

Premier mini Projet, Visualisation d'une fractale

Frédéric Hecht et Xavier Claeys

8 octobre 2019

Le but est d'afficher, la fonction f de \mathbb{C} dans \mathbb{C} , défini par

$$f : z \longrightarrow \lim_{n \rightarrow \infty} z_{n+1} = z_n - p(z_n)/p'(z_n); \quad \text{avec } z_0 = z$$

où $p(z) = z^k - 1$ avec $k = 3$ ou $4, \dots$ et où $p'(z)$ est le polynôme dérivé de $p(z)$.

Note, cette suite z_n est l'écriture de méthode de Newton pour trouver les racines de p , on cherche donc à afficher les bassins d'attractions de cette méthode dans le plan complexe.

Pour cela nous allons modifier la classe R2 défini plus loin pour faire votre classe Complexe.

1 Modélisation du Plan

Les points du plan sont modélisé avec une abscisse x et une ordonnée y , et comme un espace vectoriel sur le corps des réels Les sources des classes définies sont accessibles sur la toile à l'adresse suivante :

<https://www.ljll.math.upmc.fr/hecht/ftp/4M016/s4/R2.hpp>

1.1 La Classe R2

Mais avant toute chose voilà les entêtes :

```
#include <cmath>
#include <iostream>
#include <cstdio>
#include <cassert>
```

Voici une modélisation de \mathbb{R}^2 qui permet de faire des opérations vectorielles et qui définit le produit scalaire de deux points A et B , le produit scalaire sera défini par (A, B) . Cette version est sans compilation séparé et toutes les fonctions sont définies dans R2.hpp avec des inline.

```
typedef double R; // The class R2

class R2 {
public:
    R x,y; // declaration de membre
    // les 3 constructeurs ---
    R2 () :x(0.),y(0.) {} // rappel : x(0), y(0) sont initialiser via le constructeur de double
    R2 (R a,R b):x(a),y(b) {}
    R2 (const R2 & a,const R2 & b):x(b.x-a.x),y(b.y-a.y) {} // le constructeur par copie est inutile
    // opérateur affectation (*this) = P est inutile par défaut il fait le travail correctement // d'autre opérateurs affectations
    R2 & operator+=(const R2 & P) {x += P.x;y += P.y;return *this;}
    R2 & operator-=(const R2 & P) {x -= P.x;y -= P.y;return *this;} // opérateur binaire + - * , ^ /
    R2 operator+(const R2 & P) const {return R2(x+P.x,y+P.y);}
    R2 operator-(const R2 & P) const {return R2(x-P.x,y-P.y);}
}
```

```

R   operator,(const R2 & P) const {return x*P.x+y*P.y;} // produit scalaire
R   operator^(const R2 & P) const {return x*P.y-y*P.x;} // produit mixte
R2  operator*(R c) const {return R2(x*c,y*c);}
R2  operator/(R c) const {return R2(x/c,y/c);} // operateur unaire

R2  operator-() const {return R2(-x,-y);}
R2  operator+() const {return *this;}

R2  perp() const {return R2(-y,x);} // une méthode
// les opérateurs tableaux // la perpendiculaire
// version qui peut modifier la classe via l'adresse de x ou y
R & operator[](int i) { if(i==0) return x; else if (i==1) return y; else {assert(0);exit(1);} ;}
// version qui retourne une référence const qui ne modifie pas la class
const R & operator[](int i) const { if(i==0) return x; else if (i==1) return y;
                                     else {assert(0);exit(1);} ;}

// les opérateurs tableaux
// version qui modifie la classe via l'adresse de x ou y
R & operator()(int i) { if(i==1) return x;
                       else if (i==2) return y;
                       else {assert(0);abort();} ;}

// version qui retourne une référence const qui ne modifie pas la classe
const R & operator()(int i) const { if(i==1) return x; else if (i==2) return y;
                                     else {assert(0);abort();} ;}

R norme() const { return std::sqrt(x*x+y*y);}
};
inline std::ostream& operator <<(std::ostream& f, const R2 & P )
{ f << P.x << ' ' << P.y ; return f; }
inline std::istream& operator >>(std::istream& f, R2 & P)
{ f >> P.x >> P.y ; return f; }
inline R2 operator*(R c,const R2 & P) {return P*c;}
inline R2 perp(const R2 & P) { return R2(-P.y,P.x) ; }

```

Quelques remarques sur la syntaxe :

- Les opérateurs binaires dans une classe n'ont seulement qu'un paramètre. Le premier paramètre étant l'instance de la classe et le second étant le paramètre fourni;
- si un opérateur ou une fonction membre d'une classe ne modifie pas la classe alors il est conseillé de dire au compilateur que cette fonction est « constante » en ajoutant le mots clef `const` après la définition des paramètres;
- dans le cas d'un opérateur défini hors d'une classe le nombre de paramètres est donné par le type de l'opérateur unaire (+ - * ! [] etc.. : 1 paramètre), binaire (+ - * / | & || &&^ == <= >= < > etc.. 2 paramètres), n-aire (() : n paramètres).
- `ostream`, `istream` sont les deux types standards pour respectivement écrire et lire dans un fichier ou sur les entrées sorties standard. Ces types sont définis dans le fichier « `iostream` » inclue avec l'ordre `#include<iostream>` qui est mis en tête de fichier ;
- les deux opérateurs `<<` et `>>` sont les deux opérateurs qui généralement et respectivement écrivent ou lisent dans un type `ostream`, `istream`, ou `iostream`.

1.1.1 Utilisation de la classe R2

Cette classe modélise le plan \mathbb{R}^2 , de façon que les opérateurs classiques fonctionnent, c'est-à-dire :

Un point P du plan est modélisé par ces 2 coordonnés x, y , nous pouvons écrire des lignes suivantes par exemple :

```

R2 P(1.0,-0.5),Q(0,10);
R2 O,PQ(P,Q); // le point O est initialiser à 0,0
R2 M=(P+Q)/2; // espace vectoriel à droite
R2 A = 0.5*(P+Q); // espace vectoriel à gauche
R ps = (A,M); // le produit scalaire de R2
R pm = A^M; // le déterminant de A,M
// l'aire du parallélogramme formé par A,M
R2 B = A.perp(); // B est la rotation de A par π/2
B = perp(A); // même chose avec la fonction perp
R a= A.x + B.y;
A = -B;
A += M; // ie. A = A + M;
A -= B; // ie. A = -A;

```

```

double abscisse= A.x = A[0] = A(1) ; // la composante x de A
double ordonne = A.y = A[1] = A(2); // la composante y de A
cout << A ; // imprime sur la console A.x et A.y
cint >> A ; // vous devez entrer 2 double à la console

```

1.2 Graphique en mode texte

Le graphique du pauvre, écrire dans un fichier `ff-XXX`, n nombres par ligne sépare par un espace sur m lignes (où `XXX` est le numéro de votre dessin). Bien sûr n et m doivent être des nombre raisonnable (100 au maximum, 10 au minimum pour tester).

L'on visualisera se fichier avec le logiciel `gnuplot` comme suit :

```

gnuplot
set pm3d
splot "ff-XXX" w l

```

Les nombres sur les lignes correspondrons a une numérotations des solutions de votre problème (pour numéroté les solutions vous pourrez utiliser la fonction `atan2` défini dans `include cmath`).

Le code pour a génération du fichier doit ressemblé à :

```

#include <fstream>
#include <cstdio>
....
int nufichier=0;
....
{
nufichier++;
char filename[100]; // change le numero du dessin
sprintf(filename,"ff-%3d",nufichier); // la chaine doit faire moins de 99 caractères.
// génère un chaine "ff-XXX"
// où XXX est la valeur de la variable nufichier
ofstream ff(filename); // ouvre le fichier "ff-XXX" en ecriture (output file)
assert(ff);
for (int j=0;j<m;++j) // boucle sur les pixels du dessin
{
for (int i=0;i<n;++i)
{
int k=0; // soit k le numero de la racine à calculer
// calcul du point de départ en fonction de la pixel (i,j)
ff << k << " ";
}
ff << endl; // fin de la ligne
}
} // fermeture du fichier car la variable ff est detruite
...

```

2 A Faire

Pour faire l'exercice ;

1. il faut faire une classe `Complexe` à partir de la classe `R2` qui modéliser le corps des nombres complexe.
2. Vous écrire un programme pour valider votre classe
3. Construire une transformation $\mathcal{T}_{a,b,c,d}$ affine (une translation et une mise à l'échelle) qui va de l'écran $[0, width] \times [0, height]$ sur le rectangle du plan complexe à dessiner $[a, b] \times [c, d]$.

4. Vous écrirez un programme qui affiche chaque point de l'écran du pauvre (i,j) , avec une couleur dépendant de la racine obtenu avec la méthode de Newton partant de $z = T(i, j)$..
Pour afficher le rectangle $[-1, 1] \times [-\frac{height}{width}, \frac{height}{width}]$, en utilisant la transformation affine $T = \mathcal{T}_{-1,1,-\frac{height}{width},\frac{height}{width}}$, pour parcourir les points de l'écran nommés *pixels*.

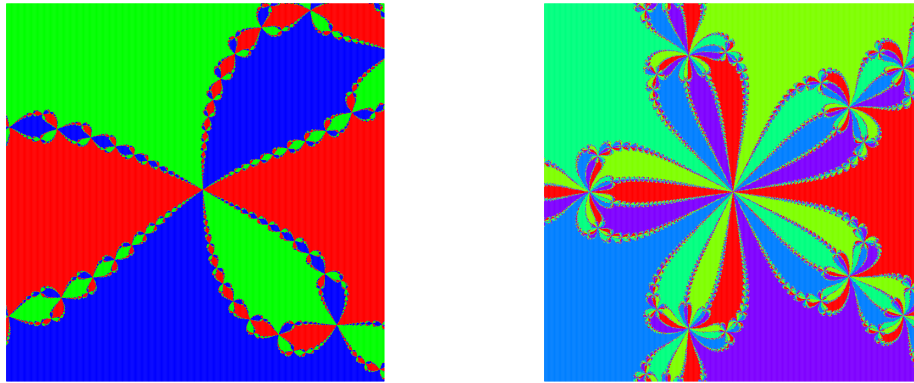


FIGURE 1 – Visualisation pour $k=3$ et $k=5$, une couleur est associée chaque racine