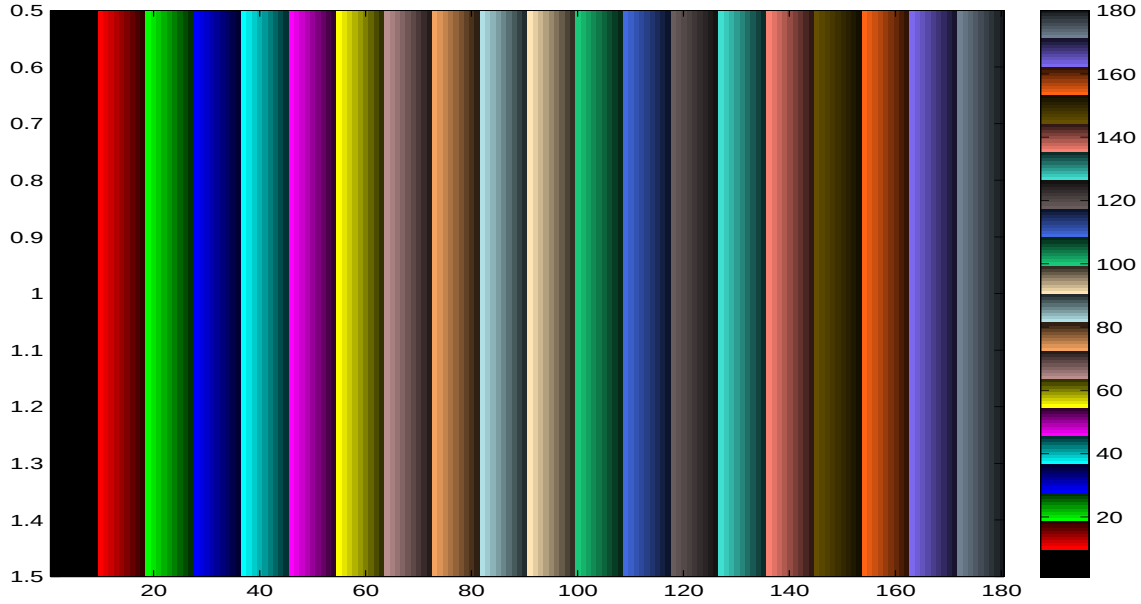

Figura 1. Mapa de colores.

Ahora se presenta el Algoritmo 2, el cual es utilizado para generar la cuenca de atracción de los ceros simples o múltiples de la función f. Dicho código se explica a continuación:

1. Lineas (12–14): corresponde al cálculo de derivadas necesitadas por el método a estudiar.
2. Lineas (15–18): se calculan las raíces de la función introducida con el comando **solve**.
3. Lineas (20–21): construcción del mallado de puntos iniciales.
4. Linea (23): se establece el número de tonalidades y luego se calcula una matriz con la cantidad de colores y degradaciones de los mismo utilizando el Algoritmo 1.
5. Lineas (24–35): se realiza un doble ciclo **for** para recorrer todos los puntos del mallado realizado anteriormente y los cuales serán puntos iniciales de los métodos seleccionados.
6. Linea (28): en esta parte se introduce el método a estudiar para la aproximación de la raíz, como ejemplo tenemos los métodos (1–6), cuyos código los podemos ver en los Algoritmos (3–8). Notar que el algoritmo se adapta a cualquier método numérico de su interés.
7. Lineas (30–33): se emplea la función **find**, para conocer a la raíz a la cual el método converge y así poder ubicar en la barra de colores el color del punto inicial utilizado. En la linea (32) se utiliza la función

$$g(iter) = a - b * exp\left(\frac{-log(15)(iter-1)}{10}\right) \quad (a > b) \tag{7}$$

para establecer la velocidad de convergencia tomando en cuenta la cantidad de iteraciones realizadas; mientras menor sea el número de iteraciones se tiene colores claros, en el caso de que aumente la cantidad de iteraciones los colores se tornarán oscuros.
La función dada en (7) satisface las siguientes propiedades:

- g es creciente.
- $a - b \leq g(iter) \leq a$.
- $g(1) = a - b$ y $g(iter) = a$ cuando $iter \to \infty$

Cuando no ocurra convergencia o la misma sea lenta se le asignará a los puntos iniciales el color negro.

8. Lineas (37–41): se crea la cuenca de atracción utilizando el comando **colormap** junto con la matriz de colores realizada anteriormente, el comando **caxis** para fijar los intervalos de los colores en la barra colores y el comando **pcolor**, por último sobre esta imagen se grafícan las raíces de la función con el comando **plot**.

Algoritmo 2. Basins of attraction

```
function [] = Basinsofattraction(f,a,b,c,d,max_X,max_Y,Niter,tol,NDeri)
% Basinsofattraction obtains the basins of attraction for numerical methods to find
    simple roots of nonlinear equations
% in
% f: nonlinear function (@(x)x^3-1)
% a,b,c,d: Represent the closed interval [a,b] and [c,d] in the axis x and define the
    rectangle for possible values to initializing the methods
% max_X, max_Y: partition size of [a,b] and [c,d]
% Niter: is the maximum number of iterations of the method.
% tol and Nderi: tolerance and number of derivative for the numerical method
% test: Basinsofattraction(@(x)x^3-1,-3,3,-3,3,600,600,30,10^-6,1);
syms x
% Derivatives are calculated and converted to inline functions
for k=1:NDeri
        eval(['d' num2str(k) 'f' ' = matlabFunction(diff(f(x),k));' ]);
end
r1=solve(f(x)==0); r1=unique(r1); r=zeros(size(r1,2)); % roots
for k=1:size(r1,1)
    r(k)=r1(k,1);
end
%Mesh of points
x = linspace(a,b,max_X); y = linspace(c,d,max_Y); [X,Y] = meshgrid(x,y);
W = zeros(max_X,max_Y);%Preallocating Arrays
%colour matrix for the colormap
Ntonal=40; map = MatrixColors(size(r,2)+1,0.3,Ntonal)
for k = 1:size(X,2)
    for j = 1:size(X,1)
        z0=X(j,k)+1i*Y(j,k); % Initial value
        %Examples methods (call your method here)
        [ra,iter]= Met_Newton(z0,f,d1f,Niter,tol);
        %Colour of each point
        if iter<=Niter && abs(f(ra))<tol
            pos=find(abs(r-ra)<0.1,1,'first'); %find the position of the aproximate root
            W(j,k)= (pos+1)*s-s*exp(-log(15)*(iter-1)/10);%Light to Dark
        end
    end
end
%Image display
axis([a,b,c,d]); axis square; colormap(map)
hold on; caxis([0,(size(r,2)+1)*Ntonal])
pcolor(X,Y,W); hold on
plot(real(r),imag(r),'o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',2);
shading flat;
end
```

Se recomienda guardar la imágenes utilizando las siguientes lineas de comando, las cuales nos permiten definir la resolución y formato de la misma, para este caso tenemos un imagen en formato .eps y resolución de 500pp.

```
figure
Basinsofattraction(@(x)x^3-1,-3,3,-3,3,1000,1000,60,10^-4,1);
print -depsc2 -r500 Figure.eps
```

A continuación se presentan sin mayor detalle los códigos de los seis métodos clásicos escogidos para mostrar el desempeño del algoritmo propuesto en este trabajo:

Algoritmo 3. The Newton method

```
function [z,iter] = Met_Newton(z,f,df,N,e)
% The Newton-Raphson method is a second-order method solving nonlinear equations.
% variable
% IN:
% z: Initial points.
% f and df: Function and derivate of f.
% e and N: tolerance and the maximan value of iteration
% OUTPUT
% z: root of f
% iter: iteration
k=1;    % Count the number of iteration
band=1; % Flag
while abs(band)>e && k<=N
    s=f(z)/df(z); z=z-s; band=f(z); k=k+1;
end
iter=k-1;
end
```

Algoritmo 4. The Halley method

```
function [r,iter] = Met_Halley(z,f,df,d2f,N,e)
% The Halley Method, is a third-order method
k=1; band=1;
while abs(band)>e && k<=N
    u=f(z); v=df(z); w=d2f(z); z=z-(u*v)/(v^2-0.5*u*w); band=f(z);   k=k+1;
end
iter=k-1;
end
```

Algoritmo 5. The Jarratt method

```
function [z,iter] = Met_Jarratt(z,f,df,N,e)
% The Jarratt method is a fourth-order method
k=1; band=1;
while abs(band)>e && k<=N
    u=f(z); v=df(z); y=z-(2*u)/(3*v); w=df(y);
    z=z-(1-(1.5)*(w-v)/(3*w-v))*(u/v); band=f(z); k=k+1;
end
iter=k-1;
end
```

Algoritmo 6. The Newton Multiple method

```
function [z,iter] = Met_Newton_Multiple(z,f,df,N,e,m)
% The Newton-Raphson method to multiple root, is a second-order method
k=1; band=1;
while abs(band)>e && k<=N
    s=m*f(z)/df(z); z=z-s; band=f(z); k=k+1;
end
iter=k-1;
end
```

Algoritmo 7. The Halley Multiple method

```
function [z,iter] = Met_Halley_Multiple(z,f,df,d2f,N,e,m)
% The Halley Method to multiple root, is a third-order methods
k=1; band=1;
while abs(band)>e && k<=N
    u=f(z); v=df(z); w=d2f(z);
    z=z-(u*v)/(((m+1)*v^2)/(2*m)-0.5*u*w); band=f(z); k=k+1;
end
iter=k-1;
end
```

Algoritmo 8. The Jarratt Multiple method

```
function [z,iter] = Met_Jarratt_Multiple(z,f,df,N,e,m)
%The Jarratt method to multiple root is a fourth-order method
k=1; band=1;
while abs(band)>e && k<=N
    u=f(z); v=df(z); y=z-(2*m*u)/((m+2)*v); w=df(y);
    z=z-(m/8)*((m^3-4*m+8)*(u/v)-(m+2)^2*(m/(m+2))^m*(2*(m-1)-(m+2)*(m/(m+2))^m*(v/w))*(u
        /w));
    band=f(z); k=k+1;
end
iter=k-1;
end
```

4. Resultados

En esta sección, para comprobar el desempeño del algoritmo propuesto, se presentan la cuenca de atración de ceros simples y múltiples de varios polinomios, ampliamente utilizados en la literatura, empleando los métodos (1)-(6). Todos los ejemplos poseen raíces en $[-3,3]\times[-3,3]$. Se utilizan 1000 puntos igualmente espaciados en el intervalo cerrado $[-3,3]$ y se realiza el producto cartesiano de las particiones construidas, para obtener un total de 1000000 puntos en el plano complejo donde cada uno de ellos se usan como puntos iniciales de los esquemas iterativos, además se utiliza un número máximo de 60 iteraciones y una tolerancia de 10^{-4}.

Ejemplo 1: se utiliza el polinomio de grado tres

$$p_1(z) = z^3 - 1$$

estudiado en [11], [12] y [13].

Se denotan las tres raíces de $p_1(z)$ como $r_1 = 1$, $r_2 = -0{,}5 + 0{,}86603i$ y $r_3 = -0{,}5 - 0{,}86603i$, donde las cuencas de atracción de r_1, r_2 y r_3 son pintadas de rojo, azul y verde respectivamente, a la no convergencia o convergencia lenta se le asigna el color negro.

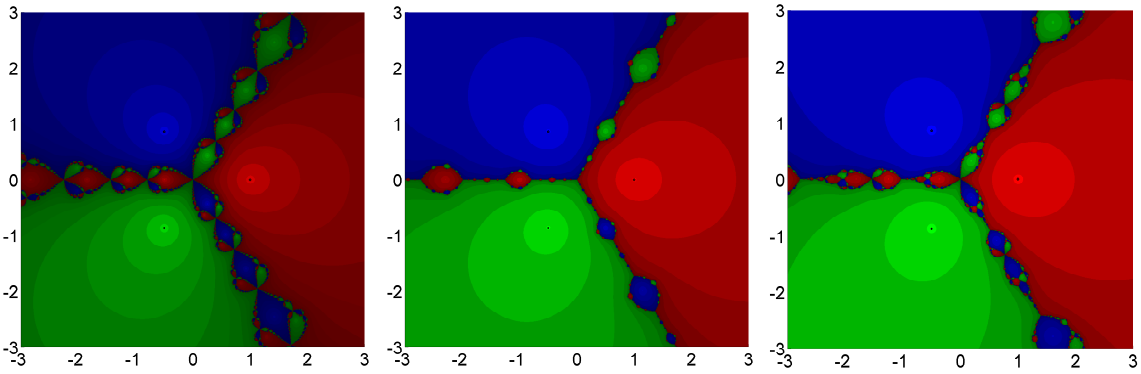

Figura 2. Cuenca de atracción del método de Newton (izquierda), método de Halley (centro) y método de Jarratt (derecha).

Ejemplo 2: se selecciona el polinomio de grado cuatro

$$p_2(z) = z^4 - 10z^2 + 9$$

estudiado en [11], [12] y [13].

Las raíces de $p(z)$ son: $r_1 = -3$ (Rojo); $r_2 = -1$ (Verde); $r_3 = 2$ (Azul); y $r_4 = 3$ (Aguamarina), la no convergencia o convergencia lenta es pintada en negro.

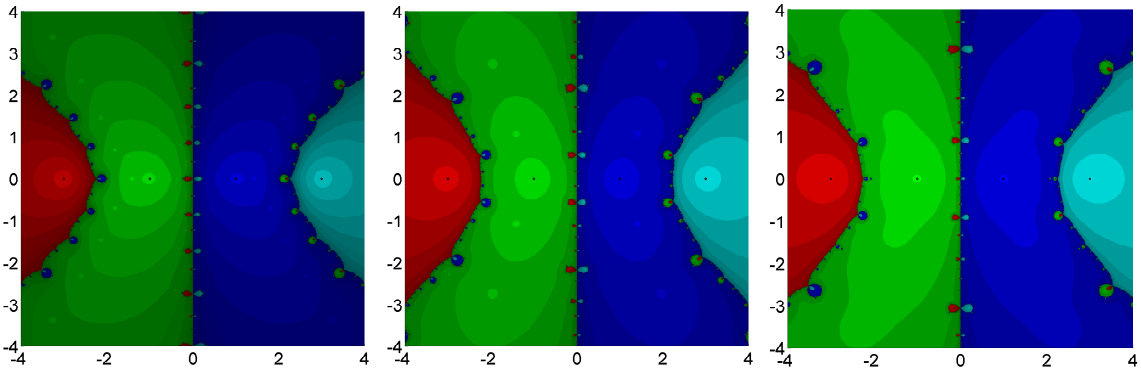

Figura 3. Cuenca de atracción del método de Newton (izquierda), método de Halley (centro) y método de Jarratt (derecha).

Ejemplo 3: polinomio de grado seis con coeficientes complejos

$$p_3(z) = z^6 - \frac{1}{2}z^5 + \frac{11}{4}(1+i)z^4 - \frac{1}{4}(19+3i)z^3 + \frac{1}{4}(11+5i)z^2 - \frac{1}{4}(11+i)z + \frac{3}{2} - 3i$$

estudiado en [11], [12] y [13].

Las seis raíces de $p_3(z)$ vienen dadas por: $r_1 = 1$; $r_2 = -1+2i$; $r_3 = -\frac{1}{2} - \frac{i}{2}$; $r_4 = i$; $r_5 = -\frac{3i}{2}$ y $r_6 = 1-i$ y los colores respectivos de sus cuencas de atracción son: rojo, aguamarina, amarillo, verde, magenta y azul, la no convergencia o convergencia lenta en negro.

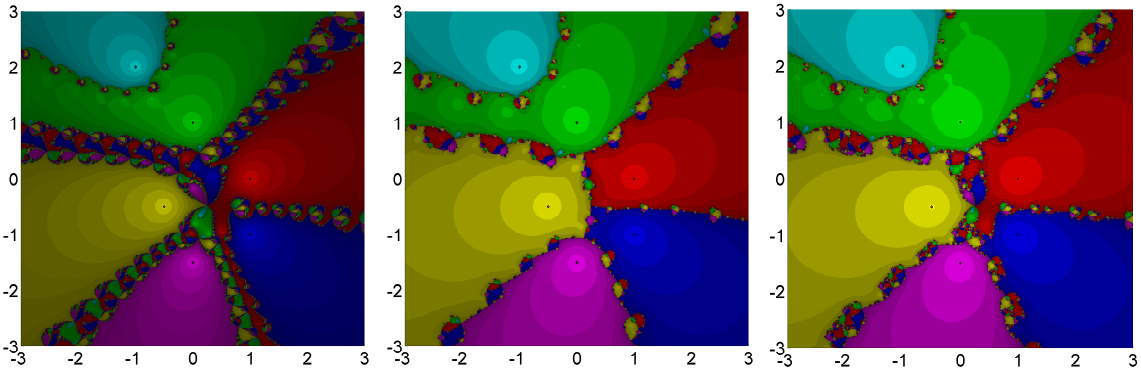

Figura 4. Cuenca de atracción del método de Newton (izquierda), método de Halley (centro) y método de Jarratt (derecha).

Ejemplo 4: polinomio de grado nueve $p_4(z) = z^9 - 1$

Se denotan las raíces de $p_4(z)$ como sigue: $r_1 = 1$ (Rojo), $r_2 = 0{,}76604 + 0{,}64279i$ (Verde), $r_3 = 0{,}17365 + 0{,}98481i$ (Azul), $r_4 = -0{,}50000 + 0{,}86603i$ (Amarillo), $r_5 = -0{,}93969 + 0{,}34202i$ (Aguamarina), $r_6 = -0{,}93969 - 0{,}34202i$ (Morado), $r_7 = -0{,}50000 - 0{,}86603i$ (Magenta), $r_8 = 0{,}17365 - 0{,}98481i$ (Marrón) y $r_9 = 0{,}76604 - 0{,}64279i$ (Cobre) la no convergencia en negro.

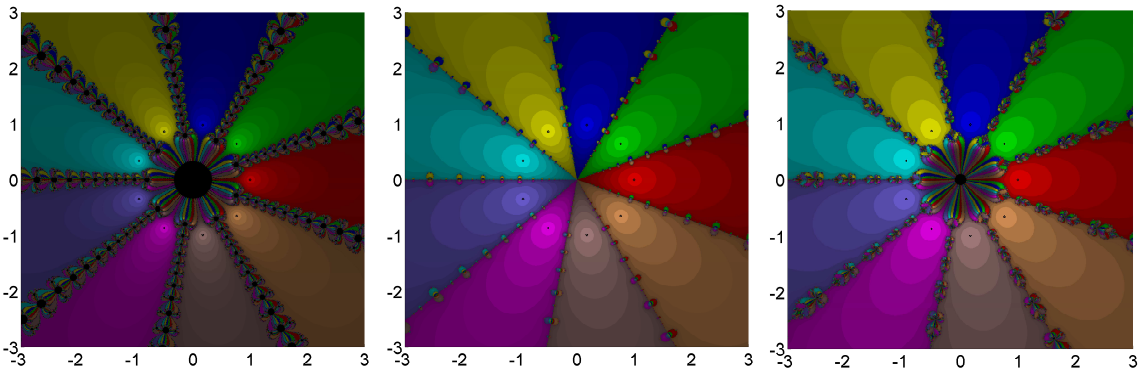

Figura 5. Cuenca de atracción del método de Newton (izquierda), método de Halley (centro) y método de Jarratt (derecha).

Ejemplo 5: polinomio de grado cuatro

$$p_5(z) = (z^2 - 1)^2$$

estudiado en [14].

Las raíces $r_1 = -1$ y $r_2 = 1$ son ambas reales y de multiplicidad $m = 2$, la cuenca de atracción de r_1 y r_2 se pinta de rojo y verde respectivamente, la no convergencia o convergencia lenta en negro.

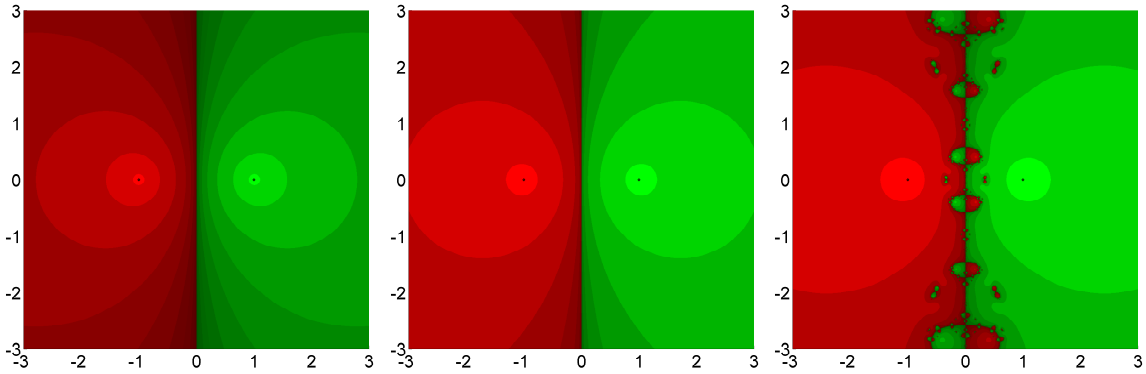

Figura 6. Cuenca de atracción (raíces múltiples) de la variante del método de Newton (izquierda), adaptación del método de Halley (centro) y modificación del método de Jarratt (derecha).

Ejemplo 6: polinomio cuyas raíces son todas de multiplicidad tres

$$p_6(z) = (z^3 + 4z^2 - 10)^3$$

estudiado en [14].

Las raíces de $p_6(z)$ son: $r_1 = -2{,}68261500670705 - 0{,}35829359924043i$ (Azul); $r_2 = -2{,}68261500670705 + 0{,}35829359924043i$ (Verde) y $r_3 = 1{,}3652300134141$ (Rojo), la no convergencia o convergencia lenta en negro.

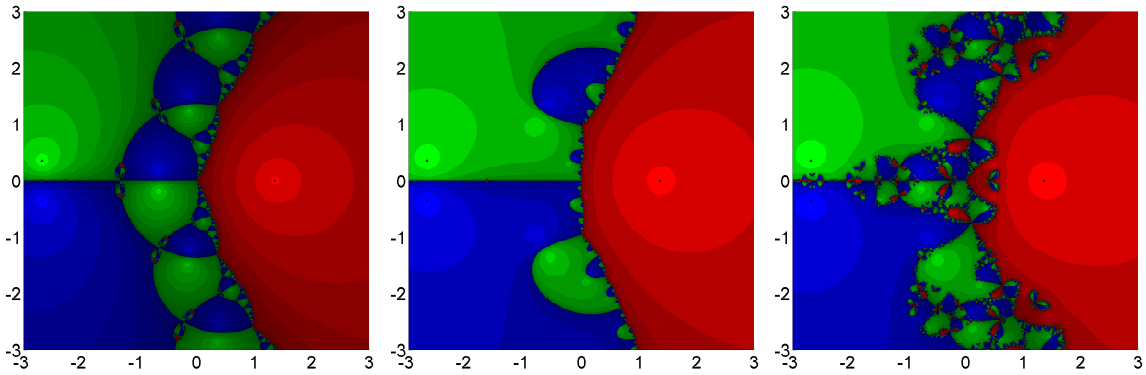

Figura 7. Cuenca de atracción (raíces múltiples) de la variante del método de Newton (izquierda), adaptación del método de Halley (centro) y modificación del método de Jarratt (derecha).

Ejemplo 7: polinomio cuyas raíces son todas de multiplicidad cuatro, las raices son las tres raíces de la unidad

$$p_7(z) = (z^3 - 1)^4$$

estudiado en [14].

Se denotan las tres raíces de $p_7(z)$ como $r_1 = 1$, $r_2 = -0{,}5 + 0{,}86603i$ y $r_3 = -0{,}5 - 0{,}86603i$, donde los colores de las cuencas de atracción de r_1, r_2 y r_3 se pintan de rojo, azul y verde respectivamente, la no convergencia o convergencia lenta en negro.

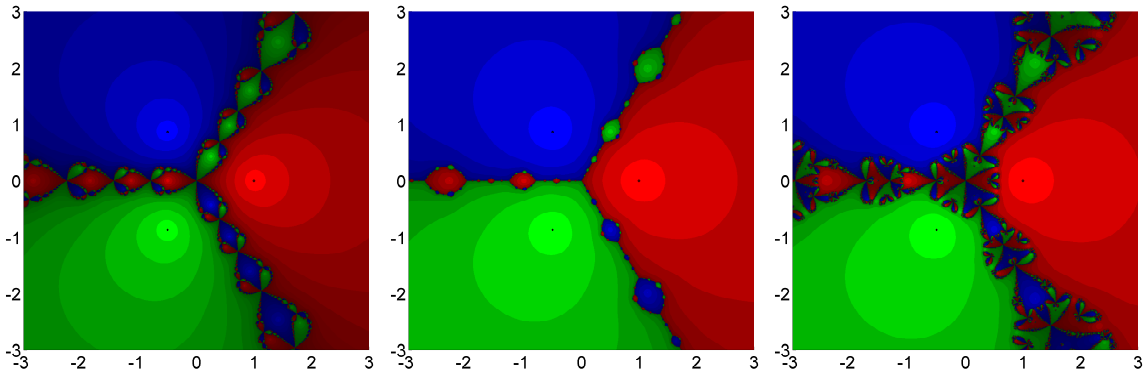

Figura 8. Cuenca de atracción (raíces múltiples) de la variante del método de Newton (izquierda), adaptación del método de Halley (centro) y modificación del método de Jarratt (derecha).

Ejemplo 8: el último polinomio

$$p_8(z) = (z^4 - 1)^5$$

Se denotan las raíces de $p_8(z)$ como sigue: $r_1 = 1$ (Verde), $r_2 = -1$ (Rojo), $r_3 = 1i$ (Aguamarina) y$r_4 = -1i$ (Azul), la no convergencia o convergencia lenta en negro.

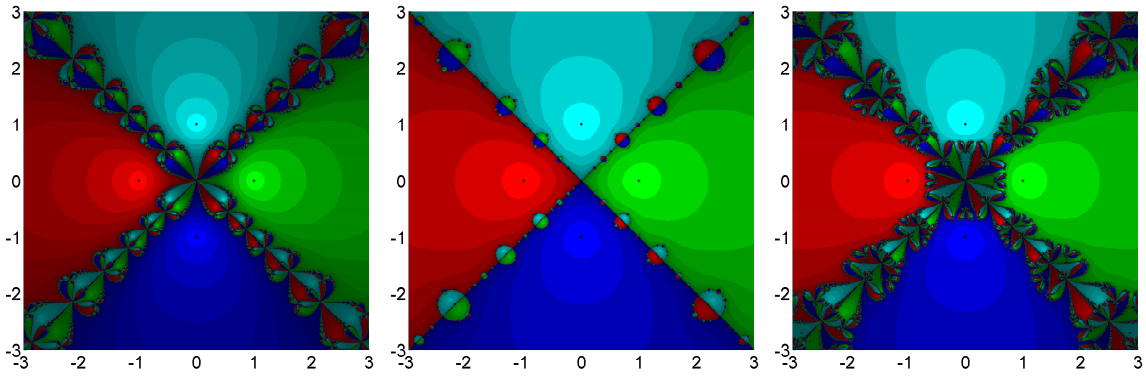

Figura 9. Cuenca de atracción (raíces múltiples) de la variante del método de Newton (izquierda), adaptación del método de Halley (centro) y modificación del método de Jarratt (derecha)

5. Conclusión

Se presenta un algoritmo flexible y eficiente en MatLab® que permite la realización de la cuenca de atracción de los puntos fijos de las funciones de iteración utilizadas para resolver ecuaciones no lineales de la forma $f(x) = 0$, donde $f : A \subset \mathbb{C} \to \mathbb{C}$. El algoritmo muestra un excelente comportamiento como se muestra en las figuras 2–5. El mismo es utilizado en tres famosos métodos numéricos: Newton, Halley y Jarratt aplicados a funciones que han sido estudiadas en la literatura. De igual manera se muestra el comportamiento del algoritmo propuesto para el caso de estar en presencia de ceros múltiples como se ve en las figuras 6–9, utilizando las respectivas adaptaciones de los método mencionados anteriormente para el cálculo de ceros múltiples. Además en la figura 1 se aprecian diferentes colores y tonalidades, los cuales pueden ser utilizado por el usuario para realizar cuencas de atracciones de problemas que involucren a lo sumo 20 raíces, permitiendo la posibilidad de aumentar el número de colores así como el orden en el que se presentan en este trabajo. Esta herramienta también puede ser utilizada para construir los espacios de parámetro que permiten comparar elementos de familias de métodos de uno y varios puntos para resolver ecuaciones no lineales con raíces simples y/o múltiples.

Referencias

[1] Varona, J. L. Graphic and numerical comparison between iterative methods, *The Mathematical Intelligencer*, 24 (1) (2002), 37–46.

[2] Shaw, W. Newton–Raphson iteration and complex fractals. In Complex Analysis with MATHEMATICA (pp. 56-77). Cambridge: Cambridge University Press. doi:10.1017/CBO9781316036549.005, 2006.

[3] Getz, C. y Helmstedt, J. , Graphics with Mathematica: Fractals, Julia Sets, Patterns and Natural Forms, Elsevier B. V., Amsterdam, The Netherlands, 2004.

[4] McClure, M. , Newton's method for complex polynomials, preprint version of a *Mathematical graphics column from Mathematica in Education and Research*, 1–15.

[5] Conte S.D. y De Boor C., Elementary Numerical Analysis: An Algorithmic Approach, *McGraw Hill Co., New York*, 1973.

[6] Halley E. , A new, exact and easy method of finding the roots of equations generally and that without any previous reduction, *Phil. Trans. Roy. Soc. London*, 18 (1694) 136–148.

[7] Jarratt P., Some fourth order multipoint iterative methods for solving equations, *Mathematics of Computation*, 20 (1966) 434–437.

[8] Schröder E., Uber unendlich viele Algorithmen zur Auflösung der Gleichungen, Math. Ann. 2 (1870), 317–365.

[9] Hansen E. y Patrick M., A family of root finding methods, Numer. Math. 27 (1977) 257–269.

[10] Sharma J.R. y Sharma R., Modified Jarratt method for computing multiple roots, Appl. Math. Comput. 217 (2010) 878–881.

[11] Neta B., Scott M. y Chun C., Basins of attraction for several methods to find simple roots of nonlinear equations, *Applied Mathematics and Computationn*, 218 (2012) 10548-10556.

[12] Chun C., Neta B. y Scott M., Basins of attraction for optimal eighth order methods to find simple roots of nonlinear equations, *Applied Mathematics and Computationn*, 227 (2014) 567–592.

[13] Chun C., Neta B., Kozdon J. y Scott M., Choosing weight functions in iterative methods for simple roots, *Applied Mathematics and Computationn*, 227 (2014) 788–800.

[14] Chun C. y Neta B., Basins of attraction for Zhou-Chen-Song fourth order family of methods for multiple roots, Math. Comput. Simul. 109 (2015) 74–91.