CAHIERS GUTenberg

LUALATEX POUR LES NON-SORCIERS, DEUX EXEMPLES

Maxime CHUPIN

Cahiers GUTenberg, nº 54-55 (2010), p. 37-56.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_2010___54-55_37_0>

LUALATEX POUR LES NON-SORCIERS, DEUX EXEMPLES

Maxime CHUPIN

RÉSUMÉ. — Cet article présente une utilisation de LuaTEX à un « petit niveau », c'est-à-dire sans être spécialiste ni de TEX, ni de Lua. Les exemples d'utilisation illustrent le traitement de fichiers externes par Lua, ainsi que l'utilisation de Lua pour certains calculs difficilement implémentables avec TEX. Ces exemples sont la création du code LATEX de tableaux à partir d'un fichier externe de valeurs, et la mise en place de la méthode des moindres carrés et de sa représentation graphique.

ABSTRACT. — This article present a way to use LuaTEX without being a expert of TEX or Lua. The examples illustrate the treatment of external files by Lua, and the use of Lua in order to perform some computations hardly implementable in TEX. These examples are the generation of LATEX tabular code from an external data file and the implementation of the method of least squares and its graphical presentation.

NOTE. — Merci à Anne-Sophie Philippe pour ses lectures attentives ainsi qu'à la communauté, notamment sur le groupe Usenet `fctt`.

1. INTRODUCTION

On entend de plus en plus parler du nouveau moteur LuaTEX [1]. Mais pour un *simple* utilisateur de LATEX, à quoi cela peut-il bien servir ? C'est le propos de cet article. En effet, n'étant qu'un utilisateur de base de LATEX, je me suis lancé dans l'aventure LuaTEX en utilisant le format LATEX. Je ne me suis pas plongé dans tous les aspects de LuaLATEX, à savoir, la gestion de l'UTF-8, la gestion des formats modernes de fontes, ni

1. Thème récurrent dans les journées GUTenberg `http://www.gutenberg.eu.org/spip.php?rubrique4` ainsi que dans les publications du TEX Users Group `http://www.tug.org/TUGboat/`.

dans l'utilisation de METAPOST grâce à `mplib`... J'ai cherché à montrer comment, sans être expert ni de TEX ni de Lua, on peut utiliser ce langage de script qu'est Lua pour se faciliter la vie.

Dans cet article deux exemples d'utilisation sont présentés. Dans un premier temps, je vais montrer comment on peut utiliser LuaTEX pour la fabrication automatique des tableaux à partir de fichiers externes de données, puis dans un second temps, comment la méthode des moindres carrés peut être implémentée grâce à Lua et Ti*k*z.

Cet article n'est en aucun cas un modèle de programmation, il est certain que les exemples présentés peuvent être améliorés, peut-être même les codes sont-ils maladroits. Ce n'est pas le propos de l'article, il montre comment un débutant, comme moi, peut se servir de LuaTEX pour se faciliter la vie et entreprendre la création de fonctionnalités plus facilement qu'avec TEX seul. Je suppose quand même que le lecteur a quelques bases en programmation. Le langage Lua s'apprend et se comprend assez vite.

2. Fabrication de tableaux à partir d'un fichier externe

Il arrive fréquemment, du moins en tant qu'étudiant en sciences, d'avoir à retranscrire des résultats sous forme de tableaux et cela souvent à partir de fichiers générés par un programme externe (Octave, Open Office, R, Scilab, etc.). La création de tels tableaux dans le document LATEX est parfois fastidieuse et, si par malheur, les données venaient à être modifiées, la saisie serait alors à recommencer. Ayant déjà fabriqué un petit script perl créant le code LATEX d'un tableau automatiquement à partir d'un fichier texte[2], je me suis dit que l'utilisation de LuaTEX présentait l'avantage de ne pas avoir à multiplier les compilations (perl et pdfTEX). De plus, nous allons voir que le traitement de chaînes de caractères par Lua est très facile.

2.1. Principe de fonctionnement

Soit, à notre disposition, un fichier de données sous forme de texte `donnees.txt`. On suppose que les séparateurs de colonnes sont une ou plusieurs espaces et que les séparateurs de lignes sont un ou plusieurs retours à la ligne.

2. `http://melusine.eu.org/syracuse/wiki/doku.php/mc/ptab/`.

Voici un exemple d'un tel fichier :

```
 2,01    4,78   3,0    7,7    8,2      9,1
 3,31    1,8    7,0   16,7    9,1      7,13
 7,41    4,8    2,1    8,1    5,22    10,1
12,31   45,78   1,0   88,7    8,3     12,1
```

Cet exemple est volontairement court et la macro que je vais présenter n'a que peu d'intérêt pour un tel tableau !

L'idée de cette macro est de produire automatiquement ce code LaTeX :

```
2,01&4,78&3,0&7,7&8,2&9,1\\
3,31&1,8&7,0&16,7&9,1&7,13\\
7,41&4,8&2,1&8,1&5,22&10,1\\
12,31&45,78&1,0&88,7&8,3&12,1\\
```

Voyons maintenant comment procéder.

2.2. Passage à Lua

La syntaxe imaginée ici est la suivante. On considère toujours notre exemple de fichier de données `donnees.txt`, c'est-à-dire un tableau 4×6. L'utilisateur doit écrire l'environnement d'encadrement du tableau (`tabular`, `matrix`, etc.) ainsi que, s'il le souhaite, la ligne de dénomination des colonnes. Voici un exemple d'invocation par appel de la macro avec, pour encadrement du tableau, l'environnement `tabular`.

```
\begin{tabular}{|c|c|c|c|c|c|}
  \hline
  Titre1&Titre2&Titre3&Titre4&Titre5&Titre6\\\hline
  \luatableau{donnees.txt}
\end{tabular}
```

Reste alors à définir la commande `\luatableau`. Ceci peut se faire de la façon suivante.

```
\directlua{%
  dofile(kpse.find_file("split.lua"))
}
\newcommand\luatableau[2][]{%%o
  \directlua{
    tableau("#2","#1")
  }
}
```

Décorticons un peu ce simple code. Ceci est une macro à deux arguments. Le premier est optionnel et est vide par défaut. Il s'agit de l'option de séparation des lignes du tableau. Pour un tableau classique, il faut spécifier `hline` si l'on souhaite séparer les lignes du tableau par un trait. Le second argument est la chaîne de caractères indiquant le chemin du fichier de données que l'on souhaite traiter.

Nous voyons que cette définition de macro ne contient qu'un seul appel à Lua. Celui-ci se fait grâce à la nouvelle primitive du moteur LuaTEX `\directlua`. Cette commande permet d'écrire du code Lua qui sera développé par TEX puis envoyé à l'interpréteur Lua. Notons qu'il faut bien garder à l'esprit que l'argument de `\directlua` sera développé par TEX. C'est en partie pour cela qu'il est préférable d'écrire un fichier `.lua` dans lequel sont implémentées les fonctions Lua. C'est ici le cas avec le fichier `split.lua`.

On le charge avec la fonction Lua `dofile`[3] qui définit la fonction Lua `tableau` qui prend en argument les deux arguments de la macro LATEX. Notons qu'il est préférable de charger le fichier de définition des fonctions Lua en dehors de la définition de la macro comme il est présenté dans le code. Lorsque l'on charge ainsi le fichier `.lua`, les fonctions définies dans celui-ci sont accessibles dans chaque appel à Lua via `\directlua`.

Voyons maintenant comment on utilise Lua.

2.3. Le script Lua

Décrivons l'algorithme du script que voici.

Listing 1. — Fichier `split.lua` qui définit la fonction `tableau`.

```
1  function tableau(fichier,sepligne)
2     local default_input=io.input();
3     io.input(fichier);
4     if sepligne~="" then
5        sepligne="\\"..sepligne;
6     end
```

3. La fonction `kpse.find_file` est une fonction Lua propre à LuaTEX qui permet la recherche dans l'arborescence de la distribution TEX. En effet, contrairement aux `\input` de TEX, le `dofile` ne parcourt pas l'arborescence TeX.

```
7     local case={};
8     -- on lit le fichier ligne par ligne
9     for ligne in io.lines() do
10       case=string.explode(ligne," +");
11       -- création du tableau
12       for j=1,#case do
13     -- s'il y a une virgule on l'encadre par des
           accolades
14          case[j]=string.gsub(case[j],",","{,}")
15    -- la syntaxe #case donne la taille du tableau case
16          if j==#case and i==#ligne then
17             tex.print(case[j]..\"\\\\\");
18          elseif j==#case then
19             tex.print(case[j]..\"\\\\\"..sepligne);
20       -- concaténation de \ avec la valeur de sepligne
21          else
22             tex.print(case[j]..\"&\");
23          end
24       end
25    end
26    io.input(default_input);
27 end
```

Nous voyons tout d'abord l'utilisation de la bibliothèque `io` de Lua. On sauvegarde tout d'abord l'entrée par défaut de `io` pour pouvoir la restituer à la fin de la fonction (ligne 25).[4] Ensuite, on met le fichier en entrée de `io`. Ensuite, en ligne 9, on rentre dans une boucle qui permet de lire le fichier en entrée ligne par ligne avec la fonction `io.lines()`.

Une fois que l'on a récupéré une ligne du fichier, il nous faut ń casser ż cette ligne pour en obtenir les cases. Ceci est réalisé par la fonction propre à LuaTEX `string.explode`[5]. Cette fonction prend deux arguments, la chaîne de caractères que l'on souhaite séparer et le *motif* par lequel cela doit se faire, à savoir `"\n+"` pour une suite de retour à la ligne, et `" +"` pour une suite d'espace, (ligne 7 et 10 du code ci-dessus).

4. On voit aussi l'utilisation de `local` pour la déclaration de `default_input` qui signifie que cette variable est interne à la fonction est n'existe pas en dehors. Ceci est une bonne habitude à prendre.

5. Elle a été rajoutée à Lua par les développeurs de LuaTEX, on ne peut donc pas l'utiliser avec Lua sans TEX.

On construit ainsi le tableau `case` qui contient toutes les cellules du tableau que l'on souhaite fabriquer.

Il ne reste donc plus qu'à faire écrire par Lua du code TEX. Ceci se fait grâce à la commande `tex.print`. Cette fonction permet, en quelque sorte, d'écrire une ligne virtuelle dans le source TEX. Il est donc facile de produire le code du tableau. Les lignes 12 à 24 du code ci-dessus se lisent assez facilement et ne nécessitent pas plus d'explications [6].

La gestion de l'argument optionnel se fait aussi très simplement. On voit à la ligne 4 du script que si l'argument de séparation n'est pas vide, alors la séquence `..` permet de concaténer `\\` avec `sepligne`. Par exemple, si `sepligne` vaut `hline` alors cela produit `\\hline`. En effet, comme avec le langage C, il existe des séquences d'échappement pour produire certains caractères : `\n` pour un retour à la ligne, `\t` pour une tabulation, `\\` pour un backslash, etc. Pour plus de détails sur la gestion des caractères en Lua, nous renvoyons à la section 2.4 de [2]. Finalement ce qui passe dans le `tex.print`, n'est pas `\\hline` mais `\hline` qui arrive alors dans le source TEX virtuellement. On obtient donc ce que l'on souhaite.

Pour permettre l'écriture des nombres décimaux avec un virgule quel que soit le mode de représentation du tableau choisi, il suffit de prendre la précaution de remplacer `,` par `{,}` dans la chaîne de caractère de chaque cellule. Ceci s'effectue à la ligne 16 du code ci-dessus, on y voit l'utilisation de la fonction `gsub` de la bibliothèque `string`. En effet, celle-ci prend 3 arguments, une chaîne *sujette*, un *motif*, et une chaîne de caractère de remplacement. Elle sert donc à remplacer dans la chaîne sujette, ici `case[i][j]`, toutes les occurrences du motif, ici `,`, par la chaîne de remplacement, ici `{,}`. Ainsi, en mode math de TEX, la virgule n'est plus traitée comme un signe de ponctuation, on obtient ainsi les bons espacements (voir le résultat plus bas sous forme de matrice). Si la chaîne de caractères sujette ne contient pas de virgule, la fonction `gsub` s'exécute sans problème, elle ne fait juste rien.

6. Je suppose que la lecture des boucles `for` ne pose pas de problème. Si tel n'est pas le cas, je vous invite vivement à vous familiariser avec ces bases de l'informatique pour que Lua puisse vous être utile.

Voici le résultat de la fonction ainsi définie. Ceci

```
\begin{tabular}{|c|c|c|c|c|c|}
  \hline
  Titre1&Titre2&Titre3&Titre4&Titre5&Titre6\\\hline
  \luatableau[hline]{donnees.txt}
\end{tabular}
```

produit

Titre1	Titre2	Titre3	Titre4	Titre5	Titre6
2,01	4,78	3,0	7,7	8,2	9,1
3,31	1,8	7,0	16,7	9,1	7,13
7,41	4,8	2,1	8,1	5,22	10,1
12,31	45,78	1,0	88,7	8,3	12,1

On peut le produire sous forme de matrice avec le code suivant :

```
\[
\begin{pmatrix}
  \luatableau{donnees.txt}
\end{pmatrix}
\]
```

$$\begin{pmatrix} 2,01 & 4,78 & 3,0 & 7,7 & 8,2 & 9,1 \\ 3,31 & 1,8 & 7,0 & 16,7 & 9,1 & 7,13 \\ 7,41 & 4,8 & 2,1 & 8,1 & 5,22 & 10,1 \\ 12,31 & 45,78 & 1,0 & 88,7 & 8,3 & 12,1 \end{pmatrix}$$

3. Les moindres carrés

Nous allons décrire ici l'implémentation de la méthode des moindres carrés et de sa représentation. Le principe de fonctionnement reste le même que dans l'exemple précédent, nous allons créer une macro LaTeX appelant simplement des fonctions Lua codées dans un fichier externe `.lua`. Ici, tout le travail est transféré sur Lua qui est bien meilleur calculateur que TeX…

3.1. Une méthode matricielle

L'implémentation des moindres carrés est assez simple à réaliser avec un langage de script car les calculs se mettent simplement sous forme

matricielle [7]. De plus, il existe une bibliothèque Lua qui met en place les opérations usuelles sur les matrices, à savoir, la multiplication, l'inversion, l'addition, etc. Les fichiers de cette bibliothèque sont à récupérer ici `http://lua-users.org/wiki/LuaMatrix` et à mettre soit dans le répertoire courant de travail, soit dans le `LUA_PATH` [8]. Cela peut aussi se mettre dans l'arborescence `texmf` de la distribution TEX mais je pense que, comme il ne s'agit pas d'une bibliothèque propre à LuaTEX, ça n'est pas judicieux. Cette bibliothèque pourrait vous servir dans un autre contexte que celui de TEX...

Pour utiliser cette bibliothèque, il faudra naturellement la charger au début de votre fichier `.lua`. Cela se fait grâce à la ligne suivante.

```
matrix = require "matrix"
```

Pour plus de précisions sur les bibliothèques Lua, nous renvoyons à [2] chapitre 15. Une fois ces fichiers installés, nous pouvons commencer.

3.2. Un brin de théorie

On ne va rien prouver ici, juste montrer le système matriciel à résoudre. Pour une bonne description de la méthode, allez jeter un coup d'œil sur le site `http://serge.mehl.free.fr/anx/meth_carr.html`.

Soit à notre disposition un ensemble de points $\{(x_i, y_i)\}_{i=1}^{n}$, on cherche alors à trouver le polynôme d'ordre p, $x \mapsto f(x)$, qui minimise

$$\Delta = \sum_{i=1}^{n} \big(y_i - f(x_i)\big)^2.$$

On définit pour cela

$$\forall k \in [\![0, 2p]\!],\ \mathrm{S}_k = \sum_{i=1}^{n} x_i^k,$$

et

$$\forall k \in [\![0, p]\!],\ \mathrm{W}_k = \sum_{i=1}^{n} y_i x_i^k.$$

7. Cela doit être possible avec TEX mais ma culture informatique fait que cela me semble bien plus compliqué qu'avec le langage Lua.

8. Sur mon système Debian, je les ai mis `/usr/local/share/lua/5.1/`.

En nommant $\{c_i\}_{i=1}^{p}$ les coefficients du polynôme, le système matriciel à résoudre est le suivant

$$\begin{pmatrix} S_0 & S_1 & S_2 & \cdots & S_p \\ S_1 & S_2 & \cdots & \cdots & S_{p+1} \\ S_2 & \cdots & & \cdots & S_{p+2} \\ \vdots & & & & \vdots \\ S_p & S_{p+1} & S_{p+2} & \cdots & S_{2p} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_p \end{pmatrix} = \begin{pmatrix} W_0 \\ W_1 \\ W_2 \\ \vdots \\ W_p \end{pmatrix}.$$

Ainsi il suffit d'inverser $\mathbf{S}$:

$$\mathbf{c} = \mathbf{S}^{-1}\mathbf{W}.$$

3.3. L'implémentation Lua

Commençons par la fonction permettant, à partir de la liste des points et de l'ordre de régression souhaité, d'obtenir les coefficients du polynôme. En utilisant la bibliothèque, la chose est simple. Voici le code.

Listing 2. — Fonction générant les coefficients du polynôme.

```
function coefficients(points, ordre)
   -- Cette fonction retourne la liste des coeffs de la
         régression d'ordre p
   -- f(x)=c0+c1*x+...+cp*x^p
   -- points : une liste de points (x,y) --> points[i].
        x et points[i].y
   -- ordre : p
   local taille = #points
 -- S matrice Sk (somme des x^k) et Wk (somme y*x^k)
   local tS=ordre+1
   local S=matrix(tS,tS)
   local W=matrix(tS,1)
   local p
   local xp
   local xi
   for i=1,(ordre+1) do
      for j=i,(ordre+1) do
         p=i+j-2
         for k=1,taille do
            if i==1 and j==1 then
               xp=1
            else
```

```
21                  xp=(points[k].x)^p
22               end
23               S[i][j]=S[i][j]+xp
24            end
25            S[j][i]=S[i][j]
26         end
27      end
28
29      for i=1,(ordre+1) do
30         for k=1,taille do
31            if i==1 then
32               xi=1
33            else
34               xi=(points[k].x)^(i-1)
35            end
36            W[i][1]=W[i][1]+(points[k].y)*xi
37         end
38      end
39
40      --la matrice des coeff du polynome de regression
41      local C=matrix{}
42      C=S^-1*W
43      return C
44   end
```

La liste de points est sous la forme d'un tableau (indexé par un entier) de tableaux (indexés par deux champs, `x` et `y`). Les tableaux en Lua sont des objets dynamiques et, par conséquent, ils sont facilement manipulables. Ici, pour obtenir la coordonnée x du 6e point, il suffit d'écrire `points[6].x`. Une fois les deux matrices **S** et **W** formées, la matrice **c** s'obtient très facilement grâce à la bibliothèque `matrix` (lignes 38 et 39 du code ci-dessus).

Une fois que l'on sait obtenir les coefficients du polynôme, il faut créer le code TEX de représentation graphique.

3.4. La fonction principale

J'ai choisi de créer une fonction principale `moindre` dont l'entête est la suivante :

```
function moindre(mode, versPoints, largeurPage, ordre)
```

Celle-ci sera précédemment appelée à partir d'une macro LaTeX et de la primitive `\directlua`. Le code de cette macro est présentée à la section 3.6.

Décrivons un peu les arguments de cette fonction Lua :

— L'argument `mode` valant soit `int` soit `ext` selon que l'on traite une série de points directement en argument de la commande LaTeX ou que l'on traite un fichier externe.

— L'argument `versPoints` peut lui aussi être sous deux formes, selon le mode. Pour le mode `int`, c'est une chaîne de caractères sous la forme `(x1,y1),...,(xn,yn)`. Pour le mode `ext` c'est simplement le chemin et le nom du fichier texte à traiter qui doivent être composés de lignes de deux coordonnées séparées par des espaces.

— L'argument `largeurPage` est la sortie de `\the\linewidth`, c'est-à-dire une chaîne de caractères : un nombre suivi de l'unité `pt`.

— L'argument `ordre` est l'ordre de régression souhaité.

Cette fonction gère le mode choisi et construit la liste des points en fonction du mode. Après avoir construit cette liste, elle délègue à la fonction `coefficients` l'obtention des coefficients du polynôme de régression, puis délègue encore la génération du code Tikz pour la représentation à une fonction `trace` que nous décrirons plus tard.

Voici le code de la fonction `moindre`.

Listing 3. — Fonction principale.

```
1  function moindre(mode, versPoints, largeurPage, ordre)
2     -- mode : int ou ext
3     -- versPoints : soit liste (x1,y1),...,(xn,yn)
4     --               soit fichier externe
5     -- largeur de la page courante en pt TeX (sortie de
          \the\linewidth)
6     ----------------------------------------
7     local default_input=io.input();
8     -- traitement des points
9     local tablePoints={}
10    if mode == "int" then
11       local points = string.explode(versPoints, ";")
12       local taille=#points
13       for i=1,taille do
14          Sx,Sy=string.match(points[i],"%((.+),(.+)%)")
```

```
15            tablePoints[i]={x=Sx,y=Sy}
16         end
17      elseif mode == "ext" then
18         io.input(versPoints) -- on ouvre le fichier
19         local xy
20         for ligne in io.lines() do
21            xy=string.explode(ligne," +")
22            table.insert(tablePoints,{x=xy[1],y=xy[2]})
23         end
24      else
25         print("Mode non existant")
26      end
27
28      local C=coefficients(tablePoints,ordre)
29      trace(tablePoints,C,largeurPage)
30      tex.sprint("\\par")
31      io.input(default_input)
32   end
```

Voyons comment on construit la liste de points suivant le mode.

Le mode `int`. — La liste est à construire à partir de l'argument de la commande LaTeX qui est de la forme `(x1,y1);(x2,y2);...;(xn,yn)`. Ainsi, on voit ligne 11 du code ci-dessus que si l'on se trouve dans le mode `int`, on commence à mettre dans un tableau `points` les sous-chaînes de caractères formées par ce qui se trouve entre les « ; » de la chaîne `versPoints`. Ceci se fait grâce à la commande Lua `string.explode` que nous avons déjà rencontrée. Une fois cela effectué, il nous faut extraire les valeurs x_i et y_i. Celles-ci se trouvent entre « (» et « , » pour les x et entre « , » et «) » pour les y. Pour extraire ces sous-chaînes de caractères, nous avons utilisé la fonction `string.match` ainsi que les *classes de caractères* (voir section 20.3 de [2]). En supposant que la variable `points[i]` est une chaîne de caractères de la forme `(<une suite quelconque de caractère>,<une suite quelconque de caractère>)`, le code Lua est alors le suivant.

```
Sx,Sy=string.match(points[i],"%((.+),(.+)%)")
```

Dans la recherche par classe de caractères « . » désigne n'importe quel caractère, et par conséquent, « .+ », n'importe quelle suite de caractères. « %(» désigne la parenthèse ouvrante et « %) » la parenthèse fermante. Les autres parenthèses désignent les groupes qui seront affectés à `Sx` et

Sy. Ainsi, dans la chaîne de caractères `points[i]`, tout ce qui est entre le caractère « (» et « , » sera affecté à Sx et tout ce qui est entre le caractère « , » et «) » sera affecté à Sy. Voyez comme il devient simple de traiter les chaînes de caractères grâce à Lua. Il suffit ensuite de construire le tableau de points comme on le voit à la ligne 15 du code ci-dessus.

Le mode `ext`. — La création du tableau de points pour ce mode ressemble au code exposé dans la section 2. En effet, le fichier texte externe contient, par ligne, deux valeurs séparées par une ou plusieurs espaces. En utilisant la bibliothèque `io` de Lua pour l'ouverture et la lecture du fichier, puis en utilisant la fonction `string.explode`, on construit facilement le tableau de points (lignes 17 à 24 du code précédent).

3.5. L'enrobage

J'ai choisi d'utiliser Ti*k*z pour la représentation des points et de la régression[9]. Il s'agit donc de faire générer à Lua du code Ti*k*z pour la représentation graphique. On va utiliser de façon abondante la fonction `tex.sprint` qui a l'avantage, par rapport à `tex.print`, d'inclure la chaîne de caractères dans la ligne courante. Le comportement est donc plus naturel. Pour plus d'explications nous renvoyons à la lecture de l'article du présent cahier intitulé « Un guide pour LuaLaTeX ». Je vais décortiquer le code petit à petit en omettant certaines parties qu'il ne me semble pas pertinent d'expliciter. Je laisse au lecteur le soin de lire tout le code présenté plus bas pour les détails sur la génération du code Ti*k*z.

Voici l'entête de notre fonction.

```
function trace(points,coeff,largeurPage)
```

Notons tout d'abord que c'est cette fonction qui va se servir de l'argument de la fonction principale `largeurPage`. En effet, nous voulons que notre graphique tienne dans notre page, ainsi la largeur de celui-ci doit être inférieure à `\linewidth`. Pour cela, il nous faut récupérer l'amplitude selon *x* et l'amplitude selon *y* et ajuster les unités Ti*k*z pour adapter

9. Ceci sera bientôt possible avec MetaPost (que je connais mieux) puisque le moteur LuaTeX contient la bibliothèque `mplib` qui permet d'utiliser le language MetaPost directement avec TeX. Malheureusement, son utilisation avec LaTeX n'est pas encore tout à fait au point.

notre graphique à la largeur de page. J'ai choisi 80% de la largeur de page pour l'axe des abscisses et 60% pour l'axe des ordonnées.

Il est intéressant de voir comment le tri du tableau de points peut se faire. Chaque élément `points[i]` contient deux champs, `x` et `y`, en choisissant le champ `x`, le tri de ce tableau peut se faire comme ceci :

```
table.sort(points, function (a,b) return (a.x<b.x) end)
```

La fonction `sort` de la bibliothèque `table` est une fonction *higher-order*, on peux donc redéfinir son fonctionnement en second argument par une fonction Lua. Ce fonctionnement est décrit dans [2], chapitre 6. Ici, le choix est de comparer les valeurs selon le champ `x`. Après avoir effectué ce tri, il devient simple de récupérer les maximum et minimum :

```
-- min max et pas (50 points sur la courbe)
local minX=points[1].x
local maxX=points[#points].x
local pas=math.abs(maxX-minX)/50
```

J'ai choisi de définir la courbe de régression par 50 points, je définis donc le `pas` qui servira lors du tracé de la courbe.

Pour récupérer les minimum et maximum selon le champ `y`, on procède de la même manière en triant de nouveau notre tableau selon `y`.

```
table.sort(points, function (a,b) return (a.y<b.y) end)
local minY=points[1].y
local maxY=points[#points].y
```

Une fois obtenues ces amplitudes, on peut définir les unités en x et y pour la `tikpicture`. Pour cela, il faut traiter l'argument de largeur de page. Celui-ci est, rappelons-le, le résultat de `\the\linewidth`, donc c'est une valeur en `pt` suivie de l'unité. On souhaite le transformer en `cm` et supprimer la chaîne `pt`. J'ai tout de suite appliqué la réduction à 80%.

```
local conv=0.035145979
local larg=string.gsub(largeurPage,"pt","")
local largeurPageCm=0.8*larg*conv
```

On peut alors définir les unités.

```
local Xunit=largeurPageCm/math.abs(maxX-minX)
local Yunit=(0.6*largeurPageCm)/math.abs(maxY-minY)
```

Une fois que l'on a ce matériel, il suffit de générer le code Ti*k*z. La description de cette partie du code ne présente pas d'intérêt. Je vous invite donc à le lire par vous-même. Notons tout de même que ce code utilise deux couleurs `point` et `regression` qui sont les couleurs de tracé des points et de la courbe de régression et qui doivent être définies dans le préambule du document LaTeX.

Voici le code complet de la fonction.

Listing 4. — Définition de la fonction de représentation.

```
1  function trace(points,coeff,largeurPage)
2     -- code de tracé avec tikz, points table de (x,y),
           coeff matrix de
3     -- coeff et largeurPage sortie de \the\linewidth
4     ------------------------------------------------
5
6     -- on trie les points selon x utilisation de la
7     -- fonction sort (higher-order function)
8     table.sort(points, function (a,b) return (a.x<b.x)
           end)
9     -- min max et pas (50 points sur la courbe)
10    local minX=points[1].x
11    local maxX=points[#points].x
12    local pas=math.abs(maxX-minX)/50
13    -- idem selon y, l'odre dans lequel
14    -- est le tableau n'importe pas
15    table.sort(points, function (a,b) return (a.y<b.y)
           end)
16    local minY=points[1].y
17    local maxY=points[#points].y
18
19    --pt to centimètre
20    local conv=0.035145979
21    local larg=string.gsub(largeurPage,"pt","")
22    local largeurPageCm=0.8*larg*conv
23
24    -- unité
25    local Xunit=largeurPageCm/math.abs(maxX-minX)
26    local Yunit=(0.6*largeurPageCm)/math.abs(maxY-minY)
27
28
```

```
29      -- on écrit du code tikz
30      -- on ouvre la figure en réglant les unités
31      tex.sprint("\\noindent\\begin{tikzpicture}[x="..
              Xunit.."cm,y="..Yunit.."cm]")
32      -- les axes
33      tex.sprint("\\draw[->,line width=0.7pt] ("..points
              [1].x ..","..(minY-0.5)..")--("..(points[#
              points].x+0.4) ..","..(minY-0.5)..")")
34      for i=1,9 do
35         tex.sprint("node[pos="..(0.1*i)..
              "]{\\begin{
                 tikzpicture}\\draw[-](0,0)--(0,0.1);\\end{
                 tikzpicture}}")
36         tex.sprint("node[pos="..(0.1*i)..",anchor=north]{
                 "..(string.format(" %.3f",(0.1*i*(points[#
                 points].x+0.4-points[1].x))))..
              "}")
37      end
38      tex.sprint("node[pos=0.95,anchor=north] {$x$}")
39      tex.sprint(";")
40      tex.sprint("\\draw[->,line width=0.7pt] ("..(minX)..
              ","..(minY-0.5)..")--("..(minX)..","..(maxY)..
              ")")
41      for i=1,9 do
42         tex.sprint("node[pos="..(0.1*i)..
              "]{\\begin{
                 tikzpicture}\\draw[-](0,0)--(0.1,0);\\end{
                 tikzpicture}}")
43         tex.sprint("node[pos="..(0.1*i)..",anchor=east]{"
                 ..(string.format(" %.3f",(0.1*i*(maxY-minY
                 +0.5))))..
              "}")
44      end
45      tex.sprint("node[pos=0.95,anchor=east] {$y$}")
46      tex.sprint(";")
47
48      -- on met les points
49      for i=1,#points do
50         tex.sprint("\\fill[color=point]("..points[i].x ..
                 ","..points[i].y ..") circle (2pt) ;")
51      end
52      -- on trace la regression
53      tex.sprint("\\draw[color=regression,line width=1pt]"
              )
54      for i=1,51 do
```

```
55        local x=minX+(i-1)*pas
56        local y=tostring(f(x,coeff))
57        if i~=51 then
58           tex.sprint("("..x..","..y..")--")
59        else
60           tex.sprint("("..x..","..y..");")
61        end
62     end
63     tex.sprint("\\node[draw,fill=white,text width="
          ..(1.4*math.abs(maxX-minX)).."cm] at("..(0.8*
          maxX)..","..(0.5*maxX)..") {régression : $");
64     for i=1,matrix.size(coeff) do
65        if i==1 then
66           tex.sprint(string.format(" %.4f", coeff[i][1])
                )
67        elseif i==2 then
68           tex.sprint(string.format(" %.4f", coeff[i][1])
                ..."x")
69        else
70           tex.sprint(string.format(" %.4f", coeff[i][1])
                ..."x^{"..(i-1).."}")
71        end
72
73        if i~= matrix.size(coeff) then
74           if coeff[i+1][1]>=0 then
75              tex.sprint("+")
76           end
77        end
78     end
79     tex.sprint("$};\\end{tikzpicture}")
80  end
```

Pour des questions de lisibilité, j'ai aussi défini une fonction `f` qui permet d'obtenir l'image de x par le polynôme de régression. La voici.

```
function f(x,coeff)
   -- fonction qui a x renvoie f(x)
   local taille=matrix.size(coeff)
   local puiss=matrix(taille,1)
   for i=1,taille do
      puiss[i][1]=(x)^(i-1)
   end
```

```
    return matrix.transpose(coeff)*puiss
end
```

3.6. Les macros LaTeX

J'ai choisi de créer deux macros LaTeX, une pour chaque mode, à savoir le mode `int` et le mode `ext` de la fonction Lua `moindre`. Chacune de ces macros est construite sur le même schéma, elles ont deux arguments, le premier est l'ordre de régression souhaité et le deuxième est, suivant le cas, la liste des points sous la forme `(x1,y1);(x2,y2);...;(xn,yn)` ou le chemin d'accès et le nom du fichier contenant la liste des points, par exemple `./le/chemin/vers/monfichier.txt`.

```
\directlua{%
  dofile("moindre.lua")
}
\newcommand\moindreInt[2]{%
  % deux arguments
  %% #1 l'ordre de régression
  %% #2 la suite de points sous la forme :
  %%% (x1,y1);(x2,y2);...;(xn,yn)
  \directlua{%
    moindre("int","\luatexluaescapestring{#2}","\the\
          linewidth","\luatexluaescapestring{#1}")
  }
}

\newcommand\moindreExt[2]{%
  % deux arguments
  %% #1 l'ordre de régression
  %% #2 le chemin et le nom du fichier à traiter, par
       exemple
  %%% ./le/chemin/vers/monfichier.txt
  \directlua{%
    moindre("ext","\luatexluaescapestring{#2}","\the\
          linewidth","\luatexluaescapestring{#1}")
  }
}
```

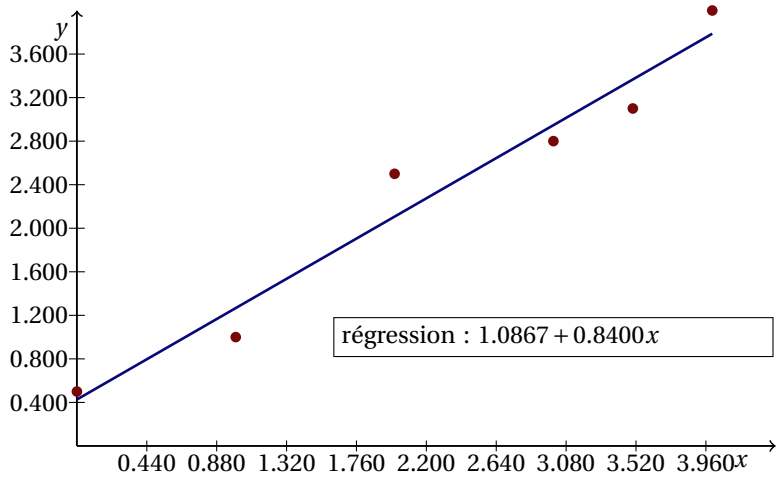

Figure 1. — Exécution en mode `ext`, ordre de régression $p = 1$

```
\definecolor{point}{RGB}{120,8,8}
\definecolor{regression}{RGB}{8,8,120}
```

Tout d'abord, et comme dans l'exemple précédent, on charge le fichier `moindre.lua` pour avoir accès dans les `\direclua` suivants aux fonctions définies dans ce fichier. Ensuite, ces macros sont assez simples, elles exécutent le fichier `moindre.lua` pour charger les fonctions, et exécutent ensuite la fonction principale `moindre`. Notons l'emploi de la nouvelle macro de LuaTEX, `\luatexluaescapestring`, qui permet de gérer les caractères d'échappement de TEX, car, rappelons-le, l'argument de `\directlua` est tout d'abord développé par TEX avant d'être envoyé à l'interpréteur Lua. Ceci est donc une précaution.

3.7. Le résultat

Les figures 1 et 2 montrent deux graphiques obtenus respectivement en mode `ext` et `int`.

Celles-ci sont simplement produites par les deux lignes suivantes :

```
\moindreExt{1}{test.txt}
\moindreInt{4}{(1,2);(2,2.5);(3,4);(4,4.3);(5,5.5)
      ;(4.5,4.6)}
```

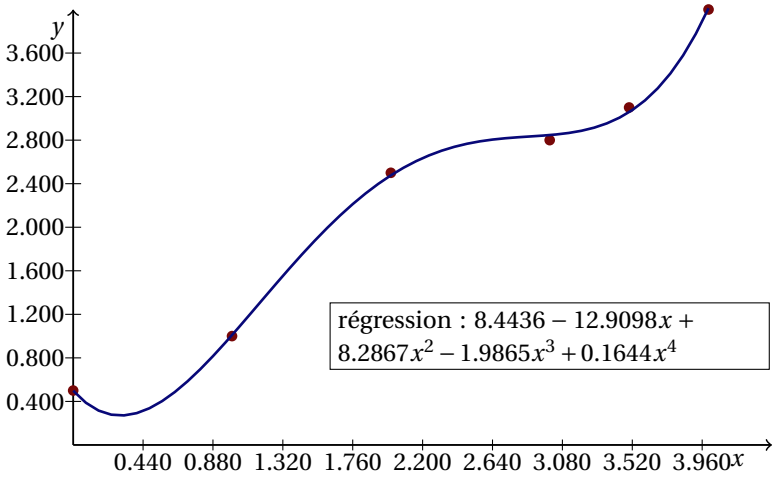

Figure 2. — Exécution en mode `int`, ordre de régression $p = 4$

La régression exponentielle n'est pas implémentée, il suffit pour cela de passer au logarithme et de faire une régression d'ordre 1. Ceci peut être un exercice pour se familiariser avec LuaLaTeX…

Bibliographie

1. J. André & J.-C. Charpentier, « Lexique anglo-français du *Companion* », *Cahiers GUTenberg* **49** (2007), p. 19-45, disponible en ligne : `http://cahiers.gutenberg.eu.org/fitem?id=CG_2007___49_19_0`.

2. R. Ierusalimschy, *Programming in lua*, 2e éd., Lua.Org, 2006, 1re édition disponible en ligne : `http://lua.org/pil/`.

3. LuaTeX development team, *LuaTeX reference manual*, 2010, dernière version en ligne disponible sur : `http://www.luatex.org/documentation.html`.

◖ Maxime Chupin
Fougères
16310 Cherves-Châtelars
mc@melusine.eu.org
`http://mc.notezik.com/`