Journées mathématiques X-UPS

Année 2001

Pavages

Jean-René GEOFFROY

Programme et galerie

Journées mathématiques X-UPS (2001), p. 87-100.
https://doi.org/10.5802/xups.2001-02b

Les Éditions de l'École polytechnique
Route de Saclay
F-91128 PALAISEAU CEDEX
https://www.editions.polytechnique.fr

Centre de mathématiques Laurent Schwartz
CMLS, École polytechnique, CNRS,
Institut polytechnique de Paris
F-91128 PALAISEAU CEDEX
https://portail.polytechnique.edu/cmls/

Publication membre du
Centre Mersenne pour l'édition scientifique ouverte
www.centre-mersenne.org

Journées mathématiques X-UPS
2001, p. 87–100 doi: 10.5802/xups.2001-02b

PROGRAMME ET GALERIE

par

Jean-René Geoffroy

On présente ici un programme **Maple** permettant de représenter les tuiles des pavages autosimilaires. On indiquera par un symbole `//` les coupures de lignes dues à la mise en page.

Il s'agit essentiellement de calculer des développements en base β, de décrire une partie de l'ensemble des développements faiblement propres puis de colorer leurs projections sur le plan selon la partie fractionnaire des β-représentations.

On utilise la proposition 2.3 pour caractériser les développements faiblement propres.

Du point de vue informatique, on dispose de trois paramètres dans cette construction :

- la base β de représentation (entièrement déterminée par son polynôme minimal)
- les entiers p et q correspondant à la taille des développements manipulés :

p indique la taille des β-parties entières et q celle des β-parties fractionnaires.

Remarque. La valeur de q (la taille de la partie fractionnaire) influe sur le nombre de tuiles du pavage qui sont représentées (ce nombre est le cardinal des développements faiblement propres de taille q).
La valeur de p ne modifie pas le nombre de tuiles, mais, tout comme q, le nombre de représentations envisagées : on peut le voir comme un paramètre de « densité visuelle » du dessin.

Publication originelle dans Journées X-UPS 2001. Pavages. Éditions de l'École polytechnique, 2001.

On commence par remettre à zéro la feuille de calcul :

```
#remise à zéro de la feuille
> restart :
```

La procédure `devfaible` permet de calculer le développement faiblement propre d'un réel positif `x` dans une base quelconque `b` avec `n` chiffres après la virgule.
La valeur retournée est une liste dont le format est le suivant :

$$[\mathtt{p}, \mathtt{c}_1, \ldots, \mathtt{c}_k]$$

Le premier élément indique la position de la virgule : on a $k = p+n$ et le réel `x` s'écrit $\mathtt{x} = \mathtt{c}_1 \ldots \mathtt{c}_p, \mathtt{c}_{p+1} \ldots \mathtt{c}_k$ en base `b`.

Il est nécessaire de réaliser les calculs avec une grande précision pour éviter les erreurs d'arrondi lors de l'utilisation des parties entières.
On utilise la formule `floor(y)-floor(1-frac(y))` pour obtenir une « partie entière stricte » : lorsque le paramètre `x` possède une représentation finie, on choisit sa représentation infinie.

```
> devfaible := proc(b::realcons,x::realcons,n::integer)
   local y,i,dev,digits_Backup ;

   #sauvegarde du paramètre de précision
      digits_Backup := Digits ;
      Digits := 50 + n * 100 ;

      if not(type(evalf(x),positive)) then
        ERROR("Le paramètre x doit être positif") ;
      fi ;

      if not(type(evalf(b),positive)) then
        ERROR("Le paramètre x doit être positif") ;
      fi ;

      i := floor(evalf(ln(x)/ln(b))) ;
      dev := [i+1] ;
      y := x * (b^(-i)) ;
      for i from 1 to n+max(0,dev[1]) do
         dev := [op(dev),floor(y)-floor(1-frac(y))] ;
         y := b * (y-dev[-1]) ;
      od ;
```

```
   #restauration du paramètre de précision
      Digits := digits_Backup ;

    RETURN(dev) ;
  end :
```

Par exemple

```
> devfaible(10,1,5) ;
```

[1, 0, 9, 9, 9, 9, 9]

La procédure `calcule_pisot` permet de calculer le nombre de Pisot associé à un polynôme :
On commence par vérifier que le polynôme `p` en la variable `x` passée en paramètre définit bien une unité de Pisot (si ce n'est pas le cas, une erreur est générée pour avertir l'utilisateur), puis on renvoie un couple de valeurs $[\beta, \gamma]$ correspondant aux valeurs numériques du nombre de Pisot et de son conjugué de Galois.
Comme on manipule des polynômes à coefficients réels, pour obtenir une unité de Pisot, il doit y avoir exactement une racine réelle (et plus grande que 1).

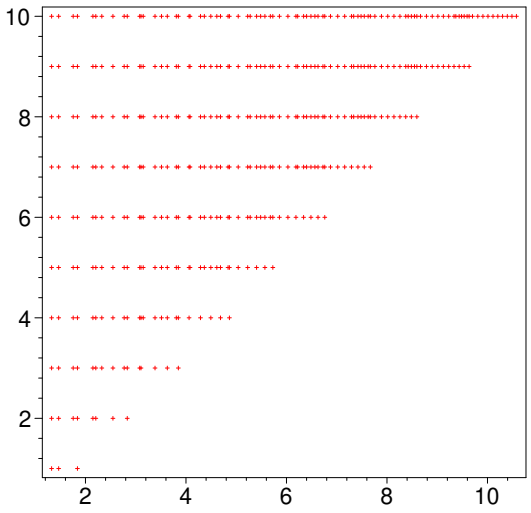
```
> calcule_pisot := proc(p::polynom,x::name)
      local racines,pisot,expan :

      if not(lcoeff(p,x)=1) then
          ERROR(sprintf("Le polynôme %a n'est pas unitaire !",p)) :
      fi :

      if not(type(p,polynom(integer,x))) then
          ERROR(sprintf("Le polynôme %a n'est pas à coefficients          //
                entiers !",p)) :
      fi :

      if not(abs(tcoeff(p,x))=1) then
          ERROR(sprintf("Le terme constant de %a n'est pas une            //
                unité !",p)) :
      fi :

      racines := [fsolve(p=0,x,complex,fulldigits)] ;
      pisot := select(z->type(z,realcons),racines) ;
      pisot := select(x->evalb(x>1),pisot) ;
```

```
    if nops(pisot)>1 then
        ERROR(sprintf("Le polynôme %a possède trop de racines          //
              réelles >1 :\n %a",p,mr)) :
    fi :

    if nops(pisot)=0 then
        ERROR(sprintf("Le polynôme %a ne possède pas de racine         //
              réelle > 1 !",p)) :
    fi :

    expan := select(z->evalb(Im(z)>0),racines) ;

    RETURN([op(pisot),op(expan)]) ;
end:
```

Exemple d'appel :

```
> calcule_pisot(X^3-X-1,X) ;
```

*[1.324717957, -.6623589786+.5622795121*I]*



Ce tableau représente en abscisse les unités de Pisot positives de degré 3 dont le polynôme minimal a tous ses coefficients inférieurs en valeur absolue au nombre porté en ordonnée.

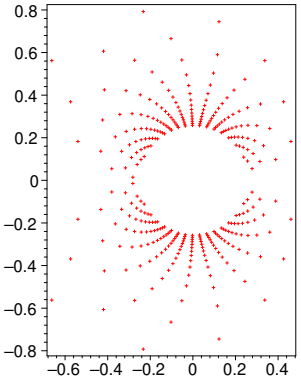
Pour réaliser la figure 2.3, on a simplement fait appel à cette procédure (en filtrant les erreurs), par exemple de la façon suivante :

```
> Liste_Pisot:=[] :
  for cmax from 1 to 10 do
     for c1 from -cmax to cmax do
        for c2 from -cmax to cmax do
           for c3 from 0 to 1 do
              pisot := [] ;
              pisot := traperror(                                    //
                     calcule_pisot(X^3+c2*X^2+c1*X+(-1)^c3,X)) ;
              if nops(pisot)>1 then
             Liste_Pisot := [op(Liste_Pisot), [pisot[1],cmax]] ;
              fi ;
           od ;
        od ;
     od ;
  od ;
 PLOT(POINTS(op(Liste_Pisot),COLOR(RGB,1,0,0),SYMBOL(CROSS)),       //
                         AXESSTYLE(BOX),SCALING(CONSTRAINED)) ;
```

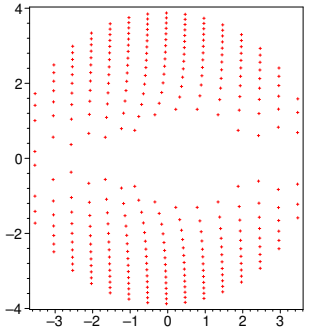
On vérifie de cette façon qu'il existe 89 nombres de Pisot obtenus comme racines de polynômes dont les coefficients sont inférieurs en valeur absolue à 10.
En modifiant un peu cette boucle, on peut également représenter l'ensemble des constantes d'expansion associées (et leurs inverses) :

Les deux procédures suivantes permettent de tester si un développement donné est faiblement propre pour la base `b`.

```
> lex_large_list := proc(l1::list,l2::list,d::integer)
    local m,i ;

  m := min(nops(l1)-d,nops(l2))-1 ;
 for i from 1 to m while l1[i+d]=l2[i] do : od :

 RETURN(evalb(l1[i+d]<=l2[i]))
 end :
> rep_stricte := proc(l1::list,b::numeric)
    local d,carry_seq ;

carry_seq := subsop(1=NULL,2=NULL,devfaible(b,1,nops(l1))) ;
for d from 0 to (nops(l1)-2) while lex_large_list(l1,carry_seq,d) do: od:
```



Conjugués de Galois des unités de Pisot de degré 3 *obtenues comme racines de polynômes de coefficients majorés en valeur absolue par* 15.



Constantes d'expansion pour les polynômes de coefficients majorés en valeur absolue par 15.

```
  RETURN(lex_large_list(l1,carry_seq,d)) ;
 end :
```

Exemple d'appel :

```
> rep_stricte([1,0,0,0,1,0,0,1],1.324717957) ;
```

false

```
> rep_stricte([1,0,0,0,0,0,1,0],1.324717957) ;
```

true

La routine `pavage` est le moteur principal de l'affichage des tuiles. Il s'agit d'un parcours des développements faiblement propres réalisé dans le sens décroissant ; on utilise les propriétés suivantes :
Soit un β-développement faiblement propre, voici la méthode pour obtenir le β-développement faiblement propre immédiatement inférieur :

◦ Si on décrémente le dernier coefficient non nul de la suite, tous les translatés de la représentation sont diminués ou inchangés : elle reste faiblement propre.

◦ Si on doit poser une retenue, il suffit de remplacer le translaté correspondant par la suite de retenue (tronquée de manière adéquate) pour que la représentation vérifie encore la proposition 2.3.
De cette façon, il n'est pas nécessaire de tester qu'une représentation est faiblement propre à chaque étape.
Les paramètres de `pavage` sont, comme on l'a indiqué au début du paragraphe : `R` le polynôme minimal en `X` définissant la base β, et `p`, `q` définissant la taille des représentations manipulées.
Avant de présenter la routine `pavage` proprement dite, on commence par mettre en œuvre la technique de parcours pour déterminer le nombre de développements faiblement propres de longueur donnée :

```
> ntuile := proc(R::polynom,X::name,q::integer)

  local rep,lrep,b,pisot,i,j,carry_seq :

  pisot := calcule_pisot(R,X) ;
  b := op(1,pisot) ;
  carry_seq := [op(3..-1,devfaible(b,1,q))] ;
  rep := carry_seq ;
  j := nops(rep) :
  lrep := [] :
  while(j>0) do
   lrep:=[op(lrep),rep] ;
    if rep[j]>0 then
      rep:=subsop(j=rep[j]-1,rep) :
    else
      # on insère la suite de retenue
      for i from j to 1 by -1 while (rep[i]=0) do: od:
      if (i=0) then
        j:=0 ;
      else
        rep:=subsop(i=rep[i]-1,rep) :
        for j from (i+1) to q do
          rep := subsop(j=carry_seq[j-i],rep):
        od:
        j:=nops(rep) :
      fi:
    fi:
  od:
```

```
 print(q,nops(lrep)) ;
 print(lrep) ;
end:
```

```
> pavage := proc(R::polynom,X::name,p::integer,q::integer)

 global dPoints, rep :
 local b,lambda,pisot,i,j,k,pos,v,carry_seq,time_backup,          //
         RPOL,LPOL,POL,SIMPX,SIMPY :

 pisot := calcule_pisot(R,X) ;
 b := op(1,pisot) ;
 lambda := op(2,pisot) ;
 carry_seq := [op(3..-1,devfaible(b,1,p+q))] ;

 SIMPX := sum('-coeff(R,X,k)*X^k','k'=0..degree(R,X)-1) :
 SIMPY := sum('-coeff(R,X,0)*coeff(R,X,k)*X^(k-1)',               //
                                        'k'=1..degree(R,X)) :

 # Informations de début de calcul
 printf("Début du calcul des tuiles associées à %a\n             //
     le coefficient d'expansion est %a\n                          //
     la suite de retenue : %a\n                                   //
     la précision demandée est %a:%a\n",b,1/lambda,carry_seq,p,q) ;
 time_backup := time() ;

 # Pour reprendre un calcul interrompu, il suffit de commenter
 # les deux lignes suivantes et d'invoquer à nouveau pavage
 rep := carry_seq ;
 unassign(dPoints) ;

 j := nops(rep) :
 rep := subsop(j=rep[j]+1,rep) :

while(j>0) do
  if rep[j]>0 then
    rep:=subsop(j=rep[j]-1,rep) :
  else
    # on insère la suite de retenue
    for i from j to 1 by -1 while (rep[i]=0) do : od:
    if (i=0) then
```

```
      j:=0 ;
      printf("\n Calcul terminé (après %a secondes)\n",          //
                                        time()-time_backup) ;
    else
      rep:=subsop(i=rep[i]-1,rep) :
      for j from (i+1) to (p+q) do
        rep := subsop(j=carry_seq[j-i],rep):
      od:
      j:=nops(rep) :
    fi:
  fi:

  LPOL := sum('rep[p-i]*X^(i+1)','i'=0..p-1);
  RPOL := sum('rep[p+i]*Y^i','i'=1..q);

  # on associe un chiffre à chaque tuile
  v := subs(Y=ceil(b),RPOL) ;

  LPOL := simplify(LPOL,{X^3=SIMPX}):
  RPOL := simplify(RPOL,{Y=SIMPY}):
  RPOL := simplify(RPOL,{X^3=SIMPX}) :

  POL := collect(LPOL+RPOL,X) ;
  pos := subs(X=lambda,POL) ;

  if not(type(dPoints[v],list)) then
    printf("Création de la tuile %a\n",v) ;
    dPoints[v]:=[[Re(pos),Im(pos)]];
  else
    dPoints[v]:=[op(dPoints[v]),[Re(pos),Im(pos)]];
  fi ;
 od:
 end:
```

Exemple d'appel :

```
> pavage(X^3-X-1,X,20,0) :
```

Début du calcul des tuiles associées à 1.324717957
*le coefficient d'expansion est -.8774388331-.7448617667*I*
la suite de retenue : [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0]
la précision demandée est 20:0
Création de la tuile 0
Calcul terminé (après 5.277 secondes)

Remarque. Il est préférable d'invoquer cette procédure sur une ligne terminée par un caractère deux points : cela évite d'afficher la liste des points calculés (qui peut être très volumineuse).

Pour afficher le résultat du calcul, on peut utiliser une commande de la forme :

```
> c:=rand(255)/255 ;
> plotsetup(default) ;
> PLOT(op(map(x->POINTS(op(dPoints[x]),COLOR(RGB,c(),c(),c())      //
         ,SYMBOL(CROSS)),select(x->evalb(nops(dPoints[x])>1),      //
            [$0..2^10]))),AXESSTYLE(BOX),SCALING(CONSTRAINED)) ;
```

et pour stocker le résultat dans un fichier au format postscript, il suffit de remplacer la ligne `plotsetup(default)` par :

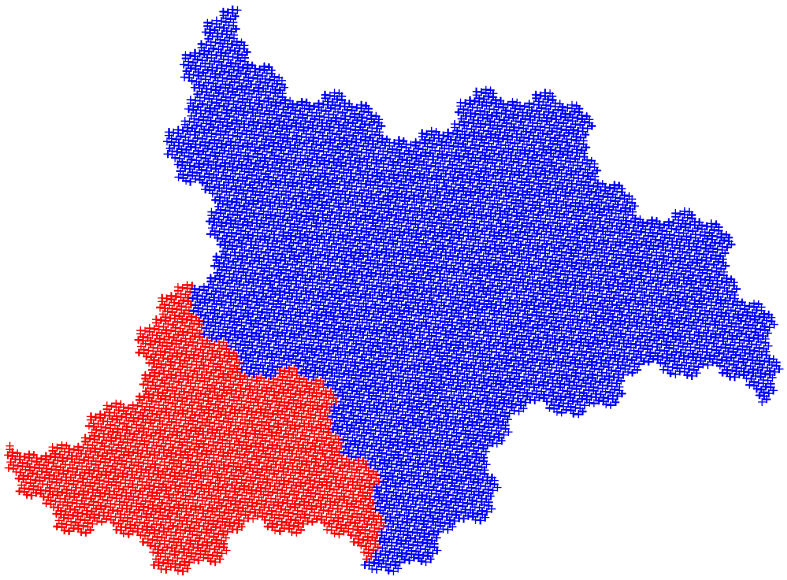
```
plotsetup(ps,plotoutput='Pisot',plotoptions='colour=rgb,           //
                                        noborder,portrait');
```

Les temps de calcul sont très variables suivant la précision demandée : il est possible d'afficher un résultat partiel après l'interruption d'un calcul, mais aussi de relancer un calcul stoppé (en commentant les deux lignes indiquées dans `pavage`).
La procédure `pavage` stocke le résultat des calculs dans la variable global `dPoints` : il s'agit d'un tableau dont les éléments sont indexés par les entiers affichés lors du calcul (`subs(Y=ceil(b),RPOL)`). On peut donc afficher et manipuler chacune des tuiles indépendamment. Par exemple, on va vérifier la nature fractale du pavage en superposant une tuile et son image par la multiplication par $1/\lambda$ (inverse du conjugué de Galois de β) :

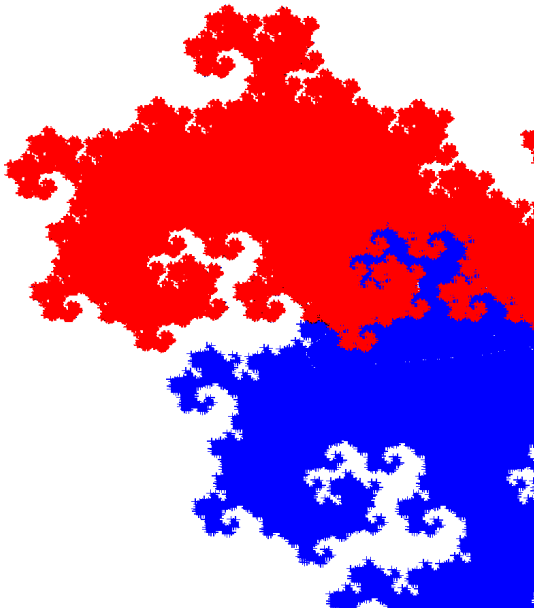
```
> pavage(X^3-X-1,X,20,0) ;
  lambda := op(2,calcule_pisot(X^3-X-1,X)) ;
  L := [Re(1/lambda),Im(1/lambda)] ;
  # on redéfinit la mutliplication complexe...
  f := (a,b)->[a[1]*b[1]-a[2]*b[2],a[1]*b[2]+a[2]*b[1]] ;
  # on transforme la première tuile
  d1Points:=map(z->f(z,L),dPoints[0]) :
  # et on la superpose à son image (attention à l'ordre)
  PLOT(POINTS(op(dPoints[0]),COLOR(RGB,1,0,0),SYMBOL(CROSS)),      //
      POINTS(op(d1Points),COLOR(RGB,0,0,1),SYMBOL(CROSS)),         //
      AXESSTYLE(BOX),SCALING(CONSTRAINED)) ;
```

On obtient (avec beaucoup moins de calculs) la figure de division des tuiles :

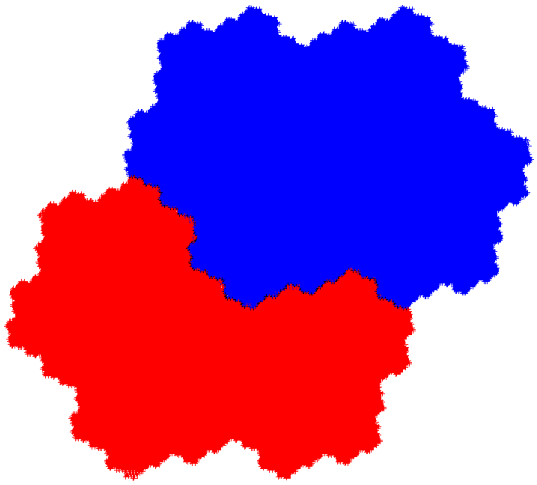
- La partie rouge représente la « tuile 0 », obtenue pour une précision $p : 0$.
- La partie bleue représente la portion de l'image de la « tuile 0 » par multiplication par $1/\lambda$ qui n'est pas recouverte par la « tuile 0 ».



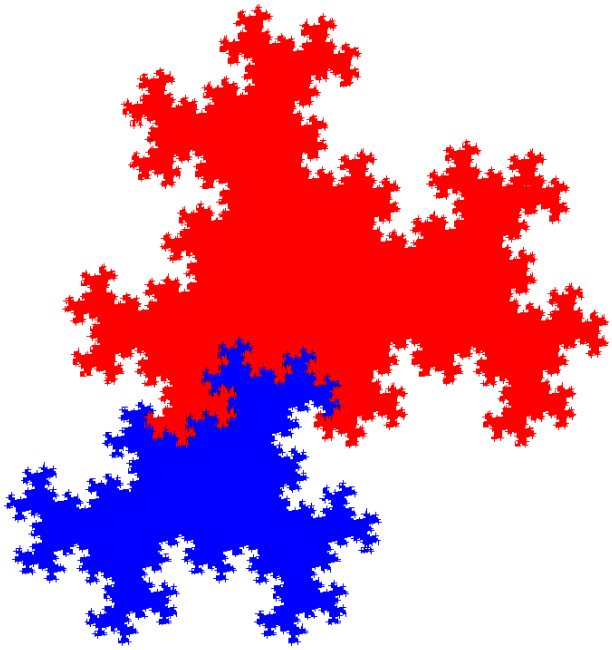
Motif du pavage associé à l'unité de Pisot $\beta = 1.32471795724475\ldots$, racine du polynôme $X^3 - X - 1$ (de suite de retenue $carry(\beta) = 0.(10000)$).
Le pavage ne contient qu'un seul type de tuile à rotations et homothéties près.



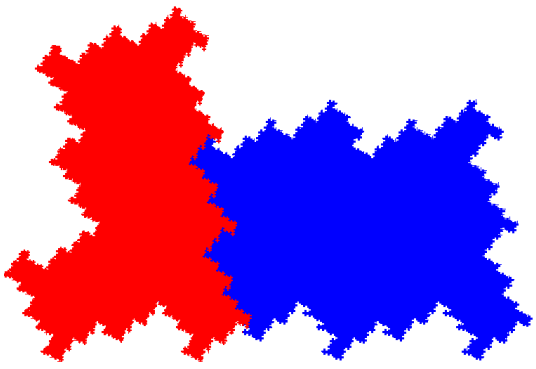
Motif du pavage associé à l'unité de Pisot $\beta = 2.3247179572447460260\ldots$, racine du polynôme X^3-3X^2+2X-1 (de suite de retenue $carry(\beta) = 0.20(1)$). Le pavage est constitué de deux types de tuiles à rotations et homothéties près.



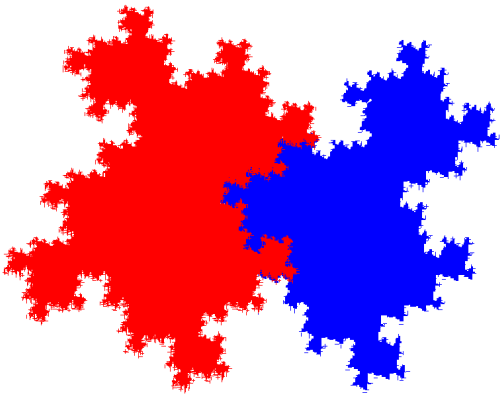
Motif du pavage associé à l'unité de Pisot $\beta = 1.8392867552141611326\ldots$, racine du polynôme $X^3 - X^2 - X - 1$ (de suite de retenue $carry(\beta) = 0.(110)$). Le pavage est constitué de deux types de tuiles à rotations et homothéties près.



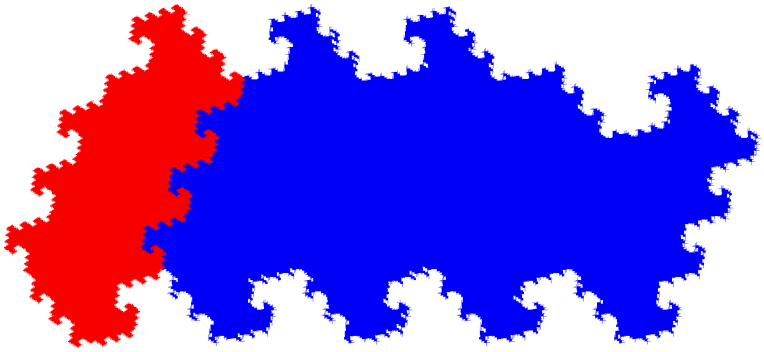
Motif du pavage associé à l'unité de Pisot $\beta = 1.4655712318767680267\ldots$, racine du polynôme $X^3 - X^2 - 1$ (de suite de retenue $carry(\beta) = 0.(100)$). Le pavage est constitué de deux types de tuiles à rotations et homothéties près.



Motif du pavage associé à l'unité de Pisot $\beta = 2.2055694304005903117\ldots$, racine du polynôme $X^3 - 2X^2 - 1$ (de suite de retenue $carry(\beta) = 0.(200)$). Le pavage est constitué de deux types de tuiles à rotations et homothéties près.



Motif du pavage associé à l'unité de Pisot $\beta = 1.7548776662466927600\ldots$, racine du polynôme X^3-2X^2+X-1 (de suite de retenue $carry(\beta) = 0.(1100)$). Le pavage est constitué de deux types de tuiles à rotations et homothéties près.



Motif du pavage associé à l'unité de Pisot $\beta = 4.613470268\ldots$, racine du polynôme $X^3 - 5X^2 + 2X - 1$ (de suite de retenue $carry(\beta) = 0.42(3)$). Le pavage est constitué d'un seul type de tuile à rotations et homothéties près.

On pourra trouver les fichiers source, des images ainsi que des liens vers d'autres pages concernant les fractales de Rauzy sur la page :

`http://www.math.u-psud.fr/~geoffroy/pisot`

Jean-René Geoffroy, Laboratoire de Mathématiques, UMR 8628 du CNRS, Bât 425, Université Paris-Sud, 91405 Orsay, France