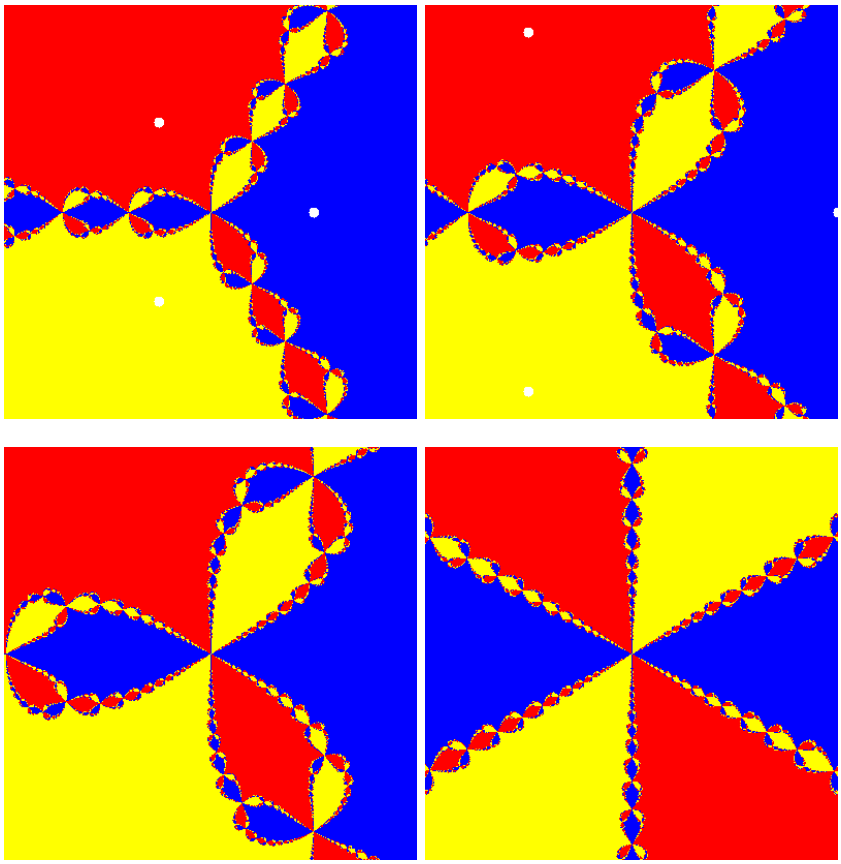
Algorithme de Newton

19 novembre 2008

Programmer la méthode de Newton pour trouver les racines cubiques de 1. Pour cela vous traduirez l'équation $z^3 - 1 = 0$ en un système de 2 équations et 2 inconnues, en posant $z = x + iy$, où $i^2 = -1$. Pour les 400×400 pixels d'un carré $[-c, +c]^2$, vous calculerez vers quelle racine converge la méthode de Newton, et vous afficherez chaque pixel avec une couleur dépendant de cette racine. Pour $c = 2.$ (en haut à gauche), $c = 1.$ (en haut à droite), $c = 0.8$ (en bas à gauche), $c = 0.1$ (en bas à droite), vous devez obtenir des images comme celles ci :



1 Le programme

```
module L=List;;
module A=Array;;
open Graphics;;

let f x y = x*.x *. x -. 3. *. x *. y *. y -. 1. ;;
let g x y = y *. y *. y -. 3. *. x *. x *. y ;;
let fx x y = 3. *. x *. x -. 3. *. y *. y;;
let fy x y = -. 6. *. x *. y ;;
let gx x y = -. 6. *. x *. y ;;
let gy x y = 3. *. y *. y -. 3. *. x *. x ;;

let step_newton (x,y) =
        let a = fx x y in let c = fy x y in
        let b = gx x y in let d = gy x y in
        let det = a *. d -. b *. c in
        let a' = d /. det in let b' = -. b /. det in
        let c' = -. c /. det in let d' = a /. det in
        ( x -. (f x y)*. a' -. (g x y) *. c',
          y -. (f x y)*. b' -. (g x y) *. d' );;
let sqr x = x *. x;;

let epsilon = 0.000001;;

let newton (x,y) =
        let rec boucle (x,y) n =
                if n=100 then (x,y)
                else let (x',y')= step_newton (x,y) in
                     if (sqr (x-.x') +. sqr(y-.y') <= epsilon*. epsilon)
                        then (x', y') else boucle (x', y') (n+1) in
        boucle (x,y) 0;;

let distance (x,y) (x',y') = sqrt (sqr (x-.x') +. sqr (y-.y'));;

let rec closest pt pts =
        let rec scan i iclosest =
                if i = A.length pts then iclosest
                else scan (i+1) (if distance pt pts.(i) < distance pt pts.(iclosest)
                                 then i  else iclosest)
        in scan 1 0;;

let foi x = float_of_int x;;
let iof x = int_of_float x;;

let go cote =
        open_graph "_400x400"; clear_graph ();
        let n= min (size_x()) (size_y()) in
        let roots = [| -1. /. 2., ((sqrt 3.)/. 2. );
                 -1. /. 2., -. ((sqrt 3.)/. 2. ); 1., 0. |] in
        for i= 0 to n-1 do for j= 0 to n-1 do
                let x= cote *. ( 2. *. (foi i) /. (foi n) -. 1.)  in
                let y= cote *. ( 2. *. (foi j) /. (foi n) -. 1.)  in
                let (x', y') = newton (x, y) in
                set_color (match closest (x',y') roots  with
                                | 0 -> yellow | 1 -> red | _ ->  blue );
                plot i j
        done done;
        let pixel x y = iof( (x +. cote) *. (foi n) /. 2. /. cote )
,
                        iof( (y +. cote) *. (foi n) /. 2. /. cote ) in
        A. iter (function (x,y)-> let (i,j)=pixel x y in
```

```
                set_color white; fill_circle i j 5) roots ;;

let decompose_color rgb= (*rouge,vert,bleu*)
   let r=rgb/256/256 in let v=(rgb/256) mod 256 in let b=rgb mod 256 in (r,v,b);;
module G=Graphics;;
let save_img  nom =
        let tx= G.size_x() and ty=G.size_y() in
        let img= G.dump_image (G.get_image 0 0 tx ty) in
        let file = open_out nom in
        Printf.fprintf file "P6 %d %d 255\n" (A.length img) (A.length img.(0));
        for lig= A.length img - 1 downto 0 do for col=0 to A.length img.(0) - 1 do
                let (red,green,blue)= decompose_color img.(lig).(col) in
                Printf.fprintf file "%c%c%c"
                  (Char.chr red) (Char.chr green) (Char.chr blue)
        done done; close_out file ;;
go 2.;;
save_img "2.ppm";;
go 1. ;;
save_img "1.ppm";;
go 0.8 ;;
save_img "0.8.ppm";;
go 0.1;;
save_img "0.1.ppm";;
```