La Tortue

Algo & Prog avec R

A. Malapert, B. Martin, M. Pelleau, et J.-P. Roy

10 septembre 2021

Université Côte d'Azur, CNRS, I3S, France
firstname.lastname@univ-cotedazur.fr

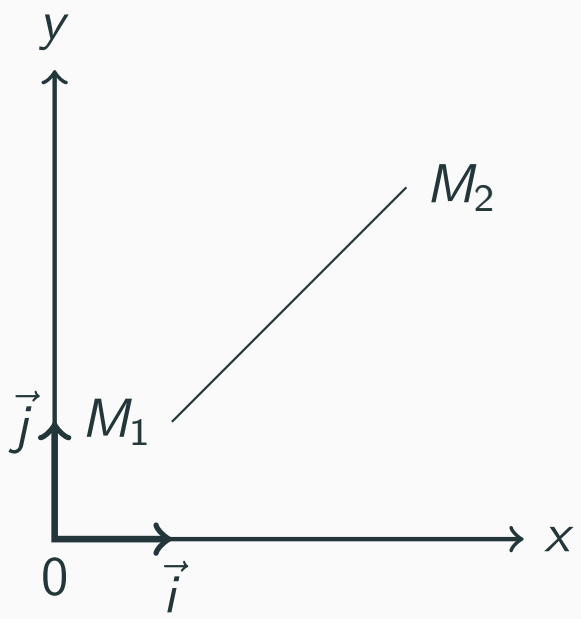
Les deux types de graphisme dans le plan I

Il y a deux types de graphisme 2D, mathématiquement parlant :

Le graphisme CARTESIEN (global)
Le plan est rapporté à un repère orthonormé direct $(0, \vec{i}, \vec{j})$.

Une seule opération essentielle

Tracer un segment du point $M_1(x_1, y_1)$ au point $M_2(x_2, y_2)$.



Les deux types de graphisme dans le plan II

Le graphisme POLAIRE (local)
Aucune notion de coordonnées.

Deux opérations essentielles

- Tourner à droite ou à gauche sur place d'un angle a.
- Avancer dans la direction courante d'une distance d.

L'animal traceur porte un repère mobile orthonormé avec une notion de droite et de gauche.
la tortue va tourner à gauche

- Opérateurs de translation et de rotation plane, qui engendrent le groupe des déplacements. La tortue se déplace dans le plan !
- Graphisme moins matheux, plus intuitif. Inutile de calculer les coordonnées des points . . .
- Une trajectoire qui semble lisse sera en fait un polygone !

Le module TurtleGraphics de R

Le graphisme de la tortue a été inventé au Laboratoire d'Intelligence Artificielle du MIT vers 1968 avec le langage LOGO.

- Il est disponible dans quasiment tous les langages de programmation qui offrent des facilités graphiques.
- Et en particulier en R avec le module TurtleGraphics.

Installation et chargement
Ce module n'est pas livré avec la distribution R standard.

```
install.packages("TurtleGraphics")
```

Il faut en importer les noms pour pouvoir les utiliser.

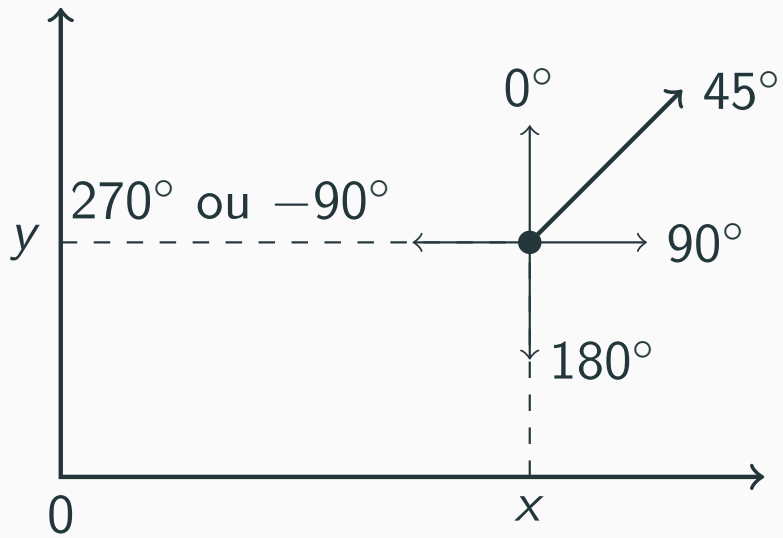
```
library(TurtleGraphics)
```

Graphisme cartésien

C'est celui des matheux dans la mesure où il faut calculer les coordonnées des points à relier.

Représentation de la tortue

- une flèche qui indique son cap en degrés ;
- une position : une abscisse et une ordonnée ;
- un crayon (*pen*) qui peut être baissé (*down*) ou levé (*up*). Si le crayon est baissé, la tortue laisse une trace en se déplaçant. On peut choisir la couleur du crayon ainsi que le type et l'épaisseur de la ligne.



État et opération de la tortue

Une tortue a donc un ETAT représenté mathématiquement par trois données : position ; cap ; crayon.

Position

```
turtle_getpos()
turtle_setpos(x,y)
```

Cap

```
turtle_getangle()
turtle_setangle(a)
```

Crayon (état)

```
turtle_down()
turtle_up()
```

Crayon (style)

```
turtle_param(col, lwd ,lty)
turtle_col(col)
turtle_lwd(lwd)
turtle_lty(lty)
```

Tracer un segment

```
turtle_goto(x, y)
```

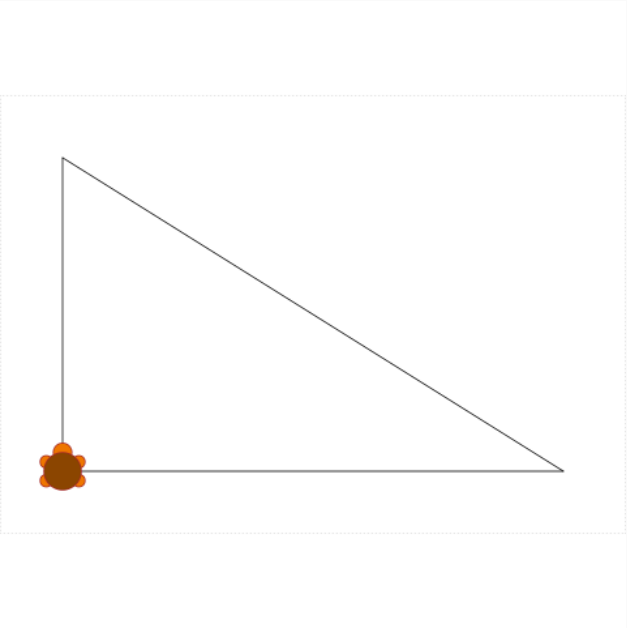
Dessin d'un triangle rectangle

Agir sur le bac à sable (canevas)

```
turtle_init(width = 100, height = 100,
            mode = c("error", "clip", "cycle"))
turtle_reset()
```

```
TriRect <- function(a, b, c = 10) {
  turtle_up()
  turtle_goto(c, c);
  turtle_down()
  turtle_goto(a + c, c)
  turtle_goto(c, b + c)
  turtle_goto(c, c)
}
```

```
turtle_init(width = 100, height = 70)
turtle_do(TriRect(80, 50))
```



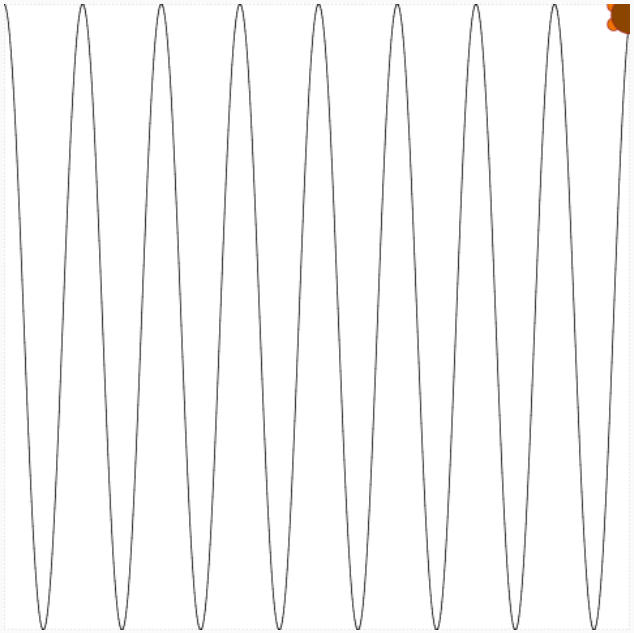
ATTENTION, les points du canevas ont des coordonnées positives.

L'origine du repère est donc en bas à gauche.

Tracé de la courbe du cosinus

```
TraceFunction <- function(f, a, b, n)
      {
  turtle_up()
  turtle_goto(a, f(a))
  turtle_down()
  for(x in seq(a,b, length.out=n)) {
    turtle_goto(x, f(x))
  }
}
```

```
b <- 50
n <- 1000
turtle_init(width= b, height= b)
f <- function(x) b * (cos(x)+1) / 2
turtle_do(TraceFunction(f, 0, b, n))
```



Comme `turtle_goto` ou `TraceFunction` , la plupart des fonctions de dessin n'ont pas de résultat, seulement des effets.

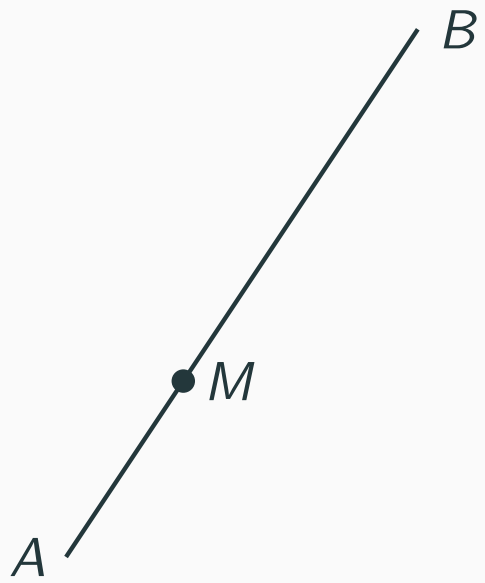
Courbes en coordonnées paramétriques

Cinématique (étude du mouvement)
La cinématique s'intéresse à la trajectoire d'un corps dont les coordonnées (x, y) sont fonction d'un paramètre t. Autrement dit :
$x = x(t)$ et $y = y(t)$
Ces courbes englobent les courbes $y = f(x)$ mais sont plus générales !

Le segment
Le segment AB joignant le point $A(x_A, y_A)$ au point $B(x_B, y_B)$ est la trajectoire d'un mobile M paramétrée par $t \in [0, 1]$:

$$x(t) = tx_A + (1 - t)x_B$$
$$y(t) = ty_A + (1 - t)y_B$$

De manière vectorielle : $\overrightarrow{MB} = t\overrightarrow{AB}$

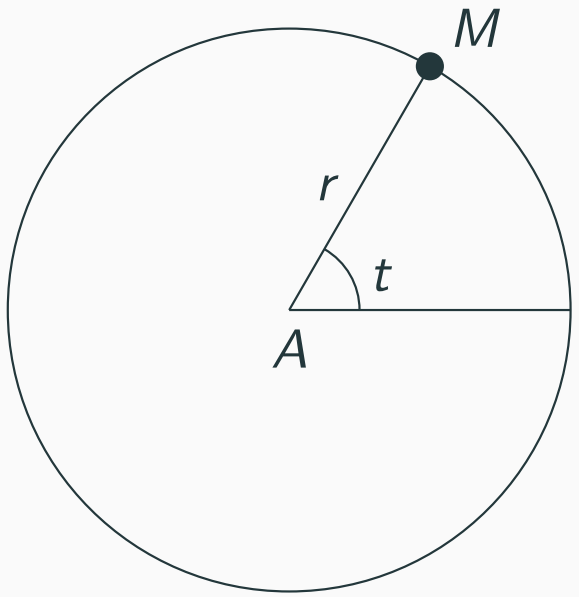


Animation de la tortue parcourant un cercle

Le cercle

le cercle de centre $A(x, y)$ et de rayon r n'est autre que la trajectoire d'un mobile M dont les coordonnées sont paramétrés par $t \in [0, 2\pi]$:

$$x(t) = x + r\cos(t)$$
$$y(t) = y + r\sin(t)$$

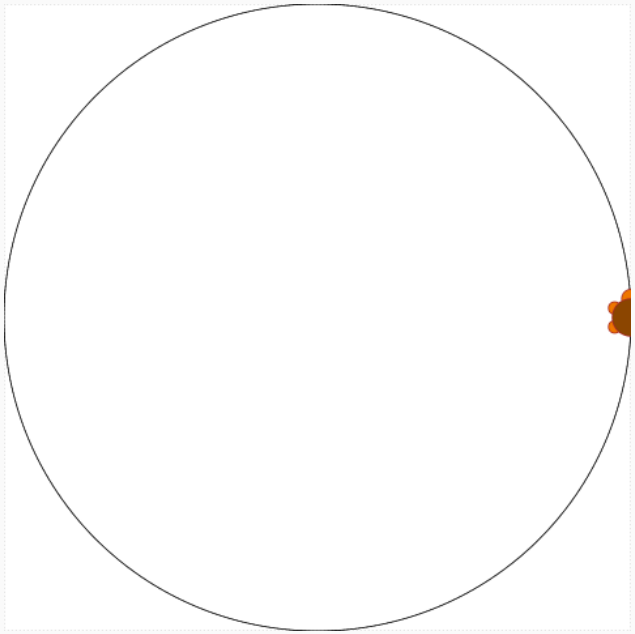



```
Cercle <- function(r, n) {
  turtle_up()
  turtle_goto(2*r, r)
  turtle_down()
  for(x in seq(0,2*pi, length.out=n)) {
    turtle_goto(r + r*cos(x), r + r*sin(x))
  }
}
```

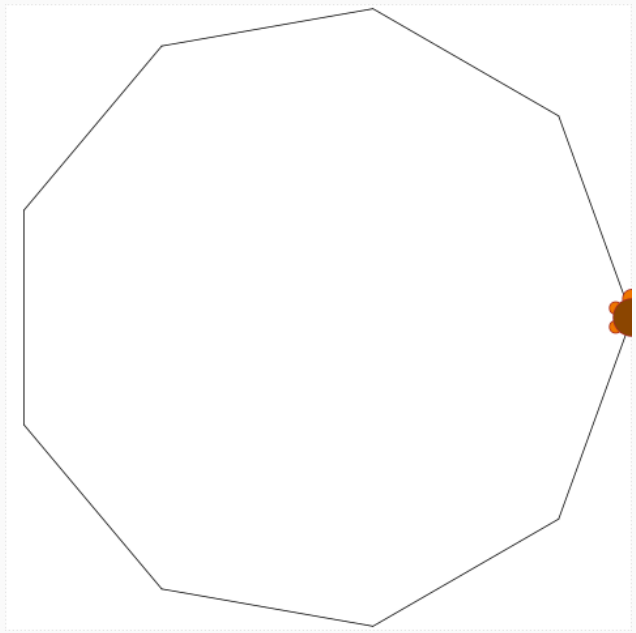
Le caractère continu du mouvement est une illusion d'optique

En fait, il est discrétisé. Le paramètre t avance chaque fois de $\frac{2\pi}{n}$.

```
turtle_init()
turtle_do(
  Cercle(r = 50, n = 1000)
)
```



```
turtle_init()
Cercle(r = 50, n = 10)
```



Le choix de n peut être empirique, guidé par l'esthétique de la simulation.

Le graphisme polaire

Il s'agit du vrai graphisme tortue pour les puristes ...
Nous ignorons la valeur du cap et de la position dans le graphisme polaire pur.

Le cap

```
turtle_left(a)
turtle_right(a)
turtle_turn(a,dir)
```

La position

```
turtle_forward(d)
turtle_backward(d)
turtle_move(d,dir)
```

`turtle_right(a)` $\Leftrightarrow$ `turtle_left(-a)`

`turtle_backward(d)` $\Leftrightarrow$ `turtle_forward(-d)`

Dessiner
Une suite d'appels à ces fonctions `turtle_left` et `turtle_forward` permet donc de décrire une courbe d'un seul tenant. En levant le crayon, on peut tracer plusieurs courbes non reliées entre elles.

Dessiner un triangle et un carré

Il s'agit du vrai graphisme tortue pour les puristes . . . Nous ignorons la valeur du cap et de la position dans le graphisme polaire pur.

```
Triangle <- function(c) {
  for(i in 1:3) {
    turtle_forward(c)
    turtle_left(120)
  }
}
```

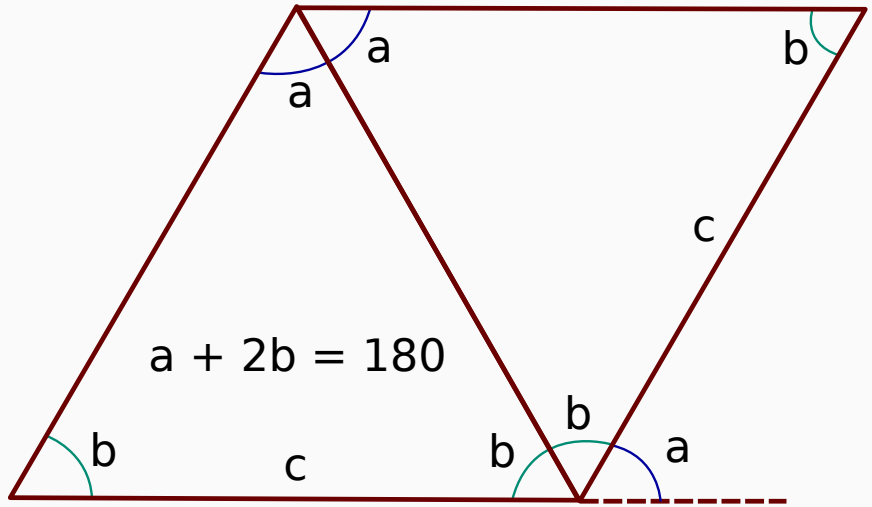
```
Carre <- function(c) {
  for(i in 1:4) {
    turtle_forward(c)
    turtle_left(90)
  }
}
```

Remarquez la notation `i:j` pour parcourir l'intervalle [i,j].

Généralisation : dessiner un polygone régulier

```
Polygone <- function(n, c) {
  a <- 360 / n
  for(i in seq(n)) {
    turtle_forward(c)
    turtle_left(a)
  }
}
```

```
Polygone <- function(n, c) {
  a <- 360 / n
  while(n > 0) {
    turtle_forward(c)
    turtle_left(a)
    n <- n - 1
  }
}
```

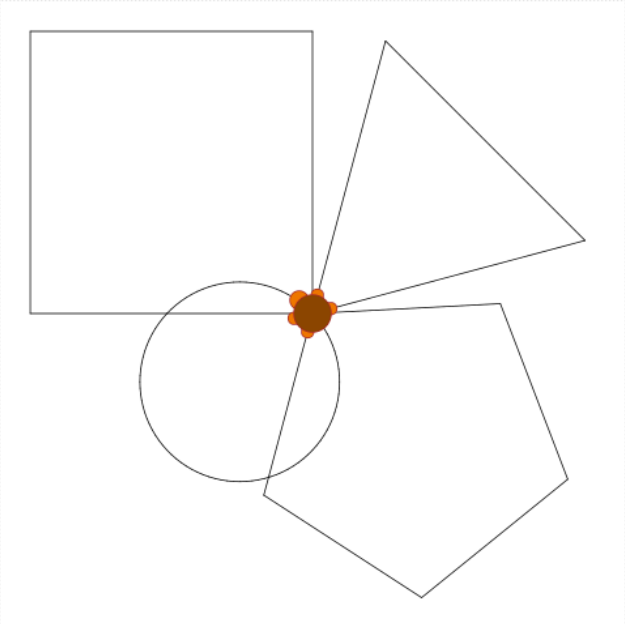


Remarquez la notation `seq(n)` pour parcourir [1,n]. La boucle `for` est bien pratique lorsque l'on connaît à l'avance le nombre d'itérations

En pratique, dessiner et exporter une figures

```
Carre <- function(c) Polygone(4,c)
```

```
png("fig/polygones.png")
turtle_init()
Carre(45)
turtle_right(75)
Triangle(45)
turtle_right(120)
Polygone(5, 30)
turtle_right(120)
turtle_do(Polygone(100, 1))
dev.off()
```



Carrés en fleur

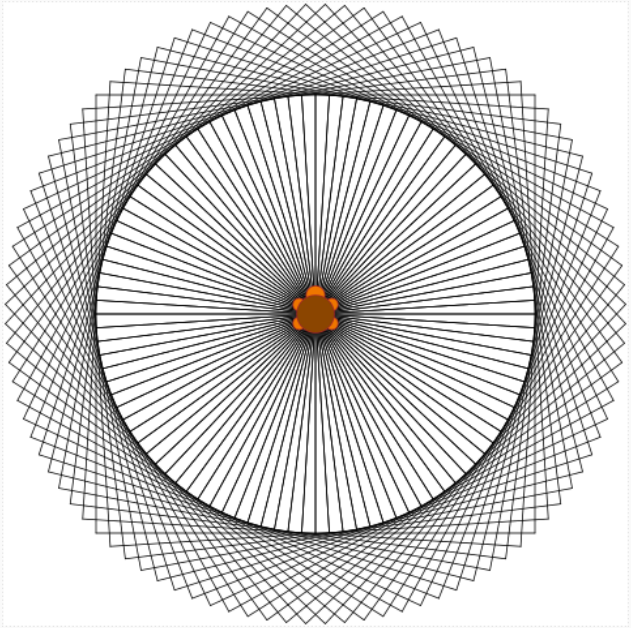
Spécialisons `Polygone` en une fonction auxiliaire globale `Carre`.

```
Carre <- function(c) Polygone(4,c)
```

Dessinons une fleur avec des carrés en rotation par composition et répétition de fonctions.

```
Fleur <- function(n, c) {
  a <- 360 / n
  for(i in seq(n)) {
    Carre(c)
    turtle_left(a)
  }
}
```

```
turtle_init()
turtle_do(Fleur(100, 35))
```

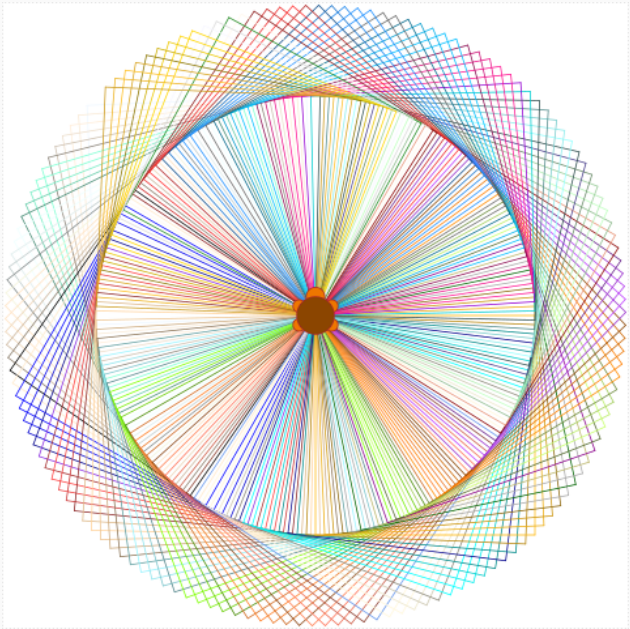


Carrés colorés en fleur

- Il est possible, mais non obligatoire, de localiser la fonction auxiliaire. Elle ne sera plus utilisable par ailleurs !
- Remarquez qu'une fonction locale a accès aux arguments de la fonction globale.
- On change la couleur de chaque carré en créant un vecteur de `n` couleurs.

```
Fleur <- function(n, c) {
  Carre<- function() Polygone(4,c)
  a <- 360 / n
  cols <- rep_len(colors(), n)
  for(col in cols) {
    turtle_col(col)
    Carre()
    turtle_left(a)
  }
}
```

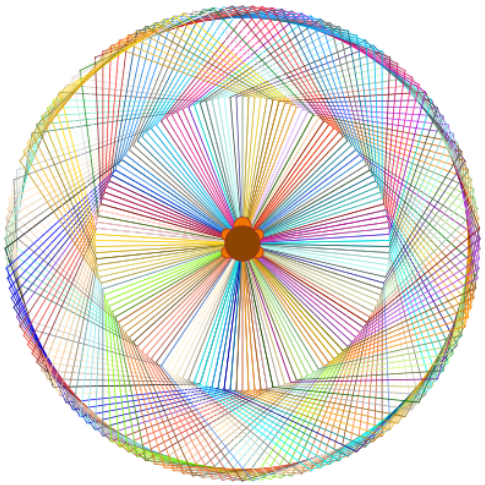
```
turtle_init()
turtle_do(Fleur(150, 35))
```



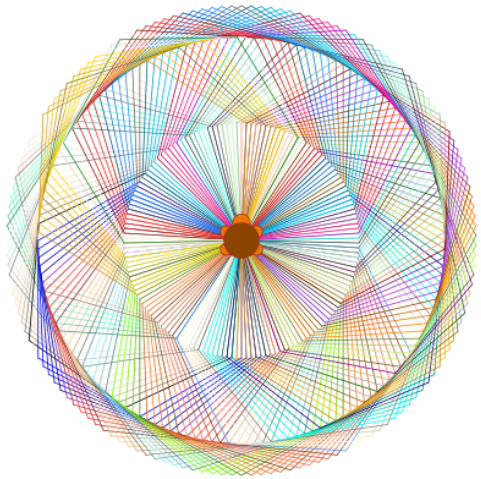
Généralisation : polygones colorés en fleur

```
Fleur <- function(n, m, c) {
  a <- 360 / n
  cols <- rep_len(colors(), n)
  for(col in cols) {
    turtle_col(col)
    Polygone(m, c)
    turtle_left(a)
  }
}
```

```
turtle_init()
turtle_do(Fleur(150, 5, 25))
```



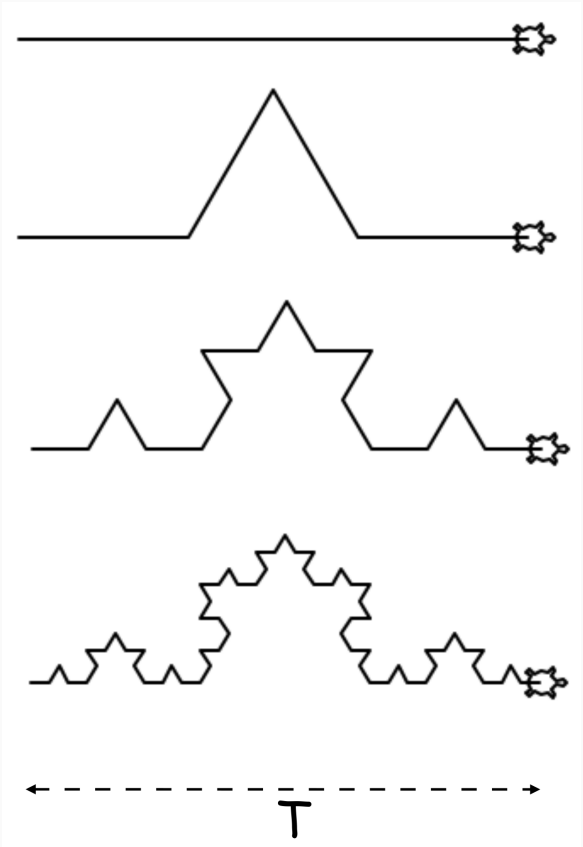
```
turtle_init()
turtle_do(Fleur(150, 6, 20))
```



La courbe fractale de Von Koch

Petite incursion dans la récurrence graphique. La suite (VK_n) des courbes de Von Koch de base T est construite de proche en proche :

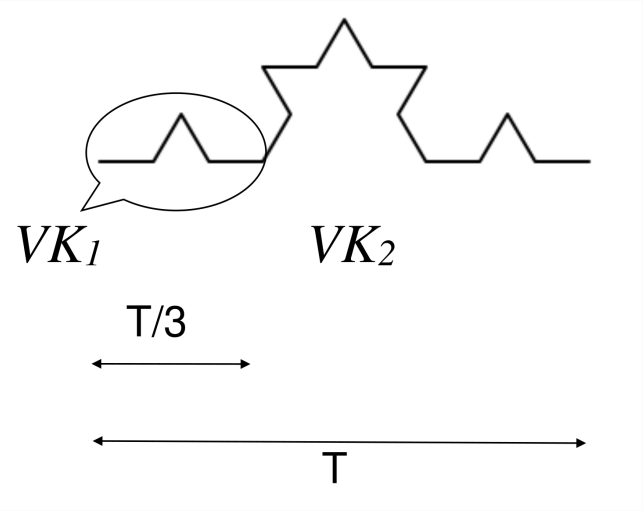
- VK_0 est un segment de longueur T.
- VK_1 s'obtient par chirurgie sur VK_0.
- VK_2 s'obtient par la même chirurgie sur chaque segment de VK_1.
- VK_3 s'obtient par la même chirurgie sur chaque segment de VK_2.
- ...



Dessin de la courbe fractale de Von Koch

Mathématiquement, la courbe VK_n s'obtient donc comme assemblage de quatre courbes VK_{n-1}. Il s'agit donc d'une RÉCURRENCE sur n.

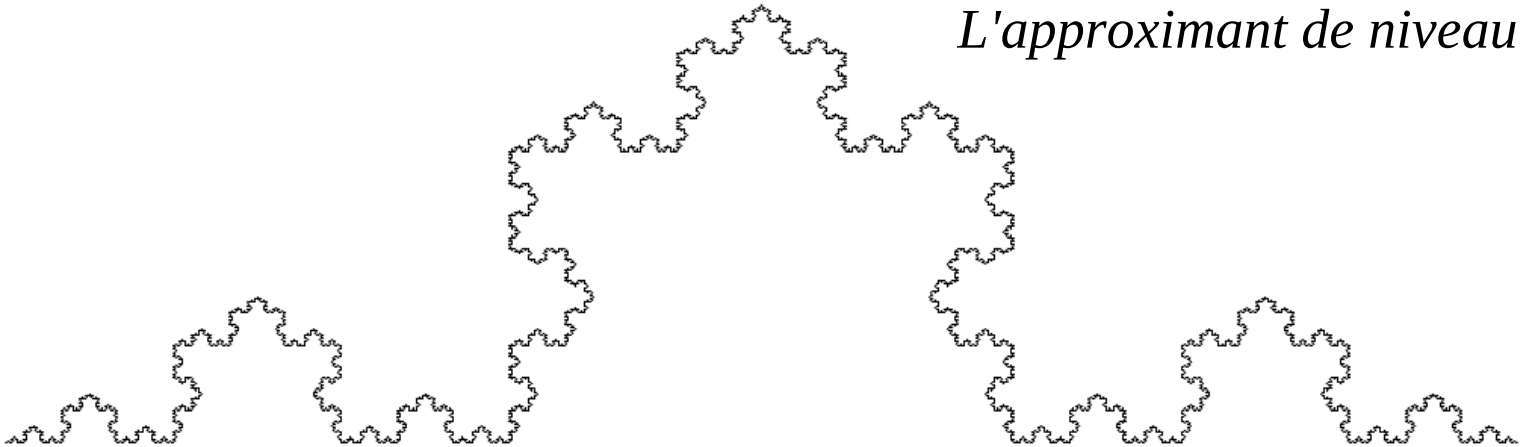
```
VK <- function(n,T) {
# Approximant de niveau n et base T
  if (n == 0) turtle_forward(T)
  else {
    VK(n-1,T/3)
    turtle_left(60)
    VK(n-1,T/3)
    turtle_right(120)
    VK(n-1,T/3)
    turtle_left(60)
    VK(n-1,T/3)
  }
}
```



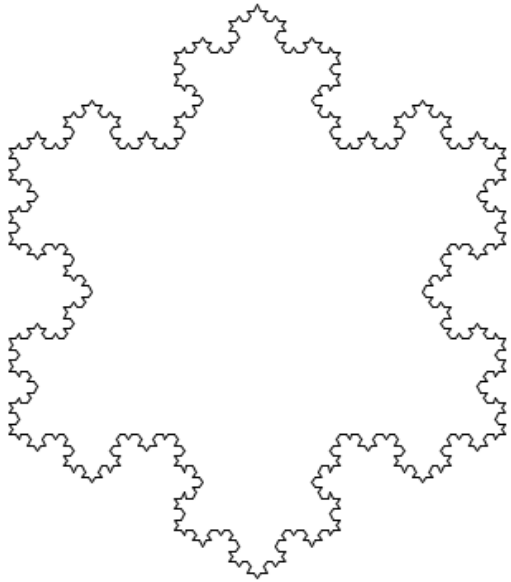
La courbe de Von Koch est la limite de la suite : $VK = \lim_{n\to+\infty} VK_n$.

Découverte en 1906, VK possède d'étranges propriétés. Par exemple, elle est continue mais n'admet de tangente en aucun point ! !

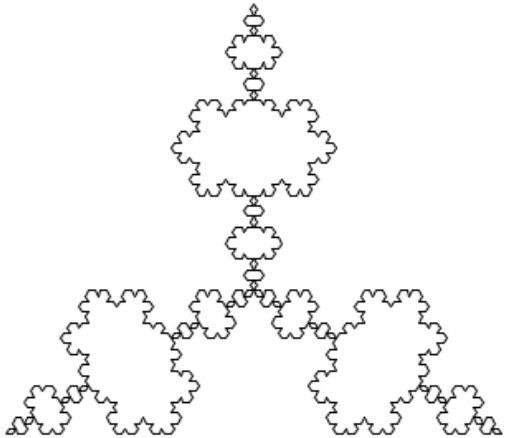
Variations sur la courbe de Von Koch



L'approximant de niveau 6



Le flocon de Von Koch



L'antiflocon

Questions?

Retrouvez ce cours sur le site web

`www.i3s.unice.fr/~malapert/R`