Le package `lyalgo` : taper facilement de jolis algorithmes

Code source disponible sur `https://github.com/bc-latex/ly-algo`.

Version `0.0.0-beta` développée et testée sur Mac OS X.

Christophe BAL

2019-10-16

Table des matières

---

1 Introduction

Le but de ce package est d'avoir facilement des algorithmes [1] et aussi des contenus `verbatim` [2]. Les algorithmes mis en forme ne sont pas des flottants et utilisent une mise en forme proche de ce que l'on pourrait faire en `Python`.

2 `lymath`, un bon complément

Le package `lymath` est un bon complément à `lyalgo` : voir à l'adresse `https://github.com/bc-latex/ly-math`.

3 Écriture `pseudo-verbatim`

En complément à l'environnement `verbatim` est proposé l'nevironnement `pseudoverb`, pour « pseudo verbatim », qui permet d'écrire du contenu presque verbatim : ci-après, la macro `\squaremacro` définie par `\newcommand\squaremacro{$x^2$}` est interprétée mais pas la formule de mathématiques.

Code LaTeX

```
\begin{pseudoverb*}
  Prix1 = 14 euros
+ Prix2 = 30 euros
--------------------
  Total = 44 euros

========
Remarque
========
Attention car les macros comme \squaremacro{} sont interprétées mais pas les formules
de maths comme $x^2$ !
\end{pseudoverb*}
```

La mise en forme correspondante est la suivante sans cadre autour.

Rendu réel
ATTENTION ! Le cadre ne fait pas partie de la mise en forme.

```
  Prix1 = 14 euros
+ Prix2 = 30 euros
--------------------
  Total = 44 euros

========
Remarque
========
Attention car les macros comme x^2 sont interprétées mais pas les formules
de maths comme $x^2$ !
```

Il est en fait plus pratique de pouvoir taper quelque chose comme ci-dessous avec un cadre autour où le titre est un argument obligatoire *(voir plus bas comment ne pas avoir de titre)*.

1. Le gros du travail est fait par `algorithm2e`.
2. Tout, ou presque, est géré par `alltt`.

Une sortie console

```
  Prix1 = 14 euros
+ Prix2 = 30 euros
--------------------
  Total = 44 euros
```

Le contenu précédent s'obtient via le code suivant.

Code LaTeX

```
\begin{pseudoverb}{Une sortie console}
  Prix1 = 14 euros
+ Prix2 = 30 euros
--------------------
  Total = 44 euros
\end{pseudoverb}
```

Finissons avec une version bien moins large et sans titre de la sortie console ci-dessus. Le principe est de donner un titre vide via `{}`, c'est obligatoire, et en utilisant l'unique argument optionnel pour indiquer la largeur relativement à celle des lignes. En utilisant `\begin{pseudoverb}[.275]{}` au lieu `\begin{pseudoverb}{Une sortie console}`, le code précédent nous donne ce qui suit.

```
  Prix1 = 14 euros
+ Prix2 = 30 euros
--------------------
  Total = 44 euros
```

A retenir. C'est la version étoilée de `pseudoverb` qui en fait le moins. Ce principe sera aussi suivi pour les algorithmes.

4 Algorithmes en language naturel

4.1 Numérotation des algorithmes

Avant de faire les présentations, il faut savoir que les algorithmes sont numérotés globalement à l'ensemble du document. C'est tout simple et efficace pour une lecture sur papier.

4.2 `algorithm2e` tout en français

Le package `algorithm2e` permet de taper des algorithmes avec une syntaxe simple. La mise en forme par défaut de `algorithm2e` utilise des flottants, chose qui peut poser des problèmes pour de longs algorithmes ou, c'est plus gênant, pour des algorithmes en bas de page. Dans `lyalgo`, il a été fait le choix de ne pas utiliser de flottants, un choix lié à l'utilisation faite de `lyalgo` par l'auteur pour rédiger des cours de niveau lycée.

Dans la section suivante, nous verrons comment encadrer les algorithmes. Pour l'instant, voyons juste comment taper l'algorithme suivant où tous les mots clés sont en français.

Algorithme 1 : Suite de Collatz (u_k) – Conjecture de Syracuse

Donnée : $n \in \mathrm{N}$
Résultat : le premier indice i tel que $u_i = 1$ ou (-1) en cas d'échec

Actions
- $i, imax \leftarrow 0, 10^5$
- $u \leftarrow n$
- $continuer \leftarrow \top$
- **Tant Que** $continuer = \top$ **et** $i \leq imax$ **:**
  - **Si** $u = 1$ **:**
    - `# C'est gagné !`
    - $continuer \leftarrow \bot$
  - **Sinon :**
    - `# Calcul du terme suivant`
    - **Si** $u \equiv 0\ [2]$ **:**
      - $u \leftarrow u/2$; `# Quotient de la division euclidienne.`
    - **Sinon :**
      - $u \leftarrow 3u + 1$
    - $i \leftarrow i + 1$
- **Si** $i > imax$ **:**
  - $i \leftarrow (-1)$
- **Renvoyer** i

La rédaction d'un tel algorithme est facile car il suffit de taper le code suivant proche de ce que pourrait proposer un langage classique de programmation. Le code utilise certaines des macros additionnelles proposées par `lyalgo` *(voir la section 4.5)*. Si besoin voir juste après les lignes de code propres à la syntaxe `algorithm2e`.

Code LaTeX

```
\begin{algo*}
    \caption{Suite de Collatz $(u_k)$ -- Conjecture de Syracuse}

    \Data{$n \in \mathbb{N}$}
    \Result{le premier indice $i$ tel que $u_i = 1$ ou $(-1)$ en cas d'échec}

    \BlankLine    % Pour aérer un peu la mise en forme.

    \Actions{
        $i, imax \Store 0, 10^5$
        \\
        $u \Store n$
        \\
        $continuer \Store \top$
        \\
        \While{$continuer = \top$ \And $i \leq imax$}{
            \uIf{$u = 1$}{
                \Comment{C'est gagné !}
                $continuer \Store \bot$
            } \Else {
                \Comment{Calcul du terme suivant}
                \uIf{$u \equiv 0 \,\, [2]$}{
```

```
                    $u \Store u / 2$
                    \Comment*{Quotient de la division euclidienne.}
                } \Else {
                    $u \Store 3u + 1$
                }
                $i \Store i + 1$
            }
        }
        \If{$i > imax$}{
            $i \Store (-1)$
        }
        \Return{$i$}
    }
\end{algo*}
```

Le squelette du code précédent est le suivant.

***Squelette du code** `algorithm2e`*

```
\caption{...}

\Data{...}
\Result{...}

\Actions{
    ...
    \While{...}{
        \uIf{...}{
            ...
        } \Else {
            ...
            \uIf{...}{
                ...
            } \Else {
                ...
            }
            ...
        }
    }
    \If{...}{
        ...
    }
    \Return{...}
}
```

4.3 Des algorithmes encadrés

La version non étoilée de l'environnement `algo` ajoute un cadre, comme ci-dessous, afin de rendre plus visibles les algorithmes.

Algorithme 2 : Un truc bidon

Donnée : $n \in \mathrm{N}^\star$

Résultat : $\displaystyle\sum_{i=1}^{n} i$

Actions

$s \leftarrow 0$

Pour i **allant de** *1* **jusqu'à** n **:**

$s \leftarrow s + i$

Renvoyer s

Le code utilisé pour obtenir le rendu ci-dessus est le suivant où sont utilisées certaines des macros additionnelles proposées par `lyalgo` *(voir la section 4.5)*.

Code LaTeX

```
\begin{algo}
    \caption{Un truc bidon}

    \Data{$n \in \mathds{N}^\star$}
    \addalgoblank
    \Result{$\displaystyle \sum_{i = 1}^{n} i$}

    \Actions{
        $s \Store 0$
        \\
        \ForRange{$i$}{$1$}{$n$}{
            $s \Store s + i$
        }
        \Return{$s$}
    }
\end{algo}
```

L'environnement `algo` propose un argument optionnel pour indiquer la largeur relativement à celle des lignes. Ainsi via `\begin{algo}[.45] ... \end{algo}`, on obtient la version suivante bien moins large de l'algorithme précédent.

Algorithme 3 : Un truc bidon

Donnée : $n \in \mathrm{N}^\star$

Résultat : $\displaystyle\sum_{i=1}^{n} i$

Actions

$s \leftarrow 0$

Pour i **de** *1* **à** n **:**

$s \leftarrow s + i$

Renvoyer s

On peut utiliser un environnement `multicols` pour un effet sympa.

Algorithme 4 : Un truc bidon

Donnée : $n \in \mathrm{N}^\star$
Résultat : $\sum_{i=1}^{n} i$
Actions
  $s \leftarrow 0$
  Pour i **de** *1* **à** n **:**
    $s \leftarrow s + i$
  Renvoyer s

Algorithme 5 : Un truc bidon

Donnée : $n \in \mathrm{N}^\star$
Résultat : $\sum_{i=1}^{n} i$
Actions
  $s \leftarrow 0$
  Pour i **allant de** *1* **jusqu'à** n **:**
    $s \leftarrow s + i$
  Renvoyer s

4.4 Un titre minimaliste

Dans l'exemple ci-dessous on voit un problème à gauche où l'on a utilisé `\caption{}` avec un argument vide pour la macro `\caption`. Si vous avec besoin juste de numéroter votre algorithme comme ci-dessous à droite, utiliser à la place `\algovoidcaption`.

Algorithme 6 :

Données : Un titre vide ci-dessus !
Résultat : ...
Actions
  ...

Algorithme 7

Données : ...
Résultat : ...
Actions
  ...

4.5 Quelques macros additionnelles

4.5.1 Convention en bosses de chameau

Le package `algorithm2e` utilise, et abuse[3], de la notation en bosses de chameau comme par exemple `\uIf` et `\Return` au lieu de `\uif` et `\return`. Par cohérence, les nouvelles macros ajoutées par `lyalgo` utilisent aussi cette convention même si l'auteur aurait préféré proposer `\putin` et `\forrange` au lieu de `\PutIn` et `\ForRange` par exemple.

4.5.2 Affectations

Les affectations $x \leftarrow 3$ et $3 \rightarrow x$ se tapent `$x \Store 3$` et `$3 \PutIn x$` respectivement où sont utilisées les macros de type mathématique `\Store` et `\PutIn`.

4.5.3 Listes

AVERTISSEMENT – Premier indice

Pour le package, les indices des listes commencent toujours à un.

Opérations de base.

Voici les premières macros pour travailler avec des listes c'est à dire des tableaux de taille modifiable.

3. Ce type de convention est un peu pénible à l'usage.

1. *Liste vide.*
   `\EmptyList` imprime une liste vide [ ].
2. *Liste en extension.*
   `\List{4 ; 7 ; 7 ; -1}` produit $[4;7;7;-1]$.
3. *Le k^e élément d'une liste.*
   `\ListElt{L}{1}` produit $L[1]$.
4. *La sous-liste des éléments jusqu'à celui à la position k.*
   `\ListUntil{L}{2}` produit $L[..2]$.
5. *La sous-liste des éléments à partir de celui à la position k.*
   `\ListFrom{L}{2}` produit $L[2..]$.
6. *Concaténer deux listes.*
   `\AddList` est l'opérateur binaire $\boxplus$ qui permet d'indiquer la concaténation de deux listes.
7. *Taille ou longueur d'une liste.*
   La macro `\Len(L)` produit taille(L).

Modifier une liste – Versions textuelles.

1. *Ajout d'un nouvel élément à droite.*
   `\Append{L}{5}` produit *« Ajouter le nouvel élément 5 après la fin de la liste L. »* [4].
2. *Ajout d'un nouvel élément à gauche.*
   `\Prepend{L}{5}` produit *« Ajouter le nouvel élément 5 avant le début de la liste L. »* [5].
3. *Extraction d'un élément.*
   `\PopAt{L}{3}` produit *« Élément à la position 3 dans la liste L, cet élément étant retiré de la liste. »*.

Modifier une liste – Versions POO [6].

Les versions étoilées des macros précédentes fournissent une autre mise en forme à la fois concise et aisée à comprendre [7].

1. *Ajout d'un nouvel élément à droite.*
   `\Append*{L}{5}` fournit L`.`ajouter-droite(5).
2. *Ajout d'un nouvel élément à gauche.*
   `\Prepend*{L}{5}` fournit L`.`ajouter-gauche(5).
3. *Extraction d'un élément.*
   `\PopAt*{L}{3}` fournit L`.`extraire(3).

Modifier une liste – Versions symboliques.

Des versions doublement étoilées permettent d'obtenir des notations symboliques qui sont très efficaces lorsque l'on rédige les algorithmes à la main [8].

1. *Ajout d'un nouvel élément à droite.*
   `\Append**{L}{5}` donne $L \leftarrow L \boxplus [5]$.

4. Le verbe anglais *« append »* signifie *« ajouter »*.
5. Le verbe anglais *« prepend »* signifie *« préfixer »*.
6. *« POO »* est l'acronyme de *« Programmation Orientée Objet »*.
7. L'opérateur point `.` est défini dans la macro `\POOpoint`.
8. L'opérateur $\boxplus$ est défini dans la macro `\AddList`.

2. *Ajout d'un nouvel élément à gauche.*
   `\Prepend**{L}{5}` donne $L \leftarrow [\,5\,] \boxplus L$.
3. *Extraction d'un élément – Version pseudo-automatique.*
   `\PopAt**{L}{3}` donne $L \leftarrow L[\,..\,2\,] \boxplus L[\,4\,..\,]$ avec un calcul fait automatiquement par la macro. Bien entendu `\PopAt**{L}{1}` produit $L \leftarrow L[\,2\,..\,]$ au lieu de $L \leftarrow L[\,..\,0\,] \boxplus L[\,2\,..\,]$ puisque pour le package les indices des listes commencent toujours à un.
   Il est autorisé de taper `\PopAt**{L}{k}` pour obtenir $L \leftarrow L[\,..\,k-1\,] \boxplus L[\,k+1\,..\,]$. Par contre, `\PopAt**{L}{k-1}` aboutit au truc très moche $L \leftarrow L[\,..\,k-1-1\,] \boxplus L[\,k-1+1\,..\,]$. Les items suivants expliquent comment gérer à la main les cas problématiques via des macros plus généralistes.
   Attention ! *On notera que contrairement aux versions* `\PopAt` *et* `\PopAt*`*, l'écriture symbolique agit juste sur la liste. Si besoin, avec* `\PopAt**` *il faudra donc indiquer au préalable où stocker l'élément extrait via* $\cdots \leftarrow L[\,k\,]$.
4. *Extraction d'éléments consécutifs.*
   Lorsque les calculs automatiques ne sont pas faisables, on devra tout indiquer comme dans `\KeepLR{L}{k - 2}{k}`[9] afin d'avoir $L \leftarrow L[\,..\,k-2\,] \boxplus L[\,k\,..\,]$ qui est bien mieux que ce que nous avions obtenu ci-dessus : $L \leftarrow L[\,..\,k-1-1\,] \boxplus L[\,k-1+1\,..\,]$.
5. *Extractions juste à droite, ou juste à gauche.*
   `\KeepL{L}{k}` permet d'afficher $L \leftarrow L[\,..\,k\,]$ et `\KeepR{L}{k}` permet quant à lui d'écrire $L \leftarrow L[\,k\,..\,]$[10].

4.5.4 Boucles

Une boucle `POUR` peut s'écrire de façon succincte via `\ForRange*{a}{0}{12}{...}` pour obtenir ce qui suit.

Pour a **de** 0 **à** 12 **:**
⎣ ⋯

Pour une version sans ambiguïté possible, on utilisera `\ForRange{a}{0}{12}{...}` afin d'obtenir la rédaction plus longue suivante.

Pour a **allant de** 0 **jusqu'à** 12 **:**
⎣ ⋯

Pour en finir avec les boucles, il est facile d'obtenir l'écriture symbolique ci-dessous via `\For` qui est proposé par le package `algorithm2e` : il suffit de taper `\For{$a \in \CSinterval{0}{12}$}{...}` où la macro `\CSinterval` est proposée par le package `lymath`.

Pour $a \in 0\,..\,12$ **:**
⎣ ⋯

9. Le nom de la macro vient de *« keep left and right »* soit *« garder à droite et à gauche »*.
10. Les noms des macros viennent de *« keep left »* et *« keep right »* soit *« garder à gauche »* et *« garder à droite »*.