ACT-2001 Introduction à l'actuariat 2

Atelier de la semaine du 12 janvier 2026 (atelier 0)

Étienne Marceau
avec Anthony Gélinas et Philippe Leblanc (H2026)
et Jérémie Barde, Ève Busque et Alexandre Dubeau (semestres précédents)

École d'actuariat
Université Laval, Québec, Canada

2026-01-14

Table des matières I

Partie 1 - Bases

Logiciel R

Lois continues paramétriques et logiciel R :

- Loi uniforme : `dunif()`, `punif()`, `qunif()`, `runif()` ;
- Loi exponentielle : `dexp()`, `pexp()`, `qexp()`, `rexp()` ;
- Loi gamma : `dgamma()`, `pgamma()`, `qgamma()`, `rgamma()` ;
- Loi lognormale : `dlnorm()`, `plnorm()`, `qlnorm()`, `rlnorm()`.
- Loi normale : `dnorm()`, `pnorm()`, `qnorm()`, `rnorm()`.

Lois discrètes paramétriques et logiciel R :

- Loi Poisson : `dpois()`, `ppois()`, `qpois()`, `rpois()` ;
- Loi binomiale négative : `dnbinom()`, `pnbinom()`, `qnbinom()`, `rnbinom()`.
- Loi binomiale : `dbinom()`, `pbinom()`, `qbinom()`, `rbinom()`.

Logiciel R

- Minimum et maximum parallèle : `pmin()`, `pmax()`
- Outils d'optimisation : `optimize()`, `uniroot()`
- Fonctions d'application : `sapply()`, `apply()`
- Création d'objets : `seq()`, `numeric()`, `matrix()`

Lien GitHub

Vous pouvez accéder au répertoire GitHub du cours ici :
https://github.com/pleblanc0/act2001-h26

Partie 2 - Raisonnement computationnel

Raisonnement Computationnel

Il n'y a pas de définition explicite pour le raisonnement computationnel (ou pensée computationnelle).

On parle ici du raisonnement utilisé pour résoudre un problème à l'aide d'un ordinateur.

Le raisonnement computationnel est de plus en plus important des les domaines de sciences, technologie et mathématiques, et forme un mode de pensée important pour la **résolution de problèmes** quelconques.

« Raisonnement computationnel » est la traduction française de « Computationnal Thinking ». L'expression « Pensée computationnelle » est aussi parfois utilisée.

Raisonnement Computationnel et Mathématique

Les auteurs de [Broley et al., 2024] ont remarqué un lien très fort entre le raisonnement mathématique et le raisonnement computationnel.

La façon de penser en programmation utilise souvent des notions mathématiques et vice-versa.

C'est pourquoi comprendre et utiliser le raisonnement computationnel peut, en plus d'améliorer son niveau en programmation, acquérir des outils supplémentaires et un raisonnement utile pour la résolution de problèmes mathématiques.

Pour des références supplémentaires, voir [Weintrop et al., 2016] et [Barana et al., 2025].

Raisonnement Computationnel - Compétences

Les auteurs de [Weintrop et al., 2016] ont identifié 10 compétences, importantes dans n'importe quel domaine, reliées au raisonnement computationnel :

- L'aptitude de traiter des problèmes ouverts ;
- La persévérance en travaillant sur des problèmes difficiles ;
- La confiance à travailler avec de la complexité ;
- La représentation d'idées de façon computationnellement significative ;
- La division de gros problèmes en plus petits problèmes ;
- La création d'abstractions pour des aspects de problèmes ;
- La reformulation d'un problème en un problème connu ;
- L'évaluation de forces et faiblesses d'une représentation ;
- La génération de solutions algorithmiques ;
- La reconnaissance et le traitement d'ambiguïtés dans des algorithmes.

Exemples

Exemples de Raisonnement Computationnel

On illustre le raisonnement computationnel en 4 exemples de différents niveaux de complexité.

Les illustrations seront effectuées avec R.

Note : les solutions aux problèmes présentées ne sont pas les meilleures solutions et c'est volontaire, seul le raisonnement est important.

Exemple 1 - Base

Commençons par un exemple de base. On veut compter le nombre de fois que la lettre *a* se trouve dans le mot *actuariat*. Il est évident que la réponse est 3, mais comment peut-on utiliser la programmation pour déterminer la réponse ?

Commençons par diviser le problème en 3 étapes :

1. Déterminer les lettres du mot *actuariat*.
2. Déterminer si chaque lettre est un *a*.
3. Compter le nombre de *a*.

Exemple 1 - Base

Grâce au cours IFT-4902, vous connaissez la fonction `strsplit` de `R` qui vous permet de séparer les caractères d'une chaîne. On peut résoudre la première étape du problème :

```
> strsplit("actuariat", "")
[[1]]
[1] "a" "c" "t" "u" "a" "r" "i" "a" "t"
```

Exemple 1 - Base

On souhaite maintenant utiliser l'opérateur == pour résoudre la deuxième étape de notre problème :

```
> strsplit("actuariat", "") == "a"
[1] FALSE
```

On remarque que l'on obtient pas le résultat désiré. On devrait avoir 9 valeurs booléennes et non une seule. En observant attentivement la sortie précédente, on remarque que la fonction `strsplit` retourne une liste d'un élément.

Exemple 1 - Base

On peut alors indicer pour obtenir le résultat désiré.

```
> strsplit("actuariat", "")[[1]] == "a"
[1]  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE
```

Exemple 1 - Base

On utilise finalement la conversion forcée de R à notre avantage pour compter le nombre de valeurs TRUE retournées pour résoudre notre problème :

```
> sum(strsplit("actuariat", "")[[1]] == "a")
[1] 3
```

Exemple 1 - Base

L'avantage de cette méthode est qu'elle nous permet d'adapter facilement notre solution à une fonction générale pour compter le nombre de lettre <letter> dans un mot <word> :

```
> count <- function(letter, word) sum(strsplit(word, "")[[1]] == letter)
> count("a", "actuariat")
[1] 3
> count("i", "introduction")
[1] 2
```

Exemple 2 - Prime stop-loss

Pour ce deuxième exemple, on souhaite construire une fonction `R` qui calcule la prime stop-loss donnée par

$$\pi_N(d) = E[\max(N - d; 0)],$$

pour n'importe quelle valeur d et n'importe quelle fonction de masse de probabilité discrète $f_N(k)$ sur un support fini A.

Exemple 2 - Prime stop-loss

À première vue, ce problème peut sembler difficile à résoudre numériquement. Cependant, grâce à la définition de l'espérance, on peut reformuler notre problème de la façon suivante :

Coder la fonction g telle que

$$g(d; A, f_N(k)) = \sum_{k \in A} \max(k - d; 0) f_N(k).$$

Ce problème est beaucoup plus simple à résoudre numériquement. Il suffit d'appliquer quelques fonctions apprises en IFT-4902 et l'arithmétique vectorielle de `R`.

Exemple 2 - Prime stop-loss

On obtient la fonction suivante :

```
g <- function(d, A, f) sum(pmax(A - d, 0) * f)
```

Ici, la fonction `pmax` calcule le maximum entre $k - d$ et 0 pour chaque élément $k \in A$, donc la partie $\max(k - d; 0)$ de notre équation.

On multiplie ensuite ce vecteur avec le vecteur `f` contenant la fonction de masse de probabilités $f_N(k)$, calculant chaque élément de la somme séparément.

Finalement, on effectue la somme de ces éléments à l'aide de la fonction `sum`, retournant la valeur désirée.

Exemple 2 - Prime stop-loss

Vérifions maintenant notre fonction. On remarque que lorsque $d = 0$, le calcul revient à l'espérance de N (lorsque tous les éléments du support sont positifs). On utilise $N \sim Bin(3, 0.5)$ pour vérifier qu'on obtient bien 1.5 :

```
> f <- dbinom(A, 3, 0.5)
> g(0, A, f)
[1] 1.5
```

On peut aussi vérifier notre réponse à l'aide de l'annexe. On a que $\pi_N(3) = 0.2180$ pour $N \sim Pois(2)$:

```
> f <- dpois(A, 2)
> g(3, A, f)
[1] 0.2180175
```

Les vérifications sont réussies !

Exemple 3 - Fonction récursive

Programmer la fonction vectorielle f suivante, définie récursivement.

$$f(\boldsymbol{x}; p) = u_1,$$

où $\boldsymbol{x}$ est un vecteur de dimension d où $\boldsymbol{x} \in [-1, 1]^d$, $p \in (0, 1)$ est un paramètre et

$$\begin{aligned} u_k &= (p + (1-p)x_k)\tfrac{1}{2}u_{k+1} + \tfrac{1}{2}v_{k+1} \\ v_k &= (px_{k-1} + (1-p)x_k)\tfrac{1}{2}u_{k+1} + (1 - p + px_{k-1})\tfrac{1}{2}v_{k+1} \end{aligned}$$

avec les valeurs de départ $u_{d+1} = 1$ et $v_{d+1} = 1 - p + px_d$.

Ceci est un exemple réel simplifié d'un contexte actuariel.

Exemple 3 - Fonction récursive

Encore une fois, ce problème semble complexe à première vue. Tentons de le décomposer.

On remarque par la relation récursive que nous devrons calculer les éléments des vecteurs $\boldsymbol{u}$ et $\boldsymbol{v}$, tous les deux de dimension $d+1$, en commençant par l'élément $d+1$ et en finissant par 1.

Réécrivons notre problème sous la forme d'un algorithme.

Exemple 3 - Fonction récursive

Algorithme 1: Calcul de $f(\boldsymbol{x}; p)$.

Entrée: Vecteur de d dimensions x et paramètre p.
Sortie: Valeur calculée de $f(\boldsymbol{x}; p)$.

1 $d \leftarrow$ Longueur de x;
2 $u, v \leftarrow$ Vecteurs de $d+1$ éléments;
3 $u[d+1] \leftarrow 1$;
4 $v[d+1] \leftarrow 1 - p + p \times x[d]$;
5 **pour** k *allant de* d *à* 1 **faire**
6 $\quad u[k] \leftarrow (p + (1-p)x[k])\frac{1}{2}u[k+1] + \frac{1}{2}v[k+1]$;
7 $\quad v[k] \leftarrow (px[k-1] + (1-p)x[k])\frac{1}{2}u[k+1] + (1 - p + px[k-1])\frac{1}{2}v[k+1]$;
8 **Retourne** $u[1]$

Exemple 3 - Fonction récursive

Nous sommes maintenant prêts à appliquer notre algorithme en R :

```
f <- function(x, p)
{
    d <- length(x)
    u <- numeric(d + 1)
    v <- numeric(d + 1)
    u[d + 1] <- 1
    v[d + 1] <- 1 - p + p * x[d]

    for (k in d:1)
    {
        u[k] <- (p + (1 - p) * x[k]) / 2 * u[k + 1] + v[k + 1] / 2
        v[k] <- (p * x[k - 1] + (1 - p) * x[k]) / 2 * u[k + 1] +
                (1 - p + p * x[k - 1]) / 2 * v[k + 1]
    }
    u[1]
}
```

Exemple 3 - Fonction récursive

Vérifions notre fonction ! On peut procéder par les cas limites ou un exemple simple.

En observant la construction de f, on remarque que lorsque $\boldsymbol{x}$ ne contient que des 1, le résultat est 1.

On peut aussi essayer un exemple simple à la main. En posant $\boldsymbol{x} = (0.1, 0.2)$ et $p = 0.5$, on obtient $u_3 = 1$, $v_3 = 0.6$, $u_2 = 0.6$, $v_2 = 0.24$ et $f(\boldsymbol{x}, p) = u_1 = 0.285$.

On obtient :

```
> f(c(1, 1, 1), 0.5)
Error in v[k] <- (p * x[k - 1] + (1 - p) * x[k])/2 * u[k + 1] + (1 - p +  :
  replacement has length zero
> f(c(0.1, 0.2), 0.5)
Error in v[k] <- (p * x[k - 1] + (1 - p) * x[k])/2 * u[k + 1] + (1 - p +  :
  replacement has length zero
```

Exemple 3 - Fonction récursive

Cette erreur inattendue relève un problème dans notre algorithme, à la ligne 7 :

$$v[k] \leftarrow (px[k-1] + (1-p)x[k])\tfrac{1}{2}u[k+1] + (1-p+px[k-1])\tfrac{1}{2}v[k+1]$$

En s'aidant du message d'erreur de `R`, on remarque que lorsque $k = 1$, on essaie de trouver l'élément x_0, qui n'existe pas.

Cependant, on remarque aussi qu'il n'est pas nécessaire de calculer v_1.

On peut alors appliquer un changement à notre code pour que la boucle soit de d à 2 et on calcule u_1 séparément ensuite.

Exemple 3 - Fonction récursive

Voici notre nouvelle fonction :

```
f <- function(x, p)
{
    d <- length(x)
    u <- numeric(d + 1)
    v <- numeric(d + 1)
    u[d + 1] <- 1
    v[d + 1] <- 1 - p + p * x[d]
    for (k in d:2)
    {
        u[k] <- (p + (1 - p) * x[k]) / 2 * u[k + 1] + v[k + 1] / 2
        v[k] <- (p * x[k - 1] + (1 - p) * x[k]) / 2 * u[k + 1] +
                (1 - p + p * x[k - 1]) / 2 * v[k + 1]
    }
    u[1] <- (p + (1 - p) * x[1]) / 2 * u[1 + 1] + v[1 + 1] / 2
    u[1]
}
```

Exemple 3 - Fonction récursive

Vérifions :

```
> f(c(1, 1, 1), 0.5)
[1] 1
> f(c(0.1, 0.2), 0.5)
[1] 0.285
```

On obtient les résultats espérés !

Exemple 4 - Application mathématique

Le raisonnement computationnel peut aussi nous aider à résoudre des problèmes purement mathématiques sans programmation.

On démontre le fort lien entre le raisonnement computationnel et le raisonnement mathématique dans le prochain exemple.

Soit la fonction de densité conjointe

$$f_{X,Y}(x,y) = \frac{18y\mathrm{e}^{-xy}}{(3+y)^3}, \quad x > 0, y > 0.$$

Calculer $E\left[\mathrm{e}^{-X}|Y=y\right]$ en fonction de y.

On a ici un contexte de probabilités, sans aucune programmation.

Exemple 4 - Application mathématique

Encore une fois, le problème semble complexe, mais décomposons le.

On sait que pour calculer $E[e^{-X}|Y = y]$, nous aurons besoin de la fonction de densité conditionnelle $f_{X|Y=y}(x)$.

Pour calculer $f_{X|Y=y}(x)$, on sait qu'on aura besoin de $f_Y(y)$.

On a alors un plan précis à exécuter :

1. Calculer $f_Y(y)$;
2. Calculer $f_{X|Y=y}(x)$;
3. Calculer $E\left[\mathrm{e}^{-X}|Y = y\right]$.

Exemple 4 - Application mathématique

Commençons par la première étape, on sait trouver la fonction de densité marginale de Y, par le cours ACT-1002, en intégrant sur le support de X :

$$\begin{aligned} f_Y(y) &= \int_0^\infty \frac{18y\mathrm{e}^{-xy}}{(3+y)^3}\,\mathrm{d}x \\ &= \frac{-18y\mathrm{e}^{-xy}}{y(3+y)^3}\Big|_0^\infty \\ &= \frac{18}{(3+y)^3}. \end{aligned}$$

Exemple 4 - Application mathématique

Passons à la deuxième étape :

$$f_{X|Y=y}(x) = \frac{f_{X,Y}(x,y)}{f_Y(y)} = \frac{\frac{18y\mathrm{e}^{-xy}}{(3+y)^3}}{\frac{18}{(3+y)^3}} = y\mathrm{e}^{-xy}.$$

Exemple 4 - Application mathématique

Finalement, on passe à la troisième étape :

$$\begin{aligned} E\left[\mathrm{e}^{-X}|Y=y\right] &= \int_0^\infty \mathrm{e}^{-x} f_{X|Y=y}(x)\,\mathrm{d}x \\ &= \int_0^\infty \mathrm{e}^{-x} y\mathrm{e}^{-xy}\,\mathrm{d}x \\ &= \int_0^\infty y\mathrm{e}^{-x(y+1)}\,\mathrm{d}x \\ &= \frac{y}{y+1}\int_0^\infty (y+1)\mathrm{e}^{-x(y+1)}\,\mathrm{d}x \\ &= \frac{y}{y+1}, \end{aligned}$$

en reconnaissant la densité d'une exponentielle.

Note : on aurait aussi pu reconnaître que $(X|Y=y) \sim Exp(y)$ et que $E\left[\mathrm{e}^{-X}|Y=y\right] = \mathcal{L}_{X|Y=y}(1) = \frac{y}{y+1}$ pour une solution plus courte.

Exemple 4 - Application mathématique

On remarque que même en faisant face à un problème de probabilité à première vue complexe, on peut utiliser le raisonnement computationnel à notre avantage et décomposer le problème en plus petits problèmes connus.

Au final, le problème initial n'était pas si complexe !

Conseils pour la réussite du cours

Conseils pour la réussite du cours

- Éviter le retard + être alerte pendant le cours.
- Exercices traditionnels à faire en  aussi pour les réponses ;
- Utiliser des vérifications pour les calculs informatiques ;
- Bien comprendre les preuves qui sont présentées ;
- Utilise activement le document d'annexes ;
- Essayer les exercices des ateliers avant le début de ces derniers ;
- Utiliser les ateliers adéquatement ;
- « Essayer » jusqu'à développer l'intuition.

Après-propos

Après-propos

Site du cours : monportail.ulaval.

Livre de référence : [Cossette and Marceau, 2023].

Diapositives

- Logiciel : LaTeX
- Package : Beamer
- Éditeur en ligne : Overleaf

Calculs et illustrations

- Toutes les calculs et les illustrations ont été réalisés dans le langage R.
- Les codes R ont été conçus dans l'environnement de développement intégré RStudio.
- Le logiciel GNU R et les bibliothèques sont disponibles sur le site du R Project et du Comprehensive R Archive Network CRAN.

Références

Références I

Barana, A., Boetti, G., Conte, M. M., and Perrotta, A. (2025).
Rethinking computational practices in financial mathematics to boost collaboration : a comparative analysis to explore the redesign of lab activities and its impact on the knowledge co-construction process.
Teaching Mathematics and its Applications : An International Journal of the IMA, page hraf010.

Broley, L., Buteau, C., Modeste, S., Rafalska, M., and Stephens, M. (2024).
Computational thinking and mathematics.
In *Handbook of Digital Resources in Mathematics Education*, pages 323–360. Springer.

Références II

Cossette, H. and Marceau, E. (2023).
Mathématiques actuarielles du risque : modèles, mesures de risque et méthodes quantitatives.
Monographie.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., and Wilensky, U. (2016).
Defining computational thinking for mathematics and science classrooms.
Journal of science education and technology, 25(1) :127–147.