

Projet R-Dyndoc

R. Drouilhet

Cqls Team

Plan

- 1 Motivation
- 2 Découverte de dyndoc par des exemples
- 3 Programmation : environnements R et ruby
- 4 Programmation : variables dyndoc
- 5 Programmation : condition, sélection et boucles
- 6 Exemples plus avancés
- 7 dyndoc en résumé
- 8 Modes d'utilisation

Motivation initiale pour le projet dyndoc

A l'origine, (il y a plus de 15 ans)

- **Enseignement** : gain de temps lors de la rédaction des sujets d'examens et préparation de documents de cours
⇒ Marre du copier-coller-modifier et du "je sais plus où c'est!"
- **Consulting statistique** : idée de proposer des outils de génération automatique de rapports en intégrant les résultats extraits de traitements R .

Puis, (il y a \simeq 10 ans)

- **Programmation** : création d'un langage de type "chef d'orchestre"
 - ① dédié à la manipulation de contenu de **document** latex, html, ...
⇒ pas besoin de manipulation de `print` et "..."
 - ② utilisant tel quel ("as is") plusieurs langages (R , ruby, ...) aux caractéristiques complémentaires pour **dynamiser** le contenu.

Plan

- 1 Motivation
- 2 Découverte de dyndoc par des exemples
- 3 Programmation : environnements R et ruby
- 4 Programmation : variables dyndoc
- 5 Programmation : condition, sélection et boucles
- 6 Exemples plus avancés
- 7 dyndoc en résumé
- 8 Modes d'utilisation

L'habituel "Hello code" :

```
1  [#R<] mister <- "misteR" #R code
2  [#rb<] mister = "mister" #ruby code
3  [#<] This is not output!
4      {#def}hello[#,]name[Miss][#>]
5          [hello #{name}]
6      [#}
7  [#>]
8      [from Dyn, {#hello#}
9          |from Dyn, {#hello}Mister[#}
10         |from ruby, hello :{mister}
11         |from R, hello :r{mister}]
```

Résultat:

```
from Dyn, hello Miss
from Dyn, hello Mister
from ruby, hello mister
from R, hello misterR
```

Le “Hello code” avec librairie :

Le code ci-dessous étant le contenu du fichier helloLib.dyn

```
1 No output inside a library
2
3 [#def]hello[# ,]name[Miss][#>]
4   [Hello #{name}]
5   [#]
```

le code dyndoc suivant donne le même résultat que précédemment :

```
1 [#require]helloLib
2 [#main][#R<] mister <- "mister" #R code
3 [#rb<] mister = "mister" #ruby code
4 [#>]
5   [from Dyn, {#hello#}
6   |from Dyn, {#hello}Mister[#}
7   |from ruby, hello :{mister}
8   |from R, hello :r{mister}]
```

L'habituel "Hello code" en latex :

```
1  [#R<] mister <- "misteR" #R code
2  [#rb<] mister = "mister" #ruby code
3  [#<] This is not output!
4      {#def}hello[#,]name[Miss][#>]
5          [hello \textbf{#{name}}]
6      [#}
7  [#>]
8      [from Dyn, {#hello#}\
9      |from Dyn, {#hello}Mister[#]\
10     |from ruby, hello \textit{:#{mister}}\
11     |from R, hello \underline{:r{mister}}]
```

Résultat:

```
from Dyn, hello \textbf{Miss}\
from Dyn, hello \textbf{Mister}\
from ruby, hello \textit{mister}\
from R, hello \underline{r{mister}}
```

L'habituel "Hello code" en latex :

```
1  [#R<] mister <- "misteR" #R code
2  [#rb<] mister = "mister" #ruby code
3  [#<] This is not output!
4      {#def}hello[#,]name[Miss][#>]
5          [hello \textbf{#{name}}]
6      [#}
7  [#>]
8          [from Dyn, {#hello#}\\
9          |from Dyn, {#hello}Mister[#]\\
10         |from ruby, hello \textit{:#{mister}}\\
11         |from R, hello \underline{:r{mister}}]
```

Résultat:

from Dyn, hello **Miss**
from Dyn, hello **Mister**
from ruby, hello *mister*
from R, hello misteR

L'habituel "Hello code" en latex via textile :

```
1  [#R<] mister <- "misteR" #R code
2  [#rb<] mister = "mister" #ruby code
3  [#<] This is not output!
4      {#def}hello[# ,]name[Miss][#>]
5          [hello *#{name}*]
6      [#}
7  [#txtl>]
8          [from Dyn, {#hello#}
9          |from Dyn, {#hello}Mister[#}
10         |from ruby, hello __:{mister}__
11         |from R, hello +:r{mister}+]
```

Résultat:

```
from Dyn, hello \textbf{Miss}
from Dyn, hello \textbf{Mister}
from ruby, hello \textit{mister}
from R, hello \underline{misteR}
```

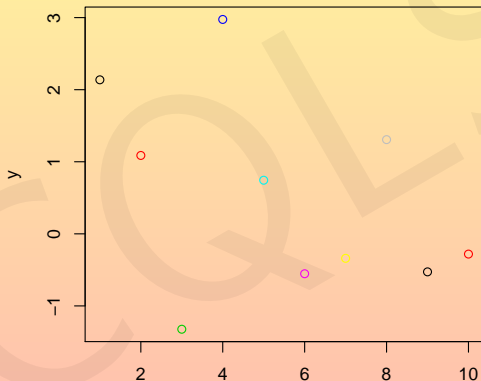
Le code dyndoc suivant :

```
1  {#VerbFrame}
2  [#>]\noindent \textbf{Session \texttt{R}}~ :
3  {#beamer}
4  y<- rnorm(10)
5  y
6  plot(1:10,y,col=1:10)
7  ##fig | img=test | opt=width=7cm
8  rexp(10)
9  {#beamer}
10 {#VerbFrame}
```

permet à un simple utilisateur de générer la page beamer suivante (modulo la définition des fonctions VerbFrame et beamer en mode développeur dyndoc) :

Session R :

```
> y<-rnorm(10)
> y
[1]  2.1372479  1.0876475 -1.3234185  2.9746359  0.7439182 -0.5534129
[7] -0.3396379  1.3072045 -0.5273199 -0.2807362
> plot(1:10,y,col=1:10)
```



```
> rexp(10)
[1] 0.2245419 1.2491204 0.6625015 0.7512126 0.2334184 1.6684865
[7] 0.5930386 0.3608202 0.3920849 0.5529308
```

Exemple Sweave (très basique)

```
1 \documentclass{article}
2 \begin{document}
3 <<=>
4 a<-c(1,3,2)
5 a
6 @
7 La moyenne est \Sexpr{mean(a)}.
8 \end{document}
```

Son équivalent dyndoc :

```
1 {#rverb}
2 a<-c(1,3,2)
3 a
4 [#}
5 La moyenne est :r{mean(a)}.
```

N.B. pas d'entête latex grâce à un système de "layout"

Plan

- 1 Motivation
- 2 Découverte de dyndoc par des exemples
- 3 Programmation : environnements R et ruby**
- 4 Programmation : variables dyndoc
- 5 Programmation : condition, sélection et boucles
- 6 Exemples plus avancés
- 7 dyndoc en résumé
- 8 Modes d'utilisation

Equivalent Sweave (par défaut) :

```
1  [#>] \textit{Mode séquentiel}~ :
2  [#R»] x <- rnorm(3)
3  x
4  [#>]
5  \textit{Mode imbriqué}~ :
6  {#rverb}
7  c(mean(x),sd(x))
8  [#]
```

Résultat:

Mode séquentiel :

```
> x <- rnorm(3)
> x
[1] -0.5738537 -0.6617936  0.8380576
```

Mode imbriqué :

```
> c(mean(x),sd(x))
[1] -0.1325299  0.8417027
```

Equivalent Sweave (echo=FALSE,results='hide') et \Sexpr :

```
1  [#R<]# No output even for the following "cat(...)"
2  a <- c("toto","titi")
3  cat(a, '\n')
4  a2 <- paste(a, 1:2, sep="")
5  cat(a2, '\n')
6  a3 <- paste(a2, collapse=" & ")
7  cat(a3)
8  [#>]Extracting the R results inside dyndoc:
9  a is :r{a}
10 a2 is #r{a2}
11 a3 is :r{a3}
```

Résultat:

Extracting the R results inside dyndoc:

a is toto,titi

a2 is toto1,titi2

a3 is toto1 & titi2

Idem en ruby :

```
1  [#rb<]# No output even for the following "puts"  
2  a=["toto","titi"]  
3  p a  
4  a2=a.each.with_index.map{|e,i| e+(i+1).to_s}  
5  puts a2  
6  a3=a2.join(" & ")  
7  puts a3  
8  [#>]Extracting the ruby results inside dyndoc:  
9  a is :{a}  
10 a2 is #rb{a2}  
11 a3 is :rb{a3}
```

Résultat:

Extracting the ruby results inside dyndoc:

a is toto,titi

a2 is totol,titi2

a3 is totol & titi2

Equivalent Sweave (echo=FALSE) :

```
1  [#R>]# Each output goes to the document
2  a <- c("toto", "titi")
3  cat(a, '\n')
4  a2 <- paste(a, 1:2, sep="")
5  cat(a2, '\n')
6  a3 <- paste(a2, collapse=" & ")
7  cat(a3)
```

Résultat:

```
toto titi
total titi2
total & titi2
```

Idem en ruby :

```
1  [#rb>]# Each output goes to the document
2  a=["toto","titi"]
3  p a
4  a2=a.each_with_index.map{|e,i| e+(i+1).to_s}
5  puts a2
6  a3=a2.join(" & ")
7  puts a3
```

Résultat:

```
["toto", "titi"]
toto1
titi2
toto1 & titi2
```

Plan

- 1 Motivation
- 2 Découverte de dyndoc par des exemples
- 3 Programmation : environnements R et ruby
- 4 Programmation : variables dyndoc**
- 5 Programmation : condition, sélection et boucles
- 6 Exemples plus avancés
- 7 dyndoc en résumé
- 8 Modes d'utilisation

Variables dyndoc :

```
1  [#>]toto[TOT0]      [#=]titi[TITI]
2  [#>]First step: ({toto}) AND ({titi})
3  [#>]toto+[ and TOT02] [#+]titi[ and TITI2]
4  [#>]Second step: ({toto}) AND ({titi})
5  [#>]toto[NewTOT0]   [#=]titi[NewTITI]
6  [#>]Third step: ({toto}) AND ({titi})
```

Résultat:

First step: (TOT0) AND (TITI)

Second step: (TOT0 and TOT02) AND (TITI and TITI2)

Third step: (NewTOT0) AND (NewTITI)

Variables dyndoc pour R et ruby :

```
1  [#=] toto[TOTO]
2  [#%] Ruby variables from Dyndoc environment
3  [#=] toto@[1,3,2]
4  [#>] [Content of toto@: #{toto@}<\n>]
5  [#%] R variables from Dyndoc environment
6  [#=] toto$c(1,3,2)
7  [#>] [Content of toto$: #{toto$}<\n>]
```

Résultat:

Content of toto@: 1,3,2

Content of toto\$: 1.0,3.0,2.0

Utiliser ces variables en R :

```
1  [#R<]
2  <toto@>[1]= sum(<toto@>)
3  <toto:> = tolower(<toto:>)
4  <toto$>[1] = diff(<toto$>[1:2])
5  [#>]After R,
6  toto=#{toto}
7  toto@=#{toto@}
8  toto$=#{toto$}
```

Résultat:

After R,

toto=toto

toto@=6,3,2

toto\$=2.0,3.0,2.0

Utiliser ces variables en ruby :

```
1  [#rb<]
2  <toto@[0]= 1
3  <toto:> = <toto:>.uppercase
4  <toto[1]$> -= 1
5  [#>]After ruby,
6  toto=#{toto}
7  toto@=#{toto@}
8  toto$=#{toto$}
```

Résultat:

After ruby,

toto=TOTO

toto@=1,3,2

toto\$=1.0,3.0,2.0

Plan

- 1 Motivation
- 2 Découverte de dyndoc par des exemples
- 3 Programmation : environnements R et ruby
- 4 Programmation : variables dyndoc
- 5 Programmation : condition, sélection et boucles**
- 6 Exemples plus avancés
- 7 dyndoc en résumé
- 8 Modes d'utilisation

Instruction `if` (mode imbriqué) :

```
1  [#rb<]n=11
2  [#>]:{n} is {#if}n%2==1[#>]odd[#else]even[#if} and |
3  [#R<]n <- 28
4  [#>]:r{n} is {#if}:r{n%%2==1}[#>]odd[#else]even[#if}
```

Résultat:

11 is odd and 28 is even

Instruction `if` (mode séquentiel) :

```
1  [#rb<]n=14[#R<]n <- 25
2  [#>]:{n} is [#?]n%2==1[#>]odd[#?]else[#>]even[#?]end
3  [#>] and :r{n} is |
4  [#?]:r{n%%2==1}[#>]odd[#?]else[#>]even
```

Résultat:

14 is even and 25 is odd

Instruction if en R :

```
1  [#R>]n=31
2      if(n%%2==1) {#>}:r{n} is odd[#>}
3      else {#>}:r{n} is even[#>}
```

Résultat:

31 is odd

Instruction if en ruby :

```
1  [#rb>]n=33
2      if n%2==1
3          {#>}:{n} is odd[#>}
4      else
5          {#>}:{n} is even[#>}
6      end
```

Résultat:

33 is odd

Instruction case :

```
1  [#=]todo[first,third,second,first]
2  [#>]List of the translated words in French:
3  {#case}#{todo}
4  [#when]first[#>][premier ]
5  [#when]second[#>][deuxième ]
6  [#when]third[#>][troisième ]
7  [#case}
```

Résultat:

List of the translated words in French:
premier troisième deuxième premier

L'équivalent en R (même si pas trop d'intérêt, ici) :

```
1  [#=]todo[first,third,second,first]
2  [#>]List of the translated words in French:
3  [#R>]
4      for(todo in strsplit("#{=todo}","")[[1]]) {
5          switch(todo,
6              first={#>}premier [#>},
7              second={#>}deuxième [#>},
8              third={#>}troisième [#>})
9      }
```

Résultat:

```
List of the translated words in French:
premier troisième deuxième premier
```

L'équivalent en ruby (même si pas trop d'intérêt, ici) :

```
1  [#=]todo[first,third,second,first]
2  [#>]List of the translated words in French:
3  [#rb>]#={todo}.split(",").each do |todo|
4      case todo.to_sym
5      when :first then [#>]premier [#>}
6      when :second then [#>]deuxième [#>}
7      when :third then [#>]troisième [#>}
8      end
9  end
```

Résultat:

```
List of the translated words in French:
premier troisième deuxième premier
```

Boucles directement en R :

```
1  [#>] Loop in R:
2  [#R>]
3  for(cpt in 1:4) {#>}item#r{cpt-1} [#>}
4  [#>]<\n>but also,
5  [#R>] sapply(5:8,function(cpt) {
6      cpt <- cpt -1
7      {#>}item#r{cpt} [#>}
8  })
```

Résultat:

```
Loop in R:
item0 item1 item2 item3
but also,
item4 item5 item6 item7
```

Boucles directement en ruby :

```
1  [#>] Loop in ruby:
2  [#rb>]
3  for cpt in 1..4 do
4      [#>] item:{cpt-1} [#>}
5  end
6  [#>] <\n>but also,
7  [#rb>] (5..8).each do |cpt|
8      cpt=cpt-1
9      [#>] item#rb{cpt} [#>}
10 end
```

Résultat:

```
Loop in ruby:
item0 item1 item2 item3
but also,
item4 item5 item6 item7
```

Plan

- 1 Motivation
- 2 Découverte de dyndoc par des exemples
- 3 Programmation : environnements R et ruby
- 4 Programmation : variables dyndoc
- 5 Programmation : condition, sélection et boucles
- 6 Exemples plus avancés**
- 7 dyndoc en résumé
- 8 Modes d'utilisation

Libraries d'objets dyndoc (ici classe Data) :

```
1  {#meth}new.Data[#,] .objR[] [#,].rdata[] [#<][#}  
2  
3  {#meth}begin.Data  
4    [#yield]default  
5    [#rb>].rdata[#{=.rdata}.strip]  
6    [#rb>].objR[#{=.objR}.strip]  
7    [#?]{+?.rdata}[#r<]attach('#{.rdata}')  
8    [#?]{+?.objR}[#r<]attach('#{.objR}')  
9  [#}  
10  
11 {#meth}end.Data  
12 [#?]{+?.objR}[#r<]detach('#{.objR}')  
13 [#?]{+?.rdata}[#r<]detach('file:#{.rdata}')[#?]end  
14 [#yield]default  
15 [#}
```

Déclaration d'objets et appels de méthodes :

```
1  [#require] Import/LM/lm[#main] [#<]
2  {#new} data[#of] Data[#in] lm1
3  [# ,] chomage[# ,] ~ /cqls/data/chomage.RData[#]
4  {#new} lm1[#of] LM[#in] lm1
5  [# ,] chom ~ txpib+deppub+pfisc[# ,]:data[#]
6  [#>] {#summary} lm1[#]
7  [#<] {#end} lm1[#]
```

Résultat:

Régression linéaire : $\text{chom} \sim \text{txpib} + \text{deppub} + \text{pfisc}$ (data=chomage)

$\mathbf{x}^{(j)}$	$\widehat{\beta}_j(\mathbf{y} \mathbf{x})$	$\widehat{\sigma}_{\widehat{\beta}_j}(\mathbf{y} \mathbf{x})$	$\widehat{\delta}_{\beta_j,0}(\mathbf{y} \mathbf{x})$	p -valeur bilatérale
1	-22.05345	1.96807	-11.20562	$\simeq 0$
txpib	0.01359	0.10423	0.13035	0.89716
deppub	0.27974	0.17793	1.57221	0.12639
pfisc	0.34835	0.20287	1.71711	0.09627

Rédaction abrégée d'un hypothèse de test :

```
1  [#require] Import/StatDevel/HypoTest[#main] [#<]
2  {#new} pChom[#of] ParamFrame[#in] pChom[#,] p[#,] asymp
3  {#.nomPb} C{#.thetaEstR} 16/200{#.ny} 200{#}
4
5  {#new} pChomTest[#of] TestParam[#in] pChom[#,]:pChom
6  {#.theta0R} 0.1{#.theta0Tex} 10\%{#.side}<
7  {#.assertion} le taux de chômage en France serait
8  cette année inférieur à $10\%$ {#}
9  {#>} {#redaction} pChomTest[#,] a{#}
```

Résultat:

Assertion d'intérêt : $H_1 : p^C < 10\%$

Application numérique : puisqu'au vu des données,

$p\text{-valeur} \stackrel{R}{=} \text{pnorm}((16/200 - 0.1)/\text{sqrt}(0.1 * (1 - 0.1)/200)) \simeq 34.58\% \not< 5\%$,

on ne peut pas plutôt penser (avec un risque de 5%) que le taux de chômage en France serait cette année inférieur à 10%.

Flag + environnement R + Paramètre nommé :

```
1  [#<] {#r}a<-rnorm(5) [#in]exo[#]
2  {#def]test[#,]opt[m][#,]l[5]
3  [#<]{#opt}#{opt}[#][#>]
4  {#rverb][#??]m[#>]mean(a)
5  [#??]s[#>]sd(a) [#in]exo[#rverb}
6  #r{exo:a[1:#{l}]}[#def}
7  [#>]{#test}m[#] {#test}s[#l]3[#]
```

Résultat:

```
> mean(a)
[1] 1.061335
0.3596606,2.154655,0.8119914,1.640756,0.3396102
> mean(a)
[1] 1.061335
> sd(a)
[1] 0.8072303
0.3596606,2.154655,0.8119914
```

Intégrer knitr (et Sweave)

```
1  [#R>] for(i in c(5,6)) {
2    {#>}{#knitr}
3    |«»=
4    |rnorm(:r{i})
5    |@
6    |[#knitr]{#>}
7
8    if(i==6) {#>}{#knitr}
9    |«fig,echo=FALSE,out.width='.3\\linewidth'»=
10   |plot(1::r{i},rnorm(:r{i}),type='l',lwd=5)
11   |@
12   |[#knitr]{#>}
13 }
```

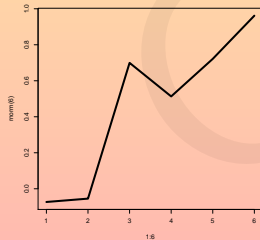
Résultat :

```
rnorm(5)
```

```
## [1] -0.25861  0.02974 -0.62782 -0.26184  1.32447
```

```
rnorm(6)
```

```
## [1] -0.04186  1.26791 -0.99090 -0.99889  1.04388 -0.41159
```



Plan

- 1 Motivation
- 2 Découverte de dyndoc par des exemples
- 3 Programmation : environnements R et ruby
- 4 Programmation : variables dyndoc
- 5 Programmation : condition, sélection et boucles
- 6 Exemples plus avancés
- 7 dyndoc en résumé**
- 8 Modes d'utilisation

En résumé, dyndoc c'est

Un outil développé en ruby avec pour principales caractéristiques :

- **système de modèles (“templating system”)** : basé sur un langage de balises (de type “domino”) suffisamment exotique pour s'intégrer facilement dans tout document au format “human readable” (tel que latex, html, ...) afin d'en dynamiser le contenu.
- **langage (de script)** : permettant d'intégrer pleinement les langages ruby, R et Mathematica (expérimental) et éventuellement python ou tout autre langage interfaçable en C.

Principaux avantages

- *mode développeur*: créer des bibliothèques regroupant fonctions et objets dyndoc (dédiés à la génération de parties textuelles relatives à certaines expertises)
- *mode utilisateur*: rassembler dans un même document dyndoc à la fois les dernières étapes des analyses statistiques (fait avec R , ruby et Mathematica) ainsi que le contenu textuel. A partir de ce document dyndoc, permettre de générer un ou plusieurs documents finaux dans différents formats (latex, html, ...).

Plan

- 1 Motivation
- 2 Découverte de dyndoc par des exemples
- 3 Programmation : environnements R et ruby
- 4 Programmation : variables dyndoc
- 5 Programmation : condition, sélection et boucles
- 6 Exemples plus avancés
- 7 dyndoc en résumé
- 8 Modes d'utilisation**

Modes d'utilisation de dyndoc

- **Ligne de commande en version autonome** : (sans hors-connexion)
`dyndoc-ruby [options] all [options] document.dyn`
- **Ligne de commande en version client-serveur** :
`dyndoc-client all dyn://sag6/document.dyn`
client (léger) se connectant à un serveur dyndoc pour traitement de petit projet (pas un livre sur R !)
- **Interface web** : (hors-connexion)
`dyndoc-web [start|stop|status]`
dédié à la création de documents html5 incluant du javascript pour dynamiser les traitements (en particulier, graphique).
- **Interface web** : (connexion server)
Zéro install, dédié à la création en ligne de pages web (ou documents pdf) avec possibilité de proposer directement des traitements R .

Informations complémentaires

Page web dyndoc

<http://dyndoc.upmf-grenoble.fr/Dyndoc.html>

Esprit du projet :

- *sur un plan général* : dyndoc est né d'une quête de sens, d'une mise en veille de l'égo, d'un éveil de conscience, de l'envie de partager des valeurs
- *sur un plan plus pratique* : dyndoc c'est une tentative de participer au développement d'une activité (ici, autour du reporting statistique) dans un but de regrouper une communauté (ici, de développeurs, d'utilisateurs) où la dimension sociale prévaudrait.