

Introdução ao L^AT_EX

Bruna Gabriela Wendpap
Djair Durand Ramalho Frade

Fernando de Pol Mayer

Luiz Ricardo Nakamura

Maria Cristina Martins

Thiago de Paula Oliveira

Thiago Gentil Ramires

Profa. responsável: Dra. Roseli Aparecida Leandro

Universidade de São Paulo (USP)

Escola Superior de Agricultura "Luiz de Queiroz" (ESALQ)

04 de Outubro, 2013



Sumário

- 1 Introdução
- 2 Configuração
- 3 Opções
- 4 Tabelas
- 5 Figuras
- 6 *Tangling*

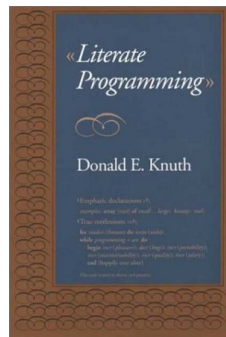


Plano de aula

- 1 Introdução
- 2 Configuração
- 3 Opções
- 4 Tabelas
- 5 Figuras
- 6 *Tangling*



Introdução



Introdução

- O conceito de *Literate Programming* foi introduzida por Knuth em 1984
- Representa uma mudança de paradigma na computação
 - Deixar de escrever programas na ordem imposta pelo computador
 - Passar a escrever programas na ordem e lógica dos pensamentos
- O conceito é o de misturar *literatura* (o texto em uma linguagem humana) com códigos de programação, tornando claro cada etapa de um programa
- Obviamente, Knuth criou um sistema chamado **WEB** para fazer essa mistura dos seus textos em T_EX com a linguagem Pascal
- Atualmente muitos outros sistemas existem para misturar códigos com texto em várias linguagens



Introdução

Na Estatística, com a ascensão do **R** no início dos anos 2000, Friedrich Leisch criou o **Sweave** em 2002

- **S + weave**
- Permite “entrelaçar” textos do \LaTeX com códigos do **R**
- É o *engine* padrão para gerar a documentação de todas as funções do **R**
- Ainda é muito utilizado e já é distribuído como uma função do **R** dentro do pacote **utils**
- No entanto, com o passar do tempo, as necessidades dos programadores e a falta de flexibilidade do Sweave fizeram com que diversos pacotes auxiliares tivessem que ser criados



Introdução

No final de 2011, Yihui Xie criou o pacote **knitr** com a proposta de ser mais flexível, fácil e preparado para a Web

*knitr = Sweave + cacheSweave + pgfSweave + weaver +
animation::saveLatex + R2HTML::RweaveHTML +
highlight::HighlightWeaveLatex + 0.2 * brew + 0.1 *
SweaveListingUtils + more*



Introdução

O knitr

- Uma re-implementação mais moderna do Sweave
- Adiciona muitas facilidades como
 - Cache
 - Decoração e formatação automática de códigos
 - Geração de gráficos mais direta
- Extremamente extensível e customizável
- Suporta a geração de documentos para a Web
 - Markdown
 - HTML
- Já é possível usar o **knitr** como *engine* de documentação do R



Introdução

O knitr

- Página com diversas informações <http://yihui.name/knitr>
- Acompanhe o desenvolvimento pelo GitHub
<https://github.com/yihui/knitr>
- A sintaxe do **knitr** é parecida, mas não é a mesma que a do **Sweave**. Se você era um usuário do Sweave antes, leia <http://yihui.name/knitr/demo/sweave>



Plano de aula

- 1 Introdução
- 2 Configuração
- 3 Opções
- 4 Tabelas
- 5 Figuras
- 6 *Tangling*



Configuração

- O **knitr** pode ser utilizado em qualquer editor de texto, mas alguns facilitadores são
 - **Emacs** com ESS
 - **LyX**
 - **RStudio**
- A primeira coisa a fazer no **R** é instalar os pacotes necessário:

```
install.packages(c("knitr", "xtable"), dependencies = TRUE)
```



Configuração

A ideia é fazer a seguinte sequência:

- 1 Criar um arquivo com a extensão `.Rnw`
- 2 Inserir o preâmbulo tradicional do \LaTeX , texto e código
- 3 Compilar o arquivo com a função `knit()` → vai gerar um arquivo `.tex`
- 4 Compilar o arquivo `.tex` no \TeX Maker (ou outros) → gera o arquivo `.pdf`



Configuração

Expressões do **R** são inseridas normalmente dentro de um ambiente especial no arquivo `.Rnw`:

```
<<>>=
```

```
...
```

```
@
```

- Toda expressão do **R** que estiver dentro deste **chunk** será interpretada quando compilada pelo **knitr**, gerando a saída, gráficos, etc.
- Para inserir resultados no meio do texto (*inline*) use `\Sexpr{}`



Configuração

Um exemplo mínimo:

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[brazil]{babel}
\usepackage[margin=2.5cm]{geometry}
```

```
\begin{document}
```

Definindo a variável aleatória X com distribuição Normal padrão, ou seja, $X \sim \text{N}(0,1)$

```
<<>>=
```

```
set.seed(1)
```

```
(x <- rnorm(10))
```

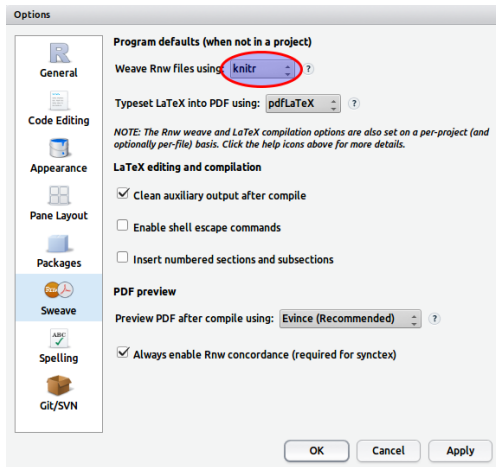
```
@
```

A média desta variável aleatória é $\% \text{\%Sexpr{mean(x)}} \%$. O primeiro valor é $X_i = \% \text{\%Sexpr{x[1]}} \%$.

```
\end{document}
```

Configuração

No **RStudio**, altere as opções para deixar o **knitr** como padrão



Configuração

Com isso, o botão **Compile PDF**:

- Compila o arquivo `.Rnw` com o **knitr**
- Compila o arquivo `.tex` resultante com o **PDF \LaTeX**
- Abre o PDF no editor padrão

Tudo isso é feito em um ambiente separado da sua área de trabalho original, para não haver confusão com objetos gerados



Plano de aula

- 1 Introdução
- 2 Configuração
- 3 **Opções**
- 4 Tabelas
- 5 Figuras
- 6 *Tangling*



Opções

- **Opções específicas:** são opções definidas para cada *chunk* de código, especificadas entre `<<` e `>>=`
- **Opções globais:** são opções definidas para todos os *chunks* do documento, especificadas pela função do **knitr**

```
opts_chunk$set()
```

Ainda assim, é possível especificar opções para *chunks* individuais!



Opções

A lista completa de opções está em <http://yihui.name/knitr/options>

Algumas opções gerais (* indica o padrão):

- `eval`: TRUE*, FALSE, ou `c(1,3,4)` ou `-(4:6)`. Se o código deve ser interpretado ou não. Podem ser especificadas apenas linhas.
- `echo`: TRUE*, FALSE, ou `c(1,3,4)` ou `-(4:6)`. Se deve mostrar ou não o código ou pedaços dele. O resultado sempre aparece. (Diferente de `eval`, porque sempre será interpretado).
- `include`: TRUE*, FALSE. Se deve incluir ou não o código no documento final. Útil por exemplo para fazer cálculos auxiliares que não precisam ser mostrados no documento.



Opções

Algumas considerações sobre as opções:

- Cada chunk pode ter um nome, por exemplo

```
<<bloco1>>=
```

@

que será útil posteriormente para dar nomes à figuras e extração de código.

- Evite usar espaços e pontos nos nomes dos chunks (use - e _).
- As opções em <<>>= devem estar sempre na **mesma linha**
- **Todas** as opções devem ser expressões válidas do R
 - Caracteres entre aspas
 - Valores lógicos: TRUE ou FALSE



Opções

Exercício

- Insira mais 3 chunks no seu arquivo, cada um com pelo menos 3 linhas de código do **R**
- Teste as opções `eval`, `echo`, e `include` em cada uma delas



Opções

Opções para decoração de código:

- `prompt`: TRUE ou FALSE*. Se deve ou não inserir o *prompt* (>) do R no início de cada linha.
- `comment`: "##"* ou NA. Se deve ou não comentar a saída dos comandos
- `highlight`: TRUE* ou FALSE. Se deve colorir os códigos.
- `size`: "normalsize"* ou qualquer tamanho de fonte do \LaTeX ("small", "footnotesize", ...)
- `tidy`: TRUE* ou FALSE. Se o código deve ser formatado para um padrão geral pré-definido.



Opções

Exercício

- Altere estas opções em alguns chunks e veja o resultado
- Em especial escreva esse código dentro de dois chunks separados

```
rmnorm(10,  
        10, 5)
```

cada um com `tidy=TRUE` e `tidy=FALSE`.



Opções globais

Para definir um padrão para todos os chunks globalmente, podemos especificar logo no início do documento as opções que queremos. Por exemplo:

```
<<setup, include=FALSE, cache=FALSE, tidy=FALSE>>=  
# Minhas configurações globais para os chunks  
opts_chunk$set(size = "small",  
                prompt = FALSE,  
                comment = NA,  
                tidy = FALSE,  
                cache = TRUE)
```

@



Opções

O cache (TRUE*/FALSE):

- Armazena os resultados dos chunks em disco
 - Cria um diretório cache
- Esse resultado será reaproveitado nas próximas compilações, portanto um chunk em cache não será interpretado novamente
- Muito útil para documentos muito longos ou com comandos demorados ou bases de dados grandes



Plano de aula

- 1 Introdução
- 2 Configuração
- 3 Opções
- 4 Tabelas**
- 5 Figuras
- 6 *Tangling*



Tabelas

Tabelas geradas pelo **R** podem ser incluídas no documento \LaTeX com o pacote `xtable`

```
<<results="asis", echo=FALSE>>=
```

```
## Carrega o pacote
require(xtable, quietly = TRUE)
## Tira uma amostra de 10 linhas da base de dados Iris
am <- sample(1:nrow(iris), size = 10)
iris.am <- iris[am, ]
## Gera a tabela com código do LaTeX
xtable(iris.am)
```

@



Tabelas

As opções usadas são:

- `results="asis"`: para o resultado ser a saída pura do **R** (ao invés de tentar decorar). Outras opções são: `markup*`, `hold` e `hide`
- `echo=FALSE`: para que o código em \LaTeX que gera a tabela não seja mostrado. Outra opção seria: `echo=c(1:5)` para mostrar o código até a geração da tabela.

Tente com `echo=TRUE`



Tabelas

Inserindo legendas e referências pelo xtable

```
xtable(iris.am,  
       caption = "Uma legenda para a tabela",  
       label = "tab:iris")
```

No texto você pode referenciar como `\ref{tab:iris}`



Tabelas

Para alterar a posição da legenda, precisamos do método `print()` para o `xtable`

```
tab <- xtable(iris.am,  
              caption = "Uma legenda para a tabela",  
              label = "tab:iris2")  
print(tab, caption.placement = "top")
```

Muitas outras opções de formatação estão disponíveis nestas duas funções.
Veja

`?xtable`

`?print.xtable`



Tabelas

Tente com a saída de um modelo

```
mod <- lm(Petal.Length ~ Petal.Width, iris)
xtable(summary(mod))
```

; -)



Plano de aula

- 1 Introdução
- 2 Configuração
- 3 Opções
- 4 Tabelas
- 5 Figuras**
- 6 *Tangling*



Figuras

As figuras geradas pelo **R** são incluídas automaticamente no documento final.

```
<<fig1>=>
```

```
plot(iris)
```

@

Um diretório `figure` é criado automaticamente (veja!) para armazenar as figuras. Por isso aqui é importante **nomear** o chunk.



Figuras

Por padrão, a figura ocupa a largura da página. Podemos alterar o tamanho com as opções `out.width` e `out.height` como no `\includegraphics` do \LaTeX . Por exemplo:

```
<<fig1, out.width=".5\\linewidth">=>
```

```
plot(iris)
```

@



Figuras

Para alinhar a figura usamos a opção `fig.align`. Por exemplo:

```
<<fig1, out.width=".5\\linewidth", fig.align="center">=
```

```
plot(iris)
```

@



Figuras

Para adicionar uma legenda escrevemos na opção `fig.cap`. O alinhamento deve ser feito em `fig.pos`, como na opção do ambiente `figure` do \LaTeX :

```
\begin{figura}[fig.pos]
```

```
<<fig1, ..., fig.cap="Legenda da figura", fig.pos="!htb">=>
```

```
plot(iris)
```

@

Para referenciar a figura no texto, use `\ref{fig:<nome do chunk>}`.
Nesse caso `\ref{fig:fig1}`



Figuras

Para duas figuras:

```
<<fig2, out.width=".5\\linewidth", fig.align="center">=
```

```
plot(Petal.Length ~ Petal.Width, iris)  
plot(Sepal.Length ~ Petal.Length, iris)
```

@

Serão plotadas separadamente



Figuras

Para que elas fiquem lado a lado use a opção `fig.show`:

```
<<fig2, out.width=".45\\linewidth", ..., fig.show="hold">=>
```

```
plot(Petal.Length ~ Petal.Width, iris)  
plot(Sepal.Length ~ Petal.Length, iris)
```

@

Repare que `out.width=".45\\linewidth"` para que cada uma ocupe 45% da largura da linha.



Plano de aula

- 1 Introdução
- 2 Configuração
- 3 Opções
- 4 Tabelas
- 5 Figuras
- 6 *Tangling*



Tangling

Tangling é o processo inverso de *weaving*

- Serve para extrair apenas o código que está “emaranhado” no meio do texto do \LaTeX

No **R** use a função `purL()` do **knitr**:

```
## Extrai apenas o código e os comentários  
purL("knitr-template.Rnw", documentation = 0L)  
## Traz também as opções dos chunks (nome, ...)  
purL("knitr-template.Rnw", documentation = 1L)
```

