Méthodes numériques en actuariat avec R

Analyse numérique

Vincent Goulet

Université Laval

Méthodes numériques en actuariat avec R

Analyse numérique

Vincent Goulet

Professeur titulaire
École d'actuariat, Université Laval

Avec la collaboration de

Laurent Caron

Édition 2019.05

Code source
Voir sur GitLab

Couverture
Le reptile en couverture est un caméléon panthère Ambilobe (*Furcifer pardalis*) originaire de Madagascar.

Crédit photo : Stefan Overmann ; `http://fc-foto.de/25896357`

Introduction

Les ordinateurs ne savent pas compter. Ou, en fait, très peu. Ils ne savent traiter que des 0 et des 1 et sont incapables de représenter tous les nombres réels — chose qu'un humain peut faire, du moins conceptuellement. Cela signifie qu'à peu près tout calcul effectué dans un ordinateur comporte une part d'erreur d'arrondi et de troncature. Comme on ne souhaite généralement pas que cette erreur devienne trop grande, il importe de connaître ses sources afin de la diminuer le plus possible. C'est, entre autres choses, l'objet du chapitre 4.

Les procédures numériques pour résoudre des équations à une variable, optimiser une fonction ou calculer une intégrale définie sont aujourd'hui aisément accessibles dans une foule de logiciels à connotation mathématique ou même dans une simple calculatrice. Or, comment ces calculs sont-ils effectués, quels sont les algorithmes à l'œuvre en arrière-scène ? Le chapitre 5 se penche sur les méthodes de base de résolution d'équations à une variable et le chapitre 6 sur celles d'intégration numérique.

On présente également au chapitre 5 les principales fonctions d'optimisation disponibles dans Excel et dans R.

Utilisation de l'ouvrage

Chaque chapitre de cet ouvrage propose un problème à résoudre au fil du texte. L'énoncé du problème, les indices en cours de chapitre et la solution complète se présentent dans des sections marquées des symboles ⚙, 🛟 et ⚡.

L'étude de l'ouvrage implique des allers-retours entre le texte et le code R à la fin de chaque chapitre. Ce code informatique et les commentaires qui l'accompagnent vise à enrichir vos apprentissages. Assurez-vous donc de lire attentivement tant les commentaires que le code, d'exécuter le code pas-à-pas et de bien comprendre ses effets.

Le code informatique est distribué avec l'ouvrage sous forme de fichiers de script. De plus, à chaque fichier `.R` correspond un fichier `.Rout` contenant les résultats de son évaluation non interactive.

Fonctionnalités interactives

En consultation électronique, ce document se trouve enrichi de plusieurs fonctionnalités interactives.

- Intraliens du texte vers une ligne précise d'une section de code informatique et, en sens inverse, du numéro de la ligne vers le point de la référence dans le texte. Ces intraliens sont marqués par la couleur ▬.
- Intraliens entre le numéro d'un exercice et sa solution, et vice versa. Ces intraliens sont aussi marqués par la couleur ▬.
- Intraliens entre les citations dans le texte et leur entrée dans la bibliographie. Ces intraliens sont marqués par la couleur ▬.
- Hyperliens vers des ressources externes marqués par le symbole ↗ et la couleur ▬.
- Table des matières, liste des tableaux, liste des figures et liste des vidéos permettant d'accéder rapidement à des ressources du document.

Blocs signalétiques

Le document est parsemé de divers types de blocs signalétiques inspirés de AsciiDoc ↗ qui visent à attirer votre attention sur une notion. Vous pourrez rencontrer l'un ou l'autre des blocs suivants.

Astuce ! Ces blocs contiennent un truc, une astuce, ou tout autre type d'information utile.

Avertissement ! Ces blocs mettent l'accent sur une notion ou fournissent une information importante pour la suite.

Attention ! Vous risquez de vous brûler — métaphoriquement s'entend — si vous ne suivez pas les recommandations de ces blocs.

Important ! Ces blocs contiennent les remarques les plus importantes. Veillez à en tenir compte.

Ces blocs contiennent des remarques additionnelles sur la matière ou des informations amusantes, mais non essentielles.

Ces blocs contiennent des liens vers des vidéos dans ma chaine YouTube dédiée à ce document de référence. Les vidéos sont répertoriées dans la liste des vidéos.

Ces blocs vous invitent à interrompre la lecture du texte pour passer à l'étude du code R des sections d'exemples.

Remarques spécifiques à macOS.

Document libre

Tout comme R et l'ensemble des outils présentés dans ce document, le projet « Méthodes numériques en actuariat avec R » s'inscrit dans le mouvement de l'informatique libre. Vous pouvez accéder à l'ensemble du code source en format LaTeX en suivant le lien dans la page de copyright. Vous trouverez dans le fichier `README.md` toutes les informations utiles pour composer le document.

Votre contribution à l'amélioration du document est également la bienvenue ; consultez le fichier `CONTRIBUTING.md` fourni avec ce document et voyez votre nom ajouté au fichier `COLLABORATEURS`.

Remerciements

Je tiens à souligner la précieuse collaboration de MM. Mathieu Boudreault, Sébastien Auclair et Louis-Philippe Pouliot lors de la rédaction des exercices et des solutions.

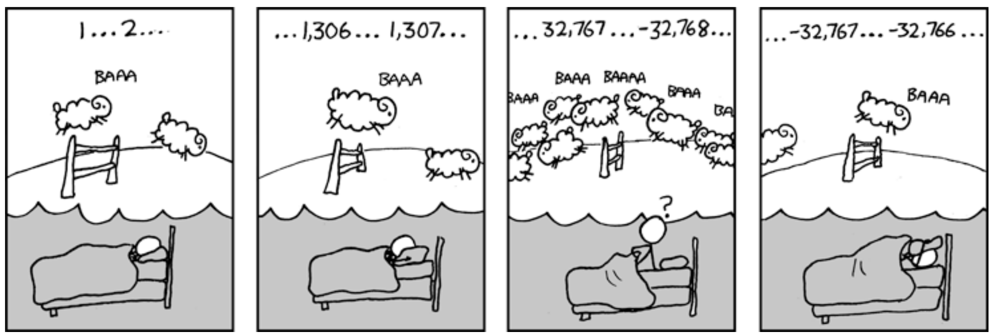
Table des matières

Liste des tableaux

Liste des figures

Liste des vidéos

Le numéro indiqué à gauche est celui de la section dans laquelle se trouve le bloc signalétique.



Tiré de XKCD.com

4 Arithmétique des ordinateurs

Objectifs du chapitre

- Concevoir l'effet de la représentation interne des nombres dans un ordinateur sur la précision des calculs.
- Convertir un nombre décimal vers une base quelconque, et vice versa.
- Exprimer un nombre décimal dans la représentation interne d'un ordinateur selon la norme IEEE 754.
- Planifier ses calculs avec un ordinateur de manière à diminuer l'impact des erreurs d'arrondis et de troncature et optimiser le temps de calcul.

4.1 Énoncé du problème

Les étudiants d'actuariat utilisent pour la plupart la calculatrice TI-30XS MultiView de Texas Instruments puisque c'est un modèle agréé pour les examens des organismes professionnels.

Peut-on se fier aux résultats de cette calculatrice ? Effectuons un calcul simple pour une valeur de x donnée :

$$(x-1)(x+1)-x^2+1.$$

Le résultat devrait bien entendu être égal à zéro, et ce, peu importe la valeur de x.

Si l'on fait le calcul avec $x = 10^6$, le résultat est bien 0.

Si l'on fait le calcul avec $x = 10^8$, le résultat est maintenant de… 1.

Que s'est-il passé ? Doit-on douter de la qualité de la calculatrice ?

4.2 Les ordinateurs ne savent pas compter

Le type de bogue le plus fréquemment rapporté dans les forums de discussion de R a trait au fait que le logiciel ne retourne pas le bon résultat lors d'opérations arithmétiques simples.

> Étudiez le code des lignes 15-44 du fichier de script `arithmetique_ordinateurs.R` reproduit à la section 4.10. Il contient des exemples de calculs simples que R n'arrive apparemment pas à effectuer correctement. Les exemples sont tous tirés de véritables messages envoyés à la liste de discussion `r-help`.

On devine que les créateurs de R n'ont pas négligé une fonctionnalité aussi fondamentale pour un langage mathématique que le calcul arithmétique. Les supposées erreurs ci-dessus relèvent toutes de la représentation interne des nombres dans un ordinateur et d'erreurs d'arrondi inhérentes aux opérations arithmétiques avec ces nombres. En effet, un des plus célèbres principes de programmation proposés par Kernighan et Plauger (1978) est :

$$10{,}0 \text{ fois } 0{,}1 \text{ ne donne jamais vraiment } 1{,}0.$$

D'ailleurs, les auteurs des faux rapports de bogues dans `r-help` sont invariablement renvoyés à l'entrée 7.31 de la foire aux questions de R.

Quiconque utilise un ordinateur pour le calcul scientifique devrait connaître les grands principes du fonctionnement interne d'un ordinateur afin, entre autres :

- d'éviter les erreurs d'arrondi et de troncature ;
- d'éviter les dépassements et soupassements de capacité (*overflow* et *underflow*) ;
- d'optimiser le code informatique.

Ce chapitre se penche sur les grands principes de l'arithmétique des ordinateurs. Quoique, en fait, les ordinateurs ne savent pas faire d'arithmétique à proprement parler. Ils ne connaissent que deux états : ouvert (1) et fermé (0). Nous en profiterons donc d'abord pour réviser la conversion des nombres décimaux vers, et de, n'importe quelle base. Après avoir, à la section 4.4, introduit les principales unités de mesure en informatique, nous expliquerons la représentation interne des nombres dans un ordinateur en nous concentrant sur la double précision. Nous pourrons ainsi justifier que les ordinateurs ne peuvent représenter qu'un sous-ensemble des nombres réels, d'où les erreurs d'arrondi et de troncature. La section 4.6 explique comment ces erreurs surviennent. On y donne également quelques pistes pour éviter ou pour diminuer l'impact de ces erreurs. Enfin, le chapitre se clôt par un survol d'un sujet quelque peu périphérique : la représentation interne des caractères.

Avant d'entamer la lecture du chapitre, je vous recommande de visionner une vidéo d'introduction qui offre un survol du sujet de l'arithmétique des ordinateurs.

4.3 Conversion de base

La *base*, dans un système de numération, est le nombre de symboles (habituellement les chiffres) qui pourront servir à exprimer des nombres. L'humain s'est habitué à compter en base 10, le système *décimal*, où les symboles sont $0, 1, \dots, 9$. De nos jours, la grande majorité des ordinateurs travaillent toutefois en base 2, le système *binaire*. Les autres systèmes d'usage courant, principalement en informatique, sont l'*octal* (base 8) et l'*hexadécimal* (base 16). Dans ce dernier système, les symboles utilisés pour les nombres 10–15 sont généralement les lettres A–F. (C'est pourquoi l'on retrouve ces six lettres sur le pavé des calculatrices scientifiques.)

Cette section étudie la conversion des nombres entre la base 10 et une autre base quelconque.

4.3.1 Notation et définitions

De manière générale, soit x un nombre (entier pour le moment) dans la base de numération b composé de m chiffres ou symboles, c'est-à-dire

$$x = x_{m-1}x_{m-2} \cdots x_1x_0,$$

où $0 \leq x_i \leq b - 1$. On a donc

$$x = \sum_{i=0}^{m-1} x_i b^i. \tag{4.1}$$

Lorsque le contexte ne permet pas de déterminer avec certitude la base d'un nombre, celle-ci est identifiée en indice du nombre par un nombre décimal. Par exemple, 10011_2 est le nombre binaire 10011.

Exemple 4.1. Soit le nombre décimal 348. Selon la notation ci-dessus, on a $x_0 = 8$, $x_1 = 4$, $x_2 = 3$ et $b = 10$. En effet,

$$348 = 3 \times 10^2 + 4 \times 10^1 + 8 \times 10^0.$$

Ce nombre a les représentations suivantes dans d'autres bases. En binaire :

$$\begin{aligned} 101011100_2 = 1 \times 2^8 &+ 0 \times 2^7 + 1 \times 2^6 \\ &+ 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 \\ &+ 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0. \end{aligned}$$

En octal :

$$534_8 = 5 \times 8^2 + 3 \times 8^1 + 4 \times 8^0.$$

En hexadécimal :

$$15C_{16} = 1 \times 16^2 + 5 \times 16^1 + 12 \times 16^0.$$

Des représentations ci-dessus, l'hexadécimale est la plus compacte : elle permet de représenter avec un seul symbole un nombre binaire comptant jusqu'à quatre chiffres. C'est, entre autres, pourquoi c'est une représentation populaire en informatique. □

Dans un ordinateur réel (par opposition à théorique), l'espace disponible pour stocker un nombre est fini, c'est-à-dire que $m < \infty$. Le plus grand nombre que l'on peut représenter avec m chiffres ou symboles en base b est

$$\begin{aligned} x_{\max} &= \sum_{i=0}^{m-1} (b-1) b^i \\ &= (b-1) \sum_{i=0}^{m-1} b^i \\ &= (b-1) \left(\frac{b^m - 1}{b - 1} \right) \\ &= b^m - 1. \end{aligned}$$

Par exemple, le plus grand nombre décimal représentable avec $m = 4$ symboles est

$$x_{\max} = 9\,999 = 10\,000 - 1 = 10^4 - 1,$$

alors que le plus grand nombre binaire est seulement

$$x_{\max} = 1111 = 10000 - 1 = 2^4 - 1 = 15_{10}.$$

Par une extension naturelle de ce qui précède, un nombre composé de $m \geq 0$ symboles dans sa partie entière et $n \geq 0$ symboles dans sa partie fractionnaire est représenté en base b comme

$$\begin{aligned} x &= x_{m-1}x_{m-2}\cdots x_1x_0{,}x_{-1}x_{-2}\cdots x_{-n} \\ &= \sum_{i=-n}^{m-1} x_i b^i. \end{aligned}$$

Le symbole qui sépare les parties entière et fractionnaire du nombre est la *séparation fractionnaire* (une virgule en français, un point en anglais).

4.3.2 Conversion vers une base quelconque

Avant de discuter de la conversion de nombres décimaux vers une base quelconque, il convient de définir les notions de *quotient* et de *reste* d'une division.

Le quotient est la partie entière de la division de deux entiers a et d ; sa représentation mathématique habituelle est

$$q = \left\lfloor \frac{a}{d} \right\rfloor, \tag{4.2}$$

où $\lfloor x \rfloor$ est la fonction qui retourne le plus grand entier inférieur ou égal à x. Le reste de la division est simplement la valeur

$$r = a - d \left\lfloor \frac{a}{d} \right\rfloor. \tag{4.3}$$

Évidemment, on a $r \in \{0, 1, \dots, d-1\}$. Le reste est le résultat de l'opération modulo, notée $r = a \bmod d$.

On remarquera que le premier chiffre en partant de la droite d'un entier décimal est le reste de la division de ce nombre par 10, que le second chiffre est le reste de la division par 10 du quotient de la division précédente, et ainsi de suite.

Exemple 4.2. Cet exemple semblera plutôt artificiel dans la mesure où nous sommes habitués à la représentation des nombres en base 10. Néanmoins, il permet d'illustrer les propos du paragraphe précédent. Soit le nombre 2 547. Dans ce nombre :

$$\begin{aligned} &7 \text{ est le reste de } \frac{2\,547}{10} && 7 = 2\,547 - 10 \left\lfloor \frac{2\,547}{10} \right\rfloor \\ &4 \text{ est le reste de } \frac{254}{10} && 4 = 254 - 10 \left\lfloor \frac{254}{10} \right\rfloor \end{aligned}$$

$$5 \text{ est le reste de } \frac{25}{10} \qquad 5 = 25 - 10\left\lfloor \frac{25}{10} \right\rfloor$$
$$2 \text{ est le reste de } \frac{2}{10} \qquad 2 = 2 - 10\left\lfloor \frac{2}{10} \right\rfloor .$$

□

La conversion d'un nombre décimal en une base b implique simplement de diviser par b plutôt que par 10 et de déterminer le symbole dans la base b correspondant au reste de chaque division.

L'algorithme suivant reprend ces idées de manière plus formelle et en ajoutant le traitement de la partie fractionnaire. Nous démontrerons plus loin qu'il n'est pas réellement nécessaire de savoir effectuer la conversion de la partie fractionnaire d'un nombre réel.

Algorithme 4.1 (Conversion de la base 10 vers la base b)**.** *Soit x un nombre réel en base* 10*.*

1. *Poser $i \leftarrow 0$ et $v \leftarrow \lfloor x \rfloor$.*
2. *Répéter les étapes suivantes jusqu'à ce que $v = 0$:*
   a) *Poser $d_i \leftarrow v \bmod b$ et trouver x_i, le symbole dans la base b correspondant à d_i ;*
   b) *Poser $v \leftarrow \lfloor v/b \rfloor$;*
   c) *Poser $i \leftarrow i + 1$.*
3. *Poser $i \leftarrow 1$ et $v \leftarrow x - \lfloor x \rfloor$.*
4. *Répéter les étapes suivantes jusqu'à ce que $v = 0$ ou que $i = n$ (le nombre voulu ou maximal de chiffres après la séparation fractionnaire) :*
   a) *Poser $d_{-i} \leftarrow \lfloor bv \rfloor$ et trouver x_{-i}, le symbole dans la base b correspondant à d_{-i} ;*
   b) *Poser $v \leftarrow bv - d_{-i}$;*
   c) *Poser $i \leftarrow i + 1$.*
5. *Retourner*
$$x_b = x_{m-1}x_{m-2} \cdots x_1 x_0{,}x_{-1}x_{-2} \cdots x_{-n}.$$

Exemple 4.3. Soit le nombre décimal 23,31 que l'on convertit en binaire (base 2) avec un maximum de cinq chiffres après la séparation fractionnaire ($n = 5$). Le tableau 4.1 montre le processus en suivant les étapes de l'algorithme 4.1. On obtient le résultat final en combinant la dernière colonne du tableau 4.1 (a) lue de bas en haut avec la dernière colonne du tableau 4.1 (b)

Tab. 4.1 – Conversion d'un nombre décimal en binaire

(a) partie entière

i	v	$\lfloor v/2 \rfloor$	$v \bmod 2$	x_i
0	23	11	1	1
1	11	5	1	1
2	5	2	1	1
3	2	1	0	0
4	1	0	1	1

(b) partie fractionnaire

i	v	$2v$	$\lfloor 2v \rfloor$	x_{-i}
1	0,31	0,62	0	0
2	0,62	1,24	1	1
3	0,24	0,48	0	0
4	0,48	0,96	0	0
5	0,96	1,92	1	1

lue de haut en bas. Ainsi, la représentation binaire du 23,31 limitée à cinq chiffres après la virgule est 10111,01001.

□

Tel que mentionné ci-dessus, il est tout à fait possible de se passer de la conversion de la partie fractionnaire. Pour convertir le nombre réel x en base b avec un maximum de n chiffres dans la partie fractionnaire, il suffit de :

1. multiplier x par b^n ;
2. convertir l'entier $\lfloor xb^n \rfloor$ en base b ;
3. déplacer la virgule de n positions vers la gauche.

Exemple 4.4. Soit de nouveau la conversion de 23,31 en binaire avec une partie fractionnaire d'au plus cinq chiffres. En suivant les étapes de la remarque ci-dessus, on a :

1. $23{,}31 \times 2^5 = 23{,}31 \times 32 = 745{,}92$;
2. la représentation binaire de $745 = 2^9 + 2^7 + 2^6 + 2^5 + 2^3 + 1$ est 1011101001 ;
3. enfin, en déplaçant la virgule de cinq positions vers la gauche, on obtient 10111,01001, comme à l'exemple 4.3.

□

On tire une conclusion intéressante de la procédure ci-dessus. S'il existe un entier $k \leq n$ tel que la partie fractionnaire de x est un multiple de $1/b^k$, alors la multiplication de x par b^k résultera en un entier. Par conséquent, la représentation de x en base b aura une partie fractionnaire de longueur finie. Autrement dit, pour obtenir une représentation binaire finie, la partie fractionnaire d'un nombre réel doit être un multiple de $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \dots$ Ce n'était pas le cas à l'exemple 4.4 (la partie fractionnaire étant 0,31), d'où la nécessité de limiter le nombre de chiffres après la virgule.

4.3.3 Conversion en décimal

La conversion d'un nombre en base b vers la base 10 repose essentiellement sur la définition (4.1), mais avec chaque symbole x_i remplacé pour son équivalent décimal. Un algorithme de conversion est le suivant.

Algorithme 4.2 (Conversion de la base b vers la base 10)**.** *Soit x*

$$x_b = x_{m-1}x_{m-2} \cdots x_1x_0{,}x_{-1}x_{-2} \cdots x_{-n}$$

un nombre réel en base b.

1. *Poser $x \leftarrow 0$ et $y \leftarrow 0$.*
2. *Pour $i = m-1, m-2, \dots, 0$, faire les étapes suivantes :*
   a) *Trouver d_i, le nombre décimal correspondant au symbole x_i ;*
   b) *Poser $x \leftarrow xb + d_i$.*
3. *Pour $i = -n, -n+1, \dots, -1$, faire les étapes suivantes :*
   a) *Trouver d_i, le nombre décimal correspondant au symbole x_i ;*
   b) *Poser $y \leftarrow (y + d_i)/b$.*
4. *Retourner $x + y$.*

On peut, ici aussi, aisément éviter de convertir la partie fractionnaire. Il suffit de déplacer la virgule de n positions vers la droite dans le nombre x_b de manière à obtenir un entier, de convertir ce nombre en décimal avec les étapes 1 et 2 de l'algorithme 4.2 et, finalement, de diviser le nombre obtenu par b^n.

Exemple 4.5. On convertit le nombre binaire 101011,011 en décimal. Le tableau 4.2 illustre le processus de conversion selon l'algorithme 4.2 (le chiffre en surbrillance est celui traité lors de chaque étape de l'algorithme). Le résultat final est 43,375.

De manière équivalente, mais plus simple, on peut déplacer la virgule de trois positions vers la droite pour obtenir l'entier binaire 101011011. Ce nombre est $2^8 + 2^6 + 2^4 + 2^3 + 2 + 1 = 347$ en décimal. Enfin, $347 \div 2^3 = 43{,}375$. □

TAB. 4.2 – Conversion d'un nombre binaire en décimal

(a) partie entière

i	d_i	x
5	101011,011	1
4	101011,011	2
3	101011,011	5
2	101011,011	10
1	101011,011	21
0	101011,011	43

(b) partie fractionnaire

i	d_{-i}	y
3	101011,011	0,5
2	101011,011	0,75
1	101011,011	0,375

Exemple 4.6. Soit le nombre hexadécimal AC2,3D8. On peut le convertir en base 10 à partir de la définition :

$$\begin{aligned}\text{AC2,3D8} &= 10 \times 16^2 + 12 \times 16 + 2 + 3 \times 16^{-1} + 13 \times 16^{-2} + 8 \times 16^{-3} \\ &= 2\,754{,}240\,234\,375.\end{aligned}$$

□

4.3.4 Conversion avec des bases générales

Il existe de nombreuses bases de numération d'usage courant où chaque symbole est (potentiellement) exprimé dans une base différente. On a qu'à penser à l'heure ou à l'ensemble du système de mesure impérial. Or, il peut s'avérer utile de savoir convertir de et vers une base quelconque. Le matériel de cette section se retrouve dans très peu d'ouvrages d'analyse numérique ou de statistique numérique. Pourtant, il existe quelques applications intéressantes de la conversion de et vers des bases générales, comme nous le verrons.

On peut généraliser la notion de nombre présentée dans les sections précédentes à une collection de « symboles », chacun dans une base différente. On restreint la discussion aux entiers sans perte de généralité. On a donc

$$x = x_{m-1}x_{m-2} \cdots x_1x_0,$$

où x_{m-1} est un nombre en base b_{m-1}, x_{m-2} est un nombre en base b_{m-2}, etc. Nous dirons que le nombre x est exprimé en base $[b_{m-1}\ b_{m-2}\ \dots\ b_0]$. On a alors

$$x = \sum_{i=0}^{m-1} x_i \prod_{j=-1}^{i-1} b_j, \tag{4.4}$$

TAB. 4.3 – Conversion d'un nombre décimal dans une base générale

i	v	b_i	$\lfloor v/b_i \rfloor$	$v \bmod b_i$	x_i
0	91 492	60	1 524	52	52
1	1 524	60	25	24	24
2	25	24	1	1	1
3	1	365	0	1	1

avec $b_{-1} = 1$.

Les algorithmes de conversion 4.1 et 4.2 demeurent essentiellement valides ici. Il suffit de remplacer chaque mention de b par b_i.

Exemple 4.7. Le jour de l'année et l'heure du jour est un « nombre » exprimé en base $[365\ 24\ 60\ 60]$. En utilisant directement l'équation (4.4), le nombre de secondes correspondant à 1 jour, 2 heures, 33 minutes et 20 secondes est :

$$\begin{aligned} 1\text{ j } 2\text{ h } 33\text{ min } 20\text{ sec} &= 1 \times (24)(60)(60) + 2 \times (60)(60) + 33 \times 60 + 20 \\ &= 95\,600 \text{ secondes.} \end{aligned}$$

Si l'on fait la conversion en sens inverse, le nombre de jours, d'heures, de minutes et de secondes correspondant à 91 492 secondes est, en utilisant l'algorithme 4.1 avec la base $[365\ 24\ 60\ 60]$: 1 jour, 1 heure, 24 minutes et 52 secondes (voir le tableau 4.3 pour les calculs). □

Les bases générales sont particulièrement utiles pour assigner ou extraire les éléments d'une matrice ou d'un tableau dans un ordre non séquentiel. Typiquement, l'index de l'élément à traiter est le résultat d'un calcul. L'exemple suivant illustre cette idée.

Exemple 4.8. Soit $\mathbf{A}$ une matrice 4×5 que l'on suppose remplie en ordre lexicographique (par ligne). Il est simple de déterminer, ici, que le 14ᵉ élément est a_{34}. Or, on observe que la conversion du nombre $14 - 1 = 13$ dans la base $[4\ 5]$ donne :

$$\begin{aligned} 13 \div 5 = 2 \text{ reste } 3 \quad &\Rightarrow \quad x_0 = 3 \\ 2 \div 4 = 0 \text{ reste } 2 \quad &\Rightarrow \quad x_1 = 2, \end{aligned}$$

soit le « nombre » $(2, 3)$. En additionnant 1 à ce résultat, on obtient précisément la position du 14ᵉ élément dans la matrice. Les opérations -1 et $+1$

sont rendues nécessaires par le fait que les lignes et les colonnes sont numérotées à partir de 1, alors que les systèmes de numération débutent à 0. □

> La fonction `arrayInd` de R permet de faire exactement ce qui est décrit dans l'exemple précédent, soit retourner les coordonnées du i^{e} élément d'un tableau. Consultez le code des lignes 47-67 du fichier de script `arithmetique_ordinateurs.R` reproduit à la section 4.10 pour quelques exemples d'utilisation.

4.4 Unités de mesure

Les ordinateurs que nous utilisons couramment fonctionnent en base 2. La plus petite quantité d'information qu'un ordinateur peut traiter est un *bit* (compression de l'anglais *binary digit*). En informatique, un bit est égal à $\boxed{0}$ ou à $\boxed{1}$. Le symbole du bit est b.

Un *octet* (*byte*) est un groupe de huit bits. Son symbole est o ou B. L'octet est l'unité de mesure la plus fréquemment utilisée en informatique, en grande partie parce que le codage d'un caractère dans la plupart des langues occidentales requiert un octet ; voir la section 4.8. Les termes pour de plus grandes quantités de bits ou d'octets utilisent généralement les préfixes usuels du système international d'unités. Par exemple :

- 1 kilooctet (ko) est $2^{10} = 1\,024$ octets ;
- 1 mégaoctet (Mo) est $2^{20} = 1\,048\,576$ octets ;
- 1 gigaoctet (Go) est $2^{30} = 1\,073\,741\,824$ octets.

On voit que cette pratique fort répandue entraîne des distorsions par rapport aux définitions usuelles de kilo, méga, giga, etc., et que cette distorsion s'amplifie au fur et à mesure que l'on grimpe dans l'échelle des tailles d'objets. Pour cette raison, on a défini les préfixes kibi, mébi, gibi, etc., mais ceux-ci demeurent peu utilisés à ce jour.

Dans l'industrie de l'informatique, on joue beaucoup avec les unités pour montrer son produit sous un jour favorable. Deux exemples :

1. Les fournisseurs d'accès Internet expriment la vitesse de téléchargement en Mb/sec, alors que la taille des fichiers est généralement affichée en octets. Il faut donc diviser par 8 la vitesse annoncée pour avoir une idée plus juste du temps requis pour télécharger un fichier.

2. Les fabricants de disques durs divisent la capacité réelle de leurs disques par 10^9 pour l'exprimer en gigaoctets. Ainsi, un disque dont la capacité annoncée est de 100 Go ne peut contenir, en fait, que $100 \times 10^9 \div 2^{30} = 93{,}132$ Go de données. Cette confusion serait éliminée si les fabricants affichaient plutôt une capacité de 93,132 gibioctets. Moins vendeur…

4.5 Représentation en virgule flottante

Cette section décrit la manière standard de représenter les nombres réels dans les ordinateurs d'usage courant. Il est utile de connaître les grandes lignes afin de comprendre pourquoi les nombres réels n'ont pas tous une représentation exacte dans un ordinateur et comment surviennent et se propagent les erreurs d'arrondi. L'étude du sujet est également intéressante en soi pour constater comment les ingénieurs et les informaticiens sont parvenus à stocker un maximum d'information dans un espace limité.

Tel que mentionné précédemment, la capacité de stockage d'un ordinateur, bien que vaste de nos jours, n'en demeure pas moins limitée. Par conséquent, les nombres y sont représentés par un ensemble de m bits. Habituellement, m est un multiple de 2 et sa valeur détermine le type de nombre. Par exemple, des définitions usuelles sont $m = 2^4 = 16$ pour un entier, $m = 2^5 = 32$ pour un nombre réel en simple précision (*float* dans plusieurs langages de programmation) et $m = 2^6 = 64$ pour un nombre réel en double précision (*double*).

Il existe deux grandes façons de représenter les nombres (réels) à l'aide de m bits.

Virgule fixe Dans cette représentation, la position de la virgule dans le nombre est prédéterminée et, par conséquent, n'a pas besoin d'être conservée en mémoire. On réserve alors m_e positions pour la partie entière et m_f pour la partie fractionnaire (avec $m = m_e + m_f$). La représentation en virgule fixe est particulièrement utile dans les applications financières.

Virgule flottante Dans la représentation des nombres en virgule flottante, celle-ci peut être placée n'importe où dans le nombre. L'étendue de cette représentation est beaucoup plus grande que la virgule fixe, mais ceci au détriment de la précision puisque quelques bits doivent être réservés pour stocker la position de la virgule dans le nombre.

La représentation en virgule flottante est celle utilisée dans les applications scientifiques et celle sur laquelle nous nous concentrons dans la suite.

$$\underbrace{\underbrace{\boxed{S}}_{m_S=1}\;\underbrace{\boxed{\quad E \quad}}_{m_E=11}\;\underbrace{\boxed{\qquad\qquad F \qquad\qquad}}_{m_F=52}}_{m=64}$$

FIG. 4.1 – Représentation schématique d'un nombre en double précision dans la norme IEEE 754.

La représentation en virgule flottante est analogue à la notation scientifique. Le nombre réel x est entièrement défini par un bit de signe S, un exposant positif E et une mantisse M tel que

$$x = (-1)^S \times B^{E-e} \times M, \tag{4.5}$$

où B est la base de la représentation et e est un entier prédéterminé appelé le décalage, ou le biais, de l'exposant. Le recours au décalage facilite les calculs internes et évite de devoir réserver un autre bit pour le signe de l'exposant en le stockant sous forme non signée (c'est-à-dire sous forme d'entier positif).

La norme IEEE 754 définit la manière standard de représenter les nombres en virgule flottante dans les ordinateurs (IEEE, 2003 ; Wikipedia, 2012, pour une excellente présentation). En premier lieu, la norme stipule que la base dans la représentation est $B = 2$. Ceci est implicite et n'est pas stocké où que ce soit dans l'ordinateur. En second lieu, la norme suppose que, pour les nombres dits *normalisés*, la mantisse est toujours de la forme $M = 1{,}F$, où F est un entier binaire. Le bit à gauche de la virgule est appelé le *bit caché* puisqu'il est lui aussi implicite et non stocké dans l'ordinateur.

Nous décrivons plus en détail la norme pour les nombres réels en *double précision* puisque c'est le type avec lequel R travaille toujours[1]. Tout d'abord, un nombre en double précision est stocké dans un mot de $m = 64$ bits (8 octets) divisé ainsi de gauche à droite : $m_S = 1$ bit pour le signe, $m_E = 11$ bits pour l'exposant et $m_F = 52$ bits pour la partie fractionnaire de la mantisse. Le premier bit du mot utilisé pour le signe est appelé le *bit fort*. Voir la figure 4.1 pour une représentation schématique.

On remarque que le bit caché confère à la mantisse une longueur de mot effective (ou précision) de 53 bits.

1. Il est possible de créer des vrais entiers dans R en ajoutant un suffixe `L` à un nombre entier. Ainsi, `1L` est un nombre entier dans le sens où `is.integer(1L)` est `TRUE`. Cela est relativement peu utilisé en programmation normale.

TAB. 4.4 – Valeurs spéciales dans la norme IEEE 754

S	E	F	Représentation binaire	Valeur spéciale
0	0	0	$\boxed{0}\boxed{00000000000}\boxed{00\cdots00}$	0
1	0	0	$\boxed{1}\boxed{00000000000}\boxed{00\cdots00}$	-0
0	2 047	0	$\boxed{0}\boxed{11111111111}\boxed{00\cdots00}$	$+\infty$
1	2 047	0	$\boxed{1}\boxed{11111111111}\boxed{00\cdots00}$	$-\infty$
0 ou 1	2 047	$F \neq 0$	$\boxed{S}\boxed{11111111111}\boxed{\quad F \quad}$	NaN[a]

[a] *Not a number*, par exemple $\frac{0}{0}$ ou $\frac{\infty}{\infty}$.

Le décalage est $e = 2^{m_E-1} - 1 = 2^{10} - 1 = 1\,023$. Avec une longueur de mot de 11 bits, les valeurs possibles de E vont de 0 à $2^{11} - 1 = 2\,047$. Cependant, les valeurs 0 et 2 047 sont réservées pour des usages spéciaux ; voir le tableau 4.4. Par conséquent, les valeurs possibles de l'exposant (une fois décalé) pour les nombres en double précision vont de $-1\,022$ à $1\,023$.

Exemple 4.9. Tel qu'expliqué à la section 4.3.2, le nombre $\frac{1}{2}$ possède une représentation binaire exacte :

$$\begin{aligned}\frac{1}{2} &= 2^{-1}\\ &= (-1)^0 \times 2^{1\,022-1\,023} \times 1{,}0.\end{aligned}$$

Puisque la représentation de l'exposant est $1\,022_{10} = 1111111110_2$, la représentation interne de $\frac{1}{2}$ selon la norme IEEE 754 est

$$\boxed{0}\boxed{01111111110}\boxed{00\cdots00}.$$

En revanche, le nombre $\frac{1}{10}$ n'a pas une représentation binaire finie. En utilisant l'algorithme 4.1, on trouve (le côté droit de l'égalité étant en binaire) :

$$\begin{aligned}\frac{1}{10} &= 0{,}0001100110011001100110011001\ldots\\ &= 2^{-4} \times 1{,}1001100110011001\ldots\\ &= (-1)^0 \times 2^{1\,019-1\,023} \times 1{,}1001100110011001\ldots\end{aligned}$$

Or, $1\,019_{10} = 1111111011_2$, d'où la représentation interne finie de $\frac{1}{10}$ est

$$\boxed{0}\boxed{01111111011}\boxed{1001\cdots1001}.$$

Si l'on reconvertit ce nombre en décimal, on obtient

$$2^{-4} \times \left(1 + \sum_{k=0}^{12} (2^{-4k-1} + 2^{-4k-4})\right),$$

un nombre près de, mais pas exactement égal à, $\frac{1}{10}$ (le programme de calcul en multiprécision `bc` donne 0,09999999999999999166). Dès lors, la multiplication de ce nombre par 10 ne donnera pas 1. Voilà qui justifie le principe de programmation énoncé à la page 2. □

Les principales caractéristiques des nombres en double précision sont les suivantes.

1. Soit $x_{\max}$ le plus grand nombre représentable. Parce que l'exposant $E = 2\,047$ est réservé, on a

$$\begin{aligned} x_{\max} &= \boxed{0}\boxed{11111111110}\boxed{11 \cdots 11} \\ &= (-1)^0 \times 2^{1\,023} \times 1{,}11 \cdots 11 \\ &= 2^{1\,023} \times (2 - 2^{-52}) \\ &= 1{,}797\,693 \times 10^{308}. \end{aligned}$$

2. Soit $x_{\min}$ le plus petit nombre normalisé représentable. Parce que l'exposant $E = 0$ est réservé, on a

$$\begin{aligned} x_{\min} &= \boxed{0}\boxed{00000000001}\boxed{00 \cdots 00} \\ &= (-1)^0 \times 2^{-1\,022} \times 1{,}00 \cdots 00 \\ &= 2^{-1\,022} \\ &= 2{,}225\,074 \times 10^{-308}. \end{aligned}$$

Afin de rendre la transition vers 0 moins abrupte, la norme IEEE 754 définit également des nombres *dénormalisés*. Ceux-ci sont identifiés par $E = 0$ et une partie fractionnaire F non nulle. Cependant, par définition, les nombres dénormalisés ont $E - e = -1\,022$ et leur bit caché est 0. Par conséquent, le plus petit nombre dénormalisé est stocké sous la forme

$$\boxed{0}\boxed{00000000000}\boxed{00 \cdots 01}$$

et sa valeur est

$$\begin{aligned} x_{\min} &= (-1)^0 \times 2^{-1\,022} \times 0{,}00 \cdots 01 \\ &= 2^{-1\,022} \times 2^{-52} \\ &= 2^{-1\,074} \\ &= 4{,}940\,656 \times 10^{-324}. \end{aligned}$$

3. On a la même étendue pour les nombres négatifs, soit

$$[-1{,}797\,693 \times 10^{308}, -2{,}225\,074 \times 10^{-308}]$$

ou

$$[-1{,}797\,693 \times 10^{308}, -4{,}940\,656 \times 10^{-324}].$$

en considérant les nombres dénormalisés.

4. Soit ε la plus petite valeur tel que $1 + \varepsilon \neq 1$ dans la représentation en virgule flottante. Cette valeur est appelée l'*epsilon de la machine* ou la *précision de la machine*. Or, puisque

$$1 = (-1)^0 \times 2^0 \times 1{,}00 \cdots 00,$$

le nombre suivant est

$$(-1)^0 \times 2^0 \times 1{,}00 \cdots 01.$$

Par conséquent, on a $\varepsilon = 2^{-52} = 2{,}220\,446 \times 10^{-16}$.

5. Tout nombre x représente en fait un intervalle de $\mathbb{R}$. Par exemple, tout nombre dans l'intervalle $[1, 1 + \varepsilon)$ est représenté par le nombre 1 dans l'ordinateur.

6. On a un ensemble fini de nombres pour représenter $\mathbb{R}$. Or, cet ensemble est plus dense près de 0 que pour les grandes valeurs. En effet, le nombre suivant x_{min} est

$$\begin{aligned} x_{\text{min}}^{+} &= (1 + \varepsilon) \times 2^{-1\,022} \\ &= x_{\text{min}} + \varepsilon \times 2^{-1\,022} \\ &= x_{\text{min}} + 2^{-1\,074}, \end{aligned}$$

alors que le nombre précédant x_{max} est

$$\begin{aligned} x_{\text{max}}^{-} &= (2 - 2^{-52} - \varepsilon) \times 2^{1\,023} \\ &= x_{\text{max}} - \varepsilon \times 2^{1\,023} \\ &= x_{\text{max}} - 2^{971}. \end{aligned}$$

L'écart entre les deux plus petits nombres représentables est donc de $2^{-1\,074} \approx 10^{-324}$, alors que celui entre les deux plus grands est de $2^{971} \approx 10^{292}$!

Tout calcul impliquant un nombre $x > x_{\max}$ ($x < -x_{\max}$) entraîne un *dépassement de capacité*. Habituellement, cela entraîne l'arrêt immédiat des calculs et un résultat de $+\infty$ ($-\infty$). À l'autre bout du spectre, un calcul impliquant un nombre $x < x_{\min}$ entraîne un *soupassement de capacité* et le résultat est habituellement considéré égal à 0.

R conserve dans une liste nommée `.Machine` les valeurs mentionnées ci-dessus — ainsi que plusieurs autres — pour l'architecture de l'ordinateur courant. Par exemple, on confirme les valeurs de ε, $x_{\min}$, $x_{\max}$, B, m_f et m_E, dans l'ordre, pour l'architecture x86 :

```
> .Machine[c(1, 3:6, 11)]
$double.eps
[1] 2.220446e-16

$double.xmin
[1] 2.225074e-308

$double.xmax
[1] 1.797693e+308

$double.base
[1] 2

$double.digits
[1] 53

$double.exponent
[1] 11
```

Le 1er décembre 2014, YouTube a annoncé que la vidéo de la chanson *Gangnam Style* avait « brisé » son compteur en dépassant le nombre maximal de visionnements admissible, soit 2 147 483 647. Cette limite, bien inférieure à celle des nombres réels en double précision, provient de la représentation des nombres sous forme d'entier signé sur 32 bits. Voir l'exercice 4.4 pour les détails.

TAB. 4.5 – Représentation en virgule flottante simplifiée

x	$\mathrm{fl}(x)$
$2{,}5$	$0{,}25000 \times 10^1$
$-42{,}182$	$-0{,}42182 \times 10^2$
$0{,}214356$	$0{,}21436 \times 10^0$

4.6 Éléments d'arithmétique en virgule flottante

La section précédente expliquait pourquoi les nombres stockés dans un ordinateur ne sont pas toujours ceux que l'on croit — ou que l'on voudrait. Puisque l'ordinateur ne peut représenter tous les nombres réels, toute opération arithmétique avec des nombres en virgule flottante implique une erreur d'arrondi ou de troncature (selon l'architecture de l'ordinateur) dont il importe de tenir compte lors de la mise en oeuvre de certains algorithmes. Cette section présente quelques grands principes d'arithmétique en virgule flottante ainsi que de bons usages pour minimiser les erreurs d'arrondi. Consulter les ouvrages Monahan (2001) ; Burden et Faires (1988) ; Knuth (1997), entre autres, pour une discussion plus complète de ce vaste sujet.

Aux fins de cette section, on note $\mathrm{fl}(x)$ la représentation en virgule flottante du nombre x et l'on définit la représentation simplifiée

$$\mathrm{fl}(x) = (-1)^S \times 10^E \times 0{,}F, \tag{4.6}$$

où F compte cinq chiffres significatifs, dont le dernier est arrondi. Le tableau 4.5 fournit quelques exemples.

Dans la norme IEEE 754, les règles d'arrondi de base sont : arrondir à la valeur la plus près (0,14 devient 0,1 et 0,16 devient 0,2) ; arrondir un nombre exactement à mi-chemin au nombre avec un dernier chiffre significatif pair ou nul (0,05 devient 0,0 alors que 0,15 devient 0,2).

4.6.1 Erreurs d'arrondi ou de troncature

Lors de l'addition et de la soustraction de nombres en virgule flottante, on rend les exposants égaux en déplaçant les bits de la mantisse du plus petit nombre vers la droite. Il en résulte une perte de bits significatifs et donc une *erreur d'arrondi*. Dans les cas extrêmes où les deux opérandes sont de grandeur très différente, le plus petit opérande devient nul suite à la perte de tous ses bits significatifs.

La situation pour la multiplication et la division est quelque peu différente, mais la source d'erreur d'arrondi demeure la même.

À toute fin pratique, tout calcul fait naître de l'erreur d'arrondi et celle-ci augmente avec le nombre d'opérations. Les quelques principes de programmation ci-dessous, lorsque suivis par le programmeur prudent et consciencieux, permettent d'atténuer l'impact de l'erreur d'arrondi.

(P1) L'addition et la soustraction en virgule flottante ne sont pas associatives. Additionner les nombres en ordre croissant de grandeur afin d'accumuler des bits significatifs.

(P2) Éviter de soustraire un petit nombre d'un grand, ou alors le faire le plus tard possible dans la chaîne des calculs.

(P3) Pour calculer une somme alternée, additionner tous les termes positifs et tous les termes négatifs, puis faire la soustraction.

(P4) Multiplier et diviser des nombres d'un même ordre de grandeur. Si x et y sont presque égaux, le calcul x/y sera plus précis en virgule flottante que $y^{-1}x$.

Les exemples ci-dessous illustrent ces principes.

Exemple 4.10. Soit $x = 7$, $y = 4$ et $z = 100\,000$. Dans la représentation simplifiée (4.6), on a $\mathrm{fl}(x) = 0{,}70000 \times 10^1$, $\mathrm{fl}(y) = 0{,}40000 \times 10^1$ et $\mathrm{fl}(z) = 0{,}10000 \times 10^6$. Or, $x + y + z = 100\,011$ et

$$\begin{aligned}
[\mathrm{fl}(x) + \mathrm{fl}(y)] + \mathrm{fl}(z) &= (0{,}70000 \times 10^1 + 0{,}40000 \times 10^1) + 0{,}10000 \times 10^6 \\
&= 0{,}11000 \times 10^2 + 0{,}10000 \times 10^6 \\
&= 0{,}00001 \times 10^6 + 0{,}10000 \times 10^6 \\
&= 0{,}10001 \times 10^6,
\end{aligned}$$

soit un résultat raisonnablement précis. Cependant,

$$\begin{aligned}
\mathrm{fl}(x) + [\mathrm{fl}(y) + \mathrm{fl}(z)] &= 0{,}70000 \times 10^1 + (0{,}00000 \times 10^6 + 0{,}10000 \times 10^6) \\
&= 0{,}00000 \times 10^6 + 0{,}10000 \times 10^6 \\
&= 0{,}10000 \times 10^6.
\end{aligned}$$

En effectuant les opérations dans un mauvais ordre, on obtient $x + y + z = z$ même si $x \neq 0$ et $y \neq 0$. □

Exemple 4.11. Soit $x = 7$, $y = 100\,006$ et $z = 100\,002$. On a $z-y-x = -11$. Or,

$$\begin{aligned}\mathrm{fl}(z) - [\mathrm{fl}(y) + \mathrm{fl}(x)] &= 0{,}10000 \times 10^6 - (0{,}10000 \times 10^6 + 0{,}00000 \times 10^6) \\ &= 0{,}00000 \times 10^0,\end{aligned}$$

alors que

$$\begin{aligned}[\mathrm{fl}(z) - \mathrm{fl}(y)] - \mathrm{fl}(x) &= (0{,}10000 \times 10^6 - 0{,}10000 \times 10^6) - 0{,}70000 \times 10^1 \\ &= 0{,}00000 \times 10^1 - 0{,}70000 \times 10^1 \\ &= -0{,}70000 \times 10^1.\end{aligned}$$

□

L'exemple suivant est adapté de Monahan (2001).

Exemple 4.12. L'évaluation numérique de probabilités loin dans les queues d'une fonction de répartition peut s'avérer imprécise selon la technique employée. Par exemple, la fonction de répartition de la distribution logistique est

$$F(x) = (1 + e^{-x})^{-1}.$$

La vraie valeur de $\Pr[X > 6]$ est $1 - F(6) = 1 - (1 + e^{-6})^{-1} = 0{,}002\,472\,623$. L'évaluation de cette probabilité dans notre représentation en virgule flottante avec la formule ci-dessus est

$$\begin{aligned}1 - [1 \div (1 + \mathrm{fl}(e^{-6}))] &= 1 - [1 \div (1 + 0{,}24788 \times 10^{-2})] \\ &= 1 - [0{,}10000 \times 10^1 \div 0{,}10025 \times 10^1)] \\ &= 1 - 0{,}99751 \times 10^0 \\ &= 0{,}10000 \times 10^1 - 0{,}09975 \times 10^1 \\ &= 0{,}25000 \times 10^{-2}.\end{aligned}$$

(Nous n'avons pas écrit les entiers en virgule flottante ci-dessus pour alléger la notation.) Toutefois, si l'on utilise plutôt la formule équivalente

$$\begin{aligned}1 - F(6) &= \frac{e^{-6}}{1 + e^{-6}} \\ &= \frac{1}{1 + e^{6}}\end{aligned}$$

dans laquelle l'opération de soustraction entre deux nombres presque égaux est déjà effectuée, on obtient le résultat bien plus précis

$$\begin{aligned}
1 \div (1 + \text{fl}(e^6)) &= 1 \div (1 + 0{,}40343 \times 10^3)) \\
&= 1 \div (0{,}00100 \times 10^3 + 0{,}40343 \times 10^3) \\
&= 1 \div 0{,}40443 \times 10^3 \\
&= 0{,}24726 \times 10^{-2}.
\end{aligned}$$

□

> Toutes les fonctions de calcul de fonction de répartition de R permettent d'évaluer plus précisément les probabilités $1 - F(x)$ pour une grande valeur de x en spécifiant l'option `lower.tail = FALSE`.

Les deux principes ci-dessous permettent d'éviter des dépassements ou des soupassements de capacité et d'améliorer la précision des calculs.

(P5) Chercher des formules mathématiques équivalentes évitant de devoir traiter des très grands ou des très petits nombres.

(P6) Travailler en échelle logarithmique. Par exemple, le produit de deux grands nombres x et y dépassera la capacité plus tard et demeurera précis plus longtemps s'il est calculé sous la forme $e^{\log x + \log y}$.

Exemple 4.13. Soit $X_1, \dots, X_n$ un échantillon aléatoire tiré d'une distribution de Pareto translatée, dont la fonction de répartition est

$$F(x; \mu, \alpha) = 1 - \left(\frac{\mu}{x}\right)^{\alpha}, \quad x > \mu.$$

On peut démontrer que l'estimateur du maximum de vraisemblance de α est

$$\hat{\alpha} = \frac{n}{\ln(X_1 \cdots X_n / X_{(1)}^n)},$$

où $X_{(1)} = \min(X_1, \dots, X_n)$. Soit `x` un objet R contenant un échantillon de 1 000 valeurs d'une distribution Pareto translatée de moyenne 5 000 (le contenu de cet objet n'est pas affiché ici pour des raisons évidentes). Le calcul de l'estimateur $\hat{\alpha}$ directement par la formule entraîne rapidement un dépassement de capacité, tant lors du produit $X_1 \cdots X_n$ que lors de l'opération $X_{(1)}^n$:

```
> prod(x)
[1] Inf
> min(x)^length(x)
[1] Inf
```

Le résultat est donc `NaN` :

```
> length(x)/log(prod(x)/min(x)^length(x))
[1] NaN
```

Selon la grandeur des données dans l'échantillon, la formule équivalente

$$\hat{\alpha} = \frac{n}{\ln \prod_{i=1}^{n} X_i / X_{(1)}}$$

peut éviter le dépassement de capacité. Elle n'est toutefois d'aucun secours dans le présent exemple :

```
> length(x)/log(prod(x/min(x)))
[1] 0
```

On remarquera de plus que cette approche nécessite un grand nombre de multiplications (voir la section 4.6.2), en plus d'ouvrir la porte à des erreurs d'arrondi.

Pour obtenir une réponse on utilisera plutôt une autre formule algébriquement équivalente :

$$\hat{\alpha} = \frac{n}{\sum_{i=1}^{n} \ln X_i - n \ln X_{(1)}}.$$

On obtient alors

```
> length(x)/(sum(log(x)) - length(x) * log(min(x)))
[1] 1.398935
```

Cet exemple illustre combien des calculs *algébriquement* équivalents ne sont pas nécessairement *numériquement* équivalents. □

4.6.2 Coût des opérations

Les ordinateurs modernes disposent d'une unité de calcul en virgule flottante (FPU) intégrée au processeur. Cette unité est évidemment très optimisée et, par conséquent, elle accélère beaucoup le calcul en virgule flottante. Néanmoins, certaines opérations sont plus coûteuses à réaliser que d'autres

TAB. 4.6 – Coût relatif de quelques opérations en virgule flottante

Opération arithmétique	Coût relatif
Addition et soustraction	1,0
Multiplication	1,3
Division	3,0
Racine carrée	4,0
Logarithme	15,4

en termes de temps de calcul. À titre indicatif, on trouve au tableau 4.6 le coût relatif approximatif de quelques opérations en virgule flottante.

À titre d'exemple, considérons le calcul d'une simple moyenne arithmétique

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

Si l'on effectue le calcul tel qu'écrit ci-dessus, cela requiert $n-1$ additions et une division. En revanche, le calcul algébriquement équivalent

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n},$$

requiert $n-1$ divisions et $n-1$ additions. Pour n grand, utiliser la première approche plutôt que la seconde fera une différence.

4.7 Indice pour le problème

La calculatrice TI-30XS MultiView conserve 13 chiffres significatifs en mémoire pour les calculs. Nous pourrons tenter d'expliquer les résultats du calcul de $(x-1)(x+1) - x^2 + 1$ à l'aide de la représentation simplifiée (4.6) en utilisant une mantisse F comptant 13 chiffres significatifs.

4.8 Codage de caractères

Pendant que l'on discute de la représentation interne des nombres dans un ordinateur, on peut toucher un mot de la représentation interne, ou le codage, des caractères. Ce sujet demeure une cible mouvante puisque de nouveaux standards apparaissent encore après de nombreuses années de systèmes incompatibles et basés sur la langue anglaise.

Peu importe le système retenu, les caractères doivent être codés en binaire dans l'ordinateur. La partie la plus difficile consiste à créer un système standard permettant aux ordinateurs et autres appareils numériques de communiquer entre eux. Le premier standard d'usage courant fut le Code américain normalisé pour l'échange d'information, mieux connu sous son acronyme ASCII (ANSI, 1986). La norme ASCII définit des codes numériques pour 128 caractères, soit 95 affichables (lettres, chiffres, symboles divers et l'espace) et 33 non affichables (essentiellement des caractères de contrôle tels que saut de ligne, retour de chariot ou espacement arrière).

À titre d'exemple, les lettres majuscules A–Z occupent les cases 65–90 (1000001–1011010 en binaire) alors que les lettres minuscules a–z occupent les cases 97–122 (1100001–1111010 en binaire). On constate que les versions majuscule et minuscule d'une même lettre ne diffèrent, dans leur représentation binaire, que par leur second bit le plus significatif. Le changement de casse d'une lettre ou d'un mot est donc une opération très simple et rapide.

La représentation ASCII ne requiert en soi que sept bits. Néanmoins, on a rapidement codé les caractères sur un octet (huit bits) puisqu'il s'agit du type de donnée natif des ordinateurs depuis les années 1970. L'ajout d'un bit a créé de l'espace pour 128 caractères additionnels (cases 128–255). Une myriade de systèmes de codage différents sont alors apparus pour supporter les caractères des langues autres que l'anglais (les caractères accentués, par exemple) ainsi que d'autres symboles graphiques. La norme ISO 8859-1, ou Latin 1 (ISO, 1998), a fini par s'imposer comme l'un des standards les plus répandus. Ces listes de codes fixes se révèlent toutefois problématiques lors de l'apparition d'un nouveau symbole. Par exemple, pour faire de la place pour le symbole de l'euro à la fin des années 1990, il a fallu retirer un symbole de ISO 8859-1. Avec quelques autres changements, la nouvelle liste de code est devenue ISO 8859-15. De plus, comment doit-on traiter les langues CJC (chinois, japonais, coréen) qui comptent des milliers de symboles différents ?

Depuis le début des années 1990, la mondialisation de l'informatique a suscité un effort concerté de création d'un système de codage standard couvrant la presque totalité des systèmes d'écriture du monde. Le système devait aussi permettre la composition de textes en plusieurs langues, par exemple en français et dans une écriture de droite à gauche comme l'hébreu ou l'arabe. Le standard ainsi créé est Unicode (Unicode Consortium, 2007). Le plus populaire système de codage capable de représenter l'ensemble des caractères Unicode est *Unicode Transformation Format* 8 bits, ou UTF-8 (Unicode Consortium, 2007, section 3.9). Dans l'UTF-8, chaque caractère est codé sur une suite d'un à quatre octets et les premiers 128 caractères sont identiques à la norme ASCII. L'UTF-8 est le système de codage par défaut dans macOS et les plus récentes distributions GNU/Linux.

Le système R supporte les caractères multi-octets de manière assez exhaustive. Sans doute aidés en cela par le fait qu'il s'agit d'un projet international, les développeurs de R ont mis beaucoup d'effort pour assurer l'internationalisation et la localisation du logiciel ; voir Ripley (2005).

4.9 Solution du problème

Nous calculons $(x-1)(x+1)-x^2+1$ avec, en premier lieu, $x = 10^6$. Dans la représentation interne de la calculatrice, ce nombre est stocké sous la forme :

$$\mathrm{fl}(x) = 0{,}1000000000000 \times 10^7.$$

Aux fins des calculs, le nombre 1 sera, quant à lui, traité comme

$$\mathrm{fl}(1) = 0{,}0000001000000 \times 10^7.$$

Nous avons donc

$$\begin{aligned}
[\mathrm{fl}(x) - &\mathrm{fl}(1)][\mathrm{fl}(x) + \mathrm{fl}(1)] - \mathrm{fl}(x)^2 + \mathrm{fl}(1) \\
&= (0{,}0999999000000 \times 10^7)(0{,}1000001000000 \times 10^7) \\
&\quad - 0{,}1000000000000 \times 10^{13} + 0{,}0000001000000 \times 10^7 \\
&= 0{,}0999999999999 \times 10^{13} - 0{,}1000000000000 \times 10^{13} \\
&\quad + 0{,}0000000000001 \times 10^{13} \\
&= -0{,}0000000000001 \times 10^{13} + 0{,}0000000000001 \times 10^{13} \\
&= 0,
\end{aligned}$$

ce qui est le résultat attendu. On remarque que dans les opérations ci-dessus, le nombre de chiffres significatifs nécessaire ne dépasse jamais la capacité de 13 de la calculatrice.

Répétons ces calculs avec, cette fois, la valeur $x = 10^8$ et les représentations internes

$$\begin{aligned}
\mathrm{fl}(x) &= 0{,}1000000000000 \times 10^9 \\
\mathrm{fl}(1) &= 0{,}0000000010000 \times 10^9.
\end{aligned}$$

Nous avons

$$
\begin{aligned}
[\mathrm{fl}(x)-&\mathrm{fl}(1)][\mathrm{fl}(x)+\mathrm{fl}(1)]-\mathrm{fl}(x)^2+\mathrm{fl}(1) \\
&= (0{,}0999999990000\times 10^9)(0{,}1000000010000\times 10^9) \\
&\quad - 0{,}1000000000000\times 10^{17} + 0{,}0000000010000\times 10^9 \\
&= 0{,}1000000000000\times 10^{17} - 0{,}1000000000000\times 10^{17} \\
&\quad + 0{,}0000000010000\times 10^9 \\
&= 0 + 0{,}0000000010000\times 10^9 \\
&= 1.
\end{aligned}
$$

Là où le calcul de la calculatrice s'écarte de son équivalent mathématique, c'est lors du passage de la première à la seconde égalité. Le produit des deux nombres entre parenthèses doit être arrondi à 13 décimales et il en résulte un nombre numériquement équivalent à $\mathrm{fl}(x)^2$.

Afin de prévenir ce genre d'erreurs, il conviendra donc de respecter les principes de la section 4.6.

4.10 Code informatique

Fichier d'accompagnement `arithmetique_ordinateurs.R`

```
###
### EXEMPLES DE CALCULS EN APPARENCE ERRONÉS
###

## Opérateurs d'addition et de soustraction ne respectant pas les
## règles d'arithmétique de base.
1.2 + 1.4 + 2.8              # 5.4
1.2 + 1.4 + 2.8 == 5.4       # non?!?
2.8 + 1.2 + 1.4              # encore 5.4
2.8 + 1.2 + 1.4 == 5.4       # donc addition non commutative?
2.6 - 1.4 - 1.2              # devrait donner 0
2.6 - 1.5 - 1.1              # correct cette fois-ci

## Division donnant parfois des résultats erronés.
0.2/0.1 == 2                 # ok
(1.2 - 1.0)/0.1 == 2         # pourtant équivalent
0.3/0.1 == 3                 # fonctionnait pourtant avec 0.2

## Distance plus grande entre 3,2 et 3,15 qu'entre 3,10 et
## 3,15.
```

```
3.2 - 3.15 > 0.05          # écart supérieur à 0,05...
3.15 - 3.1 < 0.05          # et cette fois inférieur à 0,05

## Valeurs inexactes dans les suites de nombres générées avec
## 'seq', qui sont générées algébriquement (par additions ou
## soustractions successives).
(a <- seq(0.9, 0.95, by = 0.01))
a == 0.94                  # 0.94 n'est pas dans le vecteur!
(b <- c(0.90, 0.91, 0.92, 0.93, 0.94, 0.95))
b == 0.94                  # mais ici, oui!
a - b                      # remarquer le 5e élément

###
### CONVERSION DANS DES BASES GÉNÉRALES
###

## La fonction 'arrayInd', dont la syntaxe (simplifiée) est
##
##     arrayInd(ind, dim)
##
## retourne les coordonnées des éléments aux positions 'ind'
## dans un tableau de dimensions 'dim'. Par exemple, le 14e
## élément d'une matrice 4 x 5 remplie par colonne (ordre R)
## est en position (2, 4).
arrayInd(14, c(4, 5))      # comparer avec l'exemple 10.7

## C'est toujours un peu plus compliqué avec les tableaux. Par
## exemple, un tableau 3 x 4 x 5 doit être vu comme cinq
## matrices 3 x 4 placées les unes derrière les autres. Où se
## trouve le 'i'ème élément?
arrayInd(8, 3:5)            # élément (2, 3) de 1ère matrice
arrayInd(13, 3:5)           # élément (1, 1) de 2e matrice
arrayInd(59, 3:5)           # élément (2, 4) de 5e matrice
arrayInd(c(8, 13, 59), 3:5) # en un seul appel
```

4.11 Exercices

4.1 Convertir les nombres décimaux suivants en base 6, puis en binaire.

a) 119

b) 343

c) 96

d) 43

4.2 Convertir les nombres hexadécimaux suivants en nombres décimaux.

a) A1B

b) 12A

c) B41

d) BAFFE

4.3 a) Utiliser l'algorithme de conversion des nombres en base b vers la base 10 et les idées de l'exemple 4.8 pour trouver une formule générale donnant la position de l'élément a_{ijk} d'un tableau de dimensions $I \times J \times K$ dans l'ordre de la liste des éléments du tableau. Utiliser l'ordre lexicographique où le tableau est rempli dans l'ordre

$$a_{111}, a_{112}, \dots, a_{11K}, a_{121}, a_{122}, \dots$$

b) Répéter la partie a) en utilisant l'ordre R où le tableau est plutôt rempli dans l'ordre $a_{111}, a_{211}, \dots, a_{I11}, a_{121}, a_{221}, \dots$.

4.4 La plupart des langages de programmation — dont R, voir la note au bas de la page 13 — permettent de définir des nombres entiers « stricts », par opposition à des nombres réels qui s'avèrent simplement être entiers. La représentation de ces nombres entiers est complètement différente de celle de la norme IEEE 754, sauf en ce qui a trait au bit fort, qui sert toujours à stocker le signe du nombre.

Les nombres entiers sont stockés dans une représentation dite « en complément à deux » (Wikipedia, 2013). Dans cette notation :

i) un entier positif est représenté en binaire de manière usuelle, avec le bit fort égal à 0 ;

ii) pour un entier négatif, on inverse les bits de la représentation de sa valeur absolue et on ajoute 1 au résultat.

Le bit fort est automatiquement mis à 1 (pour représenter un nombre négatif) par l'opération d'inversion. Les deux opérations ci-dessus sont équivalentes à stocker un nombre négatif x en tant que $2^n - |x|$.

Par exemple, la représentation sur 8 bits du nombre 5 est

$$\boxed{0}\,\boxed{0000101}.$$

Pour représenter -5, on inverse ces bits et on ajoute 1 :

$$\boxed{1}\,\boxed{1111011}.$$

On remarque que $11111011_2 = 251_{10} = 2^8 - 5$.

a) L'un des avantages de la notation en complément à deux est qu'elle évite d'avoir des représentations pour $+0$ et -0. Pour des entiers de 8 bits, $\boxed{0}\boxed{0000000}$ correspond tout naturellement à 0. À quel nombre correspond $\boxed{1}\boxed{0000000}$ selon cette notation ?

b) Un autre avantage, plus important que celui mentionné en a), de la notation en complément à deux est qu'elle permet d'effectuer les opérations arithmétiques en binaire naturellement. Pour constater ce fait, effectuer directement sur 8 bits l'opération $15 - 5$ en utilisant les notations suivantes :

   i) la notation naturelle où le bit fort représente le signe et les sept autres bits pour représentent la valeur absolue du nombre ;

   ii) la notation en complément à deux.

   Dans les deux cas, ignorer les éventuels dépassements de capacité vers la gauche.

c) C'est maintenant le temps d'expliquer le trait d'humour dans le strip de XKCD.com au début du document. Pour ce faire, convertir en décimal les représentations en complément à deux sur 8 bits des nombres de $\boxed{00000000}$ à $\boxed{11111111}$. Répéter l'exercice sur 16 bits. (En fait, convertir non pas toutes les représentations, mais bien seulement quelques valeurs clés qui permettront d'observer la séquence des nombres décimaux correspondants.)

d) Plusieurs langages de programmation permettent de choisir parmi divers types de structures de données pour représenter des nombres réels ou des nombres entiers. Nous avons déjà mentionné les types `float` et `double` pour les nombres réels en simple et en double précision. Cette nomenclature du Fortran se perpétue dans le C et ses nombreux dérivés. Le C dispose aussi des types `int` et `long` pour représenter des nombres entiers, positifs ou négatifs. Le type `int` utilise 16 bits d'espace mémoire et le type `long`, 32 bits. Tous deux ont recours à la notation en complément à deux. Déterminer l'ensemble des entiers représentables pour chacun des types de données `int` et `long`.

4.5 La norme IEEE 754 pour les nombres en virgule flottante (S, E, F) en simple précision est le suivant :

- longueur totale de $m = 32$ bits ;
- 1 bit pour le signe S (valeur de 0 pour un nombre positif) ;
- 8 bits pour l'exposant E, avec un biais de 127 ;

- 23 bits pour la partie fractionnaire F.

Un nombre x est donc représenté comme

$$x = (-1)^S \times 2^{E-127} \times 1{,}F.$$

Trouver les valeurs ε, $x_{\max}$ et $x_{\min}$ pour les nombres en simple précision. Comparer les résultats avec les limites du type `Single` en VBA.

4.6 Représenter les nombres suivants comme des nombres en virgule flottante en simple précision selon la norme IEEE 754.

a) $-1\,234$

b) 55

c) $8\,191$

d) -10

e) $\frac{2}{3}$

f) $\frac{1}{100}$

4.7 Les nombres ci-dessous sont représentés en format binaire selon la norme IEEE 754 pour les nombres en simple précision. Convertir ces nombres en décimal.

a) $\boxed{0}\,\boxed{00111101}\,\boxed{10010000100000000000000}$

b) $\boxed{1}\,\boxed{00111101}\,\boxed{10010000100000000000000}$

c) $\boxed{0}\,\boxed{10000100}\,\boxed{10010000100000000000000}$

d) $\boxed{1}\,\boxed{10000100}\,\boxed{10010000100000000000000}$

4.8 Trouver, pour les nombres des parties a) et c) de l'exercice 4.7, le nombre suivant et le nombre précédent en représentation binaire.

Réponses

La notation x_b signifie que le nombre x est en base b. On omet généralement b pour les nombres en base 10.

4.1 a) 315_6, 1110111_2

b) 1331_6, 101010111_2

c) 240_6, 1100000_2

d) 111_6, 101011_2

4.2 a) 2 587 b) 298 c) 2 881 d) 765 950

4.3 a) $k + K(j - 1 + J(i - 1))$

b) $i + I(j - 1 + J(k - 1))$

4.4 a) -128

b) i) -20 ii) 10

d) Type `Byte` : $\{0, \dots, 255\}$; type `Integer` : $\{-32\,768, \dots, 32\,767\}$;
type `Long` : $\{-2\,147\,483\,648, \dots, 2\,147\,483\,647\}$

4.5 $\varepsilon = 2^{-23} = 1{,}192 \times 10^{-7}$, $x_{\max} = (2 - 2^{-23}) \times 2^{127} = 3{,}403 \times 10^{38}$, $x_{\min} = 2^{-126} = 1{,}175 \times 10^{-38}$ (nombre normalisé) ou $x_{\min} = 2^{-149} = 1{,}401 \times 10^{-45}$ (nombre dénormalisé)

4.6 a) $\boxed{1}\,\boxed{10001001}\,\boxed{00110100100000000000000}$

b) $\boxed{0}\,\boxed{10000100}\,\boxed{10111000000000000000000}$

c) $\boxed{0}\,\boxed{10001011}\,\boxed{11111111111100000000000}$

d) $\boxed{1}\,\boxed{10000010}\,\boxed{01000000000000000000000}$

e) $\boxed{0}\,\boxed{01111110}\,\boxed{01010101010101010101010}$

f) $\boxed{0}\,\boxed{01111000}\,\boxed{01000111101011100001010}$

4.7 a) $2{,}120\,229\,346 \times 10^{-20}$ b) $-2{,}120\,229\,346 \times 10^{-20}$ c) 50,0625 d) $-50{,}0625$

4.8 a) $2{,}120\,229\,508 \times 10^{-20}$ et $2{,}120\,229\,185 \times 10^{-20}$

b) 50,062 503 815 et 50,062 496 185

5 Résolution d'équations à une variable

Objectifs du chapitre

- Résoudre une équation à une variable à l'aide des méthodes de la bissection, du point fixe et de Newton-Raphson.
- Faire un choix parmi l'une des méthodes ci-dessus pour résoudre un problème donné.
- Déterminer si une fonction admet un point fixe dans un intervalle donné.
- Utiliser les fonctions de résolution d'équations et d'optimisation de Excel et R.

5.1 Énoncé du problème

Les fonctions décrivant les heures de lever et de coucher du soleil à Québec pour le mois de janvier 2016 sont représentées par $L(x)$ et $C(x)$, respectivement :

$$\begin{aligned} L(x) &= e^{-0.001431x+2{,}016340} \\ C(x) &= e^{0{,}001312x+2{,}778538}, \end{aligned}$$

où x est le jour du mois (un entier). La fonction du nombre d'heures d'ensoleillement pour une journée est donc $H(x) = C(x) - L(x)$.

Pendant quel jour du mois les Québécois bénéficient-ils de 9 heures d'ensoleillement, à une minute près ?

5.2 Mise en contexte

Ce chapitre passe en revue les méthodes les plus populaires pour résoudre numériquement une équation à une variable. Ce problème, simple seulement en apparence, survient dans une foule de domaines. Débutons par deux exemples, tirés des mathématiques financières et de l'inférence statistique.

Exemple 5.1. Un investisseur est disposé à payer k \$ aujourd'hui en retour d'une série de n versements de 1 \$, le premier dans un an. Pour déterminer le taux de rendement qu'il exige, on doit trouver la valeur de i tel que $a_{\overline{n}|i} = k$ ou, énoncé autrement, la valeur de i tel que

$$f(i) = \frac{1 - (1 + i)^{-n}}{i} - k = 0.$$

□

Exemple 5.2. On a un échantillon aléatoire $X_1, \dots, X_n$ provenant d'une distribution binomiale négative avec fonction de masse de probabilité

$$\Pr[X = x] = \binom{x + r - 1}{r - 1} p^r (1 - p)^x, \quad x = 0, 1, \dots,$$

où p est supposé connu. On souhaite trouver l'estimateur du maximum de vraisemblance du paramètre r.

La fonction de vraisemblance est

$$\begin{aligned} L(r) &= \prod_{i=1}^{n} \Pr[X = x_i] \\ &= \prod_{i=1}^{n} \binom{x_i + r - 1}{r - 1} p^r (1 - p)^{x_i} \end{aligned}$$

et la fonction de log-vraisemblance est

$$\begin{aligned} l(r) = \sum_{i=1}^{n} [&\ln \Gamma(x_i + r) - \ln \Gamma(r) - \ln \Gamma(x_i + 1) \\ &+ r \ln p + x_i \ln(1 - p)]. \end{aligned}$$

Par conséquent, l'estimateur du maximum de vraisemblance $\hat{r}$ est la solution de

$$f(r) = \frac{d}{dr} l(r) = \sum_{i=1}^{n} (\Psi(x_i + r) - \Psi(r) + \ln p) = 0,$$

où $\Psi(x) = \Gamma'(x)/\Gamma(x)$ est la *fonction digamma*. □

Le problème sur lequel nous allons nous pencher consiste donc à trouver la racine d'une fonction $f(x)$, c'est-à-dire la solution de $f(x) = 0$.

5.3 Indice pour le problème

Nous cherchons à déterminer pour quelle valeur de $x \in \{1, 2, \dots, 31\}$ la fonction $H(x)$ vaut 9 ou, de manière équivalente, la racine de l'équation $H(x) - 9 = 0$.

Une technique d'essai-erreur intuitive suivrait probablement à peu près les étapes suivantes :

1. vérifier qu'il existe bel et bien une racine dans l'intervalle $[1, 31]$ en évaluant le nombre d'heures d'ensoleillement aux bornes de l'intervalle, puis déterminer si la fonction est croissante ou décroissante sur l'intervalle (elle est croissante dans le cas présent);
2. essayer une première valeur pour x et calculer la valeur de $H(x)$;
3. si le nombre d'heures d'ensoleillement est trop élevé, diminuer la valeur de x et si, à l'inverse, le nombre d'heures d'ensoleillement est trop faible, augmenter la valeur de x ;
4. répéter les deux étapes précédentes jusqu'à trouver la bonne valeur de x au degré de précision désiré.

Voici une mise en œuvre simple en R. Définissons d'abord une fonction pour calculer le nombre d'heures d'ensoleillement $H(x)$.

```
> H <- function(x)
+ {
+     exp(0.001312 * x + 2.778538) -
+         exp(-0.001431 * x + 2.016340)
+ }
```

Calculons les valeurs au début et à la fin de l'intervalle

```
> H(1)
[1] 8.616558
> H(31)
[1] 9.578721
```

Selon les valeurs à $x = 1$ et $x = 31$, le jour où l'ensoleillement total vaut 9 semble se situer environ à la mi-mois. Essayons donc $x = 15$.

```
> H(15)
[1] 9.064084
```

La valeur est supérieure à 9. Le jour recherché se trouve donc entre 1 et 15. Essayons $x = 10$.

```
> H(10)
[1] 8.903965
```

La valeur est inférieure à 9. Le jour recherché se trouve donc entre 10 et 15. Essayons $x = 13$.

```
> H(13)
[1] 8.999997
```

Nous avons trouvé la bonne valeur à une minute près.

5.4 Méthode de bissection

Toutes les méthodes numériques de résolution d'une équation qui seront étudiées dans ce chapitre reposent sur une procédure systématique d'essais et d'erreurs. La procédure — ou algorithme — utilisée déterminera l'efficacité de la méthode. À ce chapitre, la méthode de bissection est la plus simple et la plus intuitive méthode de résolution, mais aussi la moins efficace.

Soit f une fonction continue sur un intervalle $[a, b]$ choisi de telle sorte que $f(a)$ et $f(b)$ sont de signes opposés. Par le théorème de la valeur intermédiaire, il existe un point x^* tel que $f(x^*) = 0$. En d'autres mots, si une fonction continue est d'un côté de l'axe des abscisses au début d'un intervalle et de l'autre coté à l'autre extrémité de l'intervalle, alors elle a forcément croisé l'axe quelque part dans l'intervalle.

L'idée de la méthode de bissection consiste à trouver la solution x^* par essais successifs en utilisant le point milieu d'intervalles de plus en plus petits, mais que l'on sait toujours contenir le point cherché. L'algorithme 5.1 systématise cette idée et la figure 5.1 l'illustre pour quelques itérations de l'algorithme.

Algorithme 5.1 (Méthode de la bissection)**.** *Soit f une fonction continue sur l'intervalle $[a, b]$ où $f(a)$ et $f(b)$ sont de signes opposés. On cherche x^* tel que $f(x^*) = 0$ en un maximum de $N_{\max}$ itérations.*

1. *Poser $n = 1$.*
2. *Répéter les étapes suivantes.*
    - *2.1. Poser $x_n = (a + b)/2$.*
    - *2.2. Si $f(a)f(x_n) > 0$, poser $a = x_n$, sinon poser $b = x_n$.*
    - *2.3. Si $|x_n - x_{n-1}|/|x_n| < \varepsilon$, poser $x^* = x_n$ et terminer.*
    - *2.4. Si $n \geq N_{\max}$, terminer sans convergence.*

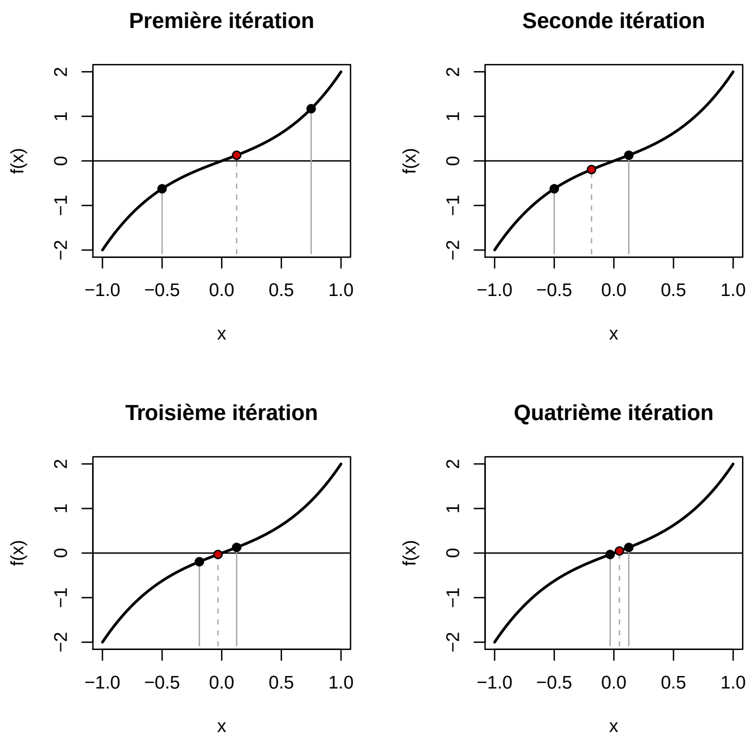


FIG. 5.1 – Illustration de la méthode de bissection. À chaque itération, les points noirs reliés par des segments verticaux en ligne pleine identifient les bornes de l'intervalle dans lequel on sait que se trouve la solution de l'équation $f(x) = 0$. Le point rouge relié par un segment en pointillé est l'essai de l'itération, c'est-à-dire le point milieu de l'intervalle. On remarquera comment ce point devient la borne inférieure ou supérieure de l'intervalle à l'itération suivante.

2.5. *Poser* $n = n + 1$.

L'application de l'algorithme 5.1 génère une suite $\{x_n\}$, $n = 1, 2, \ldots$ qui, si tout se passe bien, converge vers la valeur x^*.

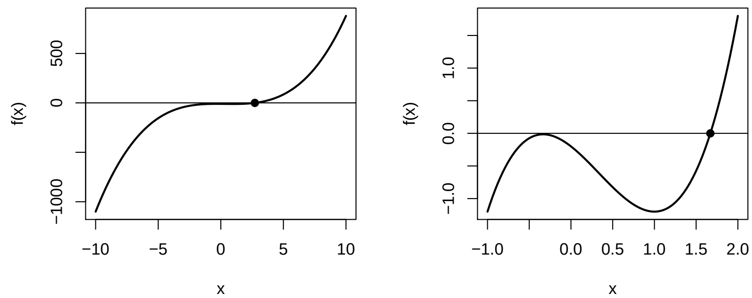


FIG. 5.2 – Deux exemples de fonctions pour lesquelles on pourrait avoir $|f(x_n)| < \varepsilon$ même si $x_n \gg x^*$ ou $x_n \ll x^*$. Le point identifie la véritable racine dans chaque graphique.

À l'étape 2.2 de l'algorithme, le produit $f(a)f(x_n)$ permet de déterminer si l'on se trouve du même côté de l'axe des abscisses aux points a et x_n (produit de deux valeurs négatives ou de deux valeurs positives) ou de part et d'autre de l'axe (produit de valeurs de signes opposés).

L'étape 2.3 de l'algorithme permet de valider si la procédure a convergé vers une réponse. La valeur de ε sera donc en général « petite ». Ici, nous avons utilisé l'erreur relative comme critère d'arrêt. Il s'agit en général du meilleur choix. En effet, un critère tel que $|f(x_n)| < \varepsilon$ peut être satisfait même si $x_n \gg x^*$ ou $x_n \ll x^*$ (voir la figure 5.2 pour deux exemples). Quant à l'erreur absolue $|x_n - x_{n-1}| < \varepsilon$, l'inégalité peut être satisfaite même si la suite $\{x_n\}$ diverge.

Le code informatique de la section 5.15 contient la définition d'une fonction `bissection` dont les arguments sont :

1. la fonction $f(x)$;
2. la valeur de a ;
3. la valeur de b ;
4. la valeur de ε (10^{-6} par défaut) ;
5. le nombre maximal d'itérations $N_{\max}$ (100 par défaut) ;

6. une valeur booléenne indiquant si les calculs intermédiaires doivent ou non être affichés à l'écran : si l'argument est `TRUE`, les valeurs de a, b, x_n et $f(x_n)$ sont affichées à chaque itération.

La fonction retourne une liste dont l'élément `$root` contient la réponse et l'élément `$nb.iter` le nombre d'itérations nécessaire pour obtenir cette réponse.

> Étudiez attentivement les lignes 15-59 du fichier de script `resolution_equations.R` reproduit à la section 5.15. Nous aurons recours à la fonction `bissection` dans les exemples qui suivent.

Exemple 5.3. On cherche la racine de $f(x) = x^3 + 4x^2 - 10$ dans l'intervalle $[1, 2]$. On peut vérifier que la fonction f n'a qu'une seule racine dans cet l'intervalle et que, de plus, $f(1) = -5$ et $f(2) = 14$. Ainsi, $a = 1$ et $b = 2$ constituent des valeurs de départ adéquates.

Après 17 itérations de la fonction `bissection`, la réponse obtenue est, à neuf décimales, 1,365 226 746. L'erreur d'approximation est au maximum de

$$|x^* - x_{17}| < |b_{18} - a_{18}| = |1{,}365234 - 1{,}365227| = 0{,}000007$$

et, puisque $|a_{18}| < |x^*|$,

$$\frac{|x^* - x_{17}|}{|x^*|} < \frac{|b_{18} - a_{18}|}{|a_{18}|} = 5{,}13 \times 10^{-6}.$$

La réponse est donc exacte à au moins cinq décimales près.

> Vérifiez les résultats de cet exemple à l'aide du code des lignes 61-75 du fichier de script `resolution_equations.R`.

□

Exemple 5.4. La valeur actuelle d'une série de dix paiements de fin d'année est 8,2218. Pour déterminer le taux d'intérêt (ou taux de rendement, selon le point de vue), on cherche i tel que $f(i) = a_{\overline{10|}i} - 8{,}2218 = 0$.

> Complétez cet exemple en exécutant les lignes 77-92 du fichier de script `resolution_equations.R`.

□

5.5 Indice pour le problème

La méthode de bissection est en fait une systématisation de la technique d'essai-erreur intuitive présentée précédemment.

On utilise l'algorithme 5.1 avec $f(x) = H(x) - 9$, $a = 1$ et $b = 31$.

5.6 Méthode du point fixe

La solution de l'équation

$$g(x) = x$$

est un *point fixe* de la fonction g. Il s'agit du point où les fonctions $g(x)$ et $y = x$ sont égales, c'est-à-dire là où la fonction g croise la droite à 45° dans le plan.

Tout problème de recherche de racine peut être exprimé comme un problème de point fixe. Par exemple, en définissant

$$g(x) = x - f(x),$$

les problèmes de trouver x tel que $g(x) = x$ ou tel que $f(x) = 0$ sont équivalents.

La méthode du point fixe repose sur le théorème du même nom. Celui-ci fournit les conditions garantissant, d'une part, l'existence d'un point fixe pour une fonction g dans un intervalle donné et, d'autre part, l'unicité de ce point fixe. Un point fixe unique peut néanmoins exister sans que ces conditions ne soient remplies.

Théorème 5.1 (Théorème du point fixe)**.** *Soit g une fonction continue sur un intervalle $[a, b]$.*

1. *Si $g(x) \in [a, b]$ pour tout $x \in [a, b]$, alors g a un point fixe dans $[a, b]$.*
2. *Si, de plus, $g'(x)$ existe sur (a, b) et qu'il existe une constante k tel que*

$$|g'(x)| \leq k < 1$$

pour tout $x \in (a, b)$, alors g a un point fixe unique *dans $[a, b]$.*

Démonstration. Le théorème compte deux volets. Nous en faisons une démonstration simplifiée ne reposant que sur des arguments graphiques.

1. Le premier volet garantit l'existence d'un point fixe sur $[a, b]$ si l'image de la fonction g se trouve aussi dans un intervalle $[a, b]$. Cette paire d'intervalles engendre le carré en pointillé que l'on retrouve dans les graphiques de la figure 5.3.

En définitive, le premier volet du théorème établit que si la fonction continue g entre dans l'intervalle $[a, b]$ par la gauche du carré ($g(x) \in [a, b]$) et en ressort par la droite (idem), alors elle doit forcément croiser la droite $y = x$. Observons le graphique (a) de la figure 5.3 :

- $g(x) \in [a, b]$ pour tout $x \in [a, b]$, donc un point fixe existe nécessairement ;
- $h(x) \notin [a, b]$ pour tout $x \in [a, b]$, donc la fonction n'a pas nécessairement de point fixe dans $[a, b]$;
- $k(x) \notin [a, b]$ pour tout $x \in [a, b]$, mais un point fixe demeure néanmoins possible.

2. Le second volet garantit l'unicité du point fixe si la pente de la fonction sur (a, b) n'excède jamais 1. Cette condition empêche la fonction de croiser la droite $y = x$ une première fois et de la croiser de nouveau suite à une forte augmentation. Comparons les deux fonctions du graphique (b) de la figure 5.3 :
   - $g'(x) < 1$ pour tout $x \in (a, b)$, donc $g(x)$ croise $y = x$ en un seul point et le point fixe est unique ;
   - $h(x) \in [a, b]$ pour tout $x \in [a, b]$, mais $h'(x) > 1$ pour $x \in (a, b)$, ce qui ouvre la porte à des points fixes multiples, tel qu'illustré.

$\square$

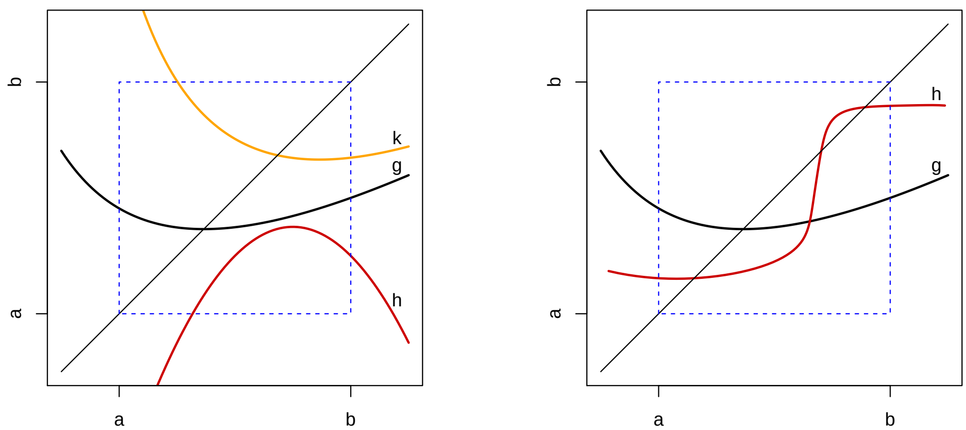
Le théorème suivant, donné sans preuve, établit véritablement la procédure qui sera utilisée par la méthode numérique pour trouver le point fixe d'une fonction.

Théorème 5.2. *Si les deux conditions du théorème 5.1 sont satisfaites, alors la série $\{x_n = g(x_{n-1})\}$ converge vers un point fixe unique dans $[a, b]$.*

Tel que mentionné au début de la section 5.4, toutes les méthodes numériques — et la méthode du point fixe ne fait pas exception — procèdent par essais et erreurs, mais de manière systématique. Le théorème 5.2 nous dit que pour trouver un point fixe, le meilleur essai x_n à faire est la valeur de la fonction g à l'essai précédent, $g(x_{n-1})$. La figure 5.4 illustre comment la suite des essais successifs peut converger vers le point fixe.

Nous pouvons dès lors composer un algorithme de résolution par la méthode du point fixe.

Algorithme 5.2 (Méthode du point fixe)**.** *Soit g une fonction et x_0 une valeur de départ. On cherche x^* tel que $g(x^*) = x^*$ en un maximum de $N_{\max}$ itérations.*



(a) existence du point fixe (b) unicité du point fixe

FIG. 5.3 – Exemples de graphiques permettant de faire la démonstration du théorème du point fixe

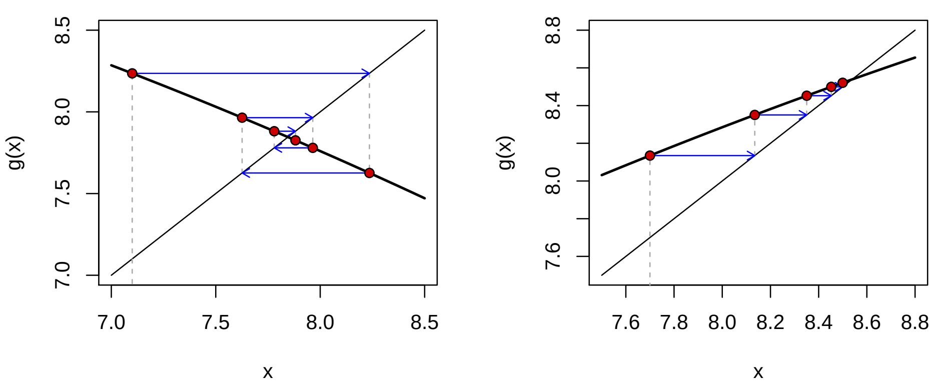


FIG. 5.4 – Illustrations de la méthode du point fixe pour une fonction décroissante et pour une fonction croissante. Les points rouges identifient les essais successifs, le premier étant, ici, le point le plus à gauche dans les graphiques. Les flèches illustrent comment est déterminé l'essai suivant. La relation $x_n = g(x_{n-1})$ implique que l'abscisse d'un essai correspond à la projection de l'ordonnée de l'essai précédent sur la droite $y = x$.

1. *Poser* $n = 1$.
2. *Répéter les étapes suivantes.*
   a) *Poser* $x_n = g(x_{n-1})$.
   b) *Si* $|x_n - x_{n-1}|/|x_n| < \varepsilon$, *alors poser* $x^* = x_n$ *et terminer.*
   c) *Si* $n \geq N_{\max}$, *terminer sans convergence.*
   d) *Poser* $n = n + 1$.

Comme on peut le déduire des illustrations de la figure 5.4, la rapidité de la convergence est fonction de $g'(x)$: plus la pente est *faible*, plus la convergence est *rapide*, et vice versa.

Le code informatique de la section 5.15 définit une fonction `pointfixe` similaire à la fonction `bissection` présentée à la section 5.4. Ses arguments sont d'ailleurs les mêmes, sinon que les points a et b sont remplacés par une valeur de départ x_0.

Étudiez attentivement les lignes 95-126 du fichier de script `resolution_equations.R` reproduit à la section 5.15. Nous aurons recours à la fonction `pointfixe` dans les exemples qui suivent.

Exemple 5.5. On répète l'exemple 5.4 à l'aide de la méthode du point fixe, soit trouver le taux d'intérêt i tel que $a_{\overline{10}|i} = 8{,}2218$. On a déjà déterminé à l'exemple 5.4 que le taux d'intérêt se trouve dans l'intervalle $[0{,}035, 0{,}040]$. On peut exprimer le problème sous forme de point fixe ainsi : trouver la valeur de i tel que

$$g(i) = \frac{1 - (1+i)^{-10}}{8{,}2218} = i.$$

Exécutez le code des lignes 128-147 du fichier de script `resolution_equations.R` pour calculer le point fixe de la fonction $g(i)$ ci-dessus.

Il faut 34 itérations pour obtenir la convergence avec un critère identique à celui utilisé avec la bissection. Pour ce type de problème, la méthode du point fixe n'est pas la plus efficace. □

Exemple 5.6. On répète l'exemple 5.3 à l'aide de la méthode du point fixe, à savoir : trouver la racine de $f(x) = x^3 + 4x^2 - 10 = 0$ dans l'intervalle $[1, 2]$.

Il y a plusieurs façons d'exprimer le problème de recherche de racine en termes de point fixe. Examinons cinq différentes fonctions toutes algébriquement équivalentes :

$$\begin{aligned}
g_1(x) &= x - x^3 - 4x^2 + 10 \\
g_2(x) &= \left(\frac{10}{x} - 4x\right)^{1/2} \\
g_3(x) &= \frac{1}{2}(10 - x^3)^{1/2} \\
g_4(x) &= \left(\frac{10}{4 + x}\right)^{1/2} \\
g_5(x) &= x - \frac{x^3 + 4x^2 - 10}{3x^2 + 8x}.
\end{aligned}$$

Il est laissé en exercice de vérifier que, dans chacun des cas, la racine de $f(x)$ est le point fixe de $g_i(x)$, $i = 1, \dots, 5$.

Les graphiques des cinq fonctions ci-dessus se trouvent à la figure 5.5. L'examen de ces graphiques permet de déterminer, avant de faire quelque calcul que ce soit, pour quelles fonctions la procédure itérative du point fixe convergera ainsi que les taux de convergence relatifs, le cas échéant.

Complétez cet exemple en exécutant les lignes 149-181 du fichier de script `resolution_equations.R`.

□

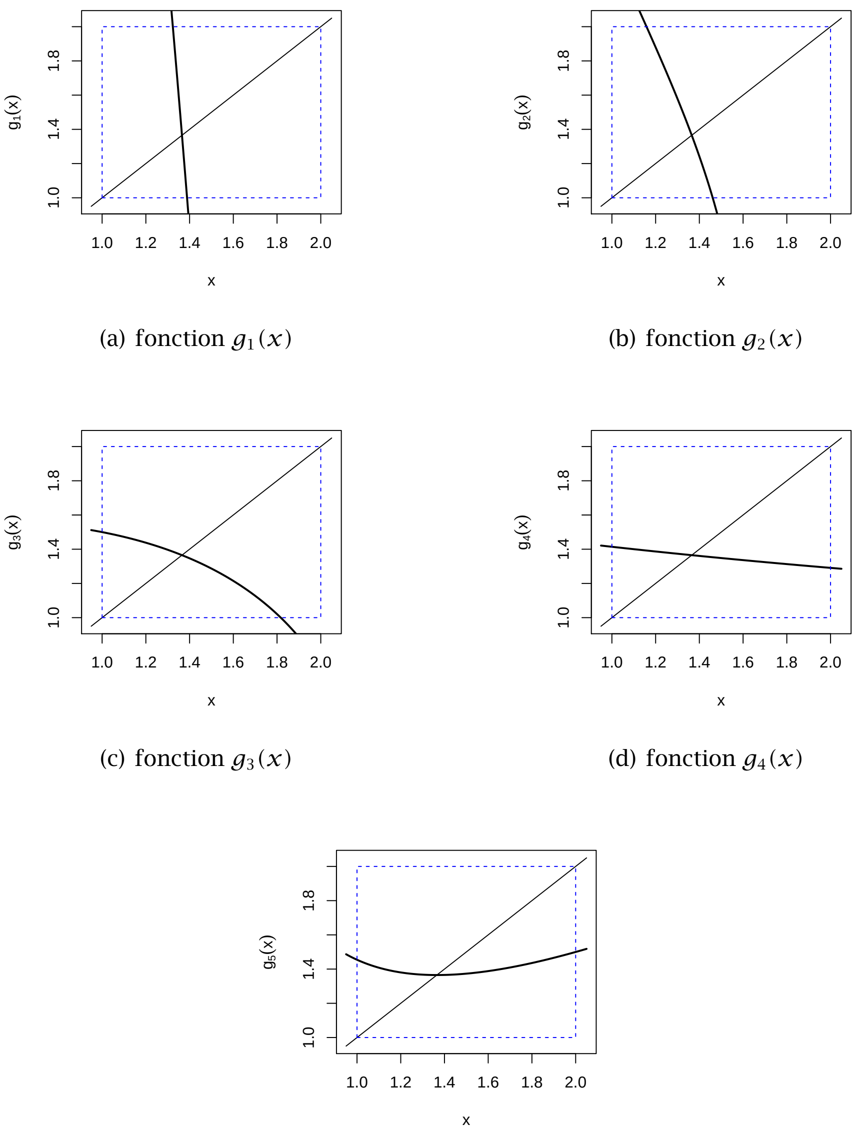
5.7 Indice pour le problème

Pour utiliser l'algorithme du point fixe 5.2, il faut exprimer notre problème de recherche du jour du mois où l'ensoleillement est de 9 heures sous la forme $g(x) = x$.

Nous pourrions simplement réécrire l'équation de départ $H(x) = 9$ sous la forme $H(x) - 9 + x = x$. Seulement, la pente de cette fonction est très près de 1 et la méthode du point fixe serait donc très lente.

Isolons donc plutôt une valeur de x dans l'équation de départ

$$e^{0{,}001312x+2{,}778538} - e^{-0{,}001431x+2{,}016340} = 9.$$



(a) fonction $g_1(x)$

(b) fonction $g_2(x)$

(c) fonction $g_3(x)$

(d) fonction $g_4(x)$

(e) fonction $g_5(x)$

FIG. 5.5 – Graphiques des cinq fonctions de l'exemple 5.6

En isolant le x dans la première exponentielle, cette équation se réécrit

$$x = \frac{\ln(9 + e^{-0{,}001431x+2{,}016340}) - 2{,}778538}{0{,}001312}.$$

Nous pourrions donc utiliser l'algorithme du point fixe avec la fonction

$$g(x) = \frac{\ln(9 + e^{-0{,}001431x+2{,}016340}) - 2{,}778538}{0{,}001312}.$$

On peut vérifier graphiquement que la pente de cette fonction est beaucoup plus faible que celle de $H(x) - 9 + x$.

5.8 Méthode de Newton–Raphson

La méthode de Newton–Raphson est l'une des plus populaires et puissantes méthodes numériques de résolution d'équations à une variable. Les algorithmes plus élaborés qui existent aujourd'hui sont d'ailleurs souvent basés sur cet algorithme.

L'étude de la méthode de Newton–Raphson nous ramène au problème de recherche de la racine x^* d'une fonction $f(x)$, c'est-à-dire la solution de $f(x) = 0$. Il y a plusieurs manières d'introduire cette méthode de résolution. Nous privilégierons une approche graphique.

Dans la méthode de Newton–Raphson, les essais successifs sont déterminés en utilisant les points où la tangente de la fonction f à l'essai précédent croise l'axe des abscisses ; voir la figure 5.6 pour une illustration.

Une première justification mathématique de la méthode va comme suit. Soit $\tilde{x}$ un point « près » de la racine x^* et tel que $f(\tilde{x}) \neq 0$. Comme on peut le voir à la figure 5.6, la tangente de f en $\tilde{x}$ croise l'abscisse en un point $\hat{x}$ « près » de x^*. Ainsi, le point $\hat{x}$ est tel que

$$\frac{f(\tilde{x}) - 0}{\tilde{x} - \hat{x}} = f'(\tilde{x}),$$

d'où

$$\begin{aligned} x^* &\simeq \hat{x} \\ &= \tilde{x} - \frac{f(\tilde{x})}{f'(\tilde{x})}. \end{aligned}$$

On peut également justifier la formule précédente à l'aide du développement de Taylor de $f(x)$ autour de $\tilde{x}$:

$$f(x) = f(\tilde{x}) + (x - \tilde{x})f'(\tilde{x}) + \frac{(x - \tilde{x})^2}{2} f''(\tilde{x}) + \dots$$

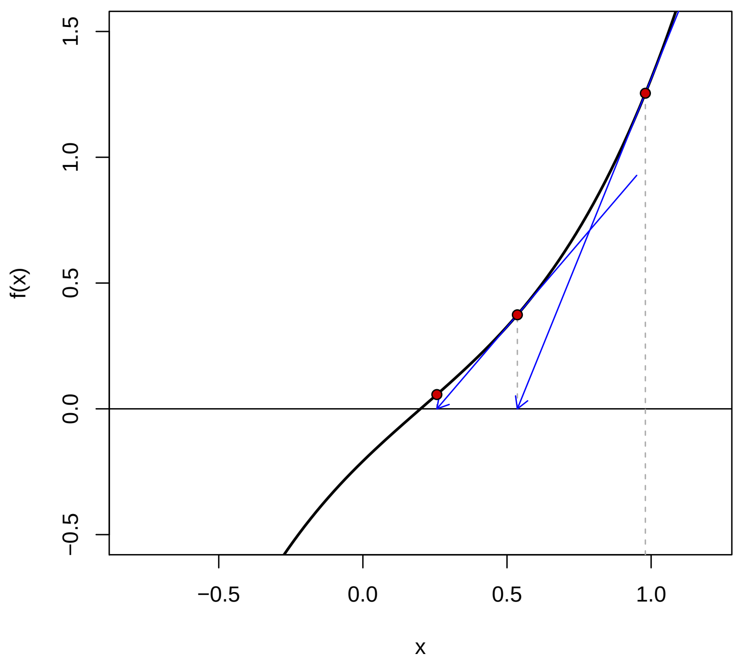


FIG. 5.6 – Illustration de la méthode de Newton–Raphson. Les points rouges identifient les essais successifs, le premier étant, ici, le point le plus à droite dans le graphique. Les flèches illustrent comment est déterminé l'essai suivant. On trace la tangente en un point ; le prochain essai est la valeur en abscisse où cette tangente croise l'axe.

Toujours en supposant que $\tilde{x}$ est près de x^*, on a

$$f(x^*) = 0 = f(\tilde{x}) + (x^* - \tilde{x})f'(\tilde{x}) + \frac{(x^* - \tilde{x})^2}{2} f''(\tilde{x}) + \dots,$$

où l'on peut considérer les termes de puissance 2 et plus dans le développement comme négligeables. On obtient alors l'approximation

$$x^* \simeq \tilde{x} - \frac{f(\tilde{x})}{f'(\tilde{x})}.$$

Tel qu'illustré à la figure 5.6, on peut répéter la procédure ci-dessus si le point $\hat{x}$ est trop éloigné de la racine x^*. On utilise alors $\hat{x}$ comme nouveau point où l'on calcule la tangente, et ainsi de suite jusqu'à l'obtention d'une « bonne » approximation.

Formellement, soit f une fonction différentiable sur $[a, b]$ et $x^* \in [a, b]$. Alors x^* peut être obtenu comme le point de convergence de la série $\{x_n\}$ définie par

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})},$$

avec x_0 une valeur de départ quelconque. L'algorithme 5.3 systématise cette procédure.

Algorithme 5.3 (Méthode de Newton–Raphson)**.** *Soit f une fonction continue et différentiable sur l'intervalle $[a, b]$. On cherche x^* tel que $f(x^*) = 0$ en un maximum de N_{max} itérations avec une valeur de départ x_0.*

1. *Poser $n = 1$.*
2. *Répéter les étapes suivantes.*
   a) *Poser $x_n = x_{n-1} - f(x_{n-1})/f'(x_{n-1})$.*
   b) *Si $|x_n - x_{n-1}|/|x_n| < \varepsilon$, alors poser $x^* = x_n$ et terminer.*
   c) *Si $n \geq N_{\text{max}}$, terminer sans convergence.*
   d) *Poser $n = n + 1$.*

L'examen attentif de la méthode de Newton–Raphson revèle qu'il s'agit en fait d'une procédure de point fixe avec

$$g(x) = x - \frac{f(x)}{f'(x)}.$$

La méthode de Newton–Raphson nous fournit donc la « bonne » fonction g à utiliser dans la méthode du point fixe pour trouver la racine de la fonction f.

On peut démontrer que si f est doublement différentiable sur $[a, b]$ et que $x^* \in [a, b]$ est tel que $f(x^*) = 0$ et $f'(x^*) \neq 0$, alors il existe un $\delta > 0$ tel que la série $\{x_n\}$ converge vers x^* pour tout $x_0 \in [x^* - \delta, x^* + \delta]$. Autrement dit :

- les hypothèses du théorème 5.1 sont satisfaites ;
- la méthode de Newton–Raphson fonctionne toujours avec un « bon » choix de valeur de départ x_0.

Le code informatique de la section 5.15 définit une fonction `nr` similaire à la fonction `pointfixe` présentée à la section 5.6, sauf qu'on y a ajouté un argument pour la dérivée de la fonction $f(x)$.

> Étudiez attentivement les lignes 184-221 du fichier de script `resolution_equations.R` reproduit à la section 5.15. Nous aurons recours à la fonction `nr` dans les exemples qui suivent.

Exemple 5.7. Reprenons l'exemple 5.5 de calcul d'un taux de rendement, cette fois à l'aide de la méthode de Newton–Raphson. Vous savez peut-être de votre cours de mathématiques financières que la méthode de Newton–Raphson est la plus efficace pour résoudre ce genre de problème. La figure 5.7 illustre pourquoi elle est plus efficace que la méthode du point fixe : la fonction

$$g(i) = i - \frac{f(i)}{f'(i)}$$

avec $f(i) = a_{\overline{n}|i} - k$ est beaucoup plus plate que la fonction

$$g(i) = \frac{1 - v^n}{k}$$

utilisée à l'exemple 5.5.

> Exécutez le code des lignes 223-244 du fichier de script `resolution_equations.R` pour calculer la racine de $f(i) = a_{\overline{n}|i} - k$.

La méthode de Newton–Raphson ne requiert donc que 2 itérations pour obtenir un résultat équivalent à celui obtenu par la méthode du point fixe avec la fonction g de l'exemple 5.5 en 34 itérations. □

Exemple 5.8. Revisitons l'exemple 5.3 où nous recherchions la racine de la fonction $f(x) = x^3 + 4x^2 - 10$ dans l'intervalle $[1, 2]$. On a $f'(x) = 3x^2 + 8x$, d'où $f'(x) \neq 0$ pour tout $x \in [1, 2]$. La méthode de Newton–Raphson dicte en définitive de rechercher le point fixe de la fonction

$$\begin{aligned} g(x) &= x - \frac{f(x)}{f'(x)} \\ &= x - \frac{x^3 + 4x^2 - 10}{3x^2 + 8x}. \end{aligned}$$

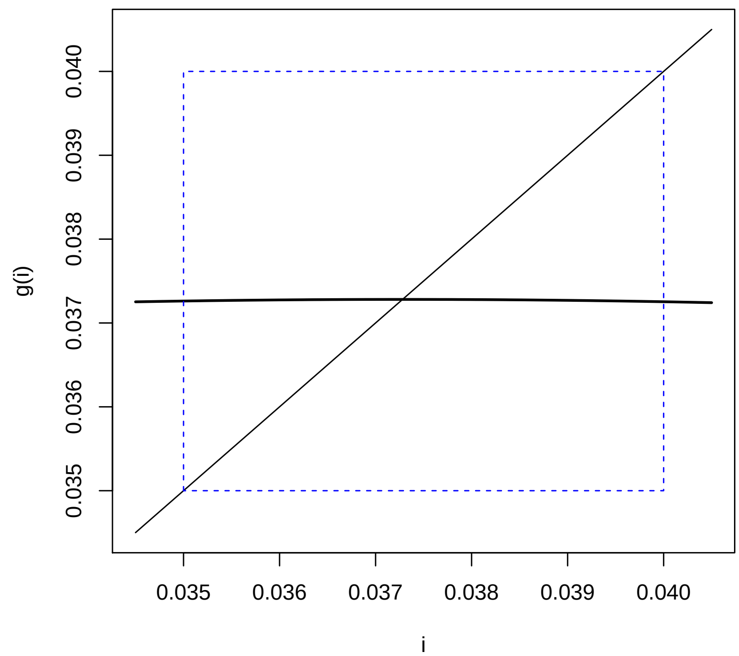


FIG. 5.7 – Fonction $g(i) = i - f(i)/f'(i)$ où $f(i) = (1-(1+i)^{-10})/i - 8{,}2218$ pour $0{,}035 \leq i \leq 0{,}040$ (ligne épaisse) et la droite $y = i$ (ligne mince)

Celle-ci est présentée à la figure 5.8. On constate que la procédure itérative de Newton–Raphson convergera rapidement vers une racine unique dans l'intervalle $[1, 2]$. En effet :

```
> f <- function(x) x^3 + 4*x^ 2 - 10
> fp <- function(x) 3*x^2 + 8 * x
> nr(f, fp, start = 1.5)
$root
[1] 1.36523

$nb.iter
[1] 3
```

Si vous êtes attentif, vous aurez remarqué que la fonction g utilisée ici

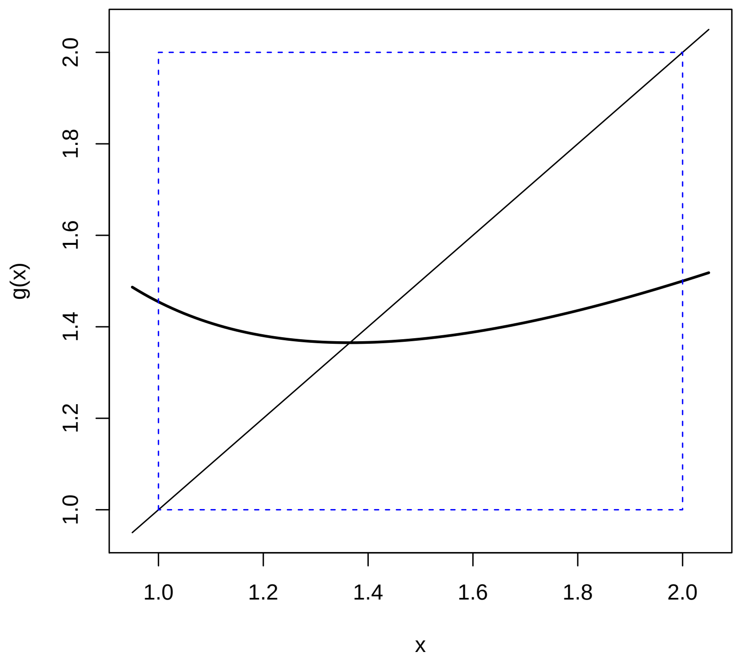


FIG. 5.8 – Fonction $g(x) = x - f(x)/f'(x)$, où $f(x) = x^3 + 4x^2 - 10$

était la fonction g_5 dans l'exemple 5.6, celle pour laquelle la convergence de la méthode du point fixe était la plus rapide. C'est précisément ce que mentionnait la remarque suivant l'algorithme 5.3, à la page 48. □

Exemple 5.9. Cet exemple illustre l'importance d'utiliser, pour certaines fonctions, une valeur de départ près de la racine. La fonction $f(x) = (4x - 7)/(x - 2)$ a une racine en $x = 1{,}75$ et une asymptote en $x = 2$, ce qui cause quelques soucis. (Voir la figure 5.9 pour un graphique de cette fonction.)

On peut tenter de trouver numériquement la racine de la fonction $f(x)$ avec l'algorithme de Newton–Raphson en utilisant

$$f(x) = \frac{4x - 7}{x - 2}$$

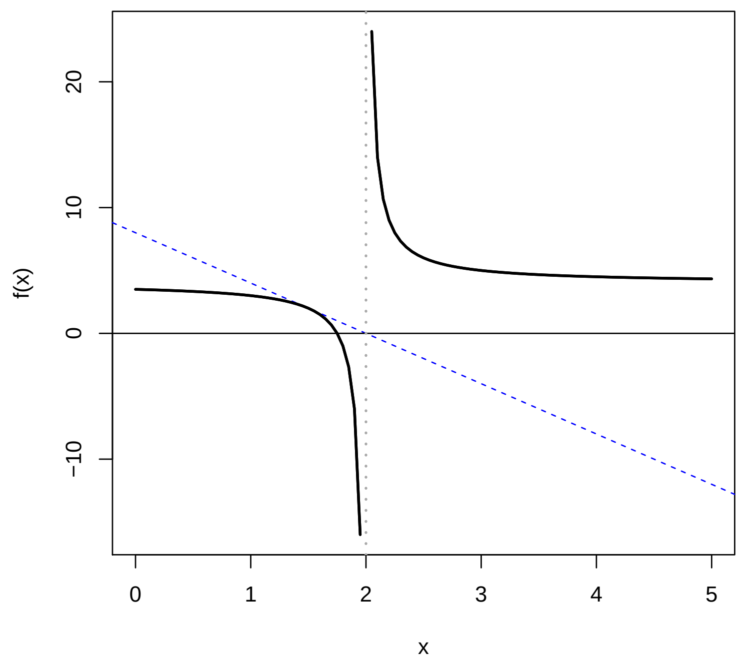


FIG. 5.9 – Fonction $f(x) = (4x-7)/(x-2)$ (ligne épaisse), asymptote (ligne pointillée) et tangente en $x = 1{,}5$ (ligne traitillée)

et

$$f'(x) = -\frac{1}{(x-2)^2},$$

ou encore directement par l'algorithme du point fixe avec

$$\begin{aligned} g(x) &= x - \frac{f(x)}{f'(x)} \\ &= 4x^2 - 14x + 14. \end{aligned}$$

Le graphique de la fonction $g(x)$ ci-dessus se trouve à la figure 5.10. On notera toutefois que le point fixe en $x = 2$ n'est pas une solution acceptable dans le problème d'origine.

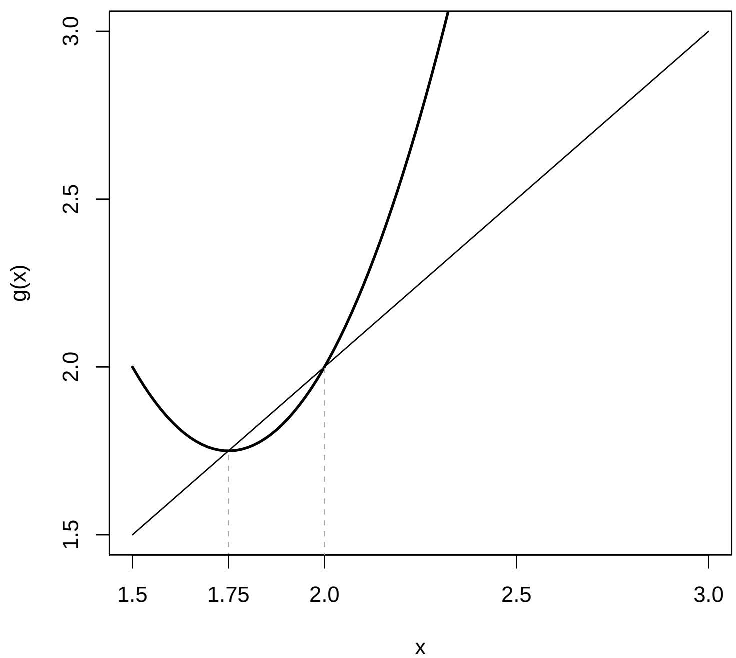


FIG. 5.10 – Fonction $g(x) = 4x^2 - 14x + 14$

On constate également dans cette dernière figure que les hypothèses pour l'existence et l'unicité d'un point fixe — et, donc, pour la convergence de la méthode de Newton–Raphson — ne sont pas rencontrées. La valeur de départ utilisée aura donc un impact sur la réponse obtenue.

Complétez cet exemple en exécutant les lignes 246–279 du fichier de script `resolution_equations.R`.

□

Le principal inconvénient de la méthode de Newton–Raphson demeure le fait de devoir connaître la dérivée de la fonction f. Dans les cas où celle-ci s'avère difficile ou inefficace à calculer, on peut utiliser la *méthode de la sécante*. Avec cette méthode, plutôt que d'utiliser la tangente en un essai pour déterminer la valeur de l'essai suivant, on a recours à la sécante entre les deux essais précédents ; voir la figure 5.11. Les valeurs des essais successifs sont alors données par :

$$x_n = x_{n-1} - \frac{f(x_{n-1})(x_{n-1} - x_{n-2})}{f(x_{n-1}) - f(x_{n-2})},$$

avec x_0, x_1 des valeurs de départ.

5.9 Indice pour le problème

Nous revenons au problème de départ exprimé sous la forme $f(x) = H(x) - 9 = 0$. La fonction des heures d'ensoleillement H étant simple à dériver, la méthode de Newton–Raphson est simple à appliquer. En effet,

$$\begin{aligned} f(x) &= H(x) - 9 \\ &= e^{0{,}001312x+2{,}778538} - e^{-0{,}001431x+2{,}016340} - 9, \end{aligned}$$

d'où

$$f'(x) = 0{,}001312e^{0{,}001312x+2{,}778538} + 0{,}001431e^{-0{,}001431x+2{,}016340}.$$

Il ne reste qu'à appliquer l'algorithme 5.3 avec les fonctions $f(x)$ et $f'(x)$ ainsi qu'une valeur de départ $x_0 \in \{1, 2, \dots, 31\}$.

5.10 Fonctions d'optimisation dans Excel et R

Les méthodes de bissection, du point fixe, de Newton–Raphson et autres permettent de résoudre des équations à une variable de la forme $f(x) = 0$ ou $g(x) = x$. Il existe également des versions de ces méthodes pour les systèmes à plusieurs variables comme

$$\begin{aligned} f_1(x_1, x_2, x_3) &= 0 \\ f_2(x_1, x_2, x_3) &= 0 \\ f_3(x_1, x_2, x_3) &= 0. \end{aligned}$$

De tels systèmes d'équations surviennent plus souvent qu'autrement lors de l'optimisation d'une fonction. Par exemple, en recherchant le maximum

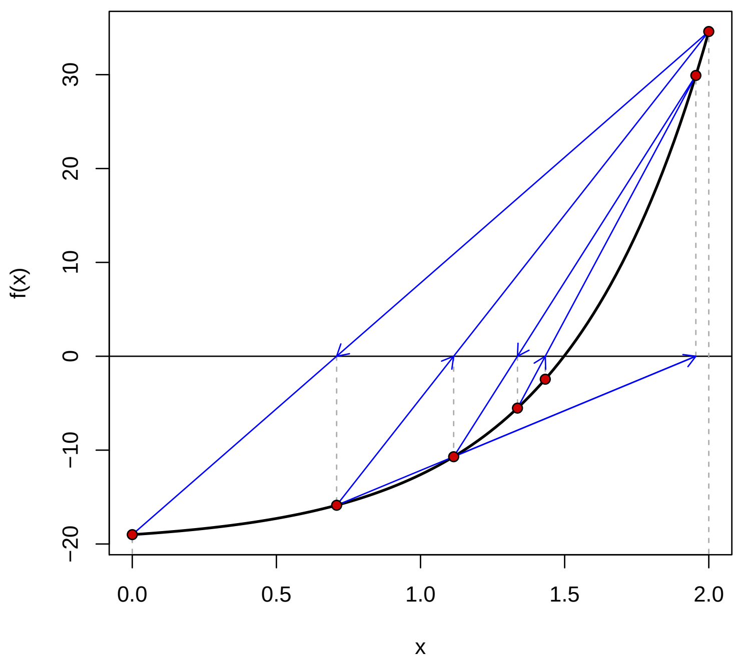


FIG. 5.11 – Illustration de la méthode de la sécante. Cette méthode requiert deux essais initiaux. Dans le graphique, il s'agit, dans l'ordre, du point à l'extrême gauche et celui à l'extrême droite. Les flèches illustrent comment est déterminé l'essai suivant. On trace la sécante entre deux points et le prochain essai est la valeur en abscisse où cette sécante croise l'axe.

ou le minimum d'une fonction $f(x, y)$, on souhaitera résoudre le système d'équations

$$\begin{aligned}\frac{\partial}{\partial x} f(x, y) &= 0 \\ \frac{\partial}{\partial y} f(x, y) &= 0.\end{aligned}$$

La grande majorité des suites logicielles comportent des outils d'optimisation de fonctions. Ce document passe en revue les fonctions disponibles dans Excel et R.

5.10.1 Solveur de Excel

Le principal outil d'optimisation utilisé dans Excel est le Solveur. La rubrique d'aide de cet outil est complète et on trouvera plusieurs exemples dans le classeur `SOLVAMP.XLS` livré avec Excel.

Le reste de cette section est dévolu à des fonctions d'optimisation de R.

5.10.2 Fonction `uniroot`

La fonction `uniroot` recherche la racine d'une fonction dans un intervalle. C'est donc la fonction de base pour trouver la solution (unique) de l'équation $f(x) = 0$ dans un intervalle déterminé.

5.10.3 Fonction `optimize`

La fonction `optimize` recherche le minimum local (par défaut) ou le maximum local d'une fonction dans un intervalle donné.

5.10.4 Fonction `nlm`

La fonction `nlm` minimise une fonction non linéaire sur un nombre arbitraire de paramètres.

5.10.5 Fonction `nlminb`

La fonction `nlminb` est similaire à `nlm`, sauf qu'elle permet de spécifier des bornes inférieure ou supérieure pour les paramètres. Attention, toutefois : les arguments de la fonction ne sont ni les mêmes, ni dans le même ordre que ceux de `nlm`.

5.10.6 Fonction `optim`

La fonction `optim` est l'outil d'optimisation tout usage de R. À ce titre, la fonction est souvent utilisée par d'autres fonctions. Elle permet de choisir parmi plusieurs algorithmes d'optimisation différents et, selon l'algorithme choisi, de fixer des seuils minimum ou maximum aux paramètres à optimiser.

5.10.7 `polyroot`

En terminant, un mot sur `polyroot()`, qui n'est pas à proprement parler une fonction d'optimisation, mais qui pourrait être utilisée dans ce contexte.

La fonction `polyroot` calcule toutes les racines (complexes) du polynôme $\sum_{i=0}^{n} a_i x^i$. Le premier argument est le vecteur des coefficients $a_0, a_1, \dots, a_n$, dans cet ordre.

> Les lignes 282-432 du fichier de script `resolution_equations.R` reproduit à la section 5.15 fournissent des exemples d'utilisation des fonctions R d'optimisation. Prenez le temps d'étudier ce code attentivement.

5.11 Indice pour le problème

La fonction `uniroot` permet de trouver la solution à l'équation $f(x) = 0$ dans un intervalle déterminé. Nous pourrons donc l'utiliser dans la solution en fin de chapitre pour confirmer les autres résultats.

5.12 Astuce Ripley

J'ai appris le truc suivant dans une publication de Brian Ripley — un important développeur de R — dans les forums de discussion de R. Il m'a été très utile à de nombreuses reprises, alors je le dissémine.

Une application statistique fréquente de l'optimisation est la maximisation numérique d'une fonction de vraisemblance ou, plus communément, la minimisation de la log-vraisemblance négative

$$-l(\theta) = -\sum_{i=1}^{n} \ln f(x_i; \theta).$$

Les fonctions d'optimisation sont d'ailleurs illustrées dans ce contexte dans le code informatique de la section 5.15.

En actuariat, nous utilisons principalement des lois de probabilité dont les paramètres sont strictement positifs. Or, en pratique, il n'est pas rare que les fonctions d'optimisation s'égarent dans les valeurs négatives des paramètres. La fonction de densité n'étant pas définie, la log-vraisemblance vaut alors `NaN` et cela peut faire complètement dérailler la procédure d'optimisation ou, à tout le moins, susciter des doutes sur la validité de la réponse.

Afin de remédier à ce problème, l'Astuce Ripley™ propose d'estimer non pas les paramètres de la loi eux-mêmes, mais plutôt leurs logarithmes. Si l'on définit $\tilde{\theta} = \ln \theta$, alors on peut écrire la fonction de log-vraisemblance ci-dessus sous la forme

$$-l(\tilde{\theta}) = -\sum_{i=1}^{n} \ln f(x_i; e^{\tilde{\theta}}).$$

Dès lors, $\tilde{\theta}$ (qui peut représenter un ou plusieurs paramètres) demeure valide sur tout l'axe des réels, ce qui permet d'éviter bien des soucis de nature numérique lors de la minimisation de $-l(\tilde{\theta})$.

Évidemment, le résultat de l'optimisation est l'estimateur du maximum de vraisemblance de $\tilde{\theta}$. Il faudra donc veiller à faire la transformation inverse pour retrouver l'estimateur de θ.

L'utilisation de l'Astuce est illustrée dans le code informatique du fichier de script `resolution_equations.R` reproduit à la section 5.15.

Une vidéo explicative vous aidera sans doute à mieux comprendre le fonctionnement de l'Astuce Ripley. Vous verrez, c'est tout simple, en fait.

5.13 Outils additionnels

Les paquetages disponibles sur CRAN fournissent plusieurs autres outils d'optimisation pour R. Pour un bon résumé des options disponibles, consulter la *CRAN Task View* consacrée à l'optimisation.

5.14 Solution du problème

Nous résolvons avec R notre problème consistant à trouver la valeur de x tel que $H(x) = 9$ avec les méthodes numériques étudiées dans ce chapitre. Les fonctions `bissection`, `pointfixe` et `nr` ci-dessous sont celles tirées du code informatique de la section 5.15.

Méthode de bissection : résolution de $H(x) - 9 = 0$.

```
> H <- function(x)
+     exp(0.001312 * x + 2.778538) -
+         exp(-0.001431 * x + 2.016340)
```

```
> bissection(FUN = function(x) H(x) - 9,
+            lower = 1, upper = 31)
$root
[1] 13.00008

$nb.iter
[1] 22
```

Méthode du point fixe : résolution de $g(x) = x$ avec la fonction g de l'encadré de la page 46.

```
> g <- function(x)
+     (log(9 + exp(-0.001431*x+2.016340)) - 2.778538) /
+     0.001312
> pointfixe(g, start = 15)
$fixed.point
[1] 13.00008

$nb.iter
[1] 18
```

Méthode de Newton-Raphson : résolution de $f(x) = H(x) - 9 = 0$ avec les fonctions f et f' de l'encadré de la page 54.

```
> f <- function(x) H(x) - 9
> fp <- function(x)
+     0.001312 * exp(0.001312 * x + 2.778538) +
+         0.001431 * exp(-0.001431 * x + 2.016340)
> nr(f, fp, start = 15)
$root
[1] 13.00008

$nb.iter
[1] 2
```

Vérification avec la fonction `uniroot`.

```
> uniroot(f, c(1, 31))$root
[1] 13.00008
```

5.15 Code informatique

Fichier d'accompagnement `resolution_equations.R`

```
###
### MÉTHODE DE BISSECTION
###

## Fonction pour trouver la solution de 'FUN'(x) = 0 par la
## méthode de bissection, où 'FUN' est une fonction continue
## sur l'intervalle ['lower', 'upper'].
bissection <- function(FUN, lower, upper, TOL = 1E-6,
                       MAX.ITER = 100, echo = FALSE)
{
    ## Cas triviaux où une borne est la solution
    if (identical(FUN(lower), 0))
        return(lower)
    if (identical(FUN(upper), 0))
        return(upper)

    ## Bornes de départ inadéquates
    if (FUN(lower) * FUN(upper) > 0)
        stop('FUN(lower) and FUN(upper) must be of opposite signs')

    x <- lower
    i <- 1

    repeat
    {
        xt <- x
        x <- (lower + upper)/2
        fx <- FUN(x)

        if (echo)
            print(c(lower, upper, x, fx))

        if (abs(x - xt)/abs(x) < TOL)
            break

        if (MAX.ITER < (i <- i + 1))
            stop('Maximum number of iterations reached
                  without convergence')

        if (FUN(lower) * fx > 0)
            lower <- x
```

```
        else
            upper <- x
    }
    list(root = x, nb.iter = i)
}

## RACINE D'UN POLYNÔME
##
## La fonction dont on cherche la racine
f <- function(x) x^3 + 4*x^2 - 10

## Un graphique de la fonction permet de vérifier qu'elle ne
## possède qu'une seule racine sur un intervalle donné ainsi
## que des valeurs de départ pour l'algorithme.
curve(f(x), xlim = c(0, 3), lwd = 2) # graphique sur [0, 3]
abline(h = 0)                        # axe des abscisses

## Résolution. Nous utilisons a = 1 et b = 2 comme valeurs de
## départ. Nous pourrions choisir des valeurs de départ plus
## près de la racine, ce qui accélèrerait la résolution.
bissection(f, lower = 1, upper = 2, TOL = 1E-5, echo = TRUE)

## VALEUR ACTUELLE
##
## La fonction dont on cherche la racine
f <- function(x) (1 - (1 + x)^(-10))/x - 8.2218

## Graphique de la fonction sur un intervalle "raisonnable"
## dans lequel on peut pressentir que se trouvera la réponse.
## Ici, on y va pour un taux d'intérêt se trouvant entre 1 %
## et 10 %.
curve(f(x), xlim = c(0.01, 0.10), lwd = 2)
abline(h = 0)

## Le graphique permet de déterminer à l'oeil que la racine se
## trouve entre 3 % et 4 %. Résolution avec la fonction
## 'bissection'.
bissection(f, lower = 0.03, upper = 0.04)

###
### MÉTHODE DU POINT FIXE
###

## Fonction pour trouver la solution de 'FUN'(x) = x par la
## méthode du point fixe à partir d'un essai initial 'start'.
```

```
pointfixe <- function(FUN, start, TOL = 1E-6, MAX.ITER = 100,
                      echo = FALSE)
{
    x <- start

    if (echo)
        expr <- expression(print(xt <- x))
    else
        expr <- expression(xt <- x)

    i <- 0

    repeat
    {
        eval(expr)

        x <- FUN(xt)

        if (abs(x - xt)/abs(x) < TOL)
            break

        if (MAX.ITER < (i <- i + 1))
            stop('Maximum number of iterations reached
                  without convergence')
    }
    list(fixed.point = x, nb.iter = i)
}

## VALEUR ACTUELLE (bis)
##
## La fonction dont on cherche le point fixe.
g <- function(x) (1 - (1 + x)^(-10))/8.2218

## Un graphique de la fonction permet de vérifier que les
## conditions pour qu'il existe un point fixe unique dans
## l'intervalle [0,035, 0,040] sont satisfaites. Cependant, la
## forme de la courbe (pente près de 1) nous indique que la
## convergence sera relativement lente.
f <- function(x) x          # pour tracer la droite y = x
lim <- c(0.035, 0.040)      # intervalle [a, b]
curve(g, xlim = lim, ylim = lim, lwd = 2,
      xlab = "i", ylab = "g(i)")
curve(f, add = TRUE)
polygon(rep(lim, each = 2), c(lim, rev(lim)),
        lty = "dashed", border = "blue")
```

```
## Résolution avec la fonction 'pointfixe'.
pointfixe(g, start = 0.0375, echo = TRUE)

## POLYNÔME (bis)
##
## Nous cherchons la racine de f(x) = x^3 + 4 * x^2 - 10 en
## exprimant le problème sous forme de point fixe. Les cinq
## fonctions ci-dessous sont toutes algébriquement
## équivalentes, c'est-à-dire que g(x) = x lorsque f(x) = 0.
g1 <- function(x) x - x^3 - 4 * x^2 + 10
g2 <- function(x) sqrt(10/x - 4*x)
g3 <- function(x) sqrt(10 - x^3)/2
g4 <- function(x) sqrt(10/(4 + x))
g5 <- function(x) x - (x^3 + 4*x^2 - 10)/(3*x^2 + 8*x)

## Si les fonctions sont algébriquement équivalentes, elles ne
## le sont pas numériquement devant la méthode du point fixe.
## Ainsi, avec la fonction 'g1', la procédure diverge.
## (Remarque: la fonction 'poinfixe' ne prévoit pas de message
## d'erreur pour ce cas. Qu'ajouteriez-vous à la fonction?)
pointfixe(g1, 1.5)
pointfixe(g1, 1.5, echo = TRUE) # plus évident ainsi

## Avec la fonction 'g2', la procédure s'arrête lorsqu'il faut
## calculer la racine carrée d'un nombre négatif.
pointfixe(g2, 1.5)

## Avec les trois autres fonctions, la méthode du point fixe
## est extrêmement rapide et précise. Une analyse rapide des
## graphiques fournis dans le chapitre nous permettrait de
## déterminer avec quelle fonction la convergence serait la
## plus rapide. En effet, c'est la fonction 'g5' qui a la
## pente la plus faible près de son point fixe.
pointfixe(g3, 1.5)
pointfixe(g4, 1.5)
pointfixe(g5, 1.5)

###
### MÉTHODE DE NEWTON-RAPHSON
###

## Fonction pour trouver la solution de 'FUN'(x) = x par la
## méthode de Newton-Raphson à partir de sa dérivée 'FUNp' et
## d'un essai initial 'start'.
```

```
##
## On ajoute une amélioration par rapport aux fonctions
## 'bissection' et 'pointfixe', soit la possibilité de passer
## des arguments additionnels aux fonctions 'FUN' et 'FUNp'
## via l'argument '...'
nr <- function(FUN, FUNp, start, TOL = 1E-6,
               MAX.ITER = 100, echo = FALSE, ...)
{
    x <- start

    if (echo)
        expr <- expression(print(xt <- x))
    else
        expr <- expression(xt <- x)

    i <- 0

    repeat
    {
        eval(expr)

        x <- xt - FUN(xt, ...)/FUNp(xt, ...)

        if (abs(x - xt)/abs(x) < TOL)
            break

        if (MAX.ITER < (i <- i + 1))
            stop('Maximum number of iterations reached
                  without convergence')
    }
    list(root = x, nb.iter = i)
}

## VALEUR ACTUELLE (ter)
##
## Nous effectuons de nouveau le calcul d'un taux de
## rendement, cette fois avec la méthode de Newton-Raphson. La
## convergence devrait être plus rapide qu'avec la méthode du
## point fixe puisque la fonction g(i) = i - f(i)/f'(i) est
## plus plate que celle utilisée avec la méthode du point
## fixe. Premièrement, la fonction dont on cherche la racine.
f <- function(i) (1 - (1 + i)^(-10))/i - 8.2218

## Sa dérivée.
fp <- function(i) (10 * i * (1 + i)^(-11) + (1 + i)^(-10) - 1)/i^2
```

```r

## Résolution avec la fonction 'nr'.
nr(f, fp, start = 0.0375, echo = TRUE)

## À noter que si la fonction g est définie adéquatement, nous
## pouvons de manière tout aussi équivalente utiliser la
## fonction 'pointfixe' pour effectuer les itérations de la
## méthode de Newton-Raphson.
g <- function(i) i - f(i)/fp(i)
pointfixe(g, 0.0375, echo = TRUE)

## CAS PATHOLOGIQUE
##
## Les fonctions f(x), f'(x) et g(x) définies dans le texte de
## l'exemple.
f <- function(x) ifelse(x == 2, NA, (4 * x - 7)/(x - 2))
fp <- function(x) ifelse(x == 2, NA, -1/(x - 2)^2)
g <- function(x) 4 * x^2 - 14 * x + 14

## Vérifions que, étant donné la forme des fonctions 'f' et
## 'g', la valeur de départ utilisée dans les méthodes de
## Newton-Raphson ou du point fixe a un impact sur la réponse
## obtenue. Avec une valeur de départ 'start' = 1,625 < 1,75,
## la convergence se fait vers la bonne racine.
nr(f, fp, 1.625, echo = TRUE)
pointfixe(g, 1.625)

## Nous pouvons vérifier sur le graphique de la fonction 'g'
## qu'avec une valeur de départ entre 1,75 et 2, la procédure
## itérative convergera aussi vers la bonne réponse.
nr(f, fp, 1.875, echo = TRUE)
pointfixe(g, 1.875, echo=TRUE)

## La tangente en x = 1,5 tracée sur le graphique de la
## fonction 'f' montre que 'start' = 1,5 constitue un mauvais
## choix car g(1,5) = 2. La procédure itérative converge donc
## vers une valeur non admissible.
nr(f, fp, 1.5)            # division par 0
pointfixe(g, 1.5)         # point fixe non admissible

## Pour toute valeur de départ supérieure à 2, la procédure
## itérative diverge. On peut vérifier ce fait dans les
## graphiques de 'f' et de 'g'.
nr(f, fp, 3, echo = TRUE)
pointfixe(g, 3, echo = TRUE)
```

```

###
### FONCTIONS D'OPTIMISATION DE R
###

## FONCTION 'uniroot'
##
## La fonction 'uniroot' recherche la racine d'une fonction
## 'f' dans un intervalle spécifié soit comme une paire de
## valeurs dans un argument 'interval', soit via des arguments
## 'lower' et 'upper'.
##
## On calcule la solution de l'équation x - 2^(-x) = 0 dans
## l'intervalle [0, 1].
f <- function(x) x - 2^(-x)      # fonction
uniroot(f, c(0, 1))              # appel simple
uniroot(f, lower = 0, upper = 1) # équivalent

## On peut aussi utiliser 'uniroot' avec une fonction anonyme.
uniroot(function(x) x - 2^(-x), lower = 0, upper = 1)

## FONCTION 'optimize'
##
## On cherche le maximum local de la densité d'une loi bêta
## dans l'intervalle (0, 1), son domaine de définition. (Ce
## problème est facile à résoudre explicitement.)
##
## Les arguments de 'optimize' sont essentiellement les mêmes
## que ceux de 'uniroot'. Ici, on utilise aussi l'argument
## '...' pour passer les paramètres de la loi bêta à 'dbeta'.
##
## Par défaut, la fonction recherche un minimum. Il faut donc
## lui indiquer de rechercher plutôt un maximum.
optimize(dbeta, interval = c(0, 1), maximum = TRUE,
         shape1 = 3, shape2 = 2)

## On pourrait aussi avoir recours à une fonction auxiliaire.
## Moins élégant et moins flexible.
f <- function(x) dbeta(x, 3, 2)
optimize(f, lower = 0, upper = 1, maximum = TRUE)

## FONCTION 'nlm'
##
## Pour la suite, nous allons donner des exemples
## d'utilisation des fonctions d'optimisation dans un contexte
```

```
## d'estimation des paramètres d'une loi gamma par la méthode
## du maximum de vraisemblance.
##
## On commence par se donner un échantillon aléatoire de la
## loi. Évidemment, pour ce faire nous devons connaître les
## paramètres de la loi. C'est un exemple fictif.
set.seed(1)                 # toujours le même échantillon
x <- rgamma(10, 5, 2)

## Les estimateurs du maximum de vraisemblance des paramètres
## 'shape' et 'rate' de la loi gamma sont les valeurs qui
## maximisent la fonction de vraisemblance
##
##     prod(dgamma(x, shape, rate))
##
## ou, de manière équivalente, qui minimisent la fonction de
## log-vraisemblance négative
##
##     -sum(log(dgamma(x, shape, rate))).
##
## On remarquera au passage que les fonctions de calcul de
## densités de lois de probabilité dans R ont un argument
## 'log' qui, lorsque TRUE, retourne la valeur du logarithme
## (naturel) de la densité de manière plus précise qu'en
## prenant le logarithme après coup. Ainsi, pour faire le
## calcul ci-dessus, on optera plutôt, pour l'expression
##
##     -sum(dgamma(x, shape, rate, log = TRUE))
##
## La fonction 'nlm' suppose que la fonction à optimiser
## passée en premier argument a elle-même comme premier
## argument le vecteur 'p' des paramètres à optimiser. Le
## second argument de 'nlm' est un vecteur de valeurs de
## départ, une pour chaque paramètre.
##
## Ainsi, pour trouver les estimateurs du maximum de
## vraisemblance avec la fonction 'nlm' pour l'échantillon
## ci-dessus, on doit d'abord définir une fonction auxiliaire
## conforme aux attentes de 'nlm' pour calculer la fonction de
## log-vraisemblance (à un signe près).
f <- function(p, x) -sum(dgamma(x, p[1], p[2], log = TRUE))

## L'appel de 'nlm' est ensuite tout simple. Remarquer comment
## on passe notre échantillon aléatoire (contenu dans l'objet
## 'x') comme second argument à 'f' via l'argument '...' de
```

```
## 'nlm'. Le fait que l'argument de 'f' et l'objet contenant
## les valeurs portent le même nom est sans importance. R sait
## faire la différence entre l'un et l'autre.
nlm(f, c(1, 1), x = x)

## === ASTUCE RIPLEY ===
## L'optimisation ci-dessus a généré des avertissements? C'est
## parce que la fonction d'optimisation s'est égarée dans les
## valeurs négatives, alors que les paramètres d'une gamma
## sont strictement positifs. Cela arrive souvent en pratique
## et cela peut faire complètement dérailler la procédure
## d'optimisation (c'est-à-dire: pas de convergence).
##
## L'Astuce Ripley consiste à remédier à ce problème en
## estimant plutôt les logarithmes des paramètres. Pour ce
## faire, il s'agit de réécrire la log-vraisemblance comme une
## fonction du logarithme des paramètres, mais de la calculer
## avec les véritables paramètres.
f2 <- function(logp, x)
{
    p <- exp(logp)          # retour aux paramètres originaux
    -sum(dgamma(x, p[1], p[2], log = TRUE))
}
nlm(f2, c(0, 0), x = x)

## Les valeurs obtenues ci-dessus sont toutefois les
## estimateurs des logarithmes des paramètres de la loi gamma.
## On retrouve les estimateurs des paramètres en prenant
## l'exponentielle des réponses.
exp(nlm(f2, c(0, 0), x = x)$estimate)
## ====================

## FONCTION 'nlminb'
##
## L'utilisation de la fonction 'nlminb' peut s'avérer
## intéressante dans notre contexte puisque l'on sait que les
## paramètres d'une loi gamma sont strictement positifs.
nlminb(c(1, 1), f, x = x, lower = 0, upper = Inf)

### FONCTION 'optim'
##
## La fonction 'optim' est très puissante, mais requiert aussi
## une bonne dose de prudence. Ses principaux arguments sont:
##
##  par: un vecteur contenant les valeurs initiales des
```

```
##       paramètres;
##   fn: la fonction à minimiser. Le premier argument de fn
##       doit être le vecteur des paramètres.
##
## Comme pour les autres fonctions étudiées ci-dessus, on peut
## passer des arguments à 'fn' (les données, par exemple) par
## le biais de l'argument '...' de 'optim'.
optim(c(1, 1), f, x = x)

## FONCTION 'polyroot'
##
## Racines du polynôme x^3 + 4 x^2 - 10. Les réponses sont
## données sous forme de nombre complexe. Utiliser les
## fonctions 'Re' et 'Im' pour extraire les parties réelles et
## imaginaires des nombres, respectivement.
polyroot(c(-10, 0, 4, 1))     # racines
Re(polyroot(c(-10, 0, 4, 1))) # parties réelles
Im(polyroot(c(-10, 0, 4, 1))) # parties imaginaires
```

5.16 Exercices

5.1 En vous basant sur les fonctions `bissection`, `pointfixe` et `nr` présentées dans le code de la section 5.15, écrire une fonction R pour effectuer les calculs de l'algorithme de la sécante. Outre les arguments communs à toutes les fonctions que sont le niveau de tolérance ε, le nombre maximal d'itérations $N_{\max}$ et une valeur booléenne spécifiant si les valeurs successives des itérations doivent être affichées à l'écran, la fonction doit compter les arguments $f(x)$, x_0 et x_1.

5.2 Trouver la solution des équations suivantes par les méthodes de bissection, de Newton-Raphson et de la sécante.

a) $x^3 - 2x^2 - 5 = 0$ pour $1 \leq x \leq 4$

b) $x^3 + 3x^2 - 1 = 0$ pour $-4 \leq x \leq 0$

c) $x - 2^{-x} = 0$ pour $0 \leq x \leq 1$

d) $e^x + 2^{-x} + 2\cos x - 6 = 0$ pour $1 \leq x \leq 2$

e) $e^x - x^2 + 3x - 2 = 0$ pour $0 \leq x \leq 1$

5.3 Déterminer la valeur numérique de $\sqrt{2}$ à l'aide de la méthode de bissection dans l'intervalle $[0, 2]$ avec dix itérations. Comparer avec la vraie valeur.

5.4 Soit $\{x_n\}$ une suite définie par

$$x_n = \sum_{k=1}^{n} \frac{1}{k}.$$

Démontrer que $\lim_{n\to\infty}(x_n - x_{n-1}) = 0$, mais que la suite diverge néanmoins. Ceci illustre que l'erreur absolue peut être un mauvais critère d'arrêt dans les méthodes numériques.

5.5 Soit la fonction $g(x) = 2^{-x}$ sur l'intervalle $[0, 1]$.

a) Vérifier si les hypothèses du théorème 5.1 quant à l'existence et l'unicité d'un point fixe dans $[0, 1]$ sont satisfaites.

b) Déterminer graphiquement l'existence et l'unicité d'un point fixe de $g(x)$ dans $[0, 1]$ puis, le cas échéant, calculer ce point fixe.

5.6 Vérifier que les cinq fonctions $g_1, \dots, g_5$ de l'exemple 5.6 ont toutes un point fixe en x^* lorsque $f(x^*) = 0$, où $f(x) = x^3 + 4x^2 - 10$.

5.7 Soit $g(x) = 4x^2 - 14x + 14$. Pour quels intervalles de valeurs de départ la procédure de point fixe converge-t-elle et diverge-t-elle ?

5.8 Les trois fonctions ci-dessous sont toutes des candidates pour faire l'approximation, par la méthode du point fixe, de $\sqrt[3]{21}$:

$$\begin{aligned} g_1(x) &= \frac{20x + 21x^{-2}}{21} \\ g_2(x) &= x - \frac{x^3 - 21}{3x^2} \\ g_3(x) &= x - \frac{x^4 - 21x}{x^2 - 21}. \end{aligned}$$

Classer ces fonctions en ordre décroissant de vitesse de convergence de l'algorithme du point fixe. *Astuce* : comparer les valeurs des dérivées autour du point fixe.

5.9 Démontrer, à l'aide du théorème du point fixe, que la fonction $g(x) = 2^{-x}$ possède un point fixe unique dans l'intervalle $[\frac{1}{3}, 1]$. Calculer par la suite ce point fixe à l'aide de la fonction `pointfixe`.

5.10 Vérifier graphiquement que le théorème 5.1 demeure valide si la condition $|g'(x)| \leq k < 1$ est remplacée par $g'(x) \leq k < 1$.

5.11 Utiliser la méthode de Newton–Raphson pour trouver le point sur la courbe $y = x^2$ le plus près du point $(1, 0)$. *Astuce* : minimiser la distance entre le point $(1, 0)$ et le point (x, x^2).

5.12 Le taux de rendement interne d'une série de flux financiers $\{CF_t\}$ est le taux i tel que

$$\sum_{t=0}^{n} \frac{CF_t}{(1+i)^t} = 0.$$

Écrire une fonction R permettant de calculer le taux de rendement interne d'une série de flux financiers quelconque à l'aide de la méthode de Newton–Raphson. Les arguments de la fonction sont un vecteur `CF` et un scalaire `erreur.max`. Au moins un élément de `CF` doit être négatif pour représenter une sortie de fonds. Dans tous les cas, utiliser $i = 0{,}05$ comme valeur de départ.

5.13 a) Composer une fonction R pour estimer par la méthode du maximum de vraisemblance le paramètre θ d'une loi gamma de paramètre de forme $\alpha = 3$ et de paramètre d'échelle $\theta = 1/\lambda$ — de moyenne 3θ, donc — à partir d'un échantillon aléatoire $x_1, \dots, x_n$. *Astuce* : maximiser la fonction de log-vraisemblance plutôt que la fonction de vraisemblance.

b) Simuler 20 observations d'une loi gamma avec paramètre de forme $\alpha = 3$ et moyenne 3 000, puis estimer le paramètre d'échelle θ par le maximum de vraisemblance.

5.14 À l'aide de la méthode du point fixe, trouver le taux d'intérêt i tel que

$$a^{(12)}_{\overline{10}|i} = \frac{1-(1+i)^{-10}}{i^{(12)}} = 8,$$

où

$$\left(1 + \frac{i^{(12)}}{12}\right)^{12} = 1 + i.$$

Comparer les résultats obtenus avec la méthode de Newton–Raphson.

5.15 Trouver la solution des équations suivantes à l'aide des fonctions R appropriées.

a) $x^3 - 2x^2 - 5 = 0$ pour $1 \leq x \leq 4$

b) $x^3 + 3x^2 - 1 = 0$ pour $-4 \leq x \leq 0$

c) $x - 2^{-x} = 0$ pour $0 \leq x \leq 1$

d) $e^x + 2^{-x} + 2\cos x - 6 = 0$ pour $1 \leq x \leq 2$

e) $e^x - x^2 + 3x - 2 = 0$ pour $0 \leq x \leq 1$

5.16 La fonction de densité de probabilité et la fonction de répartition de la loi de Pareto de paramètres α et λ sont, respectivement,

$$f(x) = \frac{\alpha\lambda^\alpha}{(x+\lambda)^{\alpha+1}}$$

et

$$F(x) = 1 - \left(\frac{\lambda}{x+\lambda}\right)^\alpha.$$

Calculer les estimateurs du maximum de vraisemblance des paramètres de la Pareto à partir d'un échantillon aléatoire obtenu par simulation avec la commande

```
> x <- lambda * (runif(100)^(-1/alpha) - 1)
```

pour des valeurs de `alpha` et `lambda` choisies.

Réponses

5.7 Converge pour $x_0 \in [1{,}5, 2]$; diverge pour $x_0 \leq 1{,}5$ et $x_0 > 2$

5.8 g_2, g_1 ; g_3 ne converge pas

5.11 0,589755

5.14 0,0470806

6 Intégration numérique

Objectifs du chapitre

- Avoir une connaissance générale de ce qu'est un polynôme d'interpolation de Lagrange et comment il peut servir dans le contexte de l'intégration numérique.
- Calculer la valeur approximative d'une intégrale définie à l'aide des méthodes du point milieu, du trapèze, de Simpson et de Simpson 3/8.
- Développer les formules d'approximation composées de chacune des méthodes d'intégration numériques présentées dans le chapitre.

Il n'est pas rare de devoir calculer l'intégrale d'une fonction qui n'admet pas de primitive sous forme explicite ou dont la primitive est très difficile à calculer. Dans de tels cas, l'intégration numérique permet d'obtenir une approximation — parfois excellente — de l'intégrale recherchée.

Les méthodes d'approximation qui seront étudiées dans ce chapitre reposent toutes sur le remplacement de la fonction à intégrer par une « bonne » approximation sur un intervalle donné. Cette approximation devra évidemment être simple à intégrer, autrement aucun gain n'est réalisé. Or, quelles fonctions sont simples à intégrer sinon les polynômes ?

6.1 Énoncé du problème

Au chapitre 3, nous avons tiré profit de la méthode Monte Carlo pour calculer l'intégrale de Gauss sur tout son domaine sans passer par les coordonnées polaires. Nous reprenons ce contexte ici.

Cette fois, nous cherchons à calculer l'intégrale

$$I = \int_{-1{,}645}^{1{,}645} \frac{1}{\sqrt{2\pi}}\, e^{-x^2/2}\, dx.$$

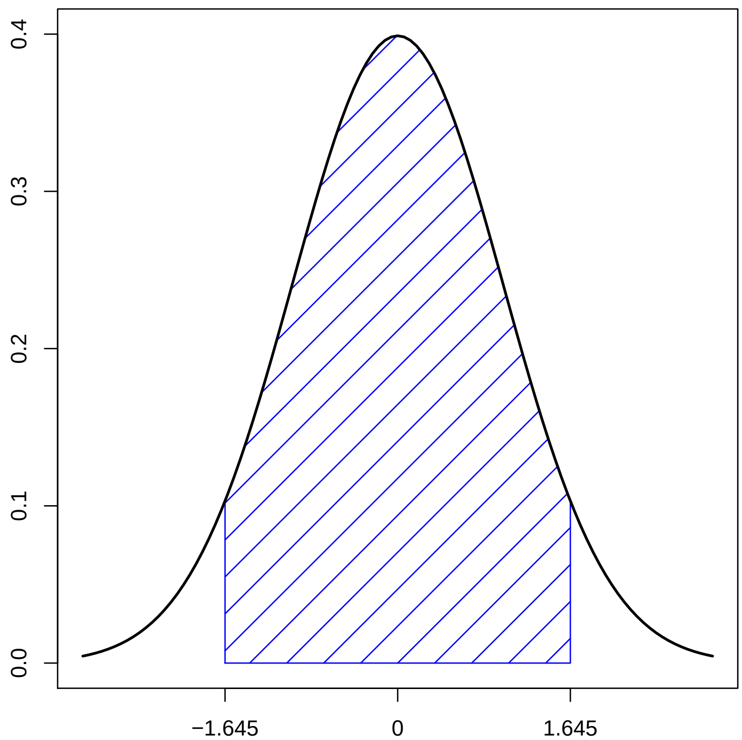


FIG. 6.1 – Aire à calculer avec l'intégrale I

Il s'agit bien sûr d'un résultat connu : nous pouvons reconnaître l'expression de la probabilité sous la densité d'une loi normale centrée réduite entre ses 5e et 95e centiles ; voir la figure 6.1. Par conséquent, nous savons que la valeur de l'intégrale est 0,90. Cela nous donnera un point de référence.

6.2 Polynômes d'interpolation de Lagrange

Un résultat connu en analyse mathématique sous le nom de Théorème d'approximation de Weierstrass établit que l'on peut faire l'approximation de toute fonction continue par un polynôme de degré suffisant. Formellement, soit

$$P_n(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} + a_n x^n$$

un polynôme de degré n. Si f est une fonction continue sur un intervalle $[a, b]$, alors pour tout $\varepsilon > 0$ il existe un polynôme $P(x)$ tel que

$$|f(x) - P(x)| < \varepsilon$$

pour tout $x \in [a, b]$.

La notion d'approximation d'une fonction par un polynôme est beaucoup utilisée en analyse numérique puisque les polynômes sont simples à dériver et à intégrer, et que leurs dérivées et intégrales sont elles-mêmes des polynômes.

Un type de polynôme d'approximation que vous connaissez déjà est le polynôme (ou développement) de Taylor autour d'un point x_0. Cependant, les polynômes de Taylor sont conçus pour être précis autour de x_0 et non sur tout un intervalle. Dans le contexte de l'intégration numérique, où nous voudrons remplacer la fonction à intégrer par un polynôme, nous aurons besoin d'une bonne approximation sur tout le domaine d'intégration.

C'est là qu'interviennent les polynômes d'interpolation de Lagrange. Soit $x_0, x_1, \dots, x_n$ un ensemble de $n + 1$ points distincts et f une fonction qui passe par ces points (ou *nœuds*). Le polynôme d'interpolation de Lagrange de degré n de la fonction f est

$$P_n(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + \cdots + f(x_n)L_n(x) = \sum_{k=0}^{n} f(x_k)L_k(x),$$

où

$$\begin{aligned} L_k(x) &= \frac{(x - x_0)(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0)(x_k - x_1) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)} \\ &= \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{(x - x_i)}{(x_k - x_i)}. \end{aligned}$$

Remarquer que le terme $(x - x_k)$ n'apparait pas au numérateur de la fonction $L_k(x)$.

On peut démontrer (voir, par exemple, Burden et Faires, 2011) que le polynôme $P(x)$ peut être aussi près que l'on veut de la fonction $f(x)$.

Exemple 6.1. Le polynôme d'interpolation de Lagrange de premier degré de la fonction f est

$$\begin{aligned} P_1(x) &= f(x_0)L_0(x) + f(x_1)L_1(x) \\ &= f(x_0)\frac{(x - x_1)}{(x_0 - x_1)} + f(x_1)\frac{(x - x_0)}{(x_1 - x_0)}. \end{aligned}$$

Le polynôme de second degré est, quant à lui :

$$\begin{aligned}P_2(x) &= f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x)\\ &= f(x_0)\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + f(x_1)\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}\\ &\quad + f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}.\end{aligned}$$

Supposons que $f(x) = 1/x$ et que les polynômes d'interpolation passent par les nœuds $x_0 = 2$, $x_1 = 2{,}75$ et $x_2 = 4$. On a alors

$$\begin{aligned}P_1(x) &= \frac{1}{2}\frac{(x-2{,}75)}{(2-2{,}75)} + \frac{1}{2{,}75}\frac{(x-2)}{(2{,}75-2)}\\ &= -\frac{2}{3}(x-2{,}75) + \frac{16}{33}(x-2)\\ &= -\frac{6}{33}\,x + \frac{171}{198}\end{aligned}$$

et

$$\begin{aligned}P_2(x) &= \frac{1}{2}\frac{(x-2{,}75)(x-4)}{(2-2{,}75)(2-4)} + \frac{1}{2{,}75}\frac{(x-2)(x-4)}{(2{,}75-2)(2{,}75-4)}\\ &\quad + \frac{1}{4}\frac{(x-2)(x-2{,}75)}{(4-2)(4-2{,}75)}\\ &= \frac{1}{3}(x-2{,}75)(x-4) - \frac{64}{165}(x-2)(x-4) + \frac{1}{10}(x-2)(x-2{,}75)\\ &= \frac{1}{22}\,x^2 - \frac{35}{88}\,x + \frac{49}{44}.\end{aligned}$$

Des approximations de $f(3) = 1/3$ sont donc $P_1(3) \approx 0{,}31818$ et $P_2(3) \approx 0{,}32955$. □

La fonction `poly.calc` du paquetage **polynom** permet de calculer le polynôme d'interpolation de Lagrange de degré n dans R. Étudiez et exécutez les lignes 15-38 du fichier de script `integration_numerique.R` reproduit à la section 5.15 pour vérifier les calculs de l'exemple 6.1.

6.3 Principes généraux d'intégration numérique

Supposons que l'on cherche à calculer $\int_a^b f(x)\,dx$. Toutes les méthodes d'intégration numérique reposent, en premier lieu, sur le découpage du domaine (a,b) en n intervalles. Cela permet d'évaluer l'intégrale comme une somme d'intégrales sur chacun de ces intervalles :

$$\int_a^b f(x)\,dx = \sum_{j=0}^{n-1} \int_{x_j}^{x_{j+1}} f(x)\,dx.$$

Les diverses méthodes d'intégration numérique diffèrent par la suite essentiellement par l'approximation de la fonction $f(x)$ sur l'intervalle (x_j, x_{j+1}) utilisée afin de rendre l'intégrale $\int_{x_j}^{x_{j+1}} f(x)\,dx$ simple à calculer.

La figure 6.2 illustre les procédures d'approximation de quatre méthodes courantes d'intégration numérique.

Les formules d'intégration numérique présentées dans la suite découlent de la procédure suivante : le domaine d'intégration (a,b) est découpé en n intervalles de longueur égale. Chacun de ces intervalles est à son tour divisé en m sous-intervalles de longueur égale, pour un total de nm points. On a alors

$$\begin{aligned}\int_a^b f(x)\,dx &= \int_{x_0}^{x_m} f(x)\,dx + \int_{x_m}^{x_{2m}} f(x)\,dx + \cdots + \int_{x_{(n-1)m}}^{x_{nm}} f(x)\,dx \\ &= \sum_{j=0}^{n-1} \int_{x_{jm}}^{x_{(j+1)m}} f(x)\,dx,\end{aligned}$$

où $x_0 = a$ et $x_{nm} = b$. L'approximation numérique se trouve dans l'évaluation de l'intégrale du côté droit de la dernière équation. Pour toutes les méthodes étudiées dans ce chapitre sauf la méthode du point milieu, la fonction f sera remplacée par un polynôme d'interpolation de Lagrange.

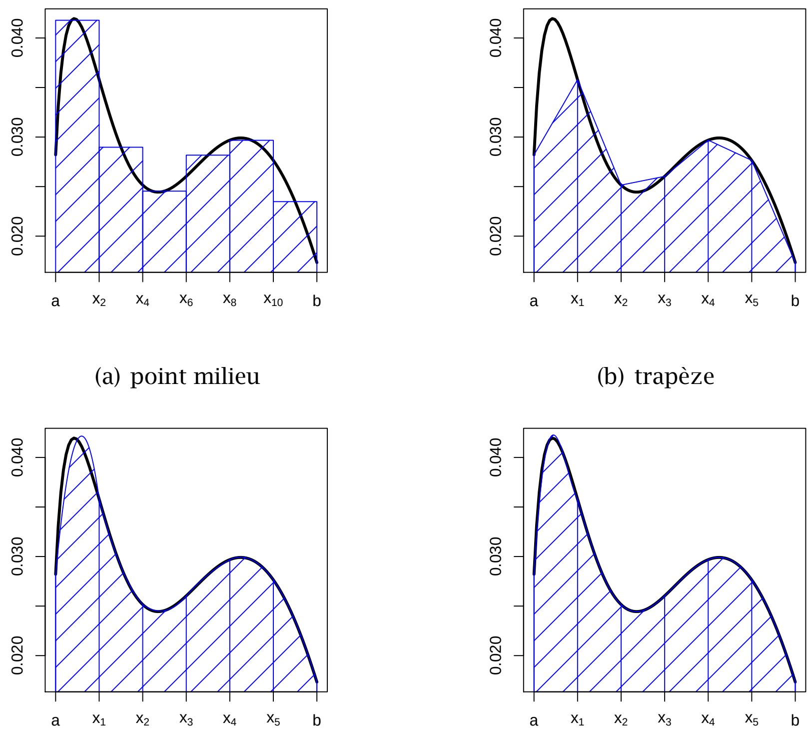
Afin de ne pas alourdir inutilement la notation, les formules d'approximation de l'intégrale

$$\int_{x_{jm}}^{x_{(j+1)m}} f(x)\,dx$$

seront présentées pour le cas $j = 0$ seulement dans les sections suivantes.

6.4 Méthode du point milieu

La méthode du point milieu est la plus simple et la plus intuitive méthode d'intégration numérique. Les intervalles sont divisés en $m = 2$ parties et



(a) point milieu

(b) trapèze

(c) Simpson

(d) Simpson 3/8

FIG. 6.2 – Procédures d'approximation de quatre méthodes d'intégration numérique

la valeur de la fonction f sur l'intervalle (x_0, x_2) est estimée par $f(x_1)$ (figure 6.3(a)). Ainsi, on a l'approximation

$$\int_{x_0}^{x_2} f(x)\,dx \approx 2hf(x_1), \tag{6.1}$$

où $h = x_2 - x_1 = x_1 - x_0$.

La formule composée pour l'approximation par la méthode du point milieu de $\int_a^b f(x)\,dx$ avec n intervalles est

$$\int_a^b f(x)\,dx \approx 2h \sum_{j=0}^{n-1} f(x_{2j+1}) \tag{6.2}$$

où $h = (b-a)/(2n)$ et $x_j = a + jh$.

6.5 Méthode du trapèze

Les trois prochaines méthodes d'intégration numérique sont basées sur l'approximation de la fonction f sur un intervalle par un polynôme d'interpolation de Lagrange de degré m.

La méthode du trapèze utilise un polynôme du premier degré ($m = 1$) pour faire l'approximation de la valeur de $f(x)$ sur l'intervalle (x_0, x_1), ce qui est équivalent à une simple interpolation linéaire (figure 6.3(b)). On a donc

$$\int_{x_0}^{x_1} f(x)\,dx \approx \frac{h}{2}[f(x_0) + f(x_1)], \tag{6.3}$$

où $h = x_1 - x_0$. Contrairement à ce que peut laisser croire la figure 6.2, cette méthode est généralement plus précise que la méthode du point milieu.

La formule composée pour l'approximation de $\int_a^b f(x)\,dx$ par la méthode du trapèze avec n intervalles est

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left[f(a) + f(b) + 2\sum_{j=1}^{n-1} f(x_j)\right], \tag{6.4}$$

où $h = (b - a)/n$ et $x_j = a + jh$.

6.6 Méthode de Simpson

La méthode d'approximation de Simpson est la plus usuelle des méthodes d'intégration numérique, encore que pas nécessairement la plus précise. La fonction $f(x)$ est remplacée, sur l'intervalle (x_0, x_2), par un polynôme d'interpolation de Lagrange du second degré (figure 6.3(c)). On a donc $m = 2$ et on peut démontrer que

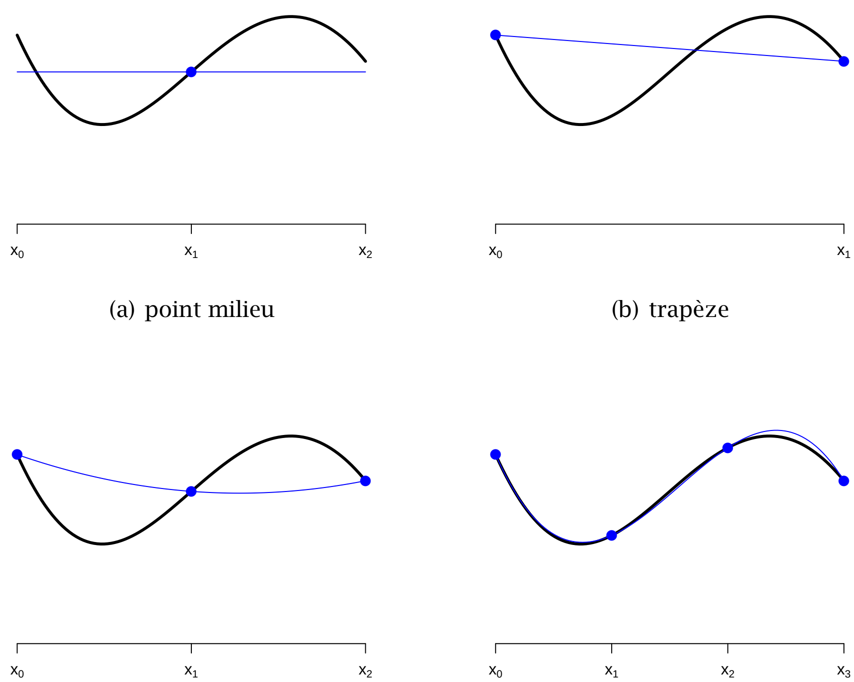
$$\int_{x_0}^{x_2} f(x)\,dx \approx \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)], \tag{6.5}$$

où $h = x_2 - x_1 = x_1 - x_0$. Cette méthode d'approximation numérique s'avère exacte pour les fonctions polynomiales de degré trois ou moins.

La formule composée pour l'approximation par la méthode de Simpson de $\int_a^b f(x)\,dx$ avec n intervalles est

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(a) + 2\sum_{j=1}^{n-1} f(x_{2j}) + 4\sum_{j=1}^{n} f(x_{2j-1}) + f(b)\right], \tag{6.6}$$

où $h = (b - a)/(2n)$ et $x_j = a + jh$.



(a) point milieu (b) trapèze

(c) Simpson (d) Simpson 3/8

FIG. 6.3 – Approximation de $f(x)$ sur un intervalle

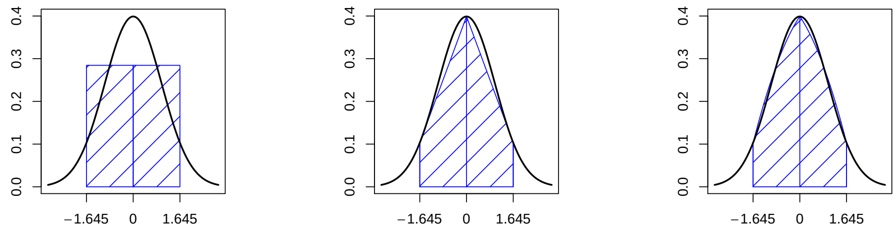
6.7 Méthode de Simpson 3/8

La méthode de Simpson 3/8 constitue une extension de la méthode de Simpson où la fonction f est remplacée par un polynôme d'interpolation de degré $m = 3$ (figure 6.3(d)). On peut alors démontrer que

$$\int_{x_0}^{x_3} f(x)\,dx \approx \frac{3h}{8}[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)], \tag{6.7}$$

où $h = x_3 - x_2 = x_2 - x_1 = x_1 - x_0$. La dérivation de la formule composée d'approximation de $\int_a^b f(x)\,dx$ avec n intervalles est laissée en exercice.

Consulter Burden et Faires (2011) pour de plus amples détails sur les polynômes d'interpolation de Lagrange et les méthodes d'intégration numérique présentées ci-dessus.



(a) point milieu (b) trapèze (c) Simpson

FIG. 6.4 – Comparaison des approximations de l'aire correspondant à l'intégrale I selon trois méthodes d'intégration numérique avec $n = 2$

Il n'y a pas de fonction dans la distribution de base de R pour le calcul d'intégrales avec précisément l'une ou l'autre des méthodes mentionnées dans ce chapitre. Cependant, la fonction `integrate` réalise à peu près la même chose avec un algorithme différent. Étudiez les lignes 41-53 du fichier de script `integration_numerique.R` reproduit à la section 6.9 en apprendre un peu plus sur cette fonction.

6.8 Solution du problème

Nous allons calculer l'intégrale

$$I = \int_{-1,645}^{1,645} \frac{1}{\sqrt{2\pi}} \, e^{-x^2/2} \, dx$$

à l'aide de trois méthodes d'intégration numérique : point milieu, trapèze et Simpson. Dans tous les cas, pour simplifier les calculs, nous allons utiliser $n = 2$ intervalles. Il va donc sans dire que l'approximation sera relativement grossière. La figure 6.4 fournit une comparaison graphique des trois approximations.

Dans tous les cas, nous avons $a = -1,645$, $b = 1,645$ et

$$f(x) = \phi(x) = \frac{1}{\sqrt{2\pi}} \, e^{-x^2/2}.$$

Tout d'abord, avec la méthode du point milieu, nous avons :

$$\begin{aligned} h &= \frac{b-a}{2n} = \frac{1{,}645 + 1{,}645}{4} = 0{,}8225 \\ x_j &= a + jh = -1{,}645 + 0{,}8225j \end{aligned}$$

et donc

$$\begin{aligned} I &\approx 2h \sum_{j=0}^{n-1} f(x_{2j+1}) \\ &= 2(0{,}8225)[f(x_1) + f(x_3)] \\ &= 1{,}645[f(-0{,}8225) + f(0{,}8225)] \\ &= 0{,}9358462. \end{aligned}$$

Avec la méthode du trapèze, maintenant :

$$\begin{aligned} h &= \frac{b-a}{n} = \frac{1{,}645 + 1{,}645}{2} = 1{,}645 \\ x_j &= a + jh = -1{,}645 + 1{,}645j, \end{aligned}$$

d'où

$$\begin{aligned} I &\approx \frac{h}{2}\left[f(a) + f(b) + 2\sum_{j=1}^{n-1} f(x_j)\right] \\ &= \frac{1{,}645}{2}[f(a) + f(b) + 2f(x_1)] \\ &= 0{,}8225[f(-1{,}645) + f(1{,}645) + 2f(0)] \\ &= 0{,}8258773. \end{aligned}$$

Enfin, avec la méthode de Simpson, nous avons :

$$\begin{aligned} h &= \frac{b-a}{2n} = \frac{1{,}645 + 1{,}645}{4} = 0{,}8225 \\ x_j &= a + jh = -1{,}645 + 0{,}8225j \end{aligned}$$

et

$$
\begin{aligned}
I &\approx \frac{h}{3}\left[f(a) + 2\sum_{j=1}^{n-1} f(x_{2j}) + 4\sum_{j=1}^{n} f(x_{2j-1}) + f(b)\right] \\
&= \frac{0{,}8225}{3}[f(a) + 2f(x_2) + 4(f(x_1) + f(x_3)) + f(b)] \\
&= \frac{0{,}8225}{3}[f(-1{,}645) + 2f(0) \\
&\quad + 4(f(-0{,}8225) + f(0{,}8225)) + f(1{,}645)] \\
&= 0{,}8991899.
\end{aligned}
$$

6.9 Code informatique

Fichier d'accompagnement `integration_numerique.R`

```r
###
### POLYNÔMES D'INTERPOLATION DE LAGRANGE
###

## La fonction 'poly.calc' du package 'polynom' permet de
## calculer le polynôme d'interpolation de Lagrange passant
## par un ensemble de points (x, y), où x et y sont des
## vecteurs de même longueur. Le package n'est pas livré avec
## R, il faut donc l'installer depuis CRAN, puis le charger en
## mémoire. [Décommenter la ligne ci-dessous pour installer
## (une seule fois!) le package.]
#install.packages("polynom", repos = "https://cran.ca.r-project.org")
library(polynom)

## Soit f(x) = 1/x. Calculons les coefficients des polynômes
## d'interpolation de Lagrange de degré 1 et 2 pour cette
## fonction.
x <- c(2, 2.75, 4)                  # noeuds de f
(P1 <- poly.calc(x[1:2], 1/x[1:2])) # polynôme de degré 1
(P2 <- poly.calc(x, 1/x))           # polynôme de degré 2

## Pour calculer la valeur des polynômes en un point, il faut
## utiliser la fonction 'predict'.
predict(P1, 3)             # tel qu'obtenu dans le texte
predict(P2, 3)             # idem

```

```
###
### MÉTHODES D'INTÉGRATION NUMÉRIQUE
###

## Pas de démonstration en R des méthodes étudiées dans le
## chapitre. Il est laissé en exercice d'écrire des fonctions
## 'pointmilieu', 'trapeze', 'simpson' et 'simpson38'.
## Cependant, il y a dans R une fonction 'integrate' qui
## permet d'intégrer numériquement une fonction 'f' entre des
## bornes 'lower' et 'upper'.
integrate(sin, 0, 2)              # intégrale de sin(x)
f <- function(x) x^2 * exp(-x)   # une autre fonction
integrate(f, 0, 1)                # intégrale sur [0, 1]
integrate(dnorm, -1.645, 1.645) # exemple du chapitre
```

6.10 Exercices

6.1 À partir des formules d'approximation de $\int_{x_0}^{x_m} f(x)\,dx$, $m = 1, 2, 3$, développer les formules composées pour les méthodes d'approximation suivantes.

a) Trapèze ($m = 1$)

b) Simpson ($m = 2$)

c) Simpson 3/8 ($m = 3$)

6.2 Évaluer numériquement les intégrales suivantes avec les méthodes du point milieu, du trapèze, de Simpson et de Simpson 3/8. Dans chaque cas, n'utiliser qu'un seul sous-intervalle, c'est-à-dire $n = 1$.

a) $\displaystyle\int_{0,5}^{1} x^4\,dx$

b) $\displaystyle\int_{0}^{0,5} \frac{2}{x-4}\,dx$

c) $\displaystyle\int_{1}^{1,5} x^2 \ln x\,dx$

d) $\displaystyle\int_{0}^{\pi/4} e^{3x} \sin 2x\,dx$

6.3 L'approximation de $\int_0^2 f(x)\,dx$ avec $n = 1$ donne 4 avec la méthode du trapèze et 2 avec la méthode de Simpson. Déterminer la valeur de $f(1)$.

Réponses

6.2 a) 0,158203 ; 0,265625 ; 0,194010 ; 0,193866

b) −0,266667 ; −0,267857 ; −0,267064 ; −0,267063

c) 0,174331 ; 0,228074 ; 0,192245 ; 0,192253

d) 1,803915 ; 4,143260 ; 2,583696 ; 2,585789

6.3 $\frac{1}{2}$

A Solutions des exercices

Chapitre 4

La notation x_b signifie que le nombre x est en base b. On omet généralement b pour les nombres en base 10.

4.1 L'algorithme de conversion des nombres décimaux en une base b se résume essentiellement à ceci pour la partie entière :

1. les chiffres du nombre en base b sont obtenus de droite à gauche en prenant le reste de divisions par b ;
2. on divise par b d'abord le nombre décimal d'origine, puis la partie entière de la division précédente, jusqu'à ce que celle-ci soit égale à 0.

On a donc les résultats suivants.

a) Conversion en base 6 :

$$\begin{aligned} 119 \div 6 &= 19 \text{ reste } 5 \\ 19 \div 6 &= 3 \text{ reste } 1 \\ 3 \div 6 &= 0 \text{ reste } 3, \end{aligned}$$

d'où $119 \equiv 315_6$.

Conversion en binaire :

$$\begin{aligned} 119 \div 2 &= 59 \text{ reste } 1 \\ 59 \div 2 &= 29 \text{ reste } 1 \\ 29 \div 2 &= 14 \text{ reste } 1 \\ 14 \div 2 &= 7 \text{ reste } 0 \\ 7 \div 2 &= 3 \text{ reste } 1 \\ 3 \div 2 &= 1 \text{ reste } 1 \\ 1 \div 2 &= 0 \text{ reste } 1, \end{aligned}$$

d'où $119 \equiv 1110111_2$.

b) Conversion en base 6 :

$$\begin{aligned} 343 \div 6 &= 57 \text{ reste } 1\\ 57 \div 6 &= 9 \text{ reste } 3\\ 9 \div 6 &= 1 \text{ reste } 3\\ 1 \div 6 &= 0 \text{ reste } 1, \end{aligned}$$

d'où $343 \equiv 1331_6$.

Conversion en binaire :

$$\begin{aligned} 343 \div 2 &= 171 \text{ reste } 1\\ 171 \div 2 &= 85 \text{ reste } 1\\ 85 \div 2 &= 42 \text{ reste } 1\\ 42 \div 2 &= 21 \text{ reste } 0\\ 21 \div 2 &= 10 \text{ reste } 1\\ 10 \div 2 &= 5 \text{ reste } 0\\ 5 \div 2 &= 2 \text{ reste } 1\\ 2 \div 2 &= 1 \text{ reste } 0\\ 1 \div 2 &= 0 \text{ reste } 1, \end{aligned}$$

d'où $343 \equiv 101010111_2$.

c) Conversion en base 6 :

$$\begin{aligned} 96 \div 6 &= 16 \text{ reste } 0\\ 16 \div 6 &= 2 \text{ reste } 4\\ 2 \div 6 &= 0 \text{ reste } 2, \end{aligned}$$

d'où $96 \equiv 240_6$.

Conversion en binaire :

$$\begin{aligned} 96 \div 2 &= 48 \text{ reste } 0\\ 48 \div 2 &= 24 \text{ reste } 0\\ 24 \div 2 &= 12 \text{ reste } 0\\ 12 \div 2 &= 6 \text{ reste } 0\\ 6 \div 2 &= 3 \text{ reste } 0\\ 3 \div 2 &= 1 \text{ reste } 1\\ 1 \div 2 &= 0 \text{ reste } 1, \end{aligned}$$

d'où $96 \equiv 1100000_2$.

d) Conversion en base 6 :

$$\begin{aligned} 43 \div 6 &= 7 \text{ reste } 1\\ 7 \div 6 &= 1 \text{ reste } 1\\ 1 \div 6 &= 0 \text{ reste } 1, \end{aligned}$$

d'où $43 \equiv 111_6$.

Conversion en binaire :

$$\begin{aligned} 43 \div 2 &= 21 \text{ reste } 1\\ 21 \div 2 &= 10 \text{ reste } 1\\ 10 \div 2 &= 5 \text{ reste } 0\\ 5 \div 2 &= 2 \text{ reste } 1\\ 2 \div 2 &= 1 \text{ reste } 0\\ 1 \div 2 &= 0 \text{ reste } 1, \end{aligned}$$

d'où $43 \equiv 101011_2$.

4.2 On fait les deux premières conversions à l'aide de la définition d'un nombre hexadécimal, puis les deux dernières à l'aide de l'algorithme de conversion des nombres en base b vers la base 10.

a) $\mathrm{A1B}_{16} = 10 \times 16^2 + 1 \times 16 + 11 = 2\,587$

b) $12\mathrm{A}_{16} = 1 \times 16^2 + 2 \times 16 + 10 = 298$

c) $\mathrm{B41}_{16} = (11 \times 16 + 4) \times 16 + 1 = 2\,881$

d) $\mathrm{BAFFE}_{16} = ((((11{\times}16{+}10){\times}16){+}15){\times}16){+}15){\times}16{+}14 = 765\,950$

4.3 La généralisation de l'algorithme de conversion des nombres en base b vers la base 10 à la conversion d'un nombre

$$x = x_{m-1}x_{m-2} \cdots x_1x_0$$

en base $[b_{m-1} \ \ldots \ b_0]$ vers la base 10 est la suivante (nombre entiers seulement) :

1. Poser $x = 0$.
2. Pour $i = m-1, m-2, \ldots, 0$, faire les étapes suivantes.
   i) Trouver d_i, le nombre décimal correspondant au symbole x_i.
   ii) Poser $x = xb_{i-1} + d_i$, avec $b_{-1} = 1$.

Cet algorithme permet de trouver les formules demandées.

a) On trouve la position de l'élément a_{ijk} dans l'ordre de la liste des éléments du tableau en convertissant le nombre $[i-1 \ \ j-1 \ \ k-1]$ de la base $[I \ \ J \ \ K]$ à la base 10, puis à additionnant 1. À l'aide de l'algorithme ci-dessus, on obtient

$$[((i-1) \times J + j - 1) \times K + k - 1] + 1 = k + K(j - 1 + J(i-1))$$

b) Dans l'ordre R, on convertit le nombre $[k-1 \ \ j-1 \ \ i-1]$ exprimé dans la base $[K \ J \ I]$ en base 10. On obtient alors

$$[((k-1) \times J + j - 1) \times I + i - 1] + 1 = i + I(j - 1 + J(k-1)).$$

4.4 a) Par la valeur du bit fort, on sait que $\boxed{1}\boxed{0000000}$ est un nombre négatif. Pour retrouver sa valeur absolue, on soustrait d'abord 1 (ce qui donne 01111111) et on inverse les bits : 10000000. Or $10000000_2 = 2^7 = 128$. Le nombre décimal correspondant à $\boxed{1}\boxed{0000000}$ en notation en complément à deux est donc -128.

b) Tout d'abord, $15_{10} = 1111_2$ et $5_{10} = 101_2$.

   i) Selon la notation « naturelle », on aurait pour 15 et -5 les représentations sur 7 bits suivantes, dans l'ordre :

$$\boxed{0}\,\boxed{0001111} \quad \text{et} \quad \boxed{1}\,\boxed{0000101}.$$

Or, en additionnant en binaire, on obtient

$$\begin{array}{r} 0\,0001111 \\ +\quad 1\,0000101 \\ \hline 1\,0010100 \end{array}$$

et $10010100_2 = -20_{10} \neq 15 + (-5)$.

ii) En notation en complément à deux, on a plutôt les représentations suivantes pour 15 et -5 :

$$\boxed{0}\,\boxed{0001111} \quad \text{et} \quad \boxed{1}\,\boxed{1111011}.$$

En additionnant en binaire et en ignorant le neuvième bit qui apparaîtrait à gauche suite à la retenue, on obtient :

$$\begin{array}{r} 0\,0001111 \\ +\quad 1\,1111011 \\ \hline 0\,0001010 \end{array}$$

Toujours en complément à deux, $0001010_2 = 10_{10} = 15 + (-5)$. On voit donc que l'addition en complément à deux donne directement le bon résultat.

c) Voir les tableaux A.1 et A.2. La troisième colonne des tableaux est obtenue par l'opération $-(2^n - x)$, où $n = 8$ ou $n = 16$ et x est la valeur dans la seconde colonne.

d) De la partie c), l'ensemble des nombres admissibles pour le type `int` est

$$\{-2^{15}, \dots, 2^{15} - 1\} = \{-32\,768, \dots, 32\,767\}.$$

Par analogie, on trouve que l'ensemble des entiers admissibles pour le type `long` est

$$\{-2^{31}, \dots, 2^{31} - 1\} = \{-2\,147\,483\,648, \dots, 2\,147\,483\,647\}.$$

(C'est le type de nombre qui était utilisé par YouTube pour comptabiliser le nombre de visionnements d'une vidéo jusqu'à ce que *Gangnam Style* excède 2 147 483 647. Le nombre de visionnements est désormais stocké sous forme d'entier sur 64 bits (`long long`), ce qui a fait passer le maximum à $2^{63} - 1 = 9\,223\,372\,036\,854\,775\,808$.)

4.5 Voir la section 4.5. Les calculs sont exactement les mêmes que pour les nombres en double précision.

4.6 Dans les égalités ci-dessous, le côté droit est en binaire.

Tab. A.1 – Séquences des nombres en notation en complément à deux sur 8 bits

Bits	Valeur brute	Valeur en complément à deux
0000 0000	0	0
0000 0001	1	1
0000 0010	2	2
0111 1110	126	126
0111 1111	127	127
1000 0000	128	−128
1000 0001	129	−127
1000 0010	130	−126
1111 1110	254	−2
1111 1111	255	−1

Tab. A.2 – Séquences des nombres en notation en complément à deux sur 16 bits

Bits	Valeur brute	Valeur en complément à deux
0000 0000 0000 0000	0	0
0000 0000 0000 0001	1	1
0000 0000 0000 0010	2	2
0111 1111 1111 1110	32 766	32 766
0111 1111 1111 1111	32 767	32 767
1000 0000 0000 0000	32 768	−32 768
1000 0000 0000 0001	32 769	−32 767
1000 0000 0000 0010	32 770	−32 766
1111 1111 1111 1110	65 534	−2
1111 1111 1111 1111	65 535	−1

a) Premièrement, $1234 \equiv 10011010010_2$. On a donc

$$\begin{aligned}-1\,234 &= (-1)^1 \times 2^{10} \times 1{,}001101001\\ &= (-1)^1 \times 2^{137-127} \times 1{,}0011010010.\end{aligned}$$

Or, puisque $137 \equiv 10001001_2$, on a la représentation en simple précision

$$\boxed{1}\,\boxed{10001001}\,\boxed{00110100100000000000000}$$

b) On a $55 \equiv 110111_2$, d'où

$$\begin{aligned}55 &= (-1)^0 \times 2^5 \times 1{,}10111\\ &= (-1)^0 \times 2^{132-127} \times 1{,}10111.\end{aligned}$$

Or, puisque $132 \equiv 10000100_2$, on a la représentation en simple précision

$$\boxed{0}\,\boxed{10000100}\,\boxed{10111000000000000000000}$$

c) On a $8\,191 \equiv 1111111111111_2$ et $139 \equiv 10001011$, d'où

$$\begin{aligned}8\,191 &= (-1)^0 \times 2^{12} \times 1{,}111111111111\\ &= (-1)^0 \times 2^{139-127} \times 1{,}111111111111\\ &= \boxed{0}\,\boxed{10001011}\,\boxed{11111111111100000000000}.\end{aligned}$$

d) On a $10 \equiv 1010_2$ et $130 \equiv 10000010$, d'où

$$\begin{aligned}-10 &= (-1)^1 \times 2^3 \times 1{,}010\\ &= (-1)^1 \times 2^{130-127} \times 1{,}010\\ &= \boxed{1}\,\boxed{10000010}\,\boxed{01000000000000000000000}.\end{aligned}$$

e) La représentation de $\frac{2}{3}$ en binaire est $0{,}101010\dots$. (La façon la plus simple d'obtenir ce résultat consiste à convertir $\frac{2}{3} \times 2^n$, où n est le nombre de bits souhaité après la virgule). Puisque $126 \equiv 1111110_2$, on a

$$\begin{aligned}\frac{2}{3} &= (-1)^0 \times 2^{-1} \times 1{,}01010101010101010101010\\ &= (-1)^0 \times 2^{126-127} \times 1{,}01010101010101010101010\\ &= \boxed{0}\,\boxed{01111110}\,\boxed{01010101010101010101010}.\end{aligned}$$

f) La représentation binaire de $\frac{1}{100}$ est infinie : 0,000000101000111101 … .
Puisque $120 \equiv 01111000_2$, on a

$$\begin{aligned}\frac{1}{100} &= (-1)^0 \times 2^{-7} \times 1{,}01000111101011100001010\\ &= (-1)^0 \times 2^{120-127} \times 1{,}01000111101011100001010\\ &= \boxed{0}\,\boxed{01111000}\,\boxed{01000111101011100001010}\end{aligned}$$

4.7 a) Puisque $111101_2 \equiv 61$, on a le nombre

$$\begin{aligned}(-1)^0 \times 2^{61-127} \times 1{,}100100001 &= (-1)^0 \times 2^{-66} \times 1{,}100100001\\ &= 2^{-66}(1 + 2^{-1} + 2^{-4} + 2^{-9})\\ &\equiv 2{,}120\,229\,346 \times 10^{-20}.\end{aligned}$$

b) Signe inversé par rapport à la partie a).

c) Puisque $10000100_2 = 2^7 + 2^2 \equiv 132$, on a le nombre

$$\begin{aligned}(-1)^0 \times 2^{132-127} \times 1{,}100100001 &= (-1)^0 \times 2^5 \times 1{,}100100001\\ &= 2^5(1 + 2^{-1} + 2^{-4} + 2^{-9})\\ &\equiv 50{,}062\,5.\end{aligned}$$

d) Signe inversé par rapport à la partie c).

4.8 a) Le nombre suivant est

$$\boxed{0}\,\boxed{00111101}\,\boxed{10010000100000000000001},$$

soit

$$2^{-66}(1 + 2^{-1} + 2^{-4} + 2^{-9} + 2^{-23}) \equiv 2{,}120\,229\,508 \times 10^{-20}.$$

Le nombre précédent est

$$\boxed{0}\,\boxed{00111101}\,\boxed{10010000011111111111111},$$

soit

$$2^{-66}(1 + 2^{-1} + 2^{-4} + 2^{-9} - 2^{-23}) \equiv 2{,}120\,229\,185 \times 10^{-20}.$$

c) Le nombre suivant est

$$\boxed{0}\,\boxed{10000100}\,\boxed{10010000100000000000001},$$

soit
$$2^5(1 + 2^{-1} + 2^{-4} + 2^{-9} + 2^{-23}) \equiv 50{,}062\,503\,815.$$

Le nombre précédent est

$$\boxed{0}\,\boxed{10000100}\,\boxed{10010000011111111111111},$$

soit
$$2^5(1 + 2^{-1} + 2^{-4} + 2^{-9} - 2^{-23}) \equiv 50{,}062\,496\,185.$$

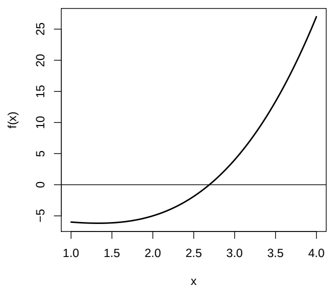
On remarque que les nombres sont beaucoup plus éloignés les uns des autres ici qu'en a).

Chapitre 5

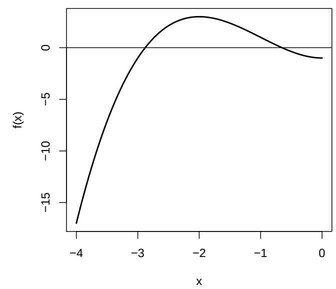
5.2 Les solutions suivantes utilisent nos fonctions de résolution d'équations à une variable. Les valeurs intermédiaires sont affichées pour montrer la convergence. La figure A.1 contient les graphiques des cinq fonctions pour les intervalles mentionnés dans l'énoncé.

a) La fonction n'a qu'une seule racine dans $[1, 4]$.

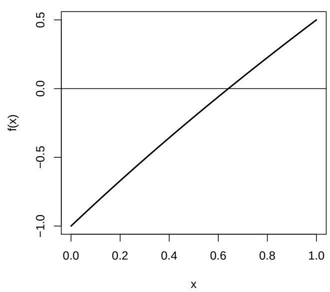
```
> f1 <- function(x) x^3 - 2 * x^2 - 5
> f1p <- function(x) 3 * x^2 - 4 * x
> bissection(f1, lower = 1, upper = 4, echo = TRUE)
[1]  1.000  4.000  2.500 -1.875
[1] 2.5000 4.0000 3.2500 8.2031
[1] 2.5000 3.2500 2.8750 2.2324
[1]  2.500000  2.875000  2.687500 -0.034424
[1] 2.6875 2.8750 2.7812 1.0432
[1] 2.68750 2.78125 2.73438 0.49078
[1] 2.68750 2.73438 2.71094 0.22481
[1] 2.687500 2.710938 2.699219 0.094355
[1] 2.687500 2.699219 2.693359 0.029757
[1]  2.6875000  2.6933594  2.6904297 -0.0023855
[1] 2.690430 2.693359 2.691895 0.013673
[1] 2.6904297 2.6918945 2.6911621 0.0056403
[1] 2.6904297 2.6911621 2.6907959 0.0016266
[1]  2.69042969  2.69079590  2.69061279 -0.00037968
[1] 2.6906128 2.6907959 2.6907043 0.0006234
[1] 2.69061279 2.69070435 2.69065857 0.00012185
[1]  2.69061279  2.69065857  2.69063568 -0.00012892
```



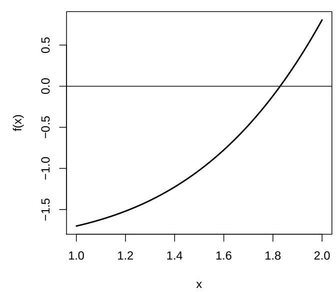
(a) $f(x) = x^3 - 2x^2 - 5$



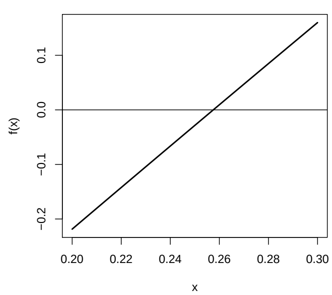
(b) $f(x) = x^3 + 3x^2 - 1$



(c) $f(x) = x - 2^{-x}$



(d) $f(x) = e^x + 2^{-x} + 2\cos x - 6$



(e) $f(x) = e^x - x^2 + 3x - 2$

FIG. A.1 – Fonctions de l'exercice 5.2

```
[1]  2.6906e+00  2.6907e+00  2.6906e+00 -3.5365e-06
[1] 2.6906e+00 2.6907e+00 2.6907e+00 5.9155e-05
[1] 2.6906e+00 2.6907e+00 2.6906e+00 2.7809e-05
[1] 2.6906e+00 2.6906e+00 2.6906e+00 1.2136e-05
$root
[1] 2.6906

$nb.iter
[1] 21
> nr(f1, f1p, start = 2.5, echo = TRUE)
[1] 2.5
[1] 2.7143
[1] 2.691
[1] 2.6906
$root
[1] 2.6906

$nb.iter
[1] 3
> secante(f1, start0 = 1, start1 = 4, echo = TRUE)
[1] 1.5455
[1] 1.9969
[1] 4.1051
[1] 2.2947
[1] 2.4787
[1] 2.7514
[1] 2.6831
[1] 2.6904
[1] 2.6906
$root
[1] 2.6906

$nb.iter
[1] 9
```

b) La fonction possède deux racines dans $[-4, 0]$. La convergence se fera vers l'une ou l'autre selon la position de la ou des valeurs de départ par rapport à l'extremum de la fonction dans l'intervalle. Ici, il s'agit d'un maximum $x = -2$. Ainsi, avec des valeurs de départ inférieures au maximum, on trouve la première racine :

```
> f2 <- function(x) x^3 + 3 * x^2 - 1
> f2p <- function(x) 3 * x^2 + 6 * x
> bissection(f2, lower = -3, upper = -2.8, TRUE)
$root
[1] -2.9

$nb.iter
[1] 1
> nr(f2, f2p, start = -3, TRUE)
$root
[1] -2.8889

$nb.iter
[1] 0
> secante(f2, start0 = -3, start1 = -2, TRUE)
$root
[1] -2.75

$nb.iter
[1] 0
```

Pour trouver la seconde racine, on utilise des valeurs de départ supérieures au maximum :

```
> bissection(f2, lower = -1, upper = 0.5, TRUE)
$root
[1] -0.625

$nb.iter
[1] 2
> nr(f2, f2p, start = -1, TRUE)
$root
[1] -0.66667

$nb.iter
[1] 0
> secante(f2, start0 = -2, start1 = -1, TRUE)
```

```
$root
[1] -0.63636

$nb.iter
[1] 1
```

On remarquera que les deux valeurs de départ de la méthode de la sécante n'ont pas à se trouver de part et d'autre de la racine.

c) La fonction n'a qu'une seule racine dans $[0, 1]$ et elle est légèrement supérieure à 0,6.

```
> f3 <- function(x) x - 2^(-x)
> f3p <- function(x) 1 + log(2) * 2^(-x)
> bissection(f3, lower = 0.6, upper = 0.65, TRUE)
$root
[1] 0.625

$nb.iter
[1] 1
> nr(f3, f3p, start = 0.6, TRUE)
$root
[1] 0.641

$nb.iter
[1] 0
> secante(f3, start0 = 0.6, start1 = 0.65, TRUE)
$root
[1] 0.64122

$nb.iter
[1] 0
```

d) La fonction n'a qu'une seule racine dans $[1, 2]$ et elle est légèrement supérieure à 1,8.

```
> f4 <- function(x)
+     exp(x) + 2^(-x) + 2 * cos(x) - 6
> f4p <- function(x)
+     exp(x) - 2^(-x) * log(2) - 2 * sin(x)
> bissection(f4, lower = 1.8, upper = 1.85, TRUE)
```

```
$root
[1] 1.825

$nb.iter
[1] 1
> nr(f4, f4p, start = 1.8, TRUE)
$root
[1] 1.8301

$nb.iter
[1] 0
> secante(f4, start0 = 1.8, start1 = 1.85, TRUE)
$root
[1] 1.8289

$nb.iter
[1] 0
```

e) Encore ici, la fonction n'a qu'une seule racine dans l'intervalle mentionné et elle se situe autour de 0,25.

```
> f5 <- function(x) exp(x) - x^2 + 3 * x - 2
> f5p <- function(x) exp(x) - 2 * x + 3
> bissection(f5, lower = 0.24, upper = 0.26, TRUE)
$root
[1] 0.25

$nb.iter
[1] 1
> nr(f5, f5p, start = 0.26, TRUE)
$root
[1] 0.25753

$nb.iter
[1] 0
> secante(f5, start0 = 0.24, start1 = 0.26, TRUE)
$root
[1] 0.25753
```

Tab. A.3 – Valeurs successives de la méthode de bissection pour l'exercice 5.3

n	a_n	b_n	x_n	$f(x_n)$
1	0	2	1	−1
2	1,00	2,00	1,50	0,25
3	1,0000	1,5000	1,2500	−0,4375
4	1,250000	1,500000	1,375000	−0,109375
5	1,37500000	1,50000000	1,43750000	0,06640625
6	1,37500000	1,43750000	1,40625000	−0,02246094
7	1,40625000	1,43750000	1,42187500	0,02172852
8	1,4062500000	1,4218750000	1,4140625000	−0,0004272461
9	1,41406250	1,42187500	1,41796875	0,01063538
10	1,41406250	1,41796875	1,41601562	0,00510025

```
$nb.iter
[1] 0
```

5.3 On trouve la racine de $f(x) = x^2 - 2$ dans l'intervalle $[0, 2]$ par la méthode de bissection avec un maximum de dix itérations. Le tableau A.3 contient les valeurs successives de a, b, $x = (a+b)/2$ et $f(x)$. On obtient donc une réponse de 1,41601562, alors que la vraie réponse est le nombre irrationnel $\sqrt{2} = 1{,}414214$.

5.4 On a que

$$\begin{aligned} x_n - x_{n-1} &= \sum_{k=1}^{n} \frac{1}{k} - \sum_{k=1}^{n-1} \frac{1}{k} \\ &= \frac{1}{n}, \end{aligned}$$

d'où $\lim_{n\to\infty}(x_n - x_{n-1}) = \lim_{n\to\infty} n^{-1} = 0$. Or, $\sum_{k=1}^{n} k^{-1}$ est la série harmonique qui est connue pour diverger. On peut, par exemple, justifier ceci par le fait que l'intégrale

$$\int_1^\infty \frac{1}{x}\, dx$$

diverge elle-même. Cet exercice illustre donc que le critère $|x_n - x_{n-1}| < \varepsilon$ peut être satisfait même pour une série divergente.

5.5 a) On considère la fonction $g(x) = 2^{-x}$ dans l'intervalle $[0, 1]$. En premier lieu, $1/2 \leq 2^{-x} \leq 1$ pour $0 \leq x \leq 1$, donc $g(x) \in [0, 1]$. De plus, $g'(x) = -2^{-x}(\ln 2)$, d'où

$$|g'(x)| = \frac{\ln 2}{2^x} \leq \ln 2 < 1, \quad 0 \leq x \leq 1.$$

Par le théorème 5.1, la fonction g possède un point fixe unique dans l'intervalle $[0, 1]$.

b) Le graphique de la fonction g se trouve à la figure A.2. On constate que la fonction a effectivement un point fixe unique dans l'intervalle $[0, 1]$ et que celui-ci se trouve près de $x = 0{,}6$. On peut donc utiliser cette valeur comme point de départ de l'algorithme du point fixe :

```
> pointfixe(function(x) 2^(-x), start = 0.6)
$fixed.point
[1] 0.64119

$nb.iter
[1] 15
```

On remarque que la réponse est indépendante de la valeur de départ :

```
> pointfixe(function(x) 2^(-x), start = 0)
$fixed.point
[1] 0.64119

$nb.iter
[1] 18
> pointfixe(function(x) 2^(-x), start = 1)
$fixed.point
[1] 0.64119

$nb.iter
[1] 17
> pointfixe(function(x) 2^(-x), start = 2)
$fixed.point
[1] 0.64119

$nb.iter
[1] 18
```

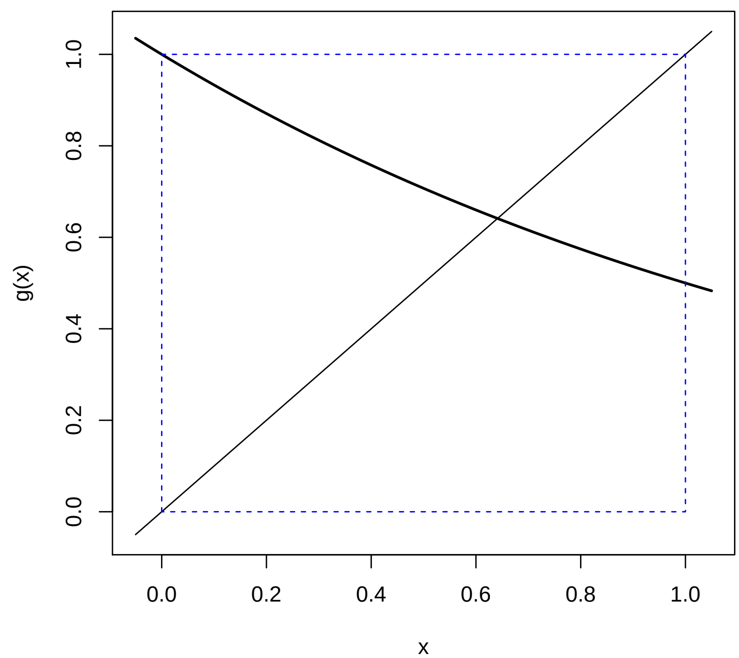


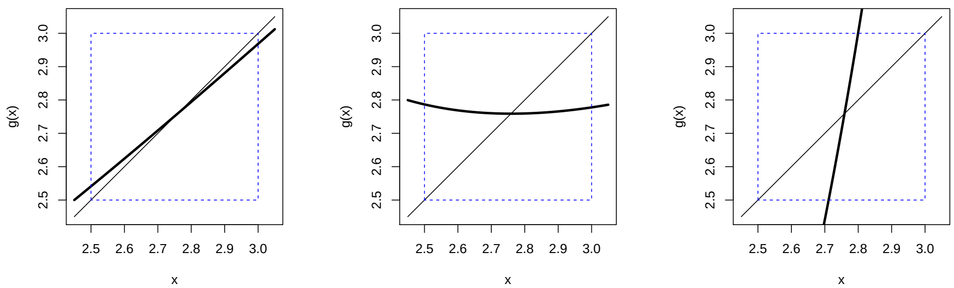
FIG. A.2 – Fonction $g(x) = 2^{-x}$ dans $[0, 1]$

5.6 Les cinq fonctions sont les suivantes :

$$
\begin{aligned}
g_1(x) &= x - x^3 - 4x^2 + 10 \\
g_2(x) &= \left(\frac{10}{x} - 4x\right)^{1/2} \\
g_3(x) &= \frac{1}{2}(10 - x^3)^{1/2} \\
g_4(x) &= \left(\frac{10}{4+x}\right)^{1/2} \\
g_5(x) &= x - \frac{x^3 + 4x^2 - 10}{3x^2 + 8x}.
\end{aligned}
$$

Soit $f(x) = x^3 + 4x^2 - 10$. On peut réécrire l'équation $f(x) = 0$ de différentes façons :

a) $x = x - f(x)$;



(a) fonction $g_1(x)$ (b) fonction $g_2(x)$ (c) fonction $g_3(x)$

FIG. A.3 – Fonctions de l'exercice 5.8

b) $x^3 = 10 - 4x^2 \Rightarrow x^2 = 10/x - 4x \Rightarrow x = (10/x - 4x)^{1/2}$;

c) $4x^2 = 10 - x^3 \Rightarrow x^2 = (10 - x^3)/4 \Rightarrow x = \frac{1}{2}(10 - x^3)^{1/2}$;

d) $4x^2 + x^3 = 10 \Rightarrow x^2 = 10/(4 + x) \Rightarrow x = (10/(4 + x))^{1/2}$;

e) $x = x - f(x)/f'(x)$.

Les fonctions g_1 à g_5 correspondent, dans l'ordre, au côté droit de chacune des équations ci-dessus. On remarquera que la fonction g_5 est celle préconisée par la méthode de Newton–Raphson — et celle qui converge le plus rapidement.

5.7 La fonction $g(x) = 4x^2 - 14x + 14$ est une parabole ayant deux points fixes. On observe graphiquement que l'un se trouve en $x = 2$ et que $g(1{,}5) = g(2)$. Or, pour toute valeur de départ $x_0 < 1{,}5$ de l'algorithme du point fixe, on constate que les valeurs de $x_1, x_2, \dots$ se retrouveront de plus en plus loin sur la branche droite de la parabole. Il en va de même pour tout $x_0 > 2$. La procédure diverge donc pour de telles valeurs de départ. En revanche, il est facile de vérifier graphiquement que la procédure converge pour toute valeur de départ dans l'intervalle $[1{,}5, 2]$. Si $x_0 = 1{,}5$, on obtient le point fixe $x = 2$ après une seule itération.

5.8 En premier lieu, on remarque que les fonctions g_1, g_2 et g_3 sont développées à partir de l'équation $x^3 = 21$ pour calculer $\sqrt[3]{21}$ par la méthode du point fixe. La figure A.3 présente ces trois fonctions. On a

$$\begin{aligned}
g_1'(x) &= \frac{20}{21} - \frac{2}{x^3} \\
g_2'(x) &= \frac{2}{3} - \frac{14}{x^3} \\
g_3'(x) &= 1 - \frac{2x^5 - 84x^3 + 21x^2 + 441}{(x^2-21)^2}.
\end{aligned}$$

Considérons l'intervalle $[2{,}5, 3]$ autour du point fixe. On a que $|g_2'(x)| < |g_1'(x)|$ pour tout x dans cet intervalle, donc la procédure du point fixe converge plus rapidement pour g_2 (la fonction de la méthode de Newton–Raphson). Cela se vérifie aisément sur les graphiques de la figure A.3. Toujours à l'aide des graphiques, on voit que la procédure diverge avec la fonction g_3.

5.9 On considère la fonction $g(x) = 2^{-x}$ dans l'intervalle $[\frac{1}{3}, 1]$. En premier lieu, on a $g(x) \in [\frac{1}{2}, 1/\sqrt[3]{2}] \subset [\frac{1}{3}, 1]$ pour $x \in [\frac{1}{3}, 1]$. De plus, on a $g'(x) = -2^{-x}(\ln 2)$. Puisque $\ln 2 < 1$, on a aussi que $|g'(x)| = g(x)(\ln 2) \in [\frac{1}{3}, 1]$ pour $x \in [\frac{1}{3}, 1]$.

Par le théorème du point fixe, la fonction g possède donc un point fixe unique dans l'intervalle $[\frac{1}{3}, 1]$. La valeur de ce point fixe est

```
> pointfixe(function(x) 2^(-x), start = 2/3)
$fixed.point
[1] 0.64119

$nb.iter
[1] 14
```

5.10 La condition $|g'(x)| \leq k < 1$ du théorème 5.1 assure l'unicité d'un point fixe. Cette condition est nécessaire pour que la fonction $g(x)$ ne puisse repasser au-dessus de la droite $y = x$ après l'avoir déjà croisée une première fois pour créer un point fixe. Or, il est seulement nécessaire de limiter la croissance de la fonction — soit les valeurs positives de sa pente. Peu importe la vitesse de décroissance de la fonction dans l'intervalle $[a, b]$ (mais toujours sujet à ce que $g(x) \in [a, b]$), la fonction ne peut croiser la droite $y = x$ qu'une seule fois. La condition $g'(x) \leq k < 1$ est donc suffisante pour assurer l'unicité du point fixe dans $[a, b]$.

5.11 La distance entre deux points (x_1, y_1) et (x_2, y_2) de $\mathbb{R}^2$ est

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Par conséquent, la distance entre le point (x, x^2) et le point $(1, 0)$ est une fonction de x

$$d(x) = \sqrt{(x-1)^2 + (x^2-0)^2} = \sqrt{x^4 + x^2 - 2x + 1}.$$

On cherche la valeur de minimisant $d(x)$ ou, de manière équivalente, $d^2(x)$. On doit donc résoudre

$$\frac{d}{dx} d^2(x) = f(x) = 4x^3 + 2x - 2 = 0$$

à l'aide de la méthode de Newton–Raphson. Cela requiert également

$$f'(x) = 12x^2 + 2.$$

Le point sur la courbe $y = x^2$ le plus du point $(1, 0)$ est donc la limite de la suite

$$x_n = x_{n-1} - \frac{4x_{n-1}^3 + 2x_{n-1} - 2}{12x_{n-1}^2 + 2}, \quad n = 1, 2, \dots,$$

avec x_0 une valeur de départ « près » de la solution. On obtient

```
> nr(function(x) 4 * x^3 + 2 * x - 2,
+    function(x) 12 * x^2 + 2, start = 0.6)
$root
[1] 0.58975

$nb.iter
[1] 2
```

On voit à la figure A.4 que la solution, disons le point $(x^*, (x^*)^2)$, est la projection du point $(1, 0)$ sur la courbe $y = x^2$; autrement dit, la droite passant par $(x^*, (x^*)^2)$ et $(1, 0)$ est perpendiculaire à la tangente de $y = x^2$ en $x = x^*$.

5.12 La fonction devra utiliser la méthode de Newton–Raphson pour trouver la racine de $f(i) = 0$, où

$$f(i) = \sum_{t=0}^{n} \frac{CF_t}{(1+i)^t}$$

et

$$f'(i) = -\sum_{t=0}^{n} \frac{t\,CF_t}{(1+i)^{t+1}}.$$

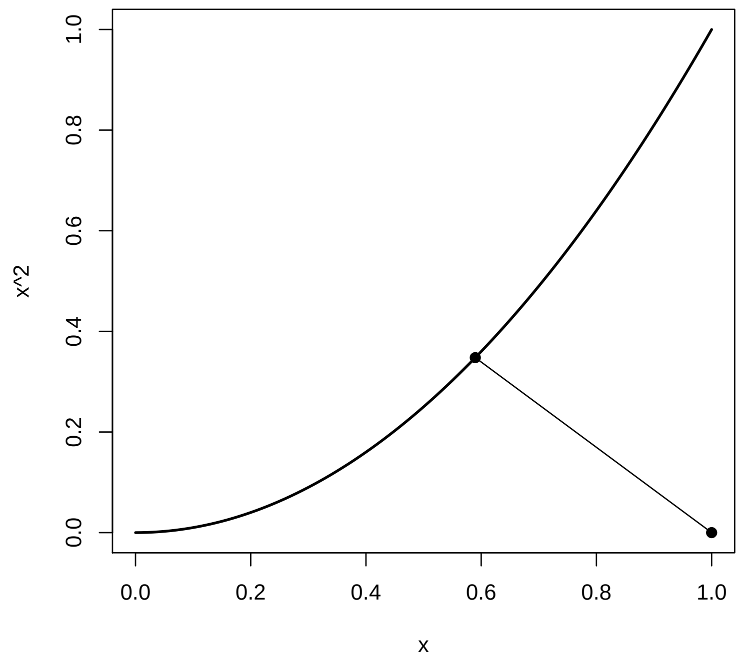


FIG. A.4 – Point de $y = x^2$ le plus près du point $(1, 0)$

La fonction `tri` de la figure A.5 calcule le taux de rendement interne d'un vecteur de flux financiers en utilisant la fonction de Newton–Raphson `nr`.

5.13 a) On veut estimer par la méthode du maximum de vraisemblance le paramètre θ d'une loi Gamma$(3, 1/\theta)$, dont la fonction de densité de probabilité est

$$f(x;\theta) = \frac{1}{2\theta^3}\, x^2 e^{-x/\theta}, \quad x > 0.$$

La fonction de log-vraisemblance d'un échantillon aléatoire $x_1, \dots, x_n$

```
tri <- function(CF, erreur.max = 1E-6)
{
    if (!any(CF < 0))
        stop("au moins un flux financier doit être
              négatif")

    t <- 0:(length(CF) - 1)

    f <- function(i) sum(CF/(1 + i)^t)
    fp <- function(i) sum((-CF * t)/((1 + i)^(t + 1)))

    nr(f, fp, start = 0.05, TOL = erreur.max)$root
}
```

FIG. A.5 – Fonction R de calcul du taux de rendement interne d'une série de flux financiers

tiré de cette loi est

$$\begin{aligned} l(\theta) &= \sum_{i=1}^{n} \ln f(x_i;\theta) \\ &= \sum_{i=1}^{n} \left(2\ln x_i - \frac{x_i}{\theta} - \ln 2 - 3\ln\theta\right) \\ &= 2\sum_{i=1}^{n} \ln x_i - \frac{1}{\theta}\sum_{i=1}^{n} x_i - n\ln 2 - 3n\ln\theta. \end{aligned}$$

On cherche le maximum de la fonction $l(\theta)$, soit la valeur de θ tel que

$$l'(\theta) = \frac{1}{\theta^2}\sum_{i=1}^{n} x_i - \frac{3n}{\theta} = 0.$$

Résoudre cette équation à l'aide de la méthode de Newton–Raphson requiert également

$$l''(\theta) = -\frac{2}{\theta^3}\sum_{i=1}^{n} x_i + \frac{3n}{\theta^2}.$$

La figure A.6 présente une fonction R pour effectuer le calcul à l'aide de notre fonction de Newton–Raphson `nr`. On remarquera que la

```
emv.gamma <- function(x, erreur.max = 1E-6)
{
    n <- length(x)
    xsum <- sum(x)
    f <- function(theta) xsum/theta^2 -
         (3 * n)/theta
    fp <- function(theta) -2 * xsum / theta^3 +
          (3 * n)/theta^2
    nr(f, fp, xsum/n/3, TOL = erreur.max)$root
}
```

FIG. A.6 – Fonction R d'estimation du paramètre d'échelle d'une loi gamma par le maximum de vraisemblance

somme des valeurs de l'échantillon — qui ne change pas durant la procédure itérative — est calculée une fois pour toute dès le début de la fonction. De plus, on utilise comme valeur de départ l'estimateur des moments de θ, $\bar{x}/3$.

b) On a, par exemple,

```
> x <- rgamma(20, shape = 3, scale = 1000)
> emv.gamma(x)
[1] 1065
```

5.14 On cherche à résoudre l'équation

$$a^{(12)}_{\overline{10}|i} = \frac{1 - (1+i)^{-10}}{i^{(12)}} = 8$$

ou, de manière équivalente,

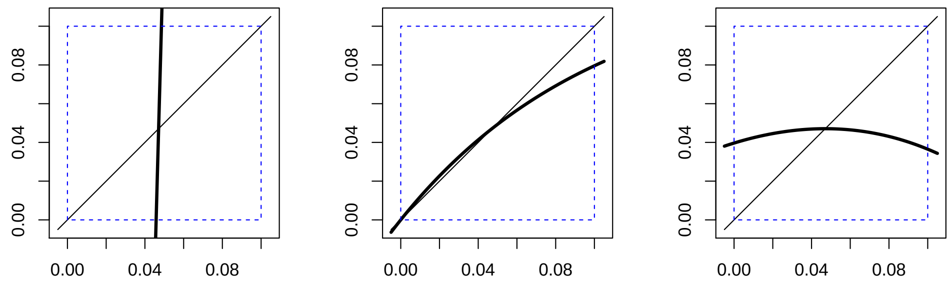
$$f(i) = \frac{1 - (1+i)^{-10}}{12[(1+i)^{1/12} - 1]} - 8 = 0.$$

Pour résoudre le problème à l'aide de la méthode du point fixe, on peut considérer la fonction

$$g_1(i) = i - f(i),$$

mais, comme le graphique de la figure A.7(a) le montre, la procédure itérative divergera. En posant

$$\frac{1 - (1+i)^{-10}}{8} = 12[(1+i)^{1/12} - 1],$$



(a) fonction $g_1(x)$ (b) fonction $g_2(x)$ (c) fonction $g_3(x)$

FIG. A.7 – Trois fonctions pour résoudre $a_{\overline{10}|i}^{(12)} = 8$ par la méthode du point fixe

puis en isolant i du côté droit de l'équation, on obtient une alternative

$$g_2(i) = \left(\frac{97 - (1+i)^{-10}}{96}\right)^{12} - 1.$$

Cette fonction satisfait les hypothèses pour l'existence et l'unicité d'un point fixe, mais sa pente est toutefois près de 1, donc la convergence sera lente ; voir la figure A.7(b). Enfin, on peut considérer la fonction

$$g_3(i) = i - \frac{f(i)}{f'(i)}$$

avec

$$f'(i) = \frac{120(1+i)^{-11}[(1+i)^{1/12} - 1] - [1 - (1+i)^{-10}](1+i)^{-11/12}}{144[(1+i)^{1/12} - 1]^2}.$$

On remarquera que cette dernière est la fonction utilisée dans la méthode de Newton–Raphson et sa pente est presque nulle, d'où une convergence très rapide ; voir la figure A.7(c). On obtient les résultats suivants :

```
> f <- function(i)
+     (1 - (1 + i)^(-10))/(12 * ((1 + i)^(1/12) - 1)) -
+     8
> fp <- function(i)
+     (120 * (1 + i)^(-11) * ((1 + i)^(1/12) - 1) -
```

```
+      (1 - (1 + i)^(-10)) * (1 + i)^(-11/12)) /
+     (144 * ((1 + i)^(1/12) - 1)^2)
> g2 <- function(i) ((97 - (1 + i)^(-10))/96)^12 - 1
> g3 <- function(i) i - f(i)/fp(i)
> pointfixe(g2, start = 0.05)
$fixed.point
[1] 0.047081

$nb.iter
[1] 40
> pointfixe(g3, start = 0.05)
$fixed.point
[1] 0.047081

$nb.iter
[1] 2
```

5.15 a)

```
> f <- function(x) x^3 - 2 * x^2 - 5
> uniroot(f, lower = 1, upper = 4)
```

b) Comme un simple graphique le démontre, il y a deux racines dans l'intervalle.

```
> f <- function(x) x^3 + 3 * x^2 - 1
> curve(f, xlim = c(-4, 0))
> uniroot(f, lower = -4, upper = -1)
> uniroot(f, lower = -1, upper = 0)
```

c)

```
> f <- function(x) x - 2^(-x)
> uniroot(f, lower = 0, upper = 1)
```

d)

```
> f <- function(x) exp(x) + 2^(-x) + 2 * cos(x) - 6
> uniroot(f, lower = 1, upper = 2)
```

e)

```
> f <- function(x) exp(x) - x^2 + 3 * x - 2
> uniroot(f, lower = 0, upper = 1)
```

5.16 On a

```
> dpareto <- function(x, alpha, lambda)
+     (alpha * lambda^alpha)/(x + lambda)^(alpha + 1)
> f <- function(par, x)
+     -sum(log(dpareto(x, par[1], par[2])))
> optim(c(1, 1000), f, x = x)
```

ou, en utilisant l'Astuce Ripley (section 5.12) consistant à estimer le logarithme des paramètres pour éviter les soucis de convergence :

```
> dpareto <- function(x, logAlpha, logLambda)
+ {
+     alpha <- exp(logAlpha)
+     lambda <- exp(logLambda)
+     (alpha * lambda^alpha)/(x + lambda)^(alpha+1)
+ }
> optim(c(log(2), log(1000)), f, x = x)
> exp(optim(c(log(2), log(1000)), f, x = x)$par)
```

Chapitre 6

6.1 On fait le cas de la méthode Simpson seulement, la procédure à suivre dans les autres cas étant tout à fait similaire.

On souhaite développer la formule (6.6) à partir de l'approximation sur un sous intervalle donnée par l'équation (6.5) pour l'intervalle $[x_0, x_2]$. De manière plus générale, nous avons

$$\int_{x_{2j}}^{x_{2(j+1)}} f(x)\,dx \approx \frac{h}{3}[f(x_{2j})+4f(x_{2j+1})+f(x_{2(j+1)})], \quad j = 0, \dots, n-1,$$

d'où

$$\begin{aligned}\int_a^b f(x)\,dx &\approx \sum_{j=0}^{n-1} \int_{x_{2j}}^{x_{2(j+1)}} f(x)\,dx \\ &= \frac{h}{3}\sum_{j=0}^{n-1} [f(x_{2j}) + 4f(x_{2j+1}) + f(x_{2(j+1)})]\end{aligned}$$

$$= \frac{h}{3}\left[f(x_0) + \sum_{j=1}^{n-1} f(x_{2j}) + 4\sum_{j=0}^{n-1} f(x_{2j+1}) \right.$$
$$\left. + \sum_{j=0}^{n-2} f(x_{2(j+1)}) + f(x_{2n})\right]$$
$$= \frac{h}{3}\left[f(a) + 2\sum_{j=1}^{n-1} f(x_{2j}) + 4\sum_{j=0}^{n-1} f(x_{2j+1}) + f(b)\right]$$

puisque, par définition, $f(x_0) = f(a)$ et $f(x_{2n}) = f(b)$.

6.2 Puisque $n = 1$, on peut utiliser directement les formules (6.1), (6.3), (6.5) et (6.7).

a) Point milieu :

$$\int_{0,5}^{1} x^4\,dx \approx 2(0{,}25)(0{,}75)^4 = 0{,}158203$$

Trapèze :

$$\int_{0,5}^{1} x^4\,dx \approx \frac{0{,}5}{2}[(0{,}5)^4 + 1] = 0{,}265625$$

Simpson :

$$\int_{0,5}^{1} x^4\,dx \approx \frac{0{,}25}{3}[(0{,}5)^4 + 4(0{,}75)^4 + 1] = 0{,}194010$$

Simpson 3/8 :

$$\int_{0,5}^{1} x^4\,dx \approx \frac{3(0{,}5/3)}{8}[(0{,}5)^4 + 3(2/3)^4 + 3(5/6)^4 + 1] = 0{,}193866$$

b) Point milieu :

$$\int_{0}^{0,5} \frac{2}{x-4}\,dx \approx 2(0{,}25)(-2/3{,}75) = -0{,}266667$$

Trapèze :

$$\int_{0}^{0,5} \frac{2}{x-4}\,dx \approx \frac{0{,}5}{2}[(-2/4) + (-2/3{,}5)] = -0.267857$$

Simpson :

$$\int_0^{0,5} \frac{2}{x-4}\,dx \approx \frac{0,25}{3}[(-2/4) + 4(-2/3,75) + (-2/3,5)]$$
$$= -0,267064$$

Simpson 3/8 :

$$\int_0^{0,5} \frac{2}{x-4}\,dx \approx \frac{3(0,5/3)}{8}[(-2/4) + 3(-12/23) + 3(12/22) + (-2/3,5)]$$
$$= -0,267063$$

c) Point milieu :

$$\int_1^{1,5} x^2 \ln x\,dx \approx 2(0,25)(1,75)^2 \ln 1,75 = 0,174331$$

Trapèze :

$$\int_1^{1,5} x^2 \ln x\,dx \approx \frac{0,5}{2}[0 + (1,5)^2 \ln 1,5] = 0,228074$$

Simpson :

$$\int_1^{1,5} x^2 \ln x\,dx \approx \frac{0,25}{3}[0 + 4(1,25)^2 \ln 1,25 + (1,5)^2 \ln 1,5]$$
$$= 0,192245$$

Simpson 3/8 :

$$\int_1^{1,5} x^2 \ln x\,dx \approx \frac{3(0,5/3)}{8}[0 + (7/6)^2 \ln(7/6) + (4/3)^2 \ln(4/3)$$
$$+ (1,5)^2 \ln 1,5] = 0,192253$$

d) Point milieu :

$$\int_0^{\pi/4} e^{3x} \sin 2x\,dx \approx 2(\pi/8)e^{3\pi/8} \sin(\pi/4) = 1,803915$$

Trapèze :

$$\int_0^{\pi/4} e^{3x} \sin 2x\,dx \approx \frac{\pi/4}{2}[0 + e^{3\pi/4} \sin(\pi/2)] = 4,143260$$

Simpson :

$$\int_0^{\pi/4} e^{3x} \sin 2x\, dx \approx \frac{\pi/8}{3}[0 + 4e^{3\pi/8}\sin(\pi/4) + e^{3\pi/4}\sin(\pi/2)]$$
$$= 2{,}583696$$

Simpson 3/8 :

$$\int_0^{\pi/4} e^{3x} \sin 2x\, dx \approx \frac{3(\pi/12)}{8}[0 + 3e^{3\pi/12}\sin(\pi/6)$$
$$+ 3e^{3\pi/6}\sin(\pi/3) + e^{3\pi/4}\sin(\pi/2)]$$
$$= 2{,}585789$$

6.3 Avec la méthode du trapèze et $n = 1$, on a $h = 2 - 0 = 2$ et l'approximation de l'intégrale est

$$f(0) + f(2) = 4.$$

Avec la méthode de Simpson, $h = (2-0)/2 = 1$ et l'approximation est

$$\frac{1}{3}[f(0) + 4f(1) + f(2)] = 2.$$

En remplaçant la première équation dans la seconde et en résolvant, on trouve facilement $f(1) = 1/2$.

Bibliographie

ANSI. 1986, *ANSI X3.4-1986: Coded Character Set. 7-Bit American Standard Code for Information Interchange*, American National Standards Institute, New York.

Burden, R. L. et J. D. Faires. 1988, *Numerical Analysis*, 4e éd., PWS-Kent, Boston, ISBN 0-5349158-5-X.

Burden, R. L. et J. D. Faires. 2011, *Numerical Analysis*, 9e éd., Brooks/Cole, ISBN 978-0-5387335-1-9.

Unicode Consortium, T. 2007, *The Unicode Standard, Version 5.0.0*, Addison-Wesley, Boston, ISBN 0-32148091-0.

IEEE. 2003, *754-1985 IEEE Standard for Binary Floating-Point Arithmetic*, IEEE, Piscataway, NJ.

ISO. 1998, *ISO/IEC 8859-1:1998. Information Technology – 8-bit Single-Byte Coded Graphic Character Sets – Part 1: Latin Alphabet No. 1*, International Organization for Standardization, Geneva.

Kernighan, B. W. et P. J. Plauger. 1978, *The Elements of Programming Style*, 2e éd., McGraw-Hill, ISBN ISBN 0-07034207-5.

Knuth, D. E. 1997, *The Art of Computer Programming*, vol. 2, Seminumerical Algorithms, Addison-Wesley, Reading, MA.

Monahan, J. F. 2001, *Numerical Methods of Statistics*, Cambridge University Press, Cambridge, UK, ISBN 0-52179168-5.

Ripley, B. D. 2005, «Internationalization features of R 2.1.0», *R News*, vol. 5, no 1, p. 2–7. URL http://cran.r-project.org/doc/Rnews/.

Wikipedia. 2012, «IEEE 754-2008 — Wikipedia, the free encyclopedia», URL http://en.wikipedia.org/wiki/IEEE_floating-point_standard.

Wikipedia. 2013, « Complément à deux — Wikipédia, l'encyclopédie libre », URL http://fr.wikipedia.org/wiki/Complément_à_deux.

Ce document a été produit avec le système de mise en page XƎLaTeX. Le texte principal est composé en Lucida Bright OT 11 points, les mathématiques en Lucida Bright Math OT, le code informatique en Lucida Grande Mono DK et les titres en Fira Sans. Des icônes proviennent de la police Font Awesome. Les graphiques ont été réalisés avec R.