

Rapports dynamiques avec Shiny



Rapports dynamiques avec Shiny

Vincent Goulet

Professeur titulaire

École d'actuariat, Université Laval

Version 2020.03



Vincent Goulet, 2020

© 2020 par Vincent Goulet. « Rapports dynamiques avec Shiny » est mis à disposition sous licence **Attribution-Partage dans les mêmes conditions 4.0 International** de Creative Commons. En vertu de cette licence, vous êtes autorisé à :

- **partager** — copier, distribuer et communiquer le matériel par tous moyens et sous tous formats;
- **adapter** — remixer, transformer et créer à partir du matériel pour toute utilisation, y compris commerciale.

L'Offrant ne peut retirer les autorisations concédées par la licence tant que vous appliquez les termes de cette licence.

Selon les conditions suivantes :



Attribution — Vous devez créditer l'œuvre, intégrer un lien vers la licence et indiquer si des modifications ont été effectuées à l'œuvre. Vous devez indiquer ces informations par tous les moyens raisonnables, sans toutefois suggérer que l'Offrant vous soutient ou soutient la façon dont vous avez utilisé son œuvre.



Partage dans les mêmes conditions — Dans le cas où vous effectuez un remix, que vous transformez, ou créez à partir du matériel composant l'œuvre originale, vous devez diffuser l'œuvre modifiée dans les mêmes conditions, c'est-à-dire avec la même licence avec laquelle l'œuvre originale a été diffusée.

Code source

 [Voir sur GitLab](#)

Couverture

Tête d'un sarcoramphé roi (*Sarcoramphus papa*), photographié dans le Parque del Este, à Caracas, au Venezuela.

Crédit photo : © Paolo Costa Baldi **CC BY-SA 3.0 non transposé**, via [Wikimedia Commons](#).

Vous souhaitez accompagner votre étude d'une narration ?

Vous trouverez un lien vers une vidéo dans chacune
des cinq principales sections de la formation.

Présentation générale

Notre projet d'application

Programmation Shiny 101

Construction d'une application

- Entrée et sortie


- Recalcul des valeurs




- Texte statique et mathématiques

Dernières remarques

Présentation générale

 Vidéo 1 de 5 — Présentation générale

Shiny  est une technologie de RStudio pour créer des applications web interactives (*apps*) directement dans R.

- Basé sur le packaging **shiny**
- Exemple simple : **Analyse de régression** 
- Exemple élaboré : **SuperZip** 
- Plusieurs autres exemples dans la **galerie Shiny** 

Préalables pour exécuter des applications

- Installer le paquetage **shiny** (si ce n'est pas déjà fait)

```
> install.packages("shiny")
```

- Charger le paquetage dans la session R

```
> library("shiny")
```

Le paquetage **shiny** est livré avec quelques exemples d'applications que vous pouvez exécuter localement.

1. Exécuter l'exemple `01_hello` et analyser le code de l'application

```
> runExample("01_hello")
```

2. Interrompre l'application `01_hello`
3. Exécuter l'exemple `04_mpg` et analyser le code de l'application

```
> runExample("04_mpg")
```

4. Interrompre l'application `04_mpg`

Notre projet d'application

 Vidéo 2 de 5 — Notre projet d'application

Calcul interactif de l'*Expected Shortfall*

La mesure de risque *Expected Shortfall* est équivalente à la prime stop-loss évaluée à la *Value at Risk* :

$$\begin{aligned} \text{ES}_\alpha(S) &= E[(S - \text{VaR}_\alpha(S))_+] \\ &= \int_{\text{VaR}_\alpha(S)}^{\infty} (x - \text{VaR}_\alpha(S)) f_S(x) dx \\ &= \int_{\text{VaR}_\alpha(S)}^{\infty} (1 - F_S(x)) dx. \end{aligned}$$

Nous allons :

1. simuler un échantillon d'une distribution Poisson composée
2. calculer empiriquement VaR_α et ES_α
3. représenter la région sur la fonction de répartition empirique

Exécuter l'application `shortfall` livrée avec le matériel pédagogique.

- Extraire les fichiers de `laboratoire-shiny.zip` dans un répertoire quelconque
- Faire de ce répertoire le répertoire de travail de R
- Démarrer l'application

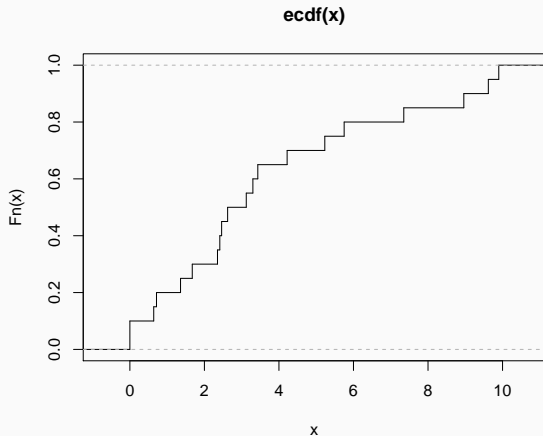
```
> runApp("shortfall")
```

Création du graphique (sommaire)

1. Fonction `ecdf` retourne une fonction pour évaluer la fonction de répartition empirique d'un échantillon
2. Méthode de `plot` pour tracer la fonction
3. Déterminer les points d'un polygone à remplir
4. Remplir le polygone avec `polygon`

Création du graphique (étape 1)

```
> Fn <- ecdf(x)  
> plot(Fn, verticals = TRUE, do.points = FALSE)
```



Création du graphique (étape 2)

- Calcul de la VaR

```
> VaR <- quantile(x, 0.9, type = 1)
```

- Nœuds de la fonction de répartition

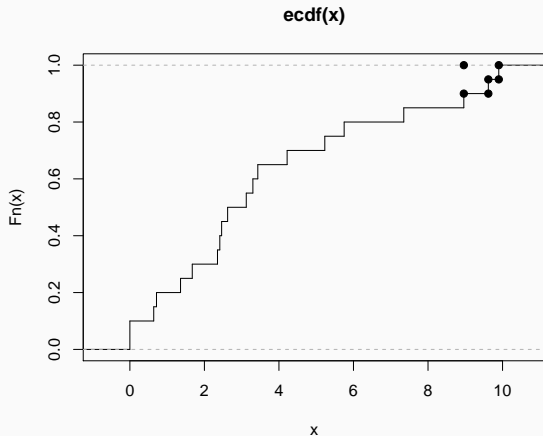
```
> k <- knots(Fn)
```

- Sommets du polygone

```
> xx <- rep(c(VaR, k[k > VaR]), each = 2)
```

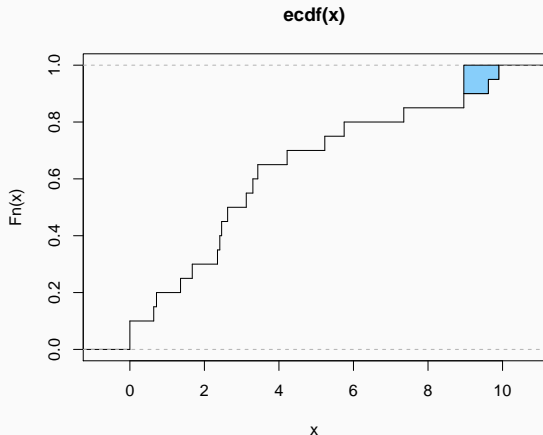
Création du graphique (étape 2.5)

```
> plot(Fn, verticals = TRUE, do.points = FALSE)  
> points(xx, c(1, Fn(head(xx, -1))), pch = 19)
```



Création du graphique (étape 3)

```
> plot(Fn, verticals = TRUE, do.points = FALSE)  
> polygon(xx, c(1, Fn(head(xx, -1))), col = "lightskyblue")
```



Programming Shiny 101

 Vidéo 3 de 5 — Programmation Shiny 101

Structure de fichiers d'une application

Shiny exige une structure de fichiers bien précise pour les applications.

- Pour une application nommée `foobar`

```
mon projet/  
└─ foobar/  
   └─ app.R
```

- Micro-exercice : identifiez l'application `shortfall` dans le présent matériel pédagogique

Structure du code d'une application

```
> runExample("01_hello")
```

Structure du code d'une application

```
> runExample("01_hello")
```

Le fichier `app.R` contient au minimum trois éléments.

1. Création d'un objet `ui` avec la fonction `fluidPage` pour décrire l'interface de l'application
 - crée des éléments d'un objet `input` (implicite)
 - utilise éléments d'un objet `output` (implicite)
2. Création d'une fonction `server` pour effectuer le traitement
 - arguments `input` et `output`
 - utilise éléments de l'objet `input`
 - place contenu dans objet `output`
3. Appel à la fonction `shinyApp` pour construire l'application
 - arguments `ui` et `server`

Construction d'une application

 Vidéo 4 de 5 — Construction d'une application

Construction d'une application

Entrée et sortie

Contrôles d'entrée de paramètres (*widgets*)

http://127.0.0.1:3771 | Open in Browser | Publish

Basic widgets

Buttons

Action

Submit

Single checkbox

☒ Choice A

Checkbox group

☒ Choice 1
☐ Choice 2
☐ Choice 3

Date input

2014-01-01

Date range

2017-06-21 to 2017-06-21

File input

Browse... No file selected

Help text

Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

Numeric input

1

Radio buttons

☒ Choice 1
☐ Choice 2
☐ Choice 3

Select box

Choice 1

Sliders

0 50 100

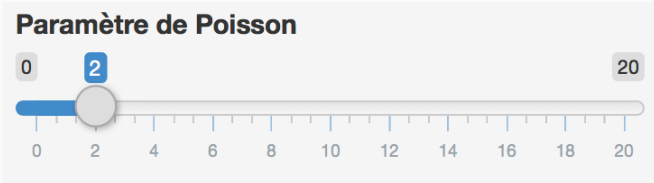
0 25 75 100

Text input

Enter text...

Liste complète des fonctions [↗](#)

Exemple de syntaxe



```
sliderInput(inputId = "lambda",  
            label = "Paramètre de Poisson",  
            value = 2,  
            min = 0,  
            max = 20,  
            step = 0.1)
```

Fonctions d'affichage dans l'interface

Nous utiliserons deux fonctions d'affichage de contenu dynamique.

1. `textOutput` : texte
2. `plotOutput` : graphique

Liste complète des fonctions [↗](#)

Valeur du paramètre de Poisson: 2

```
textOutput(outputId = "valeur_lambda")
```

Fonctions de traitement dynamique (*rendering*)

Ces fonctions sont utilisées dans la fonction `server` pour ajouter des éléments à l'objet `output`.

- `renderPlot` : graphiques
- `renderText` : chaînes de caractères

Liste complète des fonctions [↗](#)

Exemple de syntaxe

Les fonctions **render*** prennent en argument une **expression** R.

```
output$valeur_lambda <- renderText({  
  paste("Valeur du paramètre:", input$lambda)  
})
```

Compléter le code de l'application `exercice-widgets` pour obtenir le résultat ci-dessous.

Exercice sur les contrôles d'entrée



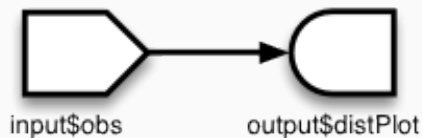
Valeur du paramètre de Poisson: 2

Construction d'une application

Recalcul des valeurs

Sources et destinations réactives (*reactive*)

Modèle de base que nous avons utilisé précédemment.



- **Source** réactive via l'objet `input`
- **Destination** réactive via l'objet `output`

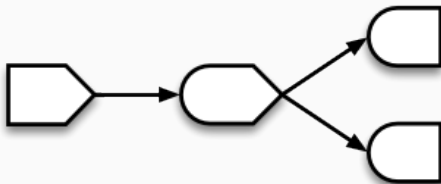
Éviter les calculs inutiles

Notre application requiert de simuler un échantillon aléatoire.

- À recommencer lorsque les paramètres de la simulation changent
 - modèle de simulation
 - nombre de simulations
- Ne **pas** recommencer lorsque les paramètres de la simulation de changent pas
 - niveau de la VaR
 - paramètres du graphique (pas mis en œuvre)
- Résultats de la **même simulation** pour le graphique et pour les résultats numériques
- Besoin de quelque chose **entre** la source et la destination réactive

Expression réactive

Une **expression réactive** est à la fois une destination et une source réactive.



- Effectue un calcul lorsqu'une source change
- Fournit son résultat à répétition à des destinations
- Résultat gardé en mémoire

Exemple de syntaxe

Fonction **reactive** crée une expression réactive

```
data <- reactive({  
  rcomppois(input$n, input$lambda, rgamma(2, 0.002))  
})
```

Appel de fonction pour obtenir la valeur de l'expression

```
output$VaR <- renderText({  
  paste("Valeur de la VaR:",  
        quantile(data(), as.numeric(input$alpha), type = 1))  
})
```



Si vous devez utiliser les mêmes résultats plus d'une fois dans `server`, vous avez besoin d'une expression réactive.



L'utilisation d'expressions réactives permet de mieux segmenter le code et d'éviter les répétitions.



Un appel à une source ou à une expression réactive ne peut se trouver que dans une fonction `render*` ou dans la fonction `reactive`.

Compléter le code de l'application `exercice-reactive`.

Exercice sur le recalcul des valeurs

Paramètre de Poisson

0 2 20

0 2 4 6 8 10 12 14 16 18 20

Taille de l'échantillon

1000

Niveau de la VaR

☒ 0.9

☐ 0.95

☐ 0.99

Moyenne de l'échantillon: 2003

Valeur de la VaR: 4385

- Moyenne **et** VaR changent quand les paramètres de simulation changent
- Seule VaR change quand le niveau de la VaR change

Construction d'une application

Texte statique et mathématiques

- Fonction `helpText` sert pour afficher du texte sur la page.
 - entête de la page
 - dans `sidebarPanel`
 - dans `mainPanel`

```
helpText("Ce texte n'est pas dynamique")
```

- Fonctions `h1`, `h2`, ..., `h6` pour les niveaux de titre

- Charger MathJax dans `fluidPage`

```
withMathJax(),
```

- Entrer les équations dans le texte de `helpText` avec la syntaxe \LaTeX
- **Attention** Doubler les `\` à l'intérieur des équations

```
$$y = \alpha x + \beta$$
```

- **Attention** Utiliser `\(\)` pour délimiter les équations au fil du texte


```
Nous avons \((y = \alpha x + \beta)\).
```


Étudier le code complet de l'application `shortfall`.

Dernières remarques

 Vidéo 5 de 5 — Dernières remarques

Publier ses applications Shiny

Vous pouvez publier vos applications Shiny de **diverses manières** .

- Envoyer le fichier **app.R** à une autre personne, qui l'exécute **localement** avec **runApp**
 - *some assembly required*
- Publier l'application sur un site quelconque ou dans GitHub et indiquer l'url à une autre personne, qui l'exécute **localement** avec **runUrl** ou **runGitHub**
 - *some assembly required*
- Publier l'application dans **shinyapps.io**  et fournir un lien vers la page web
 - publication très simple à partir de RStudio
 - restrictions pour l'hébergement gratuit

Il est possible de définir une application Shiny à l'intérieur d'un document R Markdown. C'est même très simple.

Consulter la [documentation](#) .

Ce document a été produit par le système de mise en page \LaTeX avec la classe **beamer** et le thème Metropolis. Les titres et le texte sont composés en Fira Sans, les mathématiques en Arev Math et le code informatique en Fira Mono. Les icônes proviennent de la police Font Awesome. Les graphiques ont été réalisés avec R.

