Explorations mathématiques avec ordinateur Technologie d'algèbre

John Harris,
Karen Kohl, et
John Perry,
tous employés à l'Université du sud du Mississippi
(au moins jusqu'à ce que le bureau du prévôt lise cette débâcle)

2000 Classification des matières mathématiques. 97U30, 97N80

Brouillon 2·3 $_2$, juin (x +1)(x +5), où x est le sens de la vie, de l'univers et de tout

Contenu

Remerciements

Les auteurs tiennent à remercier :
• Le Bureau du Provost de l'Université du sud du Mississippi, qui en 2016 a soutenu
a porté la création de ce travail avec une bourse d'été pour l'amélioration des
tion.
• William Stein et divers développeurs et utilisateurs de Sage pour leur soutien moral et
occasionnel
un soutien financier pour assister aux ateliers Sage Days.
• Tous les développeurs qui ont contribué au projet Sage, directement ou
correctement.
• Amber Desrosiers et Candice Bardwell Mitchell ont souffert d'un premier repêchage de cette
texte et trouvé plus d'erreurs typographiques que nous ne voulons l'admettre.

• Valerio de Angelis a aimablement signalé un grand nombre d'erreurs typographiques et a suggéré
Gested une amélioration à une blague sur les boucles infinies. Il a également contribué à quelques-uns des laboratoires
dans l'Encyclopédie Laboratorica.
• Deux d'entre nous ont des conjoints et des enfants qui méritent des excuses plus que des remerciements, deux d'entre nous
ont des chats qui méritent tout ce que les chats méritent (tout ?), et l'un de nous a un lapin et une tortue, qui méritent tous deux plus d'attention.
En plus:
• L'image de Glenda, le lapin Plan 9 à la p. 209 a été téléchargé à partir du plan de Bell Labs 9 et est utilisé selon les termes donnés.
• L'image de Rabbit Island à la p. 195 a été prise par Kim Bui et est utilisé sous un licence accordée (CC BY-SA 2.0).
7

Préface

Pourquoi avez-vous écrit ce texte ?
L'objectif principal de ce texte est de justifier un salaire d'été partiel qui nous est prolongé par notre employeur
Provost, qui sans doute ne se trompera plus ainsi.
Non, vraiment, pourquoi avez-vous écrit ce texte ?
Notre institution offre aux majors un cours sur la résolution de problèmes avec la technologie intitulé, Mathematical
Calcul. Nous ne trouvons pas de texte qui corresponde à son caractère unique. Nous croyons au cours et pensons
c'est une bonne idée; beaucoup d'étudiants qui le prennent finissent par être d'accord.[1]
À l'origine, la classe utilisait un manuel basé sur un autre système de calcul formel, mais pour raisons énumérées ci-dessous, nous sommes passés à Sage. L'interface de Sage repose sur Python, nous avons donc utilisé un très
bon livre sur la programmation Python pendant un certain temps, mais il n'y a jusqu'à présent qu'un livre sur Python pro-
gramming vous emmènera avec Sage.
D'où ce livre. Avec un peu de chance, le texte résultant surgira comme un champignon, ici et là,
imperméable à l'éradication, jusqu'à ce qu'elle domine le paysage de l'enseignement des mathématiques.
Quelle valeur pédagogique ce texte offre-t-il ?
Nous aimerions penser que ce livre serait utile pour toute situation où un individu est passer des mathématiques « lycées », dans lesquelles nous incluons le calcul de base, à « l'université »

mathématiques, qui incluent le calcul intermédiaire et beaucoup de choses en plus. Beaucoup de nos étu-
les dents luttent avec la transition des mathématiques informatiques principalement concrètes qu'elles
expérimentés dans leur jeunesse, aux mathématiques théoriques de plus en plus abstraites qu'ils rencontrent
en mathématiques supérieures.
La technologie est devenue un aspect indispensable de la plupart des cours de mathématiques. La quantité
de puissance de calcul dans les téléphones portables d'aujourd'hui est de plusieurs ordres de grandeur plus que
puters avaient dans l'enfance des auteurs - et la plupart des familles ne pouvaient même pas se permettre un ordinateur à la maison.
Le nôtre est un monde magique, mais les étudiants sont souvent réticents à jouer dans ce monde. S'ils ne le font pas
savent comment résoudre un problème tout de suite, ils pensent que quelque chose ne va pas. Nous pouvons leur apprendre
sur les groupes (disons) et donnez-leur quelques exemples, mais les élèves sont souvent trop réticents à explorer
les exemples par eux-mêmes.
Pourquoi Sauge ?
Nous préférons Sage pour plusieurs raisons :
• La sauge est « libre comme dans la bière ». Les auteurs travaillent dans une université dans l'un des pays
des États plus modestes ; tandis que les éditions étudiantes de nombreux systèmes de calcul formel sont

1 Cet accord n'intervient souvent qu'après l'expérience d'un stage ou d'un premier emploi, mais ça compte !

---

sans doute abordables, ils sont néanmoins un ajout non négligeable au coût élevé de nos études.
les bosses portent déjà.
• La sauge est « libre comme dans la parole ». Les instructeurs peuvent montrer aux étudiants talentueux des parties du code, et
encouragez-les à s'impliquer. Il y a sans aucun doute un certain nombre de bons étudiants de premier cycle
projets de recherche pouvant conduire à des contributions à la base de code Sage. Instructeur talentueux
Les tors peuvent contribuer au code de manière à améliorer l'utilisation de Sage dans l'éducation. D'ailleurs,

deux d'entre nous ont montré que même des instructeurs sans talent peuvent contribuer au code de Sage. Ne pas
soyez timide : la communauté est solidaire !

• L'interface de Sage repose sur Python, un langage de programmation standard en haute demande des employeurs. Enseigner aux étudiants Sage signifie que nous leur enseignons pas mal
de Python également, augmentant leurs perspectives d'emploi.

N'y a-t-il pas déjà de bonnes références sur Sage ?

Oui, et c'est le point; ce sont des références pour les mathématiciens et les étudiants pour apprendre Sage.

Ce texte vise à aider les élèves à apprendre les mathématiques via Sage. Cela le place dans un créneau différent,
qui, nous l'espérons, s'avérera non seulement rentable,[2] mais aussi utile,[3] à lire et à éditer.

Comment utilisez-vous généralement ce texte ?

Aucune section de laboratoire n'est rattachée au cours de notre établissement, nous avons donc tendance à offrir des journées de laboratoire en classe;
procéder dans un semblant de l'ordre fourni ici ; et attribuer des questions de manuels, des laboratoires et
essais.

Nous utilisons différents labos, selon l'instructeur et l'année. Nous modifions également les laboratoires sur
occasion. Modifier les laboratoires au goût est conseillé à une époque où un nombre non négligeable d'étudiants a
appris que l'externalisation de la production vers un moteur de recherche en ligne fournit des solutions préfabriquées
plus rapidement que leurs moteurs de calcul internes. Même si vous ne modifiez aucun d'entre eux
au goût, le grand nombre de laboratoires signifie que le cours peut varier selon les intérêts de recherche
ou le goût pédagogique.

Si vous nourrissez une hostilité particulière envers vos élèves, n'hésitez pas à contacter les auteurs. Nous pouvons
indiquer quels laboratoires se sont avérés particulièrement difficiles dans la pratique.

Un dernier mot ?

Après cinq ans de développement, ce texte et sa source sont sortis en 2021 sur un plateforme de diffusion.

Concernant les errata. La tradition mathématique raconte qu'un professeur a une fois entrepris d'écrire un manuel
exempt d'erreur. Un critique du texte résultant a observé sèchement que l'auteur a fait « plutôt bien :
la première erreur apparaît à la page 9.

Nos objectifs sont plus modestes : nous promettons seulement que les erreurs que le lecteur

trouvera iront de
du purement typographique au carrément menteur.
• Pour les erreurs typographiques, nous demandons au lecteur de contacter les mainteneurs. Si l'un des
le dossier de publication des auteurs est un indicateur quelconque, les corrections typographiques seront
avoir des erreurs typographiques. Avertissement lecteur.

2 Arrêtez de rire.

3 Vraiment. Arrêter de rire.

• Les erreurs du mensonge ont le mérite de sonner mieux que la vérité, donc dans l'esprit des fois où nous avons décidé de les inclure.[4] Nous laissons au lecteur le soin de trier le blé de l'ivraie, bien que nous assurons au lecteur dévoué qu'il y a plus de blé que d'ivraie.[5]
Le lecteur intéressé par le développement historique de ce texte peut souhaiter visiter www.math.usm.edu/dont_panic/
Existe-t-il une copie papier ? Non. Il y en avait, mais les manuels en couleur sont chers et vous perdez
l'expérience des animations intégrées dans le texte. Si vous souhaitez une copie papier, n'hésitez pas à imprimer
un, de préférence en utilisant une imprimante couleur sur du papier de haute qualité.

4 Au moins un mensonge pur et simple est démenti en tant que tel, et la justification du mensonge est expliquée.

5 À moins que cette affirmation ne soit aussi un mensonge.

Partie 1
Cours

CHAPITRE 1
Arrière-plan

Une chose dont ils n'avaient jamais eu besoin auparavant est devenue une nécessité.
(Narrateur, [11])

Pour explorer le monde des mathématiques, vous ne pouvez pas simplement lire à ce sujet ; vous devez l'engager. Quelque
les gens sont doués d'une confiance et/ou d'une aptitude suffisantes pour s'y attaquer seuls ; certains ont eu la chance d'être bien élevés, et depuis leur jeunesse sont habitués à s'engager le monde des mathématiques.
La plupart ont moins de chance, et bien qu'ils puissent trouver le monde plus vaste des mathématiques intrigant, ils

lutte pour se frayer un chemin à travers un territoire nouveau et inconnu. Ce texte vise à encourager
vous d'explorer le monde des mathématiques supérieures à l'aide d'un système de calcul formel,
s'appuyant sur un système particulier nommé Sage. Nous vous encourageons à expérimenter avec des problèmes
lems et forment des conjectures sur leurs solutions. Parfois, nous vous encourageons à utiliser le
l'expérimentation pour formuler une explication quant à la raison pour laquelle la conjecture est, en fait, vraie.
Mais pour utiliser efficacement les ordinateurs, vous devez apprendre à programmer.
Est-ce juste un autre manuel de programmation?
Non.
Peux-tu être plus précis? Oui.
[Grogner.] Veuillez entrer dans les détails. Ce texte traite des mathématiques, qui elles-mêmes résoudre les problèmes. C'est assez important pour être souligné, pour que même les skimmers le remarquent.
Les mathématiques sont un outil pour résoudre des problèmes.
En particulier, nous voulons vous présenter des idées et des techniques de mathématiques supérieures à travers
problèmes qui sont sans doute mieux abordés avec un ordinateur, car
• ils sont longs ;
• ils sont répétitifs ou fastidieux ; et/ou
• ils nécessitent une expérimentation.
Les ordinateurs ont besoin d'instructions et un groupe d'instructions s'appelle un programme. [1] Nous étudions donc
programmation, mais en référence à un objectif précis : à savoir, résoudre des problèmes en mathématiques. Cette
rend ce texte différent des manuels de « programmation », qui étudient la programmation en référence
à un autre objectif : à savoir, résoudre des problèmes en informatique. Encore une fois, cette distinction est
assez important pour être mis en évidence, de sorte que même les écumeurs le remarqueront.
Nous étudions la programmation afin de résoudre des problèmes mathématiques.

1 Si vous lisez ceci en tant que document électronique, vous remarquerez de temps en temps du texte dans un Couleur. Si vous cliquez dessus, vous trouverez sur Internet des liens vers des informations supplémentaires, dont très peu sont
due aux auteurs de ce texte et, à ce titre, est probablement beaucoup plus fiable et utile. Nous espérons que vous suivez ces
liens et familiarisez-vous avec ces informations — c'est pourquoi nous les avons incluses ! - mais à proprement parler ce n'est pas
nécessaire.

Pourquoi programmer ? Si je voulais écrire des programmes, je me spécialiserais en informatique. Commencer
avec, certains problèmes sont trop difficiles à faire à la main, vous devez donc utiliser un ordinateur. Le but
de ce texte est de vous engager dans cette voie sans faire de vous une majeure en informatique. Il
vous présentera non seulement certains outils d'un système de calcul formel qui vous aident à calculer, il
vous encouragera à prendre des mesures pour devenir un pionnier des mathématiques. Pour résoudre le
problèmes que nous présentons, vous devrez vous arrêter à une caractéristique particulière juste à l'extérieur de votre quartier
et apprenez-en un peu plus que vous ne le feriez en l'absence de nos encouragements.
Dans certains cas, nous vous ramènerons même sur des terres que vous avez déjà parcourues et vous demanderons de réexaminer
quelque chose d'un peu plus attentivement. Bref, et en répétition, ce texte parle d'explorer le monde
de mathématiques supérieures, avec l'aide d'un ordinateur.
D'une part, les ordinateurs ne comprennent pas les langues humaines. Les humains sont intuitifs et
poétique, en recourant au langage figuratif et abstrait. Les ordinateurs ne comprennent rien de tout cela ; elles ou ils
ne comprends vraiment qu'une chose: allumé et éteint. Tout ce que votre téléphone portable, ordinateur portable ou ordinateur de bureau
implique un arrangement intelligent d'interrupteurs par des êtres humains bien formés dans l'art
de diriger le courant électrique au bon endroit au bon moment.
D'un autre côté, la plupart des humains ne comprennent pas le langage d'activation et de désactivation de l'ordinateur.
Réduire chaque problème à ces deux symboles a été extrêmement efficace, mais c'est inconfortable-
capable aux humains (ne serait-ce que parce que c'est tellement fastidieux!).
Apprendre à programmer vous donne le contrôle des circuits de l'ordinateur et vous permet de travailler
à un niveau beaucoup plus confortable. Même les experts travaillent généralement avec des interfaces qui sont elles-mêmes converties en signaux d'activation et de désactivation en plusieurs étapes. Apprendre à pro-
gram vous donne également une compréhension plus approfondie de la technologie informatique et une appréciation de la

quantité de travail nécessaire à la construction de ce monde magique dans lequel nous vivons, où vous pouvez parler
une petite boîte à la main et être entendu par quelqu'un à l'autre bout du monde.
Quels types de langages de programmation existe-t-il ? Sans entrer dans trop de détails, il existe aujourd'hui trois types de langages de programmation informatique :
• Dans un langage de programmation interprété, l'ordinateur lit un fichier qui contient des mandements dans la langue. Il traduit chaque symbole et exécute une séquence de commandats basés sur ce symbole. (Ici, « symbole » peut inclure des mots ainsi que des nombres et
caractères abstraits.) Il passe ensuite au symbole suivant, oubliant finalement sa transtion de la précédente. Des exemples de langages interprétés incluent BASIC , Python, et les langages « shell » des invites de ligne de commande.
• Dans un langage de programmation compilé, l'ordinateur lit un fichier qui contient des commandes
dans la langue. Il traduit chaque symbole, mais au lieu d'exécuter des commandes, il enregistre la traduction en signaux marche et arrêt et les stocke dans un nouveau fichier, généralement
appelé un exécutable. (Nous écrivons « habituellement » car il produit parfois une bibliothèque à la place,
selon la demande du programmeur.) Des exemples de langages compilés incluent Fortran , C/ C++, et allez.
Avant de décrire le troisième type de langage, mentionnons quelques avantages et inconvénients
de chaque type. Les langages interprétés sont par nature généralement beaucoup, beaucoup plus lents que compilés
langues, car l'ordinateur doit retraduire chaque symbole, quel que soit le nombre de fois il l'a déjà traduit. (C'est quelque peu simpliste, mais ce n'est pas si loin non plus
de la vérité.) D'autre part, les langues interprétées sont généralement beaucoup plus interactives
et flexible que les langages compilés, au point que de nombreux utilisateurs n'écrivent jamais un programme réel
pour eux, mais simplement émettre une commande à la fois.

De même, la nature des langages compilés signifie que les signaux d'activation et de désactivation précis sont liés
à une architecture particulière, l'une des raisons pour lesquelles les programmes compilés pour s'exécuter sur un appareil Windows ne
fonctionner sur Macintosh ou Linux. Les développeurs C++ doivent recompiler chaque programme pour un

architecture, et pour de nombreux programmes, cela devient assez difficile, surtout si le programme repose sur
fortement sur l'interface graphique particulière de Windows. En conséquence, les langues interprétées peuvent être
beaucoup plus « portables », ce qui signifie que vous pouvez simplement les copier sur une autre machine. Python
Les programmes sont particulièrement réputés pour leur portabilité. On peut soutenir que Microsoft Corporation
le succès est principalement dû à son interprète BASIC brillant et omniprésent pour les ordinateurs personnels de
la fin des années 70 et le début des années 80 ; il vit aujourd'hui sous le nom de Visual BASIC.

• Les langages de bytecode cherchent à combler l'écart entre les deux. Dans ce cas, l'ordinateur
lit un fichier qui contient des commandes dans la langue et traduit chaque symbole, mais pas dans les signaux marche et arrêt natifs de l'architecture de l'ordinateur. Il se traduit plutôt en signaux conçus pour un ordinateur abstrait appelé machine virtuelle, puis stocke dans un exécutable ou une bibliothèque qui ne fonctionnera que sur les ordinateurs qui contiennent des pro-
grammes qui comprennent les signaux de la machine virtuelle. Les langues de bytecode notables incluent
Pascal, [2] Java, et le Common Language Runtime des langages .NET. [3]
Étant donné que les exécutables et les bibliothèques ne sont pas dans les propres signaux de l'ordinateur, les langages de bytecode
sont techniquement un type particulier de langage interprété, et leur dépendance à l'égard d'une machine virtuelle
signifie qu'ils sont théoriquement plus lents que les langages compilés. En pratique, la sanction est généralement
assez petit, car les signaux de la machine virtuelle sont conçus pour être traduits très facilement en
les signaux d'un ordinateur réel. Les techniques modernes les rendent si efficaces que certains bytecode
langages surpassent fréquemment de nombreux langages compilés. Le recours à la ma- virtuelle abstraite
chine signifie que les langages de bytecode sont hautement portables ; nous avons écrit plus haut que les exécutables fonctionnent
« uniquement » sur les ordinateurs qui contiennent des programmes qui comprennent les signaux de la machine virtuelle, mais
cela signifie également qu'ils fonctionnent sur "n'importe quel" ordinateur avec un programme qui comprend la ma-
signaux de la Chine. La popularité de Java - il semblait autrefois impossible de trouver une page Web sans Java

applet - était due à sa philosophie "Write Once, Run Anywhere", qui dépendait de son capacités de la machine.
Lequel de ces types de langage utilisons-nous dans ce texte ? Tous, en fait.
Le langage de programmation principal dans Sage est Python, que nous avons répertorié ci-dessus comme un
Langue. Sage et Python ne sont pas tout à fait les mêmes, cependant ; Les programmes Sage ne fonctionneront pas en clair,
vanilla Python, et certaines fonctionnalités Python sont différentes dans Sage.
Vous pouvez parfois compiler des programmes Sage en utilisant un programme Cython; nous couvrons cela près du
fin du texte. Cependant, « Cythonized » Sage ne se suffira pas à lui-même ; vous devez l'exécuter à l'intérieur
un environnement Sage.
Comme vous l'apprendrez ci-dessous, Sage est en fait assemblé à partir d'un grand nombre de pièces distinctes.
Certains d'entre eux sont écrits avec Java, ce qui signifie que vous utilisez un langage de bytecode, cependant
vous n'écrirez en fait aucun programme Java.
C'est quoi ce truc de Sage sur lequel tu n'arrêtes pas de japper ?
Sage est un système de calcul formel gratuit et open source.

2 Dans ses incarnations originales, Pascal a été traduit en une idée similaire appelée P-code, plus tard en bytecode proprement dit.

3 De nombreux langages interprétés sont désormais compilés « de manière incrémentielle ». Ils enregistrent l'interprétation de chaque commande, et
au fur et à mesure de leur exécution, vérifiez si chaque nouvelle commande a déjà été interprétée.

Qu'est-ce qu'un « système de calcul formel » ? Traditionnellement, il existe trois grands types de
progiciels mathématiques à grande échelle :
• Les systèmes de calcul numérique visent un calcul rapide, en s'appuyant généralement sur nombres ponctuels. Nous n'entrerons pas dans les détails pour le moment, mais vous pouvez penser à flotter
point comme une sorte d'« estimation précise », semblable à l'utilisation de chiffres significatifs dans le
les sciences. La science derrière le calcul numérique est généralement du ressort de la numéri-analyse cal. Presque tout le monde dans le monde développé a utilisé un ordinateur numérique système à un moment donné en se tournant vers une calculatrice. Les progiciels numériques incluent
MATLAB et Octave.
• Les progiciels statistiques sont un type particulier de système de calcul numérique qui se

concentre
sur des outils spéciaux propres à l'analyse statistique. Les exemples incluent SAS et le projet R.

• Les systèmes de calcul formel visent un calcul exact, même si cela se fait au détriment de la vitesse que l'on associe aux systèmes numériques. Plutôt que de manipuler environ valeurs de contrainte, les systèmes de calcul formel manipulent des symboles qui représentent des valeurs exactes :
ils ne voient pas π comme un nombre décimal mais comme un symbole avec les propriétés mathématiques
s'y associe, et ils nous permettent de manipuler des expressions qui impliquent des variables. Être-
pour cette raison, la science derrière les systèmes d'algèbre informatique est appelée calcul symbolique,
algèbre informatique ou algèbre computationnelle. Quelques-unes des calculatrices les plus chères utilisent
calcul symbolique, mais typiquement on travaille avec un progiciel comme Maxima, Érable, ou Sauge.

Pourquoi sacrifierait-on l'exactitude aux valeurs approximatives d'un système numérique ? Le principal
la raison est la vitesse ! En sacrifiant une petite quantité de précision, un système numérique peut facilement fonctionner
avec à la fois de grands et de petits nombres, des vecteurs et des matrices, le tout sans trop de problèmes ; ce n'est pas
rare de travailler avec des centaines voire des milliers de variables dans un système d'équations.

Par exemple, que se passe-t-il si vous additionnez les fractions $1/2$, $1/3$, $1/5$ et $1/7$? Chacun d'eux nécessite
seulement deux chiffres à écrire (numérateur et dénominateur), mais la somme exacte, $247/210$, nécessite six
chiffres - une augmentation de 300% de la taille! Si vous utilisez des nombres à virgule flottante avec au plus 4 chiffres, le
somme devient à la place
0,5000+0,333+0,2000+0,1429 = 1,176.
La somme n'est pas plus grande que les nombres d'origine. C'est vrai, c'est un peu faux, mais l'erreur est moindre
que $1/500$; c'est beaucoup plus précis que la plupart des tâches quotidiennes n'en ont besoin.

La croissance rapide de la taille est l'une des raisons pour lesquelles les gens n'aiment généralement pas les fractions ; personne n'aime travailler
avec des objets dont la complexité croît rapidement. Dans ce cas, ce qui est vrai sur les gens est vrai
sur les ordinateurs aussi; si vous travaillez avec un problème qui nécessite une division d'entiers dans Sage, vous

rencontrera presque certainement un ralentissement massif à mesure que les fractions se compliqueront et,
par conséquent, plus difficile à manipuler pour l'ordinateur.
Dans ce cas, pourquoi s'embêter avec le calcul symbolique et les valeurs exactes ? Pour de nombreux problèmes, les imprécisions du calcul en virgule flottante le rendent absolument inadapté.
Cela est particulièrement vrai lorsque la division est une partie incontournable du problème. Par exemple, supposons que l'ordinateur doive diviser par le nombre à virgule flottante à 4 chiffres 0,0001.
Le quotient résultant devient très grand. Pourtant, il est tout à fait possible que 0,0001 soit le résultat d'un
erreur à virgule flottante, et la valeur correcte est en fait 0. Comme vous le savez sûrement, la division
par 0 est mauvais. Si l'ordinateur avait su que la valeur était 0, il n'aurait pas du tout divisé ! Problèmes
où cela peut se produire sont souvent appelés « mal conditionnés », et les analystes numériques passent beaucoup de temps
essayant de les éviter.

Dans certains cas, cependant, vous ne pouvez pas, vous avez donc recours à des valeurs exactes. Cela est d'autant plus vrai que
on se dirige vers des domaines mathématiques plus abstraits. Certaines personnes pensent que les mathématiques « abstraites » sont
mathématiques « inutiles », mais elles se trompent complètement : ce texte vous présentera plusieurs
objets mathématiques dont l'exactitude même rend possible l'extrême précision et l'extrême communication sécurisée que vous obtenez sur Internet chaque fois que vous consultez votre compte bancaire ou
acheter auprès d'un vendeur en ligne.
Quelle est la particularité de Sage ? Sage a été « démarré » par William Stein, un théoricien des nombres. Il était
frustré par plusieurs inconvénients des systèmes de calcul formel disponibles à l'époque :
Les systèmes commerciaux, comme Maple, ne permettent pas à l'utilisateur de visualiser le code, encore moins de le modifier.
Dans le monde du logiciel, ces systèmes sont appelés « propriétaires », « fermés » ou « non libres » (par certains
gens). Des systèmes «ouverts» ou «libres» existaient également et, fruit d'une recherche de pointe, ils
étaient souvent meilleurs dans une tâche particulière que les packages commerciaux.

Cependant, ces paquets
n'exceller que dans un domaine particulier des mathématiques :
• pour le calcul, vous utiliseriez probablement Maxima ;
• pour l'algèbre linéaire, vous utiliseriez probablement Linbox;
• pour la théorie des groupes, vous utiliseriez probablement GAP;
• pour la théorie des nombres, vous utiliseriez probablement Pari;
• pour l'algèbre commutative, vous utiliseriez probablement CoCoA , Macaulay, ou singulier;
et ainsi de suite. Pire encore, supposons que vous deviez transférer le résultat d'un package à un autre :
après avoir effectué une théorie des nombres avec Pari, par exemple, vous voudrez peut-être analyser le
résultats en utilisant la théorie des groupes, auquel cas GAP serait l'outil de choix. Mais il n'y avait pas
moyen simple de copier les résultats de Pari dans GAP.
Sage a donc été organisé pour lier ces outils brillants ensemble en un seul pack relativement simple.
âge. Des fonctionnalités supplémentaires ont été programmées dans Sage proprement dit, et dans certains cas, Sage a été le
leader à résoudre certains problèmes. En prime, les développeurs de Sage ont permis d'interagir
avec de nombreux packages propriétaires via Sage, de sorte que si Maple dispose des outils les plus rapides pour résoudre certains
problème, et vous possédez une copie, vous pouvez le faire via Maple, puis manipuler le résultat
par Sage.
Pourquoi se préoccuper des logiciels « libres » ? Dans le monde du logiciel, le terme « gratuit » a deux
sens :
Gratuit comme dans la bière : vous n'avez pas à payer pour cela.
Libre comme dans la parole : le code est ouvert et visible, plutôt que « censuré ».
Le logiciel peut être « libre comme dans la bière » mais pas « libre comme dans la parole » ; c'est-à-dire que cela ne coûte rien, mais vous ne pouvez pas
voir le code source. Les exemples incluent les nombreux programmes « gratuits » que vous pouvez télécharger pour un
ordinateur ou téléphone portable.
Il existe des raisons importantes pour lesquelles un chercheur ou même un ingénieur devrait pouvoir visualiser et/ou
modifier le code dans un logiciel mathématique :
• Une bonne pratique scientifique requiert la reproductibilité et la vérifiabilité. Mais un chercheur ne peut
vérifier les résultats d'un calcul mathématique si elle ne peut pas vérifier la façon dont il a été com-

mis.
• Tout progiciel de taille significative comportera des erreurs, appelées bogues. Si deux soft-
les paquets de ware produisent une réponse différente, un chercheur ne peut pas décider lequel est correct
s'il ne peut pas voir le code et évaluer les algorithmes.

• Presque toutes les recherches mathématiques s'appuient sur des travaux antérieurs. Il en est de même pour les soft-
logiciels, et les chercheurs ont souvent besoin d'étendre un package avec de nouvelles fonctionnalités afin
pour accomplir une tâche. Cela peut être beaucoup plus difficile à faire correctement si le chercheur
ne peut pas voir le code, encore moins le modifier.
Par exemple, supposons qu'un mathématicien prétende avoir une preuve qu'il existe une infinité de
nombres premiers. La plupart des mathématiciens voudraient voir la preuve ; c'est une façon d'apprendre de chacun
autre. (Dans certains cas, la preuve est beaucoup plus intéressante que le théorème.) En effet, vous pouvez trouver
cette preuve dans beaucoup, beaucoup de manuels, parce que le mathématicien hellénique Euclide d'Alexandrie
enregistré ce qui est considéré comme l'une des plus belles preuves de ce fait sur deux mille ans
depuis [dix, Livre IX, Proposition 20] :
T HEOREM . Il existe une infinité de nombres premiers.
P TOIT . Considérons tout ensemble fini et non vide de nombres premiers, $P = \{p_1, ..., p_n\}$. Soit $q = p_1 \cdots p_n + 1$. Puisque $q = 1$, au moins un nombre premier le divise, mais le reste der de diviser q par n'importe quel $p \in P$ est 1, donc aucun des nombres premiers de P ne le divise. Donc là
doit être un nombre premier non répertorié dans P. Mais P est un ensemble fini arbitraire de nombres premiers, ce qui signifie qu'aucun ensemble fini de nombres premiers ne peut tous les énumérer. En d'autre
mots, il y a une infinité de nombres premiers.
ré
En exposant clairement la preuve, Euclide facilite la vérification du résultat. Il facilite aussi remettre en question l'argument : vous pourriez vous demander, par exemple, comment Euclide sait qu'au moins un
premier divise tout entier positif qui n'est pas 1. (En fait, il l'a prouvé plus tôt.)
Comparez cela à un autre théorème célèbre attribué à Pierre de Fermat, un juriste français qui

a étudié les mathématiques comme passe-temps [9, p. 61, Observatio Domini Petri de Fermat] :

T HEOREM . Si $n > 2$, l'équation $a_n + b_n = c_n$ n'a pas de solution avec des nombres entiers a, b, c 1.

P TOIT . J'ai trouvé une preuve vraiment merveilleuse de ce fait. Cette marge est la petitesse ne le contiendra pas.

ré

Ces deux phrases déclenchèrent une recherche de preuve qui dura plus de trois siècles ; André Wiles a trouvé une preuve en 1994, et à ce jour il n'y a pas d'autres preuves. La plupart des gens conviennent que Fermat n'avait pas, en fait, de preuve, mais il ne faut pas penser du mal de lui : il n'a en fait jamais dit à personne qu'il avait une preuve; il a simplement écrit ce commentaire dans la marge d'un livre. Son fils a publié un copie du livre après la mort de Fermat, et inclus les notes que Fermat a écrites dans la marge.

Pour pousser l'analogie plus loin, supposons que nous devions affirmer ce qui suit :

T HEOREM . Le nombre

2_n -1

est premier pour n = 2,3,5,7,13,17,19,31,67,127,257.

P TOIT . Secret de commerce.

ré

Vous auriez raison de douter de notre affirmation, pour au moins deux raisons : il n'y a pas de moyen facile de vérifier ça, et en fait c'est faux ! Pourtant, cette affirmation a été faite par un mathématicien très respecté nommé Marin Mersenne [8], offert sans preuve, et fut pendant un certain temps généralement accepté. Vous serez répondre à cette revendication à nouveau dans un laboratoire plus tard.

Quels sont les avantages de Sage ? Comme mentionné précédemment, Sage facilite l'expérimentation avec des objets mathématiques que vous utiliserez de plus en plus dans les cours après celui-ci. Plus tard

---

classes n'exigeront probablement pas explicitement Sage, mais si vous n'utilisez plus jamais Sage après cette classe, ce serait comme suivre un cours de statistiques et faire tout le travail à la main :[4]

POURQUOI ?!?

Un autre avantage de Sage est que vous interagissez avec lui via une interface Python ; programme-

ming dans Sage est, dans une certaine mesure, impossible à distinguer de la programmation en Python. Cela confère
les avantages suivants :
• Rappelons que Python est l'un des langages interprétés les plus répandus.
– De nombreux employeurs veulent une expérience Python. Bien apprendre Sage vous aide à apprendre Python,
et contribue à vous rendre plus employable.
– De nombreux packages sont disponibles pour améliorer Python et fonctionnent bien avec Sage. En effet,
de nombreux packages de ce type sont déjà fournis avec Sage.
– Comme mentionné précédemment, vous pouvez souvent accélérer un programme en le « cythonisant ».
• Python est un langage moderne, offrant de nombreuses façons d'exprimer une solution élégante à un
problème. En apprenant Sage, vous apprenez les bonnes pratiques de programmation.
Gardez à l'esprit que Python et Sage ne sont pas la même chose, ni un sous-ensemble de l'autre. sauge
les commandes ne fonctionnent pas en Python simple, et certaines commandes Python ne fonctionnent pas de la même manière
en Sage qu'ils le feraient en Python.
Comment démarrer avec Sage ?
Le moyen le plus simple est probablement de visiter le serveur CoCalc sur cocalc.com, Enregistrez-vous pour avoir accès à un compte gratuit,
démarrer un projet et créer une feuille de calcul Sage. Nous exhortons fortement le lecteur à trouver la pâte pour
un abonnement, qui au moment de la rédaction de cet article coûte 7 $/mois et donne accès à plus rapidement, plus
serveurs fiables. Vous pouvez l'utiliser gratuitement, mais pour diverses raisons, les serveurs gratuits se réinitialisent parfois
sur toi. (Si vous faites partie d'une classe, cependant, demandez à l'instructeur si le département a poney
pour un forfait classe; la remise est substantielle.) L'inconvénient de cette approche est que vous avez
payer pour obtenir un bon service. L'avantage est que vous disposez toujours d'une version raisonnablement à jour
de Sage à portée de main, et vous n'avez pas à vous soucier d'un crash matériel qui efface tout vos données, car elles sont stockées sur des serveurs qui reposent sur plusieurs sauvegardes et mécanismes de secours.
Une autre façon d'utiliser Sage consiste à utiliser un serveur en ligne qui n'est pas un serveur CoCalc. Cela vous oblige
connaître quelqu'un qui connaît quelqu'un qui... connaît quelqu'un qui gère un serveur, vous pouvez

utiliser. De nombreuses institutions le font, dont celle qui emploie les auteurs ; en fait, notre département maintient au moins deux de ces serveurs au moment de la rédaction de cet article. L'inconvénient de cette
approche est que vous dépendez du mainteneur du serveur pour maintenir le logiciel à jour et le
données sauvegardées. En effet, l'un des serveurs de notre service exécute une version de Sage dépassée depuis des années
de date.
La dernière façon d'utiliser Sage est à partir d'une copie sur votre propre machine. Vous pouvez le télécharger depuis
www.sagemath.org et installez-le sur votre ordinateur. (Recherchez le lien vers « Téléchargements ».)
• Si votre machine exécute Linux, il s'agit d'un processus relativement simple, même si vous avez peut-être
pour installer certains packages via votre gestionnaire de packages. (Dans le passé, nous avons dû
installer un système appelé m4 . Nous ne nous souvenons pas de ce que c'est.) Des binaires sont disponibles pour certains
distributions Linux ; Ubuntu est typiquement l'un d'entre eux, et Fedora a été de temps à autre temps. Pour le reste, vous devrez probablement télécharger la source Sage et la compiler sur votre
l'ordinateur. La bonne nouvelle est que cela est généralement assez indolore, tant que vous avez déjà

4 Au moins un des auteurs a effectivement essayé cela lorsqu'il était à l'université. En fait, les statistiques étaient la seule classe
qui a brisé sa résistance à l'utilisation d'une calculatrice. La capacité de la calculatrice à effectuer un calcul exact de fractions
l'impressionna ; jusque-là, il n'avait vu que des calculatrices faire de l'arithmétique avec des fractions en virgule flottante.

---

installé les packages requis, et ceux-ci sont répertoriés dans les instructions. La mauvaise nouvelle est que
l'installation à partir des sources prend beaucoup de temps, alors préparez-vous à attendre vite.
• Si votre machine exécute OSX, essayez de télécharger un binaire pour votre architecture. Tu peux essayer
compiler à partir des sources, comme avec Linux, mais dans ce cas, espérons qu'Apple n'a pas récemment
Xcode « mis à niveau », car vous avez besoin de Xcode pour installer Sage et de chaque version majeure de
Xcode casse généralement Sage d'une manière ou d'une autre.5

• Si votre machine fonctionne sous Windows, vous êtes dans la position inhabituelle d'avoir à subir ce
Les utilisateurs Linux et OSX souffrent généralement : Sage ne fonctionne pas nativement sur Windows, vous
devez l'utiliser via une machine virtuelle. Cela peut être un peu fastidieux à mettre en place, mais une fois
vous comprenez, ça marche bien. La bonne façon de procéder a changé plusieurs fois
au fil des ans, nous sommes donc réticents à donner des conseils plus précis que cela. La bonne nouvelle
est que les instructions sur l'installation et l'exécution de Sage seront disponibles sur le site Web.
Une fois que l'une de ces méthodes est opérationnelle, vous commencez à l'utiliser !
Feuille de calcul ou ligne de commande ?
Il existe deux manières typiques d'utiliser Sage : à partir d'un navigateur, dans ce qu'on appelle une feuille de calcul Sage, ou
à partir de la ligne de commande. Si vous avez installé Sage sur votre machine, vous pouvez faire les deux ; voir le
section sur la ligne de commande Sage pour voir comment démarrer une feuille de calcul Sage à partir de la ligne de commande.
Feuilles de travail de la sauge. Si vous avez accès à Sage via un navigateur Web (soit CoCalc ou un autre
serveur de ligne), vous préférerez probablement travailler avec une feuille de calcul Sage. Nous recommandons nos étudiants
pour commencer avec Sage de cette manière, car la feuille de calcul fournit un environnement plus convivial
ment : il est facile de revoir et de réexécuter les commandes, d'ajouter du texte pour l'organisation et la narration,
et en outre pour enregistrer votre session entière, puis rechargez plus tard le travail et la sortie. Tu ne peux pas faire
tout cela avec la ligne de commande.
Lorsque vous démarrez une feuille de calcul, vous devriez voir l'un de ces deux écrans :
Serveur indépendant

5 Sage n'est en aucun cas le seul progiciel à subir cette conséquence.

CoCalc
Vous pouvez (et devez) changer le titre.
• Dans le serveur indépendant, vous pouvez le faire au début avec le "Renommer le travail-feuille de dialogue" que vous voyez dans la capture d'écran. Vous pouvez le faire plus tard en cliquant sur le titre (dans le
en haut à gauche, actuellement "Sans titre") et la même boîte de dialogue apparaîtra.

• Dans CoCalc, vous pouvez le faire en cliquant sur le i entouré en haut à gauche et en choisissant
"Renommer...". Un nouvel écran apparaîtra, vous invitant à renommer le fichier. S'assurer vous gardez le .sagews ajouté à la fin.
Il existe d'autres options avec lesquelles vous pouvez jouer, mais pour l'instant, nous vous recommandons de passer à
chapitre suivant, car la plupart de ces options sont de peu d'importance pour notre propos. Ceux
nous avons besoin de discuter en temps voulu.
Sage en ligne de commande. Si vous choisissez d'exécuter Sage à partir de la ligne de commande, vous devez ouvrir un
shell, également appelé invite de ligne de commande. Vous verrez une sorte d'invite, qui peut varier assez
un peu; chaque fois que nous entendons une invite shell, nous mettrons simplement un symbole bleu supérieur à : > . Au
invite, tapez sage , appuyez sur Entrée , puis attendez que Sage démarre. Cela peut prendre quelques secondes, mais
vous finirez par voir quelque chose à cet effet :

```
> sauge
SageMath version 6.7, date de sortie :
2015-05-17
Tapez "notebook()" pour l'interface de notebook basée sur un navigateur.
Tapez "help()" pour obtenir de l'aide.
sauge:
_
```

Le trait de soulignement (_) peut en fait ressembler à une petite boîte sur votre système. Une fois que vous voyez cela, vous êtes
en forme pour le prochain chapitre. Si vous ne le voyez pas, ou si vous voyez une sorte d'erreur, vous devez
pour parler avec votre instructeur ou le support technique et voir ce qui s'est mal passé.
Si vous souhaitez exécuter une feuille de calcul Sage dans un navigateur, mais ne souhaitez pas exécuter CoCalc et ne
avoir accès à un autre serveur, tapez notebook() et appuyez sur Entrée . Vous verrez beaucoup de messages, par
exemple:

```
sauge:
carnet()
Les fichiers du bloc-notes sont stockés dans :
sage_notebook.sagenb
Ouvrez votre navigateur Web sur http://localhost:8080
Exécution de twistd --pidfile="sage_notebook.sagenb/sagenb.pid" -ny
"sage_notebook.sagenb/twistedconf.tac"
```

/Applications/sage-6.7-untouched/local/lib/python2.7/site-packages/
Crypto/Util/numéro.py:57 :
PowmInsecureAvertissement :
N'utilise pas
mpz_powm_sec.
Vous devez reconstruire en utilisant libgmp >= 5 pour éviter le timing
vulnérabilité aux attaques.
_warn("N'utilise pas mpz_powm_sec.
Vous devriez reconstruire en utilisant libgmp >=
5 pour éviter la vulnérabilité d'attaque de synchronisation.", PowmInsecureWarning)
2016-05-27 14:30:49+0300 [-] Journal ouvert.
2016-05-27 14:30:49+0300 [-] twisted 14.0.2
(/Applications/sage-6.7-untouched/local/bin/python 2.7.8) démarrage
en haut.
2016-05-27 14:30:49+0300 [-] classe de réacteur :
twisted.internet.selectreactor.SelectReactor.
2016-05-27 14:30:49+0300 [-] QuietSite à partir du 8080 2016-05-27
14:30:49+0300 [-] Démarrage de l'usine <__builtin__.QuietSite instance
au 0x1181bb638>

Pour la plupart, vous n'avez pas à vous soucier de ces messages.[6] Vous n'avez même pas besoin de
suivez les instructions pour ouvrir votre navigateur Web sur ce site ; sur de nombreuses machines, le navigateur
ouvrir la page Web automatiquement. De plus, il faudra quelques secondes pour que les choses commencent, alors
asseyez-vous et détendez-vous quelques secondes. Si votre navigateur ne s'ouvre pas, pas de panique ! Ouvrez-le vous-même
et voyez si l'adresse Web que votre Sage vous conseille fonctionne. Si c'est le cas, vous êtes en forme pour la prochaine
chapitre.

Si ce n'est pas le cas,

PANIQUE!

Oui, il est vraiment normal de paniquer de temps en temps. Sortez-le de votre système. Une fois que vous avez terminé, regardez
attentivement les messages, et voyez s'il y en a des messages d'erreur ; ceux-ci seraient utiles. Puis visite
https://groups.google.com/forum/#!forum/sage-support
et recherchez pour voir si ce message d'erreur a été discuté. Si ce n'est pas le cas, commencez un nouveau message, en demandant
ce qui ne va pas.

Obtenir de l'aide

Si vous lisez ceci dans le cadre d'un cours, votre instructeur devrait vous aider. (Peut-être pas très
utile, mais utile tout de même.) Nous avons déjà mentionné le dernier chapitre du forum sage-support .

En dehors de ces options, Sage répondra seul à de nombreuses questions.

Docstrings. Si vous voulez savoir comment fonctionne une commande, tapez le nom de la commande,
suivi d'un point d'interrogation, puis exécutez la commande. Sage vous fournira des informations utiles.
sur la commande, généralement avec des exemples.

6 Eh bien, peut-être les avertissements de sécurité concernant libgmp , si vous les voyez. Je devrais me renseigner là-dessus.

OBTENIR DE L'AIDE
22

```
sage : simplifier ?
Signature:
simplifier(f)
Chaîne de documentation :
Simplifier l'expression f.
EXEMPLES : On simplifie l'expression i + x - x.
sauge:
f = I + x - x; simplifier(f)
je
En fait, l'impression de f donne la même chose - c'est-à-dire la
forme.
Init docstring :
x.__init__(...)
initialise x ; voir l'aide (tapez (x))
à signer
Déposer:
/Applications/sage-6.7-untouched/local/lib/python2.7/site-
packages/sage/calcul/functional.py
Taper:
une fonction
```

Recherche de méthodes pour les objets. Une autre façon d'obtenir de l'aide est de voir quelles commandes un objet
acceptera comme méthodes. Une « méthode » est une commande très spécifique à un objet Sage particulier ;
vous l'invoquez en tapant le nom de l'objet, suivi d'un point, puis le nom de la méthode.[7] à recherchez les noms de toutes les méthodes qu'un objet accepte, tapez son nom, suivi d'un point, puis appuyez sur
la touche Tab .

Par exemple, simplifie() ne fonctionne pas très bien sur l'expression suivante :[8]

```
sauge : rats = 1/x + 1/2
sauge : simplifier(rats)
1/x + 1/2
```

Nous voulons vraiment simplifier cette expression aussi complètement que possible, alors voyons si elle accepte une méthode
qui effectuera une simplification plus poussée. Tapez les rats. (y compris la période !) et puis appuyez sur Tabulation ; vous devriez voir plus de 200 méthodes possibles. Certains d'entre eux ne sont pas vraiment appropriés pour
l'expression; nous n'entrerons pas dans les raisons de cela, mais si vous regardez

attentivement, vous devriez trouver
au moins deux méthodes utiles. L'un d'eux est full_simplify() .

```
sage : rats.full_simplify()
1/2*(x + 2)/x
```

C'est une façon d'écrire un peu alambiquée (x+2)/(2x) , mais au moins c'est simplifié !

7 Un autre terme pour « méthode » est « message » ; les deux termes sont utilisés en informatique, mais "méthode" est le jargon
pour Sage.

8 Cela fonctionne réellement dans certaines versions de Sage, telles que Sage 6.7, mais pas dans d'autres, telles que 8.1. Si ça simplifie
dans votre version, prétendez simplement que ce n'est pas le cas pour l'argument, et suivez, car ce n'est pas le point en tous cas.

L'idée de chasser à travers plus de 200 méthodes possibles peut sembler intimidante, mais beaucoup
les objets acceptent très peu de méthodes. En pratique, ce n'est pas une technique si difficile, car il y a
plusieurs façons de rechercher dans la liste.

Des exercices

Vrai faux. Si la déclaration est fausse, remplacez-la par une déclaration vraie.

1. Ceci est juste un autre manuel de programmation.
2. Les mathématiques consistent à compter des nombres.
3. Alors que les étudiants ne se soucient pas beaucoup des fractions, les ordinateurs trouvent plus facile de travailler avec des fractions
qu'avec les nombres à virgule flottante.
4. Les logiciels « gratuits » sont écrits par des fainéants sans emploi vivant dans les sous-sols de leurs parents.
5. Tous les résultats mathématiques célèbres ont été appréciés dès le départ pour leurs preuves claires et élégantes.
6. Le bytecode n'est techniquement pas un langage interprété.
7. Les langues interprétées sont appréciées avant tout pour leur rapidité.
8. Certains logiciels de taille importante sont exempts de bogues.
9. Les mathématiques abstraites sont inutiles dans le monde réel.
10. Il est facile de vérifier et d'améliorer les progiciels mathématiques propriétaires.
11. Bonus : La réponse à toutes ces questions Vrai/Faux est « Faux ».

Choix multiple.

1. Lequel des éléments suivants n'est pas un exemple de système de calcul formel ?

A. Un package qui exécute l'arithmétique des fractions sans aucune erreur d'arrondi.
B. Un package qui attribue des numéros aux notes de musique et utilise des fractales pour produire de la nouvelle musique.
C. Un progiciel qui exécute l'arithmétique polynomiale avec des valeurs approximatives pour

les coefficients.
D. Un package qui se concentre sur des ensembles d'objets abstraits définis par des propriétés précises.
2. Laquelle des étapes suivantes n'est pas une étape du processus habituel de compilation d'un programme ?
A. Le compilateur lit la source à partir d'un fichier.
B. Le compilateur traduit chaque symbole en une combinaison de signaux d'activation et de désactivation.
C. Le compilateur enregistre la traduction dans un fichier, appelé exécutable ou bibliothèque.
D. Le compilateur exécute les commandes traduites immédiatement avant de quitter.
3. L'objectif principal de Sage est quel type de mathématiques computationnelles ?
A. calcul numérique avec des valeurs approximatives
B. calcul statistique avec des valeurs probables
C. calcul symbolique avec des valeurs exactes
D. calcul flou avec des valeurs incertaines
4. Lequel des systèmes de calcul formel suivants utiliseriez-vous pour calculer une dérivée ?
A. Maxima
B. PARI
C. Bourse
D. S INGULIER
5. Lequel des langages informatiques bien connus suivants essaie de combler le fossé entre langages interprétés et compilés ?
A. C/C++
B. Fortran
C. Java
D. Python

---

6. Parmi les propositions suivantes, laquelle est la principale motivation du mouvement pour les mathématiques « libres » ?
Logiciel?
A. Antipathie pour la censure
B. Collaboration internationale
C. Manque de subventions
D. Vérifiabilité des résultats
7. Lequel des mathématiciens suivants est célèbre malgré avoir popularisé un « fait » qui était très mal?
A. Pierre de Fermat
B. Marin Mersenne
C. William Stein

D. Andrew Wiles
8. Lequel des mathématiciens suivants est célèbre malgré son travail quotidien non scientifique ?
A. Pierre de Fermat
B. Marin Mersenne
C. William Stein
D. Andrew Wiles
9. Parmi les propositions suivantes, laquelle n'est pas un avantage à travailler avec Sage ?
A. Vous acquérez des compétences pratiques que les employeurs apprécient.
B. Vous travaillez avec des logiciels de pointe.
C. Vous n'avez plus à vous soucier d'obtenir la bonne réponse.
D. Vous pouvez utiliser le matériel appris ici dans d'autres classes.
10. De quelle manière Sage fonctionne-t-il avec Python ?
A. L'interface de Sage est essentiellement une interface Python.
B. Sage est une bibliothèque Python que vous pouvez charger dans n'importe quel interpréteur Python.
C. Python est l'un des systèmes de calcul formel de Sage.
D. Sage utilise le compilateur Cython pour compiler tous les programmes Python.
Bonus : quelle réponse est correcte ?
R. Le suivant.
B. Le suivant.
C. Le suivant.
D. Le premier.
Réponse courte.
1. Expliquez comment la citation au début de ce chapitre est liée à sa thèse principale.
2. Décrire une analogie du monde réel pour la différence entre compilé, interprété et bytecode langues.
3. Un problème courant dans les manuels de mathématiques est de calculer la 1001e dérivée de cos x.
La meilleure façon de trouver la réponse n'est pas de calculer toutes les 1001 dérivées ; à la place, vous calculez un
quelques-uns, remarquent un motif et déduire rapidement ce que devrait être la 1001e dérivée. Expliquer comment
cela se compare à ce que nous voulons que vous reteniez de ce texte.
4. Tous les mathématiciens ne trouvent pas convaincants les arguments en faveur du logiciel libre, et l'utilisation
de logiciels propriétaires est répandu. Pourquoi pensez-vous que ce soit le cas?
5. Même si Sage est moins populaire qu'un package propriétaire comme Maple ou Mathematica, apprendre
Sage peut également faciliter le travail avec ces packages. Pourquoi?

CHAPITRE 2

Calculs de base

Qu'est-ce qu'il y a dans un nom? ce que nous appelons une rose / Par n'importe quel autre nom sentirait comme
doux... (Shakespeare)

Créez une nouvelle feuille de calcul et appelez-la « Ma première feuille de calcul Sage ».

Pour décrire l'interaction avec Sage, nous utilisons le format suivant :

```
sage : quelques entrées
une certaine sortie
```

Le texte " une entrée " indique une commande que vous tapez dans Sage. Si vous utilisez Sage à partir du
ligne de commande, vous le taperez à l' invite bleue " sage: " ; d'où la couleur bleue et l'étiquette.

Si vous utilisez Sage à partir d'une feuille de calcul, vous ne verrez pas d'invite ; à la place, vous taperez le
commande dans une « cellule » qui, dans certaines versions de Sage, est délimitée par une petite boîte.

Pour exécuter la commande :

• sur la ligne de commande, appuyez sur la touche Entrée ou Retour ;

• dans une feuille de calcul, maintenez la touche Maj enfoncée et appuyez sur la touche Entrée ou Retour .

Sage interprétera et traitera ensuite votre commande. L'interface de la feuille de calcul affichera une couleur
ou une barre clignotante pendant que Sage le fait ; l'interface de ligne de commande fera simplement une pause.

Une fois la sortie terminée, vous verrez le texte indiqué par " une sortie ". Si le com-
mand réussi, vous verrez une réponse qui semble plus ou moins sensée. Si une erreur s'est produite,
le texte contiendra beaucoup d'informations, certaines en rouge (surtout la dernière ligne, qui est tout ce que nous
copiera) et n'aura probablement aucun sens à moins que vous ne connaissiez déjà Python. Par exemple:

```
sage : une mauvaise entrée
SomeError :
un message pour expliquer l'erreur
```

Nous expliquerons plusieurs types d'erreurs au fur et à mesure que nous progressons dans nos explorations de Sage. En guise de con-

Pour plus de commodité, nous ajoutons des erreurs à l'index du manuel. Si vous travaillez sur un calcul,
et vous rencontrez une erreur que vous ne reconnaissez pas, voyez si vous pouvez la trouver dans l'index ; si c'est le cas, le
les numéros de page auxquels il fait référence s'avéreront utiles.

Voici un exemple de calcul réussi ; vous devriez essayer vous-même maintenant et faire

sûr que vous obtenez le même résultat.

```
sauge : 2 + 3
5
```

Cela semble rassurant; au moins Sage peut le faire ! En voici un où l'utilisateur a oublié de lâcher le
touche Shift avant d'appuyer sur le chiffre 3.

```
25
```

YER ARITHMÉTIQUE DE BASE

```
26
sauge : 2 + #
Erreur de syntaxe:
Syntaxe invalide
```

Si vous l'exécutez réellement dans Sage, vous remarquerez qu'il y a un peu plus de texte dans l'erreur, ainsi que beaucoup
des changements de couleur. Dans notre cas, cela ressemble à :

```
Fichier "<ipython-input-3-7c2c726856a7>" , ligne 1
Entier(2) + #
^
Erreur de syntaxe:
Syntaxe invalide
```

Vous remarquerez que nous n'avons copié que la dernière ligne, qui précise le type d'erreur ! Lorsque cela est nécessaire pour
la discussion, nous inclurons parfois aussi le reste des informations, mais vous pouvez souvent
comprendre le problème juste en regardant la dernière ligne.

Il peut arriver que la ligne que vous devez taper soit trop longue pour tenir sur une seule ligne. Il arrivera à
les auteurs de ce texte, car nous n'avons pas beaucoup d'espace horizontal sur la page. Dans des cas comme
ceci, vous pouvez continuer à taper, ce qui peut rendre le code plus difficile à lire, ou vous pouvez dire à Sage que vous voulez
pour continuer sur la ligne suivante en tapant une barre oblique inverse \ , que Sage interprète comme un « saut de ligne »,
puis en appuyant sur Entrée . Nous le ferons assez régulièrement pour aider à rendre le code plus lisible. Il y a
pas besoin d'attendre jusqu'à la fin de la ligne pour cela, et parfois cela peut être plus
lisible pour ajouter le saut de ligne au début. Par exemple:

```
sage : 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 \
.... :
+ 11 + 12 + 13 + 14 + 15
120
```

(Lors de l'utilisation d'une feuille de calcul Sage). L'un des avantages de l'utilisation des feuilles de calcul Sage est que vous pouvez utiliser
Commandes HTML pour ajouter une narration explicative à votre travail. Tapez simplement

```
sage : %html
```

...et tout ce qui suit cette ligne sera considéré comme du texte HTML. La barre d'outils va changer
pour permettre le formatage HTML, mais si vous êtes familier avec les balises HTML, vous pouvez les ajouter
directement. Vous pouvez le formater de la même manière que vous exécutez une commande Sage : maintenez Shift , appuyez sur Enter .
Si nécessaire, vous pouvez ensuite modifier à nouveau la cellule HTML en double-cliquant dessus. C'est énormément
utile pour diviser une longue feuille de calcul en sections, avec des en-têtes qui organisent les parties liées
du travail.

Yer arithmétique de base

Comme vous pouvez vous y attendre, Sage propose les mêmes opérations arithmétiques de base que vous pouvez trouver dans n'importe quel
calculatrice. En voici quelques-uns qui s'avéreront utiles.[1]

1 Vous pouvez également taper a ^ b pour l'exponentiation, mais pour diverses raisons nous ne le recommandons pas : dans certaines situations,
Sage l'interprétera comme un opérateur différent.

---

a + b
ajoute a et b

a - b
soustrait a et b

a * b
multiplie a et b

a / b
trouve le rapport de division de a par b

a // b
trouve le quotient de la division de a par b

un % b
trouve le reste de la division de a par b

a ** b
élève a à la puissance b

sqrt(a) la racine carrée d'un

abdos (un)
la valeur absolue d'un

T ABLEAU 1. Opérateurs Sage pour l'arithmétique de base

Allez-y et essayez-les avec quelques chiffres, à la fois approximatifs et exacts.

Yer comparaisons de base. Sage peut aussi comparer des objets, dans une certaine mesure.

a > b est a strictement supérieur à b ?

a >= b est a supérieur ou égal à b ?
a == b est a égal à b ?
a <= b est a inférieur ou égal à b ?
a < b est a strictement inférieur à b
TABLEAU 2. Opérateurs Sage pour les comparaisons de base
Lors de l'utilisation de ces comparaisons, Sage renverra *True* ou *False* , qui correspondent évidemment à
"Oui ou non."

```
sauge : 2 < 3
Vrai
sauge : 2 > 3
Faux
```

Il faut faire attention à la comparaison d'égalité, car le signe est doublé. Des problèmes surgiront si
Tu oublies ça.

```
sauge : 2 = 3
Erreur de valeur :
Le nom "2" n'est pas un identifiant Python valide.
```

Si vous voyez ce message d'erreur, le problème est presque certainement dû à l'utilisation d'un seul égal
signe quand vous en avez besoin de deux.
En plus de comparer des nombres, de nombreux objets symboliques peuvent être comparés. nous ne parlerons pas
sur les ensembles depuis un certain temps, par exemple, mais il existe un ordre naturel des ensembles basé sur des sous-ensembles
properties, et Sage comparera les ensembles en fonction de ce fait. Considérons, par exemple, les ensembles {2,3},
{2,4} et {2,3,4,5} :

```
sauge : {2,3} < {2,3,4,5}
Vrai
sauge : {2,3,4,5} > {2,3}
Vrai
sage : {2,3} < {2,4}
Faux
```

Valeurs exactes ou approximatives. Sage propose des symboles pour représenter les valeurs exactes de certains im-
nombres importants; ceux-ci doivent être faciles à mémoriser.[2]

pi

π , le rapport de la circonférence d'un cercle à son rayon

e

e = limite
$x \to \infty$
1+

1 X x , ou alors, la valeur de a telle que la dérivée de a x soit a x	
je i, le nombre « imaginaire » (i 2 = −1)	
oo ou ∞, infini (illimité ci-dessus)	
+Infini	
-oo ou −∞ (non borné ci-dessous)	
-Infini	

TABLEAU 3. Symboles sages pour les constantes importantes

Nous mettons ces identifiants en italique dans le texte, à la fois pour faciliter la lisibilité et pour souligner qu'ils
doit représenter des valeurs fixes.[3] Sage ne les met pas en italique en sortie.

Nous avons déjà expliqué que la force d'un système de calcul formel comme Sage réside dans sa capacité
pour manipuler des valeurs exactes, plutôt que des valeurs approximatives. Néanmoins, nous avons parfois besoin de
calculer avec des valeurs approximatives, et Sage le permet également.

Pour utiliser l'arithmétique approximative, tapez au moins un nombre sous forme décimale ! Pour voir comment cela fonctionne, considérez les commandes suivantes et leurs résultats.

```
sauge : 2/3
2/3
sauge : 2./3
0.666666666666667
```

Dans le premier cas, vous avez entré des valeurs exactes, donc Sage vous donne le quotient exact de la division 2 par 3,
la fraction 2/3 . Dans le second cas, vous spécifiez au moins une décimale, vous obtenez donc le
quotient approximatif de la division de 2 par 3. Cela peut ressembler à une virgule flottante, mais ce n'est pas tout à fait ; c'est
en fait, un objet Sage appelle un RealNumber . Ne vous inquiétez pas de ces détails pour le moment.

2 Il est également possible d'utiliser *i* pour le nombre imaginaire, mais comme il est courant d'utiliser i comme variable, nous essayons d'éviter
cette.

3 Comme nous le notons à la page 29, cependant, Sage n'a pas de vraies constantes, donc un utilisateur ou un programme peut changer les symboles'
significations.

Constantes, variables et indéterminés

Une subtile distinction d'usage. Les symboles mathématiques e, π et i représentent des constantes,
nous entendons par là que leurs valeurs sont définies et fixes. Si quelqu'un écrit l'équation d'Euler,

$$e^{i\pi} + 1 = 0,$$

alors tous les mathématiciens instruits connaîtront les valeurs de e, i, et π, même si cela est le premier
fois qu'ils voient l'équation. Grâce aux symboles que nous avons décrits ci-dessus, cela est vrai dans Sage,
ainsi que:

```
sage : e**(I*pi) + 1
0
```

Bien sûr, il y a des moments où l'un de ces symboles peut signifier autre chose. Par exemple,

$$x_i$$

fait référence au ième nombre dans une liste, pas à une manipulation de x par le nombre imaginaire.

D'autres symboles à une lettre en mathématiques représentent l'un des deux types de variables, par lesquelles
nous voulons dire que leurs valeurs ne sont pas nécessairement fixées lors d'un problème. Le symbole x, par exemple, peut
représentent l'une quelconque d'un nombre infini de valeurs. Les mathématiciens travaillent généralement avec deux types de
variables; et dans le langage ordinaire, nous avons tendance à qualifier les deux types de variables, mais en fait il y a
une distinction importante, que l'on peut voir dans une expression aussi simple que le polynôme

$$x^2 - c^2.$$

• Dans de nombreuses situations, des valeurs spécifiques pour x et c sont très importantes. L'équation

$$x^2 - c^2 = 2$$

n'est pas toujours vrai ; cela dépend des valeurs de x et c, et en pratique on essaie souvent de trouver les valeurs pour lesquelles cela est vrai.

• Dans d'autres situations, cependant, x et c représentent des valeurs arbitraires. L'équation

$$x^2 - c^2 = (x + c)(x - c)$$

est vrai quelles que soient les valeurs de x et c.

Cela est très important pour les systèmes de calcul formel, car un symbole peut également faire référence à
une valeur spécifique ou indéterminée, et nous devons prêter attention à cette distinction de temps en temps
temps. Nous conviendrons de la convention suivante :

• Lorsqu'un symbole a reçu une valeur spécifique, nous l'appelons généralement une variable,

car
nous pouvons modifier cette valeur à tout moment.
• Lorsque nous n'avons pas l'intention de changer la valeur d'une variable, nous l'appelons une constante.
• Lorsqu'un symbole ne doit contenir aucune valeur spécifique, mais est purement symbolique pour un
valeur d'un certain ensemble, nous l'appelons indéterminée.
Sage ne fournit qu'un indéterminé au démarrage : `x` . Si vous voulez travailler avec un autre indéterminé
nate, tel que `y` , vous devez le créer.
Sage n'offre pas de moyen de définir vos propres constantes, comme certains langages de programmation
fais. Dans Sage, une constante est juste une variable que vous essayez vraiment de ne pas changer.

Remise à zéro d'une variable ou indéterminée. Étant donné que Sage n'a aucun moyen pour vous de définir la véritable con-
stants, vous pouvez, si vous le souhaitez, réaffecter la valeur de *I* à autre chose, et il est très probable que
vous le ferez un jour, dans certaines circonstances. Si cela se produit, il est facile de corriger avec la `réinitialisation ()`
commander; inclure le symbole entre guillemets à l'intérieur des parenthèses.

```
sage : je^2 + 1
0
sage : je = 2
sage : je^2 + 1
5
sage : réinitialiser('je')
sage : je^2 + 1
0
```

Création de variables et d'indéterminés. Pour créer une variable, utilisez la construction d'affectation,
identifiant `=` expression
où identifiant est un nom légitime pour un identifiant d'un symbole et expression est un nom légitime
expression mathématique. Par exemple,

```
sage : sqrt2 = sqrt(2)
```

attribue la valeur 2 au symbole `sqrt2` . Après avoir effectué cette affectation, vous pouvez utiliser le
le symbole `sqrt2` dans n'importe quelle expression, et Sage le verra comme 2. Par exemple :

```
sauge : sqrt2**2
2
```

Contrairement aux langages de programmation qui vous permettent d'affecter une seule variable à la fois, Sage
hérite de l'affectation plus flexible de Python, qui vous permet d'affecter de nombreuses variables en une seule
va. Par example,

```
sage : sqrt2, sqrt3 = sqrt(2), sqrt(3)
sauge : sqrt2**2
2
sauge : sqrt3**4
9
```

La première ligne de cette séquence d'instructions affecte à la fois 2 et 3 aux variables `sqrt2` et `sqrt3` , dans cet ordre. Les deux déclarations suivantes montrent que les affectations étaient en effet cor-
rect. Pour cette raison, vous pouvez probablement voir que la déclaration

```
sage : sqrt2, sqrt3 = sqrt3, sqrt2
```

---

a pour effet d'échanger les valeurs de `sqrt2` et `sqrt3` .
Pour créer un indéterminé, Sage fournit une commande, `var()` . Tapez le nom que vous souhaitez
indéterminé à avoir entre parenthèses, entre guillemets. Vous pouvez créer plusieurs de ces indéter-
miner en les listant également entre les guillemets ; laissez juste un espace entre chaque nom. Si
réussi, Sage imprimera les noms des indéterminés nouvellement créés entre parenthèses.
Par example:

```
sage : var('y z')
(y, z)
```

Vous pouvez ensuite manipuler ces variables à votre guise.

```
sage : (y + z) + (z - y)
2*z
```

Identifiants valides. Chaque variable ou indéterminée a besoin d'un identifiant valide. Sage accepte l'iden-
noms de tifier en utilisant les séquences de caractères suivantes :

- Le nom doit commencer par une lettre (majuscule ou minuscule) ou le trait de soulignement (_).
- Le nom peut contenir n'importe quel mélange de lettres, de chiffres ou du trait de soulignement.
- Le nom ne peut pas être un mot réservé, également appelé mot-clé. Vous rencontrerez ces tout au long du cours, mais il est peu probable que vous les choisissiez.

Contrairement à la convention mathématique, les noms des variables et des indéterminés peuvent être plus longs que
un symbole ; nous le voyons déjà dans `pi` , que Sage propose au départ. Dans de nombreux

cas, un nom
qui est plus long qu'un symbole peut être compris beaucoup plus facilement qu'un nom qui n'est que
un symbole ; comparer, par exemple, d à la dérivée . Par contre, si un nom est trop long
et vous l'utilisez à plusieurs reprises, il devient fastidieux à taper et peut même rendre un programme plus difficile à saisir
comprendre. Une personne alphabétisée en mathématiques, par exemple, comprendrait aussi bien ddx que
dérivée , et pourrait bien préférer la première à la seconde. En général, nous ne recommanderions pas plus
plus de six caractères dans un nom, mais si un nom plus long est utile pour plus de clarté, vous pouvez ne pas en tenir compte
directive, et devrait, tout comme nous le ferons.

Expressions et commandes pour les manipuler

Une expression mathématique consiste en toute combinaison significative de symboles mathématiques.
bols. Une de ces expressions est une équation, et dans Sage, vous pouvez affecter des équations comme valeurs de
variables. Vous devez absolument le faire de temps en temps, mais le signe égal a déjà un sens : il affecte la valeur d'une expression à un symbole pour créer une variable ! Se référer à un
équation, alors, vous utilisez deux signes égal à la suite :

```
sage : eq = x**2 - 3 == 1
```

Comme précédemment, si vous oubliez de doubler les signes égal, Sage vous donnera une erreur :

```
sage : eq = x**2 - 3 = 1
Erreur de syntaxe:
impossible d'attribuer à l'opérateur
```

Nous rappelons au lecteur que si vous voyez ce message d'erreur, le problème est presque certainement dû à
l'utilisation d'un seul signe égal lorsque vous en avez besoin de deux.

Il est souvent utile de réécrire des expressions de différentes manières, et un système de calcul formel
est, essentiellement, rien de plus qu'un outil sophistiqué pour réécrire des expressions. Voilà quelque
commandes utiles pour réécrire des expressions.

facteur (exp)
factoriser l'expression exp

simplifier(exp) simplifier un peu l'expression exp

développer(exp)
effectuer des multiplications et d'autres opérations sur l'expression exp
round(exp , n)
arrondir l'expression exp à n chiffres après la virgule
TABLEAU 4. Quelques commandes pour manipuler des expressions (il y en a beaucoup plus)
Quelques mots aux sages. Le symbole mathématique traditionnel de la multiplication est le simple × ou le symbole du point ·. Hélas, les claviers d'ordinateur n'ont aucun symbole par défaut ; toi
devez généralement utiliser une solution de contournement si vous souhaitez saisir le symbole. La solution de contournement traditionnelle
est de taper l'astérisque, et c'est ce que Sage utilise.
Il est courant en mathématiques d'omettre le symbole de multiplication : 2x, par exemple, ou abcd.
Vous ne pouvez pas faire cela dans Sage ; cela vous donnera diverses erreurs:

```
sauge : 2x
Erreur de syntaxe:
Syntaxe invalide
sage : var('abc d')
(a B c d)
sage : abcd
NameError :
le nom 'abcd' n'est pas défini
```

Dans les deux cas, Sage pense que vous essayez de taper le nom d'un identifiant, ce qui n'est pas le cas.
reconnaître:
• Pour 2x , vous ne pouvez pas commencer un nom d'identifiant par un chiffre donc l'erreur est de syntaxe.
• Pour abcd , Sage n'a aucun moyen pratique de savoir que vous voulez dire les produits de a , b , c ,
et d , plutôt qu'un identificateur différent nommé abcd , donc l'erreur est en fait l'un des Nom. (Rappelez-vous que dans Sage, contrairement à la plupart des mathématiques, les variables et les indéterminés
peut avoir des noms plus longs qu'un symbole.)
Ces deux expressions fonctionnent correctement si vous tapez le symbole de multiplication là où vous le souhaitez.
être.[4]
Fonctions transcendantales. Sage offre tout ce qu'offre une calculatrice scientifique. Voici certaines fonctions transcendantales communes que vous trouverez utiles tout au long de ce texte et dans
votre programme de mathématiques :
4 Dans certaines versions de Sage, il existe un moyen de faire fonctionner la multiplication sans écrire explicitement le multiplica-
symbole de tion dans ces circonstances, mais il n'est pas disponible par défaut, vous devrez donc exécuter une commande spéciale pour cela.
C'est une mauvaise idée pour les débutants, et vous auriez quand même à taper un espace entre les symboles de toute

façon, donc nous n'allons pas
décrire cette technique.

---

sin(a) , cos(a) , tan(a) ,
les fonctions trigonométriques,
cot(a) , sec(a) , csc(a)
évalué à un
arcsin(a) , arccos(a) , arctan(a) , les fonctions trigonométriques inverses,
arccot(a) , arcsec(a) , arccsc(a)
évalué à un
exp(a)
e a (synonyme de e**a)
ln(a) , log(a)
le logarithme népérien d'un
log_b(a , b) ou log(a , b)
la base du logarithme b d'un
T ABLEAU 5. Commandes sages pour les fonctions transcendantales communes
On peut aussi calculer les fonctions trigonométriques hyperboliques, et leurs inverses, en ajoutant h
à la fin du nom usuel, et avant la parenthèse gauche.
Vous avez peut-être remarqué que log(a) est synonyme de ln(a) . Ce n'est pas la coutume habituelle dans
Les manuels américains, où log(a) est synonyme de log 10 a. Pour calculer ce que les Américains appellent
le logarithme « commun », nous devons utiliser la commande log_b() :

```
sage : journal (100)
log (100)
sage : log_b(100,10)
2
```

Soyez prudent lorsque vous essayez d'utiliser ces fonctions dans des expressions plus volumineuses. C'est courant en mathématiques
pour écrire sin 2 π / 4 quand tu veux dire (sin(π / 4)) 2 . Sage ne fait pas cette distinction. Il y a vraiment
une seule façon de le faire correctement, et c'est d'écrire ce que vous voulez dire :

```
sage: sin^2(pi/4)
Erreur-type:
L'objet 'sage.rings.integer.Integer' n'est pas appelable
sage : (péché^2)(pi/4)
Erreur-type:
type(s) d'opérande non pris en charge pour ** ou pow() :
'Function_sin' et 'int'
sage : (sin(pi/4))^2
1/2
```

Lorsque vous voyez *TypeError* avec ces messages, il y a fort à parier que vous avez "tapé" quelque chose de mal.[5]

• Le premier type de TypeError devrait être facile à déboguer : recherchez un endroit où un nombre
est immédiatement suivi de l'ouverture de parenthèses — dans notre cas, 2(*pi*)/4 . Lorsque cela se produit, Sage pense que vous essayez d'utiliser 2 comme fonction. La façon dont Sage évalue
expressions, il voit sin $2(\pi/4)$
, et bien qu'il puisse ne pas nous sembler raisonnable de considérer $2(\pi/4)$ comme un fonction 2 évaluée au point $\pi/4$, c'est ainsi que Sage la voit.

• Le deuxième type de TypeError peut être plus difficile. En général, le problème est que vous êtes
essayer d'effectuer une opération avec deux objets pour lesquels Sage ne sait pas comment

5 Techniquement, Sage fait référence au « type » d'objet, pas à ce que vous avez « tapé », mais nous sommes suffisamment désespérés
rouler avec le jeu de mots.

---

effectuer l'opération. Neuf fois sur dix, vous avez tapé quelque chose de mal, alors re-examinez ce que vous avez tapé. Dans ce cas, vous avez tapé un raccourci pratique, qui Sage ne comprend pas (avec raison).

Fonctions mathématiques et substitution. Un aspect utile des indéterminés est que vous pouvez
substituer des valeurs en eux. Contrairement à l'attribution d'une valeur à une variable, les indéterminés ne retiennent pas
cette valeur après le calcul. Nous utilisons trois façons de substituer dans Sage.

La première consiste à utiliser la méthode subs() , qui est un raccourci pour la méthode substitut() . Rappelons qu'une méthode est une commande spécifique à un objet. Vous accédez à cette fonctionnalité dans le
manière suivante:

• Tapez le nom de l'expression, puis un point, puis subs(.

• Après la parenthèse, listez les devoirs. Il y a deux façons de faire ça:

– Énumérez chaque affectation sous forme d'équation : indéterminée = valeur.[6]

– Énumérez chaque devoir comme un « dictionnaire ». Pour ce faire, ouvrez une paire d'accolades, listez les
les inscriptions sous la forme indéterminée:valeur , en séparant chaque affectation par une virgule, puis fermez les bretelles.

• Fermez maintenant les parenthèses commençant après subs : subs(assignations) .

Voici un exemple:

```
sage : f = x**2
sage : f.subs(x=2)
4
```

```
sage : f.subs({x:2})
4
```

La première substitution illustre l'affectation via des équations ; la seconde illustre l'affectation via
dictionnaires. La deuxième approche est le moyen le plus fiable, sans erreur et sans avertissement de remplacer
dans toute expression mathématique qui contient des indéterminés.[7]
Une deuxième façon est de substituer sans spécifier le nom de la méthode !

```
sage : f(x=2)
4
sage : f({x:2})
4
```

Vous ne devez pas utiliser cette approche sans spécifier le nom de l'indéterminé. Si vous n'avez que
un indéterminé, comme avec `x` ci-dessus, vous pouvez oublier que vous devez le nommer. Dans ce cas, Sage
émettra ce qu'on appelle un avertissement :

6 Cela ne fonctionne pas toujours. Nous reviendrons sur le sujet plus tard.
7 En particulier, la deuxième approche fonctionne même à l'intérieur de fonctions définies par l'utilisateur, également appelées procédure, lorsque vous
vouloir passer un indéterminé comme argument à la procédure. On en parle plus tard.

```
sauge : f(2)
DépréciationAvertissement :
Substitution à l'aide de la syntaxe d'appel de fonction
et les arguments sans nom sont obsolètes et seront supprimés d'un
future version de Sage ; vous pouvez utiliser des arguments nommés à la place, comme
EXPR(x=..., y=...)
4
```

Cet avertissement peut ne pas apparaître sur la dernière ligne. Ce n'est pas une erreur, et Sage parvient à deviner le
substitution correcte et renvoie la bonne réponse après l'avertissement. En plus c'est sympa à propos de `DeprecationWarning` s est qu'ils n'apparaissent qu'une seule fois, même si vous continuez à faire le même
"erreur." C'est quand même un peu effrayant de le voir là, et il est tout à fait possible que la menace de
supprimer cette fonctionnalité de devinette se réalisera,[8] donc essayez de ne pas faire cette erreur. Cela peut être
un vrai problème si vous fournissez trop de valeurs ; Sage ne saura pas quoi assigner où, et se plaindre:

```
sage : f(3,2,1)
Erreur de valeur :
le nombre d'arguments doit être inférieur ou égal à
1
```

Cela ne se produira pas lorsque vous spécifiez des affectations, car Sage sait ce qui va où :

```
sage : f = x^2 - y^2
sage : f(x=3,w=2,z=1)
-y^2 + 9
```

La troisième façon de substituer est utile lorsque vous utilisez une expression mathématique appelée un
une fonction. Rappelez-vous qu'une fonction mappe les éléments d'un ensemble, appelé le domaine, aux éléments de
un autre ensemble, appelé la plage, de manière à ce que chaque entrée ait une sortie bien définie. Définir
et l'utilisation des fonctions est simple dans Sage :

```
sage : f(x) = x**2
sauge : f(2)
4
```

Les fonctions ont également la propriété utile de définir tout indéterminé qui apparaît entre les parenthèses de leur définition. Vous pouvez ensuite utiliser ces indéterminés dans d'autres contextes
sans utiliser au préalable la commande var() .

8 L'avertissement est apparu depuis quelques années maintenant.

---

lim(f , x = a)
calculer la limite bilatérale de f (x) à x = a
ou limit(f , x = a)
lim(f , x = a , dir= direction)
calculer la limite unilatérale de f (x) à x = a, avec
ou limit(f , x = a , dir= direction)
direction un de 'gauche' ou 'droite'
diff(f , x)
calculer la dérivée de f (x)
ou dérivée(f , x)
par rapport à x
diff(f , x , m)
calculer la dérivée mième
ou dérivée(f , x , m)
de f (x) par rapport à x
intégrale(f , x)
calculer l'intégrale indéfinie (primitive)
ou intégrer(f , x)
de f (x) par rapport à x
intégrale(f , x , a , b)
calculer l'intégrale définie sur [a, b]
ou intégrer(f , x , a , b)

de f (x) par rapport à x

T ABLEAU 6. Commandes pour le calcul exact des limites, des dérivées et des intégrales

```
sage : f(w,z) = 4*w**2 - 4*z**2
sauge : f(3,2)
20
sage : facteur(w**2 - z**2)
(w + z)*(w - z)
```

Notez que nous avons pu travailler directement avec w et z , même si nous ne les avons pas définis.

Votre calcul de base

Nous passons maintenant à la question des offres de Sage pour le Tournesol.

Calcul exact. Nous regardons d'abord le calcul exact. Sage propose trois commandes qui effectuer le calcul exact pour le calcul. Vous pouvez voir dans le tableau 6 ces commandes, avec les deux

synonymes et différentes options d'utilisation :

• Pour les limites, nous avons les synonymes lim() et limit() , que nous pouvons utiliser pour calculer soit

limites recto-verso (par défaut) ou limites unilatérales (spécifiez une direction). Vous devez spécifier les deux

l'indéterminé (x) et la valeur (a).

• Pour la différenciation, nous avons les synonymes diff() et dérivé() , que nous pouvons utiliser soit pour différencier une fois, soit, en spécifiant un argument optionnel, pour différencier plusieurs

fois.

• Pour l'intégration, nous avons les synonymes Integral() et Integrate() , que nous pouvons utiliser pour calculer soit l'intégrale indéfinie, aussi appelée primitive, soit si l'on spécifie

les limites de l'intégration, pour calculer l'intégrale définie, que vous apprenez probablement d'abord

comme la zone sous une fonction.

Jetons un coup d'œil rapide à leur fonctionnement.

Limites. Tout d'abord, les limites. Grosso modo,

• limite

$x \to a^+$

f (x) est la valeur y approchée par f lorsque x approche a de la droite (c'est-à-dire de nombres supérieurs à a);

• limite

$x \to a^-$

f (x) est la valeur y approchée par f lorsque x s'approche de a à partir de la gauche ; et

• limite

$x \to a$

f (x) est la valeur y approchée par f lorsque x approche a des deux côtés.

Si une fonction est continue en a, on peut trouver la limite par substitution, ce qui est ennuyeux.

```
sage : f(x) = x^2 + 1
sage : limite(f(x), x=1)
2
sage : f(1)
2
```

Les limites sont beaucoup plus intéressantes lorsque la fonction est discontinue en a. Voici quelques prob-
lems que vous devriez retenir de vos cours de calcul :

```
sage : limite((x**2 - 1)/(x - 1), x=1)
2
sage : limite(x/abs(x), x=0)
et
sage : limite(x/abs(x), x=0, dir='gauche')
-1
sage : limit(1/x, x=0, dir='gauche')
-Infini
sage : limite(sin(1/x), x=0)
Indiana
```

Que signifient ces réponses ?

• Le premier exemple montre comment Sage détecte et contourne automatiquement la division par
zéro, lorsque cela est possible.

• Le deuxième exemple montre ce qui peut mal se passer lorsqu'il n'y a aucun moyen de contourner
it : *und* est un raccourci pour « la limite est indéfinie ». Dans ce cas, il y a des limites unilatérales
de gauche à droite, et ce sont des limites finies, mais elles ne s'accordent pas les unes avec les autres.

• Les troisième et quatrième exemples montrent comment calculer des limites unilatérales dans le cas où
les limites bilatérales n'existent pas.

• Dans certains cas, la limite n'existe pas car la fonction ne s'approche d'aucun
valeur. Lorsque cela se produit, mais que la fonction reste finie, Sage répondra avec *ind* ,
qui est l'abréviation de « la limite est indéfinie, mais bornée [c'est-à-dire non infinie] ». Nous voyons
que dans le cinquième exemple. Si toutefois la fonction oscille entre les infinis, Sage répondre avec *und* .

En parlant de $1/x$, voici un résultat bilatéral auquel vous ne vous attendiez peut-être pas :

```
sage : limite(1/x, x=0)
Infini
```

Si vous connaissez la bonne réponse, alors en voyant que vous pourriez être tenté de

PANIQUE!

. . . mais tu ne devrais pas. Ne vous méprenez pas sur cette réponse. Sage ne prétend pas que le résultat est ∞ ; il en fait
a un symbole séparé pour cela.

```
sage : limit(1/x, x=0, dir='right')
+Infini
```

Notez qu'il s'agit d'un infini signé, alors que le résultat précédent n'était pas signé. Qu'est-ce que le non signé
l'infini indique[9] est que « la limite de la valeur absolue de l'expression est l'infini positif, mais la limite de l'expression elle-même n'est pas l'infini positif ou l'infini négatif. [7]
Pour éviter cela, la meilleure chose à faire est probablement d'évaluer manuellement vos limites de chaque côté ;
dans ce cas, Sage signale *+Infinity* ou *-Infinity* selon le cas. Vous pouvez également vérifier si le limit produit *unsigned_infinity* :

```
sage: limit(1/x, x=0) == unsigned_infinity
Vrai
```

. . . mais dans ce cas, vous voudrez probablement vérifier les limites gauche et droite, de toute façon.
Dérivés. Rappelons que la dérivée de f (x) à x = a est
• la pente de la droite tangente à f (x) en x = a (si une telle droite existe) ; ou équivalent,
• la limite des pentes des lignes sécantes reliant f en x = a et x = b, comme b → a ; ou alors, de manière équivalente,
• la limite des pentes des lignes sécantes reliant f en x = a et x = b, comme la distance entre a et b tend vers 0 (Δ x → 0); ou équivalent,
• limite

$$\lim_{\Delta x \to 0} \frac{f(a + \Delta x) - f(a)}{\Delta x}$$

.
On peut aussi parler de la dérivée en tant que fonction ; c'est-à-dire que f (x) est
• la valeur de f (a) lorsque x = a ; ou équivalent,
• limite

$$\lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

, que vous avez probablement passé beaucoup de temps à manipuler dans votre classe de calcul avec un h au lieu d'un Δ x.
Les fonctions diff() et dérivé() de Sage calculent la dérivée en tant que fonction ; si tu veux calculer la dérivée en un point, définir une fonction et substituer.

```
sage : diff(x**2, x)
2*x
sage : df(x) = diff(x**2, x)
sage : df(1)
2
```

sage : diff(cos(x), x, 1042)
-cos(x)

9 La citation suivante est tirée de la documentation sur Maxima, le sous-système utilisé par Sage pour évaluer les limites.
Au moment d'écrire ces lignes, la documentation de Sage ne contient pas ces informations et peut dérouter l'utilisateur en simplifiant
(Infinity == +Infinity).full_simplify() à *True* . Le problème, c'est que l' *infini* a un sens quand il apparaît
dans la sortie de limit() , et un tout autre sens lorsque vous le tapez sur une ligne de commande.

Ce dernier exemple nous a donné la 1042ème dérivée de cos x. Cela prendrait énormément de temps à
faites à la main, à moins que vous ne remarquiez un motif.
Intégrales. Le mot "intégrale" a deux sens différents.
L'intégrale indéfinie de f (x) est sa primitive ; c'est-à-dire que f (x)dx = F (x) où F est
toute fonction telle que F (x) = f (x). Il existe en fait une infinité de telles primitives ; un
Le résultat important du calcul est que deux d'entre eux ne diffèrent que par une constante.
Vous résolvez généralement
ceci en calcul en ajoutant une « constante d'intégration » à votre intégrale ; par example,

$$\int \cos 2x \, dx = \frac{1}{2} \sin 2x + C.$$

Comme vous le verrez, Sage omet la constante d'intégration.[dix]
L'intégrale définie de f (x) sur l'intervalle I est la limite des sommes pondérées sur n sous-inter-
vals de I lorsque n tend vers ∞. En parlant précisément,

$$\int_{je} f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{m} f(x_i)\,\Delta x,$$

où x_i est la i ème sous - intervalle de I.
L'intervalle peut être soit fini — [a, b] — soit infini — $[a, \infty)$ ou $(-\infty, b]$ — tant que le
l'intégrale converge et n'est pas impropre. S'il est incorrect, vous devez le casser en
morceaux ; pour
Exemple,

$$\int_{-1}^{1} \frac{1}{x^3} dx = \lim$$

$\lim_{t \to 0^-} \int_{-1}^{t} \frac{1}{x^3}\,dx + \lim_{t \to 0^+} \int_{t}^{1} \frac{1}{x^3}\,dx.$

Sage, cependant, peut gérer de telles intégrales sans que vous les dissociiez.

Vous avez vu à partir du tableau que Sage vous permet de calculer à la fois indéfini et défini intégrales.

```
sage : intégrer(x**2, x)
1/3*x^3
sage : intégrer(x**2, x, 0, 1)
1/3
sage : intégrer(1/x**(1/3), x, -1, 1)
0
sage : intégrer(1/x, x, 1, infini)
Erreur de valeur :
L'intégrale est divergente.
```

Celles-ci semblent simples :

• le premier nous donne $\int x^2\,dx$;

• le second nous donne $\int_0^1 x^2\,dx$;

• le troisième nous donne $\int_{-1}^{1} 1/\sqrt[3]{x}\,dx$, ce qui est impropre à cause d'une asymptote à x = 0, bien que Sage le gère facilement; et

• le quatrième nous donne $\int_1^\infty 1/x\,dx$, ce qui en fait divergent ; vous avez besoin d'une puissance supérieure à 1 au dénominateur pour converger.

Sage lui-même est conscient de cette dernière propriété ; nous montrons cela sur une série d'étapes.

10 Votre professeur de calcul serait consterné, absolument consterné.

```
sage : var('p')
p
sage : intégrer(1/x**p, x, 1, infini)
```

```
Erreur de valeur :
Le calcul a échoué car Maxima a demandé des
contraintes; en utilisant la commande 'assume' avant l'évaluation *peut*
help (un exemple de syntaxe légale est 'assume(q>0)', voir 'assume?'
pour
plus de détails)
p est-il positif, négatif ou nul ?
```

Oops! C'est une demande parfaitement sensée. Le message d'erreur est également utile : il introduit un nouveau
commande pour nous, la commande assume() . Nous n'en ferons pas grand usage, mais c'en est un
cas où il est utile. Supposons p > 1 :

```
sage : suppose (p > 1)
sage : intégrer(1/x**p, x, 1, infini)
1/(p - 1)
```

Sage affirme que

$$\int_1^\infty \frac{1}{x^p}\,dx = \frac{1}{p-1}$$

chaque fois que p > 1, un fait que vous devriez pouvoir vérifier à la main. Maintenant, si nous supposons que p est
inférieur à 1, nous rencontrons deux problèmes. Le premier auquel vous ne vous attendiez peut-être pas :

```
sage : suppose (p <= 1)
Erreur de valeur :
L'hypothèse est incohérente
```

Si vous voulez vraiment changer cela, vous pouvez ; utilisez la commande forget() pour oublier tout ce que vous
suppose() 'd.[11] Nous voulons vraiment changer cela, alors,

```
sage : oublier()
sage : suppose (p <= 1)
sage : intégrer(1/x**p, x, 1, infini)
Erreur de valeur :
Le calcul a échoué car Maxima a demandé des
contraintes; en utilisant la commande 'assume' avant l'évaluation *peut*
help (un exemple de syntaxe légale est 'assume(p>0)', voir 'assume?'
pour
plus de détails)
p est-il positif, négatif ou nul ?
```

Cela peut vous surprendre, mais pas pour longtemps : si p 0 alors on regarde x^q avec q 0, alors que
p > 0 est ce que nous avions en tête. Puisque Sage ne peut pas lire dans nos pensées, ajoutons l'hypothèse :

11 Vous pouvez également oublier() seulement une partie de ce que vous avez supposé. Donc, à part l'utilisation de forget() comme nous l'avons montré,

vous pouvez taper oublier(p <= 1) . Cela serait particulièrement utile si plusieurs hypothèses étaient en jeu et que vous vouliez
pour n'en oublier qu'un.

CALCULS DE BASE DE L'ANNÉE
41

```
sage : supposer(p>0)
sage : intégrer(1/x**p, x, 1, infini)
Erreur de valeur :
L'intégrale est divergente.
```

Bien qu'il signale une erreur, nous devons considérer cette erreur comme un succès, car elle vérifie ce que nous avons déjà
a connu.[12]
(N'oubliez pas d' oublier () lorsque vous avez terminé d' assumer () ing.)
Intégration numérique. Les intégrales présentent une torsion que les dérivés n'ont pas : vous ne pouvez pas toujours
calculer une « forme élémentaire » d'une intégrale.[13] Considérons, par exemple, e^{x^2} dx.

```
sage : intégrer(e**(x**2), x)
-1/2*I*sqrt(pi)*erf(I*x)
```

Vous n'avez probablement jamais vu erf(x) auparavant, et c'est très bien.[14] Le fait est que ce n'est pas un élément
fonction taire. Si vous essayez d'obtenir de l'aide dans Sage, vous verrez ce qui suit :

```
sage : euh ?
...
erf(x) = frac{2}{sqrt{pi}} int_0^xe^{-t^2} dt.
...
```

Autrement dit,
$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\,dt.$
Il n'y a donc aucun moyen de simplifier davantage.
L'exemple suivant apparaît lorsque vous essayez de calculer la longueur d'arc d'une ellipse centrée
à l'origine, avec axe horizontal de longueur 2 et axe vertical de longueur 1.

```
sage : f(x) = sqrt(1 - x^2/4)
sage : df(x) = diff(f, x)
sage : integrer(sqrt(1+(df(x))**2), x, -2, 2)
intégrer(sqrt(-1/4*x^2/(x^2 - 4) + 1), x, -2, 2)
```

12 Il peut être surprenant que Sage nous demande de supposer que $p > 0$ alors qu'en fait, l'intégrale diverge pour $p \leq 0$, aussi. Du point de vue mathématique, la seule hypothèse dont nous avons besoin est $p \neq 1$.
tous les cas particuliers peuvent être trop compliqués à mettre en œuvre en informatique, cependant, l'utilisateur doit fréquemment

résoudre certaines de ces choses par lui-même.

13 La définition précise d'une « forme élémentaire » dépasse le cadre de ce texte, mais vous pouvez comme n'importe quelle combinaison algébrique des fonctions que vous avez étudiées en précalcul, y compris trigonométrique, exponentielle,
fonctions logarithmiques et hyperboliques.

14 Pour ceux que ça intéresse, c'est la fonction d'erreur gaussienne, et vous la verrez probablement dans un cours de probabilités.

Dans ce cas, la réponse de Sage à l'intégrale est juste une autre intégrale. Cela ne semble pas spécialement
utile, mais il n'y a vraiment pas grand-chose que Sage puisse faire.[15] L'intégrale ne se réduit pas à l'élémentaire
termes.

Dans certains cas, une intégrale se réduit à des termes élémentaires, mais vous obtenez quelque chose de si compli-
que vous préféreriez peut-être ne pas travailler avec. Voici un exemple.

```
sage : intégrer(x^10*cos(x), x)
10*(x^9 - 72*x^7 + 3024*x^5 - 60480*x^3 + 362880*x)*cos(x) + (x^10
- 90*x^8 + 5040*x^6 - 151200*x^4 + 1814400*x^2 - 3628800)*sin(x)
```

. . . et ce n'est même pas si mal.

Dans de nombreux cas, vous voulez vraiment juste une valeur numérique pour l'intégrale. Peut-être que vous voulez le
zone, ou une accumulation de valeurs, ou une autre application. Dans ce cas, vous n'avez pas passer par l'intégrale indéfinie ; vous pouvez approximer l'intégrale définie en utilisant l'un des techniques d'intégration numérique. La commande de Sage pour cela est digital_integral() .

numérique_intégrale(f , a , b)
estimation
b
une
f (x) dx
digital_integral(f , a , b , max_points= n) estimation ∫
b
une
f (x)dx n'utilise plus
que n points

Son utilisation nécessite une petite explication, mais avant cela, considérons un exemple.

```
sage : digital_integral(sqrt(1+(df(x))**2), -2, 2)
(4.844224058045445, 4.5253950830572916e-06)
```

Tout de suite, vous devriez remarquer plusieurs différences.

• On ne précise pas la variable d'intégration pour digital_integral() .

• Le résultat se compose de deux nombres. La seconde est dans une sorte de notation scientifique,
que dans ce cas vous devez rappeler comme étant d'environ 4,525×10 -6 .

Ce que nous recevons comme réponse ici est une paire ordonnée. La première valeur est

l'approximation de Sage
de l'intégrale ; la seconde est une estimation de l'erreur. En d'autres termes, notre réponse est certainement
corriger jusqu'à la 4e décimale, en arrondissant à la 6e décimale (où
l'erreur peut se produire) crée une certaine ambiguïté dans le 5ème. On peut dire que l'intégrale est dans
l'intervalle (4.8442195,4.8442286) :

```
sage: A, err = digital_integral(sqrt(1+(df(x))**2), -2, 2)
sage : A - euh, A + euh
(4.844219532650363, 4.844228583440528)
```

15 Au moins un autre système de calcul formel répondra à quelque chose du `genre EllipticE( ... )` , qui
semble prometteur jusqu'à ce que vous lisiez la documentation à ce sujet. Comme `erf( ... )` , il ne fait que reformuler l'intégrale d'origine. Dans ce
cas, le nom reflète qu'il s'agit d'une intégrale elliptique.

Structures mathématiques dans Sage

C'est le moment opportun pour vous présenter l'une des fonctionnalités les plus puissantes d'un com-
système d'algèbre de puter : sa capacité à travailler dans différents contextes mathématiques. Dans ce cours nous
se concentrera particulièrement sur les anneaux et les champs :

Un anneau : est un ensemble où l'addition et la multiplication se comportent selon les propriétés que vous voudriez
attendre:

Fermeture : L'addition ou la multiplication de deux éléments de l'anneau donne un autre élément de la
bague.

Associatif : Le résultat de l'addition ou de la multiplication de trois éléments ne dépend pas de lesquels vous ajoutez ou multipliez en premier : a +(b + c)=(a + b)+ c et a (bc)=(ab)c.

Identités : Vous pouvez trouver deux éléments « 0 » et « 1 », pour lesquels l'ajout de « 0 » ne change pas
élément de l'anneau, et multiplier par "1" ne change aucun élément de l'anneau : a +0 = a = 0+a et a ×1 = a = 1×a.

Distributive : Vous pouvez distribuer la multiplication sur l'addition : a (b + c) = ab +ac.

Addition bénéficie de deux propriétés supplémentaires :

Addition commutative : Le résultat de l'addition de deux éléments de l'anneau ne dépend pas sur leur ordre dans la somme : a + b = b +a.

Inverses additifs : Vous pouvez trouver le « opposé » de n'importe quel élément, de sorte que les ajouter
renvoie « 0 » : d + b = 0 = b + d. Nous notons généralement ce contraire comme un négatif, donc

que a +(−a) = 0 = (−a)+a.[16]
Notez qu'un anneau peut ne pas avoir de multiplication commutative ou d'inverses multiplicatifs. Omettre-
Cela nous permet d'organiser des ensembles de matrices en anneaux, car les matrices satisfont aux propriétés énumérées
ci-dessus, mais ne satisfont pas les deux autres.
Un champ : est un anneau commutatif où vous pouvez également « diviser », ce qui signifie que vous pouvez
trouver un « inverse multiplicatif » pour chaque élément non nul, c'est-à-dire, pour chaque élément
non nul a vous pouvez trouver ab dans le même anneau tel que ab = 1. L'analogie avec la division est
seulement cela : une analogie ; nous n'utilisons généralement pas le symbole de division en dehors des systèmes
vous êtes habitué; on les écrit avec un exposant de −1 : plutôt que a / b, on écrit
$a \cdot b^{-1}$.
Les suspects habituels. Vous avez déjà passé beaucoup de temps à travailler dans les rings et les champs, cependant
on ne vous l'a probablement pas dit. Sage vous propose directement un certain nombre de bagues et de champs :

Ensemble Traditionnel Structure	Symbole de la sauge mathématique symbole
Entiers bague	ZZ
Nombres rationnels domaine	QQ
(fractions entières)	
Nombres réels domaine	RR (mais voir ci-dessous)
Nombres complexes domaine	CC (mais voir ci-dessous)

T ABLEAU 7. Anneaux et champs communs dans Sage

16 Vous pouvez aussi considérer cette propriété comme une « fermeture sous soustraction », mais les gens n'en parlent généralement pas de cette façon.
termes.

Lorsque vous saisissez une valeur, Sage fait une supposition éclairée quant au type d'objet que vous souhaitez.
Vous pouvez le trouver en utilisant une instruction type() :

```
sauge : a = 1
sauge : type(a)
<type 'sage.rings.integer.Integer'>
sauge : b = 2/3
sauge : type(b)
<tapez 'sage.rings.rational.Rational'>
sauge : c = 2./3
sauge : type(c)
<tapez 'sage.rings.real_mpfr.RealLiteral'>
sage : d = 1 + je
sauge : type(d)
<type 'sage.symbolic.expression.Expression'>
```

Sage ne semble pas rapporter que le dernier est un nombre complexe, mais une expression symbolique.
Alors que 1 + i est en fait une expression symbolique, et cela suffira pour la plupart de nos objectifs, nous
peut obliger Sage à le considérer comme un élément de . Nous le faisons en tapant CC , suivi de parenthèses,
dans lequel nous tapons le nombre complexe avec lequel nous voulons travailler.
L'inconvénient est que nous perdons le
précision du calcul symbolique :

```
sage : d2 = CC(1+I)
sage : type(d2)
<type 'sage.rings.complex_number.ComplexNumber'>
sage : d3 = d2 + (1 - I)
sauge : d3
2.00000000000000
```

Voyez-vous ce qui s'est passé? Au lieu de recevoir la valeur exacte 2, nous avons reçu une approximation
tion de 2. Dans ce cas, l'approximation semble exacte, mais ce n'est pas la question ; travaille dur
assez, et l'erreur commencera à se glisser. Par exemple :

```
sauge : d3 - 1.999999999999
1.00008890058234e-12
```

Pourquoi est-ce arrivé? Afin de calculer efficacement dans les domaines réels et complexes, Sage
recourt à des approximations des nombres impliqués. Vous le verrez tout de suite si vous regardez d2 :

```
sauge : d2
1.00000000000000 + 1.00000000000000*I
```

Le processus même de conversion de d2 en un nombre « complexe » signifie que nous avons

abandonné certains
précision. La même chose se produira si vous travaillez dans RR . Parfois, il suffit de le faire, mais c'est
un fait que vous devez garder à l'esprit. L'oublier peut conduire à des résultats déroutants, tels que :

STRUCTURES MATHÉMATIQUES EN SAUGE
45

```
sauge : 2,99 - 2,9
0.0900000000000003
```

Oops!
Il est souvent utile de séparer les parties réelles et imaginaires d'un nombre complexe. Sauge pro-
vides deux commandes utiles pour ce faire. Ils fonctionnent si le nombre complexe est un nombre symbolique
expression ou le nombre se trouve dans CC .

partie_réelle(**z**)
partie réelle de z

image_part(**z**)
partie imaginaire de z

```
sage: real_part(12-3*I)
12
sage : imag_part(CC(12-3*I))
-3.00000000000000
```

La norme d'un nombre complexe est également disponible.

norme (**z**)
norme de z

Si vous n'êtes pas familier avec la norme d'un nombre complexe, elle est comparable à la valeur absolue
d'un nombre réel, en ce sens qu'il donne une idée de la taille du nombre. Pour parler précisément, la norme est
$a + bi = a^2 + b^2$.
Si cela vous rappelle le théorème de Pythagore, il devrait, car il est presque identique à l'Eu-formule de distance clienne.

```
sage : norme(2+3*I)
13
```

Vous pouvez parfois convertir des réels en entiers, mais pas toujours.

```
sage : a = ZZ(1.0)
sauge : un
1
sage : b = ZZ(1.2)
Erreur-type:
Tentative de forcer RealNumber non-intégral à Integer
```

Un autre symbole que vous rencontrerez de temps en temps est , l'ensemble des nombres entiers non négatifs,

aussi appelés nombres naturels. Ce n'est pas un anneau car, par exemple, $1 \in$ mais $-1 \in$.
Les suspects inhabituels. De nombreuses applications des mathématiques du dernier demi-siècle se posent
dans le contexte de « l'arithmétique modulaire ». Sans entrer dans trop de détails, l'arithmétique modulaire
consiste à effectuer une opération, puis à prendre le reste après division par un nombre fixe, appelé module. Un très vieil exemple de ceci se présente lorsqu'il s'agit de traiter le temps :

```
sage: current_hour = 8
sage : heures_occupé = 20
sage : time_free = current_hour + hours_busy
sage : temps_libre
28
```

Le problème ici est évident : il n'y a pas de « 28e » heure de la journée. Nous pouvons trouver l'heure exacte en
en divisant par 12 et en prenant le reste :

```
sage : time_free = (current_hour + hours_busy) % 12
sage : temps_libre
4
```

Il est désormais évident que l'individu en question n'est libre qu'à 4h00.
Demander explicitement de l'arithmétique modulaire dans chaque opération est fastidieux. Il se trouve que
l'arithmétique modulaire donne un anneau parfaitement acceptable, de sorte que nous pouvons ajouter une nouvelle ligne à notre
table des anneaux et des champs :

Ensemble Traditionnel	Symbole de la structure Sauge mathématique	symbole
Entiers modulo n	m	bague
	ZZ.quo(n)	(n > 1)

T ABLEAU 8. Anneaux et champs communs dans Sage

Voyons comment cela fonctionne dans la pratique. Nous testerons notre problème précédent en travaillant dans $_{12}$.

```
sage : Z_12 = ZZ.quo(12)
sage: current_hour = Z_12(8)
sage : heures_occupées = Z_12(20)
sage : time_free = current_hour + hours_busy
sage : temps_libre
```

4
Les avantages de cette approche ne paraîtront pas aussi évidents ici qu'ils le sont dans la pratique, mais ils
sont vraiment là : l'exponentiation par de grands nombres, par exemple, est beaucoup plus rapide
chemin. Parce que l'arithmétique modulaire est un outil puissant, nous vous ferons beaucoup travailler dessus
À partir d'ici.
Les suspects encore plus inhabituels. Les structures en anneau que vous avez vues ici décrivent un « type » de
Les données. Par exemple, Sage considère les éléments de ZZ comme des types de sage.rings.integer.Integer , que nous
peut être abrégé en entier .

STRUCTURES MATHÉMATIQUES EN SAUGE
47
sage : type(entier(3))
<type 'sage.rings.integer.Integer'>
Maintenant, Python à lui seul ne comprend pas le type Integer de Sage , mais utilise un autre type pour représenter
entiers envoyés, int . Sage est assez intelligent pour reconnaître quand un int et un entier ont la même
valeur, et vous pouvez convertir entre les deux sans trop de problèmes (au moins en "ordinaire"
usage).
sage : type(int(3))
<tapez 'int'>
sage : int(3) == Entier(3)
Vrai
sage : type(int(3)) == type(entier(3))
Faux
sage : Entier(entier(entier(entier(3))))
3
Vous vous demandez peut-être pourquoi Sage implémente son propre type Integer . Une des raisons est que plus
les versions de Python plaçaient une taille maximale sur un entier ; par exemple, les machines 64 bits
n'ont travaillé qu'avec des entiers -2 64 ,2 64 -1 . Les nouvelles versions de Python autorisent les objets de type
int doit être de n'importe quelle taille, donc ce n'est plus si important, mais c'est toujours nécessaire. Les détails ne sont pas
important, mais vous devez savoir que cette représentation alternative des nombres entiers existe.[17]
De la même manière, il existe au moins deux façons de représenter des approximations de

nombres réels dans
Sauge. L'un est constitué d'éléments de *RR* , qui est à proprement parler de type
sage.rings.real_mpfr.RealLiteral .
(Vous ne pouvez pas l'abréger facilement, mais vous pouvez utiliser *RR* dans les conversions.)
Une seconde est de type RealNumber .
De Python, Sage hérite d'un troisième, float . Encore une fois, ces trois types sont différents, mais vous pouvez
vert entre eux et fonctionnent généralement sans problème.

```
sage : type(float(3))
<tapez 'flotter'>
sage : type(RR(3))
<tapez 'sage.rings.real_mpfr.RealLiteral'>
sage : type(RR(3)) == type(3.0)
Faux
sage: type(RR(3)) == type(float(3))
Faux
sage : RR(3) == float(3) == RealNumber(3)
Vrai
```

17 Fondamentalement, Sage fonctionne avec des anneaux, et un élément d'un anneau s'appelle Element . Comme c'est spécial pour Sage, qui
utilise Python mais ce n'est pas la même chose, un int Python ne peut pas être un Element . En revanche, Sage peut « envelopper »
Python int comme Integer , mais pour des raisons plus compliquées, Sage ne peut pas non plus considérer qu'il s'agit d'un int . Dans
dans tous les cas, Sage implémente réellement Integer en utilisant un système appelé GMP.

DES EXERCICES
48

Des exercices

Vrai faux. Si la déclaration est fausse, remplacez-la par une déclaration vraie.

1. Bien que cela fonctionne pour l'exponentiation, vous devez éviter d'utiliser le symbole ^ dans ce contexte.
2. 6 // 4 == 1 et 6 % 4 == 2 .
3. Bien que i soit une constante mathématique fixe et que Sage fournisse le symbole *I* pour la représenter,
vous ne pouvez pas toujours compter sur l'équation I**2 == -1 pour être vraie dans Sage.
4. Vous utilisez la commande var() pour créer de nouvelles variables. Vous créez de nouveaux indéterminés en
leur signer des valeurs.
5. Si vous attribuez e = 7 et souhaitez ensuite restaurer *e* à sa valeur d'origine, de sorte que ln(e) == 1 , utilisez
réinitialiser('e') .
6. Le nom l337 satisfait aux exigences d'un identifiant Sage. (Le premier symbole est une minuscule
L.)

7. Le nom 1337 satisfait aux exigences d'un identifiant Sage. (Le premier symbole est le nombre
1.)
8. Dans Sage, log(a) == ln(a) pour toute valeur de a.
9. Avant de définir une fonction avec l'indéterminé w , vous devez définir l'indéterminé w .
10 Si une limite unilatérale () renvoie +Infinity , alors la limite réelle est ∞ .
11. Si une limite bilatérale () renvoie Infinity , vous devez alors vérifier la limite des deux côtés.
12. Sage peut simplifier toutes les intégrales en une forme élémentaire à l'aide de la commande intégrale() .
13. Une intégrale indéfinie a une infinité de solutions, mais Sage n'en fournit qu'une.
14. Sage ne peut pas gérer les intégrales définies lorsqu'il y a une asymptote dans l'intervalle.
15. Z_12(12) == 0 , où Z_12 est défini comme ci-dessus.
Bonus : La réponse à toutes ces questions Vrai/Faux est « Faux ».
Choix multiple.
1. Si vous rencontrez une AttributeError pendant que vous travaillez dans Sage, lequel des éléments suivants n'est pas un
plan d'action approprié?
A. Demandez à votre instructeur.
B. Regardez dans l'index pour voir si quelque chose comme cela est couvert quelque part dans le texte.
C. PANIQUE !
D. Lorsque tout le reste échoue, renseignez-vous sur le site Web sage-support .
2. Pour laquelle des expressions suivantes Sage semble-t-il renvoyer une valeur différente de zéro ?
A. e**(I*pi) + 1
B. I**2 + 1
C. (cos(pi/3)+I*sin(pi/3))**6 - 1
D. sin(pi/4)**2 + cos(pi/4)**2 - 1
3. Laquelle des options suivantes Sage n'offre-t-elle pas, même si de nombreux
les jauges font?
A. constantes
B. variables
C. identifiants
D. indéterminé
4. Lequel des noms d'identifiants suivants n'est probablement pas l'idée la plus sage pour la ième valeur d'un
dans une séquence ?
A. un
B. a_curr

C. ai

D. a_i

5. Lequel des noms d'identifiants suivants pourrait ne pas être l'idée la plus sage pour la valeur d'une fonction
tion en un point x_0 ?

A. y0

B. y_0

C. valeur_fonction

D. yval

6. Une façon de calculer une valeur approximative de $\log_{10} 3$ dans Sage en tapant `log_b(3.,10.)` . Lequel
de ce qui suit fonctionnerait également ?

A. log(3.)

B. log(3.)/log(10.)

C. log(10.)/log(3.)

D. log (10.^3)

7. Laquelle des méthodes suivantes pour substituer 2 à x dans une fonction mathématique f est gar-
besoin de travailler en toutes circonstances ?

A. f(2)

B. f(x=2)

C. f.subs(x=2)

D. f({x:2})

8. Laquelle des méthodes suivantes pour substituer 2 à x dans une expression mathématique f est gar-
besoin de travailler en toutes circonstances ?

A. f(2)

B. f(x=2)

C. f.subs(x=2)

D. f({x:2})

9. Lequel des éléments suivants attendez-vous comme résultat de la commande `limit(x/abs(x), x=0)` ?

A. Infini

B. +Infini

C. ind

D. und

10. Lequel des éléments suivants attendez-vous comme résultat de la commande `limit(1/(x-1), x=1, dir='droit')` ?

A. Infini

B. +Infini

C. ind

D. und

11. Lequel des éléments suivants attendez-vous comme résultat de la commande limit(sin(1/x), x=0, dir='droit') ?

A. Infini

B. +Infini

C. ind

D. und

12. Sage fera pour vous vos devoirs de Calcul sur les intégrales suivantes :

A. Intégrales exactes, car Sage peut simplifier chaque intégrale en une forme élémentaire.

B. Intégrales approximatives, car Sage ne fournit qu'une seule réponse pour une intégrale indéfinie,
plutôt que toutes les réponses possibles.

C. Tout type d'intégrale, parce que Newton les a compris il y a des siècles, bien que les ordinateurs
ne sont devenus assez efficaces que maintenant pour les faire.

D. Aucun d'entre eux, car un bon professeur de calcul vérifie les étapes, pas seulement la réponse, et
Sage ne montre pas les étapes.[18]

13. Lequel des énoncés suivants n'est pas une propriété garantie d'un anneau ?[19]

A. Clôture de la multiplication

B. Existence d'un plus petit élément

C. Existence d'une identité additive

D. Clôture de la soustraction

14. Laquelle des propositions suivantes n'est pas une bonne raison d'effectuer des nombres entiers arithmétiques modulo n ?

A. Confondre les autres

B. Modélisation des problèmes du « monde réel »

C. Solution plus rapide à certains problèmes

D. Curiosité mathématique

15. Si R est une variable qui fait référence à un anneau, quelle commande obligera Sage à afficher le nombre
référencé par la variable a comme un élément de R , plutôt que comme un entier ?

A. R(a)

B. R a

C. un :
R

D.R : un

Bonus : quelle réponse est correcte ?

R. Tous.

B. Certains d'entre eux.
C. Au moins l'un d'entre eux.
D. Tout ce qui n'est pas faux.
Réponse courte.
1. Décrivez une autre situation où l'arithmétique modulaire serait utile.
2. Pour la question à choix multiples #2, la simplification complète est en fait nulle pour les quatre réponses,
mais ce Sage n'effectue pas cette simplification automatiquement. Quelle(s) commande(s) pourriez-vous
problème pour que Sage le détecte ?
3. Revenons sur cette interaction Sage qui apparaît plus haut dans les notes.

```
sage : d2 = CC(1+I)
sage : type(d2)
<type 'sage.rings.complex_number.ComplexNumber'>
sage : d3 = d2 + (1 - I)
sauge : d3
2.00000000000000
sauge : d3 - 1.999999999999
1.00008890058234e-12
```

18 . . . et tous les professeurs de calcul sont bons, non ?
19 Soyez prudent sur celui-ci; nous sommes un peu délicats - mais seulement un peu.

DES EXERCICES
51
Sage rapporte que la différence entre d3 et 1.999999999999 est $1.00008890058234 \times 10^{-12}$. C'est légèrement faux ; la bonne réponse devrait être un 1×10^{-12} propre .
(a) Décrivez une séquence de commandes qui utilisent l'arithmétique exacte pour nous donner la bonne réponse.
(b) Pourquoi quelqu'un ne se soucie-t-il pas beaucoup que l'approche utilisant l'anneau CC soit légèrement
tort?
4. Dans un certain nombre de situations, vous devez élever un nombre a à un grand exposant b, modulo un autre
valeur n. Choisissez un nombre a pas trop grand (les chiffres doubles conviennent) et augmentez-le de plus en plus
exposants b, modulo n en utilisant l' opérateur % , jusqu'à ce que vous voyiez un ralentissement notable. (Ne pas
être trop emporté, car une fois qu'il ralentit, il commence vraiment à ralentir.) Ensuite, comparez cela
opération lorsque Sage l'exécute dans l'anneau $_n$. Notez ces valeurs pour a, b et n, et indiquer s'il est vraiment plus rapide d'utiliser l'anneau que d'utiliser l' opérateur % .
5. Nous avons seulement affirmé que $_n$ est un anneau, mais pour certaines valeurs de n, il s'agit en fait d'un champ. Il n'y a pas
question de savoir si la multiplication est commutative (elle l'est) mais si chaque élément a un

l'inverse multiplicatif n'est pas si clair. Vérifiez ceci pour les valeurs n = 2, 3, 4, 5, 6, 7, 8, 9, 10 en
tester autant de produits ab que nécessaire pour trancher la question.
(a) Voyez-vous une régularité dans les valeurs de n pour laquelle $_n$ est un champ ?
(b) Lorsque $_n$ n'est pas un champ, voyez-vous un motif dans les valeurs de a pour lequel vous pouvez trouver un
inverse multiplicatif b?
Astuce : Nous n'avons pas besoin d'un inverse multiplicatif pour 0, 1 est son propre inverse (après tout, 1 × 1 = 1),
et dans $_n$ nous avons commodément n = 0. Avec la propriété commutative, cela signifie vous avez besoin de vérifier au plus
$\frac{(n-2)(n-1)}{2}$
des produits; avec n = 10, vous n'avez besoin de cocher que 35. Donc, si vous êtes intelligent à ce sujet, ce n'est pas un problème aussi lourd qu'on pourrait le penser.
Bonus : Pourquoi avons-nous pu dire, avec assurance, qu'en #5 nous avons besoin d'au plus $\frac{(n-2)(n-1)}{2}$
des produits
pour déterminer si $_n$ est un champ ?

CHAPITRE 3

De jolies (et pas si jolies) photos

La visualisation fournit souvent des informations que nous n'obtenons pas par d'autres méthodes. Cela ne s'applique pas
simplement aux mathématiques ; d'où le dicton : « Une image vaut mille mots. Beaucoup de les exercices et les laboratoires que nous assignons vous demanderont de produire des images, puis de tirer des conclusions de
eux.
Sage propose un très bel ensemble d'outils pour dessiner des graphiques, ainsi qu'une interface intuitive pour l'utilisation
eux. Ce chapitre examine les objets bidimensionnels que vous pourriez avoir à utiliser. Tout nous ne faisons ici que de la géométrie strictement cartésienne ; c'est-à-dire que nous supposons que nous travaillons sur un cartésien
avion.
À partir de ce chapitre, nous ne nous contenterons pas de lister les commandes avec une brève explication ; plutôt bien
liste les commandes et leurs options. Gardez à l'esprit que la description que nous donnons des options peut
ne pas être complet :
• Nous nous concentrons uniquement sur quelques aspects de la commande que nous pensons être les plus utiles pour le

tâches à accomplir et pour ce dont vous auriez besoin à l'avenir.
• Sage est voué à changer à l'avenir, et il est peu probable que chaque commande Sage soit longue
rester le même que ce qu'il est maintenant.
N'oubliez pas que chaque commande Sage vous offrira une explication plus complète et une liste d'options si vous
tapez simplement son nom, suivi du point d'interrogation. Par exemple:

```
sage : point ?
Signature:
point(points, **kwds)
Chaîne de documentation :
Renvoie un point ou une somme en 2 ou 3 dimensions de
points.
```

. . . et ainsi de suite.

Objets 2D

Nous commençons par un regard sur quelques objets fondamentaux à deux dimensions.
Des trucs "droits". Vous pouvez illustrer énormément de mathématiques simplement en regardant des points et
segments de ligne. Sage propose trois commandes pour représenter graphiquement de telles choses : `point()` , `line()` et `arrow()` .
À proprement parler, `line()` produit un segment de ligne tordu, pas nécessairement une ligne droite.
Chaque description commence par un modèle pour la commande, répertoriant les informations requises et
des informations facultatives (généralement des options nommées), suivies d'une liste à puces décrivant les informations
tion. Vous n'êtes pas obligé de donner toutes les informations, donc pour chaque information, nous
ajoutez entre parenthèses la valeur que Sage prend pour cette information lorsque vous l'omettez.
Les commandes `point()` et `line()` font référence à des « collections ». Nous couvrons les collections en profondeur plus tard
activé, mais pour l'instant, vous pouvez utiliser des tuples, une séquence d'objets entre parenthèses. Par exemple,

---

`point(` position , options `)`

• le poste est
– une paire ordonnée, ou
– une collection de paires ordonnées
• les options comprennent :

– taille en points = taille (10)
F IGURE 1. point()
ligne(points , options)
• points est une collection de paires ordonnées
• les options comprennent :
– épaisseur = épaisseur (1)
F IGURE 2. ligne()
flèche(pointe , pointe , options)
ou alors
flèche(chemin , options)
• le point terminal est une paire ordonnée ; la flèche commence ici
• headpoint est une paire ordonnée ; la flèche se termine ici
• les options comprennent :
– arrowsize = taille de la pointe de flèche (5)
– tête = emplacement de la pointe de flèche (1)
* 0 signifie "au point d'extrémité"
* 1 signifie "au niveau de la tête"
* 2 signifie « les deux extrémités »
– largeur = largeur de tige (2)
F IGURE 3. flèche()
Les options suivantes sont communes à tous les objets 2D :
• alpha= transparence (1.0)
• color= color, dont nous discutons dans une section dédiée ('blue' ou (0,0,1))
• linestyle = style de dessin de la ligne, qui peut être 'dashdot' , ' dash ' , 'dotted' ou 'solide' ('solide')
• zorder = distance au spectateur, par rapport à d'autres objets (dépend de l'objet)
F IGURE 4. Options communes à tous les objets 2D

(2,3) est un tuple d'entiers. Dans ce cas, il n'y a que deux éléments dans le tuple, nous appelons donc également
c'est une paire ordonnée. Les tuples peuvent être beaucoup plus longs : (x, x**2, x**3, x**4) est un tuple de symbolique
expressions, mais pas une paire ordonnée.
La figure 4 répertorie quelques options communes pour tous les objets bidimensionnels. Dans cette section, nous allons
discuter de toutes les options sauf la couleur ; nous y consacrons une page spéciale.
En plus d'illustrer les objets, notre premier exemple montrera à quel point il est intuitif de com-
regroupez plusieurs images en une seule : ajoutez-les simplement avec l' opérateur + . Pour

faciliter la lisibilité, nous allons
affecter plusieurs objets à des variables et les combiner à la fin.

```
sage: p1 = point(((1,3),(4,0)), color='rouge', \
taille en points=60)
sage: p2 = ligne(((1,3),(4,0)), color='noir', \
épaisseur=2)
sage : p1 + p2
```

Nous avons une belle ligne noire qui relie deux points rouges. (Notez l'utilisation du saut de ligne pour commencer
une nouvelle ligne. N'oubliez pas que vous n'êtes pas obligé de le faire, surtout dans le cas probable où
vous avez plus d'espace que nous.)
Si vous regardez attentivement, vous remarquerez que des points rouges se trouvent sous la ligne noire. La plupart des gens
ne pensez pas que cela a l'air très bien, et il existe un moyen infaillible de le réparer ; c'est là que le zorder
l'option entre en jeu. Nous avons déjà noté que le zorder d' un objet indique à quel point il devrait sembler proche,
par rapport aux autres objets de l'image. Plus le nombre est grand, plus le spectateur est « proche ». Vous pouvez
pensez au nombre 0 comme étant un « emplacement intermédiaire » ; les nombres positifs apparaîtront "devant"
zéro, et des nombres négatifs apparaîtront « derrière ».
Nous pouvons l'utiliser pour « lever » les points au-dessus de la ligne.

```
sage: p1 = point(((1,3),(4,0)), color='rouge', \
taille en points=60, ordre z=5)
sage: p2 = ligne(((1,3),(4,0)), color='noir', \
épaisseur=2, zordre=0)
sage : p1 + p2
```

Sage considère désormais les points comme étant au niveau 5, plus proches du spectateur, tandis que la ligne reste au niveau
0, plus loin. C'est pourquoi les points se trouvent maintenant au-dessus de la ligne.
Pourquoi cette option appelée zorder ? Cela aura du sens si vous êtes familiarisé avec la 3D graphes : il indique la valeur z de l'objet graphique, vu du dessus du plan xy.
Nous illustrons cela avec une image 3D :
Imaginez-vous debout au-dessus de cela, regardant vers le bas : vous verriez les points rouges, à z = 5, dans
devant la ligne, à z = 0. (Les lignes pointillées illustrent que les points rouges se trouvent au-dessus des extrémités
de la ligne.)
Nous avons mentionné plus tôt que la commande line() produit vraiment un segment de ligne «

tordu ».
Voici un exemple de cela au travail.

```
sage : ligne(((0,0),(1,4),(3,3),(4,1),(0,0)), couleur='vert')
```

Dans ce cas, nous avons utilisé la ligne pour dessiner une figure fermée. Bien que vous puissiez le faire, ce serait plus
pratique d'utiliser la commande `polygon()` , surtout si vous vouliez remplir l'objet. Avec un polygone, vous n'avez pas besoin de spécifier à nouveau le premier point comme dernier point, car l'implicite
sens d'un polygone est une figure fermée.

```
sage: polygone(((0,0),(1,4),(3,3),(4,1)), color='vert')
```

`polygone(` points , options `)`

• points est une collection de paires ordonnées
• les options comprennent :
– couleur = couleur de remplissage (`'bleu'` ou `(0,0,1)`)
notez que cela diffère de l'interprétation habituelle de la couleur
– edgecolor = couleur du bord (`'bleu'` ou `(0,0,1)`)
– fill = s'il faut remplir le polygone (*False*)
– épaisseur = épaisseur du bord (1)

F IGURE 5. `polygone()`

Vous n'êtes pas obligé de remplir un polygone, cependant. Ajoutons un autre polygone sous celui-ci. Pendant que
nous y sommes, nous allons ajouter un bord rouge à celui-ci, juste pour le faire ressortir.

```
sage: p1 = polygon(((0,0),(1,4),(3,3),(4,1)), color='green', \
edgecolor='rouge', épaisseur=2)
sage: p2 = polygon(((1,0),(2,5),(3,0)), épaisseur=4, remplissage=Faux, \
zordre=-5)
sage : p1 + p2
```

Dans certaines situations, nous voulons « voir à travers » la figure du haut, pour voir celle du dessous. C'est
où l' option `alpha` entre en jeu. Cette option contrôle la transparence d'un objet. Son
les valeurs vont de 0 (invisible) à 1 (opaque).

```
sage: p1 = polygon(((0,0),(1,4),(3,3),(4,1)), color='green', \
edgecolor='rouge', épaisseur=2, alpha=0,5)
sage: p2 = polygon(((1,0),(2,5),(3,0)), épaisseur=4, remplissage=Faux, \
zordre=-5)
sage : p1 + p2
```

C'est un peu trop transparent. Nous voulons juste un indice que quelque chose se cache en dessous ; nous ne voulons pas

pour donner l'impression que le triangle bleu est au-dessus. Nous modifions alpha en conséquence.

```
sage: p1 = polygon(((0,0),(1,4),(3,3),(4,1)), color='green', \
edgecolor='rouge', épaisseur=2, alpha=0.8)
sage: p2 = polygon(((1,0),(2,5),(3,0)), épaisseur=4, remplissage=Faux, \
zordre=-5)
sage : p1 + p2
```

Soyez prudent avec la commande polygon() . Il ne peut pas lire dans vos pensées ; il suit les points de la
ordre précis que vous spécifiez. Il ne faut pas grand-chose pour changer un pentagone en un pentagramme.

```
sage: polygone(((1,0), (cos(2*pi/5),sin(2*pi/5)), \
(cos(4*pi/5),sin(4*pi/5)), \
(cos(6*pi/5),sin(6*pi/5)), \
(cos(8*pi/5),sin(8*pi/5))), épaisseur=2)
```

---

cercle(centre , rayon , options)

• le centre est une paire ordonnée
• le rayon est un nombre réel
• les options incluent
– fill = s'il faut remplir le cercle (False)
– edgecolor = couleur du bord du cercle ('blue' ou (1,0,0))
– facecolor = couleur utilisée pour remplir le cercle ('blue' ou (1,0,0))
– si vous spécifiez color , alors Sage ignore edgecolor et facecolor même si vous spécifiez eux explicitement

F IGURE 6. cercle()

```
sage : polygone(((1,0), (cos(4*pi/5),sin(4*pi/5)), \
(cos(8*pi/5),sin(8*pi/5)), \
(cos(2*pi/5),sin(2*pi/5)), \
(cos(6*pi/5),sin(6*pi/5))), épaisseur=2)
```

Nous n'illustrons pas les flèches telles que définies dans cette section ; voir un exemple dans une section suivante.
N'hésitez pas à les expérimenter par vous-même.
Des trucs "courbés". Nous considérons trois autres objets : les cercles, les ellipses et les arcs. Le premier exemple est un arc, principalement pour illustrer la relation entre l' angle , le rayon, r2 ,
et secteur . À cette fin, nous ajouterons quelques lignes pointillées pour aider à illustrer, en obtenant l'avantage supplémentaire
d'illustrer l' option de style de ligne .

---

ellipse(centre , hradius , vradius , options)
• le centre est une paire ordonnée
• hradius et vradius sont des nombres réels (rayons horizontal et vertical)
• les options incluent
– fill = s'il faut remplir le cercle (*False*)
– edgecolor = couleur du bord du cercle ('blue' ou (1,0,0))
– facecolor = couleur utilisée pour remplir le cercle ('blue' ou (1,0,0))
– si vous spécifiez color , alors Sage ignore edgecolor et facecolor même si vous spécifiez eux explicitement

FIGURE 7. ellipse()

arc(centre , rayon , options)
• le centre est une paire ordonnée
• le rayon est un nombre réel
• les options incluent
– angle = angle entre l'axe de l'arc et l'axe des x ; la longueur de cet axe sera le rayon (0)
– r2 = un deuxième rayon, pour l'arc d'ellipse ; perpendiculaire à l' angle (égal au rayon, obtenir l'arc de cercle)
– secteur = une paire ordonnée, indiquant les angles qui définissent le début et la fin du secteur ((0,2 π))

FIGURE 8. arc()

```
sage : p1 = arc((0,0), 2, angle=pi/6, r2=4, secteur=(pi/4,5*pi/6), \
épaisseur=2)
sage: p2 = line(((0,0), (2*cos(pi/6),2*sin(pi/6))), color='red', \
linestyle='dashed')
sage : p3 = ligne(((0,0), (4*cos(pi/6+pi/2),4*sin(pi/6+pi/2))), \
color='red',linestyle='dashed')
sage: p4 = ellipse((0,0),2,4,pi/6,color='red',zorder=-5)
sage : p1 + p2 + p3 + p4
```

Notez que l'axe « horizontal » de l'ellipse est incliné à l'angle $\pi/6$, comme illustré par le pointillé rouge
ligne dans le premier quadrant. L'axe « vertical » est également incliné ; donc l' option d' angle a effectivement
fait pivoter toute l' affaire de $\pi/6$. Pour tracer l'arc, nous avons spécifié le secteur ($\pi/4$, $\pi/6$). Cela démarre le
arc à l'angle $\pi/4 + \pi/6 = 5\pi/12$.
Vous vous demandez peut-être comment cette dernière affirmation peut être vraie lorsque l'arc bleu commence clairement
dans le second quadrant, plutôt que le premier quadrant - après tout, $5\pi/12 < 6\pi/12 = \pi/2$ et $\pi/2$ est la
axe vertical. La réponse réside dans la distorsion causée par l'axe « vertical » de l'ellipse ; il

tire le
l'extrémité de l'arc « vers le haut », mais puisque l'ellipse a tourné, « vers le haut » signifie en fait vers le nord-
par le nord-ouest, obtenant un point dans le deuxième quadrant. Ne nous croyez pas sur parole, cependant; essayer
avec un cercle en omettant toute spécification de r2 , et voyez comment les choses ont plus de sens.

```
sage : p1 = arc((0,0), 2, angle=pi/6, secteur=(pi/4,5*pi/6), \
épaisseur=2)
sage: p2 = line(((0,0), (2*cos(pi/6),2*sin(pi/6))), color='red', \
linestyle='dashed')
sage : p3 = ligne(((0,0), (2*cos(pi/6+pi/2),2*sin(pi/6+pi/2))), \
color='red', linestyle='dashed')
sage: p4 = cercle((0,0),2,color='red',zorder=-5)
sage : p1 + p2 + p3 + p4
```

Une parenthèse, pour ceux qui manquent de confiance en leur trigonométrie (ainsi que pour ceux qui
ne le faites pas, mais devrait). Si vous vous demandez où nous avons trouvé les paires commandées pour les lignes,
Passons en revue un peu de trigonométrie. Rappelons d'abord qu'en notation radian, un tour complet est considéré comme
avoir un angle de 2π , au lieu de $180\circ$. Nous divisons traditionnellement le tour complet à partir de là en un demi
tour ($2\pi/2 = \pi$), un quart de tour ($2\pi/4 = \pi/2$), un sixième de tour ($2\pi/6 = \pi/3$), et un douzième de tour
($2\pi/12 = \pi/6$).
Pendant ce temps, les fonctions cos α et sin α représentent les valeurs x et y d'un point sur l'unité
cercle qui se trouve à l' angle α à partir de l'horizontale. Par exemple, le point sur le cercle unité à un
angle de $\pi/3$ serait à l'horizontale soit
car
π
3
, car
π
3
=
1
2
,
3
2
.

Et si vous vouliez un cercle de rayon différent ? Dans ce cas, la relation entre cosinus, sinus, et le triangle formé par le rayon et les valeurs x et y signifie que vous devez mettre le point à l'échelle de
multipliant par le rayon du cercle. Par exemple, le point sur un cercle centré de rayon 2 à un

---

OBJETS 2D
61

text(chaîne de texte , position , options)

• la chaîne de texte est l'étiquette, entourée de guillemets simples ou doubles
• la position est une paire ordonnée
• les options incluent
– fontsize = taille du texte (10)
– rotation = angle pour faire pivoter le texte, en degrés (0)

F IGURE 9. texte()

angle de $\pi/6$ pourrait être à l'horizontale
2cos
π
6
,2péché
π
6
=
3,1 .
Un point d'un système de calcul formel est de nous éviter la peine de simplifier. Plutôt que de travailler
out (sqrt(3),1) comme un point sur la ligne, nous avons alimenté Sage l'expression (2*cos(pi/6), 2*sin(pi/6))
et laissez Sage faire le reste pour nous.
De même, lorsque nous voulions placer un point le long de l'angle de l'axe « vertical » de l'ellipse,
plutôt que de déterminer la valeur simplifiée, nous avons simplement demandé à Sage de faire la rotation, puis de trouver le
points correspondants :
(
|
4cos
(
|
\
π
6
}{{}
démarrer

+
π
2
}{{}
tourner
ֆ
|
,4sin
(
|
)
π
6
}{{}
démarrer
+
π
2
}{{}
tourner
ֆ
|
)
ֆ
|
.

Nous pouvons illustrer cela dans Sage :

```
sage : p1 = cercle((0,0),2)
sage : p2 = flèche((0,0),(2*cos(pi/6),2*sin(pi/6)), \
linestyle='dashed', color='red')
sage : p3 = flèche((0,0), (2*cos(pi/6+pi/2), 2*sin(pi/6+pi/2)), \
linestyle='dashed', color='red')
sage: p4 = arc((0,0), 2, angle=pi/6, sector=(0,pi/2), color='red',\
épaisseur=2, zorder=5)
sage : p1 + p2 + p3 + p4
-2
-1
1
2
-2
-1
1
2
```

Notez l'utilisation à la fois de arrow() et de l' option linestyle .

Texte. Ne serait-il pas agréable d'ajouter des étiquettes à cette dernière photo ? La commande text() permet

nous mettons des étiquettes dans une image. Notre exemple réutilise les images de l'exemple précédent.

OBJETS 2D
62
sage : p5 = p1 + p2 + p3 + p4
sage: p6 = text('pi/2', (.5,1.5), color='red', fontsize=18, \
rotation=30)
sage : p5 + p6
Ça a l'air bien, mais ne serait-ce pas bien si ça avait l'air un peu plus « professionnel » ? Ne serait-ce pas
être plus sympa d'avoir la lettre grecque π au lieu de quelques caractères latins ? Une façon d'accomplir
ce serait de changer la disposition du clavier sur votre ordinateur afin que vous puissiez taper des lettres grecques,
mais cela ne ressemblerait toujours pas à une vraie fraction.[1] Au lieu de cela, Sage propose une solution mathématique élégante
via L A TEX.
Sans aller trop loin dans les mauvaises herbes, L A TEX est une sorte de « langage de balisage » développé au cours de
les années 1970 et 1980. Il est extrêmement courant pour les mathématiciens de taper des papiers et des livres en
L A TEX (vous lisez un exemple) car il offre des commandes extrêmement intuitives et flexibles
pour indiquer ce que vous voulez écrire, et organise automatiquement le texte d'une manière cohérente
avec la tradition mathématique, avec le placement approprié des exposants, l'étirement du groupement
symboles, etc...
Sage n'offre pas la gamme complète du balisage L A TEX ; il n'offre qu'un sous-ensemble, mais ce sous-ensemble
suffit pour les travaux pratiques en graphisme. Pour utiliser le balisage L A TEX, vous devez essentiellement effectuer uniquement
deux tâches :
• enfermer la sous-chaîne de mathématiques dans les délimiteurs L A TEX $ et $; et
• utiliser le balisage L A TEX approprié , comme décrit au chapitre 12.
Un aparté. Les utilisateurs de L A TEX remarqueront que nous utilisons des doubles barres obliques inverses alors qu'ils
attendez-vous à des barres obliques inverses simples. C'est parce que la barre oblique inverse unique est une commande de contrôle dans Sage[2]
chaînes : \n , \r , \t ont toutes des significations spéciales (comme les autres). Vous pourriez vous en tirer avec un seul
backslash parfois, mais cela peut aussi faire des ravages dans les moments les plus incommodes. Utilisant un
la double barre oblique inverse évite systématiquement les ambiguïtés potentielles.
Et encore. . . Si vous regardez la Figure 1 à la page 252, vous remarquerez que nous avons

utilisé des barres obliques inverses simples. le
la raison en est que nous parlerons plus tard d'insérer L A TEX dans la feuille de calcul, et dans ce cas un double
La barre oblique inverse est inappropriée. C'est une confusion plutôt malheureuse pour l'apprenant, mais la base
La règle est assez simple : le balisage nécessite une seule barre oblique inverse, mais dans une chaîne de texte qui signifie
vous avez besoin d'une double barre oblique inverse.
De plus... Vous pouvez aussi utiliser L A TEX dans les cellules HTML d'une feuille de calcul ! (Voir p. 26 pour un ex-
sur la façon de créer une cellule HTML.) Entourez le texte de signes dollar, en « barre oblique inverse »

1 De plus, nous l'avons essayé. Cela ne fonctionne pas.

2 C'est parce que Sage est construit sur Python.

---

entre parenthèses, ou entre crochets inversés, et vous pouvez utiliser le balisage L A TEX à votre guise.3

Par exemple, le code suivant vous donnera une belle présentation de la définition de l'intégrale :

```
sage : %html
La définition du <b>dérivé</b> est <i>la limite de
les pentes des lignes sécantes comme la distance entre les
les points approchent de 0 ;</i> ou,\[\frac{d}{dx}f(x) = \lim_{\Delta
x\rightarrow\infty}\frac{f(x+\Delta x)-f(x)}{\Delta x}\]
```

Retour à notre programmation régulière. Reprenons cette image précédente, en utilisant L A TEX
cette fois pour que ça ait l'air bien. Si vous regardez la figure 1, vous trouverez π dans la rangée sur les lettres grecques ;
pour obtenir le symbole grec, vous voudriez le balisage \pi , que nous écrivons dans une chaîne Sage sous la forme \\pi .

```
sage: p7 = text('$\\pi/2$', (.5,1.5), color='red', \
taille de police=18, rotation=30)
sage : p5 + p7
```

Cela a l'air beaucoup mieux. Mais peut-être aimeriez-vous qu'il ressemble à une vraie fraction, au lieu d'un simple
division? Regardez à nouveau la figure 1 et vous verrez un balisage \frac . Cela se traduit par \\ frac dans un
Chaîne sage, mais elle illustre aussi autre chose à propos du balisage L A TEX : certaines commandes nécessitent
Informations Complémentaires. Cette information est généralement transmise par paires d'accolades, parfois plusieurs
paires. Pour une fraction, cela devrait avoir du sens : une fraction nécessite à la fois un

numérateur et un dénominateur.
inator, il nécessite donc deux informations, donc deux paires d'accolades. Nous illustrons cela dans le
exemple ci-dessous, qui diffère légèrement des précédents.

```
sage: p8 = text('Rotation de $\\frac{\\pi}{2}$', (.5,1.5), \
couleur='rouge', taille de police=18)
sage : p5 + p8
```

3 Eh bien, peut-être pas à votre guise, à plus d'un titre. Dans les limites du raisonnable, en tout cas.

Couleurs. Il existe plusieurs façons de définir les couleurs dans Sage. L'un est de nom, et Sage propose un
beaucoup de couleurs par nom, avec des options intrigantes telles que 'chartreuse' et 'lavenderblush' . Vous pouvez obtenir une liste complète des noms en tapant ce qui suit :

```
sage : colors.keys()
dict_keys(['automatic', 'aliceblue', 'antiquewhite', 'aqua',
'Aigue-marine', ...
```

À l'heure actuelle, il existe près de 150 couleurs prédéfinies, nous en avons donc laissé beaucoup de côté.
Vous pouvez en effet obtenir plusieurs millions de couleurs (au moins 16 millions environ) par manip-
ce qu'on appelle les « valeurs RVB ». La lumière visible peut être divisée en trois composantes : rouge,
vert et bleu - et nous pouvons demander à Sage d'attribuer à chacun de ces composants, appelés canaux, une valeur
de 0 à 1. Vous pouvez penser à "0" comme signifiant que le canal est complètement éteint, et "1" comme signifiant
le canal est complètement plein. Cela nous amène à ce qui suit :

R	g	B	interprétation	résultat
1	0	0	plein de rouge, pas de vert ni de bleu	rouge
1	1	0	plein rouge et vert, pas de bleu	

jaune
0,5 0,5 0,5
demi-puissance rouge, vert, bleu
gris
0,8 0,8 0,8 huit dixièmes de puissancerouge, vert, bleu gris clair
Pour utiliser les couleurs de cette façon dans Sage, il suffit de passer les trois nombres, entre parenthèses, comme couleur
option.[4] Pour illustrer cela, nous revisitons notre pentagone d'avant.
sage: polygone(((0,0),(1,4),(3,3),(4,1)), couleur=(0.8,0.8,0.8))
4 Il serait plus précis de la spécifier comme option rgbcolor , mais Sage interprète la couleur comme rgb, ce qui est assez norme, en tout cas. Si vous connaissez le modèle de couleur Hue-Saturation-Value, vous pouvez utiliser l' option hsvcolor
à la place, mais nous ne couvrons pas cela ici.

---

show(objet graphique , options)

- ymax = plus grande valeur y visible (déterminée par l'objet)
- ymin = plus petite valeur y visible (déterminée par l'objet)
- xmax = plus grande valeur y visible (déterminée par l'objet)
- xmin = plus petite valeur y visible (déterminée par l'objet)
- aspect_ratio = rapport de 1 unité y à 1 unité x (déterminé par l'objet)
- axes = s'il faut afficher les axes (*True*)

F IGURE 10. afficher()

Autres options pour les objets 2D. Certaines options pour les objets 2D sont plus appropriées nipulé après les avoir additionnés. Il existe plusieurs manières de les manipuler ; vous pouvez, pour
exemple, ajoutez chaque option ci-dessous directement à une commande de tracé individuelle. Quand tu as plusieurs
tracés, le plus pratique peut être via la commande show() , qui offre plusieurs options
pour affiner une image. Nous listons ceux qui nous intéressent le plus dans la figure 10. Nous les mettons au travail
sur p5 , l'image ci-dessus du cercle avec la rotation.
sage : show(p5, ymax=1, aspect_ratio=2)
Vous pouvez voir que la plus grande valeur y visible est y = 1, et le cercle semble avoir été étiré
verticalement, de sorte qu'une unité de 1 sur l'axe des y est deux fois plus longue qu'une unité de 1 sur l'axe des x.
Il est souvent pratique d'afficher des graphiques sans les axes ; show offre l' option axes pour cela.
Nous illustrons cela plus tard, mais vous pouvez essayer de retravailler certains des graphiques répertoriés ici pour le voir dans
action. Il existe d'autres options pour contrôler les axes que nous ne passons pas en revue ici ;

la plupart d'entre eux peuvent
également être contrôlé via les méthodes d'un objet de tracé. Vous pouvez lire à leur sujet en tapant le nom d'un
objet de tracé, suivi du point, suivi de la touche de tabulation ; sélectionnez celui qui vous intéresse, tapez la question
marque, et exécutez la commande pour voir son aide (comme expliqué précédemment).
tracés 2D
A présent, vous devriez avoir appris trois manières de base de représenter les relations bidimensionnelles que nous
vouloir tracer. Elles sont:
• Coordonnées cartésiennes, soit
– avec y en fonction de x ;
– comme une équation en termes de x et y uniquement, mais pas nécessairement une fonction ; ou alors
– avec x et y en fonction d'une troisième variable, t, appelée paramètre ;
et

---

tracer(f (x) , xmin , xmax , options)
• f (x) est une fonction ou une expression dans une variable
• xmin est la plus petite valeur x à utiliser pour le tracé
• xmax est la plus grande valeur x à utiliser pour le tracé
• les options incluent
– detect_poles = l' un des suivants (*False*):
  * *Faux* (dessiner directement)
  * *Vrai* (détecter la division par zéro et esquisser de manière appropriée)
  * 'show' (identique à *True* , mais inclut une asymptote)
– fill = un des suivants (*False*)
  * *Faux* (ne pas remplir)
* « Axe » ou *pur* (remplissage à l'axe des x)
g (x) (remplir la fonction g (x))
  * 'min' (remplir de la courbe à sa valeur minimale)
  * 'max' (remplir de la courbe à sa valeur maximale)
– fillcolor = couleur utilisée pour le remplissage, ou 'automatic' ('automatic')
– fillalpha = transparence du remplissage, de 0 à 1 (0,5)
– plot_points = nombre de points à utiliser lors de la génération d'une table xy (200)
F IGURE 11. plot()
• coordonnées polaires.
Sage propose des commandes pour chacune de ces représentations.
Parcelles génériques. Celle que vous utiliserez le plus souvent est la commande plot() , qui

accepte une fonction
tion, ou au moins une expression dans une variable, et produit un graphique où la valeur y est déterminée
par les valeurs x, comme par une table xy. Cette description est tout à fait littérale ; la commande fonctionne par cré-
ating un certain nombre de (x,y)-paires et connecter les points. Ses options incluent celles énumérées dans la Figure 4
à la page 53. Une chose à surveiller est que xmin et xmax ne signifient pas la même chose ici que
ils le font dans la commande show() . Ici, ils correspondent aux valeurs x les plus petites et les plus grandes pour
quel plot() génère une paire xy. Il est tout à fait possible de montrer() plus ou moins que ce montant
en définissant xmin et xmax sur des valeurs différentes dans la commande show() .
Nous illustrons cette commande sur une fonction qui a des pôles.

```
sage : p = plot(x^2 - 3/(x-1), -5, 5, detect_poles='show')
sage : show(p, ymin=-10, ymax=10, aspect_ratio=1/2)
```

---

parametric_plot((x (t) , y (t)) , (t , tmin , tmax) , options)

• x (t) et y (t) sont des fonctions ou des expressions en fonction d'un paramètre t, qui définissent le
valeur x et y d'un point
• t est le paramètre
• tmin est la plus petite valeur t à utiliser
• tmax est la plus grande valeur t à utiliser
• les options incluent
– fill = *True* (à l'axe) ou *False* (aucun) (*False*)
– fillcolor , comme dans plot()
– *fillalpha* , comme dans plot()

F IGURE 12. parametric_plot()

Si nous n'avions pas contraint les valeurs y, l'image serait très différente. (Tu pourrais vouloir essayez la commande show() sans les valeurs ymin et ymax , pour voir cela par vous-même.) De même, si vous
laisser de côté detect_poles Sage erreur relier les deux points les plus proches du pôle, qui ressemble généralement à une asymptote.

```
sage : p = plot(x^2 - 3/(x-1), -5, 5)
sage : montrer(p)
```

Comme nous l'avons laissé entendre ci-dessus, c'est techniquement faux, car cette ligne apparemment verticale ne l'est pas ; il con-
relie deux points. Si vous avez vu cela se produire dans la vraie vie, vous pourriez en conclure

qu'il s'agissait d'une asymp-
fourre-tout, mais vous pourriez aussi avoir tort de le faire ; il pourrait bien y avoir une caractéristique intéressante de la
fonction qui s'y déroule.
Tracés paramétriques. Étroitement liée à la parcelle () commande est le parametric_plot () commandat. Les tracés paramétriques sont fréquemment utilisés dans les situations où les valeurs de x et y dépendent
sur le temps particulier, ce qui se passe en physique et en ingénierie. Dans de nombreux cas, t [0,1],
où « 0 » indique la « position de départ » et « 1 » indique la « position de fin », mais il y a parfaitement
raisons d'utiliser d'autres valeurs. Ce qui suit utilise t $\in$ [0,2 π].

```
sage : var('t')
sage : parametric_plot((cos(4*t),sin(t)), (t,0,2*pi))
```

---

polar_plot (r (θ) , (θ , θ_{min} , Θ_{max}) , choix)

• r (θ) est une fonction ou une expression définie en fonction d'une autre variable θ , pour laquelle vous
peut utiliser x
• θ est la variable indépendante représentant l'angle
• θ_{min} est le plus petit angle à utiliser
• θ_{max} est le plus grand angle à utiliser
• les options incluent
– fill = *True* (à l'axe) ou *False* (aucun) (*False*)
– fillcolor , comme dans plot()
– *fillalpha* , comme dans plot()

F IGURE 13. polar_plot()

C'est le graphe entier de cette fonction paramétrique. Jouez avec différentes valeurs de tmin et tmax pour déterminer le plus petit intervalle [a, b] qui nous donne une onde complète.
L'exemple suivant illustre l'utilisation de l' option de remplissage .

```
sage : parametric_plot((t*cos(4*t),t*sin(2*t)),(t,0,2*pi),fill=True)
```

C'est déjà bien, mais vous pouvez obtenir une image plus nette encore si vous doublez cette courbe dans différents
façons. Essaye le et regarde ce qu'il se passe.
Parcelles polaires. Nous nous tournons vers les coordonnées polaires. Ceux-ci sont couramment utilisés lorsqu'il est plus conve-
nient pour décrire comment la position d'un objet change lorsqu'il tourne autour de l'origine. Un nombre de

les images difficiles à décrire en paires (x,y) deviennent assez faciles avec des coordonnées polaires, telles que
comme la fleur de lys abstraite ci-dessous.

ANIMATION
69

animer(collection , options)

• la collection peut être une liste, un ensemble ou un tuple de graphiques
• les options incluent
– xmin , xmax , ymin , ymax et aspect_ratio , comme dans show()

F IGURE 14. animer()

afficher(animation , options)

• animation est une animation générée à l'aide de la commande animate()
• les options incluent
– délai = centièmes de seconde avant de passer à l'image suivante (20)
– itérations = nombre de fois pour répéter l'animation depuis le début, où 0
indique « pour toujours » (0)
– gif = s'il faut produire une animation GIF ou une animation WEBM (*False*)

F IGURE 15. afficher()

```
sage : polar_plot(cos(2*x)*sin(3*x), (x, 0, pi), fill=True, \
épaisseur=2, fillcolor='jaune', \
couleur='goldenrod', axes=False)
```

Animation

L'animation tant au cinéma qu'en programmation informatique consiste à échanger plusieurs images
plusieurs fois par seconde. Sage propose une commande animate() qui accepte une collection en entrée et
crée une animation que nous pouvons ensuite afficher() . Dans ce cas, les options que nous fournissons à show() sont un
peu différent.

Il est important d'observer que nous ne spécifions pas les options show() traditionnelles xmin , xmax , etc.
quand on montre() une animation ; ceux-ci sont vraiment plus appropriés dans la commande animate() ,
qui doit assembler tous les cadres, et doit donc savoir ce que le plus grand et le plus petit x- et les valeurs y devraient être. Les options attendues par show sont plutôt des options qui s'appliquent à l'affichage d'un
animation, et qui ne sont pas propres à l'animation elle-même. C'est-à-dire qu'il est logique de montrer le
même animation avec un délai différent entre les images, car les images elles-mêmes ne changent pas. Il

---

cela n'a pas de sens de dire que la « même » animation aurait une valeur xmin différente ; les cadres
eux-mêmes ont changé dans ce cas.
Pour illustrer le fonctionnement de la commande animate() , nous allons jouer avec notre fleur de lys de
la section précédente.

```
sage : p1 = polar_plot(cos(2*x)*sin(3*x), (x, 0, pi), fill=True, \
épaisseur=2, fillcolor='jaune', \
couleur='goldenrod', axes=False)
sage : p2 = polar_plot(cos(3*x)*sin(4*x), (x, 0, pi), fill=True, \
épaisseur=2, fillcolor='jaune', \
couleur='goldenrod', axes=False)
sage : p3 = polar_plot(cos(4*x)*sin(5*x), (x, 0, pi), fill=True, \
épaisseur=2, fillcolor='jaune', \
couleur='goldenrod', axes=False)
sage : p4 = polar_plot(cos(5*x)*sin(6*x), (x, 0, pi), fill=True, \
épaisseur=2, fillcolor='jaune', \
couleur='goldenrod', axes=False)
sage : p5 = polar_plot(cos(6*x)*sin(7*x), (x, 0, pi), fill=True, \
épaisseur=2, fillcolor='jaune', \
couleur='goldenrod', axes=False)
sage : p6 = polar_plot(cos(7*x)*sin(8*x), (x, 0, pi), fill=True, \
épaisseur=2, fillcolor='jaune', \
couleur='goldenrod', axes=False)
sage: panim = animer((p1, p2, p3, p4, p5, p6), xmin=-1, xmax=1, \
ymin=-0,5, ymax=1,5, rapport d'aspect=1)
sage : show(panim, gif=True)
```

Vous devriez obtenir une animation en conséquence. Si vous visualisez ce texte dans Acrobat Reader, vous devez
voir cette animation ci-dessous; si vous regardez une copie papier du texte, vous devriez plutôt voir
les images individuelles de l'animation :
Vous devriez prendre un moment pour jouer un peu avec les options. Par exemple, voyez ce qui se passe
lorsque vous changez la dernière ligne de cette séquence en

```
sage : show(panim, gif=True, delay=10)
```

ou pour

```
sage : show(panim, gif=True, delay=40)
```

Vous devriez remarquer un changement de comportement certain.
Une autre chose à essayer est de supprimer l' option gif=True . Nous l'avons utilisé parce que certains
les navigateurs Web ne prennent pas en charge le format WEBM au moment de la rédaction

de cet article. Voir ce qui se passe
lorsque vous l'enlevez.
En utilisant l'approche que nous avons montrée ici, il est un peu fastidieux de créer des graphiques pour tous sauf
les animations les plus triviales. Dans un chapitre ultérieur, nous montrerons comment vous pouvez utiliser des boucles pour accélérer le
traiter.
Parcelles implicites
Certains tracés cartésiens sont facilement décrits comme des relations en termes de x et y, mais sont difficiles à
décrire comme des fonctions. Géométriquement, ils ne satisfont pas au « test de la ligne verticale » qui détermine un
fonction, et il n'est pas très facile de les réécrire sous forme paramétrique ou polaire.
Considérez le cercle. Une relation qui n'est pas un bon exemple de cette difficulté est le cercle $x^2 + y^2 = a^2$, où a est n'importe quelle constante que vous aimez (dans les limites du raisonnable).[5] Nous pouvons effectivement exprimer cela
courbe utilisant un paramètre t :
(x,y)=(un coût t,un sin t)
ou en coordonnées polaires :
r = a.
Néanmoins, l'équation traditionnelle d'un cercle est utile à cet égard : elle n'admet pas de fonction y en fonction de x ; après tout, si nous résolvons pour y, nous avons
$y = \pm a^2 - x^2$,
et avoir le choix entre deux valeurs y est un peu décevant pour une fonction. À tout le moins, c'est
incommode.
Une solution consiste à le tracer à l'aide de tracés paramétriques ou polaires, mais ce n'est pas toujours disponible, ou
pas facilement. Une alternative consiste à tracer la courbe implicitement ; dans ce cas, vous spécifiez une équation, sa
variables et leur portée. Sage transforme alors l'équation
f (x,y) = g (x,y) à f (x,y)− g (x,y) = 0,
construit une grille le long de la fenêtre que vous avez spécifiée, vérifie quels points de grille viennent raisonnablement
proche de zéro, puis les relie intelligemment pour obtenir le graphe correct de la fonction.

```
sage : var('y')
sage : implicite_plot(x**2 + y**2 == 6, (x, -3, 3), (y, -3, 3), \
remplissage=Vrai)
```

5 "Dans les limites du raisonnable?" vous marmonnez probablement. « Qu'est-ce que cela signifie ? » Nous entendons a = 0. Et fini. Et
réel, certainement réel. Complex serait mauvais, du moins pour nos besoins. Donc, en gros, un \{0}.

implicite_plot(relation , (xvar , xmin , xmax) , (yvar , ymin , ymax) , options)
• la relation est une équation en termes de deux variables
• xvar et yvar sont les indéterminés dans la fonction pour laquelle vous souhaitez tracer le long
les axes x et y, respectivement
• xmin et xmax sont les valeurs x les plus petites et les plus grandes à utiliser pour générer des points
• ymin et ymax sont les valeurs y les plus petites et les plus grandes à utiliser pour générer des points
• les options incluent
– fill = s'il faut remplir « l'intérieur » de la relation ; c'est-à-dire, (x,y) paires où le trans-
la relation formée devient négative
– plot_points = nombre de points à utiliser sur chaque axe lors de la génération du plot (150)
Notez que fillColor , fillalpha , et l' épaisseur ne sont pas valides pour les options implicit_plot () .
F IGURE 16. implicite_plot()
Précisons comment Sage a décidé quels points se trouvent à l'intérieur. Nous lui demandons de représenter graphiquement
équation
$x^2 + y^2 = 6$,
donc
$f(x, y) = x^2 + y^2$
et $g(x,y) = 6$.
Sage réécrit ceci comme
$f(x,y) - g(x,y) = 0$,
ou plus précisément
$x^2 + y^2 - 6 = 0$.
Sage divise ensuite les intervalles x et y (−3,3) en 150 sous-intervalles, ce qui établit une grille sur le
plan, et substitue chaque paire (x,y) sur la grille dans le côté gauche de l'équation. Quelque de ces points peuvent correspondre exactement à 0, mais la plupart ne le seront pas ; par exemple, (1.5,1.92) et (1.5,1.96) se trouvent
près du cercle, mais le premier donne une valeur négative (−.0636) lorsqu'il est substitué en $x^2 + y^2 - 6$,
tandis que le second donne une valeur positive (.0916). Cela dit à Sage trois choses :
• Le cercle doit passer entre ces deux points ; après tout, ses points ont la valeur 0 quand substitué en $x^2 + y^2 - 6$.
• De ces deux points, le point (1.5,1.96) est plus proche du cercle que (1.5,1.92), donc c'est il est préférable de connecter cette approximation avec d'autres approximations proches de la courbe.
• Parce que sa valeur était négative, le point (1.5,1.96) se trouve à l'intérieur du cercle, et

devrait être
rempli si fill=True .

73

Pour voir cela en action, nous pouvons expérimenter avec l' option plot_points . Nous allons tracer deux vides
cercles, chacun avec un nombre très différent de points de tracé.

```
sage: p1 = implicite_plot(x^2+y^2-6, (-3,3), (-3,3), \
plot_points=5, zorder=5)
sage: p2 = implicite_plot(x^2+y^2-6, (-3,3), (-3,3), \
color='rouge', zorder=-5, \
plot_points=300)
sage : p1 + p2
```

Le graphique bleu, qui était pointé avec seulement 6 points de tracé sur chaque axe (la fin du sous-intervalle
points ajoutent 1 à l' option plot_points), semble irrégulier et inégal, tandis que le graphique rouge semble
lisse, bien qu'il soit généré de la même manière. Cette illusion de douceur est due à la
plus grand nombre de points connectés pour faire le graphique.

Pour bien comprendre l'importance du nombre de points d'intrigue dans des situations pratiques, regardons
à un autre exemple.

```
sage : implicite_plot(x*cos(x*y)-y, (x,-15,15), (y,-15,15))
```

Cette courbe a l'air tout simplement. . . impressionnant. Mais est-ce une courbe connexe ? Il semble avoir un grand
nombre d'éléments irréguliers. Certaines pièces sont également déconnectées ; ce n'est pas mauvais en soi, mais le long
avec l'irrégularité, cela suggère un problème. C'est un cas où vous vous attendriez à une augmentation
dans le nombre de points de tracé pour être extrêmement utile, et il en est ainsi. Si vous augmentez le nombre de
plot_points d'environ 50 ou plus, encore et encore jusqu'à ce que les choses commencent à devenir claires, finalement vous
arriver à quelque chose comme ça :

Page 74


```
sage : implicite_plot(x*cos(x*y)-y, (x,-20,20), (y,-20,20), \
plot_points=500)
```

Il y a bien une différence dans cette image, n'est-ce pas ! De plus, il y a une grande différence dans le
le temps qu'il a fallu pour les tracer, n'est-ce pas ? Certaines choses semblent encore un peu

floues, nous pouvons donc prendre ceci
un peu plus loin :

```
sage : implicite_plot(x*cos(x*y)-y, (x,-20,20), (y,-20,20), \
plot_points=1000)
```

Remarquez le compromis : plus de points de tracé résout plus de divergences, mais conduit également à un temps plus long.

À ce stade, il n'y a aucune raison d'augmenter `plot_points` plus haut, car il n'y a aucune raison de penser que nous avons
perdu toutes les fonctionnalités intéressantes.

Nous concluons par quelques mots d'avertissement. Il peut arriver que vous négligez de spécifier xvar et
yvar lors de la création de votre tracé implicite ; c'est une erreur assez naturelle à faire, d'autant plus que `plot()`
ne vous oblige pas à spécifier une variable. Dans ce cas, vous recevrez un *DeprecationWarning* :

```
sage : implicite_plot(x**2 + y**2 - 6, (-3,3), (-3,3))
DépréciationAvertissement :
Les plages sans nom pour plus d'une variable sont
obsolète et sera supprimée d'une future version de Sage ; toi
peut utiliser des plages nommées à la place, comme (x,0,2)
```

L'avertissement ici est à la fois explicite et utile : il vous indique quel est le problème et suggère le
solution de contournement. L'utilisation correcte est celle que nous avons décrite ci-dessus, et répétez ici pour faire bonne mesure :

```
sage : implicite_plot(x**2 + y**2 - 6, (x,-3,3), (y,-3,3))
```

Vous rencontrerez également des problèmes si vous négligez de définir `y` comme indéterminé, ou si vous négligez
pour spécifier les plages pour `x` et `y` ; ceux-ci conduisent à *NameError* (car `y` est inconnu) et *TypeError*
avertissements (car Sage attend un nombre différent d'arguments).

Des exercices

Vrai faux. Si la déclaration est fausse, remplacez-la par une déclaration vraie.

1. Nous avons répertorié toutes les options possibles pour les commandes de traçage.
2. Vous n'êtes pas obligé de fournir des valeurs pour tous les arguments que nous listons pour une commande.
3. Les valeurs des couleurs RVB vont de 0 à 255.
4. Peu importe ce que vous faites, vous êtes coincé avec des axes dans chaque parcelle de Sage.
5. Les options de `zorder` permettent de placer des objets "au-dessus" ou "en face" les uns des autres.
6. L'une des raisons d'utiliser la commande `polygon()` au lieu de la commande `line()` est que vous ne

devez re-spécifier le point de départ comme point final.
7. La transparence d'un remplissage peut différer de la transparence de la courbe en cours de remplissage.
8. Sage génère des graphiques de la même manière que nous apprenons à le faire : connecter les points entre les paires xy.
9. Comme une calculatrice graphique, Sage est incapable de détecter les endroits où se produisent des asymptotes.
10. La meilleure façon de représenter graphiquement la courbe $y^2 = x^3 + x + 1$ est de résoudre y en prenant la racine carrée,
tracer les branches positives et négatives séparément à l'aide d'une commande `plot()` , puis combiner
les en utilisant l'addition - un peu comme vous le faites sur une calculatrice graphique.
11. Bonus : lorsque vous ne connaissez pas la réponse à une question Vrai/Faux, la meilleure stratégie consiste à
répondez « Discutable », puis priez pour un crédit partiel.
Choix multiple.
1. Si les coordonnées x et y d'un objet sont des fonctions du temps, le meilleur choix pour tracer un graphique
de son mouvement est probablement :
A. `intrigue()`
B. `implicit_plot ()`
C. `parametric_plot()`
D. `polar_plot()`
2. Si la distance d'un objet à un point central varie lorsqu'il tourne autour du centre, le meilleur
choix pour tracer un graphique de son mouvement est probablement:
A. `intrigue()`
B. `implicit_plot ()`
C. `parametric_plot()`
D. `polar_plot()`
3. Si une fonction est bien définie par une variable, le meilleur choix pour tracer son graphique est probablement :
A. `intrigue()`
B. `implicit_plot ()`
C. `parametric_plot()`
D. `polar_plot()`
4. Pour modifier le degré de transparence d'un objet graphique, nous utilisons cette option :

A. `alpha`

B. opacité
C. la transparence
D. zorder
5. Pour changer quel objet semble « au-dessus » ou « devant » un autre, nous utilisons cette option :
A. alpha
B. profondeur
C. niveau
D. zorder
6. Quel aspect d'un tracé pourrait suggérer que vous ayez besoin d'augmenter la valeur de plot_points ?
A. On dirait qu'il a une asymptote.
B. Des lignes irrégulières ou des courbes « tordues » apparaissent lorsque vous vous attendez à de la douceur.
C. Une équation produit deux courbes ou plus.
D. Nous avons une courbe approximative au lieu de la courbe exacte.
7. Si un plot() produit une image dont une partie ressemble à une ligne verticale, quelle option doit
vous modifiez pour tester une asymptote?
A. detect_poles
B. draw_asymptotes
C. plot_points
D. zorder
8. Si vous obtenez une *NameError* lors de la création d'un implicit_plot() , alors, selon toute vraisemblance, vous avez fait le
erreur suivante :
A. Vous avez oublié de spécifier une équation en x et y , plutôt qu'une expression en x seule
B. Vous avez mal orthographié le nom d'une option à implicite_plot()
C. Vous avez oublié de définir le y indéterminé en utilisant var()
D. Vous avez oublié de spécifier x et y comme xvar et yvar
9. Si vous obtenez une *TypeError* lors de la création d'un implicite_plot() , alors selon toute vraisemblance vous avez fait le
erreur suivante :
A. Vous avez mal orthographié le nom, implicite_plot()
B. Vous avez mal orthographié le nom d'une option à implicite_plot()
C. Vous avez oublié de définir le y indéterminé en utilisant var()
D. Vous avez oublié de spécifier les plages pour x et y
10. Si nous enseignons le calcul et que nous voulons illustrer comment trouver l'aire entre les courbes
x^2 et x sur [0,1], quelle commande montrerait la région que nous voulons ? (L'intrigue doit également
inclure les deux courbes qui définissent la région.)

A. `plot(x**2-x, 0, 1, fill=True)`

B. `tracé(x**2, 0, 1, remplissage=x)`

C. `plot(x**2, 0, 1, fill=True) - plot(x, 0, 1, fill=True)`

D. `plot(x**2, 0, 1, fill=x) + plot(x, 0, 1)`

Bonus : vous êtes bloqué sur une île déserte. Lequel de ces articles préféreriez-vous avoir avec
tu?

A. une serviette

B. un ouvre-boîte

C. une bouteille de crème solaire

D. un ordinateur portable fonctionnel avec Sage installé

Réponse courte.

DES EXERCICES
77

1. Le professeur Proofsrock affirme que nous pouvons voir la courbe $y = x^2$ comme une courbe paramétrique $y = t^2$,
$x = t$. Tracez les deux courbes, en utilisant `plot()` pour la première et `parametric_plot()` pour la seconde. Fais
vous êtes d'accord avec son averral? Qu'est-ce qui, à part les intrigues, vous fait accepter ?

2. Pourquoi pensez-vous que les valeurs par défaut des options `show()` `ymax` , `ymin` et `aspect_ratio` dépendent de l'objet affiché, plutôt que de valeurs spécifiques ? Donnez une brève explication pour
chaque option.

Programmation.

1. Utilisez Sage pour tracer les fonctions suivantes sur l'intervalle [1,5] sur le même graphique. Donnez à chacun
une couleur différente et étiquetez-le par son nom avec un objet `text()` .

x, x^2 , $\log_{10} x$, e^x

Classez les fonctions selon lesquelles on grandit le plus rapidement sur le long terme. Ensuite, utilisez Sage
Facilités de calcul pour confirmer que votre classement est correct à $x = 10$ et également à $x = 100$. Astuce :
La dérivée vous indique le taux de variation en un point.

2. À l'aide de vos connaissances en calcul, procédez comme suit :

(a) Choisissez une fonction transcendantale $f(x)$ et un point $x = a$ auquel la fonction est définie.

(b) Calculer la dérivée de f en $x = a$. Si la dérivée est 0, retournez à l'étape (a) et choisissez un point différent.

(c) Tracer f au voisinage de a. Rendre cette courbe noire, d'épaisseur 2.

(d) Ajoutez à cela un tracé de la droite tangente à f en $x = a$. Faites cette ligne en bleu et en

pointillés.
(e) Ajoutez à cela un tracé de la droite normale à x = a. Faites cette ligne verte et en pointillés.
Astuce : La ligne normale passe par x = a et est perpendiculaire à la ligne tangente.
(f) Ajoutez un gros point rouge en (a, f (a)).
(g) Ajoutez des étiquettes pour le point, la courbe et la ligne. Utilisez L A TEX. La couleur de chaque étiquette doit
correspond à celui de l'objet qu'il étiquette.
(h) montrer () l'image à un rapport hauteur/largeur approprié à un résultat agréable.
3. Le théorème de la valeur moyenne du calcul indique que si f est une fonction continue sur l'intervalle
[a, b], alors on peut trouver un c ∈ (a, b) tel que
f (c) =
1
b−a
$\int_{une}^{b}$
f (x) dx,
et cette valeur y (f (c)) est en fait une valeur « moyenne » de la fonction sur l'intervalle. Dans ce
problème, vous illustrerez ce résultat.
(a) Choisissez une fonction quadratique f (x). Choisissez un intervalle raisonnable [a, b]. Tracer le graphique
de f sur [a, b], remplissant à la fois les valeurs y minimale et maximale. (Ceci techniquement vous oblige à créer deux tracés, puis à les combiner.)
(b) Trouvez la valeur de c qui satisfait le théorème de la valeur moyenne. (il faudra faire ça à la main, car nous n'avons pas encore expliqué comment utiliser Sage pour résoudre des problèmes. C'est un quadratique
équation, cependant, donc ça ne devrait pas être si difficile.)
(c) Ajoutez la ligne y = f (c) à votre tracé rempli de f (x). Ajoutez également une ligne de (c,0) à (c, f (c)).
Ces lignes doivent avoir une couleur différente de celle du tracé ou de son remplissage.
(d) Expliquez comment cette image finale illustre le théorème de la valeur moyenne.
4. Un nombre complexe a la forme a + bi. Le plan complexe est une représentation du champ dans le plan cartésien, avec le point (a, b) correspondant au nombre a + bi.

DES EXERCICES
78

(a) Choisissez trois nombres complexes a + bi, avec chaque point correspondant du complexe plan dans un quadrant différent. Créez des flèches reliant l'origine à chaque point, chaque flèche d'une couleur différente.
(b) Utilisez Sage pour évaluer le produit i (a + bi) pour chacun des points que vous venez de choisir. Créer

trois nouvelles flèches reliant l'origine à chaque point, dans une couleur correspondant au couleur que vous avez utilisée pour le point d'origine.
(c) Tracez les six flèches simultanément. Quel est l'effet géométrique de la multiplication (a + bi) par
je?
5. Une grande partie des mathématiques supérieures s'intéresse aux courbes elliptiques, qui sont lisses, non
courbes auto-sécantes de forme générale
$y^2 = x^3 + ax + b$.
Tracez plusieurs courbes de cette forme, en utilisant différentes valeurs pour a et b. Quelle(s) valeur(s) produit(s)
courbes elliptiques ? Quelle(s) valeur(s) ne correspond(nt) pas ? Décrivez le schéma général de ce qui se passe comme
les valeurs de a et b changent.
6. Nous construisons sur le problème précédent. Une grande partie des mathématiques supérieures considère les courbes elliptiques
sur un corps fini, tel que $_p$ où p est premier. Choisissez l'une des courbes elliptiques de la problème précédent. Soit p = 5, et substitue toutes les valeurs possibles de x et y dans l'équation.
(Il n'y en a que 25, donc ce n'est pas trop lourd, et vous pouvez probablement en faire beaucoup dans
ta tête. N'oubliez pas de les faire modulo 5.) Tracez les points dans le plan. Est ce que ca courbe ressemble-t-elle à celle que vous aviez auparavant ?
Bonus : Pourquoi avons-nous pu dire, avec confiance, que dans #6 il y a 25 valeurs possibles de x et
y substituer ? Et si on avait choisi p = 7 ; comment le nombre de points aurait-il changé ?

CHAPITRE 4

Définir vos propres procédures

Résoudre des problèmes plus difficiles nécessite un peu plus qu'une simple liste directe de commandes. Il est souvent utile d'organiser de nombreuses tâches souvent réutilisées en une seule commande qui donne
nous une poignée pratique sur chacun d'eux. Faire cela nous permettrait également de traiter des tâches plus
abstraitement, nous aidant à voir la forêt plutôt que les arbres. Il est plus facile de réutiliser le même code
à différents endroits, et de même pour que quelqu'un d'autre lise et, si désiré, modifie le code à leur
propres fins. Cette idée de modularité nous rend plus « productifs », en ce sens que nous devenons
meilleurs résolveurs de problèmes.

Techniquement, vous l'avez déjà fait : chaque commande Sage que vous avez utilisée pour simplifier
expressions et dessiner des images est vraiment un ensemble d'autres commandes qui a été conçu (généralement
soigneusement) pour s'attaquer à cette tâche particulière. Vous pouvez en fait afficher une partie du code pour la plupart des Sage
commandes utilisant un double point d'interrogation :

```
sage : simplifier ??
Signature:
simplifier(f)
La source:
def simplifier(f):
r"""
Simplifiez l'expression 'f'.
EXEMPLES : Nous simplifions l'expression 'i + x - x'.
::
sauge:
f = I + x - x; simplifier(f)
je
En fait, l'impression de 'f' donne la même chose - c'est-à-dire le
forme simplifiée.
"""
essayer:
return f.simplify()
sauf AttributeError :
retour f
Déposer:
/Applications/sage-6.7-untouched/local/lib/python2.7/
site-packages/sage/calcul/functional.py
Taper:
une fonction
```

Cela nous donne en fait un peu plus que le code. Il répertorie d'abord la signature de la commande, un
modèle pour la façon dont la commande est utilisée. Il répertorie ensuite le "code source", la liste des commandes
qui définissent la commande `simplifier()` . Ces commandes commencent après la ligne intitulée `Source :` et
continuer jusqu'à la ligne intitulée `File:` ; cette dernière ligne indique le fichier sur le disque où la commande
est trouvé. Si vous ouvriez ce fichier, vous y trouveriez la même source répertoriée ci-dessus. Remarquer
que la plupart du code source dans ce cas se compose de la même documentation que vous voyez lorsque vous
tapez `simplifier?` , reflétant que la documentation de Sage est intégrée dans le code lui-même.

Une grande partie du reste de ce texte est dédiée à vous aider à comprendre ce que tout ce code source
moyens, et comment l'utiliser à vos propres fins. Nous ne nous attendons pas à ce que vous deveniez un sage à part entière
développeur au moment où vous avez terminé (cela irait bien au-delà de sa portée) mais nous nous attendons à ce que vous
être capable d'écrire de nouvelles commandes simples qui accomplissent des tâches spécifiques qui seraient utiles pour
la recherche et l'enseignement. Cela demande un certain dévouement de votre part, mais ne vous inquiétez pas : cela demande
dévouement de notre part également, pour lequel nous retrouvons maintenant nos manches figuratives et approfondissons.

Définir une procédure

Les commandes que vous définissez dans Sage sont souvent appelées fonctions, ce qui est en accord avec les
utilisation mathématique en ce sens que vous fournissez des arguments à une fonction et qu'elle fournit un résultat. Cependant, un
La fonction Sage et une fonction mathématique ne sont pas tout à fait la même chose, donc pour que les choses soient claires
nous adopterons le terme procédure pour une nouvelle commande que vous définissez dans Sage.

Le format de base. Pour définir une procédure, on utilise le format suivant :

```
sage : def nom_procédure(arguments) :
première_déclaration
deuxième_instruction
...
```

Vous pouvez considérer def comme un raccourci pour "définir". C'est un exemple de mot clé ou de mot réservé,
une séquence de caractères qui a une signification particulière dans un langage de programmation, et qui
le programmeur ne peut pas attribuer une autre signification. Ce qui suit est un nom d'identifiant valide
(voir les règles p. 31) et est le nom de votre nouvelle procédure. Vous devez ensuite ouvrir et fermer
parenthèses. En option, vous pouvez inclure des identifiants supplémentaires entre parenthèses ; alors que ce
n'est pas nécessaire, il constitue le moyen le plus fiable et le plus efficace de transmettre des informations à votre procédure.

Nous appelons ces identifiants arguments,[1] et les examinera plus attentivement dans un instant.

La prochaine chose à noter est que les « déclarations » sont en retrait. C'est un principe important
que Sage hérite de Python, dont vous vous souviendrez qu'il s'agit du langage de

programmation que nous utilisons pour interagir
avec Sage : chaque fois que nous arrivons à un point dans un programme où nous devons organiser des déclarations
"dans" d'autres instructions, ou lorsque certaines commandes "dépendent" d'autres, nous indiquons lesquelles
les instructions dépendent des autres utilisant l'indentation.[2] Cette configuration est appelée structure de contrôle ; ici,
la structure de contrôle est l' instruction `def` , qui définit une procédure qui, pour nos besoins, peut
être considéré comme un raccourci pour une liste d'autres déclarations.

1 Bien nommé, car écrire un programme ressemble si souvent à une dispute avec l'ordinateur. Textes d'informatique appellent souvent les arguments « paramètres ».

2 La plupart des langages informatiques ne nécessitent pas d'indentation, mais désignent des mots-clés ou des caractères spéciaux pour indiquer
où commence et se termine une structure de contrôle. Par exemple, de nombreuses langues suivent C dans l'utilisation des accolades `{ ... }` pour ouvrir
et fermer une structure; d'autres utilisent des variations sur `BEGIN ... END` .

DÉFINIR UNE PROCÉDURE
81

Pour l'instant, regardons une procédure assez simple. Il ne prend aucun argument, mais n'est pas entièrement
inutile.

```
sage : def salutations() :
print('Salutations !')
```

Vous pouvez utiliser des guillemets simples ou doubles autour du mot « Salutations », mais assurez-vous de le faire
régulièrement.

Si vous l'avez tapé correctement, Sage l'acceptera silencieusement. Si vous tapez quelque chose de mal et
Sage le détecte, il signalera une erreur. Par exemple, si vous oubliez de mettre en retrait, vous verrez ceci :

```
sage : def salutations() :
print("Salutations!")
Fichier "<ipython-input-33-33dcc1adcd0e>", ligne 2
print("Bonjour !")
^
Erreur d'indentation :
s'attendait à un bloc en retrait
Si vous souhaitez coller du code dans IPython, essayez les %paste et %cpaste
fonctions magiques.
```

Il est peu probable que vous voyiez une *IndentationError* si vous tapez votre programme dans un work-
feuille ou à la ligne de commande, car Sage indente automatiquement pour vous parfois vous devriez le faire.

Voici une autre erreur que vous pouvez voir :

```
sage: def salutations()
Fichier "<ipython-input-34-583f0b909fc6>", ligne 1
def salutations()
^
Erreur de syntaxe:
Syntaxe invalide
```

Voyez-vous le problème ici? Sinon, passez quelques minutes à comparer cette tentative avec la
un réussi. En particulier, regardez la fin de la ligne, ce qui est là le symbole carats ^ est montrer du doigt. Cette position est l'endroit où Sage rencontre ses problèmes.

Nous revenons maintenant à l'hypothèse que vous avez défini la procédure avec succès. Vous pouvez utiliser
votre nouvelle procédure en la tapant, suivie d'une paire de parenthèses, et en exécutant la ligne.

```
sage : salutations()
Les salutations!
```

Sage a fait ce que vous avez demandé. Il a recherché la procédure nommée `salutations()` , a vu que cela
la procédure doit `imprimer('Greetings!')` , et fait ce qui a été demandé.

Un aparté. La signification de la commande `print` est, espérons-le, évidente : elle imprime tout ce qui suit.
bas.

Dans les anciennes versions de Sage, `print` était le mot-clé ; vous ne pouviez pas lui attribuer une nouvelle valeur :

```
sauge : imprimer = 2
Fichier "<ipython-input-38-7c34f307b821>", ligne 1
print = Entier(2)
^
Erreur de syntaxe:
Syntaxe invalide
```

Maintenant, cependant, l' `impression` est l'une des « procédures standard » de Sage ; un nom auquel vous pouvez réattribuer
des fins plutôt malheureuses si vous ne faites pas attention :

```
sauge : imprimer = 2
sauge : imprimer
2
sauge : impression(3)
TypeError Traceback (appel le plus récent en dernier)
<ipython-input-8-fd56b57ba354> dans <module>()
----> 1 impression(Entier(3))
Erreur-type:
L'objet 'sage.rings.integer.Integer' n'est pas itérable
```

Si vous redéfinissez par inadvertance une procédure standard, n'oubliez pas que la commande `reset`
remettez-le à son sens originel :

```
sage : reset("imprimer")
sauge : impression(3)
3
```

Hélas, print ne coopère pas avec L A TEX, vous ne pouvez donc pas générer une jolie sortie de cette façon. À
faire cela, vous devez utiliser L A TEX à partir d'une cellule HTML (voir p. 26), ou créez un tracé.

Un autre aparté. Les programmeurs laissent généralement des commentaires dans le code pour expliquer comment un processus
la durée est censée fonctionner. La principale façon de le faire dans Sage est d'utiliser le symbole # ; chaque fois que Sage
voit ce symbole, il ignore tout ce qui suit. Par exemple, l'implémentation suivante
de salutations () est équivalent à celui ci-dessus.

```
sage : def salutations() :
# saluer l'utilisateur
print('Salutations !')
```

Sage ignore simplement la ligne qui commence par le symbole # et passe immédiatement à la suivante
une.

C'est une très bonne idée de laisser des commentaires dans les procédures, surtout au début d'une séquence
de lignes compliquées. Les auteurs peuvent attester du fait qu'ils ont écrit un code un an qui a semblé évident au moment de le programmer, puis y est revenu des semaines, des mois, voire
des années plus tard et je me suis demandé : « Pourquoi diable ai-je fait ça ? » Cela prend alors plusieurs minutes,
des heures, voire des jours de perplexité sur le code pour comprendre comment il fonctionnait. La plupart des procédures

---

nous écrivons dans ce texte ne sont pas très longs, mais nous inclurons quand même des commentaires dans nombre d'entre eux
façon d'illustrer le besoin.

Arguments

Un argument est une variable qu'une procédure utilise pour recevoir des informations nécessaires ou simplement utiles.
tion. Nous pouvons voir un argument comme un espace réservé pour les données ; son nom n'existe qu'à l'intérieur de la procédure,
et l'information est oubliée une fois la procédure terminée. Un argument ne contient qu'un copie des informations envoyées, et non des informations originales elles-mêmes. On peut illustrer l'idée de base
en modifiant la procédure de la section précédente :

```
sage: def salutations(nom):
print('Salutations,', nom)
```

Ici, `name` est un argument de `Greetings()` . C'est aussi un argument obligatoire, car nous avons spécifié non
valeur par défaut. Nous discuterons maintenant des arguments obligatoires et facultatifs ; pour l'instant, il suffit de
comprenez que vous devez fournir une valeur pour les arguments obligatoires. Par exemple, ce qui suit
l'utilisation de `salutations ()` est légitime dans Sage :

```
sage : salutations('Pythagore')
Salutations, Pythagore
```

Que s'est-il passé? Lorsque Sage lit la commande, il voit que vous voulez appeler `salutations ()` avec
un argument, qui est la chaîne `'Pythagore'` . Il copie cette information dans la variable `name` , puis procède à l'exécution de toutes les commandes en retrait sous la définition de `Greetings()` .

D'un autre côté, maintenant que nous avons redéfini `Greetings()` , l'utilisation suivante entraîne une erreur, même si cela fonctionnait bien plus tôt:

```
sage : salutations()
TypeError Traceback (appel le plus récent en dernier)
<ipython-input-44-deb3f0bd6096> dans <module>()
----> 1 salutations()
Erreur-type:
salutations () prend exactement 1 argument (0 donné)
```

Un exemple particulier : l'équation d'une droite tangente. Nous allons nous concentrer sur un usage particulier pour
ceci : supposons que vous vouliez écrire une procédure qui calcule automatiquement l'équation en un point
x = a d'une droite tangente à une courbe définie par une fonction f (x). Vous vous souviendrez que l'équation de
une droite de pente m et passant par un point (a, b) a la forme

$$y - b = m\,(x - a),$$

que nous pouvons réécrire comme

$$y = m\,(x - a) + b.$$

Dans ce cas, m est la dérivée de f en x = a ; ou, m = f (a). On peut donc réécrire l'équation sous la forme

$$y = f\,(a)\cdot(x - a) + b.$$

Nous voulons une procédure qui produit une telle équation, ou au moins la partie droite de celle-ci.

Pour définir cette procédure, nous devons répondre à plusieurs questions :

• De quelles informations la procédure a-t-elle besoin ?

Nous avons besoin de la fonction f (x) et de la valeur x a.
• Comment utilisons-nous ces informations pour obtenir le résultat souhaité ?
– Nous devons d'abord calculer f (a). Cela nécessite de calculer f (x), puis de substituer x = a.
– Nous devons également calculer b. C'est la valeur y à x = a, il suffit donc de calculer FA); c'est-à-dire en substituant x = a dans f (x).
• Comment communiquons-nous le résultat au client ?
Pour l'instant, nous allons simplement l' imprimer , mais nous finirons par discuter d'une meilleure façon.
La mise en commun de ces informations nous amène à la définition suivante :

```
sage : def tangent_line(f, a) :
# forme point-pente d'une ligne
b = f(a)
df(x) = diff(f,x)
m = df(a)
résultat = m*(x - a) + b
print('La ligne tangente à', f, 'à x =', a, 'est',)
print(résultat, '.')
```

Mettons cela en pratique. Un exemple simple que vous pouvez vérifier à la main est l'équation de
une droite tangente à y = x 2 en x = 1.

```
sage : tangente_line(x**2, 1)
/Applications/sage-6.7-untouched/src/bin/sage-ipython:1 :
DépréciationAvertissement :
Substitution à l'aide de la syntaxe d'appel de fonction
et les arguments sans nom sont obsolètes et seront supprimés d'un
future version de Sage ; vous pouvez utiliser des arguments nommés à la place, comme
EXPR(x=..., y=...)
Voir http://trac.sagemath.org/5930 pour plus de détails.
#!/usr/bin/env python
La droite tangente à x^2 en x = 1 est 2*x - 1 .
```

Oops! nous avons rencontré à nouveau ce *DeprecationWarning* . C'est une chose unique dans chaque Sage
session, et nous obtenons la bonne réponse à la fin (2*x - 1).
Même ainsi, nous devrions le changer. Seulement, où se passe-t-il ? Nous devons vérifier chaque déclaration
de la procédure :

• df(x) = diff(f,x)

Cela ne peut pas être le problème, car cela définit une procédure.

• m = df(a)

Cela utilise df dans la syntaxe d'appel de fonction, c'est donc un candidat pour le problème. Cependant, il
ne peut pas être le problème, car nous avons défini correctement df en tant que fonction. Il est ap-
priate d'utiliser la syntaxe d'appel de fonction avec les fonctions.

• b = f(a)

Cela utilise f dans la syntaxe d'appel de fonction, c'est donc un candidat pour le problème. Comme f est un
argument, dans lequel Sage a copié la valeur fournie lorsque nous avons appelé tangent_line() , nous devons regarder en dehors de la procédure pour voir si c'est bien le problème. Nous l'avons appelé
à l'aide de la commande,

```
tangente_line(x**2, 1)
```

Cela doit être le problème, car x**2 est une expression, pas une fonction.
Ainsi, une façon d'éviter cet avertissement consiste à définir une fonction avant d'appeler tangent_line() , comme suit :

```
sage : f(x) = x**2
sage : tangente_line(f, 1)
La droite tangente à x |--> x^2 à x = 1 est 2*x - 1 .
```

Ce n'est pas très pratique, cependant, car cela nécessite beaucoup plus de frappe. Il impose également une condition
sur le client.
Une bien meilleure solution consiste à ajuster le programme lui-même, afin qu'il ne suppose pas qu'il reçoit
une fonction alors qu'en fait, ce n'est pas le cas.[3] Nous pouvons le faire en changeant la ligne b = f(a) pour utiliser un
substitution explicite, b = f(x=a) . Essayons ça.

```
sage : def tangent_line(f, a) :
# forme point-pente d'une ligne
b = f(x=a)
df(x) = diff(f,x)
m = df(a)
résultat = m*(x - a) + b
print('La ligne tangente à', f, 'à x =', a, 'est',)
print(résultat, '.')
```

Essayons maintenant de l'exécuter :

```
sage : tangente_line(x^2, 1)
La droite tangente à x^2 en x = 1 est 2*x - 1 .
```

Étant donné qu'un *DeprecationWarning* n'apparaît qu'une seule fois, il n'est peut-être pas évident que nous ayons réellement
a résolu le problème, car Sage peut à nouveau supprimer l'avertissement. Vous pouvez le vérifier en redémarrant
Sage : en mode feuille de calcul, cliquez sur le bouton Redémarrer ; sur la ligne de commande, tapez exit , puis
redémarrez Sage.
Arguments facultatifs. Sage autorise également les arguments facultatifs. Dans ce cas, il est souvent logique
pour fournir une valeur par défaut pour un argument. De cette façon, vous n'avez pas besoin de fournir des valeurs « évidentes » pour

le cas général ; vous ne pouvez les fournir qu'en cas de besoin.

3 Si vous êtes habitué aux langages typés tels que C, Java ou Eiffel, vous remarquerez qu'il s'agit d'un cas où l'utilisation de types - qui peuvent souvent sembler trop contraignants - aide en fait le programmeur. Non seulement Sage
pas besoin de types, il n'offre vraiment aucune facilité pour eux, à moins que vous n'utilisiez Cython. Nous discutons de Cython à la fin de
le texte.

Vous avez déjà rencontré cela. Revenez, par exemple, à la figure 1 à la page 53 et à la figure ure 4 à la page 53. Les deux parlent d'options, qui sont en réalité des arguments optionnels pour le
procédures. Par exemple, vous pouvez taper

```
sage: point((3,2))
```

ou tu peux taper

```
sage: point((3,2), pointsize=10, color='blue', alpha=1, zorder=0)
```

car les deux veulent dire la même chose.

Cet exemple illustre également comment les arguments facultatifs permettent au client de se concentrer sur le problème.
Vous vous moquez souvent de la couleur d'une pointe, de sa taille, etc. tu veux juste voir où il se trouve, généralement par rapport à autre chose. Si cela est utile, vous voudrez peut-être spécifier certains
de ces autres options ; par exemple, vous pourriez regarder plusieurs points sur une courbe et aider à les distinguer, vous les coloriez différemment. Mais cela pourrait être complètement inutile,
et même si vous vouliez spécifier la couleur, la taille précise, la transparence et l'ordre z pourraient toujours
pas important. Rendre ces arguments facultatifs signifie que vous n'avez pas à les spécifier.

Alors, comment nommez-vous un argument facultatif lors de l'écriture de vos propres procédures ? Après le
le nom de la variable, placez un signe égal, suivi de la valeur par défaut qu'elle doit prendre. Faisons
voir comment nous pouvons faire cela dans notre procédure pour saluer quelqu'un : nous aimerions peut-être que la valeur par défaut soit
être 'Leonhard Euler' .

```
sage: def salutations(nom='Leonhard Euler'):
print('Salutations,', nom)
```

Le nom de l' argument est désormais facultatif. Comme avec la version précédente, nous pouvons maintenant fournir un nom :

```
sage : salutations('Pythagore')
Salutations, Pythagore
```

. . . et vous pouvez également nommer explicitement l'argument facultatif (cela est utile lorsqu'il y a plusieurs

arguments facultatifs, et vous ne voulez pas vous soucier de les lister dans le bon ordre):

```
sage : salutations(nom='Pythagore')
Salutations, Pythagore
```

...mais contrairement à la version précédente, on peut utiliser la procédure sans valeur par défaut :

```
sage : salutations()
Salutations, Léonhard Euler
```

Fournir des indéterminés comme arguments par défaut. Il est souvent utile de fournir une valeur indéterminée.

nate comme argument par défaut. Pour voir un exemple, revenons à notre exemple avec les lignes tangentes.

Cela fonctionne très bien si l'expression est définie en termes de x, mais de nombreuses expressions sont plus appropriées

---

défini en termes d'un autre indéterminé, tel que t. Notre procédure calculera-t-elle toujours la tangente

lignes pour ces expressions?

```
sage : var('t')
sage : tangente_line(t**2, 1)
La droite tangente à t^2 en x = 1 est t^2 .
```

Ce n'est certainement pas juste. La réponse aurait dû être $2t - 1$, pas t^2. Qu'est ce qui ne s'est pas bien passé?

Plusieurs choses:

• `df(x) = diff(f, x)` différencie `f` par rapport à `x`, pas par rapport à `t`. Le résultat est que `df(x)` finit par être 0, plutôt que 2t.

• `m = df(a)` travaille avec la mauvaise valeur de `df(x)`, nous nous retrouvons donc avec m = 0 à la place de m = 2.

• `b = f(x=a)` demande à Sage de substituer `a` à la place de `x`, pas à la place de `t`. Le remplacement effectivement ne fait rien, avec pour résultat que `b` finit par être t^2, plutôt que 1.

• Mettez le tout ensemble avec `m*(x - a) - b` et il devient clair pourquoi Sage renvoie `t^2 à la` place de `2*t - 1`.

Une façon de résoudre ce problème serait d'écrire une nouvelle procédure qui a fait son travail par rapport à `t à la` place de par rapport à `x`. C'est pourtant une idée terrible : le nombre d'identifiants possibles est assez grande; en plus des 52 lettres majuscules et minuscules de l'alphabet latin, on peut aussi avoir combinaisons de ces lettres avec des chiffres et le trait de soulignement. Tous les noms suivants de les indéterminés pourraient être utilisés dans un contexte mathématique parfaitement légitime :

• `x`

• x0
• x_0
• xi
• x_i

... et ce ne sont que des indéterminés qui commencent par x. Vous pouvez jouer à ce jeu avec y, t, a,... vous
avoir l'idée.
Cette approche n'est donc pas pratique. La bonne façon de le faire est de spécifier l'indéterminé comme un
argument à la procédure elle-même, quelque chose comme tangent_line(f, 1, t) . Mieux encore serait de
faire de l'indéterminé un argument facultatif, avec la valeur par défaut x , de sorte que le client n'ait besoin que
préciser l'indéterminé si nécessaire. De cette façon, les instructions tangent_line(f, 1) et tangent_line(f, 1, x) signifie la même chose. Une approche naïve serait la suivante :

```
sage : def tangent_line(f, a, x=x):
# forme point-pente d'une ligne
b = f(x=a)
df(x) = diff(f,x)
m = df(a)
résultat = m*(x - a) + b
print('La ligne tangente à', f, 'à', x, '=', a, 'est',)
print(résultat, '.')
```

L'instruction x=x peut sembler étrange, mais elle est parfaitement logique : le premier x spécifie le nom du
argument dans tangent_line() , tandis que le second x spécifie sa valeur par défaut. Essentiellement, il indique
que tangent_line() a un argument facultatif nommé x dont la valeur par défaut est x .[4]
Alors, que se passe-t-il lorsque nous l'essayons?

```
sage : tangente_line(t**2, 1, t)
La droite tangente à t^2 à t = 1 est t^2 + 2*t - 2 .
```

PANIQUE!

. . . Eh bien, non, ne paniquez pas. Analysons plutôt ce qui s'est passé et essayons de résoudre le problème. Pour faire ça,
nous introduisons une technique de débogage classique du désespoir, qui porte le nom technique,

imprimer

Qu'est-ce que ça veut dire? Nous imprimons tout ce que Sage calcule dans la procédure et essayons à partir de

là pour voir ce qui n'allait pas. Réécrivez la procédure comme suit :

```
sage : def tangent_line(f, a, x=x):
# forme point-pente d'une ligne
b = f(x=a)
imprimer(b)
df(x) = diff(f,x)
imprimer (df)
m = df(a)
imprimer(m)
résultat = m*(x - a) + b
print('La ligne tangente à', f, 'à', x, '=', a, 'est',)
print(résultat, '.')
```

Vous pouvez voir que nous avons une déclaration imprimée après presque chaque déclaration originale. Nous nous sommes arrêtés
résultat car c'est déjà imprimé sur la ligne suivante. Que se passe-t-il lorsque nous exécutons ceci
version de tangent_line() ?

```
sage : tangente_line(t**2, 1, t)
t^2
t |--> 2*t
2
La droite tangente à t^2 à t = 1 est t^2 + 2*t - 2 .
```

On voit ça:

• df est correctement calculé comme 2t ;

• m est correctement calculé comme 2 ; mais

• b est incorrectement calculé comme t 2 — comme si la substitution n'avait pas eu lieu ! De nouveau!

4 Ce type de construction se produit assez souvent dans le code source de Sage.

FIN DE LA COMMUNICATION DYSFONCTIONNELLE

89

Remarquez la différence subtile cette fois : une substitution a réellement fonctionné. L'autre a fait
ne pas. La différence est que l'une est une fonction mathématique définie dans notre procédure Sage ; les
other est une expression définie en dehors de notre procédure Sage.

La solution apparente est de redéfinir f à l' intérieur de notre procédure, comme une fonction mathématique. Nous
alors s'appuyer exclusivement sur la notation de fonction. Il n'y a aucun danger à cela ; grâce aux règles de
scope, changer la valeur de f à l' intérieur de tangent_line() ne change pas sa valeur d'origine à l'extérieur.

La procédure réussie ajoute une ligne :

```
sage : def tangent_line(f, a, x=x):
# forme point-pente d'une ligne
f(x) = f
b = f(a)
```

```
df(x) = diff(f,x)
m = df(a)
résultat = m*(x - a) + b
print('La ligne tangente à', f, 'à', x, '=', a, 'est',)
print(résultat, '.')
```

Cette première ligne redéfinit l'argument `f` en tant que fonction. Si `f` est une expression, nous avons évité le

DépréciationAvertissement ; s'il s'agit d'une fonction, Sage n'aura aucun mal à se l'attribuer. Essaye le:

```
sage : tangente_line(t**2, 1, t)
La droite tangente à t |--> t^2 à t = 1 est 2*t - 1 .
```

Mettre fin à la communication dysfonctionnelle

Jusqu'à présent, nous avons affiché le résultat d'un calcul à l'aide de la commande `print` . Alors que ce

est utile pour démontrer la commande d' `impression` , ainsi que pour afficher la sortie, c'est un obstacle

à une automatisation efficace. Vous ne voulez pas que votre code attende que vous voyiez le résultat imprimé de

une procédure, puis saisissez-la dans une autre procédure. Plutôt que de vous épuiser ainsi, il serait bien mieux que les procédures communiquent directement. De cette façon, vous pourriez appeler

une procédure dans une autre et évitez-vous complètement. Vous pouvez commencer un calcul, marcher

loin, prenez une tasse de boisson caféinée,[5] puis revenez et constatez que vos procédures sont terminées

leurs tâches et vous attendent joyeusement, queue en l'air.

Portée locale contre portée mondiale. Supposons, par exemple, que vous vouliez écrire une procédure qui trace

une fonction mathématique f (x) et la droite tangente à f en ce point. Vous pourriez, bien sûr, `demandez-` lui de calculer la ligne tangente en copiant et en collant le code dans `tangent_line()` dans le

nouvelle procédure, mais cela gaspille des ressources et rend la nouvelle procédure plus difficile à lire.

Vous pouvez plutôt utiliser la modularité en laissant `tangent_line()` calculer la ligne tangente, et laisser votre procédure se concentrer uniquement sur le tracé de la fonction et de la ligne - mais comment le

la nouvelle procédure utilise le résultat de `tangent_line()` ? D'une part, `tangent_line()` a une variable appelé `result` , alors peut-être pourrions-nous faire quelque chose comme ça?

5 Cela faisait à l'origine référence à une boisson contenant de la caféine en particulier, tout en prenant des photos d'une autre, plus populaire,
car l'un des auteurs est un peu immature. Un deuxième auteur a commenté ZOMGPONIES et par la suite convaincu le premier de le changer pour le discours générique actuel sur les boissons caféinées.

```
sage : def plot_tangent_line(f, a, x=x) :
# utilisez tangent_line() pour trouver la ligne
tangente_line(f, a, x)
tracé(résultat, a-1, b-1)
```

Cela ne fonctionnera pas, et si nous devions l'essayer, nous rencontrerions l'erreur suivante :

```
sage : plot_tangent_line(x^2, 1)
NameError :
le nom global 'résultat' n'est pas défini
```

Cette erreur nous indique que si le résultat existe quelque part, il n'existe pas dans l'environnement « global » de Sage.
ment, nous utilisons donc son nom en vain. Nous avons uniquement défini le résultat dans la procédure tangent_line() , donc il
restait local à cette procédure.
Il y a deux façons de contourner cela. L'un d'eux est de déclarer le résultat en tant que variable globale dans le
procédure tangent_line() utilisant le mot clé global ; après, le résultat serait accessible hors-
à côté aussi :

```
sage : def tangent_line(f, a, x=x):
# forme point-pente d'une ligne
résultat global
f(x) = f
b = f(a)
df(x) = diff(f,x)
m = df(a)
résultat = m*(x - a) + b
```

Cependant, c'est une idée terrible, pour plusieurs raisons. Premièrement, le résultat est à peu près aussi peu informatif qu'un nom
comme tu pourrais donner. Si chaque procédure plaçait son résultat dans une variable nommée result — eh bien, cela pourrait
travail, en fait, mais ce serait plutôt chaotique. En plus, il y a un meilleur moyen.
De retour du chaos. Sage fournit une commande pratique pour cela avec une autre clé-mot, retour . Chaque procédure Sage renvoie une valeur, même si vous omettez ce mot-clé. Vous pouvez
vérifiez cela avec les procédures que nous avons définies, en utilisant une instruction type() :

```
sage : rien = tangent_line(f, 1, t)
La droite tangente à t |--> t^2 à x = 1 est 2*t - 1 .
sage : tapez (rien)
<tapez 'AucunType'>
```

Ce que Sage renvoie dans ces instances est un objet appelé None . Les programmeurs utilisent souvent Aucun
[6] à
indiquer que quelque chose n'a pas été initialisé, mais c'est une discussion différente.
Contrairement à d'autres langages informatiques, Sage ne vous contraint pas à ne renvoyer qu'un seul résultat ; c'est
possible de retourner plusieurs résultats à la fois. Répertoriez-les simplement après le mot-clé return , en séparant

6 Ou l'analogique dans d'autres langages : NULL , NIL ,...

FIN DE LA COMMUNICATION DYSFONCTIONNELLE
91
eux avec des virgules. Par exemple, le programme suivant renverra à la fois la dérivée et la primitive d'une procédure, en supposant que les deux existent :

```
sage : def deriv_and_integ(f, x=x):
# BOOOO-RING
df(x) = diff(f, x)
F(x) = intégrale(f, x)
retour df, F
```

Si cela vous rappelle comment Sage vous permet d'affecter plus d'une variable dans une instruction
(voir p. 30), vous avez raison! Vous pouvez utiliser la procédure ci-dessus de la manière suivante :

```
sage : f(x) = x**2
sage : df, F = deriv_and_integ(f)
sauge : df
x |--> 2*x
sauge : F
x |--> 1/3*x^3
```

Si vous avez de l'expérience avec des langages qui ne le permettent pas, vous comprenez à quel point
et quelque peu peu intuitif, il peut s'agir de renvoyer plusieurs valeurs à la fois.7
Ce que nous voulons faire, c'est modifier la procédure tangent_line() pour renvoyer la ligne, plutôt que
l'imprimer . C'est assez simple :

```
sage : def tangent_line(f, a, x=x):
# forme point-pente d'une ligne
f(x) = f
b = f(a)
df(x) = diff(f,x)
m = df(a)
résultat = m*(x - a) + b
résultat de retour
```

C'est si simple. Que se passe-t-il lorsque vous l'utilisez ?

```
sage : tangente_line(t**2, 1, t)
2*t - 1
```

7 Par exemple, en C++, on peut soit passer des arguments par référence, ce qui donne la signature
void deriv_and_integ(Fonction f, Fonction & df, Fonction & F, Indéterminé x=x);
ou créez une structure ou une classe qui contient des champs pour les deux réponses :
struct DandI_result { Function df; Fonction F; } ;
DandI_result deriv_and_integ(Fonction f, Indéterminé x=x);
mais l'approche Python semble plus élégante.

PSEUDOCODE
92
La sortie peut ne pas sembler aussi informative qu'avant, mais c'est parce que nous l'avons

élaguée à sa
essence. Vous pouvez modifier la déclaration de retour pour inclure les informations textuelles que nous avions auparavant,
mais l'essence nue est beaucoup plus utile. Pourquoi? nous pouvons maintenant effectuer les opérations suivantes :

```
sage: p1 = plot(t**2, 0, 2, color='black', width=2)
sage : par_line = tangent_line(t**2, 1, t)
sage: p2 = plot(par_line, 0, 2, color='red', linestyle='dashed')
sage : p1 + p2
```

Remarquez ce que nous faisons : nous prenons le résultat de tangent_line() , en attribuant sa valeur à une variable
capable nommé par_line , puis en le passant comme argument à plot() . Vous pouvez le faire même si vous
n'envisageait pas de tracer le résultat de tangent_line() ! En effet, vous remarquerez que nous n'avons pas mentionné
la possibilité de tracer la ligne tangente avant ce point. Une grande partie de la puissance de processus
dures réside dans le fait que vous pouvez les utiliser dans des situations inimaginables lorsque les procédures ont été
conçu; les clients bénéficient toujours des avantages d'avoir le code déjà écrit pour eux.

Pseudocode

Les personnes qui veulent vraiment travailler sur ordinateur doivent communiquer leurs idées à chacun
autre. Tout le monde n'utilise pas le même langage de programmation, et par souci d'abstraction et
communication c'est une bonne idée d'éviter de se fier à une langue particulière pour décrire la solution
à un problème. Pour cette raison, les mathématiciens utilisent souvent un pseudo-code pour décrire un programme informatique.
gramme - ou, pour être plus précis, l'algorithme utilisé dans le programme. Un algorithme est une suite de
étapes avec entrée (informations utilisées) et sortie (informations renvoyées) bien définies.
Il existe différentes manières d'exprimer un pseudocode ; quelques exemples apparaissent dans la figure 1. Toi
devrait remarquer plusieurs propriétés qu'ils partagent tous :

• Tous spécifient le nom de l'algorithme, les informations requises (« entrée ») et les promesses
résultat (« sortie »).
• Tous utilisent une forme d'indentation pour montrer comment un code dépend d'autres.
• Tous s'appuient sur un anglais « simple » et des symboles mathématiques pour communiquer ce qui est
être fait.
• Aucun d'entre eux ne ressemble à un code informatique dans un langage informatique

particulier.
Ce texte vous fournira un pseudo-code à implémenter et vous demandera d'écrire votre propre pseudocode à l'occasion. Nous devons adopter une norme, et nous ferons ce qui suit :
• Les identifiants de variables ou les noms d'algorithmes apparaissent en italique.
• Les mots-clés pour les structures de contrôle et les informations fondamentales apparaîtront en caractères gras.
À présent, nous introduisons les mots-clés suivants :
– algorithme apparaît au début, et déclare le nom d'un algorithme

---

Apparaît dans [6]
Apparaît dans [3]
Apparaît dans [5]
Apparaît dans [4]
F IGURE 1. Pseudocode provenant de diverses publications mathématiques
– les entrées apparaissent immédiatement après l'algorithme, et en dessous se trouve une liste en retrait de
les informations requises, ainsi que les ensembles dont ils sont issus
– les sorties apparaissent immédiatement en dessous de la liste des entrées, et en dessous il y a une entrée
liste bosselée des informations promises, ainsi que leur lien avec les intrants
– do apparaît immédiatement en dessous de la liste des sorties, et en dessous il y a un retrait
liste d'instructions
– return apparaît comme la dernière instruction dans les instructions[8]
Nous pouvons maintenant décrire notre procédure `tangent_line()` en pseudocode comme suit :
8 Il est courant de voir des retours placés à d'autres endroits dans la liste d'instructions, mais nous adopterons la convention
tion que return est toujours la dernière instruction. Bien qu'il soit parfois gênant d'organiser le code autour de ce convention, cela peut aider à la fois au débogage et à la lisibilité, ce qui est important pour ceux qui apprennent pour la première fois.

---

algorithme tangente_line
contributions
• un
• x, un indéterminé
• f , une fonction en x dérivable en x = a
les sorties
• la droite tangente à f en x = a
fais

soit m = f (a)
soit b = f (a)
renvoie m (x −a)+ b
Les observations suivantes de ce code sont dans l'ordre.
• Nous effectuons des affectations avec l'énoncé mathématique traditionnel « let ».[9]
• Non seulement cela ne ressemble pas beaucoup à notre code Sage, mais nous omettons certaines tâches spécifiques à Sage,
comme affecter f (x) à une variable.[dix]
• Les entrées et certaines commandes sont répertoriées dans un ordre différent.
Au fur et à mesure que nous parcourrons le texte, nous ajouterons des mots-clés de pseudocode supplémentaires et expliquerons comment
accomplir des tâches supplémentaires.
En pratique, nous formulons presque toujours du pseudocode avant de l'implémenter dans le code. On a
inversé cette pratique ici, principalement pour se mouiller les mains dès le début, mais un aspect important
de travailler avec un ordinateur est de réfléchir à la façon dont vous résolvez le problème avant d'écrire réellement
la solution - et, vraiment, un programme n'est rien de plus que la rédaction d'une solution à la problème : « Comment puis-je faire ceci, cela et l'autre, étant donné tel et tel et tel et tel ? » Si vous avez
eu un autre cours de programmation auparavant, vous avez peut-être été élevé dans la mauvaise tradition de
ne pas penser à quoi faire avant de le faire. Essayez de résister à cela ; nous vous aiderons dans les exercices, en
exigeant que vous formuliez un pseudo-code ainsi que l'écriture d'un programme.
Script
De nombreux mathématiciens ont un ensemble de tâches qu'ils doivent répéter chaque fois qu'ils travaillent sur
quelque chose. L'écriture de procédures Sage peut aider à simplifier cela, mais si vous les saisissez dans un
session, ils s'appliquent uniquement à cette session, soit sur la ligne de commande, soit dans une feuille de calcul. Si tu
quitter la session ou en ouvrir une autre, Sage oublie ces procédures et vous devez redéfinir eux. Il n'est pas très pratique de copier et coller ou (pire) de retaper ces procédures chaque fois que
vous démarrez une nouvelle session, il est donc pratique de conserver une bibliothèque de fichiers qui les enregistrent, et que nous
peut charger et utiliser à volonté.
C'est là qu'interviennent les scripts Sage. Un « script » est essentiellement une séquence d'instructions Sage enregistrées
à un fichier. Il est pratique de charger le fichier dans Sage, évitant ainsi les tracas de retaper

ev-
tout. Le format d'un script Sage est identique à celui que nous écririons dans une cellule Sage : il peut être
une séquence d'instructions simples, mais il peut également définir une ou plusieurs procédures. En effet, une grande

9 Comme vous pouvez le constater à partir des exemples de la figure 1, il y a un peu de variété ici. Où nous avons "let m = f (a),"
certains écriraient « m := f (a) » et certains langages informatiques ont adopté cette convention de sorte que l'affectation a
la forme `m := df(a)` (par exemple, Pascal, Eiffel, Maple). Certains écriront même simplement « m = f (a) ».
10 Techniquement, nous n'avons pas non plus à le faire dans Sage, mais cela rend les choses beaucoup plus faciles.

quantité de Sage se compose de scripts Sage que les gens ont écrits en raison d'un besoin et, en raison de l'utilité de la tâche, le script a ensuite fait son chemin dans Sage lui-même.
Pour montrer comment créer un script Sage, nous rappelons notre programme sur le calcul de la droite tangente à
une courbe :

```
def tangente_line(f, a, x=x):
# forme point-pente d'une ligne
f(x) = f
b = f(a)
df(x) = diff(f,x)
m = df(a)
résultat = m*(x - a) + b
résultat de retour
```

Pour créer un script, effectuez l'une des opérations suivantes.
• Si vous exécutez Sage en tant que feuille de calcul gérée par un serveur indépendant, vous devrez parler
à votre instructeur ou à l'administrateur du serveur. Ce n'est pas difficile à faire, mais c'est un peu compliqué, alors nous leur laissons le soin.
• Si vous exécutez Sage à partir de la ligne de commande, ouvrez un éditeur de texte sur votre ordinateur, tapez
le programme ci-dessus dans l'éditeur, puis enregistrez le fichier sous le nom `calc_utils.sage` . Nous
recommande de l'enregistrer dans un répertoire spécial de votre répertoire personnel intitulé `SageScripts` ,
mais vous devriez choisir un endroit qui sera à la fois facile à retenir et facile d'accès.
• Si vous exécutez Sage via le serveur CoCalc, ouvrez un projet, puis sélectionnez dans le menu du haut
`Nouveau` .
Dans la zone de texte sous les instructions pour « Nommez votre fichier, dossier ou collez-le dans un lien »,
tapez le nom `calc_utils.sage` . Cliquez ensuite sur "Fichier" (vers la droite) et Sage

faire apparaître un nouvel onglet avec ce nom.
Allez-y et tapez la procédure. Si vous trouvez le texte trop petit, cliquez sur le "plus grand" A sous Paramètres dans la capture d'écran ci-dessous. Cela agrandira le texte.[11]

11 L'un des auteurs clique régulièrement dessus six à huit fois, et ce n'est pas parce qu'il est vieux. - Enfin, pas ça vieille.

SCRIPT
96
Sauvegarde le. (Il devrait en fait enregistrer automatiquement tous les quelques instants, mais cela ne fait jamais de mal
pour donner au bouton un clic supplémentaire.)
Une fois que vous avez écrit et enregistré votre script, vous pouvez passer à une session Sage (soit au
ligne de commande ou dans la feuille de calcul) et, à ce stade, tapez ce qui suit (ne tapez pas avant
vous lisez ce qui apparaît ci-dessous):

```
sage : %attacher /chemin/vers/fichier/calc_utils.sage
```

Avant de faire cela, nous voulons décrire deux erreurs potentielles, ainsi qu'indiquer une conven-
vous devez être au courant.
Tout d'abord, le congrès. Nous avons écrit /chemin/vers/fichier/ comme "modèle" à remplir. C'est essentiellement un champ à remplir utilisé couramment pour indiquer que vous devez remplir le bon
valeur:
• Si vous utilisez Sage à partir d'une feuille de calcul d'un serveur indépendant, vous devez demander à votre
l'instructeur ou l'administrateur du serveur.
• Si vous utilisez Sage à partir de la ligne de commande, vous devriez essayer '~/SageScripts' , car cela
correspond au répertoire que nous avons suggéré plus tôt.[12]
• Si vous utilisez le serveur CoCalc et que vous avez suivi nos instructions ci-dessus, vous n'avez pas besoin
tapez /chemin/vers/fichier du tout ! Sinon, si vous avez créé un répertoire spécial pour garder votre Sage
scripts, vous devez taper le chemin d'accès à ce répertoire.
Une fois que nous avons dépassé cette convention, nous passons aux erreurs possibles.
Si vous êtes dans le mauvais répertoire, ou si vous avez mal tapé le nom de fichier, ou si vous n'avez pas suivi notre
conseil ci-dessus d'attendre pour le saisir, vous verrez probablement une erreur comme suit :

```
sage : %attacher /chemin/vers/fichier/util_calc.sage
IOError :
n'a pas trouvé le fichier u'calc_util.sage' à joindre
```

Cela indique qu'il y a quelque chose qui ne va pas dans ce que vous avez rempli pour

/chemin/vers/fichier/ ou dans le fichier-
nom lui-même. Dans cet exemple particulier, les deux sont faux : nous n'avons pas réellement modifié /chemin/vers/fichier/
au bon chemin, et nous avons laissé le s du nom de fichier calc_utils.sage .
Maintenant pour la deuxième erreur. Nous tapons parfois mal les choses. C'est bien; ça fait partie d'être
Humain; pas besoin de s'énerver à ce sujet. Après tout, l'ordinateur s'énervera sur son propre. Il est possible de taper quelque chose de mal, et lorsque cela se produit, vous obtiendrez une erreur de syntaxe.
Un exemple courant pour les débutants sera probablement celui-ci :

12 Le caractère tilde (~) est un moyen standard de référencer le répertoire personnel d'un utilisateur. Nous avions proposé de créer un
répertoire nommé SageScripts dans votre répertoire personnel, cela devrait donc faire l'affaire.

```
sage : %attacher /chemin/vers/fichier/calc_utils.sage
Erreur d'indentation :
unindent ne correspond à aucune indentation externe
niveau
```

Dans ce cas, faites attention à la ligne où Sage s'est plaint et assurez-vous que les lignes d'indentation
avec un bloc de code précédent. Une erreur similaire sera celle-ci :

```
sage : %attacher /chemin/vers/fichier/calc_utils.sage
b = f(a)
^
Erreur de syntaxe:
Syntaxe invalide
```

Cela ne devrait pas vous arriver, mais au cas où cela arriverait, et tout semble parfaitement parfait : le
le problème est que vous avez copié et collé du texte. Les apparences dans un éditeur de texte peuvent être trompeuses, et
lorsque vous copiez d'un fichier à un autre, vous pouvez récupérer des caractères de formatage cachés, ou certains-
chose de semblable. La solution dans ce cas est d'éliminer le caractère de formatage caché en supprimant
ces espaces, puis les retaper.
Avec ces clarifications à l'écart, essayez maintenant de charger le fichier. Si tout se passe bien, Sage
garder le silence. Vous devriez alors pouvoir exécuter votre procédure de la manière habituelle :

```
sage : %attacher /chemin/vers/fichier/calc_utils.sage
sage : tangente_line(x**2, 2)
4*x - 4
```

Une fois que vous avez attaché avec succès calc_utils.sage à une session Sage, vous pouvez y

apporter des modifications
et Sage intégrera automatiquement les modifications, laissant un message du type :
rechargement du fichier joint calc_utils.sage modifié à 08:39:00
###
Feuilles de travail interactives
Si vous travaillez dans une feuille de calcul Sage, vous avez accès à une fonctionnalité qui vous permet de créer facilement un
démonstration manipulée. Nous appelons cette fonctionnalité des feuilles de travail interactives. Une feuille de travail interactive
a une ou plusieurs procédures interactives qui permettent à l'utilisateur de manipuler leurs arguments dans un
manière « pratique » à travers les objets d'interface tels que les zones de texte, les curseurs, etc. A part le
procédure elle-même, il y a deux étapes clés pour créer une feuille de travail interactive :
• Sur la ligne précédant la procédure, mais dans la même cellule, tapez @interact .
• Décidez lequel des objets d'interface suivants vous voulez et fournissez la procédure avec arguments « facultatifs » dont les valeurs par défaut sont initialisées à ces objets :
– une zone de saisie (pour fournir une valeur)
– un curseur (pour sélectionner une valeur dans une plage)
– une case à cocher (pour activer ou désactiver certaines propriétés)
– un sélecteur (pour sélectionner une propriété parmi plusieurs)
– un sélecteur de couleur (pour... eh bien, j'espère que la raison en est évidente)

---

zone de saisie()
une zone permettant à l'utilisateur de saisir des valeurs, généralement des nombres ; spécifique les options incluent
• largeur= largeur de la boîte (104)
curseur (a , b)
une ligne avec un bouton que l'utilisateur peut faire glisser à gauche et à droite, allant
de a à b avec des valeurs intermédiaires
curseur(a , b , par défaut= c)
similaire à slider(a , b) , mais l'utilisateur ne peut glisser que dans-
créments de c
curseur (L)
similaire à slider(a , b) , mais l'utilisateur ne peut sélectionner que op-
dans la collection L (on ne vous dit toujours pas ce
une collection est, mais vous pouvez toujours utiliser un tuple, comme avant)
range_slider(a , b)
similaire à slider(a , b) , mais l'utilisateur sélectionne un sous-intervalle
[c,d] de [a,b]

range_slider(a , b ,
par défaut=(c , d))
similaire à range_slider(a , b) , mais le sous-intervalle par défaut
est [c,d]
case à cocher()
une case sur laquelle l'utilisateur peut cliquer ou désactiver
sélecteur()
un menu déroulant ou des options spécifiques à la barre de boutons incluent
• *values*= collection de valeurs parmi lesquelles choisir
• boutons= s'il faut utiliser les boutons (*False*)
• nrows= nombre de lignes si vous utilisez les boutons (1)
• ncols= nombre de colonnes si utilisation des boutons (de-
dépend du nombre)
• width= largeur du bouton, si les boutons (dépend de
valeur)
Couleur (nom)
un bouton qui affiche un sélecteur de couleur ; la couleur par défaut est la
un fourni par son nom, tel que « rouge »
Couleur(r , g , b)
Identique à Color(name) , mais la couleur par défaut est le rouge-
combinaison vert-bleu donnée par r, g, b
F IGURE 2. Modèles d'objets d'interface dans une procédure interactive
étiquette= chaîne
une étiquette, placée à gauche ; il peut inclure L A TEX entre les signes dollar
default= value la valeur initiale de l'objet
F IGURE 3. Options communes à tous les objets d'interface dans une procédure interactive
sauf Couleur()
Vous trouverez des modèles pour ces objets dans la figure 2, et quelques options communes pour tous les éléments
dans la figure 3. Cette liste n'est pas exhaustive, car nous avons omis certaines options que nous pensons que vous
sont moins susceptibles de trouver utiles. D'autres objets d'interface peuvent également être disponibles dont nous ne sommes pas conscients
(ils ont peut-être été ajoutés après que nous ayons écrit ceci).

F IGURE 4. Un écrasement malheureux
Les curseurs et les sélecteurs sont similaires en ce sens qu'ils peuvent vous permettre de choisir parmi une petite plage de valeurs ;
par exemple, le curseur((0, 0,25, 0,5, 0,75, 1)) et le sélecteur((0, 0,25, 0,5, 0,75, 1)) tous les deux
vous permet de choisir dans la même liste. Dans ce cas, cependant, un curseur serait le

meilleur choix, car
vous pouvez ordonner les valeurs de gauche à droite, et un curseur communique mieux ce sens. Si la
les valeurs ne s'intègrent pas bien dans un paradigme de gauche à droite, vous feriez mieux d'utiliser un sélecteur. Aussi, les curseurs
n'étiquetez pas les valeurs individuelles, donc si vous pensez qu'il est particulièrement important que chaque valeur soit étiquetée,
il est préférable d'utiliser un sélecteur même si les valeurs correspondent au paradigme de gauche à droite.[13]
Nous illustrons cela sur deux exemples. La première est assez simple, simplement pour illustrer le principe :
il permet à l'utilisateur de taper une fonction f , et trace à la fois f et sa dérivée sur un intervalle [a, b].

```
sage : @interact
def plot_f_and_df(f=input_box(default=x**3,label='$f$'),\
a=input_box(default=-5,label='$a$'), \
b=input_box(default=5,label='$b$'), \
w=curseur(-19,19,1,défaut=0, \
label='facteur d'écrasement')):
p = plot(f, a, b, color='black')
df = diff(f)
p = p + plot(df, a, b, color='red')
show(p, aspect_ratio=(20-w)/20)
```

Si vous saisissez correctement ceci dans une feuille de calcul Sage, vous devriez voir quelque chose qui ressemble à la figure 4. Toi
pouvez voir que les zones de saisie ont des étiquettes bien formatées. (Comme vous l'avez peut-être deviné dans notre
l'utilisation de signes dollar, c'est s'appuyer sur L A TEX.) Malheureusement, ce n'est pas une intrigue particulièrement claire ; son
un peu trop écrasé horizontalement. Nous pouvons résoudre ce problème en déplaçant le facteur de courge vers la droite ; si tu glisses

13 Bien sûr, quelqu'un trouvera une exception à chacune de ces directives ; ce ne sont que des recommandations.

le bouton du curseur « facteur de courge » complètement à droite, vous obtenez une image plutôt agréable le
moment où tu lâches prise :
Vous pouvez également modifier f , a et b, mais ce sont des zones de saisie, donc plutôt que de faire glisser quoi que ce soit, vous avez
pour changer le texte. En tant que $\prec$ b, vous devriez obtenir un graphique d'apparence raisonnable au moment où quelque chose
changements. Essayez une autre fonction, telle que sin x, avec a = 0 et b = 2 π , et le graphique devrait changer

à cela (assurez-vous de remettre également le facteur de courge à 0) :
Vous pouvez bien sûr appeler des procédures que vous avez définies depuis une procédure interactive. Faisons
écrire une procédure interactive qui trace f et sa dérivée sur [a, b], ainsi qu'une ligne tangente à f en un point c quelque part entre a et b. Nous pouvons appeler la procédure tangent_line() que nous
déjà écrit pour générer la ligne tangente. Pendant que nous y sommes, nous allons passer les valeurs par défaut à sin(x)
sur [0,2 π].

```
sage : @interact
def plot_f_and_df(f=input_box(default=sin(x),label='$f$'), \
subint=range_slider(-10,10,default=(0,2*pi), \
label='$(a,b)$'), \
t=slider(0,1,default=0.375,label='$t$'), \
w=slider(-19,19,1,default=0,label='squash factor')):
(a,b) = sousnt
p = plot(f, a, b, color='black')
df = diff(f)
p = p + plot(df, a, b, color='red')
# c nous indique jusqu'où aller le long de l'intervalle [a,b]
# 0 signifie un; 1 signifie b; valeurs intermédiaires proportionnelles
c = a + t*(ba)
p = p + plot(tangent_line(f, c), a, b)
show(p, aspect_ratio=(20-w)/20)
```

Lorsque vous essayez ceci dans Sage, vous devriez voir ce qui suit :

Vous avez peut-être remarqué que nous n'avons pas demandé à l'utilisateur de spécifier directement le point c. Au lieu de cela, elle
spécifie c indirectement, en choisissant une valeur t de 0 à 1. Le code calcule alors une valeur de c
entre a et b en utilisant le paramétrage suivant :
c = a + t (b − a).
Ce paramétrage est un outil extrêmement utile, en partie parce qu'il est relativement simple :
• lorsque t = 0, alors c = a +0 = a ;
• lorsque t = 1, alors c = a +(b −a) = b ;
• lorsque t = .5, alors c = a +.5(b −a) = .5a +.5b = $a+b/2$, à mi-chemin entre a et b ;
• et ainsi de suite.
Expérimentez avec le curseur pour voir comment différentes valeurs de t permettent naturellement à l'utilisateur de sélectionner un bon point
entre A et B.
Nous n'avons pas montré comment fonctionnent tous les objets d'interface, mais nous avons montré ceux que nous pensons
sont les plus importants, ainsi qu'une manière élégante de demander à l'utilisateur de

sélectionner un point dans un intervalle sans
avoir à vérifier si le point se situe réellement dans l'intervalle. Nous vous encourageons à expérimenter
avec les autres objets, notamment en sélectionnant une couleur.
Des exercices
Vrai faux. Si la déclaration est fausse, remplacez-la par une déclaration vraie.

DES EXERCICES
102

1. Les procédures nous permettent de réutiliser des groupes communs de commandes.
2. Si un argument contient la valeur d'une variable en dehors de la procédure, changer l'argument
la valeur de l'élément change également la valeur de cette variable.
3. Sage vous permet de spécifier des arguments facultatifs.
4. Si une procédure Sage ne renvoie pas explicitement une valeur, elle renvoie *None* .
5. Lorsque nous passons des indéterminés comme arguments aux procédures Sage, nous devons redéfinir tout calcul mathématique.
fonctions ematiques qui en dépendent.
6. Les procédures définies par l'utilisateur ne peuvent pas utiliser les procédures Sage.
7. Personne n'utilise de pseudocode dans la vraie vie.
8. Le pseudocode communique des idées en utilisant un « anglais simple » et des symboles mathématiques, plutôt
que les symboles et les formats d'une langue particulière.
9. La meilleure façon de réutiliser un groupe de procédures Sage dans différentes sessions est de copier-coller
le code.
10. La commande %attach permet à Sage de lire et d'exécuter un script, ainsi que de recharger automatiquement
quand vous le changez.
11. Vous pouvez utiliser des procédures interactives à partir de la ligne de commande Sage.
12. Si l'utilisateur doit choisir parmi 100 valeurs équidistantes sur une droite numérique, il est préférable d'utiliser un
glissière.
13. Si l'utilisateur doit sélectionner une fonction dans l'ensemble $\{\sin x, \cos x, \ln x, e^x\}$, il est préférable d'utiliser un sélecteur.
14. Pour trouver le nombre un quart de la distance de 2 à 10, substituez t = 0,25 dans le expression 2+8t.
15. Un inconvénient des procédures interactives est que vous ne pouvez pas diviser leurs tâches en tâches par-
formé par d'autres procédures.
Choix multiple.

1. On démarre la définition d'une procédure dans Sage en utilisant :
A. algorithme
B. son type
C. son nom
D. déf
2. Le format d'un argument facultatif est :
A. de ne pas l'énumérer
B. faire précéder son identifiant du mot-clé facultatif
C. faire précéder le nom de la procédure par le mot-clé def
D. à placer = après son identifiant, suivi d'une valeur par défaut
3. Lequel des énoncés suivants n'est pas une description précise d'un argument ?
A. un remplaçant pour une variable extérieure à la procédure, dont la valeur peut être modifiée par la procédure
B. une copie de certaines données en dehors de la procédure
C. une variable dont vous ne connaissez peut-être pas la valeur à l'avance
D. informations dont la procédure peut avoir besoin ou non pour exécuter ses tâches
4. La meilleure façon de rapporter le résultat d'une procédure au client est de
A. lister le résultat dans une déclaration de retour
B. attribuer la valeur à l'un des arguments
C. l'affecter à une variable globale appelée result
D. attribuer une valeur au nom de la procédure
5. Une procédure qui n'a pas d' instruction return :
A. soulève une erreur

B. renvoie l'ensemble vide
C. renvoie *Aucun*
D. ne fait rien de spécial
6. Lorsqu'une liste de commandes est soumise à un mot-clé qui démarre une structure de contrôle, tel que def ,
tu devrais:
A. indenter les commandes en question
B. ajouter deux points à la fin de l'instruction de la structure de contrôle
C. inverser l'indentation lorsque la liste est terminée
Tout ce qui précède
7. Laquelle des propriétés suivantes notre norme de pseudocode partage-t-elle avec le code Sage ?
A. Les déclarations soumises aux structures de contrôle sont en retrait.
B. Nous pouvons utiliser def à la fois dans le pseudocode et dans le code Sage.
C. Nous spécifions les types d'entrées dans Sage, tout comme nous spécifions de quel

ensemble provient une entrée dans
pseudo-code.
Tout ce qui précède
8. La meilleure façon dans Sage de se substituer à un indéterminé qui a été passé comme argument est
par:
A. substitution d'appel de fonction
B. substitution de dictionnaire
C. attribution de mots clés
D. substitution d'appel de fonction après redéfinition de la fonction
9. Lequel des identifiants suivants ne pouvons-nous pas attribuer de valeurs ?
A. un identifiant valide
B. un mot-clé
C. une constante
D. le nom d'une procédure déjà définie
10. Lorsque l'utilisateur modifie un paramètre dans une procédure interactive, le seul effet garanti est de
changer quelle partie de la procédure?
A. une variable globale
B. une variable locale
C. un argument de procédure
D. le nom de la procédure
Réponse courte.
1. Expliquez la modularité et pourquoi c'est une bonne pratique lors de l'écriture de programmes.
2. La modularité et l'arithmétique modulaire sont-elles la même chose ?
Programmation.
1. Sage n'ajoute pas automatiquement de pointes de flèches à une courbe. Puisque la dérivée nous indique la pente
de la courbe en un point donné, nous pouvons déterminer dans quelles directions les flèches à la fin devraient
point en calculant les dérivées aux extrémités et en y mettant des flèches. Ecrire un procedure arrowplot() qui prend comme argument une fonction f , deux extrémités a et b, et une « flèche
length" Δ x, puis trace f sur [a, b], en ajoutant des pointes de flèche en utilisant le principe que (a, f (a)) et
(a − Δ x, f (a)− f (a) Δ x) nous donnerait deux points sur une droite tangente à f en a (et qui spécifierait une flèche appropriée). (Nous vous laissons modifier cette définition pour une pointe de flèche
à b.)

2. Écrivez un pseudocode pour une procédure qui calcule et renvoie la ligne normale à f à x = c.
Puis:
(a) Implémentez cela dans le code Sage.
(b) Choisissez n'importe quelle fonction transcendantale f , et n'importe quel point c où f est transcendant.
(c) Utilisez ce code et la procédure `tangent_line()` pour tracer à la fois les lignes normales et tangentes
à f en x = c.
(d) Écrivez une procédure interactive pour tracer le graphique d'une fonction, une ligne tangente à un point
x = c, et la droite normale en x = c. Mis à part les éléments d'interface évidents nécessaires, inclure au moins un curseur pour contrôler le rapport hauteur/largeur du graphique résultant.
Votre implémentation échouera parfois ; par exemple, si vous utilisez $f = (x-1)^2 + 2$ et c = 1, vous devriez rencontrer une *ZeroDivisionError* .

PAS DE PANIQUE !

Vous devriez en fait vous attendre à cette erreur, car le calcul d'une perpendiculaire nécessite une réciproque,
ce qui signifie division, ce qui ouvre la possibilité de division par zéro. Nous abordons ce problème
au chapitre 7 sur la prise de décision. Assurez-vous que votre code fonctionne avec des fonctions et des points qui
ne vous comportez pas mal.
3. Écrivez un pseudocode pour une procédure nommée valeur_avg dont les entrées sont une fonction f , et indé-
terminer x, et les extrémités a et b d'un intervalle I. La procédure renvoie la moyenne
valeur de f sur l'intervalle. (Vous devrez peut-être revoir certains calculs pour résoudre ce problème.)
4. Définissons l'opération $a * b = ab + (a + 2b + 1)$.
(a) Écrivez un pseudocode pour une procédure nommée `star()` qui accepte deux entiers a et b et renvoie a b.
(b) Implémentez ce pseudocode en tant que procédure Sage.
(c) Testez votre procédure sur plusieurs valeurs différentes de a et b.
(d) Existe-t-il une valeur de a telle que a b = b, quelle que soit la valeur de b ?
5. Ecrivez une procédure Sage nommée `dotted_line_segment()` qui accepte sept arguments nommés `x1` ,
`y1` , `x2` , `y2` , `couleur` , `taille des points` et `couleur des points` . Les valeurs par défaut des couleurs doivent être `'noir'` ,
et la couleur par défaut de la `taille` des `points` doit être 60. La procédure renvoie la somme de :

• un segment de droite qui relie ces points et dont la couleur est couleur , et
• deux points, tous deux de couleur pointcolor et de taille pointsize , dont les emplacements sont (x_1 ,y_1)
et (x_2 ,y_2).
(a) Utilisez cette procédure pour tracer un segment de droite reliant les points (0,0) et (1,4) dont
la couleur et la couleur du point sont toutes deux noires.
(b) Utilisez cette procédure pour tracer un segment de ligne noire reliant les points rouges (0,6) et (2,0).
Vous n'avez pas besoin d'écrire de pseudocode pour cette procédure.
6. Implémentez le pseudocode suivant dans Sage.

DES EXERCICES
105
algorithme Taylor_Truncated4
contributions
• un
• x, un indéterminé
• f , une fonction en x intégrable en x = a
les sorties
• la série de Taylor tronquée pour f (x) autour de x = a
fais
soit résultat = f (a)
ajouter $f(a)\cdot(x-a)$ au résultat[une]
ajouter $f(a)\cdot(x-a)^2/2$ au résultat
ajouter $f(a)\cdot(x-a)^3/6$ au résultat
ajouter f
$^{(4)}(a)\cdot(x-a)^4/24$ au résultat
résultat de retour
[a] Voici un indice sur celui-ci. Pour "ajouter ... au résultat", utilisez la construction result = result + ...
Vérifiez-le en trouvant la série de Taylor tronquée pour e^x à x = 1, et en comparant la série tronquée
valeur de la série à x = 1,1 avec la valeur de $e^{1,1}$.
7. Expliquez comment le « facteur d'écrasement » que nous avons utilisé dans nos procédures interactives a été utilisé pour définir le
rapport hauteur/largeur de l'intrigue. Votre explication doit expliquer pourquoi la valeur la plus à gauche écrase le
graphique le plus à gauche possible, pourquoi la valeur la plus à droite étire le graphique le plus à droite possible,
et pourquoi la valeur intermédiaire se situe quelque part entre les deux.

CHAPITRE 5

Se répéter définitivement avec les collections

Il existe un certain nombre de cas où vous souhaitez répéter une tâche plusieurs fois. Un exemple classique
vient des équations différentielles; supposons que nous sachions
$dy/dx = \text{siny} + 2\cos x$,
c'est-à-dire qu'en tout point (x,y), la valeur de y change de siny + 2cos x. C'est difficile et souvent
impossible, de trouver la formule exacte de y en fonction de y, il est donc nécessaire d'approximer le « futur »
valeurs de y à partir d'un point de départ (x,y).[1] Dans ce cas, vous décidez généralement de faire un certain nombre de
petits « pas en avant » avec x et réévaluez y à chaque point, en traçant le comportement résultant.
Dans ce cas, supposons que nous commençons à (0,0); alors y (0,0) = 2, ce qui suggère que la fonction veut
avancer le long d'une droite de pente 2. Après tout, vous calculez la dérivée, qui est
la pente de la ligne tangente, qui va dans la même direction que la courbe en ce point. Cette
La technique de déplacement le long de la ligne tangente est connue sous le nom de méthode d'Euler.
Cependant, la courbe ne veut aller dans la même direction que sa ligne tangente que pendant un instant ;
dès que nous descendons du point (0,0), la dérivée change très légèrement et la ligne tangente
n'est plus valable. Pour éviter de faire trop d'erreurs, nous faisons un petit pas, disons $\Delta x = 1/10$, le long
cette ligne, qui nous amène au point (.1,.2). Ici, y (.1,.2) 2.1, nous sortons donc le long d'une ligne de
pente 2.1, ce qui nous amène au point (.2,.41). Ici, y (.2,.41) 2.0, ce qui nous amène à (.3,.61).
Ici, y (.3,.61) 1.9, ce qui nous amène à (.4,.8). Ici, y (.4,.8) 1.8, ce qui nous amène à (.5,.98).
Ici, y (.5,.98) 1.6, ce qui nous amène à (.6,1.14). Allez assez loin, et vous tracerez une image quelque chose comme ça:
Pour générer cette courbe, on a utilisé une taille de pas de $\Delta x = 1/10$. Répétez le processus à partir de (0,0) avec un
taille de pas plus petite, disons $\Delta x = 1/100$, et vous obtiendrez un résultat légèrement différent :

[1] Par exemple, c'est ainsi que fonctionnent les prévisions météorologiques.

Cela reflète le fait que notre approximation incorpore une certaine erreur. Pourtant, ce n'est

pas si mal; les
les approximations sont assez proches. Là encore, nous avons obtenu la courbe bleue en utilisant 80 points, et le
courbe rouge utilisant 800 points. Vous ne voulez pas le faire à la main, n'est-ce pas ?
Lorsqu'une tâche (ou un ensemble de tâches) doit être répétée plus d'une fois sur le résultat du précédent
notre application, nous appelons cette itération de répétition. L'itération apparaît à plusieurs reprises dans le calcul
mathématiques, les langages de programmation offrent donc généralement une structure de contrôle appelée boucle. ONU-
comme la méthode d'Euler, on ne sait pas toujours à l'avance combien de fois la boucle doit se répéter, donc
de nombreuses boucles sont indéfinies. Néanmoins, il arrive très souvent que l'on puisse déterminer l'exacte
nombre de fois qu'une tâche doit se répéter depuis le début ; nous appelons cela une boucle définie. Ce chapitre
introduit des boucles définies; nous reportons les boucles indéfinies pour le chapitre 8.
Comment faire répéter un ordinateur un nombre fixe de fois ?
Pseudocode. Nous pouvons décrire la méthode d'Euler en pseudocode comme suit.
algorithme Eulers_method
contributions
• df , la dérivée d'une fonction
• (x_0, y_0), valeurs initiales de x et y
• Δx, taille de pas
• n, nombre de pas à faire
les sorties
• approximation $(x_0 + n \Delta x, f(x_0 + n \Delta x))$
fais
soit $a = x_0$, $b = y_0$
répéter n fois
ajouter $\Delta x \cdot df(a, b)$ à b
ajouter Δx à un
retour (a, b)
On utilise repeat en pseudocode pour indiquer qu'un ensemble de tâches doit être répété un certain nombre de
fois, et indenter les tâches à répéter.

Code sage. Comme avec la plupart des langages informatiques, Sage n'a pas de mot-clé nommé repeat .2 pour

boucles définies, Sage utilise un mot-clé plus général, pour . Quand tu sais que tu dois répéter une tâche n fois, la construction est assez simple :

```
pour la variable dans la plage (n):
```

Pour variable vous choisissez un identifiant qui contiendra le numéro de la boucle à chaque passage
à travers.

Nous pouvons maintenant implémenter la méthode d'Euler dans Sage :

```
sage : def eulers_method(df, x0, y0, Delta_x, n):
# point de départ
a, b = x0, y0
# calcule les lignes tangentes et avance
pour i dans la plage (n):
b = Delta_x * df(a, b) + b
a = Delta_x + a
retourner a, b
```

Cela semble assez simple. Essayons :

```
sage : df(x,y) = sin(y) + 2*cos(x)
sage : eulers_method(df, 0, 0, 1/10, 80)
```

Si vous essayez cela, vous remarquerez qu'il faut énormément de temps pour s'arranger, certainement plus que
quelques secondes. Pour voir pourquoi cela se produit, interrompez le calcul (appuyez sur le bouton « Stop » dans le
cloud, cliquez sur le menu "Action" et cliquez sur "Interrompre" sur un serveur indépendant, ou maintenez Ctrl et
appuyez sur C sur la ligne de commande) et exécutez à nouveau le code avec une valeur plus petite de n :

```
sage : eulers_method(df, 0, 0, 1/10, 10)
(1, 1/5*cos(9/10) + 1/5*cos(4/5) + 1/5*cos(7/10) + 1/5*cos(3/5) +
1/5*cos(1/2) ...
```

(Les ellipses à la fin indiquent qu'il y a beaucoup plus après cela. Beaucoup plus !)

Vous voyez ce qui se passe ? Sage calcule des valeurs exactes ; et la valeur exacte de cette nombre devient de plus en plus compliqué à chaque itération. Vous pouvez modifier le code pour
simplifier b après chaque calcul, mais cela n'aide pas vraiment. Cela illustre simplement un inconvénient de
calcul symbolique : pour obtenir des valeurs « exactes », vous sacrifiez du temps. Mais il n'y a pas besoin de sacrifier
c'est ici! Après tout, nous nous rapprochons de la valeur de toute façon. Dans ce cas, passons au flottant-
valeurs en points, et voyez si cela accélère les choses. Remplaçons 1/10 par .1 , et voyons comment cela se passe
en dehors.

2 Notre utilisation de la répétition est destinée à illustrer comment le pseudocode vise la clarté de la communication, plutôt que
mimétisme d'un langage particulier, et il n'est pas rare de voir la répétition utilisée en pseudocode. Cela dit, certains pro-

les langages de grammaire comportent une structure de répétition pour les boucles.


```
sage : eulers_method(df, 0, 0, .1, 80)
(7.999999999999999, 4.340418570291038)
```

Cela revient à (8,4.34). Non seulement nous avons obtenu une réponse rapidement, mais elle a été quasi instantanée.

nouveau ! C'est clairement une meilleure idée de s'appuyer sur la virgule flottante lorsque vous savez que vous vous rapprochez

en tous cas.

Qu'est-ce qui vient de se passer? Que se passe-t-il lorsque nous exécutons une boucle ? Examinons ce qui s'est passé

stylos, en regardant attentivement les valeurs de a et b à chaque passage dans la boucle.

Lorsque nous avons exécuté le programme, a et b prennent la valeur 0, car c'est ce que nous avons transmis

comme arguments pour la valeur initiale. Nous arrivons ensuite à la ligne

```
pour i dans la plage (n):
```

Rappelez-vous que cela demande à Sage de répéter les tâches suivantes n fois. La valeur de la boucle est

stocké dans la variable i, au cas où vous en auriez besoin ; nous n'avons pas besoin de i pour ce problème, mais un exemple ultérieur

illustrera comment cela peut être utile.

Maintenant, n est un argument, et dans ce cas nous lui avons attribué la valeur 80. Ainsi, les tâches indentées seront

répéter 80 fois. Illustrons maintenant ce qui se passe les premières fois :

Lorsque i = 0 : la première ligne indique à Sage de calculer Δ xdf (a, b) et de l'ajouter à b, puis d'attribuer le résultat

à b. Après cela, b = .1f (0,0)+0 = .2.

La deuxième ligne indique à Sage d'ajouter Δ x à a, puis d'affecter le résultat à a. Après ça, a = .1+0 = .1.

Lorsque i = 1 : la première ligne indique à Sage de calculer Δ xdf (a, b) et de l'ajouter à b, puis d'attribuer le résultat

à b. Après cela, b = .1f (.1,.2)+.2 .42.

La deuxième ligne indique à Sage d'ajouter Δ x à a, puis d'affecter le résultat à a. Après ça, a = .1+.1 = .2.

Lorsque i = 2 : la première ligne indique à Sage de calculer Δ xdf (a, b) et de l'ajouter à b, puis d'attribuer le résultat

à b. Après cela, b = .1f (.2,.42)+.42 .66.

La deuxième ligne indique à Sage d'ajouter Δ x à a, puis d'affecter le résultat à a. Après ça, a = .1+.2 = .3.

. . . et ainsi de suite. Répétez cette opération 80 fois et vous obtenez la valeur signalée par

Sage.
Comment cela marche-t-il? ou, une introduction aux collections
Nous discutons plus en détail du fonctionnement de ce processus. Les boucles définies fonctionnent en passant sur un
collection; en général, vous pouvez utiliser le mot-clé for sous la forme
pour variable *dans la* collection :
et Sage effectuera les tâches en retrait suivantes autant de fois qu'il y a d'objets dans la collection.
Au premier passage dans la boucle, la variable prend la valeur du « premier » élément de la collection ;
à chaque passage suivant, la variable prend la valeur de l'élément de la collection qui « suit » la valeur actuelle de la variable.
Nous mettons « premier » et « suivre » entre guillemets car dans certaines collections, ce que Sage considère comme le
L'élément « premier » peut ne pas être ce que vous attendez, et la valeur qu'il considère pour « suivre » le courant
valeur peut ne pas être ce que vous considérez pour le suivre. Ce n'est pas un problème comme vous pouvez le penser,
car cela ne se produit que dans les collections où vous ne devriez pas vous attendre à un « premier », « deuxième », etc.
Nous en parlerons dans une seconde.

Mais qu'est-ce qu'une collection ? Comme on peut s'y attendre d'après son nom, une collection est un objet qui « con-
contient » d'autres objets. Nous pouvons classer les collections de deux manières.
• La première classification est de savoir si une collection est indexée.
– Les collections indexées ordonnent leurs éléments qui vous permettent d'accéder à n'importe quel élément ac-
selon son emplacement.
– Les collections non indexées n'ordonnent pas leurs éléments, vous pouvez donc accéder à n'importe quel élément,
mais pas par son emplacement.
• La deuxième classification est de savoir si une collection est modifiable.
– Les collections mutables vous permettent de modifier leurs valeurs.
– Les collections immuables ne permettent pas de modifier leurs valeurs.
Nous utilisons cinq types de collections.
Un tuple est une collection indexée et immuable ; nous nous référons à un tuple de n éléments comme un n-uplet.
Vous créez un tuple en utilisant des parenthèses ou la commande tuple() , en insérant une autre

collection entre-
entre ses parenthèses ; par exemple, les deux commandes suivantes font la même chose :

```
sage : a_tuple = (3, pi, -I)
sage : b_tuple = tuple([3, pi, -I])
sage : a_tuple == b_tuple
Vrai
```

Vous pouvez également créer un "tuple vide" en ne plaçant rien entre les parenthèses, que ce soit `()` ou
`tuple()` . En tant que collections immuables, les tuples sont utiles pour communiquer des données qui ne devraient pas
être changé; c'est-à-dire des constantes. Vous avez déjà vu et utilisé des tuples ; nous les avons largement utilisés pour
fournir des points aux commandes de traçage.
Une liste est une collection indexée et modifiable. Vous créez une liste en utilisant des crochets ou la `liste ()` com-
mand, en insérant une autre collection entre les crochets ou les parenthèses ; par exemple,

```
sage : a_list = [3, pi, -I]
```

Vous pouvez également créer une "liste vide" en ne plaçant rien entre les parenthèses ou les crochets, soit
`[]` ou `liste()` . En tant que collections mutables, il est facile de modifier à la fois une liste et ses éléments. Si tu as besoin
pour stocker des valeurs susceptibles de changer, vous avez besoin d'une liste, pas d'un tuple, car vous ne pouvez pas modifier un tuple.
Un ensemble est une collection mutable non indexée. Vous créez un jeu à l' aide des accolades ou l' `ensemble ()` com-
mandat; par exemple,

```
sage : a_set = {3, pi, -I}
sage : a_set
{-je, 3, pi}
```

Remarquez comment "l'ordre" des éléments a changé. Vous pouvez également créer un « ensemble vide » en ne plaçant rien
entre parenthèses, comme ceci : `set()` . Comme ils sont mutables, vous pouvez modifier un ensemble. Un important
La propriété d'un ensemble est qu'il ne stocke qu'une seule copie de n'importe quel élément ; essayer d'ajouter des copies supplémentaires
nous en laisse un seul quand même. Par example:

```
sage : autre_ensemble = {2, 2, 2, 2, 2, 2}
sage : un autre_set
{2}
```

Nous n'utilisons pas souvent des ensembles dans ce texte ; bien qu'ils aient des utilisations importantes, ils peuvent être difficiles à utiliser, car

ils n'acceptent que les objets immuables, et de nombreux objets Sage sont mutables. Par exemple, vous pouvez
stocker les tuples dans un ensemble, mais pas les listes. D'ailleurs, les ensembles eux-mêmes sont modifiables, vous ne pouvez donc pas
ranger un ensemble dans un autre.

```
sage : { a_tuple }
{(3, pi, -I)}
sage : { a_set }
Erreur-type:
type non illisible :
'ensemble'
```

Afrozen set est une collection non indexée et immuable. Vous créez un ensemble gelé en utilisant le `congeléset()`
commander; par exemple,

```
sage : f_set = congeléset(a_set)
sage : f_set
gelé({-I, 3, pi})
```

Vous pouvez également créer un « ensemble gelé vide » en ne plaçant rien entre les parenthèses, `freezeset()` .
Les ensembles gelés sont particulièrement utiles lorsque vous avez besoin d'ensembles d'ensembles : vous ne pouvez pas stocker un ensemble mutable à l'intérieur
un ensemble, donc vous stockez un ensemble gelé à l'intérieur d'un ensemble.
Un dictionnaire est comme une liste, en ce sens qu'il s'agit d'une collection indexée et modifiable. C'est différent d'une liste dans
que l'indexation se fait par entrée plutôt que par position. Vous créez un dictionnaire en utilisant des accolades et
deux points ; les accolades délimitent le dictionnaire, tandis que les deux points indiquent une correspondance entre
clés (entrées du dictionnaire) et valeurs (définitions des entrées). Par example:

```
sage: a_dict = {x:2, x^2 + 1:'bonjour'}
```

Dans ce dictionnaire (plutôt idiot), l'entrée `x` correspond à la valeur `2` , tandis que l'entrée `x^2 + 1` correspond à la valeur `'bonjour'` . Une autre façon de créer un dictionnaire est d'utiliser le `dict()` commande avec une collection de 2-uplets ; la première entrée du tuple devient la clé, la seconde
devient la valeur.

```
sage : tup_dict = dict(( (x,1), (15,-71), (cos(x),3) ))
sage : tup_dict[cos(x)]
3
```

Vous pouvez également créer un "dictionnaire vide" en ne plaçant rien entre les parenthèses d'un `dict()`
commande, `dict()` . Nous avons déjà utilisé des dictionnaires lors de la substitution de dictionnaire.
Comment fonctionne l'indexation ? La réponse à cette question dépend quelque peu du type de
collection, mais cela implique toujours les crochets, `[]` .

Pour un dictionnaire, vous tapez la clé entre les crochets, et Sage renvoie la valeur attribuée à cette clé :

```
sage : a_dict[x^2 + 1]
'Bonjour'
```

Nous pouvons maintenant expliquer comment fonctionne la substitution de dictionnaire dans une expression : lorsque vous tapez

```
sage : f = x^2 + 2
sage : f({x:1})
```

puis Sage utilise le dictionnaire {x:1} pour interpréter chaque x dans f comme un 1 à la place.
Pour les tuples et les listes, l'indexation a une signification analogue à celle d'un indice en mathématiques.
Tout comme a_1 , a_2 , ..., a_i , ... indiquent les premier, deuxième, ..., ième, ... éléments d'une séquence, LorT[i]
indique l'élément en position i. La légère différence de formulation est importante ; pour Sage toi
besoin de fournir une « position juridique », ce qui n'est pas tout à fait le même que vous pourriez vous attendre :

```
sage : a_tuple
(3, pi, -I)
sage : a_tuple[1]
pi
sage : a_tuple[2]
-JE
```

Si vous regardez attentivement la numérotation, vous remarquerez que Sage commence à numéroter ses éléments à po-
position 0, pas position 1. Pour lire le premier élément de a_tuple , vous devez en fait taper a_tuple[0] .
Supposons que la collection C est soit un tuple, soit une liste, et a n éléments. Le sens de « juridique
position » correspond au tableau suivant :

C[0]
C[1]
C[2]
C[n-2]
C[n-1]
C[n]

quel élément ? PANIQUE! première seconde
la troisième
. . . pénultième
dernier
PANIQUE!

C[-n-1]
C[-n] C[-n+1] C[-n+2]
C[-2]
C[-1]

Comme d'habitude, PANIQUE ! signifie « une sorte d'erreur se produit ! » Nous en discutons dans la section suivante.
tion, mais remarquez quelque chose de surprenant : les indices négatifs ont un sens ! Ils te font reculer
à travers les éléments de C. Nous n'utiliserons pas cette fonctionnalité dans ce texte, mais il y a des occasions
où il peut être utilisé à bon escient.
Ce que vous pouvez faire avec les collections. Nous n'abordons pas ici les applications des collections,
autant que les commandes utilisées peuvent les utiliser et les méthodes que vous pouvez leur envoyer. Nous avons
déjà vu comment vous pouvez accéder aux éléments des collections indexées via l'opérateur bracket.
Les cinq collections. Les procédures et opérations suivantes sont communes aux cinq
tion :

---

len (collection)
le nombre d'éléments dans la collection
élément de collection
Vrai si et seulement si l'élément est dans la collection (si col-
lection est un dictionnaire, cela signifie que l'élément ap-
poires comme clé)
max(collecte)
le plus grand élément de la collection
min(collecte)
le plus petit élément de la collection
trié(collection ,
reverse= *True* ou *False*)
renvoie une copie triée de la collection (ordre inverse
si inverse= *Vrai*)
trié(collection , clé ,
reverse= *True* ou *False*)
renvoie une copie de la collection, triée selon
clé (ordre inverse si reverse= *True*)
La plupart du temps, un programme ne sait pas à l'avance avec combien d'éléments il doit travailler,
donc avoir un moyen de déterminer ce nombre est extrêmement utile.

```
sage : len(a_set)
3
sage : len(another_set)
```

```
1
sage : sqrt(2) dans a_list
Faux
sage : -sqrt(-1) dans a_tuple
Vrai
sage : trié(un_tuple)
[3, pi, -I]
```

Remarquez comment la commande `sorted()` renvoie toujours une liste, même si nous avons fourni un tuple pour son
contribution.

Il y a des moments où vous voudrez peut-être trier une collection d'une manière différente de celle de Sage.
défaut. Vous pouvez modifier les critères de tri à l'aide de l' option `clé` , à laquelle vous affectez une procédure
qui renvoie un objet qui sert de clé de tri, un peu comme un dictionnaire. Nous n'utiliserons pas cela
souvent, mais l'exemple ci-dessus illustre le point : du point de vue de l'analyse complexe, il semble étrange de trier −i après 3 et π , alors que sa « norme » est plus petite. Pour trier par la norme, nous pouvons
écrivez une procédure qui calcule la norme de tout nombre complexe et utilisez-la comme clé :

```
sage : def by_norm(z):
renvoie real_part(z)**2 + imag_part(z)**2
sage : trié(a_tuple, key=by_norm)
[-I, 3, pi]
```

Listes et tuples. Les méthodes et opérations suivantes s'appliquent uniquement aux listes et aux tuples :

LorT `.count(` élément `)`
le nombre d'élément de temps apparaît dans LorT

LorT `.index(` élément `)`
l'emplacement de l'élément dans LorT

LorT1 `+` LorT2
crée une nouvelle liste/tuple dont les éléments sont ceux
de LorT1, suivis de ceux de LorT2

Il devrait être logique que vous ne puissiez pas appliquer ces techniques à des ensembles, des ensembles figés ou des dictionnaires.

Aucun élément ou clé ne peut apparaître plus d'une fois dans un ensemble, un ensemble gelé ou des dictionnaires, donc `count()`
renverrait au plus 1 dans chacun. Les ensembles et les ensembles gelés ne sont pas indexables, donc `index()` ne fait pas
sens. Pour les dictionnaires, `index()` peut avoir du sens, mais il n'est pas directement

implémenté.[3]
Nous ne pouvons « ajouter » que deux listes ou deux tuples ; vous ne pouvez pas ajouter une liste à un tuple, ou vice-versa.
Nous avons épuisé les commandes disponibles pour les tuples, mais vous pouvez faire un peu plus avec
une liste. Puisque les listes sont à la fois indexables et modifiables, nous pouvons modifier un élément particulier d'une liste
en utilisant l'affectation d'élément :

```
sage : a_list[0] = 1
sage : a_list
[1, pi, -I]
sage : a_list[0] = 3
sage : a_list
[3, pi, -I]
```

Comme d'habitude, nous utilisons 0 car Sage considère que le premier élément a l'emplacement 0.
Outre l'affectation d'éléments, les listes présentent certaines méthodes non disponibles pour les tuples ; voir tableau 1. [4] A
quelques distinctions valent la peine d'être faites à propos de ces commandes :

• .pop() et .remove() diffèrent en ce que l'un fait référence à un emplacement, tandis que l'autre fait référence à un
élément particulier.

```
sage : a_list.pop(1)
pi
sage : a_list.remove(3)
sage : a_list
[-JE]
```

Il n'y a jamais eu d'élément à l'emplacement 3, soulignant que remove() recherchait un élément dont la valeur était 3. Cela peut être assez sophistiqué, car Sage effectuera des réductions pour vérifier si un élément a une valeur donnée :

3 C'est en fait faisable, mais quelque peu alambiqué.
4 Comme d'habitude, la liste n'est peut-être pas exhaustive, et ne l'est probablement pas. Nous n'adressons que les commandes que nous pensons
dont vous aurez besoin le plus souvent, et ceux disponibles au moment de la rédaction de cet article. Pour voir si la liste est exhaustive, n'oubliez pas
que vous pouvez obtenir une liste en tapant a_list. , puis en appuyant sur la touche Tab .

L .append(élément)
ajouter un élément à la fin de L
L .extend(collection)
ajouter les éléments de collection à L
L .insert(emplacement ,
élément)

addelement à l'emplacement donné (à partir de 0)
L .pop()
supprime (et renvoie) le dernier élément de la liste
L .pop(emplacement)
supprime et renvoie l'élément à l'endroit indiqué
emplacement (à partir de 0)
L .remove(élément)
supprime l'élément nommé
L .reverse()
renverse la liste
L .sort()
trie la liste selon le mécanisme par défaut de Sage
nisme
L .sort(clé)
trie la liste selon la clé donnée
L .sort(clé ,
reverse= *True* ou *False*)
trie la liste dans l'ordre inverse de celui donné par
clé
TABLEAU 1. Opérations propres aux listes

```
sage : a_list = [3, pi, -I, (x+1)*(x-1)]
sage : a_list.remove(x**2 - 1)
sage : a_list
[3, pi, -I]
```

• .Sort () diffère de tri () en ce qu'elle ne copie pas la première liste et retourne *Aucun* .
La liste est triée « sur place ». Vous pouvez penser à trié () comme laissant la liste d'origine intacte,
et .sort() pour modifier la liste.
Ensembles et ensembles gelés. Les méthodes du tableau 2 s'appliquent uniquement aux ensembles et aux ensembles gelés. Opérations
qui modifient l'ensemble ne s'appliquent pas aux ensembles gelés.
Plusieurs des méthodes correspondent à des opérations ou relations mathématiques. Un important
la distinction à faire est que les méthodes qui se terminent par _update modifient l'ensemble lui-même et renvoient *None* ,
tandis que les méthodes _update -less correspondantes renvoient un nouvel ensemble et laissent l'original intact.
Dictionnaires. Nous n'utiliserons pas la plupart des fonctionnalités disponibles pour un dictionnaire. Les seuls que nous
mention apparaissent dans le tableau 3. Nous avons déjà montré comment accéder aux éléments d'un dictionnaire en utilisant
l'opérateur crochet [] , nous notons donc simplement que vous pouvez ajouter ou modifier des entrées dans un dictionnaire le

de la même manière que vous modifiez les entrées d'une liste.

S .add(élément)
ajouter un élément à l'ensemble S

S .différence(C)
renvoie une copie de l'ensemble ou ensemble gelé S, mi-
nus tous les éléments de la collection C

S .difference_update(C)
supprime les éléments de la collection C de
l'ensemble S lui-même

S .intersection(C)
renvoie l'ensemble des éléments à la fois dans l'ensemble
ou ensemble congelé S et la collection C

S .intersection_update(C)
supprime de l'ensemble S tous les éléments de C

S .isdisjoint(C)
Vrai si et seulement si l'ensemble ou l'ensemble gelé S a
aucun élément en commun avec la collection
C

S .issubset(C)
Vrai si et seulement si tous les éléments de l'ensemble
ou l'ensemble congelé S sont également dans la collection C

S .issuperset(C)
Vrai si et seulement si tous les éléments du
la collection C est également dans l'ensemble ou l'ensemble congelé
S

S .pop()
supprime et renvoie un élément de l'ensemble S

S .remove(élément)
supprime l'élément s'il apparaît dans S ; soulève un
KeyError si ce n'est pas le cas

S .symmetric_difference(C)
renvoie la différence symétrique entre S
et T; c'est-à-dire les éléments non communs à
S et T

S .symmetric_difference_update(C)
retirer de S tout élément apparaissant dans
C, et ajoute à S les éléments de T qui
ne sont pas en S

S .union(C)
renvoie l'union de l'ensemble ou ensemble figé S
avec la collection C
S .update(C)
ajoute tous les éléments de la collection C à
l'ensemble S
TABLEAU 2. Opérations propres aux postes

```
sage : a_dict
{x^2 + 1 :
'bonjour', x :
2}
sage : a_dict[0] = 'au revoir'
sage : a_dict[0] = -3
sage : a_dict
{0 :
-3, x^2 + 1 :
'bonjour', x :
2}
```

D .clear()
supprimer toutes les entrées dans D
D .has_key(clé)
renvoie *True* si et seulement si la clé a une valeur dans D
D .pop(clé)
supprime l'entrée pour la clé de D et renvoie son
valeur
D .popitem()
supprimer une entrée de D et retourner le tuple
(clé , valeur)
D .update(E)
ajoute les définitions de E à D
TABLEAU 3. Opérations propres aux dictionnaires
Un assistant pour la création de collections. Nous avons déjà utilisé la procédure range() , mais nous
n'ont pas encore discuté de sa signification. Son but est de créer un objet de plage , la manière de Sage de suivre un
suite d'entiers. Voici trois façons de créer une plage :
plage (b)
la séquence 0, 1, . . . , b -1
plage(a , b)
la séquence a,a +1,..., b -1
plage(a , b , d)

la séquence a,a + d,...,a + kd où
a + kd ≺ b et a +(k +1)d b
Vous pouvez considérer a comme une valeur de "début", b comme une valeur "d'arrêt" et d comme une valeur de "pas", de sorte que
range(-3,8,3) représente la séquence finie où a = -3, b = 8 et d = 3, ou
−3, −3+3, −3+2×3, −3+3×3 −→ −3, 0, 3, 6 .
Notez que 8 n'est pas un élément de la séquence ; en effet, b n'est jamais un élément de la séquence.
La procédure range() ne crée pas de séquence concrète d'entiers ![5] Au contraire, il crée un objet range : le début et la fin d'une séquence, le long du pas d'un élément au suivant.

```
sage : L = plage (1, 10, 2)
sauge : L
plage (1, 10, 2)
sage: L.start
1
sage: L.stop
dix
sage: L.step
2
sauge : 5 en L
Vrai
sauge : 6 en L
Faux
```

5 C'était le cas dans les versions antérieures de Python, et donc de Sage, mais cela a changé avec Python 3.

Erreurs lors de la création ou de l'accès aux collections. Il serait sage d'examiner quel type d'erreurs
apparaissent lorsque vous essayez d'en créer un ou d'accéder à un élément.

• Lors de la création d'une collection à l'aide d'une commande plutôt que de symboles, ne négligez pas
plier une autre collection. Sage n'acceptera pas une simple séquence d'éléments séparés par des virgules.
ments.

```
sage : nouveau_tuple = tuple(2, 3, 4)
Erreur-type:
tuple() prend au plus 1 argument (3 donnés)
```

• Les tuples sont immuables. N'essayez pas de changer les valeurs. Si vous pensez que vous pourriez avoir besoin de changer
la valeur d'un élément, utilisez plutôt une liste.

```
sage : a_tuple[0] = 1
Erreur-type:
L'objet 'tuple' ne prend pas en charge l'affectation d'élément
```

• Bien que les listes et les tuples acceptent les indices négatifs, ils ne « bouclent » pas plus que cette. Si la collection a une longueur n, n'essayez pas d'accéder aux éléments n ou −n −1.

```
sage : len(a_list)
```

3
sage : a_list[3]
Erreur d'index :
index de liste hors de portée
• Comme déjà mentionné, n'essayez pas d'ajouter une liste à un tuple, ou vice-versa.
sage : a_list + a_tuple
Erreur-type:
ne peut concaténer qu'une liste (pas un "tuple") en liste
• Les ensembles ne sont pas indexés, vous ne pouvez donc pas accéder à un élément particulier d'un ensemble.
sage : len(a_set)
3
sage : a_set[1]
Erreur-type:
L'objet 'set' ne prend pas en charge l'indexation
• Sage génère une erreur chaque fois que vous essayez d'accéder à une entrée inexistante d'un dictionnaire.
sage : a_dict[12]
Erreur clé :
12
Répéter sur une collection
Nous pouvons maintenant revenir à l'objectif principal de ce chapitre, Sage est pour mot - clé. Nous vous rappelons
que sa forme générale est

---

pour variable dans la collection :
suivi d'une liste de tâches en retrait ; cela correspond au pseudo code
pour variable ∈ collection
suivi d'une liste de tâches en retrait. Au premier passage dans la boucle, la variable prend le valeur du « premier » élément de collection ; à chaque passage suivant, la variable prend la valeur de
l'élément de la collection qui « suit » la valeur actuelle de la variable. On appelle variable la boucle
variable et collection le domaine de la boucle, ou simplement "domaine".
L'effet est que si l'élément est dans le domaine C, la boucle for attribuera à un moment donné sa valeur
à variable, puis effectuez toutes les tâches en retrait. Cette propriété d'un traitement complet de C ne
ne change pas même si vous modifiez la variable pendant la boucle ; Sage se souvient de l'élément qu'il a sélectionné
à partir de C, et sélectionne l'élément qui le « suit », plutôt que la valeur de la variable. Afin que vous puissiez
modifiez la variable de boucle si nécessaire, sans vous soucier du comportement de la boucle.[6]

D'autre part! si vous modifiez le domaine, des choses désagréables peuvent arriver. La boucle for
veut parcourir chaque élément de son domaine, mais il ne fait pas de copie du domaine au au départ, donc si le corps de la boucle modifie le domaine, des choses étranges peuvent se produire, y compris un
boucle infinie.[7] L'un des exercices de programmation illustrera cela — vous ne devriez pas essayez-le dans Sage à moins que vous ne soyez prêt à appuyer sur le bouton Arrêter (nuage), sélectionnez le menu Action, puis
Interruption (autre serveur) ou maintenez la touche Ctrl enfoncée et appuyez sur C (ligne de commande).
Il existe de nombreux cas où vous voudrez utiliser la valeur de la variable de boucle.
Premier exemple : calcul d'une intégrale de Riemann. Rappelons qu'on ne peut pas toujours simplifier un
intégrale indéfinie à la « forme élémentaire », donc lorsque nous avons besoin d'une intégrale définie, nous pouvons approximer
sa valeur en utilisant l'une de plusieurs méthodes. L'intégrale est définie comme

$$\int_{une}^{b} f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{m} f(x_i^*)\Delta x, \Delta x = \frac{b-a}{m}$$

et x_i^* se trouve dans le ième sous-intervalle.
En « anglais simple », la valeur de l'intégrale est la limite des « sommes de Riemann ». Ces sommes environ
mate l'intégrale en utilisant n rectangles de largeur Δx et de hauteur $f x_j^*$. Il y a trois coutumes
façons de sélectionner x_j^* :
• par l'extrémité gauche, où $x_i^* = a + (i - 1)\Delta x$;
• par extrémité droite, où $x_i^* = a\, i + \Delta x$; et
• par milieu, où $x_i^* = a + (i - 1/2)\Delta ix$.
Lorsque n augmente, l'erreur diminue, il est donc possible d'approcher l'intégrale en évaluant
m

$$\sum_{i=1} f(x_i^*)\Delta x$$

pour une grande valeur de n. Le symbole de sommation Σ nous ordonne de laisser la variable de sommation i
croître de 1 à n, et pour chaque valeur d'évaluer l'expression à sa droite, et l'ajouter à la somme croissante.

6 Ceci est très différent de nombreux langages informatiques, tels que C, C++ et Java. Dans ces langues, modifier la variable de boucle pendant une boucle for peut avoir des résultats catastrophiques.

7 Si vous avez besoin d'une définition de « boucle infinie », consultez l'index, où l'on a volé une blague au glossaire de l'Amiga-
Manuel DOS 3.1.

RÉPÉTITION SUR UNE COLLECTION
120

Remarquez le langage que nous utilisons ici : « chaque » et « tous ». Quand la solution d'un problème en-
implique ce genre de mots, c'est un signe révélateur que vous avez besoin d'une boucle définie sur la collection
dont les éléments sont en cause.

Pseudocode. Dans ce cas, nous allons créer une boucle pour l'approximation de l'extrémité gauche. C'est relativement
facile à transformer en pseudo-code :

algorithme Left_Riemann_approximation
contributions
• a, b
• f , une fonction intégrable sur [a, b]
• n, le nombre de rectangles à utiliser pour approximer $\int_a^b f(x)\,dx$
les sorties
• S, la somme de Riemann à l'extrémité gauche de $\int_a^b f(x)dx$, en utilisant n rectangles
approximer la superficie
fais
soit $\Delta x = (b - a) / n$
soit S = 0
pour i (1,...,n)
soit $x_i^* = a + (i - 1)\,\Delta x$

ajouter fx ∗
i∆ x S
Retour
Regardons ce que fait ce code. Initialement, il attribue Δ x, que nous avons vraiment besoin de savoir,
car il ne fait pas partie de l'entrée. Il initialise ensuite le résultat, S, à 0, une bonne idée quand on veut
pour créer une somme. Une fois ceci terminé, il passe dans la boucle en attribuant à i chaque valeur de 1 à n.
Avec cette valeur, il effectue les deux tâches en retrait ci-dessous : choisissez une valeur de x
∗
je
selon
à la formule pour les extrémités gauches, et ajoutez l'aire d'un rectangle à S. Une fois la boucle passée
à travers chaque valeur 1, . . . , n, il renvoie S.
Remarquez comment ce code utilise la valeur de la variable de boucle i pour construire x ∗
je
.
Implémentation sage. L'implémentation de ce pseudocode dans Sage nécessite quelques modifications mineures.
Le principal problème est que nous ne pouvons pas utiliser de symboles grecs, d'indices ou de décorateurs, donc les noms
des variables doivent changer quelque peu.
Cependant, nous devons apporter un autre changement qui peut être facile à manquer. La formule que nous utilisons
s'attend à ce que je prenne les valeurs 1, . . . , n. La façon naturelle de faire passer Sage sur de tels nombres est
avec la commande range() , mais par défaut range(n) commence par 0 et se termine par n−1 ! Il y a
plusieurs façons de vous assurer d'avoir les bons numéros ; nous optons pour range(1,n+1). C'est-à-dire que nous voulons
commencer à 1 et s'arrêter à n+1 ; cela inclut 1 mais pas n+1.

RÉPÉTITION SUR UNE COLLECTION
121

```
sage : def Left_Riemann_approximation(f, a, b, n, x=x):
f(x) = f
Delta_x = (b - a) / n
S = 0
pour i dans la plage (1, n+1) :
# extrémité gauche de l'intervalle
xi = a + (i - 1) * Delta_x
# ajouter une aire de rectangle sur l'intervalle
S = S + f(xi) * Delta_x
```

Retour
Cela fonctionne assez bien :
sage: Left_Riemann_approximation(t^2 + 1, 0, 1, 100, t)
Deuxième exemple : vérifier si n est un champ. Pour un autre exemple, rappelez-vous l'exercice
dans. 51, où nous devons vérifier si n est un champ. Nous savons déjà que c'est une bague, nous avions donc besoin
vérifiez simplement que chaque élément non nul a un inverse multiplicatif. Vérification de tous les éléments
à la main est assez pénible, d'autant plus que n grandit. Une boucle serait donc un outil souhaitable
ici. Tout ce que nous avons à faire est de vérifier si chaque élément a un inverse, et nous pouvons le faire en
vérifier le produit de chaque élément avec chaque autre élément.
Remarquez à nouveau que nous utilisons les mots « chacun » et « chaque », qui nous indiquent que nous devons utiliser un
boucle définie.
Pseudocode. Nous décrivons une implémentation assez simple de notre solution en pseudo-docode :8
algorithme produire_tous_produits
contributions
• n avec n 2
les sorties
• une liste L telle que L i est l'ensemble des produits de i ∈ n avec tous les autres
éléments de n
fais
Soit L = []
pour i n
Soit M =
pour j n
ajouter ij à M
ajouter M à L
retour L
8 Ce n'est pas une bonne solution. Nous l'améliorerons lorsque nous discuterons de la prise de décision.

Regardons ce que fait ce code. Il crée une liste L, initialement vide. (Notre pseudocode utilise crochets pour désigner une liste et crochets sans rien entre eux pour désigner une liste vide. Quelque
les auteurs utilisent des parenthèses à la place, mais ce n'est pas une règle absolue, et nous ne voulons pas risquer

confusion avec des tuples.) Le code parcourt ensuite tous les éléments de n , appelant l'élément à chaque passage i. Pour chacun de ces éléments, il crée un nouvel ensemble M, initialement vide. Il boucle maintenant
à travers n à nouveau, en appelant l'élément à chaque passage j. Il est important de noter que je reste fixe
tandis que cette boucle intérieure change j à chaque passage ; à cause de cela, nous pouvons dire avec certitude que M
contient le produit de ce i fixe avec chaque autre élément de n une fois la boucle interne terminée.
Le code ajoute ensuite cet ensemble à L, concluant les tâches dans la boucle externe. Une fois la boucle extérieure
a traversé chaque élément de n , le code peut renvoyer L.
Notez que le code utilise les valeurs des variables de boucle i et j, qui sont elles-mêmes des entrées
de n .
En gardant L sous forme de liste, nous garantissons que ses éléments correspondent à l'ordre dans lequel i passe
à travers les éléments de n . Ce n'est pas si important pour M ; tout ce que nous voulons, ce sont les produits de i et
tous les autres éléments, pas nécessairement leur ordre, bien que cela puisse être utile dans certains contextes.
Ce pseudocode présente une propriété importante des boucles : l'imbrication. Cela se produit chaque fois que
nous incluons une structure de contrôle dans une autre. C'est souvent utile, mais cela peut aussi être très
déroutant, vous devriez donc éviter d'en faire trop. Si votre nidification dépasse 3 ou 4,
c'est une bonne idée de séparer les boucles internes dans une autre procédure. Cela aide à rendre le code plus facile
à comprendre, et comme les tâches sont souvent utilisables à plusieurs endroits, cela peut aussi vous faire gagner du temps
sur la route aussi.
Implémentation sage. Il est relativement facile de le transformer en code Sage. La principale différence
avec le pseudocode est que nous devons créer un anneau pour n et nous assurer que Sage voit i et j comme
éléments de cet anneau. Nous montrons d'abord la manière "évidente" de le faire, puis une manière plus intelligente.

```
sage : def produire_tous_produits(n):
R = ZZ.quo(n)
# résultat final :
L[i] est le produit de i
L = liste()
pour i dans la plage (n):
M = set() # ensemble de multiples de i
```

```
pour j dans la plage (n):
M.ajouter(R(i)*R(j))
L.append (M)
retour L
```

Pour voir comment cela fonctionne, essayez-le avec quelques valeurs de n :

```
sage : produire_tous_produits(4)
[{0}, {0, 1, 2, 3}, {0, 2}, {0, 1, 2, 3}]
sage : produire_tous_produits(5)
[{0}, {0, 1, 2, 3, 4}, {0, 1, 2, 3, 4}, {0, 1, 2, 3, 4}, {0, 1, 2,
3, 4}]
```

On voit ça:

• Pour 4 , 1 n'apparaît pas dans les produits de 0 et 2. Cela ne nous dérange pas 0, puisque nous nous soucions seulement sur les éléments non nuls, mais 2 est fatal : 4 n'est pas un champ.

• Pour 5 , 1 apparaît dans tous les produits sauf 0, il s'agit donc en fait d'un champ.

Malheureusement, ce code n'est pas une bonne solution, car à mesure que n augmente, les listes de produits deviennent long et rapide, ce qui rend difficile de vérifier si 1 est un élément d'un ensemble. Cela peut inciter nous demander : Pourquoi vérifions-nous cela ? Il est facile de demander à Sage de vérifier si un élément apparaît dans un collection. Modifions notre pseudocode à partir de ceci :

ajouter M à L

pour ça:[9]

ajouter *True* à L si 1 M ; *Faux* sinon

Pour le code Sage, on utilise le fait que l'expression 1 dans M se simplifie automatiquement (le seul le changement apparaît en rouge) :

```
sage : def produire_tous_produits(n):
R = ZZ.quo(n)
# résultat final :
L[i] est le produit de i
L = liste()
pour i dans la plage (n):
M = set() # ensemble de multiples de i
pour j dans la plage (n):
M.ajouter(R(i)*R(j))
L.append ( 1 en M)
retour L
```

Regardez comment cela se comporte beaucoup plus commodément qu'avant :

```
sage : produire_tous_produits(4)
[Faux, vrai, faux, vrai]
sage : produire_tous_produits(5)
[Faux, vrai, vrai, vrai, vrai]
sage : produire_tous_produits(10)
```

[Faux, vrai, faux, vrai, faux, faux, faux, vrai, faux, vrai]

Dans ce cas, il est extrêmement facile de déterminer si n est un champ : voyez si *False* apparaît n'importe où
en plus de la première position (qui correspond à 0, pour laquelle aucun inverse n'est nécessaire).

Vous pourriez vous demander si nous ne pouvons pas simplifier encore plus cela. En effet, nous le pouvons. Une façon serait
nécessitent pas mal d'algèbre booléenne, nous le reportons donc à plus tard. Mais une autre façon est d'observer
que Sage considère ZZ.quo(n) comme une collection et nous pouvons également parcourir ses éléments : (changements
apparaissent en rouge)

9 Ce n'est toujours pas une bonne solution. Nous l'améliorerons lorsque nous discuterons de la prise de décision.

COMPRÉHENSION : RÉPÉTER DANS UNE COLLECTION

124

```
sage : def produire_tous_produits(n):
R = ZZ.quo(n)
# résultat final :
L[i] est le produit de i
L = liste()
pour i dans R :
M = set() # ensemble de multiples de i
pour j dans R :
M.add( i*j )
L.append (1 en M)
retour L
```

Si vous testez cela, vous verrez comment cela fonctionne comme avant, bien que le code soit plus simple. Gardez à l'esprit
à quel point Sage facilite souvent le travail avec des objets mathématiques de manière naturelle.

Compréhensions : répéter dans une collection

Sage offre un moyen spécial de créer des collections qui abrège la structure de la boucle for et rend
c'est un peu plus facile à utiliser et à lire. Celles-ci sont appelées compréhensions et imitent le constructeur d'ensembles
notation des mathématiques. Dans la notation set-builder, nous définissons un ensemble en spécifiant d'abord un domaine,
puis un critère de sélection des éléments de ce domaine. Par exemple, l'expression

$$S = n : 2 n \leq 2^{10}$$

utilise la notation set-builder pour définir S comme l'ensemble de tous les nombres naturels compris entre 2 et 2^{10} ,
compris. Les compréhensions donnent à Sage un moyen naturel d'imiter cela dans des endroits où il serait
utile et réalisable.

Pour définir une compréhension, utilisez le modèle suivant :
collection (expression pour la variable dans collection_or_range) .
Ceci est effectivement équivalent à l'une des séquences de commandes suivantes (D est une collection ;
a, b et n sont des nombres entiers) :

```
sage : C = collection ()
sage : pour d dans D :
C = C.append/insert/update( expression )
```

(Ici, vous pouvez utiliser « collection » pour n'importe quelle liste ou ensemble, en choisissant ajouter, insérer ou mettre à jour
ou :

```
sage : C = collection ()
sage : pour d dans la plage ( a , b , n ):
C = C.append/insert/update( expression )
```

Dans chaque cas, le résultat est que C contient les valeurs prises par expression pour chaque valeur de l'un ou l'autre
D (premier cas) ou la plage d'entiers spécifiée (deuxième cas).

Pour voir cela en pratique, revenons à notre exemple des sommes de Riemann. Une façon dont nous pourrions
utiliser une compréhension consiste à générer les valeurs x. Les extrémités gauches ont la forme
$a + (i - 1)\,\Delta\, x$ où $i = 1,\ldots,n$,
et nous pouvons attribuer cela à une liste X de valeurs x en utilisant une compréhension de liste comme

```
X = [a + (i - 1)*Delta_x pour i dans la plage (1, n+1)] .
```

(Rappelez-vous que nous avons besoin de n+1 car la commande range() continue jusqu'à, mais sans inclure,
le deuxième nombre mentionné.) Nous pourrions alors boucler sur X , comme ceci:

```
sage : def Left_Riemann_approximation(f, a, b, n, x=x):
f(x) = f
Delta_x = (b - a) / n
# créer tous les points de terminaison gauche en un seul passage
X = [a + (i - 1)*Delta_x pour i dans la plage (1, n+1)]
S = 0
pour xi dans X :
# ajouter une aire avec hauteur à f(xi)
S = S + f(xi) * Delta_x
Retour
```

C'est un peu plus simple qu'avant, et cela a l'avantage de définir les valeurs x d'une manière ça a l'air mathématique.
Nous pouvons également décrire une manière plus simple de contourner la pénalité de la

création de la liste x . sauge
a une commande sum() qui est facile à comprendre. Plutôt que d'initialiser une variable somme S pour
zéro, et y ajouter des sommes partielles à chaque passage dans la boucle, nous pourrions utiliser la commande sum()
avec une compréhension de liste pour simplifier le programme et le faire ressembler davantage au programme mathématique
idée. Puisque nous utilisons des sommes de gauche, nous pouvons réécrire
$\sum_{i=1}^{m} f(x_i) \Delta x$
comme
$\sum_{i=1}^{m} f(a + i \Delta x) \Delta x$
qui devient
sum(f(a + (i*Delta_x))*Delta_x pour i dans la plage (1, n+1)) .
Tout ce que nous avons fait ici est de « traduire » l'idée mathématique en une commande Sage correspondante.
Cela peut sembler plus difficile à lire au début, mais une fois que vous vous y êtes habitué, c'est tout à fait naturel. La résultante
Le code Sage est

```
sage : def Left_Riemann_approximation(f, a, b, n, x=x):
f(x) = f
Delta_x = (b - a) / n
return sum(f(a + (i*Delta_x))*Delta_x \
pour i dans la plage (1, n+1))
```

DES EXERCICES
126
C'est beaucoup plus court que ce que nous avions avant.
Animation à nouveau
Rappel de p. 70 que nous avons créé une courte animation d'une fleur de lys modifiée en créant plusieurs
cadres, puis les joindre avec la commande animate() . Le plus intéressant ou instructif
les animations nécessitent un certain nombre d'images ; les créer à la main peut être une tâche fastidieuse, si
pas irréalisable. D'un autre côté, la plupart des animations dépendent également de manière cruciale des modèles ;
par exemple, le motif de notre animation p. 70 peut se résumer à
cos(nx)sin((n+1)x),

où n est compris entre 2 et 7 inclus. Bien sûr, nous pourrions vouloir plus d'images que cela. Lister
les compréhensions nous permettent de créer une liste de cadres, tels que les suivants :

```
sage: frames = [polar_plot(cos(n*x)*sin((n+1)*x), (x, 0, pi),
fill=Vrai, \
épaisseur=2, fillcolor='jaune', \
color='goldenrod', axes=False) \
pour n dans la plage (2,20)]
sage : fdl_anim = animer(frames, xmin=-1, xmax=1, \
ymin=-0,5, ymax=1,5, rapport d'aspect=1)
sage : show(fdl_anim, gif=True, delay=8)
```

Le résultat devrait être une animation d'une vitesse et d'une fluidité raisonnables. Si vous visualisez ce texte dans
Acrobat Reader, vous devriez voir la même animation ci-dessous, bien que la vitesse puisse différer légèrement ;
si vous regardez une copie papier du texte, vous devriez plutôt voir l'individu de l'animation cadres:

Des exercices

Vrai faux. Si la déclaration est fausse, remplacez-la par une déclaration vraie.

1. Une instruction for implémente une boucle définie.

---

2. Le concept que nous exprimons en pseudocode comme « répéter n fois » n'a pas de mot-clé correspondant dans
la plupart des langages informatiques.

3. Vous ne devez pas modifier la valeur d'une variable de boucle dans une boucle `for` , car cela peut affecter la prochaine passe
à travers la boucle.

4. Vous ne devez pas modifier les entrées du domaine d'une boucle `for` , car cela peut affecter le passage suivant
la boucle.

5. Les boucles définies ne sont pas un modèle utile pour l'itération.

6. La seule structure immuable que Sage vous offre est un tuple, donc si vous voulez l'immuabilité, vous êtes
bloqué avec l'indexation.

7. Vous ne pouvez construire un ensemble qu'à l'aide d'accolades ; par exemple, `{3, x^2, -pi }` .

8. Vous ne pouvez créer un ensemble vide que sur un ordinateur dont le clavier comporte un symbole.

9. La clé de tri renvoie *True* si et seulement si le premier argument est plus petit que le second.

10. Les compréhensions de liste vous permettent de créer une liste sans utiliser la structure de boucle for conventionnelle.
ture.

Choix multiple.

1. L'indexation en tuples et en listes correspond à quelle notation mathématique ?

A. fonctions

B. expressions

C. indices

D. apostrophes

2. On ne peut pas accéder à un élément d'un ensemble à l'emplacement numérique i car :

Les programmeurs de A. Sage étaient paresseux.

B. Les ensembles ne sont pas ordonnés et ne sont donc pas indexables.

C. Cela pourrait casser la boucle `for` .

D. La prémisse est incorrecte ; les parenthèses permettent d'accéder à un élément d'un ensemble à
emplacement i.

3. Nous ne pouvons pas accéder à un élément d'un dictionnaire à l'emplacement numérique i car :

Les programmeurs de A. Sage étaient paresseux.

B. Les dictionnaires ne sont pas ordonnés et ne sont donc pas indexables.

C. Les dictionnaires sont classés par entrée, ou « clé », plutôt que par emplacement numérique.

D. La prémisse est incorrecte ; les parenthèses nous permettent d'accéder à un élément d'un dictionnaire à nombre-
emplacement géographique i.

4. Nous ne pouvons pas accéder à un élément d'un tuple à l'emplacement numérique i car :

Les programmeurs de A. Sage étaient paresseux.

B. Les tuples ne sont pas ordonnés et ne sont donc pas indexables.

C. Les tuples sont censés être immuables, et accéder à un élément particulier violerait cela régner.

D. La prémisse est incorrecte ; les parenthèses nous permettent d'accéder à un élément d'un tuple au niveau numérique
emplacement i.

5. Laquelle des techniques mathématiques suivantes pourrait motiver l'utilisation d'une boucle `for` ?

A. itération

B. vérifier tous les éléments d'un ensemble

C. répéter un ensemble de tâches n fois

Tout ce qui précède

6. Laquelle des collections suivantes est immuable ?

A. un tuple

B. une liste
C. un ensemble
D. une plage ()
7. Laquelle des collections suivantes n'est pas indexable ?
A. un tuple
B. une liste
C. un ensemble congelé
D. une plage ()
8. Lequel des symboles ou identifiants suivants utilisons-nous dans notre norme de pseudocode pour tester
pour l'appartenance à une séquence ou à un ensemble ?
A.
B. ε
C. dans
D. dans
9. Modèle de compréhension quelle notation mathématique ?
Une fonction
B. ensemble
C. constructeur de décors
D. indices
10. L'imbrication se produit lorsque :
A. une ou plusieurs boucles for sont placées à l'intérieur d'une autre
B. nous créons une collection en utilisant la notation set-builder
C. nous créons un ensemble pour contenir un certain nombre d'éléments du même type
D. deux oiseaux s'engagent dans le sacre du printemps[dix]
Réponse courte.
1. Résumez le comportement de la commande range() lorsqu'elle fournit 1, 2 ou 3 entrées.
2. La propriété associative de multiplication est valable pour un ensemble S chaque fois qu'un triplet d'élément
s, t, u S satisfait s (tu)=(st) u. Supposons qu'un ensemble particulier S a 9 éléments. Combien de
produits un examen de tous les produits possibles exigerait-il? Astuce : La réponse est grande assez que vous ne voulez pas le faire à la main.
3. Considérez le code Sage suivant.

```
sage : pour i dans la plage (10) :
pour j dans la plage (i,10) :
pour k dans la plage (j,10) :
imprimer(i, j, k)
```

(a) Qu'imprime-t-il ? Ne donnez pas tout, juste assez pour montrer le modèle. Utiliser mots pour expliquer l'ordre dans lequel les lignes sont imprimées.
(b) Trouver une quadruple inégalité[11] impliquant i , j et k qui est valable pour chaque ligne.
4. Ce problème prend en compte le code Sage suivant.

[dix] Nous apprécierions si Stravinksyles héritiers ne poursuivraient pas.

11 Une quadruple inégalité a la forme a ≤ b ≤ c ≤ d ≤ e. Mis à part i , j et k , vous aurez besoin d'utiliser deux constantes.

DES EXERCICES
129

```
sage : def ec(k) :
varier')
p = Graphiques()
pour i dans la plage (2, k+1) :
p = p + implicite_plot(x**2 + y**2/(1-1/sqrt(i))==1,
(x,-2,2), (y,-2,2), couleur=(0,i/k,.8-i/k))
retour p
```

(a) Décrivez ce que l'appel ec(10) renvoie.

(b) Expliquez ce qui se passe si nous utilisons des valeurs de k de plus en plus grandes dans l'appel ec(k) .

5. Cet exercice considère la question de l'addition d'entiers consécutifs.

(a) Quelle est la formule pour additionner les n premiers entiers positifs ? (Tu aurais dû voir ça
avant, peut-être en Calcul II dans la section sur les sommes de Riemann.)

(b) Suzy écrit la procédure suivante pour additionner les n premiers entiers positifs.

```
sage : def sum_through(N) :
total = 0
pour i dans la plage (N):
total = total + je
retourner le total
```

Quel est le résultat de son invocation, sum_through(5) ? Indiquez la valeur du total après chaque passage dans la boucle.

(c) Utilisez la notation de sommation pour décrire la valeur que le programme de Suzy calcule réellement. Quoi
serait la formule correspondante ?

(d) Comment Suzy devrait-elle corriger son programme ?

6. Supposons que vous ayez un bébé qui demande[12] à nourrir toutes les 3 heures.

(a) Si vous commencez à la nourrir à 7 heures du matin, à quelle heure l'alimentation aura-t-elle lieu ?

(b) Écrivez une compréhension qui génère une liste avec chacun de ces temps pour les prochaines 24 heures.

(c) Répétez la partie (b), uniquement en supposant que le nourrisson est un peu plus âgé et ne demande que
à nourrir toutes les 5 heures.

Programmation.

1. Écrivez une procédure pour calculer la valeur moyenne des éléments d'une collection. (Tu devrais
rappelez-vous que "valeur moyenne" est un nom fantaisiste pour "moyenne".) Ce sera plus facile avec un
hension à l'intérieur d'un sum() , mais vous pouvez également utiliser une boucle for .

2. Écrivez une procédure pour calculer la valeur médiane des éléments d'une collection. (Tu devrais
rappelez-vous que « valeur médiane » est un nom fantaisiste pour « valeur moyenne » ; c'est-à-dire que la moitié des valeurs sont
plus grand, et la moitié sont plus bas.) La meilleure façon de le faire est probablement de convertir la collection en un
liste, triez-la, puis retournez le nombre au milieu.
3. Ecrire une procédure qui prend une liste de points (x_i, y_i) et renvoie deux listes : la liste des abscisses
et la liste des coordonnées y.
4. Créez une animation qui illustre comment une ligne sécante s'approche d'une tangente. Utiliser la fonction
$f(x) = x^2$, avec la ligne tangente passant par le point x = 2, avec les lignes sécantes qui joignent x = 2
[12] Pourquoi, oui, certains d'entre nous ont une expérience intime avec cela. Qu'est-ce qui te fait demander ?

---

avec 30 points entre x = −1 et x = 2. Rendre le tracé de f noir, avec une épaisseur
de 2 ; colorie la ligne tangente en bleu et colorie les lignes sécantes en rouge. Inclure un point bleu à (2,4)
pour mettre en évidence l'endroit où la courbe et ses lignes tangentes et sécantes se rencontrent. Le résultat devrait être
comparable ou meilleure que l'animation que vous verrez ci-dessous si vous visualisez ce texte dans Acrobat
Lecteur:
5. À la p. 104 nous avons parlé d'une série de Taylor tronquée à 4 termes. Nous n'avions pas de boucles for
disponibles, de sorte que cette approche était à la fois fastidieuse et inflexible.
(a) Adaptez le pseudocode pour que l'utilisateur puisse saisir un nombre arbitraire de termes.
(b) Implémentez le pseudocode en tant que programme Sage.
6. (Ce problème s'adresse aux élèves qui ont suivi le calcul multivariable.) L'intégrale double
$$\iint_D f(x, y)\, dx\, dy$$
apparaît souvent dans le calcul tridimensionnel. Ici, D est un sous-ensemble du plan xy^2, appelé
le domaine de l'intégrale. Lorsque D est une région rectangulaire définie par l'intervalle x (a, b)
et l'intervalle y (c,d), on peut approximer cette intégrale en divisant le domaine en m×n sous-rectangles et évaluer la valeur z de f sur chaque sous-rectangle :
$$\iint_D$$

f (x,y) dx dy

$$\sum_{i=1}^{m} \sum_{j=1}^{m} f\left(x_i^*, y_j^*\right) \Delta x \Delta y$$

où

• m est le nombre de sous-intervalles que nous voulons le long de (a, b) ;

• n est le nombre de sous-intervalles que nous voulons le long de (c,d) ;

• $\hat{o}\, x = \frac{(b-a)}{m}$ est la largeur de chaque sous-intervalle de (a, b);

• $\hat{o}\, y = \frac{(d-c)}{m}$ est la largeur de chaque sous-intervalle de (c,d); et

• x_{je}^*, y_j^* est un point du sous-rectangle défini par le ième sous-intervalle de (a, b) et le jième sous-intervalle de (c,d).

Nous pouvons sélectionner x_{je}^*, y_j^* en utilisant une règle du point médian comme

$x_i^* = a + i - \frac{1}{2} \Delta x$ et $y_j^* = c + j - \frac{1}{2} \Delta y$.

Ecrivez une procédure double_integral_midpoint() qui prend comme arguments f , a, b, c, d, m, et n, et renvoie une approximation de l'intégrale double de f sur (a, b)×(c,d) en utilisant m sous-intervalles le long de (a, b) et n intervalles le long de (c,d).

Astuce : Ce sera très similaire au code que nous avons utilisé pour approximer une seule intégrale, mais

vous voudrez imbriquer une boucle for pour y dans une boucle for pour x. Vous pouvez également le faire avec un
compréhension, mais c'est un peu plus compliqué.

-1
-0,5
0,5
1
1.5
2
1
2
3
4
5

DES EXERCICES
131

7. Soit n un entier positif. La factorielle de n, notée n!, est le produit
$n! = n \times (n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1.$
Sage propose déjà une procédure `.factorial()` , mais dans cet exercice, vous utiliserez des boucles pour le faire,
deux manières différentes.

(a) La commande `product()` de Sage fonctionne de la même manière que la commande `sum()` : elle calcule le
produit de tout ce qui se trouve entre les parenthèses, et vous pouvez utiliser une compréhension pour
spécifier une plage, plutôt que d'en créer une explicitement. Cela correspond à la mathématique
expression
$\prod_{i=1}^{m} F_i)$
qui calcule le produit de tous les f (i) où i = 1,2,...,n. Dans cette notation,
$n! = \prod_{i=1}^{m} i = n \times (n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1.$
Écrivez une procédure appelée `factorial_comprehension()` s qui accepte un argument, n,
construit ensuite le produit à l'aide d'une compréhension.

(b) Une façon plus traditionnelle de calculer la factorielle consiste à initialiser une variable de produit P
à 1 (le « produit vide »), puis effectuez une boucle définie de 1 à n, en multipliant P par chaque numéro en cours de route. Ecrivez une procédure appelée `factorial_loop()` qui en accepte un
argument, n, construit ensuite le produit en utilisant une boucle définie.

8. Écrivez un programme pour calculer la factorielle croissante d'un nombre n sur k pas. La montée
factorielle rf(n,k) est
$rf(n,k) = n \times(n+1)\times\cdots\times(n + k - 1)$.
Bien que k soit toujours un entier non négatif, votre programme devrait fonctionner même si n ne l'est pas. Pour
exemple, $rf(1/2,5)=(1/2)(3/2)(5/2)(7/2)(9/2) = 945/32$.
9. Écrivez un programme pour calculer la factorielle décroissante d'un nombre n sur k pas. La chute
factorielle ff(n,k) est
$ff(n,k) = n \times(n - 1)\times\cdots\times(n -(k - 1))$.
Bien que k soit toujours un entier non négatif, votre programme devrait fonctionner même si n ne l'est pas. Pour
exemple, $ff(9/2,5)=(9/2)(7/2)(5/2)(3/2)(1/2) = 945/32$.
10. Une permutation est une manière d'ordonner des objets distincts. Par exemple, dans la liste (1,2,3) il
sont six permutations : (1,2,3) lui-même, puis (2,1,3), (3,2,1), (3,1,2), (2,3,1), et (3, 2,1). Quelque-
fois vous voulez permuter seulement c des objets dans un ensemble de n. La formule pour cela est
$${}_nP_c = \frac{n!}{c!}$$
.
(Voir l'exercice précédent pour les factorielles.) Il y a deux façons de le faire.
(a) La façon évidente, la « force brute » de le faire est de calculer n ! et c!, puis diviser. Écrivez
une procédure Sage nommée `n_P_c_brute()` , qui accepte les deux arguments n et c, puis
appelle soit `factorial_loop()` soit `factorial_comprehension()` pour déterminer n ! et C!.
(b) Une façon plus intelligente de procéder consiste à utiliser votre caboche. Développer à la main les factorielles dans
${}_nP_c$ pour voir le motif. Transformez la formule mathématique résultante en une procédure Sage
nommé `n_P_c_smarter()` , qui accepte les deux arguments n et c, puis détermine ${}_nP_c$
sans calculer n! ou c!.

DES EXERCICES
132

11. Supposons que $b > 1$ soit un entier. Pour écrire un nombre positif n en base b, on peut répéter le
Etapes suivantes:

• Soit d = $\log_b n + 1$; cela nous indique combien de chiffres le résultat aura.
• Soit L une liste vide.
• Répétez d fois :
– Soit r le reste de n divisé par b.
– Soustraire r de n.
– Remplacez n par la valeur de n lorsqu'il est divisé par b.
– Ajouter r à L.
• Inversez L et retournez le résultat.
Convertissez cette liste informelle d'instructions en pseudocode formel et traduisez le pseudocode
au code Sage.
12. Écrivez une procédure Sage nommée by_degree_then_lc() qui, lorsqu'on lui donne un polynôme f as
input, renvoie un 2-uplet composé du degré du polynôme, suivi de son coef-
efficace. Nous ne vous avons pas dit les commandes Sage pour cela, mais elles fonctionnent comme des méthodes pour un
polynôme; vous pouvez le trouver de la manière suivante :

```
sage : f = 3*x**2 + x + 1
sage : f. <Tab>
```

...où <Tab> indique que vous devez appuyer sur la touche Tab du clavier. Les deux meth-
ods attend l'indéterminé comme argument, qui est une autre méthode pour un polynôme qui vous pouvez trouver de la même manière. Assurez-vous que vous pouvez utiliser les méthodes avec succès avant
écrire le programme.
Testez soigneusement cette procédure. Il devrait produire les résultats suivants :

```
sage : f = 3*x**2 + x + 1
sage : by_degree_then_lc(f)
(2, 3)
sauge : g = 2*t**4 - t**2
sage : by_degree_then_lc(g)
(4, 2)
```

Une fois que cela fonctionne, assurez-vous qu'il trie la liste suivante de polynômes en t
dans le bon ordre. Notez que, puisque nous utilisons t dans cette liste, votre procédure ne peut pas supposer
ce qu'est l'indéterminé, cela devrait donc aussi être un argument de la procédure, bien qu'il puisse
par défaut à x .

```
[ t^2 + t + 1, 2*t - 1, 3*t^2 + 4, -3*t^10 + 1]
```

CHAPITRE 6

Résolution d'équations

Jusqu'à présent, nous vous avons demandé de résoudre des équations à la main, bien qu'il soit logique que nous voudrions

Sage pour les résoudre pour nous. Ce chapitre examine la commande solve() et certains de ses proches,
à la fois sur des équations simples et sur des systèmes d'équations. Cela nous conduit dans des matrices, donc nous regardons
certains d'entre eux aussi.
Il n'est généralement pas possible de décrire la solution exacte de chaque équation en « purement algé-
braic », par lesquels nous entendons l'utilisation de l'arithmétique et des radicaux avec des nombres rationnels. C'est
généralement vrai même lorsque l'équation se compose simplement de polynômes ! Donc, même si nous nous concentrons
principalement sur les méthodes qui trouvent des solutions exactes, Sage propose également des méthodes pour calculer des
solutions aux équations, et nous les examinons brièvement.

Les bases

Pour trouver la solution exacte d'une équation ou d'un système d'équations, l'outil principal dont vous avez besoin est
la commande résoudre() .

résoudre(eq_or_ineq , indet)
résout l'équation simple ou en-
égalité eq_or_ineq pour indet

résoudre(eq_or_ineq_list , indet_list)
résout la collection d'équations
eq_or_ineq_list pour l'indéterminé
nates répertoriés dans la collection en-
liste_det

Alors que eq_or_ineq devrait impliquer une équation ou une inégalité, du point de vue de Sage, nous pouvons utiliser
une expression, une fonction, une équation ou une inégalité.

Une équation ou une inégalité, pour un indéterminé

La commande solve() résoudra bien sûr les problèmes d'algèbre de base du lycée.
Équations.

```
sage : résoudre(x**2 - 1 == 0, x)
[x == -1, x == 1]
```

Il est très fréquent de rencontrer des équations avec 0 d'un côté ; la solution de ce genre d'équation
s'appelle une racine. Nous n'avons pas besoin de spécifier qu'une expression est égale à 0 lorsque nous voulons
résoudre pour une racine ; nous pouvons simplement fournir l'expression, et la commande solve() en déduit que nous
veut ses racines.

sage : résoudre(x**2 - 1, x)
[x == -1, x == 1]
Sage peut également résoudre des équations dont tous les coefficients sont symboliques.
sage : var('ab c')
(a, b, c)
sage : résoudre(a*x**2 + b*x + c == 0, x)
[x == -1/2*(b + sqrt(b^2 - 4*a*c))/a, x == -1/2*(b - sqrt(b^2 -
4*a*c))/a]
L'axe polynomial 2 + bc + x est dit quadratique car son degré est 2.1 Vous pouvez distinguer le formule quadratique dans les deux réponses :
x = -
1
2
×
b + b 2 -4ac
une
=
−b − b 2 −4ac
2a
et
x = -
1
2
×
b − b 2 −4ac
une
=
−b + b 2 −4ac
2a
.
Cette réponse correspond à celle que vous avez apprise à l'école,
x =
−b ± b 2 −4ac
2a
.
Dans les deux cas, le résultat était une liste d'équations. Chaque équation indique une valeur de l'indétermination.
miner qui résoudra l'équation. Parce que le résultat est sous forme de liste, vous pouvez accéder aux solutions
en utilisant des crochets, [] . Si vous voulez manipuler les solutions d'une manière ou d'une autre, il est probablement préférable d'attribuer

la liste à une variable, que nous appellerons généralement par un nom comme sols , puis utiliser sols[i] pour accéder au
avec la solution.

```
sauge : sols = résoudre(x**5 + x**3 + x, x)
sauge : len(sols)
5
sauge : sols[0]
x == -sqrt(1/2*I*sqrt(3) - 1/2)
```

Donc x = − i 3 / 2 − 1 / 2 est l'une des solutions.
Vous voudrez parfois juste la valeur de la solution. Les équations de Sage offrent deux méthodes utiles
ods qui extraient les côtés gauche et droit :

eq_or_ineq .rhs()
côté droit de eq_or_ineq

eq_or_ineq .lhs()
côté gauche de eq_or_ineq

Poursuivant notre exemple précédent,

1 Le degré d'un polynôme dans un seul indéterminé est le plus grand exposant qui apparaît sur cet indéterminé. La méthode .degree() vous le signalera.

```
sage : sols[0].rhs()
-sqrt(1/2*I*sqrt(3) - 1/2)
```

Encore une fois parce que le résultat de solve() est sous forme de liste, vous pouvez également parcourir la liste des solutions
en utilisant une boucle for . Voici une façon dont vous pourriez trouver cela utile. Traçons à la fois un polynôme
et ses racines.

```
sage : f(x) = x^3 - 2*x^2 - 5*x + 6
sauge : sols = résoudre(f, x)
sage : X = [sol.rhs() pour sol en sols]
sage: p = plot(f, min(X) - 1, max(X) + 1, color='black', \
épaisseur=2)
sage : pour xi dans X :
p = p + point((xi,0), taille en points=60, couleur='rouge', \
zordre=5)
sauge : p
```

Alors que vous auriez dû être capable de le faire vous-même, cela implique de tirer beaucoup de choses différentes
idées ensemble, alors décomposons chaque étape.

• Nous avons d'abord défini la procédure f , un polynôme de degré 3.
• Nous avons utilisé solve() pour trouver ses racines et les avons stockées dans la variable sols . Étiez-vous à
coup d'œil à sols , vous verriez ce qui suit :

```
sauge : sols
[x == 3, x == -2, x == 1]
```

• Sage renvoie des solutions sous la forme d'une liste d'équations, et nous voulons juste la valeur de la
racine. Pour ce faire, nous avons affecté à une nouvelle variable, `X` , une liste formée à l'aide d'une compréhension.
L'expression `pour sol in sols` boucle sur la liste stockée dans `sols` et stocke chaque valeur
dans la variable `sol` . L'expression `sol.rhs()` nous donne le côté droit de cette valeur.
Le résultat est que `X` est une liste qui contient le côté droit de chaque solution en `sols` .
Si vous jetiez un coup d'œil à `X` , vous verriez ce qui suit :

```
sage : X
[3, -2, 1]
```

• Nous avons défini un objet graphique `p` comme le tracé de `f` sur l'intervalle `min(X) - 1` et `max(X) + 1` .
Cela donne au graphique un peu d'espace pour respirer au-delà des racines.
• Nous avons effectué une boucle définie sur les solutions stockées dans `X` . La boucle stocke l'entrée de `X`
dans la variable de boucle `xi` , que Sage utilise pour ajouter un point à `p` . Par exemple, le premier
passer par la boucle, il assigne `xi=3` , donc il ajoute le point à (3,0).
Multiplicités. Les expressions sages offrent également une méthode qui nous fournit parfois à la fois
racines et multiplicités.

`f .roots()`
renvoie les racines de f, avec leurs
multiplicités

Les multiplicités, rappelez-vous, nous disent combien de fois une racine apparaît dans la factorisation d'un
polynôme. Par exemple, le polynôme $(x-1)(x+2)^2(x-4)^3$ a trois racines, 1, −2 et 4,
avec des multiplicités respectives 1, 2 et 3.

```
sage : f = (x - 1)*(x + 2)**2*(x - 4)**3
sage: f.roots()
[(-2, 2), (1, 1), (4, 3)]
```

Vous voyez que le résultat est une liste de tuples ; chaque tuple répertorie la racine en premier et sa multiplicité en second.
Les multiplicités sont importantes pour de nombreuses raisons ; dans l'un des exercices de ce chapitre, vous pouvez
examinez la zone autour des racines d'un polynôme et voyez comment la multiplicité affecte la géométrie.
Le plan complexe. De nombreux polynômes ont des racines qui incluent des parties imaginaires. Nous obvi-
ne peut pas représenter graphiquement ces racines avec leurs fonctions sur le plan réel, car il

n'y a pas de place sur le
plan réel pour inclure, par exemple, le nombre i.
Néanmoins, il peut être assez instructif de visualiser les racines sans leurs fonctions.
Quelconque
nombre complexe a la forme a + bi, où a est la partie réelle et b est la partie imaginaire. Nous
tracer ceci comme le point (a, b) sur le plan réel. Cette cartographie attribue un point unique
sur le réel
plan $_2$ à tout nombre complexe z $\in$, ce qui motive le nom de ce modèle, le complexe
avion.
La procédure suivante mappe un nombre complexe à un point du plan réel.

```
sage : def complex_point(z, *args, **kwds) :
point de retour ((real_part(z), imag_part(z)), *args, **kwds)
sage : sum(complex_point(sol.rhs(), pointsize=90) \
pour sol dans résoudre (x**4 + 4*x + 5, x))
```

Que fait ce code?

• Les deux premières lignes définissent une procédure, complex_point() , dont le seul argument obligatoire
est z , un nombre complexe.
– Il crée un point dont la valeur x est la partie réelle de z et dont la valeur y est l'imagi-partie naire de z.
– Pour éviter de reformuler les arguments obligatoires et facultatifs d'un point, complex_point() utilise une astuce spéciale pour accepter les arguments requis et facultatifs pour un point régulier () ,
puis les ignore sauf pour les transmettre à point() .
• Les dernières lignes créent un graphique en utilisant la commande sum() avec une compréhension.
– La compréhension boucle les solutions sur x $_4$ +4x+5 , qu'elle trouve avec le commande résoudre(x**4 +4*x + 5, x) . Chaque solution est stockée dans la variable de boucle sol .
– La boucle envoie à chaque valeur de sol la méthode .rhs() , obtenant en retour le droit-côté de l'équation que Sage utilise pour décrire la solution. La boucle passe que
nombre complexe à complex_point () , ainsi que l'option Point () l' argument
taille en points=90 . Le résultat est un point dans le plan complexe, que la commande sum()
se combine enfin dans l'image que vous voyez.
Résoudre les inégalités. Vous vous souviendrez peut-être que les inégalités entraînent des complications. Commencer avec,
il existe généralement une infinité de solutions qui se situent généralement sur un ou plusieurs intervalles de la
vraie ligne. Ces intervalles sont parfois bornés, parfois non bornés. Par exemple,

• la solution de $x^2 - 1 \geq 0$ est $(-\infty, -1] \cup [1, \infty)$, tandis que
• la solution de $x^2 - 1 < 0$ est $(-1, 1)$.
Cette complication se reflète dans la façon dont Sage décrit les solutions d'une inégalité. Les solutions
à une inégalité sont décrits dans une liste de listes. Chaque liste interne décrit un intervalle qui contient
solutions. Cet intervalle lui-même contient soit une inégalité linéaire, qui représente un intervalle
non borné dans une direction, ou deux inégalités linéaires, qui représentent un intervalle borné en
les deux directions.
Par exemple, il n'est pas difficile de vérifier à la main que l'inégalité$(x-3)(x-1)(x+1)(x+3) \geq 0$ a la solution suivante :
$(-\infty, -3] \cup [-1, 1] \cup [3, \infty)$,
de sorte que x = -5, x = 0 et x = 12 sont des solutions. Ceci est facile à schématiser sur une droite numérique :[2]

-3

-1

1

3

Sur la base de la description que nous avons donnée dans le paragraphe précédent, comment devriez-vous vous attendre à la solution
regarder? Vous savez que c'est une liste de listes, il devrait donc y avoir plusieurs paires de crochets dans une paire de
supports. Il y a trois intervalles, il devrait donc y avoir trois listes internes. Les intervalles les plus éloignés
sont illimitées, donc Sage le décrira en utilisant une seule inégalité linéaire ; l'intervalle du milieu est
borné, donc Sage utilisera deux inégalités linéaires pour le décrire.

```
sage : résoudre((x-3)*(x-1)*(x+1)*(x+3) >= 0, x)
[[x <= -3], [x >= -1, x <= 1], [x >= 3]]
```

[2] Cela ne devrait pas faire de mal de penser à la façon dont vous pourriez schématiser cela dans Sage. Si ça fait mal, ça veut probablement dire
votre cerveau grandit, alors continuez comme ça.

Effectivement, Sage décrit l'intervalle le plus à gauche en utilisant $x \leq -3$; le plus à droite en utilisant x 3, et
le plus interne en utilisant deux intervalles, $x \geq -1$ et $x \leq 1$. C'est aussi ainsi qu'il faut le lire,

accessoirement : « x −1 et x ≤ 1 ». Cela équivaut à l'inégalité −1 ≤ x ≤ 1.
Comme les résultats sont des listes, vous pouvez considérer chaque intervalle par accès par parenthèse ou par itération. le
Les méthodes .rhs() et .lhs() sépareront les côtés gauche et droit d'une inégalité tout comme ils le feront pour une équation.

Erreurs ou surprises qui surviennent lors de la résolution

Certaines équations donneront un résultat étrange. Considérez cette équation générique du 5e degré :

```
sage : résoudre(a*x**5 + b*x**4 + c*x**3 + d*x**2 + e*x + f, x)
[0 == a*x^5 + b*x^4 + c*x^3 + d*x^2 + e*x + f]
```

Sage semble renvoyer la même équation que vous lui avez demandé de résoudre. Si cela vous rappelle le
exemple à la p. 41 où Sage a « refusé » de calculer une intégrale indéfinie, félicitations ! Il est plus ou moins le même phénomène : Sage ne peut pas trouver un moyen d'exprimer la solution par un calcul algébrique
expression sur les coefficients, sauf en vous renvoyant l'équation elle-même. C'est bien connu que nous ne pouvons pas résoudre des équations polynomiales génériques de degré 5 ou supérieur d'une manière aussi « simple »
comme formule quadratique. Cela touche à un sujet appelé solvabilité par radicaux.
N'oubliez pas que, dans Sage, une équation utilise deux signes égaux. Si vous oubliez d'utiliser deux égaux
signes lors de l'utilisation de la commande solve() , des choses désagréables se produiront. Tu étais prévenu. MT

```
sage : résoudre(2*x + 1 = 3*x - 2, x)
Erreur de syntaxe:
le mot-clé ne peut pas être une expression
```

Solutions approximatives d'une équation

Nous avons vu il y a un instant que Sage ne peut pas décrire les racines du polynôme générique du cinquième degré
en termes exacts. Cela reste également vrai pour de nombreux polynômes spécifiques du cinquième degré.

```
sage : résoudre(x**5 - 6*x + 4, x)
[0 == x^5 - 6*x + 4]
```

Il faut donc parfois se contenter de solutions approximatives. Une autre fois pour opter pour une approximation
La solution mate est lorsque la solution exacte est tout simplement trop lourde pour être traitée.

```
sage : résoudre(x**3 + x + 1, x)[0].rhs()
-1/2*(1/18*sqrt(31)*sqrt(3) - 1/2)^(1/3)*(I*sqrt(3) + 1) +
1/6*(-I*sqrt(3) + 1)/(1/18*sqrt(31)*sqrt(3) - 1/2)^(1/3)
```

Aie.
Pour trouver une solution approximative à une équation, nous utilisons quelque chose de différent de solve()
commander.

find_root(eq , a , b)

trouver une solution de eq sur l'intervalle (a, b)

SOLUTIONS APPROXIMATIVES À UNE ÉQUATION
139

Pour l'utiliser, nous avons besoin d'une idée de l'emplacement de la racine, afin que vous puissiez spécifier a et b. Même si
le terme « racine » implique que l'équation doit avoir 0 d'un côté, ce n'est pas strictement nécessaire ;
nous pouvons fournir une équation avec des expressions non nulles des deux côtés, et Sage la résoudra, tout
le même.

```
sage: find_root(x**5 - 6*x == -4, -5, 5)
-1.7000399860584985
```

Notez qu'il ne renvoie qu'une seule racine, même lorsque plusieurs racines existent sur le même intervalle. Lorsque
nous savons que plusieurs racines existent, nous pouvons modifier l'intervalle en conséquence.

```
sage: find_root(x**5 - 6*x == -4, 0, 5)
1.3102349335013999
sage: find_root(x**5 - 6*x == -4, 0, 1.3)
0.693378264514721
```

Si nous spécifions un intervalle où l'équation n'a pas de racines, Sage génère une erreur.

```
sage: find_root(x**5 - 6*x == -4, 1.3, 5)
Erreur d'exécution:
f semble n'avoir aucun zéro sur l'intervalle
```

Une autre approche à essayer lorsque nous ne savons pas quel intervalle utiliser est une deuxième forme de
méthode .roots() , qui nous permet de spécifier un anneau dans lequel chercher des solutions. En particulier,
nous pouvons spécifier quelle bague rechercher.

f .roots(anneau= R)

trouve les racines de f dans l'anneau R, avec
leurs multiplicités

L'anneau par défaut est , l'anneau des rationnels, mais nous pouvons demander à Sage de résoudre les approximations de
racines en dehors des rationnels en spécifiant les anneaux réels ou complexes ; en particulier *RR* et *CC* .

```
sage : f = x**5 - 6*x + 4
sage: f.roots()
Erreur d'exécution:
aucune racine explicite trouvée
sage: f.roots(ring=RR)
[(-1.70003998605850, 1), (0.693378264514721, 1), (1.31023493350140,
1)]
sage: f.roots(ring=CC)
[(-1.70003998605850, 1),
```

(0.693378264514721, 1),
(1.31023493350140, 1),
(-0,151786605978811 - 1,60213970994664*I, 1),
(-0,151786605978811 + 1,60213970994664*I, 1)]

L' approche .roots() n'est pas toujours réussie, même lorsque find_root() l' est. A moins que nous ne soyons
particulièrement attaché à la recherche de multiplicités, find_root() est la méthode de choix.

SYSTÈMES D'ÉQUATIONS
140

```
sage : f = sin(x) + x - 1
sage: f.roots()
Erreur d'exécution:
aucune racine explicite trouvée
sage: f.roots(ring=RR)
Erreur-type:
Impossible d'évaluer l'expression symbolique à une valeur numérique.
sage: find_root(f, 0, 1)
0.510973429388569
```

Systèmes d'équations

De nombreux problèmes du monde réel impliquent plus d'une variable et plus d'une relation entre ces variables. Celles-ci donnent lieu à des équations à plusieurs variables ; quand on essaie de
résoudre plusieurs à la fois, nous l'appelons un système d'équations. Nous pouvons utiliser solve() pour résoudre un
système d'équations dans Sage; il suffit de fournir le système dans une liste ou un tuple, suivi d'une liste ou d'un tuple
des indéterminés des polynômes. La solution d'un système d'équations est similaire à celle d'un
inégalité : Sage renvoie une liste de listes. Chaque liste interne correspond à une solution distincte à la
équation; il contient des équations qui indiquent la solution de chaque variable du système.

```
sage : résoudre((x**2 + y == 1, x - y == 1), (x,y))
[[y == -3, x == -2], [y == 0, x == 1]]
```

Cette équation a deux solutions, correspondant aux points (−2,−3) et (1,0). On peut illustrer la relation géométrique entre les courbes et ces points avec un graphique :

```
sage : p = implicite_plot(x**2 + y == 1, (x, -3, 3), (y, -4, 2), \
couleur='noir')
sage : p = p + implicite_plot(x - y == 1, (x, -3, 3), (y, -4, 2), \
couleur='bleu')
sage: p = p + point((1,0), color='red', pointssize=60, zorder=5)
sage: p = p + point((-2,-3), color='red', pointssize=60, zorder=5)
sauge : p
-3
-2
-1
0
1
2
```

3
-4
-3
-2
-1
0
1
2

MATRICES
141

Matrices

Si le degré de chaque équation est au plus égal à 1, nous les appelons équations linéaires. Donc un système de linéaire

Les équations consistent en un ensemble d'équations que nous essayons de résoudre simultanément. Ce sont juste un spécial

cas des systèmes d'équations.

Les matrices sont intimement liées aux systèmes d'équations linéaires. Tout système d'équation linéaire

tion

$a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n = b_1$

$a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n = b_2$

...

$a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n = b_m$

correspond à une « équation matricielle »

Hache = b

où

A =

$$\begin{pmatrix} un_{11} & un_{12} & \cdots & un_{1n} \\ un_{21} & un_{22} & \cdots & un_{2n} \\ ... & ... & ... & ... \\ un_{m1} & un_{m2} & & \end{pmatrix}$$

··· une minute
$$\begin{pmatrix} \\ \\ \\ \end{pmatrix}$$
, x =
$$\begin{pmatrix} \\ \\ \\ \end{pmatrix}$$
x_1
x_2
...
x_n
$$\begin{pmatrix} \\ \\ \\ \end{pmatrix}$$
, et b =
$$\begin{pmatrix} \\ \\ \\ \end{pmatrix}$$
b_1
b_2
...
b_m
$$\begin{pmatrix} \\ \\ \\ \end{pmatrix}$$
.
Les matrices x et b sont des matrices spéciales en ce qu'elles n'ont qu'une seule colonne ; nous appelons de telles matrices
vecteurs.
Nous supposons que vous connaissez les règles de l'arithmétique matricielle, nous ne les passons donc en revue
ici, mais si vous êtes encore en train d'acquérir votre « pied marin » en matière d'arithmétique matricielle, nous vous suggérons
réfléchir à la façon dont les règles de multiplication matricielle transforment Ax = b en le système d'équations

ci-dessus. L'analyse matricielle est essentielle à la compréhension des systèmes d'équations linéaires, et puisque
de nombreuses approches des équations non linéaires impliquent d'abord de les transformer en un système d'équations linéaires
équations, les matrices occupent une position fondamentale en mathématiques.
Création de matrices, accès aux éléments et modification des propriétés fondamentales. Vous pouvez
créer une matrice dans Sage à l'aide de la commande matrix() . Il y a plusieurs moyens de le faire; nous nous concentrons
sur trois.
matrice(liste_lignes)
crée une matrice à partir de list_of_rows
matrice(anneau , liste_ligne)
crée une matrice avec les entrées de l'anneau en utilisant
liste_de_lignes
La raison de ces deux formes est que Sage considère l'anneau d'une matrice lorsqu'il décide comment effectuer
la plupart des calculs matriciels, il doit donc le savoir. Vous utilisez le premier formulaire si vous vous contentez de laisser
Sage fait une supposition éclairée sur l'anneau. Vous pouvez constater que le choix de Sage ne fonctionne pas très bien
pour toi; dans ce cas, vous pouvez le corriger en douceur en utilisant la méthode pratique .change_ring() :
M .change_ring(ring)
convertit les entrées de M pour résider dans l'anneau et
renvoie le résultat
M .base_ring()
rapporte que l'anneau Sage pense que les éléments de M résident
dans

Gardez à l'esprit que Sage ne changera pas la matrice elle-même ; il produit une nouvelle matrice et la renvoie.
La matrice d'origine reste dans l'ancien anneau. Nous utilisons cette procédure un peu plus bas.
Tout d'abord, nous illustrons la création d'une matrice. Pour spécifier la matrice
M =
1 1
0 1
nous allons lister chacune de ses lignes sous forme de liste à l'intérieur d'une autre liste :
sage : M = matrice([\

```
[1, 1], \
[0, 1] \
])
```
sauge : M
[1 1]
[0 1]

Vous n'êtes pas obligé de mettre chaque ligne sur sa ligne, mais nous pensons que cela facilite la lecture. Vous pouvez
placez toute la matrice sur une seule ligne si vous le souhaitez.

Il est bien entendu possible d'utiliser des compréhensions dans la définition d'une matrice :

sage : matrice([[i, i+1, i+2] pour i dans la plage(3)])
[0 1 2]
[1 2 3]
[2 3 4]

Trois matrices spéciales que vous pouvez créer sont une matrice d'identité et une matrice diagonale.

identité_matrice(n)

renvoie la matrice identité de dimension n

diagonal_matrix(entrées)

renvoie une matrice avec 0 partout sauf le
diagonale principale, et les valeurs des entrées sur la
diagonale principale

zero_matrice(m , n)

renvoie une matrice m × n de zéros

Encore une fois, vous pouvez utiliser une compréhension de liste pour créer des listes d'entrées pour les matrices. Nous démontrons
ceci pour une matrice diagonale :

sage : diagonal_matrix([i**2 for i in range(3)])
[0 0 0]
[0 1 0]
[0 0 4]

Vous accédez aux éléments d'une matrice à l'aide de l'opérateur crochet ; pour accéder à l'élément $M_{i,j}$ utiliser M[i,j] .

Ici, « accès » ne signifie pas simplement « lire ; » cela signifie également "écrire", vous pouvez donc modifier les entrées
d'une matrice de façon pratique. Gardez à l'esprit que, comme pour les listes, la première ligne commence à la position 0,
pas la position 1, vous devrez donc en tenir compte.

---

sage : M[0,1] = 3
sauge : M
[1 3]
[0 1]

Cette capacité à modifier les éléments d'une matrice signifie que Sage considère une matrice

mutable, juste
comme une liste. Il y a des occasions où vous voudrez travailler avec une matrice immuable. Pour
exemple, vous ne pouvez stocker que des objets immuables dans un ensemble, donc si vous voulez un ensemble de matrices, vous avez
pour indiquer à Sage que vous n'avez pas l'intention de changer les matrices en question. Tu peux faire
ceci en utilisant la méthode .set_immutable() .

M .set_immutable()
rend un objet immuable

M .is_mutable()
Vrai si et seulement si l'objet est mutable

M .is_immuable()
Vrai si et seulement si l'objet est immuable

La commande ne fonctionne que dans un sens ; pour rendre une matrice mutable à nouveau, faites-en une copie avec le
commande copier() .

copier(obj)
renvoie une copie de l'objet obj

```
sage : { M }
Erreur-type:
les matrices mutables sont impossibles à hacher
sage : M.set_immutable()
sage : M.is_mutable()
Faux
sage : { M }
{[1 3]
[0 1]}
sage : M = copie(M)
sage : M.is_mutable()
Vrai
```

Les matrices ne sont pas les seuls objets que Sage peut copier ou rendre immuables, ces commandes ont donc
une applicabilité plus large.

Arithmétique matricielle et manipulation. Vous pouvez effectuer une arithmétique matricielle dans Sage en utilisant le
symboles mathématiques usuels.

```
sage: N = matrice([ \
[1 fois], \
[0, 1]])
sauge : M * N
[
1x + 3]
[
0
1]
```

Sage propose également un grand nombre de méthodes pour envoyer une matrice. Nous

n'énumérons que quelques-uns des
les dans le tableau 1. Pour les voir tous, n'oubliez pas que vous pouvez toujours découvrir les méthodes d'un objet
comprendra en tapant son identifiant, suivi du point, puis en appuyant sur la touche Tab . Pour ce qui est de

---

MATRICES
144

M .add_multiple_of_row(j , i , c)
ajoute c fois chaque entrée de la ligne i à la cor-
entrée de réponse de la ligne j

M .characteristic_polynomial()
renvoie le polynôme caractéristique de M

M .déterminant()
renvoie le déterminant de M

M .dimensions()
renvoie un tuple (m,n) où m est le nombre
ber de lignes et n le nombre de colonnes

M .echelon_form()
renvoie la forme échelonnée de M en partant
la matrice elle-même dans sa forme originale

M .échelonner()
transforme M en forme d'échelon ; Retour
Rien

M .valeurs propres()
renvoie les valeurs propres de M

M .eigenvectors_right()
renvoie les vecteurs propres de M

M .inverse()
renvoie l'inverse de M

M .noyau()
renvoie le noyau de M (pour accéder à la base
les vecteurs d'un noyau K utilisent K .basis())

M .ncols()
renvoie le nombre de colonnes dans M

M .nrows()
renvoie le nombre de lignes dans M

M .nullité()
renvoie la dimension du noyau

M .rang()
renvoie le rang de M

M .set_row_to_multiple_of_row(j , i , c)

définit chaque élément de la ligne j au produit de
c et l'entrée correspondante de la ligne i
M .swap_rows(i , j)
échange les lignes i et j de M
M .submatrix(i , j , k , *l*)
renvoie la sous-matrice k× *l* de M dont le sommet
le coin gauche est dans la ligne i, colonne j de M
M .transpose()
renvoie la transposée de M
T ABLEAU 1. méthodes comprises par une matrice de Sage
les méthodes que nous énumérons ici, ne vous inquiétez pas si vous ne comprenez pas le but de chacune, mais chacune
d'entre eux devraient à un moment donné s'avérer utiles dans les études de premier cycle.
Voyons comment certaines de ces commandes peuvent fonctionner, ainsi que certaines des erreurs qui peuvent survenir. Par exemple, supposons que nous souhaitions transformer une matrice en
forme triangulaire supérieure, et aussi en voyant certains des calculs qui se produisent en cours de route. Donc
nous effectuerons ce calcul étape par étape plutôt que de rechercher une commande Sage particulière
ça fait tout à la fois. Notre exemple de matrice sera
A=
⎛
⎜
⎜
⎝
1 2 3 4
0 2 2 3
8 3 1 2
0 1 2 3
⎞
⎟
⎟
⎠
.

Nous vous laissons le soin de mettre en place une matrice pour A dans Sage. Pour le transformer en forme triangulaire supérieure, nous
observez que $A_{0,0}$ est déjà 1, tandis que $A_{1,0}$ et $A_{3,0}$ sont tous les deux 0, nous n'avons donc rien à leur faire. Comme
$A_{2,0} = 8 = 0$, on veut ajouter un multiple de la ligne 1 qui élimine le 8 ; c'est assez simple :
sage : A.add_multiple_of_row(2, 0, -8)

[
1
2
3
4]
[
0
2
2
3]
[
0 -13 -23 -30]
[
0
1
2
3]

Nous passons à la deuxième colonne. On observe que $A_{1,1} = 2 = 1$, il faut donc diviser le rang par 2.

sage : A.set_row_to_multiple_of_row(1, 1, 1/2)
Erreur-type:
Multiplier la ligne par l'élément Rational Field
ne peut pas être fait sur Integer Ring, utilisez change_ring ou
with_row_set_to_multiple_of_row à la place.

PANIQUE!

... eh bien, non, non. Bien sûr, nous avons reçu une erreur, mais celle-ci est très utile. ça le rend parfaitement
clair que le problème est que Sage voit A comme étant au-dessus du ring. Les nombres entiers ne peuvent pas être des fractions, et
si nous multiplions la deuxième ligne par $1/2$, nous obtenons $3/2$ à la dernière place. Nous avons besoin d'Ato mentir sur le rationnel
champ, pas sur l'anneau entier. Très bien; nous avons déjà discuté de la façon de changer l'anneau de
une matrice, alors faisons-le. Il n'est pas nécessaire de conserver l'ancienne matrice A, nous allons donc attribuer le résultat de
.change_ring() en A. Rappelez-vous que le symbole de Sage pour l'anneau des nombres rationnels est QQ .

sage: A = A.change_ring(QQ)
sage : A.set_row_to_multiple_of_row(1, 1, 1/2)
sage : un
[
1
2
3
4]
[
0
1
1 3/2]
[

0 -13 -23 -30]
[
0
1
2
3]
Excellent! Nous pouvons maintenant effacer $A_{2,1}$ et $A_{3,1}$.
sage : A.add_multiple_of_row(2, 1, 13)
sage : A.add_multiple_of_row(3, 1, -1)
[
1
2
3
4]
[
0
1
1
3/2]
[
0
0
-10 -21/2]
[
0
0
1
3/2]

À partir de là, vous devriez être en mesure de terminer le processus par vous-même, pour finir avec la matrice

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ & 1 & 1 & \frac{3}{2} \\ & & 1 & \frac{21}{20} \\ & & & \frac{9}{20} \end{pmatrix}.$$

(Les entrées vides représentent 0.)

Nous nous transformons. Une application des matrices implique l'utilisation de la

transformation
matrices en infographie. Nous discutons de trois types de matrices de transformation :
• Une matrice d'échelle a la forme
$\sigma =$
0
0 s
.
Il a pour effet de redimensionner un point (x,y) au point (sx, sy).
• Une matrice de rotation a la forme
$\rho =$
cos α −sin α
sin α
cos α
.
Il a pour effet de faire tourner un point (x, y) sur l'origine d'un angle α .
• Une matrice de réflexion a la forme
$\varphi =$
cos β
sin β
sin β cos β
.
Il a pour effet de refléter un point (x,y) à travers la ligne passant par l'origine dont la pente est $1-\cos\beta / \sin\beta$.
Nous allons illustrer chacun d'eux, en introduisant également l' objet vectoriel dans Sage.
Nous avons déjà mentionné les vecteurs plus tôt ; il s'agit essentiellement d'une matrice avec une seule colonne. Un vecteur
est similaire à un tuple ou à une liste Sage en ce sens qu'il contient une séquence ordonnée de nombres. Il diffère de
un tuple Sage en ce que nous l'associons à certaines opérations mathématiques. Ce qui nous importe
le plus ici est que vous pouvez multiplier une matrice m × n en un vecteur n × 1 ; le résultat est un m × 1
vecteur. Typiquement, nous utilisons des matrices carrées, et le résultat de la multiplication d'une matrice n×n à un n×1
vector est un autre vecteur n × 1, ce qui est pratique pour une application répétée.
Sage nous permet commodément d'utiliser des vecteurs comme points dans les commandes de traçage, telles que point() ,
line() , et polygon() . Cela permet d'illustrer facilement le fonctionnement des matrices de transformation.
Pour ce faire, nous allons travailler sur un polygone défini par les points suivants, dans cet ordre :
(1,0),

car
4π
5
,péché
4π
5
,
car
8π
5
,péché
8π
5
,
car
2π
5
,péché
2π
5
,
car
6π
5
,péché
6π
5
.
Définissez une liste v dont les éléments sont les vecteurs avec ces valeurs. Tracer le polygone défini par ces
pointes et vous verrez une presque belle étoile à cinq branches :

(Si vous obtenez un pentagone au lieu d'une étoile, assurez-vous que les points sont dans l'ordre ci-dessus. Si
pas, .pop() et .insert() jusqu'à ce qu'ils le soient.) Nous disons "presque sympa" parce que c'est un peu décalé.

Ne serait-il pas agréable[3] pour qu'il pointe vers le haut ?

Nous pouvons résoudre ce problème. Nous avons la technologie. Comment allons-nous nous y prendre ? Nous pourrions le faire pivoter de

1/20 d'une rotation complète, ou par $2\pi/20$, ou par $\pi/10$. Au - dessus, on a prétendu que la matrice de rotation
accomplirait cela, si seulement nous fixions $\alpha = \pi/10$. Essayons cela. Nous allons créer un correspondant
matrice de rotation M , puis utilisez-la pour créer une nouvelle liste, U , dont les points sont les rotations des points
de V par /10. Créer U est facile avec une compréhension de liste :

```
sage : M = matrice([[cos(pi/10), -sin(pi/10)], \
[péché(pi/10), cos(pi/10)]])
sage : U = [M*v pour v dans V]
```

Tracez maintenant le polygone formé par U pour vérifier que nous avons bien redressé notre étoile :[4]

```
sauge : polygone(U)
```

Et la réflexion ? Si nous regardons l'étoile d'origine, nous pourrions la refléter sur l'axe des y, ce qui
a une pente. . .
Hmm. Eh bien, maintenant, c'est ennuyeux. L'axe des y a une pente indéfinie. Nous obtenons généralement sous-
pente amende en divisant par 0. Le dénominateur de la pente de la matrice de réflexion est sin β , de sorte que
si nous avons péché $\beta = 0$, nous devrions finir par réfléchir sur l'axe des y. Cela fonctionne pour $\beta = 0$ et
$\beta = \pi$; nous allons essayer le premier et espérer le meilleur.

3 Si vous faites partie de ces fous qui pensent que cette étoile est très bien comme elle est, et qu'il serait préférable de faire pivoter les axes
à la place, eh bien ! Vous avez un avenir en mathématiques. Si vous n'êtes pas un de ces fous et pensez qu'il vaut mieux
star... ouais, tu as peut-être aussi un avenir en mathématiques. Mais nous gardons un œil supplémentaire sur vous, juste pour nous assurer
vous ne sortez pas de la ligne. Un peu comme cette étoile ici.

4 Ou plutôt, si vous en croyez la note précédente, nous avons fait du tort à notre polygone.

```
sage : N = matrice([[cos(0), sin(0)], \
[sin(0), -cos(0)]])
sage : W = [N*v pour v dans V]
sauge : polygone (W)
```

Eh bien, c'est ennuyeux. Ça n'a pas du tout bougé !
En fait, ça a basculé ; nous ne pouvons tout simplement pas le voir. Tout d'abord, la pente devient en fait $1-\cos 0/\sin 0 = 1-1/0 =$
$0/0$; pour une pente verticale, nous avons besoin d'un non nul divisé par zéro. Un coup d'œil sur N le renforce :

```
sauge : N
[ dix]
[ 0 -1]
```

Si vous pensez au fonctionnement de la multiplication matricielle, l'effet de N sur n'importe quel vecteur sera de
préserver la valeur x et inverser la valeur y. Nous nous sommes donc retrouvés avec un flip vertical, plutôt qu'un
basculement horizontal. On dirait que nous voulons utiliser $\beta = \pi$.

```
sage : N = matrice([[cos(pi), sin(pi)], \
[sin(pi), -cos(pi)]])
sage : W = [N*v pour v dans V]
sauge : polygone (W)
```

Génial!
Pour une matrice de réflexion, les points le long de l'axe de réflexion restent fixes : si l'on plier le `N` corrigé au vecteur (0,2), cela renverrait le vecteur (0,2). C'est un cas particulier de ce que nous appelons un vecteur propre, un vecteur qui reste sur la même ligne après multiplication par un
matrice. Chaque matrice a ses propres vecteurs propres, et nous pouvons les « découvrir » en utilisant la méthode
`.eigenvectors_right()` . Dans ce cas:

```
sage : N.eigenvectors_right()
[(-1, [(1, 0)], 1), (1, [(0, 1)], 1)]
```

C'est beaucoup d'informations, alors voyons ce que cela nous dit.
La liste contient un tuple par vecteur propre. Chaque tuple contient trois éléments :
• Premièrement, cela nous donne ce qu'on appelle une valeur propre. Les valeurs propres vous indiquent comment la taille du
changements vectoriels; il se trouvera toujours sur la même ligne passant par l'origine, mais la taille sera
monnaie. La relation de base entre une matrice M, un vecteur propre e, et le correspondant aux valeurs propres λ est que Me = λ e. Vous pouvez également extraire les valeurs propres d'une matrice en utilisant
la méthode `.eigenvalues()` .
• Deuxièmement, il donne une liste de vecteurs propres qui correspondent à cette valeur propre. Dans la plupart des cas,
vous vous attendriez à ne voir qu'un seul vecteur propre par valeur propre, mais il peut arriver que vous
obtenir plus d'un.
• Enfin, il donne la multiplicité de la valeur propre. Cela renvoie à l'idée suivante :
les valeurs propres sont les racines λ_1 , λ_2 , ..., λ_m d'un polynôme appelé le minimum de la matrice
polynôme. Ce polynôme se factorise linéairement comme $(x - \lambda_1)^{a_1}(x - \lambda_2)^{a_2} \cdots (x - \lambda_m)^{a_m}$. Chaque
la puissance d'un facteur linéaire distinct est la multiplicité de la valeur propre

correspondante. Nous sommes
pas intéressé par cela pour l'application actuelle.
Vous espérez généralement que les vecteurs propres e_1 , ..., e_m soient linéairement indépendants ; c'est-à-dire le seul
façon d'écrire
$a_1 e_1 + \cdots + a_m e_m = 0$
est si $a_1 = \cdots = a_m = 0$. Nous pouvons le voir dans les vecteurs propres de N , puisque
$$un_1 \begin{matrix}1\\0\end{matrix} + a_2 \begin{matrix}0\\1\end{matrix} = 0$$
$$un_1 \quad un_2 = 0 \Rightarrow un_1 = 0, un_2 = 0.$$
Malheureusement, toutes les matrices de dimension n n'ont pas n valeurs propres distinctes ; si vous regardez le
matrice de mise à l'échelle
$$S = \begin{matrix}3 & 0\\0 & 3\end{matrix},$$
vous constaterez qu'il n'a qu'une seule valeur propre de multiplicité 2, car il traite chaque point exactement
de la même façon.
Comme nous l'avons mentionné, chaque vecteur propre reste sur la même ligne passant par l'origine lorsque nous
lui appliquer la matrice. En termes de points, cela signifie que les vecteurs propres définissent une ligne de points qui
peut se déplacer sur cette ligne, soit en inversant la direction, soit en changeant de taille, mais les points
restent sur la même ligne à travers l'origine. Dans le cas de la matrice de réflexion N , les vecteurs propres
sont $(1,0)^T$, qui se trouve sur l'axe des x, et $(0,1)^T$, qui se trouve sur l'axe des y. La valeur propre pour $(1,0)^T$
est -1 ; cela signifie que (1,0) se déplace vers (-1,0), qui se trouve toujours sur l'axe des x. La valeur propre pour
$(0,1)^T$ est 1 ; cela signifie que (0,1) passe à (0,1), qui se trouve toujours sur l'axe des y. Ce n'est pas vrai pour
points sur d'autres lignes passant par l'origine ; le point (3,5), par exemple, se déplace vers le

point (−3,5),
qui se trouve sur une ligne complètement différente à travers l'origine. Bien sûr, nous nous attendons à ce que N
est une matrice de réflexion.
Pour conclure cette enquête, considérons une matrice différente,
M =
3 1
2 2
.

DES EXERCICES
150
Utilisez Sage pour vérifier que ses vecteurs propres sont
1
1
et
1
-2
,
avec les valeurs propres correspondantes 4 et 1. Cela signifie que nous attendons des points sur la même ligne que
(1,1) T à rester sur cette ligne, mais à redimensionner d'un facteur 4, tandis que les points sur la même ligne que
(1,−2) T restera au même endroit :
reste sur place
reste sur place
(1,1) → (4,4)
(− 3
4
, − 3
4) → (− 3, − 3)
Comme vous pouvez l'imaginer, les choses deviennent un peu étranges avec une matrice de rotation, car elle se déplace
chaque point du plan à l'exception de l'origine à une ligne différente passant par l'origine. Dans ce cas, vous
trouverait que ses vecteurs propres ont des valeurs complexes. Essayez-le !
Des exercices
Vrai faux. Si la déclaration est fausse, remplacez-la par une déclaration vraie.
1. Sage a des fonctionnalités pour trouver des solutions exactes et approximatives aux équations.
2. Sage peut trouver la solution exacte à n'importe quelle équation.

c'est qu'ils sont plus faciles à regarder.
3. La commande `solve()` trouve à la fois les solutions exactes et approximatives d'une équation.
4. La méthode `.roots()` n'est pas aussi efficace que la commande `find_root()` , même lorsque nous spécifions
un anneau d'approximations de nombres, comme `RR` ou `CC` .
5. La multiplicité d'une racine n'est pas liée à la géométrie de la courbe.
6. Les résultats de la commande `solve()` sont dans un format similaire pour les inégalités et les systèmes de
équations.
7. Sage décide parfois comment résoudre un problème en fonction de l'anneau dans lequel se trouvent ses valeurs.
8. Vous devez toujours spécifier l'anneau d'une matrice lors de sa création.
9. Vous pouvez extraire les valeurs propres des résultats de `eigenvectors_right()` et `eigenvectors_left()` commandes.
10. Les valeurs propres n'ont qu'une importance théorique.
Choix multiple.
1. Laquelle des commandes et méthodes suivantes ne produit pas de solutions exactes par défaut ?
A. `résoudre()`
B. `.roots()`
C. `.valeurs propres()`
D. `find_root()`
2. Le résultat de `solve()` lorsqu'on lui donne une équation dans une variable est :
A. la valeur d'une solution approchée
B. une liste de listes de solutions et de multiplicités
C. une liste d'équations linéaires décrivant les solutions

D. une liste de listes d'équations linéaires, dont chacune décrit une solution
3. Le résultat de `solve()` lorsqu'on lui donne plusieurs équations dans plus d'une variable est :
A. la valeur d'une solution approchée
B. une liste de listes de solutions et de multiplicités
C. une liste d'équations linéaires décrivant les solutions
D. une liste de listes d'équations linéaires, dont chacune décrit une solution
4. Le résultat de `find_root()` lorsqu'on lui donne une équation dans une variable est :
A. la valeur de la solution approximative
B. une liste de listes de solutions et de multiplicités
C. une liste d'équations linéaires décrivant les solutions
D. une liste de listes d'équations linéaires, dont chacune décrit une solution
5. Le résultat de `.roots()` lorsqu'il est appliqué à une équation dans une variable est :

A. la valeur de la solution approximative
B. une liste de listes de solutions et de multiplicités
C. une liste d'équations linéaires décrivant les solutions
D. une liste de listes d'équations linéaires, dont chacune décrit une solution
6. Laquelle des commandes et méthodes suivantes nécessite que vous spécifiiez les points de terminaison d'un
intervalle qui contient une solution?
A. résoudre()
B. find_root()
C. .roots()
D. aucune de ces réponses
7. Laquelle des méthodes suivantes d'une matrice renvoie ses valeurs propres ?
A. valeurs propres()
B. vecteurs propres_right()
C. tout ce qui précède
D. aucune de ces réponses
8. Les vecteurs v_1 , v_2 ,..., v_n sont linéairement indépendants lorsque :
A. On peut trouver une matrice M dont les vecteurs propres sont v_1 ,v_2 ,...,v_n .
B. L'ensemble {v_1 ,v_2 ,...,v_n } définit un espace vectoriel.
C. La seule façon pour a_1 contre $_1$ +a_2 contre $_2$ +···+a_n v_n = 0 est si a_1 = a_2 = ··· = a_n = 0.
D. Les vecteurs déclarent leur indépendance par rapport à un espace vectoriel et créent leur propre sous-espace.
9. Pourquoi devrions-nous nous attendre à ce qu'une matrice de réflexion ait deux vecteurs propres linéairement indépendants ?
A. Chaque matrice bidimensionnelle a deux vecteurs propres linéairement indépendants.
B. Une réflexion déplace toujours un point vers un deuxième point, et c'est ainsi que nous comptons le nombre
ber de vecteurs propres linéairement indépendants.
C. Une réflexion est comme un miroir, nous devons donc nous attendre à ce que le deuxième vecteur propre reflète le
premier.
D. Les points de deux droites passant par l'origine restent sur ces droites : l'axe de symétrie, et la ligne qui lui est perpendiculaire.
10. Pourquoi s'attend-on à ce que les valeurs propres d'une matrice de rotation aient des valeurs complexes ?
A. Une matrice de rotation déplace des points sur le plan réel vers des points sur le plan complexe, et
vice versa.
B. Le seul point sur le plan réel qui reste sur la même ligne passant par l'origine est le origine elle-même.
C. Les valeurs propres sont toujours complexes ; nous ne le remarquons tout simplement pas lorsque leurs valeurs sont réelles.

D. Des problèmes complexes nécessitent des solutions complexes.
Programmation.
1. Utilisez solve() pour trouver les solutions de l'équation polynomiale générique de degré trois et quatre.
tions. Combien y en a-t-il? (Ne comptez pas à la main ! Utilisez Sage pour compter les solutions !)
2. La commande solve() ne renvoie pas les solutions d'une seule équation dans une variable en ordre de la norme complexe. Écrivez une procédure qui accepte une équation comme argument, appelle
solve() pour le résoudre, puis trie les solutions en utilisant la norme. Astuce : Vous pouvez utiliser ou
adapter la procédure clé by_norm() que nous avons définie p. 113.
4. Écrivez une procédure qui accepte une inégalité comme argument, la résout, puis produit un graphe
de ses solutions sur la droite numérique, semblable à celle de la p. 137.
5. (a) Écrivez le pseudocode pour une procédure qui accepte deux fonctions f et g, résout leur intersections, puis calcule l'aire totale entre f et g. (On trouve la surface totale par
addition de la valeur absolue des intégrales pour chaque intervalle défini par les intersections de f
et g.)
(b) Implémentez votre pseudocode en tant que procédure Sage qui utilise la commande roots() pour trouver
solutions exactes dans .
(c) Écrivez une procédure interactive qui permet à l'utilisateur de spécifier les fonctions f et g, comme
ainsi qu'une couleur. Il appelle ensuite le code Sage que vous avez écrit dans la partie (b) pour déterminer la superficie totale
entre f et g, les graphiques f et g en noir avec une épaisseur de 2, remplit la zone entre eux en utilisant la couleur spécifiée par l'utilisateur, et enfin écrit la zone au-dessus des courbes.
6. Nous avons utilisé la méthode .roots() pour trouver les racines et les multiplicités de
$f(x)=(x-1)(x+2)^2(x-4)^3$.
Créez un graphique de f sur un intervalle qui inclut les trois racines, mais n'est pas assez grand pour perdre le
des détails sur la façon dont f serpente autour d'eux. Faites ce tracé noir, d'épaisseur 1. Ajoutez-y trois
autres tracés de f , dont chacune des valeurs minimale et maximale de x est au voisinage de une racine différente. Faites ces tracés en rouge, d'épaisseur 3. Maintenant que vous avez ce tracé, décrivez un

similitude géométrique entre une racine de multiplicité m et le graphique de x ^{m} .
7. Bien que Sage ait une commande `implicite_plot()` , il lui manque une commande `implicite_diff()` pour effectuer
forme une différenciation implicite. (Vous voudrez peut-être revoir la définition de la différence implicite-
dans votre livre de calcul.) Écrivez une procédure pour ce faire, en mettant en œuvre les étapes suivantes :
(a) Déplacez tout dans `f` d'un côté de l'équation. Assurez-vous que `f` est une fonction en fonction de `x`
et `y` .
(b) Définissez `yf` comme une fonction implicite de x en utilisant la commande, `yf=function('yf')(x)` .
(c) Remplacez `y` dans `f` par `yf` . Astuce : Si vous avez défini `f` comme une fonction dans `x` et `y` , alors c'est une question de
redéfinir `f` en fonction de `x` et `yf` .
(d) Soit `df` la dérivée de `f` .
(e) Résoudre `df` pour `diff(yf)` .
(f) Renvoie la solution — pas la liste des équations, mais le membre de droite. Pour en-position, le résultat de `implicite_diff(y==cos(x*y))` devrait être `-sin(x*yf(x))*yf(x)/(x*sin(x*yf(x)) + 1)` .

CHAPITRE 7

La prise de décision

Comme mentionné précédemment, l'une des questions fondamentales des mathématiques est,
Comment trouver une racine[1] d'un polynôme ?
Comme cela est plus ou moins peu pratique dans de nombreux cas, nous passons à la question connexe,
Comment peut-on approximer la racine d'un polynôme ?
Il y a plusieurs façons de le faire, mais nous décrirons un moyen d'« approximer » la racine "exactement." Ce n'est pas vraiment la contradiction que cela puisse paraître, car notre approximation consiste en une
intervalle précis (a, b) de nombres rationnels dans lequel le polynôme a au moins une racine. Alors le
valeur sera une approximation, dans la mesure où nous ne savons pas quel c $\in$ (a, b) est la racine, mais c'est un
approximation exacte, en ce que la solution est exempte de toute erreur.[2] En bonus, la technique que nous allons
regarde à volonté :
• travailler pour n'importe quelle fonction continue, pas seulement des polynômes ; et
• en de rares occasions,[3] donne nous la valeur exacte !
Cette technique est la méthode de bissection, et elle est basée sur un fait important de Calculus :

T HE I NTERMEDIATE V ALEUR T HEOREM . Si une fonction f est continue sur l'intervalle [a, b],
alors pour toute valeur y entre f (a) et f (b), nous pouvons trouver ac (a, b) tel que f (c) est cette valeur y.
Par exemple, f (x) = cos x est continu partout, il est donc certainement continu sur le intervalle [$\pi/6$, $\pi/3$]. Maintenant, f ($\pi/6$) = $3/2$, tandis que f ($\pi/3$) = $une/2$ et $2/2$ se situe entre $une/2$ et $3/2$, de sorte que
il doit y avoir un c ∈ [$\pi/6$, $\pi/3$] tel que f (c) = $2/2$.
La méthode de la bissection
Notre exemple avec cos x ne semblera pas impressionnant ; après tout, vous saviez déjà que f ($\pi/4$) = $2/2$.
Assez juste, mais considérez ce scénario :
si
– f (a) est positif et
– f (b) est négatif
ensuite
– 0 se situe entre f (a) et f (b), donc
– une racine c doit être comprise entre a et b.
Par exemple, supposons que nous voulions identifier la racine de
f (x) = cos x − x.
1 Au cas où vous l'auriez oublié, une « racine » est la valeur d'une expression qui, lorsqu'elle est substituée dans l'expression, donne
0.
2 En revanche, les réponses à virgule flottante sont une approximation précise, plutôt qu'une approximation exacte, dans la mesure où
virgule flottante nous donne un nombre qui est légèrement faux.
3 Très rares, mais ils existent.

F IGURE 1. La méthode de bissection utilise le théorème des valeurs intermédiaires pour réduire l'intervalle contenant une racine de (0,1) jusqu'à ($1/2$, $3/4$)... et plus loin !
Si vous le tracez dans Sage, le graphique suggère qu'une racine se situe entre x = 0 et x = 1. Mais pouvons-nous faire confiance
le graphique? En effet, nous pouvons ; comme vous l'avez appris en calcul,
• cos x est continu ;
• x est un polynôme, et tous les polynômes sont continus ; et
• les sommes, les différences et les produits de fonctions continues sont continus.
Donc f est continue, et le théorème des valeurs intermédiaires confirme l'inspection visuelle.
Bien sûr, savoir qu'il y a une racine entre 0 et 1 ne crée pas vraiment beaucoup de confiance. Nous aimerions localiser cette racine plus précisément que cela. Comment?
Dans la méthode de bissection, nous avons coupé l'intervalle [a, b] en deux parties. Dans ce

cas, on le coupe
dans [0, 1/2] et [1/2 ,1]. Nous testons ensuite le nouveau point final dans la fonction : f (1/2) > 0. Nous avons déjà
savait f (0) > 0, donc le théorème des valeurs intermédiaires ne peut pas garantir qu'une racine se situe entre
0 et 1/2 . Par contre, f (1) < 0 ; puisque 0 est compris entre f (1/2) et f (1) , l'Intermédiaire
Le théorème des valeurs garantit qu'une racine se situe entre 1/2 et 1. Nous remplaçons donc [0,1] par [1/2 ,1].
Un intervalle de taille 1/2 est toujours pas très impressionnant, mais il n'y a aucune raison que nous ne pouvons pas continuer.
Coupez à nouveau l'intervalle en deux parties, [1/2 , 3/4] et [3/4 ,1]. Le nouveau point final satisfait f (3/4) < 0,
la racine doit donc se situer entre 1/2 et 3/4 . La figure 1 illustre chacune de ces trois premières étapes de la
traiter.
Nous pouvons persister dans cela aussi longtemps que nous le voulons. A chaque fois, l'incertitude diminue de 1/2 , donc
répéter cela 20 fois nous donne un intervalle de largeur 1/2 20 1,0×10 -6 , soit 0,000001. C'est un peu
difficile à faire à la main, mais c'est simple à répéter pour un ordinateur ; nous avons déjà vu comment
pour faire ça.
Mais comment devrions-nous dire à l'ordinateur de décider quel point de terminaison remplacer par le point médian ?
Entrez la structure de contrôle if/else if/else :

```
si condition1
que faire si condition1 est vraie
sinon si condition2
que faire si condition2 est vrai
...
autre
que faire si aucune des deux conditions n'est vraie
```

Cela dirige l'exécution en fonction des conditions énumérées, permettant au calcul de continuer
d'une manière adaptée à la situation.

Toutes les parties de la structure ne sont pas nécessaires. Seul un si doit apparaître ; après tout, sinon ça ne fait pas
sens sans si. Mais vous pouvez donner un sens à un s'il manque un autre ; c'est souvent utile si

un
une valeur particulière a besoin d'un "massage" supplémentaire avant que l'algorithme puisse continuer.
Dans notre cas, nous voulons que l'algorithme remplace le point final par le même signe que le point médian.
Cela implique le pseudo-code suivant :
algorithme Method_of_Bisection
contributions
• a, b
• f , une fonction telle que :
– f est continue sur [a, b]
– f (a) et f (b) ont des signes opposés
• n, le nombre de bissections à effectuer
les sorties
• [c,d] ⊆ [a, b] tel que
– $d - c = 1/2^n (b - a)$, et
– une racine de f se trouve dans [c,d]
fais
soit c = a et d = b
répéter n fois
soit $e = 1/2 (c + d)$
si f (c) et f (e) ont le même signe
remplacer c par e
autre
remplacer d par e
retourner [c,d]
Malheureusement, cela ne suffit pas encore à implémenter dans Sage, car une question demeure :
Comment décide-t-on si f (c) et f (e) ont le même signe ?
Nous considérons deux manières.
Une manière intelligente à laquelle vous avez peut-être pensé est basée sur l'observation pré-algèbre que deux
les nombres réels ont le même signe si et seulement si leur produit est positif :
$(-2)\times(-2) > 0$ et $2\times2 > 0$
mais
$(-2)\times2 < 0$ et $2\times(-2) < 0$.
On peut donc remplacer le test
si f (c) et f (e) ont le même signe
avec
si f (c) f (e) > 0

et en finir avec ça. Ceci est assez facile à implémenter en tant que code Sage, car Sage propose des mots-clés qui
correspondent presque à l'identique à notre pseudocode : if , elif et else . Comme on pouvait s'y attendre, le com-
Les commandes que vous souhaitez que Sage exécute dans chaque cas doivent apparaître en dessous d'elles, en retrait. Donc notre
pseudocode se traduit comme suit.

```
sage : def method_of_bisection(a, b, n, f, x=x):
c, d = a, b
f(x) = f
pour i dans la plage (n):
# calcule le point médian, puis compare
e = (c + d)/2
si f(c)*f(e) > 0 :
c = e
autre:
d = e
retour (c, d)
```

Remarquez les deux points ! Si vous oubliez accidentellement l'un des deux-points, Python générera une erreur.

```
sage : ...
si f(c)*f(e) > 0
c = e
...
Erreur de syntaxe:
Syntaxe invalide
```

Une fois que vous l'avez tapé correctement, vérifions que cela fonctionne.

```
sage : method_of_bisection (0, 1, 20, cos(x) - x)
(387493/524288, 774987/1048576)
```

Mais il y a encore un moyen plus excellent.

Logique booléenne

Jusqu'à présent, nous avons utilisé les identificateurs Sage *True* et *False* sans trop de commentaires ; après tout,
leurs significations sont assez évidentes à moins que vous ne les ayez réattribuées. Pourtant, ces deux idées se situent au
fondement du calcul le plus pratique. Un champ appelé logique booléenne considère les voies logiques
nous pouvons effectuer à partir des deux concepts de base de Vrai et Faux.[4]

4 Remarquez la distinction entre les identificateurs Sage des termes à sens fixe (*Vrai* et *Faux*) et les constantes mentales de la logique booléenne (Vrai et Faux).

Quatre opérations « fondamentales ». Les quatre opérations fondamentales de la logique booléenne sont

les opérateurs ou, et, xor,[5] et non.[6] Ils sont régis par les tables de « vérité » suivantes, qui indiquent comment les variables avec les valeurs True ou False se comportent sous les opérations données.

X
pas x
Vrai faux
Faux vrai

X
oui
x ou y
Vrai Vrai Vrai
Vrai Faux Vrai
Faux Vrai Vrai
Faux Faux Faux

X
oui
x et y
Vrai vrai
Vrai
Vrai faux
Faux
Faux vrai
Faux
Faux Faux
Faux

X
oui
x x ou y
Vrai vrai
Faux
Vrai faux
Vrai
Faux vrai
Vrai
Faux Faux
Faux

Après avoir précisé le sens de ces termes, nous nous empressons de signaler qu'ils devraient vraiment
être intuitif pour vous ; la seule vraie différence entre les significations précises et votre intuitif
compréhensions apparaît dans ou et xor. Nous ne trouverons généralement pas xor nécessaire dans ce texte, mais

il a ses utilités.

Opérateurs booléens dans Sage. Ces concepts se traduisent tous immédiatement en Sage :

True , *False* ,

pas , ou , et , et xor .

Avec leur aide, nous pouvons maintenant réécrire notre code Sage avec une logique booléenne au lieu d'un calcul mathématique.

astuce mathématique. Vérifier que deux nombres réels c et e ont le même signe est une question de test

cette

(f (c) ⩾ 0 et f (e) ⩾ 0) ou (f (c) ⩽ 0 et f (e) ⩽ 0)

et cela se traduit directement dans le code Sage

(f(c) > 0 et f(e) > 0) ou (f(c) < 0 et f(e) < 0)

5 Raccourci pour exclusif ou, dont le nom vient du fait qu'en général, on peut soit couper le gâteau soit l'avoir, mais pas les deux. Formulé légèrement différemment, vous pouvez avoir exclusivement une option ou l'autre. En quoi cela diffère-t-il de l'ancien ou ordinaire ? La règle ne s'applique pas aux anniversaires, car vous pouvez couper votre gâteau ou l'avoir ou les deux. (Se plaindre que vos parents n'ont pas acheté assez de cadeaux est aussi une option, mais ne nous laissons pas emporter loin.) Étant donné que vous pouvez avoir les deux, régulier vieux ou est considéré comme un « inclus » ou. Vous pourriez vous demander pourquoi nous raccourcissons exclusif ou xor. C'est parce que les informaticiens, en particulier les informaticiens ingénieurs, sont violemment allergiques à l'écriture de mots de plus de trois ou quatre lettres. (Vraiment. tu penses Je fait ce haut? — Bon, d'accord, on exagère un peu, mais seulement un peu.)

6 Ceci est un exemple du mensonge pur et simple que nous avons promis dans la préface. Comme promis, ça sonne beaucoup mieux que la vérité. Le mensonge consiste à appeler les opérations « fondamentales », car la liste contient au moins un élément redondant opération, rendant au moins l'un d'entre eux non « fondamental ». En particulier, nous pouvons réécrire a xor b comme [a et (pas b)] ou [(pas a) et b]. On peut donc déjà réduire la liste à ou, et, et non.

Mais ce n'est pas tout! Nous pouvons également réécrire a et b comme non [(pas a) ou (pas b)]. (Ce dernier fait est connu sous le nom de DeMorgan's Lois.) Nous pouvons donc en fait réduire la liste des opérations logiques à ou non. S'il est normal que le nombre de les opérations fondamentales sur deux objets (Vrai et Faux) devraient numéro deux (pas et ou), personne dans son bon sens est ce que ca. Il est bien mieux de vous mentir et de vous dire que les quatre opérations sont « fondamentales ».

Vous vous demandez peut-être si les parenthèses sont réellement nécessaires ici. Dans ce cas, non, car Sage teste toujours et avant de tester ou . En général, cependant, il est de bonne pratique d'indiquer explicitement l'ordre des opérations, car il est très important que vous entendiez

sage : False et False ou True

Vrai

sage : False et (False ou True)

Faux
sage : (Faux et Faux) ou Vrai
Vrai
Assurez-vous de comprendre pourquoi nous obtenons ces résultats.
Nous pouvons maintenant énoncer un code Sage différent pour la méthode de bissection.

```
sage : def method_of_bisection(a, b, n, f, x=x):
c, d = a, b
f(x) = f
pour i dans la plage (n):
# calcule le point médian, puis compare
e = (c + d)/2
si (f(c) > 0 et f(e) > 0) ou (f(c) < 0 et f(e) < 0) :
c = e
autre
d = e
retour (c, d)
```

Une autre relation booléenne, et une mise en garde sur son utilisation. Jusqu'à présent, nous nous sommes appuyés sur le
Relations booléennes `==` , `<` , `>` , `<=` et `>=` . Vous avez peut-être remarqué que nous n'avons pas répertorié d'opérateur
pour l'inégalité. Techniquement, on peut traduire le pseudocode
a = b
comme

```
sage : non (a == b)
```

mais ce n'est pas très élégant. Sage nous offre un symbole différent dans ces circonstances, `!=` , qui
ressemble en quelque sorte à une façon bâclée de couper à travers un signe égal, puis de laisser tomber de l'encre.
L'égalité et l'inégalité peuvent être dangereuses lors des comparaisons. nombre exact-
Les bers ne posent pas de problème, mais si votre valeur a une virgule flottante, alors l'ordinateur peut par erreur
pense que deux nombres sont différents alors qu'ils ne le sont pas. Par example:

```
sauge : 14035706228479900.
- 14035706228479899.99 != 0
Faux
```

C'est une conclusion très étrange à faire, pour laquelle on peut remercier l'arrondi de la virgule flottante
Nombres. D'autre part, rappelons p. 109 le résultat de la méthode d'Euler, que nous avons conclu
avait 8 ans.

```
sage : 8 - 7.99999999999999 != 0
Vrai
```

Imaginez un algorithme qui ne se termine que s'il a réellement calculé 0. Nous avons probablement

l'égalité ici, mais l'ordinateur ne la détecte pas encore (voire pas du tout).
La solution dans des cas comme celui-ci est de tester que la différence entre les valeurs est suffisamment
petits, plutôt que s'ils sont égaux. Par exemple, si se situer à moins de dix chiffres décimaux suffit,
le test suivant corrigerait le précédent.

```
sauge : 8 - 7.99999999999999 > 10**(-10)
Faux
```

Briser une boucle

Il y aura des occasions où il ne servira à rien de continuer une boucle. Un exemple serait
se produire si nous appliquons la méthode de bissection à une fonction dont la racine est un nombre rationnel avec
une puissance de 2 au dénominateur. Par exemple, f (x) = 4x + 3 a une racine en x = $-3/4$. Même si
vous demandez à la méthode de bissection de répéter 20 fois, selon toute vraisemblance, elle finira loin, très loin de l'oreille-
menteur. Supposons, par exemple, que nous commençons sur l'intervalle (−1,1). La première bissection nous donne
intervalle (−1, $-1/2$), tandis que le second nous donne l'intervalle (−1, $-3/4$). À ce stade, nous avons déjà
trouvé une racine f ($-3/4$) = 0, donc nous devrions vraiment quitter, mais l'algorithme n'inclut pas de moyen de
teste ça. Au lieu de cela, il continue pendant 18 itérations supplémentaires, se terminant par le ($-393217/524288$, $-3/4$).

Une meilleure approche serait de reformuler la boucle pour qu'elle s'arrête au moment où elle trouve un
racine. Le test est simple dans Sage, mais comment lui dire d'arrêter la boucle ? Une manière serait de placer une instruction return dans la boucle, mais ce n'est pas une bonne idée si un algorithme est
à la recherche d'une valeur afin qu'il puisse ensuite effectuer un calcul avec elle avant de revenir.

C'est là qu'intervient le mot-clé break . Une instruction break indique à Sage d'arrêter le
boucle et procédez à partir de la première instruction en dehors du bloc indenté de la boucle. Une pause s'applique
une seule fois, donc si la boucle est imbriquée dans une autre boucle, Sage n'en sortira pas.

Nous pouvons maintenant reformuler notre procédure method_of_bisection() comme suit.

```
sage : def method_of_bisection(a, b, n, f, x=x):
c, d = a, b
f(x) = f
pour i dans la plage (n):
```

```
# calcule le point médian, puis compare
e = (c + d)/2
si (f(c) > 0 et f(e) > 0) ou (f(c) < 0 et f(e) < 0) :
c = e
elif f(e) == 0 :
c = d = e
Pause
autre:
d = e
retour (c, d)
```

Cette fois, lorsque le code calcule `e == -3/4` , il détermine que

• la condition de la première instruction `if` est fausse, car ni `f(e) > 0` ni `f(e) < 0` ; mais

• la condition pour la seconde instruction `if` est vraie, comme `f(e) == 0` .

Le code doit donc affecter $-3/4$ à `c` et `d` , puis exécuter l' instruction `break` . La première l'instruction en dehors du bloc indenté de la boucle est `return (c, d)` , donc le code renverra le tuple

$(-3/4, -3/4)$.

```
sage : method_of_bisection(-1, 1, 20, 4*x + 3)
(-3/4, -3/4)
```

Exceptions

Les exceptions sont l'un des concepts les plus récents de la programmation informatique, et elles sont liées

à la prise de décision, ne serait-ce que parce qu'elles peuvent remplacer les instructions `if` / `else` dans de nombreuses situations. Nous

discutez-en brièvement ici.

Supposons que vous ayez un algorithme qui doit diviser :

soit c = n / d

Par exemple, vous avez un algorithme de division pour certains objets mathématiques. Cette anodine

énoncé masque un danger mathématique : et si d = 0 ?

Nous pourrions bien sûr garder la déclaration avec une sentinelle if/else :

si d = 0

soit c = n / d

autre

faire quelque chose d'intelligent à propos du cas d = 0 - gronder l'utilisateur, disons

---

Bien entendu, le code peut contenir d'autres variables avec leurs propres potentiels de valeurs dangereuses,

menant au code d'apparence moche suivant :

si condition1

si condition2

si condition3

si condition4

...
sinon si condition5
...
autre
...
sinon si condition6
...
autre
...
Ce n'est pas seulement laid, c'est difficile à lire. Les informaticiens ont un nom technique pour cela : le
Pyramide du Destin.[7]
En pseudocode, on essaie de lister les problèmes avant qu'ils ne surviennent, en précisant dans les entrées
section qui, par exemple, d = 0. Par exemple, un pseudocode qui s'attend à prendre une dérivée à un
point arbitraire indiquerait probablement quelque chose comme ceci:
contributions
• f , une fonction dérivable partout sur la droite réelle
. . . mais nous ne le faisons pas tout le temps, surtout quand cela devrait être évident d'après la recherche de l'algorithme.
pose. Après tout, c'est du pseudocode, qui est censé être lisible. Si nous spécifions chaque contrainte,
la section des entrées deviendrait une fastidieuse récitation de pédanteries. Les mathématiciens comptent souvent
sur l'intuition du lecteur.
Nous n'avons pas cette sortie dans le code du programme réel, nous adoptons donc une approche différente, appelée
gestion des exceptions. La gestion des exceptions repose sur ce que nous appelons un bloc `try` / `except` . Comme le nom
suggère, il essaie un bloc de code qui est en retrait sous l' instruction `try` , et si une erreur
se produit, il applique un bloc de code différent qui apparaît sous une instruction `except` . Les deux `essaient`
et `except` sont des mots-clés, vous ne pouvez donc pas les utiliser comme identifiants.
L'utilisation précise est la suivante :
[7] Honnêtement, je n'invente pas ça.

---

sage : essayez :
first_try_statement
second_try_statement
...

```
sauf ExceptionList :
first_exception_statement
seconde_exception_déclaration
```

Lorsque Sage rencontre un tel bloc de code, il essaie `first_try_statement` , `second_try_statement` , ... S'il peut effectuer toutes les tâches répertoriées sous l' instruction `try` , il ignore le bloc `except` entièrement, et continue.

Cependant, si Sage rencontre une erreur, il la compare aux exceptions répertoriées dans `ExceptionList` .

Si l'un d'eux correspond, il exécute `first_exception_statement` , `second_exception_statement` , et ainsi de suite, jusqu'à l'achèvement - à moins qu'il ne rencontre une autre erreur. Dans ce cas, Sage abandonne et renvoie l'erreur au client.[8]

Voici un exemple que vous pouvez tester sans même écrire de procédure :

```
sage : essayez :
1/0
sauf ZeroDivisionError :
Infini
```

Si vous tapez ceci dans une cellule ou sur la ligne de commande, puis exécutez-le, Sage essaie d'abord de diviser 1 par 0.

Cela ne fonctionnera évidemment pas, et Sage lève une *ZeroDivisionError* . Notre clause `except` attrape ceci erreur, donc Sage passe dans ce bloc, trouvant l'instruction *Infinity* . Vous ne devriez pas être surpris, alors, que le résultat est :[9]

8 Si vous ne connaissez pas le type d'erreur, ou si vous souhaitez traiter toutes les erreurs possibles avec le même code, ou si vous vous sentez particulièrement paresseux, il est en fait possible de détecter toutes les erreurs en omettant le `ExceptionType` . Les auteurs de ce texte se sentent particulièrement paresseux la plupart du temps, mais pour des raisons pédagogiques, nous pensons qu'il vaut mieux éviter cela.

Vous pouvez parfois souhaiter traiter des informations provenant de l'exception. Dans ce cas, vous pouvez utiliser la construction

```
sauf ExceptionType comme e :
```

puis regardez les détails de `e` . Nous n'entrons pas dans ces possibilités.

9 Il est également possible d'« imbriquer » `essayer ... sauf` clauses ; c'est-à-dire, placez un `essai ... sauf` clause dans un autre. Tu peux faire ceci dans une clause `try` ou `except` . Par exemple:

```
sage : essayez :
1/0
sauf ZeroDivisionError :
essayer:
0**Infini
sauf NotImplementedError :
print('Pas de dés')
```

```
+Infini
```

Plutôt que de traiter toutes les exceptions dans un seul bloc `except` , il est possible de traiter les

exceptions
à un bloc d' essai dans plusieurs sauf les blocs qui suivent. Alignez chaque clause except avec l' essai
clause, répertoriez l'erreur précise pour chaque exception et fournissez le code en retrait en conséquence. CA aide
continuez à essayer / sauf que les structures ne se transforment en une pyramide du destin moderne.
En plus de détecter les erreurs, une procédure peut invoquer un autre mot-clé, raise , pour « relever »
erreurs qui lui sont propres. L'utilisation est la suivante :

```
sage : augmenter le type d'erreur (message)
```

Pour message , vous pouvez utiliser n'importe quelle chaîne. Pour ErrorType , on peut lever n'importe quelle exception de l'appropri-
type mangé; les deux suspects habituels sont :
• *TypeError* , où l'entrée est du mauvais type : par exemple, l'algorithme attend un matrice, mais a reçu un numéro; et
• *ValueError* , où l'entrée a le bon type, mais la mauvaise valeur : par exemple, le l'algorithme attend un nombre différent de zéro, mais a reçu zéro.
Nous n'utiliserons pas le mot-clé raise en général, mais vous devez savoir que ce mécanisme est
comment les auteurs de Sage vous communiquent les erreurs.
Essayons cette approche sur quelque chose de plus substantiel.
Un exemple "normal". Un exercice p. 104 vous a demandé d'écrire une procédure Sage pour calculer
la droite normale à une fonction mathématique f en x = a. La procédure que vous avez écrite devrait fonctionner
dans la plupart des situations.

```
sage : var('t')
t
sage : ligne_normale(t**2, 1, t)
1/2*t + 1/2
```

Selon toute vraisemblance, cependant, votre code rencontrerait l'erreur suivante :

```
sage : ligne_normale((t - 1)**2 + 2, 1, t)
ZeroDivisionError :
Division symbolique par zéro
```

Eh bien, bien sûr, cela se produirait:
• la normale est perpendiculaire à la tangente ;
• une droite perpendiculaire est une réciproque négative ; et
• la pente de la droite tangente à $(t-1)^2+2$ à t = 1 est 0.
« Reciprocal » signifie « division » et la division par zéro signifie *ZeroDivisionError* .
Ce problème est inévitable. Ce dont nous avons besoin, c'est d'un moyen d'y faire face, et en l'occurrence, nous avons
au moins deux à notre disposition. Mais nous devons d'abord réfléchir au problème sous-jacent ; à savoir,

la dérivée est 0. Dans ce cas, la ligne normale est en fait la ligne verticale x = a. nous illustrons
ceci avec la fonction f (t)=(t −1) 2 +2 à t = 1 :

EXCEPTIONS
164
-1
1
2
3
1
2
3
4
5
6
Une ligne verticale n'est pas une fonction, donc une option consiste à lever notre propre exception. Quelque chose
comme ça, peut-être ?
raise ValueError('La ligne normale est verticale.')
Cela fonctionne, et *ValueError* semble l'exception appropriée, puisque le type est très bien (une fonction
tion, et une fonction différentiable, pour démarrer) mais sa valeur est inappropriée.
Le problème avec cette approche est qu'elle signale une erreur lorsque la situation est vraiment en train de se sauver.
pouvoir! Après tout, une ligne normale existe ; ce n'est tout simplement pas une fonction. Une approche plus appropriée pourrait
être de retourner une équation pour la ligne normale.
renvoie x==a
Maintenant, cela peut rendre votre algorithme d'origine incohérent si vous retournez une fonction (qui,
très probablement, vous l'avez fait). Nous pouvons toujours rendre l'algorithme cohérent en modifiant le
cas pour retourner l'équation y == m(x - a) + f(a) au lieu du membre de droite seul.
Alors, comment implémenter cela dans Sage ? Encore une fois, nous avons deux options. La première consiste à utiliser un if / else
structure comme une « garde » autour du calcul de m . De cette façon, aucune exception ne se produit.
sage : def normal_line_with_guard(f, a, x=x):
f(x) = f
df(x) = diff(f, x)
si df(a) == 0 :
# horizontal perpendiculaire à vertical
résultat = x == a
autre:
m = 1/df(a)
résultat = y == m*(x - a) + f(a)
résultat de retour

L'autre approche consiste à utiliser une structure try / except pour intercepter une *ZeroDivisionError* .

EXCEPTIONS
165

```
sage : def normal_line_with_catch(f, a, x=x):
f(x) = f
df(x) = diff(f, x)
essayer:
m = 1/df(a)
résultat = y == m*(x - a) + f(a)
sauf ZeroDivisionError :
# horizontal perpendiculaire à vertical
résultat = x == a
résultat de retour
```

Essayez-les tous les deux pour voir que les deux procédures fonctionnent avec les lignes normales verticales et obliques :

```
sage: normal_line_with_guard((t - 1)**2 + 2, 0, t)
y == -1/2*t + 3
sage : ligne_normale_avec_garde((t - 1)**2 + 2, 1, t)
t == 1
sage : normal_line_with_catch((t - 1)**2 + 2, 0, t)
y == -1/2*t + 3
sage : normal_line_with_catch((t - 1)**2 + 2, 1, t)
t == 1
```

Quelle approche est « la meilleure ? » La deuxième version est généralement recommandée pour deux raisons :

• La logique coule mieux, elle est donc plus lisible. La construction if / else ne le rend pas obvious pourquoi nous testons df(a)==0 , juste que nous le testons. Cela oblige le lecteur à réfléchir davantage à
ce que nous faisons. En revanche, la construction try / except indique clairement qu'une erreur pourrait se produire (la seule raison d'utiliser un try); si oui, quelle est cette erreur (*ZeroDivisionError*);
et de plus, que faire au cas où cela se produirait (le bloc except).

• Rappelons que l'interface de Sage utilise Python. Au fur et à mesure que les langages informatiques disparaissent, les exceptions de Python
sont efficaces. Il y a une petite pénalité à utiliser des exceptions ; dans nos tests, la garde if/else est environ 20% plus rapide que try / except catch lorsque la division par zéro se produit réellement.

Ne sautez pas encore à cette conclusion ! La division par zéro se produit rarement, nous devrions donc
nous demander comment les approches se comparent dans les circonstances habituelles. Il se trouve que
le try / except catch est plus de 30% plus rapide que le if / else guard !

Un exemple plus impliqué. Dans. 281 nous décrivons la méthode de Dodgson pour calculer une détermi-
nant. La méthode de Dodgson est un algorithme que vous pouvez implémenter à l'aide d'une

boucle for, de sorte que votre instructeur
l'a peut-être déjà attribué. C'est un algorithme mignon ; si vous ne l'avez pas encore essayé à la main, essayez-le maintenant
vraiment rapide.
Malheureusement, il a des problèmes. Ces problèmes sont difficiles à résoudre. Un autre algorithme à com-
pute déterminants est basé sur l'élimination gaussienne. Il a aussi des problèmes, mais ceux-ci sont faciles à
réparer. Nous commençons par le pseudocode ci-dessous, qui commence à compter les lignes et les matrices à partir de 1.

---

algorithme Gaussian_determinant
contributions
• M, une matrice n × n
les sorties
• detM
fais
soit N une copie de M
soit d = 1
pour i $\{1,...,n-1\}$
pour j $\in \{i+1,...,n\}$
si $N_{j,i} = 0$
soit a = $N_{j,i}$
multiplier la ligne j par $N_{i,i}$
soustraire une fois la ligne i de la ligne j
diviser d par $N_{i,i}$
retour $d \times N_{1,1} \times N_{2,2} \times \cdots \times N_{n,n}$

Avant de le mettre en œuvre, voyons-le en action sur

$$M = \begin{pmatrix} 3 & 2 & 1 \\ 5 & 4 & 3 \\ 6 & 3 & 4 \end{pmatrix}.$$

L'algorithme commence en copiant M dans N et en définissant d = 1. Il parcourt ensuite tout sauf le dernier
ligne de N, stockant la valeur de ligne dans i.
Avec i = 1, les boucles externes trouvent une boucle interne avec j = 2. Comme $N_{2,1} = 5 = 0$,

l'algorithme
stocke 5 dans a, multiplie la ligne 2 par $N_{i,i} = 3$, soustrait a = 5 fois la ligne 1 de la ligne 2, puis divise
d par $N_{i,i} = 3$, obtenant $d = 1 \div 3 = 1/3$. La matrice résultante est
N =
⎛
⎝
3 2 1
0 2 4
6 3 4
⎞
⎠
.
La boucle interne définit ensuite j = 3. Comme $N_{3,1} = 6 = 0$, l'algorithme exécute le bloc le plus interne
à nouveau sur la ligne 3, obtenant
N =
⎛
⎝
3 2 1
0 2 4
0 −3 6
⎞
⎠
.
Il a également modifié d, de sorte que $d = 1/9$. Jusqu'ici tout va bien. L'algorithme a terminé la boucle interne
sur j quand i = 1.
La boucle externe passe à i = 2, et trouve une boucle interne avec j = 3. Comme $N_{3,2} = -3 = 0$, le
l'algorithme stocke -3 dans a, multiplie la ligne 3 par $N_{i,i} = 2$, soustrait a = -3 fois la ligne 2 de la ligne 3,

---

puis divise d par $N_{i,i} = 2$, obtenant $d = (1/9) \div 2 = 1/18$. La matrice résultante est
N =
⎛
⎝
3 2 1
0 2 4
0 0 24
⎞
⎠
.
L'algorithme a terminé la boucle interne sur j lorsque i = 2. C'était la dernière valeur de i, donc

le
les boucles sont terminées ; l'algorithme renvoie
$d \times N_{1,1} \times N_{2,2} \times N_{3,3} = 1/18 \times (3 \times 2 \times 24) = 8$.
Soit Sage, soit une méthode plus traditionnelle confirmera que detM = 8.
Nous pouvons implémenter cela dans Sage dans le code suivant :

```
sage : def gaussian_determinant(M) :
N = copie(M)
d = 1
n = N.nrows()
pour i dans la plage (n-1) :
pour j dans la plage(i+1,n) :
sinon (N[j,i] == 0) :
# effacer la colonne sous cette ligne
d = d/N[i,i]
a = N[j,i]
N.set_row_to_multiple_of_row(j,j,N[i,i])
N.add_multiple_of_row(j,i,-a)
renvoie d*prod(N[i,i] pour i dans range(n))
```

Si nous l'essayons sur les matrices restantes données dans l'exercice p. 281, on voit que ça marche pour
la troisième matrice, mais pas la seconde. Qu'est-ce qui ne va pas ?

```
ZeroDivisionError :
Division rationnelle par zéro
```

La division n'a lieu qu'à un seul endroit de l'algorithme : `d` par `N[i,i]` . Nous devons avoir rencontré un 0
sur la diagonale principale.
Comment est-ce arrivé? Nous avons commencé avec
M =
⎛
⎜
⎜
⎝
1 −4 1
2
-1
4 4
1
3
3 3
4
2
5 2 -1
5
⎞
⎟
⎟
⎠
,

et le premier passage dans la boucle extérieure nous laisse avec
N =
⎛
⎜
⎜
⎝
1 −4 1
2
0
0 5
3
0 15 0 −2
0 13 0 −5
5
⎞
⎟
⎟
⎠
.
C'est parti ! Lors du deuxième passage dans la boucle, l'algorithme divise d par 0, un désastre.

Contrairement à la méthode de Dodgson, une solution simple est disponible. Rappelez-vous que l'élimination gaussienne
nous permet d'échanger des lignes, car cela réorganise simplement les équations qui correspondent à chaque ligne. Si
nous pouvons trouver une autre ligne en dessous de i = 2 avec un élément non nul dans la colonne 2, nous pouvons échanger cela
rang avec le rang 2 et procédez comme avant. Dans l'exemple ci-dessus, la ligne 3 a un élément non nul dans
colonne 2, donc nous l'échangeons et la ligne 2, obtenant
N =
⎛
⎜
⎜
⎝
1 −4 1
2
0 15 0 −2
0
0 5
3
0 13 0 −5
5

|
|
.
Avec un élément non nul dans la ligne 1, colonne 1, nous pouvons maintenant reprendre l'algorithme. Quand j = 3
il n'y a rien à faire, donc la seule boucle interne qui fait quelque chose est quand j = 4, obtenant
N =
⎛
|
|
⎝
1 −4 1
2
0 15 0
-2
0
0 5
3
0
0 0 −49
5
|
|
⎞
et d = 1/15 . Rien ne se passe non plus sur la valeur suivante de i, donc l'algorithme renvoie
1
15
×[1×15×5×(−49)] = −245.
Encore une fois, nous pouvons vérifier en utilisant Sage ou un algorithme traditionnel de calcul des déterminants qui detM
est. . . . euh, 245.
Quelle? Nous avons −245 !

PANIQUE!

Une fois que nous avons fini de paniquer, réfléchissons à ce qui a pu changer, en particulier, ce qui pourrait
ont fait que le signe tourne mal.
La seule chose qui a changé, c'est que nous avons échangé des lignes. Vous devez vous souvenir de vos expériences passées.
rien avec les matrices que l'échange de lignes change le signe du déterminant. Là! Notre méthode
de « sauver » l'algorithme impliquait d'échanger des lignes, nous devons donc nous assurer

que nous multiplions
le déterminant par -1. Ce n'est pas trop dur ; il suffit de multiplier `d` par -1.
Avant d'implémenter cet algorithme amélioré, nous devons nous demander : et si nous n'avions pas
été en mesure de trouver un élément non nul dans la colonne 2 parmi les lignes en dessous de la ligne 2 ? Dans ce cas, le
La structure de la matrice nous dit que le déterminant est 0. Cela couvre toutes nos bases.
Nous pourrions utiliser une structure de contrôle if/else pour implémenter l'algorithme modifié, et cela
fonctionnerait très bien (voir les exercices). Nous utilisons un `essai` / `sauf` cette fois, en partie pour illustrer comment
Ça marche. Cependant, plutôt que de mettre tout ce matériel dans la même procédure, séparons-nous
tâches dans leurs procédures distinctes, dont chacune gère un sous-problème « de petite taille ».
Une procédure intitulée `clear_column()` pourrait gérer la tâche ordinaire de réduction de lignes. Tous-
une chose impliquant la sauvegarde d'une division zéro pourrait entrer dans une procédure intitulée, `unzero()` .

EXCEPTIONS
169

```
sage: def unzero(M, d, i):
# utiliser pour échanger les lignes lorsque M[i,i] = 0
n = M.nrows()
# recherchez la ligne kw/non nulle dans la colonne j
pour k dans la plage(i+1,n) :
sinon (M[k,i] == 0) :
M.swap_rows(k,i)
d = -d
Pause
# M[i,i] == 0?
colonne claire ("échec")
si M[i,i] == 0 :
succès = Faux
autre:
d = d/M[i,i]
succès = vrai
succès de retour, M, d
```

La procédure `clear_column()` pourrait alors gérer la tâche simplifiée d'effacer le reste éléments de la colonne.

```
sage : def clear_column(M, d, i):
n = M.nrows()
# effacer la colonne sous cette ligne
pour j dans la plage(i+1, n) :
sinon (M[j,i] == 0) :
essayer:
d = d/M[i,i]
```

```
sauf ZeroDivisionError :
# recherchez le pivot dans la rangée inférieure
succès, M, d = unzéro(M, d, i)
sinon succès :
augmenter ZeroDivisionError('Le résultat est 0')
a = M[j,i]
M.set_row_to_multiple_of_row(j, j, M[i,i])
M.add_multiple_of_row(j, i, -a)
retourner M, d
```

Cela nous permet de simplifier considérablement la procédure `gaussian_determinant()` , ce qui la rend loin
plus compréhensible.

```
sage : def gaussian_determinant(M) :
N = copie(M)
d = 1
n = N.nrows()
# colonnes claires de gauche à droite
pour i dans la plage (n-1) :
essayer:
N, d = clear_column(N, d, i)
sauf ZeroDivisionError :
retourner 0
renvoie d*prod(N[i,i] pour i dans range(n))
```

La procédure renvoie maintenant le déterminant correct pour toute matrice que vous lui lancez.

```
sage : M = matrice([[1,-4,1,2],[-1,4,4,1],[3,3,3,4],[2,5,2,-1]] )
sage : gaussien_determinant(M)
```

Des exercices

Vrai faux. Si la déclaration est fausse, remplacez-la par une déclaration vraie.

1. Sage propose la structure de contrôle `if` / `else if` / `else` pour prendre des décisions.
2. Sage soulève un *SyntaxError* si vous oubliez d'inclure les deux points à la fin d'une `si` ou `autre` state-
ment.
3. La logique booléenne est basée sur les trois valeurs Vrai, Faux et Parfois.
4. Vrai et (pas (Faux ou Vrai))
5. Vrai ou (pas (Faux et Vrai))
6. Vrai et (pas (Faux x ou Vrai))
7. Vrai ou (pas (Faux x ou Vrai))
8. Vous devriez toujours utiliser l'inclusif ou parce que la discrimination est toujours et partout mauvaise.
9. Si un algorithme peut prendre trois actions possibles, la seule façon de l'implémenter est avec
la structure suivante :

```
si condition1 :
bloc1
```

autre:
si *condition2* :
bloc2
autre:
bloc3

10. Un `try` / `except` catch est moins efficace qu'un `if` / `else` guard, mais cette pénalité en vaut la peine pour le
souci de lisibilité.

Choix multiple.

1. La meilleure structure de contrôle pour une décision générale utilise quel(s) mot(s) clé(s) ?

A. `essayer` / `sauf`

B. `déf`

C. `si` / `elif` / `sinon`

D. `pour` / `pause`

2. Une instruction booléenne dans Sage peut avoir quelles valeurs ?

A. *Activé* , *Désactivé*

B. *vrai* , *faux*

C. *Vrai* , *Faux*

D. `1` , `-1`

3. Que signifie « imbriquer » des structures de contrôle ?

A. Placer une structure de contrôle (comme une instruction `if`) dans une structure de contrôle différente
(comme une déclaration `pour`).

B. Placer une structure de contrôle (comme une instruction `if`) dans la même structure de contrôle
(une autre instruction `if`).

C. Programmer de telle manière que les structures de contrôle ne soient pas nécessaires.

D. Pour construire deux structures de contrôle une maison agréable et confortable où ils peuvent élever une famille
des structures de contrôle.

4. Qu'est-ce qui indique les commandes à exécuter lorsque la condition d'une instruction `if` est vraie ?

A. deux points à la fin de la ligne

B. indentation des commandes

C. accolades `{` et `}`

D. à la fois un deux-points et une indentation

5. Quel mot clé indique qu'une boucle doit se terminer plus tôt parce qu'elle est effectivement terminée ?

A. `avorter`

B. `pause`

C. sortie

D. retour

6. Quel symbole indique à Sage que deux expressions ne sont pas égales ?

A. =/=

B. !=

C. #

D. Aucun ; vous devez utiliser non (a == b)

7. Prendre la racine carrée d'un nombre négatif produit quel genre d'exception ?

A. *ZeroDivisionError*

B. *TypeError*

C. *Erreur de valeur*

D. Pourquoi devrait-il produire une exception ?

8. La meilleure structure de contrôle pour surveiller une erreur spécifique utilise quel(s) mot(s) clé(s) ?

A. essayer / sauf

B. déf

C. si / elif / sinon

D. pour / pause

9. Lequel des énoncés suivants n'est pas exécuté dans une instruction try ... except ?

A. Sage exécute chaque instruction mise en retrait sous l' instruction try qui ne soulève pas de Erreur.

B. Si aucune erreur ne se produit dans les instructions répertoriées sous le bloc try , Sage ignore les instructions
en retrait sous le bloc except .

C. Si une erreur se produit dans le bloc try et que l' instruction except suivante répertorie cette erreur, ou aucune
du tout, Sage exécute chaque instruction mise en retrait sous l' instruction except qui ne soulever une erreur.

D. Si une erreur se produit dans les instructions répertoriées sous le bloc except et le suivant except
l'instruction répertorie cette erreur, Sage exécute chaque instruction en retrait sous la suivante, sauf
énoncé qui ne soulève pas d'erreur.

10. Traditionnellement, qu'entend-on par la « pyramide du destin ? »

A. une situation où nous devons imbriquer très profondément les instructions try

B. une situation où un si a beaucoup, beaucoup d'autres si

C. une situation où nous devons imbriquer si les déclarations très profondément

D. encore une autre suite de la série Indiana Jones

11. Une bonne raison d'utiliser des parenthèses lorsqu'une condition a plus d'un opérateur

booléen est
cette:
UNE. . . . il clarifie la logique de la condition à quiconque le lit.
B. . . . cela évite les bugs subtils dus à l'ordre que Sage évalue et , ou , et non .
C. tout ce qui précède
D. aucune de ces réponses
Réponse courte.
1. Comparez et contrastez les moments appropriés pour utiliser if / elif / else par opposition à l'appropri-
J'ai souvent utilisé try / except / raise .
2. Comment modifieriez-vous le pseudocode Method_of_Bisection pour tester que f (a) et f (b) ont
signes différents avant de commencer la boucle, et en retournant (a, b) s'ils ne le font pas ?
3. Si quelqu'un devait modifier l'implémentation de method_of_bisection() pour soulever une exception
tion dans le cas f (a) et f (b) étaient le même signe :
(a) Quel type d'exception devraient-ils utiliser, *TypeError* , *ValueError* , ou autre chose alto-ensemble ?
(b) Quel type de structure devraient-ils utiliser, if / else ou try / except ?
4. Ecrivez une procédure abmatrix() qui accepte en entrée un entier positif n et deux objets a et b, renvoie alors la matrice n × n C où C est la matrice ci-dessous.
n colonnes
n lignes
⎡
⎢
⎢
⎢
⎣
{
}}
{
⎛
⎜
⎜
⎜
⎜
⎜
⎝
abab
baba ···
abab
baba
...
...
⎤
⎥

|
|
|
|
)
Puisque a et b peuvent être symboliques, vous devez créer cette matrice sur l'anneau symbolique.
(a) Calculer detC.
(b) Pourquoi ce résultat a-t-il un sens ? Astuce : Pensez à l'indépendance linéaire, ou plutôt à la
l'absence de.
Programmation.

---

1. Écrivez une procédure qui calcule le plus grand élément d'une liste. Sage a une procédure `max` intégrée ,
mais utilisez une boucle ici.
2. Rédigez des procédures Sage qui :
(a) renvoie le plus grand élément d'une matrice ;
(b) renvoie le plus petit élément d'une matrice;
(c) renvoie la somme de tous les éléments d'une matrice.
3. Ecrire un pseudocode pour une procédure Sage qui accepte en entrée une fonction mathématique f et a
nombre réel a, puis renvoie un tuple avec deux valeurs : si f augmente à x = a (Vrai si so) et si f est concave vers le haut à x = a (Vrai si oui).
4. Ecrire une procédure `lp()` qui accepte comme arguments deux listes `L` et `M` , vérifie qu'elles sont les
même longueur, et si c'est le cas, renvoie une liste `N` avec le même nombre d'éléments, où `N[i]` est le
produit de `L[i]` et `M[i]` . Si les deux listes ne sont pas de la même longueur, la procédure devrait `soulever` une
AttributeError . Par exemple, le résultat de `LP([1,3,5],[2,4,6])` serait `[2,12,30]` , tandis que l'invocation `LP([1,3],[2,4,6])` lèverait l'erreur.
5. Lorsque vous demandez à Sage de résoudre une fonction trigonométrique, il ne renvoie qu'une seule solution. Pour en-
position, la solution de $\sin 2x = 1/2$ est en fait $\{ \pi/12 + \pi k\} \cup \{ 5\pi/12 + \pi k\}$. Écrire une procédure à
ajouter des solutions périodiques à une équation trigonométrique de la forme sin(ax + b) = 0.
Astuce : Une partie de votre programme devra résoudre ax + b = 0. Vous pouvez obtenir cette information
en utilisant la méthode `.operands()` . Pour voir comment cela fonctionne, attribuez `f(x) = sin(2*x + 3)`

puis
tapez `f.operands()` .
6. Dans ce problème, vous allez écrire une procédure interactive qui approche des intégrales définies dans
l'une des trois façons.
(a) Adapter le pseudocode Left_Riemann_approximation pour écrire le pseudocode pour Riemann
approximation en utilisant les extrémités droites. Traduisez cela en code Sage sans utiliser com-
préhensions. Vérifiez que votre code produit des approximations précises.
(b) Adapter le pseudocode Left_Riemann_approximation pour écrire le pseudocode pour Riemann
approximation à l'aide de points médians. Traduisez cela en code Sage sans utiliser de compréhension
sions. Vérifiez que votre code produit des approximations précises.
(c) Écrivez une procédure interactive avec les objets d'interface suivants :
• une boîte de saisie pour une fonction f (x) ;
• une zone de saisie pour une extrémité gauche a ;
• une zone de saisie pour une extrémité droite b ;
• un curseur pour un nombre n d'approximations, avec valeur minimale 10, valeur maximale 200 et pas de 10 ; et enfin,
• un sélecteur avec les options « approximation à gauche », « approximation à droite » et « milieu
approximation."
Le corps de la procédure appellera la procédure Sage que vous avez écrite pour l'application demandée.
proximité et renvoie sa valeur.
7. Typiquement, une utilisatrice de la méthode de bissection ne saurait pas à l'avance combien d'étapes elle
veut que la boucle principale de l'algorithme s'exécute. Ce qu'elle a en tête c'est la précision décimale
nécessaires entre les points de terminaison : par exemple, ils doivent être les mêmes jusqu'au millième
endroit. Une façon de mettre en œuvre ceci est de remplacer l'entrée n par un entier positif d qui
spécifie le nombre de chiffres. Nous pouvons alors calculer n à partir de d dans le corps de l'algorithme.
(a) Modifiez le pseudocode Method_of_Bisection afin qu'il accepte d au lieu de n et calcule n comme sa toute première étape.
Astuce : En général, nous arrondissons le $\log_{10} n$ à l'entier le plus élevé suivant pour trouver le nombre de chiffres

DES EXERCICES
174
auberge. ($\log_{10} 2 \approx 0 \to 1$, $\log_{10} 9 \approx 1 \to 1$, $\log_{10} 10 = 1$ 1, ...) Donc si l'algorithme divise par 10, vous pouvez le répéter d fois pour obtenir une précision de d chiffres. Hélas! l'algorithme
divise par 2, vous devez donc utiliser un journal $_2$. Avec quelle précision vous devez l'utiliser, nous laissons comme tâche
pour vous, mais vous devrez également considérer b −a, pas seulement d.
(b) Implémenter le pseudocode dans le code Sage.
8. Ecrire des procédures Sage qui acceptent en entrée une matrice M et renvoient *Vrai* si M satisfait l'un des
ces conditions, et *Faux dans le* cas contraire :
• is_square() : M est carré
• is_zero_one_negative() : chaque entrée de M est 0, 1 ou −1
• sums_to_one() : chaque ligne et chaque colonne de M somme à 1
Écrivez ensuite une procédure Sage qui accepte en entrée une matrice M et renvoie *True* si les trois
les conditions sont remplies, et *False dans le* cas contraire. Rendre cette procédure modulaire, afin qu'elle invoque
les trois procédures précédentes plutôt que de répéter leur code !

CHAPITRE 8

Se répéter indéfiniment

Nous parlions plus haut de se répéter « définitivement », ce qui se produit quand on sait exactement comment
plusieurs fois, nous devons répéter une tâche. Nos exemples pour cela impliquaient la méthode d'Euler pour app-
rapprocher la solution d'une équation différentielle, les sommes de Riemann pour approcher une intégrale, et
vérifier si les éléments d'un anneau fini forment réellement un corps. Plus tard, nous avons utilisé les mêmes idées
trouver les racines d'une fonction continue via la méthode de bissection et modifier des matrices.
Toutes les boucles ne sont pas définitives ; parfois, vous ne pouvez pratiquement pas savoir au départ combien de fois
vous devez répéter une tâche. Un exemple est la méthode de Newton. Cela peut être plus rapide que le
Méthode de bissection, mais elle utilise des lignes tangentes, elle nécessite donc que la fonction soit dérivable, pas
juste continu.

(Quelle est la différence entre une fonction continue et une fonction différentiable, demandez-vous ?
On peut penser la différence d'une manière géométrique : le graphe d'une fonction continue est « unbro-
ken », tandis que le graphique d'une fonction différentiable est « lisse ». Une fonction lisse est nécessairement un-
cassé, alors qu'une fonction ininterrompue peut ne pas être fluide. En effet, les fonctions différentiables sont
toujours continue, tandis que les fonctions continues ne sont pas toujours lisses. Ainsi, la méthode de Bisec-
tion fonctionne partout où fonctionne la méthode de Newton, mais peut-être pas aussi rapidement ; tandis que
La méthode de Newton ne fonctionne pas partout où la méthode de bissection fonctionne. D'ailleurs,
La méthode de Newton peut ne pas fonctionner partout où elle fonctionne,[1] selon votre imprudence
avec la configuration.)
L'idée de la méthode de Newton repose sur deux faits.
• La ligne tangente à une courbe se déplace dans la même direction que cette courbe.
• Il est facile de trouver la racine d'une droite : $mx + b = 0$ implique $x = -b/m$.
Mettez-les ensemble, et l'idée principale est que si vous commencez près de la racine d'une fonction, alors
suivez la ligne tangente jusqu'à sa racine, vous ne devriez pas vous retrouver trop loin de la racine que vous avez commencée
à partir de - et, espérons-le, vous serez plus proche qu'au début. Vous pouvez répéter ce processus comme
aussi longtemps que nécessaire jusqu'à ce que vous obteniez la précision souhaitée en chiffres. Voir la figure 1.
Comment décidons-nous que nous sommes arrivés à la précision souhaitée ? L'astuce habituelle est ronde
chaque approximation au nombre de chiffres souhaité, et ne s'arrêter que lorsque nous avons deux fois le même
valeur. Comme vous pouvez l'imaginer, il n'est pas évident de déterminer à l'avance le nombre de
étapes nécessaires pour ce faire. Il est plus simple de tester, après chaque itération, si la précédente
l'approximation et l'approximation actuelle correspondent au nombre de chiffres spécifié. Jusqu'à cela
point, cependant, nous n'avons aucun moyen de le faire.

[1] Cela peut sembler une curieuse combinaison de mots, mais c'est la vérité sans fard, comme nous le montrerons sous peu. Si
vous pensez que cela n'a pas de sens, donnez-lui quelques paragraphes et vous verrez. (Modulo une certaine licence artistique dans les sens

des mots, bien sûr.)
175

Page 176

METTRE EN UVRE UNE BOUCLE INDÉFINIE
176

F IGURE 1. L'idée de base de la méthode de Newton : commencez près d'une racine, suivez le ligne tangente et aboutir à un point (espérons-le) plus proche de la racine. Répétez aussi longtemps
comme voulu.

Implémenter une boucle indéfinie

Lorsque vous ne savez pas combien de fois répéter, mais que vous pouvez appliquer une condition comme celle-ci
ci-dessus qui se simplifie en True ou False, on opte pour ce qu'on appelle une boucle while. Pseudocode pour un
la boucle while a cette forme :
en condition
et est suivi d'une liste d'instructions en retrait. L'idée est que l'algorithme exécute chaque instruction chaque fois que la condition est vraie au début de la boucle ; l'algorithme ne teste pas
condition et quitter au milieu de la boucle si la condition devient soudainement vraie après l'un des
déclarations. Cela se traduit directement en code Sage : utilisez le mot-clé while , suivi d'un booléen
condition et un côlon,
en condition :
suivi d'une liste d'énoncés en retrait. Si vous le souhaitez, vous pouvez utiliser une commande break dans un
en boucle, comme une pour la boucle, mais nous ne recommandons pas.

Pour implémenter la méthode de Newton, nous devons suivre deux valeurs : l'approximation actuelle
de la racine, disons a, et la prochaine approximation de la racine, disons b. L'utilisateur spécifie un, et nous
calculer b. Tant que a et b sont en désaccord sur au moins un des nombres spécifiés d de chiffres, nous
continuer la boucle.

Une petite complication surgit ici. Une boucle while teste l'égalité au début de la boucle, plutôt qu'à la fin. Cependant, nous calculons b à l'intérieur de la boucle, nous ne pouvons donc pas connaître sa valeur à la
début à moins d'effectuer un calcul supplémentaire avant la boucle ! Cela ferait perdre de la place dans
le programme et le rendent un peu plus difficile à lire.

Pseudocode. Pour s'assurer que l'algorithme se rapproche au moins une fois, nous profitons de la nature du problème et créer une valeur manifestement fausse pour b. Dans ce cas, il serait ex-
très étrange en effet si quelqu'un nous demandait d'arrondir au chiffre des unités, nous utilisons donc b = a + 2
plutôt.[2] Dans ce cas, il serait extrêmement étrange que a et b s'accordent jusqu'à d chiffres décimaux.
Pour décrire l'idée en pseudocode, nous utilisons une troisième variable, c, comme lieu temporaire-
titulaire pour l'approximation la plus récente à la racine. L'utilisation d'un espace réservé nous permet de
organiser le calcul de manière à ce que nous puissions « glisser » des valeurs les plus anciennes aux plus récentes sur chaque
passer par la boucle.
algorithme Newtons_method
contributions
• f , une fonction dérivable autour d'un
• a, une approximation d'une racine de f
• d, le nombre de chiffres pour approcher une racine de f près d'un
les sorties
• une racine de f , correcte à d chiffres décimaux
fais
soit b = a +2, c = a
tandis que les d premiers chiffres décimaux de a et b diffèrent
soit a = c
soit b la racine de la droite tangente à f en x = a
soit c = b
retourner un
Voyons comment cela fonctionne sur le même exemple que nous avons utilisé avec la méthode de bissection, f (x) =
cos x − x. x et cos x sont tous les deux différentiables, donc leur différence l'est aussi ; nous pouvons continuer. Inspectez-
tion du graphique nous montre qu'une racine doit apparaître quelque part entre x = 0 et x = 1 ; puisque
x = 1 regarde de plus près, nous allons commencer l'algorithme avec a = 1 et d = 3.
L'algorithme initialise b à 3 et c à 1 avant d'entrer dans la boucle.
• Au début du premier passage dans la boucle, l'algorithme vérifie que le premier
trois chiffres de a = 1 et b = 3 diffèrent. Il affecte a = 1, puis b la racine de la droite tangente à cos x − x. Pour trouver cela, nous avons besoin de l'équation de la ligne tangente ; qui

nécessite un point,

[2] Un autre moyen plus sûr de le faire est de définir $b = a + 10^{-(d-1)}$, mais nous n'avons pas envie d'expliquer cela au-delà de dire
que si quelqu'un est assez étrange pour spécifier d = 0, alors cela nous donnerait $b = a + 10^{1}$, et si quelqu'un disait quelque chose de plus raisonnable comme d = 3, alors $b = a + 10^{-2}$, de sorte que $b - a = 10^{-2} > 10^{-3} = d$. Et ainsi de suite. Au-delà
que, nous laissons au lecteur le soin de comprendre pourquoi $10^{-(d-1)}$ est une si bonne idée - à moins que l'instructeur ne doive attribuer le
exercice correspondant. . .

---

que nous avons à[3] $(1, \cos 1 - 1) \approx (1, -0.459698)$, et une pente, ce que nous n'avons pas. La pente, la descente
de la tangente est la valeur de la dérivée, donc on calcule
$f(x) = -\sin x - 1$
et évaluer f (1) -1,84147. Notre tangente est donc $y + 0.459698 = -1.84147(x - 1)$.
Pour trouver la racine, définissez y = 0 et résolvez pour x, obtenant la nouvelle approximation b = c =
0,7503638678.

• Au début du deuxième passage dans la boucle, l'algorithme vérifie que le
les trois premiers chiffres de a = 1 et b = 0,75038678 diffèrent. Il attribue a = 0,75038678, puis
b la racine de la droite tangente à cos x − x. On connaît déjà la dérivée f (x), et
évaluer f (0,75038678) -1,6736325442. Notre tangente est donc $y + 0{,}0000464559 = -1.6736325442(x - 0.75038678)$. (Remarquez que la valeur y $4{,}6 \times 10^{-5}$ est déjà très, très proche de 0.) Pour trouver la racine, définissez y = 0 et résolvez pour x, obtenant la nouvelle approximation
b = c = 0,7391128909.

• Au début de la troisième passe, l'algorithme vérifie que a = 0,75038678 et b = 0,7391128909 diffèrent par leurs 3 premiers chiffres décimaux. Il attribue a = 0,7391128909, puis
b la racine de la droite tangente à cos x − x. En passant sur les détails, nous avons le nouveau approximation b = c = 0,7390851334.

• Au début de la quatrième passe, l'algorithme remarque que les trois premières décimales
les chiffres de a = 0,7391128909 et b = 0,7390851334 concordent. La boucle while se termine, et
l'algorithme n'a rien d'autre à faire que de renvoyer un $\approx$ 0,7391128909.

Remarquez à quelle vitesse la méthode de Newton a atteint son objectif ! Comment cela se compare-t-il à la méthode
de bissection ? Rappelez-vous que la méthode de bissection divise par deux l'intervalle sur chaque étape, donc en fonction
sur le choix de a et b, il aurait fallu exécuter n étapes telles que $2^{-n}(b - a) < 10^{-3}$, soit

$n > -\log_2 10^{-3}/_{b-a} = 3\log_2 10 + \log_2(b-a) \approx 10 + \log_2(b-a)$. La meilleure estimation que nous puissions faire à
(a, b) en regardant le graphe est probablement $(1/2, 3/4)$, donc $\log_2(b-a) = \log_2 1/2 = -1$. D'où le
La méthode de bissection a besoin de $n > 9$ pour approcher la racine au millième. Utilisant La méthode de Newton nécessitait moins d'un tiers de temps.
Code sage. Sage n'a pas de commande qui teste immédiatement si deux nombres diffèrent dans leurs premiers d chiffres décimaux, nous devons donc trouver comment le faire nous-mêmes. Ce n'est pas trop
dur, cependant; rappelez-vous que la commande round() arrondira un nombre au nombre spécifié de
chiffres. Donc, tout ce que nous avons à faire est d'arrondir a et b à trois chiffres, puis de vérifier s'ils sont égaux.
Nous devons également dire à Sage de résoudre l'équation de la ligne tangente à la courbe.
Retour à la p. 83
nous avons discuté de la façon de construire l'équation d'une ligne tangente, nous ne le répétons donc pas ici. le
La commande solve() nous donnera la racine, mais rappelez-vous que solve() renvoie une liste de solutions dans
la forme des équations, nous devons donc extraire la solution en utilisant [0] pour obtenir le premier élément de
la liste et la méthode .rhs() pour obtenir le membre de droite de l'équation.
Ces considérations suggèrent le code Sage suivant :
3 Eh bien, environ à. Nous devons recourir à des valeurs à virgule flottante car dans ce cas, les valeurs exactes deviennent laides
rapide et ralentit énormément le calcul.

```
sage : def newtons_method(f, a, d, x=x) :
f(x) = f
df(x) = diff(f, x)
b, c = a + 2, a
# boucle jusqu'à ce que la précision souhaitée soit trouvée
tandis que rond(a, d) != rond(b, d) :
a = c
m = df(a)
# trouver la racine de la tangente
sol = résoudre(m*(x - a) + f(a), x)
b = sol[0].rhs()
c = b
retourner un
```

Comment cela fonctionne-t-il sur l'exemple que nous avons spécifié précédemment ?

```
sage : newtons_method(cos(x) - x, 1, 3)
((cos(cos(1)/(sin(1) + 1) + sin(1)/(sin(1) + 1)) +
```

```
péché(cos(1)/(péché(1) + 1) + péché(1)/(péché(1) + 1)))*péché(1)
+ cos(1)*sin(cos(1)/(péché(1) + 1) + péché(1)/(péché(1)
+ 1)) + cos(cos(1)/(péché(1) + 1) + péché(1)/(péché(1) +
1)))/((sin(cos(1)/(sin(1) + 1) + sin(1)/(sin(1) + 1)) + 1)*sin(1) +
sin(cos(1)/(sin(1) + 1) + sin(1)/(sin(1) + 1)) + 1)
```

Oops! Cela illustre ce que nous entendions par avoir besoin d'approximer nos réponses. Modifions le
code de sorte que l'affectation de `b` à l'intérieur de la boucle change en

```
sauge:
b = rond(sol[0].rhs(), d+2)
```

Arrondir à d+2 places devrait garantir que nous ne perdons aucune précision dans les premières d places, et
cela garantit également que nous avons une approximation, au lieu de l'expression symbolique compliquée
nous avons trouvé plus tôt. Nous utiliserons également une première approximation à virgule flottante, `1.` .

```
sage : newtons_method(cos(x) - x, 1., 3)
0,73911
```

La relation entre les boucles définies et indéfinies. Il s'avère que des boucles définies sont vraiment juste un type spécial de boucle indéfinie, et toute boucle for peut être implémentée comme une boucle while.
Dans Sage, par exemple, on peut passer à travers une liste `L` en utilisant un `en` boucle comme suit:

```
sage : je = 0
sage : tandis que i < len(L):
# faire quelque chose avec L[i]
je = je + 1
```

Les collections non indexées comme les ensembles ne sont qu'un peu plus difficiles. Supposons que `S` est un ensemble ; nous pouvons le `copier()` , `pop()`
éléments hors de celui-ci et travailler avec eux. Lorsque nous avons terminé, `S` est toujours là.

```
sage : T = copie(S)
sage : sans len(T) == 0 :
t = T.pop()
# faire quelque chose avec t
```

Néanmoins, il est utile du point de vue de quelqu'un qui lit un programme de distinguer quand une boucle est définie à partir du moment où elle est indéfinie. Dans le second cas, par exemple, il est tout à fait possible
que les lignes en retrait en dessous `en n'étant pas len(T) == 0 :` modifiez `T` . Dans ce cas, la boucle n'est pas
précis; même `en` boucle de ce type peut être défini que si la collection est pas modifiée.[4]

Qu'est ce qui pourrait aller mal?

Nous avons dit que les boucles for étaient « définies » car elles itèrent sur une collection finie bien spécifiée.

tion. Cela garantit leur résiliation. Nous avons vu que les boucles while fonctionnent différemment : elles
continuer tant qu'une condition est vraie, donc en fait, ils pourraient ne jamais se terminer. Pour un exemple très simple
(n'essayez pas ça à la maison, les enfants !) :

```
sage : tandis que True :
a = 1
```

Dans cette boucle, la condition est spécifiée comme étant *True* . Cela ne changera jamais en rien d'autre. Rien dans
la boucle change le fait que *True* est *True* , et la boucle while ne se terminera jamais. C'est un bon moyen
pour « accrocher » l'ordinateur, et si vous étiez assez méchant pour ignorer notre mise en garde (vraiment, n'essayez pas
ceci à la maison, les enfants !), vous devrez alors arrêter Sage en cliquant sur le bouton Arrêter (nuage),
en sélectionnant le menu Action, puis en cliquant sur Interrompre (autre serveur), ou en maintenant la touche Ctrl enfoncée et appuyez sur C
(ligne de commande).
Ce n'est pas le seul endroit où il peut apparaître ; des conditions soigneusement préparées peuvent également mal tourner.
La méthode de Newton peut conduire à une boucle infinie lorsque nous faisons ce qui suit (une dernière fois : n'essayez pas
ça à la maison, les enfants !):

```
sage : f(x) = x**3 - 2*x + 2
sage : newtons_method(f(x), 0, 3)
```

4 Techniquement, on pourrait modifier T même dans une boucle for , mais le sens du mot rend moins probable une personne
ferait ça. Il est courant, cependant, d'effectuer une en boucle sur une collection qui est modifiée au cours de la boucle.

Dans ce cas, la dérivée est f (x) = 3x 2 − 2. La droite tangente à f en x = 0 est y = −2x + 2 ; son racine est x = 1. La droite tangente à f en x = 1 est y = (x −1) + 1 ; sa racine est x = 0. Nous sommes de retour
où nous avons commencé... oups !
Le lecteur qui regardera plus attentivement cet exemple pourra objecter qu'il est irréaliste, dans la mesure où
le choix de x = 0 semble évidemment imprudent, étant quelque peu éloigné d'une racine à
−
3 81 2· 3 81+3
3
9−57
27
3

9–57
-1,7693.
C'est encore plus évident d'après le graphique, qui illustre l'oscillation des approximations entre
entre x = 0 et x = 1 :
Bien que cela soit vrai, il est également vrai que commencer à x = 0,5 vous conduira également dans cette
oscillation; cela prend juste plus d'étapes avant que l'algorithme commence à fouetter entre 0 et 1.
S'égarer. Un exemple de boucle infinie théorique se produit avec f (x) = $1/1+x^2 - 1/2$.
Supposons que nous commençons la méthode de Newton à x = 3, qui n'est pas trop loin de la racine à x = 1. Dans ce
cas, Sage quitte soudainement avec l'erreur,

```
sage : newtons_method(f(x), 3., 3)
Erreur d'index :
index de liste hors de portée
```

Si vous regardez les informations supplémentaires fournies par Sage, vous le verrez se plaindre de la
ligne `b = round(sol[0].rhs(), d+2)` . Fondamentalement, *IndexError* signifie qu'une entrée n'existe pas dans
la liste à l'index spécifié. L'index apparaît entre parenthèses : `[0]` dans ce cas, donc Sage ne peut pas
trouver une entrée dans `sol` . Cela suggère qu'il n'y avait pas de solution.
Que s'est-il passé? Si vous tracez les calculs à la main, ou bien placez une `impression (a)` à la be-égrenage du `tout` en boucle, vous verrez les approximations thrash sauvagement entre négatif et valeurs positives de taille de plus en plus grande :
3, -3,66667, 8,58926, -149,80063, 840240,29866, $-1{,}48303\times10^{17}$, $8{,}15439\times10^{50}$.

Cette erreur est due au fait que nous travaillons avec des approximations, et les nombres deviennent ainsi
grand que l'ordinateur décide qu'il n'y a pas de solution et `solve()` abandonne, renvoyant un vide liste. Essayez de tracer les lignes tangentes sur le tracé pour voir comment cela se produit.
Encore une fois, le lecteur pourrait objecter que x = 3 ne semble pas être un point particulièrement convaincant
pour démarrer la méthode de Newton. C'est vrai, mais x = 0,09 c'est moins inquiétant, et ça fera quelque chose
similaire.
Un phénomène similaire peut se produire dans la méthode de Newton lorsque vous visez une racine, mais que vous terminez
à un autre. Ceci n'est pas un texte sur l'analyse numérique, nous n'entrerons donc pas dans ces

détails, ou certains
d'autres difficultés avec la méthode de Newton ; nous encourageons le lecteur intéressé à poursuivre un texte sur
cette.

Division des entiers gaussiens

Nous nous tournons vers un anneau intéressant appelé les entiers gaussiens, écrit [i] en abrégé. gaussien
les entiers ont la forme a+bi, où a et b sont des entiers. Cela diffère des nombres complexes, qui ont la forme superficiellement similaire a + bi, avec la différence que pour les nombres complexes
a et b sont des nombres réels. En bref,

• 2+3i est à la fois un entier complexe et un entier gaussien, tandis que
• 2+ i/3 et 2+ i 3 sont des entiers complexes, mais pas des entiers gaussiens.

L'addition, la soustraction et la multiplication d'entiers gaussiens sont identiques aux nombres complexes :

$$(a + bi)+(c + di)=(a + c)+(b + d)i$$
$$(a + bi)−(c + di)=(a − c)+(b − d)i$$
$$(a + bi)(c + di)=(ac − bd)+(ad + bc)i .$$

Pour la division, cependant, nous ne voulons pas passer des entiers gaussiens aux nombres complexes, comme Sage
pourrait nous donner par défaut :

```
sage : (2 + 3*I) / (1 - I)
5/2*I - 1/2
```

Nous aimerions plutôt un moyen de calculer un quotient et un reste, tout comme Sage propose le // et
% opérateurs pour les entiers. En particulier, nous aimerions un algorithme qui se comporte un peu comme une division de
entiers :

T HE D IVISION T HEOREM POUR I NTEGERS . Étant donné un entier n, appelé le dividende, et un
entier non nul d , appelé diviseur, on peut trouver un entier q, appelé quotient, et un entier non négatif
entier r , appelé le reste, tel que

$$n = qd + r$$

et

$$r < |d|.$$

Rappelez-vous que la valeur absolue d'un nombre réel nous donne une idée de sa taille ; l'analogue pour
nombres complexes est la norme z. Traduit en entiers gaussiens, le théorème deviendrait

T HE D IVISION T HEOREM POUR G AUSSIAN I NTEGERS . Étant donné un entier gaussien z, appelé
le dividende, et un entier gaussien non nul d, appelé diviseur, on peut trouver un entier

gaussien
q, appelé le quotient, et un autre entier gaussien r , appelé le reste, tel que
$z = qd + r$
et
$r \prec d$.

Ce théorème ne dit rien sur le fait que le reste est non négatif car « négatif » ne signifie pas vraiment du sens pour les entiers gaussiens.
Le théorème de division pour les entiers gaussiens est en fait vrai, et nous allons le prouver par construction-
ing un algorithme qui fait ce qu'il prétend. Mais comment construire un tel algorithme ? Di-
la vision des nombres entiers est une soustraction répétée, et vous pouvez créer un algorithme de division parfaitement bon
de la manière suivante :
algorithme divise_entiers
contributions
• $n,d \in$ tel que $d = 0$
les sorties
• $q \in$ et $r \in$ tels que $n = qd + r$ et $r \prec |d|$
fais
soit $r = |n|$, $q = 0$
si n et d ont le même signe
soit $s = 1$
autre
soit $s = -1$
tant que $r \prec 0$ ou $r |d|$
ajouter s à q
remplacer r par $r - sd$
renvoie q, r
Cet algorithme utilise s pour « échelonner » le quotient dans la bonne direction vers n. Par exemple,
si on divise −17 par 5, alors r et q ont les valeurs suivantes au début de chaque boucle (q diminue car on a $s = -1$) :
r −17 −12 −7 −2 3
q
0
-1 -2 -3 -4
Une fois $r = 3$ et $q = -4$, l'algorithme remarque que $r \prec |d|$, donc la boucle while se termine, nous donnant le

expression correcte
−17 = qd + r = (−4)×5+3.
Il est logique que la division des entiers gaussiens fonctionne de la même manière. Notre général
stratégie, alors, adoptera cette approche:
soit r = z, q = 0
tandis que r ≥ d
« step q up » en utilisant une valeur appropriée s
remplacer r par r − sd
renvoie q, r
Maintenant, nous devons comprendre ce que signifie « intensifier q ». La difficulté réside dans le fait que nous
peut « augmenter » q dans deux directions différentes : la direction réelle et la direction imaginaire. Comme
vous avez vu dans l'introduction au plan complexe p. 77, en multipliant un nombre complexe par i
fait tourner 90 ∘ sens inverse des aiguilles:

---

4 + je
i(4 +i)
× je
Il est donc possible de « monter q up » de deux manières différentes.
Nous allons résoudre ce problème en utilisant une approche en deux étapes : nous minimisons d'abord la distance lors de la multiplication
par des nombres entiers ; alors nous minimisons la distance en multipliant par des multiples entiers de i. Pour
exemple, supposons que nous divisons 8+7i par 2+i. Le diviseur se trouve le long de la ligne de pente $1/2$, donc nous "pas"
le long de cette ligne dans la direction appropriée jusqu'à ce qu'un pas supplémentaire augmente la distance.
De la géométrie, nous savons que cela signifie se rapprocher au plus près de l'altitude tombée de (8,7) à la
ligne $y = x/2$:
(8 , 7)
y = x/ 2
Malheureusement, le point le plus proche est $(9\ 1/5, 4\ 3/5)$ (en rouge), ce qui correspond à un nombre complexe,
mais pas un entier gaussien. L'entier gaussien le plus proche est le point bleu qui se trouve au-delà du
point rouge; la seule façon de vérifier cela est de comparer la norme de chaque point au fur et

à mesure que nous nous déplaçons le long de la
ligne.
L'étape suivante consiste à passer de notre entier gaussien minimal le long d'une ligne perpendiculaire à la
bleu, correspondant à la multiplication répétée de d par i. Encore une fois, nous marchons le long de la ligne jusqu'à ce que
toute mesure supplémentaire augmenterait la norme.
(10 , 5)
(8 , 7)
$y = -2x + 23$
Le point le plus proche sur la ligne avec des valeurs entières est (9,7), ce qui correspond à 9+7i. Nous avons obtenu
cette valeur après avoir multiplié d'abord par 5, puis par i :
(2+ i)(5+ i) = 10+2i +5i -1 = 9+7i.
Le reste est -1, dont la norme est plus petite que celle du diviseur.

Écrire ceci en tant que pseudocode nous donne ce qui suit.

```
algorithme divise_entiers_gaussiens
contributions
• z,d [i] tel que d = 0
les sorties
• q, r [i] tels que z = qd + r et r ≺ d
fais
soit r = z, q = 0
si r − d ≺ r
soit s = 1
autre
soit s = −1
tandis que r − sd ≺ r
ajouter s à q
remplacer r par r − sd
si r − id ≺ r
soit s = je
autre
soit s = −i
tandis que r − sd ≺ r
ajouter s à q
remplacer r par r − sd
renvoie q, r
```

Comme nous l'avions prévu, cet algorithme utilise une instruction if pour vérifier d'abord s'il est préférable d'effectuer un pas
en avant ou en arrière par un facteur de valeur réelle. Il sélectionne ensuite s à 1 ou -1 selon ce qui est
plus approprié. La boucle while va ensuite dans la direction appropriée, diminuant la norme jusqu'à ce que d'autres mesures l'augmentent. On ne peut pas se fier au test r d comme pour les entiers,
car nous aurons peut-être besoin d'une minimisation supplémentaire après la première étape car nous avons toujours r > d.
C'est pourquoi la boucle vérifie r −sd < r à la place ; grosso modo, cela se traduit par
réduit encore la norme de r.
L'algorithme utilise ensuite une seconde instruction if pour vérifier s'il est préférable d'avancer ou en arrière par un facteur à valeur imaginaire. Il sélectionne s comme i ou −i selon ce qui est plus approprié. La boucle while va ensuite dans la direction appropriée, diminuant la norme jusqu'à ce que d'autres mesures l'augmentent. On pourrait en fait tester r ≥ d ici, mais par souci de
cohérence, nous avons utilisé la même structure de base.
Cela se traduit par le code Sage de la figure 2. Comment fonctionne-t-il sur certains exemples ?

```
sage : diviser_entiers_gaussiens (4 + I, 1 - I)
(2*I + 1, 1)
sage : diviser_entiers_gaussiens(8 + 7*I, 2 + I)
(5 + moi, -1)
```

```
sage : def diviser_entiers_gaussiens(z, d) :
r, q = z, 0
# quelle vraie façon de marcher ?
si norme(r - d) < norme(r) :
s = 1
autre:
s = -1
# boucle à pas
tant que norme(r - s*d) < norme(r) :
q = q + s
r = r - s*d
# quelle manière imaginaire de marcher ?
si norme(r - I*d) < norme(r) :
s = je
autre:
s = -je
# boucle à pas
tant que norme(r - s*d) < norme(r) :
q = q + s
r = r - s*d
renvoie q, r
```

F IGURE 2. Code Sage pour la division des entiers gaussiens

Nous avons déjà vérifié le deuxième calcul en le parcourant plus tôt ; le premier n'est pas difficile à
fais toi-même.
Une retraite bien ordonnée. Avec la méthode de Newton, nous avons rencontré une boucle infinie avec un-
combinaisons chanceuses d'une fonction f et d'une estimation initiale a. Est-il possible de trouver deux gaussiennes
entiers tels que le tout boucles de divide_gaussian_integers () deviennent infinies?
Pas dans ce cas, non. La différence est que les boucles sont construites de telle manière que nous pouvons décrire
la condition utilisant un entier non négatif qui diminue après chaque passage réussi à travers le
boucle. Dans les deux boucles, cette valeur est z − qd. Pourquoi? L'algorithme initialise r = z, puis "steps"
q plus proche de z si et seulement si cela réduirait la distance entre z et qd ; autrement dit, z − qd. La norme est toujours un entier, et les entiers positifs bénéficient d'une propriété appelée le bien-
propriété de commande :
Chaque sous-ensemble non vide de a un moindre élément.
Si la boucle était infinie, nous pourrions considérer toutes les valeurs de z − qd, ayant un sous-ensemble de . le
l'algorithme effectue la boucle si cette distance diminue, et la propriété de bon ordre garantit que cela ne peut se produire qu'un nombre fini de fois, ce qui contredit l'hypothèse selon laquelle la boucle est
infini.
Si possible, c'est une très bonne idée lorsque vous travaillez avec des boucles while de les formuler dans un tel
manière dont vous exploitez la propriété de bon ordre d'une manière ou d'une autre, garantissant ainsi la résiliation. Toi
peut le faire dans à peu près toutes les circonstances, bien que la conséquence puisse être indésirable.
Une façon de le faire avec la méthode de Newton, par exemple, consiste à définir une variable max_iterations sur

---

un grand nombre, diminuez-le d'un à chaque passage dans la boucle, puis reformulez le tout la condition de la boucle pour qu'elle se ferme au moment où max_iterations atteint 0. Vous pouvez même choisir de
lever une exception. Nous concluons cette section avec un terminating_newtons_method () modifié qui illustre cette technique.

```
sage : def terncing_newtons_method(f, a, d, x=x) :
```

```
f(x) = f
df(x) = diff(f, x)
# a + 2 devrait être trop loin pour une résiliation prématurée
b, c = a + 2, a
max_itérations = 100
# boucle pour la précision souhaitée, jusqu'au seuil
tandis que max_iterations > 0 et round(a, d) != round(b, d):
a = c
m = df(a)
sol = résoudre(m*(x - a) + f(a), x)
b = sol[0].rhs()
c = b
itérations_max = itérations_max - 1
# Erreur?
signaler si oui
si max_iterations == 0 :
raise ValueError('Trop d'itérations :
')
+ 'essayez une valeur initiale différente'
retourner un
```

Essayez-le avec l'exemple non terminé d'avant ($f = x^3 - 2x + 2$, $a = 0$) et voyez comment cela se comporte beaucoup mieux.

Des exercices

Vrai faux. Si la déclaration est fausse, remplacez-la par une déclaration vraie.

1. Dans Sage, le mot-clé `while` implémente une boucle indéfinie.
2. A `en` boucle arrête dès que la condition devient fausse, même si elle n'a pas rempli tous les déclarations en retrait en dessous.
3. La meilleure façon de faire une boucle à travers une collection utilise une `tandis que` la boucle.
4. Une boucle indéfinie est un type particulier de boucle définie.
5. Il est impossible de créer une boucle infinie lors de l'utilisation d'une boucle indéfinie.
6. Il est impossible d'empêcher chaque boucle indéfinie de devenir une boucle infinie.
7. La méthode de Newton trouve chaque racine que la méthode de bissection trouvera.
8. La méthode de Newton trouve des racines que la méthode de bissection ne trouvera pas.
9. La division gaussienne est plus compliquée que la division complexe car elle nécessite un quotient
et un reste.
10. La propriété de bon ordre stipule que chaque sous-ensemble des nombres naturels
a au moins
élément.

Choix multiple.

1. Une boucle qui parcourt une collection spécifique est :

A. boucle définie
B. boucle indéfinie
C. boucle infinie

D. boucle spécifique
2. Une boucle qui se répète un certain nombre de fois est :
A. boucle définie
B. boucle indéfinie
C. boucle infinie
D. boucle spécifique
3. Une boucle qui se répète selon qu'une condition reste vraie ou fausse est :
A. boucle définie
B. boucle indéfinie
C. boucle infinie
D. boucle spécifique
4. La principale raison d'utiliser une boucle indéfinie au lieu d'une boucle définie est :
A. Une boucle indéfinie est plus facile à garantir la terminaison.
B. Une boucle indéfinie est plus difficile à garantir la terminaison.
C. Pour décider de se terminer, la boucle doit tester une condition booléenne.
D. Les boucles définies ne sont qu'un type spécial de boucle indéfinie, il est donc plus cohérent d'utiliser
boucles définies exclusivement.
5. Lequel des énoncés suivants caractérise le mieux l'idée derrière la méthode de Newton ?
A. Utilisez la ligne tangente pour trouver une nouvelle approximation, si tout va bien plus proche de la racine correcte.
B. Utilisez la ligne tangente pour trouver une nouvelle approximation, nettement plus proche de la racine correcte.
C. Coupez en deux l'intervalle à plusieurs reprises jusqu'à ce que les extrémités soient d'accord sur les d premiers chiffres décimaux.
D. Tracez le long de la courbe avec une précision extrême, en zoomant et en vous déplaçant d'avant en arrière jusqu'à
vous avez le bon résultat.
6. Pour laquelle des fonctions suivantes seriez-vous mieux avisé d'utiliser la méthode de Bisec-
tion pour trouver une racine, plutôt que la méthode de Newton ?
A. $|x+3|$
B. $\sin(x+\pi)$
C. $(x-3)^2$
D. $\ln(x-3)$
7. Dans quelles conditions la méthode de Newton peut-elle ne pas trouver de racine ?
A. la fonction est discontinue
B. la fonction est indifférenciable
C. le point de départ s'éloigne de la racine
Tout ce qui précède
8. Laquelle des propositions suivantes est une interprétation géométrique correcte de ce qui se passe lorsque vous mul-

tiply un nombre complexe par –i? (Notez qu'il est négatif!)
A. Le nouveau numéro correspond à un 90 ∘ rotation dans le sens horaire dans le plan complexe.
B. Le nouveau numéro correspond à un 90 ∘ rotation dans le sens antihoraire dans le plan complexe.
C. Le nouveau numéro correspond à un 180 ∘ rotation dans le plan complexe.
D. Le nouveau numéro est dans le même sens que le numéro d'origine, mais plus longtemps.
9. Pour laquelle des paires de nombres suivantes la division gaussienne devrait-elle échouer ? (le dividende vient
premier, diviseur deuxième)
A. 4+ i, 1+ i

B. 0, 1+ je
C. 1+ i, 0
D. aucune de ces réponses
10. La raison pour laquelle nous avons généré une *ValueError* dans la dernière version de la méthode de Newton est que :
A. Nous considérons que la fonction était une mauvaise valeur, car elle n'est pas continue en a.
B. On considère le point de départ comme une mauvaise valeur, car la fonction n'est pas dérivable
là.
C. Les valeurs sont devenues trop importantes et Sage n'a pas pu trouver de solution.
D. Nous considérons le point de départ comme une mauvaise valeur, car il semble pris dans une boucle infinie.
Réponse courte.
1. Considérez la procédure Sage suivante :

```
sage: def aw(n):
t = 0
je = 1
tandis que i <= n :
t = t + i**2
je = je + 1
retour t
```

(a) Calculez `aw(4)` à la main, en montrant votre travail.
(b) Réécrivez `aw(n)` comme une expression mathématique sur une ligne, en utilisant la notation de sommation.
(c) Traduire `aw(n)` en une nouvelle procédure `af(n)` , qui fait la même chose, mais utilise un `for` boucle à la place.
(d) Tant la boucle while `que la` boucle `for` sont inefficaces. Comment serait-il plus intelligent d'implémenter

ment cette procédure?
2. La séquence de Syracuse est définie de la manière suivante. Soit S_1 , et
$S_{i+1} =$
$S_i/2$,
S_i est pair ;
$3S_i + 1$, S_i est impair.
La séquence se termine une fois $S_i = 1$. On pense que la séquence se termine avec $S_i = 1$ pour chaque valeur de S_1 , mais personne ne le sait réellement. Calculer à la main la séquence de
plusieurs valeurs différentes de S_1 , et remarquez comment il semble toujours atteindre $S_i = 1$.
3. La division de polynômes univariés f par g fonctionne de la manière suivante (ici, deg(r) signifie
« degré de r » et lc(r) signifie « coefficient principal de r » :
• soit r = f , q = 0
• tant que deg(r) deg(g)
– soit $t = x^{deg(r)-deg(g)}$, $c = lc(r)/lc(g)$
– remplacer r par r − ctg
– ajouter ct à q
• renvoie q, r
(a) Choisissez 5 paires de polynômes et exécutez l'algorithme de division dessus. Vous pouvez utiliser
Sage pour aider à accélérer les soustractions, mais si vous utilisez la commande Division de Sage, vous ne le ferez pas
être en mesure de répondre à au moins un des prochains problèmes.
(b) Pensez-vous que cet algorithme exploite la propriété du bon ordre ? Pourquoi ou pourquoi pas?

(c) Existe-t-il un modèle pour le nombre de fois que l'algorithme passe par la boucle ?
(d) La boucle pourrait-elle être écrite comme une boucle for ? Si c'est le cas, comment? Si non, pourquoi pas ?
4. Montrer comment ceux - ci pour les boucles peuvent être réécrites en utilisant un en boucle dans chaque cas:
(une)

```
pour k dans la plage (num):
<body> # éventuellement en fonction de k
```

(b)

```
pour elem dans L : # L est une collection (list/tuple/set)
<corps>
# selon élément
```

5. Peut toutes en boucle être réécrite en une pour la boucle? Expliquer!

Programmation.

1. Dans la réponse courte n° 1, vous avez travaillé avec la séquence de Syracuse. Écrivez un pseudo-code pour savoir comment vous
pourrait mettre en œuvre cela comme un programme. Il doit accepter un argument pour S_1, définir $s = S_1$, appliquer le
formule à s pour trouver sa valeur suivante, et se termine lorsque s = 1. Implémentez votre pseudocode.
(Vous pouvez tester si s utilise l' opérateur % de Sage , car un nombre est pair lorsque le reste après division par 2 est 0.)

2. L'algorithme traditionnel de division entière à la p. 183 trouve le reste non négatif le plus proche -
der. Dans de nombreux cas, autoriser un reste négatif laisserait une distance plus petite entre n et dq. Par exemple, la division traditionnelle aurait
$-17 = -4\times5+3$,
tout en permettant des restes négatifs aurait
$-17 = -3\times5+(-2)$.
Il est possible de formuler un algorithme qui effectue cette division du « plus petit reste ».
(a) Essayez quelques exemples où vous utilisez des soustractions répétées pour effectuer le « plus petit reste »
division.
(b) Comment apprendriez-vous à un ami à exécuter cette division ? Pensez à l'instruction précise
tions que vous auriez à donner et à formuler un pseudocode.
(c) Implémentez votre algorithme en tant que programme Sage et testez-le.

3. Lorsque nous avons mis en œuvre la méthode de bissection, nous avons utilisé un logarithme pour déterminer la ré-
nombre d'étapes requis. C'était la seule façon pour nous d'utiliser une boucle définie. Il est pos-
ble de reformuler l'algorithme avec une boucle indéfinie, similaire à la méthode de Newton : terminate
une fois que a et b correspondent au nombre de chiffres spécifié.
(a) Faites ceci. Réécrivez à la fois le pseudocode puis le code Sage pour voir cela en action.
(b) Quelle méthode préférez-vous et pourquoi ?
(c) Selon vous, quelle méthode est la plus facile à lire, à comprendre et/ou à modifier pour quelqu'un d'autre ?
ifier ? Encore une fois, pourquoi ?

4. Dans la réponse courte n° 3, vous avez travaillé sur la division de polynômes univariés. Écrire le pseudo-code complet
pour cela, c'est-à-dire donner un nom à l'algorithme, spécifier précisément les entrées et les sorties pré-
avec précision. Ensuite, implémentez-le en tant que programme Sage et testez-le sur les

mêmes exemples que vous avez fait dans ce
problème.

DES EXERCICES
191

5. Un entier positif n est premier si aucun entier de 2 à n ne divise n.
(a) Écrivez une procédure Sage qui accepte une valeur de n en entrée et trouve le plus petit diviseur positif de n.[5] Si le nombre est premier, la procédure doit générer une *ValueError* .
(b) Écrivez une autre procédure Sage qui accepte un entier positif k en entrée et renvoie une liste
de tout nombre premier inférieur ou égal à k. Demandez à la procédure d'invoquer la réponse à
partie (a).
(c) La conjecture de Goldbach affirme que tout nombre pair est la somme de deux nombres premiers
bres. Bien qu'ayant été vérifiée jusqu'à des valeurs incroyablement gigantesques, personne n'a encore
prouvé que la conjecture de Goldbach est vraie pour tous les entiers positifs. Écrivez une procédure qui
accepte un entier positif m en entrée, lève une *ValueError* si m n'est pas pair ; sinon, il
trouve deux nombres premiers dont la somme est m. Demandez à la procédure d'invoquer la réponse à la partie
(b).
6. Le plus grand diviseur commun de deux entiers est le plus grand entier qui les divise tous les deux.
Un algorithme pour calculer le plus grand diviseur commun était déjà connu d'Euclide d'Alexandre.
dria il y a plus de 2000 ans :
algorithme pgcd
contributions
• a, b
les sorties
• le plus grand commun diviseur de a et b
fais
soit m = |a|, n = |b|
tandis que n = 0
soit d = n
remplacer n par le reste de la division de m par n
remplacer m par d
retour m
(a) Implémentez ceci en tant que code Sage et vérifiez qu'il donne les plus grands diviseurs

communs corrects pour
plusieurs grands nombres.
(b) Choisissez une petite paire d'entiers non nuls et suivez l'algorithme, étape par étape, montrant comment il arrive au pgcd.
7. L'algorithme d'Euclide étendu ne trouve pas seulement le pgcd de deux entiers non nuls a et b,
il trouve les nombres c et d tels que
pgcd(a, b) = ac + bd.
Par exemple, pgcd(4,6) = -1×4+1×6. Nous pouvons le décrire ainsi :
5 Un diviseur positif n'est pas trivial s'il n'est pas 1.

DES EXERCICES
192
algorithme étendu_euclidean_algorithm
contributions
• a, b
les sorties
• c,d ∈ tel que pgcd(a, b) = ac + bd
fais
soit m = |a|, n = |b|
soit s = 0, c = 1, t = 1, d = 0
tandis que n = 0
soit q le quotient de la division de m par n
soit r le reste de la division de m par n
soit m = n et n = r
soit w = c, alors c = s et s = w − qs
soit w = d, alors d = t et t = w − qt
retour c, d
Implémentez cela en tant que code Sage et vérifiez qu'il donne les valeurs correctes de c et d pour plusieurs
grandes valeurs de a et b; c'est-à-dire vérifier que les valeurs c et d qu'il renvoie satisfont ac + bd =
pgcd(a, b).

CHAPITRE 9

Se répéter inductivement

Une autre forme de répétition est appelée récursivité. Le terme signifie littéralement « courir à nouveau »,
parce que l'idée est qu'un algorithme résout un problème en le divisant en cas plus simples, en invoquant

lui-même sur ces cas plus petits, puis reconstituer les résultats. Ces petits cas généralement diviser leurs cas en cas encore plus petits. Cela peut sembler un peu louche : qu'est-ce qui nous empêche de
frapper une chaîne infinie?
Lors de l'organisation d'une récursivité, nous essayons de la mettre en place de telle sorte que cette stratégie crée une chaîne
de problèmes qui conduisent à un élément le plus petit possible. Si nous pouvons les relier aux nombres naturels,
alors encore mieux, car les nombres naturels bénéficient d'une propriété appelée propriété de bon ordre :
Chaque sous-ensemble non vide de a un moindre élément.
(Si cela vous semble familier, c'est parce que nous y avons fait référence à la page 186 du chapitre 8.)
Récursivité
Un exemple classique. Un certain nombre de problèmes se traduisent naturellement par une récursivité. Un exemple
est le triangle de Pascal, qui ressemble à ceci :
1
1
1
1
2
1
1
3
3
1
1
4
6
4
1
...
...
...
Les lignes de ce triangle apparaissent à de nombreux endroits, comme le développement de binômes :
$(x+1)^0$
=
1
$(x+1)^1$
=

1 fois
+
1
$(x+1)^2$
=
$1x^2$
+
2x +
1
$(x+1)^3$
=
$1x^3$
$+ 3x^2$
$+ 3x + 1$
$(x+1)^4$
=
$1x^4$
$+ 4x^3$
$+ 6x^2$
$+ 4x + 1$
...
...
...
Prenez un moment pour essayer de déduire le motif du triangle.
Est-ce que tu le vois? sinon, prenez un autre moment pour essayer de le résoudre.
J'espère que vous avez vu que chaque ligne est définie de la manière suivante. Que la ligne du haut soit la ligne 1,
la rangée suivante en bas de la rangée 2, et ainsi de suite.
• Le premier et le dernier élément de chaque ligne sont tous les deux à 1.
• Les éléments restants satisfont $q_j = p_{j-1} + p_j$, où
– q_j est l'élément de l'entrée j sur la ligne i, et

– p_{j-1}
et p_j sont les éléments des entrées j -1 et j sur la ligne i -1.
Pour connaître les entrées de la ligne i, nous devons donc d'abord connaître les entrées de la ligne i -1. On peut boucler
à travers ces entrées pour nous donner les entrées suivantes. Nous savons combien d'entrées il y a dans

cette ligne : i −1, bien que nous puissions également nous renseigner à partir de la ligne elle-même.[1] Nous pouvons donc appliquer une boucle définie.

Cela nous donne le pseudocode récursif suivant pour calculer la ligne i :

algorithme pascals_row
contributions
• i ∈ , la ligne désirée du triangle de Pascal
les sorties
• la suite des nombres de la ligne i du triangle de Pascal
fais
si je = 1
résultat = [1]
sinon si je = 2
résultat = [1,1]
autre
précédent = pascals_row(i −1)
résultat = [1]
pour j (2,3,...,i −1)
ajouter un précédent
$_{j-1}$ +préc $_{j}$ au résultat
ajouter 1 au résultat
résultat de retour

Cela se traduit par le code Sage suivant :

```
sage : def pascals_row(i) :
si je == 1 :
résultat = [1]
elif i == 2:
résultat = [1, 1]
autre:
# calcule la ligne précédente en premier
précédent = pascals_row(i - 1)
# cette ligne commence par 1...
résultat = [1]
# ...ajoute deux au-dessus suivant dans cette ligne...
pour j dans la plage (1, i - 1) :
result.append(prev[j-1] + prev[j])
# ...
et se termine par 1
result.append(1)
résultat de retour
```

1 Dans le pseudocode, on aurait pu écrire « |prev| » pour l'indiquer. Dans le code Sage, on pourrait simplement écrire prev.len() .

Nous pouvons le vérifier sur les lignes que nous avons vues ci-dessus et en générer plus que vous pouvez vérifier à la main :

```
sage : pascals_row(1)
```

[1]
sage : pascals_row(5)
[1, 4, 6, 4, 1]
sage : pascals_row(10)
[1, 9, 36, 84, 126, 126, 84, 36, 9, 1]

Un exemple si classique qu'il en est médiéval. L'exemple suivant est l'un des problèmes fondamentaux
des mathématiques,
À quelle vitesse une population de lapins va-t-elle croître?
Tu dois demander?!?
Vous pensez que nous plaisantons, mais ce n'est pas le cas.[2] Comme vous pouvez le deviner, ces lapins ne sont pas ordinaires
lapins; ils obéissent à des lois spéciales de reproduction :
• Au début, il n'y avait qu'une paire, mais ils étaient immatures.
• Chaque paire de lapins prend une saison pour mûrir.
• Une fois mature, chaque paire de lapins produit une nouvelle paire de lapins à chaque saison.
• Les lapins sont immortels. Ils n'ont pas de prédateurs et tirent leur énergie de
le soleil, pour qu'ils ne manquent jamais de nourriture. Étonnamment, ils ne se lassent jamais des nouveaux enfants
en les réveillant la nuit, en rampant dessus et en gênant généralement
poursuites plus nobles, telles que la contemplation de la forme du lapin idéal.[3]
La formule pour le nombre de paires de lapins en une saison est définie par le nombre dans le les deux saisons précédentes :

$b_n = b_{n-1} + b_{n-2}$

.
C'est un parfait exemple de récursivité. Il est naturel d'essayer de l'implémenter de manière récursive avec un
procédure qui accepte un entier pour le nombre de saisons écoulées. Si c'est 1 ou 2, le
la réponse est simple : 1. Sinon, on calcule le nombre de lapins des deux saisons précédentes.

2 Au moins un journal est dédié à ces lapins.
3 Référence gratuite (et non pertinente) à la philosophie platonicienne incluse sans frais supplémentaires.

---

algorithme funny_bunnies
contributions
• n ∈ , le nombre de saisons
les sorties
• le nombre de couples de lapins après n saisons
fais
si n = 1 ou n = 2
laisser résultat = 1
autre

laisser résultat = funny_bunnies(n −1) + funny_bunnies(n −2)
résultat de retour
Pour traduire cela en code Sage, gardez à l'esprit que range() n'exécute pas d'itération sur la dernière valeur, nous
besoin que sa limite soit n-1 :

```
sage : def funny_bunnies(n):
si n == 1 ou n == 2 :
résultat = 1
autre:
résultat = funny_bunnies(n-1) + funny_bunnies(n-2)
résultat de retour
sage : [drôle_bunnies(i) pour i dans la plage (2,10)]
[2, 3, 5, 8, 13, 21, 34]
```

C'est une liste de chiffres très intéressante. C'est tellement intéressant qu'il porte le nom d'un mathématicien
qui l'a décrit au Moyen Âge, Léonard de Pise, plus connu sous le nom de Fibonacci,[4] donc les gens

4 Fibonacci a vécu à une époque connue sous le nom de Haut Moyen Âge et ne se souciait pas tant des lapins que du fait
que les gens utilisaient toujours I, V, X, L, C, D et M pour écrire des nombres et faire de l'arithmétique — des chiffres romains, c'est-à-dire.
Il s'est rendu compte que c'était très inefficace et qu'il serait préférable d'utiliser la manière hindou-arabe d'écrire les nombres : 0, 1,
2, ..., 9 et tout ça. Le problème était que la plupart des gens étaient habitués à travailler avec une vieille calculatrice appelée
boulier, et avait été formé à l'utilisation d'un boulier avec des chiffres romains. A l'époque comme aujourd'hui, les gens hésitaient à
abandonner une béquille informatique.
Pour contourner ce problème, il a écrit un livre intitulé Liber Abaci («Livre de l'abaque»). Le point principal du livre était de
montrez comment vous pourriez résoudre tous les mêmes problèmes avec des nombres hindous-arabes sans utiliser un boulier, et travailler
beaucoup plus efficacement, pour démarrer. Faire le travail plus rapidement permettait de faire plus d'affaires. Les lapins platoniciens étaient
l'un des exemples qu'il a utilisés dans le livre. C'était probablement la première fois que quelqu'un utilisait un drôle de lapin pour faire
de l'argent.
Oh, nous n'avons pas encore fini de te faire gémir, pas de loin. Vous pouvez penser que Liber Abaci est un titre ennuyeux
– certainement pas aussi accrocheur que le slogan « Think Different », bien que Fibonacci aurait eu de meilleures raisons de
utiliser ce slogan que quiconque l'a popularisé aujourd'hui - mais les gens à l'époque étaient plus intéressés par le fond que dans le style, donc Fibonacci a réussi à la fois à sevrer les gens des chiffres romains et à faire attacher son nom
à une séquence de nombres qui, ironiquement, avait été étudiée par des mathématiciens indiens (hindous) bien avant que Leo ne soit
lui-même un nombre de Fibonacci.
Pourtant, pas mal, pour un groupe d'âge moyen intellectuellement arriérés.
(Et, oui, nous reconnaissons l'ironie en se référant à un boulier comme une « béquille informatique » dans un texte consacré à
faire des problèmes de mathématiques avec le système de calcul formel Sage, mais cette note de bas de page est conçue pour être riche en ironie.)

appelons cela la suite de Fibonacci. Nous pouvons le décrire ainsi :[5]

$F_{\text{suivant}} = F_{\text{total}} + F_{\text{mature}}$.

Cela signifie que "le nombre de lapins la saison prochaine est la somme du nombre total de lapins et du
nombre de lapins matures. On pourrait aussi dire que le nombre total de lapins est le « actuel » nombre de lapins, tandis que le nombre de lapins matures est le nombre de lapins que nous avons eu en dernier
saison. Cela signifie que "le nombre de lapins la saison prochaine est la somme du nombre de lapins cette
saison et le nombre de lapins la saison dernière. On pourrait donc écrire :

$F_{\text{suivant}} = F_{\text{actuel}} + F_{\text{précédent}}$.

Un exemple que vous ne pouvez pas réellement programmer. Un autre exemple est la technique de preuve appelée
induction. Le principe de l'induction est que :

• si vous pouvez prouver :
– une propriété s'applique à 1, et
– chaque fois qu'il s'applique à n, il s'applique également à n+1,
• alors la propriété s'applique à tous les entiers positifs.

L'idée est que si nous pouvons montrer que la première puce est vraie, alors la deuxième puce est vraie pour tout élément positif
entier m car

• si c'est vrai pour m −1, alors c'est vrai pour m ; et
• si c'est vrai pour m −2, alors c'est vrai pour m −1, et
...
• si c'est vrai pour 1, alors c'est vrai pour 2.

Nous savons que c'est vrai pour 1 (c'est dans la première puce), donc la chaîne d'implications nous dit que c'est vrai pour
m, qui était un entier positif arbitraire.

Alternatives à la récursivité

La récursivité est assez courante et raisonnablement facile à mettre en œuvre, mais ce n'est pas toujours la
meilleure approche. Il n'est pas difficile du tout de calculer le triangle de Pascal même jusqu'à la 900e rangée, et vous
pourrait être en mesure d'aller plus loin si Sage n'a pas limité le nombre de fois que vous pouvez postuler
récursivité :

```
sage : P = pascals_row(900)
sauge : len(P)
900
```

sauge : P[-3]
403651
sage : P = pascals_row(1000)
Erreur d'exécution:
profondeur de récursivité maximale dépassée lors de l'appel d'un
objet Python
5 F signifie « drôle » et non « Fibonacci ». Pourquoi demandez-vous?

Cette limite est là pour une bonne raison, d'ailleurs ; la récursivité peut utiliser un peu un type spécial de
Mémoire,[6] et une récursivité trop profonde peut aussi indiquer une boucle infinie, ou du moins que quelque chose
a terriblement mal tourné.
Avec la séquence de Fibonacci, les choses tournent mal assez vite, mais d'une manière différente. Essayer
calculant le nième nombre de Fibonacci pour les valeurs vraiment pas très grandes de n, et vous verrez le
longueur les problèmes prennent un virage brusque vers le haut rapidement. Sur les machines des auteurs, par exemple,
il y a déjà un décalage notable à n = 30 ; n = 35 prend plusieurs secondes ; et pour n = 40 vous pourriez
juste pouvoir te faire une tasse de café[7] avant la fin. Vous pourriez penser que vous pourriez obtenir
autour de ce délai avec un ordinateur plus rapide, et vous auriez tort, très tort. Le problème est que
il y a trop de branches à chaque étape. Pour voir cela clairement, développons un calcul de la 7e nombre de Fibonacci jusqu'à l'un des deux cas de base :

$F_7 = F_6 + F_5$
$= (F_5 + F_4)+(F_4 + F_3)$
$= [(F_4 + F_3)+(F_3 + F_2)] + [(F_3 + F_2)+(F_2 + F_1)]$
$= [[(F_3 + F_2)+(F_2 + F_1)] + (F_2 + F_1)+ F_2] + [[(F_2 + F_1)+ F_2]+(F_2 + F_1)]$
$= [[[(F_2 + F_1)+ F_2]+(F_2 + F_1)] + (F_2 + F_1)+ F_2]$
$+ [[(F_2 + F_1)+ F_2]+(F_2 + F_1)]$

Douze F ne sont pas les cas de base F_1 et F_2 ; chacun de ceux-ci nécessite une récursivité. C'est un affreux
beaucoup, mais ça ne sonne pas si mal. D'un autre côté, considérons le nombre suivant, $F_8 = F_7 + F_6$.
Nous avons déjà compté 12 récursions pour F_7 , et vous pouvez regarder vers le haut pour voir que F_6 nécessite 7. Donc F_8
voudra 12+7+1 = 20 récursions. (Nous avons besoin de 1 supplémentaire pour l'expansion de F_7 lui-même.) Vous pouvez voir

immédiatement que le nombre de récursions de la séquence de Fibonacci croît dans un Fibonacci-like
mode:
0,0,1,2,4,7,12,20,33,54,...
Déjà F_{10} nécessite $54 \approx 5 \times 10$ récursions ; remarquez le saut de taille à partir de F_7, qui nécessitait
« seulement » 12 2×7. Avec la suite de Fibonacci, le nombre de récursions croît trop vite !
Le calcul du 100e nombre de Fibonacci peut donc sembler hors de portée, mais il existe plusieurs façons de contourner cette énigme. Les deux premières approches pour résoudre ce problème reposent sur
le fait que lors du calcul de F_7, nous avons calculé F_5 deux fois :
$F_7 = F_6 + F_5 = (F_5 + F_4) + F_5$.
Nous avons ensuite calculé F_4 trois fois :
$F_7 = [(F_4 + F_3) + F_4] + (F_4 + F_3)$.
...et ainsi de suite. Le problème n'est donc pas que le Fibonacci nécessite intrinsèquement une quantité de récursivité; c'est que nous gaspillons beaucoup de calculs que nous pourrions réutiliser.

6 Pour ceux qui connaissent un peu l'informatique, on parle de pile.

7 Que vous vouliez le boire est une autre histoire, mais avant que les auteurs n'entrent dans un autre argument sur la caféine
boissons, nous nous arrêterons là.

Mise en cache. Une façon d'éviter de gaspiller des calculs est de les « s'en souvenir » dans ce qu'on appelle
une cachette. Vous pouvez le voir de cette façon : nous stockons les résultats précédents dans le cache. Pour mettre en œuvre le
cache, nous utilisons une variable globale ; c'est-à-dire une variable qui existe en dehors d'une procédure, que nous pouvons
accès depuis l'intérieur. La raison en est que si le cache était simplement local à la procédure, alors les mêmes valeurs ne seraient pas disponibles lorsque nous aborderons le sous-problème.
Nous décrivons deux façons de mettre en cache les résultats. L'un d'eux est de le faire explicitement, avec une liste dont
La ième valeur est le ième nombre de Fibonacci. Nous initialisons la liste comme F = [1,1]. Chaque fois que nous calculons
un nombre de Fibonacci, nous vérifions d'abord si F a déjà la valeur. Si c'est le cas, nous le retournons
valeur plutôt que de récidive. Si ce n'est pas le cas, nous le calculons, puis l'ajoutons à F à la fin. Par exemple,
lors du calcul de F_5 :

• Nous devons calculer F_4 et F_3 . Nous n'avons ni l'un ni l'autre, alors on récidive.
– Pour F_4 , nous devons calculer F_3 et F_2 . On n'a pas F_3 , donc on récidive.
* Pour F_3 , nous devons calculer F_2 et F_1 . On a les deux, on calcule donc $F_3 = F_2 + F_1 = 1+1 = 2$ et mémoriser cela à la fin de la liste F, qui est maintenant [1,1,2].
Nous le retournons ensuite
* Pour F_2 , nous l'avons déjà, donc nous retournons $F_2 = 1$.
* Nous avons maintenant F_3 et F_2 , donc nous calculons $F_4 = F_3 + F_2 = 2 + 1 = 3$, stockons que
valeur à la fin de la liste et la renvoyer.
– Pour F_3 , nous l'avons maintenant dans la liste (il a été calculé ci-dessus) donc nous le retournons simplement.
• Nous avons maintenant F_4 et F_3 , donc nous calculons $F_5 = F_4 + F_3 = 3 + 2 = 5$, stockons cette valeur au
fin de la liste, et le retourner.
Cela nous a épargné la peine de calculer F_4 plus d'une fois, et de calculer F_3 plus d'une fois. De plus, la définition des nombres de Fibonacci convient à cette méthode : lorsqu'elle
vient le temps de stocker F_n , nous avons déjà F_1 , F_2 , ..., F_{n-1}
dans la liste, afin que nous puissions le stocker dans le
bon endroit.[8] Pour écrire un pseudo-code pour cela, nous introduisons un nouveau « mot en gras » pour notre pseudo-code :
global. Ceci est en tête d'une liste de variables globales utilisées par l'algorithme ; pour plus de clarté, nous l'énumérons après
entrées et sorties.

8 Toutes les séquences récursives n'ont pas cette chance. Si la séquence était définie par $S_n = S_{n-2} + S_{n-3}$, par exemple, vous auriez
il faut s'assurer que le cache a la bonne longueur avant de le stocker. Nous avons donc un peu de chance avec la séquence de Fibonacci.

---

algorithme cached_fibonacci
contributions
• n avec n 1
les sorties
• F_n , le nième nombre de Fibonacci
mondiales
• F, une liste de nombres de Fibonacci (dans l'ordre)
fais
si n |F|
soit résultat = F_n
autre
soit résultat = cached_fibonacci(n −1) + cached_fibonacci(n −2)

ajouter le résultat à F
résultat de retour
Pour transformer cela en pseudocode, nous utilisons le mot-clé Sage `global` , qui indique que `F` est une variable
qui vit en dehors de la procédure. Ce n'est pas strictement nécessaire pour Sage, mais cela permet d'alerter les lecteurs
qu'une variable globale est attendue.

```
sage : F = [1,1]
sage : def cached_fibonacci(n):
cache global F # des valeurs précédentes
# si déjà calculé, retourne-le
si n <= longueur(F) :
résultat = F[n-1]
autre:
# besoin de récidiver :
dommage
résultat = cached_fibonacci(n-1) \
+ cached_fibonacci(n-2)
F.append (résultat)
résultat de retour
```

Notez que nous avons dû ajuster un élément du pseudocode : au lieu de renvoyer `F[n]` , nous re-
tourné `F[n-1]` . Nous l'avons fait parce que le pseudocode suppose que les listes commencent par l'index 1, mais Sage
commence ses indices à l'indice 0.
Vous pouvez tester et comparer les résultats entre les deux. ça deviendra très vite évident que `cached_fibonacci()` est beaucoup, beaucoup plus rapide que `funny_bunnies()` une fois que n devient n'importe quoi
plus grand que le plus petit des nombres :

```
sage : cached_fibonacci(10)
55
sage : cached_fibonacci(25)
75025
sage : cached_fibonacci(35)
9227465
```

Pour vraiment enfoncer le clou, nous en calculerons un qui prendrait beaucoup trop de temps avec le récursif
la mise en oeuvre:

```
sage : cached_fibonacci(100)
354224848179261915075
```

Ne négligez pas la première ligne du code ci-dessus, où nous définissons `F = [1,1]` . Si tu l'oublies,
vous rencontrerez une erreur. Si vous étiez un bon lecteur et que vous n'avez pas fait cette erreur, nous

simulez l'erreur en réinitialisant F , puis en exécutant cached_fibonacci :

```
sage : réinitialiser('F')
sage : cached_fibonacci(10)
NameError :
le nom global 'F' n'est pas défini
```

Si vous étiez toujours un bon lecteur et réinitialisez F pour voir ce qui se passerait, et maintenant vous voulez l'exécuter
encore une fois, attribuez-le à nouveau :

```
sage : F = [1,1]
sage : cached_fibonacci(10)
55
```

Deux derniers mots aux sages. Bien que cet algorithme soit mis en cache, il est toujours récursif ; nous n'avons pas
changé le fait que, lorsque nous rencontrons un « manque de cache » (n est plus grand que la taille de F), le
l'algorithme doit s'appeler lui-même. Étant donné que Sage a une limite intégrée pour la récursivité, nous rencontrons toujours le
même problème que nous avons fait avec pascals_row :

```
sage : cached_fibonacci(1000)
Erreur d'exécution:
profondeur de récursivité maximale dépassée lors de l'appel d'un
objet Python
```

Vous pouvez réellement changer cela, mais c'est dangereux, nous ne vous dirons donc pas comment. Il y a généralement
de meilleures options, de toute façon - pour en trouver une, voir la section suivante.
Enfin, vous n'avez généralement pas à implémenter vous-même la mise en cache. Dans le cloud, Sage
vous offrir une cache gratuitement ; tapez simplement @cached_function puis lancez la définition de la procédure sur
la ligne ci-dessous.

```
sage : @cached_function
def décoré_fibonacci(n):
si n == 1 ou n == 2 :
résultat = 1
autre:
résultat = décoré_fibonacci(n-1) \
+ décoré_fibonacci(n-2)
résultat de retour
sauge : décoré_fibonacci(100)
354224848179261915075
```

Notez que nous avons obtenu le même résultat qu'avec notre cache maison.
Transformer la récursivité en boucle. Une autre façon d'éviter de gaspiller des calculs est de reformuler
tard la récursivité comme une boucle. Laissez b mat compter les paires de lapins matures,

laissez b_{tot} compter le total
paires de lapins, et laissez $b_{ensuite}$ compter les paires de lapins auxquelles nous pouvons nous attendre la saison prochaine. Depuis le
les lapins sont immortels, b_{next} comptera tous les b_{tot} des lapins que nous avons maintenant, mais depuis chaque maturité
paire produit une nouvelle paire, b_{next} doit également compter toutes les paires matures b_{mat} deux fois pour tenir compte de la prochaine
les nouvelles paires de la saison. Donc
$b_{suivant} = b_{tot} + b_{mat}$.
Une fois la prochaine saison arrivée, b_{tot} sera le nombre de lapins matures qui produisent maintenant
paires, tandis que b_{next} sera le nouveau total de lapin, nous pourrions donc remplacer b_{mat} par b_{tot} et b_{tot} par b à $_{côté}$ de
calculer le total de la saison suivante. Et ainsi de suite. Puisque nous connaissons les valeurs des deux premières saisons
déjà, nous n'avons jamais besoin de les compter. En gardant à l'esprit que n = 1 est la première saison et n = 2
la seconde, nous avons besoin de la boucle pour calculer b_{tot} pour n = 3, n = 4, ... Cela suggère ce qui suit
pseudo-code.
algorithme immortal_bunnies
contributions
• $n \in$, le nombre de saisons
les sorties
• le nombre de lapins « immortels » après n saisons
fais
soit $b_{mat} = b_{tot} = 1$
répéter n -2 fois
soit $b_{suivant} = b_{tot} + b_{mat}$
soit $b_{mat} = b_{tot}$
soit $b_{tot} = b_{suivant}$
retour b_{tot}
Pour traduire cela en code Sage, gardez à l'esprit que range() n'exécute pas d'itération sur la dernière valeur, nous
besoin que sa limite soit $_{n-1}$. Il est également plus « naturel » dans Sage de commencer à compter à n = 0, donc la boucle
différera légèrement du pseudocode :

---

sage: def immortal_bunnies(n):
b_mat = b_tot = 1

```
# boucle ascendante
pour chacun dans la plage (n-1) :
b_next = b_tot + b_mat
b_mat, b_tot = b_tot, b_next
retour b_tot
sage : [immortal_bunnies(i) for i in range(2,10)]
[2, 3, 5, 8, 13, 21, 34, 55]
```

Contrairement à la mise en cache, Sage ne fournit pas de méthode automatique qui transforme une procédure récursive
dans une procédure en boucle. Faire cela est quelque chose que nous devons travailler par nous-mêmes, raisonner
de la logique du problème. Comme cela est considéré comme un sujet avancé en informatique, nous faisons
ne pas s'y attarder davantage. Le lecteur intéressé peut trouver de plus amples informations en informatique
textes qui traitent de la « programmation dynamique ». [9]

Les valeurs propres et les vecteurs propres résolvent un dilemme de lapin. Vous n'êtes peut-être pas familier avec
valeurs propres et vecteurs propres, même si vous les avez vus dans une classe. Nous concluons ce chapitre par
une illustration de l'immense utilité qu'ils peuvent être.

Comme vous l'avez probablement remarqué, même les implémentations non récursives du système de Fibonacci
s'appuyer sur sa description récursive, dont la nature récursive nous oblige à connaître
nos valeurs. Ce serait bien d'avoir une formule qui ne dépende pas des valeurs précédentes, du moins pas
explicitement. Une telle formule est appelée une forme fermée d'une séquence.

Une autre façon de décrire cela est une équation matricielle. Les matrices

F_{curr}

$F_{précédent}$

et

$F_{suivant}$

F_{curr}

représentent les informations dont nous avons besoin à la fois cette saison et la saison prochaine pour calculer les éléments suivants
lapins de saison. Maintenant, la somme des nombres dans la première matrice nous donne l'élément supérieur de
la deuxième matrice, tandis que le numéro du haut de la première matrice nous donne le numéro du bas de la
deuxième. Nous pouvons décrire cette relation à l'aide d'une équation matricielle :

1 1

dix

F_{curr}

$F_{précédent}$

=
F suivant
F curr
.
De plus, en multipliant par les puissances de la matrice
M =
1 1
dix
clique les nombres plus haut dans la liste. En être témoin,
1 1
dix
5
1
1
=
13
8
,
9 L'inventeur de la programmation dynamique a dit un jour qu'il avait inventé le terme parce que son entreprise
répondit à un homme qui abritait «une peur et une haine pathologiques du mot recherche. ...Tu peux imaginer
comment il se sentait, alors, à propos du terme mathématique.» L'utilisation d'un terme comme « programmation dynamique » l'a aidé
éviter la colère de son supérieur. Il a ajouté : « Je pensais que la programmation dynamique était un bon nom. . . ne pas
même un membre du Congrès pourrait s'y opposer.

---

que vous reconnaîtrez comme les 4e et 5e numéros de la liste produite par notre programme, qui
sont F 6 et F 7 , c'est-à-dire le nombre de lapins aux 6e et 7e saisons. En général,
1 1
dix
n−2
1
1
=
F n
F
n-1
,

car après tout
1 1
dix
1
1
1
=
2
1
,
1 1
dix
2
1
1
=
1 1
dix
1
2
1
=
3
2
,
et ainsi de suite.
Si nous pouvons trouver une formule simple pour les puissances de la matrice, nous devrions pouvoir l'utiliser pour trouver
notre forme fermée désirée. En l'occurrence, un résultat très commode de l'algèbre linéaire vient dans
pratique ici. Un vecteur propre e d'une matrice M sur un anneau R est un vecteur lié à une valeur propre
$\lambda \in$ R tel que
Me = λ e;
c'est-à-dire que M change la « longueur » du vecteur. Les vecteurs propres et leurs valeurs propres ont un nombre
d'usages, dont l'un vient désormais à notre secours.
E IGENDECOMPOSITION T HEOREM .. Soit M une matrice n × n avec
vecteurs propres l' e $_1$, ..., e $_n$ et des valeurs propres correspondant X_1 , ..., λ_n . Nous pouvons réécrire M comme Q Λ Q $^{-1}$ où
Q = (e $_1$ |e $_2$ |···|e $_n$) et Λ =
⎛
|

⎜
⎜
⎝
λ_1
λ_2
...
λ_n
⎛
⎜
⎜
⎜
⎞
.
(La notation pour Q signifie que la ième colonne de Q est le ième vecteur propre de M.)
Il est facile de le vérifier dans Sage for M. Calculons d'abord les vecteurs propres.

```
sage : evecs = M.eigenvectors_right()
sauge : evecs
[(-0.6180339887749895?, [(1, -1.6180339887749895?)], 1),
(1.6180339887749895?, [(1, 0.6180339887749895?)], 1)]
```

Qu'est-ce que tout cela signifie? Si vous étudiez le texte d'aide...

```
sage : M.eigenvectors_right ?
```

... vous verrez que la liste contient un tuple par vecteur propre, et le tuple lui-même contient le valeur propre, le vecteur propre et la multiplicité de la valeur propre.
Pour nos besoins, les valeurs propres et les vecteurs propres suffisent. Il est relativement facile de les extraire,
mais
• nous voulons des valeurs exactes, pas approximatives, et
• que signifient ces points d'interrogation, de toute façon ?
Cela peut vous surprendre d'apprendre que les points d'interrogation n'indiquent pas l'incertitude. Ils indiquent
que les nombres se trouvent dans un domaine spécial appelé le domaine des nombres algébriques. Sage désigne ce champ
avec le symbole *AA* .

```
sage : a, b = evecs[0][0], evecs[1][0]
sauge : type(a)
<classe 'sage.rings.qqbar.AlgebraicNumber'>
```

Les nombres algébriques sont le plus petit corps contenant toutes les racines de polynômes avec un entier
coefficients, de sorte que nous pouvons toujours trouver un polynôme « joli » qui les a comme racine. En addition-
dition, peut souvent les réécrire comme une expression en radicaux. Les méthodes suivantes pour algébrique

les numéros effectuent ces tâches :
un .minpoly()
renvoie le polynôme de plus petit degré qui
a comme racine
a .radical_expression() renvoie une expression radicale équivalente à un
Vous pouvez trouver beaucoup plus de méthodes de la manière habituelle, en tapant le nom d'une variable dont
value est un nombre algébrique, suivi d'un point, puis appuyez sur la touche Tab .
Pour l'instant, essayons-les sur notre vecteur propre actuel.

```
sage : a.minpoly()
x^2 - x - 1
sage : a.expression_radicale()
-1/2*carré (5) + 1/2
sage : b.minpoly()
x^2 - x - 1
sage : b.expression_radicale()
1/2*carré (5) + 1/2
```

Les deux nombres ont le même polynôme minimal,[10] mais leurs expressions radicales sont différentes.
Ils ne sont pas si différents, cependant; nous les appellerons[11]
$\psi =$
$1- \sqrt{5}$
2
et $\varphi =$
$1+ \sqrt{5}$
2
.
Le second en particulier est très célèbre, car c'est le nombre d' or. Il apparaît dans de nombreux
différents endroits en mathématiques, et sans doute dans de nombreuses situations du « monde réel ». Notre Fibonacci
les lapins peuvent ne pas sembler une situation particulièrement convaincante dans le «monde réel», mais ce nombre suggère
que les chiffres peuvent apparaître dans des situations « du monde réel », et en effet ils le font.
Nous revenons à notre problème principal, celui de trouver une forme fermée pour la suite de Fibonacci.
Nous avons d'abord voulu vérifier le théorème de composition propre. Pour construire les matrices Q et Λ , nous
extraire les données d' evecs à l'aide de l'opérateur crochet [] , puis construire les matrices à l'aide de compre-
hensions. Assurez-vous de bien comprendre comment nous construisons ces matrices.

10 Cela a du sens si vous avez étudié cela. Si ce n'est pas le cas, surveillez bientôt un cours d'algèbre linéaire dans un semestre près de chez vous !

11 Quelques anecdotes. Puisque ϕ est une racine de $x^2 - x - 1$, on sait que $\phi^2 - \phi - 1 = 0$, donc $\phi^2 = \phi + 1$, ou $\phi = \phi + 1/\phi = 1 + 1/\phi$,

ou $1/\varphi = \varphi - 1$. Notez que $\psi = 1 - \varphi$, donc $-\psi$ et φ sont en fait inverses.

```
sage: Q = matrice([e[1][0] pour e dans evecs]).transpose()
sage : L = diagonal_matrix([e[0] pour e dans E])
sage : M == Q*L*Q.inverse()
Vrai
```

Donc le théorème fonctionne. En quoi est-il utile à notre objectif ? Rappelez-vous que

$$M^{n-2} \begin{matrix} 1 \\ 1 \end{matrix} = \begin{matrix} F_n \\ F_{n-1} \end{matrix}.$$

Par substitution, cela devient

$$M^{n-2} = \underbrace{Q \Lambda Q^{-1} \cdots Q \Lambda Q^{-1}}_{n-2 \text{ fois}}.$$

Les propriétés associatives nous permettent de réécrire cela comme

$$M^{n-2} = (Q \Lambda) Q^{-1} Q \Lambda Q^{-1} Q \cdots Q^{-1} Q \Lambda Q^{-1}.$$

Cela simplifie à

$$M^{n-2} = Q \Lambda^{n-2} Q^{-1}.$$

De plus, il est « facile » de montrer que pour une matrice diagonale comme Λ

$$X^k = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \cdots \\ \lambda_n \end{pmatrix}$$

k
=
⎛
⎜
⎜
⎜
⎝
λ k
1
λ k
2
⋱
λ k
m
⎞
⎟
⎟
⎟
⎠
,
nous pouvons donc écrire Λ k sous une forme plus simple.

```
sage: Lk = diagonal_matrix([L[i,i]^n pour i dans la plage(L.nrows())])
Erreur-type:
pas de coercition canonique de Symbolic Ring à Rational
Domaine
```

Eh bien, c'est étrange. Et si on lui donnait d'abord une expression radicale ?

```
sage: Lk = diagonal_matrix([L[i,i].radical_expression()^(n-2) \
pour i dans la plage (L.nrows())])
sage : Lk
[(-1/2*carré (5) + 1/2)^(n-2)
0]
[
0 (1/2*carré (5) + 1/2)^(n-2)]
```

Ça a marché. Sage semble vouloir qu'une expression radicale procède ; nous pouvons nous adapter à cela avec Q,
ainsi que.

DES EXERCICES
207

```
sage: Q = matrice([[Q[0,0].expression_radicale(), \
Q[0,1].expression_radicale()], \
[Q[1,0].expression_radicale(), \
Q[1,1].expression_radicale()] \
])
sage : Q
[
1
1]
[-1/2*carré (5) - 1/2 1/2*carré (5) - 1/2]
```

Nous pouvons maintenant calculer directement Q Λ k Q −1 .

```
sage : Q*Lk*Q.inverse()*matrice([[1],[1]])
[ 1/20*carré (5)*(1/2*carré (5) + ...
```

. . . eh bien, c'est un peu le bordel. En fait, tout ce qui nous intéresse, c'est l'élément supérieur, et nous préférerions l'avoir
entièrement simplifié.

```
sage : (Q*Lk*Q.inverse()*matrix([[1],[1]]))[0,0].full_simplify()
1/5*sqrt(5)*((1/2*sqrt(5) + 1/2)^n - (-1/2*sqrt(5) + 1/2)^n)
```

Eh bien, c'est pratique ! C'est
1
5
5
1
2
5+
1
2
m
− −
1
2
5+
1
2
m
,
donc avec une légère réécriture on trouve que
F n =
1
5
($\phi_n - \psi_n$),
une forme fermée terriblement élégante !

Des exercices

Vrai faux. Si la déclaration est fausse, remplacez-la par une déclaration vraie.

1. Un algorithme récursif à un problème est celui où l'algorithme s'invoque sur un plus petit cas du problème.
2. Un algorithme récursif doit essayer d'exploiter la propriété de bon ordre des entiers ().
3. Les lignes du triangle de Pascal apparaissent à de nombreux endroits, comme dans le développement des trinômes.
4. Comme le triangle de Pascal n'a qu'une seule récursivité, il n'y a aucun risque de rencontrer une *RuntimeError* .
5. La croissance d'une population de lapins est l'une des propriétés fondamentales des mathématiques.
6. Chaque instruction récursive peut être résolue à l'aide d'une procédure facile à mettre en œuvre.

7. Si un problème peut être résolu par récursivité, alors l'implémenter par récursivité est la meilleure ap-
approcher.
8. Une récursivité très profonde peut indiquer que quelque chose a mal tourné, comme une boucle infinie.
9. Vous pouvez contourner le retard de calcul du 40e nombre de Fibonacci en achetant un l'ordinateur.
10. Un cache est une variable locale qui stocke les résultats précédents de la procédure.

---

11. Sage fournit un décorateur `@cached_function` qui met automatiquement en cache le précédent d'une procédure
résultats.
12. Vous pouvez parfois transformer une procédure récursive en boucle, évitant ainsi les pénalités associées.
avec récursivité.
13. Comme le nombre d'or est unique aux nombres de Fibonacci, ils sont d'un intérêt purement théorique.
14. La composition propre nous permet de réécrire une matrice M en termes d'autres matrices qui sont elles-mêmes
eux-mêmes liés à des vecteurs que M redimensionne.
15. Vous pouvez obtenir une représentation radicale d'un nombre algébrique en utilisant `.radical_simplify()`
méthode.

Choix multiple.

1. La propriété de bon ordre s'applique à lequel de ces ensembles ?
UNE.
B.
C.
RÉ.
2. Le triangle de Pascal est un bon exemple de récursivité car :
A. Il a de nombreuses applications.
B. Chaque ligne peut être définie en fonction de la ligne précédente.
C. C'est l'un des problèmes fondamentaux des mathématiques.
D. Il fait une critique efficace de la philosophie du doute systématique de Descartes.[12]
3. Les nombres de Fibonacci sont un bon exemple de récursivité car :
A. Ils ont de nombreuses applications.
B. Chaque numéro peut être défini en fonction des numéros précédents.
C. Ils se rapportent à l'un des problèmes fondamentaux des mathématiques.
D. Si vous pouvez obtenir une séquence de nombres portant votre nom, vous devez avoir raison - vous venez de

devoir!

4. Quelle technique de preuve est essentiellement une sorte de récursivité ?

A. contradiction

B. contraposée

C. induction

D. test d'hypothèse

5. Que se passe-t-il dans Sage si vous programmez accidentellement une récursivité infinie ?

A. l'ordinateur explose

B. le programme se bloque

C. *Erreur d'exécution*

D. *RecursionError*

6. Pourquoi est-ce en fait une mauvaise idée d'implémenter la séquence de Fibonacci avec une récursivité naïve ?

A. Une implémentation naïve répète la grande majorité de ses calculs.

B. À l'exception des cas de base, chaque récursivité double le nombre de calculs nécessaires.

C. Le temps nécessaire pour calculer les nombres augmente à peu près aussi vite que les nombres eux-
moi-même.

Tout ce qui précède.

7. Laquelle des alternatives suivantes à la récursivité devez-vous déterminer en analysant les algorithme, plutôt que d'utiliser une combinaison de commandes ou de variables Sage ?

12 Référence gratuite et non pertinente à la philosophie moderne incluse sans frais supplémentaires.

A. mise en cache

B. trouver une forme fermée

C. induction

D. bouclage

8. Lequel des énoncés suivants définit le mieux l'anneau des nombres algébriques ?

A. *AA*

B. l'ensemble de toutes les racines de polynômes à coefficients entiers

C. le plus petit corps contenant les racines de polynômes à coefficients entiers

D. le plus petit champ contenant tous les nombres irrationnels

9. La meilleure commande ou méthode pour transformer un nombre algébrique en un nombre exact, mais facile à lire
l'expression numérique est :

A. .exactif ()

B. .full_simplify()

C. .minpoly()

D. .expression_radicale()

10. Lequel des énoncés suivants décrit le mieux la signification d'un vecteur propre ?

A. un vecteur utile pour trouver des formes fermées de séquences
B. un vecteur que la matrice redimensionne, mais part dans la même direction
C. un vecteur linéairement indépendant des autres vecteurs propres
D. un vecteur qui peut être utilisé pour décomposer une matrice en une forme informatiquement utile

Prime. Étant donné la définition récursive d'une séquence de lapins mathématiques, la meilleure façon de trouver
une « forme fermée » pour décrire la séquence est de
A. acquérir des chiens
B. les emballer
C. les clôturer
D. les habiller
E. Enfermez-les dans des combinaisons spatiales, comme Glenda :[13]

Réponse courte.

1. Pourquoi pascals_row n'aurait-il pas autant de problèmes de récursivité que la séquence de Fibonacci ?
C'est-à-dire quelle différence dans le pseudocode (et, par conséquent, le code Sage) fait pascals_row
plus récursive que la suite de Fibonacci ?
2. Un cache serait-il aussi utile pour pascals_row que pour la séquence de Fibonacci ? Pourquoi ou pourquoi
ne pas?
3. Retour à la p. 77 nous vous avons demandé de tracer x, x^2, $\log_{10} x$ et e^x et de les classer dans l'ordre du plus grand
au plus petit. Recréer ce graphe sur l'intervalle [1,15] en y ajoutant une courbe dont les points sont
défini par les nombres de Fibonacci de 1 à 15. Encore une fois, classez ces fonctions selon qui croît le plus vite. Astuce : rappelez-vous que la procédure `line()` trace une « courbe » définie par un
ensemble fini de points. Vous devrez peut-être ajuster `ymax` afin de bien visualiser le tracé.

[13] Téléchargé à partir du site Web Plan 9 de Bell Labs et utilisé avec permission (nous pensons – le libellé est un peu vague).

Programmation.

1. Supposons $y = x^m q$, où x ne divise pas q. Par exemple, si y = 12 et x = 2, alors nous avons m = 2. Nous appelons m la multiplicité de x, et pouvons la calculer en utilisant la récursive suivante
algorithme:
multiplicité d'algorithmes
contributions

• x et y, deux objets tels que « x divise y » a du sens
les sorties
• le nombre de fois où x divise y
fais
si x ne divise pas y
résultat = 0
autre
résultat = 1 + multiplicité(y/x)
résultat de retour
(a) Mettre en œuvre ceci en tant que programme Sage. Vérifiez qu'il donne les bons résultats pour les
ment :
(i) y = 12, x = 2
(ii) y = 12, x = 3
(iii) $y = t, x = t^4 + t^2$
(b) Il est possible de résoudre ce problème de manière non récursive, en utilisant une boucle while. Écrire un pseudo-code
pour un tel algorithme, puis implémentez-le dans Sage. Testez votre programme sur le même examen-
pls.
2. Réécrivez pascals_row pour qu'il utilise un cache manuel. N'utilisez pas la capacité `@cached_function` de Sage
capacité; utilisez plutôt une liste où la ième entrée est aussi une liste : la ième ligne du triangle de Pascal.
3. Nous pouvons compter à quel point la récursivité d'une procédure est compliquée avec une variable globale appelée `invocations` .
Nous ajoutons deux lignes au début de la procédure récursive qui déclarent que les `invocations` sont
`global` , puis ajoutez-y 1. Par exemple, nous pourrions le faire avec `funny_bunnies()` comme suit :

```
sage : def funny_bunnies(n):
appels globaux
appels = appels + 1
si n == 1 ou n == 2 :
résultat = 1
autre:
résultat = funny_bunnies(n-1) + funny_bunnies(n-2)
résultat de retour
```

Maintenant, chaque fois que nous l'exécutons, nous définissons d'abord les `invocations` à 0, puis exécutons le programme, puis imprimons
`invocations` . Par example:

```
sage : invocations = 0
```

```
sage: funny_bunnies(7)
13
sage : invocations
25
```

(Ce n'est pas le même que le nombre de récursions.)

(a) Faites ceci pour les valeurs n = 7, 8, 9, 10 et 11. Diriez-vous que cette séquence se comporte comme les nombres de Fibonacci, comme on l'a vu avec le nombre de récursions ? Si oui, trouvez un formule récursive pour décrire la régularité des nombres. Si non, comment décririez-vous il?

(b) Modifiez le programme pascals_row pour calculer le nombre d'appels, puis exécutez-le pendant quelques valeurs de n. Le nombre d'invocations dans ce cas est-il semblable à celui de Fibonacci ? Si oui, trouvez un formule récursive pour décrire la régularité des nombres. Si non, comment décririez-vous il?

4. John von Neumann a montré qu'il est possible de représenter chaque nombre naturel en termes de les symboles { } et :

0
1 ⇔ { }
2 ⇔ {,{ }}
3 ⇔ {,{ },{{,{ }}}}
...
n ⇔ {n -1,{n -1}}

Comme vous pouvez le voir à partir de la dernière ligne en particulier, il s'agit d'une définition récursive :

$n = (n-1) \cup \{n-1\}$.

Implémentez cela en tant que code Sage.

Astuce: Ce n'est pas difficile, mais vous devez être prudent. Premièrement, la nature récursive de l'algorithme signifie que la procédure doit retourner un ensemble gelé. Pour créer un ensemble gelé R qui contient un autre l'ensemble gelé S, ainsi que d'autres éléments, créez d'abord R en tant qu'ensemble mutable (`R = set()`), ajoutez-y S, ajoutez les autres éléments (en utilisant `.add()` , `.update()` , et ainsi de suite selon le cas), et enfin convertir `R` à un ensemble gelé.

5. Supposons que vous ayez un ensemble de n objets et que vous vouliez répertorier ses sous-ensembles de m éléments (c'est-à-dire ses sous-ensembles avec exactement m éléments).[14] L'algorithme suivant accomplirait cela pour vous :

[14] Cela peut sembler être le genre de question dont seuls les mathématiciens les plus purs se soucieraient, mais c'est

en fait lié à la raison pour laquelle vous semblez ne jamais acheter de ticket gagnant pour PowerBall.

DES EXERCICES
212
combinaisons d'algorithmes
contributions
• S, un ensemble de n objets
• m, le nombre d'objets que vous voudriez choisir dans S, avec m n
les sorties
• les sous-ensembles de S de taille n
fais
si |S| = m
laisser résultat = {S}
autre
laisser résultat =
pour s S
soit U = S\s
ajouter des combinaisons (U, m) au résultat
résultat de retour
(a) Mettre en œuvre ceci en tant que programme Sage.
Astuce: Ce n'est pas difficile, mais vous devez être prudent. Premièrement, la nature récursive de l'algorithme
rithm signifie que le résultat doit être un ensemble figé. Cependant, la création d'un ensemble gelé contenant un ensemble
s ne fonctionne pas si vous utilisez freezeset(S) , car cela crée un ensemble gelé dont les éléments sont les éléments de s , convertissant essentiellement s d'un ensemble mutable en un ensemble gelé, plutôt
que de créer un ensemble figé dont l'un des éléments est s . Pour créer un ensemble figé dont l'élément est
un autre ensemble, créez d'abord un ensemble mutable, ajoutez-y un ensemble gelé, puis convertissez l'ensemble mutable
à un ensemble gelé. Essentiellement, la ligne du premier cas du pseudocode se transforme en trois
lignes de code Sage.
(b) Soit S = {1,2,3,4,5,6,7,8} et évalue la taille des combinaisons (S, i) pour chaque i = 0,...,8. Les tailles résultantes apparaissent plus tôt dans le texte. Où?
(c) Pensez-vous qu'il s'agit d'un accident anormal, ou est-ce vrai en général ? Qu'est-ce qui te fait dire ça ?

CHAPITRE 10

Faire vos photos en 3 dimensions

Ce chapitre examine les objets et les graphiques tridimensionnels que vous pouvez utiliser.
La plupart des commandes de deux dimensions ont une extension 3D, avec quelques options car-
en train de pleurer aussi. Comme précédemment, nous n'illustrons pas toutes les options, en espérant que les options
changement dans les futures versions de Sage.
La combinaison de graphiques se fait à nouveau par addition, et la commande `show()` s'applique également à
tracés 3D. Pour les tracés 3D, la commande `show()` ajoute une option qui vous permet de sélectionner un "viewer"
ou « rendeur » :
• Lorsque vous utilisez `show ()` comme une commande, comme `show (p, ...)` , utilisez la `renderer` option de sé-
lectez l'un de `'webgl'` (le plus rapide), `'canvas'` (peut mieux fonctionner avec la transparence) ou `'tachyon'` .
Cette dernière option donne une image statique, alors que `'webgl'` et `'canvas'` sont interactifs.
• Lorsque vous utilisez `show()` comme méthode, telle que `p.show(...)` , utilisez l' option `viewer` pour sélectionner
l'un de `'jmol'` (nécessite Java), `'java3d'` (nécessite également Java), `'canvas3d'` (navigateur uniquement,
utilise JavaScript), et encore une fois `tachyon'` . Encore une fois, tous sauf le dernier sont interactifs.
Qu'entendons-nous par « interactif » ? La nouveauté des tracés 3D est que les graphiques peuvent être ma-
nipulé – vous pouvez changer de point de vue ! Pour ce faire, cliquez pour l'activer et
puis en cliquant et en faisant glisser pour modifier l'angle de vue. Vous pouvez également zoomer en faisant défiler. Autre
les options de visualisation sont disponibles via un menu accessible par un clic droit ; on vous laisse découvrir
ceux.

Objets 3D

Des trucs "droits". La plupart des procédures de représentation graphique 2D que nous avons étudiées précédemment ont un
procédure correspondante en 3 dimensions : `point()` , `line()` , `arrow()` , `polygon()` . Étant donné que tout
ceux-ci nécessitent un ou plusieurs points, Sage reconnaîtra que les points sont donnés en 3D et
les représenter graphiquement de manière appropriée. De nombreuses options sont reportées.
Si nous donnons à la commande `point()` un ou plusieurs triplets (plutôt que des paires), il tracera des points, chacun
dont les coordonnées sont déterminées par un triplet répertorié. Les lignes sont à nouveau

tracées avec la ligne ()
commander. En 3D, la commande line() a une option arrow_head ; s'il est défini sur True, la pointe de la flèche
n'est affiché qu'au dernier point.

```
sage : pts = ((-2,1,3),(4,2,-1),(2,-3,1))
sage : p1 = point(pts,couleur='rouge',taille=20)
sage: p2 = line(pts,color='blue',thickness=7,arrow_head=True)
sage : p1+p2
```

213

OBJETS 3D
214

Les polygones sont représentés graphiquement en 3D comme en 2D avec la commande polygon() , ce qui nécessite une collecte
tion de points. Les seules options disponibles en 3D sont la couleur et l' opacité . Voici un triangle :

```
sage: polygone([(1,0,2),(3,1,5),(4,2,1)],color='orange')
```

Des trucs "courbés". La procédure de cercle peut également produire des cercles en 3 dimensions, nécessitant à nouveau
un point central et une taille. Les cercles tracés avec la commande circle() seront placés parallèlement à
le plan xy ; si vous voulez d'autres orientations, vous devrez utiliser d'autres méthodes décrites plus loin.
Il n'y a pas d' options de couleur de bord ou de couleur de face pour les cercles en 3D. Comme en 2D, si vous voulez un disque, réglez
l' option de remplissage à True.

```
sauge : cercle((2,1,3),4,épaisseur=10)
```

En 3D, l'ensemble de tous les points équidistants d'un point particulier est une sphère. Nous pouvons représenter graphiquement
sphères à l'aide de la commande sphere() , qui nécessite un centre et un rayon (taille).

PARCELLES 3D DE BASE
215

```
sage: sphere(center=(1,2,.5),size=3)
```

Tracés 3D de base
Si vous avez suivi le calcul multivariable, n'auriez-vous pas souhaité que votre calculatrice de poche puisse
avoir tracé ces surfaces ? Heureusement, Sage le peut ! La plupart des procédures de création de graphiques 2D qui
nous avons étudié précédemment ont une procédure correspondante en 3 dimensions. Il existe plusieurs façons de
représentent des relations tridimensionnelles :

• Coordonnées cartésiennes avec soit

– z en fonction de x et y ;
– une équation implicite en termes de x, y et z, mais pas nécessairement une fonction ;
– avec x, y et z en fonction d'une troisième variable t, un paramètre ;
– avec x, y et z en fonction de deux autres paramètres u et v ;
• coordonnées sphériques ; et
• coordonnées cylindriques.
Tracés cartésiens génériques. La relation tridimensionnelle la plus connue définit la valeur de une variable (souvent z) en fonction de deux autres (typiquement x et y). Contrairement à la commande 2D plot() ,
la commande plot3d() n'a pas d'option pour détecter les pôles et afficher les asymptotes.
Dans les quelques exemples suivants, nous représentons graphiquement $f(x,y) = -x/1+x^2+y^2$ avec deux styles de tracés différents et
avec diverses options.

```
sage : f(x,y) = -x/(1+x^2+y^2)
sage : plot3d(f(x,y), (x,-3,3), (y,-3,3), opacité=.8)
```

Définir l' option de maillage sur True produit des graphiques avec une grille xy sur la surface, comme ceux de
de nombreux manuels :

```
sage: plot3d(f(x,y), (x,-3,3), (y,-3,3), opacity=.8, \
color='lightblue',mesh=True)
```

Lorsqu'une partie du graphique 3D est obscurcie, il peut être plus facile de visualiser la même surface à l'aide d'un « con-
parcelle de tournée. Considérez ce graphique comme une vue aérienne du graphique, avec des points de la même couleur ayant
valeurs z dans la même plage, comme indiqué sur la barre de couleur à droite. Avec la couleur 'Spectral'
carte utilisée ci-dessous, les points les plus bas sont en rouge et les points les plus hauts sont en bleu. Le contour
niveaux (les incréments de valeur z indiqués sur la barre de couleur) ont été déterminés automatiquement, mais ils
peut être spécifié à l'aide d'une liste pour l' option contours .

```
sage: contour_plot(f(x,y),(x,-3,3),(y,-3,3), cmap='Spectral', \
barre de couleur=Vrai)
```

Les limites entre les régions de couleur sont appelées "courbes de niveau", ce qui signifie que f (x,y) = k pour
une constante k. Si vous préférez, pour ne voir que les courbes de niveau, utilisez l'option fill= *False* . (Vous serez
notez que de nombreux auteurs de manuels utilisent cette option.)

```
sage : var('y')
sage : contour_plot(f(x,y), (x,-3,3), (y,-3,3), \
cmap='Spectral', fill=False, colorbar=True)
```

PARCELLES 3D DE BASE

217

Plans tangents : combinaison de tracés 3D. Représentons graphiquement une fonction f (x,y) et le plan tangent à
f (x,y) en un point $P_0 = (x_0, y_0, z_0)$. Rappelons que le plan tangent dépend de la première dérivée partielle
tives de f à P_0 et est donnée par

$$z = f_x(x_0, y_0)(x - x_0) + f_y(x_0, y_0)(y - y_0) + z_0,$$

où f_x et f_y sont les premières dérivées partielles. On peut facilement écrire une procédure Sage pour calculer
le plan tangent et représenter graphiquement à la fois f (x,y) et le plan tangent sur la région rectangulaire[a, b]×
[CD]:

```
sage : def plot_f_tanplane(f,pt,a,b,c,d):
f(x,y)=f
x0,y0=pt
fx = diff(f,x)
fy = diff(f,y)
z0 = f(x0,y0)
tanplane = fx(x0,y0)*(x-x0) + fy(x0,y0)*(y-y0) + z0
p1 = plot3d(f,(x,a,b),(y,c,d))
p2 = plot3d(tanplane,(x,a,b),(y,c,d), \
color='limegreen',opacity=.6)
p3 = point((x0,y0,z0),couleur='rouge',taille=30)
renvoie p1 + p2 + p3
```

Maintenant, nous pouvons produire des graphiques de plans tangents pour toute fonction où les dérivées partielles sont
défini. Par example,

```
sage : var('y')
sage : plot_f_tanplane(1-x^2-y^2,(1,-2),-3,3,-3,3)
```

PARCELLES 3D DE BASE

218

Paramétrages en 3D. Une autre façon de tracer des objets en trois dimensions consiste à étérisation. Les courbes sont paramétrées en une seule variable, souvent appelée t. Nous avons vu plus haut que le
La commande `circle()` trace toujours un cercle 3D parallèle au plan xy. Avec le paramétrage, nous
peut orienter les cercles d'autres manières, par exemple, parallèlement au plan xz :

```
sage : var('t')
sage : parametric_plot3d((1,cos(t),sin(t)),(t,0,2*pi),épaisseur=10)
```

Le nombre par défaut de points à tracer est de 75, mais il est souvent nécessaire d'utiliser plus de points pour
lisser le graphique. Voici une « spirale toroïdale » qui n'a pas l'air spécialement spiralée.

```
sage : var('t')
sage : p = parametric_plot3d(((4+sin(25*t))*cos(t), \
```

(4+sin(25*t))*sin(t), cos(25*t)), (t,0,2*pi), \
couleur='rouge', épaisseur=10)
sage : show(p,aspect_ratio=1)

La raison en est qu'il a besoin d'environ 200 points pour le rendre lisse.

sage : var('t')
sage : p = parametric_plot3d(((4+sin(25*t))*cos(t), \
(4+sin(25*t))*sin(t), cos(25*t)), (t,0,2*pi), \
couleur='rouge', épaisseur=10, plot_points=200)
sage : show(p,aspect_ratio=1)

Nous pouvons utiliser la même commande parametric_plot3d() pour représenter graphiquement des surfaces paramétriques. Définir
surfaces paramétriques nécessite deux variables indépendantes, souvent appelées u et v. Par exemple, nous pouvons
représenter graphiquement la surface définie par x = u^3 , y = u sin(v), z = u cos(v) pour $-4 \leq u \leq 4$ et $0 \leq v \leq 2\pi$
en Sauge :

sage : var('u v')
sage : parametric_plot3d((u^3,u*sin(v),u*cos(v)), \
(u,-4,4), (v,0,2*pi), opacité=.8)

Coordonnées cylindriques et sphériques. Surfaces données en coordonnées cylindriques ou sphériques

Les systèmes nés peuvent être considérés comme des types particuliers de paramétrisation. A partir de coordonnées cylindriques
(r, θ , z), on convertit en coordonnées rectangulaires par x = r cos θ , y = r sin θ et z = z. Ci-dessous est
la surface r = sin(z), pour $0 \leq \theta \leq 2\pi$ et 0 z $\leq 2\pi$.

sage : var('u v')
sage: show(parametric_plot3d((sin(u)*cos(v),sin(u)*sin(v),u), \
(u,0,2*pi), (v,0,2*pi)), aspect_ratio=1)

Une autre méthode utilise plot3d() avec une transformation cylindrique. Le Cylindrical() commande, tel qu'il est utilisé ci - après, voir le rayon en fonction de l' azimut (θ) et de la hauteur (z).

sage : S=Cylindrique('rayon', ['azimut', 'hauteur'])
sage : var('theta, z')
sage: show(plot3d(sin(z), (theta,0,2*pi), (z,0,2*pi), \
transformation=S, couleur='violet'), aspect_ratio=1)

Les deux méthodes produisent le même graphique.

La conversion à partir des coordonnées sphériques (ρ , θ , *& phiv*) en coordonnées rectangulaires est donnée par x =
ρ sin φ cos θ , y = *de* sin φ sin *de* θ et z = ρ cos φ . La surface définie par ρ = 1+ $\frac{1}{6}$ sin (5 θ) sin (6 φ)
pour 0 $\leq$ θ $\leq$ 2 π et pour 0 $\leq$ φ $\leq$ π est connu comme une « sphère bosselée » et serait représenté graphiquement dans Sage
par

```
sage : var('theta phi')
sage : rho = 1+(1/6)*sin(5*thêta)*sin(6*phi)
sage : parametric_plot3d((rho*sin(phi)*cos(theta), \
rho*sin(phi)*sin(thêta), rho*cos(phi)), \
(theta,0,2*pi), (phi,0,pi), plot_points=[200,200], \
couleur='citron vert')
```

Encore une fois, nous pouvons former le même graphe en utilisant la commande plot3d() et un trans-
formation. La transformation est donnée par la commande Spherical() , en considérant le rayon comme un
fonction de l'azimut (θ) et de l'inclinaison (φ).

```
sage : T = Sphérique('rayon', ['azimut', 'inclinaison'])
sage : var('phi thêta')
sage : plot3d(1+(1/6)*sin(5*thêta)*sin(6*phi), (thêta,0,2*pi), \
(phi,0,pi), transformation=T, plot_points=[200,200], \
couleur='citron vert')
```

Outils avancés pour les tracés 3D

Sage propose un certain nombre d'outils qui aident à la production de types spéciaux de tracés 3D dans
Coordonnées cartésiennes, cylindriques ou sphériques.
Surfaces de révolution. Un type particulier de surface est une surface de révolution. C'est possible
pour les représenter graphiquement à l'aide de paramétrisations, mais il est certainement utile que Sage dispose d'une procédure
surtout pour de tels graphiques. Utilisez revolution_plot3d() pour les représenter graphiquement en se basant uniquement sur la fonction,
l'axe de révolution et l'intervalle. Rappelons que si nous tournons une fonction f (x) autour de l'axe des x,
sa surface est donnée par
S =
$\int_{\text{une}}^{b}$ 2 π f (x) 1 + f (x) ^{2} dx
Si f (x) = 3x −2, on peut demander à Sage de calculer l'aire de la surface de révolution et

visualiser
à la fois la courbe f (x) et la surface 3D.

```
sage : f(x) = sqrt(3*x-2)
sage : intégrale(2*pi*f(x)*sqrt(1+diff(f,x)**2), x, 0, 5)
1/18*pi*(61*sqrt(61) - 1)
sage: show(revolution_plot3d(f, (x,0,5), parallel_axis='x', \
show_curve=True, opacity=0.7), aspect_ratio=1)
```

Tracés implicites en 3D. Comme en 2D, la façon la plus pratique de décrire certaines surfaces est
plicitement. Cette fois, les équations peuvent impliquer une ou plusieurs des variables x, y et z. Avec
la commande implicite_plot3d() , nous devons spécifier une équation, ainsi que le maximum et valeurs minimales pour les trois variables. Dans cet exemple, la commande implicite_plot3d() est utilisée
pour produire un hyperboloïde d'une feuille :

```
sage : var('xy z')
sage : implicite_plot3d(x^2/2+y^2/2-z^2/3==1, (x,-4,4), (y,-4,4), \
(z,-4,4), opacité=.7, plot_points=200)
```

Après tout cela, n'êtes-vous pas prêt pour une boule de neige aux cerises sauvages ?[1]

1 Cerise sauvage parce que le gâteau de mariage n'apparaîtra pas.

```
sage : var('z theta r phi')
sauge : bh=4 # coordonnée z pour la base de la tasse
sauge : th=8 # coordonnée z pour le haut de la tasse
sauge : br=bh/(th-bh) # rayon de la base de la tasse
sage : tr=th/(th-bh) # rayon du haut de la tasse
sage: a = .1 #rayon de paille
sage : cupside=parametric_plot3d((z*cos(theta)/(th-bh), \
z*sin(thêta)/(th-bh),z), (thêta,0,2*pi), \
(z,bh,th), color='aliceblue')
sage : cupbase=parametric_plot3d((r*cos(thêta), r*sin(thêta), bh), \
(theta,0,2*pi), (r,0,br),color='aliceblue')
sage : snow=parametric_plot3d((tr*sin(theta)*cos(phi), \
tr*cos(thêta)*cos(phi), tr*sin(phi)+th), \
(thêta,0,2*pi), (phi,0,pi/2),couleur='rouge')
sage: paille = parametric_plot3d((a*cos(theta),a*sin(theta),z), \
(thêta,0,2*pi),(z,bh+.01,th+2*tr))
sage: show(cupside+cupbase+neige+paille,aspect_ratio=1)
```

Champs vectoriels. Un champ vectoriel attribue à chaque point du domaine (soit 2 ou 3) un 2- ou 3-
vecteur dimensionnel. Les champs vectoriels bidimensionnels peuvent être représentés graphiquement avec plot_vector_field() .
Les champs vectoriels tridimensionnels peuvent être représentés graphiquement avec la

commande plot_vector_field3d() . Dans
l'exemple ci-dessous, on trace le champ de vecteurs F(x,y, z) = −2xyi−3zj+ zk
sage : plot_vector_field3d((2*x*y,-3*z,z),(x,0,2),(y,0,2),(z,0,2))

Monter une colline

Quel est le chemin le plus rapide pour monter une colline ? Vous devriez commencer par regarder immédiatement autour de vous, trouver
le chemin le plus raide possible, continuez un peu dans cette direction. Ensuite, répétez ce processus, reeval-
uant votre parcours après avoir parcouru une petite distance. Continuez ainsi et éventuellement
vous atteignez le sommet de la colline. Vous n'avez qu'à regarder immédiatement autour de votre position actuelle
à tout moment du parcours. Cette stratégie décrit une technique mathématique importante appelée
« montée en pente. »[2]

Comme vous le savez, le sommet de la colline s'appellerait un maximum local. Les méthodes exactes peuvent
pouvoir trouver les coordonnées du maximum local. Lorsque les méthodes exactes échouent, la méthode de
la montée en pente donnera une excellente approximation. Mieux que cela, il se rapproche également de la
chemin de la montée la plus raide.

Comment savoir dans quelle direction procéder ? Le gradient d'une fonction f (x,y) est le fonction vectorielle ∇f définie par

$$f(x,y) = \frac{\partial f}{\partial x}\vec{i} + \frac{\partial f}{\partial y}\vec{j}.$$

(Ici, $\vec{i}$ et $\vec{j}$ représentent les vecteurs de base canoniques le long des axes x et y.) Lors de l'évaluation en un point (x_0, y_0), le gradient donne un vecteur dans la direction du taux de variation

maximum. Pour
exemple, si f (x,y) = $-x / 1+x^2+y^2$ comme précédemment, nous pouvons tracer son champ vectoriel de gradient 2D. Chaque vecteur
pointe dans la direction de la montée la plus raide depuis son point de queue.

```
sage : var('y')
sage : f(x,y)=-x/(1+x^2+y^2)
sage : cp = contour_plot(f(x,y),(x,-3,3),(y,-3,3), \
cmap='Spectral', fill=False,colorbar=True)
sage : vf = plot_vector_field(f.gradient(),(x,-3,3),(y,-3,3))
sage : afficher(cp+vf)
```

2 Après avoir gravi la colline, vous voudrez peut-être redescendre — ou peut-être n'avez-vous jamais voulu la gravir au début
endroit! Dans ce cas, vous pouvez utiliser la méthode de descente de gradient. Vous voulez deviner comment cela fonctionne ?

Revenons à notre montée en pente : si nous calculons la pente à notre point de départ, prenons un petit
pas dans cette direction, nous pouvons répéter le processus à partir de ce point suivant jusqu'à ce que nous soyons au sommet de
la colline. Les petits pas sont des vecteurs tout gradient réduites en utilisant une faible valeur de ε . De n'importe quel
point (x_n, y_n), on passe au point suivant x_{n+1}, y_{n+1} par la formule

$$X_{n+1}, y_{n+1} = (x_n, y_n) + \varepsilon \nabla f(x_n, y_n).$$

Étant donné que chaque point ne dépend que du point précédent, une implémentation n'a besoin de se souvenir que
la position actuelle afin de calculer la suivante. Plutôt que d'utiliser des indices comme ci-dessus, nous
peut réécrire la formule plus simplement en utilisant q comme point courant et p comme point précédent :

$$q = p + \epsilon F(p)$$

Comment sait-on qu'on a atteint le sommet ? Près du maximum local, l'amplitude de
le vecteur de gradient sera presque nul. (Cela signifie que le plan tangent sera presque horizontal.)
Lorsque ce vecteur court gradient est réduite par ε , en ajoutant le vecteur $\varepsilon \nabla f(p)$ pour p gardera q
très proche de p. On peut donc dire qu'on a atteint le maximum local quand nos pas deviennent

très petite, c'est-à-dire lorsque la distance entre les points p et q est inférieure à une certaine précision souhaitée.
Puisque nous ne pouvons pas prédire le nombre d'étapes avant de commencer, nous utilisons un tout en boucle pour la répétition.
Nous pouvons tirer parti de la nature de la condition dans la boucle while en commençant p et q
de manière appropriée.

L'ASCENSION D'UNE COLLINE
227
algorithme gradient_ascent
contributions
• f (x,y), une fonction dérivable donnée sous forme d'expression ou de fonction
en x et y
• p, le point (x_0, y_0)
• d, le nombre de décimales de précision souhaité
• ϵ , le facteur d'échelle d'un pas
les sorties
• une liste L des points le long du chemin approximatif de la montée la plus raide
fais
convertir p en vecteur
soit L = (p)
soit q = p
ajouter
$\vec{-}$
je à p
répéter tant que p et q sont plus éloignés que la précision souhaitée :
soit p = q
soit q = p+ $\varepsilon \nabla$f (p)
ajouter q à la liste
retourner la liste
Cela se traduit par le code Sage suivant :

```
sage : def gradient_ascent(f,p,eps,d):
p = vecteur(p).n()
L = [p]
q = p
p = p + vecteur([1,0])
grad = f.gradient()
tandis que round(norm(pq), d) > 0 :
p = q
q = p + eps*grad(x=p[0], y=p[1])
L.append(q)
retour L
```

Ajoutons une visualisation de nos pas le long du chemin. Si nous avons un chemin sous forme

de liste de (x,y) points
sortie de l'algorithme de montée de gradient, nous pouvons alors représenter graphiquement ces points sur le tracé 3D avec un
flèche vers le dernier point. Afin de connaître les dimensions du tracé de f (x,y), nous aurons besoin
les valeurs x et y minimales et maximales de tous les points du chemin. Une procédure d'aide à
trouver ces valeurs est donné ici:

```
sage : def xyminmax(L) :
xvals = [pt[0] pour pt dans L]
yvals = [pt[1] pour pt dans L]
xmin = min(xvals)
xmax = max(xvals)
ymin = min(yvals)
ymax = max(yvals)
retourner xmin, xmax, ymin, ymax
```

Dans notre visualisation, nous inclurons le tracé 3D de f (x,y) et les segments de ligne avec un pointe de flèche. Nous étendons le graphique de f (x,y) pour inclure un peu plus au-delà du minimum et
valeurs maximales x et y.

```
sage : def visual_gradient_ascent3d(f,p,eps,d):
L = gradient_ascent(f,p,eps,d)
xmin, xmax, ymin, ymax = xyminmax(L)
xrange = xmax - xmin
yrange = ymax - ymin
g = plot3d(f, (x, xmin - .1*xrange, xmax+.1*xrange), \
(y, ymin - .1*yrange, ymax+.1*yrange), opacité=.5)
g = g + line3d([(pt[0], pt[1], f(x=pt[0], y=pt[1])) \
for pt in L], color='red', arrow=True, size=30)
montrer(g)
q = L[-1]
renvoie q, f(x=q[0], y=q[1])
```

A partir de (1,1) sur g(x,y) = $\frac{1-x}{1+x^2+y^2}$, l'algorithme trouve que 25 étapes sont nécessaires pour 3
décimales de précision avec $\epsilon = .2$. La valeur maximale est d'environ z = 1,2071 à
(–0.4141,0.00069), et le chemin a cette forme :

Des exercices

Vrai faux. Si la déclaration est fausse, remplacez-la par une déclaration vraie.

1. Les courbes paramétriques et les surfaces paramétriques peuvent être représentées graphiquement à l'aide de la même commande Sage.

2. Chaque procédure graphique 3D dans Sage a une procédure 2D correspondante.
3. En 3D, les graphiques sont combinés en les ajoutant.
4. *L'* option detect_poles= *True* de Sage détectera l'asymptote dans f (x,y) = $3/x^2+y^2$.
5. Si vous voulez représenter graphiquement $\rho = \theta / \varphi$ pour $\pi / 12 \leq \varphi \leq \pi$ et $0 \leq \theta \leq 6\pi$, le moyen le plus simple est d' utiliser le
commande plot3d() .
6. La surface r = z sin(2 θ) pour 0 z $\leq$ 1 et $0 \leq \theta \leq 2\pi$ est la même que pour $0 \leq \theta \leq 8\pi$, donc
Le graphique de Sage de plot3d(z*sin(2*theta), (theta,0,2*pi), (z,0,2), transformation=C) est
le même que celui de plot3d(z*sin(2*theta), (theta,0,8*pi), (z,0,2), transformation=C) ,
après avoir laissé C=Cylindrical('rayon', ['azimut', 'hauteur']) ,
7. La meilleure façon de représenter graphiquement le paraboloïde $y = x^2 + z^2$ est d'isoler z en prenant la racine carrée,
tracer les branches positives et négatives séparément en utilisant plot3d() , et les combiner en utilisant
une addition.
8. Si vous voulez voir les courbes de niveau de f (x,y), vous utilisez plot3d() avec l' option mesh=True .
9. Le domaine de ∇f (x,y) est un sous-ensemble de ^{2} .
10. Si la fonction n'a pas de maximum local, l'algorithme de montée de gradient renvoie le point initial
(x_0 ,y_0).
Choix multiple.
1. Supposons que pts soit une liste de points 3D. Si vous souhaitez utiliser ces sommets pour tracer un contour non rempli et délimité
polygone en 3D, que faire ?
A. polygone(pts, remplissage=Faux)
B. polygone(pts, color=None, edgecolor='green')
C. ligne(pts+[pts[0]])
D. demandez à votre instructeur
2. Combien de paramètres sont nécessaires pour définir paramétriquement des courbes tridimensionnelles ?
A. 1
B. 2
C. 3
D.4
3. Combien de paramètres sont nécessaires pour définir paramétriquement des surfaces tridimensionnelles ?
A. 1
B. 2
C. 3
D.4

4. Comparez un vecteur de gradient à la courbe de niveau dont il provient.
A. Le vecteur de gradient est toujours tangent à la courbe de niveau.
B. Le vecteur gradient est toujours perpendiculaire à la courbe de niveau.
C. Le vecteur gradient n'a aucune relation avec la courbe de niveau.
D. Le vecteur de gradient a toujours la même couleur que la courbe de niveau.
5. L'erreur produite par `plot3d(sin(x*y)/(x*y), (x,5,-5), (y,-3,3))` est due à
A. la fonction étant indéfinie à l'origine
B. parenthèses mal placées
C. la région n'étant pas carrée

D. l'inversion de `xmin` et `xmax`
6. Si vous tracez une surface 3D avec `plot3d()` , combien de paires de valeurs min et max font tu dois donner ?
A. 1
B. 2
C. 3
D.4
7. Si vous tracez une surface 3D avec `implicite_plot3d()` , combien de paires de min et max valeurs devez-vous donner?
A. 1
B. 2
C. 3
D.4
8. Pour modifier l'algorithme de montée de gradient pour faire une descente de gradient, nous devrions laisser
A. $q = p + \varepsilon \nabla f(p)$
B. $q = p + \varepsilon \nabla f(-p)$
C. $q = -p- \varepsilon \nabla f(p)$
D. $q = p \varepsilon \nabla f(p)$
9. L'algorithme de montée de gradient pourrait être modifié pour trouver un maximum local de fonctions de
combien de variables ?
A. 1
B. 3
C.4
D. Tous ces
10. Lequel des énoncés suivants décrit le mieux la signification du gradient ?
A. un nombre qui vous indique si vous êtes à un maximum local
B. un nombre qui vous indique s'il faut monter ou descendre

C. une fonction à valeur vectorielle qui peut être évaluée pour vous dire dans quelle direction aller pour rester
le même niveau
D. une fonction à valeur vectorielle qui peut être évaluée pour vous dire dans quelle direction aller pour monter
le plus rapidement
Réponse courte.
1. Différentes couleurs sont-elles nécessaires pour visualiser un champ vectoriel ou pourriez-vous encore comprendre le
champ vectoriel si tous les vecteurs étaient affichés dans la même couleur ?
2. Qu'est-ce qui peut mal tourner avec l'algorithme de descente de gradient ? Explorez avec diverses fonctions et
points initiaux. Comment pourriez-vous modifier le code pour éviter ces problèmes ?
3. Tracez l'ellipsoïde $2x^2 + y^2 + z^2 = 3$ avec chacun des
(a) `plot3d()`
(b) `implicite_plot3d()`
(c) `revolution_plot3d()`
(d) `parametric_plot3d()` .
Lequel produit le meilleur graphique, même avec plus de points tracés ? Quels problèmes voyez-vous avec
les autres?
Programmation.

DES EXERCICES
231
1. Écrivez une procédure Sage qui représenterait graphiquement des segments de ligne avec des pointes de flèche sur tous les segments.
Assurez-vous que cela fonctionne pour les collections de points 2D ou 3D.
2. Réécrivez l'algorithme de descente de gradient (sans tracer) de manière récursive.
3. Réécrivez l'algorithme graphique de descente de gradient pour tracer sur la surface à l'aide de `contour_plot()`
plutôt que `plot3d()` . Incluez également le tracé du champ vectoriel de gradient dans la visualisation.
4. Réécrivez votre code du problème 3 pour afficher une animation sur le tracé de contour, où chaque
l'image de l'animation contient une étape de plus que l'image précédente.
5. Écrivez une procédure de Sage pour visualiser la région dont l'aire est calculée par une droite intégrale avec
par rapport à la longueur de l'arc $\int_C f(x,y)ds$ dans le cas où la courbe C paramétrée par x(t) et y(t)
pour a t b. Le graphique doit être formé en combinant

• la frontière en bleu–y compris les lignes verticales du plan xy à f à x = a et x = b
ainsi que la courbe C, tracée sur le plan xy et projetée sur la surface f (x,y),
• l'intérieur de la région, non ombré, mais affiché en utilisant au moins 50 lignes verticales, montrant
toute zone positive avec des lignes noires et toute zone négative avec des lignes rouges, et
• le graphique de f (x,y) dans une couleur aliceblue semi-transparente .
Par exemple, avec 80 lignes verticales, visualize_line_int(x+y, (t,t^2), -1, 1) devrait pro-
un graphique comme celui ci-dessous.

CHAPITRE 11

Techniques avancées

Ce dernier chapitre présente deux techniques que l'utilisateur sérieux de Sage trouvera utile, et probablement nécessaire, de temps en temps.

Fabriquer ses propres objets

Sage comprend beaucoup de mathématiques. — Non, ce n'est pas vrai. La sauge comprend

BEAUCOUP

des mathématiques. Vous n'aurez probablement jamais, jamais besoin d'utiliser quoi que ce soit qui n'est pas dans Sage.
Pourtant, certains mathématiciens travaillent parfois avec un groupe d'objets que Sage ne propose pas
immédiatement. Si vous étudiez les jeux combinatoires, qui sont tout simplement géniaux[1] alors
Sage n'offre pas immédiatement un moyen d'effectuer ce qu'on appelle l'arithmétique Nimber.[2] Cette rubrique
présente l'arithmétique Nimber et montre comment vous pouvez créer votre propre type pour permettre à Sage de faire
Arithmétique des nombres.
Arrière-plan. L'idée de base de Nimbers est née d'un jeu nommé Nim, décrit pour la première fois dans
une revue mathématique au début du 20e siècle. Les règles sont simples :
• Le jeu se joue avec des cailloux disposés en rangées.
• Chaque joueur peut prendre autant de cailloux qu'il le souhaite sur une rangée.
• Le dernier joueur à enlever un caillou gagne.
Par exemple, supposons que David et Emmy décident de jouer à Nim avec trois rangées de cailloux,
où la première rangée a sept cailloux, la deuxième rangée en a cinq et la dernière rangée en a trois.
David passe le premier ; supposons qu'il prenne quatre cailloux de la première rangée.

1 Hackenbush, Nim, Chomp, Ideal Nim, Dots, Sprouts, ... voir [2] ou [1] pour plus d'informations que vous ne pouvez

secouer un bâton à.
2 ...à partir de la version utilisée par les auteurs au moment de la rédaction de cet article. Il y a eu des discussions sur y compris la suite de jeux combinatoires, ce qui résoudrait probablement cela en un clin d'œil et offrirait encore plus que des nimbers :
même des nombres surréalistes.

Ce n'était pas une décision très intelligente de la part de David, car Emmy peut prendre tous les cailloux de la
deuxième rangée et quittez cette configuration.
David voit maintenant que quel que soit le mouvement qu'il fait dans une rangée, Emmy peut « refléter » ce mouvement dans le
autre rangée. Autrement dit, il a déjà perdu.
Arithmétique des nombres. L'idée de l'arithmétique de Nimber découle de cette observation qu'une fois
le jeu se réduit à deux rangées visuellement symétriques, le jeu est essentiellement terminé.
L'arithmétique fonctionne comme ceci :
• Le plus petit Nimber est 0, suivi de 1, 2, ...
• Étant donné un ensemble S de Nimbers, le plus petit Nimber non présent dans S est appelé le minimum excluant,
ou mex pour faire court.
• Pour additionner deux Nimbers, écrivez chacun comme une somme de puissances de 2, annulez des puissances identiques, puis
simplifier.
• La soustraction équivaut à l'addition.
• Pour multiplier deux nombres m et n, trouver mex{in + mj + ij : i ≺ m, j ≺ n}.
Ainsi, par exemple, l'addition fonctionne comme suit :
0+ x = x
1+1 = 0
1+2 = 3
2+2 = 0
1+3 = 1+(1+2) = 2
2+3 = 2+(1+2) = 1
3+3 = 0

---

tandis que la multiplication fonctionne comme ceci :
2×3 = mex{0×3+2×0+0×0,1×3+2×0+1×0,0×3+2×1+0×1,
1×3+2×1+1×1,0×3+2×2+0×2,1×3+2×2+1×2}

= mex{0,3,2}
= 1.
Remarquez que le deuxième mex ne laisse rien de côté : les sommes et les produits du premier mex sont
nombre de sommes et de produits, de sorte que (par exemple)
1×3+2×1+1×1 = 3+2+1 = 0 .
C'est lourd à faire à la main, c'est donc une bonne idée de l'automatiser dans Sage.
Nous pourrions, bien sûr, écrire une séquence de procédures qui effectuent simplement les opérations di-
correctement, mais il est également possible de donner à Sage un moyen de traiter les nombres automatiquement, et
en les utilisant de manière naturelle. Par exemple, si nous créons les procédures `nim_add()` et `nim_mult()` ,
puis pour additionner et multiplier a × b + c × d, il faudrait taper
`nim_add(nim_mult(a,b),nim_mult(c,d))` .
C'est difficile à comprendre. Il est beaucoup plus naturel de taper
`a*b + c*d`
et puis, tant que Sage reconnaît `a` , `b` , `c` et `d` comme Nimbers, il effectue l'auto-arithmétique matiquement.
Faites-le avec classe ! Le moyen d'amener Sage à le faire est avec une classe. Une classe décrit un objet
type à un ordinateur et indique à l'ordinateur quelles méthodes peuvent être envoyées à un objet de ce type.
Les classes nous permettent d'organiser des programmes autour des données et des choses que nous pouvons faire avec ces données, et
aider à garder les choses en un seul endroit. Dans Sage, les données associées à une classe, aussi appelées ses attributs,
sont accessibles à l'intérieur de la classe en utilisant la constante *self* , que nous devons écrire comme premier argument
à chacune des méthodes de la classe.[3] Vous le verrez dans un instant.
Pour créer une classe, utilisez le mot - clé `class` :

```
sage : classe NomClasse :
#liste des procédures associées à cette classe
```

Comme le suggèrent les deux points, les procédures associées à cette classe doivent être en retrait. Lorsque l'in-
la denture s'arrête, la classe aussi. Chaque classe doit avoir au moins une méthode nommée `.__init__()` ,
qui indique à Sage comment initialiser un élément de la classe. Cette méthode s'appelle un constructeur,
et est appelé silencieusement chaque fois que nous créons un nouvel objet d'une classe.[4] Ce nouvel objet est appelé un
instance de la classe.

3 Nous n'avons pas besoin d'utiliser le nom *self* ; nous pourrions utiliser `a` si nous voulions, ou `ceci` ou `moi` , mais le *soi* est

le
convention Sage hérite de Python.
4 Il est également possible de l'appeler à nouveau après avoir créé un nouvel objet `a` en tapant `a.__init__(...)` , bien que cela devrait
être fait avec parcimonie, voire pas du tout.

Initialisation : Quelles informations sont propres à la classe ? Nous allons définir une classe nommée `Nimber` .
Un nombre est juste un nombre dont l'arithmétique fonctionne différemment de la normale, nous pouvons donc simplement
initialisez-le avec un nombre, qui devrait vraiment être un entier, nous allons donc tester cela et élever un
exception sinon. De plus, l'arithmétique de Nimber est basée sur des puissances de 2, ce serait donc une bonne
idée de garder une trace des représentations du nombre en termes de puissances de 2 ; on pourrait aussi bien mettre
cela dans le code d'initialisation. Notre méthode `.__init__()` devrait donc initialiser deux éléments
de données avec la classe `Nimber` :
• *self* `.value` , l'entier associé à cette valeur ; et
• *self* `.powers` , les puissances de 2 dont la somme est *self* `.value` .
Pour déterminer quelles puissances de 2 additionnent à *self* `.value` , nous allons tester si le nombre est divisible
par 2, puis diviser par 2 pour supprimer les puissances, en ne gardant que le quotient. Par exemple,
• 11 n'est pas divisible par 2, donc 1 est une puissance de 2 qui fait la somme de 11. Si on divise 11 = 1 + 2 + 8
par 2, nous avons maintenant 5 = 0+1+4.
• 5 n'est pas divisible par 2, donc 1 est une puissance de 2 qui fait la somme de 5. On avait divisé par 2, donc 1×2 = 2
est une puissance de 2 dont la somme est 11. Si nous divisons 5 = 0+1+4 par 2, nous avons maintenant 2 = 0+0+2.
• 2 est divisible par 2, donc 1 n'est pas une puissance de 2 qui fait la somme de 2. Nous avions divisé par 2 deux fois,
donc $1\times 2^2 = 4$ n'est pas une puissance de 2 qui s'additionne à 11. Si nous divisons 0 = 0+0+2 par 2, nous
avoir 1 = 0+0+1.
• 1 n'est pas divisible par 2, donc 1 est une puissance de 2 qui somme à 1. Nous avions divisé par 2 trois
fois, donc $1 \times 2^3 = 8$ est une puissance de 2 dont la somme est 11. Si nous divisons 1 = 0 + 0

+ 1 par 2, nous
ont maintenant 0 = 0+0+0.
Une fois que nous avons atteint 0, nous avons terminé, donc les puissances de 2 qui totalisent 11 correspondent aux divisions qui ont donné
us reste 1 : c'est-à-dire 1, 2 et 8.
Cela suggère le code d'initialisation suivant.

```
sage : classe Nimber :
def __init__(self, n):
# vérifier le type valide
if type(n) != Entier et type(n) != int ou n < 0 :
augmenter ValueError( \
'Les nombres doivent être des entiers non négatifs'
)
self.value = n
self.powers = set()
# trouver des puissances de 2 qui s'ajoutent à n
je = 1
tant que n != 0 :
si n % 2 == 1 :
self.powers.add(i)
n = n // 2
je = je * 2
```

Si vous travaillez avec une feuille de calcul, allez-y et saisissez-la dans une cellule. Plus tard, nous ajouterons plus
méthodes, et vous devez les taper dans la même cellule. Si vous travaillez à partir de la commande

---

ligne, il est plus facile de l'écrire sous forme de script et de l'attacher ; au fur et à mesure que vous ajoutez plus de méthodes à la classe et
enregistrer le fichier modifié, Sage le rechargera automatiquement.
Une fois que nous avons tapé ce qui précède et l'avons exécuté ou attaché à Sage, comment créons-nous une instance Nimber ?
Nous utilisons le nom de la classe comme s'il s'agissait d'une procédure, dont les arguments sont appropriés pour passer à son
méthode __init__() .

```
sage : a = Nombre(4)
sage: b = Nimber(1/2)
Erreur de valeur :
Les nombres doivent être des entiers non négatifs
```

Jusqu'ici tout va bien. Look Let à un un peu plus près.

```
sauge : un
<__main__.Nimber instance à 0x161c9e5a8>
```

Ce n'est pas une sortie très utile, n'est-ce pas ?
Représentations d'un objet. Pour donner une sortie plus utile, vous pouvez ajouter une méthode .__repr__() .

La meilleure chose à faire est probablement d' imprimer *self* .value . Sage veut que __repr__() renvoie une valeur de chaîne,
et vous pouvez convertir *self* .value en chaîne à l'aide de la commande str() :

```
sage : classe Nimber :
...
def __repr__(self):
return str(self.value)
```

(N'oubliez pas que vous avez besoin de la définition .__init__() dans l'espace indiqué par les ellipses.)
Maintenant, lors de l'initialisation d'un number, nous pouvons afficher sa valeur :

```
sage : a = Nombre(4)
sauge : un
4
```

D'autre part, il pourrait être utile de voir quelles puissances de 2 la classe a trouvé cette somme à la
numéro. Les méthodes __init__() et __repr__() sont appelées « noms de méthodes spéciales », mais il
n'est pas un nom de méthode spécial qui convient à une représentation alternative plus détaillée de
un objet. Pour cela, nous pouvons définir une méthode avec un nom de notre propre conception ; nous l'appellerons
.power_repr() , et il renverra simplement *self* .powers .[5]

```
sage : classe Nimber :
...
def power_repr(self):
retourner self.powers
```

5 Il n'est pas nécessaire d'entourer power_repr de traits de soulignement, car il ne s'agit pas d'une « méthode spéciale ». Le but de
le soulignement est de s'assurer que les gens ne redéfinissent pas accidentellement une méthode spéciale.

Nous pouvons maintenant voir comment cela fonctionne :

```
sage : a = nombre (28)
sage : a.power_repr()
{4, 8, 16}
```

Jusqu'ici tout va bien.
Implémentation de l'arithmétique. Ce dont nous avons vraiment besoin, cependant, c'est d'un moyen d'automatiser l'ajout
et multiplication de Nimbers - de préférence d'une manière qui nous permette d'utiliser l'arithmétique ordinaire
les opérateurs. Encore une fois, Sage nous propose des méthodes spéciales pour cela ; nous utiliserons .__add__() et .__mul__() .
Mais comment allons-nous les mettre en œuvre ?
Pour l'addition, il devrait être simple de choisir les puissances de 2 qui ne sont pas répliquées dans les deux puissances des nombres . Il y a un moyen simple de le faire : puisque nous avons

conservé l'ensemble des pouvoirs qui
ajouter au nombre, nous pouvons utiliser la méthode .symmetric_difference() pour un ensemble. Par exemple, le
les nombres 14 = 2 + 4 + 8 et 20 = 4 + 16 devraient s'additionner à 2 + 8 + 16 = 26. En appliquant le
différence aux ensembles de puissances de 2 nous donne ce résultat :

```
sage : {2, 4, 8}.symmetric_difference({4,16})
{2, 8, 16}
```

L'addition est donc relativement simple à mettre en œuvre : renvoie le Nimber défini par sum de la différence symétrique.

```
sage : classe Nimber :
...
def __add__(self, b):
return Nimber(somme( \
self.powers.symmetric_difference( \
b.pouvoirs \
)))
```

Testons ceci sur notre exemple :

```
sage : Nimber (14) + Nimber (20)
26
```

Excellent!
La multiplication est un peu plus difficile, pour deux raisons. Tout d'abord, rappelons la définition de nimber mul-
conseil :
$m \times n = \text{mex}\{in + mj + ij : i \prec m, j \prec n\}$.
Cela nous oblige à calculer le mex d'un ensemble avec beaucoup de nombres que nous devons générer.
Construire l'ensemble n'est en fait pas si difficile : nous pouvons utiliser une compréhension de liste pour nous en occuper.
Cependant, Sage n'a pas de procédure mex intégrée, nous devons donc définir une procédure à com-
pute-le. Cela ne devrait pas faire partie de la classe Nimber , car bien que le mex fonctionne sur les nimbers, il
n'est pas la propriété d'un nimber, et si vous voulez vous devriez en principe pouvoir trouver le mex d'un

---

ensemble d'entiers non négatifs sans changer le code. Nous allons donc implémenter mex() comme un
procédure.
Comment doit-on le mettre en œuvre ? Nous devons trouver le plus petit nombre qui n'est pas dans l'ensemble, donc
pourquoi ne pas commencer par i = 0, tester pour voir si i est dans l'ensemble, et augmenter et

répéter si c'est le cas ? Cette
l'approche fonctionnera :

```
sage: def mex(S):
je = 0
tandis que Nimber(i) dans S :
je = je + 1
retour Nombre(i)
```

Essayons un ou deux exemples rapides.

```
sage: mex([Nimbre(0), Nimbre(2), Nimbre(4)])
sauge : 1
sage: mex([Nimbre (0), Nimbre (1), Nimbre (2)])
sauge : 3
```

Les choses vont bien jusqu'à ce que vous y regardiez d'un peu plus près :

```
sage: S = { Nimber(0), Nimber(1), Nimber(2), Nimber(1), Nimber(2)}
sauge : mex(S)
0
```

La réponse devrait être 3.

PANIQUE!

... eh bien, non, non. Réfléchissons à ce qui pourrait causer cela. La procédure `mex` doit décider si `Nimber(0)` est dans `S` . Vérifions S :

```
sage : S
{0, 1, 1, 2, 2}
sage : a, b = Nimber(2), Nimber(2)
sage : a == a
Vrai
sage : a == b
Faux
```

Que se passe t-il ici? Il y a deux problèmes :

• Nous rencontrons une réplication dans l'ensemble.

• Sage semble penser que `Nimber(2)` n'est pas égal à lui-même !

Ces deux problèmes proviennent du fait que nous n'avons pas dit à Sage comment décider si deux
`Les nombres` sont égaux. Vous pourriez penser que c'est évident, mais en fait ce n'est pas le cas ; l'égalité peut signifier deux
différentes choses:

• `a` et `b` sont le même objet (égalité), et

• `a` et `b` sont des objets différents avec la même valeur (équivalence).

Le comportement par défaut de `==` est de vérifier le premier ; donc, `a == a` nous donne *True* , parce que nous sommes comparables
ing un objet à lui-même, tandis que `a == b` nous donne *False* , car nous comparons deux objets distincts.

De notre point de vue, bien sûr, il s'agit d'une distinction sans différence, comme dit le

proverbe. Si
nous voulons que Sage nous dise que deux Nimbers distincts sont en fait égaux - c'est-à-dire, si nous voulons que Sage
dites-nous quand deux Nimbers distincts sont équivalents - alors nous devons dire à Sage comment décider cela.
Comparer les Nimbers. Nous pouvons résoudre ce problème. Nous avons la technologie. Nous pouvons également résoudre le problème
avec mex, en ajoutant des méthodes spéciales qui enseignent à Sage comment comparer Nimber s. Ceux-ci sont:

.__eq__(self , other) teste si self et other sont identiques
.__ne__(self , other) teste si self et other sont différents
.__lt__(self , other) teste si self est inférieur à other
.__le__(self , other) teste si self est inférieur ou égal à other
.__ge__(self , other) teste si self est supérieur ou égal à other
.__gt__(self , other) teste si self est supérieur à other

Nous devons définir les six, mais heureusement, ce n'est pas trop difficile, car nous pouvons simplement comparer les valeurs s.

```
sage : classe Nimber :
...
def __eq__(self, b):
return self.value == b.value
def __ne__(self, b):
return self.value != b.value
def __lt__(self, b):
return self.value < b.value
def __le__(self, b):
return self.value <= b.value
def __ge__(self, b):
return self.value >= b.value
def __gt__(self, b):
renvoyer self.value > b.value
```

Cela nous permet au moins de faire les choses suivantes :

```
sage : a, b, c = Nimber(2), Nimber(2), Nimber(3)
sage : a == b
Vrai
sage : a == c
Faux
```

Malheureusement, les ensembles ne fonctionneront toujours pas :

```
sage : S = {Nimbre (0), Nimbre (1), Nimbre (2), Nimbre (1), Nimbre (2)}
Erreur-type:
instance non illisible
```

Aie. Qu'est-ce qu'une « instance hachable ? »
Un objet est hachable lorsque vous pouvez l'associer à un entier (de type int , pas de type Integer)

que Sage utilise pour créer des ensembles et des dictionnaires. Cet entier s'appelle un hachage et ne devrait jamais changer,
car le changer rendrait les choses étranges dans les ensembles et les dictionnaires. Cela signifie l'objet lui-même
ne devrait jamais changer. Ces deux critères s'appliquent aux Nimber s, car nous n'avons créé aucune méthode
qui changent ses données, et la valeur d'un Nimber est un entier qui ne changera pas. Pour créer ceci comme
son hachage, nous le configurons simplement en tant que méthode .__hash__() :

```
sage : classe Nimber :
...
def __hash__(self):
return int(self.value)
```

. . . et maintenant les ensembles fonctionnent très bien :

```
sage : S = {Nimbre (0), Nimbre (1), Nimbre (2), Nimbre (1), Nimbre (2)}
sage : S
{0, 1, 2}
```

Revenons à la multiplication. Maintenant que nous avons des ensembles de Nimbers fonctionnels, nous pouvons enfin multiplier
Nimber s. La définition de la multiplication Nimber est récursive ; c'est-à-dire que nous devons d'abord calculer
les produits de paires plus petites. Notre récursivité a besoin de cas de base ; nous pourrions aussi bien utiliser 0 et 1, comme
la définition de la multiplication de nombre montre que 0× x = 0 et 1× x = x pour chaque nombre x :

```
sage : classe Nimber :
...
def __mul__(self, b):
si self.value == 0 ou b.value == 0 :
retour Nombre(0)
elif self.value == 1 :
retour b
elif b.value == 1 :
retourner soi-même
autre:
return mex({self*Nimber(j) + Nimber(i)*b \
+ Nombre(i)*Nombre(j) \
pour i dans la plage (self.value)
pour j dans la plage (b.value)})
```

Que diriez-vous d'une table de multiplication Nimber ?

```
sage : pour i dans la plage (10) :
pour j dans la plage (10) :
print(Nombre(i) *Nombre(j))
imprimer()
```

Vous n'aurez peut-être pas beaucoup de chance de le faire imprimer plus de quelques lignes.

Pourquoi? Une fois que nous obtenons
au-delà d'un certain point, la récursivité dans la multiplication la rend très, très lente. La sauge a un
décorateur `@cached_method` qui devrait fonctionner comme le décorateur `@cached_function` , uniquement pour la classe
méthodes. Cependant, cela ne fonctionne pas dans cette situation, car le cache est lié à chaque objet,
et notre boucle crée constamment de nouveaux `Nimber` s. Une meilleure approche serait de créer un
cache; nous laissons cela en exercice au lecteur.

Des trucs que nous avons laissés de côté. Nous avons omis un certain nombre de choses que les classes nous permettent de faire, avec
l'héritage et la surcharge étant deux sujets majeurs.

L'idée de l'héritage est d'éviter la duplication de code en héritant automatiquement des méthodes pour
un type qui s'appliquent également légitimement à un autre type. Par exemple, l'ensemble des nombres forme
un champ; ainsi, si `Nimber` héritait de la classe `Field` de Sage , il acquerrait automatiquement de la méth-
ods qui sont prédéfinis pour `Field` . L'inconvénient est qu'un champ est un objet plutôt abstrait.
ject, donc bien que `Field` fournisse beaucoup de méthodes, il n'implémente pas la plupart d'entre elles, mais augmente
NotImplementedError lorsque vous essayez d'y accéder. Cela signifie que vous devez mettre en œuvre cela
vous-même, ce qui n'est pas toujours facile. (Par exemple : comment trouvez-vous l'indice multiplicatif d'un nombre
verset?)

L'idée de la surcharge est que l'on peut implémenter une procédure avec différents types. Pour en-
position, vous voudrez peut-être multiplier un `Nimber` par un entier régulier, plutôt que d'autoriser la multiplication
par `Nimber` s seul. Dans ce cas, vous devrez modifier la `procédure __mul__()` pour tester ses entrées.
Dans d'autres cas, une méthode peut parfois prendre deux arguments et trois arguments d'autres fois.

Ceci est accompli dans Sage en fournissant des valeurs par défaut pour certains arguments et en vérifiant les arguments.
types de ment.

Cython

Sage est construit sur Python, un langage interprété. Nous avons parlé de la différence entre preted et d'autres types de code à la p. 13; essentiellement, l'interprétation du code entraîne une pénalité pour traduire
lignes encore et encore.

De plus, Python s'appuie sur ce qu'on appelle le « typage dynamique ». Cela signifie que Python ne
connaître le type d'une variable jusqu'à ce qu'il lui attribue une valeur. Par exemple, dans la méthode `.__mul__()` pour
`Nimber`, Sage ne sait pas quel type `b` a jusqu'à ce qu'il exécute réellement `.__mul__()`. Ça signifie que Sage doit fréquemment s'arrêter pour vérifier le type d'une variable avant d'exécuter une opération ; après tout,
il ne peut pas évaluer l'expression `b.value` sur un objet `b` auquel il manque un attribut nommé `value`.
L'autre approche majeure de la dactylographie utilisée dans la programmation est la « dactylographie statique ». Dans cette approche,
le programmeur doit déclarer le type de chaque variable avant l'exécution du programme. Cela peut être un problème
au programmeur, mais cela évite à l'ordinateur de vérifier à chaque fois s'il
doit faire ça. Cela peut conduire à des augmentations massives de la vitesse.
Cas de test : la méthode de bissection. Pour voir quelle différence cela peut faire, nous représentons
notre procédure `method_of_bisection()` de la page 160 et expliquez comment la chronométrer.

Page 242

CYTHON
242

```
sage : def method_of_bisection(a, b, n, f, x=x):
c, d = a, b
f(x) = f
pour i dans la plage (n):
# calcule le point médian, puis compare
e = (c + d)/2
si (f(c) > 0 et f(e) > 0) ou (f(c) < 0 et f(e) < 0) :
c = e
elif f(e) == 0 :
c = d = e
Pause
autre:
d = e
retour (c, d)
```

Rappelons que `a` et `b` sont des extrémités, `n` est le nombre de fois où la boucle doit être exécutée, `f` est une fonction,
et `x` l'indéterminé. Pour avoir une idée de la façon de chronométrer le code, nous allons utiliser une procédure spéciale
appelé `%timeit`. Cette procédure répète une instruction qui la suit plusieurs fois et renvoie le le plus rapide de ces moments.

```
sage : %timeit method_of_bisection(1, 2, 40, x**2 - 2)
100 boucles, le meilleur des 3 :
17 ms par boucle
```

(Vous pouvez obtenir un nombre différent de 17 ms ; ce n'est pas grave, car cela dépend de beaucoup de choses en plus

le programme : vitesse du CPU, vitesse du bus, ...) Qu'est-il arrivé? Sage a exécuté le programme en trois séries de
100 boucles, quelque chose comme ça :

```
sage : pour i dans la plage(3) :
pour j dans la plage (100) :
method_of_bisection(1, 2, 40, x**2 - 2)
```

Pour chacun de ces trois ensembles, il a enregistré combien de temps il a fallu pour effectuer les 100 boucles, obtenant
trois horaires. Enfin, il a rapporté le meilleur de ces trois timings : 17 millisecondes. Bref, un l'exécution de `method_of_bisection() a` pris environ 0,17 milliseconde, ou 170 microsecondes.
Pour voir quel effet la compilation peut avoir, tapez ce qui suit, soit dans une cellule Sage, soit dans un
script à attacher à la ligne de commande. (Si vous utilisez une feuille de calcul, tapez `%cython` sur la première ligne. Si
vous utilisez la ligne de commande, enregistrez le script avec le suffixe `.spyx` au lieu de `.sage` .) Ne vous inquiétez pas
sur les changements pour l'instant ; nous les expliquerons dans un instant.

```
de sage.symbolic.ring importer SR
de sage.symbolic.callable import \
CallableSymbolicExpression Ring
def method_of_bisection_cython(a, b, n, f, x=SR.var('x')):
c, d = a, b
# f(x) = f
f = CallableSymbolicExpressionRing([x])(f)
pour i dans la plage (n):
# calcule le point médian, puis compare
e = (c + d)/2
si (f(c) > 0 et f(e) > 0) ou \
(f(c) < 0 et f(e) < 0):
c = e
elif f(e) == 0 :
c = d = e
Pause
autre:
d = e
retour (c, d)
```

Lorsque vous exécutez cette cellule ou attachez ce script, Sage s'arrêtera un moment pour compiler le code.
Si vous avez mal tapé, vous verrez probablement une ou plusieurs erreurs répertoriées. Dans ce cas, regarde
parcourez attentivement cette liste et essayez de corriger les erreurs. Une fois la compilation réussie (vous saurez
parce que Sage ne signale aucune erreur), vous pouvez le chronométrer comme vous le faisiez auparavant.

```
sage : %timeit method_of_bisection_cython (1, 2, 40, x**2 - 2)
100 boucles, le meilleur des 3 :
16 ms par boucle
```

Le code s'est un peu accéléré : $16/17 \approx 94{,}1\%$ aussi long, soit une accélération d'environ 5%. Nous ne blâmons pas
vous si cela ne vous impressionne pas. Il est possible de faire mieux dans certains cas ; le site Cython donne
un tel exemple.

Avant de décrire comment nous pouvons améliorer ce code, expliquons un peu les différences de
le code Cython du code Sage original.

• Cython a besoin que nous « importions » des objets que Sage fournit habituellement gratuitement.

Les instructions qui font cela utilisent les mots-clés `from` et `import` . Nous avons dû importer le anneau symbolique `SR` et un nouveau type curieux appelé `CallableSymbolicExpressionRing` qui nous discuterons un peu plus bas.

• Cython a besoin que nous déclarions nos indéterminés — même `x` !

...et pour déclarer nos indéterminés, nous avons besoin de la commande `var()` , qui est elle-même
pas disponible immédiatement pour Cython ; nous devons y accéder en tant que méthode de `SR` .

• Cython a besoin de nous pour définir les fonctions mathématiques d'une manière différente.

Lorsque Sage lit tout ce que vous tapez dans une feuille de calcul, sur la ligne de commande ou dans un Sage
script, il en réécrit une partie de la forme pratique que nous utilisons vers une forme plus pratique
à sa fondation Python. Un exemple est la déclaration d'une fonction ; comme tu pourrais devinez à partir du code ci-dessus, la déclaration `f(x) = ...` est un raccourci pratique pour `f`

`= CallableSymbolicExpressionRing([x])(...)` . Un autre exemple est l'opéra « carat »
tor `^` ; cela a un sens complètement différent en Python, qui considère le "double product" opérateur `**` le véritable opérateur d'exponentiation. C'est la raison pour laquelle nous avons toujours
vous a conseillé d'utiliser le double-produit pour l'exponentiation, mais si l'habitude devait nous faire trébucher,
et nous tapons un carat, Sage l'acceptera silencieusement comme une exponentiation - à moins qu'il n'apparaisse dans un
Script Cython.

Ces modifications sont donc nécessaires pour que le code soit compilé en Cython.

Nous pouvons résoudre ce problème. Nous avons la technologie. Rappelez-vous comment

nous avons mentionné que beaucoup de temps
est gaspillé à vérifier le type d'un objet. Et si on attribuait un type aux articles qui sont réutilisés
le plus? Le code passe beaucoup de temps à travailler sur le calcul de f(c) et f(e) , alors peut-être nous ferions bien de les calculer une fois, puis de stocker leurs valeurs. Modifions le programme comme
suit :

```
de sage.symbolic.ring importer SR
de sage.symbolic.callable import \
CallableSymbolicExpression Ring
def method_of_bisection_cython(a, b, n, f, x=SR.var('x')):
c, d = a, b
# f(x) = f
f = CallableSymbolicExpressionRing([x])(f)
pour i dans la plage (n):
# calcule le point médian, puis compare
e = (c + d)/2
fc, fe = f(c), f(e)
si (fc > 0 et fe > 0) ou \
(fc < 0 et fe < 0) :
c = e
elif fe == 0 :
c = d = e
Pause
autre:
d = e
retour (c, d)
```

Cela nous donne une amélioration notable des performances.

```
sage : %timeit method_of_bisection_cython (1, 2, 40, x**2 - 2)
100 boucles, le meilleur des 3 :
11,6 ms par boucle
```

C'est une amélioration d'un peu plus de 25 % par rapport à la version précédente, nous faisons donc mieux.
Mais! — le lecteur avisé se plaindra — nous aurions pu pré-calculer ces valeurs sans Cython aussi. Quelle différence cela fait-il ? Cela fait vraiment une différence saine. Nous ne le ferons pas
montrer le code ici, mais si vous modifiez method_of_bisection() pour pré-calculer ces valeurs, nous
voir un timing le long de ces lignes:

```
sage : %timeit method_of_bisection(1, 2, 40, x**2 - 2)
100 boucles, le meilleur des 3 :
12,7 ms par boucle
```

C'est toujours plus lent que la version pré-calculée de Cython, mais c'est plus rapide que le premier Cython
version! Encore une fois, vous pouvez trouver une amélioration aussi légère moins

qu'impressionnante.
D'accord, mais nous avons encore un tour dans notre manche.
Attribution de types statiques. Comme nous l'avons mentionné précédemment, Python est un langage à typage dynamique.
Cython est plus un hybride. Comme nous l'avons vu ci-dessus, vous n'avez rien à taper, mais vous pouvez taper
certaines choses. Vous êtes sur le point de voir que cela peut avoir un effet significatif.
En plus de ne calculer `f(c)` et `f(e)` qu'une seule fois, essayons de leur attribuer un type. Quoi type devraient-ils avoir? Certaines des options habituelles sont

- `int` et `long` , qui correspondent aux types Python pour les entiers « plus petits » et « plus grands » ;
- `float` et `double` , qui correspondent aux types Python pour "smaller" et "larger" nombres à virgule flottante;
- `Integer` et `Rational` , qui correspondent aux types Sage pour les entiers et les nombres rationnels bres.

Il existe de nombreux autres types, dont certains sont très utiles dans diverses circonstances. Ré-
membre que vous pouvez généralement trouver le type d'un objet en demandant à sage : `type(2/3)` , par exemple,
nous donnera `<type 'sage.rings.rational.Rational'>` . Notez-le, nous y reviendrons
dans un moment.
Si vous regardez où nous avons introduit la méthode de bissection, nous avons mentionné qu'il s'agit d'un
sorte d'« approximation exacte », en ce sens qu'elle spécifie un intervalle (donc, une approximation) sur lequel le
racine ment certainement (donc, exact). Vous vous souviendrez que la méthode de bissection nous donne des fractions,
ce qui exclut plus ou moins `int` , `long` et `Integer` . Devrions-nous utiliser `float` , `double` ou `Rational` , ensuite? N'importe lequel d'entre eux fonctionnera, et les deux premiers sont beaucoup, beaucoup plus rapides que le troisième, mais
les auteurs ont un penchant pour les nombres exacts dans toute leur splendeur longue et compliquée, nous allons donc opter pour
pour ça.
Rappelons que la commande `type()` nous a dit qu'un `Rational` est `sage.rings.rational.Rational` . De ces quatre mots, les trois premiers nous disent d'où nous importons, et le dernier nous dit ce que nous importons.
Nous allons ajouter une ligne à notre liste d'importation qui s'occupe de cela :

```
de sage.rings.rational cimport Rational
```

Si vous regardez attentivement, vous remarquerez que nous avons utilisé le mot-clé `cimport` au lieu du mot-clé `import`
comme nous l'avons fait avec les deux autres. C'est parce que nous importons un type implémenté en tant que classe dans

un autre script Cython. Vous ne pouvez pas importer des types implémentés en tant que classes dans des scripts Sage ou Python.
Une fois que nous avons cela, nous ajoutons une ligne au début du code qui définit fc et fe comme
Rationnel . Nous devons alors contraindre les calculs f(c) et f(e) à Rational , car ils
sont par défaut une simple Expression . Ce sont tous les changements que nous devons apporter afin que nous puissions énumérer les
code modifié ci-dessous :

CYTHON
246

```
de sage.symbolic.ring importer SR
de sage.rings.rational cimport Rational
de sage.symbolic.callable import \
CallableSymbolicExpression Ring
def method_of_bisection_cython(a, b, n, f, x=SR.var('x')):
cdef Rational fc, fe
c, d = a, b
f = CallableSymbolicExpressionRing([x])(f)
pour i dans la plage (n):
# calcule le point médian, puis compare
e = (c + d)/2
fc, fe = Rationnel(f(c)), Rationnel(f(e))
si (fc > 0 et fe > 0) ou (fc < 0 et fe < 0) :
c = e
elif fe == 0 :
c = d = e
Pause
autre:
d = e
retour (c, d)
```

Le timing sur ce code est remarquablement plus rapide :

```
sage : %timeit method_of_bisection_cython (1, 2, 40, x**2 - 2)
100 boucles, le meilleur des 3 :
6,27 ms par boucle
```

C'est presque la moitié du temps qu'il fallait avant, et certainement la moitié aussi vite que le temps Python le plus rapide
nous avons vu. Au total, le code prend maintenant environ un tiers de la longueur de l'original.
Et si nous ajoutions ces informations de type au script Python ? - Oh, attendez. Nous ne pouvons pas.
Une mise en garde. L'ajout d'informations de type pour les variables restantes n'améliore pas les performances.
mance de manière mesurable et, dans certains cas, cela ralentit en fait le programme. (Nous avons vérifié
pour être sûr.) Si vous jugez nécessaire de suivre la voie de l'écriture de scripts Cython, alors opti-

En les mettant en place en ajoutant des informations de type, vous voudrez en savoir plus sur le profilage de votre code pour voir
ce qui prend le plus de temps. Nous ne donnons ici que quelques brèves indications.
Sage propose un outil de profilage appelé prun() . Il compte le nombre de fois que chaque procédure est
appelé, mesure le temps que Sage passe dans chaque procédure et devine à partir de là lequel il serait préférable de calculer les procédures. Cela fonctionne avec les deux procédures Sage/Python ordinaires
ainsi qu'avec les procédures Cython.
Lors de la rédaction de ce chapitre, nous avons essayé prun() sur la méthode originale Cythonized method_of_bisection()
comme suit:

```
sage : prun(method_of_bisection_cython(1, 2, 30, f))
```

Sage a exécuté le programme et a immédiatement fourni une sortie sous la forme d'un tableau, avec chaque colonne
indiquant
• nombre d'appels (ncalls) ;
• temps total passé dans la procédure (tottime) ;
• durée moyenne de chaque appel (premier appel) ;
• temps total cumulé passé dans cette procédure et toutes les procédures qu'elle appelle (cumtime);
• temps cumulé moyen passé dans chaque appel (seconde par appel) ; et
• le lieu de l'appel.
Lorsque nous l'avons lancé pour la première fois, trois des quatre premières lignes indiquaient que Sage avait fait
• 106 appels à la méthode de substitution pour les objets sage.symbolic.expression.Expression ;
• 107 appels à callable.py:343(_element_constructor_) ; et
• 106 appels à callable.py:448(_call_element_) .
Sans avoir vu ces lignes auparavant, il n'était pas difficile de deviner que Sage passait beaucoup de temps
calculer la valeur d'une fonction en un point. Cela nous a motivé à essayer d'éviter le re-calcul de f(c) et f(e) . Lorsque cela a été fait, le nombre d'appels à ces trois procédures a été réduit de plus d'un tiers, respectivement à 60, 61 et 60 à nouveau.
Avec cela à l'écart, l'indice que nous devions attribuer un type à quelque chose serait
utile est qu'il y a eu 209 appels à {isinstance} . Cela, avec des invocations mystérieuses de procédures dans le fichier complex_interval_field.py , indiquaient que de nombreuses vérifications de type étaient en cours
sur, et en fait la version finale du programme ne rapporte qu'une seule innovation de {ininstance} , et

aucun appel de complex_interval_field.py .
Décider quelles variables précises ont besoin d'informations de type reste une sorte d'art avec des compétences acquises par l'expérience et le tâtonnement, c'est pourquoi nous le classons, avec le
création de cours, rubrique « Techniques avancées ». Si tu vas assez loin dans le monde merveilleux
mathématiques, vous pourriez en avoir besoin un jour.
Des exercices
Vrai faux. Si la déclaration est fausse, remplacez-la par une déclaration vraie.
1. La plupart des personnes qui travaillent avec Sage devront, à un moment donné, faire leur propre calcul mathématique
objets.
2. Les jeux combinatoires sont tout simplement géniaux.
3. L'ajout de Nimber s'annule automatiquement ; c'est-à-dire x + x = 0 pour tout nombre x.
4. La multiplication Nimber utilise l'élément maximum d'un ensemble de nombres.
5. L'une des raisons d'encapsuler Nimbers dans une classe est d'écrire les opérations arithmétiques d'une manière plus
format pratique et naturel.
6. Les attributs d'une classe sont des variables qui incluent des données propres à la classe.
7. La modification des comportements de base d'une classe est soumise à des méthodes spéciales : __init__() , __repr__() ,
__eq__() , et ainsi de suite.
8. Dans Sage, un ensemble détermine automatiquement quand deux objets d'une classe sont équivalents.
9. Un hachage est un terme technique désignant un objet dont la classe est trop compliquée à ajouter à un ensemble .
10. Le dragon en colère de la récursivité vole dans ce chapitre et enflamme au moins une méthode de classe.
11. Cython nous permet d'accélérer les programmes Sage en compilant et en attribuant des types statiques.
12. Les langages dynamiques connaissent le type de chaque objet avant l'exécution du programme ; langages statiques
restent agnostiques jusqu'à ce que la valeur soit attribuée.

---

13. Nous ne pouvons utiliser Cython de la feuille, en plaçant le % cython déclaration
avant un bloc de code.
14. Nous pouvons « Cythoniser » le code Sage sans avoir à nous soucier de le modifier.
15. Nous ne pouvons pas importer de types implémentés en tant que classes dans des scripts Cython, uniquement des types natifs

à Sage comme `int` et `Rational` .

Choix multiple.

1. Les méthodes utilisées pour initialiser les données propres à une classe sont appelées :

A. les attributs

B. constructeurs

C. initialiseurs

D. fabricants

2. Le terme correct pour désigner une variable dont le type est de classe `C` est :

A. un attribut de `C` .

B. un constructeur pour `C` .

C. un descendant de `C` .

D. une instance de `C` .

3. La valeur de mex{1,2,4,5,7,8,9} est :

A. 0

B. 3

C. 6

D. 9

4. En arithmétique Nimber, la valeur de 3+10 est :

A. 0

B. 2

C. 9

D. 13

5. Pourquoi la multiplication Nimber est-elle difficile à faire à la main ?

A. cela nécessite une récursivité

B. c'est nouveau et différent

C. auto-annulation de l'ajout

D. ce n'est pas difficile à faire à la main

6. La bonne façon de créer une nouvelle instance d'une classe nommée `C` , dont le constructeur prend un entier
argument que nous voulons être 2, est en tapant:

A. `a = C(2)`

B. `a = nouveau C(2)`

C. `a = nouveau(C, 2)`

D. `a = C.__init__(2)`

7. Le terme désignant les méthodes spécialement nommées que Sage attend pour certains comportements « naturels » d'un
classe sont :

A. méthodes d'attribut

B. méthodes magiques

C. méthodes spéciales

D. méthodes par points

8. Un objet est hashable quand on peut l'associer à une valeur de quel type ?

A. tout type bien ordonné

B. haschich

C. Entier

D. int

9. Quel mot clé permet d'attribuer un type à une variable dans Cython ?

A. cdef

B. déf

C. type

D. aucun mot clé ; il suffit de placer le type avant le nom de la variable

10. Quel(s) mot(s) clé(s) utilisons-nous pour accéder aux objets et types Sage définis dans d'autres fichiers Cython ?

A. accès ... à partir de ...

B. de ... importer ...

C. #inclure

D. PANIQUE !

Réponse courte.

1. Résumez pourquoi il peut être utile de créer une classe pour un nouvel objet mathématique, plutôt que
implémenter des opérations sous forme de procédures avec les noms `my_object_add()` , `my_object_mul()` ,
et ainsi de suite.

2. Résumez les similitudes et les différences entre les attributs et les variables d'instance.

3. Alors que Cython compile les instructions Python en vrai code machine, le résultat ne peut pas être exécuté
indépendamment d'un interpréteur Python (comme Sage). Pensez-vous que cela fait de Cython un véritable
langage compilé ? Pourquoi ou pourquoi pas?

4. La méthode de bissection peut être exécutée avec des types `flottants` au lieu de types `rationnels` . Penses-tu
cela le rendra plus rapide ou plus lent? Essayez-le et discutez de la façon dont le résultat se compare à votre estimation.

Programmation.

1. Une autre méthode spéciale pour une classe est `__nonzero__()` , qui permet à Sage de décider si un
instance d'une classe est, eh bien, "différent de zéro". Implémentez cette méthode pour `Nimber` , renvoyant *True* si
et seulement si `self.value==0` .

2. Il est un peu plus correct en Python que `__repr__()` renvoie une chaîne avec laquelle vous

pourrait recréer l'objet ; c'est-à-dire que la séquence de commandes suivante devrait en fait avoir
la sortie donnée, pas seulement 2 .

```
sage : a = Nimber(2)
sauge : un
Nombre(2)
```

(a) Modifier Nimber de __repr __ () méthode pour renvoyer la chaîne « propre ». Astuce : vous pouvez rejoindre
chaînes en « ajoutant ».
(b) Python recommande que la méthode __str__() soit utilisée pour renvoyer un « informel », « plus
pratique » ou une représentation « concise » de l'objet. Cette valeur est renvoyée chaque fois nous utilisons la procédure str() . Ajoutez une méthode __str__() à la classe Nimber qui renvoie
ce que nous avons initialement écrit pour __repr__() . Une fois que vous avez terminé avec ce problème, la classe
devrait fonctionner comme suit :

```
sage : a = Nimber(2)
sauge : un
Nombre(2)
sage : str(a)
2
```

3. Nous avons dit que la multiplication Nimber pouvait être améliorée en utilisant une variable de cache. Modifiez votre
Classe Nimber pour introduire une variable globale nommée cached_multiplications , qui est initialisée comme un dictionnaire vide. Modifiez ensuite Nimber « s .__ mul __ () méthode pour effectuer les opérations suivantes:

- Soit c=self.value et d=b.value .
- Si c < d , intervertissez les deux.
- Essayez d'affecter au résultat la valeur associée à la clé (c,d) dans cached_multiplications .
- Si cela génère une erreur, détectez-la, effectuez la multiplication de la manière habituelle, attribuez sa valeur
au résultat , et l'enregistrer dans le dictionnaire.
- Enfin, retournez result .

Essayez maintenant de générer la table de multiplication qui a pris trop de temps dans le texte.
4. Il n'est pas déraisonnable d'imaginer que quelqu'un puisse vouloir instancier un Nimber en utilisant un ensemble de
puissances de 2, ce qui nous évite d'avoir à le faire comprendre par l'ordinateur. Modifier le __iter__()
procédure pour qu'elle teste le type de n . Si n n'est pas un tuple, une liste ou un ensemble, il procède comme avant ; autrement,
il convertit n en un ensemble, affecte cette valeur à self.powers , puis calcule self.value . La bonne

l'implémentation devrait fournir les résultats suivants (en utilisant notre implémentation originale de
__repr__() , pas celui assigné dans un exercice précédent) :

```
sage: a = Nimber({1, 4})
sauge : un
5
sage : a.power_repr()
{1, 4}
```

Bonus : Ajoutez une vérification que la collection ne contient que des puissances de 2, de sorte que l'entrée {1, 3}
soulèverait une *ValueError* .

CHAPITRE 12

L A TEX utile

De nombreux endroits de ce texte utilisent L A TEX, et pas mal d'exercices et de travaux pratiques vous orientent vers son utilisation.
Ce chapitre résume toutes les commandes dont vous avez besoin pour ce que nous faisons dans ce texte, plus
un peu plus si vous voulez expérimenter. Il ne servira pas d'introduction générale à L A TEX, car
qu'il y a beaucoup de textes beaucoup plus appropriés dont tout le but est d'exposer sur ce merveilleux outil.

Commandes de base

La figure 1 donne une liste pratique de commandes L A TEX. Pour les utiliser dans une chaîne de texte, vous devez
entourez de signes dollar, par exemple $x \in \mathbb{R}$. Vous pouvez également utiliser \(et \) comme délimiteurs ; pour
exemple, \(x \in \mathbb{R}\) .

Délimiteurs

Si vous avez une expression complexe, vous voudrez peut-être que les délimiteurs (parenthèses, crochets, etc.)
de grandir avec. Vous pouvez le voir dans la différence entre
(x x xxx
...
)
et
⎛
x
x xxx
...
⎞
.
Pour ce faire, placez la commande \left ou la commande \right juste avant le délimiteur.
Chaque \left doit correspondre à un \right , mais si vous n'en voulez qu'un, vous pouvez placer

un point après le
autre pour indiquer que vous ne voulez rien.
La deuxième expression ci-dessus provient de la saisie :
$\left(x^{x^{x^{x^{x^{\iddots}}}}} \right)$.
Vous pouvez obtenir l'intervalle $2^{\ln 5}, \infty$ en tapant ce qui suit :
$\left[2^{\ln5},\infty\right)$.
Cela semblera mieux que $[2^{\ln5},\infty)$ (ce qui donne $[2^{\ln 5}, \infty)$), car dans le premier exemple, les délimiteurs s'étirent pour correspondre à la hauteur des objets concernés.
Matrices
Il est préférable de définir des expressions matricielles sur des lignes distinctes ; nous pouvons le faire en utilisant les délimiteurs \[
(« commencer l'affichage mathématique ») et \] (« arrêter l'affichage mathématique »).

Notation LaTeX

concept représenté	exemple dans LaTeX	résultat
{ ... }		
regroupement	voir ci-dessous	voir ci-dessous
^		
exposant	x^2	x^2
\sqrt		
racine carrée	sqrt{x^2+1}	$\sqrt{x^2+1}$
_		
indice	x_{\mathrm{suivant}}	x_{suivant}
\dans		
élément de	x\en S	$x \in S$
\{ ... \}		
un ensemble contenant . . .	\{1,5,7\}	$\{1,5,7\}$
\frac{ a }{ b }		
fraction de a sur b		

\frac{2}{5}

$\frac{2}{5}$

\alpha , \beta , etc.

lettres grecques

2\pi

2π

\infty

infini

(-\infty,\infty)

$(-\infty,\infty)$

\sin , \cos , etc.

correctement formaté

\sin(\frac{pi}{6})

péché(

$\frac{\pi}{6}$)

les fonctions

\rightarrow ,

flèches

\lim_{x\rightarrow2}

limite

$\lim_{x\rightarrow 2}$

\leftarrow , etc.

\somme , \int , \prod

somme,

\int_a^bf(x)dx

$\int_{a}^{b} f(x)\,dx$

intégral,

\lim_

limite

$\lim_{n\rightarrow\infty}$

$\sum_{i=1}^{m} f(x_i)\,\Delta x_i$

produit

\somme_{i=1}^nf(x_i)

\Delta x

\leq , \geq

,

a\leq b

un b

\notin , \neq

$\notin$, $\neq$

a\pas\dans S

a / S

\subset , \not\subset

,

S\pas\sous-ensemble T

S T

\ldots , \cdots

..., ···

\mathbb{N}

= {1,2,...}

=\{1,2,\ldots\}

\cap , \cup

carrefour, union

S\cap(T\tasse U)

S (T U)

\mathrm{ ... }

ne pas mettre en italique. . .

\mathrm{suivant}

suivant

\mathbb{ ... }

inscrire

\mathbb{R}

"tableau gras"

\mathbf{ ... }

écrivez . . .

a\mathbf{F}

un F

en caractères gras

\mathcal{ ... }

inscrire

\mathcal{S}

police calligraphique

FIGURE 1. Balisage LATEX utile pour Sage

Nous commençons une matrice à l'aide de la commande \begin{pmatrix} , et la terminons à l'aide de la commande \end{pmatrix} .

(Si nous ne voulons pas de parenthèses autour de la matrice, nous utilisons la paire de commandes \begin{matrix}

et \end{matrice} .)

Nous spécifions les entrées de la matrice ligne par ligne. Les colonnes sont séparées par l'esperluette (&), tandis que

les lignes sont séparées par une double barre oblique inverse (\\ ).

Par exemple, on peut obtenir la matrice
⎛
⎜
⎜
⎜
⎜
⎝
cos x 2 -1
X
et 2x
x +1
péché x 2 -1
1− x
e -2x
X
⎞
⎟
⎟
⎟
⎟
⎠
en tapant ce qui suit :

```
\[ \begin{pmatrix}
\cos x^{2}-1 & & & & x\\
& e^{2x}\\
x+1 & & & \sin x^{2}-1\\
& 1-x & & & e^{-2x}\\
& & & X
\end{pmatrice} \]
```

Partie 2
Laboratoires

Prérequis pour chaque laboratoire

Les laboratoires de cette partie sont organisés selon le domaine général des mathématiques auquel
ils s'appliquent le plus. Nous indiquons ici quels chapitres du texte sont nécessaires pour qu'un étudiant puisse
pour terminer ce laboratoire selon l'approche que nous avions en tête. Si l'instructeur a un autre
approche à l'esprit, les conditions préalables pourraient ne pas être si fermes.

Différents types de tracés : ce laboratoire nécessite le chapitre 2 sur les « calculs de base » et

le chapitre 3
sur « Jolies (et pas si jolies) images ».
Cauchy, le cercle et la parabole de croisement : ce laboratoire nécessite le chapitre 2 sur
tations » et le chapitre 3 sur « De jolies (et pas si jolies) images ». Cela peut être un peu plus facile une fois que le
l'élève a terminé le chapitre 6 sur « Résoudre des équations ».
Un quotient de différence important : ce laboratoire ne nécessite que quelques calculs de base à partir de
Chapitre 2.
Illustrer le calcul : ce laboratoire nécessite le calcul, qui apparaît dans le chapitre 2 sur
putations », et le traçage, qui apparaît au chapitre 3 sur « De jolies (et pas si jolies) images ».
Règle de Simpson : ce laboratoire nécessite des calculs, qui apparaissent au chapitre 2 sur « Calcul de base »
tations », des feuilles de travail interactives, qui apparaissent à la fin du chapitre 4 sur « Écrire votre propre
procédures » et les boucles for, qui apparaissent au chapitre 5 sur « Se répéter avec des collections ».
La méthode de Runge-Kutta : ce laboratoire nécessite le calcul, qui apparaît dans le chapitre 2 sur « Basic
calculs » et les boucles les plus élémentaires, qui apparaissent au chapitre 5 sur « Se répéter définitivement avec des collections.
Coefficients de Maclaurin : ce laboratoire nécessite des calculs, qui apparaissent dans le chapitre 2 sur
putations » et les boucles for, qui apparaissent au chapitre 5 sur « Se répéter avec les collections ».
p-series : ce laboratoire nécessite un tracé, qui apparaît au chapitre 3 sur « Jolie (et pas si jolie)
images » et des boucles définies, qui apparaissent au chapitre 5 sur « Se répéter avec les collections ».
Sur une tangente : cet atelier nécessite le chapitre 2 sur « Calculs de base », le chapitre 3 sur « Pretty
images (et pas si jolies) » et le chapitre 6 sur « Résoudre des équations ».
Maxima et minima en 3D : ce laboratoire nécessite la résolution d'équations, qui apparaît dans le chapitre
6, création de tracés 3D à partir du chapitre 10, et les dictionnaires, qui apparaissent au chapitre 5 sur « La répétition
vous-même avec des collections.
Propriétés algébriques et géométriques des systèmes linéaires : ce laboratoire nécessite l'algèbre linéaire,
qui apparaît au chapitre 6 sur « Résoudre des équations ».
Matrices de transformation : ce laboratoire nécessite l'algèbre linéaire, qui apparaît dans le chapitre 6 sur

« Résoudre des équations » et le tracé des vecteurs, qui apparaît au chapitre 3 sur « Jolie (et pas-
si jolies) images. Il serait utile de connaître les boucles for, qui apparaissent au chapitre 5 sur
vous-même avec des collections.
Visualiser les vecteurs propres et les valeurs propres : ce laboratoire nécessite l'algèbre
linéaire, qui apparaît dans
Chapitre 6 sur « Résoudre des équations », le tracé des vecteurs, qui apparaît au chapitre 3 sur
« Pretty
(et pas si jolies) images », et pour les boucles, qui apparaissent au chapitre 5 sur « Se répéter
avec des collections.

Moyenne des moindres carrés : ce laboratoire nécessite l'algèbre linéaire, qui apparaît dans le
chapitre 6 sur
"Résoudre des équations" et le chapitre 3 sur "Jolies (et pas si jolies) images".
Méthode de Bareiss : ce laboratoire nécessite l'algèbre linéaire, qui apparaît dans le chapitre 6
sur « Résoudre
équations » et les boucles for, qui apparaissent au chapitre 5 sur « Se répéter avec des
collections ».
Une façon de répondre plus efficacement à la dernière partie impliquerait des expressions
try/except, qui
apparaissent au chapitre 7 sur la « Prise de décision ».
Méthode de Dodgson : ce laboratoire nécessite l'algèbre linéaire, qui apparaît dans le chapitre
6 sur « Résoudre
équations » et les boucles for, qui apparaissent au chapitre 5 sur « Se répéter avec des
collections ».
Fonctions un-à-un : ce laboratoire nécessite des dictionnaires, qui apparaissent au chapitre 5
sur « Re-
vous tourmenter définitivement avec des collections.
Le jeu de décor : ce laboratoire nécessite des collections, qui apparaissent au chapitre 5 sur
« Répéter votre-
auto définitivement avec des collections.
Le nombre de façons de sélectionner m éléments à partir d'un ensemble de n : Ce projet
nécessite le chapitre
9 sur « Se répéter de manière inductive ».
Propriétés des anneaux finis : ce laboratoire ne nécessite que quelques calculs de base du
chapitre 2.
Ce serait plus simple avec les boucles for, qui apparaissent au chapitre 5 sur « Se répéter
définitivement
avec des collections », mais les nombres sont suffisamment petits pour que ce ne soit pas

strictement nécessaire (être
averti, cependant, que vos élèves peuvent vous détester pour l'avoir attribué avant de couvrir les boucles).
Le Restes Chinois Clock: Ce laboratoire nécessite l' arithmétique modulaire, du chapitre 2 sur « Calculs de base » et divers objets de tracé, du chapitre 3 sur « Joli (et pas si joli)
des photos."
La géométrie des racines radicales : ce laboratoire ne nécessite que quelques calculs de base du chapitre
2. Ce serait plus simple avec les boucles for, qui apparaissent au chapitre 5 sur « Se répéter définitivement
avec des collections », mais les nombres sont suffisamment petits pour que ce ne soit pas strictement nécessaire. (ONU-
comme un autre laboratoire, ils ne devraient pas vous détester si vous attribuez ce laboratoire avant de couvrir les boucles.)
Séquences de Lucas : ce laboratoire nécessite la section « Les valeurs propres et les vecteurs propres résolvent un lapin
dilemme » du chapitre 9 sur « Se répéter inductivement ».
Introduction à la théorie des groupes : ce laboratoire nécessite le chapitre 5 sur « Se répéter avec les collections » et le chapitre 7 sur « La prise de décision ».
Théorie du codage et cryptographie : la majeure partie de ce laboratoire peut être effectuée après le chapitre 9 sur « Re-
vous tourmenter par induction », mais le bonus nécessite la section sur Cython du chapitre 11 sur
"Techniques avancées."
Fractions continues : ce laboratoire nécessite le chapitre 5 sur « Se répéter indéfiniment » et Chapitre 7 sur « Prise de décision ».

Page 257

Mathématiques générales

257

Page 258

Polynômes irréductibles

Supposons qu'un polynôme a des coefficients entiers. Parfois, il est possible de le diviser en deux
ou plusieurs polynômes qui ont également des coefficients entiers :
$x^2 - x - 6 = (x - 3)(x + 2)$.
Parfois, cependant, il est possible de ne prendre en compte que lorsque nous autorisons des coefficients irrationnels ou même complexes.
ficients (vérifiez cela en appliquant la commande `expand()` de Sage au côté droit de l'équation):

$x^2 - x + 1 = x -$
$1 + i\sqrt{3}$
2
$x -$
$1 - i\sqrt{3}$
2
.
Nous appelons irréductible ce deuxième type de polynôme.
Ce devoir étudie l'irréductibilité d'une famille simple mais importante de polynomiaux. Laisser
$p_n = x^n - 1$,
où n est un entier positif. C'est-à-dire,
$p_1 = x - 1$, $p_2 = x^2 - 1$, $p_3 = x^3 - 1$, ...
Ces polynômes sont-ils irréductibles ? Non, pour $p_n(1) = 0$, donc par le théorème du facteur, $x-1$ est un facteur
de p_n .
Comment prend-il en compte ? Rappelons la formule d'une série géométrique :
(1)
$a + ar + ar^2 + \cdots + ar^{n-1} = a \times$
$r^n - 1$
$r - 1$
.
(Une série est « géométrique » si le rapport entre n'importe quel terme et celui qui le suit reste le même
quel que soit le terme que vous choisissez. Dans la série ci-dessus, ce rapport est r.)
Remarquez la main droite
côté $a(r^n - 1)$; c'est la même chose que $p_n(r)$. Soit a = 1 et r = x, de sorte que l'équation 1 devient
$1 + x + x^2 + \cdots + x^{n-1} =$
$x^n - 1$
$x - 1$
.
Multipliez les deux côtés par x -1, puis réarrangez un peu, et nous trouvons que
$x^n - 1 = (x - 1)(x^{n-1} + \cdots + x^2 + x + 1)$.
Nous avons factorisé p_n !
Donc p_n n'est pas irréductible lorsque $n > 1$. Nous nous intéressons à ses facteurs : $x - 1$ est irréduible, mais qu'en est-il
$q_n = x^{n-1} + \cdots + x^2 + x + 1$?
Les trois premiers exemples sont
$q_2 = x + 1$
(irréductible)

$q_3 = x^2 + x + 1$
(irréductible : essayez le
formule quadratique pour voir pourquoi)
$q_4 = x^3 + x^2 + x + 1 = (x + 1)\, x^2 + 1$
(les facteurs!)
Une fois que nous arrivons à $q_5 = x^4 + x^3 + x^2 + x + 1$, nous commençons à penser qu'il pourrait être bien que quelqu'un d'autre
faire le travail. C'est là qu'intervient Sage.
1. Créez une nouvelle feuille de calcul. Intitulez-le « Polynômes irréductibles ». Assurez-vous que votre nom apparaît dans
une cellule de texte en haut de la feuille de calcul.

---

2. Passez en revue les commandes `factor()` et `expand()` du chapitre 2 (pages 31– 32). Utilisez-les pour vérifier
les factorisations que nous avons trouvées de q_2, q_3 et q_4.
3. Utilisez la commande `factor()` de Sage pour déterminer quels q_n sont irréductibles.
Astuce : pour des valeurs plus grandes de n, taper `x^20 + x^19 +` ... et ainsi de suite peut demander beaucoup de travail .
Il est plus facile de simplement factoriser `x^21 - 1` .
4. Examinez vos découvertes et formulez une conjecture sur les valeurs de n qui rendent q_n irréductible.
5. Testez votre estimation :
• Choisissez quelques $n > 100$ pour lesquels vous pensez que q_n est irréductible, et utilisez Sage pour vérifier si
c'est.
• Choisissez quelques $n > 100$ pour lesquels vous pensez q_n facteurs, et utilisez Sage pour vérifier si c'est le cas.
6. Écrivez un bref résumé de ce que vous avez fait dans une cellule de texte à la fin de la feuille de calcul. Utiliser L A TEX
lorsque vous tapez des polynômes, afin qu'il soit facile à lire pour les yeux.

Divers types de parcelles
1. Créez une nouvelle feuille de calcul. Définissez le titre sur « Laboratoire : divers types de tracés ». Ajouter d'autres informations
pour vous identifier, si nécessaire.
Partie 1 : intrigues implicites.
2. Sélectionnez un problème selon le schéma suivant.

Si votre ID se termine par. . . . . utiliser cette équation.

0,1,2

$x^3 + x = y^2$

3,4,5

$x^2 + y^2 2 = 25$

4

xy^2

6,7,8,9

$x^4 - x^2 y + y^4 = 1$

3. Tout d'abord, utilisez au moins 300 points pour créer et afficher un tracé implicite de l'équation sur la région
[-2,25,2,25]×[-2,25,2,25]. La courbe peut être de n'importe quelle couleur, tant qu'elle est noire.[2]

4. Choisissez une valeur x dans l'intervalle [-2,2]. En utilisant vos compétences en mathématiques (ni celles de Sage, ni celles de quelqu'un d'autre)
- obtenir de l'aide de moi est d'accord) trouvez l'équation d'une ligne tangente à la courbe à ce point.
Écrivez l'équation de cette ligne dans une cellule HTML. Utilisez les commandes de base de L A TEX pour vous assurer qu'il
à l'air cool. Remarque : Il peut y avoir plus d'une valeur y pour cette valeur x, il peut donc y avoir plus
qu'une ligne tangente à la courbe pour cette valeur de x, mais vous n'avez besoin d'en tracer qu'une seule. Ne choisissez pas
un point où la ligne tangente est horizontale ou verticale ; ce serait de la triche.

5. Vérifiez votre affirmation en combinant le tracé de la courbe avec un tracé de la ligne tangente. La ligne
devrait être rouge. Mettez un point rouge au point d'intersection de la ligne et de la courbe. Faire le point grand
assez pour se démarquer; la taille de point par défaut n'est pas suffisante.

Partie 2 : Tracés paramétriques.

6. Une courbe de Bézier est une courbe paramétrique qui fait intervenir quatre points de contrôle (x_0, y_0), (x_1, y_1), (x_2, y_2),
(x_3, y_3) et les équations

$$x(t) = x_0(1-t)^3 + 3x_1 t(1-t)^2 + 3x_2 t^2(1-t) + x_3 t^3$$

$$y(t) = y_0(1-t)^3 + 3y_1 t(1-t)^2 + 3y_2 t^2(1-t) + y_3 t^3$$

où t [0,1]. Tracez une courbe de Bézier noire. Vous pouvez choisir les quatre points de contrôle au hasard,
mais n'en utilisez pas que vous ayez vu auparavant. N'hésitez pas à faire une boucle, si vous le pouvez. Ajouter des lignes pointillées rouges
qui relient les 1er et 2ème points de contrôle (x_0, y_0) et (x_1, y_1), et les 3ème et 4ème contrôles
points (x_2, y_2) et (x_3, y_3), permettant au spectateur de voir comment les points de contrôle

se rapportent à la tangente
lignes.
7. Utilisez votre graphique pour écrire une brève description de la relation entre les points de contrôle et
les lignes.
Partie 3 : Animez !
8. Animez la courbe de Bézier :
• Modifiez les valeurs des 2e et 3e points 12 fois. (Ne les changez pas beaucoup, seulement un peu, pour que la courbe ne saute pas trop brusquement.)
• Pour chaque changement, créez un nouveau graphique. Chaque graphique doit être identique à celui que vous avez créé
à l'étape 6, sauf que les 2e et 3e points de contrôle sont différents.
• Animez tous les graphiques à l'aide de la commande animate() et show() .
2 Aucune excuse à Henry Ford.

Partie 4 : Faites-le bien paraître !
9. Ajoutez des cellules HTML dans tout le document qui délimitent clairement les parties. ça doit être clair
où chaque partie commence et se termine. Dans la partie 1, une cellule HTML doit indiquer précisément quelle valeur
de x vous avez choisi ; dans la partie 2, une cellule HTML doit indiquer précisément les points de contrôle que vous avez utilisés ;
dans la partie 3, une cellule HTML doit indiquer clairement la séquence d'ajustements apportés à la 2e et
3ème points de contrôle. Vous devez également ajouter un texte explicatif avant chaque tracé : « Ceci est le
premier tracé, avec des points...", "C'est le deuxième tracé, avec des points...", ... et enfin "C'est
l'animation de tous les cadres. Utilisez L A TEX pour les expressions et les déclarations mathématiques ;
n'oubliez pas la référence des commandes utiles de L A TEX p. 251.

Cauchy, le cercle et la parabole de croisement

Laisser

$$f(x) = \frac{1}{1+x^2}$$

.
1. Créez un tracé de f (x) sur le domaine [−1,1] et appelez-le p_1 .
2. Créez une image du demi-cercle dans les quadrants I et II qui est centré à l'origine, a un rayon
1, et est de couleur verte. Appelez-le p_2 .
3. Trouvez les coordonnées exactes des points d'intersection entre le demi-cercle et le graphique
de f (x). Ces points sont au nombre de trois :
• un sur l'axe des y, que nous appelons P_0 ;
• un dans le premier quadrant, que nous appelons P_+
, avec les coordonnées x_+
, y_+
; et
• un dans le deuxième quadrant, que nous appelons P_-
, avec les coordonnées x_-
, y_-
.
Astuce : Utilisez le fait que le demi-cercle est décrit par l'équation $x^2 + y^2 = 1$ pour y 0. Il est OK pour trouver cela en utilisant un crayon et du papier. Il est également possible de le trouver à l'aide de Sage, mais vous devez
afficher les commandes que vous avez utilisées. Gardez à l'esprit qu'il peut être plus facile de résoudre ce problème avec un crayon et
papier.
4. Créez deux points rouges avec une taille de point 30, un à P_+
et l'autre à P_-
. Combinez-les en un seul
image appelée p_3 .
5. Tracez la fonction $y = x^2$ sur le domaine [−1,1] et appelez-la p_4 .
6. Combinez p_1 , p_2 , p_3 et p_4 en un seul graphique. Assurez-vous que les points rouges se superposent
graphiques, et non en dessous.
7. Expliquez pourquoi les trois graphiques se rencontrent en P_-
et P_+
.
Indice : carré x_+

et rationaliser toutes les fractions.
8. Considérons le rectangle dont le coin inférieur gauche se trouve sur l'origine et dont le coin supérieur droit
se trouve à P
+
. Trouvez le rapport entre sa hauteur et sa largeur.
9. Le nombre d'or est le nombre
$\varphi =$
$\frac{1+\sqrt{5}}{2}$
.
Exprimez votre réponse à #8 en termes de φ .

Calcul et équations différentielles

Un quotient différentiel important

Dans cet atelier, nous montrons comment vous pouvez utiliser Sage pour prouver que
$\frac{d}{dx}(x^n) = nx^{n-1}$.
1. Créez une nouvelle feuille de calcul. Définissez le titre sur « Laboratoire : un quotient de différence important ».
2. Créez une cellule HTML. Écrivez votre nom, et ce semestre. Changez-le pour une couleur. Vous pouvez
choisissez la couleur que vous aimez, tant qu'elle n'est pas noire. — Ou blanc. Le blanc serait mauvais aussi.
3. Dans la première cellule de calcul, utilisez la commande `var()` pour définir les indéterminés x et h.
4. Dans les quelques cellules de calcul suivantes, définissez $f(x) = x^n$ et demandez à Sage de développer le produit
$(x + h)^n$
pour plusieurs valeurs concrètes de n. La commande `expand()` a été montrée p. 32, et
vous devez choisir plusieurs valeurs séquentielles de n : par exemple, n = 1, n = 2, et ainsi de suite.
5. Dans une cellule HTML qui suit ces cellules de calcul, faites une conjecture quant à ce que le dernier
deux termes de $(x + h)^m$

sera toujours, et quel facteur commun les termes restants toujours
avoir. C'est-à-dire que nous avons
tg (x, h)+ u + v
où u et v sont les deux derniers termes que Sage rapporte, et t est le facteur commun pour le termes restants.
6. Dans Calculus, on vous dit que
• la définition de
ré
dx f (x) est
limite
h→0
f (x + h)− f (x)
h
,
• et
ré
dx
x n = nx n−1 .
Dans une dernière cellule HTML, expliquez pourquoi vos réponses aux étapes 4 et 5 démontrent ce fait. Indice:
Utilisez les informations pour le calculer algébriquement. Vous n'avez pas besoin de connaître les détails de g (x, h);
sur papier, vous pouvez simplement substituer tg (x, h) + u + v pour f (x + h), puis effectuer la soustraction,
puis la division, et voyez ce qui ressort.

Illustrer le calcul

1. Créez une nouvelle feuille de calcul. Définissez le titre sur "Lab: Illustrating Calculus". Ajouter d'autres informations
pour vous identifier, si nécessaire.
2. Sélectionnez un problème selon le schéma suivant.

Si votre ID se termine par. . . . . utiliser cette fonction. . . . . sur cet intervalle.

0,1,2

f (x) = sin x

−
π
3
,
2 π
3

3,4,5

f (x) = cos x

−

π
3
,
2 π
3
6,7,8,9
f (x) = tan x
−
π
6
,
π
3
Partie 1 : Dérivés.
3. Trouvez l'équation de la droite tangente à f à x = / 4. Tout calcul qui peut être fait avec Sage devrait être évident dans votre feuille de travail !
4. Combinez les tracés de f et de la ligne tangente à celui-ci sur l'intervalle donné. La courbe pour
f doit être noir et avoir une largeur de 2. La ligne doit être bleue et avoir une largeur de 2.
5. Créez une animation avec au moins 8 images qui montre l'approche de la ligne sécante à la ligne tangente comme x → π/ 4 à partir de la gauche. Réutilisez les tracés de f et la ligne tangente d'en haut.
Les lignes sécantes doivent être rouges et avoir une largeur de 1. Vous êtes libre de choisir les points que vous
comme pour la sécante, tant que x → π/ 4 à partir de la gauche. Lorsque vous avez terminé, votre animation
doit ressembler à celui du programme du cours : par exemple, la ligne sécante doit continuer dans les deux sens, pas seulement dans une direction.
Partie 2 : Intégrales exactes.
6. Calculez l'aire nette entre f et g (x) = 1− x 2 sur l'intervalle donné.
7. Combinez les tracés de f et de g sur l'intervalle donné. Remplissez la zone entre f et g.
Les courbes pour f et g doivent être noires, avec une largeur de 2. Le remplissage peut être de n'importe quelle couleur
vous aimez, mais rendez-le à moitié transparent. Ajoutez une étiquette de texte à l'intérieur du remplissage qui contient le
surface. Utilisez L A TEX pour que l'étiquette de texte soit agréable.
Partie 3 : Intégrales approximatives.
8. Revenez à votre texte Calcul et révisez le calcul de la longueur de l'arc avec les intégrales. Écrivez
la formule, et dans une cellule HTML expliquez brièvement quel outil de la géométrie du lycée est utilisé
pour dériver la formule.
9. Utilisez Sage pour approximer la longueur de l'arc de l'ellipse x 2 / 4+y 2 / 9 = 1. Limitez l'approximation
à 5 points d'échantillonnage et arrondissez votre réponse à 5 décimales.

10. Répétez le problème 9, en limitant cette fois l'approximation à 10 points d'échantillonnage. Quelle partie de
la réponse indique que vous avez une réponse plus précise ? Comparez-la à la valeur lorsque vous utilisez digital_integral() avec son paramètre par défaut (actuellement 87 points d'échantillonnage).
Partie 4 : BONUS ! (pour ceux qui ont un temps et/ou une motivation exceptionnels). Animer env-
images de la longueur de l'arc, où chaque image montre
• pas d'axes ;
• l'ellipse, en noir, de largeur 2 de la courbe ;
• six cadres de 5 segments de ligne pointillée, puis 6 segments de ligne pointillée, ..., et enfin 10
segments de ligne pointillée, en rouge, de largeur 1 ;

• une étiquette de texte dans chaque cadre avec l'approximation correspondante à la longueur de l'arc, au
centre de l'ellipse, en noir.
Ce bonus vaut autant que la mission entière. Si vous le souhaitez, vous pouvez faire la partie 4 au lieu de
Parties 1-3. Assurez-vous de savoir ce que vous faites ; cela peut prendre un certain temps.

La règle de Simpson
Dans Calculus II, vous devriez avoir appris la règle de Simpson pour approximer la valeur d'une intégrale
sur un intervalle.
1. Dans une cellule HTML,
(a) énoncer la formule de la règle de Simpson;
(b) résumer brièvement l'idée qui donne lieu à la formule.
2. Ecrire une procédure Sage qui accepte en entrée une fonction intégrable f , les extrémités a et b
d'un intervalle, et un entier positif n, utilise alors la règle de Simpson pour estimer $\int_{une}^{b} f(x)\,dx$.
3. Ecrire une procédure Sage qui accepte en entrée une fonction intégrable f , les extrémités a et b
d'un intervalle, et un entier positif n, illustre alors la règle de Simpson avec un tracé des deux f et les chiffres dont les aires sont utilisées pour estimer la valeur de l'intégrale. Avoir la

procédure
invoquer la solution à la question précédente et imprimer la zone comme une étiquette L A TEX quelque part
sur la courbe.
4. Faites une procédure interactive pour la solution à la question précédente. En plus d'accepter-
ing f , a, b et n comme entrées, la procédure interactive doit permettre à l'utilisateur de sélectionner la couleur
pour les chiffres utilisés pour estimer la valeur de l'intégrale.

LA MÉTHODE RUNGE-KUTTA
268

La méthode Runge-Kutta

La méthode de Runge-Kutta d'approximation de la solution d'une équation différentielle est similaire à
Méthode d'Euler. Le pseudocode suivant décrit Runge-Kutta :

algorithme Runge_Kutta
contributions
• df , la dérivée d'une fonction
• (x_0 ,y_0), valeurs initiales de x et y
• Δ x, taille de pas
• n, nombre de pas à faire
fais
soit a = x_0 , b = y_0
répéter n fois
soit k_1 = df (a, b)
soit k_2 = df ($a + \Delta_{x/2}$, $b + \Delta_{x/2} \cdot k_1$)
soit k_3 = df ($a + \Delta_{x/2}$, $b + \Delta_{x/2} \cdot k_2$)
soit k_4 = df ($a + \Delta x$, $b + \Delta x \cdot k_3$)
ajouter $\Delta_{x/6}$ ($k_1 + 2k_2 + 2k_3 + k_4$) à b
ajouter Δ x à un
retour (a, b)

Implémentez Runge-Kutta dans le code Sage. Essayez-le sur la même équation différentielle et initiale
condition que nous avons essayée dans la méthode d'Euler et comparer le résultat.

COEFFICIENTS DE MACLAURINE
269

Coefficients de Maclaurin

Dans Calculus III, vous devriez avoir appris les extensions de Maclaurin. Les coefficients de Maclaurin sont

les coefficients numériques des termes dans le développement de Maclaurin.
1. Ecrire une procédure maclaurin_coeffs() qui accepte en entrée une fonction f et un entier positif d . Il renvoie une liste des coefficients de Maclaurin de f (x) jusqu'au degré d inclus. Indice: Vous devriez probablement utiliser les procédures Sage taylor() et coefficient() .
2. Testez votre procédure avec des fonctions élémentaires connues telles que e^x , sin x et cos x. Vérifier que
vos réponses correspondent à celles qui apparaissent dans un manuel de calcul.
3. Utilisez votre procédure pour trouver les coefficients de Maclaurin de f (x) = $\arcsin^2(x)$ jusqu'au degré 20.
4. Regardez les coefficients de Maclaurin non nuls de f (x) pour le degré 6 et plus. Quel commun
facteur voyez-vous dans les numérateurs?
5. En utilisant votre réponse au problème précédent, modifiez votre invocation en
maclaurin_coeffs(arcsin(???)**2, 20)
de sorte que tous les numérateurs soient 1. Enregistrez cette liste sous L1 . Pourquoi est-il logique que vous puissiez faire
cette?
6. A partir de L1 , utilisez une compréhension de liste pour faire L2 , une liste des réciproques du coefficient non nul
cients.
7. Quel est le plus petit facteur commun non trivial des éléments de L2 ? Écrivez une autre liste com-
préhension de créer L3 , qui contient les éléments de L2 , mais divisé par ce facteur commun.
8. Recherchez cette nouvelle liste sur l' Encyclopédie en ligne des séquences entières. Quel nombre se-
séquence trouvez-vous?
9. Revenez en arrière sur vos dernières étapes pour répondre au problème initial :
$\arcsin^2 x = \sum_{k=1}^{\infty} ???$.
(Nous n'avons pas prouvé qu'il s'agit de la série correcte ; nous l'avons seulement conjecturé sur la base de quelques
coefficients.)

---

série p

Dans ce qui suit, ⌈xs désigne le plafond de x, et ⌊xs désigne le plancher de x.
1. Écrivez le code pour calculer la somme
$\sum_{}^{m}$

$\sum_{k=1}$
$\frac{1}{k^p}$
,
et utilisez des boucles pour générer des listes des 50 premières sommes, pour p = 1,2,3.
2. Générez des tracés de ces sommes pour p = 1, 2, 3 en traçant les fonctions échelonnées
$s_p(x) = \sum_{k=1}^{\lceil x \rceil} \frac{1}{k^p}$
sur l'intervalle [1,50]. À l'aide de vos graphiques, devinez si oui ou non la série $\sum_{k=1}^{\infty} \frac{1}{k^p}$ converge, pour p = 1,2,3.
3. Générer des tracés des séquences pour p = 1, 2, 3 en traçant les fonctions
$f_p(x) = \frac{1}{\lfloor x \rfloor_p}$, $g_p(x) = \frac{1}{x^p}$, $h_p(x) = \frac{1}{\lceil x \rceil_p}$
ensemble (en utilisant des couleurs différentes) sur l'intervalle [1,10].
4. Que suggèrent ces graphiques sur $\sum_{k=2}^{m} \frac{1}{k^p}$, $\int_1^n \frac{1}{x^p} dx$, et $\sum_{k=1}^{n-1} 1$

k^p
,
et que suggèrent-ils à propos de
$\sum_{k=1}^{\infty} \frac{1}{k^p}$
et
$\int_1^{\infty} \frac{1}{x^p}\,dx$?
5. À l'aide de vos réponses au problème 4, déterminez si oui ou non
$\sum_{k=1}^{\infty} \frac{1}{k^p}$
converge, pour p = 1,2,3.

OFF SUR UNE TANGENTE
271

Départ sur une tangente

Laisser
$f(x) = \frac{1}{1+x^2}$
.

1. Tracez la fonction f (x) sur [−3,3].
2. Définissez dans Sage la fonction mathématique L(a, x) comme étant la droite tangente à f en x = a.
Remarque : Il doit s'agir d'une fonction à deux variables.
3. Tracer f (x) et L(a, x) ensemble pour chaque $a \in \{1,1.5,2\}$.
4. Notez que chaque droite coupe f deux fois. Notons $Q_a(x)$ la deuxième intersection. Utiliser la sauge
pour trouver les coordonnées de Q_a en fonction de a.
Astuce : Résoudre l'équation L(a, x) = f (x) pour x. Une solution doit être x = a. le
La coordonnée x de Q_a sera l'autre solution de cette équation. Appelez-le h (a). Alors Q_a = (h (a), f (h (a))).
5. Soit F (a) l'aire entre le graphique de f (x) et L(a, x).
6. Utilisez Sage pour trouver une formule explicite pour F (a) en fonction de a. Simplifiez-le

au maximum.
7. Maintenant, soit $a_0 = 3$ et trace un graphique qui inclut les points $P = (a_0, f(a_0))$ et Q_{a_0}, aussi bien que
les fonctions f (x), et $L(a_0, x)$ sur le domaine [−3,3].
8. Calculez la valeur exacte de $F(a_0)$ à l'aide de Sage. Simplifiez le résultat au maximum.
Remarque : Trouvez la valeur exacte, pas une approximation décimale.

TRAVERSER EN ANGLE
272

Traversée en biais

Dans « Écrire vos propres procédures », nous avons appris à définir une procédure Sage qui calcule le
tangente à f (x) en x = a. Puis à la page 104, on vous a demandé d'écrire le code d'une procédure
qui renvoie la ligne normale à f (x) à x = a.
Nous pouvons considérer la ligne tangente comme la ligne qui fait un angle $\theta = 0$ avec le graphique de f (x)
en x = a, et la droite normale comme droite qui fait un angle $\theta = \pi/2$. Dans ce laboratoire, nous généralisons
la construction et définir la ligne qui fait un angle arbitraire θ avec le graphique de f (x).
Supposons que f (x) soit dérivable en x = a. Soit $L_\theta(a, x)$ la droite passant par (a, f (a)) qui fait
un angle θ avec la ligne tangente à f (x) pour x = a. Donc $L_0(a, x)$ est la tangente, et $L_{/2}(a, x)$ est
la ligne normale.
1. Trouvez une formule explicite pour la pente de $L_\theta(a, x)$.
Astuce : faites un dessin. Interprétez la dérivée f (a) comme la valeur d'une fonction trigonométrique, puis utilisez un
identité de déclenchement.
2. Écrivez une procédure Sage qui renvoie $L_\theta(a, x)$.
Remarque : ce problème n'est pas aussi simple que de remplacer m = f (a) dans le code de la ligne tangente
à la page 91 avec la réponse que vous avez trouvée à l'item précédent, car votre procédure doit également
travailler quand $\theta = \pi/2$ ou quand $\tan(\theta)f(a) = 1$
3. Soit maintenant $f(x) = x^2$. Trouver toutes les valeurs de a pour lesquelles $L_\theta(a, x)$ coupera f (x) en un point autre
que (a, f (a)).
Astuce : laissez Sage faire le travail. Résoudre l'équation L

$_\theta$(a, x) = f (x) et voir quelles valeurs d'une marque
sens.
4. Pour chacun un que vous avez trouvé dans l'élément précédent, soit F (a) soit la zone située entre f (x) et L $_\theta$ (a, x). Utiliser
Sage pour trouver une expression explicite pour F(a). Votre expression doit être une expression dans θ et
a, et aussi simplifié que possible.
Astuce : utilisez `factor(expand(...))` .
5. Trouvez la valeur exacte de la valeur minimale possible de F(a) (pas d'approximation décimale).
6. Trouvez maintenant la valeur minimale de F(a) lorsque $\theta = 0$, $\theta = \pi/6$, $\theta = 4$ et $\theta = \pi/2$.

MAXIMA ET MINIMA EN 3D
273

Maxima et Minima en 3D

1. Ecrivez une procédure nommée `critic_points()` qui accepte en entrée un doublement différentiable
fonction f en deux variables x et y, et renvoie une liste de points (x_i ,y_i) qui sont des points critiques de
F . Assurez-vous de ne trouver que des points critiques à valeur réelle ; vous pouvez utiliser la procédure `imag_part()`
pour tester chaque point.
2. Ecrivez une seconde procédure, `second_derivs_test()` , qui accepte en entrée un doublement différentiable
fonction f en deux variables x et y, ainsi qu'un point critique (x,y), et renvoie l'un des
Suivant:
• `'max'` si le point est un maximum ;
• `'min'` si le point est un minimum ;
• `'selle'` si le point est un point selle ; et
• `'non concluant'` si le test de la dérivée seconde n'est pas concluant.
3. Écrivez une troisième procédure, `plot_critical_points()` , qui accepte en entrée un doublement différentiable
fonction f dans deux variables x et y, et renvoie un graphique en 3 dimensions des éléments suivants :
• f (x,y) dans la couleur par défaut ;
• les maxima locaux en rouge ;
• minima locaux en jaune ;
• points-selles en vert ;
• autres points critiques en noir.
Utilisez un dictionnaire pour rechercher la couleur appropriée pour les points. Utilisez `max()` et `min()` de Sage

procédures pour déterminer les valeurs x et y minimales et maximales du tracé afin de s'assurer que
tous les points critiques apparaissent dans le tracé.

Algèbre linéaire

Propriétés algébriques et géométriques des systèmes linéaires

Un système d'équations.

1. Sage a une procédure `randint()` qui choisit des nombres apparemment au hasard. Cela fonctionne dans le
façon suivante : `randint(m,n)` donne un entier de m à n . Utilisez cette commande pour générer 6 entiers aléatoires distincts a, b, c, d, e, f compris entre -20 et 20, puis les substituer dans le système d'équations suivant. (Rappelez-vous que "distinct" signifie qu'il n'y a pas deux nombres entiers
même.)

hache + par = e

cx + dy = f

2. Tracez chaque équation sur le plan xy. Faites une ligne bleue, l'autre rouge. (Ce n'est pas grave
qui est qui.) Selon toute vraisemblance, les deux lignes ne seront pas parallèles, mais se couperont exactement
un point. Ajustez les axes x et y pour que cette intersection soit visible. Si les droites sont parallèles ou
coïncident, modifiez l'un des a, b, ..., f de sorte que les lignes se coupent exactement en un point.

3. Utilisez Sage pour trouver la solution exacte du système d'équations. Cela devrait être un point (x_0, y_0).

Créez un nouveau tracé avec les deux lignes (toujours de couleurs différentes) et un gros point jaune gras au-dessus de
leur carrefour. (Pas trop gros, mais assez gros pour voir.) Assurez-vous que la pointe se trouve au-dessus du
lignes.

Un invariant.

4. Définir les indéterminés a, b, ..., f . Utilisez-les pour définir le système d'équations suivant sans aucun chiffre :

hache + par = e

cx + dy = f

5. Résolvez le système ci-dessus pour x et y.
6. Vous avez peut-être remarqué que la solution a un dénominateur commun. (Si vous ne l'avez pas remarqué,
ce serait le bon moment pour le remarquer.) Qu'est-ce qui est assez étonnant, mais pas-vraiment-que-étonnant
à propos de ce dénominateur ?
Astuce : Pensez à quelques opérations matricielles de base sur la matrice qui correspond à la gauche
côtés des équations originales. C'est quelque chose que vous auriez dû calculer au lycée soutien-gorge, et aurait certainement calculé en algèbre linéaire.
7. Supposons que a et b aient des valeurs concrètes connues. Utilisez votre réponse à #6 pour expliquer pourquoi
l'existence d'une solution dépend entièrement de c et d — et n'a rien à voir avec e et F !!!
Indice : je pose des questions sur l'existence d'une solution, pas sur la valeur spécifique de la solution une fois
ça existe. La valeur spécifique dépend certainement des valeurs e et f , mais son existence dépend
uniquement sur les valeurs de c et d. La question vous demande donc d'utiliser la réponse précédente pour expliquer
cette question d'existence, non de valeur.
8. Soit g le dénominateur de la solution trouvée dans #6. Substituer les valeurs de a et b de #1 dans l'équation g = 0. Vous avez maintenant une équation linéaire à deux variables, c et d. Changer
c à x et d à y.
9. Tracez la ligne déterminée (sans jeu de mots) par cette équation, qui a l'ordonnée à l'origine 0. Aussi
tracer l'équation originale ax + by = c. Comment ces deux lignes sont-elles liées ? (Ce n'est peut-être pas clair
à moins que vous n'utilisiez l'option de tracé aspect_ratio=1 pour que ce soit clair.)

---

10. Cette relation entre les deux lignes serait-elle valable quelles que soient les valeurs de a et b ? Cette
est, si la seule chose que vous avez changé était a et b à la fois dans la ligne et la fonction g (x), serait
la ligne ax + by = c a toujours la même relation avec la valeur correspondante de g (x) ?
Un point à l'infini.
11. Nous revenons à votre système d'équations d'origine. Définir les listes D et E pour que les valeurs de D

se déplacent en 10 étapes de d presque à a et les valeurs de E se déplacent en 10 étapes de e presque à
b. (La différence entre les valeurs finales et a ou b doit être minuscule.) Par exemple, si votre système d'origine est
1x +2y = 3
4x +5y = 6
alors vous pourriez avoir quelque chose comme D =
(4,3,2,1.5,1.1,1.01,1.001,1.0001,1.00001,1.000001)
et E = (5,4,3,2.5,2.1,2.01,2.001,2.0001,2.00001,2.000001).
12. Parcourez les listes D et E pour créer un tracé pour chaque système
hache + par = c
$d_i x + e_i y = f$
.
(Ici, d_i et $e_{i\,se}$ réfèrent respectivement aux ième éléments de D et E.) Combinez les graphiques en
une animation séquentielle. L'animer.
13. Décrivez la relation éventuelle entre les lignes, surtout si vous laissez $d_i \to a$ et $e_i \to b$.
14. Le domaine de la géométrie projective introduit un nouveau point de sorte que toutes les lignes, même parallèles,
se croisent au moins une fois. Utilisez l'animation pour expliquer pourquoi il est approprié d'appeler ce point un
"point à l'infini."
Astuce : vous souhaiterez peut-être ajuster les valeurs x et y minimales et maximales de votre animation
pour y voir plus clair.

Matrices de transformation

Définissez deux variables symboliques a et b . Laisser
A=
cosa sina
−sina cosa
et
B =
cos b péché b
−sin b cos b
.
1. Calculez AB dans Sage. Utilisez vos connaissances en trigonométrie pour spécifier une forme plus simple que
ce que donne Sage. Utilisez L A TEX pour écrire cette forme plus simple dans une zone de

texte sous le calcul de
UN B.
2. Extrayez et nommez l'entrée dans la première ligne et colonne de AB.
celui-ci
pas celui-ci
ni celui-ci certainement pas celui-ci
Essayez de lui donner un nom significatif, pas seulement x, ce qui est de toute façon une mauvaise idée.
3. Dans une nouvelle cellule de calcul, saisissez le nom que vous venez de créer pour cette entrée. Ensuite, tapez un point
(période). Appuyez ensuite sur la tabulation. Recherchez parmi les noms qui s'affichent une commande qui pourrait
réduire l'expression trigonométrique à quelque chose de plus simple. Utilisez cette commande pour confirmer
votre résultat dans la partie (b).
4. Définir une nouvelle matrice, C, obtenue en substituant la valeur $a = \pi / 3$ dans A.
5. Soit v le vecteur défini par (5,2).
6. Calculez les vecteurs Cv, C^2 v, C^3 v, C^4 v, C^5 v.
7. Tracez les vecteurs v, Cv, C^2 v, C^3 v, C^4 v, C^5 v dans différentes couleurs (votre choix). Tracez-les comme
des flèches ou des points, mais pas en tant que fonctions d'étape. Veuillez les combiner en une seule parcelle, plutôt
que de faire six parcelles différentes.
8. Quel est l'effet géométrique de la multiplication répétée de v par la matrice C ? Que prédisez-vous
C^{60} v ressemblerait? Qu'en est-il du C^{1042} v ?

Visualisation des valeurs propres et des vecteurs propres

Sage a une procédure `randint()` qui choisit des nombres apparemment au hasard. Cela fonctionne dans
de la façon suivante : `randint(m,n)` donne un entier de m à n . Utilisez cette commande pour générer 4
entiers aléatoires distincts compris entre 0 et 10, puis utilisez-les pour définir une matrice 2×2 M. (Rappelez-vous
que "distinct" signifie qu'il n'y a pas deux nombres entiers identiques.)
1. Utilisez Sage pour trouver les valeurs propres et les vecteurs propres de M. Il devrait y avoir deux
vecteurs, mais il est possible que vous n'en obteniez qu'un, avec une multiplicité de 2. Dans ce cas, modifiez le

entrées de votre matrice très légèrement pour qu'elle produise deux vecteurs propres distincts.
2. Extrayez les vecteurs propres et nommez-les v1 et v2 .
Astuce : pour un crédit complet, extrayez-les à l'aide de l'opérateur bracket ; ne pas définir de nouveaux vecteurs.
3. Soit K la liste des vecteurs obtenus à partir des 4 produits $M^i v_1$ pour i = 1, 2, . . . , 4. Utilisez une boucle for
définir cette liste ; vous perdrez la grande majorité des points sur cette partie si vous en faites un à
un temps.
Astuce : soyez prudent. L'expression range(10) commence à 0 ; vous voulez commencer à 1.
4. Soit L la liste des vecteurs obtenus à partir des 4 produits $M^i v_2$ pour i = 1, 2, . . . , 4. Utilisez une boucle for
définir cette liste ; vous perdrez la grande majorité des points sur cette partie si vous en faites un à
un temps.
5. Définissez p=Graphics() et q=Graphics() . (Cela définit une parcelle « vide ».)
6. Utilisez une boucle for pour ajouter un tracé de chaque vecteur de K à p . Le premier vecteur doit être rouge ; successif
les vecteurs devraient se déplacer de plus en plus vers le bleu, jusqu'à ce que le dernier soit complètement bleu. Voici
comment je suggérerais de passer du rouge au vert :

```
pour i dans la plage (len(L)):
p += tracé(L[i],couleur=((10-i)/10,i/10,0),zorder=-i)
```

7. Utilisez une boucle for pour ajouter un tracé de chaque vecteur de L à q . Les vecteurs devraient passer du violet (rouge et
bleu) au jaune (rouge et jaune).
8. Afficher p dans une cellule ; montrer q dans un autre.
9. Décrivez comment le résultat est cohérent avec la discussion des valeurs propres et des vecteurs propres sur
p. 204. Si ce que nous voulons dire n'est pas clair, essayez d'ajuster les valeurs xmin , xmax , ymin , ymax pour voir un
petite partie du graphique.

Moyenne des moindres carrés

Un système de deux équations linéaires à deux variables trouvera une solution exacte pour des données exactes.
Malheureusement, le monde réel manque souvent de données exactes, de sorte que les scientifiques et les ingénieurs compensent en
collecter des données supplémentaires et « faire la moyenne » des résultats. Par exemple, dans le cas de deux inconnues x et

y, nous pourrions prendre m mesures qui correspondent au système

$x + a_1 y = b_1$

$x + a_2 y = b_2$

...

$x + a_m y = b_m$.

La meilleure solution (x,y) minimise l'erreur ; c'est-à-dire qu'il minimise

$$\begin{pmatrix} b_1 \\ \dots \\ b_m \end{pmatrix} - \begin{pmatrix} x + a_1 y \\ \dots \\ x + a_{mon}\, y \end{pmatrix}$$

$$= [b_1 - (x + a_1 y)]^2 + \cdots + [b_m - (x + a_m y)]^2 .$$

Nous pouvons minimiser cela de la manière suivante :

$$A = \begin{pmatrix} 1 & un_1 \\ \dots & \dots \\ 1 & heure \end{pmatrix},$$

$X = X$ oui ,

B =
⎛
⎜
⎝
b 1
...
b m
⎞
⎟
⎠
.
Le système d'équations ci-dessus se traduit par
AX = B.
Si nous multiplions les deux côtés à gauche par A T , nous avons
A T AX = A T B.
Le produit de A T et A est une matrice carrée. Si au moins deux points (a i , b i) ont des abscisses distinctes,
la matrice est inversible, et on peut la résoudre en multipliant des deux côtés :
X = A T A
-1
A T B.
On peut montrer que ce X minimise l'erreur (bien que nous ne le montrons pas).
1. Écrivez un pseudo-code pour une procédure qui accepte deux listes de mesures correspondantesa 1 ,...a m
et b 1 ,..., b m et résout pour x 1 ,..., x m .
2. Implémentez le pseudocode en tant que procédure Sage nommée least_squares_2d() .
3. Rédigez une procédure Sage qui accepte deux listes de mesures correspondantes a 1 ,...a m et b 1 ,..., b m , trace chaque point (a i , b i) et trace la ligne qui se trouve le long du vecteur de solution (x 0 ,y 0).
Facultativement, la procédure doit accepter deux arguments supplémentaires color_points et color_line ,
qui indiquent respectivement les couleurs des points et la couleur de la ligne. Assurez-vous d'utiliser
Les commandes min() et max() de Sage pour définir ymin , ymax , xmin et xmax de manière appropriée.

MÉTHODE DE BAREISS
280

Méthode de Bareiss

La méthode de Bareiss pour calculer un déterminant peut être décrite en pseudocode comme suit :
algorithme Bareiss_determinant
contributions
• une matrice n × n M

les sorties
• detM
fais
soit D une copie de M
soit a = 1
pour k {1,...,n −1}
pour i {k +1,...,n}
pour j k +1,...,n
soit d $_{i,j}$ = d $_{i,j}$ d $_{k,k}$ −d $_{i,k}$ d $_{k,j}$
une
soit a = d $_{k,k}$
renvoie D $_{n,n}$

1. Implémentez ce pseudocode en tant que programme Sage, et testez-le sur les matrices suivantes. Lorsque
il produit un résultat, le résultat est-il réellement correct ? Si cela ne donne pas de résultat, qu'est-ce exactement
fait-il? Dans ce dernier cas, étudiez les entrées de la matrice pour voir ce qui ne va pas.

(a) M =

⎛
⎜
3 2 1
5 4 3
6 3 4
⎞
⎟

(b) M =

⎛
⎜
⎜
⎜
1 −4 1
2
-1
4 4
1
3
3 3
4
2
5 2 -1
⎞
⎟
⎟
⎟

(c) M =
⎛
⎜
⎜
⎝
1 2 3 4
0 1 1 1
1 2 1 2
1 1 3 3
⎞
⎟
⎟
⎠
2. Comme pour la méthode gaussienne, on peut récupérer un calcul de l'algorithme de Bareiss lorsqu'on
rencontre l'échec — et plus ou moins de la même manière. Modifiez votre implémentation Bareiss pour
tenez-en compte, puis testez-le à nouveau sur les trois matrices pour vous assurer qu'il se comporte correctement.

La méthode de Dodgson

La méthode de Dodgson est un moyen « intuitivement simple » de calculer un déterminant.[3] Les pseudo-
docode le décrit précisément. (Ici, la numérotation des lignes et des colonnes commence à 1, plutôt que
à 0 comme dans Sage.)

algorithme Dodgsons_method
contributions
• M, une matrice n × n
les sorties
• detM
fais
soit L une matrice (n −1)×(n −1) de 1
pour i {1,...,n −1}
soit m le nombre de lignes de M
soit N une matrice (m −1)×(m −1)
pour j {1,..., m −1}
pour k {1,..., m −1}
soit $N_{j,k} = (M_{j,k} \times M_{j+1,k+1} - M_{j+1,k} \times M_{j,k+1}) / L_{j,k}$
soit L la sous-matrice (m −2)×(m −2) de M commençant à la ligne 2, colonne 2
remplacer M par N

retour M

1. Implémentez ce pseudocode en tant que programme Sage, et testez-le sur les matrices suivantes. (Quelque-
la chose va mal tourner avec l'un d'eux.)

(a) M =

⎛
⎜
3 2 1
5 4 3
6 3 4
⎞
⎟

(b) M =

⎛
⎜
⎜
⎜
1 −4 1
2
-1
4 4
1
3
3 3
4
2
5 2 -1
⎞
⎟
⎟
⎟

(c) M =

⎛
⎜
⎜
⎜
1 2 3 4
0 1 1 1
1 2 1 2
1 1 3 3
⎞
⎟
⎟
⎟

2. Lorsqu'il produit un résultat, le résultat est-il réellement correct ?

3. S'il ne produit pas de résultat, que fait-il exactement ?
4. Dans ce dernier cas, essayez d'effectuer le calcul à la main. Expliquez ce qui ne va pas.[4]

[3] Nommé pour son inventeur, Charles Lutwidge Dodgson, un diacre de l'Église d'Angleterre qui a également occupé une chaise
en mathématiques. Il est mieux connu pour une paire de livres pour enfants qu'il a écrits sous le nom de plume "Lewis Carroll",
Alice et le pays des merveilles et Alice à travers le miroir.

[4] Ne poursuivez pas la tentation d'y remédier à moins que vous ne vouliez un projet de recherche stimulant.

Fonctions individuelles

Dans ce laboratoire, nous considérons un dictionnaire comme une carte ou une fonction à partir d'un ensemble A, le domaine, l'ensemble des
clés du dictionnaire, à un autre ensemble B, l'intervalle, parfois appelé codomaine. La gamme peut
contient en fait plus d'éléments que ceux qui apparaissent dans les valeurs du dictionnaire.

1. Ecrivez une procédure Sage qui accepte un dictionnaire D en entrée et renvoie *True* si D est un-
fonction to-one et *False* sinon.
2. Ecrivez une procédure Sage qui accepte un dictionnaire D en entrée et renvoie *True* si D est un-
fonction à un; sinon, il renvoie à la fois *False* et un contre-exemple.
3. Ecrivez une procédure Sage qui accepte un dictionnaire D et un ensemble B comme entrées et renvoie *True*
si D est sur B, et *False* sinon. Dans ce dernier cas, il doit également retourner tous les éléments de B
qui ne sont pas des valeurs de D.
4. Ecrivez une procédure Sage qui accepte un dictionnaire D et un ensemble B en entrées et renvoie *True* si
D est à la fois un-à-un et sur B, et *Faux* sinon. Dans ce dernier cas, il imprime (ne renvoie pas !)
quelle propriété échoue et le contre-exemple approprié.

Le jeu d'ensemble

L'ensemble de jeu se compose de 81 cartes, chacune ayant quatre propriétés :

• Le nombre d'objets sur la carte est 1, 2 ou 3.
• La couleur du ou des objets est identique : rouge, vert ou violet.
• L'ombrage du ou des objets est identique : plein, rayé ou vide.
• La forme du ou des objets est identique : losanges, ovales ou gribouillis.
Notez que chaque propriété a 3 possibilités, expliquant pourquoi il y a 81 = 3^4 cartes.
Le but du jeu est d'identifier autant de « sets » qu'il y a dans un groupe de cartes. Un « ensemble » est
toute collection de cartes dans laquelle chaque propriété est identique ou différente.
1. Ecrivez une procédure Sage `make_cards()` qui crée et renvoie une liste de toutes les cartes. Chaque carte
doit être un tuple de quatre entiers, où chaque entier indique la propriété. Ainsi, (1,1,3,2) représenterait une carte avec 1 ovale vide rouge.
2. Ecrivez une procédure Sage `is_set()` qui accepte un argument, `L` , qui est une liste de 3 cartes. Elle renvoie *True* si `L` forme un ensemble, et *False* sinon. Vous voudrez peut-être avoir cette procédure
appeler d'autres procédures qui testent si une propriété est la même parmi toutes les cartes ou différente
parmi toutes les cartes.
3. Dans une nouvelle cellule, tapez `import itertools` . Ecrivez une procédure Sage `has_set()` qui en accepte un
argument, `B` , une liste d'au moins 3 cartes, et renvoie *True* si `B` a un ensemble, et *False* sinon. Vous pouvez utiliser la commande `BI = itertools.combinations(B,3)` pour créer un itérateur qui choisissez toutes les combinaisons possibles de 3 cartes de `B` . Vous pouvez ensuite boucler sur `BI` à l' aide d'une boucle `for` .
4. Ecrivez une procédure Sage `prob_set_in(N)` qui prend un entier d'au moins 3 et renvoie le probabilité qu'une planche de taille N choisie au hasard contienne un ensemble. Vous ne voudrez pas essayer
ceci sur une grande valeur de N, car cela prendra beaucoup de temps à s'exécuter. Testez-le uniquement avec N = 3
(quelques secondes, max) et N = 4 (quelques minutes, max). Vous pouvez utiliser la commande `CN = itertools.combinations(CL,N)` pour créer un itérateur qui choisira toutes les combinaisons possibles de N cartes d'une liste `CL` (ici une liste de cartes).

LE NOMBRE DE FAÇONS DE SÉLECTIONNER m ÉLÉMENTS DANS UN ENSEMBLE DE n
285

Le nombre de façons de sélectionner m éléments à partir d'un ensemble de n
Supposons que vous ayez un sac avec n objets distincts et que vous vouliez en sélectionner m. Un tel choix
de balles est appelée une combinaison, par opposition à une permutation où l'ordre compte. Il y a beaucoup de

applications où l'on veut compter le nombre de façons de sélectionner m balles dans un sac de n distincts
objets.[5] Nous écrivons ceci comme $_n C_m$ ou $\binom{n}{m}$. La formule de cette valeur est

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}.$$

1. Ecrivez une procédure Sage qui accepte en entrée n et m et calcule $\binom{n}{m}$.
2. Utilisez votre procédure pour calculer $\binom{n}{m}$ pour n = 2, 3, . . . , 8 et m = 0,...,n.
3. Remarquez-vous un motif dans les nombres ? Astuce : Revenez au Triangle de Pascal.
4. Décrire comment on pourrait utiliser des combinaisons pour implémenter une procédure qui calcule le Pascal
Triangle sans utiliser la récursivité.

5 Par exemple, la loterie PowerBall attribue des prix selon que l'on sélectionne les mêmes cinq numéros
balles sur un total de 69 balles sélectionnées dans une émission télévisée, pas nécessairement dans le même ordre. (il y a un autre critère
aussi, mais à tout le moins, il faut connaître le nombre de façons de sélectionner 5 des 69 balles.

Propriétés des anneaux finis

1. Créez une nouvelle feuille de calcul. Définissez le titre sur « Lab : Propriétés des anneaux finis ». Ajouter d'autres informations
afin de vous identifier, si nécessaire.
Relisez la section sur les « Structures mathématiques dans Sage » dans le chapitre intitulé « Calcul de base
tion.
2. Créez une section intitulée « Arithmétique modulaire : démonstration », puis :
(a) Définir un anneau R comme étant $_{10}$, l'anneau fini de 10 éléments.
(Indice : les notes révisées montrent une façon plus simple de le faire que la démonstration en classe.)
(b) Définissez m comme la valeur de 2 dans R, et calculez 1× m, 2× m, 3× m,...,10× m.

(Indice : si votre réponse au dernier produit est 20, vous vous trompez. Vous devez vert 2 à une valeur de l'anneau R. Les notes montrent comment faire cela.)
(c) Définissez n comme la valeur de 3 dans R, et calculez 1× n, 2× n, 3× n,...,10× n.
(d) Définissez r comme la valeur de 5 dans R, et calculez 1× r, 2× r, 3× r,...,10× r.
(e) Définissez s comme la valeur de 7 dans R, et calculez 1× s, 2× s, 3× s,...,10× s.
(f) Définissez t comme la valeur de 9 dans R, et calculez 1× t, 2× t, 3× t,...,10× t.
3. Pour ce qui suit, écrivez votre réponse dans une zone de texte à la fin de la feuille de calcul. Le sommet de
la zone de texte doit avoir le titre « arithmétique modulaire : analyse ».
(a) Remarquez que $10 \times 2 = 10 \times 3 = \cdots = 10 \times 9$ dans cet anneau. Pourquoi cela a-t-il du sens, étant donné
le module ?
(b) Lequel de m,n, r, s, t énumère tous les nombres de 0 à 9 ?
(c) Quelle propriété les nombres que vous avez énumérés en (b) partagent-ils que les autres nombres ne partagent pas ?

L'horloge des restes chinois

Dans le College Mathematics Journal de mars 2017, Antonella Perucca a décrit un «chinois ré-
Horloge" qui schématise le temps en utilisant les restes de la division.
(1) Pour l'heure h, soit a, b les restes de la division de h par 3 et 4, respectivement.
(2) Pour la minute m, soit c,d,e les restes de la division de m par 3, 4 et 5, respectivement tivement.
(3) Dessinez cinq anneaux concentriques noirs, en marquant :
• le premier et le troisième (à partir du centre) avec trois cercles vides équidistants,
• le deuxième et le quatrième (à partir du centre) avec quatre cercles vides équidistants, et
• le plus à l'extérieur avec cinq cercles vides équidistants.
(4) Enfin, placez
• un cercle bleu sur le cercle ath dans l'anneau le plus intérieur (en comptant dans le sens des aiguilles d'une montre à partir du
Haut),
• un cercle bleu sur le deuxième cercle dans le deuxième anneau le plus à l'intérieur,
• un cercle rouge sur le cième cercle dans le troisième anneau le plus à l'intérieur,
• un cercle rouge sur le dième cercle dans le quatrième anneau le plus à l'intérieur, et
• un cercle rouge sur le cercle eth dans l'anneau le plus à l'extérieur.
Un théorème appelé le théorème des restes chinois nous dit que ce diagramme donne un unique
représentation de l'heure actuelle.
Par exemple, supposons que l'heure actuelle est 5:22. Alors a = 2, b = 1, c = 1, d = 2 et e = 2,

ce qui donne le diagramme
Écrivez un programme de Sage qui, étant donné une heure h et une minute m, produit le chinois correspondant
Horloge restante.

La géométrie des racines radicales

R EMARQUE . Dans ce devoir, nous considérons toutes les solutions comme des nombres complexes a + bi, où $i^2 =$ -1. Lorsqu'on vous demande de « tracer a+bi sur le plan complexe », tracez-le comme le point (a, b). Donc pour exemple, un tracé du nombre complexe 2+3i vous donnerait le point (2,3) :

2 +3i

1. Créez une nouvelle feuille de calcul. Définissez le titre sur « Lab : La géométrie des racines radicales ». Ajouter d'autres informations pour vous identifier, si nécessaire.
2. Utilisez Sage pour résoudre l'équation $x^2 - 1 = 0$. Tracez toutes les solutions sur un plan complexe. Ce n'est pas très intéressant, n'est-ce pas ?
3. Utilisez Sage pour résoudre l'équation $x^3 - 1 = 0$. Tracez toutes les solutions sur un plan complexe, pas le pareil qu'avant. C'est un peu plus intéressant.
4. Utilisez Sage pour résoudre l'équation $x^4 - 1 = 0$. Tracez toutes les solutions sur un troisième plan complexe. Cette peut-être un peu ennuyeux, encore une fois.
5. Utilisez Sage pour résoudre l'équation $x^5 - 1 = 0$. Tracez toutes les solutions sur un quatrième plan complexe. (Vous aurez probablement besoin d'utiliser real_part() et imag_part() ici.) Ce chiffre devrait être arrêtant, surtout si vous reliez les points, mais ne le faites pas ; il suffit de tracer les points, notez le résultat et passer à autre chose.
6. Pouvez-vous deviner? Oui, utilisez Sage pour résoudre l'équation $x^6 - 1 = 0$. Tracez toutes les solutions pour le moment un autre plan complexe. Cela devient probablement ennuyeux, ne serait-ce que parce que vous pouvez probablement devinez le résultat non seulement de celui-ci, mais aussi de. . .
7. Utilisez Sage pour résoudre l'équation $x^7 - 1 = 0$. Tracez toutes les solutions sur un autre plan complexe. Si vous comprenez le motif géométrique, passez à autre chose; sinon, plaignez-vous. Ou, si vous voulez enregistrer temps, vous pouvez probablement deviner que je vais vous dire de tracer les solutions pour

quelques exemples supplémentaires
de $x_n - 1 = 0$, uniquement avec des valeurs plus grandes de n. Tôt ou tard, vous devriez percevoir une géométrie
schéma.
8. Dans une cellule HTML, expliquez pourquoi le motif géométrique des images justifie l'affirmation selon laquelle
les solutions de $x_n - 1$ ont toutes la forme
car
$2\pi k$
m
+ je pèche
$2\pi k$
m
.
Assurez-vous d'expliquer ce que diable k et n ont à voir avec cela. Si cela peut vous aider, regardez d'abord le
parcelle paramétrique de cos $(2\pi t)$ + sin i $(2\pi t)$ pour $t \in [0,1]$, se souvenant qu'avant que vous intrigue
a + bi comme (a, b).
Avant de lever la main en panique, prenez un moment, puis respirez profondément, et repensez
au sens du sinus et du cosinus. Regardez dans un manuel s'il le faut. Vous pouvez le faire, honnêtement !
De même, ne me demandez pas comment créer un tracé paramétrique. Nous l'avons examiné, alors si vous ne le faites pas
rappelez-vous, cherchez-le dans les notes.
9. Définir $\omega = \cos \frac{2\pi}{6} + \text{je pèche} \frac{2\pi}{6}$. En utilisant une boucle à Sage, calculer ω , ω_2 , ..., Q_6 . Tracer chacun
nombre sur le plan complexe et décrivez comment le résultat est cohérent avec vos réponses aux #6 et #8.

---

10. Supposons maintenant $\omega = \cos \frac{2\pi}{n} + i \sin \frac{2\pi}{m}$. À la main, expliquez pourquoi
(2)
$\omega_k = \cos$
$2\pi k$

m
+ je pèche
2 π k
m
.
Tapez cette explication dans votre feuille de travail Sage, en utilisant L A TEX pour que tout soit pur. Si tu
connaître la preuve par induction, cela fonctionnera. Sinon, adoptez l'approche suivante :
• Expliquez pourquoi l'équation (2) est vrai pour k = 1.
• Développez ω 2 et utilisez des identités trigonométriques dont vous êtes censé vous souvenir pour simuler
se plier à la valeur de l'équation (2) pour k = 2.
• Faites la même chose pour ω 3 .
• Enfin, pointez sur un modèle des deux étapes précédentes que vous pouvez répéter à l'infini, donc
que si l'équation (2) est vrai pour une certaine valeur de k, c'est aussi vrai pour la prochaine valeur de k.
11. Il est temps d'améliorer notre jeu. Expérimentez avec x n − a pour de belles valeurs de a et a suffisamment
grande valeur de n. Que vois-tu? Formuler une conjecture quant à la forme géométrique de ces
solutions.
(Le terme « bonnes valeurs d'un » peut signifier l'une des deux choses. Premièrement, cela peut signifier des nombres
de la forme a = b n ; par exemple, si n = 6, alors vous pourriez laisser b = 3, et vous auriez a = 3 6 = 729. Vous travailleriez alors avec l'équation x 6 −729, ce qui n'est pas aussi effrayant qu'il n'y paraît.
Honnête! Deuxièmement, vous pouvez adopter l' approche de Bob Ross , auquel cas n'importe quel nombre est un bon
valeur parce que, vraiment, tous les nombres sont des nombres agréables et heureux. C'est plus difficile, cependant.)
12. Comme avant, faites en sorte que tout soit joli, avec des sections, des commentaires dans des zones de texte, au moins un peu
L A TEX, etc.

---

séquences de Lucas

La suite de Fibonacci est définie comme
f 1 = 1, f 2 = 1, f n+2 = f n + f n+1
.

Les nombres de Fibonacci sont un exemple de ce que les mathématiciens appellent maintenant une séquence de Lucas.
(Plus d'informations sur le lien.) Nous définissons généralement les séquences de Lucas de manière récursive, mais vous pouvez trouver
une « formule fermée » d'une manière similaire à ce que nous avons fait en classe pour la séquence de Fibonacci.
Soit a, b, c et d les deux premiers chiffres de votre carte d'étudiant. Si deux nombres sont les même, changez-les de sorte que les quatre nombres diffèrent. La séquence
$l_1 = a, l_2 = b, l_{n+2} = c\, l_n + d\, l_{n+1}$
est une séquence Lucas que nous appellerons la séquence « *[* insérez votre nom ici *]* ».
1. Dans une cellule Sage HTML, indiquez la définition de la séquence *[* insérez votre nom ici *]* . Utiliser
L A TEX !
2. Dans la même cellule, répertoriez les cinq premiers chiffres de la séquence *[* insérez votre nom ici *]* .
3. Définir une matrice L et un vecteur v qui génèrent la séquence. Par exemple, si les quatre premiers
les chiffres de votre identifiant sont 1, 2, 8 et 9 puis
L =
9 8
dix
et v =
2
1
.
Calculez Lv, L_2 v, L_3 v, L_4 v et L_5 v dans Sage et comparez les résultats à #2. S'ils diffèrent, soit #2 ou #3 est faux. Ou il y a une faute de frappe. Demandez et/ou corrigez avant de continuer.
4. Calculez les « données propres » de L. Extraire les vecteurs propres et les valeurs propres et demander à Sage de les convertir
à la forme radicale. (Les nombres ne devraient plus se terminer par des points d'interrogation.)
5. Construire des matrices Q et Λ telles que $L = Q\, \Lambda\, Q^{-1}$. Utilisation Sage pour vérifier que $L = Q\, \Lambda\, Q^{-1}$.
6. Construire la matrice $M = Q\, \Lambda\, Q^{-1}{}_n$.
Astuce : les notes de cours en discutent ; il nécessite une certaine connaissance de l'algèbre linéaire.
7. Utilisez le produit de M et v pour trouver la forme fermée de la séquence *[* insérez votre nom ici *]* .
Astuce : Encore une fois, les notes devraient vous être utiles si vous avez besoin d'aide.
8. Utilisez la forme fermée pour calculer les cinq premiers nombres de la séquence *[* insérez votre nom ici *]* ,
et comparez vos résultats à ce que vous avez trouvé dans #2 et #3. S'ils diffèrent, vous avez un

problème
ou il y a une faute de frappe ; demandez-moi et/ou corrigez-le !

Introduction à la théorie des groupes

Arrière-plan.

D ÉFINITION . Si un ensemble S et une opération satisfont à la fermeture, associative, identité et
propriétés inverses, alors nous appelons S un groupe sous . Ces propriétés sont définies dans ce qui suit
chemin:

• clôture : $x \otimes y \in S$ pour tout x,y S ;
• associatif : $x \otimes (y \otimes z) = (x \otimes y) \otimes z$ pour tout x,y,z S ;
• identité : on peut trouver $\iota \in S$ tel que $x\, \iota = x$ et $\iota \otimes x = x$ pour tout $x \in S$;
• inverse : pour tout $x \in S$, on peut trouver $y \in S$ tel que $x \otimes y = y \otimes x = \iota$.

E XEMPLE . Les entiers forment un groupe sous addition, car

• l'addition de deux nombres entiers quelconques vous donne un nombre entier ($x + y \in$ pour tout $x,y \in$) ;
• l'addition d'entiers est associative ;
• il existe une identité additive ($x + 0 = x$ et $0 + x = x$ pour tout $x \in$) ; et
• tout entier x a un inverse additif qui est aussi un entier ($x + (-x) = (-x) + x = 0$).

E XEMPLE . Les entiers ne forment pas un groupe sous multiplication, pour deux raisons :

• 0 n'a pas d'inverse multiplicatif 0^{-1} ; et
• les autres entiers a ont des inverses multiplicatifs $1/a$, mais la plupart ne sont pas des entiers.

Un groupe
ne satisfait la propriété inverse que s'il contient les inverses de chaque élément.

Dans cet atelier, vous utiliserez du pseudocode pour écrire du code afin de tester si un ensemble fini est un groupe sous
multiplication. Vous le testerez ensuite sur trois sets, dont deux réussis, et dont un réussit ne pas. Une complication dans ce projet est que la procédure doit dépendre de l'opération, donc vous
ne peut pas simplement écrire une procédure pour une seule opération, seulement.

Pseudocode.

Fermeture. Nous devons vérifier chaque paire $x,y \in S$. Nous pouvons tester si cela est vrai pour « chaque »
élément d'un ensemble fini utilisant des boucles définies.

l'algorithme est_fermé
contributions
S, un ensemble fini
les sorties

vrai si S est fermé par multiplication ; faux sinon
fais
pour s S
pour t S
si st ∈ S
print("échec de fermeture pour", s, t)
retourner faux
retourner vrai
Associatif. Nous devons vérifier chaque triplet x,y, z S, nécessitant des boucles définies. Le pseudo-code
est un exercice.

Identité. Nous pouvons tester si « nous pouvons trouver » une identité à l'aide d'une variable spéciale appelée indicateur
avec une valeur booléenne (parfois appelée un signal). Nous ajustons la valeur du drapeau selon qu'un
candidat continue à satisfaire une propriété connue. Lorsque la boucle se termine, le drapeau indique si
nous avons terminé (c'est-à-dire si nous avons trouvé une identité). La structure des quantificateurs (« on peut trouver... pour
quelconque. . . ”) exige que le pseudocode présume qu'une identité existe jusqu'à preuve du contraire.
algorithme find_identity
contributions
S, un ensemble fini
les sorties
une identité, s'il peut la trouver ; autrement,
fais
pour s S
let may_identity = true
pour t S
si st = t ou ts = t
let may_identity = false
si peut-être_identité = vrai
Retour
print ("pas d'identité")
revenir
Inverse. On cherche un inverse pour chaque élément. Ici encore, nous utilisons un drapeau un drapeau, comme

la logique nous oblige à trouver un inverse. Contrairement au pseudocode précédent, nous supposons une inverse
n'existe pas jusqu'à preuve du contraire ; c'est parce que l'ordre des quantificateurs est inversé (« pour
tout... nous pouvons trouver..." au lieu de "nous pouvons trouver... pour tout..."). Ce pseudocode nécessite également
que nous identifions l'identité de l'ensemble dans l'entrée.

```
algorithme has_inverses
contributions
S, un ensemble fini
ι , une identité de S sous la multiplication
les sorties
vrai si chaque élément de S a un inverse multiplicatif ; faux sinon
fais
pour s S
let found_inverse = false
pour t S
si st = ι et ts = ι
laissez found_inverse = vrai
si found_inverse = false
print("pas d'inverse pour", s)
retourner faux
retourner vrai
```

Les mettre ensemble. Ce pseudocode teste si un ensemble est un groupe sous une opération en en invoquant les quatre algorithmes définis ci-dessus.

```
algorithme is_a_group
contributions
S, un ensemble fini
les sorties
vrai si S est un groupe sous multiplication ; faux sinon
fais
si is_closed(S) et is_associative(S)
soit ι = find_identity(S)
si ι = et has_inverses(S, ι )
retourner vrai
retourner faux
```

Vos tâches. Utilisez L A TEX dans vos feuilles de travail Sage chaque fois que cela est approprié. Deux des ensembles de

3 à 5 sont des groupes ; on ne l'est pas.
1. Étudiez le pseudocode pour la fermeture et écrivez le pseudocode pour un algorithme nommé is_associative
qui teste si un ensemble S est associatif sous multiplication. Vous modifiez essentiellement le pseudo-
docode pour is_closed avec une troisième boucle et modifiez la condition de if de manière appropriée.
2. Écrivez le code Sage pour chacun des cinq algorithmes définis ci-dessus en pseudocode. tu testeras
les sur les ensembles suivants.
3. Définir un anneau R comme étant $_{101}$, l'anneau fini de 101 éléments. (Vous voudrez revoir le laboratoire n°2 si
vous avez oublié comment faire.) Soit S = {1,2,...,100} R; c'est-à-dire que S doit inclure chaque élément
de R sauf 0. Assurez-vous de définir S en utilisant des éléments de R et non des entiers simples. (Encore une fois, vous
voulez revoir le laboratoire n°2 si vous avez oublié comment faire.) Testez votre code Sage sur S ; est-ce que S est un groupe
sous multiplication ? Si non, quelle propriété échoue ?
4. Redéfinissez l'anneau R à $_{102}$, l'anneau fini de 102 éléments. Soit S = {1,2,...,101} R; cette
c'est-à-dire que S doit inclure tous les éléments de R sauf 0. Assurez-vous de définir S en utilisant les éléments de R, et
pas des entiers simples. Testez votre code Sage sur S ; S est-il un groupe sous multiplication ? Si non, lequel
propriété échoue?
5. Définir les matrices
je $_2$ =
dix
0 1
je =
je
0
0 - je
j =
0 1
-1 0
k =
0 je
je 0
et l'ensemble
Q = {I $_2$,–I $_2$,i,–i,j,–j,k,–k}.
Testez votre code Sage sur Q ; Q est-il un groupe sous multiplication ? Si non, quelle

propriété échoue ?
R EMARQUE . Cet ensemble est parfois appelé l'ensemble des quaternions.
6. En utilisant les matrices du problème #4, définissez l'ensemble
$S = \{I_2, -I_2, j, -j\}$.

(a) Vous avez probablement remarqué que S Q. S est-il aussi un groupe ? Si tel est le cas, nous appelons S un sous-groupe de Q.
Si non, quelle propriété échoue ?
(b) L'ensemble S est en fait constitué de matrices de la forme A du Lab #6, Problème #1. Indiquer
dans une cellule HTML la valeur correcte de a pour chaque matrice.

Théorie du codage et cryptographie
Deux applications de l'algèbre et de la théorie des nombres sont la théorie du codage et la cryptographie :
• L'objectif de la théorie du codage est de transmettre des informations de manière fiable, de détecter les erreurs et, lorsque
possible, en les corrigeant.
• Le but de la cryptographie est de transmettre des informations en toute sécurité, afin qu'un espion puisse
ni comprendre ni comprendre le sens de la transmission.
Pour appliquer ces idées mathématiques, les deux nécessitent une étape préalable de transformation du texte en
nombres, et vice versa. Vous rédigerez au moins deux procédures pour aider quelqu'un à cela ; une
sera interactif.
Lors de la transformation de texte en nombres, il est d'abord nécessaire de regrouper le texte en
blocs de taille. Par exemple, la phrase
SORTIR DE L'ESQUIVITE
peuvent être regroupés de plusieurs manières différentes. Une façon consiste à le grouper en blocs de deux :
GE TO UT OF DO DG EX
et une autre consiste à le grouper en blocs de quatre :
GETO UTOF DODG EXXX .
Dans les deux cas, la longueur du message ne divise pas la longueur du groupe de manière égale, nous remplissons donc le message.

sage avec x s ».
Comment est-que quelqu'un peut faire ça? Python a une commande appelée ord() qui vous permet de convertir chaque caractère
agir en nombre. Cette commande convertirait notre phrase en nombres
71, 69, 84, 79, 85, 84, 79, 70, 68, 79, 68, 71, 69 .
Vous remarquerez que j'ai supprimé les espaces et que je n'ai pas encore créé les groupes. Comment grouper ensuite
eux? D'abord, transformez-les en nombres de 0 à 25 en soustrayant le nombre correspondant à A .
Cela nous donne
6, 4, 19, 14, 20, 19, 14, 5, 3, 14, 3, 6, 4 .
Pour les regrouper, nous raisonnons comme suit :
• Nous n'avons pas besoin de lettres minuscules ou de signes de ponctuation pour faire comprendre notre sens.[6]
• Ainsi, chaque valeur à encoder est comprise entre 1 et 26.
• Nous pouvons utiliser un système de numérotation en base 26 pour coder n'importe quel groupe de lettres.
Ainsi, pour coder un groupe de quatre lettres avec des valeurs numériques a, b, c, d, nous pouvons calculer
$m = a + 26b + 26^2 c + 26^3 d$.
En général, pour coder un groupe de n lettres avec des valeurs numériques a_1 , a_2 ,..., a_n , calculez
$m = a_1 + 26a_2 + 26^2 a_3 + \cdots + 26^{n-1} a_n$.
Pour décoder un groupe encodé m de n caractères, procédez n fois comme suit :
• déterminer le reste de la division de m par 26 et l'appeler a ;
• convertir un dos en personnage ; et enfin,
• remplacer m par $m-a/26$.
Vous devez écrire au moins deux procédures.
6 Anecdotes connexes : Les langues écrites anciennes utilisaient généralement toutes les lettres majuscules sans ponctuation ni même
espacement.

---

1. La première procédure, encode() , prend deux entrées : un entier n et une liste L de caractères. Il
convertit chaque caractère en nombre, organise les nombres en groupes de n caractères, convertit ensuite chaque groupe en un nombre en utilisant la formule ci-dessus. Il renvoie une liste M de
numéros, un pour chaque groupe. Vous pouvez supposer que n divise la longueur de L. (C'est un peu
plus difficile si vous devez comprendre le rembourrage.)

2. La deuxième procédure est interactive. Il offre à l'utilisateur une zone de texte et un curseur. Dans la zone de texte,
l'utilisateur saisit un message. Dans le curseur, l'utilisateur sélectionne de 2 à 6 caractères. La procédure
prend ces valeurs et les envoie à encoder, puis imprime les numéros de liste renvoyés par `encode()` .
Pour un crédit supplémentaire, vous pouvez également :
3. Implémentez une troisième procédure, `decode()` , prend deux entrées : un entier n et une liste M d'entiers.
Il convertit chaque entier en un groupe de n entiers, puis convertit chaque entier en un caractère.
Pour cela, vous aurez peut-être besoin de la commande Python `chr()` .
4. Cythonize `encode()` ou `decode()` . Au moins un type de variable doit être déclaré.

Fractions continues

Une représentation de fraction continue d'un nombre a la forme suivante :

$$un_0 + \cfrac{1}{un_1 + \cfrac{1}{un_2 + \cfrac{1}{a3 + \cfrac{1}{\ddots}}}}.$$

Les fractions continues ont un certain nombre de propriétés fascinantes, dont certaines seront explorées dans
ce laboratoire. Le pseudocode suivant calculera une fraction continue jusqu'à m places :

algorithme fraction_continue
contributions
• b, un nombre réel
• m, le nombre de places pour développer la fraction continue
les sorties
• une liste $(a_0, a_1, ..., a_m)$ listant les entrées de la fraction continue
fais
soit L une liste de m +1 zéros
soit i = 0
tandis que $i \leq m$ et b = 0
mettre L_i au plancher de b
soit $d = b - L_i$
si d = 0
remplacer d par $1/d$

soit b = d
ajouter 1 à je
retour L
1. Implémentez ce pseudocode en tant que procédure Sage. Indice : la procédure floor() de Sage pourrait s'avérer
utile ici.
2. Utilisez votre code pour calculer les approximations de fraction continue à au plus 20 places pour $12/17$,
$17/12$, $4/15$, $15/4$, $729/1001$ et $1001/729$.
3. Notez au moins deux propriétés que vous avez remarquées concernant les fractions continues ci-dessus. Expliquer
pourquoi ces propriétés ont un sens.
4. Utilisez votre code pour calculer les approximations de fraction continue à au plus 20 places pour
41, $1/41$, $1+\sqrt{5}/2$, $2/1+\sqrt{5}$, $1+\sqrt{7}/3$ et $3/1+\sqrt{7}$.
5. Notez au moins deux propriétés que vous remarquez à propos de ces fractions continues. L'un des
les propriétés peuvent être les mêmes qu'avec les nombres rationnels, mais les autres ne le devraient certainement pas.
6. Utilisez votre code pour calculer les approximations de fraction continue à au plus 20 places pour e,
e, $\sqrt[3]{e}$, $\sqrt[4]{e}$, $\sqrt[5]{e}$ et $\sqrt[6]{e}$.
7. Notez toutes les régularités que vous remarquez à propos de ces fractions continues. (Il est peu probable que vous trouviez
un qui est commun aux numéros précédents, mais en trouver un serait bien intéressant.)

Bibliographie

[1] Michael H. Albert, Richard J. Nowakowski et David Wolfe. Leçons en jeu. AK Peters, Ltd., Wellesley, Massachussetts, 2007.
[2] Elwyn Berlekamp, John Conway et Richard Guy. Des moyens gagnants pour vos jeux mathématiques. AK Peters /
CRC Press, deuxième édition, 2001.
[3] Anna M. Bigatti. Calcul de la série de Hilbert-Poincaré. Journal d'algèbre pure et appliquée, 119 (3): 237-253, 1997.
[4] Anna M. Bigatti, Massimo Caboara et Lorenzo Robbiano. Calcul de bases de Gröbner inhomogènes. Journal de Symbolic Computation, 46(5):498-510, 2010.
[5] Alberto Damiano, Graziano Gentili et Daniele Struppa. Calculs dans l'anneau de polynomiaux. Journal of Symbolic Computation, 45:38-45, janvier 2010.
[6] Vladimir Gerdt et Youri Blinkov. Bases involutives des idéaux polynomiaux. Mathématiques et informatique en simulation
lation, 45(5-6):419-541, mars 1998.
[7] Maxime. Maxima, un système de calcul formel. Version 5.38.1. http://maxima.sourceforge.net/ , 2016.
[8] Marin Mersenne. Cogitata Physica-Mathematica. 1644.
[9] Diophante d'Alexandrie. Arithmétique. 1670. Avec notes marginales de Pierre de Fermat.

[10] Euclide d'Alexandrie. Éléments. c. 300 avant JC.
[11] Jamie Uys. Les dieux doivent être fous, 1980. film distribué par 20th Century Fox.

Indice
