# Décomposition en série de Fourier d'un caractère avec LuaTeX et MPLIB

Maxime Chupin

# DÉCOMPOSITION EN SÉRIE DE FOURIER D'UN CARACTÈRE AVEC LUATEX ET MPLIB

Le vidéaste 3Blue1Brown [1], vulgarisateur mathématique sur YouTube, a réalisé une vidéo où il illustre la décomposition de Fourier d'un chemin fermé par des animations de mécanismes d'engrenages de cercles mis bout à bout. Le résultat est magnifique et envoûtant (voir figure 1).

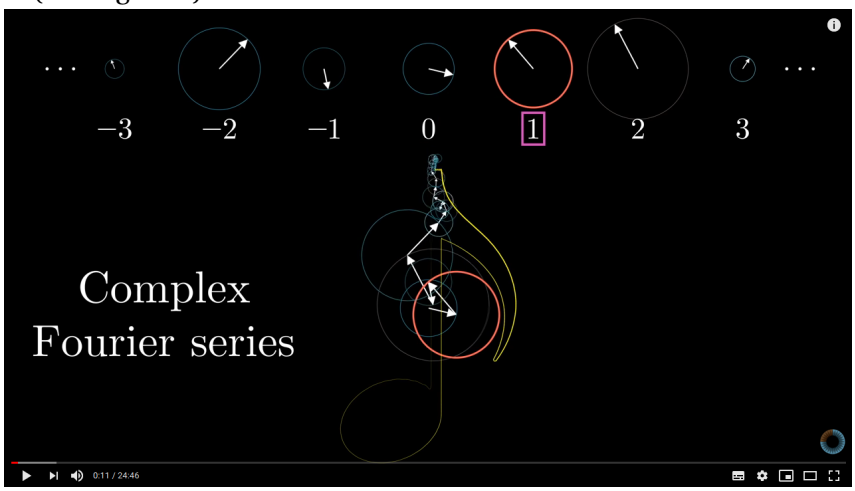



FIGURE 1 – Vidéo *Mais qu'est-ce qu'une série de Fourier ? Du transfert thermique à des dessins avec des cercles* du vidéaste 3Blue1Brown sur YouTube.

Il n'est pas facile de décrire avec des mots ces animations, mais il s'agit de mettre bout à bout des cercles de différents diamètres avec des vecteurs inscrits qui tournent à différentes vitesses, et l'extrémité de cette ligne brisée trace la courbe fermée considérée (la note de musique sur la figure 1).
Ce visionnage m'a donné l'envie [2] de réaliser cela avec nos outils préférés et plus particulièrement avec LuaLaTeX et METAPOST, lui-même inclus dans LuaTeX *via* la librairie Mplib. Je me suis aussi dit que cela serait intéressant de réaliser ces animations avec le contour d'un glyphe d'un caractère (d'une seule partie évidemment, connexe). C'est donc à ce chantier que je me suis attelé.

**Avertissement :** Cet article parle de LaTeX, de Lua, et de METAPOST [5]. Pour bien le comprendre, il faudra avoir quelques notions de ces trois aspects. Pour une bonne introduction à LuaLaTeX je conseille la lecture des cahiers GUTenberg dédiés [2]. Pour une bonne introduction à la programmation en Lua, allez voir *Programming in Lua* [7] (`lua.org`). Je ne peux malheureusement pas revenir sur toutes les commandes ou fonctions des divers langages dans cet article ; cependant, j'ai essayé de dégager l'essentiel.

## Le principe mathématique de l'affaire

On considère donc une courbe fermée dans $\mathbf{R}^2$ que l'on peut considérer comme une fonction $f : \mathbf{R} \to \mathbf{C}$ périodique. On considère que la période est égale à 1. Sans rentrer dans les détails, la décomposition en série de Fourier de $f$ est :

$$\forall t \in [0,1], \quad f(t) = \sum_{n=-\infty}^{+\infty} c_n(f) \mathrm{e}^{\mathrm{i}n2\pi t},$$

1. `https://www.youtube.com/watch?v=-qgreAUpPwM`
2. Pour un cours du traitement du signal que je dispense en troisième année d'étude supérieures.

où

$$c_n(f) = \int_{-1/2}^{1/2} f(t)\mathrm{e}^{-\mathrm{i}n2\pi t}\mathrm{d}t.$$

Numériquement, on travaillera avec des versions discrètes de cette décomposition en série de Fourier. Considérons deux entiers N et M *assez grands* et M pair. Dans le monde discret, nous n'aurons plus la fonction $f$ continue mais des échantillons le long du chemin, que l'on notera $(f_1, f_2, \ldots, f_\mathrm{N})$ où les $f_i \in \mathbf{C}$. On a alors :

$$\forall t \in [0,1], \quad f(t) \simeq \sum_{n=-\mathrm{M}/2}^{\mathrm{M}/2} \tilde{c}_n(f)\mathrm{e}^{\mathrm{i}n2\pi t}, \tag{1}$$

où

$$\tilde{c}_n(f) = \sum_{k=0}^{\mathrm{N}} \frac{1}{\mathrm{N}} f_{k+1}\mathrm{e}^{-\mathrm{i}\frac{2nk}{\mathrm{N}}}. \tag{2}$$

On appellera les $\tilde{c}_n(f)$ les coefficients de Fourier qui sont au nombre de $\mathrm{M}+1$.
Géométriquement, on peut voir la relation (1) comme une somme de vecteurs de $\mathbf{R}^2$ (donc mis bout à bout) de norme le module du nombre complexe $\tilde{c}_n(f)\mathrm{e}^{\mathrm{i}n2\pi t}$ et, comme orientation, son argument. Ainsi, quand $t$ parcourt l'intervalle $[0,1]$, ces vecteurs bout à bout tournent et l'extrémité du dernier vecteur dessine la courbe fermée [3] que l'on a décomposée.

## Récupération du contour en un ensemble de points de $\mathbf{R}^2$

Il nous faut donc construire, à partir d'un contour d'un glyphe, la suite $(f_1, f_2, \ldots, f_\mathrm{N})$ présentée ci-dessus.

### Grâce à METAPOST

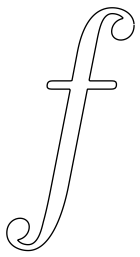
Dans un premier temps, il nous faut obtenir une discrétisation du contour fermé d'un glyphe donné. METAPOST, avec le format MetaFun [3], permet de faire cela assez facilement. Pour l'exemple, nous allons prendre le glyphe *f*, un nombre de points de 500 pour la discrétisation, et un certain facteur homothétique fixé à 0,1 pour l'affichage dans ce document :

```
fontmapfile "=lm-ec.map";
picture lettre;
path contourLettre;
lettre := glyph "f" of "ec-lmri10"; % les lettres sont en général
                                    % composées de plusieurs parties
path p;
nbrPoints := 500;
scale := 0.1;
beginfig(1);
   for item within lettre: % on parcourt toutes les parties (même si
                           % pour les lettres que nous choisirons il
                           % n'y en aura qu'une)
      contourLettre := pathpart item;
      for i:=1 upto nbrPoints:
          if i=1:
```

3. Ou plutôt une approximation du contour du glyphe.

```
            p := point i/nbrPoints along contourLettre; % on stocke
                                % dans un path tous les points
         else:
            p:= p--(point i/nbrPoints along contourLettre);
         fi;
      endfor;
      draw p scaled scale;
   endfor;
endfig;
end;
```

Ce code produit le résultat suivant (obtenu directement avec l'extension `luamplib` [4]).



Nous n'allons pas détailler ce code ici. Il paraît un peu complexe mais cela est dû au fait que la structure de `glyph` de METAPOST permet (heureusement) d'avoir plusieurs parties pour un glyphe. Même si, ici, nous ne considérerons que des lettres d'une seule partie (connexes), il faut bien se plier à la structure de données. Ce qu'il faut retenir, c'est que nous disposons d'un code METAPOST qui permet d'obtenir un ensemble de points constituant une discrétisation du contour du glyphe du caractère.

### La liste de points avec Lua

Le calcul de la décomposition en série de Fourier de cette courbe fermée doit être possible en TEX, cependant Lua nous offre plus de capacités, plus de rapidité, et plus de facilité de codage [4]. On veut donc récupérer cette liste de points du côté de Lua. L'avantage aussi, c'est que comme nous avons déjà dit, LuaTEX inclut METAPOST via la librairie Lua, Mplib (voir [6]).

### Chercher dans les fichiers de fontes.

Avec METAPOST (ou Mplib), on ne peut malheureusement utiliser qu'une fonte Type1. Il y a une petite subtilité concernant l'ouverture du fichier de fonte. En effet, METAPOST demande un type de fichier `pfb`, alors que `kpse` demande un type de fichiers `type1 fonts`. Taco Hoekwater, sur la liste de diffusion `metapost@tug.org` m'a fourni la fonction de recherche qui gère ce petit souci et que je fournis ci-dessous.

```
local mpkpse = kpse.new('luatex', 'mpost')

local function finder(name, mode, ftype)
  if mode == "w" then
     return name
  else
     if ftype == 'pfb' then
```

4. Au moins pour moi!

```
      ftype='type1 fonts'
    end
    return mpkpse:find_file(name,ftype)
  end
end
```

### Depuis METAPOST vers un tableau Lua

Passée cette petite difficulté technique, nous allons ici présenter une fonction Lua qui permet de construire un tableau Lua qui contient les points générés par METAPOST.

```
function getpathfrommp(s,nbrPoints,scale)
-- on definit une fonction qui récupère la liste de
-- nbrPoints points faite à partir du contour du caractere s
-- scale est un paramètre d'homothétie

   --on lance une session MetaPost avec la fonction de recherche
   local mp = mplib.new({find_file = finder,})
   -- on charge le format metafun
   mp:execute('input metafun ;')
   -- on stocke la sortie de l'exécution du code MetaPost
   local rettable
   rettable = mp:execute('fontmapfile "=lm-ec.map"; picture lettre;
    path contourLettre; lettre := glyph "' .. s .. '" of "ec-lmri10";
    path p; beginfig(1);
    for item within lettre: contourLettre := pathpart item; for i:=1
    upto'.. nbrPoints ..':
    if i=1: p := point i/'..nbrPoints..' along contourLettre;
    else: p:= p--(point i/'..nbrPoints..' along contourLettre);fi;
    endfor;
    draw p scaled '..scale..';
    endfor; endfig;end;') -- Code MetaPost presenté plus haut
   -- initialisation du tableau de points de sortie
   output = {}
   -- si l'exécution du code MetaPost s'est bien passée
   if rettable.status == 0 then
      figures = rettable.fig -- la liste des figures
      figure = figures[1] -- la première figure (la seule)
      local objects = figure:objects() -- la liste des objets
                                       -- composant la figure
      local segment = objects[1] -- le premier objet (le seul)
      for point =1, #segment.path do
          output[point] = {}
          output[point].x = segment.path[point].x_coord
          output[point].y = segment.path[point].y_coord
      end
  else
      print("error")
  end
  return output
end
```

Pour expliquer grossièrement le code ci-dessus, il s'agit de récupérer la sortie de l'exécution d'un code par METAPOST (Mplib ici). Cette sortie a une structure en Lua (il faut explorer la documentation de LuaTEX, section Mplib [6]). On parcourt donc cette structure pour extraire la liste de points qui nous intéressent : tout d'abord la liste de `figures`, qui ici est limitée à une seule `figure`, puis à l'intérieur de la première `figure`, on va chercher les « `objects` » qui là encore sont limités à une seul « `object` » (notre courbe fermée), puis on parcourt le « `path` » (METAPOST) de l'`object`, nommé ici « `segment` », et enfin on récupère les coordonnées $x$ et $y$ qu'on stocke dans notre variable de sortie « `output` ». Pour avoir accès aux fonctions Lua permettant de parcourir la structure de l'objet produit par l'exécution du code METAPOST, il faut se référer à la documentation de LuaTEX.
On notera qu'on a passé les paramètres du caractère à traiter, du nombre de points, et d'homothétie en arguments de la fonction Lua, le caractère dont on veut tracer le contour étant l'argument `s`. Notons aussi que ce code n'est pas robuste car si le glyphe correspondant au caractère `s` n'est pas connexe alors il y a de fortes chances que le code ne fonctionne pas.

## La décomposition en séries de Fourier avec Lua

### Appel à une bibliothèque extérieure

La décomposition en série de Fourier se fait avec des nombres complexes comme nous l'avons présenté précédemment. Les nombres complexes ne sont pas nativement gérés par Lua, mais il existe de nombreux codes disponibles sur le Web où les fonctions de calcul sur les nombres complexes sont implémentées. J'ai choisi le fichier `complex.lua` disponible à cette adresse :

`http://lua-users.org/wiki/ComplexNumbers`

Pour charger ce code, il faut :

1. du côté de LaTeX, charger l'extension `luapackageloader` (voir à la fin de cet article pour le code LaTeX complet) ;
2. du côté de Lua, appeler le fichier `complex.lua` par la ligne

```
complex = require "complex"
```

### Convertir la liste de coordonnées de $\mathbf{R}^2$ en une liste de nombres complexes.

Pour pouvoir programmer les calculs facilement, on crée une fonction de conversion de la liste de coordonnées obtenue par la fonction Lua `getpathfrommp` en une liste de nombres complexes. Ceci se fait par la fonction suivante qui ne nécessite pas plus d'explications.

```
function pathToComplex(path)
  local complexPath
  complexPath = {}
  for i=1,#path do
     complexPath[i] = complex.new(path[i].x,path[i].y)
  end
  return complexPath
end
```

**Implémentation de calcul des coefficients de Fourier.**

Le calcul des coefficients de Fourier $\tilde{c}_n(f)$ de l'égalité (2) s'implémente très facilement avec Lua, comme le montre le code suivant.

```
function cn(f,n)
  local CN
  CN = complex.new(0.0,0.0)
  local N = #f
  for i=0,N-1 do
     exposant = complex.new(0.0,-2.0*math.pi*n*i/N)
     Exp = complex.exp(exposant)
     CN = complex.add(CN,complex.mulnum(complex.mul(f[i+1],Exp),1.0/N
         ))
  end
  return CN
end
```

À partir de cela, il nous faut construire la liste

$$(c_{-M/2}, c_{-M/2+1}, \dots, c_{-1}, c_0, c_1, \dots, c_{M/2}),$$

ce qui se fait par la fonction Lua suivante :

```
function cnList(f)
  local CNlist = {}
  local M = #f
  for i=0,M do
    CNlist[i] = cn(f,math.floor(i-M/2))
  end
  return CNlist
end
```

## Tracé avec `mplibcode`

Une fois toutes ces briques de codes préparées, il nous suffit d'*implémenter le tracé* avec l'aide de l'environnement `mplibcode` de l'extension `luamplib` [5] [4]. Cette fonction a plusieurs arguments :

- un chemin discrétisé (`path`), c'est-à-dire l'ensemble des coordonnées $(x, y)$ du contour du glyphe ;
- sa conversion en complexes (`complexPath`) ;
- une liste des coefficients de Fourier (`cnList`) ;
- un nombre de coefficients de Fourier que l'on souhaite (M dans les équations précédentes), c'est-à-dire le nombre de cercles et vecteurs tracés (`nbrFourier`) ;
- un temps $t \in [0, 1]$.

```
function coreDecomp(path, complexPath, cnList,nbrFourier,t)
-- path : liste de points de R^2
-- complexePath : conversion de cette liste en complexes
```

5. Mais nous aurions pu générer du code Tikz par exemple.

```
-- cnList : liste des coefficients de Fourier
-- nbrFourier : nombre de coefficients M+1 dans les équations
  --initialisations
  local str
  local cnListRotated = {}
  local zero = math.floor(#cnList/2)
  local NFourier = math.floor(nbrFourier/2)
  cnListRotated[zero] = cnList[zero]
  -- on multiplie les coefficients par l'exponentielle e^{2i k pi t}
  for k=1, zero do
    cnListRotated[zero+k] = complex.mul(cnList[zero+k],complex.exp(
        complex.new(0.0, k*2*math.pi*t)))
    cnListRotated[zero-k] = complex.mul(cnList[zero-k],complex.exp(
        complex.new(0.0, -k*2*math.pi*t)))
  end
  -- on commence le code mplibcode
  local str = "\\begin{mplibcode}\nverbatimtex \\leavevmode etex;
      beginfig(1);"
  local mpCodeLetter = mpCodePath(path) -- code metapost du tracé du
      glyphe
                                        -- appel fonction extérieure
  str = str..mpCodeLetter -- concaténation de chaînes de caractères
  local currentC = complex.new(0,0) -- le point complexe courant (sur
                                    -- lequel on va placer le cercle
                                    -- suivant)
  -- on ajoute le code de tracé du cercle et du vecteur au point courant
  str = str .. mpCodeCircle(cnListRotated[zero],currentC)
  currentC = complex.add(currentC,cnListRotated[zero])
  for i=1,NFourier do -- on itère le procédé pour tous les modes de
      Fourier
    str = str .. mpCodeCircle(cnListRotated[zero+i],currentC)
    currentC = complex.add(currentC,cnListRotated[zero+i])
    str = str .. mpCodeCircle(cnListRotated[zero-i],currentC)
    currentC = complex.add(currentC,cnListRotated[zero-i])
  end
  str = str .. "endfig;\n\\end{mplibcode}\n\\newpage" -- on ferme tout
  return str
end
```

Pour aider à la lecture du code, notons que `cnListRotated[i]` correspond au termes dans la somme (1) : $\tilde{c}_n(f)\mathrm{e}^{2\mathrm{i}k\pi t}$, puisque la multiplication par $\mathrm{e}^{2\mathrm{i}k\pi t}$ peut être vue comme une rotation dans le plan complexe.
Cette fonction a pour but principal de construire une chaîne de caractères qui contient le code `mplibcode` qui sera *envoyé* à LaTeX *via* la fonction Lua `tex.sprint()`.
La fonction `coreDecomp` ci-dessus fait appel à deux autres fonctions Lua qui produisent le code METAPOST du tracé du graphique :

— la fonction `mpCodePath(path)` qui prend en argument la liste des points du contour du glyphe et qui trace ce dernier [6] ;

6. Il y a un cadre tracé autour aussi pour faire en sorte que toutes les images aient la même dimension et ainsi pouvoir enchaîner les images pour produire une animation.

— la fonction `mpCodeCircle(cn,shift)` qui prend en argument un coefficient de Fourier `cn` (après rotation) et un couple de $\mathbf{R}^2$ `shift` qui est l'extrémité de la ligne brisée au bout de laquelle le vecteur et le cercle doivent être dessinés.

Le code de ces deux fonctions est à lire ci-dessous. Elles consistent principalement à la concaténation de chaînes de caractères pour produire du code METAPOST.

```
function mpCodePath(path)
-- trace juste le chemin contour du glyphe en reliant simplement les
-- points entre eux
  local str
  str = ""
  str = str.."path p;\n"
  str = str.." p:="
  for i=1,#path do
     str = str.."("..string.format("%f",path[i].x)..","
         ..string.format("%f",path[i].y)..")--"
  end
  str = str.."cycle;\n"
  str = str.."draw p;\n"
  str = str.. "pair ll, lr, ur, ul; ll := llcorner p; ur := urcorner
     p; lr := lrcorner p; ul := ulcorner p;\n"
  str = str.. "Wdth := abs(xpart lr - xpart ll); Hght := abs(ypart ul
      - ypart ll); prcW := 0.8;prcH := 0.3;\n"
  str = str.. "draw (ll+(-prcW*Wdth,-prcH*Hght))--(lr+(+prcW*Wdth,-
     prcH*Hght))--(ur+(+prcW*Wdth,+prcH*Hght))--(ul+(-prcW*Wdth,+
     prcH*Hght))--cycle;\n"
  return str
end

function mpCodeCircle(cn,shift)
-- trace le cercle et le vecteur correspondant au coefficient de
-- Fourier cn (module et argument), centré au point de R^2 shift
  local str
  local abs,arg
  abs,arg= complex.polar(cn)
  str = "draw fullcircle scaled "..string.format("%f",2*abs)..
     shifted ("..string.format("%f",shift[1])..","..string.format("
     %f",shift[2])..") withcolor (0.7,0.7,0.7);\n"
  str = str.."drawarrow ((0,0)--("..string.format("%f",cn[1])..","
     ..string.format("%f",cn[2])..") )  shifted ("..string.format("
     %f",shift[1])..","..string.format("%f",shift[2])..") withpen
     pencircle scaled 1pt withcolor (0.7,0.3,0.3);"
  return str
end
```

### Générer les images pour tout $t \in [0,1]$

Pour créer l'animation, il suffit de générer les images avec une discrétisation de l'intervalle de temps $[0,1]$. Cela peut se faire avec la fonction suivante.

```
function plotDecompAnim(letter,nbrPoints,nbrFourier,scale,nbrFrame)
-- letter : caractère dont on veut faire la décomposition du glyphe
-- nbrPoints : nombre de points pour la discrétisation du contour
-- nbrFourier : nombre de modes pour la décomposition de Fourier
-- scale : facteur d'homothétie
-- nbrFrame : nombre pour diviser l'intervalle [0,1], et donc nombre d'
    images
  local str
  local path = getpathfrommp(letter,nbrPoints,scale)
  local complexPath = pathToComplex(path)
  local cnList = cnList(complexPath)
  local cnListRotated = {}
  local zero = math.floor(#cnList/2)
  local NFourier = math.floor(nbrFourier/2)
  for frame=0,nbrFrame-1 do
    t=frame/nbrFrame
    str = coreDecomp(path,complexPath,cnList,nbrFourier,t)
    tex.sprint(str)
  end
end
```

Cette fonction fait appel aux fonctions précédemment présentées ainsi qu'à la fonction de LuaTEX `tex.sprint()` qui permet d'envoyer la chaîne de caractères à LaTEX pour la composition.
Les fonctions Lua présentées sont toutes mises dans un fichier unique `Fourier.lua`.

## Animations et code

Une fois toutes ces fonctions Lua implémentées, il suffit de faire appel à la fonction Lua `plotDecompAnim` grâce à la macro `\directlua` comme le montre le code suivant.

```
\documentclass{article}
\usepackage{luapackageloader}
\usepackage{luamplib}

\directlua{dofile("Fourier.lua")}
\pagestyle{empty}

\begin{document}
\directlua{
  plotDecompAnim("f",300,50,0.26,360)
}
\end{document}
```

Pour conclure, nous n'avons considéré que 3 fichiers, le fichier LaTEX ci-dessous `fourier.tex`, le fichier `Fourier.lua` qui contient toutes nos fonctions Lua présentées ici, et le fichier `complex.lua` récupéré sur le web. Pour compiler et produire le PDF qui contient autant de pages que d'images, il suffit de mettre ces trois fichiers dans le même répertoire et d'exécuter `lualatex` sur notre fichier `fourier.tex`.

```
user $> lualatex fourier.tex
```

Si vous lisez ce PDF avec Acrobat Reader, vous pourrez lire l'animation générée (cf. figure 2) avec l'extension `animate` [1].

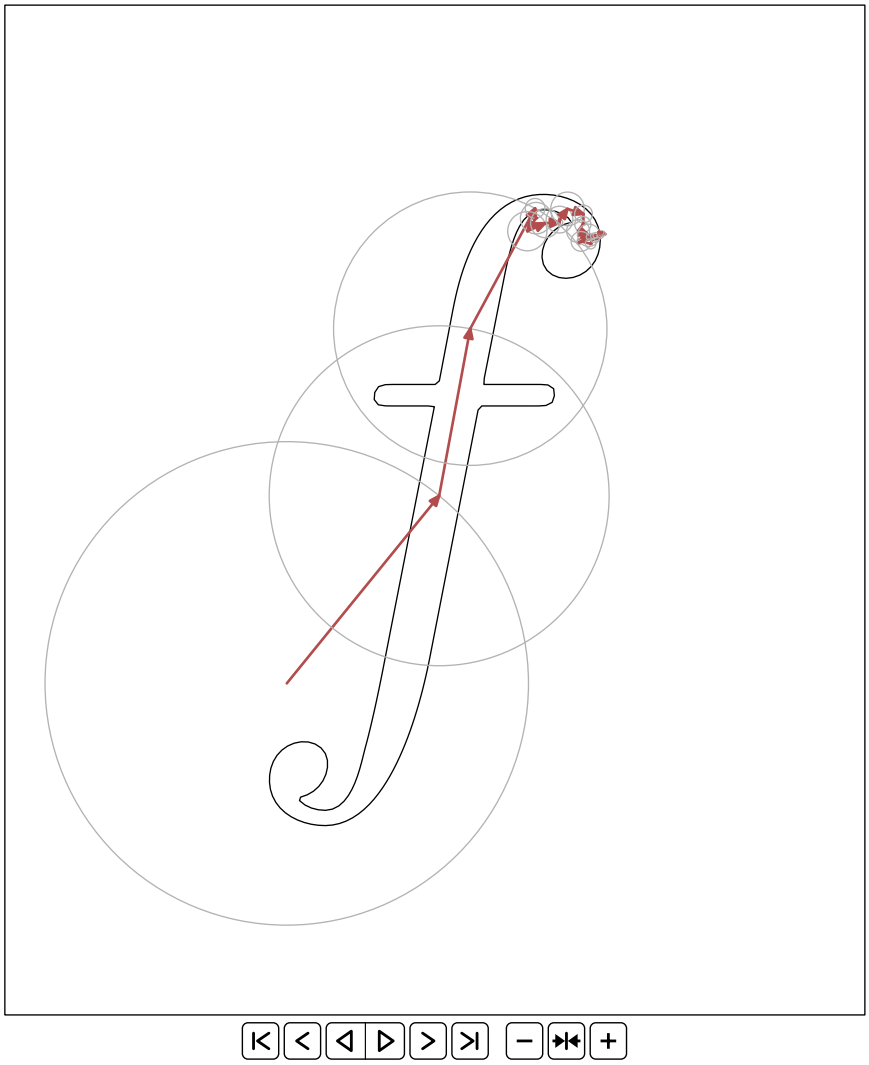


FIGURE 2 – Animation générée

Sinon, tous les codes et l'animation sont visibles et téléchargeables à l'adresse suivante :

`https://fougeriens.org/~mc/?page=exemples&dir=fourier`

Maxime CHUPIN

## Références


[1] G. ALEXANDER, *The animate package*, 2020 `https://ctan.org/pkg/animate`,

[2] ASSOCIATION GUTENBERG, *Cahiers n° 54-55 : Introduction à LuaTEX*, Octobre 2010, `http://cahiers.gutenberg.eu.org/cg-bin/feuilleter?id=CG_2010___54-55`.

[3] HANS HAGEN, *Metafun*, 2017, `http://www.pragma-ade.com/general/manuals/metafun-p.pdf`, v 2.0

[4] Hans Hagen, Taco Hoekwater, Elie Roux, Philipp Gesang et Kim Dohyun, *The `luamplib` package*, 2020, `https://ctan.org/pkg/luamplib`, v. 2.11.3.

[5] John D. Hobby and the MetaPost development team, *Metapost, a user's manual*, 2019, `https://ctan.org/pkg/metapost`, v. 2.0.

[6] LuaTeX development team, *LuaTeX reference manual*, March 2020, dernière version en ligne v. 1.12 : `http://www.luatex.org/svn/trunk/manual/luatex.pdf`.

[7] Roberto Ierusalimschy, *Programming in Lua*, 2016, `https://www.lua.org/`.