Machine Translated by Google

Sage pour les étudiants de premier cycle (version en ligne)

Grégory V. Bard

Département de mathématiques, de statistiques et d'informatique, Université du Wisconsin-Stout, Menomonie, Wi, 54751 Adresse e-mail : bardg@uwstout.edu

Machine Translated by Google

Classification des matières mathématiques 2010. Primaire : 15-04, 34-04, 65-04, 90-04, 97M10. Secondaire : 11-04, 12-04, 28-04, 40-04, 68U05.

Machine Translated by Google

C'est avec chaleur que je dédie ce livre à mon mari, Patrick, qui m'a soutenu tout au long de ma carrière, dans les bonnes années comme dans les mauvaises années, et qui a passé de nombreuses soirées seule parce que je travaillais sur ce livre.

Machine Translated by Google

Machine Translated by Google

Contenu

Préface — Comment utiliser ce livre xv

Remerciements XIX

Chapitre 1. Bienvenue chez Sage ! 1
1.1. Utiliser Sage comme calculatrice 1.2. 1
Utiliser Sage avec des fonctions courantes 1.3. Utiliser 3
Sage pour la trigonométrie 1.4. Utiliser Sage 7
pour représenter graphiquement en deux dimensions 9
1.4.1. Contrôler la fenêtre d'affichage d'un graphique 1.4.2. 11
Superposer plusieurs graphiques dans un même graphique 1.5. 14
Matrices et Sage, première partie 1.5.1. Un 19
avant-goût des matrices 1.5.2. Complications 19
de la conversion de systèmes linéaires 1.5.3. Exécuter le RREF 20
dans Sage 1.5.4. Résumé de base 1.5.5. 22
La matrice identité 1.5.6. Un 24
défi à relever pour s'entraîner seul 25
1.5.7. La matrice de Vandermonde 1.5.8. Les cas semi- 25
rares 1.6. Créer vos propres fonctions 26
dans Sage 1.7. Utiliser Sage pour 28
manipuler des polynômes 1.8. Utiliser Sage pour 30
résoudre des problèmes symboliquement 1.8.1. Résolution 34
de formules à une variable 1.8.2. Résolution de formules à 37
plusieurs variables 1.8.3. Systèmes d'équations 37
linéaires 1.8.4. Systèmes d'équations non linéaires 38
1.8.5. Cas avancés 1.9. Utiliser Sage comme 39
solveur numérique 1.10. Obtenir de l'aide en cas de 40
besoin 1.11. Utiliser Sage pour 43
calculer des dérivées 1.11.1. Représentation 43
graphique de f(x) et f(x) 47
49
50

Machine Translated by Google

1.11.2. Dérivées d'ordre supérieur ; 1.12. 50
Calcul d'intégrales avec Sage ; 1.13. Partage des 51
résultats de vos travaux. 60

Chapitre 2. Projets amusants avec Sage 2.1. 63
Microéconomie : Calcul d'un prix de vente 2.2. Biologie : Artères 64
bouchées et loi de Poiseuille 2.3. Optimisation industrielle : Expédition 69
de taconite 2.4. Chimie : Équilibrage des réactions avec des 71
matrices 2.4.1. Contexte 2.4.2. Cinq exemples intéressants 2.4.3. La 73
méthode lente : Dérivation 73
de la méthode 2.4.4. Équilibrage 74
rapide 2.5. Physique : Projectiles balistiques 2.5.1. Notre 75
premier exemple de trajectoires balistiques 77
2.5.2. Notre deuxième exemple de trajectoires 79
balistiques 2.5.3. Tir de contre-batterie : 2.5.4. Votre défi : Une fusée 80
à plusieurs étages 2.6. Cryptologie : Attaque p − 1 de Pollard sur RSA 82
2.6.1. Contexte : Nombres B-lisses et B ! 83
84
85
85
2.6.2. La théorie derrière l'attaque 2.6.3. Calcul 2(B!) 87
mod N 2.6.4. Votre défi : Réaliser tout cela 88
dans Sage 2.6.5. Protections contre l'attaque factorielle de Pollard 2.7. Mini- 89
projet sur les tracés vectoriels de champ électrique 89
90

Chapitre 3. Techniques avancées de traçage 3.1. Annotation des 91
graphiques pour plus de clarté 3.1.1. Étiquetage des axes 91
des graphiques 3.1.2. Grilles et représentation graphique 91
de graphiques de type calculatrice 3.1.3. Ajout de flèches et de texte pour étiqueter 93
des entités 3.1.4. Représentation graphique d'une intégrale 3.1.5. Lignes 94
pointillées et discontinues 3.2. Graphiques de 95
certaines fonctions hyperactives 3.3. Tracé polaire 96
3.3.1. Exemples de graphiques polaires 99 97
98

3.3.2. Problèmes pouvant survenir occasionnellement 100
3.4. Représentation graphique d'une fonction implicite 102
3.5. Tracés de contour et ensembles de niveaux 104
3.5.1. Une application à la thermodynamique 107
3.5.2. Application à la microéconomie (équations de Cobb-Douglas) 110
3.6. Tracé paramétrique 2D 112
3.7. Graphiques de champs vectoriels 114
3.7.1. Gradients et tracés de champs vectoriels 3.7.2. 115
Une application de la physique 116

Machine Translated by Google

3.7.3. Gradients versus courbes de niveau 3.8. 119
Courbes log-log 3.9. 120
Situations rares 122

Chapitre 4. Fonctionnalités avancées de Sage 127
4.1. Utiliser Sage avec des fonctions et des équations multivariables 4.2. Travailler avec de grandes 127
formules dans Sage 4.2.1. Finances personnelles : Prêts hypothécaires 129
4.2.2. Physique : Gravitation et satellites 4.3. Dérivées et 129
gradients en calcul multivarié 4.3.1. Dérivées partielles 4.3.2. 132
Gradients 4.4. Matrices dans Sage, deuxième partie 4.4.1. Définition de quelques exemples 134
4.4.2. Multiplication et exponentiation de 134
matrices 4.4.3. Résolution 135
de systèmes droite et gauche 4.4.4. Inverses de 135
matrices 4.4.5. Calcul du noyau d'une matrice 4.4.6. 135
Déterminants 4.5. Opérations vectorielles 4.6. Travailler avec les entiers et la 136
théorie des nombres 4.6.1. Le pgcd et le pcm 4.6.2. En savoir 137
plus sur les nombres premiers 4.6.3. 139
À propos de la fonction phi d'Euler 4.6.4. Les diviseurs d'un nombre 140
4.6.5. Autre signification de Tau 142
4.6.6. Arithmétique modulaire 4.6.7. 144
Lectures complémentaires en théorie des nombres 4.7. Quelques commandes 145
mineures de Sage 4.7.1. Arrondi, planchers 146
et plafonds 4.7.2. Combinaisons et permutations 4.7.3. 147
Les fonctions trigonométriques hyperboliques 4.8. 148
Calculer des limites de manière expresse 4.9. Nuages 149
de points dans Sage 4.10. Créer vos propres 151
régressions dans Sage 4.11. Calculer en 152
octal, binaire et hexadécimal 163 153
153
153
153
155
156
157
161

4.12. Sage peut-il faire du Sudoku ? 163
4.13. Mesurer la vitesse de Sage 164
4.14. Nombres énormes et Sage 165
4.15. Utilisation de Sage et LATEX 166
4.16. Matrices dans Sage, troisième partie 167
4.16.1. Introduction aux vecteurs propres 167
4.16.2. Trouver efficacement les valeurs propres dans Sage 169
4.16.3. Factorisations matricielles 170
4.16.4. Résolution approximative de systèmes linéaires par la méthode des moindres carrés 171

Machine Translated by Google

4.17. Calcul des polynômes de Taylor ou de MacLaurin 4.17.1. Exemples de 173
polynômes de Taylor 4.17.2. Une application : comprendre 174
comment g change 4.18. Minimisations et multiplicateurs de Lagrange 4.18.1. 175
Optimisation sans contrainte 4.18.2. Optimisation sous contrainte par 177
multiplicateurs de Lagrange 4.18.3. Un exemple de 177
multiplicateurs de Lagrange dans Sage 4.18.4. Quelques problèmes appliqués 4.19. 179
Sommes et séries infinies 4.19.1. Vérification des identités de sommation 179
4.19.2. Les séries géométriques 4.19.3. Utilisation 181
de Sage pour guider une preuve de sommation 182
4.20. Arithmétique de très haute précision 4.21. Systèmes 183
d'inéquations et programmation linéaire 183
4.21.1. Un exemple simple 4.21.2. Fonctionnalités pratiques en pratique 184
4.21.3. Le polyèdre d'un programme linéaire 4.21.4. 186
Programmes linéaires entiers et variables booléennes 4.21.5. Lectures 186
complémentaires sur la programmation 186
linéaire 4.22. Équations différentielles 4.22.1. Quelques 188
exemples simples 4.22.2. Un problème à valeur initiale 4.22.3. 190
Représentation graphique d'un champ de pentes 4.22.4. Le problème de la torpille : 190
travailler avec des paramètres 4.23. Transformées de Laplace 4.23.1. 191
Transformer f(t) en L(s) 191
192
194
195
197
199
199
4.23.2. Calcul de la transformée de Laplace « La voie longue » 200
4.23.3. Transformation de L(s) en f(t) 201
4.24. Calcul vectoriel dans Sage 4.24.1. 202
Notation des fonctions vectorielles 4.24.2. Calcul de la 202
matrice hessienne 4.24.3. Calcul du laplacien 203
4.24.4. La matrice jacobienne 4.24.5. 204
Divergence 4.24.6. Vérification d'une 205
ancienne identité 4.24.7. Le 206
recourbement d'une fonction vectorielle 207
4.24.8. Un défi : vérifier certaines identités de recourbement 208
4.24.9. Intégrales multiples 212
212

Chapitre 5. Programmation en Sage et Python 215
5.1. Répétition sans ennui : la boucle For ; 5.1.1. Générer des 216
tableaux avec Sage ; 5.1.2. Formater 216
soigneusement la sortie ; 5.1.3. Listes arbitraires ; 217
5.1.4. Boucles avec 218
accumulateurs. 219

Machine Translated by Google

5.1.5. Utiliser Sage pour trouver une limite numériquement 5.1.6. 220
Boucles For et polynômes de Taylor 5.1.7. Si vous savez 222
déjà programmer… 223
5.2. Écrire des sous-routines 5.2.1. Un 224
exemple : travailler avec des pièces de monnaie 5.2.2. Un défi : 224
une caisse enregistreuse 5.2.3. Un exemple : concevoir 226
des aquariums 5.2.4. Un défi : un silo cylindrique 5.2.5. Combiner 226
des boucles et des sous-routines 5.2.6. Un autre défi : 229
totaliser une séquence 5.3. Boucles et méthode de Newton 229
5.3.1. Qu'est-ce que la méthode de Newton ? 230
231
231
5.3.2. Méthode de Newton avec boucle For 5.3.3. Test du 233
code 235
5.3.4. Représentations numériques et exactes 237
5.3.5. Utilisation des paramètres facultatifs et obligatoires 237
5.3.6. Renvoyer une valeur et imbriquer des sous-routines 239
5.3.7. Un défi : trouver des tangentes parallèles 242
5.3.8. Un défi : la méthode Halley 243
5.4. Introduction au contrôle de flux 244
5.4.1. Contrôle de la verbosité 244
5.4.2. Interlude théorique : Quand la méthode de Newton devient folle 247
5.4.3. Arrêt prématuré de la méthode de Newton 249
5.4.4. La liste des comparateurs 252
5.4.5. Un défi : combien de nombres premiers sont inférieurs à x ? 252
5.5. Autres concepts de contrôle de flux 252
5.5.1. Déclencher une « exception » 253
5.5.2. La construction If-Then-Else 5.5.3. Défi 255
facile : Fin de semestre du registraire, partie 1 5.5.4. Un défi plus difficile : Fin de semestre, 257
partie 2 5.6. Boucles While contre boucles For 5.6.1. Une question sur les 257
factorielles 5.6.2. Un défi : Trouver le prochain 258
nombre premier 5.6.3. Méthode de Newton avec 258
une boucle While 5.6.4. L'impact des racines répétées 5.6.5. Factorisation 259
par division d'essai 5.6.6. Un mini-défi : Division d'essai, arrêt 259
anticipé 5.6.7. Un défi : Une caisse enregistreuse 260
améliorée 5.7. Fonctionnement des tableaux et des 262
listes 5.7.1. Listes de points et tracé 5.7.2. Un défi : Créer la commande Plot 263
5.7.3. Opérations sur les listes 5.7.4. Boucle sur un tableau 5.7.5. 264
Un défi : Rapport de résultats d'entreprise 265
5.7.6. Moyennage de quelques nombres 265
266
267
269
270
270

Machine Translated by Google

5.7.7. Mini-défi : moyenne avec et sans chute de 272
5.7.8. Mini-défi : Quiz « Doubler le comptage du plus haut » 272
5.7.9. Un défi : La course du registraire, troisième partie 272
5.7.10. Un utilitaire : renvoyer uniquement des racines réelles, rationnelles ou entières 272
5.8. Où allez-vous maintenant ? 274
5.8.1. Ressources supplémentaires sur la programmation Python 274
5.8.2. Qu'est-ce qui a été laissé de côté ? 275

Chapitre 6. Création de pages web interactives avec Sage 6.1. Le processus en 279
six étapes pour la création d'interactions 6.2. L'interaction tangente Étape 0 : 280
Développement du concept Étape 1 : Conception 281
d'une fonction Sage Étape 2 : Peaufinage de 281
cette fonction Étape 3 : Conversion en fonction 282
interactive Étape 4 : Insertion dans un 284
modèle web Étape 5 : Développement de la page web 6.3. Un défi 285
pour le lecteur 6.4. L'interaction optimale en aquarium 286
6.5. Sélecteurs et cases à cocher 6.6. L'interaction 288
intégrale définie 288
288
289
290

Annexe A. Que faire en cas de frustration ! 293

Annexe B. Transition vers Sage Cloud B.1. Qu'est-ce que 299
Sage Cloud ? 299
B.2. Premiers pas dans le Cloud B.3. Autres 300
fonctionnalités du Cloud 301

Annexe C. Autres ressources pour Sage 303

Annexe D. Systèmes linéaires à solutions infinies D.1. Exemple d'ouverture D.2. 307
Autre exemple D.3. Approfondissement 307
D.3.1. Réexamen intéressant 310
des solutions uniques D.3.2. 312
Deux équations et quatre inconnues D.3.3. Deux équations et quatre inconnues, 312
mais aucune solution : D.3.4. Quatre équations et trois 313
inconnues D.4. Idées fausses D.5. Définitions formelles de REF et RREF D.6. 313
Notations alternatives D.7. Interprétations géométriques en 3 314
dimensions D.7.1. 314
Visualisation de ces cas D.7.2. Interprétations géométriques 315
en dimensions supérieures D.8. 316
Notation des variables muettes D.9. Solutions aux défis 316
317
318
318
319

Machine Translated by Google

Annexe E. Installation de Sage sur votre ordinateur personnel 321

Annexe F. Index des commandes par nom et par section 325

Machine Translated by Google

Machine Translated by Google

Préface — Comment l'utiliser Livre

Concurrent open source et gratuit de logiciels coûteux comme Maple, Mathematica, Magma et Matlab, Sage offre à toute personne disposant d'un navigateur web la possibilité d'utiliser des logiciels mathématiques de pointe et d'afficher ses résultats, souvent avec des graphismes époustouflants. Je suis sûr que vous trouverez Sage bien plus simple d'utilisation qu'une calculatrice graphique et bien plus puissant.

Il n'est pas nécessaire de lire ce document en entier, tout comme vous ne lirez jamais un dictionnaire du début à la fin. Ce livre est idéal pour l'auto-apprentissage, mais si vous le lisez en classe, votre professeur vous indiquera les numéros de section correspondant à ce que vous devez lire.

Pour les autres lecteurs, qui lisent indépendamment d'une classe spécifique, j'ai Voici quelques suggestions : •

Le chapitre 1 contient les bases, c'est-à-dire ce que vous devez absolument savoir. La plupart des tâches courantes dans Sage ont été soigneusement conçues pour être aussi intuitives que possible, avec une notation aussi proche que possible de la méthode mathématique au tableau blanc ou au crayon. Je vous recommande de ne jamais essayer de lire plus de trois sections entières en une journée. Sinon, votre cerveau aura trop de choses à assimiler tout en conservant une expérience amusante et nouvelle. Cependant, une à deux portions par jour devraient être faciles à digérer.

Remarque : Personnellement, je recommande de lire uniquement le chapitre 1, puis de commencer à jouer seul. La meilleure façon d'apprendre de nouvelles choses

Machine Translated by Google

L'utilisation d'un logiciel informatique est récréative. La table des matières (au début du livre) ou l'index des commandes (à la fin) vous permettront de trouver comment effectuer une tâche que vous n'avez pas encore apprise. Si vous êtes perdu, consultez l'annexe A, « Que faire en cas de frustration ». • Le chapitre 2 propose des projets utilisant Sage pour résoudre des problèmes mathématiques tels qu'ils se présentent dans d'autres matières : cryptographie, physique, chimie, biologie, etc. Ces projets sont principalement destinés à être traités après la lecture du chapitre 1. Cependant, dans certains cas, une autre section d'un chapitre plus numéroté sera également requise. Bien entendu, dans ces cas, la dépendance est clairement indiquée. Ces projets sont destinés à être assignés pendant les week-ends.

Ils ne sont pas assez grands pour être des projets semestriels, mais ils sont trop grands pour être des problèmes de devoirs.

• Le chapitre 3 est consacré à la création de tracés et de graphiques élégants. J'y ai mis tout ce que j'ai pu. Cependant, certains types de tracés, notamment les tracés 3D, animés et multicolores, sont difficiles à représenter dans un livre imprimé en noir et blanc. C'est pourquoi ces types de tracés sont traités dans l'annexe en ligne, exclusivement électronique, de ce livre « Tracés en couleur, en 3D et animations », téléchargeable sur mon site web www.gregorybard.com. • Le chapitre 4 est volumineux et contient de courtes sections qui abordent des sujets mathématiques avancés. Sage peut traiter une grande variété de problèmes mathématiques. Ce chapitre a été conçu pour que les sections individuelles puissent être lues indépendamment. Il n'est pas nécessaire de lire les sections du chapitre 4 dans l'ordre, sauf si vous le souhaitez. • Le chapitre 5 explique comment programmer en Python avec Sage. En tant que système de calcul formel, Sage est basé sur le langage informatique Python. Vous pouvez donc utiliser les commandes Python dans Sage. Parfois, c'est très utile : par exemple, vous pouvez écrire des boucles for pour créer des tables très facilement. D'autres fois, il est pratique d'écrire des programmes informatiques entiers en Python avec Sage comme interface. C'est encore plus vrai avec Sage Cloud. • Le chapitre 6 aborde la création de pages web interactives (appelées applications) avec Sage. Le processus est remarquablement simple. J'ai moi-même un processus en six étapes que j'utilise. Vous pouvez créer des interactions extrêmement intéressantes avec Sage, et je parie que vous serez surpris de la facilité avec laquelle vous pouvez créer les vôtres. • L'annexe A est intitulée « Que faire en cas de frustration », où je propose une liste de questions à vous poser si vous êtes bloqué dans une situation où Sage refuse de répondre à vos commandes comme vous le souhaiteriez. Dans presque tous les cas, l'un des éléments listés vous proposera une solution à la frustration.

Machine Translated by Google

• L'annexe B concerne la familiarisation avec Sage Cloud. Le cloud computing est une nouvelle façon innovante de gérer vos données en utilisant Internet, plutôt que votre ordinateur local, comme lieu de stockage de vos fichiers.

Remarque : En général, les utilisateurs souhaitent passer à Sage Cloud après avoir terminé le chapitre 1, avant d'aborder les projets du chapitre 2 ou les travaux avancés des chapitres 4 à 6. Je pense que le chapitre 3 peut être appris avant ou après l'apprentissage de Sage Cloud, sans aucun inconvénient pour l'utilisateur. •

L'annexe C propose des ressources complémentaires sur Sage, destinées aux lecteurs. qui voudrait en savoir plus que ce que contient ce livre.

L'annexe D est ma réponse au fait que les élèves ont souvent du mal à comprendre les systèmes d'équations linéaires ayant un nombre infini de solutions. Ce concept est souvent perçu comme une idée, mais les détails manquent souvent lorsque je souhaite que les élèves les connaissent.

J'ai donc décidé d'être extrêmement minutieux et de consacrer un grand nombre de pages à expliquer ce sujet (certes difficile). J'espère que cela sera utile aux étudiants.

• L'annexe E est à lire si vous souhaitez apprendre à installez Sage sur votre machine locale.

• L'annexe F est un index de seize pages (!) de toutes les commandes de ce livre, pour votre référence. Un merci spécial à Thomas Suiter, qui l'a préparé.

• L'annexe électronique en ligne couvrira le traçage en couleur, les animations et les graphiques 3D. Ces sujets ne sont pas adaptés à un livre noir et blanc et ne peuvent donc pas être imprimés avec ces couvertures. Vous pouvez la télécharger sur mon site web.

www.gregorybard.com en cliquant sur le lien, à gauche de l'écran, pour Sage.

Plongeons-nous ! Maintenant !

Sans hésiter, ouvrez immédiatement un navigateur web et saisissez l'URL suivante :

https://sagecell.sagemath.org/ Vous êtes maintenant connecté au « Sage Cell Server ». Vous verrez alors une grande fenêtre (appelée cellule) et un gros bouton « Évaluer ». Dans cette cellule, saisissez ce qui suit.

```
résoudre( x^2 - 16 == 0, x )
```

puis cliquez sur « Évaluer ».

Vous devriez recevoir la réponse

```
[x == -4, x == 4]
```

qui est clairement l'ensemble correct de deux solutions à l'équation $x^2 - 16 = 0$. Vous avez maintenant résolu votre premier problème mathématique avec Sage. Félicitations.

Bienvenue chez Sage ! Vous êtes maintenant prêt à commencer la lecture du chapitre 1.

Machine Translated by Google

Machine Translated by Google

Remerciements

Tout d'abord, je tiens à remercier William Stein, le créateur de Sage, pour ses encouragements, son aide et son accès aux ressources de supercalcul qui ont rendu possibles mes recherches en cryptanalyse. Le professeur Stein est le fondateur de Sage, son architecte et organisateur en chef, et la source d'enthousiasme de la communauté Sage, qui compte aujourd'hui des centaines de personnes réparties dans de nombreux pays et continents. De nombreuses lignes originales de Sage, ainsi que de Sage Cloud, ont été personnellement codées par le professeur Stein.

Ce livre a débuté sous le titre « Guide de Sage » alors que j'enseignais pendant cinq semaines à Pékin à l'Académie chinoise des sciences, Académie de mathématiques et des sciences des systèmes, Laboratoire clé de mécanisation des mathématiques, durant l'été 2010. Je suis très reconnaissant d'avoir eu l'occasion unique de travailler avec des étudiants aussi avancés. J'enseignais à partir d'un autre ouvrage que j'ai écrit, Algebraic Cryptanalysis, publié par Springer en 2009. J'ai donné plusieurs heures de cours sur Sage, mais il est devenu évident que des instructions écrites seraient plus utiles aux étudiants. On peut les feuilleter tout en travaillant sur Sage, et c'est ainsi qu'est né le document qui allait devenir ce livre. Je suis extrêmement reconnaissant d'avoir eu l'occasion de visiter cette excellente institution. À mon départ, ce document comptait 37 pages, mais de minuscules glands naissent de grands chênes.

Comme de nombreux chercheurs qui inventent un nouvel algorithme dans le cadre de leur thèse de doctorat à l'Université du Maryland, j'ai dû écrire du code pour prouver son efficacité, sa justesse et son efficience. Cependant, ce code est souvent abandonné après l'obtention du doctorat. Je tiens à remercier Martin Albrecht, qui m'a fait découvrir la communauté Sage et qui a permis l'intégration du code de ma thèse dans les bibliothèques Sage, dans le cadre de ce qui est devenu le projet M4RI. Sans Martin, je n'aurais jamais entendu parler de Sage, et le code de ma thèse ne serait pas utilisé par des personnes dispersées dans le monde.

Il s'avère que M4RI est un projet en évolution, continuellement mis à jour par de nombreux chercheurs différents, et contient probablement aujourd'hui très peu de mes travaux originaux.

Machine Translated by Google

Lignes. C'est un exemple de l'évolution d'un logiciel. Vous trouverez des informations sur l'algorithme original au chapitre 9 de Cryptanalyse algébrique.

Je tiens également à remercier Robert Beezer et Robert Miller pour m'avoir encouragé à utiliser Sage dans mon enseignement et mes recherches, respectivement. L'excellent livre numérique (et gratuit) de Robert Beezer, « A First Course in Linear Algebra », a été une source d'inspiration pour moi lors de l'écriture de ce livre.

Le Sage Cell Server a rendu Sage accessible à un public bien plus large qu'on ne l'aurait jamais imaginé. Même pour les utilisateurs expérimentés, l'expérience utilisateur s'est considérablement améliorée. Ce travail important et complexe a été réalisé par Jason Grout, Ira Hanson, Steven Johnson, Alex Kramer et William Stein. À l'origine, Sage était destiné aux étudiants de troisième cycle, aux professeurs et peut-être aux étudiants de dernière année en mathématiques. Puis, le Sage Notebook Server est arrivé, permettant aux étudiants de premier cycle en milieu de carrière de l'utiliser. Aujourd'hui, grâce au Sage Cell Server, il est possible d'utiliser Sage même en cours de pré-calcul, si on le souhaite.

L'American Mathematics Society a fait preuve d'une grande souplesse à mon égard concernant les délais et les nombreuses illustrations de ce manuel. Produire un livre requiert un nombre étonnamment élevé d'étapes, et je suis reconnaissant du soutien d'Ina Mette, responsable des acquisitions, et de Marcia Almeida, son assistante. Je suis particulièrement reconnaissant à Ina Mette pour son dynamisme et son enthousiasme qui m'ont encouragé à transformer ce qui n'était qu'un petit guide en un livre plus complet (publié). Ce projet n'aurait pas vu le jour sans son soutien.

Je tiens à remercier Andrew Novocin de m'avoir appris à inclure des fichiers *.png dans des documents LaTeX, ce qui m'a épargné le temps de conversion manuelle des formats de fichiers de la multitude de captures d'écran qui composent ce livre. Sans cela, j'aurais peut-être abandonné ce projet de livre.

Aucun auteur ne peut relire efficacement son propre travail, mais c'est particulièrement mon cas. Je suis très reconnaissant de l'aide apportée par les relecteurs Joseph Robert Bertino, Joseph Loeffler et Thomas Suiter, qui ont chacun apporté des suggestions et des corrections sur certaines parties de ce document. Thomas Suiter a particulièrement œuvré pour l'index, et Joseph Bertino a assumé avec noblesse (sans se plaindre) l'énorme charge de travail que j'ai dû accomplir ces dernières années, tant pour ce livre que pour d'autres projets. Bien entendu, toute erreur subsistante est de ma responsabilité.

Après l'obtention de mon doctorat à l'Université du Maryland, j'ai occupé mon premier poste à l'Université Fordham, dans le Bronx, à New York, où j'ai été nommé professeur assistant invité Peter M. Curran pour une durée de quatre ans. J'ai progressé sur ce projet pendant mon séjour à Fordham, après mon été en Chine. J'ai reçu les encouragements et les suggestions de Shaun Ault, Melkana Brakalova, Armand Brumer, Michael Burr, Bill Campbell, Janusz Golec, Maryam Hastings, William Hastings, Nick Kintos, Robert Lewis, Damian Lyons, Ian Morrison, Leonard Nissim, Cris Poor, Benjamin Shemmer, Shapoor Vali et Kris Wolff. L'ouvrage de Fred Marotto, Introduction to Mathematical Modeling Using Discrete Dynamical Systems, publié par Cengage en 2005, a également été une source d'inspiration pour ce travail.

Machine Translated by Google

Je n'ai pas eu souvent l'occasion d'échanger sur nos centres d'intérêt communs. Jack Kamis, Mila Martynovsky et Michael « Misha » Zigelbaum m'ont prodigué des conseils pédagogiques précieux et pratiques.

Shapoor Vali et moi avons écrit nos livres quasiment en même temps. Nous avons donc dû composer les chapitres, relire les brouillons et rédiger les propositions de livres simultanément. Nous avons tous deux traversé un grand désert, non sans souffrance, et nous pouvons désormais profiter ensemble de l'oasis qui se trouve de l'autre côté.

De nombreux étudiants de premier cycle de Fordham m'ont prodigué de nombreux conseils, corrections, pistes d'amélioration et idées. Au risque d'exclure accidentellement quelqu'un, je souhaite mentionner les étudiants suivants de Fordham : James Compagnoni, Gray Crenshaw, Dan DiPasquale, Patrick Fox, Alex Golec, Simon Kaluza, Kyle Kloster, Peter Muller, Luigi Patruno, Seena Vali et Sean Woodward. Ed Casimiro m'a également apporté de nombreux encouragements, et je lui en suis très reconnaissant.

Après quatre ans passés à Fordham, j'ai eu la chance d'être embauché à l'Université du Wisconsin-Stout, l'école polytechnique du système universitaire du Wisconsin. Ce fut une véritable aubaine pour moi, car j'ai obtenu ma licence en ingénierie informatique et systèmes à l'Institut polytechnique Rens-selaer en 1999, et mon père, de son côté, a obtenu sa licence en génie mécanique à l'ETH de Zurich (l'école polytechnique suisse) en 1951. Par conséquent, retrouver l'atmosphère d'une école polytechnique était un peu comme rentrer chez soi. Si les trois institutions sont loin d'être identiques, le noble objectif de préparer des étudiants assidus à devenir ingénieurs, scientifiques et managers, ou à exercer d'autres professions techniques lucratives, incite à privilégier le pragmatisme, le travail acharné et l'exactitude. Cette « attitude polytechnique » se reflète à tous les niveaux académiques, du chancelier aux étudiants de première année. Elle nourrit le respect des mathématiques en général, et du calcul différentiel et intégral en particulier. À l'UW Stout, les mathématiques sont perçues comme une passerelle vers des cours importants liés à la carrière, et non comme un obstacle à l'obtention du diplôme, comme c'était souvent le cas à Fordham, tant par les étudiants que par les hauts fonctionnaires.

La majeure partie de ce livre a été écrite alors que je travaillais à l'Université de Washington Stout. Les doyens Jeffery Anderson et Charles Bomar ont tous deux apporté un soutien administratif, financier et émotionnel à ce projet. Le degré de coopération au sein de notre département de mathématiques, de statistiques et d'informatique est extrêmement rare et réconfortant. Sauf erreur grave, chaque membre du département a contribué à ce livre par une suggestion, une correction, un commentaire, en me remplaçant en cours lors d'une conférence ou en m'encourageant. Enseigner 24 crédits par an dans un climat totalement hostile à l'habitation humaine, tout en

---

1Ce n'est pas une exagération. Deux exemples suffiront. Le premier jour de cours en 2014, soit le 28 janvier, alors que je quittais la maison pour me rendre sur le campus, weather.com indiquait une température de -22 °F au thermomètre et de -41 °F au thermomètre.

Machine Translated by Google

C'est en essayant de maintenir une part de notre vie de chercheur que nous nous sommes réunis. Avec le plus grand respect, j'inscris ici leurs noms : Anne Antonippillai, Wan Bae, Alexander Basyrov, le président Christopher Ben-del, Seth Berrier, Jeffrey Boerner, Diane Christie, Steven Deckelman, Brent Dingle, Seth Dutter, Jeanne Foley, Eugen « Andre » Ghenciu, Petre « Nelu ».

Ghenciu, Nasser Hadidi, Matt Horak, Ayub Hossain, Amitava Karmaker, Deborah Kruschwitz-List, Terrence Mason, Valentina « Diana » Postelnicu, Laura Schmidt, Loretta Thielman, Peter Thielman, Keith Wojciechowski et Mingshen Wu. De plus, les anciens professeurs Joy Becker, John Hunt, Benjamin Jones, Dennis Mikkelson et Eileen Zito ont fourni des conseils, des encouragements ou du matériel de cours. Par ailleurs, plusieurs de nos intérimaires ont volontairement accepté des cours extrêmement désagréables ou les obligations élevées d'un professeur à temps plein, sans compensation financière supplémentaire, et méritent donc particulièrement d'être félicités : Jim Church, Brian Knaeble, Shing Lee, Olga Lopukhova, John Neiderhauser, Mark Pedersen et Dennis Schmidt. Le président, Chris Bendel, est particulièrement dévoué et un artisan de paix exceptionnel qui a créé un environnement de travail de collaboration, de coopération et de respect mutuel souvent rêvé, mais presque jamais trouvé, dans la vie universitaire.

Au MSCS, nous travaillons pour nous entraider, contrairement à un autre département de notre établissement que je ne nommerai pas explicitement, où les professeurs seniors ont fait un passe-temps de briser les carrières et de briser les rêves de leurs propres professeurs juniors.

En dehors de mon département à l'UW Stout, Alan Block, Mark Fenton, Julie Furst-Bowe, Jennifer Grant, Mary Hopkins-Best, Michael Levy, Steve Nold, Marlann Patterson et Todd Zimmermann m'ont apporté un soutien précieux. Je tiens également à remercier l'ensemble du département de biologie, qui a établi un niveau très élevé de recherche, d'enseignement interactif, de rédaction de demandes de subvention compétente, de cours innovants et axés sur le travail en laboratoire, et qui nous a tous appris à impliquer activement et concrètement les étudiants de premier cycle dans la recherche.

Il serait habituel de citer ici les étudiants de l'UW Stout qui ont enrichi mon enseignement et m'ont apporté des commentaires utiles. Une telle liste alourdirait cependant considérablement le livre. J'ai eu beaucoup de chance de bénéficier de cours très appliqués, dont le contenu s'appliquait parfaitement à la pratique professionnelle, tant en ingénierie qu'en gestion. Vous le constaterez dans le corps de cet ouvrage, car de nombreux exemples tirés de la physique et de l'économie correspondent aux aspirations professionnelles de mes étudiants. La grande majorité d'entre eux m'ont, à un moment ou à un autre, fait une suggestion, une correction ou m'ont montré une nouvelle application des mathématiques à leurs domaines d'études. J'ai étudié et enseigné dans d'autres établissements, mais ce phénomène est un phénomène que je n'ai rencontré qu'à

Température ressentie. Pour mes lecteurs métriques, cela correspond à −29,4 °C au thermomètre et à −40,6 °C avec la température ressentie. Les 2 et 3 mai 2013, nous avons eu 43,2 cm de neige.

Machine Translated by Google

UW Stout et à Oxford au Royaume-Uni lorsque j'étais étudiant invité là-bas. Comme je ne peux pas citer tous les étudiants par leur nom, je les remercierai en numéro de cours : Math-123, Mathématiques finies et financières ; Math-154, Calcul II; Math-380/580, Cryptographie; Math-447/747, Analyse numérique II; et CS-480/680, Sécurité informatique.

En plus du corps professoral des universités où j'ai enseigné, la faculté de l'Université Columbia et de la City University of New York (CUNY), deux des meilleures institutions de « la ville » et du pays, ont donné Cela m'a été d'une grande aide. J'ai apprécié les échanges sur la recherche et l'enseignement avec David Allen, David Bayer, Maria Chudnovsky, Kenneth Kramer, Hunter Johnson et Vladimir Shpilrain.

En physique et en chimie, j'ai eu l'immense avantage de consulter des experts. Pour la physique, c'était Gabriel Hannah, et pour la chimie, c'était C'était Gergely Sirokman. Ils ont soigneusement vérifié les sections concernées, et par conséquent, le lecteur peut être sûr que ces sections sont exemptes de erreurs, à une exception près. Dans les problèmes de physique, je n'ai pas effectué les unités à travers les formules et dans les équations, comme l'exige ce En mathématiques appliquées, nous ne le faisons généralement pas, et cela cela serait très désorientant et fastidieux pour moi de modifier ce qui est déjà terminé livre pour correspondre à la convention des physiciens.

J'ai reçu beaucoup d'encouragements dans ce projet, notamment suggestions détaillées, parmi celles énumérées ci-dessus, ainsi que celles de Martin Brock, Bruce Cohen, Carmi Gressel, Marshall Hampton, Benjamin Jones, David Joyner, Craig Larson, Clément Pernet et Susan Schmoyer.

Mon ancien directeur de thèse et ami de confiance Lawrence Washington m'a inspiré avec ses manuels extrêmement réussis. Il a établi un très La barre est haute pour nous tous. Je recommande souvent la cryptographie. avec Coding Theory, par Wade Trappe et Lawrence Washington, à tous qui veut apprendre la cryptographie, du premier cycle à la deuxième année le membre principal du corps professoral.

Le financement de Sage provient en grande partie de la National Science Foundation, et plus particulièrement des divisions listées ci-dessous. La communauté Sage est reconnaissant pour le soutien des subventions de la National Science Foundation :

- DMS-0545904, Division des sciences mathématiques, « CARRIÈRE : Méthodes cohomologiques en géométrie algébrique et en théorie des nombres »,
- DMS-0555776, Division des sciences mathématiques, « Approches explicites des formes modulaires et des variétés abéliennes modulaires »,
- DMS-0713225, Division des sciences mathématiques, « SAGE : Logiciel-logiciel pour l'expérimentation en algèbre et en géométrie »,
- DMS-0757627, Division des sciences mathématiques, « FRG : fonctions L et formes modulaires »,
- DMS-0821725, Division des sciences mathématiques, « SCREMS : Le Frontières computationnelles de la théorie des nombres, de la théorie des représentations et de la physique mathématique »,

Machine Translated by Google

- DMS-0838212, Division des sciences mathématiques, « EMSW21–RTG : Groupe de formation à la recherche sur les problèmes inverses et les équations aux dérivées partielles »,
- DMS-1015114, Division des sciences mathématiques, « Sage : Unifier Logiciels mathématiques pour scientifiques, ingénieurs et mathématiciens »,
- DUE-1022574, Division de l'enseignement de premier cycle, « Recherche collaborative : UTMOST : Enseignement de premier cycle en mathématiques avec des logiciels et des manuels libres »,
- ACI-1147463, Division de cyberinfrastructure avancée, « Recherche collaborative : SI2-SSE : Sage-Combinat : Développement et partage Logiciel Open Source pour la combinatoire algébrique »,

ainsi que des financements de Microsoft, Sun Microsystems, Enthought Inc. et Google.

Comme je l'ai mentionné précédemment, la communauté Sage est immense, avec des centaines de développeurs répartis dans de nombreux pays et sur plusieurs continents. Malgré le soutien financier mentionné ci-dessus, la quasi-totalité du travail est réalisée sur une base entièrement bénévole et non rémunérée. La communauté Sage tout entière est extrêmement reconnaissante du temps libre que tant de personnes y consacrent.

Machine Translated by Google

Chapitre 1

Bienvenue chez Sage !

Les mathématiques sont une discipline que l'on pratique, pas une discipline que l'on regarde. Par conséquent, la meilleure façon d'apprendre, quelle que soit leur branche, est de se lancer et d'essayer. Il en va de même pour l'apprentissage de Sage ! Connectez-vous simplement au serveur cellulaire Sage et lancez-vous sans plus attendre en essayant les exemples des pages suivantes. https://

sagecell.sagemath.org/

1.1. Utiliser Sage comme calculatrice

Tout d'abord, vous pouvez toujours utiliser Sage comme une simple calculatrice. Par exemple, si vous tapez 2+3 et cliquez sur « Évaluer », vous obtiendrez la réponse 5.

Les expressions peuvent être aussi complexes que vous le souhaitez. Pour calculer le montant d'un prêt simple à 6 % par an sur 90 jours et un capital de 900 $, nous connaissons tous la formule suivante : A = P(1 + rt). Il suffit donc de saisir

900*(1+0,06*(90/365))

et cliquez sur « Évaluer ». Vous apprendrez que la réponse est 913,315068493151, soit 913,32 $ après arrondi au centime le plus proche.

Notez que le symbole de l'addition est le signe plus et celui de la soustraction, le trait d'union. Le symbole de la multiplication est l'astérisque et celui de la division, la barre oblique (celle qui accompagne le point d'interrogation, et non la barre oblique inverse). Ces règles sont identiques à celles utilisées par MS-Excel et de nombreux langages informatiques, notamment C, C++, Perl, Python et Java.

Pour calculer des intérêts composés, il faudrait utiliser des exposants. Le symbole des exposants est le caret, situé au-dessus du chiffre six sur la plupart des claviers. Il ressemble à ceci : « ˆ ».

Prenons l'exemple d'un prêt personnel à taux composé annuel de 12 % sur 3 ans, avec un capital de 11 000 $. La formule est la suivante : A = P(1 + i) n ; il suffit donc de saisir ,

11000*(1+0,12)^3

Machine Translated by Google

et cliquez sur « Évaluer ». Vous apprendrez que la réponse est 15 454,2080000000, soit 15 454,21 $ après arrondi au centime le plus proche. Au fait, au lieu de cliquer sur « Évaluer », appuyez simplement sur Maj+Entrée sur votre clavier, et le résultat sera identique.

Attention : Il est très important de ne pas saisir de virgule en cas de nombres trop grands avec Sage. Il faut saisir 11 000 et non 11 000.

Pour ceux qui n'aiment pas le signe d'insertion, vous pouvez utiliser deux astérisques d'affilée.

```
11000*(1+0,12)**3
```

qui est en fait un retour au langage de programmation historique FORTRAN qui était le plus populaire1 dans les années 1970 et 1980.

Et si vous faites une erreur ? Imaginons que je voulais dire 13 000 $ et non 11 000 $. Cliquez sur l'erreur, corrigez-la avec le clavier et remplacez 11 par 13. Cliquez à nouveau sur « Évaluer » et tout rentrera dans l'ordre.

Symboles de regroupement :

Lorsqu'une formule comporte plusieurs ensembles de parenthèses, les mathématiciens utilisent parfois des crochets comme une sorte de « super parenthèses ». Or, Sage a besoin des crochets pour d'autres choses, comme les listes. Il faut donc toujours utiliser des parenthèses pour regrouper des éléments dans les formules.

Par exemple, disons que vous devez évaluer

$$550 \frac{1 - (1 + 0{,}05)^{-30}}{0{,}05}$$

Vous ne devriez donc pas taper

```
550 [ 1 - (1+0,05)^(-30) ]/0,05
```

mais plutôt

```
550 ( 1 - (1+0,05)^(-30) )/0,05
```

où les crochets sont devenus des parenthèses.

Certains livres de mathématiques très anciens utilisent les accolades { et } comme auxiliaires, ainsi que les parenthèses et les crochets. Ces derniers, s'ils servent à grouper dans une formule, doivent également devenir des parenthèses. Il s'avère que Sage, comme les livres de mathématiques modernes, utilise les accolades { et } pour désigner des ensembles.

Au fait, la formule ci-dessus n'était pas artificielle. Il s'agit de la valeur d'un prêt à 5 % composé annuellement sur 30 ans, avec un versement annuel de 550 $.
La formule s'appelle « la valeur actuelle d'une rente ».

1Une parenthèse historique sans rapport : comme la réécriture de logiciels est un processus pénible et long, de nombreux logiciels scientifiques importants sont restés écrits en FORTRAN jusqu'à il y a quelques années, et il en va de même pour les logiciels d'entreprise en COBOL.
Chaque nouveau langage de programmation intègre des caractéristiques de la génération précédente de langages, pour faciliter son apprentissage, et c'est ainsi que l'on voit de temps en temps des graines vraiment anciennes de langages morts.

Machine Translated by Google

Trois erreurs qui représentent plus de 90 % de mes erreurs dans Sage : Je fais souvent trois erreurs avec Sage. Par exemple, pour écrire 11 000x + 1 200, il faut taper

11000*x+1200

La première erreur que je commets parfois est d'omettre l'astérisque entre 11 000 et x. Dans Sage, il est obligatoire d'inclure cet astérisque, car c'est le symbole de la multiplication. La deuxième erreur est que j'ajoute souvent une virgule à l'intérieur de 11 000, mais il n'est pas acceptable dans Sage d'écrire 11 000 à la place de 11 000. La troisième erreur est que je vais avoir des parenthèses mal assorties. Toute expression Sage doit avoir le même nombre de (s) que de s, ni plus ni moins.

Une liste plus complète est donnée à l'annexe A, « Que faire en cas de frustration », à la page 293.

1.2. Utilisation de Sage avec les fonctions courantes

Je vais maintenant expliquer comment Sage fonctionne avec les racines carrées, les logarithmes, les exponentielles, etc.

Racines carrées : Les fonctions standard du lycée sont également intégrées. Par exemple, tapez

sqrt(144)

Cliquez ensuite sur « Évaluer » et vous apprendrez que la réponse est 12. À partir de maintenant, je ne dirai plus « Cliquer sur Évaluer », car répéter cela pourrait être fastidieux ; je supposerai que vous savez que vous devez le faire. Puisque Sage privilégie les réponses exactes, essayez sqrt(8)

et obtenez 2*sqrt(2) comme réponse. Si vous avez vraiment besoin d'une décimale, essayez plutôt N(sqrt(8))

et obtenez

2,82842712475. qui est une approximation décimale.

La fonction N(), également écrite n(), convertit le nombre entre parenthèses en nombre réel, si possible. Il s'agit généralement d'un développement décimal ; à moins que le nombre entre parenthèses ne soit un entier 2, il s'agira nécessairement d'une approximation. Sage supposera que toutes les décimales sont de simples approximations, par exemple

---

2Pour être mathématiquement correct, je devrais dire « un entier ou une fraction dont le dénominateur est inscriptible comme le produit d'une puissance de 5 et d'une puissance de 2 ». Par exemple, les 25e, 16e et 80e peuvent être écrits exactement avec des décimales, alors que les 3e, 15e et 14e ne le peuvent pas. Notez que 25 = 52 et 16 = 24 , ainsi que 80 = 24 × 5. Ces dénominateurs n'ont que des nombres premiers 2 et 5 dans leur factorisation. 3 = 3 et 15 = 5 × 3, tandis que 14 = 7 × 2. Comme vous pouvez le constater, ces dénominateurs ont d'autres nombres premiers que 2 et 5 dans leur factorisation. Si cela vous intéresse, vous devriez lire « Utiliser Sage pour travailler avec des entiers » à la page 145.

Machine Translated by Google

sqrt(3.4) sera
évalué à 1,84390889145858, sans avoir besoin d'utiliser N().

Racines d'ordre supérieur : Les

racines d'ordre supérieur peuvent être calculées comme les exposants. Par exemple, pour trouver la racine sixième de 64, utilisez 64^(1/6).

et obtenir que la réponse est 2.

Les constantes spéciales π et e : En

mathématiques, nous avons souvent besoin des constantes spéciales π et e. Il s'avère que π est intégré dans Sage sous la forme « pi » (les deux lettres sont en minuscules) et e est intégré sous la forme « e » (encore une fois, en minuscule).

Vous pouvez utiliser des

commandes comme « digital_approx(pi,

prec=200) » pour obtenir plusieurs chiffres de pi,

ou « digital_approx(sqrt(2), prec=200) » pour obtenir un

développement précis de $\sqrt{2}$. Ces développements sont effectués sur 200 bits, et pour 100 chiffres, vous pouvez utiliser 332 bits ; 200 chiffres3 correspondent à 664 bits, et ainsi de suite. Vous pouvez également utiliser l'abréviation suivante :

N(sqrt(2), prec=200)

Comme nous l'avons vu précédemment. C'est l'abréviation de la fonction N() ou n() (identique), à savoir « approximation numérique ».

Saisie semi-automatique :

Il est temps d'expliquer une fonctionnalité intéressante de Sage. Si vous saisissez une commande longue, comme « approx numérique », et qu'en cours de route vous oubliez la fin exacte (est-ce approximation ? approximations ? appr ?), vous pouvez appuyer sur la touche Tabulation. Si une seule commande de la bibliothèque possède ce préfixe, Sage complètera le reste. Sinon, une liste de suggestions s'affiche. C'est une fonctionnalité très utile lorsque vous avez oublié les commandes exactes. Nous verrons d'autres exemples de fonctionnalités d'aide intégrées à la section 1.10, page 47.

Il existe un autre raccourci clavier que j'apprécie. Comme je l'ai déjà mentionné, si vous en avez assez de cliquer sans cesse sur « Évaluer », appuyez simplement sur Maj et Entrée simultanément. Cela revient à cliquer sur « Évaluer ».

Sage est-il sensible à la casse ?

Vous avez probablement déjà eu une situation dans la vie où vous avez entré votre mot de passe correctement, mais vous avez découvert qu'il avait été rejeté parce que vous aviez le mauvais mot de passe.

3Cela s'avère être dû au fait que 10200 ≈ 2 664, si vous êtes curieux.

Machine Translated by Google

Majuscules. Par exemple, vous avez peut-être laissé la touche « VERR. MAJ » activée. Quoi qu'il en soit, Sage considère que Pi est différent de pi et Sin est différent de sin.

Les quatre seules exceptions que je connaisse sont i et n(), ainsi que Vrai et Faux. Nous n'utiliserons pas Vrai et Faux de manière approfondie avant le chapitre 5, mais il est inutile de préciser que vous pouvez également les saisir comme vrai et faux, et que Sage les reconnaît comme identiques. Vous en avez peut-être entendu parler.

$\sqrt{-1}$ est souvent appelé « le nombre imaginaire » et est noté i.

Ensuite, vous pouvez représenter $\sqrt{-1}$ par i ou I, et dans les deux cas, Sage comprendra ce que vous vouliez dire, à savoir $\sqrt{-1}$. Si vous n'avez jamais entendu parler des nombres imaginaires ou de i, vous pouvez ignorer ce fait. Enfin, vous pouvez écrire n(sqrt(2)) ou N(sqrt(2)) ; Sage les traitera comme identiques.

Autant que je sache, ces servitudes ne s'appliquent qu'à $\sqrt{-1}$ et n(), ainsi qu'à True et False comme il s'avère ; dans tous les autres cas, la majuscule est importante.

Exponentielles :

Tout aussi

2^3

vous donne 23 et de même

3^3

vous donne 33 , si tu veux dire e^3 alors dis juste

et^3

et c'est très bien. Ou pour une approximation décimale, on peut faire

N(e^3)

Il convient également de mentionner que parfois les livres écrivent

exp(5 · 11 + 8) au lieu de e, et $^{5\cdot 11+8}$

Sage trouve cela correct. Par exemple, exp(5*11+8) -

e^(5*11+8) donne 0. N'oubliez pas

l'astérisque entre le 5 et le 11.

Logarithmes :

Bien sûr, Sage connaît les logarithmes, mais il y a un problème. Il existe plusieurs types de logarithmes, notamment le logarithme décimal, le logarithme népérien, le logarithme binaire et le logarithme dans toute autre base que vous souhaitez utiliser. Au lycée, « log » désigne le logarithme décimal et « ln » le logarithme népérien. Cependant, après le calcul différentiel et intégral, « log » désigne le logarithme népérien. Sage étant principalement destiné aux étudiants de niveau universitaire et supérieur, il est tout naturel qu'ils aient choisi d'utiliser le logarithme népérien pour « log ».

Donc pour trouver le logarithme naturel de 100, tapez

N(log(100))

Machine Translated by Google

pour le type logarithme décimal de 100

N(log(100,10))

et pour le logarithme binaire de type 100

N(log(100,2))

Ce qui, bien sûr, se généralise. Par exemple, pour trouver le logarithme de 100 en base 42, tapez

N(log(100,42))

Notez que Sage est assez efficace pour obtenir des réponses exactes. Essayez

log (sqrt(100^3), 10) et vous

obtiendrez 3, la réponse exacte.

Exemple financier : Imaginez

que vous déposez 5 000 $ sur un compte rémunéré à 4,5 % de capitalisation mensuelle. Vous vous demandez peut-être quand votre total atteindra 7 000 $. En utilisant la formule A = P(1 + i) n , où P est le capital, A le montant final, i le taux d'intérêt mensuel et n le nombre de périodes de capitalisation (nombre de mois), nous obtenons

A = P(1 + i) n 7000

= 5000(1 + 0,045/12)n 7000/5000

= 5000(1 + 0,045/12)n 1,4 = (1 + 0,045/12)n

log 1,4 = log(1 + 0,045/12)n

log 1,4 = n log(1 + 0,045/12) log 1,4

log(1 + 0,045/12)

_____ = n

Et donc, nous tapons dans Sage

log(1,4)/log(1+0,045/12)

et nous obtenons la réponse 89.8940609330801, ce qui nous permet de savoir que 90 mois seront nécessaires. Nous écrivons donc la réponse 90 mois, ou mieux encore, 7 ans et 6 mois.

Voici un exemple de résolution de problème assistée par ordinateur. Outre cette approche, Sage est également disposé à résoudre le problème du début à la fin, sans intervention humaine. Nous verrons cela à la page 46.

Mathématiques pures et racines carrées : Voici

quelques notes sur les racines carrées qui intéresseront les étudiants en mathématiques ou toute personne souhaitant travailler avec les nombres complexes. La plupart des étudiants préféreront ignorer cette sous-section, ainsi que la suivante sur les nombres complexes, et passer à « Utiliser Sage pour la trigonométrie » (page 7) ou à « Utiliser Sage pour la représentation graphique en 2 dimensions » (page 9).

Machine Translated by Google

Dans un esprit de pompe absolue, vous savez peut-être que (−2)2 = 4 ainsi que 22 = 4. Donc, si vous voulez toutes les racines carrées d'un nombre, vous pouvez faire sqrt(4,all=True) pour obtenir

[2, -2]

Les crochets indiquent une liste. Toute séquence de nombres séparés par des virgules et entre crochets est une liste. Nous verrons d'autres exemples de listes tout au long du livre.

Un avant-goût des nombres complexes :

Si vous connaissez les nombres complexes, vous pouvez faire sqrt(-4,all=True) pour obtenir

[2*I, -2*I]

où la lettre I majuscule représente $\sqrt{-1}$, la constante imaginaire. Comme mentionné précédemment, vous pouvez utiliser le i minuscule à la place du i majuscule, si vous préférez.

Bien que Sage soit performant en analyse complexe et puisse produire de jolis tracés de fonctions à variables complexes, nous n'entrerons pas dans les détails ici. Les tracés en couleur de fonctions à valeurs complexes seront traités dans l'annexe en ligne de ce livre, « Traçage en couleur, en 3D et animations », téléchargeable sur ma page web www.gregorybard.com.

1.3. Utilisation de Sage pour la trigonométrie

Certains étudiants ne connaissent pas la trigonométrie ou suivent un cours qui n'en traite pas. Dans ce cas, ils peuvent ignorer cette section et passer à la suivante, qui commence à la page 9.

Les commandes trigonométriques de Sage sont très intuitives, mais il est important de se rappeler que Sage fonctionne en radians. Ainsi, pour connaître le sinus de π/3, saisissez sin(pi/3).

Vous obtiendrez la réponse 1/2*sqrt(3). Il s'agit d'une réponse exacte, et non d'une simple approximation décimale. Vous constaterez que Sage privilégie les réponses exactes aux approximations. C'est parfois irritant, car si vous demandez le cosinus de π/12, vous taperez cos(pi/12).

et obtenez 1/12*(sqrt(3) + 3)*sqrt(6), ce qui est particulièrement insatisfaisant si vous souhaitez une décimale. En revanche, si vous tapez N(cos(pi/12)), vous obtiendrez 0,965925826289, une assez bonne approximation décimale.

Machine Translated by Google

Vous découvrirez que Sage est assez avisé lorsqu'il s'agit de savoir lorsque les fonctions échouent. En particulier, essayez d'évaluer la tangente à une de ses asymptotes. Par exemple,

bronzage(pi/2)

produira la réponse utile « Infini ». Les fonctions trigonométriques « rares » ou « réciproques » de cotangente, sécante et cosécante, qui sont importantes en calcul, mais gênants sur les calculatrices portables, sont intégrés à Sage. sont identifiés comme cot, sec et csc.

Les fonctions trigonométriques inverses sont également disponibles. Elles sont utilisées tout comme les fonctions trigonométriques. Par exemple, si vous tapez

arc sinus (1/2)

vous obtiendrez

1/6*pi

comme prévu. De même

arccos(1/2)

produit

1/3*pi

Les abréviations usuelles sont toutes connues et utilisées par Sage. En voici un exemple. liste complète :

Notation mathématique :	Commande longue :	Commande courte :
sin−1 x	arcsin(x)	et(x)
cos−1 x	arccos(x)	acos(x)
tan−1 x	arctan(x)	en attente(x)
cot−1 x	arccot(x)	acot(x)
sec−1 x	arcsec(x)	asec(x)
csc−1 x	arccsc(x)	acsc(x)

Vous pouvez également utiliser Sage pour représenter graphiquement les fonctions trigonométriques. Nous allons le faire. dans la section intitulée « Utilisation de Sage pour créer un graphique en 2 dimensions », à la page 11.

Conversion entre degrés et radians :

Nous nous souvenons tous du cours de trigonométrie selon lequel pour convertir les radians en degrés, il suffit de multiplier par 180/π, et pour convertir des degrés en radians, il suffit de multiplier par π/180. Voici donc ce qu'il faut saisir pour convertir π/3 en degrés :

(pi/3) * (180/pi)

produit 60 tandis que

60 * (pi/180)

produit 1/3*pi.

La façon dont j'aime me souvenir de cette technique est qu'un rapporteur mesure 180 degrés, et le mot « rapporteur » commence par « p », tandis que π est le grec « p. » Par exemple, 90 degrés correspondent à la moitié d'un rapporteur et 60 degrés à un tiers d'un rapporteur. C'est pourquoi je me souviens qu'il s'agit de π/2 et π/3.

Machine Translated by Google

De même, si je vois π/6, alors je sais que c'est un sixième d'un rapporteur ou 30 degrés.

1.4. Utilisation de Sage pour créer un graphique bidimensionnel

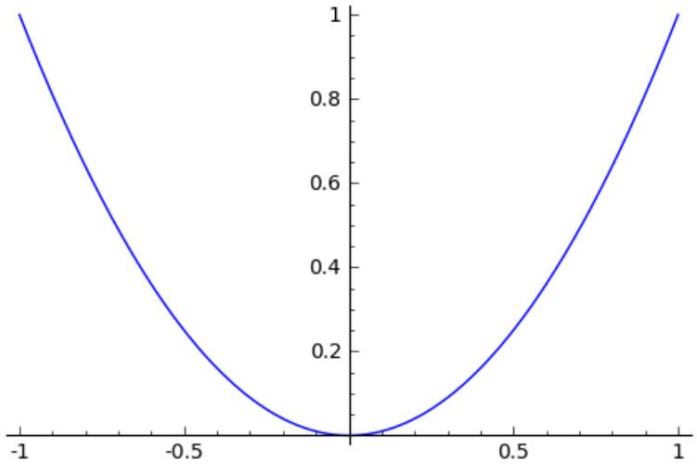
Ma forme préférée est la parabole, alors commençons par là. Tapez plot(x^2)

et vous

obtiendrez un joli tracé d'une parabole dans l'intervalle

$$-1 < x < 1$$

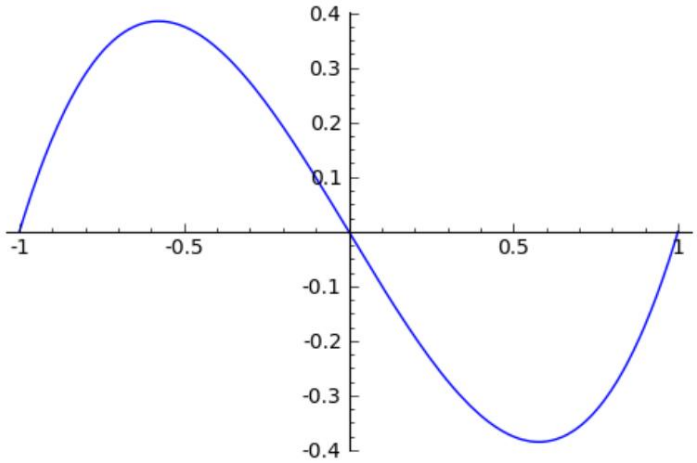
Il s'agit de la plage par défaut de la commande plot. Elle limite les valeurs y à la valeur nécessaire pour afficher les valeurs x. Voici une capture d'écran de ce que vous devriez voir :



De même, vous pouvez

faire plot(x^3-x)

qui est agréable et visuellement attrayant, comme vous pouvez le voir :

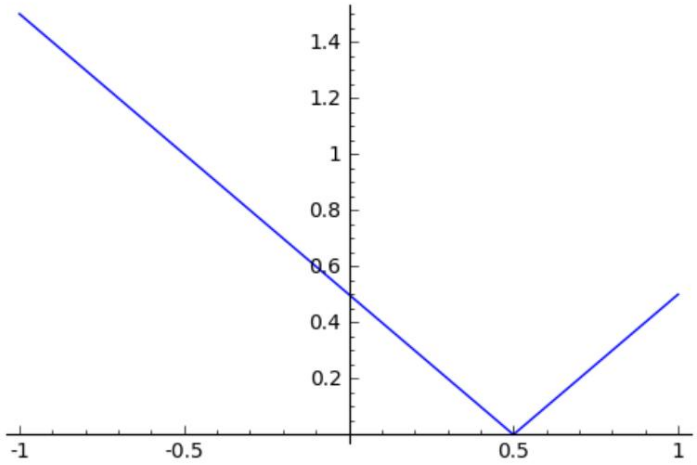


Machine Translated by Google

Pour |x − 1/2| vous pouvez faire

plot(abs(x-1/2)) qui

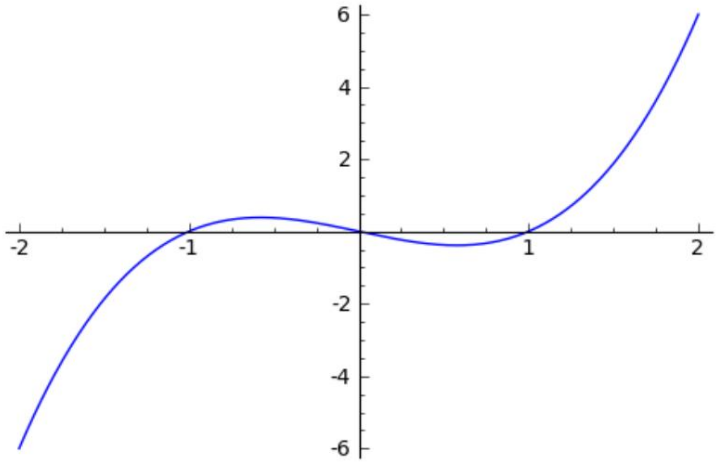
produit la capture d'écran suivante



Et si vous souhaitiez une plage x différente ? Par exemple, pour représenter graphiquement −2 < x < 2, tapez plot(x^3-x, -2, 2)

et vous obtiendrez le

graphique souhaité, à savoir :

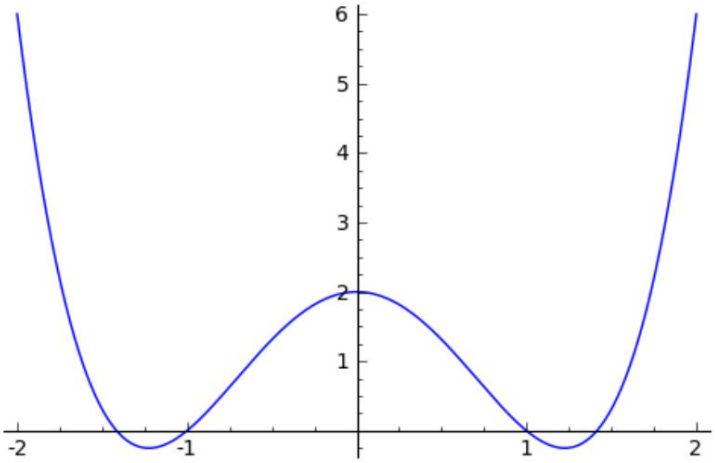


Il est temps de préciser que vous pouvez enregistrer ces tracés dans un fichier après les avoir générés par Sage. Il vous suffit de faire un clic droit sur le graphique affiché dans votre navigateur web et d'enregistrer le fichier sur votre ordinateur.
Vous pouvez ensuite importer l'image dans n'importe quel rapport ou article que vous rédigez. Un graphique très intéressant est :

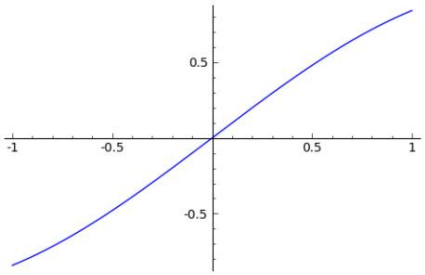
plot(x^4 - 3*x^2+2,-2,2).

mais remarquez l'astérisque entre 3 et x si vous « 3*x ”. Vous obtiendrez une erreur si l'omettez ! L'intrigue est

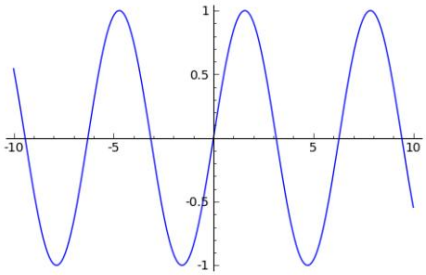
Machine Translated by Google



Un autre petit point de friction est que pour y = sin(t) vous devez dire
plot(sin(x)) ou
mieux encore
plot(sin(x),-10,10)
car l'intrigue n'attend pas t, elle attend x. Les images que vous obtenez sont



intrigue(sin(x))



plot(sin(x),-10,10)

En fait, si vous deviez taper
plot(sin(t)) vous
verriez

```
NameError : le nom « t » n'est pas défini
```

car dans ce cas, Sage ne sait pas ce que signifie « t ». Une autre solution est proposée à la page 104.

1.4.1. Contrôle de la fenêtre d'affichage d'un graphique. La

plupart du temps, la fenêtre d'affichage par défaut d'un graphique correspond exactement à ce que vous souhaitez, mais pas toujours. L'exception la plus courante est une fonction avec des asymptotes verticales. Nous allons voir ici comment ajuster la fenêtre d'affichage.

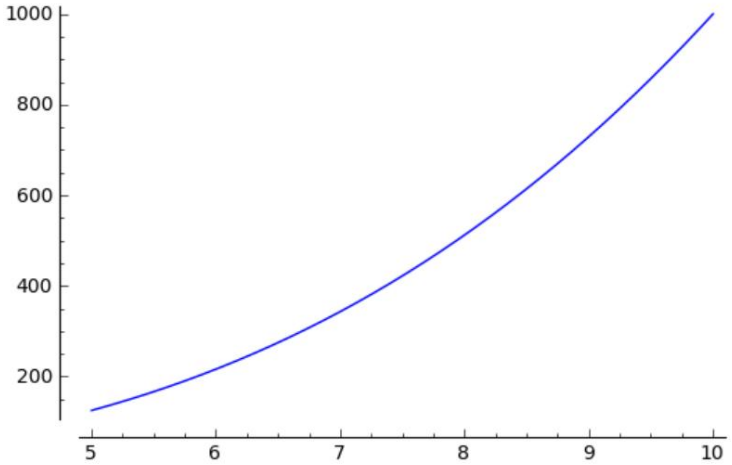
Machine Translated by Google

Sortir de l'échelle :

Considérons le tracé de x^3 de x = 5 à x = 10, qui est donné par

```
plot(x^3, 5, 10)
```

Comme vous pouvez l'imaginer, la fonction va de 53 = 125 à 103 = 1000 ; l'origine devrait donc apparaître très loin sous le graphique. Voici le tracé :

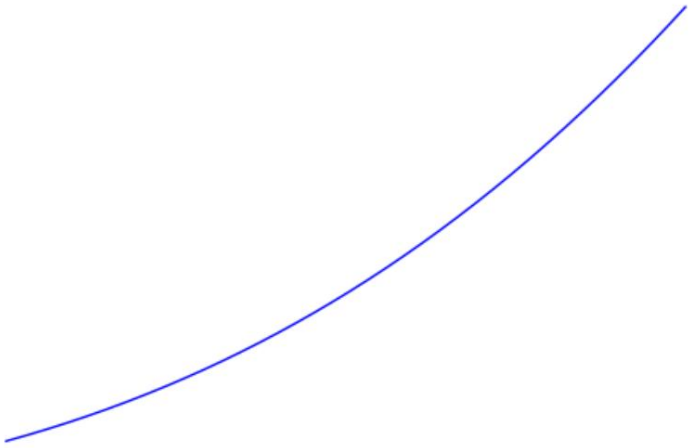


Votre indice selon lequel l'axe des x n'est pas à son emplacement normal à l'écran signifie que les axes ne se coupent pas. Cela signifie que l'origine est éloignée. Lorsque les axes se coupent à l'écran, l'origine se trouve (en réalité et à l'écran) à leur intersection.

De plus, si vous le souhaitez, vous pouvez masquer les axes avec

```
plot(x^3, 5, 10, axes=Faux)
```

et cela produit



ce qui est considérablement moins informatif.

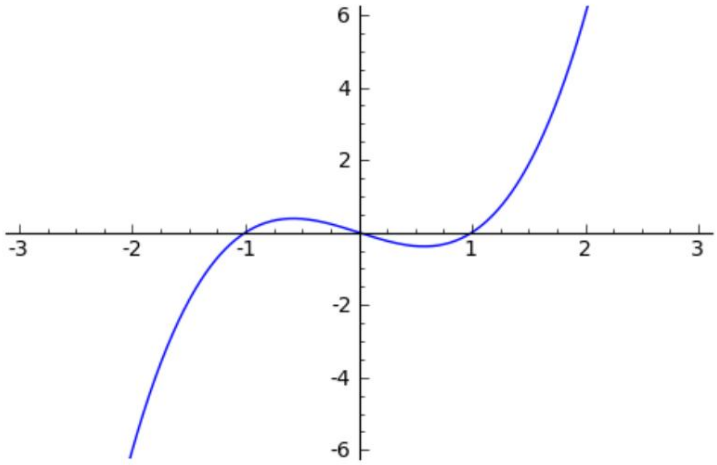
Machine Translated by Google

Pour forcer l'étendue Y d'un graphique : si, pour une

raison quelconque, vous souhaitez contraindre l'étendue Y d'un graphique à être limitée entre deux valeurs, vous pouvez le faire. Par exemple, pour maintenir −6 < y < 6, nous pouvons utiliser

tracé (x^3-x, -3, 3, ymin = -6, ymax = 6)

afin d'obtenir



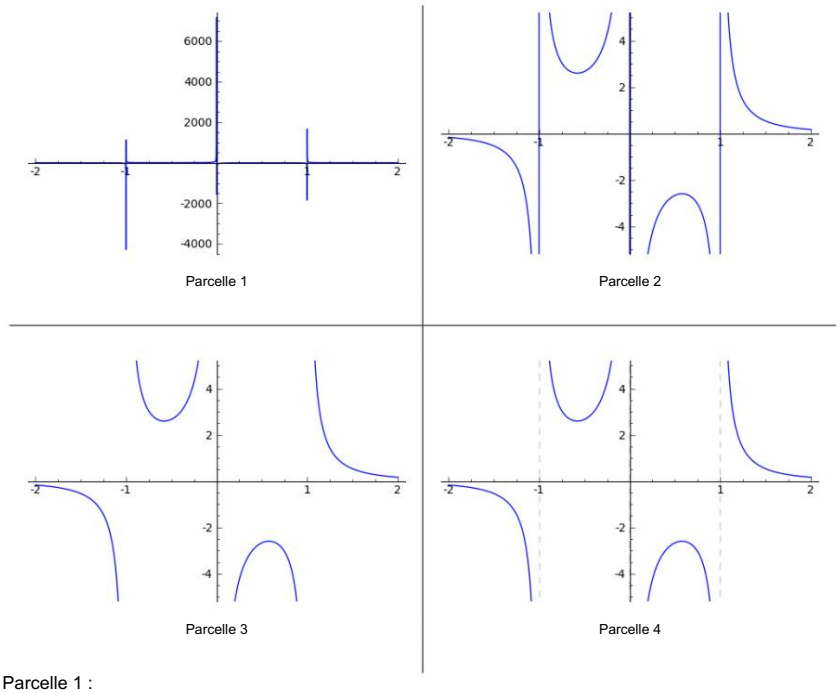
Ceci est important, car Sage souhaite généralement afficher l'intégralité du graphique. Par conséquent, il s'assurera que l'axe des Y est suffisamment grand pour inclure chaque point, du maximum au minimum. Pour certaines fonctions, le maximum, le minimum, ou les deux, peuvent être très grands.

Tracés de fonctions avec asymptotes : Sage

(et tous les outils de calcul formel) calcule le tracé d'une fonction en générant un très grand nombre de points dans l'intervalle des x et en évaluant la fonction en chacun de ces points. Plus précisément, pour représenter graphiquement f(x) = 1/ x2 entre x = −4 et x = 4, l'ordinateur peut choisir 10 000 valeurs aléatoires de x comprises entre −4 et 4, trouver les valeurs de y en les insérant dans f(x), puis tracer les points aux emplacements appropriés sur le graphique.

Ainsi, si le graphique présente une asymptote verticale, la valeur sera énorme à proximité de cette asymptote. Par conséquent, lorsque vous tracez le graphique d'une fonction rationnelle, veillez à limiter les valeurs de y. Par exemple, comparez les valeurs suivantes :

Machine Translated by Google



Parcelle 1 :

```
plot(1/(x^3-x), -2, 2)
```

Parcelle 2 :

```
plot(1/(x^3-x), -2, 2, ymin = -5, ymax = 5)
```

Parcelle 3 :

```
plot(1/(x^3-x), (x,-2, 2), detect_poles=True, ymin = -5, ymax
                = 5)
```

Parcelle 4 :

```
plot(1/(x^3-x), (x,-2, 2), detect_poles='show',
                ymin = -5, ymax = 5)
```

Comme vous pouvez le constater, la première méthode est un désastre. La seconde supprime les valeurs y très élevées et très basses, mais tente constamment de relier les différents « membres » du graphique. Si vous définissez « détecter les pôles » sur « Vrai », la méthode détecte que les pièces ne sont pas connectées.

Il reste un point mineur. Avez-vous remarqué que dans les quatre graphiques ci-dessus, « show » est entre guillemets, mais que « Vrai » ne l'est pas ? C'est parce que « Vrai » et « Faux » sont si fréquents en mathématiques qu'ils sont des mots-clés intégrés dans Sage. Cependant, le « show » apparaît moins souvent et n'est donc pas intégré, et nous devons le mettre entre guillemets.

1.4.2. Superposition de plusieurs graphiques dans un même tracé

Nous allons maintenant voir comment superposer des tracés les uns sur les autres. Il s'avère que Sage considère cela comme une « addition » des tracés, à l'aide d'un signe plus.

Machine Translated by Google

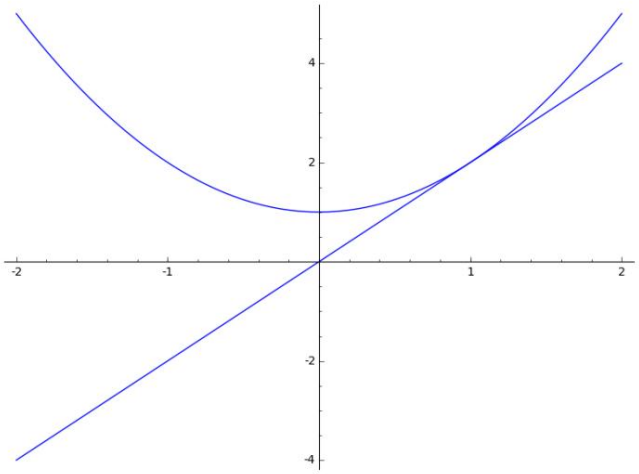
Un exemple de calcul : la tangente

Supposons que l'on veuille tracer une image de $f(x) = x^2 + 1$ sur l'intervalle $-2 < x < 2$ et la tangente à cette parabole en $x = 1$. Puisque $f(1) = 2$ et $f(1) = 2$, nous savons que la droite doit passer par le point (1, 2) et aura une pente de 2. Il n'est pas difficile de calculer que l'équation de cette droite est $y = 2x$. L'astuce est de les représenter graphiquement toutes les deux en même temps, sur la même image.

La commande pour cela sera

```
plot( 2*x, -2, 2 ) + plot( x^2+1, -2, 2 )
```

et nous obtiendrons l'image :

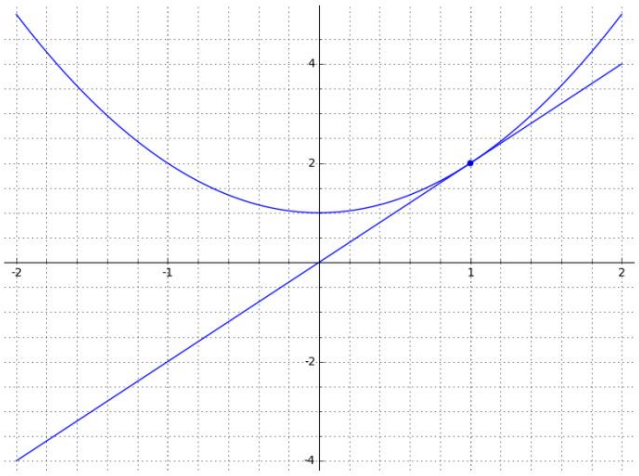


Comme vous pouvez le constater, l'addition des deux tracés crée une superposition. Le signe plus indique à Sage de tracer les deux courbes l'une sur l'autre. Nous pouvons maintenant améliorer la situation en ajoutant un point au point de tangence (le point (1, 2)) et éventuellement des lignes de quadrillage. La commande correspondante sera :

```
plot( 2*x, -2, 2, gridlines='minor' ) + plot( x^2+1, -2, 2 ) + point( (1,2), size=30 )
```

et nous obtiendrons l'image :

Machine Translated by Google



Comme vous pouvez le constater, nous avons ajouté le point à l'aide de la commande point, et celui-ci a également été superposé à l'aide du signe plus. Le paramètre gridlines='minor' nous a permis d'obtenir une grille très satisfaisante, très utile pour créer de beaux graphiques lisibles, pour des rapports de laboratoire, des cours ou des articles destinés à la publication. Nous discuterons d'autres façons de créer des lignes de quadrillage, ainsi que d'autres façons d'annoter un graphique, dans la section 3.1 à la page 91. Ce graphique spécifique apparaîtra de manière hautement annotée à la page 95.

Au fait, il est important que la commande ci-dessus, qui additionne les deux tracés et un point, soit sur la même ligne. Il est impossible d'utiliser un saut de ligne. Je ne peux pas la composer sur une seule ligne, car la page sur laquelle je tape n'est pas infinie. Assurez-vous qu'il n'y a pas de saut de ligne lorsque vous saisissez ces commandes.

Un exemple plus complexe :

Supposons que vous étudiiez les polynômes et que vous souhaitiez connaître l'effet de la variation du dernier chiffre de

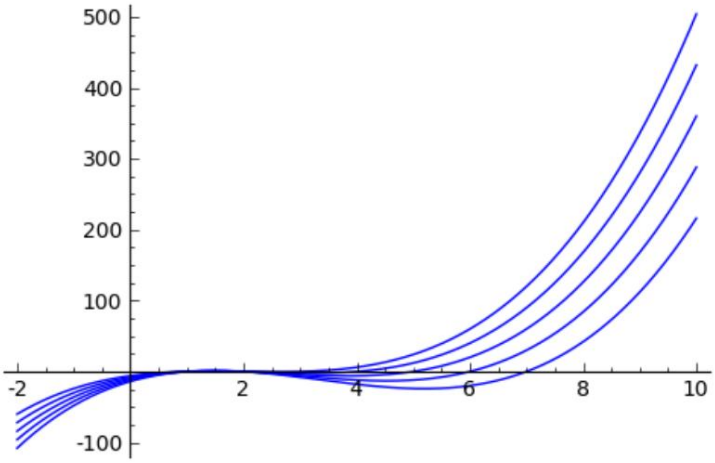
$$y = (x - 1)(x - 2)(x - 5)$$

sur son graphique. (En d'autres termes, vous souhaitez savoir ce qui se passe si vous remplacez le « 5 » de cette équation par d'autres nombres.) Vous pourriez alors envisager de considérer (x − 3) et (x − 4) comme remplaçants de (x − 5), ainsi que (x − 6) et (x − 7). Cela pourrait être fait comme

```
tracé((x-1)*(x-2)*(x-3), -2, 10) + tracé((x-1)*(x-2)*(x-4), -2, 10) + tracé((x-1)*(x-2)*(x-5),
-2,10) + tracé((x-1)*(x-2)*(x-6), -2, 10) + tracé((x-1)*(x-2)*(x-7), -2, 10)
```

J'ai choisi arbitrairement le domaine de définition, x = −2 jusqu'à x = 10. À propos, lorsque vous saisissez la commande précédente, il est important de noter qu'elle doit tenir sur une seule (très longue) ligne. Si elle est correctement exécutée, on obtient

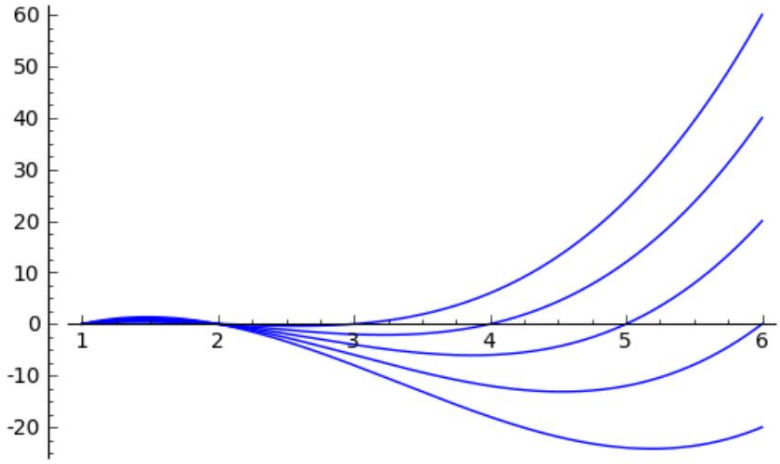
Machine Translated by Google



Il semble que je devrais zoomer un peu pour voir plus de détails. Si je voulais changer le domaine de −2 < x < 10 à 1 < x < 6, je devrais changer chaque -2 en 1 et chaque 10 en 6. Je taperais donc

tracé((x-1)*(x-2)*(x-3), 1,6) + tracé((x-1)*(x-2)*(x-4), 1, 6) + tracé((x-1)*(x-2)*(x-5), 1,6) + tracé((x-1)*(x-2)*(x-6), 1, 6) + tracé((x-1)*(x-2)*(x-7), 1,6)

et obtenir en conséquence



Il y a un problème important ici. Nous devons veiller à effectuer les modifications correctement, et cette exigence, certes complexe, est d'autant plus complexe que le code est difficile à lire. Cependant, nous apprendrons bientôt à le rendre plus lisible, ce qui facilitera également sa modification.

Machine Translated by Google

Quoi qu'il en soit, je comprends mieux maintenant la fonction de ce dernier chiffre. Il contrôle la position, ou l'abscisse, du dernier croisement de la courbe avec l'axe des abscisses. Comme vous pouvez le constater, le signe plus parmi les commandes de tracé signifie les superposer.

Il y a un point technique ici. Il est interdit de couper une ligne entre les tracés et les signes plus. Par conséquent, si vous rencontrez des difficultés pour faire fonctionner ces deux dernières commandes, vérifiez qu'elles ne contiennent pas de sauts de ligne accidentels. Pour Sage, ces commandes doivent être saisies sans saut de ligne, un peu comme la saisie d'un long paragraphe dans un traitement de texte classique : on peut insérer un retour à la ligne en fin de ligne, mais pas de saut de ligne. Cet exemple de code est fonctionnel, mais difficile à lire, à modifier et encore moins à comprendre. Nous pouvons corriger cela avec un peu d'effort. Les différentes commandes de tracé et leurs signes plus doivent être enchaînées, comme un paragraphe, passant naturellement d'une ligne à l'autre. La touche Entrée n'est pas nécessaire pour placer chaque tracé sur une ligne distincte.

Ainsi, vous ne pouvez pas faire :

```
tracé( (x-1)*(x-2)*(x-3), 1, 6) +
tracé( (x-1)*(x-2)*(x-4), 1, 6) +
tracé( (x-1)*(x-2)*(x-5), 1, 6) +
tracé( (x-1)*(x-2)*(x-6), 1, 6) +
tracé( (x-1)*(x-2)*(x-7), 1, 6)
```

ce que Sage considère comme cinq commandes distinctes, au lieu d'une seule grande commande.

Il existe cependant une solution pratique à ce problème : vous pouvez saisir la commande suivante pour obtenir le graphique souhaité.

```
P1 = plot( (x-1)*(x-2)*(x-3), 1, 6)
P2 = plot( (x-1)*(x-2)*(x-4), 1, 6)
P3 = plot( (x-1)*(x-2)*(x-5), 1, 6)
P4 = plot( (x-1)*(x-2)*(x-6), 1, 6)
P5 = plot( (x-1)*(x-2)*(x-7), 1, 6)

P = P1 + P2 + P3 + P4 + P5 P.show()
```

Je suis sûr que nous pouvons tous convenir qu'il s'agit d'un code beaucoup plus lisible.

Autres sujets sur les graphiques et les tracés :

Nous n'avons fait qu'effleurer les bases des tracés dans Sage. Sage propose une riche bibliothèque de types de tracés et de graphiques. Un chapitre entier de ce livre est consacré aux « Techniques avancées de tracé », à partir de la page 91. Si vous souhaitez savoir comment Sage génère les images, nous en parlons à la page 265, mais vous devrez peut-être lire la majeure partie du chapitre 5 pour suivre cette discussion.

Au fait, il n'est pas nécessaire de lire le chapitre en entier dans l'ordre. Vous pouvez simplement utiliser l'index ou la table des matières pour sélectionner précisément le type de chapitre.

Machine Translated by Google

du graphique ou du tracé que vous souhaitez, et lisez simplement cette petite sous-section de ce chapitre.

1.5. Matrices et Sage, première partie

Dans cette section, nous allons apprendre à résoudre des systèmes d'équations linéaires à l'aide de matrices dans Sage. Cette section suppose que vous n'avez jamais travaillé avec des matrices, ou que vous les avez oubliées. Les experts en algèbre linéaire peuvent passer à la section 1.5.3, page 22. En revanche, certains lecteurs ne s'intéressant pas particulièrement aux matrices peuvent passer à la section 1.6, page 30, où ils apprendront à définir leurs propres fonctions dans Sage.

1.5.1. Un premier aperçu des matrices

Supposons que vous souhaitiez résoudre ce système d'équations linéaires :

$$\begin{aligned} 3x - 4y + 5z &= 14 \\ x + y - 8z &= -5 \\ 2x + y + z &= 7 \end{aligned}$$

Tout d'abord, vous devez convertir ces équations dans la matrice suivante :

$$A = \left[\begin{array}{ccc|c} 3 & -4 & 5 & 14 \\ 1 & 1 & -8 & -5 \\ 2 & 1 & 1 & 7 \end{array}\right]$$

Notez que les coefficients des x apparaissent tous dans la première colonne ; les coefficients des y apparaissent tous dans la deuxième colonne ; les coefficients des z apparaissent tous dans la troisième colonne. La quatrième colonne contient les constantes. Nous avons ainsi résumé et abrégé toutes les informations du problème. De plus, notez que les additions sont représentées par un coefficient positif et les soustractions par un coefficient négatif. À propos, la ligne verticale entre la troisième et la quatrième colonne est purement décorative : elle sert à montrer que la quatrième colonne est « spéciale » car elle contient des nombres, tandis que les autres colonnes représentent les coefficients des variables x, y et z.

Les matrices ont une forme particulière appelée « Forme Échelonnée Réduite », souvent abrégée en RREF. La RREF est, d'un certain point de vue, la description la plus simple du système d'équations encore valable. La forme Échelonnée Réduite de cette matrice A est

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 1 \end{array}\right]$$

Machine Translated by Google

Nous pouvons traduire cela littéralement par

$$1x + 0y + 0z = 3 \quad 0x + 1y + 0z = 0 \quad 0x + 0y + 1z = 1$$

ou en notation plus simple

$$x = 3 \qquad y = 0 \qquad \text{avec } = 1$$

Quelle est la solution ? Prenez un moment pour saisir ces trois valeurs et vérifiez qu'elles sont correctes.

1.5.2. Complications dans la conversion des systèmes linéaires

Vous pourriez regarder la discussion précédente et imaginer que ce que nous faisons est une sorte de solution miracle, capable de résoudre rapidement n'importe lequel d'un grand nombre de problèmes extrêmement fastidieux qui prennent (avec un crayon) beaucoup de temps.

L'algèbre matricielle est une solution miracle, maintenant que nous disposons d'ordinateurs. Avant l'avènement de l'ordinateur, les grands problèmes d'algèbre linéaire étaient considérés comme extrêmement pénibles. Pourtant, grâce à l'avènement des ordinateurs, et parce qu'ils peuvent effectuer ces opérations quasi instantanément, cette discipline est un excellent moyen de résoudre rapidement d'énormes systèmes d'équations linéaires. De leur côté, les mathématiciens compétents peuvent résoudre d'importants problèmes scientifiques et industriels, car les problèmes du monde réel comportent souvent de nombreuses variables.

Il serait toutefois plus judicieux de qualifier ce sujet de « semi-automatique » plutôt que « automatique ». Je précise cela parce que l'humain doit d'abord prendre grand soin de convertir le problème verbal en un système d'équations linéaires.
Nous n'aborderons pas ce sujet ici. La deuxième étape consiste à convertir ce système en matrice, ce qui n'est pas si simple. En effet, les erreurs des élèves sont fréquentes à cette étape ; nous allons donc consacrer un peu de temps à un exemple détaillé.

Considérez ce système d'équations :

$$2x - 5z + y = 6 + w$$
$$5 + z - y = 0 \quad w +$$
$$3(x + y) = z \quad 1 + 2x - y =$$
$$w - 3x$$

Il s'avère que ce système d'équations présente les pièges suivants :

- Dans la première équation, les variables sont désordonnées, ce qui est extrêmement fréquent. Très souvent, étudiants et mathématiciens professionnels ne le remarquent pas et placent les mauvais coefficients aux mauvais endroits. Vous pouvez utiliser l'ordre de votre choix, à condition d'utiliser le même ordre pour chaque équation. Cependant, pour éviter les erreurs, les mathématiciens classent généralement les variables par ordre alphabétique.

Machine Translated by Google

• Dans la première équation, le w est du mauvais côté, et nous devons le déplacer à gauche du signe égal. Il est nécessaire que toutes les variables se trouvent d'un côté du signe égal, et tous les nombres de l'autre. Grâce à ces deux corrections, la première équation est maintenant

$$-w + 2x + y - 5z = 6$$

• La deuxième équation a la constante du mauvais côté, et nous devons donc la déplacer sur le signe égal, en pensant à la négativer. • Comme si cela ne suffisait pas, x et w manquent tous les deux dans la deuxième équation. Nous traitons cela comme si les coefficients étaient nuls. Avec ces deux corrections, la deuxième équation est maintenant

$$0w + 0x - y + z = -5$$

• La troisième équation contient des parenthèses, ce qui n'est pas autorisé. Nous avons de se rappeler que $3(x + y) = 3x + 3y$.
• Dans la troisième équation également, le z est du mauvais côté. Il faut veiller à transformer +z en −z lors du croisement du signe égal. • Un troisième défaut de la troisième équation est qu'il n'y a pas de constante. Nous traitons cela comme une constante nulle. En corrigeant ces trois problèmes, la troisième équation devient

$$w + 3x + 3y - z = 0$$

Le véritable problème est la quatrième équation. Nous avons w du mauvais côté, et le pire, c'est qu'il y a deux occurrences de x. En déplaçant le −3x de la droite du signe égal vers la gauche, il devient +3x et se combine avec le +2x déjà présent pour former 5x. Comme si cela ne suffisait pas, la constante est du mauvais côté. Enfin, z est totalement absent. En corrigeant tous ces défauts, nous avons

$$-w + 5x - y + 0z = -1$$

Avec ces équations (modifiées) à l'esprit, nous avons la matrice :

$$B = \begin{vmatrix} -1 & 2 & 1 & -5 & 6 \\ 0 & 0 & -1 & 1 & -5 \\ 1 & 3 & 3 & -1 & 0 \\ -1 & 5 & -1 & 0 & -1 \end{vmatrix}$$

Le RREF de cette matrice serait fastidieux à trouver manuellement. Cependant, il serait bien plus compliqué de résoudre ce système d'équations manuellement, sans matrices, en utilisant l'algèbre ordinaire. Cependant, avec Sage, nous calculerons facilement le RREF dans la sous-section suivante. Pour l'instant, la réponse est la suivante :

dans = −107/7 x = −12/7 y = 54/7 avec = 19/7

que vous pouvez facilement vérifier maintenant si vous le souhaitez.

Machine Translated by Google

1.5.3. Calculer le RREF dans Sage. Cette

sous-section explique comment calculer le RREF d'une matrice, vraisemblablement pour résoudre un système d'équations linéaires. Ce calcul dans Sage ne nécessite que quatre étapes, la dernière étant facultative. Nous allons d'abord définir la matrice.

$$A = \begin{array}{l} 3\ -4\ 5\ 14\ 1\ 1\ -8\ -5 \\ 2\ 1\ 1\ 7 \end{array}$$

Extrait de la sous-section précédente. Comme vous pouvez le constater, il comporte trois lignes et quatre colonnes. Nous allons le faire avec la commande

Sage : A = matrix(3, 4, [3, -4, 5, 14, 1, 1, -8, -5, 2, 1, 1, 7])

Pour comprendre la commande Sage, il faut comprendre que le premier nombre correspond au nombre de lignes, suivi du nombre de colonnes. Vient ensuite la liste des entrées, séparées par des virgules et entre crochets.

Au fait, conservez cette définition de la matrice A dans la cellule Sage tant que vous l'utilisez. Ne l'effacez pas avant d'avoir terminé. Chaque fois que vous cliquez sur « Évaluer », vous envoyez le contenu de la cellule Sage au serveur Sage Cell pour évaluation. Le serveur ne se souviendra pas de la définition de la matrice A si vous la supprimez. En revanche, Sage Cloud (voir annexe B) dispose d'une mémoire permanente beaucoup plus importante.

Notez que Sage ne répond pas à cette commande. C'est parce que nous lui avons indiqué la matrice A, mais que nous ne lui avons rien donné à faire avec A.

La deuxième étape consiste à vérifier que la matrice saisie correspond bien à ce que vous souhaitiez saisir. Cette étape est inévitable, car les fautes de frappe sont très fréquentes. Pour cela, saisissez simplement « print A » sur une ligne, sous la définition de la matrice A. Cliquez ensuite sur « Évaluer ». La matrice s'affiche et vous pouvez vérifier que vous avez saisi le texte souhaité.

La troisième étape consiste à calculer le RREF ou « Réduction de ligne »
Forme d'échelon. » Ceci est fait avec la commande print A.rref( )

et Sage vous indique
la forme d'échelon de ligne réduite, ou

$$\begin{array}{l} 1\ 0\ 0\ 3\ 0\ 1 \mid 0 \\ 0\ 0\ 0\ 1\ 1 \end{array}$$

La quatrième étape, facultative, consiste à vérifier votre travail, mais je vous le recommande vivement ! Nous commençons par saisir (sur une ligne

distincte) print 3*3 + -4*0 + 5*1 et
constatons que la réponse est 14, comme prévu. Ensuite, nous saisissons

---

Voici ce dont je me souviens : les colonnes, comme celles d'une banque, d'un Congrès ou d'un temple grec, sont verticales. Les rangées, par élimination, sont horizontales. Mais faut-il d'abord saisir les rangées ou les colonnes ? Je pense toujours à « RC Cola », une petite marque de soda, qui me fait penser aux rangées d'abord, aux colonnes ensuite. Heureusement, il n'existe pas de « CR Cola ».

Machine Translated by Google

imprimer 1*3 + 1*0 - 8*1
et apprenez que la réponse est -5, comme prévu. La dernière équation est vérifiée de la même manière.

Notez qu'il est crucial de vérifier les trois équations ! Sinon, vous échouerez. pour détecter la solution « imposteur »

x = 30 y = 29 avec = 8

qui satisfait les deux premières équations, mais pas la troisième.

Considérons maintenant l'exemple « compliqué » de la sous-section précédente. Il s'agissait

$$B = \begin{vmatrix} -1 & 2 & 1 & -5 & 6 \\ 0 & 0 & -1 & 1 & -5 \\ 1 & 3 & 3 & -1 & 0 \\ -1 & 5 & -1 & 0 & -1 \end{vmatrix}$$

et nous pouvons entrer cela dans Sage avec

B = matrice(4, 5, [-1, 2, 1, -5, 6, 0, 0, -1, 1, -5, 1, 3, 3, -1, 0, -1, 5, -1, 0, -1])

La matrice B étant construite à partir d'une très longue liste de nombres, il faut veiller à les saisir correctement, en n'omettant ni en dupliquant aucun nombre. On vérifie l'exactitude de B en tapant « print B » seul sur sa propre ligne, et on constate que l'on a saisi ce que l'on voulait saisir.

Avant de continuer, j'aimerais partager avec vous une notation simple et utile. On souhaite parfois faire référence à une position dans la matrice. Par exemple, le 6 de la première ligne, cinquième colonne de B peut s'écrire B15.
De même, le 5 de la quatrième ligne, deuxième colonne de B peut s'écrire B42. De même, les deux -5 se trouvent en B14 et B25. N'oubliez pas que la ligne vient en premier et la colonne en second, comme pour RC Cola.

Dans Sage, la dénomination des coordonnées des entrées de matrice commence à 0 et non à 1. Cela signifie que le coin supérieur gauche est B[0][0] dans Sage, mais B11 en mathématiques. De même, B14 est saisi sous la forme B[0][3] et B23 sous la forme B[1][2]. Cette anomalie est héritée par Sage du langage informatique Python et ne peut être corrigée, car Sage fonctionne sur Python.

Au fait, une autre façon de saisir une matrice assez grande est la suivante :

B = matrice([[-1, 2, 1, -5, 6], [0, 0, -1, 1, -5], [1, 3, 3, -1, 0], [-1, 5, -1, 0, -1]
])

Le code ci-dessus est une liste de listes. Chaque liste de cinq nombres entre parenthèses et séparés par des virgules représente une ligne ; il y a ensuite quatre listes de ce type, séparées par des virgules et entre parenthèses, pour représenter la matrice entière. Comme ce format indique le nombre de lignes et de colonnes, il n'est pas nécessaire d'indiquer « 4, 5 » pour indiquer à Sage que B comporte 4 lignes et 5.

Machine Translated by Google

colonnes. Vous pouvez utiliser le format qui vous convient le mieux.

Maintenant que B a été saisi (par l'une ou l'autre méthode), nous devons ajouter la commande print B.rref() sur sa propre ligne, et nous apprenons que le RREF est

$$\left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & -107/7 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & -12/7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 54/7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 19/7 & & & \end{array}\right)$$

ce qui donne la réponse finale de

dans = −107/7 x = −12/7 y = 54/7 avec = 19/7

pour le système d'équations original. Il faut maintenant vérifier notre travail. Par exemple, vérifions la première équation avec

Imprimons 2*(-12/7) - 5*(19/7) + 54/7 et 6 + -107/7, ce qui donne -65/7. L'important n'est pas qu'ils donnent -65/7, mais plutôt qu'ils soient égaux. La première équation est donc satisfaite. Vérifions les trois autres équations de la même manière.

Nous devons vraiment insérer chacune des quatre valeurs dans chacune des quatre équations, soit dit en passant. Par exemple, la solution de l'« imposteur »

dans = −139/7 x = −8/7 y = 64/7 avec = 29/7

satisfait les trois premières de ces équations, mais ne parvient pas à satisfaire la dernière.

Si vous avez lu la section 1.5.2, vous vous souviendrez que nous avons fourni un travail considérable pour obtenir B. Ces nombreuses étapes auraient pu contenir une erreur si nous avions été négligents. C'est une autre raison pour laquelle nous devons vérifier notre travail.

1.5.4. Résumé de base.

Nous avons maintenant appris à utiliser les commandes matrix et rref pour calculer la RREF d'une matrice et ainsi résoudre un système d'équations linéaires. Pour effectuer toutes les étapes en une seule fois,

je tape souvent B = matrix(4, 5, [-1, 2, 1, -5, 6, 0, 0, -1, 1, -5, 1, 3, 3, -1, 0, -1, 5, -1, 0, -1])

```
imprimer "Question :"
imprimer
B imprimer
"Réponse :" imprimer B.rref()
```

Les lignes contenant « Question : » et « Réponse : » sont facultatives, mais contribuent à la lisibilité du résultat. Elles m'encouragent également à vérifier que j'ai bien saisi la matrice souhaitée.

Machine Translated by Google

à saisir, sans faute de frappe. Je fais presque toujours une erreur typographique, que je dois corriger. Le résultat est le suivant :

ci-dessous:

Question:

```
[-1 2 1 -5 6]
[ 0 0 -1 1 -5]
[ 1 3 3 -1 0]
[-1 5 -1 0 -1]
```

Répondre:

```
[[[[  1   0   0   0 -107/7]
      0   1   0   0 -12/7]
      0   0   1   0 54/7]
      0   0   0   1 19/7]
```

1.5.5. La matrice identitaire

Examinons les RREF que nous avons calculés jusqu'à présent. Voyez-vous une tendance ? en eux ? Ou devrais-je dire, un motif dans toutes les colonnes sauf la dernière ?

$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{array}\right) \quad \text{et} \quad \left(\begin{array}{ccccc} 1 & 0 & 0 & 0 & -107/7 \\ 0 & 1 & 0 & 0 & -12/7 \\ 0 & 0 & 1 & 0 & 54/7 \\ 0 & 0 & 0 & 1 & 19/7 \end{array}\right)$$

Nous pouvons voir que toutes les colonnes (sauf la dernière colonne) ont des zéros partout, à l'exception d'une diagonale très visible de uns. Ce motif (de zéros) partout sauf ceux dans les positions A11, A22, A33, . . . , est un modèle qui se produit extrêmement souvent. Sans surprise, il porte un nom : les mathématiciens l'appellent cette « matrice identitaire ».

Pensez au moment où vous êtes obligé de présenter une « pièce d'identité », par exemple lorsque vous êtes arrêté pour excès de vitesse ou en essayant d'entrer dans un bar. La carte d'identité affiche rapidement vos informations vitales, telles que votre nom, votre date de naissance, votre taille, votre sexe, votre adresse, etc. De même, la matrice d'identité présente les faits essentiels concernant un système d'équations linéaires. Vous pouvez lire le valeurs des variables désactivées, en lisant simplement la dernière colonne.

1.5.6. Un défi à relever pour s'entraîner seul

À l'aide de Sage, résolvez le système d'équations linéaires suivant. Vérifiez votre réponses avec une calculatrice à main.

$$3481x + 59y + z = 0{,}87$$
$$6241x + 79y + z = 0{,}61$$
$$9801x + 99y + z = 0{,}42$$

Machine Translated by Google

Remarque : ce problème peut paraître artificiel, mais il s'agit en réalité d'un problème appliqué que nous aborderons prochainement. Comme de nombreux problèmes applicatifs, les réponses comportent des décimales, ce qui, espérons-le, ne devrait pas vous effrayer. Nous verrons la réponse prochainement.

1.5.7. Matrice de Vandermonde

En économie comme en affaires, on parle souvent de la courbe prix-demande. Après tout, lorsque le prix d'un produit augmente, la demande devrait diminuer, et lorsque le prix baisse, la demande devrait augmenter, du moins dans des conditions normales.

Il est parfois utile de connaître la courbe, mais elle ne peut généralement être calculée qu'en interrogeant des clients potentiels. La technique standard consiste à trouver 2 000 personnes, à leur montrer le produit à chacune et à leur demander si elles l'achèteraient ou non. Elles reçoivent le produit gratuitement en échange de leur participation à l'enquête. L'essentiel est que, parmi ces 2 000 personnes, une vingtaine de prix différents soient utilisés, chaque prix étant présenté à 100 personnes.

Ce processus fournira 20 points de données, ce qui est souvent tout à fait suffisant. Cependant, il est parfois préférable d'avoir une fonction réelle. On recherche généralement une fonction quadratique. Il y a de bonnes raisons à cela, mais ce serait une digression6 de les explorer ici. Imaginez qu'un de vos amis, qui dirige une start-up, ait réalisé une telle enquête, mais avec 3 prix et 300 personnes, pour un nouveau produit attrayant. • Au prix de 59 $, l'enquête indique que 87 %

l'achèteraient. • Au prix de 79 $, l'enquête indique que 61 % l'achèteraient. • Au prix de 99 $, l'enquête indique que 42 % l'achèteraient.

Toute fonction quadratique peut être écrite sous la forme

$$f(x) = ax2 + bx + c$$

mais la question devient : comment pouvons-nous trouver a, b ou c ?

Dans cette optique, et en supposant que f(x) modélise bien la relation prix-demande, on peut alors supposer que f(59) = 0,87, f(79) = 0,61 et f(99) = 0,42. Dans cette optique, nous pourrions écrire les équations suivantes.

$$a(592) + b(59) + c = 0,87 \quad a(792) + b(79) + c = 0,61 \quad a(992) + b(99) + c = 0,42$$

Étonnamment, ces équations sont linéaires, ce qui n'est pas forcément attendu puisqu'il s'agit d'une fonction quadratique. Il suffit d'écrire

---

Sans surprise, cette technique est plus courante dans les petits appareils électroniques grand public ou dans l'industrie alimentaire. articles, et moins courants chez les constructeurs de voitures de sport de luxe.

6Il faut dire que l'ajustement de polynômes de haut degré à des points de données est tout à fait faisable, mais extrêmement imprudent, en raison d'un phénomène appelé « surajustement ».

Machine Translated by Google

matrice vers le bas :

592 59 1 0,87 792 79
1 0,61 992 99 1 0,42

et demandez à Sage de calculer le RREF. Or, vous l'avez déjà fait vous-même ! Vous avez déjà calculé la solution.

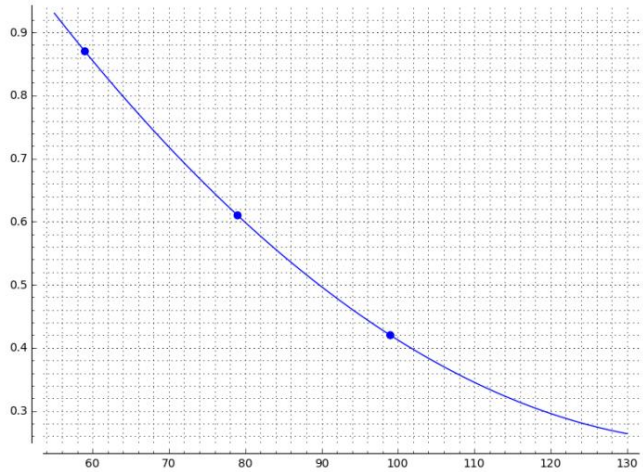
a = 0,0000875 b = −0,025075 c = 2,0448375

comme pratique, à la page 25 comme section 1.5.6.

Finalement, nous concluons que la fonction est

$$f(p) = 0{,}0000875p^2 - 0{,}025075p + 2{,}0448375$$

Le graphique de cette fonction est donné ci-dessous :



Si vous êtes curieux, le code qui a produit ce complot était

```
f(x) = 0,0000875*x^2 - 0,025075*x + 2,0448375

plot( f, 55, 130, gridlines='minor') + point( [59, 0.87], size=50 ) + point([79,
0.61], size=50) + point( [99, 0.42], size=50 )
```

La technique générale d'ajustement d'un polynôme de degré n à n+1 points à l'aide d'une matrice est due à Alexandre-Théophile Vandermonde (1735–1796), et c'est pourquoi les matrices sont appelées matrices de Vandermonde en son honneur. Par exemple, pour ajuster un polynôme de quatrième degré à cinq points (x1, y1), (x2, y2), . . . , (x5, y5), on utiliserait la matrice suivante :

$$\begin{matrix} (x1)^4 & (x1)^3 & (x1)^2 & (x1)\ 1 \\ (x2)^4 & (x2)^3 & (x2)^2 & (x2)\ 1 \\ (x3)^4 & (x3)^3 & (x3)^2 & (x3)\ 1 \\ (x4)^4 & (x4)^3 & (x4)^2 & (x4)\ 1 \\ (x5)^4 & (x5)^3 & (x5)^2 & (x5)\ 1 \end{matrix}$$

Machine Translated by Google

1.5.8. Les cas semi-rares

Nos exemples précédents nous ont fourni une solution unique. Nous allons maintenant examiner les deux cas semi-rares. Ceux-ci se produisent lorsqu'une ligne contenant uniquement des zéros se trouve en bas du RREF, se terminant par un nombre différent de zéro ou par zéro. Le système d'équations est le suivant :

$$x + 2y + 3z = 7 \quad 4x + 5y + 6z = 16 \quad 7x + 8y + 9z = 24$$

Cela donne la matrice

$$C1 = \begin{pmatrix} 1 & 2 & 3 & 7 \\ 4 & 5 & 6 & 16 \\ 7 & 8 & 9 & 24 \end{pmatrix}$$

et que nous entrerons dans Sage avec C1 =
matrix(3,4, [1, 2, 3, 7, 4, 5, 6, 16, 7, 8, 9, 24])
Ensuite, pour trouver le RREF, nous tapons
C1.rref( ) et
recevons en retour

[1 0 -1 0] [0 1 2 0]
[0 0 0 1]

Cela se traduit par

$$x - z = 0$$

$$y + 2z = 0 \quad 0 = 1$$

L'équation du bas dit 0 = 1, ce qui est clairement faux ! Il est impossible de satisfaire à l'exigence 0 = 1. Par conséquent, cette équation n'a pas de solution. Au lieu d'être une solution, une liste de solutions ou une infinité de solutions, la réponse est la phrase « Ce système d'équations linéaires n'a pas de solutions ». Ce sera le résultat chaque fois que le RREF aura une ligne contenant uniquement des zéros, sauf la dernière colonne, qui est un nombre non nul.

Chaque fois que vous voyez une ligne avec tous les zéros, sauf dans la dernière colonne où un nombre autre que zéro est trouvé, vous devez vous rappeler que le système n'a pas de solutions.

Un autre cas intéressant consiste à changer le 24 dans la dernière équation en 25. Cela donne la matrice

$$C2 = \begin{pmatrix} 1 & 2 & 3 & 7 \\ 4 & 5 & 6 & 16 \\ 7 & 8 & 9 & 25 \end{pmatrix}$$

et que nous entrerons dans Sage avec
C2 = matrice(3,4, [1, 2, 3, 7, 4, 5, 6, 16, 7, 8, 9, 25])

Machine Translated by Google

Maintenant, pour trouver la « solution », nous tapons C2.rref( )
et recevons en retour

[1 0 -1 -1] [0 1 2 4] [0
0 0 0]

Comme vous pouvez le constater, ce RREF comporte une ligne entièrement composée de zéros, se terminant par un zéro. Cela indique généralement une infinité de solutions, un point que nous clarifierons prochainement. Certains instructeurs vous permettent d'écrire « une infinité de solutions » et dans certains problèmes d'application, vous ne vous soucierez pas de fournir un moyen d'indiquer quelles sont ces solutions.

Cependant, parfois, vous aimeriez savoir quelles sont les solutions. C'est particulièrement vrai pour les problèmes de géométrie. De toute façon, on ne peut pas dresser une liste, car il existe une infinité de solutions. Souvent, même si l'espace des solutions est infini, il possède ce que les mathématiciens appellent « un degré de liberté ». Cela signifie qu'il existe une variable libre, que l'on peut définir à la valeur de son choix. Cependant, une fois cette définition établie, toutes les autres variables sont déterminées par ce choix. Ce processus est expliqué à l'annexe D, ainsi que de nombreux autres faits concernant les systèmes linéaires à infinité de solutions.

Quand y a-t-il une infinité de solutions ?

Alors, que nous apprend réellement une ligne de zéros ? La plupart du temps, votre système d'équations comporte deux équations à deux inconnues ; trois équations à trois inconnues ; quatre équations à quatre inconnues ; ou cinq équations à cinq inconnues, et ainsi de suite. On parle alors de système linéaire « défini avec précision ». Un système linéaire défini avec précision donne une matrice 2 × 3, 3 × 4, 4 × 5, 5 × 6 ou n × (n + 1). La règle empirique que je vais vous présenter fonctionne dès lors que le nombre d'équations est égal à celui des inconnues, ce qui signifie que le système est défini avec précision, c'est-à-dire qu'il y a une colonne de plus que de lignes.

Voici la règle générale : si nous voyons le signe révélateur « aucune solution », alors nous ignorons toutes les autres informations et déclarons « aucune solution ». À défaut, s'il y a une rangée de zéros, alors il y aura une infinité de solutions.

Attention cependant ! Cette règle ne fonctionne pas s'il y a « trop peu » ou « trop » d'équations. S'il y a trop d'équations (trop de lignes), le système est dit « surdéfini ». S'il y a trop de variables (trop de colonnes), le système est dit « sous-défini ». Par exemple, dans la section D.3, page 312, je vais énumérer quelques contre-exemples démontrant que cette règle empirique est fausse pour les matrices surdéfinies ou sous-définies. Pour votre curiosité, les systèmes d'équations linéaires sous-définis sont abordés dans des questions intéressantes de géométrie analytique, et les systèmes d'équations linéaires surdéfinis sont abordés en cryptanalyse – et ont été utilisés dans ma thèse de doctorat.

Machine Translated by Google

Enfin, il convient de noter que l'annexe D présente une technique relativement simple qui explique précisément comment traiter les systèmes d'équations linéaires à solutions infinies, quel que soit le nombre de lignes et de colonnes. L'astuce est légèrement plus complexe, mais elle fonctionne toujours. De plus, elle fournit des formules permettant de produire autant de solutions que vous le souhaitez à partir de l'ensemble des solutions infinies. Vous pouvez également utiliser ces formules à d'autres fins, par exemple pour vérifier si une solution candidate est bien l'une des solutions infinies.

N'allez pas croire que les systèmes linéaires avec une infinité de solutions sont extrêmement rares ; ils apparaissent parfois dans des applications. Cependant, ils sont suffisamment rares pour que nous ne vous ennuyions pas avec ces détails pour l'instant, mais nous attendrons l'annexe D.

1.6. Créer vos propres fonctions dans Sage

Définir ses propres fonctions dans Sage est extrêmement simple. À mon avis, c'est l'une des fonctionnalités les plus remarquables de Sage : les créateurs ont rendu les fonctions très faciles à utiliser. Vous pouvez créer votre propre fonction en utilisant exactement les symboles que vous utiliseriez habituellement pour écrire au crayon ou au tableau.

Définition d'une fonction :

Par exemple, si vous souhaitez définir $f(x) = x^3 - x$ puis vous tapez

```
f(x) = x^3-xf(2)
```

Nous avons défini f(x) et lui avons demandé la valeur de f(2). On pourrait également lui demander f(3), ou tout autre nombre. De même, si l'on voulait définir $g(x) = \sqrt{1 - x^2}$ tu taperais

```
g(x) = sqrt( 1-x^2 ) g(1/4)
```

Nous avons défini g(x) et cherché à obtenir l'évaluation de g(1/4). On pourrait tout aussi bien se demander quelle est la valeur de g(1/2) ou de tout autre nombre du domaine de définition de g.

Pour évaluer plusieurs valeurs, vous pouvez utiliser la commande print.

```
g(x) = sqrt( 1-x^2 ) imprimer
g(0,1) imprimer
g(0,01) imprimer
g(0,001) imprimer
g(0,0001) imprimer
g(0,00001) imprimer
g(0,000001)
```

Machine Translated by Google

Si vous avez peut-être commencé le calcul, vous pouvez voir que le résultat ci-dessus vous aidera à évaluer

$$\lim_{x \to 0+} g(x) = 1$$

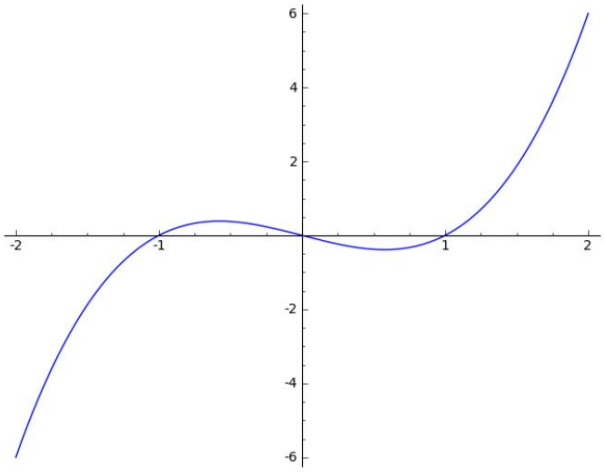
numériquement, ou aidez à confirmer votre réponse algébrique. Si vous n'avez pas étudié le calcul, vous pouvez ignorer cette remarque !

Fonctions de traçage :

Maintenant que nous avons défini f(x) et g(x), nous pouvons les tracer avec

```
f(x) = x^3-x plot( f,
-2, 2 )
```

vous obtenez l'intrigue attendue



De même si vous tapez

```
g(x) = sqrt( 1-x^2 ) plot( g, 0,
1 )
```

ce qui équivaut à

```
g(x) = sqrt( 1-x^2 ) plot( g(x),
0, 1 )
```

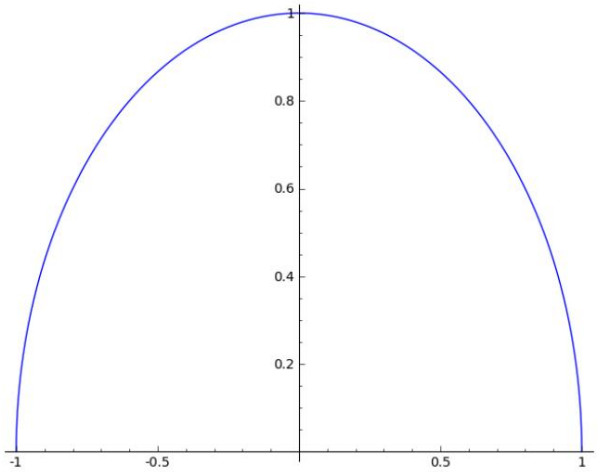
alors vous obtenez le graphique de g(x) sur $0 \le x \le 1$.

Enfin et surtout, si vous tapez

```
g(x) = sqrt( 1-x^2 ) plot( g )
```

Si la formule est plutôt abrégée (en omettant l'intervalle souhaité sur l'axe des abscisses), Sage supposera par défaut que vous souhaitez $-1 \le x \le 1$. Le graphique est alors généré.

Machine Translated by Google



Notez que vous devez absolument conserver les définitions de f(x) ou g(x) dans la fenêtre et les écrire chaque fois que vous cliquez sur « Évaluer ». C'est une fonctionnalité du serveur cellulaire Sage. Il est doté d'une fonction d'amnésie et traite chaque pression sur le bouton « Évaluer » comme un univers mathématique distinct. Ceci est utile aux débutants, car cela facilite grandement la correction des petites erreurs typographiques. Il suffit de corriger l'erreur, de cliquer sur « Évaluer » et tout rentre dans l'ordre. Sage Cloud fonctionne légèrement différemment, comme expliqué page 299.

Utilisation de variables intermédiaires :

Parfois, vous utiliserez souvent la même valeur et souhaiterez la stocker. Par exemple, si vous saisissez souvent 355/113, vous pouvez saisir

k=355/113

qui stocke silencieusement la valeur 355/113 dans la variable k. Vous pouvez l'utiliser dans n'importe quelle ligne ultérieure de la fenêtre du serveur de cellules Sage, par exemple

k=355/113,

imprimez 2+k,

imprimez 1+k,

ce qui affiche les valeurs correctes de 581/113 et 468/113. Cependant, si vous modifiez la valeur de k, assurez-vous d'utiliser le bouton « Évaluer » pour réévaluer vos formules.

De plus, si vous remplacez 2+k par 2,0+k, vous obtenez des approximations décimales. Si une affirmation mathématique est entièrement exprimée en termes d'entiers et de nombres rationnels, Sage fournira une réponse exacte. Si une décimale est introduite, il fournira une approximation décimale.

La technique des variables intermédiaires est idéale pour des éléments de la physique comme l'accélération due à la gravité ou la vitesse de la lumière, qui ne changent pas pendant un problème.

Il s'avère que ce k particulier est une approximation de π, un peu comme 22/7, mais en bien meilleur. Si nous voulions trouver l'erreur relative de k comme

Machine Translated by Google

approximation de π nous nous souviendrions de la formule

$$\text{relative} = \text{vérité} \quad \frac{\text{approximation} - \text{vérité erreur}}{}$$

et tapez

k=355/113

N((k-pi)/pi) et apprenez

que l'erreur relative est d'environ 84,9 parties par milliard. Pas mal.

Cette approximation a été trouvée par l'astronome chinois Zu Chongzhi, également connu sous le nom de Tsu Ch'ung-Chih. On peut comparer 355/113 à 22/7 avec N((22/7-pi)/pi) et constater que cette approximation

beaucoup plus courante de π comme 22/7 présente une erreur relative d'environ 402,5 parties par million. C'est bien pire ! En fait, 22/7 est 4 740,9 fois pire ; il est donc dommage que nous n'enseignions pas 355/113 aux élèves, mais plutôt 22/7.

Composition de fonctions : Sage

permet également de composer très facilement deux fonctions. Prenons les deux fonctions suivantes : f(x) = 3x + 5.

Nous souhaitons et $g(x) = x^3 + 1$

peut-être explorer f(g(x)). La méthode la plus simple consiste à définir une troisième fonction, h(x) = f(g(x)). Nous pourrions ensuite afficher cette fonction et les valeurs de h(x) pour x = 1, x = 2 et x = 3. Le code Sage est simple : f(x) = 3*x + 5 ; g(x) = x^3 + 1 ; h(x) = f(g(x)).

```
imprimer h(x)
imprimer h(1)
imprimer h(2)
imprimer h(3)
```

Cela produit la sortie

```
3*x^3 + 8
11
32
89
```

comme souhaité. Cependant, si nous remplaçons la troisième ligne par h(x) = g(f(x)), nous obtenons un résultat complètement différent

```
(3*x + 5)^3 + 1).
513
1332
2745
```

Machine Translated by Google

Bien sûr, il n'y a aucune raison de s'attendre à ce que f(g(x)) = g(f(x)) en général. La façon de voir cela dans ce cas précis est de calculer les deux fonctions en utilisant algèbre. Nous pouvons calculer

$$\begin{aligned} f(g(x)) &= 3(g(x)) + 5 \\ &= 3(x^3 + 1) + 5 \\ &= 3x^3 + 3 + 5 \\ &= 3x^3 + 8 \end{aligned}$$

ce qui est complètement différent de

$$\begin{aligned} g(f(x)) &= g(3x + 5) \\ &= (3x + 5)^3 + 1 \\ &= 27x^3 + 135x^2 + 225x + 125 + 1 \\ &= 27x^3 + 135x^2 + 225x + 126 \end{aligned}$$

Fonctions multivariées :

Cette section traite des fonctions à une variable. Sage peut très facilement et gérer simplement des fonctions de plusieurs variables. Par exemple,

$$f(x, y) = (x^4 + 4 + e^{2-5x} + e^{2})^2$$

Nous aborderons ce sujet en détail dans la section 4.1 à la page 127.

1.7. Utilisation de Sage pour manipuler les polynômes

Nous allons maintenant explorer quelques utilisations de la notation fonctionnelle, abordées dans la section précédente, pour les polynômes. Voyons d'abord comment Sage peut additionner des polynômes. très facilement. Nous allons explorer les polynômes suivants.

$$\begin{aligned} a(x) &= x^2 - 5x + 6 \\ b(x) &= x^2 - 8x + 15 \end{aligned}$$

Si nous tapons le code suivant

```
a(x) = x^2 - 5*x + 6
b(x) = x^2 - 8*x + 15

a(x) + b(x)
```

nous obtenons la bonne réponse de $2x^2 - 13x + 21$. Notez qu'il est très important de rappelez-vous de l'astérisque entre le 5 et le x ainsi que le 8 et le x.

De même, si vous changez le signe + entre a(x) et b(x) en un signe -, alors nous obtenons également la bonne réponse de $3x - 9$. Comme vous pouvez l'imaginer, le le produit peut également être calculé facilement :

```
a(x) = x^2 - 5*x + 6
b(x) = x^2 - 8*x + 15
```

Machine Translated by Google

a(x)*b(x)

mais cela fournit la réponse inachevée mais indéniablement vraie (x^2 - 5*x + 6)*(x^2 - 8*x + 15)

Au lieu de cela, nous pouvons faire ce qui suit a(x) = x^2 - 5*x + 6
b(x) = x^2 - 8*x + 15

g(x) = a(x)*b(x)
g.expand() et

nous obtenons g(x) écrit sans parenthèses. En calcul formel, on parle de « forme développée ». La forme développée de notre g(x) est x |--> x^4 - 13*x^3 + 61*x^2 - 123*x + 90

Le symbole |--> signifie « évalue à ». Ce que Sage vous dit ici, c'est que g(x) est une fonction qui envoie x à x. Au lieu de cela, nous ^{4} ^{3} − 13x + 61x ^{2} − 123x + 90.

pouvons écrire g.factor() à la place de g.expand(), auquel cas nous obtenons la sortie suivante : (x - 2)*(x - 3)^2*(x - 5)

Vous pouvez également factoriser plus directement avec
a(x) = x^2 - 5*x + 6
factoriser(a(x)) et

obtenir la bonne réponse (x − 2)(x − 3). Alternativement, nous pouvons être encore plus directs en tapant
factor(x^2 - 8*x + 15) pour obtenir

(x − 3)(x − 5).

Vous pouvez également calculer le « plus grand commun diviseur » (PGCD) de deux polynômes. Par exemple, le code suivant : a(x) = x^2 - 5*x + 6 ; b(x) = x^2 - 8*x + 15

pgcd(a(x), b(x)) fournit

la bonne réponse à (x − 3). Bien sûr, nous le savions grâce aux factorisations trouvées précédemment. Le facteur commun de (x − 2)(x − 3) et (x − 3)(x − 5) est clairement (x − 3).

Il convient de noter que ces exemples étaient triviaux, car nous avons utilisé des polynômes du second degré. Le polynôme suivant est un polynôme que je ne voudrais pas factoriser à la main, même si, il faut l'admettre, le théorème des racines entières[7]

[7] Si f(x) est un polynôme à coefficients entiers, alors toute racine entière doit être un diviseur du terme constant de f(x). La plupart des nombres ont un nombre modeste de diviseurs ; il est donc relativement facile de les vérifier tous et d'obtenir ainsi la liste complète de toutes les racines entières de f(x). Notez que cela fonctionne même pour les polynômes quintiques et de degré supérieur, qui sont par ailleurs extrêmement difficiles à manipuler. Il existe également un « théorème des racines rationnelles », mais nous n'en parlerons pas ici.

Machine Translated by Google


```
a(x) = x^2 - 5*x + 6 b(x) = x^2 -
8*x + 15

f(x) = a(b(x))

imprimer "Direct :"
imprimer f(x)
imprimer "Étendu :" imprimer
f.expand() imprimer
"Factorisé :" imprimer
f.factor()
```

Figure 1. Quelques exemples de code Sage pour explorer la composition de deux fonctions

produirait la bonne réponse en utilisant seulement un crayon et beaucoup de patience humaine (x^4 - 60*x^3 + 1330*x^2 - 12900*x + 46189)

En parlant du théorème des racines entières, nous pourrions taper factor(46189) et obtenir 11 * 13 * 17 * 19 pour nous aider à trouver les racines de ce polynôme quartique.
La commande factor gère à la fois la factorisation d'un polynôme et celle d'un entier. La commande divisors(46189), quant à elle, répertorie tous les entiers positifs qui divisent 46 189. Naturellement, cette liste est plus longue que factor, qui ne répertorie que les diviseurs premiers (et leurs exposants, le cas échéant). Comparez factor(2048) et divisors(2048), si vous le souhaitez.

La composition des fonctions ne pose pas non plus de problème. Le code présenté dans la figure 1 compose deux polynômes et produit la réponse sous trois formes. Il est à noter qu'il s'agit de notre premier véritable « programme » dans Sage, dans le sens où c'est la première fois que nous utilisons plus de deux ou trois commandes simultanément.

En particulier, notre programme produit la sortie :

```
Direct:
 (x^2 - 8*x + 15)^2 - 5*x^2 + 40*x - 69
Étendu : x |-->
x^4 - 16*x^3 + 89*x^2 - 200*x + 156
Factorisé : (x^2
 - 8*x + 13)*(x - 2)*(x - 6)
```

Le code ci-dessus illustre l'utilité d'étiqueter certaines parties de votre sortie avec des commandes d'impression si vous générez plusieurs éléments. Cet étiquetage (parfois appelé « annotation de la sortie ») est une bonne pratique générale pour gérer des tâches de plus en plus complexes dans Sage.

Comme nous l'avons vu à la fin de la section 1.6, vous obtenez une réponse différente si vous remplacez f(x) = a(b(x)) par f(x) = b(a(x)). N'hésitez pas à essayer et à vous faire votre propre opinion !

Machine Translated by Google

1.8. Utiliser Sage pour résoudre des problèmes de manière symbolique

Lorsqu'on dit qu'un ordinateur a résolu un problème, cela peut se faire de deux manières : symboliquement ou numériquement. Lorsque nous résolvons numériquement, nous obtenons un développement décimal pour une (très bonne) approximation de la réponse. Lorsque nous résolvons symboliquement, nous obtenons une réponse exacte, souvent en termes de radicaux ou d'autres fonctions complexes. Il s'agit de déterminer si les solutions à

$$\frac{2x}{2} - x - 2 = 0$$

sont $1 + \sqrt{5}$ et $1 - \sqrt{5}$ ou dire qu'ils sont

$$-1{,}23606797749979 \text{ et } 3{,}23606797749979$$

Le premier exemple est une solution symbolique, et le second une solution numérique. Une réponse numérique est souvent souhaitée. Cependant, si la réponse est une formule, la solution symbolique est la seule solution. Cette section traite de la solution symbolique, et la suivante de la solution numérique.

1.8.1. Résolution de formules à variable unique. Essayons

d'abord de résoudre des problèmes à variable unique. Nous pouvons taper solve(x^2 +

3*x+2, x)

et ensuite nous obtenons

```
[x == -2, x == -1]
```

Il est important de se souvenir de l'astérisque ! Pour être précis, pour taper

La fonction solve(x^2 + 3x+2,

x) serait erronée, en raison de l'absence d'astérisque entre le « 3 » et le « x ». Un autre exemple, plus illustratif, de calcul symbolique est solve(x^2+9*x+15==0,x), qui renvoie

```
[x == -1/2*sqrt(21) - 9/2, x == 1/2*sqrt(21) - 9/2]
```

Au fait, avez-vous remarqué que les solutions sont présentées dans une liste séparée par des virgules et entre crochets ? C'est un exemple de liste, dont la notation a été héritée par Sage du langage informatique Python.

Une façon de vraiment comprendre ce qu'est la distinction symbolique et exacte est de comparer

3+1/(7+1/(15+1/(1+1/(292+1/(1+1/(1+1/6))))))

ce qui renvoie 1 354 394 / 431 117 à

N(3+1/(7+1/(15+1/(1+1/(292+1/(1+1/(1+1/6)))))))

qui renvoie 3,14159265350241. On pourrait facilement le confondre avec π. En fait, il est très proche de π avec une erreur relative de $2{,}78 \times 10^{-11}$.

Il n'y a aucune restriction concernant les polynômes. Vous pouvez également saisir

Machine Translated by Google

var('theta')

résolvez(sin(theta)==1/2, theta) pour obtenir

[theta

== 1/6*pi], mais notez

qu'il existe de nombreuses valeurs pour lesquelles sin θ = 1/2. Par exemple :

$$\theta \quad \ldots, \quad -\frac{31p}{6}, -\frac{23p}{6}, -\frac{19p}{6}, -\frac{11p}{6}, -\frac{7p}{6}, \frac{p}{6}, \frac{5p}{6}, \frac{13p}{6}, \frac{17p}{6}, \frac{25p}{6}, \frac{29p}{6}, \ldots$$

il existe plusieurs valeurs de θ qui résolvent sin θ = 1/2.

Déclaration de variables : La

commande var ci-dessus peut vous surprendre. L'idée est d'indiquer à Sage que cette variable n'est pas encore connue. Nous avons déjà utilisé des variables, mais c'est parce que nous leur avons attribué des valeurs. Pour qu'une variable représente une quantité inconnue, il faut utiliser var.

La variable x est toujours pré-déclarée, vous n'avez pas besoin de la déclarer : Sage suppose que x est une inconnue.

1.8.2. Résolution de formules multivariables. Nous allons

maintenant utiliser Sage pour recalculer la formule quadratique. Nous devons d'abord déclarer nos variables avec :

var('ab c') puis

nous tapons

solve(a*x^2 + b*x + c == 0, x) et recevons en

retour

[x == -1/2*(b + sqrt(-4*a*c + b^2))/a, x == -1/2*(b - sqrt(-4*a*c
+ b^2))/a] dont certains termes sont dans un ordre

inhabituel, mais qui est sans aucun doute correct. Une fois de plus, les crochets et la virgule indiquent une liste dans Sage, quelque chose que Sage a hérité du langage informatique Python.

Bien sûr, vous connaissez probablement très bien la formule quadratique. (Du moins, je l'espère !) Cependant, vous ignorez probablement la formule cubique de Cardano. Dans la section 1.8.5, page 43, nous verrons comment Sage peut « redécouvrir »
La formule de Cardano sur place lorsqu'elle est contestée de la même manière.

Déclarer plusieurs variables à la fois :

Au fait var('ab

c')

est simplement une abréviation de

var('a')
var('b')
var('c')

Machine Translated by Google

Notez également que les formulaires suivants sont équivalents var("a, b, c") var("ab c") var('ab c') et vous pouvez utiliser celui existe('a, b, c') que vous préférez.

1.8.3. Systèmes d'équations linéaires. On peut résoudre simultanément plusieurs équations, qu'elles soient linéaires ou non linéaires. Un cas simple serait de résoudre

$$x + b = 6$$
$$x - b = 4$$

ce qui se ferait en tapant

```
var('b')
résoudre( [x+b == 6, xb == 4], x, b )
```

Notez l'utilisation des crochets et des virgules pour former une liste dans Sage. Vous pouvez entourer les données de [et], et séparer les entrées par des virgules, pour créer une liste.

Il est également important de souligner qu'il n'y a aucun inconvénient à déclarer b deux fois. Deux commandes var ne se nuisent pas mutuellement. En revanche, il n'est pas non plus nécessaire de déclarer b deux fois, car une fois déclaré, Sage se souviendra qu'il s'agit d'une variable.

Un système linéaire plus typique pourrait être

$$9a + 3b + 1c = 32 \quad 4a +$$
$$2b + 1c = 15$$
$$1a + 1b + 1c = 6$$

et pour résoudre cela, nous taperions

```
var('a, b, c')
résoudre( [9*a + 3*b + c == 32, 4*a + 2*b + c == 15, a + b + c == 6], a,
b, c )
```

et Sage donne la réponse

```
[[a == 4, b == -3, c == 5]]
```

Ce qui est correct. Bien sûr, c'est exactement le système d'équations linéaires que vous utiliseriez si on vous demandait de trouver la parabole reliant les points (3, 32) et (2, 15) ainsi que (1, 6). Ce serait $f(x) = 4x\ 2-3x+5$.
Ceci est un autre exemple de matrice de Vandermonde (sauf que nous n'avons pas utilisé de matrice ici), comme nous l'avons vu défini dans la section 1.5.7 à la page 26.

Naturellement, les systèmes d'équations linéaires peuvent également être résolus avec des matrices. En fait, Sage est très utile pour travailler avec des matrices et permet d'éliminer une grande partie de la lassitude généralement associée au calcul matriciel. Ce point a été abordé dans la section sur les matrices (section 1.5), page 19.

Machine Translated by Google

1.8.4. Systèmes d'équations non linéaires. Si les équations

linéaires sont très faciles à résoudre à l'aide de matrices, le cas non linéaire est généralement beaucoup plus complexe. Commençons par une seule équation, mais très non linéaire. Essayons de la résoudre.

$$6\ 5\ 4\ 3\ 2 - 21x + 175x - 735x + 1624x - 1764x + 720 = 0\ x$$

ce qui peut être fait en

résolvant(x^6 - 21*x^5 + 175*x^4 - 735*x^3 + 1624*x^2 - 1764*x + 720 == 0, x) ce qui donne [x == 5, x == 6, x == 4, x ==

2, x == 3, x ==

1]

Il s'agit d'une liste de six réponses, car ce polynôme de degré six possède six racines. La lecture était facile cette fois-ci, mais vous pouvez également accéder à chaque réponse une par une. À la place, nous saisissons answer = solve(x^6 - 21*x^5 +

175*x^4 - 735*x^3 + 1624*x^2 - 1764*x + 720 == 0, x), puis nous saisissons print answer[0] ou print answer[1] pour obtenir la première

ou la deuxième entrée. Pour obtenir la cinquième entrée de cette liste, nous saisissons print answer[4]. En effet, Sage est basé sur Python, et Python numérote ses listes à partir de 0 et non de 1. Il y a des raisons à cela, basées sur des langages informatiques très anciens.

Considérons maintenant ce problème, suggéré par le professeur Jason Grout, de l'Université Drake. Pour le résoudre,

$$p + q = 9\ qy + px = -6\ qy2 + px2 = 24\ p = 1$$

nous taperions

var('pq y') eq1 =

p+q == 9 eq2 = q*y +

p*x == -6 eq3 = q*y^2 + p*x^2 ==

24 eq4 = p == 1 solve([eq1, eq2, eq3,

eq4], p, q, x, y)

qui produit

[[p== 1, q== 8, x == -4/3*sqrt(10) - 2/3, y == 1/6*sqrt(2)*sqrt(5) -2/3], [p== 1, q== 8, x == 4/3*sqrt(10) - 2/3, y == -1/6*sqrt(2)*sqrt(5) - 2/3]]

Comme vous pouvez le voir, il s'agit d'une liste de listes, toujours avec crochets et virgules. C'est clairement un fouillis très difficile à déchiffrer. En utilisant la technique apprise lors de l'analyse du polynôme de degré six ci-dessus, nous remplaçons la dernière ligne par

Machine Translated by Google

réponse=résoudre([eq1, eq2, eq3, eq4], p, q, x, y)

Maintenant, nous pouvons taper print answer[0] print answer[1] et nous obtenons [p==1, q==8, x== -4/3*sqrt(10) - 2/3, y== 1/6*sqrt(2)*sqrt(5) - 2/3] [p==1, q==8, x== 4/3*sqrt(10) - 2/3, y== -1/6*sqrt(2)*sqrt(5) - 2/3]

C'est la façon dont Sage vous le dit :

Solution n°1 : p = 1, q = 8, x = − $\frac{4}{3}\sqrt{10} - \frac{2}{3}$, et = $\frac{\sqrt{10}}{6} - \frac{2}{3}$

Solution n°2 : p = 1, q = 8, x = $\frac{4}{3}\sqrt{10} - \frac{2}{3}$, y = − $\frac{\sqrt{10}}{6} - \frac{2}{3}$

Puisqu'il n'y a que deux réponses, nous ne pouvons pas en demander une troisième. En fait, si nous saisissons print answer[2], nous obtenons Traceback (cliquez à gauche de ce bloc pour afficher Traceback).

...

IndexError : index de liste hors plage, ce qui est la façon dont Sage vous indique qu'il vous a déjà donné toutes les réponses à ce problème : la liste ne contient pas de 3e élément.

Essayons maintenant d'intersecter l'hyperbole xx 2/4 + y ^{2} 2 − et = 1 avec l'ellipse 2/3 = 1. Nous tapons var('y') solve([x^2-y^2==1, (x^2)/4+(y^2)/3==1], x, y) et obtenons

[x == -4/7*sqrt(7), y == -3/7*sqrt(7)], [x == -4/7*sqrt(7), y == 3/7*sqrt(7)], [x == 4/7*sqrt(7), y == -3/7*sqrt(7)], [x == 4/7*sqrt(7), y == 3/7*sqrt(7)]] qui est illisible. Encore une fois, nous modifions la commande pour qu'elle soit answer=solve([x^2-y^2==1, (x^2)/4+(y^2)/3==1], x, y) puis nous tapons print answer[0] print answer[1] print answer[2] print answer[3] print answer[4] et cela produit quatre réponses et un message d'erreur

[x == -4/7*sqrt(7), y == -3/7*sqrt(7)] [x == -4/7*sqrt(7), y == 3/7*sqrt(7)] [x == 4/7*sqrt(7), y == -3/7*sqrt(7)] [x == 4/7*sqrt(7), y == 3/7*sqrt(7)]

Traceback (cliquez à gauche de ce bloc pour traceback)

...

Machine Translated by Google

IndexError : l'index de la liste est hors de portée,
ce qui est la façon dont Sage vous le dit

Solution n°1 : x = $\frac{4}{7}\sqrt{7}$ et = $\frac{3}{7}\sqrt{7}$

Solution n°2 : x = $\frac{4}{7}\sqrt{7}$, y = − $\frac{3}{7}\sqrt{7}$

Solution n°3 : x = − $\frac{4}{7}\sqrt{7}$ et = $\frac{3}{7}\sqrt{7}$

Solution n° 4 : x = − $\frac{4}{7}\sqrt{7}$, y = − $\frac{3}{7}\sqrt{7}$

mais qu'il n'y a pas de « Solution 5 ».

La recherche des racines d'un polynôme (ou des solutions d'une équation) étant une tâche mathématique courante, il existe d'autres moyens d'afficher les solutions. Par exemple, un code affiche chaque racine sur sa propre ligne, et vous pouvez choisir d'afficher uniquement les racines réelles, rationnelles ou entières. Je décrirai cela plus en détail à la section 5.7.10, page 272.

De plus, nous ne sommes pas limités aux polynômes. Nous pouvons
taper solve(log(x^2) == 5/3, x)
et nous recevons en retour
[x == -e^(5/6), x == e^(5/6)], qui satisfont
tous deux à l'équation donnée. Notez que log dans Sage signifie logarithme népérien, comme expliqué à la section 5.

Nous pouvons également faire

résoudre(sin(x+y)==0.5,x) et
obtenir

[x == 1/6*pi - y], mais

cette explication n'est clairement pas exhaustive. Par exemple, si x = 5π/6 − y, alors on a

$$\sin(x + y) = \sin \frac{5\pi}{6} - y + y$$

$$= \text{péché} \frac{5p}{6}$$

$$= 1/2$$

Nombres complexes :

Normalement, on s'attend à ce qu'un polynôme de degré 8 ait 8 racines sur le plan complexe, sauf si certaines sont des racines répétées. Par conséquent, si l'on se demande « quoi + 1 ? », on s'attend certainement à 8 racines. Ce sont les 8. racines huitièmes de −1 dans l'ensemble des nombres complexes. On peut les trouver avec

solve(x^8==-1, x)

sont les racines de x^8,

Machine Translated by Google

qui produit

```
[x == (1/2*I + 1/2)*(-1)^(1/8)*sqrt(2), x == I*(-1)^(1/8), x ==
(1/2*I - 1/2)*(-1)^(1/8)*carré(2), x == -(-1)^(1/8), x == -(1/2*I +
1/2)*(-1)^(1/8)*carré(2), x == -I*(-1)^(1/8), x == -(1/2*I -
1/2)*(-1)^(1/8)*carré(2), x == (-1)^(1/8)]
```

nous donnant huit nombres complexes distincts, qui ont tous −1 comme nombre puissance huit. Ceci peut être calculé (à la main) plus lentement avec la méthode de DeMoivre formule, mais pour Sage, c'est un problème facile. Je copie-colle simplement la première racine et l'élever à la puissance huit, comme ci-dessous

```
(        (1/2*I + 1/2)*(-1)^(1/8)*sqrt(2) et Sage renvoie        )^8
```

la réponse −1.

1.8.5. Cas avancés

Pour comprendre pourquoi nous ne demandons pas aux étudiants de premier cycle de mémoriser la formule cubique de Cardan, essayez de taper :

```
résoudre(x^3 + b*x + c==0, x)
```

Cela vous donne la formule de Cardan pour un cubique déprimé monique. Un cubique on dit qu'un polynôme est déprimé si le coefficient quadratique est nul, et monique signifie que le coefficient dominant (ici, le coefficient cubique) est un. La version complète de la formule cubique serait donnée par :

```
où('d')
résoudre(a*x^3 + b*x^2 + c*x + d == 0, x)
```

C'est incroyablement compliqué. Essayez, vous verrez. En fait, c'est tellement compliqué qu'il faudrait avouer qu'il n'est pas utilisable.

Une formule encore plus laide serait la formule générale pour une quartique, ou degré quatre, polynôme. N'importe quelle quadrique peut s'écrire

$$a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0$$

nous pouvons donc demander à Sage de le résoudre avec

```
var('a0 a1 a2 a3 a4')
résoudre( a4*x^4 + a3*x^3 + a2*x^2 + a1*x + a0 == 0, x)
```

et nous obtenons une réponse qui est une formule aux proportions horribles.

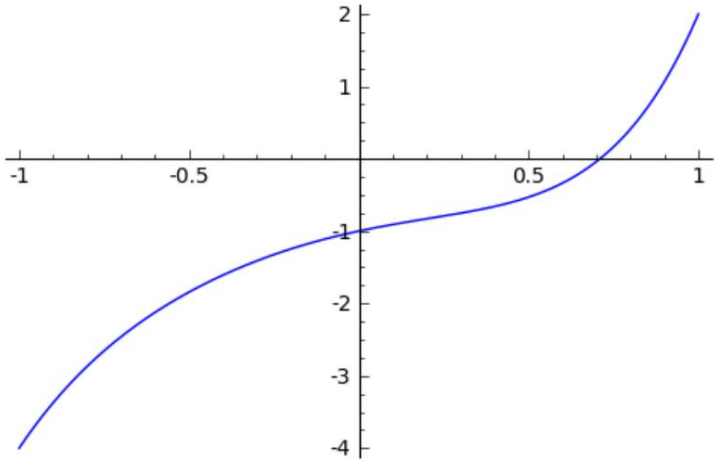
1.9. Utiliser Sage comme solveur numérique

L'ensemble des valeurs x qui font que f(x) = 0 est communément appelé « les racines » de f(x). Supposons que vous ayez besoin de connaître la valeur d'une racine de

$$f(x) = x^{4\,5\,3+x} + x^{2-x} + x - 1$$

un polynôme dont le graphique est donné ci-dessous.

Machine Translated by Google



De plus, supposons que vous soyez intéressé par une racine comprise entre −1 et 1. Cela peut se produire parce que vous avez tracé le graphique (avec ou sans Sage) et constaté la présence d'une racine dans cette région ; ou parce que vous avez constaté que f(−1) = −4 mais f(1) = 2, ce qui fait que f croise clairement l'axe des abscisses entre −1 et 1, bien que nous ne sachions pas encore où. Peut-être qu'un problème dans un manuel vous indique simplement que la racine est comprise entre −1 et 1.

Pour résoudre cette racine, il vous suffit de taper ceci

```
trouver_racine(x^5+x^4+x^3-x^2+x-1,-1,1)
```

et Sage vous dit que

0,71043425578721398

est la racine. Il s'agit d'une approximation numérique, car les polynômes quintiques ont une propriété très particulière. Si vous ne la connaissez pas, demandez à n'importe quel professeur de mathématiques.

Bien qu'il s'agisse d'une approximation, puisque l'ordinateur exécute plusieurs milliards d'instructions par seconde, Sage investit du temps de calcul pour affiner l'approximation. Il s'agit donc d'une approximation fiable avec une précision d'environ 10−16, soit un dixième de quadrillionnième. C'est une bonne approximation, quel que soit le critère.

Entre-temps, il s'avère qu'il n'y a pas de racine entre −1 et 0. On vous l'indique peut-être, ou vous le représentez graphiquement (avec ou sans Sage). Si vous tapez

```
find_root(x^5+x^4+x^3-x^2+x-1,-1,0)
```

alors Sage vous dit

```
RuntimeError : f semble ne pas avoir de zéro dans l'intervalle, ce qui est
```

parfaitement logique, car Sage ne trouve pas de racine dans l'intervalle ! Si vous ignorez où se trouve la racine, vous pouvez saisir

```
find_root(x^5+x^4+x^3-x^2+x-1, -10^12, 10^12) qui demande à Sage
```

de trouver une racine comprise entre plus/moins un billion.

Machine Translated by Google

On peut poser des questions extrêmement complexes. En voici une préférée : $-5 = x^x$ si quelqu'un vous pose une question $x^x = 5$, alors vous pourriez essayer de trouver où x sur x 0. Vous devez taper

```
trouver_racine(x^x-5, 1, 10)
```

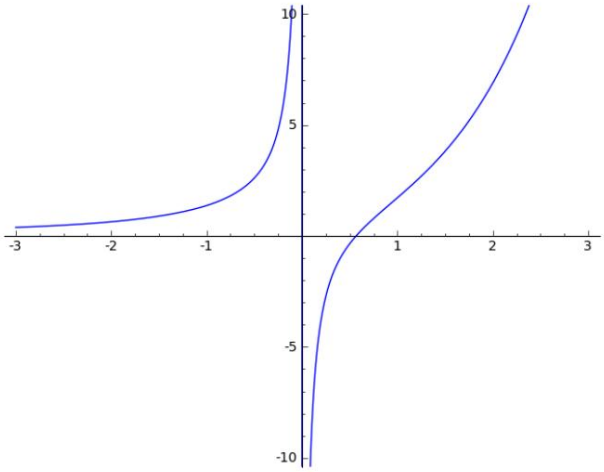
ou de manière équivalente

```
trouver_racine(x^x==5, 1, 10)
```

Une autre solution consiste à trouver où se trouve $x^x = 1/x$ et vous feriez cela en trouvant un la racine de $e^x - 1/x = 0$. Vous pourriez représenter cela graphiquement avec

```
tracé (e^x - 1/x, -3, 3, ymin=-10, ymax=10)
```

obtenir la parcelle



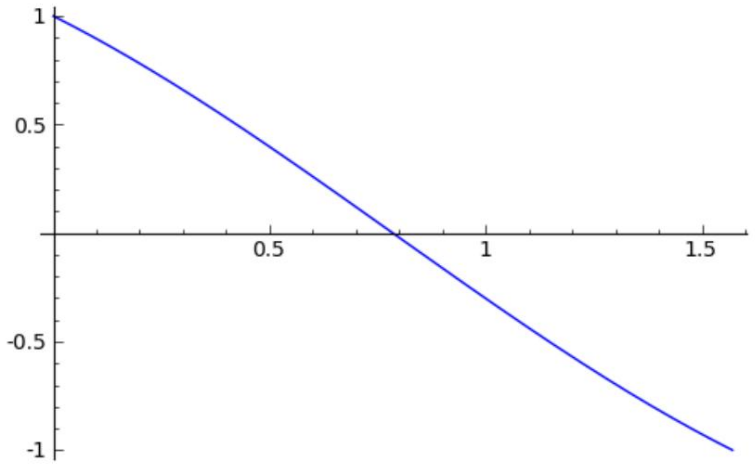
et vous pouvez alors voir que la racine est comprise entre −1 et 1. Cela signifie que nous devons taper

```
trouver_racine(e^x-1/x,-1, 1)
```

aboutira à la découverte que la racine est à x = 0,56714329040979539.

Enfin, une autre méthode courante consiste à trouver où sin x = cos x. Il existe un tel x entre 0 et π/2, comme le montre le graphique. Le graphique est le suivant :

Machine Translated by Google



Ici, vous pouvez demander la racine de (sin x) − (cos x). Vous pouvez également saisir « find_root(cos(x)

== sin(x), 0, pi/2 »).

ce qui est juste une notation plus simple.

Retour à un vieux problème sur les intérêts composés : Plus tôt, à la page 6, nous

avons vu un problème impliquant les intérêts composés et la recherche du nombre de mois nécessaires pour transformer 5 000 $ en 7 000 $. Voyons maintenant comment résoudre ce problème avec Sage en une seule ligne, plutôt que l'approche précédente qui nécessitait que l'humain fasse de l'algèbre.

La première ligne de notre analyse était

$$7000 = 5000(1 + 0{,}045/12)n$$

et traduire cela dans le langage de Sage donne

```
7000 == 5000*(1+0,045/12)^n
```

alors nous devrions taper

```
var('n')
find_root( 7000 == 5000*(1+0.045/12)^n, 0, 10000)
```

Notez que la deuxième ligne signifie que je recherche une valeur de n qui satisfait l'équation, et que cette valeur doit être comprise entre 0 et 10 000. Cela semble raisonnable, car 10 000 mois représentent un peu plus de 833 ans ; cela ne prendra sûrement pas autant de temps. Notez qu'il est essentiel de ne pas inclure de virgule à l'intérieur de 10 000, car la syntaxe de Sage l'interdit, comme nous l'avons vu aux pages 2 et 3. La première commande ci-dessus (var) déclare simplement n comme variable, comme nous l'avons vu à la page 38. Quoi qu'il en soit, nous obtenons la réponse 89,894060933080027 comme précédemment, et apprenons que 90 mois seront nécessaires.

Machine Translated by Google

1.10. Obtenir de l'aide quand vous en avez besoin

Nous avons maintenant appris plusieurs commandes et il est important d'aborder l'utilisation des « fonctionnalités d'aide » intégrées à Sage. Il en existe plusieurs, chacune destinée à un usage spécifique.

La méthode que j'utilise le plus souvent est la complétion par tabulation (décrite pour la première fois en page 4). Elle est utile lorsque l'on ne se souvient pas du nom exact d'une commande. Par exemple, je ne sais peut-être pas si la commande correcte est find root, findroot, find roots, root find, ou peut-être une toute autre commande ! Je tape find, puis j'appuie sur la touche Tabulation (sans cliquer sur « Évaluer »). J'obtiens alors une liste de choix, la plupart sans rapport avec mes intentions. Cependant, l'une d'elles est la commande que je recherche : find root. Par ailleurs, si j'ai atteint find r avant d'appuyer sur la touche Tabulation, il n'y a qu'une seule commande possible, et Sage complète la commande pour moi.

Imaginons maintenant que je me souvienne du nom de la commande – peut-être « solve » – mais pas de sa syntaxe exacte. Je dois alors saisir « solve ? » sans espace entre le point d'interrogation et la commande, puis appuyer sur Tab. Une fenêtre pratique s'ouvre, affichant de nombreuses informations utiles sur la commande. N'hésitez pas à lire la discussion si vous le souhaitez, mais je trouve plus rapide de simplement faire défiler les exemples et de voir comment ils fonctionnent, en reproduisant ce que j'y trouve si nécessaire. Au fait, l'énorme quantité d'informations contenues dans cette fenêtre s'appelle « docstring », abréviation de « documentation string ».

Il existe une autre façon d'utiliser l'opérateur ?. Si vous ajoutez deux points d'interrogation après le nom d'une commande, comme N??, vous verrez l'intégralité du code source (généralement écrit en Python) de la commande.
Dans ce cas, vous verrez le code Python pour la commande N(qui transforme des expressions algébriques exactes en nombres à virgule flottante. Bien sûr, le code est très avancé et nécessiterait beaucoup plus de connaissances de Python que ce que ce livre tentera de transmettre. (Nous allons à peine effleurer la surface au chapitre cinq).

Néanmoins, l'existence de cette fonctionnalité (d'affichage du code source) est importante pour des raisons philosophiques. Nous devons souvent comprendre comment un programme mathématique accomplira sa tâche (autrement dit, comment il fonctionnera en interne) afin de déterminer la meilleure façon de formater et de prétraiter nos données avant de l'utiliser. Ce problème se pose dans de nombreuses situations, et l'accès au code source est essentiel. Il permet également aux idées du programme mathématique d'être utilisées par d'autres programmeurs dans leurs propres programmes. Ainsi, la communauté open source fonctionne comme une sorte de « cerveau » partageant largement ses idées.
De plus, les logiciels mathématiques comportent inévitablement des bugs.
Un collègue a découvert un bug dans Mathematica et en a informé l'entreprise par e-mail. Ils ont reconnu le bug, mais trois ans plus tard, il n'est toujours pas corrigé. Dans Sage, vous pouvez simplement corriger le bug vous-même et soumettre votre demande.

Machine Translated by Google

Le changement est soumis à l'évaluation par les pairs. Si le changement est jugé utile et valable, il sera appliqué. Parfois, le processus complet peut ne prendre que six semaines.

Cela soulève maintenant un point important : vous devez connaître le nom de la commande pour utiliser l'une de ces trois premières méthodes pour obtenir de l'aide.
Que faire si vous ne vous souvenez plus du nom de la commande ? Cela arrive souvent et il existe plusieurs solutions.

Le manuel de référence officiel de Sage est disponible à l'adresse http://www.sagemath.org/doc/reference/, mais ce n'est pas pour les débutants ! Attention, le manuel compte des milliers de pages. C'est comme une encyclopédie : on ne le lit pas d'un bout à l'autre, on y cherche ce dont on a besoin. Heureusement, une fonction « Recherche rapide » est disponible sur le côté gauche.

Les tutoriels sont bien plus utiles pour la plupart des lecteurs. Certains sont organisés autour d'un sujet mathématique, d'autres en fonction d'un public cible. Il existe une collection de tutoriels Sage, dont une liste partielle se trouve à l'annexe C, page 303.

Il existe également un moyen de rechercher toutes les docstrings de toutes les commandes de Sage. Cependant, cela produit généralement un très grand nombre de résultats.
Tapez, par exemple :

search_doc("laplace") et vous

obtiendrez une réponse longue et complexe. Elle n'est pas parfaitement lisible. Voici une ligne de ce résultat : html/en/constructions/

calculus.html:231:<div class="section"

id="laplace-transforms">

La ligne entière fait référence à une page web. Le numéro du milieu, entre les colonnes, correspond à un numéro de ligne : un pour la première ligne, deux pour la deuxième ligne, et ainsi de suite. Les informations après la deuxième colonne sont écrites en HTML. Si vous maîtrisez le HTML, vous pouvez utiliser ces informations.

Jusqu'à présent, la recherche dans la documentation n'a rien révélé d'utile, mais l'information à gauche des deux points est très utile. Il s'agit d'un nom de fichier dans une arborescence de répertoires contenant la documentation de Sage. Sage est généralement utilisé via un serveur, tel que Sage Cell Server ou Sage Cloud. Cependant, il est possible d'effectuer une installation locale, ce qui signifie que vous exécutez Sage entièrement sur votre propre ordinateur, sans communiquer avec un serveur distant. Cette méthode n'est pas recommandée, sauf pour les experts.

Que devons-nous donc faire, nous autres (utilisant Sage via un navigateur web ou Sage Cloud) pour accéder à ces informations ? Nous devons accéder à l'URL suivante, qui contient essentiellement les informations fournies par le document de recherche, mais dans un ordre de mots différent : http://www.sagemath.org/doc/
constructions/calculus.html.

Comme vous pouvez le voir, nous avons remplacé html/en/ par

---

[8] Si vous ne connaissez pas le HTML, vous pourriez envisager de l'apprendre. Le langage HTML est extrêmement utile et simple. Être capable de créer ses propres pages web est une excellente compétence, très recherchée.

Machine Translated by Google

http://www.sagemath.org/doc/ pour construire cette

URL. Cette substitution fonctionnera généralement. À propos, « en » désigne l'anglais, « fr » le français, « de » l'allemand et « ru » le russe, selon les codes de langue à deux lettres utilisés sur Internet.

1.11. Utiliser Sage pour obtenir des produits dérivés

La commande pour la dérivée est simplement diff. Par exemple, pour calculer la dérivée de f(x) = x, diff(x^3 - x, x) ^{3} − x vous devez taper :

pour apprendre la réponse

3*x^2 - 1

Ou pour f(x) = sin(x ^{2}) vous devez taper :

diff(sin(x^2), x)

pour recevoir la réponse

2*x*cos(x^2)

Parfois, vous avez peut-être déjà défini votre propre fonction par g(x)=e^(-10*x),

vous pouvez donc

taper diff(g(x), x) pour obtenir la

dérivée, ce qui est

naturellement

-10*e^(-10*x)

Vous pouvez également

saisir gprime(x) = diff(g, x), ce qui

est silencieux. Vous pouvez alors saisir gprime(2) ou gprime(3) et la fonction fonctionne comme prévu. Elle vous indique les valeurs de g(2) et g(3). Vous pouvez également saisir g(x)=e^(-10*x)

g.derivative()

Nous ne pouvons cependant pas définir une fonction appelée g'(x). En effet, Sage repose sur le langage informatique Python, et l'apostrophe a une signification prédéfinie en Python. C'est regrettable, mais nous pouvons toujours écrire gprime(x) ou g prime(x), ce qui ne nous posera aucun problème.

Sage peut également faire des dérivées très difficiles :

diff(x^x, x) qui

fournit la réponse (log(x) + 1)*x^x

Machine Translated by Google

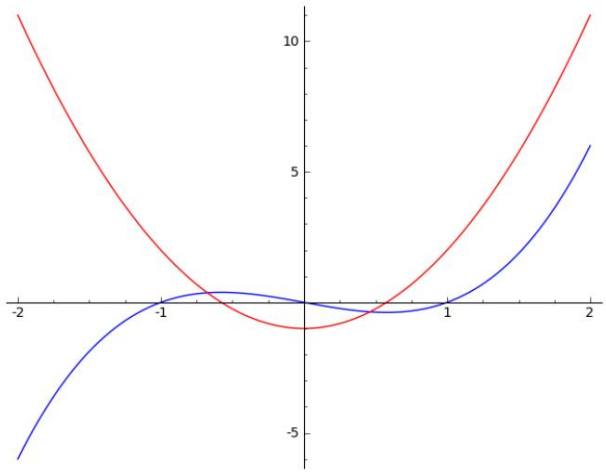
Si vous souhaitez exercer vos compétences en calcul, et peut-être même de manière très intensive, vous pouvez essayer de calculer la dérivée de x x avec un crayon. À propos, il est important de noter que log désigne le logarithme népérien, et non le logarithme décimal, comme expliqué page 5.

1.11.1. Tracer f(x) et f(x) ensemble

L'un de mes morceaux de code préférés dans Sage est f(x)=x^3-x

```
fprime(x)=f(x).derivative() plot( f(x),-2,2,
color='blue') + plot( fprime(x),-2,2, color='red')
```

Je l'utilise assez fréquemment. Ce graphique trace f(x) et f(x) sur le même graphique, avec f(x) en bleu et f(x) en rouge. Vous pouvez facilement modifier la fonction analysée en modifiant la première ligne. Ici, nous analysons $f(x) = x^3 - x$, en rouge. Le tracé est le suivant : $f(x) = 3x^2 - 1$, en bleu, et sa dérivée,



1.11.2. Dérivées d'ordre supérieur

Bien que vous puissiez trouver la dérivée seconde en utilisant diff deux fois, vous pouvez passer directement à la dérivée seconde avec diff(x^3-x, x, 2) pour obtenir la réponse

```
6*x
```

Les dérivées troisièmes ou supérieures ne posent pas non plus de problème. Par exemple, la dérivée troisième peut être trouvée avec

```
diff( x^3-x, x, 3)
```

Pour ceux qui n'ont pas peur de taper plus, vous pouvez le faire

```
dérivée( x^3-x, x, 2)
```

Machine Translated by Google

ce qui est plus lisible que d'écrire diff.

1.12. Utiliser Sage pour calculer des intégrales

La seule chose que vous devez comprendre pour faire des intégrales dans Sage est qu'il existe en réalité trois types d'intégrales qui peuvent être réalisées : une intégrale approximative numérique, une intégrale définie exacte et une intégrale indéfinie.
Les deux premiers types donnent des nombres, mais ils sont trouvés respectivement approximativement ou exactement. Le troisième est la primitive symbolique, ce qui signifie que la réponse est une fonction ; en fait, la réponse est une fonction dont la dérivée est celle que vous avez saisie. Si cela vous semble confus, prenons un exemple simple :

Un exemple d'intégration :

En utilisant une intégrale, comment calculeriez-vous l'aire entre l'axe des x et f(x) = x ^{2} + 1 sur l'intervalle 3 ≤ x ≤ 6 ? Eh bien, cette intégrale serait

$$\int_3^6 x^2 + 1\,dx = \left(\frac{1}{3}x^3 + x\right)\Big|_3^6 = \left(\frac{1}{3}6^3 + 6\right) - \left(\frac{1}{3}3^3 + 3\right) = 72+6-(9+3) = 66$$

et c'est un nombre. Ici, nous l'avons fait analytiquement, en utilisant le calcul différentiel et intégral. C'est une intégrale définie exacte. Elle est définie, car elle se termine par un nombre. C'est exact, car nous n'avons fait aucune approximation.

D'autre part,

$$\int x^2 + 1\,dx = \frac{1}{3}x^3 + x + C$$

est une intégrale indéfinie. Elle produit une fonction, et non un nombre. Bien sûr, on calcule souvent une intégrale indéfinie pour obtenir une intégrale définie ; néanmoins, les intégrales définies et indéfinies sont deux catégories distinctes. Si l'on veut une fonction, on utilise l'intégrale indéfinie ; si l'on veut un nombre, on utilise l'intégrale définie.

L'intégrale indéfinie : L'intégrale

indéfinie est la plus simple. Par exemple, si vous voulez savoir

$$\int x \text{ péché}(x^2)dx$$

vous taperiez simplement

integral(x*sin(x^2), x) et vous

apprendriez alors que la réponse est

-1/2*cos(x^2)

De même, si vous vouliez savoir

$$\int \frac{x}{x^2 + 1}dx$$

Machine Translated by Google

vous taperiez simplement

integral(x/(x^2 + 1), x) et vous

apprendriez alors que la réponse est

1/2*log(x^2 + 1)

En savoir plus sur le +C :

2 Considérons l'ensemble des fonctions dont la dérivée est 3x . Il existe une infinité de 3 + √ π et ainsi de suite.

nombre d'entre eux, y compris $x - 8$, $x^3 + 5$, $x^3 + 81$, x^3 —

C'est pourquoi les professeurs de calcul insistent pour que vous écriviez

$$2\ 3\ 3x\ dx = x + C$$

lors du calcul d'une intégrale indéfinie.

Cependant, Sage ne connaît pas +C. Il y a une bonne raison à cela : il n'est pas facile de créer une fonction Python représentant une famille infinie de fonctions. Par exemple, si je définis f(x) = 3x dx, que doit dire Python lorsque je demande f(2) ? Comment Sage peut-il savoir à quel membre de la famille je fais référence ?

Dans cette optique, vous devez vous-même suivre les +C. Leur gestion correcte est une tâche essentielle dans l'un des projets (le projet sur les projectiles balistiques), section 2.5, page 79.

L'intégrale définie :

Alternativement, nous pourrions mettre une borne inférieure de 0 et une borne supérieure de 1 sur cette intégrale, puis obtenir

$$\int_0^1 \frac{x}{2 + 1\ x}\, dx$$

Pour que Sage calcule cette intégrale définie pour nous, nous tapons integral(x/(x^2 + 1), x, 0, 1)

et nous sommes la réponse

1/2*log(2)

Cependant, n'oubliez pas que « log » signifie le logarithme naturel, et non le logarithme décimal, comme expliqué à la page 5.

Intégrales impossibles : Le mot

« impossible » est un terme dangereux en mathématiques. Certaines intégrales sont notoirement impossibles. Les deux plus célèbres sont l'intégrale de Fresnel.

$$\int_0^x \sin \frac{\pi t2}{2}\, dt$$

ce qui est important en optique, et l'intégrale gaussienne

$$\int_0^{et} \frac{2}{\sqrt{p}}\, et^{(-x^2)}\, dx$$

Machine Translated by Google

ce qui est d'une importance cruciale en probabilités. Qu'entendons-nous par « impossible » ici ? Il n'existe aucune fonction, constituée d'additions, de soustractions, de multiplications, de divisions, de racines, d'exposants, de logarithmes, de fonctions trigonométriques et de fonctions trigonométriques inverses9, qui aurait pour dérivée soit la fonction de Fresnel, soit la fonction de Gauss.

Or, en mathématiques appliquées, il est nécessaire de calculer ces intégrales ! L'intégrale de Gauss, en particulier, est extrêmement importante en statistique. On obtient ainsi l'intégrale définie en divisant l'intervalle en de très nombreuses petites régions. De nombreux rectangles et trapèzes étroits peuvent être utilisés pour approximer la courbe. Les aires numériques de ces rectangles et trapèzes sont additionnées pour obtenir une approximation numérique de l'intégrale. (D'où le nom d'« intégration numérique »). Chacun de ces rectangles ou trapèzes peut comporter une légère erreur, mais en additionnant toutes les aires, on espère que les erreurs s'annulent légèrement. Vous avez probablement déjà utilisé cette technique pendant vos cours de calcul, ou vous la ferez plus tard.

Des méthodes encore plus sophistiquées d'intégration numérique incluent l'utilisation de polynômes au lieu de trapèzes, et Sage utilise des techniques extraordinairement complexes pour obtenir la meilleure approximation possible. Ces approximations sont un sujet ancien et profond, remontant à la règle de Simpson, où Simpson utilisait de minuscules paraboles au lieu de trapèzes étroits. Cela lui permettait d'obtenir des approximations assez précises avec beaucoup moins de calculs que la règle des trapèzes. Étant donné qu'il travaillait dans la première moitié du XVIIIe siècle, c'est un véritable exploit.

Ce qui précède peut être un peu déroutant, mais l'intérêt de Sage est qu'il s'occupera de ces détails pour vous, à condition que vous sachiez ce que vous attendez de Sage.

Prenons un exemple concret. Considérons la fonction suivante :

$$f(x) = e^{3-x} \text{ péché}(x^2)$$

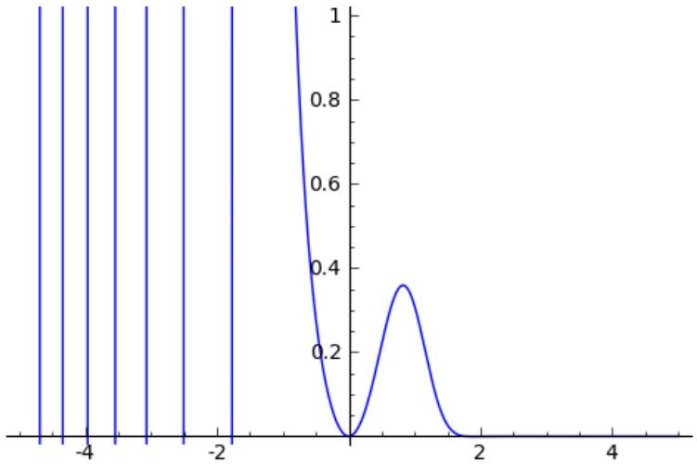
C'est une fonction continue et facile à comprendre. On peut la représenter graphiquement avec les commandes

```
plot( exp( -x^3 ) * sin(x^2), -5, 5, ymin = 0, ymax=1 )
```

Nous obtenons le tracé suivant

9Nous devrions également inclure les cousins hyperboliques des fonctions trigonométriques et fonctions trigonométriques inverses, dont on vous a peut-être ou non parlé.

Machine Translated by Google



Cette fonction f(x) semble intéressante, mais calculer son intégrale est assez frustrant. Je n'arrive pas à résoudre cette intégrale avec mon crayon, et je suis prêt à parier que vous non plus. Essayez si vous le souhaitez.

Bien qu'il existe une fonction g(x), quelque part, qui a f(x) = e sin(x l'analyse $^{3-x}$ ^{2}) comme son dérivé — une conséquence logique de quelque ad-réelle avancée et le fait que f soit continue — les aspects pratiques de la question sont qu'il n'y a aucun moyen d'écrire g(x) sous forme de formule. Pour être vraiment précis, il n'y a pas de fonction g(x), construite à partir d'additions, de soustractions, de multiplications, de divisions, de racines, d'exposants, de logarithmes, de fonctions trigonométriques et de fonctions trigonométriques inverses (ainsi que de leurs cousines hyperboliques, que vous avez peut-être ou non apprises) qui aura g (x) = f(x).

En d'autres termes, f(x) = e $^{3-x}$ péché(x^{2}) est intégrable, mais l'intégrale ne peut pas s'écrire en termes de fonctions courantes (et moins courantes). Cependant, on pourrait produire ag(x) formé d'une série infinie en notation Big-Sigma, ce qui donnerait g(x) = f(x).

Revenons maintenant à Sage, puisque

$$\text{et}^{-x^{3}} \text{ péché}(x^{2})dx$$

n'a pas une belle expression, alors quand vous tapez

```
intégrale( exp(-x^3)*sin(x^2),x)
```

le Sage ne répond pas par une réponse, mais par

```
intégrer(e^(-x^3)*sin(x^2), x)
```

Ce qui représente un aveu de défaite. Mais n'ayez crainte, nous pouvons calculer des réponses numériques approximatives avec une très grande précision, et nous le découvrirons bientôt.

Machine Translated by Google

Intégration numérique dans Sage :

Une intégrale néfaste, célèbre parmi les professeurs de calcul, est

t 20e t dt

où 20 pourrait être n'importe quel nombre relativement grand. Ce point est abordé lors de l'explication de la fonction gamma d'Euler, mais nous n'y prêtons pas attention pour le moment. Dans tous les cas, nous pourrions simplement taper integral((t^20)*(e^t), t) mais nous obtenons alors (t^20 -

20*t^19 + 380*t^18 - 6840*t^17 + 116280*t^16 -

1860480*t^15 +27907200*t^14 -

390700800*t^13 + 5079110400*t^12 - 60949324800*t^11 + 670442572800*t^10 - 6704425728000*t^9 + 60339831552000*t^8 - 482718652416000*t^7 + 3379030566912000*t^6 - 20274183401472000*t^5 + 101370917007360000*t^4 - 405483668029440000*t^3 + 1216451004088320000*t^2 - 2432902008176640000*t + 2432902008176640000)*e^t

Ce qui est tellement grand qu'il est difficile d'en saisir la signification. Si, au contraire, nous avions certaines limites, comme celles permettant de calculer

3
t 20e t dt
2

alors nous taperions

integral((t^20)*(e^t), t, 2, 3) et obtiendrions à la

place

-329257482363600896*e^2 + 121127059051462881*e^3

Ce qui nous amène à une réponse ! Si vous souhaitez vraiment une réponse numérique, utilisez notre vieille astuce : la commande N(), et tapez N(integral((t^20)*(e^t), t, 2, 3)) pour

obtenir le nombre.

8.79797452800000e9

et cela signifie 8,79797452800000 × 109. Cela met en évidence les différences et les similitudes entre les différents types de réponses. Si vous avez oublié N(), consultez la page 3.

Ce nombre que nous venons de calculer ressemble à un entier, car il se termine par 528 lorsqu'on effectue la partie 109. Mais lorsque nous examinons la réponse exacte, nous constatons que non, ce n'est pas un entier. Nous le savons parce que sont impliqués, et e est un nombre et 2 3 et e irrationnel. Lors de chaque calcul, il est essentiel de se rappeler de distinguer une approximation de ce qui est exact. Un nombre irrationnel peut parfois être exprimé exactement (comme), mais c'est l'exception et non la règle. Nous venons de le faire ici, en utilisant e. Quoi qu'il revenons donc au sujet qui 2 3 et e en soit, ce paragraphe n'a rien à voir avec Sage, nous nous occupe.

Machine Translated by Google

En parlant d'approximations, on peut aussi passer très rapidement à l'intégrale numérique. Je suis sûr que nous pouvons tous les deux calculer

$$\int_0^1 3 - x\,dx = x \qquad \frac{1}{4}x - \frac{1}{2}x \qquad \Big|_0^1 = \frac{1}{4}1^4 - \frac{1}{2}1^2 - \frac{1}{4}0^4 - \frac{1}{2}0^2 = \frac{1}{4} - \frac{1}{2} = -\frac{1}{4}$$

Cependant, si vous souhaitez demander à Sage de calculer numériquement cette intégrale, vous devez saisir la

commande « digital_integral(x^3 - x, 0, 1) » qui

affichera

(-0,24999999999999997, 2,775557561562891e-15), où le premier

nombre correspond à la meilleure estimation de Sage pour la réponse, tandis que le second représente l'incertitude. Dans ce cas, l'incertitude est de 2,77 quadrillionièmes, ce qui est très impressionnant.

Bien sûr, Sage peut également réaliser cette intégrale, grâce aux commandes que nous avons apprises plus tôt. Ce qui est intéressant avec l'intégration numérique, ce sont les cas où l'intégrale indéfinie est impossible à réaliser, car, comme l'intégrale de Fresnel ci-dessus, elle est inscriptible, mais on peut trouver de bonnes estimations numériques. Reprenons.

$$\int_1^3 e^{3-x} \text{ péché}(x^2)\,dx$$

mais maintenant avec des bornes, afin de pouvoir le faire numériquement. Nous taperions alors

intégrale_numérique(exp(-x^3)*sin(x^2),1,3)
et cela revient en arrière

(0,077997011262087815, 8,6840496851995037e-16) avec (comme

précédemment) le premier nombre comme réponse et le second comme incertitude. Dans ce cas, l'incertitude est de 868 quintillionièmes, ce qui est très impressionnant. Autrement dit, l'incertitude est de (1/1152) trillionièmes.

Intégration par fractions partielles : Une chose

qui est très importante dans les problèmes qui découlent des équations différentielles et du calcul II est la question de l'intégration par fractions partielles.
Si vous souhaitez connaître la répartition des fractions partielles de

$$\frac{x^3 - x}{x^2 + 5x + 6}$$

alors vous devez faire

f(x)=(x^3-x)/(x^2+5*x+6) suivi de

f.partial_fraction()

et obtenir la réponse x - 6/(x +

2) + 24/(x + 3) - 5

Machine Translated by Google

ce qui est correct. En fait, nous venons de calculer

$$\frac{3x - x}{2x + 5x + 6} = x - \frac{6}{x + 2} + \frac{24}{x + 3} - 5$$

Cependant, vous pouvez également faire le raccourci et demander à la place integral((x^3-x)/(x^2+5*x+6), x) ce qui donnera

1/2*x^2 - 5*x - 6*log(x + 2) + 24*log(x + 3)

Intégrales impropres :

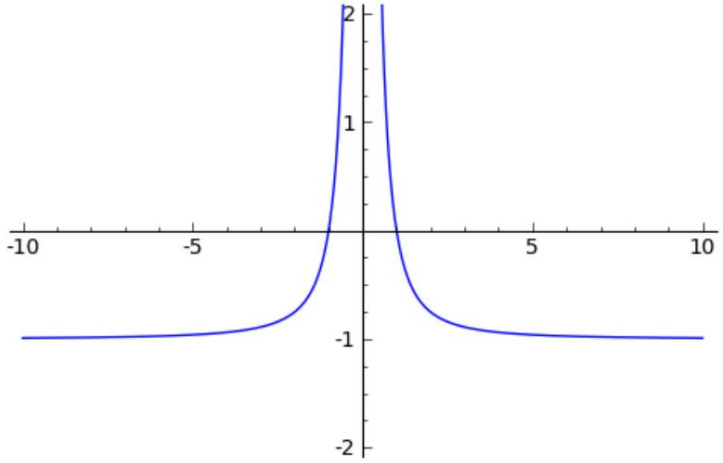
Bien que l'intégrale suivante soit mathématiquement valide

$$-1 + \frac{1}{2x} \, dx = -x - \frac{1}{x} + C$$

et le côté droit peut être évalué à x = −1 et x = 1, il n'y a pas de réponse finie à

$$\int_{-1}^{1} -1 + \frac{1}{2x} \quad \text{diagnostic}$$

car cette intégrale impropre est divergente. Le graphique illustre la raison de ce phénomène. Voici la courbe de −1 + 1/x2.



généré par

tracé(-1+1/x^2, -10, 10, ymin=-2, ymax=2)

Pourtant, Sage le sait et dira

-x - 1/x en

réponse à

l'intégrale (-1+1/x^2,x) mais

répondra à la place ValueError :

L'intégrale est divergente. à la demande

Machine Translated by Google

intégrale(-1+1/x^2,x,-1,1)

Si vous souhaitez une preuve du fait que l'intégrale ci-dessus est divergente, considérez que

$$\int_{-1}^{-1} -1 + \frac{1}{2x}\,dx = \int_{-1}^{0} -1 + \frac{1}{2x}\,dx + \int_{0}^{1} -1 + \frac{1}{2x}$$ diagnostic

Considérons l'intégrale ci-dessus, de 0 à 1, au moyen du calcul cette intégrale de a à 1, où a sera légèrement supérieur à 0.

$$\int_{un}^{1} -1 + \frac{1}{2x} = -x - \frac{1}{x}\Big|_{un}^{1}$$

$$= -1 - \frac{1}{1} - -un - \frac{1}{un}$$

$$1 = a - 2 + \frac{}{un}$$

Comme vous pouvez le voir, lorsque a s'approche de 0 à partir des nombres réels positifs, La valeur de a − 2 + 1/a devient énorme. Plus précisément, on peut écrire

$$\lim_{a\to 0+} a - 2 + \frac{1}{un} = \infty$$

mais cela signifie aussi

$$\lim_{a\to 0+} \int_{un}^{1} -1 + \frac{1}{2x}\,dx = \infty$$

et, il s'avère que la même chose est vraie pour l'intégrale de -1 à 0 également. Par conséquent, l'intégrale originale est égale à ∞ + ∞ = ∞.

Plus d'intégrales impropres :

Un autre type d'intégrale impropre est

$$\int_{2}^{\infty} \frac{1}{2x}$$ diagnostic

et la façon d'écrire cela dans Sage est de taper integral(1/

x^2,x,2,oo) auquel Sage répond

1/2, ou un autre exemple est

$$\int_{-\infty}^{\infty} et^{2-x}$$ diagnostic

qui dans Sage est

intégrale(exp(-x^2),x,-oo,oo) et donne la

réponse correcte de $\sqrt{\pi}$.

L'idée est simplement que oo est un code spécial pour « infini ». Il s'agit d'un code à deux lettres o, et l'idée est que si vous plissez les yeux, oo ressemble à un symbole de l'infini.

Machine Translated by Google

La fonction erf et les intégrales :

Parmi les mathématiciens et surtout parmi les statisticiens, l'intégrale

$$\int_0^{et} \frac{2}{\sqrt{p}} et(-x^2)\, dx = erf(y)$$

est extrêmement importante. Il s'agit de l'intégrale de Gauss dont je vous ai parlé plus tôt ; j'ai notamment précisé que cette intégrale ne peut être trouvée à l'aide des fonctions mathématiques élémentaires. Son importance est telle qu'elle porte un nom, et ce nom est erf. Bien que facile à prononcer, erf signifie « fonction d'erreur ».

Donc si vous tapez dans Sage

intégrale((2/sqrt(pi)) * exp(-x^2), x, -oo, 2) il répondra (sqrt(pi)*erf(2) +

sqrt(pi))/sqrt(pi) ce

qui est correct car

$$\int_{-\infty}^{2} \frac{2}{\sqrt{p}} et(-x^2)\, dx = \int_{-\infty}^{0} \frac{2}{\sqrt{p}} et(-x^2)\, dx + \int_{0}^{2} \frac{2}{\sqrt{p}} et(-x^2)\, dx$$

$$= \sqrt{\pi}\sqrt{\pi} \qquad \frac{\sqrt{\pi}}{\sqrt{\pi}}\sqrt{\pi}erf(2) + \sqrt{\pi} = 1 + erf(2) = (1 + erf(2))$$

Vous pourriez peut-être trouver que 1 + erf(2) est une réponse plus compacte et plus simple. Je serais plutôt d'accord. Je ne comprends pas pourquoi Sage ne précise pas davantage sa réponse finale.

Dans tous les cas, le problème est qu'il existe certaines fonctions qui n'ont pas d'intégrales inscriptibles à l'aide de fonctions élémentaires, mais que Sage calcule quand même à l'aide de la fonction erf.

La commande Assume et les intégrales. Cet exemple a

été identifié par Joseph Loeffler, qui l'a porté à mon attention. Disons que vous vouliez évaluer l'intégrale

$$\int_1^x \frac{3t + 30}{t} dt$$

et que vous avez tapé, en conséquence, les commandes Sage : var("t")

integrate((t^3+30)/t, t, 1, x)

(Au fait, « intégrer » est synonyme d'intégrale.) En fin de compte, vous recevriez une réponse contenant un message d'erreur volumineux. Il faut se rappeler que dans Sage, les dernières lignes d'un message d'erreur sont les plus importantes. Voici ces dernières : ValueError : Le calcul a échoué car Maxima a demandé des contraintes supplémentaires ;

l'utilisation de la commande « assume » avant l'évaluation de l'intégrale *peut* être utile (exemple de syntaxe valide : « assume(x-1>0) », voir « assume ? » pour plus de détails).

Machine Translated by Google

Est-ce que x-1 est positif, négatif ou nul ?

La plainte adressée à Sage par Maxima est justifiée car la fonction f(t) = (t + 30)/t explosera à t = 0, car elle tente de diviser par zéro. L'intégrale est donc impropre si 0 est compris entre 1 et x. Cependant, si x − 1 > 0, ou en clair, si x > 1, alors cela ne se produira clairement pas. Sage est donc parfaitement justifié de nous demander d'énoncer explicitement cette hypothèse.

Suivons la procédure recommandée par Sage en ajoutant la commande assume avant notre commande integrate. Nous avons maintenant var("t") assume(x>1)

integrate((t^3+30)/
t, t, 1, x) qui produit la réponse 1/3*x^3 +
30*log(x) - 1/3 comme prévu.

La commande assume est suffisamment rare pour que de nombreux utilisateurs expérimentés de Sage ignorent son existence. Il est peu probable qu'elle apparaisse dans votre travail. Cependant, nous en verrons un autre exemple à la page 184, dans une sommation, et à la page 200, dans une intégrale liée à une transformée de Laplace.

1.13. Partager les résultats de votre travail

Dans cette section, je vais vous présenter sept méthodes efficaces pour partager les résultats de vos travaux dans Sage. Elles peuvent être utiles pour la collaboration scientifique, la soumission d'un travail à un enseignant, l'aide d'un ami ou simplement l'explication d'un concept mathématique. Mes deux méthodes préférées sont accessibles via le bouton « Partager » sur l'écran de Sage Cell Server.

Lorsque vous travaillez sur Sage Cell Server, cliquez sur le bouton « Partager ». Vous aurez alors accès à des options permettant de créer un lien temporaire court, un lien permanent et un code-barres 2D, également appelé « QR code ». Le lien permanent est mon préféré. Il génère une URL volumineuse et vous prouve que la nouvelle URL fonctionne en rechargeant la page avec cette URL. Sélectionnez ensuite cette URL dans votre navigateur web, à l'endroit où vous saisissez habituellement https://sagecell.sagemath.org/, en veillant à bien la sélectionner entièrement. Cette URL peut désormais être utilisée dans vos documents, e-mails, chats ou même sur Facebook. L'URL encode le code Sage dans la fenêtre ; votre code n'est donc stocké nulle part et ne peut donc être perdu, supprimé ou détruit. Tant que vous conservez une URL complète et non modifiée, elle fonctionnera toujours, même des années plus tard.

Le permalien est long et disgracieux. Cela n'a rien d'étonnant, car il encode l'intégralité de votre bloc de code Sage. En réalité, il est si long qu'il peut poser problème dans certains navigateurs ou processeurs de messagerie. Dans la plupart des cas, il dépasse 140 caractères et ne peut donc pas être envoyé.

Machine Translated by Google

Un SMS ou un tweet. Le lien temporaire court, en revanche, est plus court et moins intimidant pour un humain. Plus court, il est moins susceptible d'être tronqué lors de son envoi par divers moyens. Cependant, il ne contient pas une copie complète de votre code. Celui-ci est stocké sur le serveur et l'URL le récupère. La durée de fonctionnement du lien temporaire court n'est pas garantie. Peut-être un jour, un an, voire une éternité, voire une heure seulement. D'après mon expérience, ils fonctionnent souvent plusieurs jours après leur création. Globalement, si la longueur et la laideur ne posent pas de problème, il est préférable d'utiliser systématiquement le lien permanent et de ne jamais utiliser le lien court.

Voici mes deux méthodes préférées pour partager du code Sage. La troisième méthode que j'aimerais partager avec vous est le grand code-barres 2D généré lorsque vous cliquez sur « Partager ». Il encode le lien permanent et fonctionne donc avec tous les smartphones compatibles. La plupart des téléphones nécessitent l'installation d'une application pour pouvoir lire ces codes-barres, parfois appelés « codes QR », « codes à réponse rapide » ou « codes-barres matriciels ». Sur mon téléphone, l'application s'appelait « Lecteur de codes QR » et était gratuite. Si vous photographiez le code-barres avec l'une de ces applications et l'appareil photo de votre smartphone, un navigateur web devrait s'ouvrir et afficher les mêmes résultats que le lien permanent. Vous n'avez même pas besoin de cliquer sur « Évaluer » ! Voici un exemple de code-barres 2D :

La quatrième méthode consiste à créer un tracé (comme nous l'avons vu à la section 1.4, page 9). Si vous souhaitez inclure cette image dans un document, quel qu'il soit, ou l'envoyer par courriel, enregistrez-la simplement depuis votre navigateur web, comme vous le feriez pour n'importe quelle autre image. Dans la plupart des navigateurs, il suffit de faire un clic droit, puis de sélectionner « Enregistrer l'image sous… », et de saisir un nom de fichier à l'emplacement approprié. Cette image sera alors dans le dossier

---

10Mais vous avez vraiment besoin de l'application. Sinon, vous n'obtiendrez qu'une photo de QR Code, ce qui est totalement ennuyeux.

Machine Translated by Google

Format « Portable Network Graphic » ou *.png. Compatible avec presque tous les programmes permettant l'importation d'images.

La cinquième méthode consiste simplement à sélectionner le code Sage, à le copier, puis à le coller dans un e-mail. Parfois, je le colle dans Sage Cloud ou dans le logiciel de gestion de cours de mon université, D2L. Envoyer le code par e-mail permet de gagner du temps et de le publier facilement.

La sixième méthode consiste à imprimer une capture d'écran du serveur cellulaire Sage. En faisant défiler et en redimensionnant légèrement, on peut souvent afficher simultanément l'intégralité du bloc de code Sage et sa sortie. Vous pouvez ensuite utiliser la commande d'impression de votre navigateur (souvent située dans le menu « Fichier ») pour imprimer la page web comme vous le feriez pour n'importe quelle autre page web. (La commande d'impression du navigateur se trouve souvent dans le menu « Fichier ».) C'est souvent ainsi que je demande à mes étudiants de soumettre un exercice réalisé dans Sage. Enfin, la septième méthode consiste à suivre les étapes d'impression d'une capture d'écran du Sage Cell Server et, au lieu de l'imprimer au format papier, à sélectionner « Enregistrer au format PDF » dans le menu d'impression du navigateur web. Ce PDF est quasiment identique à celui que vous obtiendriez en l'imprimant, à la différence qu'il s'agit d'une copie électronique et non d'une copie papier.

La collaboration scientifique est l'un des grands plaisirs du travail dans une discipline STEM[11]. J'espère que vous aurez souvent l'occasion de partager non seulement vos idées, mais aussi les détails mathématiques et quantitatifs de vos travaux avec d'autres. Sage ayant été développé par des centaines de personnes réparties dans le monde entier, il n'est pas surprenant qu'il intègre de nombreuses fonctionnalités de partage. Sage Cloud offre encore plus de fonctionnalités, notamment des projets organisés, le partage sur invitation uniquement et le chat en direct. Vous trouverez plus d'informations sur le site web de Sage Cloud.

11STEM : Sciences, Technologie, Ingénierie et Mathématiques.

Machine Translated by Google

Chapitre 2

Projets amusants utilisant Sage

Voici quelques exemples détaillés d'utilisation de Sage pour explorer un problème non trivial dans un domaine autre que le calcul formel. Nous présentons un exemple de chaque discipline. L'idée est que chaque projet nécessite environ un week-end prolongé pour être réalisé par un étudiant.

Certains de ces projets peuvent être explorés à l'aide des informations présentées au chapitre 1 et ne nécessitent aucune connaissance supplémentaire de Sage. Certains projets nécessitent un peu plus de connaissances, mais cela est clairement indiqué dans les premiers paragraphes de la description du projet.

Microéconomie : Modélisation de l'effet du prix de vente sur les ventes de glaces, en termes de fonction de demande, de fonction de revenu, de fonction de coût et de fonction de profit

Biologie : Modélisation des artères obstruées et de la loi de Poiseuille, ainsi que des effets catastrophiques d'un blocage, même faible, sur la circulation sanguine. Aucune connaissance en biologie n'est requise.

Optimisation industrielle : Résolution d'un système linéaire d'inéquations pour optimiser l'acheminement de la taconite de sept mines vers quatre ports, tout en minimisant les coûts de transport et en respectant les capacités.

Chimie : Une méthode pour équilibrer des réactions chimiques plutôt compliquées en utilisant le RREF des matrices

Physique : Résolution de problèmes de mouvement de projectiles balistiques dans Sage, notamment tir de contre-batterie

Cryptologie : Utilisation de la méthode $p - 1$ de Pollard pour factoriser des entiers, pour essayer de casser le cryptosystème RSA

Mathématiques pures : Utilisation du calcul différentiel pour analyser les caractéristiques d'un polynôme quintique, afin de trouver la fenêtre graphique optimale. Génie électrique : Un mini-projet sur la génération de tracés vectoriels de champ électrique pour le champ généré par quatre particules chargées dans le plan de coordonnées.

J'écris également un livre pour un cours appelé « Mathématiques finies » aux États-Unis et « Méthodes quantitatives » au Royaume-Uni. Ce cours s'adresse aux étudiants en commerce, comptabilité, finance, économie, marketing, etc.

Machine Translated by Google

gestion, etc. Un sujet classique de ce cours est l'analyse entrées-sorties de Leontief, méthode de modélisation des interactions entre les différents secteurs de l'économie. Ce sujet pourrait être abordé en macroéconomie, en complément du projet de microéconomie présenté ici. Bien qu'elle nécessite quelques notions d'algèbre matricielle, l'analyse de Leontief est parfaitement adaptée à Sage et relativement facile à prendre en main. Vous trouverez une version de ce projet sur ma page web : http://www.gregorybard.com/

2.1. Microéconomie : Calcul du prix de vente

On m'a dit un jour que l'objectif du premier cours de microéconomie devait être double : premièrement, enseigner le concept d'« offre et de demande ». Deuxièmement, montrer que les trois actions que sont la « maximisation du profit », la « maximisation des revenus » ou la « minimisation des coûts » sont trois objectifs extrêmement différents. Il s'agit bien sûr d'une simplification excessive, probablement une plaisanterie. Néanmoins, cela souligne l'importance primordiale de ces deux sujets.

Ce projet examinera de près la question de la « maximisation du profit » par rapport à la « maximisation des revenus » ou à la « minimisation des coûts ». Il s'agit d'une étude approfondie de la microéconomie de la vente de glaces. Imaginons un vendeur de glaces avec un chariot à une intersection très fréquentée de Central Park, à New York.

Le vendeur de glaces constate qu'en facturant 3 $ le cornet, il n'en vend que 120 par jour. En revanche, en facturant 2 $ le cornet, il en vend 200 par jour. Dans notre exemple, comme il a amorti ses frais de démarrage depuis longtemps, ses coûts d'exploitation ne s'élèvent qu'à 50 cents par cornet, auxquels s'ajoutent 100 $ par jour de frais de permis de vente de nourriture à Central Park. Comme vous pouvez l'imaginer, si son prix est trop élevé, il vendra trop peu de cornets et ne réalisera pas beaucoup de bénéfices ; s'il est trop bas, il en vendra plus, mais ne réalisera toujours pas beaucoup de bénéfices.

Le vendeur de glaces suppose que la demande varie linéairement avec le prix. Ce n'est souvent pas tout à fait exact, mais c'est souvent une approximation raisonnable. Il existe plusieurs façons de comprendre que la demande n et le prix p ont une relation.

$$n = 360 - 80p$$

ou de manière équivalente

$$p = 4{,}50 - 80\,\frac{n}{}$$

Prenons un instant pour voir si cela a du sens. En commençant par n = 200 et n = 120, nous obtenons les prix attendus de 2 $ et 3 $.
Si nous fixons p = 2,50, nous obtenons 160, comme prévu : c'est le point médian.

La méthode la plus simple pour trouver ces équations est de calculer la pente reliant les points (200, 2) et (120, 3), puis de trouver l'équation de la droite reliant ces deux points. Bien entendu, la forme de l'équation dépend de

Machine Translated by Google

que vous attribuiez ou non à p ou n le rôle de x. J'ai vu les deux conventions utilisées dans les manuels scolaires ; il n'existe pas de « norme industrielle ».

Observez maintenant qu'à un prix de 4,50 $, il ne vendra aucune glace du tout. Cela devrait être crédible : si les gens s'attendent généralement à payer environ 2 $, vous ne pouvez pas leur facturer plus du double de ce qu'ils ont l'habitude de payer. Le prix où la demande est nulle est appelé « prix maximum réalisable ». En revanche, si le prix d'un cône est de 0 $, nous vendons 360 cônes par jour ; c'est ce qu'on appelle le point de saturation, et c'est le maximum que l'on puisse espérer vendre, à savoir la demande à 0 $. Le modèle s'effondre pour les prix supérieurs à 4,50 $, car il prédit la vente d'un nombre négatif de cônes, ce qui n'est pas logique. De même, si la demande dépasse 360 cônes par jour (le « point de saturation »), un prix négatif est requis, ce qui n'est pas logique non plus. Le modèle est donc valable pour $0 \le p \le 4{,}50$ dollars, ou, de manière équivalente, $0 \le n \le 360$ cônes.

Nous souhaitons maintenant établir des fonctions de revenu, de coût et de profit en fonction du nombre de cônes vendus, n. Clairement, si l'on vend n cônes au prix p, le revenu est np. Par conséquent, nous avons

$$r(n) = np = n\,4{,}50 - 80 \qquad \frac{n}{} \quad n = 4{,}50n - 80 \quad \frac{2}{}$$

comme fonction de revenu. Parallèlement, la fonction de coût est de 50 cents par cône, plus 100 $, pour un total de

$$c(n) = 0{,}50n + 100$$

et cela nous donne une fonction de profit de

$$p(n) = \frac{2 - n}{80} + 4n - 100$$

Il s'agit d'une parabole. Il est logique d'obtenir une sorte de courbe, car si le prix est très bas ou très élevé, le profit est nul, voire pire, il y a une perte. Le point idéal, où le profit est réalisé, se situe quelque part entre les deux.

Créons un tableau et voyons si ces fonctions ont du sens et correspondent à nos données antérieures :

Machine Translated by Google

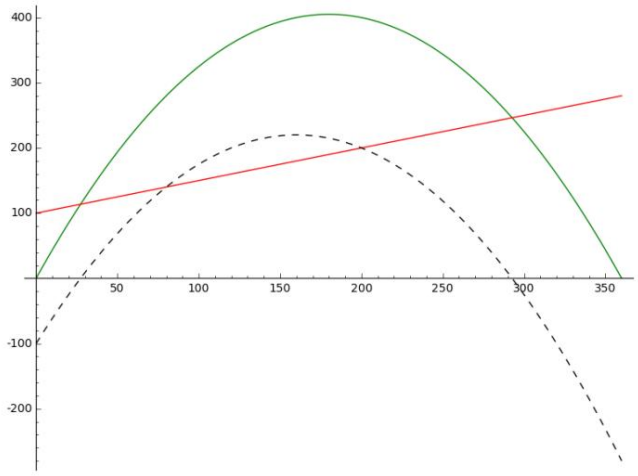


Figure 1. Le tracé du projet de microéconomie

Prix	Cônes vendus	Revenus	Coûts	Bénéfice
$ 0,00				
$ 0,50	360	0,00 $	280,00 $	-280,00 $
$ 1,00	320	160,00 $	260,00 $	-100,00 $
$ 1,50	280	280,00 $	240,00 $	40,00 $
$ 2,00	240	360,00 $	220,00 $	140,00 $
$ 2,50	200	400,00 $	200,00 $	200,00 $
$ 3,00	160	400,00 $	180,00 $	220,00 $
$ 3,50	120	360,00 $	160,00 $	200,00 $
$ 4,00	80	280,00 $	140,00 $	140,00 $
$ 4,50	40	160,00 $	120,00 $	40,00 $
$	0	0,00 $	100,00 $	-100,00 $

Je vous recommande de choisir 2 ou 3 valeurs et de vérifier manuellement ces 2 ou 3 rangées du tableau, avant de continuer.

OK, avez-vous vraiment vérifié deux ou trois lignes ci-dessus ? C'est vraiment utile. de le faire avant de lire plus loin.

Dans tous les cas, le graphique de ces trois fonctions est donné dans la figure 1. Comme vous pouvez voir, la fonction de profit est la parabole en pointillés (en noir si vous êtes (en lisant la version couleur de ce livre), la fonction de coût est la ligne (en rouge si vous lisez la version couleur de ce livre), et la fonction de revenu est la parabole solide (en vert si vous lisez la version couleur de ce livre). On peut en déduire de nombreux faits :

- Lorsque la fonction de coût et la fonction de revenu se croisent, c'est le point mort. Autrement dit, lorsque les coûts et les revenus sont égal, vous êtes à l'équilibre. Ici, nous avons deux points d'équilibre, l'un un peu moins de 30, et un un peu moins de 300. Pour les trouver, nous tapons

Machine Translated by Google


```
trouver_racine( (-x^2)/80 + 4,5*x == 0,5*x+100, 250, 350)
```

et aussi

```
trouver_racine( (-x^2)/80 + 4,5*x == 0,5*x+100, 10, 50)
```

pour découvrir qu'à environ 292,664 cônes et 27,3350 cônes, nous atteignons le seuil de rentabilité. Entre ces valeurs se trouve un bénéfice, et en dehors de ces valeurs, une perte. Cela semble correspondre aux données du tableau. • L'autre définition du seuil de rentabilité est lorsque le bénéfice est nul.

Nous aurions donc pu essayer de trouver les racines de la fonction de profit, comme autre moyen de déterminer les points morts. Pour ce faire, nous aurions tapé :

```
trouver_racine( (-x^2)/80 + 4*x - 100, 10, 50)
```

et aussi

```
trouver_racine( (-x^2)/80 + 4*x - 100, 250, 350)
```

pour trouver ces mêmes valeurs (292,664 et 27,3350), et vous verrez que la courbe en pointillés/noirs croise l'axe des x à cet endroit. • La fonction de coût n'est jamais nulle. C'est parce que le vendeur de glaces doit payer les frais de 100 $ par jour. Comme vous pouvez le voir, la ligne ne croise jamais l'axe des x pour tout n positif.

• La fonction de revenu est nulle à deux moments. Premièrement, lorsqu'aucune glace n'est vendue, ce qui est logique. On peut certainement convenir que si aucune glace n'est vendue, il n'y a aucun revenu. Nous avons constaté précédemment que cela se produit à 4,50 $ le cornet et 0 cornet par jour. Deuxièmement, le revenu est nul lorsque la glace est vendue 0 centime le cornet. De toute évidence, si vous ne facturez rien, il n'y a aucun revenu. Cela se produit au point de saturation, à 360 cornets par jour et 0 $ par cornet. Pour trouver ces valeurs dans Sage, nous devons saisir find_root(4.50*x-(x^2)/80, 0, 50)

et aussi

```
find_root( 4.50*x-(x^2)/80, 300, 400) pour obtenir les valeurs
```

0.0 et 360.0. • Nous avons maintenant trouvé quand le profit est nul et quand le revenu est nul, et déterminé que le coût n'est jamais nul. L'étape suivante consiste à essayer de minimiser ou de maximiser ces fonctions. Si vous avez fait du calcul, vous savez comment minimiser ou maximiser un polynôme. Sinon, vous vous souviendrez peut-être du fait qu'une parabole y = ax2 + bx + c a son optimum au point x = −b/2a. • Pour la fonction de revenu, nous voyons que le revenu maximal se produit à 180 cônes, un coût de 2,25 $ par cône et un revenu de 405 $. Les coûts seraient de 190 $, pour un profit de 215 $. • Pour la fonction de profit, nous voyons que le profit maximal se produit à 160 cônes, un coût de 2,50 $ par cône et un revenu de 400 $. Les coûts seraient de 180 $, pour un profit de 220 $.

Machine Translated by Google

• Le point de coûts minimum serait 0 cornets de glace, pour un coût de 100 $. Cependant, le revenu est de 0 $ et le bénéfice est négatif, soit une perte de 100 $. Ce n'est certainement pas un bon plan d'affaires. (Même si les coûts sont inférieurs dans ce point que dans les deux points précédents.) • Avez-vous remarqué que le point de profit maximum avait un revenu inférieur (400 $ contre 405 $) mais un bénéfice supérieur (220 $ contre 215 $) que le point de revenu maximum ? Avez-vous remarqué que le point de revenu maximum avait un bénéfice inférieur (215 $ contre 220 $) mais un revenu supérieur (405 $ contre 400 $) que le point de profit maximum ?

En conclusion, nous constatons que « maximiser les revenus », « minimiser les coûts » et « maximiser les profits » sont trois objectifs différents. De plus, nous devrions suggérer au vendeur de glaces de vendre sa glace à 2,50 $ le cornet, car maximiser les profits est la seule chose intelligente à faire.

Au fait, le code pour produire l'image avec les graphiques des trois fonctions est le suivant :

```
r(x) = 4,50*x - x^2/80 c(x) = 0,5*x
+ 100 p(x) = r(x) - c(x)


P1 = plot( r(x), 0, 360, couleur='vert' )
P2 = plot( c(x), 0, 360, couleur='rouge' )
P3 = plot( p(x), 0, 360, couleur='noir', style de ligne='--' )
P = P1 + P2 + P3

P.show()
```

Défi : votre

défi consiste à analyser une nouvelle situation commerciale. Un livre est prévendu par un éditeur sur deux marchés tests à 49,95 $ et 39,95 $. Il se vend respectivement à 80 et 140 exemplaires. Par le passé, les ventes annuelles ont été 100 fois supérieures à celles de l'un des marchés tests. (Pour être clair, cela signifie qu'avec un prix fixé à 49,95 $, les experts marketing prévoient 8 000 exemplaires vendus, et 14 000 exemplaires à 39,95 $.)

L'impression des livres coûte 7 $, plus 4 000 $ payables une fois pour les frais de relecture. L'éditeur perçoit 50 % des recettes des ventes. Pour information, environ 20 % des recettes des ventes sont reversées à la librairie locale, 20 % au distributeur et 10 % aux droits d'auteur. On peut supposer que le prix de vente et le nombre d'exemplaires vendus sont liés de manière linéaire.

Les livrables de ce projet sont (1) une équation

reliant le prix de vente et le nombre d'exemplaires vendus, (2) la fonction de revenu, (3) la fonction de coût, (4) la fonction
de profit,

Machine Translated by Google

(5) l'image montrant les trois fonctions représentées graphiquement simultanément, (6) le(s) point(s) d'équilibre, (7) le n et le p pour obtenir un revenu maximal, (8) le n et le p pour obtenir un profit maximal.

2.2. Biologie : Artères obstruées et loi de Poiseuille

Si un Américain décède de causes naturelles, la probabilité qu'il décède d'une maladie cardiaque est de 35,30 %, de diverses formes de cancer de 33,94 % et de toute autre cause de 30,75 %. Sachant que les maladies cardiaques tuent environ un tiers d'entre nous, la réalité des artères obstruées et du cholestérol devrait tous nous préoccuper.

Le sang circulant dans une artère est régi par la même équation que l'eau circulant dans un tuyau ou l'huile circulant dans un flexible. Ceci s'applique à tous les écoulements de fluides dans un cylindre, à l'exception des écoulements turbulents (où le fluide se déplace et culbute sur lui-même) et des écoulements compressibles (comme l'air circulant dans un flexible pneumatique). Cette équation est appelée loi de Poiseuille, du nom de Jean-Lonard Marie Poiseuille (1797–1869), dont la thèse2 appliquait cette mathématique à l'aorte humaine.

Dans ce projet, nous allons explorer les relations entre certaines variables de la loi de Poiseuille. Nous allons représenter ces interdépendances sous forme de tableaux. Pour réaliser ce projet, vous devez avoir lu la section 5.1, qui décrit comment créer des tableaux dans Sage.

Le contexte La formule
elle-même est la suivante.

$$\dot{V} = (\Delta P)\frac{\pi r^4}{8\mu L}$$

et malheureusement, nous n'avons pas le temps de le déduire ici. Cependant, je recommande vivement au lecteur curieux de se référer à l'article « Blood Vessel Branch-ing: Beyond the Standard Calculus Problem » du professeur John Adam, publié dans le Mathematics Magazine, vol. 84, par la Mathematical Association of America, en 2011.

Prenons un moment pour identifier la signification de chacune de ces variables. • Le $\dot{V}$ est le débit volumique, en unités telles que les centimètres cubes par minute. Dans une artère principale, un débit typique pourrait être de 100 cm3/minute. • Le rayon de l'artère est r, en unités telles que les centimètres. Une artère principale peut avoir un diamètre de 0,3 cm ou très environ 1/8 de pouce. • La longueur de l'artère est L, encore une fois en unités telles que les centimètres.
Une artère principale peut mesurer 20 cm ou 8 pouces de longueur.

---

1. D'après les données de 2010 du site Web « FastStats » du Centre pour le contrôle et la prévention des maladies (CDC). Ces données excluent (en tant que causes non naturelles) les homicides, les suicides et les accidents.

2Poiseuille, J-L. M., Recherches sur la force du coeur aortique, D.Sc., Ecole Polytech- nique, Paris, France. 1828.

Machine Translated by Google

Remarque : L'unité de pression peut varier. Dans le système américain, l'unité standard est la livre par pouce carré, car les forces sont mesurées en livres. De même, dans le système métrique, les forces étant mesurées en newtons, les unités de pression deviennent alors des newtons par mètre carré. L'utilisation de newtons par mètre carré est assez curieuse, car la surface des artères ne dépasse pas plusieurs mètres carrés. Une autre unité de pression utilisée est l'« atmosphère », où la pression atmosphérique est de 1 atmosphère. Cela correspond à 14,6959 livres par pouce carré, soit 101 325 N/m^2 .

• La médecine existe depuis longtemps. Pour des raisons historiques, il est courant de mesurer la tension artérielle dans une unité très archaïque, appelée « millimètres de mercure ». Lorsque vous consultez un médecin et qu'il vous annonce que votre tension artérielle est de « 120 sur 80 », ce 120 signifie « 120 millimètres de mercure », soit 120 mmHg en abrégé. Prenez-le simplement comme unité et considérez 120 comme « normal ». Si votre premier chiffre (appelé systolique) est de 145 mmHg, vous pourriez être diagnostiqué comme souffrant d'« hypertension ». Un score systolique de 180 mmHg peut entraîner des lésions organiques et est appelé « urgence hypertensive ».

• La viscosité est une variable physique que vous avez peut-être déjà vue, et elle est notée μ. Elle représente le degré d'« adhérence » ou d'« épaisseur » d'un liquide. Puisque le sang est le liquide en question et que nous ne considérerons pas d'autres liquides, nous pouvons simplement prendre μ comme une constante. L'unité habituelle est le poise. Une valeur typique pour le sang serait μ = 0,0035 poise. À titre de comparaison, l'huile de votre voiture pourrait avoir μ = 0,250 poise (car elle est collante) et l'eau μ = 0,001 millipoise (car elle n'est pas collante). Le miel est très collant et a une viscosité de 20 poises.

On ne peut pas utiliser l'équilibre comme unité, car on utilise une ancienne unité de pression. Pour nous,

$$\mu = 4{,}37535 \cdots \times 10^{-8}$$

n'est qu'une constante dans la formule. L'unité de viscosité est « sec mmHg », ce qui est extrêmement étrange, mais c'est ce qui permet d'utiliser les bonnes unités dans la formule.

Votre défi Pour ce

projet, vous allez créer des tableaux qui peuvent être utilisés pour aider un médecin à comprendre les réalités numériques de cette relation quartique dans la loi de Poiseuille entre le rayon artériel et le débit sanguin.

Les tableaux permettent la modification de certaines variables, tandis que la plupart sont fixées à des « valeurs standard » choisies pour être très typiques. Les valeurs standard de nos variables dans ce projet sont : $\mu = 4{,}375359 \times 10^{-8}$ pour la viscosité, L = 20 cm pour la longueur de l'artère, r = 0,3 cm pour le rayon artériel, $\dot{V} = 100$ centimètres cubes par seconde pour le débit sanguin, et enfin $\Delta P = 0{,}0275$ mmHg.

Machine Translated by Google

• Votre premier tableau doit conserver nos valeurs standard pour μ, L, ∆P et calculer V˙ en fonction de r, en utilisant la loi de Poiseuille. Le r doit être calculé en fonction du pourcentage de blocage, de 0 % à 50 %, par incréments de 2 %. La première colonne du tableau doit indiquer le pourcentage de blocage, la deuxième colonne doit indiquer le débit sanguin ou V˙. • Votre deuxième tableau doit conserver nos valeurs standard pour μ, L et V˙. Encore une fois, le r doit être calculé en fonction du pourcentage de blocage, de 0 % à 50 %, par incréments de 2 %. Cette fois, vous calculerez le ∆P requis pour atteindre le débit de 100 cc/minute compte tenu du blocage (le V˙ étant donné le ∆P), en utilisant la loi de Poiseuille. La première colonne du tableau doit indiquer le pourcentage de blocage, la deuxième colonne doit indiquer le ∆P. La troisième colonne doit représenter le ∆P comme le « pourcentage de la normale ». En d'autres termes, les 3/2 du ∆P normal doivent être rapportés comme étant de 150 %. • Le troisième tableau doit conserver nos valeurs standard pour μ, L et ∆P. La première colonne doit indiquer le « pourcentage de débit normal » de 100 % à 25 %, par incréments de 3 %. La deuxième colonne doit indiquer le débit, en centimètres cubes par seconde, identifié par ce pourcentage. La troisième colonne doit indiquer le rayon impliqué par ce résultat, tel que calculé par la loi de Poiseuille. La quatrième colonne doit indiquer le pourcentage de blocage. (En d'autres termes, un rayon de 0,15 cm correspond à un blocage de 50 %, car 0,15/0,3 = 0,5.)

Vous pouvez vérifier votre travail en vous connectant à l'équation d'origine, mais vous pourriez également trouver utile de comparer votre premier tableau au suivant, qui est donné par incréments de 6 %. Un blocage de 0 % implique 99,9617636500122 cc/minute. Un blocage de 6 % implique 78,0450430095128 cc/minute. Un blocage de 12 % implique 59,9466058383290 cc/minute. Un blocage de 18 % implique 45,1948885141475 cc/minute. Un blocage de 24 % implique 33,3494195216211 cc/minute. Un blocage de 30 % implique 24,0008194523679 cc/minute. Un blocage de 36 % implique 16,7708010049720 cc/minute. Un blocage de 42 % implique 11,3121689849831 cc/minute. Un blocage de 48 % implique 7,30882030491648 cc/minute. Un blocage de 54 % implique 4,47574398425329 cc/minute.

2.3. Optimisation industrielle : expédition de taconite

Dans ce projet, vous résoudrez un problème d'itinéraire maritime très simple mais réaliste, en utilisant la programmation linéaire. Ce projet suppose que vous ayez lu la section 4.21, page 186, expliquant comment utiliser la programmation linéaire dans Sage.

En particulier, un conglomérat minier possède sept mines autour du Minnesota et du Wisconsin, et dispose d'installations d'expédition à Two Harbors, Green Bay, Minneapolis, ainsi que d'un complexe d'expédition dans les ports jumeaux de Duluth, Minnesota.

Machine Translated by Google

Frais de livraison

	Duluth, MN Superior, WI	Two Harbors, Minnesota	Green Bay, Wisconsin	Minneapolis, Minnesota	Total Production
Mine 1	$ 18/ton	$ 30/ton	$ 74/ton	$ 35/ton	8,000 tons/wk
Mine 2	$ 73/ton	$ 21/ton	$ 50/ton	$ 65/ton	13,000 tons/wk
Mine 3	$ 37/ton	$ 93/ton	$ 67/ton	$ 61/ton	7,000 tons/wk
Mine 4	$ 24/ton	$ 95/ton	$ 63/ton	$ 35/ton	16,000 tons/wk
Mine 5	$ 38/ton	$ 77/ton	$ 80/ton	$ 71/ton	13,000 tons/wk
Mine 6	$ 73/ton	$ 15/ton	$ 39/ton	$ 68/ton	8,000 tons/wk
Mine 7	$ 48/ton	$ 51/ton	$ 69/ton	$ 14/ton	9,000 tons/wk

Tableau 1. Tableau des coûts d'expédition et de la production minière

et Superior, Wisconsin. Vous déterminerez la quantité de taconite à expédier de chaque La mine est reliée à chacune des quatre plateformes d'expédition. Plusieurs contraintes s'appliquent.

Premièrement, il faut que la quantité de taconite quittant chaque mine (pour les quatre destinations) doit correspondre exactement à la quantité produite cette semaine-là. Il ne devrait pas être plus grand, sinon vous ordonnez à vos mines d'en expédier davantage. taconite qu'ils n'en ont réellement ; il ne devrait pas être inférieur, sinon vous aurez une accumulation de taconite qui ne peut être stockée sur place. La production totale dans chaque mine est répertorié dans le tableau 1.

Deuxièmement, la demande et la capacité de chaque plateforme d'expédition doivent être respectées. L'installation de Two Harbors, dans le Wisconsin, peut recevoir et expédier jusqu'à 18 000 tonnes. L'usine de Green Bay peut recevoir et expédier jusqu'à 25 000 tonnes. Le complexe d'expédition Duluth/Superior peut recevoir et expédier entre 15 000 et 45 000 tonnes, le minimum étant là pour assurer des revenus suffisants pour payer les factures dans cet endroit coûteux. L'établissement de Minneapolis peut recevoir et expédier entre 10 000 et 30 000 tonnes.

Troisièmement, le calendrier d'expédition devrait permettre d'apporter les taconite jusqu'aux stations de réception et d'expédition à un coût minimal. Le coût des expéditions de chaque mine à chaque installation de réception et d'expédition sont données dans le tableau 1, en dollars par tonne.

Votre objectif est de modéliser ce problème sous forme de programme linéaire, de le coder dans Sage, et résolvez-le. Pour vous aider, voici une astuce :

d1, d2, . . . , d7 sont la quantité expédiée hors de Duluth/Superior à partir des sept mines. Soit t1, t2, . . . , t7 la quantité expédiée depuis Two Harbors. Soit

g1, g2, . . . , g7 sont la quantité expédiée depuis Green Bay. Soit m1, m2, . . . , la m7 quantité expédiée depuis Minneapolis.

Parce que vous montrerez beaucoup de travail dans ce processus, je n'ai aucune crainte de Je vous donne la réponse finale. Voici le résultat de mon programme :

```
Le coût minimum d'expédition est
1883000.0

Les quantités expédiées à Duluth/Superior à partir des sept mines sont
{1 : 8 000,0, 2 : 0,0, 3 : 7 000,0, 4 : 15 000,0, 5 : 13 000,0, 6 : 0,0, 7 : 0,0}
```

Machine Translated by Google

Les quantités expédiées à Two Harbors depuis les sept mines sont {1 : 0,0, 2 : 13 000,0, 3 : 0,0, 4 : 0,0, 5 : 0,0, 6 : 5 000,0, 7 : 0,0}

Les quantités expédiées à Green Bay depuis les sept mines sont {1 : 0,0, 2 : 0,0, 3 : 0,0, 4 : 0,0, 5 : 0,0, 6 : 3 000,0, 7 : 0,0}

Les quantités expédiées à Minneapolis depuis les sept mines sont {1 : 0,0, 2 : 0,0, 3 : 0,0, 4 : 1 000,0, 5 : 0,0, 6 : 0,0, 7 : 9 000,0}

Au fait, si vous pensez que ce problème est complexe, gardez à l'esprit ce qui suit. Un problème réaliste pourrait comporter de nombreux points de collecte intermédiaires, plusieurs dizaines de mines et des capacités limitées sur toutes les routes reliant ces points. On peut facilement imaginer un problème avec 30 mines, 12 sites de collecte intermédiaires et 5 ports, pour 360 + 60 = 420 itinéraires. Chaque itinéraire aurait un coût et une limite de capacité. Chaque limite de capacité est, en soi, une inégalité. Par conséquent, ce type de problème, « dans le monde réel », peut devenir énorme.

2.4. Chimie : Équilibrer les réactions avec des matrices

Suivre un cours de chimie offre de nombreuses activités ludiques et enrichissantes. L'une d'elles consiste à équilibrer une équation chimique, souvent facile. Cependant, certaines équations sont plus difficiles à équilibrer (pour les non-chimistes). Ici, nous allons apprendre à équilibrer rapidement et facilement même les exemples les plus complexes grâce à l'algèbre matricielle.

Tous les exemples présentés dans ce projet, ainsi que l'idée originale du projet, sont entièrement dus au professeur Gergely Sirokman, du Wentworth Institute of Technology de Boston, dans le Massachusetts. Je le remercie pour le temps qu'il a consacré à la rédaction de ces exemples pour ce livre.

Ce projet requiert des connaissances sur le traitement de systèmes d'équations linéaires à solutions infinies. L'annexe D couvre l'ensemble du sujet, mais certains lecteurs le connaissent déjà et n'auront pas besoin de la lire.

2.4.1. Contexte Commençons

par un exemple énergétique. À l'époque des dirigeables, les dirigeables américains étaient souvent remplis d'hélium. Cependant, l'Allemagne n'ayant pas accès à de grandes quantités d'hélium, elle a utilisé de l'hydrogène. C'est regrettable, car l'hydrogène peut exploser en présence d'une étincelle. Le Hindenburg, un dirigeable fabriqué par la société Zeppelin, a explosé en une boule de feu en 1937 près de Lakehurst, dans le New Jersey. L'hydrogène se combine à l'oxygène de l'air pour former de la vapeur d'eau. Voici la formule chimique de cette réaction :

$$2\ H_2 + O_2 \rightarrow 2\ H_2O$$

Machine Translated by Google

Voyons ce que signifie l'équilibre d'une équation chimique. Si nous avons deux molécules de H et une molécule de O avant la réaction, nous avons alors quatre atomes d'hydrogène et deux atomes d'oxygène avant la réaction.
Si nous avons deux molécules de H O après la réaction, nous avons alors 4 atomes d'hydrogène et 2 atomes d'oxygène. Aucun atome d'hydrogène n'a été créé ni détruit pendant la réaction chimique, car nous en avions 4 avant et 4 après ; de même, aucun atome d'oxygène n'a été créé ni détruit, car nous en avons 2 avant et 2 après. Puisque les atomes ne sont jamais créés ni détruits lors des réactions chimiques, nous devrions rejeter cette équation chimique comme déséquilibrée si nous n'avions pas la correspondance avant-après. La correspondance doit être présente pour chaque atome de la réaction chimique ; si même une paire avant-après est inégale, cela signifie qu'il y a une erreur.

La combustion de l'acétylène dans un chalumeau est une réaction chimique au cours de laquelle l'acétylène (C H) et l'oxygène de l'air (O) se combinent pour former du dioxyde de carbone (CO) et de l'eau (H O). À partir de ces formules chimiques, nous devons trouver les entiers positifs des quatre points d'interrogation ci-dessous afin d'équilibrer l'équation chimique.

$$? \, C_2H_2 + ? \, O_2 \rightarrow ? \, CO_2 + ? \, H_2O$$

Prenez un moment pour l'essayer vous-même, avec un peu de bricolage, et
Essayez de deviner et de vérifier à l'ancienne. Essayez de trouver les valeurs des points ?.

Non vraiment, je le pense vraiment, essayez.

Bon, à ce stade, j'espère que vous avez fait une tentative sérieuse et que vous avez trouvé les valeurs pour équilibrer l'équation :

$$2 \, C_2H_2 + 5 \, O_2 \rightarrow 4 \, CO_2 + 2 \, H_2O$$

C'était un peu difficile, du moins pour moi. (Peut-être que c'était plus facile pour vous.)
Avec des équations chimiques plus complexes, vous devrez peut-être deviner et vérifier pendant un certain temps. Méconnu de nombreux étudiants en chimie, il existe un moyen d'équilibrer des équations chimiques à l'aide de matrices, sans devinettes.

2.4.2. Cinq exemples intéressants Voici

cinq réactions chimiques intéressantes qui serviront d'exemples.

(1) Cette réaction se produit lors de la fusion de l'oxyde ferreux (minerai de fer) pour extraire le fer.

$$? \, Fe_2O_3 + ? \, CO \rightarrow ? \, Fe + ? \, CO_2$$

(2) Lorsque vous mettez de l'essence dans votre voiture, l'octane est l'un des composés chimiques importants. Il s'agit de la réaction qui se produit lorsque votre voiture brûle de l'octane en présence d'une abondance d'oxygène. Si vous faites tourner le moteur trop vite, d'autres réactions chimiques peuvent se produire, produisant du CO, voire du C.

$$? \, C_8H_{18} + ? \, O_2 \rightarrow ? \, CO_2 + ? \, H_2O$$

Machine Translated by Google

(3) Voici un exemple de réaction acido-basique, un type de réaction assez important en chimie. L'hydroxyde de calcium (Ca(OH)2) a de nombreuses utilisations, y compris le traitement des eaux usées humaines. L'acide phosphorique (H3PO4) peut être présent dans les eaux usées humaines provenant de sources aussi courant que les sodas. On voit ici que l'hydroxyde de calcium traitera l'acide phosphorique et le convertira en eau ordinaire et en phosphate tricalcique. Il s'avère que le phosphate tricalcique est beaucoup plus plus bénin, car il se produit dans les os des animaux et dans le lait de vache.

? Ca(OH)2+? H3PO4 →? Ca3(PO4)2+? H2O

(4) Dans l'exemple précédent, nous avons dit que l'acide phosphorique (H3PO4) Présent dans les sodas. D'où vient-il ? Il peut être facilement fabriqué à partir de phosphore élémentaire pur et d'acide hypochloreux. (HClO) en présence d'eau. De l'acide chlorhydrique (HCl) se forme comme produit complémentaire. Voici la réaction :

? P4+? HClO+? H2O →? H3PO4+? HCl

(5) C'est l'une des méthodes préférées pour extraire l'or d'échantillons de roche dilués. Elle est également terriblement toxique (NaCN est du sodium). cyanure... ce qui est aussi mauvais que ça en a l'air !)

? NaCN+? Moi+ ? O2+? H2O →? Au(CN)2+? NaOH

2.4.3. La méthode lente : dériver la méthode

Nous allons maintenant développer progressivement la méthode matricielle pour équilibrer une équation chimique. Une approche simplifiée sera présentée dans la section suivante, mais elle n'aura aucun sens si vous ne lisez pas cette dérivation très attentivement. Nous prenons comme exemple le membre du milieu des cinq exemples que nous avons listés il y a quelques instants.

? Ca(OH)2+? H3PO4 →? Ca3(PO4)2+? H2O

Si j'avais x1 molécules de Ca(OH)2, x2 molécules de H3PO4, x3 molécules de Ca3(PO4)2, et x4 molécules de H2O, alors . . .

- . . . nous aurions x1 atomes de calcium à gauche, et 3x3 atomes de calcium à droite.
- . . . nous aurions 2x1+4x2 atomes d'oxygène à gauche, et 8x3+x4 atomes d'oxygène à droite.
- . . . nous aurions 2x1 + 3x2 atomes d'hydrogène à gauche, et 2x4 atomes d'hydrogène à droite.
- . . . nous aurions x2 atomes de phosphore à gauche, et 2x3 atomes de phosphore à droite.

Machine Translated by Google

Étant donné que, pour chaque élément, nous devons avoir le même nombre d'atomes sur à gauche comme à droite, alors on a les 4 équations suivantes à 4 variables :

$$x1 = 3x3$$
$$2x1 + 4x2 = 8x3 + x4$$
$$2x1 + 3x2 = 2x4$$
$$x2 = 2x3$$

La matrice équivalente à ce système d'équations serait

$$A = \begin{vmatrix} 1 & 0 & -3 & 0 & 0 \\ 2 & 4 & -8 & -1 & 0 \\ 2 & 3 & 0 & -2 & 0 \\ 0 & 1 & -2 & 0 & 0 \end{vmatrix}$$

Selon Sage, le RREF est

```
[[[[ 1    0    0 -1/2 0    0]
     0    1    -1/3 1      0]
     0    0    -1/6 0 0    0]
     0    0                0]
```

Comme vous pouvez le voir, nous avons une ligne entièrement composée de zéros. Nous avons un pivot pour x1, x2, et x3, donc ces variables sont déterminées. Il n'y a pas de pivot pour x4, donc cette variable est libre. La « forme standard » de l'ensemble infini de solutions est écrit comme suit :

$$x1 = (1/2)x4$$
$$x2 = (1/3)x4$$
$$x3 = (1/6)x4$$
$$x4 = x4$$

Cependant, il est d'usage de ne considérer que des valeurs entières pour les coefficients de l'équation d'une réaction chimique. Que fait-on lorsque l'on souhaite Des solutions entièrement entières dans un cas comme celui-ci ? C'est une astuce extrêmement rare, non ? couramment enseignée, mais facile à apprendre. Nous prenons d'abord le PPCM (le moins courant) multiple) de tous les dénominateurs de toutes les fractions. Dans ce cas, le ppcm de {2, 3, 6} est six. Ensuite, nous insérons x4 = 6n, où n est un nouveau muet. variable. Maintenant, nous avons

$$x1 = (1/2)6n = 3n$$
$$x2 = (1/3)6n = 2n$$
$$x3 = (1/6)6n = n$$
$$x4 = 6n$$

Pour tout entier n, les quatre valeurs que vous obtiendriez forment un entier solution aux équations données. Cela inclut les valeurs absurdes comme n = 0 ou valeurs négatives de n. (Ou pour être précis, les cas n ≤ 0 sont mathématiquement

Machine Translated by Google

solutions valides au système d'équations, mais elles ne sont pas chimiquement significatives.) Nous voulons la solution entièrement positive non nulle la plus simple, nous devons donc remplacer n = 1. Maintenant, nous avons

$$x1 = 3; x2 = 2; x3 = 1; x4 = 6$$

donnant la réaction chimique finale

$$3\ Ca(OH)2 + 2\ H3PO4 \rightarrow 1\ Ca3(PO4)2 + 6\ H2O$$ mais il faut

noter que les chimistes n'écrivent généralement pas le « 1 » devant la troisième molécule.

Français Maintenant, nous devons vérifier notre travail.

- Comme vous pouvez le voir, ... nous avons 3(1)+0 = 3 atomes de calcium à gauche, et 1(3)+0 = 3
- atomes de calcium à droite. . . . nous avons 3(2) + 2(4) = 14 atomes d'oxygène à gauche, et 1(8) + 6(1) = 14 atomes
- d'oxygène à droite. . . . nous avons 3(2) + 2(3) = 12 atomes d'hydrogène à gauche, et 0 + 6(2) = 12 atomes d'hydrogène à
- droite. . . . nous avons 0 + 2(1) = 2 atomes de phosphore à gauche, et 1(2) + 0 = 2 atomes de phosphore à droite. • Toutes ces

correspondances sont exactes, nous déclarons donc le succès !

2.4.4. Équilibrer rapidement. C'était un

processus vraiment excellent mais lent. Maintenant, nous voudrions peut-être équilibrer les équations chimiques plus rapidement. La clé à retenir est que « les atomes sont des lignes, les molécules sont des colonnes ». Un indice pour s'en souvenir est d'observer que le mot « ligne » est plus court que le mot « colonne », et de même que le mot « atome » est plus court que le mot « molécule ». En associant court à court et long à long, il est naturel que les atomes correspondent aux lignes et les molécules aux colonnes.

Les atomes peuvent être codés dans n'importe quel ordre. Prenons le deuxième exemple : la combustion de l'octane.

$$?\ C8H18 + ?\ O2 \rightarrow ?\ CO2 + ?\ H2O$$

Examinons un atome à la fois. Nous allons demander à chaque molécule combien d'atomes de cet atome elle possède. Pour les réactifs (à gauche de la flèche), nous utiliserons des entiers positifs ; pour les produits (à droite de la flèche), nous utiliserons des entiers négatifs.

Carbone : Les quatre molécules ont 8, 0, 1 et 0 atomes de carbone, nous écrivons donc 8, 0, -1, 0.

Hydrogène : Les quatre molécules ont 18, 0, 0 et 2 atomes d'hydrogen, donc on écrit 18, 0, 0, -2.

Oxygène : Les quatre molécules ont 0, 2, 2 et 1 atome d'oxygène, nous écrivons donc 0, 2, -2, -1.

Nous définissons maintenant la matrice et demandons son RREF, dans la notation que nous avons apprise à la page 23. Cependant, nous devons faire attention à ajouter la colonne

Machine Translated by Google

de zéros comme colonne la plus à droite de la matrice, signifiant la partie = 0 de les équations.

```
B= matrice([ [8, 0, -1, 0, 0], [18, 0, 0, -2, 0], [0, 2, -2, -1, 0]])

imprimer « Question : »
imprimer B
imprimer « Réponse : »
imprimer B.rref()
```

Nous avons maintenant la sortie

```
Question:
 [ 8 0 -1 0 0]
 [18 0 0 -2 0]
 [ 0 2 -2 -1 0]
Répondre:
        1      0      0 -1/9 0      0]
        0      1      -25/18 1      0]
[ [ [   0      0      -8/9          0]
```

Rapidement, nous vérifions que la matrice que nous avons entrée était la matrice qui nous souhaitions entrer. Ensuite, nous prenons le PPCM des dénominateurs dans le Chronique intéressante. Le ppcm de {9, 18, 9} est clairement 18. Pour les trois premiers coefficients, nous multiplions les entrées de la colonne informative par le négatif lcm (dans ce cas, −18), et obtenir 2, 25 et 16. Pour le quatrième coefficient, c'est juste le lcm lui-même, à savoir 18. Nous concluons avec

$$2\ C_8H_{18} + 25\ O_2 \rightarrow 16\ CO_2 + 18\ H_2O$$

mais maintenant nous devons le vérifier.

- Nous avons 2(8) + 0 = 16 atomes de carbone à gauche, et 16(1) + 0 = 16 atomes de carbone à droite.
- Nous avons 2(18) + 0 = 36 atomes d'hydrogène à gauche, et 0 + 18(2) = 36 atomes d'hydrogène à droite.
- Nous avons 0+25(2) = 50 atomes d'oxygène à gauche, et 16(2)+18 = 50 atomes d'oxygène à droite.
- Puisque les chiffres correspondent, nous déclarons victoire !

Enfin, il serait bon de noter que les chimistes ont des raccourcis qu'ils utilisent qui simplifient considérablement les choses, mais qui nécessitent intuition chimique. Par exemple, PO_4 et OH sont des ions polyatomiques, qui signifie que vous pouvez « prétendre » qu'il s'agit d'atomes individuels. Cela peut supprimer une variable du système d'équations linéaires, qui est significative.

Votre défi :

Essayez maintenant les premier, quatrième et cinquième exemples par vous-même. Bonne chance ! n'oubliez pas de vérifier votre travail !

Machine Translated by Google

2.5. Physique : Projectiles balistiques

Les problèmes de mouvement de projectiles balistiques sont un élément essentiel des cours de calcul et de physique. Nous supposons ici que le lecteur maîtrise le calcul et la physique (niveau lycée). Nous allons explorer deux problèmes relativement simples, qui ne nécessitent pas l'utilisation de Sage, et un autre, plus complexe, nécessitant Sage ou un autre outil de calcul.

Pour une large classe de problèmes simples et intermédiaires, la technique en sept étapes suivante résout souvent complètement le problème.

(1) Lisez attentivement le problème et écrivez une équation de somme de forces, souvent à l'aide d'un diagramme de corps libre. Cela produit une fonction d'accélération utilisant $\Sigma F = ma$.
(2) Intégrez la fonction d'accélération pour obtenir la fonction de vitesse, qui malheureusement aura un +C.
(3) Utilisez certaines informations du problème pour calculer la valeur de C.
(4) Intégrez la fonction de vitesse pour obtenir la fonction d'altitude, qui aura malheureusement un +C (probablement différent).
(5) Encore une fois, utilisez certaines informations du problème pour calculer le valeur de C.
(6) Vérifiez que vos trois fonctions sont cohérentes avec toutes les données fournies (en insérant des valeurs) et entre elles (en prenant des dérivées).

Remarque : à ce stade, vous disposez explicitement des trois fonctions et pouvez répondre à la plupart des questions concernant le projectile.

(7) Relisez la question et voyez ce qui est réellement demandé.

Tout au long de ce projet, $y(t)$ représentera l'altitude d'un projectile et $y(t) = v(t)$ sa vitesse. De même, $y(t) = v(t) = a(t)$ est l'accélération du projectile. Nous négligerons la résistance de l'air dans tous les cas. En raison des risques d'erreur d'arrondi, il est malheureusement nécessaire d'utiliser $g = 9{,}80665$ m/s² tout au long de ce problème ; nous avons réellement besoin des six chiffres significatifs.

Avant de commencer, il convient de préciser qu'il existe deux manières d'aborder ce type de problèmes. En général, les mathématiciens appliqués utilisent de nombreux chiffres de précision à mi-parcours, et n'arrondissent qu'à la fin, afin de refléter l'incertitude des données d'entrée. Cependant, ils n'associent généralement pas les unités des équations à tous les coefficients et constantes. À l'inverse, les physiciens utilisent des unités à l'intérieur des équations (!) associées à tous les coefficients et constantes. De plus, en première année d'université, certains manuels arrondissent les nombres étape par étape aux étapes intermédiaires des calculs, afin de refléter la précision limitée des données d'entrée. Habitué à la technique des mathématiciens appliqués, je l'utilise ici afin de minimiser les risques d'erreur typographique.

Vous pouvez réaliser ces deux exercices d'échauffement avec un crayon et une calculatrice scientifique. Nous passerons ensuite à un problème utilisant Sage.

Machine Translated by Google

2.5.1. Notre premier exemple de trajectoires balistiques Supposons qu'une pièce d'artillerie soit située sur une colline, de telle sorte que la bouche du canon soit à 25 m au-dessus du paysage environnant. Elle tire son projectile à une vitesse initiale de 300 m/s, à un angle de 30◦ au-dessus de l'horizontale. Quelle est la distance parcourue par le projectile ?

Tout d'abord, nous devons construire une fonction d'accélération. La seule force agissant sur le projectile (en vol) est son poids dû à la gravité terrestre. Nous écrivons

$$\Sigma F = ma(t) - mg$$
$$= ma(t) - g = a(t)$$
$$a(t) = -9{,}80665$$

Deuxièmement, nous intégrons pour trouver v(t) avec un +C.

$$a(t)\,dt = a(t)\,dt$$

$$v\,(t)\,dt = -9{,}80665\,dt$$

$$v(t) = -9{,}80665t + C$$

Troisièmement, nous savons que v(0) est la composante verticale de la vitesse initiale. ité. Cela est calculé avec

$$(300)(\cos 30^\circ) = (300)(0{,}866025) = 259{,}807$$

et donc

$$v(t) = -9{,}80665t + C \quad v(0) = -9{,}80665(0) + C \quad 259{,}807 = C$$

Nous savons maintenant que v(t) = −9,80665t + 259,807, au moins pendant le vol. Quatrièmement, nous intégrons pour trouver y(t) avec un +C.

$$v(t)\,dt = v(t)\,dt$$

$$y\,(t)\,dt = (-9{,}80665t + 259{,}807)\,dt$$

$$y(t) = -\frac{1}{2}(9.80665)t^2 + 259.807t + C$$

$$y(t) = -4{,}90332t^2 + 259.807t + C$$

Machine Translated by Google

Cinquièmement, nous prenons en compte le fait que y(0) est l'altitude de la colline, soit 25 m.

$$y(t) = -4{,}90332ty(0) = \quad ^2 + 259.807t + C$$

$$-4{,}90332(0)2 + 259{,}807(0) + C$$

$$25 = C$$

Enfin, nous avons

$$y(t) = -4{,}90332t \text{ et } \quad ^2 + 259.807t + 25$$

nous pouvons commencer la partie la plus importante du problème : vérifier notre travail.

- Nous pouvons vérifier que y (t) = −9,80664t+ 259,807, ce qui correspond à notre v(t) à l'exception de l'erreur d'arrondi. •
- Nous pouvons vérifier que y (t) = −9,80664, ce qui correspond à notre a(t) à l'exception de l'erreur d'arrondi. •
- Nous pouvons vérifier que y (0) = v(0) = 259,807, comme souhaité. (C'est la composante verticale de la vitesse initiale.) • Nous
- pouvons vérifier que y(0) = 25, comme souhaité. C'est l'altitude du bouche d'une pièce d'artillerie.

Il faut maintenant calculer ce que la question cherche réellement à savoir. Elle cherche à connaître la portée du projectile (sa distance parcourue). Il faut donc calculer le temps de vol et le multiplier par la vitesse horizontale. Pour trouver le temps de vol, il suffit de déterminer le moment où le projectile touche le sol (fin du vol), car le vol commence à t = 0. Bien sûr, lorsque le projectile touche le sol, l'altitude est nulle. Par conséquent, on pose y(t) = 0 et on utilise la formule quadratique. 0 = y(t) 0 = −4,90332t

$$^2 + 259{,}807t + 25$$

$$t = \frac{-259{,}807 \pm (259{,}807)2 - 4(-4{,}90332)(25)}{2(-4{,}90332)}$$

$$t = -0{,}0960512 \text{ ou } 53{,}0820$$

Nous vérifions notre travail avec y(53,0820) = 0,0174218 m, soit une différence de 17 mm, ce qui n'est pas un problème, mais qui est évidemment une simple erreur d'arrondi. Ensuite, nous avons besoin de la vitesse horizontale

$$(300)(\sin 30^\circ) = (300)(1/2) = 150$$

et enfin la portée du projectile est

$$(150)(53{,}0820) = 7962{,}30$$

Puisqu'un mile équivaut à 1 609 mètres, une portée de 7 962,30 m semble plausible. Après tout, le but de l'artillerie est de frapper des cibles à plusieurs kilomètres de distance. Il faut cependant comprendre que l'altitude de la colline, soit 25 m, est une mesure précise probablement au mètre ou demi-mètre près, et non à six chiffres significatifs. La vitesse initiale n'est probablement connue qu'au mètre près.

Machine Translated by Google

1 %. Il est donc préférable de donner la réponse avec deux chiffres significatifs et de répondre que la portée du projectile est de 7 900 à 8 000 mètres.

2.5.2. Notre deuxième exemple de trajectoires balistiques Nous allons maintenant explorer un problème de défense aérienne. Nous allons simuler l'ordinateur d'un système radar de défense aérienne qui suit les ogives entrantes. Ce n'est pas un problème irréaliste. Pendant la première guerre du Golfe (1991), l'Irak a tiré des missiles SS-1 « Scud » sur de longues distances vers Israël et l'Arabie saoudite. Certains auraient pu être abattus en vol par des missiles sol-air, mais de nombreux Scud sont arrivés. Il est utile de calculer le point d'impact : après tout, si une ogive se dirige vers le désert, c'est bien, mais si elle se dirige vers un centre commercial, c'est une urgence. Au moment où ce livre allait être publié chez l'éditeur en juillet 2014, des calculs similaires étaient effectués par le système de défense aérienne israélien « Dôme de Fer », alors soyez assurés que ces techniques sont plutôt pertinentes.

Pour garder les chiffres et le modèle simples, nous allons travailler avec un problème il s'agit d'obus d'artillerie, et non de missiles balistiques régionaux.

Un projectile a été détecté à proximité de la batterie de défense aérienne dont notre ordinateur est responsable. Le système de coordonnées du radar est défini par l'ordinateur (0, 0, 0). Un instantané radar du projectile a été pris, indiquant sa position et sa vitesse. L'axe y est vertical, l'axe x est plein est et l'axe z est plein sud. Le radar indique que le projectile se trouve à une altitude de (12 000 ; 5 000 ; 7 000) mètres, avec une vitesse de (−60 ; −240 ; 80) m/s. Le point d'impact doit d'abord être calculé. Par souci de simplicité, supposons que t = 0 soit l'heure de l'instantané, et non l'heure du lancement.

Tout d'abord, nous devrions utiliser la méthode en sept étapes pour trouver y(t), la fonction d'altitude. Je vous laisse le faire vous-même pour vous entraîner. Cependant, je vous donnerai les réponses après chaque étape pour que vous puissiez vérifier votre travail.

(1) Première étape : $a(t) = -9{,}80665$ (2)
Deuxième étape : $v(t) = -9{,}80665t + C$ (3)
Troisième étape : $v(t) = -9{,}80665t - 240$ (4)
Quatrième étape : $y(t) = -4{,}90332t^2 - 240t + C - 240t + 5000$
(5) Cinquième étape : $y(t) = -4{,}90332t^2$ (6)
Sixième étape :

- La première dérivée est $y(t) = -9{,}80664t - 240$, correspondant à v(t). • La seconde dérivée est $y(t) = -9{,}80664$, correspondant à y(t). • L'altitude à t = 0 est y(0) = 5000, comme souhaité. • L'altitude à t = 0 est y (0) = −240, comme souhaité.

(7) Étape sept : Nous devons trouver le temps d'impact, tout comme le dernier problème. Cependant, cela ne se produit pas à chaque fois. Quoi qu'il en soit, l'impact est relativement proche, à t = 15,7593 secondes.

Machine Translated by Google

(8) Étape sept, suite : À une vitesse de $\dot{x} = -60$ m/s, en 2,00148, le projectile se sera déplacé de −945,558 dans la direction x ; à une vitesse de $\dot{z} = +80$, le projectile se sera déplacé de 1260,74 dans la direction z.
Bien sûr, à l'impact, y = 0.

(9) Le point d'impact est donc (−11, 054,4 ; 0 ; 8260,74).

Remarque : les symboles $\dot{x}$ et $\dot{z}$ ne sont que des abréviations pour dx/dt et dz/dt.

2.5.3. Tir de contre-batterie : Dans la réalité, une technique amusante consiste à tracer mathématiquement la parabole de vol vers l'arrière pour détecter le point de lancement. Une fois ce point connu, il est clair qu'une pièce d'artillerie s'y trouve, et il serait donc prudent de bombarder massivement cet endroit pour détruire la batterie d'artillerie ennemie. Cette technique, appelée « tir de contre-batterie », a été développée pendant la Première Guerre mondiale et est depuis lors un pilier de la guerre moderne. Un autre avantage de cette technique est qu'elle utilise la « mauvaise valeur » de t pour résoudre y(t) = 0. En effet, on utilise la racine avec t < 0, ce qui est rarement le cas dans les problèmes de mouvement de projectile.

L'instant de lancement est lorsque y(t) = 0 mais t < 0. (En fait, nous supposons implicitement que les altitudes du lanceur, de l'observateur et de l'impact sont toutes égales ou presque égales. Cependant, si elles n'étaient pas égales, les ajustements seraient simples.) L'équation quadratique nous révèle que dans ce cas, t = −64,7057 secondes. Cela semble raisonnable, car cela donne un temps de vol total de 80,4650 secondes, soit un peu plus d'une minute.

Les coordonnées de l'obusier de tir peuvent être obtenues en calculant

$$(-60)(-64{,}7057) + 12\,000 = 15\,882{,}3 \quad (+80)$$
$$(-64{,}7057) + 7\,000 = 12\,176{,}4$$

Par conséquent, nous signalons que la batterie d'artillerie ennemie se trouve aux coordonnées suivantes :

$$(15\,882{,}3 \; ; \; 0 \; ; \; 12\,176{,}4)$$

La vérité sur la résistance de l'air :

Bien sûr, nous avons négligé la résistance du vent et de l'air ; dans la réalité, il serait préférable de les inclure. Vous vous demandez probablement comment cela fonctionne concrètement.
Ce qui se passe, c'est que le temps est divisé en minuscules intervalles, d'environ 10^{-4} ou 10^{-5} secondes, et pendant ces tranches de temps, on prétend que l'effet du vent et de la résistance de l'air est une fonction constante ou une fonction linéaire.

106 ou 107 tranches de temps représentent la trajectoire totale du projectile. Grâce aux techniques que vous apprendrez dans les premières parties du chapitre 5, il est extrêmement facile d'écrire un programme informatique permettant d'avancer dans le temps de manière répétée, par tranches de temps minuscules.

La modification par une fonction constante ou une fonction linéaire ne va augmenter que légèrement la complexité du calcul pour chaque tranche de temps.
Cependant, comme les tranches de temps sont si petites, la précision de l'ensemble

Machine Translated by Google

Les calculs peuvent être très complexes. Le concept général dont nous discutons ici est appelé « simulation numérique ».

2.5.4. Votre défi : une fusée à plusieurs étages. Maintenant que vous avez pratiqué les techniques de déplacement d'un projectile, considérons le vol d'essai d'une fusée. Dans ce cas, la fusée tombe en panne et s'écrase, et nous allons modéliser sa trajectoire.

En fonctionnement normal, le moteur fournit une poussée constante de 1,27 méganewton. Cependant, onze secondes après le début du vol, l'ordinateur de commande de la pompe à carburant est tombé en panne et le débit de carburant vers le moteur a été lentement réduit à zéro. Bien sûr, la fusée s'est écrasée. Un bon modèle de la poussée du moteur après la panne serait

$$f(t) = 1{,}27e\ (t-11)/2 \text{ pour tout } t > 11$$

Encore une fois, la masse est mesurée en méganewtons. Il est utile de savoir que la masse de la fusée au lancement était de 107 000 kilogrammes.

Nous avons deux hypothèses simplificatrices. Bien que du carburant soit clairement brûlé pendant les onze premières secondes, on peut considérer que la masse est constante pendant toute la durée du problème. (C'est parce que seule une infime partie du carburant a été brûlée pendant le vol.) Deuxièmement, la fusée ne monte pas très haut, et on peut donc considérer que $g = 9{,}80665$ m/s² est une constante.
En réalité, cela change lorsque vous atteignez des altitudes de plus en plus élevées, mais un projectile doit atteindre une hauteur relativement élevée avant que cela ne devienne trop important.

Commençons. Tout d'abord, calculez la fonction d'altitude lorsque le moteur fonctionnait normalement, à savoir $0 < t < 11$. On peut supposer que la fusée est restée sur la rampe de lancement, à $y = 0$, longtemps avant le lancement. Une fois ces informations obtenues, calculez l'altitude, la vitesse et l'accélération 11 secondes après le lancement, au moment où le dysfonctionnement s'est produit.

Ensuite, à l'aide de la fonction f(t) ci-dessus, calculez une nouvelle fonction d'accélération, une nouvelle fonction de vitesse et une nouvelle fonction de position, à utiliser après le dysfonctionnement. Cela nous permettra de déterminer l'heure de l'impact.

Il y a dix éléments livrables pour ce projet :

- Premier, deuxième et troisième, fournissent les fonctions d'accélération, de vitesse et de position avant le dysfonctionnement. • Quatrième, cinquième et sixième, fournissent les fonctions d'accélération, de vitesse et de po- fonctions de fonction après le dysfonctionnement.
- Septièmement, indiquez l'heure de l'impact. • Huitièmement, fournissez un graphique unique montrant l'altitude pour l'ensemble du vol, du lancement à l'impact. • Neuvièmement, fournissez un graphique unique montrant la vitesse pour l'ensemble du vol, du lancement à l'impact. • Dixièmement, indiquez la hauteur maximale atteinte.

Pour vous aider à comprendre si vous avez résolu le problème correctement, les deux graphiques sont donnés dans la figure 2.

Machine Translated by Google

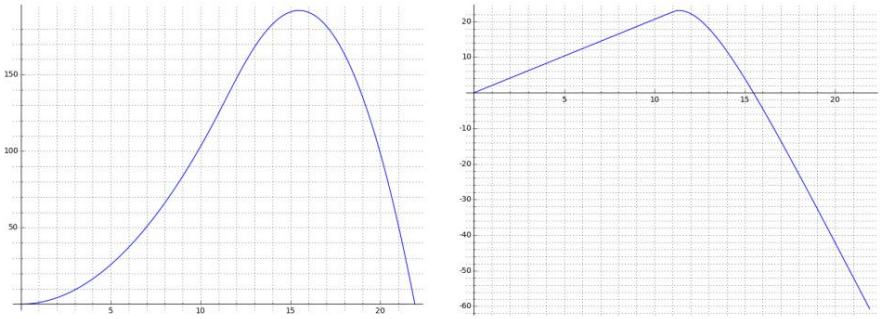


Figure 2. Les graphiques du projet de physique

2.6. Cryptologie : attaque p−1 de Pollard sur RSA

Le cryptosystème RSA est sans doute le plus célèbre des systèmes cryptographiques utilisés aujourd'hui. Presque toutes les transactions de commerce électronique sur Internet utilisent TLS (Transport Layer Security) ou SSL (Secure Sockets Layer) et utilisent donc RSA pour échanger des clés de chiffrement par blocs. La sécurité de RSA repose sur la difficulté de factoriser le produit de deux nombres premiers (énormes).

Ce projet est l'une des nombreuses façons d'attaquer le cryptosystème RSA. Ce projet suppose une connaissance de RSA et de quelques notions de théorie des nombres. Si le lecteur ne maîtrise pas RSA, ce projet ne peut être entrepris. J'encourage cependant ce type de lecteur à apprendre le RSA. Il est relativement facile à apprendre, du moins comparé à d'autres concepts mathématiques, et il offre un aperçu des applications pratiques de sujets par ailleurs « purs » comme les nombres premiers et l'arithmétique modulaire.

Ce projet suit de près la notation donnée dans Cryptography and Cod-ing Theory, deuxième édition, de Wade Trappe et Lawrence Washington, publié par Pearson en 2005. Voir le chapitre 6.4 de ce manuel. La méthode originale a été découverte par John Pollard en 1974.

Par ailleurs, Minh Van Nguyen a écrit un tutoriel sur Sage et la cryptographie, prenant RSA comme exemple. Intitulé « Théorie des nombres et cryptosystème à clé publique RSA », ce tutoriel est disponible à l'adresse http://www.sagemath.org/doc/thematic_tutorials/ numtheory_rsa.html . Enfin, ce projet nécessite une connaissance des boucles for, abordées au chapitre 5 de ce livre.

2.6.1. Contexte : Nombres B-Lisses et B!

Les théoriciens des nombres qualifient souvent certains entiers positifs de « lisses ». Ce terme vague signifie que le nombre n'a pas de grands facteurs premiers, mais que sa factorisation est entièrement composée de petits nombres premiers. Plus précisément, pour tout nombre pair k, on dit qu'un nombre est k-lisse si tous ses facteurs premiers sont inférieurs à k. Par exemple, 15, 10, 100 et 25 000 sont tous 6-lisse, mais 14 ne l'est pas, car 14 = 2 × 7 et

Machine Translated by Google

7 est plus grand que 6. Autre exemple : 14 et 1400 sont lisses en 8, mais 11 ne l'est pas. Cette notion peut être intéressante dans plusieurs branches de la théorie des nombres, mais elle est particulièrement intéressante en cryptographie. C'est la définition standard de ce qu'est un nombre « lisse ».

Il se trouve qu'une clé publique RSA peut facilement être déchiffrée (en factorisant son N) si p − 1 ou q − 1 sont lisses. C'est une affirmation assez étrange, bien sûr. Tout d'abord, p et q ne sont pas des informations publiques. Cependant, cela ne constitue pas un obstacle à cette attaque, car il suffit que p − 1 ou q − 1 soient lisses ; nous n'avons pas besoin de connaître p − 1 ni q − 1, ni même lequel des deux nombres premiers est « un plus un nombre lisse ».

Une autre particularité est que nous allons utiliser une définition non standard de la régularité. Prenons un grand nombre B et demandons-nous si p ou q divise B!. Nous utiliserons bientôt B = 10 000, mais pour l'instant, nous utiliserons B = 100.
Explorons comment cette utilisation de 100 ! se compare à un nombre lisse à 100.

La seule façon pour qu'un nombre z ne soit pas lisse à 100 est si z possède un nombre premier dans sa factorisation en nombres premiers supérieur à 100, comme par exemple 101 ou 103. Puisque 101 et 103 sont premiers, et tous deux supérieurs à 100, nous savons que ni 101 ni 103 ne divisent aucun des nombres de 1 à 100. Par conséquent, par le célèbre lemme de la théorie des nombres selon lequel p|ab si et seulement si p|a ou p|b, nous pouvons conclure que 101 ne divise pas 100!, et de même 103 ne divise pas 100!. De même, tout multiple de 101 ou 103 ne divise pas 100!. Il en va de même pour tout nombre premier supérieur, comme 107, etc.
Par conséquent, tout nombre dont le nombre premier dans sa factorisation est supérieur à 100 ne divisera certainement pas 100 !

Cependant, l'inverse pourrait nous intéresser. Si un nombre y ne divise pas 100 !, y n'est-il pas toujours lisse à 100 ? Malheureusement, nous rencontrerons ici quelques exceptions.
Utilisation des commandes

```
facteur( factoriel( 100 ) )
```

nous apprenons que

$$100! = (2)97(3)48(5)24(7)16(11)9\ (13)7\ (17)5\ (19)5\ (23)4\ (29)3\ (31)3\ (37)2 \cdot \cdot \cdot \cdot \cdot (41)2$$
$$(43)2\ (47)2\ (53)(59)(61)(67)(71)(73)(79)(83)(89)(97)$$

On voit donc que 298, 349 et 525 ne divisent pas 100!. Cependant, bien que ces trois nombres soient tous lisses à 6, ils sont donc certainement lisses à 100!. Nous avons donc trouvé trois exemples de nombres lisses à 100 qui ne divisent pas 100!.

En fait, nous avons maintenant démontré qu'exiger que les nombres soient des diviseurs de 100 ! est une norme plus stricte que d'exiger qu'ils soient lisses à 100. En général, l'ensemble des nombres divisant B ! est un sous-ensemble des nombres lisses à 100. Cependant, globalement, si l'on considère les nombres inférieurs à B !, les exceptions sont rares et les concepts sont à peu près équivalents.

Machine Translated by Google

2.6.2. La théorie derrière l'attaque Il nous faut un autre

théorème de la théorie des nombres. Pour tout nombre premier p, et pour tout a ≡ 0 mod p, le petit théorème de Fermat stipule que

$$a^p \equiv a \bmod p \text{ ou de manière équivalente } a \equiv 1 \bmod p^{p-1}$$

Supposons que p − 1 divise B!. Cela signifie qu'il existe un nombre k tel que (k)(p − 1) = B!. On peut alors conclure que = 2(k)(p−1) = (2p−1).

$$2^{B!} \qquad k \equiv 1^{k} \equiv 1 \text{ à } p$$

Puisque 2B! ≡ 1 mod p, il doit en être ainsi : (2B! − 1) ≡ 0 mod p, ou

plus simplement, c = (2B! − 1)

est un multiple de p. Ce fait essentiel, que c est un multiple de p, est au cœur de toute l'idée.

En fait, à bien y réfléchir, il doit être vrai que c est un multiple de p dans les entiers ordinaires, et que l'image de c mod N est un multiple de p dans « les entiers mod N ». La plupart des lecteurs prendront ce point plutôt technique au sérieux et pourront passer directement au paragraphe suivant. Pour ceux qui recherchent la rigueur absolue, notez que si c est un multiple de p, alors c = jp, pour un entier j. Nous n'avons aucune raison de croire que j est divisible par q, mais divisons quand même par j, ce qui donne le quotient j1 et le reste j2. Cela signifie que j = qj1 + j2 , et donc

c = jp = (qj1 + j2)p = pqj1 + pj2 = N j1 + pj2 ≡ 0 + pj2 mod N ce qui nous permet de conclure que l'image de c mod N est simplement pj2, clairement un multiple de p.

Qu'avons-nous gagné à cela ? Puisque N = pq, nous savons que N est également un multiple de p. Puisque c et N sont tous deux multiples de p, alors pgcd(c, N) est également un multiple de p. En fait, pgcd ne peut avoir que deux valeurs : p ou N.

Par conséquent, nous calculons d'abord c = 2B! − 1 mod N, ce qui n'est pas une mince affaire, puis nous calculons la valeur de pgcd(c, N). Si nous obtenons une valeur comprise entre 1 et N, nous savons que ce doit être p, et une division ordinaire nous donnera q = N/p. Nous avons factorisé N et avons déchiffré la clé publique de cette personne.

En revanche, si le pgcd est N, nous n'avons pas de chance et avons échoué. Le pgcd sera N (ce qui entraînera un échec de l'attaque) si et seulement si q divise également c. Cela ne se produira essentiellement que si p − 1 et q − 1 divisent tous deux B!.

Il est rare que p − 1 ou q − 1 divise B!. En revanche, que p − 1 et q − 1 divisent B! à la fois est extrêmement rare. Plus précisément, p et q sont supposés être générés aléatoirement, et

---

[3] Si cela vous arrive un jour en pratique, réessayez avec un B plus petit. Si le plus grand nombre premier dans la factorisation première de p − 1 est f1 et que le plus grand nombre premier dans la factorisation première de q − 1 est f2, alors choisir un B entre f1 et f2, vraisemblablement par devinette et vérification, donnera un résultat positif, à savoir pgcd(c, N) = p.

Machine Translated by Google

par conséquent, la probabilité que p−1 divise B! est indépendante de la probabilité que q−1 divise B!. La probabilité que les deux divisent B! est donc le carré de la probabilité que l'un d'eux divise B!.

En conséquence, nous nous attendons pleinement à ce que l'attaque fonctionne pour n'importe quel N, tant que comme p − 1 ou q − 1 divise B!.

2.6.3. Calcul 2 (B!) mod N Réfléchissons

au fait que le calcul

$$2\ (10\ 000\ !) \text{ contre } N$$

Ce n'est pas une tâche qu'on nous demande souvent. Il serait donc imprudent de se lancer dans ce problème sans y avoir d'abord réfléchi. On pourrait se demander : quelle sera la taille exacte de 10 000 ! ? Ou, plus précisément, combien de chiffres comportera-t-il ?

Il s'avère que cette question est facile à répondre si l'on se souvient de l'approximation de Stirling :

$$B! \approx B \text{ Soit } -B \sqrt{2\pi B}$$

et procédez en prenant le logarithme décimal

$$\log10(B!) \approx \log10\ B \quad B \quad \text{et} -B \sqrt{2\pi B}$$

$$\approx \log10\ B \quad B \quad + \log10\ \text{et}\ ^{-B} \quad + \log10 \sqrt{2\pi B}$$

$$1 \approx (B \log10\ B) - (B \log10\ e) + (\log10\ 2\pi B)\ 2$$

$$\approx B\ (\log10\ B - \log10\ e) + (\log10\ 2\pi) + 2\ 2 \quad 1 \quad - \quad 1 \quad (\log10\ B)$$

$$\approx B\ (\log10\ B - 0{,}434294481 \cdots) + 0{,}399089934 \cdots + \quad 1 \log10\ B$$

2 . . . ou dans notre cas de B = 10 000 et log10 B = 4. . . 1 ≈ 10 000 (4

− 0,434294481 · · ·) + 0,399089934 · · · + (4) ≈ 35 657,0551 · · · + 2,399089993 ·

· · ≈ 35 659,4542 · · · 2

Ayant réalisé que 10 000 ! est un nombre à 36 660 chiffres (environ), nous ne devrions probablement pas le calculer puis demander 2 (10 000 !). Le calcul pourrait être très long. Nous devons trouver une autre approche.

Machine Translated by Google

Pensez à l'informatique

$$c1 = 21 \bmod N \quad c2 = (c1)^2 \text{ vers le } N$$
$$c3 = (c2)^3 \text{ vers le } N$$
$$c4 = (c3)^4 \text{ vers le } N$$
$$c5 = (c4)^5 \text{ vers le } N$$
$$c6 = (c5)^6 \text{ vers le } N$$
$$\vdots \quad \vdots \quad \vdots$$

Alors nous aurions

$$c6 \equiv (c5)^6 \equiv ((c4)^5)^6 \equiv (((c3)^4)^5)^6 \equiv ((((c2)^3)^4)^5)^6 \equiv (((((c1)^2)^3)^4)^5)^6 \equiv 2^{(1)(2)(3)(4)(5)(6)}$$

mais observez, c'est exactement ce que nous voulions calculer, à savoir 2

$$(1)(2)(3)(4)(5)(6) \equiv 2\ (6!)$$

De la même manière, nous pouvons voir que $cB \equiv 2\ (B!) \bmod N$.

2.6.4. Votre défi : Réaliser tout cela dans Sage. Votre tâche est simple. En utilisant la stratégie ci-dessus, calculez

$$c = 2(10\ 000!) \text{ contre } N$$

Ensuite, calculez p = pgcd(c, N), suivi de q = N/p, et vérifiez votre travail en multipliant p et q. Voici sept cas de test pour

vous : 357177030895805008781319266876641103581839678151495135129133067010127
500027200225181154044657958681603934630878556564186233190586076312373 3
111293094034769304884167051458174320813595510448138274866152002067365927
189114989429752082926285457551369642787381790039260802307452110490304547
250733654482468568921620042866859718852920028026076547177974758239109177
201557389900540095613559219541299540522405259329399736824858252876376521311053006710577163057234093
862021547643631582396998212208722914288258644234791307950582916442747222039795609417741932278317121

Au fait, vous devriez pouvoir copier-coller ces chiffres dans Sage à partir du fichier PDF de ce livre, disponible sur my4 . Vous n'aurez pas besoin de retaper ces chiffres énormes !

2.6.5. Protections contre l'attaque factorielle de Pollard Comme vous pouvez le constater, les clés publiques RSA peuvent être cassées (c'est-à-dire que N peut être factorisé) si p − 1 ou q − 1 sont lisses. Un nombre lisse (par exemple, un nombre lisse à 6 ou à 100) possède un nombre relativement important de facteurs premiers, chacun contribuant faiblement au produit final. Par exemple,

$$10100 = 21005\ 100$$

4http://www.gregorybard.com/

Machine Translated by Google

est un nombre très lisse, avec 200 nombres premiers dans sa factorisation (en comptant les multiplicités) et aucun ne contribue plus de 5 à l'énorme produit final. Quel est l'extrême opposé ?

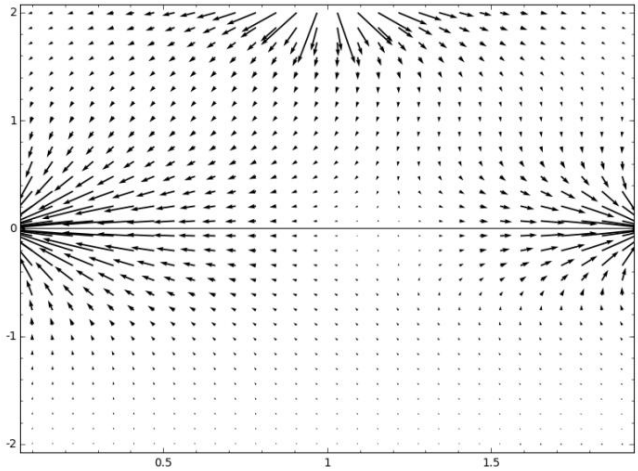
L'autre extrême serait un nombre premier. Il n'existe qu'un seul facteur premier, et il contribue à lui seul à la grandeur. Bien sûr, p et q sont impairs et donc ni p − 1 ni q − 1 ne peuvent être premiers.
Cependant, ils pourraient être deux fois un nombre premier. Les nombres premiers p où p et (p−1)/2 sont tous deux premiers simultanément sont appelés « nombres premiers sûrs ». La notion presque identique, où p et 2p+1 sont tous deux premiers, est appelée nombres premiers « Sophie Germain », du nom de Marie-Sophie Germain (–).

Pour rendre l'attaque de Pollard totalement impossible, de nombreuses implémentations RSA exigent que les deux nombres premiers soient des nombres premiers sûrs. Pour ces nombres, B devrait être égal ou supérieur à (p − 1)/2. Même si p ne mesure que 200 bits, ce qui est bien trop petit, alors (p − 1)/2 mesurera 199 bits, ce qui rendrait le calcul de 2B! impossible . Le soleil serait à court de carburant et se transformerait en géante rouge5 bien avant qu'un tel calcul ne puisse être terminé.

2.7. Mini-projet sur les tracés vectoriels du champ électrique

Prenez un moment pour lire la section 3.7 : Tracés de champs vectoriels, à partir de la page 114, si ce n'est pas déjà fait. Dans ce mini-projet, il vous sera demandé de modifier le modèle utilisé dans la sous-section « Une application de la physique » pour générer le tracé de champs vectoriels de trois charges électriques. Nous ajouterons une quatrième charge électrique, avec q3 = −1, et la collerons au point (1, 2,25).

Votre tâche consiste à modifier mon code pour intégrer cette quatrième charge électrique, puis à générer le tracé du champ vectoriel. Vous devriez obtenir l'image suivante :



Cela se produira dans environ 5 milliards d'années.

Machine Translated by Google

Chapitre 3

Tracé avancé Techniques

L'une des utilisations les plus agréables de Sage est la production d'images graphiques colorées, que ce soit pour l'enseignement ou la publication. Sage propose une vaste gamme d'utilitaires de traçage et de création de graphiques. Les bases du traçage ont été présentées dans la section 1.4, page 9. Si vous ne l'avez pas encore lu, vous devriez le lire maintenant. Presque tout ce que vous pourriez avoir besoin de tracer est abordé dans ce chapitre. L'exception concerne les tracés très colorés ou en 3D, qui ne peuvent apparaître que sur un écran d'ordinateur et non dans un livre imprimé. Ces tracés sont abordés dans l'annexe en ligne de ce livre, « Traçage en couleur, en 3D et animations », téléchargeable sur ma page web www.gregorybard.com.

De plus, comme les nuages de points ne sont utilisés que pour les régressions, leur technique est abordée à la section 4.9, page 157, par souci de simplicité. De même, les diagrammes de champ de pente ne sont utilisés (à ma connaissance) que pour l'étude des équations différentielles du premier ordre ; ces procédures sont donc abordées à la section 4.22.3, page 195, pour une meilleure continuité de l'exposé.

3.1. Annotation des graphiques pour plus de clarté

Dans cette section, nous présentons un ensemble de techniques permettant de rendre les tracés 2D plus attrayants. Parmi celles-ci, on peut citer l'étiquetage des axes, l'ajout de lignes de quadrillage, l'ajout d'un cadre, le marquage d'un point avec un gros point, le dessin d'une flèche sur un tracé, l'insertion de texte, l'ombrage des tracés (pour les intégrales) et l'utilisation de lignes pointillées ou discontinues.

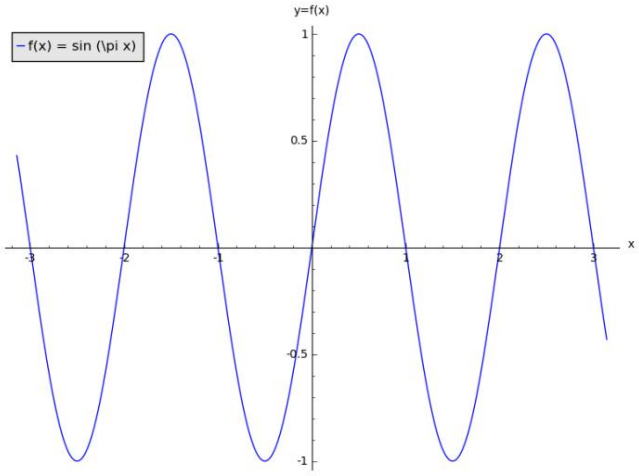
3.1.1. Étiquetage des axes des graphiques. Il est

parfois très utile d'étiqueter les axes d'un graphique, directement sur celui-ci. C'est particulièrement pratique en cas de risque de malentendu sur les unités utilisées, ou dans une application scientifique où de nombreuses fonctions sont traitées. Quoi qu'il en soit, le code suivant atteint cet objectif.

Machine Translated by Google


```
p= plot(sin(pi*x), -pi, pi, legend_label='f(x) = sin (pi x)' ) p.axes_labels(['x', 'y=f(x)'])
p.show()
```

Le code ci-dessus, fourni par Martin Albrecht et Keith Wojciechowski, produit l'image suivante :



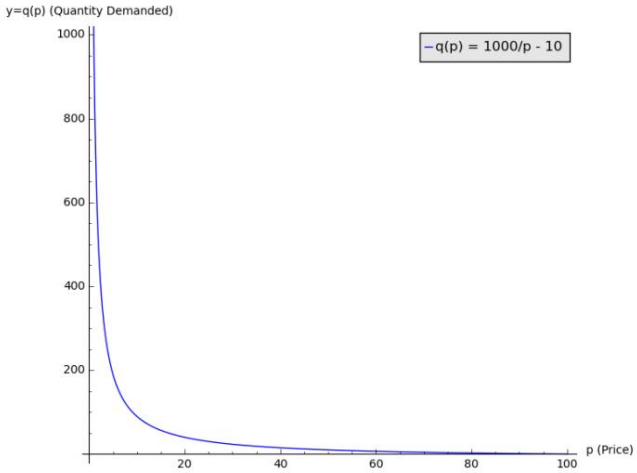
Voici un autre exemple économique : la courbe de demande.

$$q(p) = -10 \frac{1000}{p}$$

relie la quantité demandée par le marché d'un produit à son prix.
Voici un code permettant de tracer la courbe ci-dessus, avec des étiquettes informatives.

```
p = plot(1000/x - 10, 0, 100, ymax = 1000,
                legend_label= 'q(p) = 1000/p - 10' ) p.axes_labels(['p
(Price)', 'y=q(p) (Quantity Demanded)']) p.show()
```

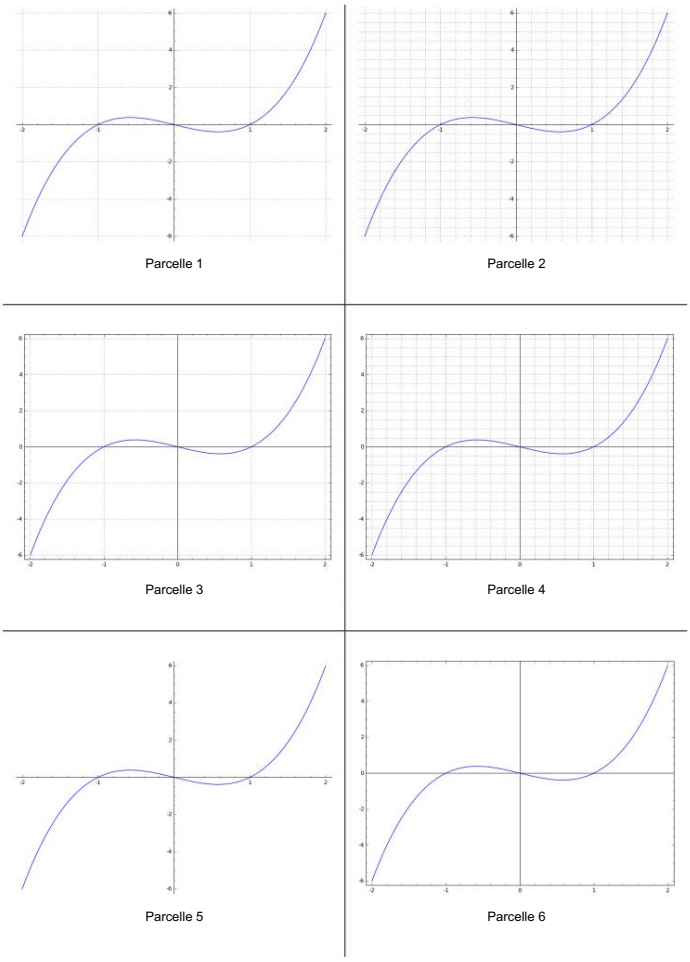
L'image produite par ce code est ci-dessous.



Machine Translated by Google

3.1.2. Grilles et graphiques de type calculatrice

Parfois, dans un graphique, il est intéressant de montrer les lignes de la grille, comme si le graphique était dessiné sur une feuille de papier millimétré. Il existe deux options : cadre, qui peut être vrai ou faux, et lignes de grille, qui peuvent être vraies, Faux, ou « mineur ». Il existe donc en réalité six choix différents disposition. Ils sont donnés dans la collection de tracés suivante.



Machine Translated by Google

Voici le code qui a généré chacun de ces tracés.
Parcelle 1 :

```
plot(x^3-x, -2, 2, gridlines=True, frame=False)
```

Parcelle 2 :

```
plot(x^3-x, -2, 2, gridlines='minor', frame=True)
```

Parcelle 3 :

```
plot(x^3-x, -2, 2, quadrillage=True, cadre=True)
```

Parcelle 4 :

```
plot(x^3-x, -2, 2, gridlines='minor', frame=True)
```

Parcelle 5 :

```
plot(x^3-x, -2, 2, gridlines=False, frame=False)
```

Parcelle 6 :

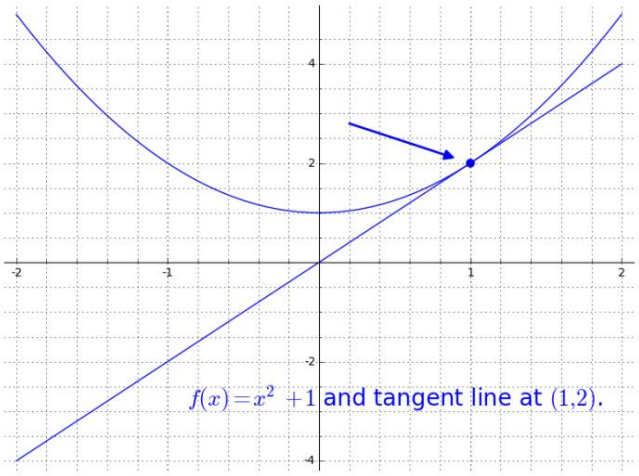
```
plot(x^3-x, -2, 2, gridlines=False, frame=True)
```

3.1.3. Ajout de flèches et de texte aux fonctionnalités d'étiquette Sage

dispose de quelques techniques intéressantes pour rendre les graphiques plus faciles à comprendre. Nous avons déjà vu comment la commande « point » permet de dessiner un gros point sur un tracé afin d'attirer l'attention du spectateur. Si nécessaire, la commande « flèche » permet de dessiner une flèche large et discrète.
La flèche est définie par deux points : sa queue et sa tête. Les coordonnées de la queue sont indiquées en premier, puis celles de la tête. La syntaxe est présentée ci-dessous. La commande texte permet également d'ajouter des mots directement au tracé. Elle possède deux paramètres : le message à ajouter et le centre des mots souhaités.

```
tangent_line = plot( 2*x, -2, 2, gridlines='minor' ) curve = plot( x^2+1, -2, 2 )
big_dot = point( (1,2), size=60 )
the_arrow = arrow( (0.2, 2.8), (0.9, 2.1) )
the_words = text( '$f(x) = x^2 + 1$ et ligne tangente à $(1,2)$.',
(0.5, -2.75), fontsize=20 )

grande_image = ligne_tangente + courbe + gros_point + la_flèche + les_mots grande_image.show()
```

Machine Translated by Google



Enfin, il est important de noter que la commande texte prend en charge le langage de préparation de documents mathématiques « LaTeX ». Si vous ne connaissez pas LaTeX, vous pouvez saisir the_words = text('f(x) = x^2 + 1 and

tangente.',

(0,5, -2,75), taille de police = 20)

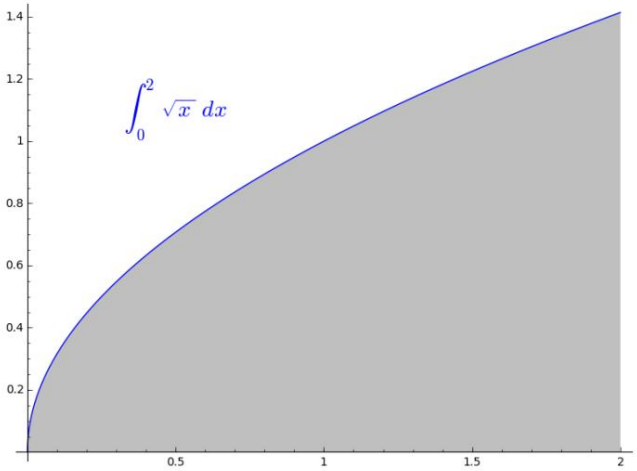
et ont presque le même effet.

3.1.4. Représentation graphique d'une

intégrale Voici un autre exemple pour montrer la puissance de LaTeX, ainsi que la manière de représenter graphiquement avec un ombrage, par exemple lors de l'enseignement des intégrales.

```
P1= texte('$\int_0^2 \sqrt{x} \, dx$', (0.5, 1.1), taille de police=20)
P2= plot( sqrt(x), 0, 2, fill=True)
P= P1 + P2
P.show()
```

L'image suivante est produite :

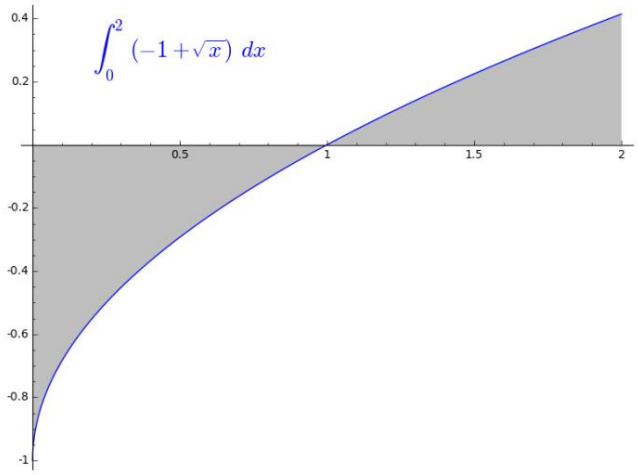


Machine Translated by Google

La commande de remplissage est excellente pour travailler avec des intégrales, même celles de fonctions qui croisent l'axe des abscisses. Cependant, pour d'autres types de tracés remplis, comme la représentation graphique de systèmes d'inéquations, d'autres techniques sont bien plus souples et précises. Ce sujet sera abordé dans l'annexe en ligne de ce livre, « Tracés en couleur, en 3D et animations », téléchargeable sur mon site web www.gregorybard.com.

```
P1 = texte('$\int_0^2 (-1+\sqrt{x}) \, dx$', (0.5, 0.3), fontsize=20)

P2 = plot( -1+sqrt(x), 0, 2, fill=True)
P = P1 + P2
P.show()
```

Comme vous pouvez le voir sur la capture d'écran ci-dessous, Sage sait dans quel sens à ombrer selon que la fonction est positive ou négative.



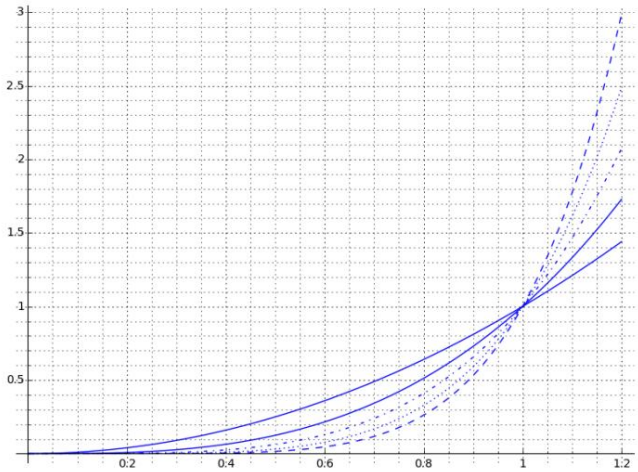
3.1.5. Lignes pointillées et discontinues

Parfois, lorsque trois fonctions ou plus sont représentées sur le même graphique, il est intéressant de les représenter visuellement. La meilleure façon d'y parvenir est d'utiliser des couleurs, mais une autre solution consiste à utiliser des lignes pointillées ou discontinues. Sage propose quatre variantes, en plus de la ligne continue classique. Le code suivant

```
P1 = plot( x^2, 0, 1.2, gridlines='minor')
P2 = plot( x^3, 0, 1.2, linestyle = P3 = plot( x^4, 0, 1.2,      '-' )
linestyle = '-.' )
P4 = plot( x^5, 0, 1.2, style de ligne = ':' )
P5 = tracé (x^6, 0, 1.2, style de ligne = '--' )
P = P1 + P2 + P3 + P4 + P5 P.show()
```

affichera chacune de ces options simultanément dans le graphique suivant :

Machine Translated by Google



D'autre part, le traçage en couleur est abordé dans l'annexe en ligne uniquement électronique de ce livre « Traçage en couleur, en 3D et animations », disponible sur ma page Web www.gregorybard.com pour téléchargement.

3.2. Graphiques de certaines fonctions hyperactives. La

fonction sin(1/x) oscille énormément près de x = 0, mais Sage le sait et trace de nombreux points supplémentaires. Il existe de nombreuses autres fonctions hyperactives en mathématiques. Voici quelques exemples et leurs images Sage. Comme vous pouvez le constater, Sage est imparfaite, mais atteint son objectif.

Graphique 1 : La fonction f(x) = sin(1/x) oscille considérablement au voisinage de x = 0. Si vous observez attentivement (0, 1) et (0, −1), vous constaterez que la « tache bleue » n'est pas uniforme. Cela est lié à la façon dont Sage trace les fonctions. Il prend l'intervalle demandé sur l'axe des x et le divise en 200 intégrales. Une valeur x aléatoire est choisie dans chaque intervalle. Sous certaines conditions, Sage peut détecter que la fonction est « compliquée » et commencer à utiliser davantage de points, plus rapprochés. Cependant, il utilisera toujours un nombre fini de points, et donc dans les régions « infiniment compliquées », il peut y avoir des non-uniformités. De plus, cela signifie qu'un graphique peut parfois avoir un aspect différent lorsqu'il est tracé deux fois de suite.

plot(sin(1/x), -0,5, 0,5)

Graphique 2 : La fonction f(x) = exp($\frac{1}{2-x}$) a une limite intéressante lorsque x tend vers 2. Si l'on approche par la gauche (avec des valeurs de x légèrement inférieures à deux), la limite est l'infini. Si l'on approche par la droite (avec des valeurs de x légèrement supérieures à deux), la limite est nulle. Par conséquent, la limite à l'approche de 2 en général « n'existe pas ». C'est un bon exemple, car je constate que les étudiants en calcul pensent souvent que les limites compliquées sont des inventions artificielles et qu'il est peu probable qu'elles se produisent un jour.

plot(exp(1/(2-x)), -1, 4, ymin=-1, ymax=5)

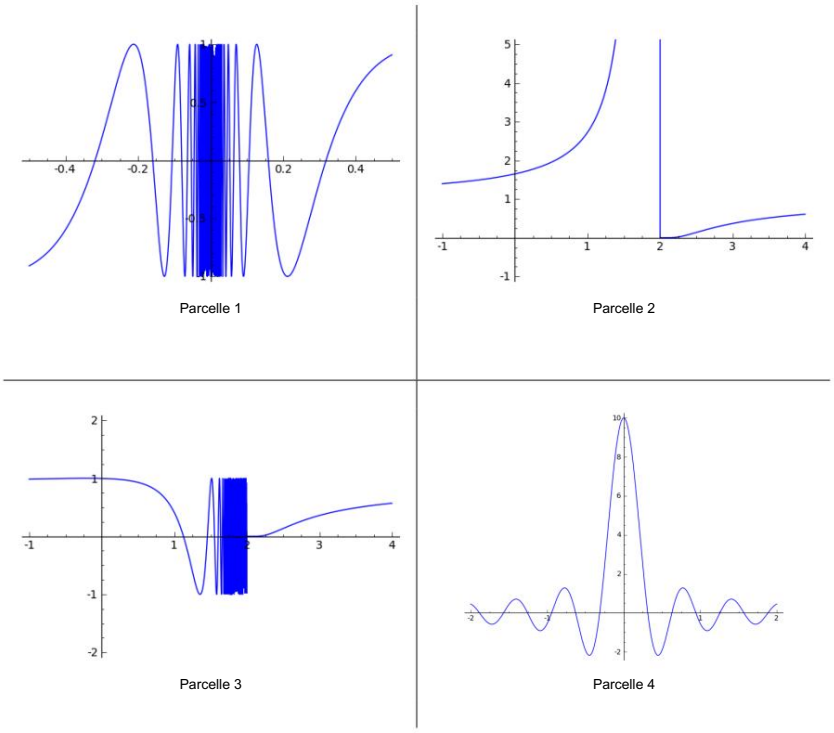
Machine Translated by Google

Graphique 3 : La fonction f(x) = sin(exp($\frac{1}{2-x}$)) est simplement le sinus de l'exemple précédent. À droite de x = 2, cela ressemble à sin(0), et il n'y a donc quasiment aucune oscillation. À gauche de x = 2, cela ressemble à une onde sinusoïdale dont la fréquence approche l'infini. Il est intéressant d'observer ce graphique à différentes échelles.

plot(sin(exp(1/(2-x))), -1, 4)

Graphique 4 : Cette fonction f(x) = sin(10x)x n'a pas de propriétés mathématiques particulières, mais elle est intéressante pour trois raisons. Premièrement, elle est visuellement attrayante. Deuxièmement, cela se produit dans l'étude des ondelettes, ce qui est intéressant en soi. Troisièmement, de nombreux étudiants annuleront les x et travailleront avec f(x) = sin(10).

plot(sin(10*x)/x, -2, 2)



3.3. Tracé polaire

L'idée générale des coordonnées polaires est qu'au lieu d'une abscisse et d'une ordonnée, on utilise un rayon r et un angle appelé θ. Pour tout point, r est la distance à l'origine. θ est défini par 0, qui correspond à l'est ou à l'horizontale à droite, et les angles positifs représentent un mouvement dans le sens inverse des aiguilles d'une montre. Ainsi, 90◦ ou π/2 radians correspondraient à une ligne droite vers le haut ou au nord, et 180◦ ou π radians à une ligne droite vers la gauche ou à l'ouest. De même, les angles négatifs représentent un mouvement dans le sens des aiguilles d'une montre. Ainsi, −90◦ ou −π/2 radians correspondraient à une ligne droite.
vers le bas ou « sud ».

Machine Translated by Google

Chaque direction de mouvement à partir de l'origine porte donc plusieurs noms. Par exemple,

$$\theta \quad \ldots, \quad \frac{-22\pi}{4}, \frac{-15\pi}{4}, \frac{-7\pi}{4}, \frac{p}{4}, \frac{9p}{4}, \frac{17p}{4}, \frac{25p}{4}, \ldots$$

sont des valeurs de θ qui représentent toutes le « nord-est ».

Dans ce contexte, représenter graphiquement une fonction en coordonnées polaires à la main est un peu complexe, car on ignore combien de tours ou de révolutions sont nécessaires autour de l'origine avant de pouvoir visualiser les parties déjà tracées de la courbe. Il est donc assez complexe de déterminer comment choisir les valeurs de θ à représenter.

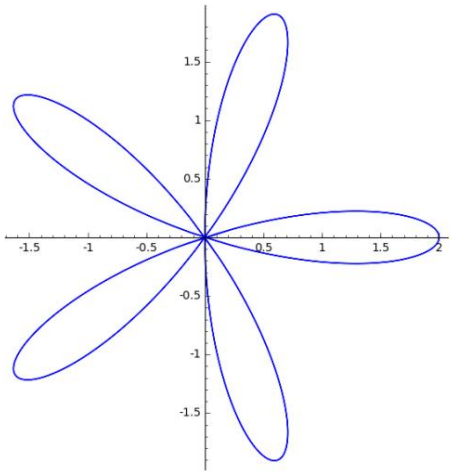
Une fonction en coordonnées polaires prend un θ en entrée et renvoie un r en sortie. Considérons

$$r(\theta) = 2 \cos 5\theta$$

qui produit un graphique appelé « rosace à 5 pédales ». Pour représenter cela dans Sage, nous saisirions

```
var("theta")
polar_plot( 2*cos(5*theta), (theta, 0, 2*pi) )
```

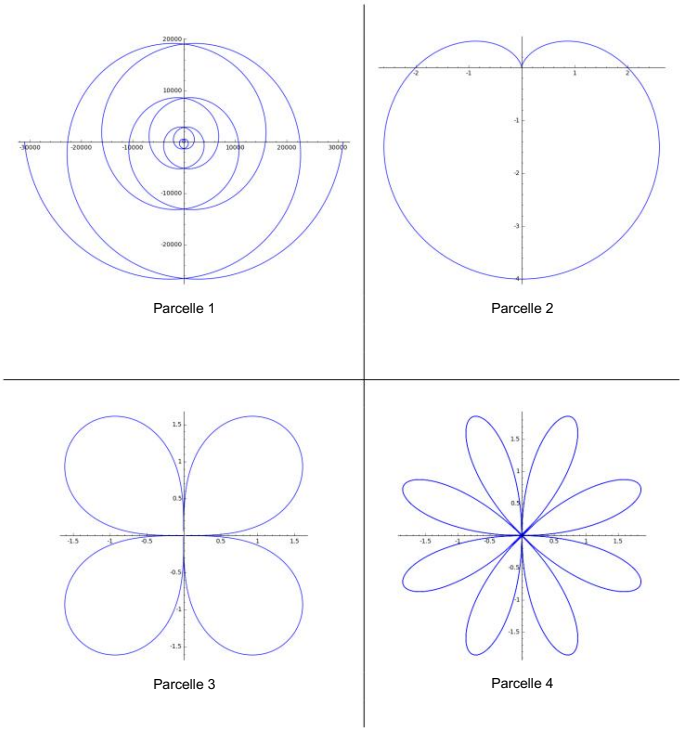
ce qui donne l'image suivante :



3.3.1. Exemples de graphiques polaires

Voici quatre autres graphiques polaires intéressants.

Machine Translated by Google



Graphique 1 : Parfois, des fonctions simples ont des graphes polaires complexes. Par exemple, $r(\theta) = \theta^3$ pour $-10\pi < \theta < 10\pi$. var("theta")

polar_plot(theta^3,

(theta, -10*pi, 10*pi))

Graphique 2 : Cette forme est appelée « cardioïde » et est donnée par

$$r(\theta) = 2 - 2\sin(\theta)$$

var("theta")

polar_plot(2 - 2*sin(theta), (theta, 0, 2*pi))

Graphique 3 : Voici une forme de trèfle. Remarquez que les pédales ont une Forme différente de celle des rosaces. La fonction est $r(\theta) = 2\sqrt{|\sin(2\theta)|}$. var("theta")

polar_plot(2*sqrt(abs(sin(2*theta))), (theta, -pi, pi))

Graphique 4 : Il s'agit d'une rosace à 8 pédales, donnée par $r(\theta) = 2\sin(4\theta)$.

var("theta")

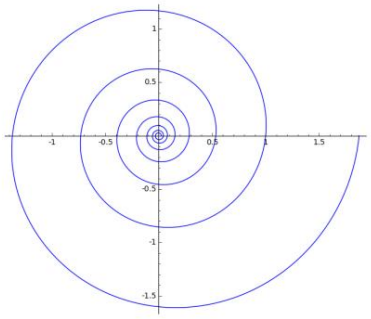
polar_plot(2*sin(4*theta), (theta, -2*pi, 2*pi))

3.3.2. Problèmes qui peuvent survenir occasionnellement La courbe suivante produit une belle spirale exponentielle. polar_plot(exp(theta/10), (theta,-12*pi,2*pi), plot_points=10000)

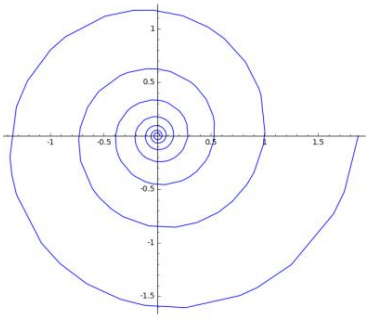
Cette courbe nécessite de spécifier un grand nombre de points – dans ce cas, 10 000 points – pour le tracé. Pour illustrer cela, considérons les deux points suivants.

Machine Translated by Google

Images. Pour l'image de gauche, l'option « Points de tracé » a été utilisée, tandis que pour l'image de droite, elle a été supprimée ; la valeur par défaut est de 200.



Tracé avec 10 000 points



Tracé avec 200 points

Comme vous pouvez le constater, la qualité de l'image est bien inférieure lorsque l'on trace seulement 200 points au lieu de 10 000. On peut comprendre ce phénomène en considérant que le domaine est $-12\pi < \theta < 2\pi$, et donc 200 points ne représentent que

$$200/14\pi \approx 4{,}54728 \cdots \text{ points par radian}$$

Ce qui pose problème, car un radian mesure environ 57 degrés. En revanche, avec 10 000 points, on obtient 227,364 points par radian, soit 50 fois plus. Si vous observez attentivement la spirale exponentielle à 200 points, vous constaterez que Sage dessine une courbe linéaire par morceaux, une séquence de segments reliant les 200 points estimés.

Voici un autre exemple, nécessitant un grand nombre de points pour une raison bien différente. Cette courbe est appelée lemniscate. Le code permettant de générer le tracé est le suivant :

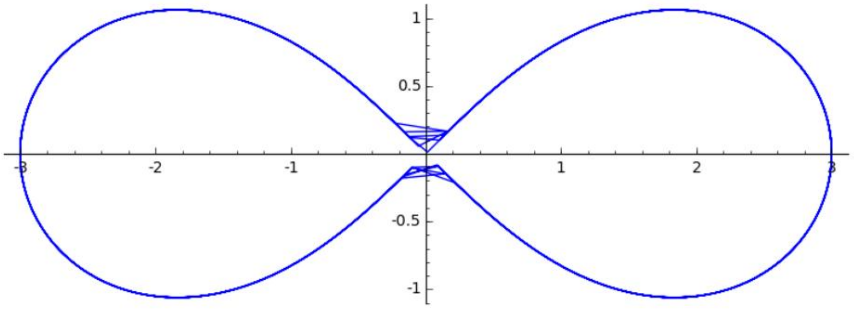
```
var("theta")
polar_plot( 3*sqrt(cos(2*theta)), (theta, -6*pi, 6*pi ), plot_points=10000)
```

La figure 3.3.2 montre les images produites avec 10 000 points (ci-dessus) et 200 points (ci-dessous). Le phénomène est en partie exagéré car nous utilisons $-6\pi < \theta < 6\pi$ comme domaine. Nous aurions pu choisir $-\pi < \theta < \pi$ à la place, et nous aurions alors obtenu plus de points par radian.

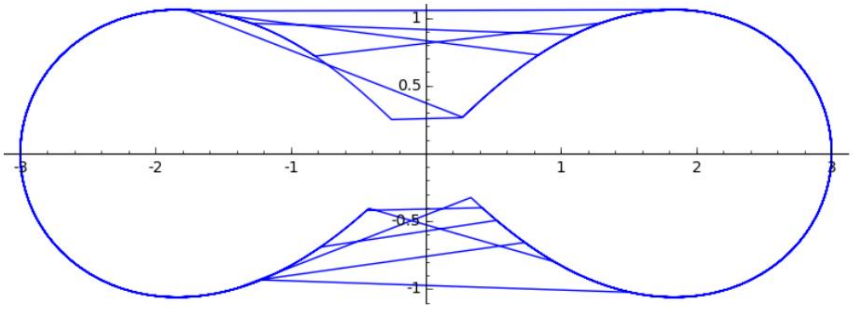
Vous vous demandez peut-être s'il existe une condition permettant de savoir à l'avance si ce problème se produira ou non. En effet, une telle condition existe. La dérivée $dr/d\theta$ n'existera en aucun point P lorsque la courbe est parallèle à la droite reliant P à l'origine.

Appelons ces points « points radialement tangents ». À ces points, l'ordinateur doit dessiner plusieurs pixels pour une valeur de θ, et cela ne se produit que pour ces points. C'est pourquoi l'algorithme ne fonctionne pas parfaitement dans ces conditions. L'analogie en coordonnées rectangulaires serait lorsque la tangente est verticale. À ces points, le rapport dy/dx n'existe pas, et le tracé est souvent erroné, même sur une calculatrice graphique bon marché.

Machine Translated by Google



Un Lemniscate tiré avec 10 000 points.



Une Lemniscate Tirée avec 200 Points.

ou dans un logiciel sophistiqué. Nous en avons vu des exemples à la page 14, où nous avons représenté graphiquement des fonctions rationnelles.

3.4. Représentation graphique d'une fonction implicite

Parfois, nous écrivons les équations des courbes sous la forme $(x - 1)^2 + (y - 2)^2 = 9$ plutôt que sous la forme $ay = f(x)$. La première est appelée « fonction implicite » et la seconde « fonction explicite ».

C'est un bon exemple, car nous aurions besoin d'écrire deux fonctions

$$f(x) = 2 + 9 - (x - 1)2 \quad g(x) = 2 - 9 - (x - 1)2$$

pour représenter cette courbe. Par conséquent, Sage vous permet de tracer des fonctions définies implicitement.

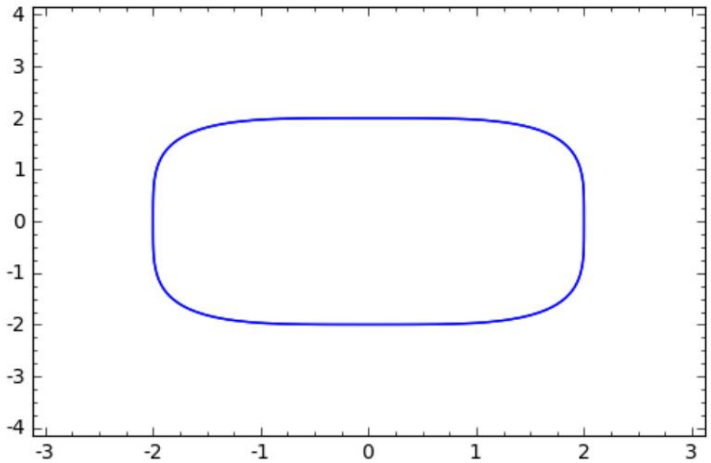
Machine Translated by Google

Par exemple,

g(x,y)=x^4+y^4-16

implicit_plot(g, (x,-3,3), (y,-4,4))

est une courbe quartique privilégiée par les concepteurs de meubles pour la conception de salles de conférence 4 tables. Comme vous pouvez le voir, sa formule mathématique est $x^4 + et^4 = 16$. Voici l'intrigue:

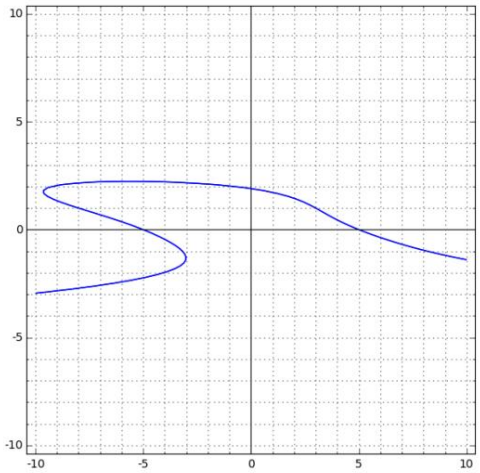


Pour d'autres courbes, telles que

$$x^2 + y^5 + 5xy = 25$$

la forme explicite serait encore plus fastidieuse, voire impossible. L'intrigue, ci-dessous, est donné par les commandes

```
varier")
implicit_plot( x^2 + y^5 + 5*x*y == 25, (x,-10,10), (y,-10,10),
                gridlines="mineur", axes=True)
```



Machine Translated by Google

Comme vous pouvez le voir, ce tracé montre un joli arrière-plan qui ressemble à du papier millimétré, en utilisant l'option de quadrillage que nous avons vue pour la première fois dans la section 1.4 à la page 9. La commande axes=True est requise pour afficher les axes dans un tracé implicite.

Notez que les fonctions implicites dans trois variables (x, y et z) sont abordées dans l'annexe en ligne uniquement électronique de ce livre « Traçage en couleur, en 3D et animations », disponible sur ma page Web www.gregorybard.com pour téléchargement.

Compatibilité ascendante : La syntaxe

suivante reproduit la syntaxe de la commande plot implicite : plot(x^2, (x, -2, 2)) alors que nous aurions normalement tapé plot(x^2,

-2, 2) comme dans la section de

traçage d'origine (section 1.4, page 9). Ceci est utile

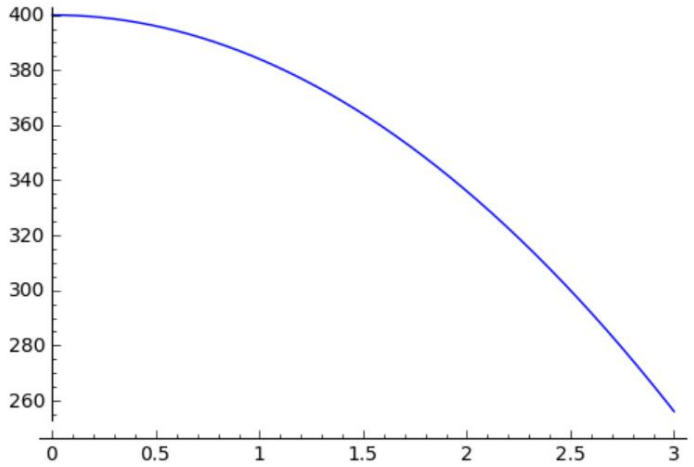
lorsque vous devez tracer des

éléments en fonction de t et non de x. Par exemple :

var('t')

plot(-16*t^2+400, (t, 0, 3)) représente la hauteur

d'une boule de bowling lâchée d'un immeuble de 120 mètres, t secondes après son lancement. Voici le graphique :



Cette même notation, consistant à utiliser (t, 0, 3) ou (x, -2, 2) pour signifier 0 < t < 3 et −2 < x < 2, sera utilisée non seulement pour le tracé implicite, mais également pour de nombreuses autres commandes de tracé que nous allons maintenant explorer.

3.5. Tracés de contour et ensembles de niveaux

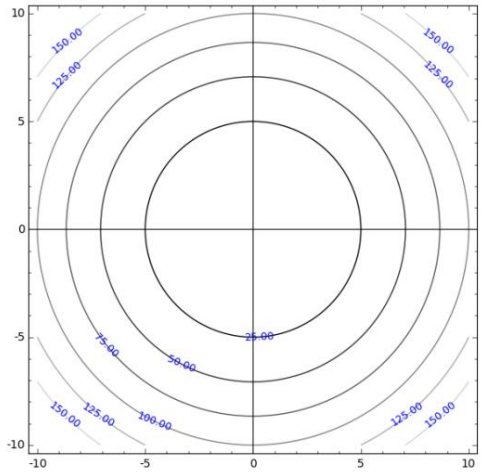
Une façon classique de visualiser une fonction de deux variables f(x, y) est d'utiliser un tracé de contour, qui affiche une collection d'« ensembles de niveaux » pour cette fonction.

Machine Translated by Google

Un ensemble de niveaux est l'ensemble des points (x, y) tels que f(x, y) = z pour un z fixe.
Par exemple, si

$$f(x, y) = x^2 + et^2$$

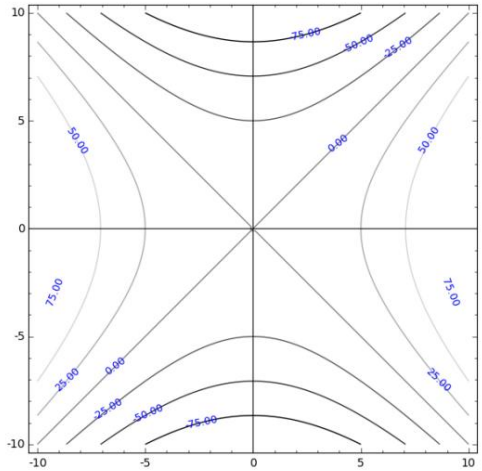
alors le graphique suivant montre les ensembles de niveaux pour z {25, 50, 75, 100, 125, 150}.



Le code pour créer cette image est

```
var("y")
contour_plot( x^2 + y^2, (x, -10, 10), (y, -10, 10),
                    fill=False, axes=True, labels=True )
```

Voici un autre tracé de contour, pour g(x, y) = x^2 entre x^2 « - » y^2. Pour générer ceci et y image, tout ce que j'ai fait a été de changer le « + » en un « - » dans le code précédent.

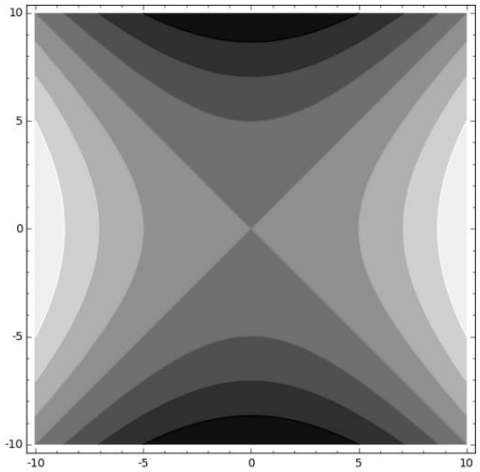


Il s'agit des tracés de contours traditionnels que l'on trouvait dans les manuels scolaires du XXe siècle. Les tracés de contours modernes utilisent des ombres, ce qui les rend généralement plus lisibles, mais au prix d'une consommation d'encre plus élevée.
Le code suivant var('y')

Machine Translated by Google

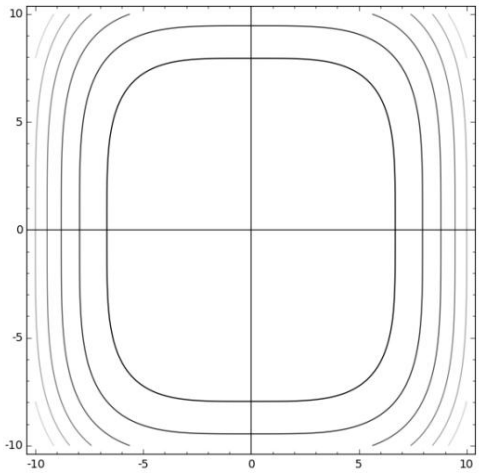

```
contour_plot( x^2-y^2, (x, -10, 10), (y, -10, 10), plot_points=200)
```

génère une image pour la même fonction que l'image précédente, g(x, y) = $2^2x - y$, mais dans le style ombré.



Dans ces tracés de contours ombrés, les zones sombres représentent les valeurs f plus petites, et les zones claires les valeurs f plus grandes. D'ailleurs, l'option de tracé de points permet à Sage de déterminer le niveau de précision souhaité pour le tracé de contour. J'utilise généralement 200 ou 300. La valeur par défaut est 100, ce qui me semble trop petit.

En toute honnêteté, je dois souligner que l'ombrage est parfois plus néfaste que bénéfique. Le schéma de contour suivant est parfaitement lisible sans ombrage.



Le code qui a généré le tracé de contour précédent était

```
var("y")
contour_plot( x^4 + (y)^4/2, (x, -10, 10), (y, -10, 10), fill=False, axes=True )
```

Machine Translated by Google

En revanche, si l'on effectue un retour arrière sur les commandes « fill=False » et « axes = True », le tracé de contour ombré obtenu présente une immense zone noire occupant la grande partie centrale du tracé, à l'intérieur de l'ovale le plus profond. Cela rend le graphique presque illisible ; essayez-le si vous le souhaitez.

Afin d'explorer les relations entre certains objets tridimensionnels et leurs diagrammes de contour, j'ai créé une page Web interactive (parfois appelée interaction ou applet) pour montrer certaines formes complexes à la fois dans des tracés 3D et dans des tracés de contour, simultanément.

3.5.1. Une application à la thermodynamique

L'exemple suivant est tiré de Elementary Differential Equations and Boundary Value Problems, de William Boyce et Richard DiPrima, 9e édition, publié par Wiley en 2009. Il s'agit de l'exemple 1 de la section 10.5, « Séparation des variables : conduction thermique dans une tige ». Cet exemple étendu nécessitera certaines techniques des sections 4.1 et 4.19. J'espère sincèrement qu'il est intelligible sans avoir étudié ces sections de trop près, mais si le lecteur trouve cet exemple confus, il pourrait être utile de lire ces sections. Avant de commencer, je dois m'excuser du fait que la composition ici ne m'a pas permis d'attacher des unités aux constantes dans tous les cas, comme c'est la pratique chez les physiciens.

La fonction u(x, t) ci-dessous indique la température u à l'instant t et à la position x à l'intérieur d'une tige. Cette tige, longue de 50 cm et isolée extérieurement, est initialement à une température uniforme de 20 °C (68 °F). Ses extrémités sont exposées brusquement à une température de 0 °C (32 °F) à t = 0 et maintenue à cette température pendant t > 0. Comme vous pouvez le constater, la fonction est définie par une série.

$$u(x, t) = \frac{80}{p} \sum_{n=1,3,5,\ldots}^{n=\infty} \frac{1}{n} e^{-n^2 \pi^2 \alpha^2 t/2500} \sin \frac{n\pi x}{50}$$

mais la variable série n s'étend sur tous les entiers positifs impairs.

Définissons plutôt n = 2j + 1. Ensuite, lorsque j passe par 0, 1, 2, 3, . . . , parcourra 1, 3, 5, 7, . . . , produisant tous les entiers positifs impairs. Cela nous permettra d'utiliser la commande sum de Sage. De plus, nous laisserons simplement j courir de 0 à 100. Après tout, lorsque j = 101, nous aurons n = 203, et ce terme $e^{-2032\pi^2 t/2500}$ contiendra e, ce qui le rendra complètement négligeable.

En réalité, on pourrait s'arrêter à j = 10 ou j = 20 et probablement produire le même graphique, mais les ordinateurs n'ont pas peur de faire beaucoup de travail fastidieux ; nous ferons donc un compromis à j = 100. Boyce et DiPrima utilisent α = 1, et nous ferons donc de même. Maintenant, nous avons

$$u(x, t) = \frac{80}{p} \sum_{j=1}^{n=\infty} \frac{1}{2j + 1} e^{-(2j+1)^2 \pi^2 t/2500} \sin \frac{(2j + 1)\pi x}{50}$$

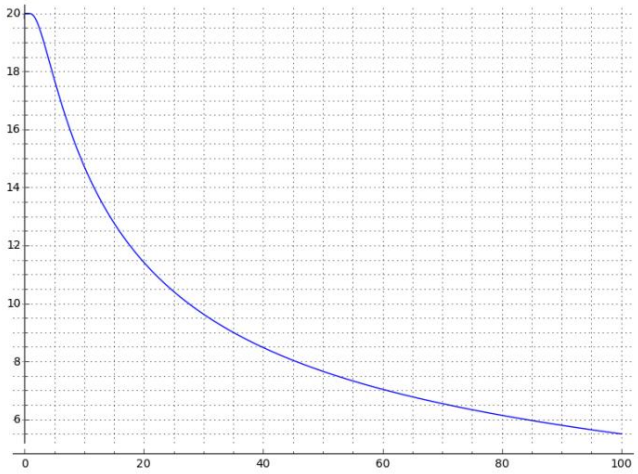
Puisqu'il s'agit d'une fonction à deux variables, une façon de la visualiser serait de la « demi-évaluation ». Nous pourrions tracer u(5, t) pour 0 < t < 100 et obtenir un

Machine Translated by Google

Une idée de l'évolution de la température au fil du temps, à 5 cm d'une extrémité de la tige. Pour cela, nous saisirions

```
var("j t")

u(x, t) = (80/pi)*somme( (1/(2*j+1))*exp(-(2*j+1)^2*pi^2*t/2500.0)
                * sin((2*j+1)*pi*x/50), j, 0, 100)

plot( u(5, t), t, 0, 100, gridlines="mineur" )
```

Cela produit le tracé suivant :



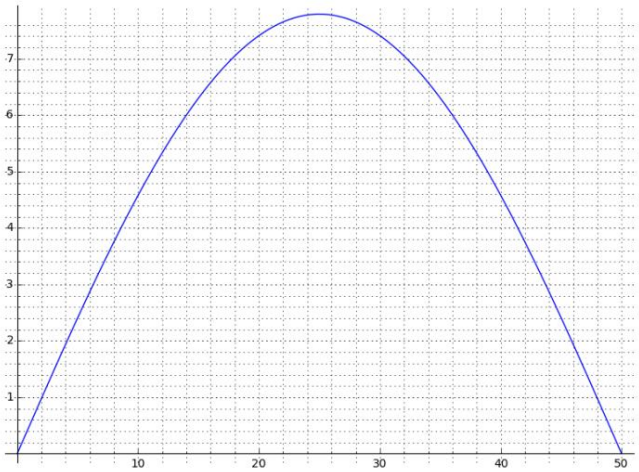
Quelques remarques rapides s'imposent. Nous avons utilisé 2500.0 au lieu de 2500 pour forcer Sage à évaluer l'expression numériquement, et non pas exactement ; sinon, l'expression algébrique exacte résultante est beaucoup trop complexe et l'ordinateur met un temps fou à effectuer le calcul. Deuxièmement, u(5, t) est un exemple de demi-évaluation (voir page 127). Troisièmement, la syntaxe pour additionner une série est donnée page 182.

Alternativement, nous pourrions tracer u(x, 300) et obtenir une idée de la température dans la tige, 300 secondes ou 5 minutes après le début du refroidissement. Nous remplacerions la dernière ligne du code ci-dessus par

```
plot( u(x, 300), x, 0, 50, gridlines="mineur" )
```

Cela produit l'image :

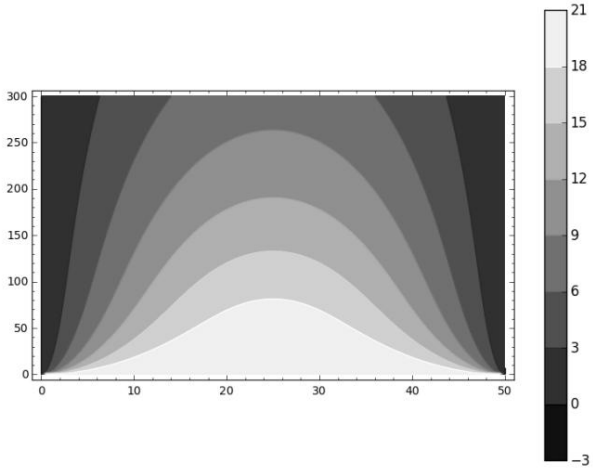
Machine Translated by Google



Cependant, un tracé de contour est bien plus performant que n'importe lequel de ces outils. Si vous tracez un tracé de contour et que vous observez uniquement une ligne verticale le long de cette ligne, vous découvrirez comment la température varie au fil du temps en un point précis. Si vous tracez une ligne horizontale le long de cette ligne, vous découvrirez comment la température varie sur la tige à un instant précis. Pour créer le tracé de contour, nous remplaçons la dernière ligne du code ci-dessus par

```
contour_plot( u(x,t), (x,0,50), (t,0,300), rapport hauteur/largeur=0,1,
                barre de couleurs = Vrai )
```

Ce code nous donne le tracé de contour suivant :



Comme vous pouvez le voir, une « barre de couleur » ou légende en niveaux de gris a été jointe pour nous indiquer ce qu'indiquent les différentes nuances de gris, en termes de degrés Celsius. C'est très utile ici en thermodynamique, mais cela n'aurait pas nécessairement été pertinent dans les exemples de mathématiques pures que nous avons vus plus tôt dans cette section.

Enfin et surtout, l'option de rapport hauteur/largeur doit être expliquée. Lors du traçage d'objets géométriques, tels que les fonctions à deux variables qui

Machine Translated by Google

Comme nous l'avons vu précédemment, il est important d'utiliser la même échelle sur l'axe des abscisses et l'axe des ordonnées. Sinon, les angles seront décalés et d'autres distorsions apparaîtront. Si nous supprimions l'option « rapport hauteur/largeur = 0,1 », Sage observerait que nous traçons 300 dans une variable et 50 dans une autre. Par conséquent, le graphique serait six fois plus étroit que haut, afin de conserver la même échelle. La meilleure façon de le constater est de revenir en arrière sur l'option « rapport hauteur/largeur = 0,1 » et d'appuyer sur « Évaluer ». Au lieu de cela, avec cette option, le 50 sur l'axe des abscisses est interprété comme 500, et nous obtenons un graphique 1,6 fois plus large que haut. Ce graphique est beaucoup plus lisible. Ce changement ne devrait pas nous gêner, car 50 cm et 300 s sont dans des unités totalement différentes, et la notion de « même échelle » n'a donc aucun sens.

3.5.2. Application à la microéconomie (équations de Cobb-Douglas)

Voici un exemple de fonction de production Cobb-Douglas

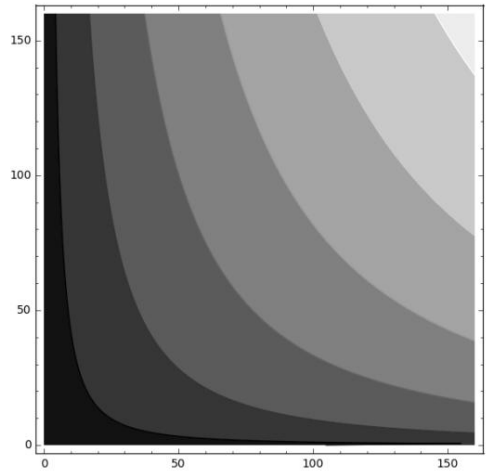
$$f(K, L) = 1400K0,51L\ 0,32$$

Ce modèle a été dérivé de données hypothétiques pour une petite entreprise produisant des tuyaux en béton. Ce modèle a été trouvé dans l'ouvrage « Programmation linéaire et non linéaire avec Maple : une approche interactive et basée sur les applications » de Paul Fishback, publié par CRC Press en 2010. La fonction apparaît à plusieurs reprises dans l'ouvrage, mais elle est dérivée de données et constitue l'exemple principal de la section 7.3.5, « Régression non linéaire ».

En utilisant le code :

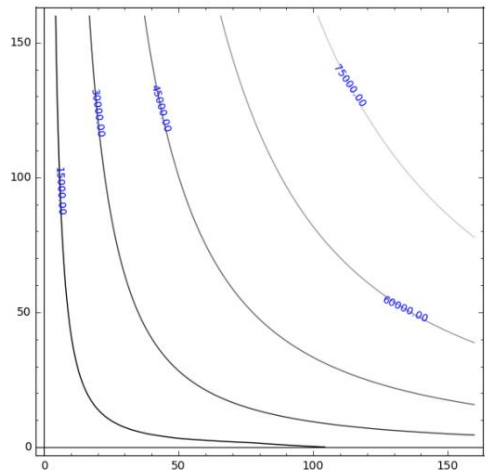
```
var("y")
contour_plot( 1400*x^0.51*y^0.32, (x, 0, 160), (y, 0, 160), plot_points=300)
```

nous obtenons le tracé de contour ombré suivant, qui est modérément informatif.



Cependant, l'intrigue sans ombres, beaucoup plus lisible

Machine Translated by Google



a été produit par le code suivant : var("y")

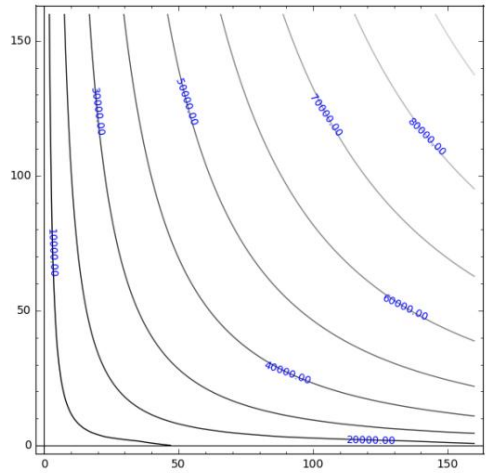
```
contour_plot( 1400*x^0.51*y^0.32, (x, 0, 160), (y, 0, 160),
                    fill=False, axes=True, labels=True )
```

Forcer des valeurs particulières : il

est parfois utile de forcer Sage à choisir des courbes de niveau particulières. Si certaines valeurs sont très importantes, ou requises par un exercice, il peut être utile d'imposer à Sage les valeurs z spécifiques à utiliser pour les ensembles de niveaux. Voici un exemple : var("y")

```
contour_plot( 1400*x^0.51*y^0.32,
(x, 0, 160),
(y, 0, 160), fill=False, axes=True, labels=True, contours=[10000, 20000, 30000, 40000,
                    50000, 60000, 70000, 80000, 90000, 100000] )
```

produit l'image suivante :



Machine Translated by Google

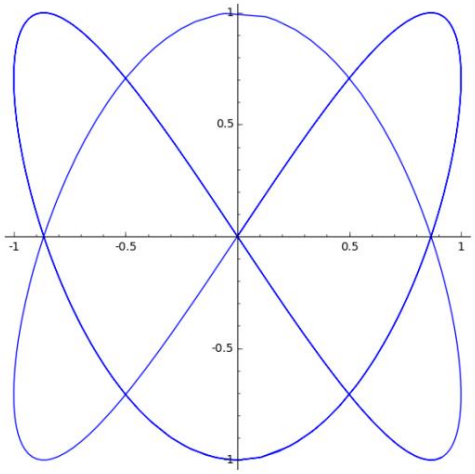
3.6. Tracé paramétrique 2D

Les mathématiciens utilisent le terme « fonction paramétrique » pour désigner les situations où l'abscisse d'un point est donnée par f(t), tandis que l'ordonnée d'un point est donnée par g(t). Par conséquent, pour tout instant t, on connaît l'abscisse et l'ordonnée. Ceci est très utile pour décrire des types de mouvement plus avancés en physique, notamment en mécanique céleste : le mouvement des planètes, des astéroïdes et des comètes.

Un exemple vraiment intéressant est la courbe de Lissajous, nommée d'après Jules Antoine Lissajous (–), mais découverte en réalité par Nathaniel Bowditch (–). Lissajous les ayant étudiées en détail, ces courbes ont été nommées en son honneur. Ces courbes sont de la forme x = sin(at + b) et y = sin(ct + d), avec a et c généralement limités aux entiers, bien que certains manuels autorisent une constante multiplicative devant chaque fonction sinus, ou des valeurs rationnelles pour a et c. Dans tous les cas, si a = c, il s'agit simplement d'une ellipse ou d'un cercle. Cependant, a = c donne des figures beaucoup plus intéressantes. Voici un code pour le cas a = 2 et c = 3.

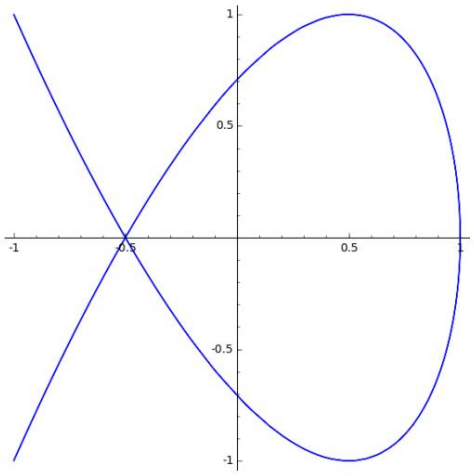
```
var("y t")

f(t) = sin(2*t) g(t) =
sin(3*t)

tracé_paramétrique( ( f(t), g(t) ), (t,0,2*pi) )
```

Ce code produit l'image suivante :

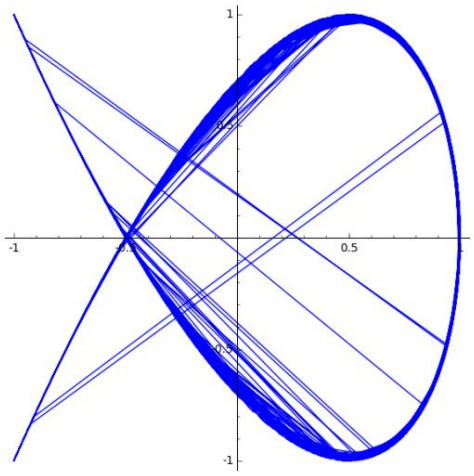


Les courbes de Lissajous sont très importantes pour le réglage des oscilloscopes et des systèmes audio. Si nous modifions la valeur de f(t) par cos(2t), nous obtenons une valeur très différente.

Machine Translated by Google



Le phénomène que nous avons vu avec les coordonnées polaires à la page 101, où nous avions trop peu de points par radian, peut également se produire dans les équations paramétriques. Imaginez modifier le domaine de mesure à −100 < t < 100 dans l'exemple précédent, en veillant à conserver y = cos(2t). Plus précisément, cela revient à remplacer (t, 0, 2*pi) par (t, -100, 100). Le résultat est extrêmement médiocre, car les points sont très éloignés. Comme le domaine de mesure a une longueur de 200 en unités de t et que le nombre de points tracés par défaut est de 200, chaque point utilisé pour construire le tracé est distant d'une unité de t. Avec un intervalle de longueur 2π, les points sont distants de π/100 ≈ 0,0314159 unités de t, ce qui représente une différence importante.



Convertir la forme paramétrique d'une fonction en fonction implicite, si possible, constitue un excellent exercice pour les étudiants. Cependant, ce n'est pas possible (avec les fonctions élémentaires) pour certaines courbes. De plus, le calcul des courbes paramétriques est simple, mais sensiblement différent de celui des fonctions explicites. La plupart des manuels de calcul traitent des fonctions paramétriques, mais certains enseignants l'ignorent, ce qui peut poser problème aux étudiants en ingénierie ou en physique confrontés à des fonctions ou des équations paramétriques au niveau supérieur.

Machine Translated by Google

Cours de niveau. Si le lecteur souhaite en savoir plus, je recommande les chapitres 9.2 et 9.3 de l'ouvrage de Soo Tan, Calculus with Early Transcendentals, 1re édition, publié par Cengage en 2011.

3.7. Diagrammes de champs vectoriels

Tout d'abord, il serait bon de rappeler à quoi ressemble un tracé de champ vectoriel. Voici un exemple : nous traçons la fonction vectorielle suivante :

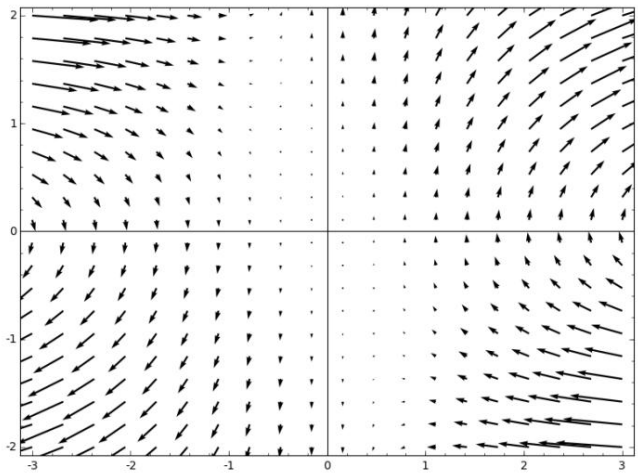
$$f(x, y) = \begin{matrix} \text{sans } x \\ \cos \text{ et} \end{matrix}$$

Pour réaliser le tracé du champ vectoriel, nous devons taper

```
var("y")
plot_vector_field( (sin(x), cos(y)), (x, -10, 10), (y, -10, 10),
                   plot_points=30 )
```

Notez qu'il s'agit d'un autre exemple illustrant comment Sage sait que x est une variable, mais nous devons lui rappeler que toutes les autres lettres sont des variables, en les déclarant avec la commande var. Nous supposons ici que la valeur x de notre fonction vectorielle est sin(x) et que sa valeur y est cos(y). La notation (x, -10, 10) signifie alors que l'étendue x souhaitée est $-10 < x < 10$, et naturellement, la notation (y, -10, 10) indique $-10 < y < 10$.

Le paramètre « Points de tracé » indique à Sage le nombre de vecteurs à dessiner dans chaque ligne et chaque colonne du tracé. Vous conviendrez sans doute que créer un tel tracé, avec 900 flèches distinctes, serait extrêmement fastidieux à la main. Nous avons maintenant le tracé suivant.



Tout comme f(x, y) renvoie un vecteur pour chaque point (x, y), nous avons (en plusieurs points du plan de coordonnées) une flèche représentant un vecteur. La direction est reproduite très fidèlement, et la longueur de la flèche est une représentation approximative de la norme de ce vecteur.

Machine Translated by Google

Comment interpréter ces tracés ? Une interprétation possible est la suivante : si un petit objet était placé dans le plan de coordonnées, il se déplacerait à chaque instant dans la direction indiquée par la flèche située sous l'objet. Les seules hypothèses sont que la masse et le rayon de l'objet soient minimes. Une autre interprétation implique les gradients, que nous allons maintenant explorer.

3.7.1. Gradients et tracés de champs vectoriels

L'une des raisons pour lesquelles il faut réaliser un tracé de champ vectoriel est d'étudier le gradient d'une fonction. Nous pouvons voir que si

$$g(x, y) = (\sin y) - (\cos x) \text{ alors}$$

les dérivées partielles deviendraient

$$\frac{\partial g}{\partial x} = \text{péché } x \qquad \text{et} \qquad \frac{\partial g}{} = \cos y \; \partial y$$

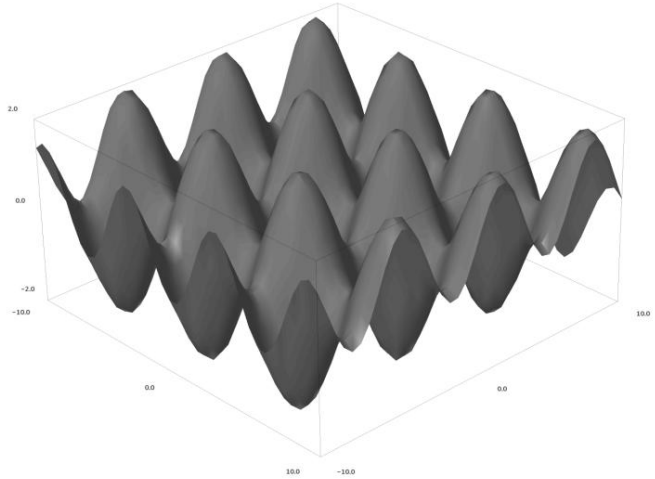
ce qui rendrait le dégradé

$$g = \begin{matrix} \frac{\partial g}{\partial x} \\ \frac{\partial g}{\partial y} \end{matrix} = \begin{matrix} \text{sans } x \\ \cos \text{ et} \end{matrix} = f(x, y)$$

Comme vous pouvez le constater, il s'agit de la même fonction vectorielle que celle que nous venons de tracer. Si nous voulions visualiser g(x, y) comme la coordonnée z de (x, y), nous pourrions créer un graphique 3D

avec g(x, y) = sin(y) - cos(x).

plot3d(g, (x,-10,10), (y,-10,10))



Comme vous pouvez le voir, nous obtenons un objet à motifs qui n'est pas sans rappeler une boîte à œufs. Regardons maintenant à nouveau le tracé du champ vectoriel. Il y a des points où plusieurs flèches pointent toutes vers le même point. Si on place une bille à proximité de ce point, elle sera attirée vers ce point et y restera. Ce sont des points où g est le vecteur nul et correspond à

Machine Translated by Google

à un minimum local de la boîte à œufs. Il existe également des points où une série de flèches pointent toutes dans la direction opposée à un point particulier. Si vous placiez une bille au voisinage de ce point, elle serait repoussée et ne reviendrait pas. Ce sont des points où g est également le vecteur nul, mais ils correspondent à un maximum local de la boîte à œufs. Un mathématicien appliqué ou un physicien dirait qu'il s'agit de points d'équilibre attractif et répulsif, respectivement pour les cas de minimum et de maximum locaux.

Comment distinguer ces minima et maxima locaux ? Vous pourriez demander à Sage de générer un champ vectoriel et de procéder à l'identification visuelle. Vous pourriez aussi considérer la matrice hessienne (ici une matrice 2 × 2 de dérivées partielles secondes), puis calculer le déterminant de cette matrice. De toute évidence, une tâche est considérablement plus difficile que l'autre.

3.7.2. Une application de la physique. Imaginez

trois charges électriques dans le plan de coordonnées. La charge 1 est liée à l'origine et la charge 2 au point (2, 0). La troisième charge est mobile, au point (x, y). Nous souhaiterions connaître la force totale de la charge électrique sur la troisième charge mobile. Pour rendre le problème plus intéressant, imaginons que les charges soient +2, +1 et -1 pour la charge 1, la charge 2 et la charge mobile. L'inégalité des charges rendra le problème plus intéressant. Un excellent moyen de visualiser cela est d'utiliser un champ vectoriel.

Avant de continuer, rappelons brièvement la loi de Coulomb.
La force F (en Newtons) d'une charge de q0 Coulombs sur une charge de q1 Coulombs est donnée par

$$F = -\frac{1}{4\pi 0}\frac{q1q0}{||r||3}r$$

« permettant » – où r est le vecteur de q1 à q0, où 0 est une constante (appelée π a sa signification usuelle. Certains manuels définissent r comme le vecteur de q0 à q1 ; dans ce cas, on supprime le signe moins devant le côté droit. (À noter que la notation ||r|| désigne la norme, ou longueur, du vecteur r.)

Certains lecteurs pourraient être surpris de voir le cube au dénominateur.
C'est parce que nous travaillons avec des vecteurs. Si nous voulions travailler uniquement avec des scalaires, c'est-à-dire le nombre réel exprimant l'intensité de la force, nous aurions plutôt

$$||F|| = -\frac{1}{4\pi 0}\frac{|q1||q0|}{||r||3} \quad ||r|| = \frac{1}{4\pi 0}\frac{|q1||q0|}{||r||2}$$

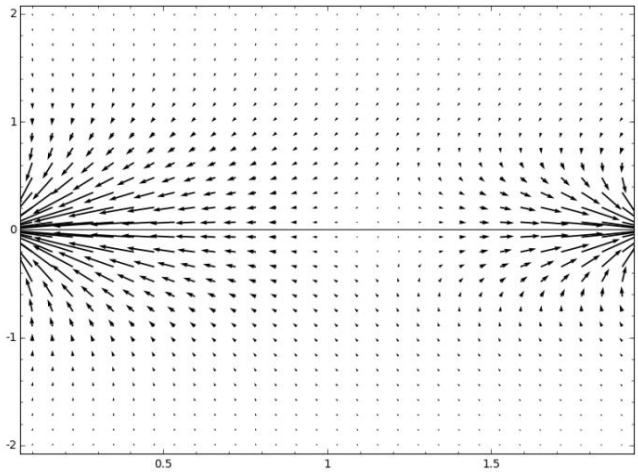
nous donnant la loi familière du carré inverse.

Nous sommes maintenant prêts à analyser le code (trouvé dans la figure 1) qui a été utilisé pour générer le tracé suivant.

Machine Translated by Google


```
plot_vector_field(net_f, (x, -1, 4), (y, -2, 2), plot_points=30 ) var("y")


x1= 0
y1= 0
x2= 2
y2= 0

r_vec_1 = vecteur( [ x1-x, y1-y ] r_vec_2 =
vecteur( [ x2-x, y2-y ]                    ) )

q0= -1
q1= 2
q2= 1

f1 = - q0*q1*r_vec_1 / r_vec_1.norm()^3 f2 = - q0*q2*r_vec_2 /
r_vec_2.norm()^3

net_f = f1 + f2

plot_vector_field(net_f, (x, 0,1, 1,9), (y, -2, 2), plot_points=30)
```

Figure 1. Le code pour l'application physique des tracés de champs vectoriels



Premièrement, nous définissons x1 et y1 comme les coordonnées de la charge 1, à l'origine. Deuxièmement, nous définissons x2 et y2 comme les coordonnées de la charge 2, situées au point (2, 0). Il est utile d'isoler ces valeurs en début de programme (plutôt que de les substituer au milieu des formules, plus tard), car cela nous permettra d'expérimenter en ajustant légèrement les valeurs.

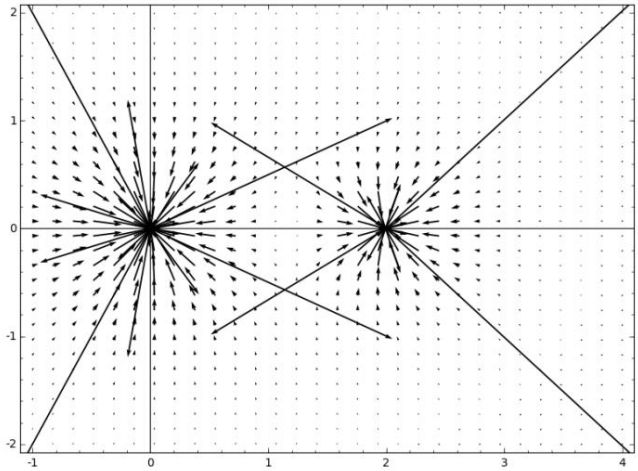
Machine Translated by Google

Troisièmement, nous définissons deux vecteurs. Le vecteur r vec 1 va de (x, y) à (x1, y1). De même, le vecteur r vec 2 va de (x, y) à (x2, y2). Chaque fois que vous devez définir un vecteur, que ce soit dans Sage ou dans toute autre situation mathématique, il est utile de se rappeler l'astuce « tête moins queue ». Si vous visualisez r vec 1 comme une flèche, alors la pointe de la flèche serait en (x1, y1) et les plumes de la queue en (x, y).

Quatrièmement, nous définissons les charges de la charge mobile, Charge 1 et Charge 2, par q0, q1 et q2. Cinquièmement, nous appliquons la loi de Coulomb. Pour la force entre la charge 1 et la charge mobile, nous calculons f1 et de même f2. Comme vous pouvez le constater, j'ai supprimé toutes les constantes, car elles n'affectent pas les directions des vecteurs. Sixièmement, f nette est la somme de ces deux forces, ce qui donne la force totale sur la charge mobile. Notez que nous n'avons pas eu besoin de calculer la force de la charge 1 sur la charge 2, ni la force égale et opposée de la charge 2 sur la charge 1, car elles ne sont pas pertinentes pour la charge mobile. Enfin, nous utilisons la commande plot vector field pour tracer le champ vectoriel pour $0{,}1 \le x \le 1{,}9$ et $-2 \le y \le 2$.

Comme Icare, ne vous approchez pas trop !

Bien que les charges soient situées aux points (0, 0) et (2, 0), vous remarquerez que nous avons délibérément défini les valeurs de x pour exclure x = 0, x = 2 et leur voisinage immédiat. Si vous remplacez (x, 0,1, 1,9) par (x, -1, 4) dans la dernière ligne du code de la figure 1, vous constaterez la présence de vecteurs énormes et insensés qui semblent indiquer des conclusions absurdes. En particulier, il semble que certains de ces grands vecteurs pointent dans la mauvaise direction.



En réalité, lorsque (x, y) se rapproche trop de l'une des charges, la distance diminue considérablement. Cela signifie que le dénominateur, qui est au cube, devient extrêmement petit. La force devient alors absolument énorme. Le vecteur devient alors si long qu'il finit par occuper

[1] C'est aussi pour cette raison que nous ne nous sommes jamais souciés des unités dans ce problème : les unités n'affecteront pas les directions des vecteurs.

Machine Translated by Google

L'espace disponible dans le graphique est bien plus important qu'il ne le devrait, ce qui crée un résultat visuellement trompeur. Dans un champ vectoriel Sage, la queue de la flèche représente l'emplacement qu'elle décrit. Or, l'œil semble imaginer que la pointe de la flèche représente cet emplacement. Ce problème est généralement rare.

Mathématiquement parlant, (0, 0) et (2, 0) sont les pôles de la fonction de force totale, car la fonction y est divisée par zéro. En général, le tracé du champ vectoriel sera bien meilleur si l'on exclut les pôles de la région représentée.

Un défi : vous

êtes maintenant prêt à tenter le mini-projet sur les tracés vectoriels du champ électrique, qui est présenté à la page 90.

3.7.3. Gradients versus tracé de contour

L'exemple suivant illustre parfaitement la relation entre les tracés de contour et le tracé du champ vectoriel du gradient. Cet exemple a été créé par Augustine O'Keefe. Nous considérons la fonction

$$f(x, y) = \cos(x) - 2\sin(y)$$

et son gradient, qui est

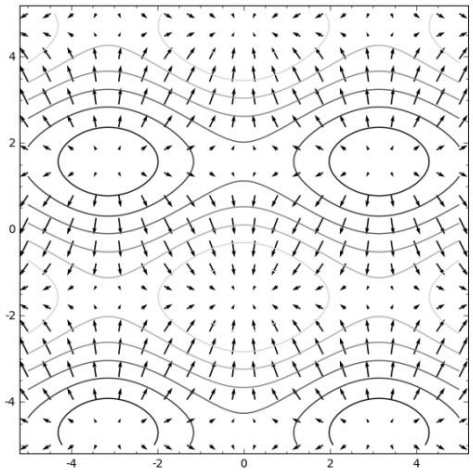
$$f(x, y) =< >=< \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} > = < -\sin(x), -2\cos(y) >$$

Nous allons tracer le tracé du contour de f(x, y), ainsi que le tracé du champ vectoriel du gradient, f(x, y), superposés sur la même image. Le code que nous utilisons est le suivant :

```
f(x,y) = cos(x)-2*sin(y) gradient =
dérivée( f )

P1 = plot_vector_field( gradient, (x,-5,5), (y,-5,5) )

P2 = contour_plot(f, (x,-5,5), (y,-5,5), fill=False)

afficher( P1 + P2 )
```

Ce code produit cette image :

Machine Translated by Google



On peut y voir une sorte de succession de collines et de vallées (le tracé des courbes de niveau), presque à la manière des cartes géologiques de courbes de niveau utilisées par les géologues et les scouts. Les flèches, représentant le gradient, illustrent la direction du déplacement d'une goutte de pluie, passant d'une altitude élevée à une altitude plus basse. On voit clairement où se situerait l'inondation en cas de fortes pluies. Enfin, on constate clairement que le vecteur gradient est toujours perpendiculaire aux courbes de niveau (les courbes qui composent le tracé des courbes).

3.8. Graphiques log-log

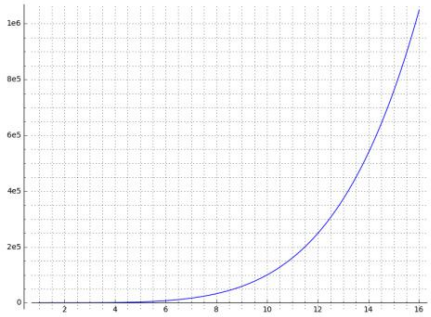
Au milieu du XXe siècle, il n'était pas rare de trouver du « papier log-log » en vente dans les librairies universitaires. Il s'agit d'un papier millimétré dont les lignes sont espacées de manière particulière : chaque ligne horizontale et verticale représente une échelle logarithmique. Par exemple, au lieu d'un pouce particulier représentant x = 1 à x = 2, divisé en incréments de 1/10 (par exemple, 1, 1,1, 1,2, 1,3, etc., 2), un pouce particulier pouvait représenter x = 10 à x = 100.
Les subdivisions représenteraient 101 102. Le pouce suivant représenterait x = 100 à x = 1000 et le pouce précédent représenterait x = 1 à x = 10.

La première question serait : « Pourquoi a-t-on procédé ainsi ? » L'objectif d'un graphique log-log est de permettre le calcul de lois de puissance par régression, ou peut-être simplement d'afficher une loi de puissance. Par exemple, le volume d'une sphère est proportionnel au cube de son rayon ; la distance d'arrêt d'une voiture est proportionnelle au carré de sa vitesse ; l'énergie du retour d'un signal radar est proportionnelle à l'inverse de la puissance quatre de la distance à la cible ; de nombreux autres exemples existent. Bien que c et n puissent varier, toutes ces relations sont de la forme y = cxn .

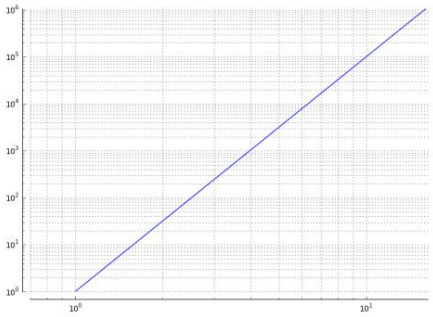
À l'ère des calculatrices graphiques et des logiciels de calcul formel, le papier log-log est extrêmement difficile à trouver. Cependant, Sage permet de tracer des tracés dans un

Machine Translated by Google

Mode log-log. Illustrons d'abord l'effet en considérant que y = x est clairement une loi [5] lequel de puissance avec c = 1 et n = 5.



Coordonnées ordinaires



Coordonnées log-log

Le tracé de gauche a été donné par

plot(x^5, (x, 1, 16), gridlines="minor") et celui de droite a été

donné par plot_loglog(x^5, (x, 1, 16),

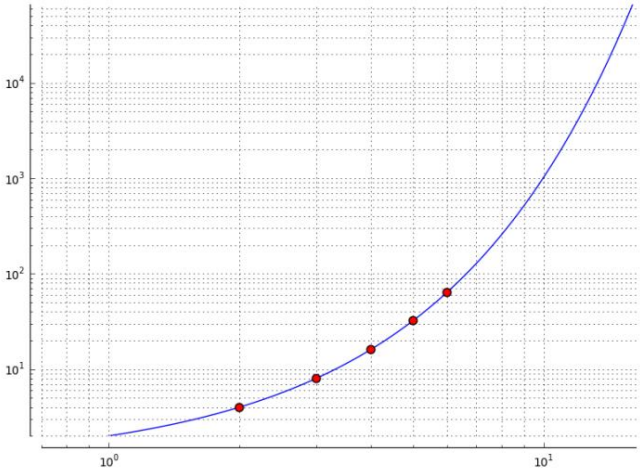
gridlines="minor")

Le code suivant trace une fonction y = 2x avec cinq points de données, à savoir {(2, 4), (3, 8), (4, 16), (5, 32), (6, 64)}. plot_loglog(2^x, (x, 1, 16),

gridlines="minor") +

nuage de points([(2,4), (3,8), (4,16), (5,32), (6, 64)], facecolor='red')

Vous trouverez ci-dessous le tracé généré par le code ci-dessus.



Pour plus d'informations sur le nuage de points, voir la section 4.9 à la page 157.)

Étant donné que cette transformation de l'espace bidimensionnel ordinaire (à savoir R dans le plan log-log) a été si utile, vous pourriez être curieux de savoir s'il existe une analogie pour l'espace tridimensionnel (à savoir l'espace R log-log-log). [2] [3]). Il y en a, et ça s'appelle Cela apparaît dans la dérivation des formules de style Cobb-Douglas en macroéconomie. En général, ces équations sont de la forme

$$P = cKaL^{b}$$

Machine Translated by Google

où P représente la production, K le capital investi et L le travail investi. Les coefficients a, b et c sont calculés en ajustant un plan d'ajustement optimal (l'analogue tridimensionnel d'une droite d'ajustement optimal), mais dans un espace log-log-log plutôt que dans un espace tridimensionnel ordinaire. Voir la section 3.5.2, page 110, pour un exemple de fonction Cobb-Douglas dérivée de données.

3.9. Situations rares

Nous avons ici deux situations rares qui peuvent se produire, ou non. L'une d'elles est une faille inhérente à la façon dont Sage gère les racines impaires des nombres négatifs. L'autre concerne les fonctions dont les valeurs x sont absentes de leurs domaines.

Contexte sur les racines cubiques

Lorsque nous disons $\sqrt[3]{64} = 4$, que voulons-nous vraiment dire ? Bien sûr, ce n'est qu'une façon d'écrire le fait que $4^3 = 64$. Cependant, lorsque les nombres complexes entrent en jeu, les choses se compliquent un peu.

Si je demande la racine cubique de 8, il y a trois nombres z dans le plan complexe tels que $z^3 = 8$. Plus précisément, ils sont

$$z = 2\cos(0^\circ) + i2\sin(0^\circ) = 2 + 0i = 2$$

$$z = 2\cos(120^\circ) + i2\sin(120^\circ) = -1 + i\sqrt{3}$$

$$z = 2\cos(240^\circ) + i2\sin(240^\circ) = -1 - i\sqrt{3}$$

Prenons un bref instant pour vérifier ce dernier, en le cubant soigneusement.

$$\begin{aligned}
\left(-1 - i\sqrt{3}\right)^3 &= \left(-1 - i\sqrt{3}\right)\left(-1 - i\sqrt{3}\right)^2 \\
&= \left(-1 - je\sqrt{3}\right)\left(1 + 2i\sqrt{3} + je^{2}3\right) \\
&= \left(-1 - je\sqrt{3}\right)\left(1 + 2i\sqrt{3} - 3\right) \\
&= \left(-1 - je\sqrt{3}\right)\left(-2 + 2i\sqrt{3}\right) \\
&= (-1)(-2) + (-1)(2i\sqrt{3}) + (-i\sqrt{3})(-2) + (-i\sqrt{3})(2i\sqrt{3}) = 2 - 2i\sqrt{3} + 2i\sqrt{3} - 2i^2(3) = 2 - (2)(-1)(3) = 2 + 6 = 8
\end{aligned}$$

Par conséquent, l'ensemble des nombres complexes qui sont au cube jusqu'à 8 serait

$$\left\{2, -1 + i\sqrt{3}, -1 - i\sqrt{3}\right\}$$

Machine Translated by Google

Cependant, de ces trois, le premier est réel — normalement c'est « celui que nous voulons » (du moins, habituellement). De même, si je demande la racine cubique de -8, il y a trois nombres z dans le plan complexe tels que $z^3 = -8$. Plus précisément, ils sont

$$z = -2\cos(0^\circ) - i2\sin(0^\circ) = -2 - 0i = -2 \quad z = -2\cos(120^\circ) - i2\sin(120^\circ) = 1 - i\sqrt{3} \quad z = -2\cos(240^\circ) - i2\sin(240^\circ) = 1 + i\sqrt{3}$$

Par conséquent, l'ensemble des nombres complexes qui s'élèvent au cube à -8 serait

$$-2,\ 1 - i\sqrt{3},\ 1 + i\sqrt{3}$$

Et comme vous pouvez le voir, il s'agit simplement de l'ensemble précédent multiplié par −1. De nouveau, de ces trois solutions, la première est réelle, et c'est généralement celle que nous recherchons (du moins, généralement). Il est intéressant de noter que je peux obtenir les trois solutions avec la commande suivante.

```
résoudre( x^3 == 8, x )
```

et bien sûr, cela fonctionne très bien avec == -8 également.

Le problème est que pour certains sujets extrêmement importants, y compris les corps de nombres algébriques et les corps finis (branches de l'algèbre abstraite), il est important que Sage renvoie l'une des deux racines complexes lorsque vous tapez

```
imprimer -8^(1/3)
```

même si nous désirons généralement la vraie racine.

Cela signifie que si vous tapez

```
plot( x^(1/3), -5, 5 )
```

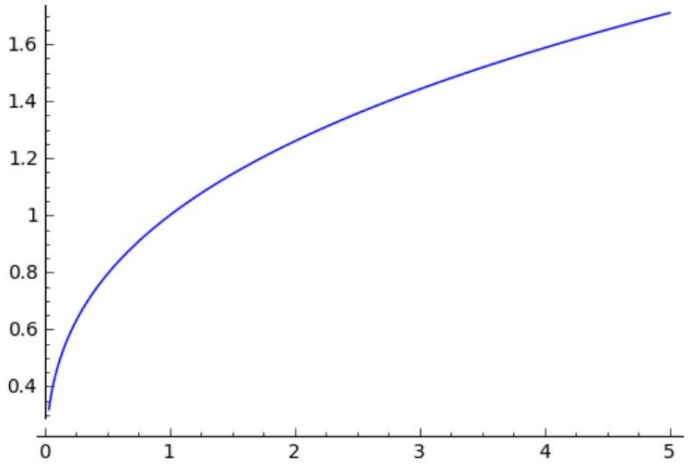
alors vous n'obtiendrez qu'un graphique montrant 0 < x < 2, accompagné du message d'erreur

```
verbeux 0 (2414 : plot.py, generate_plot_points)
AVERTISSEMENT : lors du traçage, l'évaluation de la fonction à 100 points a échoué.
verbose 0 (2414 : plot.py, generate_plot_points)
Dernier message d'erreur : « Un nombre négatif ne peut pas être élevé à une puissance fractionnaire »
```

car Sage renvoie les racines complexes (non visibles sur le plan réel) pour les valeurs x < 0. Comme vous pouvez le constater, Sage vous réprimande explicitement pour avoir élevé un nombre négatif à une puissance fractionnaire.

Le graphique est donné ci-dessous :

Machine Translated by Google



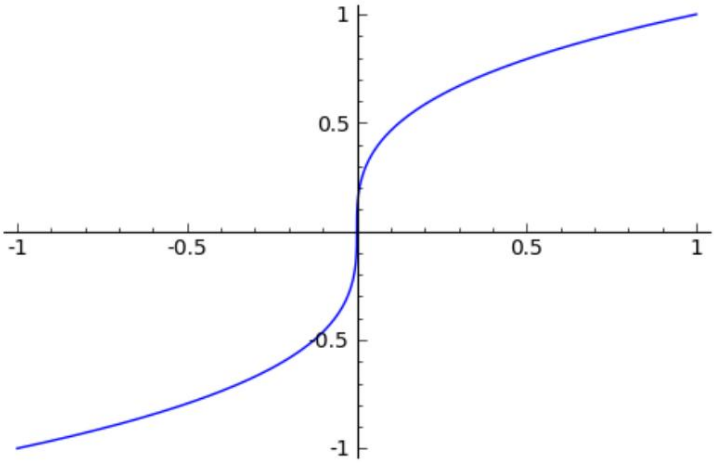
Je vais maintenant vous expliquer comment contourner ce problème.

Tracer les racines impaires des nombres négatifs Pour tracer la racine cubique réelle négative, vous devez utiliser ce qui suit : plot(lambda x : RR(x).nth_root(3), (x,-1, 1)) et non

plot(x^(1/3),-1,1) qui produit le message d'erreur dont nous venons de parler.

Le code avec le lambda produit la belle racine cubique lisse graphique que nous nous attendrions à voir.

Voici l'intrigue :



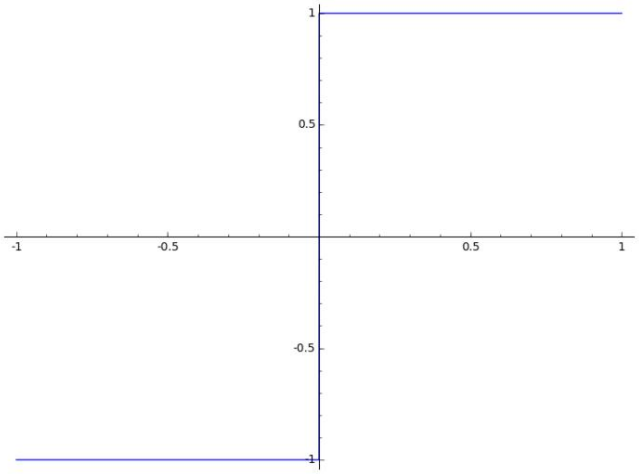
Une alternative est d'utiliser

plot(signe(x) * (abs(x)^(1/3)), -1, 1)

Ce qui est beaucoup plus simple à expliquer. La valeur absolue, calculée par abs, garantit que l'entrée à la puissance un tiers est positive. Cependant,

Machine Translated by Google

Les valeurs négatives de x doivent avoir des racines cubiques négatives, et les valeurs positives de x doivent avoir des racines cubiques positives. Par conséquent, nous multiplions par sign(x). La fonction sign(x) renvoie −1 pour tout x négatif et +1 pour tout x positif, mais renvoie 0 pour x = 0. En multipliant par sign(x), nous garantissons que la racine cubique calculée aura le bon signe. Vous pouvez voir un graphique de la fonction sign(x) ci-dessous :



Une nouvelle fonction Sage arrive !

Il existe de nombreuses raisons pour lesquelles Sage devrait disposer d'une commande permettant de contourner ce problème avec la puissance un tiers des nombres négatifs. Par exemple, on pourrait vouloir calculer les dérivées et les intégrales de la fonction racine cubique.

C'est dans cet esprit que, juste avant la publication de ce livre, j'ai lancé une discussion sur les groupes Google Sage-EDU et Sage-devel, consacrés respectivement à l'enseignement et au développement de Sage. C'est là tout l'intérêt d'un logiciel open source développé par une communauté.

On peut discuter avec les développeurs, faire des suggestions, demander des modifications, débattre des fonctionnalités et, si un consensus se forme, les changements interviennent. Il est beaucoup plus difficile de convaincre une entreprise d'apporter des modifications à son produit commercial, surtout lorsqu'il n'existe aucun moyen de communication avec les développeurs.

La nouvelle fonction sera probablement appelée racine_n-ième_réelle, ou quelque chose de très similaire, et fournira toujours la racine cubique réelle, la racine cinquième, la racine septième ou tout ce que vous pourriez vouloir.

Pour restreindre les valeurs x d'un graphique Cette

brève discussion n'est pas liée aux racines cubiques, mais plutôt aux racines carrées ou à toute autre fonction qui a un domaine autre que la ligne réelle entière.

Lorsque vous travaillez avec des fonctions qui ont des domaines limités, vous devez parfois restreindre les valeurs x d'un graphique.

Machine Translated by Google

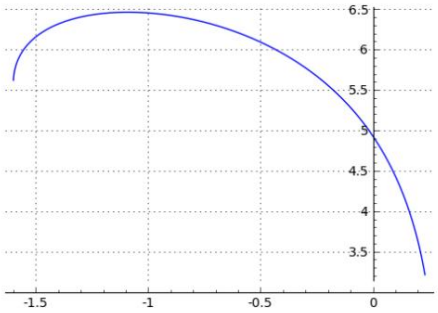
Considérez la fonction

$$f(x) = \sqrt{5x + 8} + \log(3 - 9x) + \sqrt{1 - 4x}, \text{ dont le domaine}$$

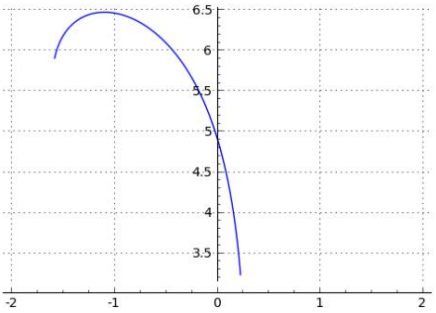
de définition est clairement $-8/5 \le x \le 1/4$. Si vous tapez plot(sqrt(5*x+8) + log(3-9*x) + sqrt(1-4*x), -2, 2, gridlines=True) alors que vous avez demandé $-2 \le x \le 2$, vous n'obtiendrez que $-8/5 \le x \le 1/4$. Un message d'erreur s'affiche également. À la place, tapez plot(sqrt(5*x+8) + log(3-9*x) + sqrt(1-4*x), -2, 2,

quadrillage=True).show(xmin=-2,xmax=2)

où xmin et xmax forcent explicitement les coordonnées x à être ce que vous voulez.



Tracer en utilisant Afficher



Intrigue sans spectacle

Machine Translated by Google

Chapitre 4

Fonctionnalités avancées de Sage

Dans ce chapitre, vous trouverez des commandes et des exemples pour des sujets mathématiques sélectionnés qui n'étaient pas suffisamment courants pour être donnés dans le chapitre un. Sauf quelques passages (comme indiqué), les sections sont totalement indépendantes les unes des autres et ne doivent certainement pas être lues dans l'ordre. Plongez-vous simplement dans les sections de ce chapitre, car vous pourriez avoir besoin de connaissances mathématiques avancées.

4.1. Utilisation de Sage avec des fonctions et équations multivariables

Considérons la fonction suivante

$$f(x, y) = (x^4 - 5x^2 + 4 + y^2)^2$$

et voyez ce que Sage peut en faire.

Comme nous l'avons vu à la section 1.8.1, page 38, lorsque nous utilisons des variables autres que x, sauf s'il s'agit de constantes comme k = 355/113, nous devons les déclarer comme variables à l'aide de la commande var. Par exemple

```
var('y') f(x,y)
= (x^4 - 5*x^2 + 4 + y^2)^2 f(1, 2)
```

Le code ci-dessus définit la fonction f(x, y) comme souhaité et renvoie correctement la valeur f(1, 2) = 16. On constate ainsi que les fonctions multivariées fonctionnent de la même manière que les fonctions univariées pour la définition et l'évaluation. Cependant, on peut également procéder à une demi-évaluation et obtenir une fonction comportant moins de variables.

Tout comme les fonctions à une variable sont représentées graphiquement sur des feuilles de papier bidimensionnelles (ou des écrans d'ordinateur), il est possible de représenter graphiquement des fonctions à deux variables dans un espace tridimensionnel. Ce sujet est abordé dans l'annexe en ligne de ce livre, exclusivement électronique, « Traçage en couleur, en 3D et animations », téléchargeable sur ma page web www.gregorybard.com.

Machine Translated by Google

Nous avons étudié les fonctions paramétriques à la section 3.6, page 112. Vous pourriez par exemple évaluer f(x, y), comme précédemment, mais avec x = 2t + 5 et y = 3t − 1, ou une formule similaire. Nous taperions var('y t') f(x,y) = (x^4 - 5*x^2 + 4 + y^2)^2 f(x=2*t+5,

y=3*t-1) et
nous obtiendrions la sortie.

```
((2*t + 5)^4 + (3*t - 1)^2 - 5*(2*t + 5)^2 + 4)^2
```

ce qui est clairement vrai mais pas très lisible.

Alternativement, si nous remplaçons la dernière ligne par

```
f(x=2*t+5, y=3*t-1).expand()
```

alors nous obtiendrons une sortie plus lisible par l'homme

```
256*t^8 + 5120*t^7 + 44448*t^6 + 217088*t^5 + 649161*t^4
        + 1214732*t^3 + 1394126*t^2 + 902940*t + 255025
```

Demi-évaluation d'une fonction :

Considérez le code suivant :

```
var('y') f(x,y)
= (x^4 - 5*x^2 + 4 + y^2)^2 imprimer f(x,y) imprimer
f(x=1) imprimer
f(y=2)
```

Ce code produit la sortie suivante :

```
(x^4 - 5*x^2 + y^2 + 4)^2 y^4 (x^4 -

5*x^2 + 8)^2
```

La première ligne est la fonction réelle f(x, y) telle que nous l'avions définie précédemment. La deuxième ligne indique ce qui se passe si x est verrouillé à x = 1. Dans ce cas, on obtient une fonction de y seul, une fonction assez simple. La troisième ligne indique ce qui se passe si y est verrouillé à y = 2. Comme vous pouvez le deviner, on obtient une fonction de x seul. Ce processus consistant à verrouiller une ou plusieurs variables d'une fonction tout en laissant une ou plusieurs variables libres est appelé « demi-évaluation ».

Travailler avec des paramètres Un

exemple plus étendu de « demi-évaluation » consiste à prendre une fonction avec quatre, cinq ou six variables et à y insérer des valeurs pour plusieurs d'entre elles (telles que la masse, la poussée, g = 9,82 ou d'autres coefficients) afin d'obtenir une équation avec moins de variables.

Nous en verrons un exemple dans la section 4.22.4 à la page 197.

Machine Translated by Google

4.2. Travailler avec de grandes formules dans Sage

Nous allons ici explorer l'utilisation de deux grandes formules dans Sage : l'une issue de la finance, l'autre de la physique. L'exemple financier est un prêt immobilier personnel. L'exemple physique est la loi de la gravitation de Newton. Dans les deux cas, nous allons travailler avec des formules et des modèles complexes, en écrivant un bloc de code Sage qui finira par atteindre 4 à 6 lignes.
Cette section suppose que vous avez lu la section précédente, Section 4.1, à partir de la page 127.

4.2.1. Finances personnelles : Prêts hypothécaires

Cet exemple suppose que vous savez calculer un prêt hypothécaire. Si ce n'est pas le cas, vous pourriez lire un autre exemple, ou peut-être souhaitez-vous en savoir plus sur les prêts hypothécaires, auquel cas vous devriez poursuivre votre lecture. La notation suivante, affectueusement appelée « la soupe à l'alphabet » par les étudiants en commerce, est assez courante dans les manuels de finance, mais avec quelques variations. • La

durée du prêt (ou d'un investissement) est de t années. • Le taux d'intérêt nominal par an est r. (Remarque : 5 % est désigné.) comme r = 0,05.)

• Le nombre de fois par an que l'intérêt est composé est m. Par exemple, trimestriellement est m = 4 et hebdomadairement est m = 52, tandis que mensuellement est m = 12 et annuellement est m = 1. • Le nombre de compositions sur la durée du prêt (ou d'un investissement) est n = mt. • Le taux d'intérêt par période de composition (au lieu de par an) est i = r/m. Par exemple, 8 % composé trimestriellement a i = 0,02 tandis que 6 % composé mensuellement est i = 0,005. • Pour les prêts (ou investissements) avec des paiements égaux régulièrement espacés, le montant de ce paiement est c. • Pour les prêts (ou investissements) avec un seul paiement initial, ce paiement Le ment est appelé le principal et est noté P.

Quelques problèmes de préparation :

Si vous déposez 1 000 $ une fois et que vous le laissez pendant trois ans au taux de 4,8 % composé mensuellement, alors r = 0,048, m = 12, i = r/m = 0,004, t = 3, n = 36, P = 1 000. La formule pour le montant que vous avez à la fin est simplement la formule des intérêts composés (que différents livres composent légèrement différemment) :

$$A = P\left(1 + \frac{r}{m}\right)^{mt} = P(1 + i)^{n}$$

La réponse peut être obtenue dans Sage en tapant

```
(1000)*(1 + 0,048/12)^{(12*3)}
```

ce qui est 1154,55 $.

Machine Translated by Google

Si vous épargnez pour votre retraite et souhaitez déposer 100 $ par chèque de paie bimensuel pendant 40 ans et investir au taux de 9,1 % composé bimensuellement.
Comme il y a 52 semaines par an, les chèques de paie bimensuels arriveront 26 fois par an et m = 26.
On a alors t = 40, r = 0,091, i = 0,0035, n = 1040 et c = 100. La formule correcte dans ce cas est la « valeur future d'une rente » ou la formule de « rente croissante » et est donnée par (1 + i) n − 1 VP = c

$$\frac{\quad}{je} = c\ \frac{(1 + tr/min)^{mt} - 1}{tr/min}$$

avec laquelle nous pouvons évaluer

(100)*((1 + 0,0035)^1040 - 1) / 0,0035

obtenant la somme plutôt jolie de 1 052 872,11 $.

Enfin, si vous souhaitez obtenir un prêt immobilier sur 30 ans, au taux de 5,4 % composé mensuellement, avec des mensualités de 900 $, vous aurez alors t = 30, m = 12, n = 360, r = 0,054, i = r/m = 0,054/12 = 0,0045 et c = 900. La formule correcte pour ce problème est la « valeur actuelle d'une rente » ou la formule de la « rente décroissante » et est donnée par

$$PV = c\ \frac{1 - (1 + i)\ n}{je} = c\ \frac{1 - (1 + tr/min)\ tr/^{mt}}{min}$$

avec laquelle nous pouvons évaluer

(900)*(1 - (1 + 0,0045)^(-360)) / 0,0045

On obtient un résultat de 160 276,16 $. Selon votre lieu de résidence, cela peut être une très belle maison de ville. Par exemple, si vous trouvez un bel appartement dont le prix demandé est de 178 084,62 $, verser un apport de 10 % impliquerait un apport de 17 808,46 $, ce qui laisserait 160 276,16 $ à la banque pour ce prêt hypothécaire.

Création d'un tableau de calcul du coût pour mille : Les trois

exercices d'échauffement ci-dessus pourraient vous être faciles, car vous êtes assez avancé en mathématiques. De nombreux employés de banque ne sont pas diplômés de l'enseignement supérieur et sont incapables de réaliser les calculs précédents. Pour permettre à ces employés de recruter des clients et de leur proposer des prêts hypothécaires, la technique du « coût pour mille » est utilisée.

Une « grille tarifaire » est publiée par la banque soit quotidiennement, soit hebdomadairement. Ces feuilles indiquent un « coût pour mille » ou « CPT » pour chaque prêt bancaire. Par exemple, il devrait y avoir un CPT pour un prêt hypothécaire à taux fixe sur 30 ans, un pour un prêt hypothécaire à taux fixe sur 15 ans, et éventuellement trois CPT pour les prêts automobiles (de 36, 48 et 60 mois), suivis des autres prêts (pour les bateaux, les études, etc.).

Poursuivons avec l'exemple ci-dessus, avec des prêts hypothécaires sur 30 ans et un taux d'intérêt mensuel de 5,4 %, le CPT s'élève à 5,6153079187. (Je vous expliquerai comment le calculer dans un instant.) Puisque vous comprenez les mathématiques,

Machine Translated by Google

Si un client vous demandait le montant des mensualités pour une maison de 180 000 $, vous pourriez insérer 180 000 $ dans l'équation de la valeur actualisée (VA) ci-dessus, insérer n = 360 et i = 0,0045, et calculer simplement pour c. Cependant, les employés de banque les plus modestes ne peuvent pas le faire. Ils chercheraient plutôt le CPT sur la grille tarifaire, trouveraient qu'il est de 5,6153079187, et calculeraient simplement (180)(5,6153079187) = 1010,75 · · ·

leur permettant d'indiquer au client que 1 010,75 $ représenteront leur paiement mensuel pour la maison avec cette hypothèque.

En attendant, la grille tarifaire doit bien provenir de quelque part. Un employé de banque de niveau intermédiaire calculerait le CPT lors de la génération de la « grille tarifaire » en prenant

$$PV = ci \quad \frac{1 - (1 + i)\, n}{}$$

résoudre pour c et obtenir

$$c = \frac{(PV\,)(i)\,1}{-\,(1 + i)\, n}$$

où l'on peut insérer les valeurs i et n. Naturellement, PV = 1 000. Calculons maintenant cette valeur avec Sage.

Puisque c = 1000 ne va pas changer, nous pouvons mettre comme première ligne

```
c=1000
```

Ensuite, nous pourrions définir notre propre fonction en tapant

« CostPerThousand (i, n) » = (c * i) / (1 - (1+i)^(-n)) sur la deuxième ligne. Notez que notre définition de c = 1000 ici est similaire à celle de k = 355/113 à la page 32.

En conservant ces deux lignes, nous pouvons utiliser la fonction à plusieurs reprises comme troisième ligne. Par exemple, pour un prêt immobilier sur 30 ans à 6,5 % composé mensuellement, nous pouvons saisir « comme troisième ligne ».

```
Coût par mille (0,065/12, 30*12)
```

et puis si le taux monte à 7%, nous pouvons remplacer cette ligne par

```
CostPerThousand (0,07/12, 30*12)
```

et ainsi de suite, et ainsi de suite.

Notez que le nom de la fonction doit être composé d'un seul mot, mais en variant les majuscules de cette façon (ce que les programmeurs appellent « Camel Case »), nous pourrions exprimer une idée en plusieurs mots. Il est certainement beaucoup plus facile de lire « CoûtPourMille » que « coûtpourmille » ou « COSTPERTHOUSAND ».

Bien que cette fonction soit intéressante (elle nous évitera par exemple de nous angoisser à chaque fois sur le placement des parenthèses), elle n'est pas non plus tout à fait parfaite. Nos données seront exprimées en nombre d'années du prêt t, et non en nombre de remboursements n = mt = 12t. De même, le taux sera le taux nominal publié r, et non le taux par période i = r/m = r/12. Un étudiant m'a un jour expliqué cela en disant que n et i sont faciles à calculer, tandis que t et r sont faciles à comprendre.

Machine Translated by Google

Cela peut être résolu en définissant une nouvelle fonction. Pour éviter toute confusion, notre bloc de code dans la fenêtre Sage Cell Server devrait ressembler à ceci :

```
c=1000
CoûtPourMille (i, n) = ( c * i ) / (1 - (1+i)^(-n) )
CoûtPourMille1 (r, t)=CoûtPourMille( i=r/12, n=12*t )
```

Nous avons maintenant créé une nouvelle formule, qui utilise le taux exprimé en r, comme souhaité, et le nombre d'années t, et non le nombre de périodes de capitalisation. Pour un prêt immobilier sur 30 ans à 6,5 % capitalisé mensuellement, nous pouvons saisir

```
Coût par mille1 ( 0,065, 30)
```

et obtenir une saisie et une sortie plus conviviales. Dernier rappel : lorsque vous travaillez avec des sommes importantes, comme le prix d'une maison, veillez à ne jamais utiliser de virgule comme séparateur de milliers. Tous les chiffres doivent être consécutifs, comme nous l'avons vu aux pages 2 et 3.

Les symboles de groupement :

Certains manuels de mathématiques anciens ont l'habitude d'utiliser des crochets, [et], comme des super-parenthèses. Cela permettait d'éviter d'avoir des groupes de parenthèses à l'intérieur d'autres parenthèses. Une personne élevée avec de tels livres pourrait être amenée à écrire.

$$f(i, n) = (c \cdot i) \div 1 - (1 + i) - n$$

au lieu du plus moderne

$$f(i, n) = (c \cdot i) \div 1 - (1 + i) - n$$

qui est identique sauf que le [devient un (et le] devient un).

Les mathématiciens peuvent écrire ce qu'ils veulent sur papier, mais le code Sage offre beaucoup moins de flexibilité. Cela est principalement dû au fait que Sage repose sur le langage informatique Python, et que les conventions Python doivent donc être respectées.

Par conséquent, on ne peut pas taper

CoûtParMille (i, n) = (c * i) / [1 - (1+i)^(-n)] mais doit plutôt taper

CoûtPourMille (i, n) = (c * i) / (1 - (1+i)^(-n))

N'oubliez pas que dans Sage, les seuls opérateurs de regroupement sont les parenthèses. Des explications complémentaires sont disponibles en page 2.

4.2.2. Physique : Gravitation et satellites Imaginons que

vous analysiez un satellite en orbite autour de la Terre. La formule de Newton pour la gravité est :

$$F = \frac{GMeMs}{r_s^2}$$

Machine Translated by Google

où G = 6,77 × 10−11N m2 /kg2 est la constante universelle de la gravitation ; Me = 5,9742 × 1024 kg est la masse de la Terre ; ensuite rs est la distance du satellite au centre de la Terre en mètres ; et enfin Ms kg est la masse du satellite.

G et moi ne changeront sûrement pas, nous pouvons donc les définir comme les deux premiers lignes de notre morceau de code en tapant G

```
= 6,77 * 10^(-11)
Moi = 5,9742 * 10^(24) comme
```

nous avions défini k = 355/113 à la page 32.

Contrairement à la masse de la Terre ou à la constante universelle de gravitation, la distance au satellite peut varier assez fréquemment, et sa masse peut également varier (notamment s'il transporte du carburant, qui sera brûlé lors des manœuvres). Ces deux quantités doivent être des variables et non des constantes. Cela signifie que notre fonction de force sera une fonction de deux variables. Nous écrirons sur la troisième ligne :

```
Force( Ms, rs ) = (G * Me * Ms)/(rs^2)
```

En utilisant ces trois lignes, vous pouvez saisir (sur la quatrième ligne) un code tel que :

```
Force( 1000, 10 * 10^6 )
```

Pour obtenir des réponses. Dans ce cas, vous affirmez que la masse du satellite est de 1 000 kg et que la distance entre le satellite et le centre de la Terre est de 107 mètres ; la réponse devrait être de 4 044,53 Newtons. Cette notation est idéale si vous avez un grand nombre de valeurs r ou Ms qui vous intéressent et que vous souhaitez économiser de la saisie. Il vous suffit de modifier un nombre, voire deux, et de cliquer plusieurs fois sur « Évaluer ».

Supposons maintenant que vous souhaitiez que la fonction fonctionne en termes d'altitude, mesurée depuis la surface de la Terre, et non depuis son centre. Il est facile de comprendre qu'il suffit d'ajouter le rayon de la Terre à l'altitude pour obtenir la distance entre le satellite et son centre. Il faut d'abord saisir le rayon de la Terre, soit 6 378,1 km, mais il faut bien sûr utiliser des mètres pour éviter les erreurs dues à des unités contradictoires.

Pour éviter toute confusion, notre bloc de code entier devrait maintenant ressembler à

```
G = 6,77 * 10^(-11)
Moi = 5,9742 * 10^(24) re = 6378100

Force( Ms, rs ) = (G * Me * Ms)/(rs^2)
Force1( Ms, alt ) = Force( Ms, rs = alt + re )
```

La cinquième ligne est très intéressante. Nous avons redéfini rs comme la somme de l'altitude et du rayon de la Terre. Notez également qu'il ne faut jamais utiliser de virgule comme « séparateur de milliers » dans les grands nombres en programmation Sage : tous les chiffres doivent être consécutifs, comme nous l'avons vu aux pages 2 et 3. Il ne nous reste plus qu'à utiliser Force1. Par exemple, comme sixième ligne, nous pouvons saisir

Machine Translated by Google


```
Force1( 1000, 3621900 )
```

Ce qui renvoie 4 044,53 N, soit exactement la valeur obtenue précédemment. En effet, une altitude de 3 621,9 km équivaut à une distance de 107 mètres du centre de la Terre. Ceci est également vrai car

$$3\,621{,}9 \text{ km} + 6\,378{,}1 \text{ km} = 10\,000 \text{ km}$$

Quelle fonction utiliser ? Force ou Force1 ? Je suppose que cela dépend si le problème posé est en termes d'altitude ou en termes de r (la distance du satellite au centre de la Terre).

L'idée de superposer des fonctions à d'autres est très puissante. J'ai souvent écrit de grandes suites de fonctions de cette manière – une fonction sur l'autre, puis sur la suivante – pour modéliser des phénomènes complexes dans Sage. De cette manière, le cerveau humain n'a besoin de considérer qu'un seul élément à la fois avant de passer au composant suivant, ce qui rend les systèmes volumineux et complexes accessibles et assimilables.

4.3. Dérivées et gradients en calcul multivarié

Dans la section 1.11, page 49, nous avons appris à calculer les dérivées de fonctions univariées. Nous allons maintenant étudier les dérivées multivariées.

4.3.1. Dérivées partielles

Vous ne serez probablement pas surpris d'apprendre que Sage peut effectuer des dérivées partielles très facilement. Par exemple, si vous avez

$$g(x, y) = xy + \sin(x^2) + e^{-x}$$

tu peux taper

```
g(x,y) = x*y + sin(x^2) + e^(-x) diff(g(x,y), x) et
obtenir
```

2*x*cos(x^2) + y - e^(-x), soit ∂g/∂x. De même, vous pouvez taper g(x,y) = x*y + sin(x^2) + e^(-x) et derivative(g(x,y), y) pour apprendre que ∂g/∂y est simplement x. Notez ici que cela montre bien que diff et derived sont synonymes.

Vous pouvez même trouver

$$\frac{\partial^2}{\partial x \partial y} f$$

en tapant « derivative(g(x,y), x, y) ». La réponse est bien sûr 1.

Machine Translated by Google

4.3.2. Gradients

La question suivante pourrait nous être posée : trouver le gradient d'une fonction multivariée. Pour obtenir f, tapez g(x,y) =

x*y + sin(x^2) + e^(-x) g.derivative() et
Sage calculera

correctement le gradient. Vous pouvez également taper diff(g) à la place de g.derivative(). Dans les deux cas, Sage affiche (x, y) |--> (2*x*cos(x^2) + y

- e^(-x), x), ce qui rappelle qu'il s'agit d'une application

d'un point (x, y) à une paire de nombres réels, et non d'une fonction ordinaire. Il est intéressant de noter que le gradient d'une fonction univariée telle que $f(x) = 2e^{-x}$, ainsi calculé, n'est que la dérivée première ordinaire, ce qui est parfaitement correct selon les lois du calcul différentiel et intégral.

4.4. Matrices dans Sage, deuxième partie

Contrairement à la section sur les matrices du chapitre 1 (voir page 19), cette section suppose une connaissance du premier semestre d'algèbre matricielle (parfois appelée algèbre linéaire). Cependant, je me suis efforcé de la rendre aussi lisible et intuitive que possible. Les lecteurs peu familiarisés avec l'algèbre matricielle et souhaitant acquérir une expertise feraient bien de lire « A First Course in Linear Algebra » de Robert Beezer, publié en version électronique en 2007. Non seulement ce manuel est entièrement gratuit, mais il a été conçu pour Sage et est extrêmement facile à lire.

4.4.1. Définition de quelques exemples.

Nous allons examiner les exemples suivants dans cette section. Considérons d'abord la très modeste matrice 3 × 3 :

$$A = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & -1 \end{matrix}$$

et considérons également b =< 24, 63, 52 > ainsi que c =< 10, 14, 8 >.

Notez que certains ouvrages préfèrent écrire b =< 24, 63, 52 > pour un vecteur et B = (24, 63, 52) pour un point de l'espace. Cette distinction est excellente, utile aux étudiants comme aux enseignants, et les manuels de mathématiques devraient l'utiliser. Cependant, cette notation est également récente et peut être méconnue de nombreux lecteurs habitués à voir des parenthèses utilisées à la fois pour les points et les vecteurs.

Comme Sage utilise déjà les symboles < et > pour les comparaisons, à savoir « inférieur à » et « supérieur à », il est malheureusement impossible pour Sage (comme tous les manuels, sauf les plus récents) d'utiliser la nouvelle notation. C'est inévitable. Cependant, la commande vector élimine toute confusion, comme vous le verrez dans un instant.

Machine Translated by Google

À ce stade, vous savez comment définir A avec
A = matrice(3, 3, [1, 2, 3, 4, 5, 6, 7, 8, -1]) comme nous l'avons vu à la
page 22, mais pour b et c, nous avons le choix. Considérons b comme une matrice à 3 lignes et 1 colonne. Dans ce cas, nous écririons
b = matrix(3, 1, [24, 63, 52]) mais il existe
également la commande vector, qui a la syntaxe suivante : c = vector([10, 14, 8]) qui
nécessite moins de saisie et est plus
lisible par l'homme.

Prenez un moment dans un serveur de cellules Sage pour définir ces trois objets, à savoir A, b et c. Pour le reste de cette section, conservez ces trois lignes dans la cellule Sage. Ainsi, vous n'aurez pas à les retaper et pourrez simplement modifier la dernière ligne de votre cellule au fur et à mesure que nous apprendrons les différentes opérations.

4.4.2. Multiplication et exponentiation de matrices. Le symbole * ou
l'astérisque s'utilise pour multiplier des matrices, tout comme pour multiplier des nombres rationnels. Par exemple, on peut taper A*b ou A*A et obtenir la réponse souhaitée. De même, l'addition et la soustraction de matrices s'effectuent avec les signes plus et moins habituels, comme pour additionner ou soustraire des nombres.

Il est parfois utile d'avoir la matrice d'identité prête. Vous pourriez définir
Vous pouvez utiliser la matrice
identité avec Id = matrix(3, 3, [1, 0, 0, 0, 1, 0, 0, 0, 1]) si vous le souhaitez.
Cependant, cela peut être fastidieux, surtout pour les grandes tailles. J'ai notamment souvent du mal à placer le bon nombre de zéros aux bons endroits. Vous pouvez donc définir la matrice identité avec Id = identity_matrix(3), une formule beaucoup plus courte. Notez que j'ai
l'habitude d'appeler la matrice
identité Id et non I. Ceci est uniquement pour des raisons de lisibilité, car un « I » majuscule peut être confondu avec un L minuscule, le chiffre un ou la barre oblique inverse. Vous pouvez appeler la matrice identité comme vous le souhaitez. Il est utile de noter que Id*A et A*Id produiront les bonnes réponses, à savoir A.

Vous pouvez également élever des matrices à différentes puissances. Le code suivant :
A = matrix(3, 3, [1, 2, 3, 4, 5, 6, 7, 8, -1]) print A*A*A print print A^3
démontre que
Sage
peut effectuer
la multiplication matricielle beaucoup plus rapidement qu'un humain. Il montre également comment les exposants de matrice utilisent la même notation que l'exposant de n'importe quel nombre.

Afin de préserver la règle (An)(Am) = An+m du nombre réel
Dans le monde, les mathématiciens définissent les exposants négatifs par A−n = (A−1) n, , pro-
à condition que A soit inversible. De plus, A0 est la matrice identité. Il existe en fait

Machine Translated by Google

une notion de « racine carrée » d'une matrice, mais elle est rarement utilisée et nous n'y reviendrons pas ici.

4.4.3. Résolution des systèmes droit et gauche

Utilisons maintenant les objets définis précédemment dans cette section pour résoudre quelques problèmes. Pour trouver un vecteur x tel que Ax = b, nous tapons A.solve right(b) et apprenons que la solution est x =< 7, 1, 5 >. Nous pouvons également vérifier cela en tapant

A*vector([7,1,5]) et nous

récupérons le vecteur < 24, 63, 52 >, comme souhaité.

Pour trouver un vecteur y tel que yA = c, nous tapons A.solve left(c) et apprenons que la solution est y =< 3, 0, 1 >. Comme précédemment, nous vérifions cela en tapant vector([3,0,1])*A

et nous obtenons le

vecteur < 10, 14, 8 >, comme souhaité. Il y a un petit point technique ici. Le vecteur < 7, 1, 5 > était un vecteur colonne, ou une matrice à 3 lignes et 1 colonne. Le vecteur < 3, 0, 1 > était un vecteur ligne, ou une matrice à 1 ligne et 3 colonnes. Sage essaiera toujours de déterminer si vous vouliez parler d'un vecteur ligne ou d'un vecteur colonne lors de l'utilisation de la commande vector. Cependant, certains enseignants peuvent être moins indulgents et vous demander d'écrire yT A au lieu de yA.

L'opérateur barre oblique inverse peut être utilisé pour abréger « solve right ». Notez que la barre oblique inverse n'est pas celle qui comporte un point d'interrogation, mais celle qui comporte une barre verticale. (La barre oblique près du point d'interrogation est appelée « barre oblique » ou barre oblique.) Nous

taperions A \ b.

comme abréviation de A.solve right(b). Cette abréviation extrêmement concise est un clin d'œil au très ancien langage MATLAB, toujours utilisé dans l'industrie, et qui possède le même opérateur de barre oblique inverse.

Nous pouvons également taper A.augment(b) pour obtenir la matrice

$$\left|\begin{matrix} 1 & 2 & 3 & 24 \\ 4 & 5 & 6 & 63 \\ 7 & 8 & -1 & 52 \end{matrix}\right|$$

En préparation à l'utilisation de la commande rref, étudiée page 22. Comme vu précédemment, Sage ne trace pas de ligne verticale marquant la dernière colonne de la matrice de manière spécifique. Certains manuels l'omettent également. Pour utiliser la stratégie augment-and-RREF, nous saisirions A = matrix(3, 3, [1, 2, 3, 4, 5, 6,

7, 8, -1])

b = matrice(3, 1, [24, 63, 52])

M = A.augmentation(b)

Machine Translated by Google


```
imprimer "Avant :"
imprimer
M
imprimer imprimer
"Après :" imprimer M.rref()
```

Bien sûr, les commandes d'impression ne sont pas vraiment nécessaires, mais elles sont très instructives pour l'utilisateur. Certains mathématiciens utilisent l'opérateur barre oblique inverse ou la commande « solve right ». Cependant, cette stratégie « augment-and-RREF » présente des avantages.

Envisagez de modifier la dernière entrée dans A, à savoir A33, en 9 au lieu de −1. Exécutez ensuite à nouveau notre code augment-and-RREF.

Nous voyons la sortie suivante

```
Avant : [ 1
 2 3 24] [ 4 5 6 63] [ 7
 8 9 52]


Après:
 [ 1 0 -1 0] [ 0 1 2
 0] [ 0 0 0 1]
```

C'est extrêmement instructif. La ligne de zéros suivie d'un 1 montre que le système d'équations initial n'a pas de solution. On constate également que les première et deuxième colonnes sont des vecteurs effectifs, ou élémentaires, tandis que la troisième colonne est un vecteur défectif, ou non élémentaire. Si vous savez ce que signifie « rang » en algèbre linéaire, vous constaterez que la matrice est de rang 2. Si vous ne savez pas ce que cela signifie, ne vous en préoccupez pas pour l'instant. Voyons ce qui se passerait si nous posions la même question à l'opérateur barre oblique inverse.

Si nous devions taper

```
A = matrice( 3, 3, [1, 2, 3, 4, 5, 6, 7, 8, 9] )

b = matrice( 3, 1, [24, 63, 52] )

Un \ b
```

nous obtiendrions la réponse :

ValueError : l'équation matricielle n'a pas de solutions, ce qui est indéniablement vrai mais pas aussi informatif que la stratégie augment-and-RREF.

Que se passe-t-il si l'on utilise la barre oblique inverse alors qu'il existe une infinité de solutions ? Essayons et découvrons-le ! Les commandes : A =

```
matrix( 3, 3, [1, 2, 3, 4, 5, 6, 7, 8, 9] )
```

Machine Translated by Google

d = vecteur([6, 15, 24])

Annonce

produire le vecteur < 0, 3, 0 >, qui est une solution parfaitement valide. Sur la base de ce résultat, nous serions en droit de supposer qu'il s'agit de l'unique solution au problème. Cependant, nous pouvons également constater que < 1, 1, 1 > est une solution. Lorsqu'il existe une infinité de solutions, l'opérateur barre oblique inverse en choisit simplement une. Plus précisément, il choisira la solution formée en mettant toutes les variables libres à zéro.

Nous pourrions plutôt taper

A = matrix(3, 3, [1, 2, 3, 4, 5, 6, 7, 8, 9])

d = vecteur([6, 15, 24])

B = A.augmentation(d)

B.rref()

pour obtenir la sortie

[1 0 -1 0] [0 1 2
3] [0 0 0 0]

Ce qui, si vous savez le lire, est beaucoup plus instructif. Le cas d'un système linéaire ayant un nombre infini de solutions (autrement dit, une variable libre) est abordé dans la section D, page 307.

Enfin, rappelons que A.rref() demande une « forme échelonnée réduite ». Cependant, il existe également la « forme échelonnée » ou REF.
Cette version semi-complète du RREF est extrêmement utile pour manipuler des matrices à la main, mais à l'ère de l'informatique, son rôle est secondaire. La commande permettant de calculer le REF au lieu du RREF est simplement A.echelon_form().

4.4.4. Matrix Inverses

Certaines matrices ont des inverses, d'autres non. On peut calculer l'inverse d'une matrice avec A.inverse(). Par exemple, A = matrix(3, 3,

[1, 2, 3, 4, 5, 6, 7, 8, -1])
B = A.inverse() print

B produit

la sortie

[-53/30 13/15 -1/10] [23/15
-11/15 1/5] [-1/10 1/5 -1/10]

Machine Translated by Google

Vous pouvez vérifier l'exactitude de l'inverse en demandant A*B ou B*A, qui renverront tous deux la matrice identité. Vous pouvez également remplacer A.inverse() par A−1 , comme ci-dessous :

```
A = matrice( 3, 3, [1, 2, 3, 4, 5, 6, 7, 8, -1] )
B = A^(-1)
imprimer B
```

Vous vous demandez peut-être ce qui se passe si vous inversez une matrice singulière ? Nous pouvons simplement changer le -1 dans le coin inférieur droit de la matrice en 9. La matrice est alors singulière. On obtient un message d'erreur composé de beaucoup de déchets, suivi de

```
ZeroDivisionError : la matrice d'entrée doit être non singulière
```

Il est parfois utile de calculer l'inverse d'une matrice « par le chemin le plus long ». Pour ce faire, nous attachons une matrice d'identité de la taille appropriée, à à droite de la matrice d'origine. Par exemple, le code

```
A = matrice( 3, 3, [1, 2, 3, 4, 5, 6, 7, 8, 9] )
B = matrice_identité(3)
C = A.augmentation(B)

imprimer « Avant : »
imprimer C

imprimer « Après : »
imprimer C.rref()
```

produit la sortie ci-dessous.

```
Avant:
[1 2 3 1 0 0]
[4 5 6 0 1 0]
[7 8 9 0 0 1]

Après:
[[[ 1 0 -1 0 -8/3 5/3]
    0 1 2 0 7/3 -4/3]
    0 0 0 1 -2 1]
```

Comme vous pouvez le constater, la matrice n'est pas inversible. Le processus RREF a obtenu « bloqué » dans la troisième colonne. C'est beaucoup plus instructif que l'erreur. message que nous obtiendrions si nous essayions simplement d'inverser A avec A.inverse().

4.4.5. Calcul du noyau d'une matrice

Pour toute matrice A spécifique, il existe de nombreuses situations où il est utile de savoir quel ensemble de vecteurs v aura vA = 0. De même, il y a de nombreuses fois lorsqu'il est utile de savoir quel ensemble de vecteurs v aura Av = 0. Ces deux les ensembles sont appelés respectivement « noyau gauche de A » et « noyau droit de A ».

Explorons un exemple :

Machine Translated by Google


```
A = matrice( 2, 2, [-2, 7, 0, 0] )

imprimer A.left_kernel()

print "Test :" print
vector([0, 1])*A print A*vector([0,
1]) produit la sortie :
```

```
Module libre de degré 2 et de rang 1 sur anneau entier
Matrice de base des échelons :
 [0 1]
Test :
 (0, 0) (7,
 0)
```

Il est à noter que notre code de test vérifie que vA = 0, comme souhaité, mais que Av = 0, comme il s'avère. Ceci souligne que les noyaux gauche et droit ne sont pas identiques.

J'aimerais maintenant expliquer les deux premières lignes du résultat ci-dessus. Comme nous le verrons tout au long de ces exemples, le « degré 2 » fait référence à la dimension des vecteurs du noyau. Le « rang » est la dimension de l'espace qui est ramené à zéro ; on parle généralement de « nullité » de A, mais il faut faire la distinction entre « nullité gauche » et « nullité droite ». Il est important de souligner que le rang du noyau gauche, le rang du noyau droit et le rang de A lui-même peuvent être différents.

Par exemple, la matrice

$$\begin{matrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 0 \end{matrix}$$

a un rang 2, mais son noyau gauche est de rang 0 (juste le vecteur zéro) et son noyau droit est de rang 3. Vous pouvez trouver ces nombres immédiatement avec les fonctions A.right nullity(), A.left nullity() et A.rank(), pour toute matrice A.

Pendant ce temps, le code équivalent utilisant le noyau droit serait le suivant : A = matrix(2, 2, [-2, 7, 0, 0])

```
imprimer A.right_kernel()

imprimer "Test :"
imprimer vector([7, 2])*A imprimer
A*vector([7, 2]) qui produit la sortie
```

```
Module libre de degré 2 et de rang 1 sur anneau entier
Matrice de base des échelons :
 [7 2]
Test:
```

Machine Translated by Google

(-14, 49) (0,
0)

On constate à nouveau que Av = 0, comme souhaité, mais que vA = 0. Il est certain que le noyau gauche et le noyau droit sont des objets mathématiques distincts. Le noyau droit est parfois appelé « espace nul » de A. C'est pourquoi, lorsqu'on parle du noyau de A en mathématiques, on désigne généralement le noyau droit.
Cependant, dans Sage, A.kernel() fait référence au noyau gauche, ce qui est extrêmement inhabituel. Un programmeur avisé sera explicite et utilisera A.right kernel() et A.left kernel() afin d'éviter toute confusion.

D'ailleurs, j'en ai discuté un jour avec le professeur William Stein, et il m'a expressément demandé de mentionner dans le livre que les « vrais programmeurs » consultent toujours la documentation. Personne ne se souvient réellement de détails précis comme la distinction entre le noyau gauche et le noyau droit. Même si vous êtes sûr de vous souvenir de la différence, il est préférable de vérifier quand même (en utilisant les techniques de la section 1.10, page 47) pour éviter toute erreur.

Pour illustrer plus efficacement le concept, j'ai écrit un programme. Voir la figure 1, page 143. Vous y voyez une matrice plus grande et nous calculons très soigneusement les noyaux gauche et droit. Vous pouvez également constater que si a et b sont dans le noyau, alors r1a + r2b le seront également, quels que soient les nombres réels r1 et r2 choisis.

Applications du calcul des noyaux Nous venons

de consacrer beaucoup de temps au calcul des noyaux gauche et droit d'une matrice. Vous vous demandez peut-être à quoi cela sert.
Il existe plusieurs utilisations en mathématiques pures. En mathématiques appliquées, l'une d'elles concerne les systèmes d'équations linéaires à solutions infinies.

Une autre utilisation, issue de la cryptographie, est l'algorithme du crible quadratique, qui permet de factoriser le produit de deux grands nombres premiers. Il s'agit d'une méthode d'attaque du cryptosystème RSA. J'aborde d'ailleurs en détail le crible quadratique au chapitre 10 d'un autre livre que j'ai écrit, Algebraic Cryptanalysis, publié par Springer en 2009. Si la majeure partie de ce livre requiert des connaissances acquises lors d'un cours de cryptographie, le chapitre consacré au crible quadratique est relativement indépendant et serait accessible à toute personne ayant suivi un cours de théorie des nombres, voire deux semestres d'algèbre abstraite, parfois appelée algèbre moderne.

4.4.6. Déterminants

Revenons brièvement à la matrice avec laquelle nous avons ouvert cette leçon. Pour connaître son déterminant, il suffit de saisir :

```
A = matrice( 3, 3, [1, 2, 3, 4, 5, 6, 7, 8, -1] ) det(A)
```

Machine Translated by Google

Code du noyau gauche

```
A = matrice( 4, 4, [1, 2, 3, 0, 4, 5, 6, -1, 7, 8, 9, -2, 12, 15, 18,
                    -3] )

imprimer « Matrice d'entrée : »
imprimer
A imprimer

imprimer "Nullspace:"
imprimer A.left_kernel()
imprimer

xvec = vecteur( [1, 1, 1, -1] ) yvec =
vecteur( [0, 3, 0, -1] )

print "Tests :" print
xvec, "correspond à", xvec*A print yvec,
"correspond à", yvec*A print (xvec+yvec),
"correspond à", (xvec+yvec)*A print (xvec-2*yvec),
"correspond à", (xvec-2*yvec)*A print (17*xvec + 47*yvec),
"correspond à", print (17*xvec + 47*yvec)*A print
(71*xvec + 888*yvec), "correspond à",
print (71*xvec + 888*yvec)*A
```

Code du noyau droit

```
A = matrice( 4, 4, [1, 2, 3, 0, 4, 5, 6, -1, 7, 8, 9, -2, 12, 15, 18,
                    -3] )

imprimer « Matrice d'entrée : »
imprimer
A imprimer

imprimer "Nullspace:"
imprimer A.right_kernel()
imprimer

xvec = vecteur( [1, 1, -1, 3] ) yvec =
vecteur( [0, 3, -2, 3] )

print "Tests :" print
xvec, "correspond à", A*xvec print yvec,
"correspond à", A*yvec print (xvec+yvec),
"correspond à", A*(xvec+yvec) print (xvec-2*yvec),
"correspond à", A*(xvec-2*yvec) print (17*xvec + 47*yvec),
"correspond à", print A*(17*xvec + 47*yvec) print
(71*xvec + 888*yvec), "correspond
à", print A*(71*xvec + 888*yvec)
```

Sortie du noyau gauche

```
%%% noyau_gauche
Matrice d'entrée :
 [ 1 2 3 0] [ 4 5 6
 -1] [ 7 8 9 -2] [12
 15 18 -3]

Espace nul :
Module libre de degré 4 et de rang 2 sur l'anneau entier
Matrice de base d'échelon : [ 1 1 1
 -1] [ 0 3 0 -1]

Tests:
(1, 1, 1, -1) correspond à (0, 0, 0, 0) (0, 3, 0, -1)
correspond à (0, 0, 0, 0) (1, 4, 1, -2) correspond
à (0, 0, 0, 0) (1, -5, 1, 1) correspond à (0, 0, 0,
0) (17, 158, 17, -64) correspond à (0, 0, 0, 0)
(71, 2735, 71, -959) correspond à (0, 0, 0, 0)
```

Sortie du noyau droit

```
 %%% ceci utilise « right_kernel »
Matrice d'entrée :
 [ 1 2 3 0] [ 4 5 6
 -1] [ 7 8 9 -2] [12
 15 18 -3]

Espace nul :
Module libre de degré 4 et de rang 2 sur l'anneau entier
Matrice de base d'échelon : [ 1 1 1 -1
 3] [ 0 3 -2 3]

Tests:
(1, 1, -1, 3) correspond à (0, 0, 0, 0) (0, 3, -2,
3) correspond à (0, 0, 0, 0) (1, 4, -3, 6)
correspond à (0, 0, 0, 0) (1, -5, 3, -3) correspond
à (0, 0, 0, 0) (17, 158, -111, 192) correspond à
(0, 0, 0, 0) (71, 2735, -1847, 2877) correspond à (0, 0, 0,
0)
```

Figure 1. Le très grand exemple pour les commandes du noyau gauche et du noyau droit.

Nous verrions alors que le déterminant est 30. Si nous remplaçons −1 par 9, le déterminant devient 0. (N'oubliez pas de cliquer sur « Évaluer » après cette modification.) Cela est logique, car nous avons constaté que A avait une solution unique lors de la résolution du système avec le vecteur < 24, 63, 52 >, et −1 toujours présent. En revanche, nous avons constaté que A \ b n'avait aucune solution.

Machine Translated by Google

Opération	Notation mathématique	Sage Syntaxe	Réponse attendue
Ajout	a + b	a + b	< 5, 7, 9 >
Soustraction	a −b	a - b	< −3, −3, −3 >
Produit scalaire	3a	3*a	< 3, 6, 9 >
produit scalaire	un · b	a.produit scalaire(b)	32
Produit vectoriel	un ×b	a.produit vectoriel(b)	< −3, 6, −3 >
Norme ou longueur	||a||	norme(a)	$\sqrt{14}$

Tableau 1. Opérations vectorielles de base dans Sage

lorsque nous avons essayé de résoudre le système avec ce même vecteur, mais après avoir modifié le −1 en 9.

Les cas de « solutions infinies » et d'« aucune solution » peuvent et vont se produire. se produit uniquement avec un déterminant nul. Le cas d'une solution unique est ce qui se produit si et seulement si le déterminant est différent de zéro.

4.5. Opérations vectorielles

Les vecteurs sont abordés dans de nombreux cours, notamment l'algèbre linéaire et le calcul multivarié, ainsi qu'en physique I et ses cours suivants. Sage est très performant. au travail avec les vecteurs. Nous explorerons ici les opérations de base : addition, soustraction, prise d'un multiple scalaire, calcul d'un produit scalaire, calcul un produit vectoriel et la mesure de la norme d'un vecteur. Nous combinerons ensuite en explorant comment calculer l'angle entre deux vecteurs. Nous allons utilisons, comme exemple courant, les deux vecteurs suivants :

$$a =< 1, 2, 3 > \text{ et } b =< 4, 5, 6 >$$

Chacune des commandes du tableau 1 est destinée à être précédée de la définitions de ces vecteurs, à savoir via :

```
a = vecteur( [1, 2, 3] )
b = vecteur( [4, 5, 6] )
```

Pour un exemple de la façon de combiner plusieurs d'entre eux pour réaliser une tâche utile Tâche : trouvons l'angle entre a et b. Il faut rappeler la formule

$$\cos\theta = \frac{\text{un} \cdot b}{||a||\ ||b||}$$

et ensuite nous pouvons taper

```
a = vecteur( [1, 2, 3] )
b = vecteur( [4, 5, 6] )

produit scalaire = a.produit_scalaire(b)
mafraction = produit scalaire / ( norme(a) * norme(b) )
thêta = arccos(mafraction)
```

Machine Translated by Google


```
imprimer N(theta), "radians" imprimer
N(theta*180/pi), "degrés"
```

Cela donnera le résultat suivant :

```
0,225726128552734 radians
12,9331544918991 degrés
```

4.6. Travailler avec les entiers et la théorie des nombres

L'étude des problèmes liés aux nombres entiers est appelée « théorie des nombres ». Ce sujet a été reconnu pour sa pureté pendant des siècles, mais ces cinquante dernières années ont vu apparaître des applications étonnantes, notamment la cryptographie et la programmation en nombres entiers.

Ce sujet peut être abordé dès le plus jeune âge (par exemple, « qu'est-ce qu'un nombre premier ? »), mais peut aussi être très avancé. De nombreuses thèses de doctorat sont rédigées chaque année avec de nouveaux résultats en théorie des nombres. La théorie des nombres consiste principalement à rédiger des preuves, et on pourrait penser que Sage n'est pas très utile pour ce type de travaux. En réalité, Sage est très efficace pour travailler avec les nombres entiers et vous aider à trouver des exemples précis de l'effet des instruments de la théorie des nombres sur des nombres entiers spécifiques. Cela peut vous aider à comprendre de nouveaux concepts plus en profondeur, et je trouve que cela peut faire le lien entre le cours magistral et le moment où vous êtes vraiment prêt à rédiger des preuves sérieuses sur un sujet.

Par exemple, vous pouvez taper factor(-2007) pour trouver la factorisation de cet entier, qui est

```
-1 * 3^2 * 223
```

et de même

```
facteur (2008)
```

produit la réponse

```
2^3 * 251
```

Pour trouver le nombre premier suivant après un million, vous pouvez taper

```
next_prime( 10^6 )
```

et découvrez que la réponse est 1 000 003.

Vous pouvez également tester la primalité d'un nombre. Considérez is prime(101) ou is prime(102). Bien sûr, Sage indique True pour 101 et False pour 102. Enfin, vous pouvez trouver tous les nombres premiers dans un intervalle donné.
Par exemple,

```
prime_range(1000,1100)
```

vous donne tous les nombres premiers entre 1000 et 1100.

Machine Translated by Google

4.6.1. Le pgcd et le pcm

Pour trouver le « plus grand dénominateur commun » ou « pgcd » de deux nombres, par exemple 120 et 64, tapez simplement

```
pgcd ( 120, 64 )
```

et vous apprendrez que la réponse est 8. Cela a du sens car 120/8 = 15
Alors que 64/8 = 8, et comme vous pouvez le constater, 15 et 8 n'ont aucun facteur commun. explicitement, nous pouvons regarder les factorisations premières 15 = 5×3 et 8 = 2×2×2.
Une autre façon de voir les choses est

$$120 = 23 \times 3 \times 5 \text{ tandis que } 64 = 26$$

et, comme vous pouvez le voir, tout ce que 120 et 64 ont en commun est seulement 23 , lequel est égal à 8. Par conséquent, le pgcd est 8.

De même, vous pouvez trouver le plus petit commun multiple (PPCM). Ceci est fait en tapant

```
lcm( 120, 64 )
```

d'où vous apprenez que le PPCM est 960. Cela est logique en raison de la ratios 960/120 = 8 et aussi 960/64 = 15, et encore une fois 8 ne partage aucun facteur avec 15. Un mathématicien dirait : « 8 est premier avec 15. » Certains auteurs disent « premiers entre eux » au lieu de « premiers entre eux ».

Une autre façon de voir les choses est que le lcm devrait être

$$6\ 2\ \ 1 \times 3\ \ 1 \times 5\ = 960$$

car le 2 apparaît 6 fois en 64 et 3 fois en 120 (donc un maximum de 6), tandis que le 3 et le 5 apparaissent tous deux 0 fois dans 64 et 1 fois dans 120 (donc un maximum de 1 chacun). En élevant chacun de ces nombres premiers (2, 3 et 5) à ceux les puissances (6, 1 et 1) donnent le lcm.

Avant de continuer, un autre exemple est utile. Travaillons avec un peu des nombres plus grands, peut-être 3600 et 1000. Tout d'abord, nous faisons les factorisations premières

$$3600 = 24 \times 3^{2} \times 5^{2} \text{ tandis que } 1000 = 23 \times 5^{3}$$

et nous pouvons calculer le lcm en prenant l'exposant maximum à chaque fois

$$4\ 2\ \ 2 \times 3\ \ 3 \times 5\ = 18\,000$$

et le pgcd en prenant l'exposant minimum à chaque fois

$$2^{3} \times 3^{0} \times 5^{2} = 200$$

Prenez un moment pour vérifier que 200 entre à la fois dans 1000 et 3600. comme le fait qu'aucun nombre supérieur ne le sera. Ensuite, prenez un moment pour vérifier que 1000 et 3600 entrent tous deux dans 18 000, mais il n'y a pas de nombre inférieur pour ce qui est vrai. Et bien sûr, le produit des nombres divisé par le pgcd est le ppcm, et le produit des nombres divisé par le ppcm est le pgcd. L'idée clé ici est de comprendre que la factorisation en nombres premiers communique une énorme quantité d'informations sur ce nombre.

Si vous souhaitez prendre le pgcd de plusieurs nombres à la fois (par exemple 120, 55, 25 et 35), il n'est pas nécessaire de le faire

Machine Translated by Google


```
pgcd( 120, pgcd(55, pgcd(25, 35 ) ) )
```

parce que tu peux faire à la place

```
pgcd( [ 120, 55, 25, 35 ] )
```

pour obtenir 5 et de même

```
lcm( [ 120, 55, 25, 35 ] )
```

pour obtenir 46 200. Voici un autre exemple de liste dans Sage. Vous pouvez entourer n'importe quelle donnée de [et], et séparer les entrées par des virgules, pour créer une liste. Sage a hérité cette notation du langage informatique Python.

4.6.2. En savoir plus sur les nombres premiers

Une astuce intéressante consiste à trouver le 54 321e nombre premier en tapant nth_prime(54321) et en

découvrant qu'il est

670 177. Vous pouvez vérifier cela avec

```
facteur(670177)
```

qui se renvoie lui-même, confirmant que le nombre est premier.

L'une des choses les plus fascinantes en mathématiques est la distribution des nombres premiers. Par exemple, vous pourriez vouloir observer comment la fonction « n-ième nombre premier » croît à mesure que x croît. En termes simples, quelle est la taille du millionième nombre premier ? Du milliardième nombre premier ? Ou du trillionième nombre premier ?

Il s'avère qu'une bonne approximation de la taille du xième nombre premier est

f(x) = 1,13x logarithme x

mais il existe aussi de meilleures approximations.

Nous pouvons tester cela avec

```
f(x) = 1,13*x*log(x)

imprimer f(10^6)
imprimer nth_prime(10^6)

imprimer f(10^7)
imprimer nth_prime(10^7)
```

Ce code produit la sortie

```
1.56115269304996e7
15485863
1.82134480855829e8
179424673
```

et après ajustement pour la notation scientifique, nous pouvons voir que l'estimation et la réalité sont à environ 1% l'une de l'autre.

Machine Translated by Google

4.6.3. À propos de la fonction Phi d'Euler

Il est parfois utile de savoir combien d'entiers sont premiers avec x, compris entre 1 et x. Par exemple, combien de nombres de 1 à 10 partagent un facteur avec 10, et combien sont premiers avec 10.

Les diviseurs premiers de 10 sont 2 et 5, et donc tout nombre dont le nombre premier la factorisation contient des 2 et des 5 qui partageront un facteur avec 10. Ceux-ci incluent

$$\{2, 4, 5, 6, 8, 10\}$$

et donc les autres nombres

$$\{1, 3, 7, 9\}$$

qui n'ont aucun 2 ni aucun 5 dans leurs factorisations, sont premiers avec 10. En particulier, si vous savez ce qu'est l'arithmétique modulaire, alors ce sont les nombres qui ont des « inverses multiplicatifs » ou des « réciproques » mod 10.
Comme 4 nombres sont premiers avec 10, nous écrirons $\varphi(10) = 4$.

En général, il existe $\varphi(n)$ nombres z avec $1 \leq z \leq n$, tels que z soit premier avec n. Il s'agit d'une façon un peu empirique de définir une fonction mathématique, mais il s'avère que φ est extrêmement utile en arithmétique modulaire, notamment en cryptographie.

Même si vous ne savez pas ce qu'est l'arithmétique modulaire, pensez à 7. Chacun d'entre eux

$$\{1, 2, 3, 4, 5, 6\}$$

est premier avec 7, car le seul nombre premier qui divise 7 est 7, et 7 ne divise aucun de ces nombres. Ainsi, 6 des 6 nombres sont premiers avec 7, et nous écririons alors $\varphi(7) = 6$.

En fait, ce sera toujours le cas pour un nombre premier que $\varphi(p) = p-1$.
D'un autre côté, considérons le 12. Alors parmi

$$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$$

nous voyons que

- 2 et 12 sont tous deux divisibles par 2 ; • 3 et 12 sont tous deux divisibles par 3 ; • 4 et 12 sont tous deux divisibles par 2 ; • 6 et 12 sont tous deux divisibles par 2 ou 3 ; • 8 et 12 sont tous deux divisibles par 2 ; • 9 et 12 sont tous deux divisibles par 3 ; • 10 et 12 sont tous deux divisibles par 2.

Pour ces raisons, nous excluons les numéros 2, 3, 4, 6, 8, 9 et 10 de notre liste.
Finalement, cela signifie que seuls 1, 5, 7 et 11 sont premiers avec 12, soit quatre entiers seulement, et nous écrivons donc $\varphi(12) = 4$.

Cette stratégie consistant à lister les nombres et à les vérifier un par un est intéressante pour les petits nombres, mais elle serait impossible pour un nombre à 100 chiffres, comme en cryptographie. Cependant, Sage peut calculer $\varphi(n)$ avec une seule commande. Il suffit de taper euler_phi(12) dans Sage.

Machine Translated by Google

et puis vous apprenez que la réponse est 4. On l'appelle « euler phi » car il a été découvert par Leonhard Euler (—), qui a également découvert le nombre

$$e \approx 2{,}718281828 \cdots$$

et de nombreux autres concepts mathématiques. Il détient toujours le record du plus grand nombre d'articles publiés par un mathématicien, même s'il est mort depuis plus de 230 ans.

Un défi amusant : En fait,

si p et q sont des nombres premiers distincts, la valeur de φ(pq) est très facile à calculer. Cependant, je ne vais pas vous révéler la formule. Je pense que ce serait très amusant d'essayer quelques nombres de la forme pq et de voir si vous pouvez découvrir la formule par vous-même ; ce n'est pas trop difficile. Rappelez-vous simplement que des nombres premiers « distincts » impliquent p = q.

4.6.4. Les diviseurs d'un nombre

Considérons l'ensemble des diviseurs d'un nombre. Pour connaître l'ensemble des diviseurs de 250, c'est-à-dire l'ensemble des entiers strictement positifs divisant 250, tapez divisors(250).

ce qui donne la liste suivante :

[1, 2, 5, 10, 25, 50, 125, 250]

De même, si vous souhaitez connaître l'ensemble des diviseurs de 312 500, tapez diviseurs(312500) qui

donne la sortie

[1, 2, 4, 5, 10, 20, 25, 50, 100, 125, 250, 500, 625, 1250, 2500,
3125, 6250, 12500, 15625, 31250, 62500, 78125, 156250, 312500]

Mais n'oubliez pas qu'il est très important de ne pas utiliser de virgule entre les milliers lors de la saisie de grands nombres dans Sage. (Cela signifie que vous ne pouvez pas utiliser de virgule entre le 2 et le 5 dans 312 500.)

Tout comme φ, il existe deux fonctions amusantes de la théorie des nombres qui génèrent des diviseurs. La taille de l'ensemble des diviseurs de n est notée τ(n), tandis que la somme de l'ensemble des diviseurs de n est 1, notée σ(n). Le symbole σ est la lettre grecque « sigma » et le symbole τ est la lettre grecque « tau ». Par exemple, nous venons de voir la liste des diviseurs pour 250. Il y avait 8 diviseurs, donc τ(250) = 8. Par ailleurs,

$$1 + 2 + 5 + 10 + 25 + 50 + 125 + 250 = 468$$

et donc σ(250) = 468. On peut calculer σ dans Sage avec sigma(250). Un point de mémoire pour éviter toute confusion est le suivant :

1Soyez prévenus, certains statisticiens sont offensés par le fait que les théoriciens des nombres ont détourné le symbole σ à cette fin, alors que dans toute la science, l'ingénierie et les statistiques, σ est le symbole de l'écart type.

Machine Translated by Google

sigma est la lettre grecque équivalente à notre « s » et « sum » commence par « s ». Cependant, il n'y a pas de « s » dans « tau » ni dans « count ». De même, « tau » et « count » ont tous deux un « t », mais « sum » et « sigma » n'ont pas de « t ». Gardez à l'esprit que σ(n) additionne les diviseurs, tandis que τ(n) compte les diviseurs.

Parfois, en théorie des nombres, nous voulons calculer les sommes des carrés ou cubes des diviseurs. Considérons que 45 a pour diviseurs

$$\text{diviseurs}(45) = \{1, 3, 5, 9, 15, 45\}$$

nous avons donc un sigma totalisant 78, car

$$1 + 3 + 5 + 9 + 15 + 45 = 78$$

En conséquence, la somme des carrés serait

$$^{2}1 + 32 + 52 + 92 + 152 + 452 = 2366$$

et avec des cubes

$$^{3}1 + 33 + 53 + 93 + 153 + 453 = 95, 382$$

mais dans Sage, les commandes pour cela sont sigma(45, 2) pour les carrés et sigma(45, 3) pour les cubes.

De même, sigma(45, 1) est identique à sigma. Il existe deux façons de calculer τ dans Sage. L'une d'elles est sigma(3600, 0) qui calcule simplement 1 pour chaque diviseur. Autrement dit, chaque diviseur est élevé à la puissance 0 et devient donc 1. Ensuite, en additionnant tous ces 1, on obtient le nombre de diviseurs.

Une autre façon de calculer τ (250) serait de taper

```
len( diviseurs( 250 ) )
```

qui renvoie la longueur de la liste des diviseurs de 250 — c'est exactement ce que τ signifie. Le mot-clé len en Python renvoie la longueur de n'importe quelle liste, quel que soit le type de problème traité.

Paires de nombres amicaux

Une tradition arabe2 de l'époque médiévale est liée à ce sujet. Prenons deux nombres particuliers : 220 et 284. Les diviseurs de 220 sont

$$\text{diviseurs}(220) = \{1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110, 220\}$$

et ensuite ils calculeraient

$$1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284$$

mais d'un autre côté, les diviseurs de 284 sont

$$\text{diviseurs}(284) = \{1, 2, 4, 71, 142, 284\}$$

2Alors que les nombres amicaux apparaissaient dans les écrits des Grecs anciens, le premier mathématicien à écrire des théorèmes à leur sujet fut Al-Sabi Thabit ibn Qurra al Harrani (–), généralement appelé Thebit ou Thebith dans les livres d'Europe occidentale, qui était plus connu pour ses travaux astronomiques et géographiques.

Machine Translated by Google

qui s'élèvent à

$$1 + 2 + 4 + 71 + 142 = 220$$

Nous voyons donc que 220 et 284 sont intimement liés. Le terme technique est que 220 et 284 forment une « paire amicale » d'entiers. Les Arabes trouvaient cela si émouvant qu'ils inscrivaient ces nombres sur les bijoux qu'ils offraient à leurs épouses, symbolisant une relation intime. On les appelle « paires amicales » : 220 représente la jeune femme et 284 le jeune homme.

Notez que nous avons omis n lors de la somme des diviseurs de n dans ce cas. Vous pouvez le trouver dans Sage via

```
sigma( 220 )-220 et

sigma( 284 )-284 Deux
```

autres exemples de paires amicales sont 1184 et 1210, ainsi que 2620 et 2924.

4.6.5. Autre signification de Tau. Comme vous le

savez, certaines fractions comme 1/4, 1/2 et 1/25 peuvent être exprimées comme des fractions décimales terminales. D'autres, comme 1/3 ou 1/9, se répètent. Un théorème intéressant montre qu'une fraction peut être exprimée comme une fraction décimale terminale exacte de n décimales (ou moins) si et seulement si le dénominateur divise 10n . Prenez un moment pour vous en convaincre, ou bien considérez l'exemple suivant. Pour savoir quels dénominateurs donnent une décimale terminale exacte de deux décimales (ou moins), il suffit de voir quels entiers positifs divisent 100. Ce sont les dénominateurs autorisés. Dans Sage, nous saisirions

```
diviseurs(100)
```

et puis obtenir

```
[1, 2, 4, 5, 10, 20, 25, 50, 100]
```

qui sont les seuls dénominateurs qui sont donnés exactement sous forme de décimale terminale avec seulement une ou deux décimales.

En binaire, la règle analogue serait 2n . Il est évident que le seul . En gen-
les entiers positifs qui divisent 2n sont les nombres 1, 2, 4, 8, . . . , 2 eral, pour les nombres ^{n}
« base b », les dénominateurs admissibles pour les fractions utilisant deux chiffres ou moins après la virgule seraient les diviseurs de b pour le binaire, 22 = 4, et les dénominateurs sont [1, 2, 4]. 2. Donc

Comme vous pouvez le constater, nous pouvons utiliser ce critère pour identifier les bases numériques adaptées aux fractions et celles qui ne le sont pas. Nous pouvons nous poser en juges et comparer les systèmes binaire, décimal, maya (base 20) et babylonien (base 60).

Considérons d’abord le système de numération babylonien qui est en base 60.
En utilisant les diviseurs (3600), nous apprenons que deux symboles (ou moins) suffiraient à décrire n'importe quelle fraction avec les dénominateurs suivants :

Machine Translated by Google

[1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, 30, 36, 40, 45, 48, 50, 60, 72, 75, 80, 90, 100, 120, 144, 150, 180, 200, 225, 240, 300, 360, 400, 450, 600, 720, 900, 1200, 1800, 3600]

Pour le système maya (base 20), nous avons une liste plus courte, obtenue avec des diviseurs (400).

[1, 2, 4, 5, 8, 10, 16, 20, 25, 40, 50, 80, 100, 200, 400]

Comme vous pouvez le voir, pour chaque base, nous savons maintenant combien de dénominateurs admissibles existent.

• En base 2, il y a 3 dénominateurs. (Si l'on exclut 1, il n'y en a que 2.) • En base 10, il y a 9 dénominateurs. (Si l'on exclut 1, il n'y en a que 8.) • En base 20, il y a 15 dénominateurs. (Si l'on exclut 1, il y en a 14.) • En base 60, il y a 45 dénominateurs. (Si l'on exclut 1, il y en a 44.)

Remarque : Ce que nous faisons en réalité, c'est calculer $\tau(n^2)$.

Comme vous pouvez le voir, le système babylonien est le plus pratique pour la division simple des produits, du bétail et de l'argent pendant le commerce. Non seulement il existe 44 façons d'écrire des fractions, exactement, en utilisant deux symboles après la place des unités, mais la liste commence par

{1, 2, 3, 4, 5, 6, 8, 9, 10, 12, . . .} soit

tous les dénominateurs possibles jusqu'à 12 inclus, à l'exception de 7 et 11. Il n'est peut-être pas surprenant que nous utilisions la base 60 pour les angles en géométrie et en trigonométrie, ainsi que pour diviser les heures en minutes et en secondes.

4.6.6. Arithmétique modulaire

Pour savoir quel est le mod 37 de 1939, vous pouvez taper

```
1939 % 37
```

mais l'opérateur % prend uniquement la réduction modulaire du nombre le plus proche, sauf si des parenthèses sont utilisées. Cela signifie que pour évaluer

(32)4 + 2 divisé par 5

vous ne devez pas taper

```
3^2*4 + 2 % 5 qui
```

donne 38, mais plutôt

```
(3^2*4 + 2) % 5
```

ce qui donne 3, la bonne réponse.

L'opposé de l'opérateur « mod » est le quotient entier. Pour trouver le quotient entier d'un nombre, utilisez 25 // 4

Pour obtenir 6, il s'agit du quotient 25/4 = 6,25 arrondi à l'entier inférieur. Plus précisément, x//y = z signifie que z est le plus grand entier tel que zy ≤ x. Puisque (7)(4) = 28, on voit que (6)(4) = 24 est la plus grande façon possible d'avoir (z)(4) ≤ 25, z étant un entier.

Machine Translated by Google

4.6.7. Lectures complémentaires en théorie des nombres

Pour le lecteur qui souhaite explorer davantage la théorie des nombres, j'ai deux Livres à recommander. Le premier est un peu plus facile : Introduction aux nombres. Théorie avec cryptographie, par James S. Kraft et Lawrence C. Washington, publié en 2013 par Chapman & Hall.

Le deuxième est davantage axé sur Sage et a été écrit par le fondateur du professeur William Stein, professeur à Sage. Le titre est : Théorie élémentaire des nombres. Nombres premiers, congruences et secrets : une approche computationnelle, publié par Springer dans le cadre de la série « Undergraduate Texts in Mathematics » en 2008.

4.7. Quelques commandements mineurs de Sage

Voici quelques commandes qui reviennent de temps à autre. Elles peuvent être très utiles dans les situations qui les nécessitent.

min(x,y)	Renvoie x ou y, selon la valeur la plus petite.
max(x,y)	Renvoie x ou y, selon la valeur la plus élevée.
plancher(x)	Arrondissez simplement x vers le bas, ou x
plafond(x)	Il suffit d'arrondir x vers le haut, ou x
factorielle(n)	Il s'agit de n!, également appelé « factorielle n ».
binôme(x,m)	Ceci est écrit $\binom{x}{m}$ ou $\frac{x!}{m!(x-m)!}$ ou xCm
binôme(x,m) * m!	Ceci est généralement écrit $\frac{x!}{m!}$ ou xPm

4.7.1. Arrondis, sols et plafonds

Comme vous pouvez le voir, la commande floor arrondit un nombre vers le bas, et la commande ceil la commande arrondit un nombre vers le haut.

Dans les livres plus anciens, la fonction « étage » est parfois appelée « Le plus grand Fonction entière. » Ceci est dû au fait qu'elle peut être définie par « le plus grand entier qui est inférieur ou égal à x. » Certains anciens livres de mathématiques écriront [x] au lieu de x. Je suppose que si vous le vouliez, vous pourriez définir la fonction plafond par « le plus petit entier supérieur ou égal à x ».

4.7.2. Combinaisons et permutations

Dans de nombreux manuels, notamment pour les mathématiques finies ou quantitatives L'alphabétisation, ainsi que les probabilités et les statistiques de base, nécessitent beaucoup de temps Parlons de combinaisons et de permutations. Par exemple, si j'ai 52 cartes distinctes dans un jeu de cartes, je pourrais demander combien de mains de 5 cartes je peux Construire avec eux.

Si l'ordre n'a pas d'importance, comme au poker, alors j'aurais

$$52C5 = \binom{52}{5} = \frac{52!}{5!\,47!} = \frac{52 \times 51 \times 50 \times 49 \times 48}{5 \times 4 \times 3 \times 2 \times 1} = 2\,598\,960$$

Machine Translated by Google

possibilités, et Sage appelle cela binomial(52, 5). Voici un exemple de la formule des « combinaisons ».

Cependant, si l'ordre compte, ce qui n'est pas le cas au poker, alors je aurait

$$52P5 = \binom{52}{5} 5! = \frac{52!}{47!} = 52 \times 51 \times 50 \times 49 \times 48 = 311,875,200$$

possibilités. Dans Sage, vous pouvez saisir

```
factorielle(52)/factorielle(52-5)
```

pour cela, ou vous pouvez taper

```
binomial(52,5)*factorial(5)
```

Les deux renvoient la même réponse. Voici un exemple de la formule des « permu-tations ».

Vous vous demandez peut-être pourquoi Sage ne propose pas de commande pour les permutations, alors qu'il en propose une pour les combinaisons. Il existe bel et bien une commande pour les permutations, mais elle offre des fonctionnalités un peu plus avancées que celles décrites ci-dessus. Dans le travail ci-dessus, nous avons calculé le nombre de permutations, mais parfois vous voudrez peut-être savoir quelles sont ces permutations.

Pour cela, vous pouvez taper

```
S= Permutations(['ace', 'king', 'queen', 'jack']) print "Nous attendons ", len( S.list() ), "
éléments ci-dessous."
S.list()
```

et il affichera docilement les 24 possibilités : Nous attendons 24 éléments ci-dessous.

```
[['as', 'roi', 'dame', 'valet'], ['as', 'roi', 'valet', 'dame'],
 ['as', 'dame', 'roi', 'valet'], ['as', 'dame', 'valet', 'roi'],
 ['as', 'valet', 'roi', 'dame'], ['as', 'valet', 'dame', 'roi'],
 ['as', 'valet', 'dame', 'roi'], ['roi', 'as', 'valet', 'dame'],
 ['roi', 'as', 'valet'], ['roi', 'dame', 'valet', 'as'], ['roi',
 'valet', 'as', 'dame'], ['roi', 'valet', 'dame', 'as'],
 ['dame', 'as', 'roi', 'valet'], ['dame', 'as', 'valet', 'roi'],
 ['dame', 'roi', 'as', 'valet'], ['dame', 'roi', 'valet', 'as'],
 ['dame', 'valet', 'as', 'roi'], ['dame', 'valet', 'roi', 'as'],
 ['valet', 'as', 'roi', 'dame'],
```

Machine Translated by Google

Paroles de mathématicien	Fonctions de Sage
Sinus hyperbolique	né(x)
cosinus hyperbolique	matraque(x)
Tangente hyperbolique	tanh(x)
cotangente hyperbolique	nourriture(x)
Sécante hyperbolique	lui-même/elle-même
cosécante hyperbolique	csch(x)
Sinus hyperbolique inverse	arcsinh(x) ou asinh(x)
cosinus hyperbolique inverse	arccosh(x) ou acosh(x)
Tangente hyperbolique inverse	arctanh(x) ou atanh(x)
Cotangente hyperbolique inverse	arccoth(x) ou acoth(x)
Sécante hyperbolique inverse	arcsearch(x) ou search(x)
cosécante hyperbolique inverse	arccsch(x) ou acsch(x)

Tableau 2. Les fonctions trigonométriques hyperboliques et
Leurs Inverses

```
['valet', 'as', 'reine', 'roi'],
['valet', 'roi', 'as', 'reine'],
['valet', 'roi', 'reine', 'as'],
['valet', 'reine', 'as', 'roi'],
['valet', 'reine', 'roi', 'as']]
```

Essayez plutôt

```
S= Permutations( [ 'as', 'roi', 'reine', 'valet', 'dix', 'neuf' ] )
imprimer "Nous nous attendons à ", len( S.list() ), "  éléments ci-dessous."
S.list()
```

si vous êtes curieux.

En fait, c’est l’une des nombreuses utilisations de l’infrastructure « Permutations » dans Sage, mais nous ne pouvons pas aborder les autres ici, car il s'agit d'un sujet très avancé.

4.7.3. Les fonctions trigonométriques hyperboliques

Si vous avez appris les fonctions trigonométriques hyperboliques, c'est parfait. Personnellement, on ne me les a jamais enseignés dans un cours (et j'ai un (doctorat), mais je sais qu'ils sont parfois utiles.
en physique, principalement dans l'étude de la courbe caténaire, qui est la forme d'un câble autorisé à pendre sous son propre poids, mais attaché à chaque extrémité. Certaines intégrales sont également beaucoup plus compactes. dans leur solution si vous connaissez les fonctions hyperboliques. Alors, ne vous sentez pas C'est dommage si vous ne les avez jamais vus. Voici les commandes pour chacun de ces 12 fonctions, et elles sont données dans le tableau 2.

Machine Translated by Google

4.8. Calcul des limites expressément

Si vous vouliez calculer

$$\lim_{x \to 0} \frac{x-2}{x-3}$$

tu devrais taper

```
limite( (x-2)/(x-3), x=0)
```

d'apprendre que la réponse est 2/3, ce qui est peut-être évident. La limite serait peut-être moins évidente.

$$\operatorname*{colle}_{x \to 1} \frac{2x - 4x + 3}{x^2 - 3x + 2}$$

pour lequel vous devez taper

```
limite( (x^2-4*x+3)/(x^2-3*x+2), x=1)
```

pour obtenir la réponse « 2 ». Si vous ne savez pas comment faire cela sans ordinateur ni calculatrice (autrement dit, uniquement avec un stylo et un crayon), alors permettez-moi de vous suggérer d'essayer de factoriser ces deux polynômes dans un premier temps.

Enfin et surtout, si vous vouliez vérifier le fait étrange et étonnant

$$\lim_{x \to 0} (\cos x)^{1/x2} = \frac{1}{\sqrt{et}}$$

alors vous devriez taper

```
limite( cos(x)^(1/x^2), x=0)
```

pour obtenir la réponse souhaitée (mais très surprenante)

```
et^(-1/2)
```

Pour montrer que la dernière limite est donnée en termes de cosinus et de x^2 c'est vrai, vous pouvez essayer des valeurs comme x = 0,001 sur n'importe quelle calculatrice scientifique ou avec Sage. Mais prouver que la limite est vraie est plutôt difficile. Cela nécessite deux applications de la règle de L'Hôpital ou une série de Taylor (mais pas les deux), pour autant que je sache.

Limites infinies :

Parfois, les limites vont jusqu'à l'infini. Par exemple,

$$\operatorname*{colle}_{x \to 0+} \frac{1}{x^2} = \lim_{x \to 0-} \frac{1}{x^2} = \operatorname*{colle}_{x \to 0} \frac{1}{x^2} = \infty$$

C'est l'exemple classique dans la plupart des manuels de calcul. Cependant, si nous devions remplacer 1/x2 par 1/x3 dans ces limites, nous aurions un problème.

Si l'on approche de 0 à partir des nombres positifs, la limite pour 1/x3 serait +∞, mais si l'on approche de 0 à partir des nombres négatifs, la limite pour 1/x3 serait −∞. Par conséquent, la limite n'existe pas si l'on approche de 0 en général.

Pour voir comment Sage gère cela, considérez le code suivant :

Machine Translated by Google


```
limite d'impression( 1/x^2, x=0)
limite d'impression( 1/x^3, x=0)
limite d'impression( 1/x^4, x=0)
limite d'impression( -1/x^4, x=0)
```

Ce code génère cette sortie :

```
+Infini Infini
+Infini
-Infini
```

Comme

vous pouvez le voir, Sage rapportera fidèlement +Infini lorsque la limite passe à +∞, et il rapportera fidèlement -Infini lorsque la limite passe à −∞. L'infini non signé est un peu surprenant. Nous pouvons clarifier davantage en tapant print limit(1/

```
x^3, x=0, dir='right') print limit( 1/x^3, x=0,
dir='left') print limit( 1/x^3, x=0 )
```

Ce code produit la sortie suivante :

```
+Infini
-Infini
Infini
```

Essentiellement, Sage signalera Infinity comme limite pour f(x) chaque fois

$$\lim_{x \to 0} |f(x)| = +\infty$$

à condition que les conditions de déclaration de +Infini ou -Infini ne soient pas également remplies. Cependant, cette convention pourrait contrarier de nombreux professeurs de calcul. Traditionnellement, la limite lorsque 1/x3 tend vers zéro devrait être « dne », abréviation de « n'existe pas ».

À propos, dir = 'droite' peut aussi être décrit par dir = '+' et dir = 'plus'. De même, dir = 'gauche' peut aussi être décrit par dir = '-' et dir = 'moins'. Chaque système de nommage est plus ou moins intuitif. Cependant, il est très important de se rappeler que la limite approchant du côté négatif et la limite approchant du côté positif ne sont (parfois) pas égales. C'est un piège fréquent en calcul différentiel et intégral.

examens.

4.9. Nuages de points dans Sage

Supposons que vous examiniez certaines données fournies par les points

$$\{(0, 7,1); (1, 5,2); (2, 2,9); (3, 1,05); (4, -0,9)\}$$

Tout d'abord, vous devez informer Sage des données à l'aide de la commande suivante, qui est un autre exemple de la façon dont les listes sont écrites dans

Sage : datapoints = [(0, 7.1), (1, 5.2), (2, 2.9), (3, 1.05), (4, -0.9)]

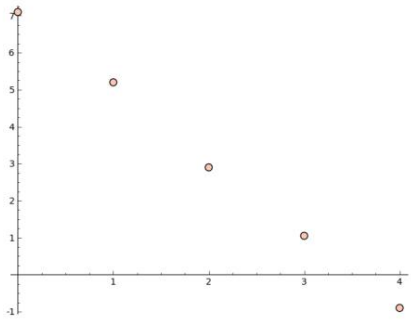
Machine Translated by Google

Remarquez comment les données, à savoir ces cinq points, sont séparées par des virgules et entourées de crochets. Voici un exemple de liste dans Sage. Vous pouvez entourer n'importe quelle donnée de [et], et séparer les entrées par des virgules, pour créer une liste. Sage a hérité cette notation du langage informatique Python.

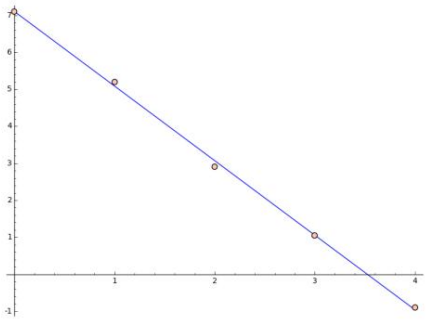
Nous pouvons maintenant facilement créer un nuage de points, en tapant simplement

```
scatter_plot(points de données)
```

Cela produira le nuage de points illustré ci-dessous à gauche. Cependant, il est important que la liste des points de données reste affichée à l'écran dans le serveur Sage Cell. Ainsi, tout au long de cette section et de la suivante, cette liste devra figurer dans le champ où vous saisissez habituellement les commandes.



Sans ligne d'ajustement optimal



Avec la ligne la mieux adaptée

Vous avez peut-être l'habitude de calculer les courbes d'ajustement optimal dans un tableur ou un logiciel statistique. Si c'est le cas, c'est parfait. Sinon, Sage pourra parfaitement le faire pour vous. Nous vous expliquerons comment procéder dans la section suivante. La courbe d'ajustement optimal pour cet exemple est y = 7,1−2,015x, et le nuage de points de droite inclut cette courbe. Comme vous pouvez le constater, l'ajustement est plutôt bon, mais imparfait. Ce graphique, avec la courbe incluse, a été créé en saisissant

```
scatter_plot(points de données) + plot(7.1-2.015*x, 0, 4)
```

Au fait, si vous vous demandez comment Sage a développé les versions 7.1 et 2.015, nous en parlerons à la page 4.16.4. Prenons maintenant un autre ensemble de données. Cet exemple vous expliquera pourquoi nous nous donnons la peine de créer un graphique, alors qu'il existe de nombreux outils, dont Sage, pour nous fournir la droite d'ajustement optimale à chaque fois que nous en avons besoin. Tout d'abord, saisissez les données.

```
autresdonnées = [ (-3, 10), (-2, 5), (-1, 2), (0, 1), (1, 2),
                  (2, 5), (3, 10) ]
```

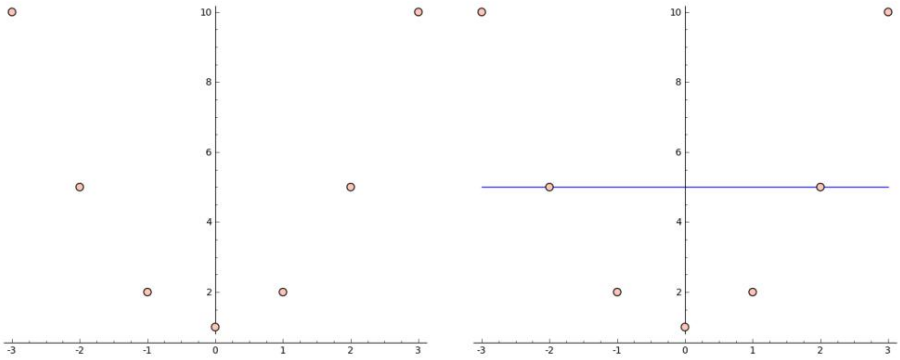
Notez que le copier-coller fonctionne généralement bien si vous ne souhaitez pas retaper la liste ci-dessus. Imaginez qu'il s'agisse de données issues d'une expérience scientifique, que votre partenaire de laboratoire les ait copiées et ait trouvé la droite d'ajustement optimale, y = 0x + 5, à l'aide d'un logiciel statistique.

Machine Translated by Google

n'a pas créé de nuage de points. Vous êtes, bien sûr, attentif aux bonnes pratiques scientifiques et vous saisissez donc

```
scatter_plot(otherdata)
```

pour obtenir le graphique ci-dessous à gauche. Ensuite, ajoutez la droite d'ajustement optimal, et vous obtenez le graphique de droite de la figure 4.9.



Sans ligne d'ajustement optimal

Avec la ligne la mieux adaptée

Comme vous pouvez le constater, c'est absurde. Les données sont clairement une parabole ordinaire. Par conséquent, une droite est une très mauvaise approximation pour ces données. Cependant, parmi toutes les droites possibles, même si elles présentent toutes un taux d'erreur inacceptable, une droite présente le moins d'erreur. Il s'avère que cette droite de moindre erreur (carrée), pour cet ensemble de données, est y = 0x + 5. De plus, il est manifeste que cette droite est mal ajustée, comme le montre le graphique de droite. C'est pourquoi tracer systématiquement le nuage de points avec la droite la mieux ajustée est considéré comme une « procédure opératoire standard » dans la plupart des laboratoires. D'ailleurs, le graphique précédent a été généré par la commande :

```
scatter_plot(autresdonnées) + plot( 0*x+5, -3, 3)
```

C'est le moment d'expliquer que ce que les scientifiques appellent « la droite de meilleur ajustement » est en réalité plus connu des mathématiciens sous le nom de « régression linéaire ». Par régression linéaire, nous entendons l'analyse d'un ensemble de données pour trouver la droite qui s'ajuste le mieux aux données. Ici, nous aurions dû effectuer une « régression quadratique » pour trouver la parabole la plus adaptée aux données. Dans quelques pages, nous verrons comment procéder.

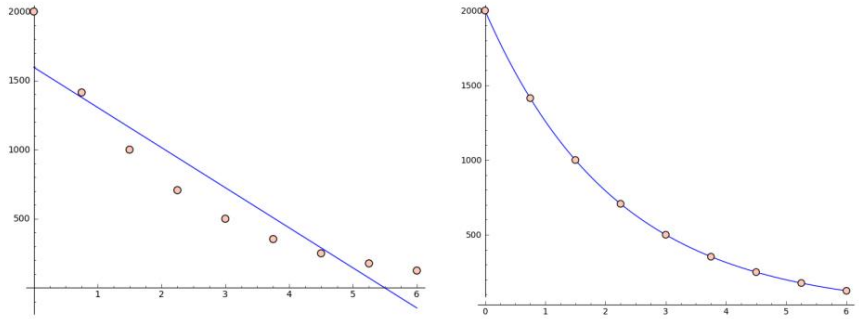
Exemple de chimie : Vous

pourriez penser que l'exemple de la parabole est assez artificiel. Vous avez raison, je l'ai inventé juste pour prouver quelque chose. Cependant, les données suivantes montrent comment un échantillon de 2 kilogrammes de cobalt 62 pourrait se désintégrer. Ici, l'abscisse correspond au temps en minutes et l'ordonnée au poids en grammes.

```
cobalt_data=[ (0, 2000), (0,75, 1414), (1,5, 1000), (2,25, 707), (3,00, 500), (3,75, 353),
        (4,50, 250), (5,25, 177), (6,00, 125)]
```

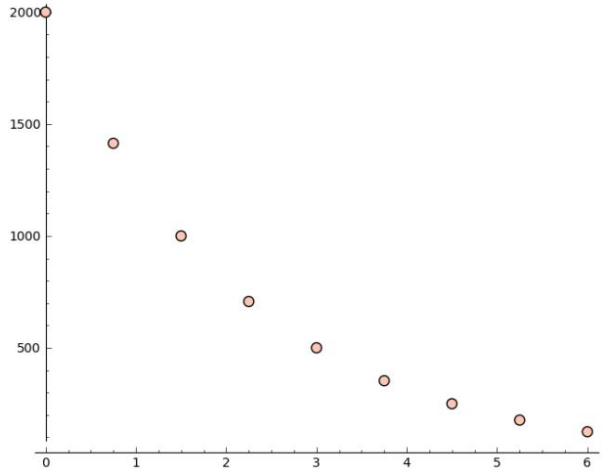
Nous pouvons taper scatter_plot(cobalt_data) et obtenir le nuage de points :

Machine Translated by Google



Régression linéaire Régression exponentielle

Figure 2. Deux tracés de l'ensemble de données Cobalt



Un étudiant imprudent pourrait être tenté d'effectuer une régression linéaire, surtout si les derniers rapports de laboratoire l'exigeaient. La régression linéaire de ces données est y = −290,333x + 1596,11. Dans ce cas, on obtient le tracé de la figure 4.9 à gauche. Cependant, un étudiant plus avisé devrait se rappeler que la désintégration radioactive suit une courbe exponentielle. Au lieu de demander y = ax + b, qui est une droite, l'étudiant avisé demanderait y = aebx , qui est une exponentielle. C'est là tout l'intérêt des outils de régression de Sage : une , régression linéaire est aussi simple qu'une régression quadratique, exponentielle, cubique ou logarithmique.

Nous verrons comment réaliser ces régressions dans la section suivante. La régression exponentielle est égale à y = 1999,96e, ce qui donne la courbe ci-dessous à droite.

La morale de l'histoire est que, même s'il est intéressant de calculer des régressions de données, il ne faut pas présumer qu'une régression linéaire est la solution recherchée. Diviser le monde des fonctions en fonctions linéaires et non linéaires revient à diviser l'ensemble des êtres vivants sur Terre en tatous et

Machine Translated by Google

non-tatous. La plupart des fonctions ne sont pas linéaires ; il ne faut donc pas présumer qu'un modèle linéaire est le meilleur choix pour vos données.

4.10. Créer vos propres régressions dans Sage

Cette leçon suppose que vous avez lu la section précédente sur les nuages de points. Nous utiliserons notamment les données Cobalt de la leçon précédente. Pour les pages suivantes, ces données seront utilisées comme première ligne de votre fenêtre Sage Cell. Si vous utilisez Sage Cloud, ces données doivent simplement figurer sur la page et au-dessus des commandes saisies ici.

Nous allons d'abord essayer d'appliquer une régression linéaire à ces données. Cela signifie que nous allons demander à Sage de nous fournir une fonction y = ax + b qui modélise au mieux ces 9 points. Les commandes pour cela sont var("a b") model(x) =

a*x + b

find_fit(cobalt_data, model)

où la commande var est déjà utilisée

(voir page 38), mais les deux suivantes sont nouvelles pour nous. La réponse obtenue est :

[a == -290,33333269040816, b == 1596,1111098318909]

c'est la façon dont Sage nous dit que la régression linéaire est

$$y = -290{,}333x + 1596{,}11$$

Nous pouvons maintenant tracer

cela avec scatter_plot(cobalt_data) + plot(-290.333*x+1596.11, 0,6)

Regardons le tracé, qui est à gauche de la paire d'images de la figure 4.9. Comme vous pouvez le constater, ce n'est pas du tout une bonne adéquation. Bien sûr, la chimie de base nous apprend que la désintégration radioactive suit une courbe exponentielle. Pour effectuer une régression

exponentielle,

nous saisissons var("a b")

model(x) = a*exp(b*x) find_fit(cobalt_data,

model) et, comme vous pouvez le constater, seule la ligne médiane de ces trois lignes a changé. Nous avons modifié le modèle de y = ax + b à y = aebx. La réponse de Sage à ces commandes est désormais la suivante :

[a == 1999,9607020042693, b == -0,46216218069391302]

c'est la façon dont Sage nous dit que le modèle est

$$1999{,}96e^{-0{,}462162x} \quad y =$$

et vous pouvez immédiatement constater une erreur d'arrondi. Ce coefficient directeur de 1 999,96 devrait en réalité être de 2 000, puisque nous avons commencé avec 2 000 grammes. Cependant, cela fait partie des avantages et des inconvénients des méthodes numériques. On peut faire presque tout avec, mais on est rarement précis.

En tout cas, on peut constater la qualité de cette régression par la commande

Machine Translated by Google


```
scatter_plot(cobalt_data) + plot(1999,96*exp(-0,462162*x), 0, 6)
```

Le tracé obtenu est l'image de droite de la figure 4.9. Enfin, on pourrait envisager une régression quadratique. Il s'agit d'une régression de la forme $y = ax^2 + bx + c$, dont la commande serait :

```
var("ab c")
modèle(x) = a*x^2 + b*x + c
find_fit(cobalt_data, modèle)
```

Vous pouvez constater que le seul changement réel se situe au milieu des trois lignes, mais nous avons également dû déclarer c comme variable, ce que nous avons fait dans la commande var. Quoi qu'il en soit, la réponse de Sage est la suivante :

```
[a == 62,755170737557854, b == -666,86435772164759,
                c == 1925.5757574133333]
```

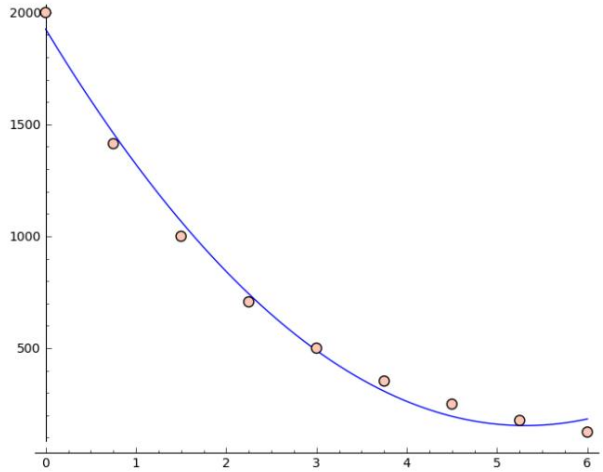
et c'est la façon dont Sage nous dit que le modèle est

$$y = 62,7551x^2 - 666,864x + 1925,57$$

Ensuite, nous pouvons dessiner ce modèle avec

```
nuage de points (données cobalt) + tracé (62,7551*x^2-666,864*x+1925,57,0,6)
```

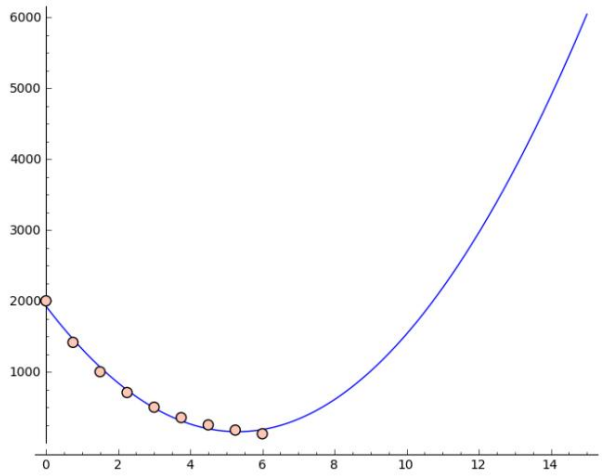
qui produit l'image



Bien que ce ne soit clairement pas une mauvaise adéquation, il est imprudent de modéliser la désintégration radioactive avec une fonction quadratique. Ce n'est pas seulement dû au fait que de nombreux points de données ont été « manqués ». Le problème est plus facile à comprendre en définissant une plage de temps de 0 à 15, au lieu de 0 à 6. Il suffit de remplacer le 6 par 15 à la fin de la dernière ligne saisie.

Nous obtenons l'image suivante

Machine Translated by Google



ce qui n'est clairement pas un bon modèle de désintégration radioactive.

4.11. Calcul en octal, binaire et hexadécimal ?

Pour connaître les valeurs décimales de l'hexadécimal 71, tapez 0x71 et obtenez 113. Notez que cela est dû au fait que 7 × 16 + 1 = 113. Au fait, il s'agit d'un zéro avant le x, et non d'un « o ». Alternativement, pour le nombre hexadécimal 0x1F, vous pouvez taper 0x1F et obtenir 31, car 1 × 16 + 15 = 31.

Le système octal est beaucoup plus rare, mais il apparaît de temps à autre. Pour l'octal 11, tapez 011. La réponse est 9, car 1×8+1 = 9.

Le binaire est similaire à l'hexadécimal, sauf que le x est remplacé par ab. Ainsi, pour trouver le nombre binaire 1101, il faut taper 0b1101 et apprendre que la réponse est 13. C'est parce que

$$(1 \times 2^{3}) + (1 \times 2^{2}) + (0 \times 2^{1}) + (1 \times 2^{0}) = 8 + 4 + 0 + 1 = 13$$

Pour inverser la fonction, tapez hex(31) et vous obtiendrez 1F. De même, pour le binaire, tapez bin(63) et vous obtiendrez le développement binaire de 63, soit 11111 (soit cinq unités consécutives). L'octal inversé est un peu complexe. Tapez oct(int(31)) pour savoir que la réponse est 37, car

$$37 \text{ octaux} = 3 \times 8 + 7 = 24 + 7 = 31$$

4.12. Sage peut-il faire du Sudoku ?

Pour résoudre le puzzle Sudoku 9×9 défini par une matrice 9×9, entrez la matrice :

```
A = matrice( 9, 9, [5,0,0, 0,8,0, 0,4,9, 0,0,0, 5,0,0, 0,3,0, 0,6,7, 3,0,0, 0,0,1, 1,5,0, 0,0,0,
    0,0,0, 0,0,0, 2,0,8, 0,0,0, 0,0,0, 0,0,0, 0,1,8, 7,0,0, 0,0,4, 1,5,0, 0,3,0, 0,0,2,
    0,0,0, 4,9,0, 0,5,0, 0,0,3])
```

Ensuite, si vous tapez print A sur une ligne seule, vous obtenez

Machine Translated by Google


```
[5 0 0 0 8 0 0 4 9] [0 0 0 5
0 0 0 3 0] [0 6 7 3 0 0 0 0
1] [1 5 0 0 0 0 0 0 0] [0 0 0
2 0 8 0 0 0] [0 0 0 0 0 0 0
1 8] [7 0 0 0 0 4 1 5 0] [0 3
0 0 0 2 0 0 0] [4 9 0 0 5 0
0 0 3]
```

Ce jeu comporte des entrées non nulles, mais contient principalement des zéros pour marquer les espaces vides dans un Sudoku classique. Vous pouvez maintenant résoudre le jeu en tapant

```
sudoku(A).
```

et obtenir

```
[5 1 3 6 8 7 2 4 9] [8 4 9 5
2 1 6 3 7] [2 6 7 3 4 9 5 8
1] [1 5 8 4 6 3 9 7 2] [9 7 4
2 1 8 3 6 5] [3 2 6 7 9 5 4
1 8] [7 8 2 9 3 4 1 5 6] [6 3
5 1 7 2 8 9 4] [4 9 1 8 5 6
7 2 3]
```

C'est une solution valable au Sudoku. Cependant, il existe peut-être d'autres solutions.

4.13. Mesurer la vitesse de Sage

La factorisation de grands nombres, par exemple, est censée être difficile et vous voudrez peut-être savoir combien de temps il faut à Sage pour exécuter la commande.

```
facteur(2250000063000000041)
```

Dans ce cas, vous devez taper

```
timeit("factor(2250000063000000041)")
```

et pour moi, sur ma machine, il est indiqué

625 boucles, le meilleur des 3 : 91,8 µs par boucle

ce qui signifie qu'il a essayé 625 fois, et la meilleure estimation du temps d'exécution est de 91,8 microsecondes, soit environ 1/10 893 secondes.

Bien sûr, le temps de réponse avec Sage est bien plus long. Que se passe-t-il ? La plupart du temps passé avec Sage est gaspillé par les commandes transitant par Internet vers le serveur, puis retournant au serveur.

De plus, l'affichage des données représente une perte de temps. Dans ce cas, vous perdez probablement environ 2 secondes en temps de transit, 1/4 de seconde sur l'affichage et l'interface, et presque 0 seconde en calcul.

Machine Translated by Google

Cependant, pour certains problèmes très complexes, vous pouvez consacrer 10 minutes aux calculs, 2 secondes au temps de transit et un quart de seconde à l'affichage et à l'interface. Cette fonctionnalité de Sage est là pour vous aider à mesurer cela.

4.14. Nombres énormes et Sage

Parfois, lorsqu'on étudie la combinatoire, on est confronté à des nombres énormes. Par exemple, demander à Sage la factorielle (52) donnera la réponse suivante :

80658175170943878571660636856403766975289505440883277824000000000000

mais j'ai du mal à saisir ce que cela signifie. Cependant, je peux taper

floor(log(factorial(52), 10))+1 ce qui me donnera 68. Cela

signifie que le nombre est un nombre à 68 chiffres, pour des raisons que nous allons maintenant expliquer.

Pourquoi cela fonctionne-t-il ? Parce que log10 107 = 7 et log10 108 = 8. Par conséquent, tout nombre x tel que 107 ≤ x < 108 a donc 7 ≤ log10 x < 8.
Mais quels nombres ont 107 ≤ x < 108 ? Précisément les nombres à huit chiffres, bien sûr ! Par conséquent, l'ensemble des nombres à huit chiffres est précisément l'ensemble des nombres avec 7 ≤ log10 x < 8. En prenant le plancher avec floor, on arrondit à l'inférieur, et ainsi tout nombre à huit chiffres a le plancher de son logarithme décimal qui est précisément 7. On corrige cela avec un +1.

Taper factorial(factorial(7)), pour calculer (7!)! est également amusant.
Vous obtenez un nombre très grand si vous procédez ainsi. En fait, grâce à la technique ci-dessus, j'ai pu apprendre que (7!)! comporte 16 474 chiffres. Vous constaterez également que ce nombre très grand est affiché sur des dizaines de lignes, chacune terminée par une barre oblique inverse. Les barres obliques inverses sont une façon pour Sage de signifier qu'il considère plusieurs lignes comme une seule grande ligne, mais que cette très grande ligne de chiffres ne tiendrait pas à l'écran. Par conséquent, Sage doit le casser pour l'adapter à votre écran.

Il est également amusant

de taper factor(factorial(factorial(7)))

Je suis impressionné par la rapidité avec laquelle Sage peut factoriser ce nombre à 16 474 chiffres. Et observez que chaque nombre premier inférieur à 5040 est présent dans la factorisation, bien qu'élevé à différentes puissances.

Les fractions peuvent être encore plus intéressantes. Par exemple, si vous tapez :

7200000*(21/20)^(2011-1867) vous récupérez

2258504452713665066998727716366996290277307841436194047141354198722979 91\
1598907301223254758487131990979658081272665237056684172677363631651004 50\
8440891079595624253782831563238566170905056995 29/27875931498163278926919\ 647840810451882475520 ...

Notez la présence d'une barre oblique aux deux tiers environ de la ligne médiane. Ce nombre de cinq lignes est donc en réalité un entier de 2,67 lignes divisé par deux.

Machine Translated by Google

par un entier de 2,33 lignes. Plus précisément, il s'agit d'un nombre à 192 chiffres divisé par un nombre à 182 chiffres. J'ai découvert ces nombres en utilisant la technique du plancher du logarithme en base 10, que nous avons apprise il y a quelques instants.

Dans tous les cas, en tapant

```
N( 7200000*(21/20)^(2011-1867) )
```

donne-moi l'approximation décimale

```
8.10198738242156e9
```

et cela représente environ 8,1 milliards, ce que je peux comprendre, même si ce n'est que partiellement.

Si vous vous demandez d'où cela vient, regardez les dates et vous comprendrez rapidement de quoi il s'agit (si vous avez étudié l'histoire américaine). En 1867, les États-Unis ont acheté l'Alaska à la Russie pour 7,2 millions de dollars. Si cette somme avait été investie à un taux annuel composé de 5 % pendant la même période, elle aurait valu 8,1 milliards de dollars en 2011, lorsque j'ai commencé ce livre. (Notez que 21/20 = 1,05.) Bien sûr, c'est un taux d'intérêt extrêmement modéré, et il illustre la puissance des intérêts composés.

4.15. Utilisation de Sage et LATEX

Peut-être que je veux résoudre l'équation quartique suivante

$$x^4 + x^3 + x^2 + x - 4 = 0$$

et je tape le code suivant : answers =

```
solve(x^4 + x^3 + x^2 + x - 4, x)

imprimer les réponses[0]
imprimer les réponses[1]
imprimer les réponses[2]
imprimer les réponses[3]
```

Alors je pourrais voir que answers[2] est la racine qui m'intéresse. Cependant, comment vais-je composer efficacement x ==

```
(5/9*sqrt(3)*sqrt(2) - 35/27)^(1/3) - 5/9/(5/9*sqrt(3)*sqrt(2) -
        35/27)^(1/3) - 2/3
```

en LATEX pour une communication de conférence ou un article de revue ? Heureusement, Sage le fera. Faites-le pour moi. Le code suivant : réponses

```
= résoudre(x^4 + x^3 + x^2 + x - 4, x)

imprimer les réponses[2]

latex(réponses[2])
```

produit la sortie suivante

Machine Translated by Google


```
x == (5/9*carré(3)*carré(2) - 35/27)^(1/3) - 5/9/(5/9*carré(3)*carré(2) - 35/27)^(1/3) - 2/3
```

```
x={\gauche(\frac{5}{9} \, \sqrt{3} \sqrt{2} - \frac{35}{27}\droite)}^{ \frac{1}{3}} - \frac{5}{9 \,
        {\gauche(\frac{5}{9} \, \sqrt{3} \sqrt{2} - \frac{35}{27}\droite)}^{\frac{1}{3}}}-\frac{2}{3}
```

Comme vous pouvez le constater, la première ligne est écrite en Sage et la seconde en LATEX. Le résultat suivant est très beau, mais encore imparfait.

$$x = \frac{5}{9}\sqrt{3}\sqrt{2} - \frac{35}{27}\Big)^{\frac{1}{3}} - \frac{5}{9\left(\frac{5}{9}\sqrt{3}\sqrt{2} - \frac{35}{27}\right)^{\frac{1}{3}}} - \frac{2}{3}$$

4.16. Matrices dans Sage, troisième partie

De grandes parties de Sage utilisent l'algèbre linéaire pour effectuer leurs tâches. Il n'est donc pas surprenant que Sage dispose d'une suite complète de commandes dédiées à l'algèbre linéaire. Il y aurait trop de choses à aborder ici, mais j'ai choisi trois sujets phares pour compléter notre troisième section sur l'algèbre linéaire : les problèmes de vecteurs propres/valeurs propres, les factorisations matricielles et les formes canoniques, ainsi que la résolution de problèmes de moindres carrés.

4.16.1. Introduction aux vecteurs propres. Un vecteur

propre à droite v = 0 de la matrice A est tout vecteur tel qu'il existe un nombre réel ou complexe λ avec Av = λv. Autrement dit, en multipliant Av, on obtient le double de v, ou un tiers de v, ou un million de fois v, ou un multiple constant de v. Considérons comme acquis que ces vecteurs sont très importants dans certaines applications scientifiques et concentrons-nous plutôt sur leur recherche. Au fait, le terme pour λ qui apparaît dans la définition est « valeur propre ».

Si A est une matrice n × n, alors soit In la matrice identité n × n.
les formulations suivantes sont équivalentes :

$$\text{Désactivé} = \lambda v$$

$$\text{Désactivé} - \lambda v = 0$$

$$\text{Désactivé} - \lambda(\text{In}v) = 0$$

$$(A - \lambda(\text{In}))v = 0$$

La dernière équation montre clairement que pour que l'on trouve des vecteurs v intéressants (c'est-à-dire non nuls), la matrice (A−λIn) doit être singulière.
Bien sûr, cela signifie que le déterminant de (A − λIn) est nul. Nous ne connaissons pas encore la valeur de λ, donc appelons-le x. Nous construisons une nouvelle matrice, où toutes les entrées hors diagonale correspondent à celles de A, mais où les entrées sur la diagonale de A ont « −x » attaché. Nous pouvons demander à Sage de prendre le déterminant de cette matrice, ou le calculer manuellement.

Machine Translated by Google

Cependant, nous savons que si x est une valeur propre, alors ce déterminant est nul.
Voyons un petit exemple :

$$A = \begin{matrix} 0 & 0 & 0 & -4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 0 \end{matrix} \qquad \text{devient } A - xI_n = \begin{matrix} -x & 0 & 0 & -4 \\ 1 & -x & 0 & 0 \\ 0 & 1 & -x & 5 \\ 0 & 0 & 1 & 0-x \end{matrix}$$

Ensuite, il s'avère que le déterminant est

$$\det(A - xI_n) = x^4 - 5x^2 + 4$$

Ce qui m'inquiétait, car c'est un polynôme du quatrième degré. Cependant, c'est un polynôme biquadratique, donc il se factorise facilement.

$$x^4 - 5x^2 + 4 = (x + 2)(x + 1)(x - 1)(x - 2)$$

Au fait, ce polynôme que nous venons de trouver s'appelle le polynôme caractéristique de A. Maintenant, nous savons que les valeurs de x qui seront des valeurs propres sont

$$x \in \{-2, -1, 1, 2\} \text{ et}$$

nous pouvons procéder à leur recherche.

Prenons simplement $\lambda = 2$ comme exemple, nous devons résoudre

$$(A - \lambda(I_n))v = (A - 2I_n)v = 0$$

Mais ce n'est qu'un problème d'algèbre linéaire standard. Nous avons appris à le résoudre à la page 137 en utilisant l'opérateur barre oblique inverse ou la commande « solve_right ». Nous devons le faire pour chacune des quatre valeurs propres, mais ce n'est clairement pas un problème difficile.

Il existe d'autres méthodes pour trouver les vecteurs propres. L'avantage ici est que le polynôme est facile à manipuler. Comme la résolution des polynômes de degré 5 et supérieur n'est pas garantie, notre technique précédente (factorisation du polynôme caractéristique) peut s'avérer irréalisable. Heureusement, Sage gère cette tâche pour nous.

Dans Sage, l'ensemble du processus ci-dessus peut être rationalisé, comme ci-dessous :

```
A = matrice([ [0,0,0,-4], [1,0,0,0], [0,1,0,5], [0,0,1,0] ])
I = matrice_identité(4)

imprimer "A:"
imprimer
A
imprimer
imprimer
"I:"
imprimer I imprimer
imprimer
(AI*x) imprimer imprimer
det(AI*x) imprimer facteur(det(AI*x))
```

Machine Translated by Google

Cela produit la sortie

```
A:
[ 0 0 0 -4] [ 1 0 0 0]
[ 0 1 0 5] [ 0 0 1 0]
```

```
Moi : [1 0 0 0]
[0 1 0 0] [0 0 1
0] [0 0 0 1]
```

```
[-x 0 0 -4] [ 1 -x 0
0] [ 0 1 -x 5] [ 0 0
1 -x]
```

```
x^4 - 5*x^2 + 4 (x + 2)*(x
+ 1)*(x - 1)*(x - 2)
```

Au fait, le polynôme caractéristique de A a une relation fascinante avec A. Si vous tapez

imprimer A^4 - 5*A^2 + 4 où

nous avons pris la matrice A et l'avons traitée comme si elle était x, nous obtenons la matrice zéro - cela se produira toujours.

4.16.2. Trouver efficacement des valeurs propres dans Sage.

La procédure décrite dans la section précédente visait à vous donner une idée de ce que signifie réellement trouver un vecteur ou une valeur propre. Les procédures utilisées peuvent être intéressantes, tant par les mathématiques qu'elles exploitent que par l'informatique algorithmique choisie pour assurer l'efficacité du processus.
Nous ne pouvons pas entrer dans les détails ici, mais je recommande le livre de Lloyd Trefethen et David Bau, Numerical Linear Algebra, publié par la Society for Industrial and Applied Mathematics en 1997.

Nous pouvons trouver les valeurs propres ou les vecteurs propres directement, avec les commandes : A.eigenvalues()
A.vecteurs propres_gauche()
A.eigenvectors_right()

Cependant, le polynôme caractéristique est parfois précieux en soi.
C'est vrai. Pour l'obtenir, il faut utiliser l'une des commandes suivantes :

A.charpoly()
factor(A.charpoly())

Machine Translated by Google

Pour démontrer les compromis entre ces méthodes, nous étudierons la matrice suivante :

$$B = \begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}$$

Considérez le code suivant :

```
B = matrice(2, 2, [1, 2, 3, 4] )

imprimer B.eigenvectors_right() imprimer
b(x) =
B.charpoly() imprimer
résoudre( b(x)==0, x )
```

Ce code produit la sortie suivante :

```
 [(-0,3722813232690144?, [(1, -0,6861406616345072?)], 1),
                    (5.372281323269015?, [(1, 2.186140661634508?)], 1)]


[ x == -1/2*sqrt(33) + 5/2, x ==
1/2*sqrt(33) + 5/2 ]
```

Nous voyons une séquence d'éléments séparés par des virgules et entre crochets, ce qui nous permet de savoir que nous avons une liste dans la syntaxe de Sage et Python. Chacun des deux éléments de la liste est un triplet ordonné, composé de la valeur propre λ et du vecteur propre v, suivi de la multiplicité de la valeur propre dans le polynôme caractéristique de la matrice.

Ci-dessous se trouve le résultat de solve, où nous avons explicitement trouvé les racines du polynôme caractéristique. Nous obtenons ainsi la forme algébrique exacte des valeurs propres. Cela me convient mieux, car sinon, nous ne connaissons pas vraiment la signification des valeurs numériques.
Cependant, dans de nombreuses applications, les valeurs numériques suffisent. Notez que Sage rappelle que les valeurs propres sont des approximations numériques dans la commande eigenvectors en utilisant le symbole ? après le chiffre le moins significatif.

J'ai déjà indiqué que si l'on insère la matrice A à la place de x dans son polynôme caractéristique, on obtient la matrice nulle. Le polynôme caractéristique est généralement, mais pas toujours, le polynôme de plus faible degré qui produit ce phénomène. Cependant, si le polynôme caractéristique a des racines répétées, il existe un autre polynôme de degré inférieur qui enverra également A à la matrice nulle. C'est le « polynôme minimal » de A, et il est trouvé par la commande A.minpoly().

4.16.3. Factorisations matricielles

Il existe de nombreuses formes canoniques et factorisations de matrices dans le monde des mathématiques. Sage en possède quatorze intégrées, d'après ce que j'ai pu trouver.

Machine Translated by Google

(il y en a peut-être plus que ce que je n'ai pas trouvé.) Si vous avez une application particulière en tête qui utilise souvent une décomposition, une factorisation ou une forme canonique particulière, j'espère que vous la trouverez ci-dessous.

A.comme somme de permutations()	A.LU()
A.cholesky()	A.QR()
A.frobenius()	A.forme rationnelle()
A.gram schmidt()	Formulaire_A.smith()
Forme A.hessenberg()	A.forme symplectique()
Forme A.hermite()	A.forme popov faible()
Formulaire A.jordan()	A.forme en zigzag()

Cependant, la factorisation matricielle la plus connue est la « factorisation LU ». Généralement, lorsqu'on parle de « factorisation LU », on fait référence à la factorisation LUP, où A = LUP, telle que L est une matrice triangulaire inférieure, U une matrice triangulaire supérieure et P une matrice de permutation. Cette matrice de permutation est souvent, mais pas toujours, la matrice identité. Cela la rend généralement « facultative », dans le sens où la plupart des A peuvent s'écrire A = LU sans nécessiter P = In. Dans Sage, cependant, A.LU() est une factorisation PLU. La matrice de permutation est à gauche et non à droite, ce qui est inhabituel. Vous êtes prévenus !

Cela m'a causé un certain embarras une fois, au cours du semestre de printemps 2014, lorsque j'enseignais l'analyse numérique II, et ma factorisation de A = P LU ne s'est pas multipliée jusqu'à l'original A. C'est parce que j'utilisais A = LUP.

4.16.4. Résolution approximative de systèmes linéaires par la méthode des moindres carrés. Que se passerait-il si je voulais résoudre un système de cinq équations à deux inconnues ? Cela paraît même étrange à l'oreille humaine : pourquoi y a-t-il cinq équations s'il n'y a que deux inconnues ? Je souhaite peut-être résoudre le système d'équations linéaires suivant :

$$0m + b = 7{,}1$$
$$1m + b = 5{,}2$$
$$2m + b = 2{,}9$$
$$3m + b = 1{,}05$$
$$4m + b = -0{,}9$$

Parce que ces équations sont linéaires, nous savons que je peux en faire un problème matriciel :

$$\begin{matrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{matrix} \begin{matrix} m \\ b \end{matrix} = \begin{matrix} 7.1 \\ 5.2 \\ 2.9 \\ 1{,}05 \\ -0{,}9 \end{matrix}$$

Machine Translated by Google

Par conséquent, dans Sage, je devrais taper

```
A = matrice(5, 2, [0, 1, 1, 1, 2, 1, 3, 1, 4, 1] ) imprimer A imprimer



b = matrice(5, 1, [7,1, 5,2, 2,9, 1,05, -0,9] ) imprimer b


Un \ b
```

Sans surprise, Sage répond qu'il n'y a pas de solution. J'ai trop de contraintes et pas assez de variables modifiables pour satisfaire les cinq équations simultanément. Que dois-je faire alors ?

Et si une solution approximative était « correcte » ? Il s'avère que je peux obtenir une solution approximative comme suit :

$$Ax = b \text{ devient à la place } A^T Ax = A^T b$$

où AT est la transposée de A. Notez que, bien que A soit une matrice 5×2, puisque AT est une matrice 2×5, AT A est une matrice 2×2. De même, ATb est un vecteur bidimensionnel. Si vous ne connaissez pas l'opération de transposition, sachez qu'elle consiste simplement à intervertir les lignes et les colonnes de la matrice.
Cela signifie que les anciennes lignes sont les nouvelles colonnes, et que les anciennes colonnes sont les nouvelles lignes. Dans notre cas, cela signifie que

$$_{UN}{}^{T} = \quad 0\ 1\ 2\ 3\ 4\ 1\ 1\ 1\ 1\ 1$$

Je tape donc dans Sage le code suivant :

```
A = matrice(5, 2, [0, 1, 1, 1, 2, 1, 3, 1, 4, 1] ) imprimer A imprimer



b = matrice(5, 1, [7,1, 5,2, 2,9, 1,05, -0,9] ) imprimer b imprimer



M = A.transpose() * A d =
A.transpose() * b

M \ d
```

Le résultat indique que ma solution est m = −403/200 ou m = −2,015 et b = 71/10 = 7,1. Ces chiffres vous semblent-ils familiers ? Il s'avère qu'il s'agit précisément du problème de régression linéaire que nous avons abordé à la page 157. Lorsqu'on calcule une régression linéaire (c'est-à-dire une droite d'ajustement optimal) sur n points de données, on résout n équations linéaires à 2 inconnues, en utilisant cette méthode originale appelée « moindres carrés ». Cette méthode a été inventée.

Machine Translated by Google

par Carl Friedrich Gauss (–), ou par Adrien-Marie Legendre (–), selon le livre d'histoire des mathématiques que vous consultez.

Si nous considérons les points de données comme

$$(x1, y1), (x2, y2), . . . (x5, y5)$$

alors nous pourrions définir l'erreur dans le ième point de données comme

$$ei = yi − yˆi = yi − mxi − b \text{ où } ˆyi =$$

mxi + b est la valeur estimée de yi , prédite par la droite de meilleur ajustement. Cette méthode des moindres carrés garantit trois résultats remarquables :

- La somme des ei est nulle.
- La longueur de ei , ou de manière équivalente, la somme : $e_1^{2\ 2 + et\ 2}\ 2 + · · · + et_n$ est comme aussi petit que possible. Notez que ||ei ||2 est égal à cette somme.
- Le point (x, y) est assuré d'être sur la droite y = mx + b. Ici x est la valeur moyenne des x, et y est la valeur moyenne des levure.

Enfin et surtout, les entrées de AT A et AT b ont une signification. Elles sont

$$UN^T A = \begin{matrix} 30 & 10 \\ 10 & 5 \end{matrix} \quad \text{ainsi que } A\ Tb = \begin{matrix} 10h55 \\ 15h35 \end{matrix}$$

Comme vous pouvez le voir, 30 est la somme des carrés des x. Les deux dizaines sont les sommes des x. 5 est le nombre de points de données. 15,35 est la somme des y. Enfin, 10,55 est la somme :

$$x1y1 + x2y2 + · · · + x5y5$$

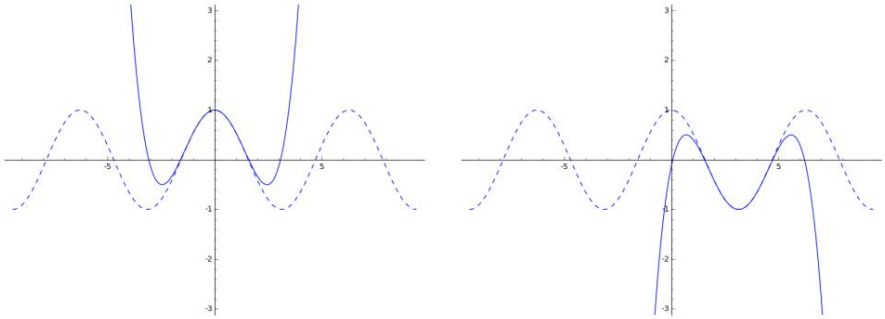
Il convient de noter que la conversion de Ax = b en AT Ax = ATb est souvent pertinente. Le calcul de régressions linéaires n'en est qu'une application. Le système plus petit d'équations linéaires produit par cette astuce est appelé « équations normales ».

4.17. Calcul des polynômes de Taylor ou de MacLaurin

Les polynômes de Taylor constituent d'excellentes approximations des fonctions et sont naturellement utilisés dans de nombreuses applications du calcul différentiel et intégral aux sciences, notamment en physique. Malheureusement, calculer un polynôme de Taylor à la main peut s'avérer complexe. Il faut souvent calculer 8 termes pour obtenir seulement 4 termes non nuls. Cela nécessite de calculer une dérivée 7. Heureusement, Sage peut le faire pour vous.

Tout d'abord, il est utile de rappeler la définition d'un polynôme de Taylor. Rappelons qu'une droite (x) = mx + b est dite tangente d'une fonction f(x) au point x = x0 si la droite tangente « touche » f(x) en x0 et a « la même pente » que f(x) en x0. Plus précisément, cela signifie que la droite a f (x0) comme pente et f(x0) = (x0). On pourrait définir une parabole p(x) comme étant la parabole tangente en x0 de la même manière. Elle devrait avoir p(x0) = f(x0), et p (x0) = f (x0), ainsi que p (x0) = f (x0).

Machine Translated by Google



Polynôme de Taylor sur x = 0 Polynôme de Taylor sur x = π

Figure 3. Deux exemples de polynômes de Taylor de f(x) = cos x.

Analogue à la tangente et à la parabole tangente, le polynôme de Taylor de degré n de f(x) en x0 est l'unique polynôme de degré n p(x) tel que p(x0) = f(x0), et p (x0) = f (x0), ainsi que p (x0) = f (x0), et ainsi de suite, jusqu'à la dérivée n. La définition ne restreint pas les dérivées n+1 ou supérieures : elles peuvent être différentes ou identiques.

Curieusement, certains professeurs de mathématiques insistent pour que si x0 = 0, le polynôme soit appelé polynôme de MacLaurin, tandis que si x0 = 0, il doit être appelé polynôme de Taylor. Cette règle est extrêmement étrange. Les mathématiciens impliqués sont Brook Taylor (–) et Colin Maclaurin (–), ainsi que James Gregory (–). Malheureusement, chacun d'eux a eu une vie inhabituellement courte, même selon les normes de leur époque.

4.17.1. Exemples de polynômes de Taylor. Nous allons

d'abord calculer le polynôme de Taylor du cinquième degré de f(x) = cos x au point x = 0. Le code ci-dessous tracera également la fonction cosinus (courbe en pointillés) et le polynôme de Taylor (courbe pleine). f(x) = cos(x) p(x) = taylor(f(x), x, 0,

5)

```
imprimer « Polynôme : »
imprimer p(x)

P1= tracé( p(x), -3*pi, 3*pi, ymax=3, ymin=-3 )
P2= plot( f(x), -3*pi, 3*pi, style de ligne='--' ) show(P1 + P2)
```

La sortie produite est le polynôme lui-même :

1/24*x^4 - 1/2*x^2 + 1

ainsi que le tracé donné à gauche dans la figure 3.

Machine Translated by Google

En attendant, nous pouvons changer le 0 dans la deuxième ligne de notre code pour qu'il soit pi. Nous obtenons alors le polynôme de Taylor du 5e degré de f(x) au point x = π.
C'est l'intrigue ci-dessus, à droite.

En résumé, la commande Taylor attend quatre entrées : premièrement, la fonction ; deuxièmement, la variable ; troisièmement, le point autour duquel l'approximation est effectuée ; quatrièmement, le degré d'approximation requis. Ici, nous avons demandé le cinquième degré et obtenu le quatrième. C'est parce que le cinquième…
Le coefficient de degré s'avère être nul. Comme on n'écrit généralement pas 0x, ce terme de ^{5}, cinquième degré est totalement absent.

Au fait, comme il s'agit d'une fonction à quatre paramètres, il m'arrive presque toujours de confondre leur ordre. Les « vrais programmeurs » consultent toujours la documentation. Personne ne se souvient vraiment des détails précis comme l'ordre des entrées. Même si vous êtes sûr de vous souvenir de chaque paramètre, il est préférable de vérifier quand même (en utilisant les techniques de la section 1.10, page 47) pour éviter toute erreur.

4.17.2. Application : Comprendre l'évolution de g En physique I, lorsque nous travaillons sur des problèmes se déroulant sur Terre (et près de sa surface), nous posons simplement g = 9,82 m/s². Nous pouvons alors calculer la force de gravité (c'est-à-dire le poids) de toute masse via w = mg, ou w = −mg, selon les conventions de signe préférées. Lorsque nous travaillons sur des problèmes impliquant des satellites, non trivialement éloignés de la surface terrestre, nous apprenons la formule de Newton pour la force de gravité :

$$F = \frac{Gmems}{2\ r}$$

où G est la constante universelle de gravitation, me est la masse de la Terre, ms est la masse de l'objet (généralement un satellite) et r est la distance de l'objet au centre de la Terre.

Tout cela est plutôt excellent, mais que faire pour les distances intermédiaires ? Si nous sommes à 1 mètre au-dessus de la surface terrestre, nous savons qu'il faut utiliser g = 9,82 m/s² . Si nous sommes à 106 mètres au-dessus, nous devrions certainement utiliser la formule de Newton. Cependant, il serait intéressant de savoir à quelle distance nous nous trouvons si nous utilisons g = 9,82 m/s² dans un problème, par exemple concernant un avion de ligne ou le mont Everest. Nous allons maintenant explorer cette question en créant une représentation polynomiale de Taylor de la formule de Newton. La taille des coefficients nous renseignera beaucoup.

Machine Translated by Google

Si nous définissons F = w = msg, alors nous découvrons quelle est la valeur de « petit g » prend cette valeur r particulière. Voir le calcul suivant : Gmems

$$F = \frac{}{2\,r}$$

$$\text{message} = \frac{Gmems}{2\,r}$$

$$g = \frac{Gme}{2\,r}$$

$$g = \frac{Gme}{(re + x)^2}$$

Comme vous pouvez le constater, la ms à gauche annule la ms à droite. Cela n'est pas surprenant, car tout comme g = 9,82 m/s² pour tous les objets à la surface de la Terre, quelle que soit leur masse, la valeur locale de g loin de la surface de la Terre ne devrait pas dépendre de ms. Enfin, vous constaterez que nous avons substitué r = re + x à la dernière étape. re représente le rayon de la Terre et x la distance entre la surface de la Terre et l'objet, généralement appelée altitude de l'objet. En effet, il est facile de mesurer l'altitude d'un objet avec un altimètre, mais il est plutôt difficile de couper la Terre entière en deux simplement pour placer l'extrémité d'une énorme règle en son centre.

Nous allons maintenant prendre cette fonction

$$g(x) = \frac{Gme}{(re + x)^2}$$

et calculer un polynôme de Taylor à x = 0, qui est la surface de la Terre.
Faut-il utiliser un polynôme du premier degré ? Du second degré ? Ou peut-être du troisième ? À ce stade du problème, ce n'est pas évident. Ma stratégie (depuis mes années d'ingénieur) consiste à effectuer une approximation de haut degré, puis à éliminer la plupart des coefficients. Leurs valeurs me permettront de déterminer lesquels éliminer. Je vais donc demander à Sage de produire une approximation du sixième degré. Le code est le suivant : G = 6,67384*10^(-11)

```
Moi = 5,97219*10^24
re = 6,371*10^6 g(x) =
G*Me/(re + x)^2

p(x) = taylor(g(x), x, 0, 6)

imprimer « Polynôme : » imprimer
p(x)

imprimer « Juste pour confirmer : »
imprimer g(0)
imprimer p(0)
```

Machine Translated by Google

Ce code produit la sortie suivante :

```
Polynôme :
(1,02788991701e-39)*x^6 - (5,61315999537e-33)*x^5 + (2,98012019421e-26)*x^4 - (1,51890766058e-19)*x^3 +
    (7,25772052919e-13)*x^2 - (3,08259583276e-06)*x + 9,81960902527

Juste pour confirmer :
9.81960902526829
9.81960902527
```

Les lignes « juste pour confirmer » sont là pour nous rappeler qu'à x = 0, nous sommes à la surface de la Terre et que nous nous attendons à utiliser g = 9,82 m/s² . Cela semble correspondre. Quant au polynôme lui-même, les coefficients des termes de troisième, quatrième, cinquième et sixième degrés sont exceptionnellement petits ; nous pouvons donc les supprimer. Pour la plupart des problèmes, nous pourrions également supprimer le coefficient quadratique, mais si x = 2 × 106 m² , alors x = 4 × 1012 m² et donc le coefficient de 7,25 × 10−13 semble valoir la peine d'être conservé dans ce cas.

Le coefficient constant est rassurant, car il nous rappelle que g = 9,82 m/s² à la surface de la Terre. Notez que le nombre 9,82 m/s² n'apparaît nulle part dans notre programme Sage ; il ne s'agit pas d'une entrée, mais d'un calcul mathématique. Le coefficient linéaire est le plus important dans ce cas. Comme il est de −3,08259 × 10−6 , nous savons que si x ≈ 1 000 m, nous n'avons rien à craindre. Nous pouvons utiliser 9,82 m/s² en toute impunité. En revanche, si x ≈ 106 m, alors 9,82 m/s² est une très mauvaise approximation.
De plus, chaque fois que l'on s'élève d'un kilomètre au-dessus de la surface terrestre, g diminue de −0,003 m/s² . Cela signifie que même à 104 m d'altitude, on pourrait envisager d'utiliser la formule de Newton au lieu de 9,82 m/s² .

4.18. Minimisations et multiplicateurs de Lagrange

Trouver le minimum ou le maximum d'une fonction à variable unique est une tâche courante en calcul différentiel et intégral (CDI). Pour une fonction à variables multiples, le problème est un peu plus complexe. Sage intègre plusieurs commandes permettant de gagner du temps pour l'optimisation multivariable.

Il existe deux catégories : l'optimisation contrainte et l'optimisation sans contrainte.

4.18.1. Optimisation sans contrainte

Supposons que vous deviez trouver les minima de la fonction

$$f(x, y, z) = 120(y - x^2 + 1)^2 + (2 - x)^2 + 110(z - y^2)^2 + 150(3 - y)^2$$

Une option, peut-être la plus classique, consiste à prendre le gradient f et à le fixer égal au vecteur nul. L'équation vectorielle f = 0 comporte trois inconnues et trois équations, mais elle est non linéaire. Le système étant petit, il pourrait être résolu par des résultantes, mais cela représente un travail complexe.

Souvent, une solution numérique suffit, mais cela nécessite une estimation initiale ou une condition initiale. Par exemple, votre estimation initiale est x = 0,1, y = 0,3 et z = 0,4. Vous devrez alors saisir ce qui suit.

Machine Translated by Google


```
var('y z') f(x,
y, z)=120*(yx^2+1)^2 + (2-x)^2 + 110*(z - y^2)^2 + 150*(3-y)^2 minimiser(f, [0.1, 0.3, 0.4])
```

Ensuite, nous recevons la sortie

```
L'optimisation s'est terminée avec succès.
        Valeur de fonction actuelle : 0,000000
        Itérations : 14
        Évaluations fonctionnelles : 22
        Évaluations de gradient : 22

(2.0000000019, 3.00000000582, 9.00000003228)
```

Nous pouvons avoir une grande confiance qu'un bon minimum est x = 2, y = 3 et z = 9. Comme vous pouvez le voir, 14 itérations de descente de gradient3 ont été effectuées.
Cependant, nous pourrions être curieux de voir dans quelle mesure notre réponse dépend des conditions initiales. Essayons une hypothèse erronée, par exemple x = 10, y = 30 et z = 40. Nous obtiendrons la même réponse, mais il faudra 50 itérations.

Dans le cas d'une fonction comportant de nombreux minima locaux, il n'est pas évident de déterminer lesquels seront trouvés. Le domaine de la fonction est divisé en « bassins d'attraction ». Ce sont des ensembles de points du domaine qui convergent tous vers le même minimum local. Cependant, les bassins eux-mêmes peuvent être très complexes et, dans des cas extrêmes, fractals. Il en va de même pour les maximisations.

Les fonctions convexes, en revanche, n'ont qu'un seul minimum local ou un seul maximum local, qui sont donc également globaux. De plus, elles ne peuvent pas avoir à la fois un minimum global et un maximum global, mais seulement l'un des deux. En calcul à variable unique, il s'agit de fonctions où f(x) est toujours positif (minimum unique) ou f(x) est toujours négatif (maximum unique). Cependant, la notion équivalente pour les fonctions multivariées n'est pas très claire.

Pour qu'une fonction multivariée telle que f(x, y, z) ait un maximum ou un minimum unique, il s'avère que la matrice hessienne H (voir page 203) doit avoir soit uniquement des valeurs propres positives (minimum unique), soit uniquement des valeurs propres négatives (maximum unique). Certains lecteurs ignorent peut-être l'existence des matrices hessiennes. Pour les fonctions à deux variables seulement, il existe un raccourci vraiment intéressant : f(x, y, z) a un minimum (ou un maximum) unique si

$$\frac{\partial 2f}{\partial x \partial x} \quad \frac{\partial 2f}{\partial y \partial y} \quad - \quad \frac{\partial 2f}{\partial x \partial y}^{2} > 0$$

mais notez qu'il ne s'agit pas d'une relation « si et seulement si ».

La descente de gradient est l'un de mes algorithmes mathématiques préférés, mais je n'ai pas vraiment la place de le décrire ici. Il suffit de dire qu'il a été découvert par Augustin-Louis Cauchy (–), plus connu pour avoir donné naissance à la notion moderne de convergence et de divergence des séries infinies, ainsi qu'à d'autres résultats en analyse réelle.

Machine Translated by Google

Ces problèmes « très bien comportés » (c'est-à-dire des problèmes convexes) apparaissent relativement souvent en économie mathématique. Ils ont la belle propriété selon laquelle la descente du gradient réussira toujours et convergera toujours vers le point optimal unique.

4.18.2. Optimisation contrainte par multiplicateurs de Lagrange

En général, si j'ai une fonction f(x1, x2, . . . , xn) et que je veux minimiser il est soumis à la fonction de contrainte gi(x1, x2, . . . , xn) = 0 pour plusieurs contraintes g1, g2, . . . , gm, alors je devrais définir une nouvelle fonction

$$F(x1, x2, \ldots, xn, \lambda 1, \lambda 2, \ldots, \lambda m) = f(x) + \lambda 1 g1(x) + \lambda 2 g2(x) + \cdots + \lambda m gm(x)$$

où les nouvelles variables, λ1, λ2, . . . , Le Les λm sont appelés « multiplicateurs de Lagragne4 ».

minimum authentique et non contraint de F sera le minimum de f sur l'ensemble de tous les points qui satisfont simultanément toutes les m contraintes— en supposant que l'ensemble n'est pas vide. Nous avons également besoin de l'hypothèse technique selon laquelle gi = 0 pour tous les points où gi(x) = 0, pour chacun des gis. Cette technique Cette hypothèse tend à être vraie dans les problèmes d'application.

4.18.3. Un exemple de multiplicateurs de Lagrange dans Sage

L'exemple suivant est tiré de Calculus, 6e édition de James Stewart, où il se trouve Exemple 4 de la section 15.8, « Multiplicateurs de Lagrange ». Trouvez le point sur la sphère x 2 + et 2 2 + avec = 4 qui est le plus proche du point (3, 1, −1). Nous pourrions essayer de minimiser

$$D = \sqrt{(x-3)2 + (y-1)2 + (z+1)2}$$

mais il est plus logique de minimiser

$$D2 = (x-3)2 + (y-1)2 + (z+1)2$$

ce qui est légal car D n'est jamais négatif. Puisque D n'est jamais négatif, le la tâche de minimiser D et de minimiser D2 sont exactement le même acte.

Nous souhaitons donc minimiser

$$f(x, y, z) = (x-3)2 + (y-1)2 + (z+1)2$$

sous réserve de

$$g(x, y, z) = x \quad 2 + et \quad 2\,2 + avec \quad - 4 = 0$$

ce qui signifie que nous devons minimiser à la place

$$F(x, y, z, \lambda) = (x-3)2 + (y-1)2 + (z+1)2 + \lambda(x \quad 2 + et \quad 2\,2 + avec \quad - 4)$$

nous devrions donc naturellement trouver F et le fixer à 0.

4Nommé en l'honneur de Joseph-Louis Lagrange (–).

Machine Translated by Google

Nous commençons par trouver les quatre dérivées partielles

$$\partial F/\partial x = 2(x - 3) + \lambda(2)(x) \quad \partial F/\partial y = 2(y - 1) + \lambda(2)(y) \quad \partial F/\partial z = 2(z + 1) + \lambda(2)(z)$$

$$\partial F/\partial \lambda = x^2 + y^2 + z^2 - 4$$

Résultant en quatre équations quadratiques à quatre variables :

$$2(x - 3) + 2x\lambda = 0 \quad 2(y - 1) + 2y\lambda = 0 \quad 2(z + 1) + 2z\lambda = 0$$

$$2x^2 + 2y^2 + z^2 - 4 = 0$$

Nous sommes à la fois déçus (car les équations sont quadratiques) et ravis (car la dernière équation garantit clairement que toute solution se trouve sur la sphère en question). Après de nombreuses opérations algèbres, nous obtenons deux solutions.

$$x = \frac{}{6\sqrt{11}}, \text{ et } = \frac{}{2\sqrt{11}}, \text{ avec } = \frac{-2}{\sqrt{11}} \text{ ainsi que } x = \frac{-6}{\sqrt{11}}, \text{ et } = \frac{-2}{\sqrt{11}}, \text{ avec } = \frac{}{2\sqrt{11}}$$

Comme vous pouvez le voir, la première solution est le point sur la sphère le plus proche de (3, 1, −1) et la deuxième solution est le point sur la sphère le plus éloigné de (3, 1, −1).

Exemple dans Sage : pour ce faire, il faudrait calculer F(x, y, z, λ) nous-mêmes. On tape var("yz L")

```
F(x, y, z, L)=(x-3)^2 + (y-1)^2 + (z+1)^2 + L*(x^2 + y^2 + z^2 - 4) minimiser(F, [0,0,0,0] )
```

La réponse de Sage est ci-dessous :

```
Avertissement : l'erreur souhaitée n'est pas nécessairement atteinte en raison d'une perte de précision.
    Valeur de fonction actuelle : 10,480396
    Itérations : 1
    Évaluations fonctionnelles : 28
    Évaluations de gradient : 16

(1,9173321311, 0,639110710366, -0,639110710366, 1,27822142073)
```

Ceci représente un échec de Sage. Nous avons un avertissement, et la « valeur actuelle de la fonction » devrait être nulle. J'ai essayé de nombreuses conditions initiales différentes, encore et encore. C'était très frustrant. Soyez assuré que la commande « minimiser » de Sage fonctionne généralement mieux. Cependant, comme cela m'est arrivé, cela peut vous arriver. Je vais donc vous expliquer comment contourner ce problème.

Machine Translated by Google

Quand la commande Minimize de Sage ne minimise pas. Lorsqu'un mathématicien souhaite minimiser une fonction multivariée, une stratégie très courante consiste à calculer le gradient et à le mettre à zéro. Commençons donc par demander à Sage de calculer le gradient pour nous. Le code ci-dessous le fait.

```
var("yz L")
F(x, y, z, L)=(x-3)^2 + (y-1)^2 + (z+1)^2 + L*(x^2 + y^2 + z^2 - 4) dérivée(F)
```

Sage répond avec (x, y, z, L) |-->(2*L*x + 2*x - 6, 2*L*y + 2*y - 2, 2*L*z + 2*z + 2, x^2 + y^2 + z^2 - 4)

En utilisant le copier-coller, soit dans une autre fenêtre de navigateur exécutant le Sage Cell Server, soit dans Sage Cloud, nous pouvons assembler les équations suivantes et les résoudre avec la commande solve. eqn1= 2*L*x + 2*x - 6 == 0 eqn2= 2*L*y + 2*y - 2 == 0 eqn3= 2*L*z + 2*z + 2 == 0 eqn4= x^2 + y^2 + z^2 - 4 == 0

```
résoudre( [eqn1, eqn2, eqn3, eqn4], [x, y, z, L] )
```

Comme vous pouvez le voir, Sage produit le minimum et le maximum exacts dans ce cas:

[[x == -6/11*carré(11), y == -2/11*carré(11), z == 2/11*carré(11), L == -1/2*sqrt(11)-1], [x == 6/11*sqrt(11), y == 2/11*sqrt(11), z == -2/11*sqrt(11), L == 1/2*sqrt(11)-1]]

De plus, cette solution présente l'avantage d'être exacte et non une simple approximation numérique. Quelle est la morale de cette histoire ?
Si les méthodes symboliques fonctionnent, évitez les méthodes numériques. Celles-ci sont sensibles aux conditions initiales, sujettes aux erreurs d'arrondi, et nécessitent souvent une sélection rigoureuse des paramètres. Les méthodes algébriques, lorsqu'elles sont possibles, contournent tous ces défauts. En revanche, elles sont souvent inapplicables, par exemple pour trouver les racines d'équations polynomiales de degré cinq et plus.

4.18.4. Quelques problèmes appliqués

Il existe de nombreux exemples de multiplicateurs de Lagrange, notamment en économie mathématique. Je recommanderais en premier lieu Principles of Mathematical Economics de Shapoor Vali, publié par Atlantis Press en 2013. En particulier, les multiplicateurs de Lagrange apparaissent au chapitre 10.4.2, « Fonction de production, combinaison de ressources au moindre coût et niveau de production maximisant le profit ».

Je recommande également l'ouvrage « Mathématiques pour la finance : Introduction à l'ingénierie financière », de Marek Capinski et Tomasz Zastawniak, publié en 2003 par Springer dans la collection Springer Undergraduate Mathematics. Les sections 5.1, 5.2 et 5.3 présentent le portefeuille optimal de Markowitz.

Machine Translated by Google

Stratégie consistant essentiellement à choisir des actions contravariantes pour réduire le risque de l'investisseur. Cette stratégie est en fait un énorme problème de multiplicateurs de Lagrange.

4.19. Sommes et séries infinies

Commençons par une somme très simple :

−4+−1+2+5+8+11+14+17+20+23+26+29+32+35+· · ·+131+134+137

Premièrement, nous identifions qu'il s'agit d'une progression arithmétique, car la différence entre termes adjacents est exactement de trois dans tous les cas. Un examen plus approfondi nous apprend que cette progression peut être considérée comme 3k − 7 pour k {1, 2, 3, . . . , 48} ou 3k − 4 pour k {0, 2, 3, . . . , 47}. La première notation me semble plus logique.

Pour représenter cela dans Sage, nous devons taper

```
var("k y")
sum( 3*k-7, k, 1, 48) et nous
```

obtenons la bonne réponse, à savoir 3192.

Nous pourrions vouloir une formule plus générale pour

$$1 + 2 + 3 + 4 + \cdots + (n-2) + (n-1) + n = \sum_{k=1}^{k=n} k$$

dans ce cas, nous devrions taper

```
var("k m") print
sum(k, k, 1, m) print factor( sum(k,
k, 1, m) )
```

Nous recevons en retour la réponse suivante :

```
1/2*m^2 + 1/2*m 1/2*(m
+ 1)*m
```

L'un des grands problèmes ouverts en mathématiques est l' hypothèse de Riemann5 et elle est liée à la fonction zêta de Riemann. Une somme de la forme

$$\frac{1}{1^4} + \frac{1}{2^4} + \frac{1}{3^4} + \frac{1}{4^4} + \frac{1}{5^4} + \cdots = \sum_{k=1}^{k=\infty} \frac{1}{k^4} = \zeta(4)$$

est un exemple d'évaluation de la fonction Zeta de Reimann à s = 4. En général,

$$\frac{1}{1^s} + \frac{1}{2^s} + \frac{1}{3^s} + \frac{1}{4^s} + \frac{1}{5^s} + \cdots = \sum_{k=1}^{k=\infty} \frac{1}{k^s} = \zeta(s)$$

où s peut être n'importe quel nombre complexe, à l'exception des entiers négatifs. Dans Sage, on peut évaluer le cas s = 4 avec

---

5 Nommé en l'honneur de Georg Friedrich Bernhard Riemann (–). Il est le « Riemann » de la géométrie de Riemann (la courbure de l'espace) et de l'hypothèse de Riemann. Il a également beaucoup contribué à l'étude des nombres premiers.

Machine Translated by Google


```
somme(1/k^4, k, 1, oui)
```

Nous obtenons la réponse : π 4/90. Les deux lettres « o » consécutives sont ici le symbole Sage de l'infini, à savoir oo. En plissant légèrement les yeux, le oo ressemble à ∞.

4.19.1. Vérification des identités de sommation

Si vous avez travaillé avec des combinaisons, vous connaissez peut-être les identités suivantes :

$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n-2} + \binom{n}{n-1} + \binom{n}{n} = 2n$$

$$\binom{n}{0} - \binom{n}{1} + \binom{n}{2} - \binom{n}{3} + \binom{n}{4} - \binom{n}{5} + \binom{n}{6} - \cdots \pm \binom{n}{n} = 0$$

Nous pouvons le vérifier avec le code Sage suivant :

```
var("k m") print
somme(binomial(m,k), k, 0, m) print
somme((-1)^k*binomial(m,k), k, 0, m)
```

La réponse révèle que Sage connaît ces identités.

```
2^m
0
```

Une autre astuce amusante consiste à calculer la somme des n premiers cubes au carré de la somme des n premiers entiers. Autrement dit :

$$1^3 + 23 + 33 + 43 + \cdots + (n-1)3 + n^3 = (1 + 2 + 3 + 4 + \cdots + n)^2$$

Au début, j'ai essayé de vérifier cela avec le code suivant

```
var("k m")
somme(k, k, 1, m)^2 - somme(k^3, k, 1, m)
```

pourtant j'ai reçu la réponse suivante :

```
-1/4*m^4 - 1/2*m^3 + 1/4*(m^2 + m)^2 - 1/4*m^2
```

J'ai ensuite essayé le code ci-dessous et j'ai obtenu un « 0 » plus satisfaisant.

```
var("k m") f(m)
= somme(k, k, 1, m)^2 - somme(k^3, k, 1, m) f(m).expand()
```

4.19.2. La série géométrique

Si nous voulions étudier une progression géométrique particulière

$$25 + 20 + 16 + + + \frac{64}{5} \frac{256}{25} \frac{1024}{125} \frac{4096}{625} + \cdots$$

On commencerait par diviser chaque élément par le précédent, pour découvrir que la raison commune est cr = 4/5. On pourrait peut-être considérer la formule générale

Machine Translated by Google

forme d'une progression géométrique :

$$a + a(cr) + a(cr)^2 + a(cr)^3 + a(cr)^4 + a(cr)^5 + \cdots + a(cr)^m = \sum_{k=0}^{k=m} un(cr)^k$$

où a désigne le premier élément et cr la « raison commune ».

Nous pouvons calculer cette somme dans Sage, en utilisant le code ci-dessous :

```
var("a c_r k m")
somme( a*(c_r)^k, k, 0, m)
```

Nous obtenons la formule ci-dessous, qui est correcte.

```
(a*c_r^(m + 1) - a)/(c_r - 1)
```

Pour la somme infinie, nous aurions simplement k compris entre 0 et ∞ au lieu de 0 et m. Cependant, Sage émet une objection lorsque nous saisissons le code suivant :

```
var("a c_r k m")
somme( a*(c_r)^k, k, 0, oo)
```

La réponse que nous recevons est un énorme message d'erreur, suivi de : ValueError:

```
Le calcul a échoué car Maxima a demandé des contraintes supplémentaires ; l'utilisation de la
commande « assume » avant la sommation *peut* aider (un exemple de syntaxe légale est
« assume(abs(c_r)-1>0) », voir « assume ? » pour plus de détails)

Est-ce que abs(c_r)-1 est positif, négatif ou nul ?
```

Le cr désigne la raison, et grâce au calcul II, nous savons que −1 < cr < 1 devrait être le cas. Par conséquent, nous pouvons facilement supposer que |cr| < 1. Nous insérons la ligne d'hypothèse comme nouvelle ligne médiane de notre code :

```
var("a c_r k m")
suppose( abs(c_r) < 1 )
somme( a*(c_r)^k, k, 0, oo)
```

Avec cette ligne ajoutée, la réponse que nous recevons est la réponse conventionnelle.

```
-a/(c_r - 1)
```

Ici, nous pouvons constater que Sage a eu la sagesse de nous obliger à formuler l'hypothèse selon laquelle −1 < cr < 1, car sinon la progression géométrique divergerait. D'ailleurs, la commande « assume » apparaît également en intégration. Voir pages 60 et 200 pour des exemples.

4.19.3. Utilisation de Sage pour guider une preuve de sommation

Supposons qu'on vous demande de calculer

$$\sum_{j=0}^{j=\infty} \frac{x^{2j+1}}{j!} = x + \frac{3x}{1!} + \frac{5x}{2!} + \frac{7x}{3!} + \frac{9x}{4!} + \frac{x^{11}}{5!} + \cdots$$

6À l'exception notable mais extrêmement stupide de a = 0, où la série converge vers 0 quel que soit le choix de cr.

Machine Translated by Google

et en donner une preuve. Nous ignorerons pour l'instant les questions de « rayon de convergence », qui ne se posent pas dans ce problème.

Tout d'abord, demandons à Sage à quoi correspond réellement cette somme, en utilisant le code suivant :

```
var("j")
somme( x^(2*j+1)/factorielle(j), j, 0, oo)
```

La réponse est

```
x*e^(x^2)
```

ce qui sera notre objectif dans notre preuve.

Disons que vous êtes presque sûr que

$$\text{et e} \qquad \frac{}{+\,2\,!} \; 3! \; \frac{}{4!} \; 5! \; 1 \; \frac{}{} \; \frac{}{} \; \frac{}{} \; \frac{6}{\text{et}} + + \cdots 6!$$

mais je n'en suis pas entièrement sûr. Vous pouvez le vérifier avec

```
var("k y")
somme( y^k/factorielle(k), k, 0, oo)
```

Comme vous pouvez le constater, Sage nous a été d'une grande aide : d'abord en nous donnant un objectif pour notre preuve, puis en nous aidant à confirmer une formule de base importante dont nous avions peut-être oublié l'existence. Le reste est assez simple, après avoir remplacé y = x. Nous obtenons la « preuve » suivante, mais un élève devrait bien sûr y ajouter quelques mots.

$$\text{et e} \qquad + 1\,2! \qquad \frac{2\,yy = 1 +}{} \; \frac{}{} \; \text{et} \; \frac{3}{+\,3\,!} \; \text{et} \; \frac{4}{+\,4\,!} \; \text{et} \; \frac{5}{+\,5\,!} \; \text{et} \; \frac{6}{+\,6\,!} + \cdots$$

$$\text{x et}^{2} = 1 + \frac{2\,x}{1} + \frac{4\,x}{2!} + \frac{6\,x}{3!} + \frac{8\,x}{4!} + \frac{x^{10}}{5!} + \frac{x^{12}}{6!} + \cdots$$

$$\text{xex}^{2} = x + \frac{3\,x}{1} + \frac{5\,x}{2!} + \frac{7\,x}{3!} + \frac{9\,x}{4!} + \frac{x^{11}}{5!} + \frac{x^{13}}{6!} + \cdots$$

$$\text{xex}^{2} = \sum_{j=0}^{j=\infty} \frac{2j+1\,x}{j!}$$

Point philosophique Bien sûr,

l'objectif de tout cela est que Sage soit un outil pédagogique pour l'apprentissage de ce sujet réputé difficile : les séries infinies. Sage peut aider les étudiants à surmonter la confusion et la confusion initiales lors de leur première exposition à ce sujet. L'objectif est qu'au moins après un certain temps, l'étudiant consulte Sage moins fréquemment, jusqu'à ce que les problèmes puissent être résolus sans son aide. Ce n'est qu'à ce moment-là qu'un examen rigoureux de Calculus II peut être un succès.

succès.

Machine Translated by Google

4.20. Arithmétique de très haute précision

En réalité, Sage se concentre sur les calculs exacts, ce qui signifie que les fractions sont considérées comme de véritables rapports d'entiers, et non comme leurs approximations décimales, ce qui évite les erreurs d'arrondi. Lorsqu'on a besoin de nombres à virgule flottante, par exemple logpq où p et q sont des nombres premiers distincts, on obtient une précision de 53 bits, soit plus de 15 décimales. C'est largement suffisant pour presque tout.

Cependant, si vous avez besoin de plus de précision, Sage vous le permettra. Grâce à MPFR (Multiple Precision Floating-point Reals), conçu en partie par le professeur Paul Zimmerman, vous pouvez atteindre une précision de 10 000 bits, soit environ 3 010 chiffres. MPFR est fourni avec Sage, mais son utilisation est un peu complexe ; nous vous laissons donc le soin de consulter le manuel MPFR pour en expliquer l'utilisation. http://

mpfr.loria.fr/mpfr-current/mpfr.html

4.21. Systèmes d'inégalités et programmation linéaire

La programmation linéaire permet de résoudre une incroyable diversité de problèmes. Parmi ceux-ci figurent l'acheminement des expéditions, la planification, les mélanges d'ingrédients pour les aliments hautement transformés, l'équilibrage des portefeuilles, la planification des stocks, les allocations publicitaires et une multitude de problèmes de fabrication, sans parler de la recherche d'équilibres de Nash dans les jeux à somme nulle.

Nous n'avons pas l'espace nécessaire pour aborder tous ces sujets (en fait, une bonne couverture de ce sujet nécessite souvent un livre entier bien plus volumineux que celui-ci). Cependant, nous verrons comment utiliser Sage pour résoudre des programmes linéaires présentés sous forme symbolique.

4.21.1. Un exemple simple

Voici un exemple simple :

$$\begin{aligned} &\text{Maximiser} \quad 2x + y \\ &\text{sous réserve} \\ &\text{de : } 1{,}5 - 2x \geq y \quad 1 - \\ &x \geq y \quad 1 - x \leq 2y \\ &x \geq 0 \quad y \geq 0 \end{aligned}$$

Au fait, si vous êtes curieux de voir à quoi ressemble cet exemple, vous pouvez le trouver (sans la fonction objective) sur www.gregorybard.com sous forme d'applet interactive. Son titre est « Finding the Feasible Region ».

Dans tous les cas, pour résoudre le programme linéaire, nous taperions ce qui suit.

```
LP = MixedIntegerLinearProgram( maximisation = True )
```

Machine Translated by Google


```
x = LP.nouvelle_variable()

LP.add_constraint( 1.5 - 2*x[1] >= x[2] )
LP.add_constraint( 1 - x[1] >= x[2] )
LP.add_constraint( 1 - x[1] <= 2*x[2] )

LP.add_constraint( x[1] >= 0 )
LP.add_constraint( x[2] >= 0 )

LP.set_objective( 5*x[1] + 4*x[2] )

imprimer LP.solve()

imprimer LP.get_values( x )
```

Nous allons maintenant parcourir ce code ligne par ligne.

(1) La première ligne est la moins intuitive. Le nom LP donne un nom à ce programme linéaire, quel que soit le nom choisi. Le mot long MixedIntegerLinearProgram doit toujours apparaître, en majuscules et sans espaces. Vous indiquerez ensuite si le problème est une maximisation, comme nous l'avons fait ici, ou vous pouvez indiquer s'il est une minimisation en indiquant = False.

(2) Nous déclarons ensuite notre famille de variables x. Cela revient à déclarer un ensemble infini de variables : x[0], x[1], x[2], x[3], ... Comme vous pouvez le voir dans cet exemple, nous n'avons besoin que de x[1] et x[2]. Vous pouvez même utiliser des indices de manière non consécutive. Nous aurions pu utiliser les indices x[420] et x[88] au lieu de x[1] et x[2] si nous en avions eu la possibilité.

(3) Viennent ensuite les contraintes du problème. Comme vous pouvez le constater, elles sont assez simples. Il faut taper >= pour ≥ et <= pour ≤.
N'oubliez pas de mettre un astérisque entre les coefficients et les variables.

(4) Ensuite, nous devons fournir les contraintes mineures, à savoir que les variables x[1] et x[2] ne doivent jamais être négatifs.

(5) La ligne suivante déclare la fonction objectif. Il s'agit là encore d'une notation Sage simple.

(6) L'instruction d'impression affichera la valeur optimale (dans ce cas, la valeur maximale) de la fonction objective.

(7) La ligne suivante peut paraître déroutante au premier abord. Elle affichera les valeurs des variables qui atteignent cet optimum. Il peut paraître étrange qu'une ligne distincte, après LP.solve(), soit nécessaire.
Cependant, lorsque nous parlons de « noms de variables multiples » ci-dessous, vous comprendrez pourquoi cela est là.

La sortie de ce code devrait être la suivante :

```
4,5
```

Machine Translated by Google

{1 : 0,5, 2 : 0,5}

Cependant, si vous essayez cet exemple peu de temps après la publication de ce livre, vous pourriez recevoir à la place ce qui est ci-dessous.

```
4.5
{1: 0.5, 2: 0.5} /home/
sageserver/sage/ipython/IPython/core/interactiveshell.py:2883: Avertissement : Le comportement par défaut de
new_variable() va bientôt changer ! Elle renverra des variables « réelles » au lieu de variables positives.

ceux.
Soyez explicite et appelez plutôt new_variable(nonnegative=True).
Voir http://trac.sagemath.org/15521 pour plus de détails.
    exec(code_obj, self.user_global_ns, self.user_ns)
```

Sage subit actuellement une modification. Celle-ci concerne la non-négativité des variables dans les programmes linéaires. Le code présenté dans cette section fonctionnera avant et après la modification. Décrire cette modification ici serait fastidieux et déroutant. Cependant, cet avertissement est affiché « pendant un certain temps » avant la modification, afin que les utilisateurs soient informés de son arrivée.

4.21.2. Fonctionnalités pratiques en pratique

Les fonctionnalités supplémentaires suivantes peuvent être trouvées dans Sage, relatives à la programmation

linéaire. • Si vous avez 20 ou 30 variables, alors avoir à plusieurs reprises la ligne

```
LP.add_constraint( x[27] >= 0 )
```

La saisie peut être fastidieuse. Vous pouvez plutôt saisir x =

```
new_variable(nonnegative=True)
```

à la place de la ligne x = new variable(). Tous les indices de x seront alors considérés comme supérieurs ou égaux à zéro, avec une seule ligne de code. Il n'est pas nécessaire d'écrire une inégalité distincte pour chaque variable x.

• Les contraintes d'intervalle sont parfois utilisées, et leur utilisation dans Sage permet de réduire de moitié leur nombre dans certains cas. Certains systèmes de calcul formel nécessitent de saisir $2 \leq 3x4 + 7x8 \leq 5$ comme deux contraintes distinctes : $2 \leq 3x4 + 7x8$ et $3x4 + 7x8 \leq 5$. Cependant, dans Sage, vous pouvez le faire sur une seule ligne :

```
LP.add_constraint( 2 <= 3*x[4] + 7*x[8] <= 5 )
```

• On souhaite parfois dire $5x1+6x2 = 300$. Certains systèmes de calcul formel exigent deux contraintes : $5x1+6x2 \leq 300$ et $5x1 + 6x2 \geq 300$. Cependant, dans Sage, cela peut se faire en une seule ligne :

```
LP.add_constraint( 5*x[1] + 6*x[2] == 300 )
```

Notez qu'il s'agit de deux signes égaux consécutifs.

Machine Translated by Google

• Dans certaines applications, il suffit de trouver une solution réalisable, sans fonction objectif. L'astuce suivante n'est probablement utile que pour les programmes linéaires très volumineux, dont le temps d'exécution peut être extrêmement long. Plutôt que de définir la fonction objectif à 0, pour accélérer le programme, il faut taper LP.set_objective(None).

Si vous tapez LP.show(), vous obtenez un résumé lisible du programme linéaire. Cela peut être très utile pour le débogage. Vous pouvez ajouter et supprimer des contraintes linéaires au fur et à mesure, avant ou après la commande solve. Dans certains cas, ajouter une contrainte, la résoudre à nouveau, ajouter une contrainte, la résoudre à nouveau, etc., sont des stratégies très utiles. Chaque commande LP.solve() connaît les contraintes qui la précèdent dans le programme (sauf celles qui ont été supprimées), mais pas celles qui la suivent. Ces situations se produisent dans la « programmation par objectifs » et dans la méthode des « coupes de Gomery ». Pour supprimer une contrainte, tapez simplement p.remove_constraint(17).

où 17 est le numéro de la contrainte dans la liste affichée lorsque vous tapez LP.show(). N'oubliez pas que les informaticiens comptent à partir de 0 ; la première contrainte affichée est donc la #0, suivie de la #1, puis de la #2.
En conséquence, la contrainte n°17 est la 18e contrainte affichée.

Imaginez que je travaille sur un programme d'expédition avec 10 usines et 300 concessionnaires. Il serait intéressant de pouvoir indiquer le nombre d'articles envoyés de l'usine i au concessionnaire j avec sij . Dans Sage, c'est possible. Il suffit de saisir

```
s = nouvelle_variable( non négatif = Vrai, dim = 2 )
```

et vous pouvez ensuite utiliser des variables telles que s[2][3], ou s[1][5], etc. Notez que s[5][7] n'est pas la même chose que s[7][5].
Bien entendu, vous pouvez également omettre l'option nonnegative=True si vous souhaitez que les quantités négatives soient expédiées des usines aux concessionnaires. • Parfois, nommer toutes les variables simplement par x est peu informatif et empêche une bonne compréhension du problème. Par exemple, dans le projet décrit à la section 2.3, page 71, j'avais tapé : d = myprogram.new_variable() t =

myprogram.new_variable() g =
myprogram.new_variable() m =
myprogram.new_variable()

Ceci a été fait pour que je puisse exécuter le plan variable suivant. Soit d1, d2, . . . , d7 la quantité expédiée depuis Du-luth/ Superior. Soit t1, t2, . . . , t7 la quantité expédiée depuis Two Harbors. Soit g1, g2, . . . , g7 la quantité expédiée depuis Green.

Machine Translated by Google

Baie. Soit m1, m2, . . . , m7 la quantité expédiée depuis Minneapolis. Pour moi, c'est beaucoup plus simple que d'avoir x1, x2, x3, x4, . . . , x26, x27, x28.

Cependant, cela implique de demander les variables séparément. Malgré les efforts déployés, le résultat est beaucoup plus lisible qu'auparavant.

```
imprimer "Duluth :"
imprimer monprogramme.get_values(d)
imprimer "Two Harbors :"
imprimer monprogramme.get_values(t) et
ainsi de suite.
```

4.21.3. Le polyèdre d'un programme linéaire. Vous connaissez probablement la méthode de résolution d'un programme linéaire en traçant des droites dans le plan. Sinon, consultez l'applet « Finding the Feasible Region » sur www.gregorybard.com.

Tout comme un problème à deux variables divise le plan de coordonnées en régions polygonales (convexes), un problème à trois variables divise l'espace tridimensionnel ordinaire en polyèdres (convexes). Sage peut les représenter graphiquement, comme le décrit l'annexe en ligne de ce livre, « Traçage en couleur, en 3D et animations », téléchargeable sur ma page web www.gregorybard.com.

4.21.4. Programmes linéaires entiers et variables booléennes L'ensemble des problèmes modélisables par programmation linéaire devient considérablement plus grand lorsque certaines variables peuvent être désignées comme « entières uniquement ». Si seulement quelques variables sont désignées ainsi, le problème est résolu sous forme de plusieurs programmes linéaires, en utilisant une méthode appelée « coupes de Gomery » ou une autre méthode appelée « branche et borne ». Si de nombreuses variables sont désignées ainsi, le problème peut prendre beaucoup de temps à résoudre.

Quoi qu'il en soit, Sage intègre cette fonctionnalité. Pour montrer que les variables x[i] ne peuvent être que des entiers non négatifs, tandis que les variables y[i] peuvent être des nombres réels non négatifs, saisissez simplement x = LP.new_variable(nonnegative=True, integer=True) y = LP.new_variable(nonnegative=True)

De plus, vous pouvez taper y = LP.new_variable(nonnegative=True, integer=False) pour vraiment faire passer le message à un lecteur humain.

Les variables booléennes sont également utilisées de temps à autre. Certaines variables ne peuvent être que 0 ou 1, et rien d'autre. Pour montrer que toutes les variables z[k] sont de cette forme, nous saisissons z = LP.new_variable(binary=True).

Machine Translated by Google

4.21.5. Lectures complémentaires sur la programmation linéaire

Comme pour de nombreux sujets, le choix du livre à lire sur la programmation linéaire dépend de si vous recherchez des informations sur la théorie ou la pratique.

- Pour la théorie de la façon dont les grands programmes linéaires sont résolus avec le méthode du simplexe, je recommande Linear Programming: An Introduction With Applications, par Alan Sultan, auto-publié sur la plateforme d'édition indépendante Cre-ateSpace en 2011. Ce livre est convient aux étudiants qui n'ont pas suivi beaucoup de cours en mathématiques, ainsi qu'à ceux qui en ont suivi.
- Pour la myriade d'applications, une excellente source (et remarquablement compréhensible) est Optimization in Operations Research de Ron-ald Rardin, publié par Prentice Hall en 1997.

4.22. Équations différentielles

Je pense que pour beaucoup d'entre nous, on nous a dit dès notre plus jeune âge que les mathématiques étaient la clé pour comprendre l'univers, ou du moins l'univers de la science.
Cette attribution a été faite par de nombreuses personnes au cours des siècles, mais sa plus La formulation célèbre7 est de Galilée (–) dans L'Essayeur,

> . . . dans ce grand livre — je veux dire l'univers — qui se dresse continuellement ouvert à notre regard, mais il ne peut être compris à moins que l'on apprenne d'abord à comprendre la langue et interpréter les caractères dans lesquels il est écrit. Il est écrit dans le langage des mathématiques, et ses caractères sont triangles, cercles et autres figures géométriques, sans dont il est humainement impossible de comprendre un seul mot de cela ; sans cela, on erre dans l'obscurité labyrinthe.

Cependant, ce n'est souvent qu'au cours d'équations différentielles, relativement tard dans l'éducation mathématique de certains étudiants, que cette promesse est livré. Soudain, avec le contenu de ce cours, beaucoup des questions les plus difficiles Les problèmes de physique, de chimie, d'écologie et de finance deviennent transparents et faciles, au point de devenir de simples devoirs.

Malheureusement, les problèmes réalistes issus d'applications réelles sont souvent sales : Les coefficients ont des décimales, les équations ne sont pas faciles à résoudre (surtout si l'on inclut la résistance de l'air), et certains problèmes sont tout simplement difficiles. C'est là que Sage peut rendre un réel service à l'étudiant.
peut se concentrer sur la modélisation des phénomènes et poser correctement le problème — le l'ennui des solutions numériques ou des solutions en série peut être mieux délégué au ordinateur. Grâce à cette division du travail entre l'homme et la machine, des problèmes plus difficiles et plus réalistes peuvent être résolus que par un seul être humain travailler de manière isolée. Ce sujet est le joyau des mathématiques appliquées,

---

7J'utilise ici la traduction standard, de Stillman Drake.

Machine Translated by Google

et l'auteur encourage fortement tout étudiant à explorer bien au-delà des limites du cursus de premier cycle et à pénétrer en profondeur dans des sujets avancés.

Heureusement, un livre complet a été écrit pour enseigner le cours d'équations différentielles avec Sage. Il s'agit d'Introduction to Differential Equations Using Sage, de David Joyner et Marshall Hampton, publié par Johns Hopkins University Press en 2012. Cet ouvrage couvre un large éventail de sujets liés aux équations différentielles ; il serait donc superflu de le reproduire intégralement ici.

Au lieu de cela, nous montrerons comment résoudre des équations différentielles simples, puis passerons aux problèmes de valeur initiale et à la construction de champs de pente. Je présente ensuite un problème clé : la modélisation de la trajectoire d'une torpille, y compris la force de traînée hydrodynamique. Sage est extrêmement compétent en résolutions numériques (avec les méthodes de Runge-Kutta), en résolutions de séries, en problèmes aux limites, en systèmes d'équations différentielles ordinaires, en séparation de variables, etc. Le lecteur est invité à se référer à l'ouvrage de Joyner-Hanson pour ces procédures.

4.22.1. Quelques exemples simples

Si nous voulions résoudre l'équation différentielle

$$\frac{dy}{dx} + y = 7$$

alors nous devons taper ce qui suit :

```
y = fonction('y',x) h =
résoudre(diff(y,x) + y == 7, y)

imprimer "Non simplifié :"
imprimer h
imprimer "Simplifié :" imprimer
h.expand()
```

La première ligne est un peu nouvelle pour nous. Tout comme nous avons appris à déclarer des variables inconnues avec la commande var page 38, nous devons également déclarer des fonctions inconnues. La méthode pour déclarer une fonction inconnue s'effectue avec la commande function, mentionnée ci-dessus. Son seul but est d'indiquer à Sage et Python que nous ne savons pas quelle sera la valeur de y(x). Dans tous les cas, nous obtenons la réponse suivante :

```
Non simplifié : (c +
 7*e^x)*e^(-x)
Simplifié : c*e^(-
x) + 7
```

Machine Translated by Google

Vous pouvez également changer diff(y,x) en diff(y,x,2) pour créer la dérivée seconde, rendant l'équation différentielle d 2y + y = 7 dx2

ce qui donne la réponse :

Non simplifié :
k2*cos(x) + k1*sin(x) + 7

Simplifié :
k2*cos(x) + k1*sin(x) + 7

Maintenant, changeons la ligne avec h en une nouvelle équation différentielle, représentant

$$\frac{dy}{} + \sin x = 0 \text{ y } dx$$

en utilisant le

code : h = desolve(y*diff(y,x)+sin(x)==0, y)

Nous obtenons un type de réponse assez

différent : Non
simplifié : -1/2*y(x)^2 == c - cos(x)
Simplifié :
-1/2*y(x)^2 == c - cos(x)

Ici, y(x) a été défini implicitement, et non explicitement. Cependant, un peu d'algèbre permet de comprendre ce qui est réellement voulu : (−1/2)(y(x))² =

$$c - \cos x \ (y(x))^2 = 2 \cos x - 2c$$

$$y(x) = \pm \sqrt{2 \cos x - 2c}$$

Le ± indique que Sage n'aurait pas pu nous fournir une seule fonction comme réponse. À cause du ±, tel qu'écrit, y(x) échouerait au « test de la ligne verticale » et ne pourrait donc pas être considéré comme une fonction. Autrement dit, pour toute fonction, si l'on choisit un x spécifique dans son domaine, il ne peut y avoir qu'une seule valeur de y pour cette valeur de x. Ici, nous aurions deux valeurs de y, ce qui n'est pas autorisé. La meilleure façon de voir les choses est d'imaginer deux solutions : $y1(x) = \sqrt{2 \cos x - 2c}$ $y2(x) = -\sqrt{2 \cos x - 2c}$

En fait, il existe une infinité de fonctions qui satisfont l'équation différentielle, car nous pouvons choisir n'importe quelle valeur de c. Supposons que nous choisissions d'inclure le signe moins (c'est-à-dire en choisissant y²(x)) et que nous choisissions c = 832. Comment vérifier qu'il s'agit bien d'une solution ?

Nous devrions taper le code suivant :

f(x) = -sqrt(-2*(832-cos(x)))

imprimer f(x)*diff(f(x), x) + sin(x)

Machine Translated by Google

Nous obtenons la réponse « 0 », ce qui signifie que l'équation différentielle initiale est bien satisfaite. J'ai l'habitude de toujours vérifier les solutions, car il existe de nombreuses possibilités de faire une erreur de frappe dans une fonction, par exemple en omettant un signe moins ou en tapant un nombre erroné. Puisque Sage se charge de tout, il est dommage de ne pas vérifier.

4.22.2. Un problème de valeur initiale

Nous allons maintenant programmer Sage pour résoudre cette équation différentielle :

$$\frac{d}{dx^2}y + y = x$$

Sous réserve de certaines contraintes de valeur initiale. Si nous essayons de résoudre le problème de la même manière que précédemment, nous

utiliserons la formule
suivante : y = function('y', x) myequation = diff(y, x,
2) + y == x desolve(myequation, y).

Ce code produit la sortie :

k2*cos(x) + k1*sin(x) + x

C'est assez facile à vérifier à la main :

$$f(x) = k2(\cos x) + k1(\sin x) + x f(x) = -k2(\sin x) + k1(\cos x) + 1 f(x) = -k2(\cos x) - k1(\sin x)$$

$$\begin{aligned} f(x) + f(x) &= (-k2(\cos x) - k1(\sin x)) + (k2(\cos x) + k1(\sin x) + x) \\ &= (-k2 + k2)(\cos x) + (-k1 + k1)(\sin x) + x \\ &= x \end{aligned}$$

En fait, si nous avons un problème de valeur initiale, tel que y = 2 en x = 10 et dy/dx = 1 en x = 10, nous n'effectuerons qu'une seule petite modification. Nous modifions la troisième ligne par desolve(de, y, ics=[10,2,1]) où

ics désigne les « conditions initiales ».

[10, 2, 1] signifie que y(0) = 10, y(0) = 2 et y(0) = 1. Nous obtenons alors la réponse suivante : x - 8*cos(10)*cos(x)/(cos(10)^2 + sin(10)^2) -

8*sin(10)*sin(x)/(cos(10)^2 + sin(10)^2)

Bien sûr, sin2 (10) + cos2 (10) = 1, nous devrions donc interpréter cela comme

$$f(x) = x - (8 \cos 10) (\cos x) - (8 \sin 10) (\sin x)$$

Si nous voulons vérifier notre travail, nous avons besoin d'un extrait de code légèrement plus long :

```
y = fonction('y', x)
monéquation = diff(y,x,2) + y == x

f(x) = desolve(myequation, y, [10,2,1] ) print "f(x) = ", f(x)
```

Machine Translated by Google


```
fprime(x) = diff( f(x), x ) fprimeprime(x) =
diff( fprime(x), x )

imprimer "f(10) = ", N(f(10)) imprimer
"f'(10) = ", fprime(10) imprimer fprimeprime(x)
+ f(x) - x
```

Le code ci-dessus produit la sortie ci-dessous :

```
f(x) = x - 8*cos(10)*cos(x)/(cos(10)^2 + sin(10)^2) - 8*sin(10)*sin(x)/(cos(10)^2 +
                    sin(10)^2)
f(10) = 2,00000000000000 f'(10) = 1 0
```

Comme vous pouvez le voir, la sortie ci-dessus vérifie nos trois exigences : premièrement, que f(10) = 2 ; deuxièmement, que f (10) = 1 ; troisièmement, que f(x) satisfait l'équation différentielle f (x) + f(x) = x.

4.22.3. Représentation graphique d'un champ

de pente Supposons que je souhaite résoudre l'équation différentielle :

$$\frac{\text{de } y + x}{x} = 2 + dx$$

avec des conditions initiales y(2) = 4. Cependant, j'aimerais savoir comment les solutions varieraient également sous d'autres conditions initiales. Je dois d'abord résoudre le problème à la valeur initiale comme dans la sous-section précédente. Je dois taper le code suivant : y = function('y',x)

```
myequation =
diff(y,x) + y/x + x == 2 h =
desolve(myequation, y, [2, 4] )

imprimer h.expand()
```

Maintenant j'ai la réponse :

```
-1/3*x^2 + x + 20/3/x
```

ce qui signifie vraiment

$$f(x) = \frac{-1}{3} \quad 2 + x + x \quad \frac{20/3}{x}$$

et donc

$$f(x) = \frac{-2}{}x + 1 - 3 \quad \frac{20/3}{2x}$$

ainsi que

$$\frac{f(x)}{x} = \frac{-1}{}x + 1 + 3 \quad \frac{20/3}{2x}$$

Machine Translated by Google

nous permettant de conclure que la solution est correcte en vérifiant :

$$(x) + \frac{f(x)\,f}{x} + x = \frac{-2}{}x + 1 - 3\,\frac{20/3}{2\,x} + \frac{-1}{}x + 1 + 3\,\frac{20/3}{2\,x} + x$$

$$= \frac{-2}{3} + \frac{-1}{3} + 1\,x + (1 + 1) + \frac{-20/3}{2\,x} + \frac{20/3}{2\,x}$$

$$= 0 + 2 + 0 = 2$$

La nouvelle étape consiste à résoudre l'équation différentielle initiale pour dy/dx. Dans ce cas, nous obtenons

$$\frac{de}{dx} = -\frac{et}{x} - x + 2$$

ce qui signifie

$$\frac{de}{dx} = -1\,x - 1 - 3\,\frac{20/3}{2\,x} - x + 2 = \frac{-2}{3}x - \frac{20/3}{2\,x} + 1$$

et notez que toutes les solutions à l'équation différentielle doivent obéir à cette relation, pas seulement celle qui satisfait nos conditions initiales.

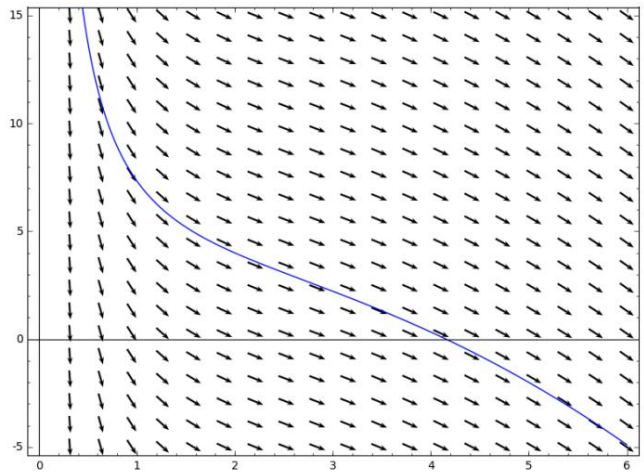
Nous allons ensuite construire un tracé de champ de pente. Pour chaque position sur le plan de coordonnées, nous pouvons utiliser l'équation ci-dessus pour déterminer le rapport dy/dx. Sage sélectionnera environ 400 points, selon un motif 20 × 20, formant une grille sur le plan de coordonnées. Pour chaque point, Sage calculera le rapport dy/dx en ce point et dessinera un vecteur avec cette pente, centré sur ce point.

Notre graphique couvrira 0 < x < 6 et −5 < y < 15, et nous tracerons également notre f(x) particulière, issue de l'exemple précédent, superposée sur la même image. Voici le code:

```
var("y") f(x)
= -1/3*x^2 + x + (20/3)/x

P1 = plot_slope_field( (-2/3)*x - (20/3)/x^2 + 1, (x, 0, 6), (y, -5, 15), headlength = 4,
                headaxislength=3 )
P2 = tracé( f(x), (x, 0, 6), ymax=15, ymin=-5 )
P = P1 + P2
P.show()
```

Ce code produit le tracé suivant :

Machine Translated by Google



Comme vous pouvez le constater, notre solution (la courbe continue) correspond à tous les vecteurs qui lui sont très proches. D'une certaine manière, notre solution revient à connecter une séquence de vecteurs adjacents, tête-bêche, pour obtenir une courbe de balayage allant de x = 0 à x = 6. De même, toutes les autres solutions seraient des courbes de balayage similaires. Avec une condition initiale différente, le point de départ serait différent, mais les vecteurs seraient toujours connectés, tête-bêche, pour obtenir une courbe de balayage.

Un bref mot s'impose sur les deux options

longueur de la tête = 4, longueur de l'axe de la tête = 3

Les experts en équations différentielles préfèrent ne pas utiliser de pointes de flèche sur leurs vecteurs lorsqu'ils tracent des champs de pente. Cela me semble logique, car elles risquent d'encombrer l'image (surtout si les vecteurs sont proches les uns des autres). De plus, la pointe de flèche se trouve toujours à droite du segment de droite représentant la tige, ce qui ne transmet aucune information supplémentaire. En revanche, je préfère vraiment que les pointes de flèche soient là, car elles sont visuellement attrayantes et utiles aux étudiants qui débutent dans l'exploration des équations différentielles.

Le lecteur pourrait vouloir expérimenter en revenant en arrière sur ces deux options pour voir comment l'image change lorsque ces options sont supprimées.

4.22.4. Le problème de la torpille : travailler avec les paramètres. Supposons que je veuille résoudre l'équation différentielle d'une torpille accélérant sous l'eau, depuis un sous-marin profond jusqu'à la surface. Bien sûr, la force de traînée hydrodynamique ne sera pas négligeable, et je dois l'inclure. Le poids de la torpille et la poussée du moteur seraient également des caractéristiques importantes. Commençons par une équation générale.

$$ma = \Sigma F = \underbrace{-kv}_{\text{traîner}} \underbrace{- mg}_{\text{poids}} \underbrace{+ T}_{\text{poussée}}$$

Machine Translated by Google

Où m est la masse de la torpille, k un coefficient de traînée, g = 9,82 m/s² l'accélération due à la pesanteur et T la poussée du moteur de la torpille. Ensuite, je peux transformer cela en une équation différentielle en définissant l'altitude y(t), la vitesse y(t) et l'accélération y(t). (Au fait, j'utilise la convention selon laquelle « haut » est positif et « bas » est négatif.) À ce stade, j'ai

$$\text{mon}(t) = \underbrace{-ky(t)}_{\text{traîner}} \underbrace{-\ mg}_{\text{poids}} \underbrace{+\ T}_{\text{poussée}}$$

que nous allons maintenant saisir dans Sage. Nous allons analyser le code ci-dessous, ligne par ligne.

```
var("mk T g t") y =
fonction('y', t)

équation_générale= m*diff(y, t, 2) == -k*diff(y, t) - m*g + T mon_équation =
équation_générale(m=1000, k=0.5, T=20000, g=9.82)

f(t) = desolve( mon_équation, y, [0, -2000, 0] ) print "y(t) = ", f(t)


plot( f(t), 0, 20, gridlines="mineur" )
```

La première ligne déclare cinq variables, comme nous l'avons vu tout au long de ce livre. La deuxième ligne déclare que la fonction y(t), l'altitude à l'instant t, est inconnue pour l'instant. On obtient alors une reproduction fidèle de l'équation différentielle dominante comme équation générale variable, à la troisième ligne.

La quatrième ligne remplace la masse de la torpille par 1 000 kg, le coefficient de traînée par 0,5, la poussée par 20 000 N et la valeur habituelle de 9,82 m/s² par g. L'unité du coefficient de traînée est le kg/s, ce qui prête à confusion, car c'est ainsi que l'on obtient les newtons après avoir multiplié ce coefficient par une vitesse en m/s.

L'affectation des paramètres en haut du programme nous permet d'expérimenter différentes valeurs de ces paramètres afin d'en évaluer l'impact sur la solution finale. Nous préciserons bientôt que la torpille a une profondeur initiale de 2 000 m et une vitesse initiale de 0 m/s.

La cinquième ligne, sans doute familière maintenant, utilisera desolve pour résoudre cette équation différentielle. Nous posons la condition initiale suivante : à t = 0, l'altitude est y = −2000 m et la vitesse est dy/dt = 0 m/s. La réponse est insérée dans f(t).

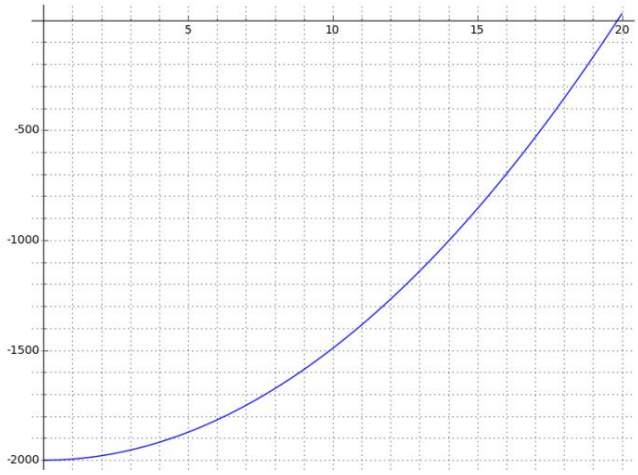
À la sixième ligne, f(t) est connue, car desolve l'a calculée. La nécessité de deux fonctions, y(t) et f(t), perturbe certains étudiants. Considérez y(t) comme un espace réservé pour la réponse finale, mais f(t) est la réponse réelle. Par exemple, dans 3x + 5 = 5x − 3, l'espace réservé pour la solution finale est x, mais la réponse réelle est 4.

Dans tous les cas, la solution que nous obtenons est

```
y(t) = 20360*t + 40720000*e^(-1/2000*t) - 40722000
```

Machine Translated by Google

et en plus, on obtient une très belle intrigue :



Si vous observez attentivement le graphique, vous constaterez que pendant les 25 % de temps les plus à droite, il ressemble à une droite. De plus, pour des valeurs élevées de t, notre fonction y(t) sera une droite. Cependant, sans la force de traînée hydrodynamique, notre fonction y(t) serait une parabole. Par conséquent, nous pouvons constater que la force de traînée ne modifie pas seulement la solution quantitativement, mais qualitativement. La pente de cette droite est appelée « vitesse terminale » et, à cette vitesse, la force de traînée est égale à la poussée. Bien sûr, il convient de noter que pour y > 0, la torpille est au-dessus de l'eau et qu'un modèle complètement différent doit donc être utilisé à ce stade.

4.23. Transformées de Laplace

La transformée de Laplace, du nom de Pierre-Simon de Laplace (1749–1827), est intéressante à bien des égards. Premièrement, elle transforme de nombreux problèmes complexes d'équations différentielles en simples problèmes d'algèbre, bien que souvent assez complexes. Deuxièmement, c'est une analogie étonnante entre deux mondes : le monde du temps, t, notre monde, et celui de l'espace des transformées, s, un monde étranger à notre pensée, mais que nous pouvons utiliser pour résoudre des problèmes importants et pratiques. Troisièmement, calculer une transformée de Laplace à la main constitue un excellent réexamen des techniques d'intégration, offrant une révision du calcul différentiel et intégral II, généralement un an ou plus après son apprentissage. Voyons maintenant comment calculer les transformées de Laplace dans Sage.

4.23.1. Transformation de f(t) en L(s)

Pour notre premier exemple, nous allons trouver le L(s) correspondant à f(t) = t cos(t) dans notre monde. Nous saisirons var("s") f(t) =
t*cos(t)

L(s) = laplace(f, t, s)

Machine Translated by Google

imprimez L(s)

et nous recevons en retour la réponse :

2*s^2/(s^2 + 1)^2 - 1/(s^2 + 1)

Remplaçons simplement f(t) par une nouvelle fonction, en conservant les quatre autres lignes. Prenons maintenant f(t) = 5tet−3 comme exemple. Modifions uniquement la deuxième ligne par : f(t) =

5*t*exp(t-3), ce qui donne

la réponse : 5*e^(-3)/(s - 1)^2

Après toutes ces années, je suis toujours étonné de voir qu'une énorme collection de fonctions, dont certaines sont assez complexes en termes de t, deviennent simplement des fonctions rationnelles de s. Bien sûr, c'est là tout l'intérêt : si les fonctions ne se simplifiaient pas considérablement dans le monde des transformations, il n'y aurait aucune raison de les transformer !

4.23.2. Calcul de la transformée de Laplace « La voie longue »

Parce que la transformée de Laplace est définie comme

$$L(s) = \int_{0+}^{\infty} f(t)e^{-st}\,dt$$

nous imaginerions normalement que le code pour calculer la transformée de Laplace de notre exemple précédent serait comme ci-dessous : var("s")

f(t) =

5*t*exp(t-3)

L(s) = intégrale(f(t)*exp(-s*t), t, 0, oo) imprimer L(s)

Cependant, nous recevons une erreur de Sage, composée de nombreuses lignes ininterprétables, suivies du texte suivant comme deux dernières lignes. N'oubliez pas : dans Sage, les dernières lignes d'un message d'erreur sont les plus importantes.

ValueError : le calcul a échoué car Maxima a demandé des contraintes supplémentaires ; l'utilisation de la commande « assume » avant l'évaluation intégrale *peut* aider (un exemple de syntaxe légale est « assume(s-1>0) », voir « assume ? » pour plus de détails)

S-1 est-il positif, négatif ou nul ?

Comme vous pouvez le constater, Maxima, le package de Sage qui calcule les intégrales, est confus quant au signe de s > 1. Par conséquent, nous insérons la ligne assume(s>1) immédiatement au-dessus de la ligne contenant l'intégrale. La première fois que vous avez vu la commande assume, c'était probablement à la page 60, et bien qu'elle puisse avoir d'autres utilisations, je ne l'ai vue utilisée que pour aider les algorithmes d'intégration et de sommation à parvenir à une conclusion. C'est très rare.
Un autre exemple, en résumé, peut être trouvé à la page 184.

Après avoir fait cela, nous obtenons le

résultat 5*e^(-3)/(s - 1)^2

Machine Translated by Google

Ce qui est sans aucun doute correct. Cependant, par curiosité, nous pourrions essayer les deux autres cas, à savoir assume(s<1) et assume(s==1). Cependant, nous obtenons le message d'erreur ci-dessous dans ce cas (après de nombreuses lignes inutiles).

ValueError : l'intégrale est divergente.

En résumé, nous pensons que Maxima est justifié d'exiger cette hypothèse explicite, car l'intégrale n'était pas convergente sans elle. De plus, la commande laplace fait tout cela pour vous, ce qui est très pratique.

4.23.3. Transformation de L(s) en f(t)

Voyons maintenant comment convertir nos L(s) de l'espace de transformation vers notre monde. Le code à saisir est le suivant :

```
var("s t")
L(s) = 5*e^(-3)/(s - 1)^2 print
inverse_laplace( L(s), s, t )
```

Ce code produit la bonne réponse : 5*t*e^(t - 3)

Alternativement, nous pouvons essayer de mettre Sage au défi avec un problème plus difficile. Imaginez qu'après un problème d'ingénierie des systèmes de contrôle, votre réponse finale soit $L(s) = 1/(s^3 + 1)$. Calculer la transformée inverse à la main sera un peu difficile, car il faudra effectuer une décomposition en fractions partielles. Sage, en revanche, peut la calculer facilement. Il suffit de modifier la ligne centrale du code ci-dessus :

```
L(s) = 1/(s^3+1)
```

Cela produit la réponse donnée ci-dessous :

```
1/3*(sqrt(3)*sin(1/2*sqrt(3)*t) - cos(1/2*sqrt(3)*t))*e^(1/2*t) + 1/3*e^(-t)
```

Cependant, cette fonction est très difficile à comprendre. Par exemple, si je dois l'intégrer dans un article de mathématiques techniques, je dois déterminer comment l'encoder en LATEX. Heureusement, Sage peut m'aider. Si je remplace la dernière ligne par latex(inverse_laplace(L(s), s, t))

Puis j'obtiens une réponse effrayante

```
\frac{1}{3} \, {\gauche(\sqrt{3} \sin\gauche(\frac{1}{2} \, \sqrt{3} t \droite) - \cos\gauche(\frac{1}{2} \, \sqrt{3} t\droite)\droite)} e^{\gauche(\frac{1}{2} \, t\droite)} + \frac{1}{3}
```

que je peux insérer dans mon travail mathématique en LATEX. Son codage est le suivant :

$$f(t) = \frac{1}{3} \sqrt{3} \text{ péché } \frac{1}{2} \sqrt{3t} - \cos \frac{1}{2} 2 t) \sqrt{3t} e \ (1 \quad 1 + \sqrt{3}$$

ce qui est beaucoup plus lisible.

Machine Translated by Google

4.24. Calcul vectoriel dans Sage

Dans cette section, nous apprendrons à utiliser Sage pour calculer les opérateurs les plus importants du calcul vectoriel, notamment la divergence, le roulis, le laplacien, le hessien et le jacobien, ainsi que pour calculer des intégrales doubles. Il peut être utile de revoir les sections sur les tracés de contour (section 3.5, page 104) et sur les tracés de champ vectoriel (section 3.7, page 114), car elles vous permettront de représenter graphiquement les nombreux exemples présentés dans cette section.

Ne vous inquiétez pas si vous trouvez ce contenu confus. Si les professeurs de physique maîtrisent généralement très bien ce sujet, la plupart des professeurs de mathématiques n'ont pas ces formules sous la main, à moins d'avoir enseigné ce cours récemment ou que ce soit leur domaine de recherche. Le guide d'étude, extrêmement célèbre, pour enseigner ce sujet aux ingénieurs est un ouvrage intitulé « Div, Grad, Curl, and All That: an Informal Text on Vector Calculus », de Harry Moritz Schey, publié par WW Norton and Company. La quatrième édition a été publiée en 2004, mais la première édition date de 1973. La longévité de ce manuel témoigne de son ton particulièrement bavard, presque sarcastique et conversationnel.

4.24.1. Notation des fonctions vectorielles

Avant de commencer, nous utiliserons la notation suivante tout au long de cette section. Une fonction mathématique f prenant en entrée un vecteur de dimension n et produisant en sortie un vecteur de dimension m s'écrit $f : R^n \to R^m$.

Notez que cette notation est cohérente avec l'algèbre linéaire. Si f(x) = Ax pour une matrice A de taille m × n, alors $f : R^n \to R^m$.

Les fonctions que nous voyons dans la plupart des calculs multivariés seraient ceux comme

$$g(x, y, z) = x^2 y^3 z^4 + y + z,$$

qui ont une entrée multivariable (ou vectorielle), mais une sortie unique. Ceci est indiqué par $g : R^3 \to R$. En revanche, les fonctions étudiées en calcul infinitésimal I sont des fonctions comme $b(x) = 2e^{-x}$, et elles s'écrivent $b : R \to R$.

Le cas le moins courant, mais pas si rare, est celui d'une fonction à une entrée et plusieurs sorties (ou à sortie vectorielle). Par exemple, une fonction donnant la position d'un avion dans R^3 à tout instant, par exemple :

$$s(t) = \langle 200 + 3t, 300 - t, 400 + 5t \rangle$$

qui serait noté $s : R \to R^3$.

En résumé, les dimensions de l'entrée vont sur le premier R et les dimensions de la sortie vont sur le deuxième R. Cependant, la dimension « 1 »

Bien sûr, le gradient est l'opérateur le plus important en calcul vectoriel (du moins en mathématiques appliquées). Il est généralement abordé dans un cours précédent, comme celui de calcul multivarié, mais nous avons appris à calculer les gradients à la section 4.3.2, page 135. Par conséquent, nous ne le répétons pas ici.

Machine Translated by Google

n'est jamais écrit. Les chapeaux sur f et s ont été inclus car f et s ont des sorties multidimensionnelles (ou vectorielles) et ils ont été exclus sur g et b car g et b ont des sorties unidimensionnelles (ou scalaires).

4.24.2. Calcul de la matrice hessienne La matrice hessienne9 d'une fonction $R^3 \to R$ est donné par

$$H = \begin{matrix} \frac{\partial^2 f}{\partial x \partial x} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y \partial y} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z \partial z} \end{matrix}$$

et se compose de toutes les dérivées partielles secondes possibles, disposées dans une matrice 3 × 3. Pour les autres dimensions, la définition est modifiée de manière évidente. Nous continuerons à utiliser notre exemple de la section sur les dégradés à la page 135. Plus précisément, nous considérons

$$g(x, y) = xy + \sin(x^2) + e^{-x}$$

Pour cet exemple, le code suivant :

```
g(x,y) = x*y + sin(x^2) + e^(-x) diff( g, 2 ) produit la
réponse : [(x, y) |-->
-4*x^2*sin(x^2) + 2*cos(x^2) +
 e^(-x) (x, y) |--> 1] [ (x, y) |--> 1 (x, y) |--> 0]
```

Ce format étrange rappelle que pour chaque point du plan de coordonnées (x, y), chacune des dérivées partielles secondes est un nombre. Or, il y en a quatre, ce qui donne quatre nombres distincts. L'arrangement naturel est une matrice 2 × 2.

Une autre façon de demander à Sage de calculer la matrice hessienne est d'utiliser le code suivant :

```
g(x,y) = x*y + sin(x^2) + e^(-x)
H = diff( g, 2 )

imprimer H(0,3)

imprimer

imprimer H(0,0) imprimer imprimer H(sqrt(pi), 2)
```

Comme vous pouvez le voir ici, nous avons demandé la matrice hessienne, mais au sens numérique. Nous avons demandé la matrice spécifique aux points (0, 3), (0, 0) et

9Nommé en l'honneur de Ludwig Otto Hesse (–), connu pour ses résultats en mathématiques et en physique, et pour avoir été le pionnier de l'étude des applications du déterminant, mais aussi pour avoir écrit un manuel élémentaire populaire sur la géométrie analytique des lignes et des cercles.

Machine Translated by Google

($\sqrt{\pi}$, 2). Les réponses sont celles attendues, comme indiqué ci-dessous :

3 1 3 1 et $-\sqrt{\pi}$ − 2 1
1 0 1 0 1 0

Notez que trois des quatre entrées de la matrice (sauf le coin supérieur gauche) sont verrouillées à des valeurs fixes. Le coin supérieur gauche varie avec x, mais pas avec y. Testez d'autres entrées pour vérifier que cela est vrai. Si vous observez maintenant le Hessien symbolique, vous constaterez que trois des quatre entrées sont des constantes, et que le coin supérieur gauche est une fonction de x, mais constante par rapport à y.

Parfois, nous souhaitons simplement connaître le déterminant de l'hessien. En économie mathématique, il est parfois très utile de produire des exemples de fonctions f(x, y) avec un minimum local (et donc global) unique.

En limitant l'étude aux fonctions à deux variables, f(x, y), on dispose d'un raccourci très utile. Il s'avère que ce phénomène (à savoir l'unicité du minimum ou du maximum local) est garanti si le déterminant de l'hessien est positif pour tous les points (x, y). Il en résulte par analogie que les fonctions univariées g(x) ont un minimum local (et donc global) unique si g(x) > 0 pour tout x. De plus, les fonctions univariées g(x) ont un maximum local (et donc global) unique si g(x) < 0 pour tout x. Ceci étant dit, on peut écrire

```
g(x,y) = (x + y + 2)*(x + y + 1)*(x + y - 1)*(x + y - 2) h(x,y) = det(diff( g, 2 )) print
h(x,y)
```

pour obtenir très rapidement le déterminant de l'hessien. Dans ce cas, le déterminant est nul partout.

4.24.3. Calcul du Laplacien Le Laplacien d'une fonction g : R selon le manuel que vous ^{3} → R est noté soit · g soit 2 g, lisez, et est défini comme

$$\cdot \quad g = \quad 2\,g = \frac{2\,\partial}{\partial x \partial x g} + g + \frac{2\,\partial}{\partial y \partial y} \frac{2\,\partial}{\partial z \partial z}$$

qui est la somme des éléments de la diagonale principale de la matrice hessienne de g. Le Laplacien doit son nom à Pierre-Simon de Laplace (–).

Il s'agit essentiellement de la somme de toutes les dérivées partielles secondes, à l'exclusion des dérivées partielles mixtes. Cet opérateur n'est pas intégré à Sage, mais il est néanmoins facile à calculer grâce aux commandes existantes. Prenons l'exemple de la fonction g : ^{3}R → R que nous avions vu à la page 202, à savoir

$$g(x, y, z) = x^2 + \text{et}^{4\,3+z} + xyz2$$

Machine Translated by Google

Les dérivées partielles secondes pertinentes sont

$$\frac{\partial^2 g}{\partial x \partial x} = 2$$

$$\frac{\partial^2 g}{\partial y \partial y} = 6 \text{ ans}$$

$$\frac{\partial^2 g}{\partial z \partial z} = 12z^2 + 2xy$$

Par conséquent, le Laplacien de g est juste

$$\cdot \quad g = \quad 2\,g = 2 + 6y + 12z^2 + 2xy$$

et était facilement calculée manuellement. Cependant, pour des fonctions plus complexes, nous préférerions utiliser Sage. Pour cette fonction g, nous saisirions g(x, y, z) = x^2 + y^3 + z^4 + x*y*z^2

```
L(x, y, z) = diff( g, x, x ) + diff( g, y, y ) + diff( g, z, z )

imprimer « Laplacien : »
imprimer L(x,y,z)

imprimer "Hessian:"
imprimer diff( g, 2 )
```

Nous obtenons le résultat suivant :

```
Laplacien :
2*x*y + 12*z^2 + 6*y + 2
Hesse : [ (x,
y, z) |--> 2*x*y + 12*z^2 + 6*y + 2 (x,
```

Machine Translated by Google

Comme vous pouvez le constater, la fonction de f1, f2 et f3 est de nous indiquer une coordonnée particulière de f. Dans ce cas, il est facile de calculer la matrice Jacobienne10. qui est défini comme suit

$$J = \begin{matrix} \frac{\partial f1}{\partial x} & \frac{\partial f1}{\partial y} & \frac{\partial f1}{\partial z} \\ \frac{\partial f2}{\partial x} & \frac{\partial f2}{\partial y} & \frac{\partial f2}{\partial z} \\ \frac{\partial f3}{\partial x} & \frac{\partial f3}{\partial y} & \frac{\partial f3}{\partial z} \end{matrix}$$

Dans cet exemple, nous aurions

$$J = \begin{matrix} 0 & 1 & 0 & 2x \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Pour la matrice jacobienne. Parfois, le terme « jacobienne » désigne simplement le déterminant de la matrice jacobienne. Dans ce cas, la réponse serait simplement 0, car la matrice J comporte une ligne de zéros et est donc évidemment singulière.

Pour calculer toutes ces choses dans Sage, nous utiliserions le code suivant :

```
f(x,y,z) = [ y, x^2, 3 ]

print "Fonction :" print
f print
"Jacobien :" print diff(f)
print "Déterminant
jacobien :" print det( f.diff() )
```

Comme vous pouvez le constater, Sage considère que le jacobien est la signification naturelle du mot « dérivée » pour une fonction ayant une entrée et une sortie vectorielles. Après réflexion, cela paraît logique. Plus précisément, si la sortie d'une fonction est unidimensionnelle, mais avec une entrée multidimensionnelle, alors le jacobien serait égal/identique au gradient en tant que vecteur ligne, et Sage considère que le gradient est la signification naturelle du mot « dérivée » pour les fonctions à entrées multiples et à sortie unique.

Cependant, si l'on désire le déterminant de la matrice jacobienne, il faut utilisez explicitement la commande det, comme tout autre déterminant.

4.24.5. La divergence. La

divergence est un opérateur important, mais il n'est pas intégré à Sage. Nous avons vu que l'opérateur laplacien n'était pas non plus intégré à Sage, et nous verrons bientôt que le curl ne l'est pas non plus. Cependant, la divergence est extrêmement facile à calculer avec les commandes existantes.

10Nommé en l'honneur de Carl Gustav Jacob Jacobi (–).

Machine Translated by Google

Pour toute fonction $f : R^3 \to R^3$, la divergence est définie par

$$\cdot f = \text{div } f = \frac{\partial f1}{\partial x} + \frac{\partial f2}{\partial y} + \frac{\partial f3}{\partial z}$$

qui peut être considéré comme la somme des entrées sur la diagonale principale de la matrice jacobienne. (Parfois, on l'appelle « la trace » de la matrice jacobienne matrice.) En ce sens, il est très similaire au Laplacien, qui est la somme des entrées sur la diagonale principale de la matrice hessienne.

Observons que pour chaque point de l'espace tridimensionnel ordinaire, la divergence nous fournira un nombre, et non un vecteur. Cela signifie que la divergence de f est une fonction $R^3 \to R$.

Prenons maintenant un exemple. Prenons la fonction assez fantaisiste

$$f(x, y, z) = xy^2 i + yz^2 j + zx^2 k$$

où nous calculerons à la fois la divergence et la matrice jacobienne. Le code est ci-dessous. Comme il s'agit d'un exemple simple, vous pouvez le calculer. avec un crayon et du papier d'abord.

```
f1(x,y,z) = x*y^2
f2(x,y,z) = y*z^2
f3(x,y,z) = z*x^2

f(x, y, z) = ( f1(x,y,z), f2(x,y,z), f3(x, y, z) )

div(x, y, z) = diff( f1, x ) + diff( f2, y ) + diff( f3, z )

imprimer « Divergence : »
imprimer div(x,y,z)

imprimer « Jacobien : »
imprimer dérivé( f )
```

Ce code produit cette sortie :

```
Divergence:
x^2 + y^2 + z^2
Jacobien :
[ (x, y, z) |--> y^2 (x, y, z) |--> 2*x*y (x, y, z) |--> 0]
[ (x, y, z) |--> 0 (x, y, z) |--> z^2 (x, y, z) |--> 2*y*z]
[(x, y, z) |--> 2*x*z (x, y, z) |--> 0 (x, y, z) |--> x^2]
```

Comme vous pouvez le voir, la divergence est égale à la somme des entrées du principal diagonale de la matrice jacobienne.

4.24.6. Vérification d'une ancienne identité

On vous a peut-être enseigné le théorème selon lequel la divergence du gradient est égal au Laplacien. À la page 204, nous avons calculé le Laplacien de

$$g(x, y, z) = x^2 + e^{t^4 3 + z} + xyz^2$$

Machine Translated by Google

Calculons maintenant la divergence de son gradient et voyons si nous obtenons la même réponse que précédemment. Pour calculer le gradient, nous tapons g(x, y, z) =
x^2 + y^3 + z^4 + x*y*z^2.

imprimer « Original : »
imprimer g(x, y, z)

gradient = dérivée(g)

imprimer « Dégradé : »
imprimer le dégradé
qui donne la sortie :

Original :
x*y*z^2 + z^4 + y^3 + x^2
Dégradé : (x,
y, z) |--> (y*z^2 + 2*x, x*z^2 + 3*y^2, 2*x*y*z + 4*z^3)

Ceci étant dit, nous pouvons interpréter les trois composantes du gradient comme f1, f2 et f3. Nous allons programmer Sage pour calculer la somme des dérivées partielles. Utilisez le code suivant : f1(x,y,z) = y*z^2 + 2*x
f2(x,y,z) = x*z^2 + 3*y^2
f3(x,y,z) = 2*x*y*z + 4*z^3

Lap(x,y,z) = diff(f1, x) + diff(f2, y) + diff(f3, z) print "Laplacien :" print Lap(x, y, z)

Finalement, nous obtenons le résultat :

Laplacien :
2*x*y + 12*z^2 + 6*y + 2 Comme

vous pouvez le voir, il s'agit d'une correspondance exacte avec le Laplacien que nous avions calculé à la page 204. Nous avons maintenant vérifié que la divergence du gradient (de cette fonction particulière) est égale au Laplacien (de cette fonction particulière). C'est loin de prouver le théorème en général, mais il est bon de savoir que nous pouvons au moins vérifier des instances individuelles.

4.24.7. Le recourbement d'une fonction vectorielle

La boucle d'une fonction $f : R^3 \to R^3$, de la forme

$$f(x, y, z) =< f1(x, y, z), f2(x, y, z), f3(x, y, z) >$$

est noté × f = curl f et est calculé par

$$\times f = \text{boucle } f = \frac{\partial}{\partial y} f3 - \frac{\partial}{\partial z} f2 \; je + \frac{\partial}{\partial z} f1 - \frac{\partial}{\partial x} f3 \; j + \frac{\partial}{\partial x} f2 - \frac{\partial}{\partial y} f1 \; k$$

Machine Translated by Google

C'est une formule qui n'est sûrement pas facile à mémoriser. Il existe donc une astuce pour s'en souvenir.

Astuce pour mémoriser la formule du curl :

Personnellement, j'ai toujours été frustré par la difficulté de calcul du curl, notamment parce qu'il s'agit de l'opérateur le plus important après le gradient dans les applications d'ingénierie ou de physique. Il existe en fait une astuce de mémorisation assez simple pour se souvenir de la formule du curl lorsqu'on fait des mathématiques à la main.

$$\text{le} \quad \begin{matrix} i & j & k \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ f_1 & f_2 & f_3 \end{matrix} =$$

$$= \text{que} \quad \begin{matrix} \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ f_2 & f_3 \end{matrix} \partial\,\partial \quad \text{dans} - \text{il} \quad \begin{matrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial z} \\ f_1 & f_3 \end{matrix} \partial\,\partial \; j + \text{ça} \quad \begin{matrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \\ f_1 & f_2 \end{matrix} \partial \; k$$

$$= \frac{\partial f_3}{\partial y} - \frac{\partial f_2}{\partial z} \; i - \frac{\partial f_3}{\partial x} - \frac{\partial f_1}{\partial z} \; j + \frac{\partial f_2}{\partial x} - \frac{\partial f_1}{\partial y} \; k$$

= boucle f

Essentiellement, si vous pouvez vous souvenir de cette matrice 3×3 et prendre sa déterminant avec succès, alors vous pouvez calculer le curl de n'importe quelle fonction $R^3 \to R^3$.

Vous vous demandez peut-être pourquoi je l'ai qualifié de semi-absurde. La raison est que $\partial/\partial x$ est un opérateur, et le calcul

$$\frac{\partial}{\partial x} f_2$$

Ce n'est en aucun cas une multiplication. Pourtant, le calcul de ces déterminants 2 × 2 correspond exactement à deux multiplications séparées par une soustraction. Cependant, en suivant le processus ci-dessus à l'aveugle, les symboles s'affichent visuellement aux bons endroits, et l'astuce mémorisée est donc utile, même si elle n'a pas vraiment de sens mathématique. Cette astuce mémorisée est l'équivalent universitaire de la blague du collège selon laquelle (3)(3) devrait en réalité être égal à 33.

Un exemple simple de la boucle : Parfois, un

exemple simple est préférable. L'exemple suivant a été trouvé dans l'article Wikipédia « Curl : (mathématiques) » du 29 juin 2014. Bien que je ne cite habituellement jamais Wikipédia, l'exposé est remarquablement clair dans ce cas ; il serait dommage de le remplacer par un exemple plus complexe, simplement pour éviter la stigmatisation de Wikipédia. J'ai pris des précautions particulières pour m'assurer qu'il ne contienne aucune erreur, car Wikipédia est réputé pour être sujet aux erreurs.

Machine Translated by Google

Considérez la fonction suivante :

$$c = y \quad 2i =< y2 , 0, 0 >$$

Puisque f2 et f3 sont la fonction nulle dans ce cas, nous pouvons nous concentrer sur f1. Nous pouvons voir que

$$\frac{\partial}{\partial x} f1 = 0 \qquad \frac{\partial}{} f1 = 2y \, \partial y \qquad \frac{\partial}{} f1 = 0 \, \partial z$$

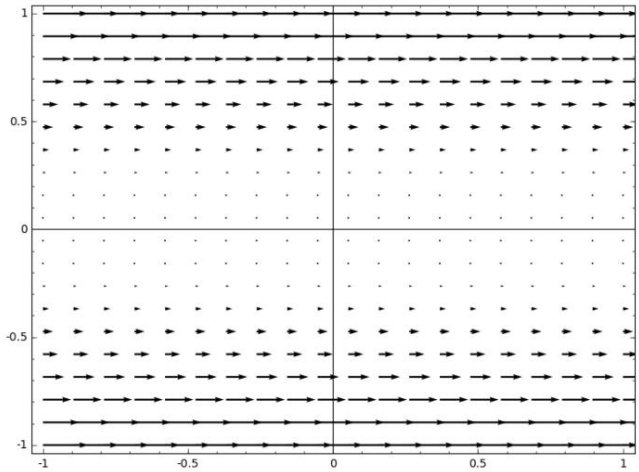
et ainsi la formule normale de la boucle se simplifie rapidement

$$\times f = \text{boucle } f = \frac{\partial}{} \frac{\partial}{} f3 - f2 \, i + \frac{\partial}{} \frac{\partial}{} z = (0 - 0)i + (0 - 0)j + (0 - 2y) \, k = -2y \, k =< 0, 0, -2y > \qquad \frac{\partial}{\partial x} f1 - \frac{\partial}{} f3j + \partial z \qquad \frac{\partial}{\partial x} f2 - \frac{\partial}{} f1 \, k \, \partial y$$

Examinons un tracé de champ vectoriel. Pour le représenter sur une page bidimensionnelle, nous devons ignorer complètement la coordonnée z. C'est correct dans ce cas, car notre exemple n'utilise pas z. Nous pouvons saisir

```
var("y")
plot_vector_field( (y^2,0), (x,-1,1), (y,-1,1) )
```

On obtient l'image :



Comme vous pouvez le constater, toutes les flèches pointent vers la droite. L'abscisse n'a pas d'importance ; les flèches proches de l'axe des x (celles avec une petite ordonnée, en valeur absolue) ont une faible magnitude, tandis que celles éloignées de l'axe des x (celles avec une grande ordonnée, en valeur absolue) ont une magnitude élevée. Maintenant, demandez-vous, si ce tracé représentait un fluide en écoulement, comment vous attendriez-vous à ce qu'une roue à aubes se comporte si elle était insérée :

- au-dessus de l'axe des x ?
- sur l'axe des x ?
- en dessous de l'axe des x ?

Machine Translated by Google

Prenez un moment pour bien y réfléchir. (Veuillez ne pas poursuivre votre lecture avant d'avoir répondu à cette expérience de pensée.)

Quoi qu'il en soit, si la roue à aubes est sur l'axe des x, il n'y aura pas de rotation. Cependant, au-dessus de l'axe des x, on s'attend à une rotation dans le sens horaire, et en dessous, dans le sens antihoraire. Cela est dû au fait qu'un côté de la roue à aubes est beaucoup plus sollicité que l'autre. Si l'on se souvient qu'une rotation dans le sens horaire sur une feuille de papier bidimensionnelle indique un vecteur entrant dans la feuille, et qu'une rotation dans le sens antihoraire indique un vecteur sortant de la feuille, notre réponse < 0, 0, −2y > est logique.

- Lorsque y est nul, toutes les coordonnées de la boucle sont nulles, donc la boucle est zéro. (Cela signifie pas de rotation de la roue à aubes.)
- Lorsque y est négatif, la troisième coordonnée est positive, donc la courbure pointe vers l'extérieur du papier. (Cela signifie une rotation de la roue à aubes dans le sens inverse des aiguilles d'une montre.)
- Lorsque y est positif, la troisième coordonnée est négative, donc la courbure pointe vers l'intérieur du papier. (De même, cela signifie une rotation de la roue à aubes dans le sens des aiguilles d'une montre.)

Il faut maintenant calculer cela dans Sage. Le calcul était facile dans cet exemple extrêmement simple, mais dans d'autres cas, il pourrait s'avérer extrêmement complexe. L'extrait de code suivant est très pratique : var("y z")

```
f1 = y^2 f2 = 0

f3 = 0

curl = ( diff(f3, y) - diff(f2, z), diff(f1, z) - diff(f3, x), diff(f2, x) - diff(f1, y) ) imprimer curl
```

Naturellement, nous obtenons la sortie attendue de (0, 0, -2*y) à partir de ce code.

Un exemple plus complexe de la fonction Curl :

Prenons maintenant un exemple plus complexe, où nous ne souhaiterions pas calculer la fonction Curl manuellement. Voici une fonction légèrement plus complexe :

f(x, y, z) =< x2 y, x + y + sin(x), 0 > pour lequel

nous pouvons utiliser le code suivant : var("y z")

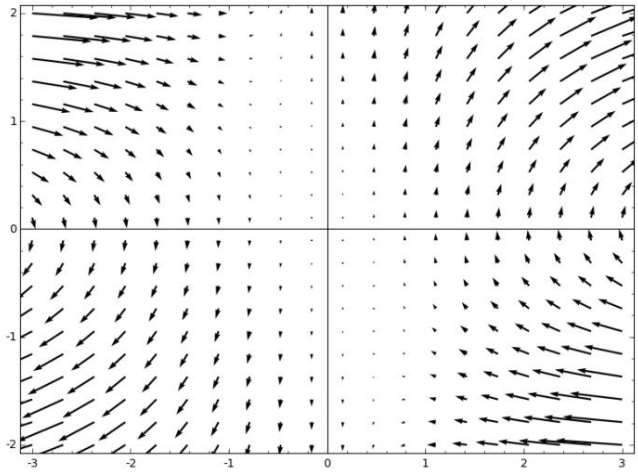
```
f1 = x^2*y f2 = x
+ y + sin(x) f3 = 0
```

Machine Translated by Google


```
curl = ( diff(f3, y) - diff(f2, z), diff(f1, z) - diff(f3, x), diff(f2, x) - diff(f1, y) ) imprimer curl
```

```
fvec = (f1, f2)
plot_vector_field( fvec, (x, -3, 3), (y, -2, 2) )
```

Comme vous pouvez le voir, nous avons ajouté deux lignes pour créer un tracé vectoriel et obtenu l'image suivante :



4.24.8. Un défi : vérifier certaines identités Curl Voici un défi pour vous.

Ouvrez un bon manuel de calcul vectoriel.

- Trouver plusieurs fonctions de la forme g : R dont ^{3} → R et vérifier que le le vecteur du gradient est nul, 0 =< 0, 0, 0 >.
- Trouver plusieurs fonctions de la forme f : R la ^{3} → R ^{3} et vérifier que le divergence du curl est bien 0.

Remarque : La deuxième méthode est un peu plus complexe, car ni la boucle ni la divergence ne sont prédéfinies dans Sage. Cependant, l'extrait de code que j'ai utilisé précédemment pour calculer deux exemples de boucle devrait être très utile.

4.24.9. Intégrales multiples. Le

processus d'intégration dans Sage est le même pour les intégrales multiples que pour les intégrales individuelles. Cependant, il existe une notation alternative qui simplifie la compréhension.

$$\int_0^1 \text{voiture}-x^2 \quad \text{diagnostic}$$

peut être écrit dans Sage

```
comme intégrale( x*exp(-x^2), x, 0, 1)
```

Machine Translated by Google

comme cela a été enseigné dans la section 1.12 à partir de la page 51. Cependant, la syntaxe suivante est

également acceptable : integral(x*exp(-x^2), (x, 0, 1))

Cette deuxième notation semble peu judicieuse, car elle comporte plus de caractères et des parenthèses imbriquées. Cependant, nous allons maintenant voir qu'elle est plus pertinente pour travailler avec plusieurs intégrales. Prenons l'intégrale suivante :

$$\int_0^1 \int_0^{e^{e^2}} dx \text{ à}$$

Dans l'ancienne notation, cela serait

```
var("y")
intégrale( intégrale( exp(y^2), x, 0, y), y, 0, 1)
```

Cependant, dans la nouvelle notation, nous

écrivons

```
var("y") integral( integral( exp(y^2), (x, 0, y) ), (y, 0, 1) )
```

Il me semble plus facile de comprendre que l'intégrale « interne » a 0 < x < y et l'intégrale « externe » a 0 < y < 1. Cette nouvelle notation est similaire à la façon d'exprimer les intervalles que nous avons vue pour la première fois à la page 104, sous le titre « Compatibilité descendante », lors de l'apprentissage des techniques de traçage avancées.

Machine Translated by Google

Machine Translated by Google

Chapitre 5

Programmation dans Sage et Python

Comme vous avez pu le constater dans les chapitres précédents, de nombreuses tâches mathématiques importantes peuvent être réalisées dans Sage sans nécessiter trop de lignes de code. Souvent, quelques commandes suffisent. D'un autre côté, il arrive que l'on cherche à réaliser des tâches plus profondes et plus complexes. Les tâches complexes nécessitent souvent plus de code, allant d'une douzaine de lignes à d'énormes programmes de 10 000 lignes, écrits par plusieurs développeurs répartis dans le monde entier sur plusieurs années.

Dans ce chapitre, nous allons apprendre à écrire des programmes de taille intermédiaire en Sage pour réaliser des tâches mathématiques intéressantes. Sage étant basé sur Python, nous apprendrons une bonne partie de Python au fil du processus. Bien sûr, Python est un langage informatique majeur et possède de nombreuses fonctionnalités, et nous ne pouvons pas espérer les aborder toutes ici. Cependant, nous allons voir chacune des commandes Python les plus courantes utilisées de manière mathématiquement pertinente.

À propos, il est utile de noter que ce chapitre est rédigé en supposant que vous connaissiez le contenu du chapitre 1 et l'utilisation de Sage Cloud, mais pas nécessairement le contenu des autres chapitres de ce livre. Le rythme suggéré pour ce chapitre est d'étudier les sections 5.1 à 5.7, chacune lors d'une soirée distincte. Les élèves patients et doués peuvent peut-être étudier les sections deux à la fois, mais même cela n'est pas recommandé. Ce contenu est assez lourd et il est préférable de l'assimiler en petites sections.

À mesure que les exemples Sage deviennent plus longs, il peut devenir fastidieux de les saisir lettre par lettre sur votre ordinateur. N'oubliez pas que vous pouvez copier-coller ces lignes dans Sage ; inutile de les retaper lettre par lettre ! Il vous suffit de sélectionner le code dans la version PDF (gratuite en ligne) de ce livre, d'exécuter la commande « Copier » dans votre lecteur PDF, d'ouvrir une fenêtre Sage et d'exécuter la commande « Coller ».

Machine Translated by Google

De même, il est essentiel d'utiliser Sage Cloud pour ce chapitre, et non Sage Cell Server. J'utilise Sage Cell Server presque tout le temps, et même si les blocs de code de 10 à 20 lignes que nous allons écrire dans ce chapitre fonctionnent correctement, il est important de conserver une trace permanente de votre code pour plusieurs raisons. Premièrement, en cas de petit problème réseau, il est dommage de perdre tout le code sur lequel vous avez travaillé pendant une heure. Deuxièmement, les experts en programmation reprennent souvent un ancien code qu'ils ont écrit et le copient-collent dans le nouveau programme. Troisièmement, si vous oubliez comment faire quelque chose, il est utile de pouvoir revenir à l'ancien code et de voir comment il a été fait. Comme je suis têtu et que j'utilise trop Sage Cell Server (et que je sous-utilise Sage Cloud), je perds souvent mon travail et je dois tout recommencer.

Enfin, je vous ai présenté des défis tout au long de ce chapitre. De temps à autre, après avoir présenté deux ou trois concepts, je vous propose un problème que je vous demande de résoudre vous-même. N'hésitez pas à les essayer, car ils amélioreront grandement votre apprentissage de ce sujet important. La seule façon d'apprendre à programmer est de programmer. Je suis convaincu que vous trouverez les exemples relativement faciles, mais utiles et non triviaux. En revanche, si vous n'êtes pas prêt à relever ces défis, la lecture de ce chapitre ne vous sera d'aucune utilité ; autant arrêter de lire maintenant.

5.1. Répétition sans ennui : la boucle For

On nous a souvent appris que les ordinateurs sont les plus performants pour automatiser des tâches répétitives, identiques (ou très similaires). C'est une excellente nouvelle, car cela peut épargner à un être humain une tâche fastidieuse et ennuyeuse, qui peut prendre des siècles. Le meilleur exemple, et le plus courant, est la « boucle for ». Nous allons la découvrir maintenant.

5.1.1. Utiliser Sage pour générer des tableaux. Imaginons que

je veuille générer une liste des 100 premiers carrés parfaits. Le code serait le suivant :

pour i dans la plage (0,100) : j^2 et cela

signifie que i sera pris sur les valeurs 0, 1, 2, 3, . . . , 99 mais pas 100.
De cette façon, la plage (0, x) contiendra toujours x éléments. C'était un exemple simple, mais peu utile. Si je trace un graphique à la main, je souhaiterais connaître la valeur d'une fonction en des points compris entre −10 ≤ x ≤ 10. Supposons que la fonction soit f(x) = x − x. Je saisirais alors for x in range(-10,11): x^3-x
3

C'était intéressant, mais pas crucial, car nous avons appris à créer directement un graphique Sage, à la page 9. Cependant, c'est ainsi que fonctionnent les ordinateurs

Machine Translated by Google

Fonctions graphiques : ils évalueront une fonction en 1 000 points appropriés très proches les uns des autres et traceront 1 000 points, un à chaque emplacement. À l'œil nu, le graphique apparaît très lisse et clair. Il s'agit d'une courbe fluide, alors qu'en réalité, il ne s'agit que d'une collection de points.

Au fait, avez-vous remarqué que la première boucle, listant les carrés parfaits, − x était sur Tout était sur une seule ligne ? Pendant ce temps, la boucle autour ^{3} deux lignes. de x. Bien qu'il soit légal de tout laisser sur une seule ligne, il est considéré comme de bon ton d'utiliser plusieurs lignes. À mesure que nous apprendrons à créer des boucles de plus en plus complexes, vous comprendrez pourquoi. Il est important que chaque sous-routine d'un programme informatique soit lisible. Sinon, il est difficile de réparer ou d'améliorer le programme. Il ne suffit pas qu'un programme informatique « fonctionne ».

Prenons maintenant un exemple financier. Imaginons que je souhaite générer tous les montants finaux possibles pour un placement de 15 000 $ en un seul versement, avec un taux d'intérêt inconnu, mais une durée de 5 ans, capitalisé annuellement. Supposons que je souhaite considérer tous les taux d'intérêt de 1 %, 2 %, 3 %, 4 %, etc., jusqu'à 29 %. La formule de l'intérêt composé est : A = P(1 + i) n , et donc, je coderais

```
pour i dans la plage (1,30) :
    N( 15000*(1+(i/100))^5 )
```

L'indentation est d'ailleurs importante. Utilisez la touche Tabulation pour cela. L'indentation montre quelles commandes doivent être répétées par la boucle for.

Notez que si vous omettez N(), vous obtenez des fractions exactes (nombres rationnels), ce qui est vraiment bizarre. Essayez de supprimer N( ) et voyez ce qui se passe, mais assurez-vous que chaque « (» correspond à «) » après la modification.

5.1.2. Formatage soigné de la sortie

Quoi qu'il en soit, ce code était assez difficile à lire. Alternativement, je peux taper

```
pour i dans la plage (1,30) :
    imprimer i,
    imprimer implique ", imprimer
    N( 15000*(1+(i/100))^5 )
```

Quelques informations générales peuvent être utiles. L'indentation sert à indiquer que les commandes indentées sont subordonnées à la commande for. Cela signifie qu'elles seront répétées. Une commande print affichera à l'écran ce qui la suit, mais la virgule lui indique de ne pas passer à une nouvelle ligne.

Je vous recommande maintenant d'essayer les trois variantes suivantes :

- essayez d'ajouter une virgule à la fin de la troisième instruction d'impression, • essayez-le sans aucune virgule, • essayez-le en supprimant uniquement la virgule après « implique », mais en gardant la virgule après i présente.

Machine Translated by Google

Rendons notre sous-routine plus puissante maintenant. Si je voulais aller à chaque 1/4 de pour cent, je pourrais faire pour

```
i dans la plage (1,120) : imprimer
    N (i/400), imprimer
    implique ", imprimer N
    (15000*(1+(i/400))^5)
```

Ce qui a nécessité trois modifications. Premièrement, le (i/100) dans le grand N() est devenu i/400, pour que les pas soient de 1/400 (un quart de pour cent) au lieu de 1/100 (ou un pour cent). Ensuite, la boucle passe à 120 au lieu de 30, ce qui m'amène à 29,75 %, au lieu de m'arrêter au quart. Enfin, j'ai décidé que l'impression la plus à gauche afficherait le taux réel, plutôt que la valeur de i, de sorte que l'impression i est passée à l'impression N(i/100). L'omission de N() dans cette première impression produit le résultat étrange de Sage essayant de réduire les fractions qui composent le taux d'intérêt à leur plus bas niveau. Essayez par vous-même et vous comprendrez.

Nous pouvons faire en sorte que notre exemple précédent avec x 3−x produise des résultats plus professionnels. Nous pourrions également chercher des résultats.

```
pour x dans la plage
    (-10,11) : imprimer
    « x = », imprimer x,
    imprimer x, imprimer
    signifie y = », imprimer x^3-x
```

À ce stade, notre code devient plutôt « long » dans le sens où nous utilisons un grand nombre de lignes mais ne faisons pas grand-chose.
Il s'avère que nous pouvons économiser de l'espace vertical en combinant les instructions d'impression. Ceci est autorisé à condition de conserver les virgules. Par exemple, essayez le code suivant.

```
pour x dans la plage
    (-10,11) : imprimer « x = », x, signifie y=", x^3-x
```

5.1.3. Listes arbitraires. Parfois,

la liste de valeurs souhaitée est arbitraire. Par exemple, si je voulais évaluer x − x aux valeurs x 2, 5, 7/4, 3/10 et −28/11, je tapérais

```
pour x dans [-2, 5, 7/4, 3/10, -28/11] : imprimer « x »,
    imprimer x,
    imprimer
    signifie y », imprimer x^3-
    x
```

Comme vous pouvez le voir, j'ai simplement remplacé range(-10,11) par la liste [-2, 5, 7/4, 3/10, -28/11]. Ceci est un exemple de la façon dont Sage utilise la construction de liste Python. Nous l'avons déjà vu à plusieurs reprises dans ce livre.

Machine Translated by Google

5.1.4. Boucles avec accumulateurs. Vous pouvez

également donner de la « mémoire » à la boucle en stockant des informations dans une variable. L'utilisation la plus courante consiste à calculer la somme des éléments d'une liste, mais vous pouvez

également l'utiliser pour trouver une valeur minimale, une valeur maximale ou le produit des éléments. Un terme très ancien pour additionner des éléments à l'aide d'une boucle for, datant des premiers temps de l'informatique, est « accumulateur ».
Je trouve que c'est un terme utile, mais pour un informaticien, il aura une consonance ancienne.

Considérez l'extrait de code suivant, qui calculera la somme des entiers compris entre 0 et 1000 (inclus).

```
total = 0
pour i dans la plage (0, 1001) : total=total+i

imprimer le total
```

Boucles inutiles : il serait

inutile et imprudent d'afficher chaque nombre de 1 à 1 000, puis de les additionner. Le code pour ce faire serait le suivant :

```
total = 0 pour i
dans la plage (0, 1001) : imprimer i
        total=total+i

imprimer le total
```

Notez la présence de « print i » entre l'instruction « for » et l'instruction « total = total+i ». Ici, le résultat à afficher est trop volumineux et la liste de nombres est énorme, défilant lentement vers le bas. Il est important d'éviter de telles techniques (surtout si l'on utilise un million ou un milliard de nombres), car cela gaspillerait des ressources informatiques importantes, comme la bande passante et le temps de calcul. Ce gaspillage est particulièrement important dans un système comme Sage, car vous utilisez Internet pour transférer toutes ces valeurs intermédiaires du serveur Sage vers votre navigateur web. Lorsque mes étudiants sont frustrés par la lenteur extrême de leur code, c'est souvent parce qu'ils ont des instructions d'impression superflues qui leur font perdre du temps.

Au fait, avez-vous remarqué que le 0 était inclus, mais que le 1001 était exclu ? Ceci est similaire aux exemples de la section 5.1.1, page 216, où nous avons évalué f(x) = x − x sur le domaine range(-10, 11), où -10 était inclus mais 11 exclu. Python fonctionne ainsi : range(0, 10) s'exécute exactement 10 fois, range(0, 100) 100 fois et range(0, 1000) 1000 fois. À propos, chaque « passage » ou « exécution » d'une boucle for est appelé « itération ».

3

Machine Translated by Google

Calculer cette somme directement :

Un dernier point doit être souligné, du point de vue des mathématiques pures.
Il existe une formule permettant d'additionner tous les nombres d'une progression arithmétique.
Une progression arithmétique est une liste de nombres dont la différence
entre deux nombres consécutifs, n'importe où dans la liste, il y a toujours un écart fixe
constante. Cette constante fixe est appelée la différence commune. Par exemple,
considérer

$$3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, \ldots$$

ou alternativement

$$72, 69, 66, 63, 60, 57, 54, 51, 48, 45, \ldots$$

ainsi que

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, \ldots$$

Comme vous pouvez le voir, ils ont une différence commune de +2, -3 et +1 respectivement. La somme d'une progression arithmétique est donnée par

$$S = \frac{(a + z)(n)}{2}$$

où a est le premier membre de la progression, z est le dernier membre de la progression, n est le nombre de membres et S est la somme finale. Nous pouvons calculer plus directement notre objectif avec

$$S = \frac{(1 + 1000)(1000)}{2} = (1001)(500) = 500, 500$$

obtenir la même réponse mais avec beaucoup moins d'effort de calcul.

5.1.5. Utiliser Sage pour trouver une limite, numériquement

Disons que vous vouliez étudier quelle est la fonction

$$f(x) = e^{1/(8-x)}$$

se produit autour de la valeur x = 8. Vous pourriez alors écrire le code suivant

```
pour i dans la plage (0,9) :
    xtemp = 8 - 10^(-i)
    imprimer N( e^( 1/( 8-xtemp ) qui interroge )  )
```

sur les valeurs de f(x) aux points

$$7, 7.9, 7.99, 7.999, 7.9999, 7.99999, 7.999999, 7.9999999, 7.9999999, 7.9999999$$

Ces numéros ont aussi les noms

8 à 100, 8−10−1, 8−10−2, 8−10−3, 8−10−4, 8−10−5, 8−10−6, 8−10−7, 8−10−8

L'évaluation de f(x) donne la sortie suivante

Machine Translated by Google

2.71828182845905
22026.4657948067
2.68811714181614e43
1.97007111401705e434
8.80681822566292e4342
2.80666336042612e43429
3.03321539680209e434294
6.59223253461844e4342944
1.54997674664843e43429448

qui tend clairement vers l'infini. (Si 1,54 · · ·×1043,429,448 n'est pas « assez proche » de l'infini, alors je ne sais pas ce que c'est !) Ensuite, en changeant 8 − 10−i en 8 + 10−i, nous obtiendrions pour i dans la plage(0,9) : xtemp=8+10^(-

i) print N(e^(1/(8-xtemp) qui
interroge sur les valeurs
de f(x) aux points))

9, 8.1, 8.01, 8.001, 8.0001, 8.00001, 8.000001, 8.0000001, 8.0000001, 8.00000001

qui portent également les noms

8+100, 8+10−1, 8+10−2, 8+10−3, 8+10−4, 8+10−5, 8+10−6, 8+10−7, 8+10−8

ce qui donne le résultat suivant

0,367879441171442
0,000045399929762484 9
3.72007597602084e-44
5.07595889754946e-435
1.13548386531474e-4343
3.56294956530937e-43430
3.29683147808856e-434295
1.51693678089873e-4342945
6.45170969282177e-43429449

qui va clairement vers zéro. (Si 6,45 · · · ×10−43 429 449 n'est pas « assez proche » de zéro, alors je ne sais pas ce que c'est !)

Nous pouvons donc finalement conclure

$$\lim_{x\to 8-} f(x) = \infty, \qquad \text{et aussi} \qquad \lim_{x\to 8+} f(x) = 0$$

deux réponses intéressantes en elles-mêmes. La conclusion finale, cependant, doit être

$$\lim_{x\to 8} f(x) = \text{dne}$$

car il faut dire qu'une limite (en général) n'existe pas si la limite venant de la gauche et la limite venant de la droite ne sont pas égales.

Machine Translated by Google

Mise en garde:

L'exploration de cette limite a demandé un peu de travail, mais pas trop, et a répondu à une question assez difficile. On voit donc que la technique de l'« évaluation numérique des limites » a sa place en mathématiques. Néanmoins, attention : les limites numériques sont terriblement peu fiables !

Par exemple, la somme des réciproques des n premiers nombres premiers tend vers l'infini lorsque n tend vers l'infini. Plus précisément, je parle de la somme

$$\frac{1}{2}+\frac{1}{3}+\frac{1}{5}+\frac{1}{7}+\frac{1}{11}+\frac{1}{13}+\frac{1}{17}+\frac{1}{19}+\frac{1}{23}+\frac{1}{29}+\cdots$$

tend vers l'infini. Ce théorème a une démonstration intéressante, mais difficile, et il ne serait pas agréable de la parcourir maintenant.

Cependant, vous ne seriez jamais en mesure de détecter cette divergence via un ordinateur. En fait, à mesure que n augmente, cette somme croît comme log(log n). (Je simplifie légèrement cette discussion en supprimant quelques détails.) Par exemple, si votre norme pour « suffisamment » de l'infini était 1 000, il faudrait attendre que soit si grand qu'il m'est difficile de décider comment le $e^{e^{1000}}$ Les nombres ont été totalisés. Le concept de $e^{e^{1000}}$ décrire. Même si votre norme pour « suffisamment » de l'infini était 10, il faudrait attendre que $e^{e^{10}}$ nombres soient tabulés, et notez que $e^{e^{10}}$ comporte 9 565 chiffres !

Il va sans dire que si vous essayez de calculer cela avec une boucle for, bien avant que ce tableau ne se termine, le soleil sera déjà devenu une géante rouge et nous serons tous brûlés vifs.

5.1.6. Boucles For et polynômes de Taylor. Vous savez peut-être que, dans les cours de calcul et de niveau supérieur, il est parfois nécessaire d'étudier de nombreuses dérivées d'une fonction, notamment pour calculer un polynôme de Taylor. Certains professeurs de calcul enseignent les polynômes de Taylor dès le début du cours, d'autres très tard ; certains passent même outre ce sujet. Si vous n'avez jamais étudié les polynômes de Taylor, parfois appelés « séries de Taylor », n'hésitez pas à passer à la section suivante, page 224.

La discussion ici est une excursion, sans rapport avec le reste du chapitre.

```
pour i dans la plage (0,10) :
    imprimer i,
    imprimer « la dérivée est », imprimer
    diff(x^3-x,x,i)
```

Comme vous pouvez le voir, cela calcule les 9 premières dérivées de $x^3 - x$, dont la plupart sont nulles. Notez également que la « dérivée 0 » est la fonction elle-même. Il est à noter, au passage, que Sage peut calculer directement la série de Taylor, sans que vous ayez à calculer toutes ces dérivées, en une seule ligne de code. Ceci sera expliqué page 173. Cet exemple est fourni à titre indicatif uniquement.

Machine Translated by Google

Une autre fonction amusante est la transformée de Lorentz de la relativité restreinte :

$$L(v) = \frac{1}{\sqrt{1 - \frac{v^2}{c^2}}}$$

Où v est la vitesse d'un objet et c celle de la lumière. Cette formule décrit comment les objets se déplaçant très rapidement rétrécissent dans leur direction de déplacement, deviennent plus massifs et voient leur temps se ralentir (ou se dilater) pendant eux.

Si nous appliquons l'outil précédent à ce problème, nous le ferions comme suit :

```
f(v,c) = 1/sqrt(1-(v^2/c^2)) pour i dans la
plage(0,10) : print i, print "la
    dérivée
    est", print diff(f,v,i) qui vous indique
    que la dérivée 9 est
893025*v/((-v^2/c^2 + 1)^(11/2)*c^10) +
13097700*v^3/((-v^2/c^2 + 1)^(13/2)*c^12) +
51081030*v^5/((-v^2/c^2 + 1)^(15/2)*c^14) + 72972900*v^7/((-
v^2/c^2 + 1)^(17/2)*c^16) + 34459425*v^9/((-v^2/c^2 +
1)^(19/2)*c^18)
```

un calcul que je ne voudrais pas faire à la main.

5.1.7. Si vous savez déjà programmer. . .

Si vous savez déjà programmer, vous comprendrez que Sage fonctionne comme un langage de programmation. En fait, Sage fonctionne sur le langage Python.

- Si vous connaissez Python, vous pouvez utiliser toutes les commandes Python de votre choix, directement dans Sage, sans problème.
- Ensuite, si vous ne savez programmer dans aucun langage, ne vous en souciez pas pour le moment. Vous n'avez absolument pas besoin de savoir programmer pour utiliser Sage. Presque toutes les tâches ne nécessitent pas de programmation, et celles qui en nécessitent n'en nécessitent généralement pas beaucoup : juste les rudiments que vous apprendrez dans les pages suivantes. Ce chapitre a été écrit pour vous, et vous serez un programmeur assez fonctionnel à la fin (à condition de relever tous les défis).
- Si vous savez programmer, mais pas en Python, sachez que Python est un langage extrêmement facile. Toute personne ayant des connaissances en C, C++, Pascal, Java ou C# peut apprendre Python avec seulement[1] un investissement minime en temps et très peu d'efforts.

---

[1] Mon ami Martin Albrecht, un important développeur de Sage, a un jour qualifié Python de « langage exécutable ». pseudo-code. »

Machine Translated by Google

– Si vous êtes un programmeur expérimenté, vous pouvez en apprendre davantage sur Python sur http://www.diveintopython.org. – Si vous êtes un programmeur intermédiaire, vous pouvez tout aussi bien lire ce chapitre. Vous le lirez probablement beaucoup plus vite qu'un vrai débutant, mais ce n'est pas grave. Au cours de ce chapitre, vous apprendrez beaucoup de Python.

5.2. Écriture de sous-routines

Les programmes informatiques sont souvent divisés en de nombreux petits éléments exécutant des tâches distinctes. Au grand dam des étudiants, le vocabulaire diffère d'un langage à l'autre. Ces fragments de code sont parfois appelés « méthodes » ou « procédures », mais généralement, on les appelle « fonctions ».
Certains langages de programmation très anciens les appelaient « sous-programmes ». Python préfère nettement le terme « fonction ».

Cela nous présente un obstacle verbal majeur dans le calcul mathématique-+ 7x − 12. En mathématiques, le mot fonction signifie quelque chose comme f(x) = x. 2
Comment alors peut-on utiliser le mot fonction pour désigner simultanément f(x) et un petit morceau de code ? Par conséquent, dans ce chapitre, j'utiliserai le mot traditionnel « sous-routine » pour désigner un segment de code effectuant une tâche, tandis que j'utiliserai le mot « fonction » pour désigner quelque chose comme f(x) = + 11x − 12. Mon utilisation du 2 x mot sous-routine ici est plutôt atypique, et la plupart des experts Python seront surpris de le voir ici. C'est un peu démodé et cela sonnerait comme quelqu'un qui dirait « mince zut » ou « c'est génial ». Cependant, je pense que le chapitre serait illisible si j'utilisais le mot fonction pour décrire à la fois des morceaux de code Python et des fonctions mathématiques comme f(x).

Que vous les appeliez méthodes, procédures, sous-programmes, fonctions ou sous-routines, la programmation en général (et la programmation mathématique en particulier) consiste à concevoir ces blocs de construction, puis à les combiner dans un système logiciel plus vaste.

5.2.1. Exemple : Travailler avec des pièces de monnaie.

Nous allons commencer par une sous-routine très simple, qui ferait partie de la programmation d'un distributeur automatique ou d'une caisse enregistreuse. Elle fournira la valeur en dollars d'un certain nombre de pièces de 25 $, 10 $, 5 $ et 1 $. Pour les lecteurs non américains, il s'agit de pièces valant 0,25 $, 0,10 $, 0,05 $ et 0,01 $. Cette tâche est simple, nous permettant de nous concentrer sur Python et non sur les mathématiques sous-jacentes.

Voici le code que nous allons taper :

```
def coinCount( quarters, dimes, nickels, pennies ): """Cette sous-
    routine récupère les données d'une pile de monnaie, dont les quatre paramètres sont le
    nombre de 25 cents, de 10 cents, de 5 cents et de 1 cent. La valeur totale en dollars
    de ces pièces est ensuite indiquée.
```

Machine Translated by Google

Les quantités sont supposées être des nombres naturels. Veuillez ne pas utiliser de nombres complexes en entrée, car l'utilisateur serait horrifié de découvrir qu'une partie de son argent est imaginaire. Total = 25 cents*0,25 + 10 cents*0,10 + 5 cents*0,05 + 1 centime*0,01. Afficher « Cela fait un total de $ », afficher le total.

La première ligne, avec la commande def, signale à Python et Sage que nous sommes sur le point de définir une nouvelle sous-routine. Nous indiquons ensuite immédiatement le nom de la sous-routine, puis la liste des variables nécessaires à sa création. Ici, nous avons des variables pour les pièces de 25, 10, 5 et 10 cents.
Alors qu'en algèbre, les variables ont tendance à être constituées d'une seule lettre, en programmation, nous utilisons des mots pour ne pas avoir à nous souvenir de la signification de chaque variable.
Les deux points sont très importants. Ils indiquent que les commandes suivantes sont subordonnées à l'instruction def. L'indentation indique précisément quelles instructions sont subordonnées.

Ensuite, sur la deuxième ligne, nous avons une formule qui calcule la valeur totale des pièces. Chaque pièce est multipliée par sa valeur en dollars, et le total est placé dans la variable appelée, sans surprise, « total ».
La troisième ligne et la quatrième ligne impriment les montants d'une manière bien formatée et lisible par l'homme, comme nous l'avons appris à la page 217.

Nous pouvons tester notre sous-routine avec 5 quarts, 4 dimes, 3 nickels et 2 pièces de un cent. Nous

saisirions coinCount(5, 4, 3, 2)

et nous obtenons en retour la bonne réponse indiquée ci-dessous.

Cela représente un total de 1,82000000000000

Il est important de se rappeler que l'ordre est important. Considérez les trois appels suivants à cette sous-routine : coinCount(1, 3, 2, 1)

coinCount(1, 1, 2, 3) coinCount(1,
2, 3, 1)

Comme vous pouvez le constater, les trois appels concernent des piles de monnaie différentes, dont les valeurs totales diffèrent. Imaginez qu'une personne utilisant la sous-routine trois jours différents, ayant oublié l'ordre, puisse deviner l'ordre et saisir les trois lignes ci-dessus. Le résultat ci-dessous illustre bien ce point : vous obtenez une mauvaise réponse si vous placez les paramètres dans le mauvais ordre.

Cela représente un total de 0,6600000000000000 Cela
représente un total de 0,480000000000000 Cela représente
un total de 0,610000000000000

Contrairement à la plupart des langages informatiques, Python propose une solution pour ceux qui ont du mal à mémoriser les ordres. En utilisant la notation suivante, nous étiquetons chaque valeur avec la variable à laquelle nous souhaitons l'affecter. Nous obtiendrons la même réponse à chaque fois.

Machine Translated by Google

coinCount(dimes=3, nickels=2, pennies=1, quarters=1) coinCount(quarters=1, pennies=1, nickels=2, dimes=3) coinCount(pennies=1, nickels=2, dimes=3, quarters=1)

Sage et Python répondront aux lignes ci-dessus avec la réponse suivante, indiquant qu'ils ont compris ce que nous espérions dire, à chaque fois.

Cela représente un total de 0,6600000000000000 $ Cela représente un total de 0,6600000000000000 $ Cela représente un total de 0,6600000000000000 $

C'est un très bon service pour les cerveaux qui ont du mal à organiser, qu'ils soient dyslexiques ou peut-être simplement temporairement distraits.
Cependant, cela pose un autre problème : il faut se souvenir de l'orthographe du nom de chaque variable. Comme vous pouvez le voir ci-dessous, j'ai un jour omis le « r » de « quarters », remplaçant « quaters ». N'ayant jamais vu ce mot auparavant, l'ordinateur ne sait pas interpréter mes remarques et affiche donc le message d'erreur suivant.

TypeError : coinCount() a reçu un argument de mot-clé inattendu « quaters »

Une dernière chose avant de continuer. Dans Sage Cloud, sous votre code utilisant coinCount, saisissez « coi » sur une ligne distincte et appuyez sur Tab sans appuyer sur Entrée. Comme vous pouvez le constater, l'une des options qui s'offrent à vous est votre propre sous-routine ! Sage Cloud la considère comme faisant partie intégrante du monde dans lequel vous évoluez actuellement.

5.2.2. Un défi : une caisse enregistreuse

Une tâche amusante pourrait être de simuler une caisse enregistreuse. Supposons qu'un petit stand de nourriture dans une station de métro vende des sodas (1,50 $ chacun), des bananes (0,75 $ chacune), des sandwichs de différents types (tous à 3,75 $ chacun) et des bouteilles d'eau (2 $ chacune).

Écrivez une sous-routine prenant en compte cinq entrées : le nombre de sodas, de bananes, de sandwichs et de bouteilles d'eau, ainsi que le montant remis (par exemple, 20 $ ou 10 $). Votre programme doit indiquer à l'utilisateur la monnaie à rendre, en dollars et en cents.

5.2.3. Un exemple : Conception d'aquariums

Nous allons maintenant écrire une sous-routine adaptée à une entreprise qui construit des aquariums d'exposition sur mesure pour les cabinets dentaires et autres commerces haut de gamme. Les clients passent commande et le coût doit être correctement calculé, en fonction des dimensions des six panneaux qui composent l'aquarium rectangulaire.

Il s'avère que les matériaux particuliers utilisés par l'entreprise ont les coûts suivants :

- Le fond de l'aquarium est en métal et coûte 1,3 centime/pouce carré. • Le dessus de l'aquarium est en plastique et coûte un demi-centime/pouce carré. • Les côtés de l'aquarium sont en verre et coûtent 5 centimes/pouce carré.

Machine Translated by Google


```
coût_supérieur = 0,5
coût_inférieur = 1,3
coût_latéral = 5

def analyzeAquariumDesign( longueur, largeur, hauteur ):
        Ici, les dimensions d'un aquarium sont les données d'entrée. Les dimensions des six
        panneaux qui forment l'aquarium rectangulaire seront affichées à l'écran, ainsi que leurs
        coûts. Les coûts sont calculés en fonction des surfaces de ces panneaux et des variables
        globales top_cost, bottom_cost et side_cost.

        Enfin, le volume et le coût total sont affichés.

        zone_bas_haut = longueur*largeur
        zone_avant_arrière = longueur*hauteur
        zone_gauche_droite = largeur*hauteur

        print "Les panneaux gauche/droite ont une superficie de", left_right_area, print "coût $",
        side_cost*left_right_area*0.01 print "Les panneaux avant/arrière ont une
        superficie de", front_back_area, print "coût $", side_cost*front_back_area*0.01 print "Le panneau
        supérieur a une superficie de", bottom_top_area, print "coût $",
        top_cost*bottom_top_area*0.01 print "Le panneau inférieur a une superficie de",
        bottom_top_area, print "coût $", bottom_cost*bottom_top_area*0.01 print




        imprimer "Le volume total est", N( longueur*largeur*hauteur )

        coût = coût_haut*zone_bas_haut + coût_bas*zone_bas_haut + 2*zone_avant_arrière*coût_latéral
          + 2*zone_gauche_droite*coût_latéral

        imprimer "Le coût total est de $", coût*0,01
```

Figure 1. Code pour calculer le coût des matériaux d'un aquarium

Étant donné la préférence du client quant au volume de l'aquarium, il existe de nombreux choix de dimensions et leurs coûts seront sensiblement différents. Prenons l'exemple d'un client souhaitant un aquarium de 12 000 pouces cubes. (Notez que 12 000 pouces cubes représentent environ 52 gallons, ce qui n'est pas un très grand aquarium.) Nous pouvons calculer les prix suivants :

- Un choix de 20 × 30 × 20 a un coût de 110,80 $.
- Un choix de 40 × 30 × 10 a un coût de 91,60 $.
- Un choix de 60 × 20 × 10 a un coût de 101,60 $.
- Un choix de 80 × 15 × 10 a un coût de 116,60 $.

Ces prix ont été calculés à l'aide de la sous-routine analyzeAquariumDesign, que vous trouverez dans la figure 1. Les commandes suivantes ont été utilisées, une par une, après la définition de la sous-routine.

Machine Translated by Google


```
analyzeAquariumDesign( 20, 30, 20 )
analyzeAquariumDesign( 40, 30, 10 )
analyzeAquariumDesign( 60, 20, 10 )
analyzeAquariumDesign( 80, 15, 10 )
```

Comme vous pouvez le constater, le code présente quelques nouveautés que vous n'avez peut-être pas vues auparavant. Le symbole, qui ressemble à six apostrophes consécutives, est en fait un guillemet triple. Entre ces guillemets, vous pouvez insérer les mots que vous souhaitez pour expliquer la fonction de votre sous-routine à une personne lisant votre code. En général, il est préférable d'utiliser quelques phrases plutôt qu'une seule. Cette description est appelée « docstring » et elle est très importante. Sans une bonne docstring, il est peu probable que quelqu'un d'autre utilise votre code. Cependant, la réutilisation du code est une stratégie extrêmement importante, tant dans l'industrie que dans le monde universitaire. Il serait extrêmement coûteux de « réinventer la roue » chaque fois qu'un ordinateur est nécessaire pour effectuer une tâche. D'ailleurs, lorsque nous avons utilisé l'opérateur « ? » (voir section 1.10, page 47), nous lisions les docstrings écrites par les anciens programmeurs Sage.

Les variables « top cost », « bottom cost » et « side cost » servent à stocker les coûts de chaque partie de l'aquarium. Si j'avais utilisé le chiffre 0,5 à chaque emplacement où apparaît la variable « top cost », le code aurait été moins lisible. De plus, le prix pourrait changer. Si le prix passe à 0,6 au lieu de 0,5, je pourrais peut-être le modifier seulement à quelques endroits, sans le modifier partout. Les calculs seraient alors irréalisables. Cependant, en utilisant une variable pour stocker la valeur en haut du programme, je n'ai besoin de la modifier qu'à un seul endroit. Faciliter la mise à jour et la maintenance du code est également essentiel, tant dans le monde industriel qu'universitaire.

Nous avons déjà vu des variables utilisées de cette manière (voir section 1.6, page 32). Cependant, il existe une terminologie utile. Lorsqu'une variable est créée et utilisée exclusivement à l'intérieur d'une sous-routine, on parle de variable locale. Cependant, ces variables sont créées en dehors de toute sous-routine et sont donc visibles par toutes les sous-routines ; on les appelle des variables globales. Dans de nombreux types de programmation, les variables globales sont déconseillées, mais en informatique scientifique ou mathématique, elles semblent plus courantes.

Il est considéré comme un bon style de programmation de placer toutes les variables globales en haut du programme, afin de les retrouver facilement. Cependant, ce n'est pas absolument nécessaire. Python et Sage ne s'opposeront pas à ce que vous enfreigniez cette règle de l'étiquette de codage.

Une ligne sur deux du code de la figure 1 correspondait soit à une instruction d'impression, soit à une équation mathématique. En informatique scientifique, ces outils très simples permettent souvent d'effectuer des tâches complexes.

Au fait, cette sous-routine analyzeAquariumDesign constituera la base d'une page web interactive avec Sage. Création de pages web interactives

Machine Translated by Google

peut être très amusant, et c'est le sujet du chapitre 6. Nous explorerons l'utilisation de cette sous-routine particulière dans la section 6.4 à la page 288.

5.2.4. Un défi : un silo cylindrique Ce problème est une

légère variante d'un problème de calcul classique. Une entreprise de fournitures agricoles fabrique des silos bon marché. Chaque silo est un cylindre (peut-être de rayon r et de hauteur h) et il est coiffé d'un hémisphère, également de même rayon r. L'objectif est de produire un silo avec un volume spécifié, mais en utilisant le coût minimum de métal. Le problème de calcul classique est de trouver le silo d'un volume spécifié et d'une surface minimale.

Cependant, pour plus de réalisme, nous allons affirmer que le métal cylindrique coûte 25 cents le pied carré, car il est facile à fabriquer, et que le métal sphérique coûte 1,50 $ le pied carré, car il est plus difficile à fabriquer. Nous espérons minimiser les coûts, ce qui revient à minimiser la surface (mais pas à la réduire).

Votre défi est d'écrire une sous-routine qui prend r et h comme entrées.
devrait calculer et imprimer • la

surface de la partie cylindrique du silo • la surface de la partie sphérique du silo • le coût de chacune d'elles et le coût total • le volume du silo

Naturellement, vous voudrez consulter Internet pour obtenir les formules appropriées pour le volume et la surface d'un cylindre ou d'un hémisphère.
Souvent, lorsque je suis engagé dans un calcul scientifique, je dois rechercher quelques formules mineures.

Grâce à Internet, il est aujourd'hui beaucoup plus facile de chercher des informations qu'à l'époque où j'ai appris la programmation en 1988. Bien sûr, je connais ces formules et je pourrais les saisir ici. Cependant, l'exercice est plus réaliste lorsque je vous force à les chercher vous-même sur Internet. C'est une meilleure simulation du travail en entreprise : vous aurez mémorisé les concepts et les procédures mathématiques, mais vous aurez depuis longtemps oublié certains détails mineurs.

5.2.5. Combinaison de boucles et de sous-routines Parfois,

en particulier lors de l'apprentissage des fonctions, il est judicieux d'évaluer une fonction aux nombres naturels : 0, 1, 2, 3, 4, 5, 6, . . . , jusqu'à une certaine valeur limite.

La sous-routine suivante nous permettra de faire exactement cela. Nous pouvons choisir la fonction de notre choix, ainsi que le nombre de termes que nous voulons dans la séquence formée. def functionToSequence(f,

howmany): """Cette sous-routine prend une fonction

comme première entrée, puis l'évalue aux premiers nombres naturels « howmany ».

Par exemple, si howmany est 4, alors f est évalué

Machine Translated by Google


```
    à 0, 1, 2, 3."""" pour k
  dans la plage(0, combien) : imprimer
        f(k),

  imprimer
```

Nous pouvons également définir les fonctions suivantes pour tester notre code :

```
a(x) = 3*x + 5 b(x)
= 2 - xc(x) = 7 +
10*xd(x) =
8*exp(-0,1*x) f(x) = 5*3^xg(x)
= (1/8)*2^xh(x) =
(0,125)*2^x
```

Nous pouvons maintenant tester notre code avec la fonction sequence(a, 20) et observer le résultat. De même, vous pouvez utiliser b(x) ou c(x) en remplaçant a par b ou c, et ainsi de suite. Vous remarquerez des différences de comportement intéressantes entre g(x) et h(x), qui diffèrent de Sage, même si pour un mathématicien, g(x) et h(x) sont la même fonction. Essayez par vous-même ; je ne vous gâcherai pas la surprise.

Ce que je trouve vraiment intéressant avec Sage, c'est que vous pouvez taper du code comme function_to_sequence(3*x, 20) et Sage comprendra ce que vous voulez dire. Il n'est pas nécessaire de définir une fonction distincte comme j(x) = 3x pour utiliser 3x comme fonction.

5.2.6. Autre défi : totaliser une séquence. Je vous invite maintenant à revenir à la fonction de sous-routine pour séquencer, vue dans la sous-section précédente. Modifiez cette sous-routine pour qu'elle totalise également les nombres imprimés. Ce total devrait s'afficher sur une ligne distincte, après l'affichage de la séquence elle-même.

Testez maintenant votre nouvelle sous-routine avec la fonction r(x) = 1/(x + 1,0)4 et observez le résultat. Un mathématicien dirait que vous calculez les sommes des inverses des puissances quatrièmes des entiers positifs.

Quel est le total obtenu ? Pour les grandes valeurs de howmany, comment se compare-t-il à l'approximation décimale du nombre π 4/90 ? Si vous connaissez la fonction zêta de Riemann2, notée ζ(x), vous comprendrez que nous venons de calculer ζ(4). Sa définition est la suivante :

$$\zeta(x) = \sum_{j=1}^{\infty} \frac{1}{j^x}$$

2Nommé en l'honneur de Georg Friedrich Bernhard Riemann (–). Il est le « Riemann » de la géométrie de Riemann (la courbure de l'espace) et de l'hypothèse de Riemann. Il a également beaucoup contribué à l'étude des nombres premiers.

Machine Translated by Google

où j parcourt les entiers positifs et x peut être réel, complexe ou imaginaire.

5.3. Boucles et méthode de Newton

Cette section est longue, mais extrêmement importante. Nous allons développer un morceau de code pour quelque chose d'utile mathématiquement : trouver les racines d'une fonction. Au fur et à mesure, nous explorerons différentes approches pour y parvenir. En résumé, nous commencerons par un code de qualité, puis nous ajouterons des fonctionnalités, une par une, jusqu'à obtenir un excellent code. Au cours de ce processus, nous découvrirons de nombreuses fonctionnalités des langages de programmation, notamment la construction « if-then-else », les boucles « while », les paramètres optionnels et bien d'autres.

5.3.1. Qu'est-ce que la méthode de Newton ?

Les sections suivantes de ce livre présenteront quelques techniques de programmation utiles et extrêmement courantes. Afin de proposer une application mathématique authentique et intéressante, nous examinerons la méthode de Newton, un célèbre algorithme permettant de trouver les racines de fonctions, ou, de manière équivalente, les solutions numériques d'équations. J'ai rédigé cette section en partant du principe que vous, lecteur, n'avez jamais vu la méthode de Newton3 auparavant. Si vous avez déjà vu la méthode de Newton, par exemple dans un cours de calcul, vous pouvez la parcourir rapidement ou la sauter et passer à la section 5.3.2, page 233. L'idée générale est de générer une sous-routine fonctionnant de manière similaire à « trouver la racine » ; ce faisant, nous apprendrons beaucoup de mathématiques et comprendrons mieux le fonctionnement de « trouver la racine ».

Certaines équations peuvent être résolues grâce à des techniques algèbres. D'autres peuvent l'être grâce à des astuces créatives, comme les identités trigonométriques. Cependant, de nombreuses équations ne peuvent tout simplement pas être résolues symboliquement. Par exemple, supposons que je cherche une solution à

$$\text{péché } x = 1 - x$$

Il n'existe aucun moyen de résoudre cette équation en utilisant des techniques ordinaires d'algèbre. Cependant, grâce à cette procédure appelée méthode de Newton, nous pouvons réaliser une série d'approximations numériques successives. Ces approximations sont faciles à calculer et tendent à devenir assez précises après seulement quelques étapes. Premièrement, nous observons qu'une solution à l'équation ci-dessus serait une racine de la fonction suivante :

$$f(x) = 1 - x - \sin x$$

Nous commençons par la célèbre formule suivante : f(xold)

$$x_{\text{nouveau}} = x_{\text{ancien}} - \frac{f(x_{\text{old}})}{}$$

3Comme vous pouvez le deviner, la méthode de Newton doit son nom à Isaac Newton (—).

Machine Translated by Google

C'est le cœur de la méthode de Newton. Nous utiliserons comme estimation initiale x = 0. Ensuite, nous avons

$$f(x) = 1 - x - \sin x \text{ et donc } f(x) = -1 - \cos x$$

que nous pouvons utiliser comme suit :

• Avec x = 0, –
On a f(0) = 1 et f (0) = −2.
– Cela signifie que f /f = −1/2.
– Ainsi, la prochaine valeur de x sera 0 − (−1/2) = 1/2. • Avec x = 1/2, – Nous avons
f(1/2) = 0,0205744 · · · et f (1/2) = −1,87758 · · · – Ainsi f /f = −0,0109579 · · · ·
– La prochaine valeur de x est 0,5 − ·
(−0,0109579 · · ·) = 0,510957 · · · • Avec x = 0,510957, – Maintenant ·
f(0,510957) =
0,00000307602 . . .
– Alors que f (0,510957) = −1,87227 · · · – Par ·
conséquent, f /f = −0,0000164293 · · · – Cela signifie ·
que la prochaine valeur de x sera

$$0{,}510957 - (-0{,}0000164293) = 0{,}510973 \cdots \quad \cdot$$

• Remarque, si vous suivez avec une calculatrice portable, assurez-vous
assurez-vous qu'il est configuré pour utiliser des « radians ».

Si nous devions discuter de ces résultats dans un courrier électronique ou une publication mathématique, nous dirions que x = 0 est la première approximation, x = 1/2 est la deuxième approximation, x = 0,510957 est la troisième approximation et x = 0,510973 est la quatrième approximation.

Ce n'était pas un effort considérable, mais le calcul n'était pas non plus ultra-rapide ; nous sommes donc peut-être curieux de savoir comment nous avons fait. Nous allons maintenant insérer notre quatrième approximation dans les deux côtés de l'équation initiale et voir à quel point nous avons réussi à nous rapprocher.

• La valeur de sin 0,510973 = 0,489026196 · · · • La .
valeur de 1 − 0,510973 = 0,489027000 · · · .

Nous avons atteint ce degré de précision phénoménal en seulement trois utilisations de la formule de Newton. Lorsque l'ordinateur appliquera la méthode de Newton, nous constaterons que nous pouvons obtenir une précision au milliardième près, relativement rapidement.

En fait, j'ai arrondi au sixième chiffre significatif à plusieurs reprises ; la quatrième approximation aurait donc peut-être été encore meilleure réalisée sur ordinateur. En utilisant le code que nous allons développer dans les pages suivantes, j'ai atteint la cinquième approximation. Cette cinquième solution approximative s'avère être

$$x = 0{,}510973429388569 \cdots$$

dont les côtés gauche et droit de notre équation sont égaux à 15 décimales près, avec une précision au quadrillionième près.

Machine Translated by Google

Résumé : Cet

exercice montre que la méthode de Newton possède deux propriétés importantes. Premièrement, une fois f(x) et f(x) ainsi que l'estimation initiale données, la méthode est très simple à mettre en œuvre. Deuxièmement, elle n'est pas particulièrement intéressante pour un humain, car elle est répétitive. Après tout, une fois l'estimation initiale et les deux fonctions f(x) et f(x) connues, on ne fait qu'utiliser plusieurs fois certaines formules. Ces deux caractéristiques font de la méthode de Newton une solution idéale pour un ordinateur, et c'est d'autant plus vrai lorsque l'on réalise que la précision de l'ordinateur est supérieure à celle que nous pourrions utiliser avec notre calculatrice portable.

Méthode tabulaire de Newton :

Petite remarque : si l'on utilise une calculatrice de poche plutôt qu'un ordinateur, je trouve beaucoup plus simple de créer un petit tableau pour organiser les étapes intermédiaires. Dans l'exemple ci-dessus, ce serait :

x	f(x)	f (x)	rapport
0	1	−2	−1/2
1/2	0,0205744	−1,87758	−0,0109579
0,510957	0,0000307602	−1,87227	−0,0000164293
0,510973			

C'est en forgeant qu'on devient forgeron :

Maintenant, si vous voulez vraiment être certain de maîtriser parfaitement cette technique de calcul, vous pouvez essayer un exemple avec un papier et un crayon, à l'aide d'une calculatrice. Essayez plutôt de résoudre

$$\text{et}^{x} = 10x$$

La condition initiale étant x = 4. Après cela, vous aurez probablement hâte de déléguer cette tâche à un ordinateur. Votre quatrième approximation, après trois utilisations de la formule de Newton, devrait être d'environ 3,57715206 · · ·, avec des variations selon la marque de calculatrice utilisée et sa gestion interne des arrondis. En effet, nous pouvons vérifier

$$\text{et}^{3.57715206395730} = 35{,}7715206395730 \cdots$$

avec n'importe quelle calculatrice portable.

5.3.2. Méthode de Newton avec une boucle For. La figure 2

présente un bon exemple d'une première tentative d'implémentation de la méthode de Newton dans Sage. Nous allons maintenant analyser ce code, ligne par ligne.

La première ligne informe Sage que nous sommes sur le point de définir une sous-routine. Elle s'appellera « méthode newton » et comportera deux paramètres : f et x old. Il est primordial que les lignes suivantes soient indentées.

Le degré d'indentation montre quelles commandes sont subordonnées à quelles

Machine Translated by Google


```
def newton_method( f, x_old ):
        """Cette sous-routine utilise la méthode de Newton pour trouver une racine
        de f, et la condition initiale est x_old. Il y aura 10 itérations.

      f_prime(x) = diff( f, x )

      pour j dans la plage (0, 10) :
            imprimer "itération=", j imprimer
            "x_old=", x_old imprimer "f(x)=",
            f(x_old) imprimer "f'(x)=",
            f_prime(x_old)

            rapport = f(x_old) / f_prime(x_old)

            imprimer "ratio=", ratio

            x_nouveau = x_ancien - rapport

            imprimer "x_new=", x_new
            imprimer

            x_ancien = N(x_nouveau)
```

Figure 2. Méthode de Newton dans Sage, version 1

autres. Puisque la méthode de Newton requiert la connaissance de la dérivée de la fonction dont on cherche la racine, il n'est pas surprenant de voir la ligne suivante définir f prime(x). Notez qu'on ne peut pas utiliser une apostrophe à la place du premier, comme dans f(x), car l'apostrophe a déjà une signification en Python. Ce point mineur a été mentionné pour la première fois page 49.

Nous arrivons maintenant à une boucle qui s'exécutera 10 fois, comme indiqué par la commande for. Comme vous pouvez le constater, la boucle est principalement composée d'instructions d'impression, destinées à informer l'utilisateur de ce qui se passe. La syntaxe de la commande d'impression vous est probablement familière, mais vous pouvez également consulter la section 5.1.2, page 217.

Les trois lignes de la boucle qui ne sont pas des instructions d'impression sont trois affectations. La première définit le « ratio », comme indiqué dans la sous-section précédente. La seconde définit xnew à partir de xold selon les règles de la méthode de Newton.

Enfin, nous voyons la ligne x old =_N(x new) qui peut être déroutante. Fondamentalement, pendant l'exécution de la boucle for identifiée par j = 3, la valeur de xold sera la valeur que xnew avait pendant l'exécution de la boucle for identifiée par j = 2. Cette commande fournit le lien entre une exécution de la boucle et l'exécution suivante de la boucle.

Machine Translated by Google

La seule surprise serait l'utilisation de la commande N( ), que nous avions auparavant utilisée uniquement pour convertir une forme exacte (algébrique) d'un certain nombre, peut-être difficile à comprendre, sous une forme à virgule flottante ou décimale, ce qui est plus facile à comprendre mais qui n'est nécessairement qu'une simple approximation.

L'utilisation de N( ) ici est de forcer Sage à calculer approximativement et Pas exactement. Cela peut paraître surprenant, car l'un des points forts de Sage est sa capacités de calcul exact. Nous verrons cependant un exemple dans le prochain sous-section qui nous dévoilera pourquoi j'ai fait ce choix.

5.3.3. Tester le code

Nos exemples de test seront les trois fonctions suivantes.

$$a(x) = x \qquad 5 + x + 1$$

$$b(x) = x \qquad x \quad - 5$$

$$c(x) = x \qquad 3 - 2x + 1$$

Prenez un moment pour vous assurer que vous avez correctement tapé ces trois fonctions a(x), b(x) et c(x) dans Sage. Pour a(x), nous pourrions choisir une valeur initiale condition de 0,5 et donc nous taperions

```
méthode_newton( a(x), 0,5 )
```

et obtenir ainsi une réponse finale de x = −0,754877 · · · raisonnable. (C'est , ce qui semble la valeur finale de xold. Il convient de noter que f(xold) se termine par une valeur finale de 8,32667 · · · × 10−17, ce qui est vraiment très proche de zéro. Nous pouvons donc conclure que l'exécution de notre sous-routine est un succès. Nous avons calculé avec succès la racine d'un polynôme quintique.

Cependant, il y a encore place à l'amélioration, comme on peut le constater en faisant défiler retour à l'itération 5. Nous voyons que f(xold) = −7,62168 · · · × 10−14 là et c'est sûrement « assez proche de zéro ». Par conséquent, nous avons gaspillé du temps de calcul. temps. Ce n'est pas important lorsqu'on analyse les racines d'un polynôme unique, Il pourrait s'agir d'une différence d'un quart de seconde ou d'une demi-seconde. Cependant, si notre projet devient un élément important d'un programme plus vaste (peut-être à exécuter un million de fois) puis perdre la moitié du temps de calcul serait extrêmement imprudent. Nous remédierons à ce gaspillage dans quelques pages. à partir de maintenant.

Nous pouvons maintenant essayer b(x). Puisque 22 = 4 et 33 = 27, et que 4 < 5 < 27, nous anticiper que la solution de x ^{x} = 5 doit être trouvé entre 2 et 3. En utilisant la condition initiale de 2, on obtient xold = 2,12937 · · · ce qui semble, raisonnable. Nous pouvons tester la réponse fournie en demandant la valeur de

```
2.12937248276016^2.12937248276016
```

en utilisant le copier-coller (plutôt que la saisie). Nous apprenons que la réponse est

```
5.00000000000003
```

qui est extrêmement proche de 5. Nous constatons à nouveau le gaspillage de notre investissement initial code car nous obtenons une excellente réponse dès l'itération 4.

Machine Translated by Google

Si nous utilisions x = 3 comme condition initiale, nous obtiendrions la même valeur finale de xold. Elle correspond à notre réponse pour x = 2 aux 14 décimales indiquées, ce qui représente une précision remarquable. Il est intéressant de s'interroger sur la précision impliquée par la concordance à 15 chiffres significatifs.

Imaginez que vous calculiez le poids d'une automobile d'environ une tonne. Une précision à 15 chiffres significatifs impliquerait une précision au nanogramme près, soit un milliardième du poids d'un trombone (un gramme). Il convient également de noter que nous n'avons pas atteint 5 chiffres exactement. Des méthodes comme la méthode de Newton produisent des approximations, souvent excellentes, mais pas des réponses exactes.

Cette comparaison entre le comportement de b(x) avec x = 2 et x = 3 vise à vous indiquer que les hypothèses initiales doivent simplement être « raisonnables » et ne sont pas forcément très bonnes. Cependant, une hypothèse initiale « folle », comme x = 10, produit un échec. En particulier, nous constatons que la valeur finale de f(xold) = 1,30099 × 106, ce qui est loin d'être nul. Cet échec ne devrait toutefois pas être surprenant, car nous cherchons des valeurs de x telles que = 5. Je suis sûr que nous sommes d'accord : 1010 n'est pas approximativement égal à 5.
x x

Maintenant que nous sommes sûrs que tout fonctionne correctement, nous pouvons conclure notre vérification avec c(x). Avec une condition initiale de x = −1, nous obtenons que xold = −1,61803 · · · est la réponse finale fournie, et f(xold) est dite nulle. Ce n'est pas tout à fait exact, mais la longue chaîne de zéros indique que f(xold) est si proche de zéro que la meilleure estimation de sa valeur, représentée (sous forme de virgule flottante) par Sage, est nulle.

Si nous voulions être précis, nous pourrions utiliser le copier-coller pour demander à

Sage c(-1,61803398874989) qui

donne 2,84217 · · · × 10−14. Avec une condition initiale de x = 0, nous obtenons une valeur finale de xold = 0,618033 · · · . Cette fois, la valeur de f(xold) = 5,55111 · · · × 10−17. C'est une excellente précision, et nous pouvons être satisfaits que notre code fonctionne pour ces exemples.

Il est intéressant de noter, dans ce cas c(x), que les conditions initiales x = −1 et x = 0 convergent toutes deux vers de bonnes réponses, mais vers des réponses différentes. Cela se produit fréquemment lorsque f a plusieurs racines.
Dans certaines conditions, la méthode de Newton convergera vers une racine, souvent, mais pas toujours, la racine la plus proche.

Pour le plaisir, nous pouvons reconsidérer notre problème d'ouverture, à partir de la section 5.3.1 Page 231. Nous voulions résoudre sin x = 1 − x, pour x.

```
u(x) = 1 - x - sin(x)
newton_method( u, 0 )
```

Nous pouvons voir que dans la cinquième itération, nous avons une réponse précise à

10−16, et cela correspond à la solution que nous avions trouvée manuellement.

```
v(x) = e^x - 10*x
méthode_newton( v, 4 )
```

c'est un autre exemple que nous avons vu plus tôt.

Machine Translated by Google

Une dernière remarque est qu'avec ces trois dernières fonctions, nos calculs ont encore une fois été inutiles.

- À l'itération 5, nous avons vu f autour d'un trillionième pour c(x). • À l'itération 5, nous avons vu f environ 1/9 d'un quadrillionième pour u(x). • À l'itération 5, nous avons vu f dit être nul pour v(x). • Tous ces éléments sont sûrement « plus que suffisants en termes de précision » pour toute application du monde réel, nous devrions donc chercher un moyen de « nous arrêter tôt ».

5.3.4. Représentations numériques et exactes Cette sous-section explorera a(x) un peu plus en détail et expliquera pourquoi nous avons utilisé la commande N( ) d'une manière nouvelle et étrange.

Essayons de transformer x ancien = N(x nouveau) en x ancien = x nouveau et voyons ce qui se passe. Si nous essayons la condition initiale x = 0,5, nous ne constatons aucune différence. Cependant, si nous essayons x = 1, nous constatons clairement une différence. Essayez maintenant et voyez ce qui se passe ; c'est amusant.

Même à l'itération 5, nous voyons la réponse

$$\frac{-1218838545901833215165300598610929129090358 6 \text{ maintenir} =}{1614617024225309571191199408413995135509180 3}$$

Ce qui est comique. Les itérations suivantes affichent encore plus de chiffres. Bien sûr, ce n'est pas ce que nous avions en tête ; nous devons donc forcer Sage à calculer numériquement. Pour ce faire, nous reconvertirions x ancien = x nouveau en x ancien = N(x nouveau). Cela signifie qu'au début de l'itération 2, nous sommes assurés de travailler numériquement.

Pourquoi cet étrange phénomène d'extrême précision s'est-il produit avec la condition initiale x = 1, mais pas avec la condition initiale x = 0,5 ? La raison tient à la façon dont Sage interprète les nombres. Le nombre 0,5 est déjà un nombre décimal et Sage suppose donc qu'il s'agit d'une simple approximation numérique. Cependant, le nombre 1/2, au lieu de 0,5, est supposé exact. Si nous utilisons 1/2 à la place de 0,5 avec x ancien = x nouveau au lieu de x ancien = N(x nouveau), nous obtiendrons un nombre de chiffres comique pour x ancien dès l'itération 4. − 5, et les calculs exacts produisent des conséquences encore plus catastrophiques et humoristiques.

Il est amusant de noter qu'en utilisant b(x) = x x

5.3.5. Travailler avec des paramètres optionnels et obligatoires. La première chose à corriger serait peut-être les calculs inutiles. Je parle spécifiquement des situations où nous avons réduit f(xold) à moins d'un milliardième, mais où l'algorithme continue car nous avons besoin de 10 itérations. Cependant, ce changement aura un avantage secondaire, qui vous sera révélé dans quelques paragraphes.

La solution la plus judicieuse est d'essayer de créer une condition mathématique permettant à la sous-routine de déterminer d'elle-même qu'il est temps de s'arrêter. Cependant, nous allons d'abord explorer une méthode moins sophistiquée, beaucoup plus simple à expliquer et à mettre en œuvre. Nous allons

Machine Translated by Google

Pour permettre à l'appelant de notre sous-routine de spécifier le nombre d'itérations à effectuer. Cela nécessite bien sûr un nouveau paramètre pour la sous-routine, qui lui indiquera le nombre d'itérations à effectuer.

Pour ce faire, nous allons effectuer deux petits changements

- def newton method(f, x old): devient
  def newton method(f, x old, max iterate):
- pour j dans la plage(0, 10) : devient pour j dans
  la plage(0, itération max) :

Le premier changement établit le troisième paramètre et le nomme max iterate, car c'est le nombre maximum de fois que la boucle for doit itérer.
Le deuxième changement fait que la valeur de max iterate remplace 10 dans la boucle for. En d'autres termes, la boucle for itérera le nombre maximal d'itérations au lieu de 10 fois précisément.

Cependant, nous pouvons faire encore mieux. Les modifications apportées ci-dessus modifieraient nos cas de test.

- la méthode Newton(b, 2) devient la méthode Newton(b, 2, 10) • la méthode
  Newton(b, 10) devient la méthode Newton(b, 10, 10)

Ces modifications ne sont pas complexes et se limitent à fournir une valeur numérique au paramètre « max iterate ». Cependant, ce n'est pas optimal pour plusieurs raisons. Premièrement, un programmeur utilisant notre ancienne sous-routine depuis longtemps dans ses propres programmes constaterait que son code est totalement inopérant. En effet, les appels à notre sous-routine dans ses programmes comporteraient deux paramètres au lieu de trois, et la sous-routine en nécessite désormais trois. Par conséquent, son code ne fonctionnerait pas correctement et il vous en voudrait. Deuxièmement, un nouveau programmeur inexpérimenté ne devrait pas s'embarrasser de connaissances approfondies sur la méthode de Newton pour pouvoir utiliser notre sous-routine. Nous souhaitons que notre code soit accessible au plus grand nombre. Il serait préférable de définir une valeur « standard » pour « max iterate », puis de permettre aux autres programmeurs de « surcharger » notre choix standard. Le mécanisme pour cela en Python est un paramètre « facultatif ».

Pour implémenter un paramètre facultatif, nous effectuerions cette modification :

- L'ancien code def newton method(f, x old): • ou
  alternativement def newton method(f, x old, max iterate): • devient à la place
  def newton method(f, x old, max iterate=10):

Ce changement a pour effet que si une valeur d'itération maximale est spécifiée, elle sera utilisée. En revanche, si aucune valeur n'est spécifiée, la valeur par défaut sera utilisée.

Ce problème, appelé « rétrocompatibilité », est extrêmement grave dans tout projet logiciel impliquant plusieurs programmeurs. Un étudiant souhaitant occuper un emploi, quel qu'il soit, doit prendre conscience qu'il ne sera pas le seul à programmer un ordinateur dans sa nouvelle entreprise. Il doit donc réfléchir attentivement à cette question. « Casser » le code d'autrui en modifiant quelque chose sans autorisation est un moyen extrêmement efficace de se faire licencier.

Machine Translated by Google

Une valeur de 10 sera utilisée à la place. Effectuez cette modification, puis prenez un moment pour tester chacun des appels suivants de notre sous-routine, un par un. newton_method(b,
2) newton_method(b, 2,
max_iterate=5) newton_method(b, 10) newton_method(b,
10, max_iterate=5)
newton_method(b, 10, max_iterate=100)

Comme vous pouvez le voir dans le dernier cas, étant donné l'horrible estimation initiale de x = 10 pour la fonction b(x) = x − 5, la méthode de Newton a finalement convergé.
Tandis que vous avez la longue sortie de 100 itérations à l'écran, il est très intéressant de voir où les performances passent de « mauvaises » à « bonnes ».
En particulier, nous voyons que l'itération 23 a f(xold) = 0,318042 · · · ce qui n'est pas vraiment une très bonne approximation de zéro, alors que l'itération 26 a f(xold) à environ 19 trillionièmes, et l'itération 27 a f(xold) à moins d'un quadrillionième.

Lorsque j'ai appris à programmer à la fin des années 1980 et au début des années 1990, il était courant, lors de l'implémentation d'un algorithme itératif comme la méthode de Newton, de demander à l'utilisateur d'appuyer sur la barre d'espace entre chaque itération, afin de donner un aperçu de l'évolution de l'algorithme. Ce n'est plus ainsi que les logiciels sont écrits aujourd'hui, mais cela constitue néanmoins une bonne expérience de pensée. Imaginez avoir appuyé 23 fois sur la barre d'espace pour signaler « prochaine itération », en espérant résoudre le problème x = 5, et obtenir une valeur de f(xold) autour de 0,318042. À quel point seriez-vous frustré ? Peut-être même très frustré. La plupart d'entre nous auraient abandonné bien avant la 23e itération, et certains seraient tentés d'abandonner dès la cinquième itération.

Cependant, une personne très patiente appuierait sur le bouton 3 ou 4 fois de plus et obtiendrait une excellente réponse à la 26e ou 27e itération. Nous pouvons être certains maintenant que « où s'arrêter » n'est pas évident, et qu'une certaine patience est nécessaire.

À propos, il convient de noter que les paramètres optionnels sont rares dans la plupart des langages de programmation les plus importants. Python met cette fonctionnalité à votre disposition, mais elle est absente de nombreux langages.

5.3.6. Renvoyer une valeur et imbriquer des sous-routines.

Travailler en calcul scientifique ou en programmation numérique consiste généralement à écrire une série de sous-routines dont chacune effectue une tâche importante, mais qui accomplissent principalement leur objectif en appelant d'autres sous-routines – parfois celles que vous avez écrites, mais le plus souvent celles que d'autres ont écrites. Dans cette sous-section, nous allons expérimenter cela en utilisant notre sous-routine de méthode de Newton pour calculer quelque chose qui ne serait pas vraiment calculable sans quelque chose de similaire à la méthode de Newton ou à l'une de ses méthodes rivales.

À titre d'exemple, jetez un œil aux extraits de code suivants : 13 + sin(3*pi + 1/5)

Machine Translated by Google


```
log( 169 ) / 2 16 -
sqrt( 9 - x^2 )
```

Chacun de ces extraits est logique et utilisable, car une sous-routine, à savoir sin, log ou sqrt, génère un nombre utilisable avec les autres nombres selon les règles de l'arithmétique. De même, notre sous-routine de méthode Newton doit également générer un nombre. Il est donc logique que ce nombre soit notre meilleure approximation, soit la valeur finale de xold et xnew à la fin de l'exécution de la sous-routine. Notez qu'à la fin de l'exécution, xold = xnew sera toujours le cas ; il n'est donc pas nécessaire de se demander lequel choisir.

Pour réaliser cette sortie d'un nombre, nous ajoutons la ligne

```
retourner x_old
```

jusqu'à la fin de la sous-routine. Cette commande Python indique à Sage d'utiliser la valeur finale de xold comme sortie de notre sous-routine.

Ce changement, ainsi que les changements de la sous-section précédente, donnent lieu au code que l'on trouve dans la figure 3. Les lignes qui ont changé depuis la version 1 ont l'annotation # changed ou # new à côté d'elles.

Le symbole # est très important. Tout ce qui suit le symbole # sur une ligne de code Python est totalement ignoré. Vous pouvez donc l'utiliser pour de petites annotations comme « # modifié » ou « # nouveau ».

Vous pouvez également utiliser # pour des commentaires plus étoffés, comme un résumé sur une seule ligne pour indiquer au lecteur du code ce que quelques lignes de code sont censées faire. Souvent, si une formule provient d'un article ou d'un chapitre de manuel, c'est une bonne idée de noter ce fait avec cette catégorie de commentaire « rapide ».

Au début de cette sous-section, je vous avais promis que nous utiliserions la méthode de Newton pour accomplir une tâche de calcul légitime. La tâche que j'ai choisie est le calcul de l'autologarithme. Étant donné un nombre réel positif y, nous souhaitons calculer un x tel que $x^x = y$. Notre exemple précédent, $b(x) = x - 5$, réaliserait cette tâche dans le cas spécifique de y = 5, mais nous allons maintenant écrire du code qui devrait fonctionner pour tout nombre réel positif y.

Vous pouvez probablement deviner qu'au lieu d'utiliser $b(x) = x^x - 5$, nous utiliserons $b(x) = x^x - y$. Cependant, ce n'est pas le seul changement, car nous devons faire une estimation initiale. Il est facile pour un être humain d'utiliser le raisonnement pour calculer une estimation initiale raisonnable, mais un ordinateur n'a pas la puissance de raisonnement.
Nous avons besoin d'une règle simple et rapide pour produire rapidement une estimation initiale, même si celle-ci n'est pas très bonne. Nous aurons besoin d'une condition initiale assez petite, même pour des valeurs élevées de y. Considérons 66 = 46, 656, et constatons que même avec un y à 5 chiffres, nous avons une estimation initiale faible de x = 6.

Finalement, j'ai choisi de prendre une estimation initiale égale à 1 + log10 y, où log10 représente le logarithme décimal de y. Ce n'est pas si terrible, car

Machine Translated by Google


```
def newton_method( f, x_old, max_iterate=10 ): # modifié """Une implémentation de la
        méthode de Newton, pour trouver une racine de f(x) étant donné une estimation initiale
        de x. Une valeur par défaut de 10 itérations sera effectuée, sauf si le paramètre
        facultatif max_iterate est défini différemment."""  # modifié


        f_prime(x) = diff( f, x )

        pour j dans la plage (0, max_iterate) :                    # changé
                imprimer "itération=", j imprimer
                "x_old=", x_old imprimer "f(x)=",
                f(x_old) imprimer "f'(x)=",
                f_prime(x_old)

                rapport = f(x_old) / f_prime(x_old)

                imprimer "ratio=", ratio

                x_nouveau = x_ancien - rapport

                imprimer "x_new=", x_new
                imprimer

                x_ancien = N(x_nouveau)

        retourner x_old # nouveau
```

Figure 3. Méthode de Newton dans Sage, version 2

les exemples suivants illustrent

1 + log10 46, 656 = 5,66890 · · · au lieu de 6 1 + log10 27 = 2,43136 · · · au lieu de 3 1 + log10 4 = 1,301029 · · · au lieu de 2

Je dois être honnête ici et vous dire que j'ai dû expérimenter plusieurs fois jusqu'à ce que je trouve une fonction de sélection initiale qui me semblait appropriée.
Enfin et surtout, nous devons décider comment utiliser les paramètres facultatifs.
J'ai décidé de définir max iterate=100 afin d'être sûr d'obtenir une réponse correcte. L'utilisation de 100 itérations présente un inconvénient : le résultat sera extrêmement long et l'utilisateur devra faire défiler la page. Cependant, nous verrons comment atténuer ce problème grâce au contrôle de la verbosité à la page 244.

L'extrait de code final se trouve dans la Figure 4. Notez que ce code nécessite absolument la version 2 ou supérieure de notre sous-routine de méthode newton, car la version 1 n'avait pas d'instruction de retour.

Nous pouvons maintenant tester ce code en tapant

Machine Translated by Google


```
def selfLogarithm( y ): """Cette
        sous-routine renvoie x tel que x^x = y, où y est un nombre réel positif."""


        g(x) = x^x - y
        estimation = 1 + log( y, 10 )

        réponse = newton_method( g, guess, max_iterate=100 )

        # ce qui suit est simplement un code de test test = réponse^réponse

        imprimer la réponse, "^", réponse, "=", test
        imprimer "Souhaité : ", y

        répondre
```

Figure 4. Code pour calculer l'auto-logarithme, en utilisant la méthode de Newton (version 2 ou supérieure).

```
selfLogarithm( 17 )
```

5.3.7. Un défi : trouver des tangentes parallèles. L'objectif de ce défi est de trouver une valeur de x notée xp, étant donné deux fonctions a(x) et b(x), telle que les tangentes des deux fonctions à x = xp soient parallèles. Cela devrait être fait en appelant notre implémentation de la méthode de Newton. Les paramètres seraient a(x) et b(x), et la sortie devrait être la valeur de xp.

Pour un défi encore plus difficile, faites en sorte que la sous-routine génère à la fois des tangentes lignes sous la forme y = mx + b. (Attention : c'est un défi difficile !)

Voici quelques exemples :

- Pour a(x) = x ^{2} + 2 et b(x) = x – ^{3} − x, une réponse est xp = 1.
  - Observez que a (1) = 2 et b (1) = 2 – Nous obtenons une ligne tangente pour a(x) définie par y = 2x + 1.
  - On obtient une tangente pour b(x) définie par y = 2x − 2.
  - Pour un graphique de cette situation, voir le tracé à gauche dans la figure 5.
- Pour a(x) = x 2+2 et b(x) = x 3−x, l'autre réponse est xp = −1/3.
  - Observez que a (−1/3) = −2/3 et b (−1/3) = −2/3 – Nous obtenons une tangente pour a(x) définie par y = (−2/3)x + 17/9.
  - On obtient une tangente pour b(x) définie par y = (−2/3)x+74/999.
  - Pour un graphique de celui-ci, voir le tracé à droite dans la figure 5.
- Votre sous-routine n'identifierait qu'une seule de ces deux solutions, à moins que vous ne fassiez quelque chose de très sophistiqué.

Machine Translated by Google

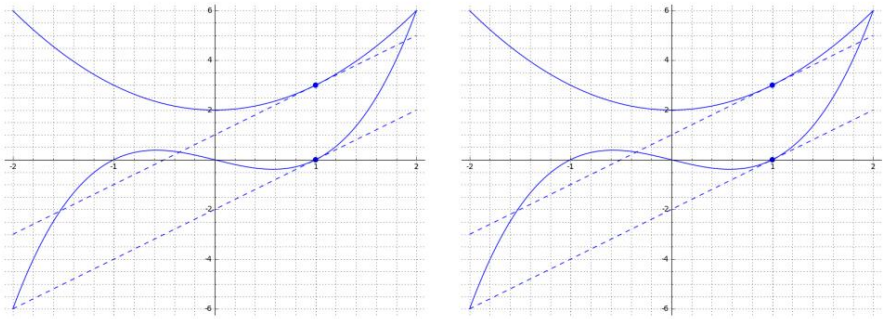


Figure 5. Les deux cas de tangentes parallèles, de la section 5.3.7

Pour un autre exemple, considérons $a(x) = \sqrt{x}$ et $b(x) = x$ la solution ^{2}. L'unique est xp = 3 1/16. Observez que

$$a(x) = b(x) = \sqrt[3]{\frac{1}{2}} \approx 0{,}793700525 \cdots$$

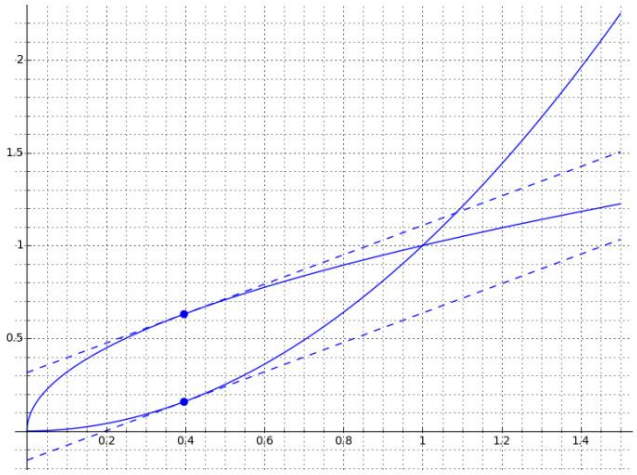
impliquant des lignes tangentes de

$$y = (0{,}793700525 \cdots)x + 0{,}314980262 \cdots$$

pour a(x) et pour b(x)

$$y = (0{,}793700525 \cdots)x - 0{,}157490130 \cdots$$

Cette dernière situation peut être résumée par l'intrigue suivante :



5.3.8. Un défi : la méthode de Halley. Edmond Halley

(–) était un astronome et physicien qui a donné son nom à la comète de Halley. Cependant, il était également ami d'Isaac Newton et a contribué à élargir le domaine d'étude qui est devenu le calcul infinitésimal.

Machine Translated by Google

Une découverte mineure de Halley, presque totalement oubliée, est une méthode de recherche de racines très similaire aux travaux de Newton avec Joseph Rapheson. Par exemple, jusqu'en 2007, aucun article ne traitait de cette méthode sur Wikipédia, où l'on trouve de nombreuses formules obscures. Quoi qu'il en soit, voici la formule sous-jacente :

$$\text{xnouveau} = \text{xancien} - \frac{f(\text{xold})f\,(\text{xold})}{[f\,(\text{xold})]2 - \frac{1}{2}f(\text{xold})f\,(\text{xold})}$$

Comme vous pouvez le constater, cette méthode est plus complexe et nécessite la connaissance de la dérivée seconde. Cependant, elle présente une excellente convergence lorsqu'elle s'approche d'une racine r de f(x), à condition que f(r) et f(r) soient tous deux égaux à 0. De plus, f(x) doit exister, ce qui exclut certaines fonctions. Dans les situations où f(x) et f(x) peuvent être calculés facilement, la méthode de Halley peut être un algorithme de recherche de racines extrêmement efficace.

Votre défi consiste à modifier notre code afin qu'il utilise la méthode de Halley plutôt que celle de Newton. J'ai cependant une suggestion. Commencez par placer f(x), f(x) et f(x) dans des « variables temporaires ». Par exemple, a = f(xold) et de même pour b et c contenant les dérivées première et seconde. La formule détaillée de Halley ci-dessus sera alors beaucoup plus facile à saisir.

5.4. Introduction au contrôle de flux

Nous allons étendre nos capacités dans cette section, tout en programmant en Python et Sage, pour inclure des structures qui contrôlent le flux d'un programme.
La plus courante est, de loin, la construction « si-alors ». Nous en verrons plusieurs exemples, ainsi que la manière d'arrêter une sous-routine prématurément (par exemple, si elle a terminé ses calculs plus tôt que prévu) et d'afficher un message d'erreur personnalisé en cas de problème.

5.4.1. Contrôle de la verbosité : le

contrôle de la verbosité consiste à limiter la quantité de sortie obtenue par une sous-routine. Nous développons actuellement la sous-routine de la méthode Newton ; nous souhaitons donc obtenir la sortie, afin de comprendre ce qui se passe réellement. Cependant, le code de notre méthode Newton pourrait un jour faire partie d'un programme plus vaste. Toute cette impression détruira les performances de l'algorithme, car toute opération impliquant des « E/S » (entrée ou sortie) est toujours beaucoup plus lente que le calcul.

Le plan consiste donc généralement à regrouper toutes les instructions d'impression au même endroit et à les « protéger » avec une instruction if.
Il s'agit de notre première rencontre avec la construction if, dont l'utilisation est très simple. Nous allons la combiner avec un nouveau paramètre appelé « verbose ». Comme le paramètre max iterate, le paramètre verbose sera facultatif.

Machine Translated by Google


```
def newton_method( f, x_old, max_iterate = 10, verbose=False ):
    # changé

    Implémentation de la méthode de Newton pour trouver la racine de f(x) à partir d'une
    estimation initiale de x. 10 itérations par défaut seront effectuées, sauf si le paramètre
    optionnel max_iterate est défini sur une autre valeur. Définissez verbose = True
    pour afficher toutes les étapes intermédiaires. # modifié



    f_prime(x) = diff( f, x )

    pour j dans la plage (0, max_iterate) :
        ratio = f(x_old) / f_prime(x_old) x_new = x_old - ratio          # déplacé
                                                  # déplacé

        si (verbose==True) : imprimer          # nouveau
                    "itération=", j imprimer "x_old=",
                    x_old imprimer "f(x)=", f(x_old)
                    imprimer "f'(x)=", f_prime(x_old)
                    imprimer "ratio=", ratio imprimer "x_new=",
                    x_new imprimer



        x_ancien = N(x_nouveau)

    retourner x_old
```

Figure 6. Méthode de Newton dans Sage, version 3

Si le paramètre verbose est défini sur True, l'impression sera effectuée ; s'il ne l'est pas, l'impression sera annulée. Le code est présenté dans la figure 5.4.1. Les lignes modifiées (depuis la version 2, présentée dans la figure 3) sont accompagnées de l'annotation # changed, et celles déplacées sont accompagnées de l'annotation # moved. Les nouvelles lignes sont accompagnées de l'annotation # new.

Même si cela ressemble à un grand nombre de changements, il ne s'agit pas vraiment d'un changement majeur. différence.

- J'ai ajouté le paramètre optionnel « verbose ». • J'ai déplacé toutes les instructions d'impression au même endroit. Leur emplacement est un peu complexe. Elles doivent être suffisamment avancées pour garantir que ratio et x new sont bien calculés. Cependant, si j'avais placé les instructions d'impression après x old = N(x new), la distinction entre x old et x new aurait été perdue.

Machine Translated by Google

• J'ai ajouté la commande if (verbose==True) : • Toutes les commandes d'impression ont été indentées d'un niveau.

L'indentation est très importante. Par exemple :

• Les instructions d'impression sont très indentées, ce qui montre qu'elles sont subordonnées à la fois à l'instruction if et à l'instruction for.

Cela signifie que l'instruction if sera exécutée à chaque itération de la boucle for, et les instructions print ne seront exécutées que si le paramètre verbose est True.

• L'instruction x old = N(x new) est moins indentée, ce qui montre qu'elle est subordonnée à l'instruction for mais pas à l'instruction if. En d'autres termes, la condition de verbose étant True ou False n'a aucune conséquence sur cette ligne, qui sera exécutée dans les deux cas. D'autre part, la ligne est subordonnée à la boucle for. Par conséquent, chaque itération de la boucle for verra cette ligne calculée. • L'instruction return x old a très peu d'intention. Cela signifie qu'elle n'est subordonnée ni à la boucle for ni à l'instruction if. Cependant, elle est toujours subordonnée à la def, ce qui signifie qu'elle fait bien partie de notre sous-routine. Elle ne sera exécutée qu'une seule fois, et non une fois pour chaque cycle de la boucle for.

Vous pouvez tester ce code avec l'une des commandes de test suivantes :

```
newton_method( b, 2 )
newton_method( b, 2, verbose=True ) newton_method( b,
2, max_iterate=5 ) newton_method( b, 2, max_iterate=5,
verbose=True ) newton_method( b, 2, verbose=True, max_iterate=5 )
```

Comme le montrent les deux dernières lignes de code de test, les paramètres optionnels peuvent être spécifiés dans n'importe quel ordre. Cependant, dans la ligne def de toute sous-routine, les paramètres obligatoires doivent être listés en premier, suivis des paramètres optionnels ; il est impossible de mélanger les deux. C'est une règle héritée de Python par Sage.

Nous avons un petit détail technique à aborder maintenant. Dans l'instruction if, nous voyons deux signes égal, ou ==. Jusqu'ici, nous n'en avons utilisé qu'un. L'essentiel est que pour tester l'égalité, il faut utiliser deux signes égal, mais pour provoquer l'égalité, il faut utiliser un seul signe égal. Par exemple, pour forcer la verbosité à True, nous saisirions verbose=True dans la sous-routine. Cependant, dans l'instruction if, nous devons vérifier l'égalité, et non la forcer ; nous utilisons donc deux signes égal avec if (verbose==True): pour y parvenir.

La présence d'un paramètre de verbosité optionnel est une caractéristique de mon style de programmation. Bien sûr, la plupart des programmeurs utilisent cette astuce à un moment ou à un autre, mais pour ma part, presque toutes les sous-routines que j'écris possèdent cette option. Cela rend le débogage beaucoup plus simple et c'est une bonne habitude de programmation.

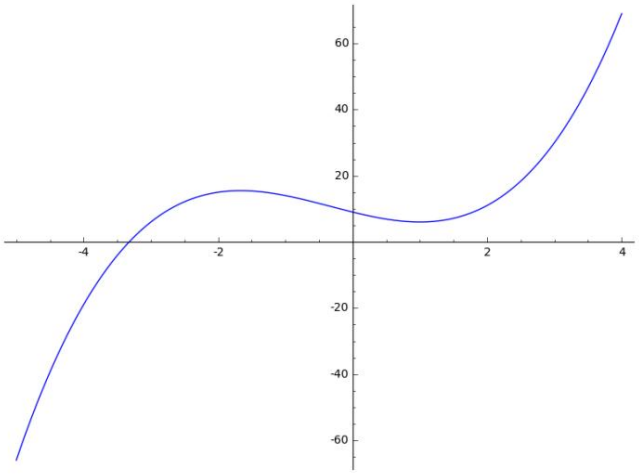
Machine Translated by Google

La notion de contrôle de la verbosité est courante en calcul numérique. Cette idée apparaît parfois dans l'humour des analystes numériques. Un jour, lors d'une réunion d'un petit comité départemental, je discutais sans cesse d'un point relativement insignifiant, et un de mes collègues s'est frustré. Il a dit en plaisantant : « Greg : Assez ! Verbeux, c'est faux ! »

5.4.2. Interlude théorique : quand la méthode de Newton devient folle Considérons la fonction suivante, qui après tout n'est qu'un polynôme cubique, et donc pas particulièrement intimidante.

$$p(x) = x^3 - 5x^2 + 9 + x$$

Même son tracé n'est pas particulièrement inhabituel, comme vous pouvez le voir ci-dessous (montrant −5 < x < 4).



Cependant, observez ce que nous obtenons en testant la condition initiale x = 2 avec la méthode de Newton et p(x). Le résultat est un message d'erreur, reproduit à la figure 7. Comme vous pouvez le constater, le problème est que, lors de l'itération 2, nous constatons que f(x) = 0.

Vous pourriez penser qu'il s'agit d'un exemple isolé, mais vous auriez tort. Voici trois autres exemples :

- $q(x) = x^3 + x^2 + x^2 - 56x + 155$ avec la condition initiale x = 3.
- $r(x) = x^3 - x - 85x + 323$ avec la condition initiale x = 3.
- $s(x) = x^3 - 8x + 15$ avec la condition initiale x = 1.

Le problème principal est que la méthode de Newton exige une division par f(x), et dans ces exemples, nous voyons que f(x) devient nul. Comme il est interdit de diviser par zéro, la sous-routine plante. (L'argot pour « diviser par zéro » est « exploser ».) Les exemples suivants sont des polynômes qui explosent à l'itération 1 et ne survivent même pas à l'itération 2. p(x) = x^2 +

8*x - 9 newton_method(p(x),
-4)

Machine Translated by Google


```
itération = 0
x_old= 2
f(x)= 11
f'(x)= 11
rapport = 1
x_new= 1

itération = 1
x_old= 1,00000000000000 f(x)=
6,00000000000000 f'(x)=
0,000000000000000

---------------------------------------------------------------------------
ZeroDivisionError                                         Traceback (dernier appel le plus récent)
<ipython-input-1-8a48587f6c88> dans <module>() renvoie x_old #
     28         modifié
     29
---> 30 newton_method( p, Integer(2), max_iterate=Integer(20))

<ipython-input-1-8a48587f6c88> dans newton_method(f, x_old, max_iterate) print "f'(x)=", f_prime(x_old)
     15
     16
---> 17                 rapport = f(x_old) / f_prime(x_old)
     18
     19                 imprimer "ratio=", ratio

/home/sageserver/sage/local/lib/python2.7/site-packages/sage/structure/element.so dans sage.structure.element.RingElement.__div__
(sage/structure/element.c:15197)()

/home/sageserver/sage/local/lib/python2.7/site-packages/sage/symbolic/expression.so dans sage.symbolic.expression.Expression._div_
(sage/symbolic/expression.cpp:15716)()

ZeroDivisionError : division symbolique par zéro
```

Figure 7. Le résultat de la méthode de Newton (version 2)
Quand il plante parce qu'il essaie de diviser par zéro

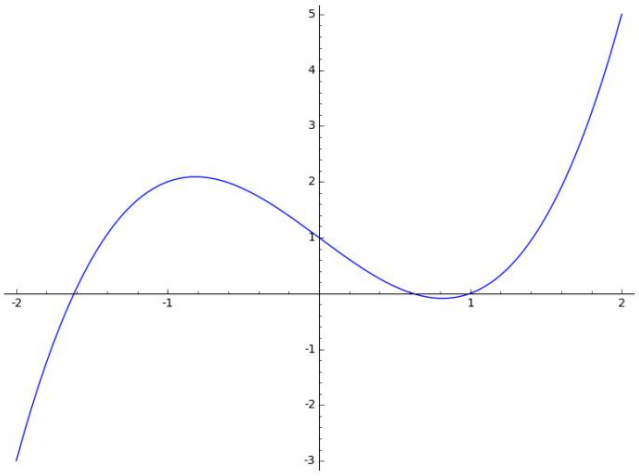
```
q(x) = x^4 - 2*x^2
méthode_newton( q(x), -1 )
méthode_newton( q(x), 0 )
méthode_newton( q(x), 1 )
```

En réalité, il est relativement rare d'atteindre exactement zéro. D'autres difficultés peuvent survenir lorsque f(x) est proche de zéro. Prenons l'exemple précédent de c(x), un polynôme cubique peu intimidant.

$$c(x) = x^3 - 2x + 1$$

Même son tracé n'est pas particulièrement inhabituel, comme vous pouvez le voir ci-dessous (montrant $-2 < x < 2$).

Machine Translated by Google



Par conséquent, nous pouvons imaginer que nous avons une fonction agréable et bien conçue, jusqu'à ce que nous

tapions ce qui suit : newton_method(c(x), -0.436485)

ou mieux encore :

méthode_newton(c(x), -0,436485285968722)

Sur mon ordinateur, le deuxième exemple ne progresse pas avant Iteration 79. C'est parce que f (x) passe extrêmement près de zéro.

La question est maintenant : que faire de tout cela ? Or, ces exemples représentent des choix de condition initiale extrêmement malchanceux. Si vous choisissez plutôt une condition initiale légèrement meilleure (en traçant la fonction, par exemple), vous constaterez que chaque exemple de cette sous-section converge rapidement et sans incident.

Par conséquent, les bonnes implémentations signalent une erreur si f(x) devient nul ou proche de zéro, permettant ainsi à l'utilisateur de recommencer avec un nouveau choix de condition initiale. Nous le ferons nous-mêmes très prochainement, à la section 5.5.1, page 253.

5.4.3. Arrêt prématuré de la méthode de Newton. Nous

avons constaté à plusieurs reprises que le calcul se poursuit longtemps après qu'une réponse relativement précise a été obtenue. C'est un gaspillage, surtout si notre sous-programme risque d'être appelé des millions de fois dans un programme plus volumineux.
Par exemple, les procédures de minimisation multivariée impliquent souvent un calcul complet de la méthode de Newton pour chacune de leurs itérations, alors qu'un million d'itérations du solveur multivarié ne serait pas très inhabituel.

Par conséquent, nous devrions établir une limite. Si f(x) est inférieur à une valeur donnée, par exemple un milliardième, nous devrions arrêter le calcul et afficher la réponse finale. Nous pourrions prendre un moment pour décider si nous souhaitons que cette valeur soit prédéfinie ou définie par l'utilisateur. Une valeur prédéfinie est préférable pour les utilisateurs qui ne sont pas très familiers avec la méthode de Newton.

Machine Translated by Google

Il serait préférable pour les utilisateurs expérimentés de le définir. L'utilisation d'un paramètre optionnel permet d'obtenir le meilleur des deux mondes. L'utilisateur débutant n'a pas à se soucier du paramètre optionnel (ni même à en être informé), mais l'expert peut le remplacer si nécessaire. De plus, comme la valeur est peu susceptible d'être remplacée trop souvent, l'utilisation d'un paramètre optionnel évite d'avoir trop d'éléments entre parenthèses à chaque appel de la sous-routine.

Le nom traditionnel d'une telle valeur limite est « epsilon ». Cette lettre grecque est souvent utilisée pour représenter une valeur « assez proche » en mathématiques computationnelles. La figure 5.4.3 présente le nouveau code, ou version 4, de la méthode de Newton. Les nouvelles lignes clés sont : if (abs(f(x_old)) < epsilon): return x_old

La première ligne indique (en Python) ce qu'un mathématicien écrirait « if |f(x)| < then » et la deuxième ligne, l'instruction return, indique à la sous-routine de renvoyer la réponse finale. Cette instruction return indique implicitement que la sous-routine a terminé son calcul et que rien d'autre ne sera exécuté jusqu'à son prochain appel. Il n'est donc pas rare qu'une sous-routine comporte plusieurs instructions return, mais dès qu'une d'entre elles est atteinte, le calcul s'arrête.

Quelques réflexions sur les valeurs absolues : La façon

d'écrire la valeur absolue dans Sage est d'utiliser la commande abs, comme nous l'avons vu pour la première fois à la page 10. Il est intéressant de noter pourquoi les signes de valeur absolue sont utilisés mathématiquement.

Imaginons que nous ayons plutôt posé la condition d'arrêt f(x) < au lieu de |f(x)| < . Considérons f(x) = 9 − x et supposons qu'un utilisateur confus utilise la condition initiale de x = 100. Nous aurions alors f(100) = −91. Est-ce que ?
est-ce un bon point d'arrêt ? Est-ce que x = 100 est une racine approximative de f(x) = 9−x ? Bien sûr que non ! Cependant, −91 < 10−9 est vrai. Pour éviter cela, nous avons besoin des signes de valeur absolue pour pouvoir écrire |−91| < 10−9 ce qui est bien sûr faux, comme souhaité.

Si vous suivez un cours d'analyse réelle, vous verrez peut-être de nombreux autres exemples où « si |f(x)| < k » est utilisé comme abréviation de −k < f(x) < k.
C'est une astuce courante dans l'analyse mathématique.

Test de résiliation anticipée :

Nous pouvons voir que ce nouveau code fonctionne réellement en tapant

```
b(x) = x^x - 5
newton_method( b(x), 2, verbose=True )
```

qui s'arrête très tôt. Après l'itération 3, xnew est affiché comme sortie finale, car f(xnew) s'est avéré inférieur à epsilon, défini comme son
valeur par défaut d'un milliardième.

Machine Translated by Google


```
def newton_method( f, x_old, max_iterate = 10,
        verbose = False, epsilon = 10^(-9) ): # modifié """Une implémentation de la
    méthode de Newton, pour trouver une racine de f(x) étant donné une estimation initiale
    pour x. Une valeur par défaut de 10 itérations sera effectuée à moins que le
    paramètre facultatif max_iterate ne soit défini sur autre chose. Définissez
    verbose = True pour voir toutes les étapes intermédiaires.

    L'algorithme s'arrêtera prématurément si |f(x)| < epsilon."""  # modifié


    f_prime(x) = diff( f, x )

    pour j dans la plage (0, max_iterate) :
        si (abs(f(x_old)) < epsilon) : renvoie x_old # nouveau          # nouveau


        ratio = f(x_old) / f_prime(x_old) x_new = x_old - ratio


        si (verbose==True) : imprimer
            "itération=", j imprimer "x_old=",
            x_old imprimer "f(x)=", f(x_old)
            imprimer "f'(x)=", f_prime(x_old)
            imprimer "ratio=", ratio imprimer "x_new=",
            x_new imprimer



        x_ancien = N(x_nouveau)

    retourner x_old
```

Figure 8. Méthode de Newton dans Sage, version 4

Bien sûr, certains utilisateurs pourraient exiger une plus grande précision. Dans ce cas, le code b(x) = x^x - 5
newton_method(b(x), 2, epsilon=10^(-15), verbose=True) montre comment atteindre une précision au quadrillionième. Curieusement, une seule itération supplémentaire est nécessaire.

En général, il est judicieux de permettre à l'utilisateur de contrôler des détails spécifiques, comme le critère de « suffisant ». Par exemple, une personne estimant le déficit budgétaire d'un pays se contentera probablement d'une précision au dollar ou au centime près, et ne voudra pas perdre de temps à calculer pour obtenir une précision au milliardième de dollar près.

Machine Translated by Google

5.4.4. La liste des comparateurs

Lors de la rédaction d'une instruction « if », il existe six constructions courantes. Elles sont : énumérés ci-dessous dans leur sens mathématique normal ainsi que le Sage correct ou du code Python pour eux.

Mathématiques	Sage ou Python
si k < 6 alors si	si (k<6) :
k > 6 alors si k	si (k>6) :
= 6 alors si k ≤	si (k==6) :
6 alors si k ≥ 6	si (k<=6) :
alors si k = 6	si (k>=6) :
alors	si (k!=6) :

Notez que, même s'il me semble plus agréable d'écrire si (k=>6) que d'écrire si (k>=6), Python n'acceptera pas si (k=>6). La syntaxe correcte est if (k>=6). à la place. La façon dont je me souviens de ce détail particulier est de me rappeler qu'il il est courant de dire « est supérieur ou égal à » mais il est rare de dire « est égal ou supérieur à. »

De plus, nous devons veiller à ne pas oublier d'utiliser == (deux signes égal) au lieu de = (un signe égal) dans toute instruction if, comme nous l'avons appris dans la section 5.4.1 à la page 246.

5.4.5. Un défi : combien de nombres premiers sont inférieurs à x ?

Nous travaillons avec la méthode de Newton depuis un certain temps, donc je pense que c'est il serait intéressant d'envisager un défi qui n'a rien à voir avec la théorie de Newton. méthode. Considérez la question suivante :

Pour tout entier x > 1, combien de nombres premiers trouve-t-on entre 1 et x ? Plus formellement, combien y a-t-il de nombres premiers p tels que 1 ≤ p ≤ x ?

Voici un indice : vous pouvez créer une boucle for qui compte à partir de 1, 2, 3, . . . , x−1, x. Ensuite, pour chacun de ces entiers z, on pourrait se demander si Sage est premier(z) qui renverra soit Vrai, soit Faux, selon le cas. En utilisant une instruction if, vous pouvez soit incrémenter un compteur en cours d'exécution, soit non, selon que vous obtenez un Réponse vrai ou faux.

Maintenant, pour un défi supplémentaire (une fois que vous avez terminé ce qui précède), faites un paramètre optionnel les répertorie, normalement défini sur False. Cependant, si l'utilisateur le définit sur True, il doit imprimer tous les nombres premiers qu'il découvre en comptant de 1 à x.

5.5. Autres concepts de contrôle de flux

Dans cette section, nous explorerons l'utilisation de la commande raise pour les situations inattendues. situations et un contrôle plus nuancé basé sur ce qu'on appelle la construction « si-alors-sinon ».

Machine Translated by Google

5.5.1. Déclencher une « exception »

Dans tous les types de programmes, il est nécessaire de gérer des situations inhabituelles qui représentent un problème. Il peut s'agir d'une division par zéro (comme nous l'avons vu à la section 5.4.2, page 247), d'un nombre négatif alors qu'on ne s'attendait pas à un, ou d'une multitude d'autres situations. Le terme formel pour ce type de situations est « exceptions ». Dans le cas de la méthode de Newton, nous ne souhaitons pas que f(x) devienne nul, ni qu'il s'approche de zéro.

La commande Python raise permet d'annoncer qu'une exception doit être générée suite à la détection d'une condition inhabituelle. Elle permet également de générer un message d'erreur lisible par l'utilisateur, qui peut aider l'utilisateur à comprendre le problème. D'ailleurs, dans d'autres langages informatiques comme Java, « lever » une exception est parfois appelé « lancer ». Le code qui lèvera une exception si f(x) = 0 lors de l'exécution de la méthode de Newton est présenté dans la figure 5.5.1.

Par exemple, le code

```
c(x) = x^3 - 2*x + 1
newton_method( c(x), -0.436485285968722, verbose=True, max_iterate=20)
```

produira la sortie suivante donnée dans la Figure 10. L'indice « Veuillez essayer une autre condition initiale » dans la dernière ligne de la sortie amènerait l'utilisateur obéissant à taper quelque chose de similaire à `c(x) = x^3 - 2*x + 1 newton_method( c(x), -0.4, verbose=True, max_iterate= 20 )` et alors Sage renvoie joyeusement la bonne solution, à savoir 0.618033. . . .

De nombreux autres nombres auraient pu être utilisés à la place de -0,4 comme condition initiale.

Le principal changement dans le code pour activer cette fonctionnalité est la ligne

```
raise RuntimeError('f_prime(x) got too small! '+ 'Please try another initial condition.')
```

qui notifie à Python et Sage qu'une exception s'est produite et que tous les calculs doivent être arrêtés. Elle informe également Sage et Python que l'exception est de la catégorie RuntimeError, et fournit un message d'erreur clair et compréhensible. Nous ne distinguerons pas les différents types d'exceptions dans ce livre, mais des ouvrages plus avancés sur Python pourraient le faire.

Notez également comment nous avons utilisé le signe + pour créer un message d'erreur plus long que celui qui peut tenir sur une seule ligne. Il est interdit de diviser une citation, comme notre message d'erreur, sur deux lignes. Nous avons donc écrit la citation en deux parties, chacune tenant sur une seule ligne, puis utilisé le signe plus pour indiquer à Sage qu'il s'agit en réalité d'un seul long message d'erreur. Vous pouvez voir le résultat généré dans la figure 10. Comme toujours, la dernière ligne d'un message d'erreur dans Sage est la plus importante.

Utiliser la commande raise pour signaler une erreur majeure présente de nombreux avantages, plutôt que de simplement arrêter le programme. Imaginons que, dans un programme volumineux, une séquence de sous-routines appelle un solveur multivarié sophistiqué, qui appellerait notre méthode Newton.

Machine Translated by Google


```
def newton_method( f, x_old, max_iterate = 10,
        verbose = False, epsilon = 10^(-9), hazard = 10^(-4) ): # modifié


    Implémentation de la méthode de Newton pour trouver la racine de f(x) à partir d'une
    estimation initiale de x. 10 itérations par défaut seront effectuées, sauf si le paramètre
    facultatif max_iterate est défini sur une autre valeur. Définissez verbose = True
    pour afficher toutes les étapes intermédiaires.

    L'algorithme s'arrêtera prématurément si |f(x)| < epsilon. Une exception sera levée si |
    f'(x)|<hazard.""" # modifié


    f_prime(x) = diff( f, x )

    pour j dans la plage (0, max_iterate) :
        si (abs(f(x_old)) < epsilon) : renvoie x_old


        si (abs(f_prime(x_old)) < hazard): # nouvelle augmentation RuntimeError('f_prime(x)
            est devenu trop petit ! '+ 'Veuillez essayer une autre condition initiale.')

                                # nouveau

        ratio = f(x_old) / f_prime(x_old) x_new = x_old - ratio


        si (verbose==True) : imprimer
            "itération=", j imprimer "x_old=",
            x_old imprimer "f(x)=", f(x_old)
            imprimer "f'(x)=", f_prime(x_old)
            imprimer "ratio=", ratio imprimer "x_new=",
            x_new imprimer



        x_ancien = N(x_nouveau)

    retourner x_old
```

Figure 9. Méthode de Newton dans Sage, version 5

Machine Translated by Google


```
Erreur dans les lignes 30-30
Traceback (dernier appel le plus récent) :
    Fichier « /projects/13762024-df4f-4043-8eca-8b21ee6f05e4/.sagemathcloud/
     sage_server.py », ligne 671, dans l'exécution
        exec compile(block+'\n', '', 'single') dans l'espace de noms, locals
    Fichier "", ligne 1, dans <module>
    Fichier "", ligne 14, dans newton_method
Erreur d'exécution : f_prime(x) est trop petit ! Veuillez essayer une autre condition initiale.
```

Figure 10. La sortie générée lorsque la commande raise est utilisée.

Sous-routine. Si nous renvoyons simplement une valeur (par exemple 0) et affichons un message d'erreur, nous ne « notifions » pas les couches supérieures qu'un problème s'est produit, et le programme entier pourrait alors se comporter de manière très étrange. Cependant, la commande raise interromprait le calcul et empêcherait le programme principal de se comporter de manière inattendue. Deuxièmement, il existe une construction try:except en Python, que nous ne pouvons pas explorer ici. Grâce à cette construction, vous pouvez réagir et gérer l'erreur, et éventuellement réessayer la sous-routine. Troisièmement, la commande raise permet d'afficher un message d'erreur clair à l'utilisateur.

Comme vous l'avez sans doute constaté, de nombreux messages d'erreur dans Sage sont extrêmement illisibles. Cela vous a sans doute parfois frustré. C'est pourquoi vous devez éviter de provoquer cette frustration chez vos utilisateurs et créer des messages d'erreur informatifs, en espérant que les futures générations de développeurs Sage feront de même.

5.5.2. La construction If-Then-Else

Dans la section 5.2.5, page 229, nous avons écrit du code évaluant une fonction sur les premiers nombres naturels (0, 1, 2, 3, etc.) pour obtenir le nombre demandé. Cependant, ce code plantera si nous divisons par zéro. Vous pouvez essayer avec cette fonction :

$$f(x) = \frac{2x - 4x + 3}{2x - 5x + 6}$$

Nous allons maintenant utiliser un concept très puissant en programmation informatique, appelé la construction « if-then-else ». Elle ressemble aux instructions if utilisées précédemment, mais nous y ajoutons une commande else. L'idée est de vérifier d'abord (avec un if) si le dénominateur est nul. S'il est nul, nous afficherons « dne » pour « n'existe pas » ; ce sera la partie « then ». S'il est différent de zéro, nous afficherons la valeur de la fonction entière à cet entier naturel ; ce sera la partie « else ».

Considérez le code suivant :

```
def rationalFunctionToSequence( num, denom, howmany ):
```

Machine Translated by Google


```
    """Cette sous-routine prend une fonction comme première entrée, puis évalue cette
    fonction aux premiers nombres naturels « combien ».
    Par exemple, si howmany est 4, alors f est évalué à 0, 1, 2, 3. Tous les
    pôles sont signalés comme dne."""  for j in range(0, howmany): if
  (denom(x=j) == 0): print "dne",


        autre:
            value = num(x=j) / denom(x=j) print
            value,
```

Nous avons maintenant deux paramètres pour communiquer la fonction à la sous-routine, car nous avons besoin du numérateur et du dénominateur séparément.
Le troisième paramètre indique le nombre d'évaluations de fonctions naturelles demandées. La deuxième ligne est une simple boucle for qui nous guide à travers les premiers nombres naturels, à savoir 0, 1, 2, . . . , n − 2, n − 1. Nous allons ensuite évaluer le dénominateur au point x = j et vérifier s'il est nul – c'est la troisième ligne. À la quatrième ligne, qui n'est exécutée que si le dénominateur est nul, nous affichons « dne ».

La cinquième ligne est la commande else, ce qui signifie que les sixième et septième lignes, indentées sous la commande else et donc subordonnées à celle-ci, ne seront exécutées que si le dénominateur est différent de zéro. Nous calculons la valeur de f(x) à x = j, qui est simplement le numérateur divisé par le dénominateur (ligne six), et nous l'affichons enfin (ligne sept).

Maintenant, testons notre code, avec les lignes suivantes :

```
f(x) = x^2 - 4*x + 3 g(x) = x^2 - 5*x
+ 6 fonctionrationnelleÀSéquence( f(x),
g(x), 10 )
```

Nous obtenons la sortie suivante et pouvons être satisfaits que la sous-routine fonctionne correctement :

```
1/2 0 jours jours 3/2 4/3 5/4 6/5 7/6 8/7
```

La commande else permet de gagner du temps et rend le code plus lisible. Le code suivant est équivalent, mais plus fastidieux à écrire et à lire :

```
def rationalFunctionToSequence( num, denom, howmany ): """Cette sous-
    routine prend une fonction comme première entrée, puis évalue cette fonction aux
    premiers nombres naturels « howmany ».
    Par exemple, si howmany est 4, alors f est évalué à 0, 1, 2, 3. Tous les
    pôles sont signalés comme dne."""  for j in range(0, howmany): if
  (denom(x=j) == 0): print "dne",



        if (denom(x=j) != 0): value =
            num(x=j) / denom(x=j) print value,
```

Machine Translated by Google

Ici, nous avons explicitement indiqué que l'impression de « dne » ne s'effectue que si le dénominateur est nul. De même, la division du numérateur et du dénominateur, son stockage dans la valeur et son impression ne s'effectuent que si le dénominateur est différent de zéro. Par conséquent, la commande else n'est pas indispensable, mais elle simplifie la tâche du programmeur. Les avantages de la disponibilité de else sont plus faciles à apprécier lorsque l'on gère des instructions if-then-else complexes les unes à la suite des autres. Nous n'aborderons pas ce sujet dans ce chapitre, mais d'autres ouvrages sur Python le feront.

Une note historique est que même s'il n'existe pas de commande « then » en Python, ni en C, C++, ni en Java, le concept est toujours appelé « if-then-else ».
C'est parce que certains langages de programmation très anciens, comme BASIC et Pascal, utilisaient le mot « then » comme commande. La nomenclature reflète cette mémoire historique.

5.5.3. Défi facile : Fin de semestre du registraire, partie 1. Pour ce défi, vous allez écrire une petite sous-routine qui vérifiera la moyenne générale d'un étudiant et déterminera son statut à l'université. Le nom de la sous-routine doit être decideStatus et comporter deux paramètres : le nom de l'étudiant et sa moyenne générale.

Avant tout traitement, si la moyenne de l'étudiant est inférieure à zéro ou supérieure à quatre, la sous-routine génère une exception. Par la suite, si la moyenne de l'étudiant est inférieure à 2, il doit être déclaré en probation académique. Sur une seule ligne, indiquez son nom suivi de la mention « est en probation académique ». En revanche, si la moyenne de l'étudiant est de 2 ou plus, sur une seule ligne, indiquez son nom suivi de la mention « est correct ».

Il est important que vous n'utilisiez qu'une seule ligne, car plus tard (dans la section 5.7.9), nous créerons une autre sous-routine qui appelle cette sous-routine, mais pour plusieurs étudiants.

5.5.4. Un défi plus difficile : Fin de semestre, partie 2. Il s'agit d'une variante du défi précédent. Rappelons qu'une moyenne pondérée cumulative (GPA) est valide si et seulement si $0 \le g \le 4$. Pour toute moyenne pondérée cumulative (GPA) valide, le statut de l'étudiant doit être déterminé selon la règle suivante :

$4 \ge g \ge 3$ Liste du doyen $3 > g \ge 2$
Normal $2 > g \ge 1$ Probation
académique $1 > g \ge 0$ Renvoi académique

Voici un indice : vous pourriez essayer trois constructions si-alors-sinon.
À savoir, un if-then-else à l'intérieur de la partie « then » et un autre à l'intérieur de la partie « else » d'un énorme if-then-else. Cela nécessite une indentation très soignée.

Machine Translated by Google

5.6. Boucles While et boucles For

Il s'avère que la boucle for est la plus courante en programmation mathématique, mais elle est suivie de près par une boucle while. La boucle while est également très importante et apparaît fréquemment. Une boucle while continue tant qu'un test logique renvoie Vrai ; si ce test logique renvoie Faux, la boucle s'arrête.

Il n'y a plus rien à ajouter, puisque nous avons maintenant décrit la seule différence entre les boucles for et while. Voyons donc un exemple.

5.6.1. Une question sur les factorielles. Souvent,

lorsqu'on enseigne les factorielles, les permutations, les combinaisons, etc., il est utile d'essayer de sélectionner des nombres d'une taille particulière. Par souci de simplicité, limitons-nous aux factorielles. Imaginons que je veuille trouver les valeurs de n autour desquelles n! est approximativement un milliard, un billion, ou peut-être un millier.

Je pourrais procéder par essais et erreurs, mais ce n'est pas très agréable. Il serait plus pratique d'avoir une sous-routine qui me fournisse simplement une réponse. Plus précisément, nous souhaitons trouver, étant donné un certain y, la plus petite valeur de x telle que x! ≥ y.

Puisque 0! = 1, nous pouvons commencer avec x = 0, et nous devons compter jusqu'à ce que nous Atteindre la première valeur de x telle que x! ≥ y. Essayez le code suivant : def

first_factorial_greater_than(y): """Cette sous-routine

renvoie le plus petit entier x tel que x! >= y.""" guess = 0 while (factorial(guess) < y):

guess = guess + 1

Ce code

commencera par une valeur de 0 pour guess. Il continuera d'exécuter la commande guess = guess + 1 tant que factorial(guess) < y est évalué à True. Cependant, dès que factorial(guess) < y est évalué à False, la boucle s'arrêtera et guess ne sera pas incrémentée. En clair, dès que factorial(guess) >= y, la boucle s'arrêtera et le programme reprendra avec la ligne suivante return.

et.

Nous pouvons tester cela avec quelques nombres et constater son efficacité. Par exemple,

first_factorial_greater_than(130) renvoie 6 car 5! =
120 et 6! = 720.

Vous pouvez également essayer de vérifier avec ce qui

suit : print first_factorial_greater_than(10^6) print factorial(9)

Machine Translated by Google

imprimer factorielle(10)

5.6.2. Un défi : trouver le prochain nombre premier. Pour ce défi, vous allez reproduire l'une des commandes déjà intégrées à Sage. Dans la section 5.4.5, page 252, vous avez été mis au défi de compter le nombre de nombres premiers p dans l'intervalle 1 ≤ p ≤ x, probablement à l'aide d'une boucle for et de la commande is prime.

Ici, nous souhaitons trouver le plus petit nombre premier supérieur à x. En clair, nous souhaitons trouver le nombre premier suivant x, bien qu'il n'y ait aucune raison de supposer que x soit premier. Par exemple, le plus petit nombre premier supérieur à 13 est 17. Le plus petit nombre premier supérieur à 15 est également 17.

Ce qui est intéressant dans ce défi, c'est que nous ne pouvons clairement pas utiliser une boucle for car nous n'avons aucune idée du nombre d'entiers que nous devrons vérifier. Par exemple, le nombre premier suivant 113 est 127, soit un saut de 14 entiers. Le nombre premier suivant 107 est 109, soit un saut de 2 entiers. Le nombre premier suivant 360 653 est 360 749, soit un saut de 96 entiers.

Pour les nombres plus grands, la situation est encore plus amusante. Le nombre premier suivant 1 693 182 318 746 371 est 1 693 182 318 747 503, soit un saut de 1 132 entiers. En revanche, le nombre premier suivant est 1 693 182 318 747 523, soit un saut de 20 entiers.

Créez donc une petite sous-routine qui compte à partir d'un nombre (l'entrée) jusqu'à atteindre un nombre premier. Elle doit renvoyer le premier nombre premier trouvé, qui sera bien sûr supérieur à son entrée, puisque nous avançons seulement et jamais en arrière.

Vous pouvez facilement tester votre travail dans ce défi, car cette commande est déjà intégrée à Sage. C'est la commande next prime. Vous pouvez constater vos progrès en codant désormais des sous-routines équivalentes aux commandes Sage.

Pour un défi supplémentaire, écrivez votre code de manière à ce qu'il ne vérifie jamais les nombres pairs supérieurs à 2, car les nombres pairs supérieurs à 2 ne peuvent évidemment pas être premiers. Si l'entrée est impaire, c'est facile : vous devez donc détecter et traiter les entrées paires en conséquence.

5.6.3. Méthode de Newton avec une boucle While Maintenant que nous comprenons les boucles while, je dois avouer que j'ai été un peu étrange en utilisant une boucle for dans la méthode de Newton. L'application nécessite en réalité une boucle while. Le plan général est de répéter le processus itératif jusqu'à ce que l'approximation atteigne la norme requise en étant inférieure à epsilon (en valeur absolue). Cependant, nous pouvons avoir une « voie de secours » si le nombre maximal d'itérations autorisé est dépassé.

Le code est présenté dans la figure 5.6.3. J'ai marqué les nouvelles lignes avec #new comme précédemment. Notez que j'ai modifié la valeur par défaut de max iterate à 1 000, afin de refléter le fait que ce sera rarement le cas pour la sous-routine.

Machine Translated by Google

se termine. Normalement, l'approximation sera inférieure à epsilon (en valeur absolue).

Se pose également la question de savoir comment terminer la boucle while après avoir atteint le nombre maximal d'itérations. Comme vous pouvez le constater, j'utilise une instruction if pour détecter la condition, puis affiche un avertissement à l'utilisateur. Ensuite, j'utilise la commande break, que nous n'avons pas encore vue. Cette commande permet de quitter la boucle.

boucle actuelle, mais reprendre le flux du programme après la boucle dans le code source. (En d'autres termes, il continue le programme au point spécifique où le programme irait une fois la boucle normalement terminée.) Ceci est en contraste avec une commande de retour, qui terminera à la fois la boucle et la sous-routine.

Il est important de noter que si vous avez plusieurs boucles imbriquées, la commande break s'applique uniquement à la boucle la plus interne. Ici, lorsque nous activons la commande break, nous exécutons des instructions d'impression lorsque le paramètre optionnel verbose est à True, puis nous renvoyons la boucle finale.

répondre.

5.6.4. L'impact des racines répétées. Enfin, je voudrais

vous montrer un phénomène intéressant dans la méthode de Newton. Avec la version 6 du code ci-dessus, définissez les deux polynômes suivants :

f(x) = (x-1)^5 g(x) =
(x-1)*(x-10)*(x-20)*(x+10)*(x+20)

Ensuite, essayez

newton_method(f(x), 3, verbose = True)

Mais vous constaterez que la valeur f(x) devient trop faible, et la méthode de Newton échoue. Nous pouvons empêcher cet arrêt automatique en définissant le paramètre optionnel hazard à 10−9 au lieu de 10−4 , valeur par défaut. Nous saisissons maintenant

newton_method(f(x), 3, verbose = True, hazard = 10^-9)

et on voit que 22 itérations ont été nécessaires ! En revanche

newton_method(f(x), 3, verbose = True, hazard = 10^-9)

Fournit une réponse en seulement quatre itérations. De plus, il s'agit d'une approximation de meilleure qualité, avec g(xold) valant −1,75 · · · × 10−11. En revanche, f(xold) valait 6,99 · · · × 10−10, une valeur nettement inférieure.

Comme vous pouvez le constater, f(x) possède une racine de multiplicité cinq. En revanche, g(x) possède cinq racines de multiplicité un. Vous pouvez choisir des polynômes très différents, mais une racine très répétée aura effectivement une convergence plus lente qu'un polynôme de même degré mais sans racines répétées. Ce théorème est souvent démontré en cours d'analyse numérique, mais pas toujours au premier semestre. Cette démonstration est trop difficile pour ici, mais expérimentez sur des polynômes à racines répétées et vous verrez le phénomène.

Machine Translated by Google


```
def newton_method( f, x_old, max_iterate = 1000, # modifié
        verbose = Faux, epsilon = 10^(-9), hazard = 10^(-4) ):

    Implémentation de la méthode de Newton pour trouver la racine de f(x) à partir d'une
    estimation initiale de x. 10 itérations par défaut seront effectuées, sauf si le paramètre
    facultatif max_iterate est défini sur une autre valeur. Définissez verbose = True
    pour afficher toutes les étapes intermédiaires.

    L'algorithme s'arrêtera prématurément si |f(x)| < epsilon. Une exception sera levée si |
    f'(x)|<hazard.

    f_prime(x) = diff( f, x )

    itérations=0

    while (abs(f(x_old)) >= epsilon): # nouvelles itérations = itérations + 1 #
        nouveau

        si (itérations >= max_iterate) : # nouveau print "Avertissement :
            limite d'itération atteinte." # nouveau break # nouveau


        si (abs(f_prime(x_old)) < hazard) : raise
            RuntimeError('f_prime(x) est devenu trop petit ! '+ 'Veuillez essayer une
                    autre condition initiale.')

        ratio = f(x_old) / f_prime(x_old) x_new = x_old - ratio


        if (verbose==True) : print
            "iteration=", itérations # modifié print "x_old=", x_old print "f(x)=",
            f(x_old) print "f'(x)=", f_prime(x_old)
            print "ratio=", ratio print "x_new=",
            x_new print




        x_ancien = N(x_nouveau)

    # maintenant la boucle est terminée. (# nouveau) si
    (verbose==True) :                                   # nouveau
        itérations d'impression, des                    # nouveau
        "
        itérations d'impression étaient nécessaires." # nouvelle impression
        "f(answer) = ", # nouvelle impression f(x_old)
                                                        # nouveau

    retourner x_old
```

Figure 11. Méthode de Newton dans Sage, version 6

Machine Translated by Google

5.6.5. Factorisation par division d'essai

Nous allons poursuivre notre réflexion sur la programmation de petits morceaux de code exécutant des tâches similaires à celles de Sage. Nous allons ici nous essayer à la factorisation, comme la commande factor de Sage, mais avec une approche extrêmement simple : nous diviserons simplement par des nombres premiers jusqu'à obtenir le nombre cible.

Le nombre cible est initialement le nombre que nous souhaitons factoriser. Nous allons diviser notre « nombre cible » par des nombres premiers successifs, en commençant par 2. Si un nombre premier divise le nombre cible, nous le remplacerons par le quotient, nous afficherons ce nombre premier dans la factorisation et nous ne passerons pas au nombre premier suivant. Si un nombre premier ne divise pas le nombre cible, nous remplacerons le nombre premier actuel par le nombre premier suivant, identifié par la commande « prochain nombre premier » de Sage, mais nous n'afficherons pas ce nombre premier dans la factorisation. Nous nous arrêterons lorsque le nombre cible sera 1, sans afficher le 1.

Un problème est de savoir si un nombre est divisible par un autre. Les mathématiciens utilisent généralement une astuce appelée « arithmétique modulaire », mais elle est peu pratique ici, car tous mes lecteurs ne savent pas ce qu'est l'arithmétique modulaire. Si vous connaissez l'arithmétique modulaire, vous savez que x est divisible par y si et seulement si x mod y est égal à zéro. Si vous ne connaissez pas l'arithmétique modulaire, croyez-moi sur parole.

```
si ( (cible % current_prime) == 0 ):
```

comme équivalent à l'inexistant mais désiré si ("a est divisible par b") :.

Le code de trialDivisionFactor peut paraître un peu déroutant. Il s'agit d'une sous-routine mathématiquement non triviale, et il vaut donc la peine de s'y attarder un peu.

```
déf trialDivisionFactor( x ):
    En utilisant la division par essai, cette sous-routine affichera la factorisation première de
    l'entrée x. On suppose que x est un entier supérieur à 1.

    while (target>1) : if ( (target %
        current_prime) == 0 ): # target est divisible par le nombre premier
            actuel. print current_prime, target = target / current_prime else :

            # la cible n'est pas divisible par le nombre premier actuel.
            current_prime = next_prime( current_prime )

    imprimer
```

Machine Translated by Google

Cette instruction d'impression à la fin de la sous-routine nous permettra d'essayer plusieurs cas de test à la fois. Essayons de tester avec

```
trialDivisionFactor( 25 )
trialDivisionFactor( 26 )
trialDivisionFactor( 27 )
```

Nous obtenons l'excellent résultat ci-dessous :

```
Calcul de la factorisation de 25 : 5 5
Calcul de la factorisation de 26 : 2 13
Calcul de la factorisation de 27 : 3 3 3
```

Vous pouvez réessayer ce test de trois lignes, après avoir supprimé l'instruction print isolée à la fin de la sous-routine. Le résultat amusant obtenu illustrera la nécessité de cette instruction print. Un meilleur test consisterait à vérifier plusieurs entiers consécutifs, comme ci-dessous.

```
pour y dans la plage (19, 50) :
    trialDivisionFactor( y )
```

Si vous essayez cette vérification, vous verrez que notre code fonctionne bien.

Cependant, la sous-routine ne fonctionne pas bien avec une entrée négative. Nous allons Il faut corriger ça. Par exemple, rien ne s'imprime si vous essayez

```
trialDivisionFactor( -10 )
```

5.6.6. Un mini-défi : division d'essai, arrêt prématuré

Notre sous-routine est très inefficace dans le sens où si vous lui demandez 1 000 003 (qui se trouve être un nombre premier), elle doit vérifier tous les nombres premiers en commençant par 2, 3, 5, 7, . . . , 999 983, 1 000 003, et ne se termine que lorsqu'elle atteint 1 000 003.

Le lemme suivant nous aidera. Si un entier positif n n'est pas divisible par l'un des nombres premiers inférieurs ou égaux à $\sqrt{n}$, alors n doit être premier.

Preuve : Supposons que n ne soit pas premier. Considérons alors la factorisation première de n, à savoir

$$n = (p1)(p2)(p3)\cdot\cdot\cdot(p)$$

où ≥ 2 puisque n n'est pas premier. On sait qu'il n'existe pas de nombre premier divisant n qui soit inférieur ou égal à $\sqrt{n}$; on a donc

$$p1 > \sqrt{n}\quad p2 > \sqrt{n}\quad p3 > \sqrt{n}$$

$$\vdots \quad \vdots \quad \vdots$$

$$p > \sqrt{n}$$

Machine Translated by Google

La combinaison de tous ces éléments (ce qui n'est autorisé que parce que tout est positif) donne :

$$(p1)(p2)(p3)\cdot\cdot\cdot(p) > (\sqrt{n})\,(p1)$$
$$(p2)(p3)\cdot\cdot\cdot(p) > n/2\ n > n/2$$

maintenant Puisque ≥ 2 alors /2 ≥ 1 et /2 > n1 . Nous avons

$$nn > n/2 \text{ et } n/2 > n \text{ donc } n > n$$

ce qui est clairement une contradiction. Par conséquent, notre supposition était fausse, et n est sûrement premier.

Nous devons donc vérifier si le nombre premier actuel est supérieur à la racine carrée de la cible. Si c'est le cas, nous pouvons alors arrêter la boucle while, car la cible est première.

Votre défi consiste à modifier le code existant ci-dessus pour appliquer ce raccourci. Un bon cas de test pour votre travail pourrait être :

x = 1 000 730 021

Lors de la mise à niveau de cette sous-routine, nous devons corriger l'impossibilité de factoriser les nombres négatifs. Sachant que nous pouvons factoriser les entiers positifs, une solution simple s'offre à nous. Étant donné un entier négatif x, nous devrions simplement factoriser −x, qui sera clairement positif. Nous devrions mettre « −1 » devant la factorisation pour indiquer que le nombre cible est négatif. Enfin, si quelqu'un demande la factorisation de 0, nous devrions lever une exception. Par exemple, si vous utilisez la commande intégrée de Sage et demandez factor(0), vous obtiendrez un long message d'erreur, suivi de

ArithmeticError : la factorisation première de 0 n'est pas définie.

5.6.7. Défi : Une caisse enregistreuse améliorée. Vous allez maintenant mettre à niveau le code que vous avez écrit dans le défi de la section 5.2.2, en simulant une caisse enregistreuse. Au lieu d'indiquer à l'utilisateur qu'il lui rend la monnaie : 4,75 $, vous devriez afficher la monnaie de quatre dollars et trois quarts. Un logiciel de ce type est nécessaire pour les caisses enregistreuses où les pièces sortent simplement du côté de la caisse dans un plateau que le client peut récupérer. Ces caisses, bien que coûteuses, sont devenues populaires dans les magasins au cours de la dernière décennie, car de nombreuses personnes qui, autrement, travailleraient aux caisses enregistreuses, sont incapables (!) de rendre la monnaie correctement.

Il est parfois difficile de trouver comment procéder tout en distribuant le moins de pièces possible. C'est un objectif louable, car il évite de recharger la machine trop souvent. De plus, un client serait irrité s'il recevait 475 centimes en retour. Voici un exemple simplifié : si vous savez que vous devez rendre 3,90 $, par exemple, vous devez d'abord donner autant de billets que possible (ici 3 billets, comme 4 $).

Machine Translated by Google

c'est trop). Maintenant, vous devez donner 90 centimes, et vous devez donner autant de pièces de 25 cents que vous pouvez (dans ce cas, 3 pièces de 25 cents, car cela fait 75 centimes, ce qui est bien, mais 1,00 $, c'est trop). Bon, maintenant il reste 15 centimes, vous devez donc donner autant de pièces de 10 cents que vous pouvez (dans ce cas, 1, car 0,20 $, c'est trop). Enfin, il reste 5 centimes, soit une pièce de 5 cents. Un moyen simple de coder cela est d'utiliser cinq boucles while (une pour les billets de 1 dollar, une pour les 25 cents, une pour les 10 cents, une pour les 5 cents et une pour les 1 cents) ou d'utiliser intelligemment la commande « floor » de Sage.

5.7. Fonctionnement des tableaux et des listes

Nous avons rencontré à plusieurs reprises le concept de liste Python tout au long de ce livre. La structure des listes en Python remplace des concepts plus anciens comme les tableaux et les listes chaînées dans d'autres langages de programmation. Si vous n'avez jamais programmé, ne vous inquiétez pas de ce que cela signifie pour l'instant. Nous allons ici apprendre quelques astuces pour manipuler les listes.

Au fait, nous verrons également comment l'intrigue génère des images. Un merci spécial à Brian Knaeble pour m'avoir encouragé à inclure ce sous-thème, car les élèves de niveau 100 sont souvent très curieux de savoir comment les calculatrices graphiques et Sage produisent des graphiques.

5.7.1. Listes de points et tracés Le petit extrait

de code va explorer comment les ordinateurs génèrent des tracés.
Nous allons examiner une fonction simple f(x) = x 3−x et écrire une sous-routine qui la représentera graphiquement sur l'intervalle −3/2 < x < 3/2.

Voici le code que nous allons analyser. f(x) =

```
x^3 - x list_of_points
= []
verbeux = Faux


pour k dans la plage (-150, 150) : x = k*0,01
        y = f(x)
        nouveau_point
        = (x,y) si (verbose==True) :
        imprimer nouveau_point



        liste_de_points = liste_de_points + [nouveau_point]

nuage de points( liste_de_points )
```

Tout d'abord, nous définissons notre fonction, puis nous partons d'une liste vide de points (ce sont nos deux premières lignes). De même que toute suite de nombres, séparés par des virgules et entre crochets, est une liste, la liste vide est simplement une paire de crochets sans aucun élément entre eux. Ensuite,

Machine Translated by Google

Nous définissons rapidement une variable verbose, pratique par la suite. Vient ensuite la boucle for, qui effectue l'essentiel du travail.

Pour chaque itération de la boucle for, nous calculons une coordonnée x.
Bien que k passe de −150 à 150, chaque abscisse doit être très proche. En fait, nous multiplions par 0,01, forçant x à passer de −1,5 à 1,5 par incréments de 0,01. L'ordonnée est simplement f(x). Ensuite, nous construisons un point, qui est une paire ordonnée, et qui est simplement « x virgule y ». Il pourrait être intéressant de définir verbose sur True plutôt que False pour visualiser cette liste. Ensuite, nous mettons à jour notre liste de points en additionnant l'ancienne liste et une liste composée uniquement du nouveau point. Enfin, lorsque toutes les valeurs k sont traitées et que la boucle for est terminée, nous créons un nuage de points.

La ligne

```
list_of_points = list_of_points + [new_point]
```

mériterait d'être expliqué.
Dans cette ligne, nous indiquons à Sage que la nouvelle liste de points doit devenir l'ancienne liste de points, plus le nouveau point. Le signe plus entre deux listes entraînera une concaténation (ou fusion) des deux listes. Cependant, contrairement à la théorie des ensembles, l'ordre est important et des doublons peuvent se produire. Nous explorerons les différences entre les ensembles et les listes Python dans un instant. Le fait d'avoir dû écrire le nom de la liste deux fois est un inconvénient, car son nom est assez long. Pour pallier ce problème, Python dispose d'un raccourci :

```
list_of_points.append( new_point ).
```

qui collera un seul élément à la fin de la liste donnée.

La question naturelle serait alors : « Est-ce ainsi que Sage procède ? » Ce qui se passe ici correspond à l'algorithme de base presque toujours utilisé dans des outils comme les calculatrices graphiques ou les anciens logiciels mathématiques. Sage prend en fait quelques points répartis sur de minuscules intervalles régulièrement espacés le long de l'axe des abscisses, au lieu d'un seul point. De plus, ces points représentatifs de l'intervalle ne sont pas régulièrement espacés, mais choisis aléatoirement. Nous avons expliqué pourquoi à la page 3.2.
Enfin, Sage détectera occasionnellement que davantage de points sont nécessaires dans certains sous-intervalles parce que la fonction « devient folle à cet endroit » et évaluera des points supplémentaires dans ces sous-intervalles.

5.7.2. Un défi : Créer la commande Plot

Modifions le code que nous venons d'écrire pour le rendre plus général. Premièrement, nous devrions en faire une sous-routine appropriée avec une définition et un nom. Deuxièmement, nous devrions prendre des entrées : la fonction à représenter graphiquement, suivie du point de départ et du point de fin de l'intervalle. Troisièmement, nous devrions autoriser un paramètre optionnel, qui indique le nombre de points souhaités par l'utilisateur sur l'ensemble de l'intervalle représenté graphiquement. Je pense que 100 est une bonne valeur par défaut pour ce paramètre optionnel.

La difficulté est d'élaborer une formule qui, étant donné (1) le nombre de points souhaités, (2) le début de l'intervalle, (3) la fin de l'intervalle et (4) la variable d'itération de la boucle, fournira la coordonnée x. C'est la principale difficulté de ce défi.

Machine Translated by Google

5.7.3. Opérations sur les listes

Comme nous l'avons vu précédemment, l'addition de deux listes les concatène.

```
a = [ 2, 3, 4 ] b = [ 1, 3 ]


imprimer a + b
imprimer b + a
```

Nous obtenons le résultat suivant :

```
[2, 3, 4, 1, 3] [1, 3, 2,
3, 4]
```

Ici, nous pouvons clairement voir la différence entre les listes et les ensembles Python. Tout d'abord, dans un ensemble mathématique, aucune entrée n'apparaît plus d'une fois. Cependant, ici, nous voyons que des entrées peuvent apparaître plusieurs fois. De plus, dans les ensembles mathématiques, l'ordre n'a pas d'importance. En Python, c'est l'ordre qui compte.

Il est parfois utile de connaître la longueur d'une liste Python. Pour cela, nous utilisons la commande len. Nous l'avons d'ailleurs brièvement vue page 150, lorsque nous avons abordé la détermination du nombre de diviseurs d'un nombre.

La commande in est utilisée avec les listes Python et les instructions if pour rendre le comportement dépendant de l'appartenance à une liste. Comme vous pouvez le voir, 2 fait partie de notre liste « a » mais pas de notre liste « b ». Par conséquent, les lignes suivantes renverront respectivement True et False, comme on pourrait s'y attendre. print 2 in a print 2 in b Pour accéder à des membres spécifiques d'une liste, nous utilisons les crochets. Les informaticiens comptent à partir de 0 et non à partir de 1, pour des raisons très difficiles à expliquer mais très valables liées à l'emplacement mémoire. Par conséquent, la première entrée est l'entrée 0, la deuxième entrée est l'entrée 1, la troisième entrée est l'entrée 2, et ainsi de suite. Il est intéressant de noter qu'il arrive que l'on veuille la dernière entrée, ou l'avant-dernière entrée, ou l'avant-dernière entrée. Bien que je ne connaisse pas d'autres langages de programmation qui le permettent (il y en a peut-être), Python permet d'y accéder directement en demandant l'entrée « -1 » de la liste, qui est la dernière, ou l'entrée « -2 » de la liste, qui est l'avant-dernière. Essayez le code ci-dessous pour illustrer cette fonctionnalité.

```
C = [1, 2, 3, 4, 5, 6] imprimer C[0]
imprimer C[1]
imprimer C[2]
imprimer
imprimer
C[-1] imprimer
C[-2] imprimer
C[-3]
```

Machine Translated by Google

Nous pouvons également supprimer des éléments des listes grâce à la commande remove.

```
C.remove(5)
imprime C
```

Parfois, dans la programmation mathématique, nous pouvons vouloir additionner une liste. La commande sum calcule automatiquement la somme d'une liste. Moins utilisée, mais parfois utile, la commande prod multiplie tous les membres d'une liste. Le code ci-dessous calcule correctement une somme de 28 et un produit de 5040 pour les sept premiers entiers positifs.

```
E = [ 1, 2, 3, 4, 5, 6, 7 ] print "Somme : ",
somme(E) print "Prod : ", prod(E)
```

Il existe également une commande shuffle en Python, idéale pour randomiser une liste. Elle peut être utile pour créer des jeux ou des expériences scientifiques où la randomisation est importante pour des raisons statistiques. Je l'utilise pour créer des listes afin de pouvoir appeler les élèves dans un ordre aléatoire et de m'assurer de ne pas appeler quelqu'un deux fois avant d'avoir appelé chaque élève une fois.

```
liste = ["Alice", "Bob", "Charlie", "Diane", "Edward",
                "Francis", "Gregory", "Harriet" ] imprimer la liste
mélangée(liste)
imprimer la liste
mélangée(liste)
imprimer la liste
mélangée(liste)
imprimer la liste
mélangée(liste)
```

Ce code a produit, pour moi, le résultat suivant, mais vous pourriez obtenir quelque chose de complètement différent. (Après tout, il s'agit d'un résultat pseudo-aléatoire.)

```
['Alice', 'Bob', 'Charlie', 'Diane', 'Edward', 'Francis', 'Gregory', 'Harriet']
['Diane', 'Bob', 'Francis', 'Alice', 'Gregory', 'Charlie', 'Harriet', 'Edward']
['Charlie', 'Diane', 'Grégory', 'Bob', 'Harriet', 'Francis', 'Edward', 'Alice']
['Francis', 'Edward', 'Harriet', 'Gregory', 'Bob', 'Diane', 'Alice', 'Charlie']
```

Il existe de nombreuses commandes pour vous aider à travailler avec des listes en Python. Vous trouverez une excellente référence à l'adresse URL ci-dessous. Elle fait partie d'un site web d'enseignement général de Python destiné aux programmeurs expérimentés.
Cependant, il n'y a aucun mal à simplement faire défiler une page Web pour voir combien de commandes liées aux listes existent dans Python.

http://www.diveintopython.net/native_data_types/lists.html Je recommande quelques tutoriels plus simples comme « Lectures complémentaires » à la page 274, qui sont plus adaptés aux débutants.

Machine Translated by Google

5.7.4. Boucle sur un tableau. Dans cette

sous-section, nous allons explorer comment générer des rapports à partir de tableaux. Il arrive souvent qu'une partie de mon programme calcule quelque chose, qu'une autre génère des statistiques à partir de ces données, et qu'une troisième les affiche dans un format utile. Ici, nous allons simuler un petit morceau d'un logiciel de gestion. Nous allons écrire une sous-routine qui génère un rapport de revenus basé sur une liste de noms et un tableau de revenus.

Nous prendrons en entrée deux listes, une de noms et une autre de revenus. Nous devons d'abord vérifier qu'ils ont la même longueur ; dans le cas contraire, nous lèverons une exception. Nous parcourrons ensuite tous les noms à l'aide d'une boucle for. Nous afficherons une ligne indiquant le nom et le revenu correspondant. Voici le code :

```
def generateRevenueReport( noms, revenus ):
    Cette sous-routine gère deux listes : une liste de noms et une liste de revenus. Si les listes sont
    de longueurs différentes, une exception est générée. Un rapport de revenus est alors généré,
    avec chaque nom et le revenu généré, imprimés sur une ligne par responsable. if
    ( len(names) != len(revenues) ): raise RuntimeError('La liste des noms et la liste des
    revenus n'étaient pas de la même longueur !
        Veuillez réessayer.')


    pour i dans la plage (0, len (noms)) : afficher
        les noms [i], afficher
        les apports ", afficher les
        revenus [i]
```

Vous trouverez ci-dessous comment nous allons tester notre code.

```
noms = [ "Ned", "Ed", "Ted", "Jed", "Fred" ] revenus =
[ 200*1000, 300*1000, 400*1000, 100*1000, 600*1000 ]

generateRevenueReport( noms, revenus )
```

Vous devriez recevoir le résultat suivant :

```
Ned a apporté 200 000 Ed a
apporté 300 000 Ted a
apporté 400 000 Jed a apporté
100 000 Fred a apporté
600 000 Vous vous demandez
```

peut-être pourquoi nous avons vérifié la longueur des listes et pourquoi nous levons une exception si elles sont de longueur différente. Disons que j'ai une liste de 5 éléments. Quand je demande names[0] ou names[1], le nombre entre crochets est appelé un index (pluriel : indices). Comme j'ai cinq éléments, les indices de liste valides sont 0, 1, 2, 3, 4. De plus, comme Python autorise les indices négatifs, je peux demander -1, -2, -3, -4 et -5. Les entiers naturels comptent du premier au dernier, et les entiers négatifs comptent du dernier au premier. Ce qui est interdit, c'est de demander des indices supérieurs à ou

Machine Translated by Google

égale à 5. Si la longueur d'une liste est différente de celle d'une autre, le dernier index demandera un élément absent de la liste la plus courte. Cela générera une exception (même sans raise) et arrêtera le programme.

Pour tester le phénomène ci-dessus, ajoutez un nom à la fin de la liste des noms et appelez à nouveau la sous-routine. La sous-routine lèvera une exception et le programme s'arrêtera. Maintenant, faites un retour arrière sur l'instruction if qui vérifie les longueurs, puis sur l'instruction raise qui la suit. Observez le comportement de la sous-routine lorsque vous ajoutez un nom erroné. Un message d'erreur sera généré dans les deux cas. Cependant, en utilisant un if suivi d'un raise, vous pouvez rendre le message d'erreur lisible.

Mini-défi : Calculer le revenu total J'aimerais maintenant vous donner un mini-défi. Pouvez-vous modifier la sous-routine que nous venons d'écrire pour qu'elle génère également le revenu total (en bas du rapport de revenus) une fois terminé ?

Vous pouvez utiliser un accumulateur (voir section 5.1.4, page 219) ou la commande sum pour les listes (voir section 5.7.3, page 268). À vous de choisir !

5.7.5. Défi : Rapport de résultat de l'entreprise. Ce défi vous demandera d'écrire une sous-routine dont les trois entrées sont des listes. La première est une liste de noms, suivie d'une liste de revenus, puis d'une liste de coûts. Ces listes représentent les responsables de chaque division de l'entreprise, ainsi que leurs revenus et dépenses. Votre sous-routine générera un rapport de résultat trimestriel, constitué d'une liste de noms suivie du bénéfice de chaque division. Cependant, si les listes saisies ne sont pas de taille égale, une exception sera générée.

Enfin, n'oubliez pas le résultat net : le bénéfice total doit figurer sur la dernière ligne du rapport de résultat. (D'où l'expression « Quel impact cela aura-t-il sur le résultat net ? »)

5.7.6. Moyenne de quelques nombres. Dans cette sous-section, nous allons écrire une sous-routine pour faire la moyenne de certaines notes, mais nous souhaitons abandonner la note la plus basse. Cette pratique étant peu courante en dehors des États-Unis, je vais prendre un moment pour expliquer le concept.

Pour nos lecteurs non américains, les étudiants américains passent souvent un court test écrit appelé « quiz » chaque semaine ou toutes les deux semaines, au lieu d'un seul examen à la fin du semestre (ou de l'année) comme c'est la norme au Royaume-Uni. En fait, lorsque j'enseigne, il peut y avoir 12 à 14 quiz, un examen final long et un examen semi-long en milieu de semestre, appelé « examen de mi-session ». À bien y réfléchir, c'est une énorme différence. Si un étudiant suit les cours de Calcul II et de Calcul III avec moi, il finira par passer

Machine Translated by Google

Environ 32 épreuves différentes, comparables à des examens, sont organisées au cours de l'année universitaire. Dans le modèle britannique classique, ils peuvent ne passer que deux examens, voire un seul. examen.

Quoi qu'il en soit, avec 13 quiz, certains enseignants « laissent tomber la note la plus basse ». C'est une façon de compenser une mauvaise journée, voire une gueule de bois. La note la plus basse est rayée du compte rendu, et la moyenne des 12 autres est calculée. Prenons un exemple concis : un cours avec six quiz, avec des notes de 100, 80, 90, 70, 80 et 90. Dans ce cas, la note de 70 est rayée et la moyenne est calculée.

$$\frac{100 + 80 + 90 + 80 + 90}{5} = \frac{440}{5} = 88$$

Ce qui correspond à un B+. C'est nettement mieux que 85,4285 · · · sans chute, ce qui , la moyenne correspond à un B. Voici le code : def averageGrades(grades) :

Cette sous-routine prend en entrée une liste de notes comprises entre 0 et 100 inclus. Elle calcule ensuite la moyenne des notes, après avoir supprimé la note la plus basse.

```
pour g dans les notes :
        si ((g<0) ou (g>100)) : raise
                RuntimeError('Note invalide dans la liste !')

        total = total + g

        si (g < lowest_seen) : lowest_seen
                = g

numérateur = total - plus bas_vu dénominateur =
len(notes) - 1

renvoie N(numérateur/dénominateur)
```

Tout d'abord, vous pourriez être surpris par la phrase :

```
pour g dans les notes :
```

qui est un type de syntaxe différent pour la boucle for de ce que nous avons vu jusqu'à présent. Ici, nous disons que g doit prendre la valeur de chacun des membres de la liste grades exactement une fois.

Comme vous pouvez le constater, nous déclenchons une exception si la note est inférieure à zéro ou supérieure à 100 %. Nous avons utilisé le mot-clé « or » ici, ce qui signifie que l'instruction « if »

---

[5] Il convient de noter que certaines universités britanniques ont adopté le modèle américain. De plus, je sais que les universités allemandes organisent souvent des séances hebdomadaires de problèmes écrits dans leurs cours de mathématiques, qui sont fondamentalement le même concept que nos quiz hebdomadaires ou bimensuels.

Machine Translated by Google

se déclenchera si g < 0 ou si g > 100. Dans d'autres situations, vous pouvez utiliser la commande et de la même manière.

Nous disposons également d'un accumulateur, appelé total. La valeur la plus basse observée nécessite quelques explications. Si, au milieu d'une boucle, nous constatons une pente g inférieure à la valeur la plus basse observée, nous devons alors mettre à jour la valeur la plus basse observée pour qu'elle corresponde à cette valeur « très basse » de g. Une fois la boucle terminée, la valeur la plus basse observée correspondra certainement à la valeur la plus basse jamais observée.

Il ne reste plus qu'à choisir une valeur pour le début de la sous-routine.
J'ai choisi 101, car aucune note ne dépassera 100. Par ailleurs, si une note supérieure à 100 est trouvée, la sous-routine lèvera une exception. Je suis toujours assuré de trouver une note dans la liste inférieure à 101.
De cette façon, la note la plus basse observée sera toujours l'une des notes de la liste, et en fait, ce sera la plus basse.

Vous pouvez tester la sous-routine ci-dessus avec la sortie suivante.

```
notesmoyennes( [ 100, 80, 90, 70, 80, 90 ] )
```

Nous obtenons le score de 88 comme prévu.

5.7.7. Mini-défi : Moyenne avec et sans abandon. Une modification amusante de la sous-routine précédente serait un bon défi pour vous. Ajoutez un paramètre facultatif appelé miséricordieux. Le quiz le plus bas doit être abandonné si miséricordieux est vrai, mais pas abandonné si miséricordieux est faux. La valeur par défaut devrait être Faux.

5.7.8. Mini-défi : doubler le comptage du quiz le plus élevé Une autre modification du mini-défi précédent serait de demander aux professeurs miséricordieux de compter le quiz le plus élevé comme double, plutôt que d'abandonner le quiz le plus bas. Essayez cela.

5.7.9. Un défi : La course du registraire, troisième partie Voici un excellent exemple de la façon dont de petites sous-routines se transforment en sous-routines plus grandes, puis en programmes. Considérez les sous-routines que vous avez écrites aux sections 5.5.3 et 5.5.4. On vous demande maintenant d'écrire une sous-routine qui prendra une liste de noms d'étudiants et une liste de moyennes. Si les listes sont de taille inégale, générez une exception. Si les listes sont de taille égale, appelez la sous-routine précédemment écrite pour chaque étudiant de la liste. Un rapport devrait être généré, indiquant le nom de l'étudiant, sa moyenne et enfin son sort. Vous devrez peut-être légèrement modifier votre ancienne sous-routine pour prendre en compte cela.

5.7.10. Un utilitaire : Renvoyer uniquement les racines réelles, rationnelles ou entières. Précédemment, dans la section 1.8, page 37, nous avons appris à utiliser la commande solve pour trouver les racines d'un polynôme. Naturellement, cela inclut les racines qui sont

Machine Translated by Google

nombres complexes. Cependant, il arrive que l'on ne souhaite connaître que les racines réelles, rationnelles ou entières.

Vous avez probablement déjà vu des notations telles que C, R, Q et Z pour représenter respectivement les nombres complexes, les nombres réels, les nombres rationnels et les entiers. Pour les représenter, Sage utilise les notations CC, RR, QQ et ZZ. Nous pouvons utiliser la commande in comme s'il s'agissait de listes Python, même s'il s'agit d' ensembles infinis6.

Par exemple, si nous sommes intéressés par

$$f(x) = x^9 - 10x^8 + 40x^7 - 100x^6 + 203x^5 - 290x^4 + 260x^3 - 200x^2 + 96x$$

nous pourrions taper

```
f(x) = x^9-10*x^8+40*x^7-100*x^6+203*x^5-290*x^4+260*x^3-200*x^2+96*x réponses = résoudre( f(x) == 0, x )


pour x dans les réponses :
    imprimer x
```

Cela imprimerait les neuf racines complexes, à savoir

{0, 1, 2, 3, 4, −2i, −i, i, 2i}, mais nous ne

souhaitons peut-être pas voir les racines imaginaires. Nous pouvons essayer ceci à la place.

```
f(x) = x^9-10*x^8+40*x^7-100*x^6+203*x^5-290*x^4+260*x^3-200*x^2+96*x réponses = résoudre( f(x) == 0, x )


pour x dans les réponses : si
    (x.rhs() dans RR) : imprimer x
```

ce qui nous limitera aux seules racines réelles, excluant les quatre racines purement imaginaires.

Au fait, rhs() signifie « côté droit ». Le côté gauche serait accessible par lhs(), mais serait inutile dans ce cas, car il s'agirait simplement de x.

Transformez maintenant f(x) en g(x) dans le code ci-dessus, où g(x)

$$= 3x^7 - 2x^6 - 15x^5 + 10x^4 + 6x^3 - 4x^2 + 24x - 16$$

En prenant soin de ne pas oublier d'effectuer la modification deux fois : une fois dans la définition du polynôme et une fois dans la commande de résolution. Si nous conservons RR, nous ne voyons que cinq racines. En revanche, si nous remplaçons RR par CC, nous voyons les sept racines, à savoir

$$-2, -\sqrt{2}, -\frac{2}{3}, \sqrt{2}, 2, -i, i$$

---

[6]En fait, si vous connaissez les différentes dimensions de l'infini, R et C sont la « grande espèce » de l'infini ou « infini indénombrable », parfois appelée 1. En revanche, Q et Z sont la « petite espèce » de l'infini ou « infini indénombrable », parfois appelée 0. Le symbole est la lettre hébraïque « aleph ».

Machine Translated by Google

$$-\frac{1}{12}\sqrt{\frac{36\left(\frac{1}{18}\sqrt{563}\sqrt{3}-\frac{65}{54}\right)^{\frac{2}{3}}+33\left(\frac{1}{18}\sqrt{563}\sqrt{3}-\frac{65}{54}\right)^{\frac{1}{3}}-56}{\left(\frac{1}{18}\sqrt{563}\sqrt{3}-\frac{65}{54}\right)^{\frac{1}{3}}}}-\frac{1}{2}\sqrt{-\left(\frac{1}{18}\sqrt{563}\sqrt{3}-\frac{65}{54}\right)^{\frac{1}{3}}+\frac{14}{9\left(\frac{1}{18}\sqrt{563}\sqrt{3}-\frac{65}{54}\right)^{\frac{1}{3}}}-\frac{39}{2\sqrt{\frac{36\left(\frac{1}{18}\sqrt{563}\sqrt{3}-\frac{65}{54}\right)^{\frac{2}{3}}+33\left(\frac{1}{18}\sqrt{563}\sqrt{3}-\frac{65}{54}\right)^{\frac{1}{3}}-56}{\left(\frac{1}{18}\sqrt{563}\sqrt{3}-\frac{65}{54}\right)^{\frac{1}{3}}}}}+\frac{11}{6}}+\frac{1}{4}$$

Figure 12. Une des racines de h(x) de la section 5.7.10

Ensuite, nous pourrions utiliser QQ et obtenir les trois racines rationnelles, à savoir {−2, 2/3, 2}. Enfin, si nous utilisons ZZ, nous obtiendrions les deux racines entières, à savoir {−2, 2}.

Nous pourrions remplacer g(x) par

$$h(x) = x^4 - x^3 - x^2 - x - 1$$

et essayons le code ci-dessus en utilisant CC. Nous obtenons quatre racines algébriques explicites (dont le format est d'une complexité déconcertante). L'une d'elles est donnée dans la figure 12, et les autres ont une structure similaire. Si nous remplaçons CC par RR, nous constatons que deux de ces racines sont réelles, ce qui signifie que les deux autres doivent être complexes. Enfin, si nous remplaçons RR par QQ, nous constatons qu'aucune de ces racines n'est rationnelle.

5.8. Où allez-vous à partir d'ici ?

J'espère que ce chapitre vous a été utile et peut-être même agréable.
Apprendre à programmer peut ouvrir de nouvelles perspectives dans la recherche, le monde universitaire ou l'industrie. Cela peut également augmenter considérablement votre salaire.

5.8.1. Ressources supplémentaires sur la programmation Python Voici

quelques ressources qui peuvent vous permettre de développer vos compétences en programmation.

- Un excellent site web pour les personnes qui débutent en programmation et souhaitent se lancer avec Python a été écrit par Alan Gauld, à l'adresse ci-dessous. Le site web est un peu obsolète (le 4 juillet 2014, il indiquait que la dernière mise à jour datait du 27 mai 2007), mais je ne vois pas en quoi cela devrait avoir d'importance.

  http://www.freenetpages.co.uk/hp/alan.gauld/

- Un autre excellent livre électronique pour apprendre à programmer, avec Python comme langage, est Comment penser comme un informaticien : version Python par Jeffrey Elkner, Allen B. Downey et Chris Meyers.

  http://openbookproject.net/thinkcs/python/english2e/ • Un wikibook hautement recommandé sur Python pour ceux qui ne le savent pas savoir programmer peut être trouvé à

http://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for_Python

- (Je ne suis pas en mesure de juger entre ces trois premières ressources.) • Un site Web très intéressant de David Sklar et Bruce Cohen est « Just Enough Python for Numerical Analysis ». Ce grand site Web ou

Machine Translated by Google

Ce petit livre numérique n'est pas destiné à enseigner Python via Sage. Il se concentre plutôt sur l'utilisation directe de Python. Les exemples sont toutefois plus mathématiques que ceux des trois premiers livres ci-dessus. http://www.tetrahedra.net/

m400/index.html • Des ressources de niveau intermédiaire sont disponibles sur le site officiel de Python.

https://www.python.org/about/gettingstarted/

• Par Mike Pilgrim, « Dive into Python ! » est disponible en livre numérique, en site web et en livre imprimé. Il s'adresse toutefois aux programmeurs expérimentés qui connaissent d'autres langages, mais pas Python.

http://www.diveintopython.net/

5.8.2. Qu'est-ce qui a été laissé de côté ?

Selon l'université, nous avons couvert environ la moitié de ce qui est normalement enseigné dans un semestre d'informatique I. Ce qui suit est un résumé de ce qui serait enseigné pendant la seconde moitié, pour ceux d'entre vous qui sont curieux.

Types de données variables : En programmation informatique, les données peuvent prendre diverses formes. Les entiers, les nombres à virgule flottante (approximations des nombres réels) et les chaînes de caractères (messages entre guillemets) sont les plus courants. Dans la plupart des langages de programmation, vous devez déclarer vos variables et nommer leur type de données. Les déclarations de variables ne sont ni complexes ni déroutantes, mais elles sont importantes car votre programme ne fonctionnera pas sans elles dans ces langages. Le choix du bon type de données peut poser quelques problèmes subtils.

Opérateurs logiques : Nous avons vu l'utilisation de ou dans la section 5.7.6, mais un cours typique offrirait quelques situations plus difficiles où l'utilisation intelligente de ou ainsi que de et devient importante.

Interception des exceptions déclenchées : Lorsqu'une exception est générée par la commande raise, une sous-routine appelant la sous-routine fautive peut remédier à la situation. Le programme n'est pas arrêté, mais un code spécifique est exécuté.

Instructions If/Then/Else imbriquées : Le constructeur if-then-else est un excellent moyen de gérer deux possibilités. Pour trois ou quatre possibilités, on peut les enchaîner dans une construction « else if ».

En Python, la commande « elif » est une abréviation de « else if ».

Gestion des instructions Case/Switch : pour les situations où il peut y avoir une douzaine de possibilités ou plus, un long flux de commandes if-then-elif-elif-elif-else serait très difficile à lire et à déboguer.

La plupart des langages informatiques ont une manière de gérer des « cas » distincts et cela peut être utile à connaître.

Portée : La brève discussion (page 228) sur les « variables locales » par rapport aux « variables globales » n'est en fait qu'un petit aperçu d'un problème de taille moyenne.

Machine Translated by Google

appelé « scope ». Comprendre son fonctionnement peut être important dans le débogage de programmes informatiques difficiles.

Récursivité : En mathématiques comme en informatique, la récursivité se produit lorsqu'une fonction ou une sous-routine s'appelle elle-même. Par exemple, $n! = (n)(n-1)!$, ou $x^n = (x)(x^{n-1})$. Ce n'est pas différent d'une preuve par induction. Les exemples que j'ai donnés ici sont très simples, mais il existe d'autres exemples, plus sophistiqués, qui peuvent être mathématiquement plutôt puissant.

Travailler avec des chaînes : nous avons beaucoup travaillé avec des données numériques abordés tout au long de ce chapitre. De même, on peut travailler sur des cordes données (messages). Vous pouvez rechercher des chaînes, effectuer des substitutions, les formater correctement et réaliser toutes sortes de tâches intéressantes. c'est ainsi que fonctionnent les traitements de texte.

Assertions : elles constituent un excellent moyen de déboguer du code qui n'est pas faire exactement ce qu'il doit faire. Une assertion est une déclaration logique, quelque chose comme ce que vous donneriez à une instruction if, mais c'est une dont vous êtes relativement certain qu'elle doit être vraie à chaque exécution de la commande assert. Par exemple, assert (x>3). Si il s'avère que l'affirmation est fausse, le programme immédiatement s'arrête complètement et vous êtes averti que l'assertion a échoué. C'est un excellent moyen de trouver des situations dans lesquelles vous avez supposé que quelque chose serait toujours vrai, mais cette hypothèse n'est pas toujours vraie. en pratique.

Autres techniques de débogage : Apprendre à déboguer des programmes est la C'est la tâche la plus vitale pour un programmeur. En pratique, elle consomme la majeure partie du temps de développement. La plupart des programmeurs développent Cette compétence se développe lentement au fil du temps, mais elle peut être un défi agréable. En fait, certaines questions d'examen amusantes peuvent consister à présenter à un étudiant une tâche et un code qui exécute presque cette tâche, mais pas exactement. L'étudiant doit identifier le bug et identifier comment modifier le code afin qu'il fonctionne correctement.

Jouer à l'interprète/compilateur : Un autre excellent exercice est pour le On donne un code à l'étudiant et on lui demande : « À quoi ça sert ? » L'étudiant doit faire semblant d'être un ordinateur et parcourir le code suivant le flux de contrôle. C'est aussi une excellente façon d'apprendre algorithmes nouveaux et compliqués.

Simulations : En général, une simulation modélise des phénomènes techniques, physiques, chimiques, biologiques, voire sociologiques complexes. Comment une entité aussi complexe peut-elle être modélisée sur ordinateur ? L'idée est de découper la situation en de nombreuses (peut-être un million) petites tranches de temps. Chaque tranche de temps peut ensuite être modélisée avec un modèle peu précis, comme un ensemble de fonctions linéaires. Parce que chaque tranche de temps est petite et qu'il y en a tellement la simulation peut néanmoins produire un résultat de haute fidélité. est fondamentalement le concept d'intégrale appliqué à l'informatique.

Machine Translated by Google

De nombreuses situations complexes peuvent être abordées de cette façon. L'art de la simulation est l'idée la plus simple et pourtant la plus utile de cette liste.

Remarque : Si un système est divisé en minuscules régions d'espace au lieu de temps, cela s'appelle « la méthode des éléments finis » et constitue la seule « technique décisive » utilisée aujourd'hui pour les situations d'ingénierie difficiles.

De nombreuses avancées technologiques des deux dernières décennies n'ont été possibles que grâce à la méthode des éléments finis. Malheureusement, cette méthode est presque toujours enseignée uniquement aux étudiants de troisième cycle, et non aux étudiants de premier cycle, et j'ignore pourquoi.

Programmation orientée objet : Il est difficile de décrire la programmation orientée objet en quelques mots. Je vais néanmoins tenter le coup. À ce stade, vous savez ce que sont les données et les sous-routines. Un ensemble de sous-routines (appelées « méthodes ») et de variables contenant des données (appelées « champs ») sont regroupées pour former une « classe ». Par exemple, dans le système d'un service de registraire, il pourrait y avoir une classe appelée « StudentClass » avec des variables pour le nom, la moyenne générale et le numéro d'étudiant. Il y aurait ensuite plusieurs objets StudentClass, un pour chaque étudiant de l'université. Puisque les données et le code sont regroupés, vous pouvez effectuer efficacement de nombreuses tâches qui seraient autrement plus difficiles et fastidieuses.

Naturellement, pour les universités qui utilisent des trimestres ou des trimestres, la répartition serait quelque peu différente.

Plus important encore, dans un véritable cours d'informatique, vous auriez eu droit à une plus grande quantité d'exemples à étudier et d'exercices à programmer. Ces deux éléments sont extrêmement précieux et constituent une part importante du développement d'un programmeur.

Machine Translated by Google

Machine Translated by Google

Chapitre 6

Bâtiment interactif Pages Web avec Sage

Dans ce chapitre, nous allons apprendre à construire une page Web interactive, parfois appelée « interaction » ou « application », qui peut être publiée sur le Web.
Contrairement à d'autres utilisations de Sage, cette fonctionnalité particulière est assez puissante pour une raison simple : toute personne ayant accès à Internet peut trouver votre page Web interactive. Sans aucune connaissance de Sage, ni même de mathématiques, ils peuvent déplacer des curseurs et observer l'évolution d'un graphique, ou observer le développement d'un autre concept mathématique. Comme de nombreux élèves sont des apprenants visuels, ce concept de page web interactive peut être d'une aide précieuse pour ceux qui, autrement, auraient des difficultés à apprendre certains éléments de mathématiques.

Par exemple, les étudiants des classes de niveau inférieur (à l'université ou même aux lycées et collèges) peuvent recevoir l'URL, et les étudiants peuvent commencer à utiliser la page Web interactive avec très peu d'intervention de l'enseignant.
Aucune expérience avec Sage ou tout autre logiciel d'algèbre informatique n'est requise.
En fait, la plupart des étudiants n'auront aucune idée que Sage a été impliqué.

Personnellement, j'ai travaillé avec Sage pendant 5 ans et 10 mois avant d'apprendre à créer des interactions pendant les vacances d'hiver 2012-2013.

Quand j'y suis enfin parvenu, j'ai été étonné de la simplicité de la création. Si j'avais compris plus tôt à quel point il est facile de créer des pages web interactives, j'aurais certainement commencé bien des années plus tôt.

Nos exemples

Ce chapitre fera référence à cinq exemples mathématiques divers mais élémentaires :

Tangente : Ceci est notre exemple principal. Je développerai progressivement cette interaction à partir de zéro, au fur et à mesure que nous progresserons dans le chapitre. L'objectif est de tracer une tangente mobile à un polynôme simple du second degré. Je détaillerai la construction de cette applet en utilisant mon processus en six étapes.

Machine Translated by Google

Onde sinusoïdale : Cet exemple constitue mon principal défi. Après avoir exploré l'interaction tangente-ligne, je vous invite à la reproduire comme un défi personnel. En cas de problème, j'ai fourni mon propre code pour chaque étape afin que vous puissiez vous y référer.

Aquarium optimal : Dans la section 5.2.3, page 226, nous avons développé une fonction permettant de calculer le coût d'un aquarium en fonction de ses dimensions. J'ai pensé qu'il serait intéressant de fournir un code permettant de développer cette fonction en une page web interactive. Il s'avère que le processus est relativement simple.

Polynomial Grapher : Il s'agit d'une applet très rapide destinée à vous montrer comment créer des menus déroulants ou des cases à cocher pour vos interactions.
C'est très simple et rapide. Vous disposez d'un menu déroulant proposant trois polynômes, et vous pouvez indiquer si vous souhaitez ou non des lignes de quadrillage dans votre graphique en cochant une case.

Intégrale définie : Avant de faire interagir l'onde sinusoïdale, j'avais pensé que c'était un bon défi pour le lecteur : le rôle désormais joué par l'« onde sinusoïdale ». Il s'avère que cette tâche comporte quelques subtilités techniques, et elle est un peu plus difficile qu'on pourrait le penser. Je la laisse comme défi aux lecteurs particulièrement motivés.

Où trouver les fichiers :

Les fichiers de ces cinq grands exemples sont disponibles sur ma page web. Rendez-vous sur http://www.gregorybard.com/, cliquez sur « Sage Stuff » et faites défiler la page vers le bas. Vous y trouverez un fichier zip et un répertoire pour chacune de ces cinq interactions.

6.1. Le processus en six étapes pour créer des interactions

Le processus en plusieurs étapes suivant décrit la manière dont je procède pour créer les interactions, et c'est ainsi que notre explication sera structurée. Nous observerons notamment comment l'interaction « Ligne tangente » a été développée à travers chacune de ces étapes successives.

Étape 0 : Développement du concept : prenez le temps de définir précisément ce que vous souhaitez que l'interaction fasse. Considérez cela non seulement comme une mise en page, mais aussi comme un modèle de comportement. (Si je clique ici, ceci se produit ; si je clique là, ceci se produit, etc.)

Étape 1 : Concevoir une fonction dans Sage (en utilisant def) qui représente un tour, cycle ou phase de l'interaction.

Étape 2 : peaufinez cette fonction jusqu'à ce qu'elle soit parfaite, avec toutes les entrées et sorties comme vous le souhaitez, y compris les couleurs, etc.

Étape 3 : Convertissez cette fonction en fonction d'interaction dans Sage Cloud, à l'aide des commandes « @interact » et « slider ». Nous allons

Machine Translated by Google

Décrivez ce processus en détail. Il est bien plus simple qu'il n'y paraît. La commande du curseur fait tout le travail.

Étape 4 : Insérez cette fonction interactive dans un modèle de page web. J'utilise un modèle depuis longtemps et vous pouvez simplement copier-coller votre code dans celui-ci.

Étape 5 : Complétez la page web comme n'importe quelle autre page web, en ajoutant quelques paragraphes sur les mathématiques, des instructions et éventuellement une motivation. Vous pouvez modifier cette page web comme n'importe quelle autre. Enfin, téléchargez cette nouvelle page web sur votre site web, comme vous le feriez pour n'importe quelle autre page web.

Les prérequis de la construction d'interactions : En écrivant

ce chapitre, j'ai essayé de viser le public le plus large possible.
Fondamentalement, il y a quatre ingrédients nécessaires dans votre parcours pour vous permettre de lire ce chapitre.

(1) Ce chapitre suppose une connaissance très basique du langage HTML. Seule une connaissance minimale de ce langage, suffisante pour permettre à l'utilisateur de créer ses propres pages web (ordinaires, non interactives), est requise.

(2) Il est nécessaire d'avoir une connaissance générale de Sage, notamment des fonctionnalités et du contenu de la majeure partie du chapitre 1 de ce livre.

(3) De plus, nous programmerons en Sage via Python ; il serait donc utile d'avoir lu également le chapitre 5. Cependant, nous n'utiliserons que nos propres fonctions def et quelques constructions de contrôle de flux ; une compréhension superficielle du chapitre 5 sera donc suffisante (par exemple, les chapitres 5.1 et 5.2). De plus, il est à noter que si le lecteur possède une bonne expérience de n'importe quel langage de programmation (par exemple, C, C++, Java, Perl, Fortran, etc.), il sera capable de lire le code Python et de comprendre ce qui se passe sans aucune connaissance préalable de Python ou du chapitre 5.

(4) Enfin, le lecteur doit être au moins quelque peu familier avec le Interface Sage Cloud.

6.2. L'interaction tangente-ligne

Nous allons maintenant suivre l'interaction « Tangente-Ligne » telle que je l'ai construite à travers ce processus en six étapes.

Étape 0 : Développement du concept. Avant de

coder quoi que ce soit, j'essaie de définir précisément ce que je veux que l'interaction fasse. En général, j'ai trois choses en tête.

1Fourni par le professeur Jason Grout.

Machine Translated by Google

Tout d'abord, je souhaite trouver le « meilleur » exemple possible pour communiquer l'idée mathématique : il ne doit pas être trop complexe, mais pas trop simple non plus. Dans ce cas, nous souhaitons tracer une tangente et montrer à un étudiant débutant en calcul (peut-être moins de trois semaines après le début du cours) à quoi ressemble une tangente. Par conséquent, il est pédagogiquement utile de choisir une fonction extrêmement simple, comme un polynôme, comme le montre le schéma ci-dessous. J'ai choisi la fonction f(x) = x. Ensuite, j'essaie de me concentrer sur ce que l'utilisateur peut modifier $^3 - x$.

et ce qu'il ne doit pas être autorisé à modifier. Ces éléments constitueront les curseurs de l'interaction. L'utilisateur doit pouvoir modifier l'abscisse du point où la tangente est tracée, mais pas l'ordonnée, qui doit être calculée par l'interaction. De cette façon, l'utilisateur ne choisira jamais un point qui ne soit pas sur la courbe y = x. Troisièmement, j'aime souvent faire un croquis de ce à quoi ressemblera le résultat. Je me demande comment présenter les circonstances de manière optimale, en fournissant $^3 - x$.

suffisamment de détails, sans toutefois perturber l'étudiant. Maintenant que mon concept est finalisé, je peux me connecter à Sage Cloud et commencer à travailler.

Étape 1 : Conception d'une fonction Sage. Le code de l'étape

1 de l'interaction de la tangente est présenté dans la figure 1. − x, puis je déclare une seconde fonction pour la dérivée. J'ai

Comme vous pouvez le voir, je déclare d'abord la fonction f(x) = x ^{3} demandé à Sage de calculer la dérivée, mais j'aurais pu simplement définir f(x) = 3x − 1.

^{2}

plutôt.

L'idée de demander à Sage de calculer la dérivée pour moi est que peut-être plus tard, je pourrais vouloir utiliser d'autres fonctions, comme f(x) = sin x par exemple.
Le fait que Sage calcule la dérivée pour moi exclut la possibilité que j'oublie de changer la dérivée de f (x) = 3x 2−1 en f (x) = cos x,
dans ce cas.

Ensuite, j'utilise une commande def pour définir ma propre fonction tangente au point.
C'est le cœur de l'étape 1 de mon processus de création d'applet. La fonction conçue à cette étape représente le noyau de l'interaction.
Il évoluera lentement vers une page Web interactive fonctionnelle.

Naturellement, une tangente peut être produite à partir de trois éléments : la pente, l'abscisse et l'ordonnée. L'abscisse sera générée par un curseur une fois l'interaction terminée, mais il s'agit désormais d'un nombre réel (appelé x0) pour la fonction créée avec la commande def. La plupart des interactions comportent plusieurs curseurs, car elles nécessitent davantage de paramètres.

Nous devons calculer nous-mêmes l'ordonnée et la pente. Ces valeurs proviennent de l'évaluation de f(x) et f(x) en x0. Une fois la pente de la tangente connue, nous avons besoin de l'ordonnée à l'origine. Pour enseigner cela, je demande aux élèves d'utiliser la formule point-pente :

$$y - y0 = m(x - x0)$$

Machine Translated by Google


```
f(x) = x^3 - x

f_prime(x) = diff( f(x), x)

def tangent_at_point( x_0 ): y_0 = f( x_0 )
        m = f_prime( x_0 )


        # parce que y - y_0 = m(x - x_0) nous savons que # y - y_0 = m*x -
        m*x_0 # y = m*x + y_0 - m*x_0 #
        donc b = y_0 - m*x_0


        b = y_0 - m*x_0

        P1 = plot( f(x), -2, 2, couleur='bleu')
        P2 = plot( m*x + b, -2, 2, couleur='tan')
        P3 = point( (x_0, y_0), couleur='rouge', taille=100)

        P = P1+P2+P3

        P.show()

        retour
```

Figure 1. Le code de l'étape 1 de « La ligne tangente » Interact

et ensuite utiliser l'algèbre pour convertir cela sous la forme y = mx + b. Cependant, lors de l'écriture d'un programme informatique, il est nécessaire de disposer d'une formule explicite pour l'ordonnée à l'origine, notée ici b.

Les quatre lignes qui suivent sont des « commentaires » et n'ont aucun effet sur l'ordinateur. Elles sont destinées à indiquer à l'utilisateur l'effet attendu d'un code proche, dans l'esprit du programmeur. Je souligne que les commentaires ne sont pas des paroles infaillibles d'un être omniscient, mais plutôt une déclaration de l'intention du programmeur. Toute ligne commençant par un # est un commentaire et sera ignorée par l'ordinateur. J'utilise ici ce symbole pour dériver ma formule abrégée : b = y0 − mx0.

Trois commandes de tracé suivent. Premièrement, nous voulons représenter la fonction f(x) sur l'intervalle −2 < x < 2, et j'ai choisi la couleur bleue. Deuxièmement, nous voulons représenter la tangente y = mx + b sur ce même intervalle, et j'ai choisi la couleur beige, car cela ressemble au début de « tangente ».
Ensuite, j'ai ajouté un gros point rouge (un point coloré) à leur intersection, pour aider l'utilisateur à voir où il doit regarder. J'ai choisi la couleur rouge, car

Machine Translated by Google

Cela signifie généralement quelque chose d'important. Rappelons que la superposition de tracés dans Sage se fait par « addition ». J'additionne donc ces trois tracés.

Enfin, je dois afficher le tracé superposé, et la fonction est enfin terminée. Cette conclusion est indiquée par une commande de retour.

Si vous avez besoin de quelques rappels sur l'ajout de tracés, pour réaliser la superposition de tracés, voir page 18. Pour des rappels sur la commande de point, voir page 15.

Étape 2 : Peaufiner cette fonction Une fois

votre fonction fonctionnelle, il est judicieux de la peaufiner un moment pour qu'elle soit exactement comme vous le souhaitez. Ce n'est pas parce qu'il est impossible d'apporter des modifications ultérieurement. J'ai même apporté des modifications après l'étape 5. Cependant, il devient de plus en plus irritant pour le programmeur d'apporter des modifications dans les étapes ultérieures. Par conséquent, prenez le temps de bien définir les couleurs, les limites du graphique et tout le reste.

Dans ce cas particulier, j'ai testé ma fonction avec ce qui suit :

```
tangente_au_point(0,5)
tangente_au_point(1,5)
tangente_au_point(-1,75)
```

Même si l'affichage était loin d'être laid, il y avait place à l'amélioration.
Par exemple, dans le dernier graphique, la tangente est relativement raide (avec une pente de 8,1875), ce qui fait qu'elle monte jusqu'à $y = 25$ avant de quitter le graphique. La courbe cubique apparaît alors minuscule, car elle ne monte que jusqu'à $y = 6$. Il me semble donc nécessaire de borner les ordonnées des graphiques. J'ai procédé de la sorte pour $f(x)$ et la tangente, en les limitant à $-6 < y < 6$, car c'est l'étendue de $f(x) = x^3 - x$ sur le domaine $-2 < x < 2$.

Si vous regardez le code de la Figure 2, vous pouvez voir les options ymin et ymax ajoutées dans les deux premières commandes plot.

Ensuite, j'ai trouvé le point rouge un peu grand, j'ai donc réduit sa taille de 100 à 50. Notez qu'il s'agit de la surface du point et que cela ne réduit pas le diamètre de moitié. Puisque la surface du point est divisée par deux, le diamètre est réduit de $1/\sqrt{2} \approx 0{,}707106\cdots$ J'ai également pensé qu'un arrière-plan de grille serait utile

pour le tracé. Si vous regardez la commande pour P1, vous pouvez voir que j'ai ajouté la commande gridlines='minor' à cette commande de tracé. Puisqu'il s'agit du « bas » des trois tracés superposés, il apparaîtra comme arrière-plan sur la superposition composite.

J'ai également pensé qu'il serait intéressant de partager avec l'utilisateur les valeurs numériques de x0, y0, m et l'équation de la tangente. Vous pouvez voir de nouvelles commandes d'impression après l'instruction P.show(). La syntaxe exacte de l'interaction entre les virgules et les guillemets peut prêter à confusion. Ne vous laissez pas décourager : cela n'a aucune importance pour notre progression.

Machine Translated by Google


```
f(x) = x^3 - x

f_prime(x) = diff( f(x), x)

def tangent_at_point( x_0 ): y_0 = f( x_0 )
        m = f_prime( x_0 )


        # parce que y - y_0 = m(x - x_0) nous savons que # y - y_0 = mx
        - m*x_0 # y = mx + y_0 - m*x_0
        # donc b = y_0 - m*x_0


        b = y_0 - m*x_0

        P1 = plot( f(x), -2, 2, color='bleu', ymin=-6, ymax=6,
                    gridlines='mineur')
        P2 = plot( m*x + b, -2, 2, color='tan', ymin=-6, ymax=6)
        P3 = point( (x_0, y_0), couleur='rouge', taille=50)

        P = P1+P2+P3

        P.show()


        imprimer imprimer "x_0 = ",
        x_0 imprimer "y_0 = f(x_0) = ", y_0
        imprimer "m = f'(x_0) = ", m imprimer
        "tangente : y = (", m, ")*x + (", b, ")"

        retour
```

Figure 2. Le code de l'étape 2 de « La ligne tangente » Interact

Ensuite, j'ai réédité les cas de test x = 0,5, x = 1,5 et x = −1,75. Ils ça a l'air beaucoup mieux maintenant, il est donc temps de passer à l'étape 3.

Étape 3 : Conversion en fonction interactive Nous allons maintenant découvrir la commande @interact et comment créer des curseurs pour les paramètres de notre sous-routine. Trois modifications seront apportées au code, dont deux mineures, mais une nécessite quelques explications.

Machine Translated by Google

En particulier, les quatre premières lignes

```
f_prime(x) = diff( f(x), x)

def tangent_at_point( x_0 ): y_0 = f( x_0 )
```

ont été à ce moment-là changés en

```
f_prime(x) = diff( f(x), x)

@interact def
tangent_at_point( x_0 = slider(-2, 2, 0.1, -1.5,
                    label="x-coordonnée") ): y_0 =
    f( x_0 )
```

Tout d'abord, vous pouvez constater que j'ai ajouté la commande @interact, sur une ligne distincte, au-dessus de la commande def qui démarre notre fonction. Cette commande signale à Sage que nous créons une interaction.

Deuxièmement, et cela nécessite quelques explications, j'ai utilisé la commande slider pour transformer x0 d'une entrée numérique en un curseur manipulable par l'utilisateur. Les deux premiers paramètres indiquent à Sage que la variable x0 sera limitée à l'intervalle $-2 \leq x \leq 2$. Le troisième paramètre indique à Sage la granularité du curseur. Dans ce cas, nous souhaitons que chaque déplacement représente 1/10.
Ainsi, si le curseur est à x0 = 0,5, les deux positions de chaque côté seront 0,4 et 0,6. Le quatrième paramètre représente la condition initiale. Dans ce cas, le curseur démarrera avec x0 défini sur −1,5, mais l'utilisateur pourra le déplacer vers d'autres valeurs. Enfin, le cinquième paramètre (facultatif) donnera un libellé lisible au curseur.

Troisièmement, dans Sage Cloud, je n'appellerai plus la fonction avec des commandes comme tangent at point(-1.75). À la place, je saisirai tangent at point(). En effet, je ne transmets pas x0 comme valeur à la fonction, mais j'utilise plutôt un curseur pour modifier fréquemment cette valeur.

Nous disposons désormais d'une fonction interactive dans Sage Cloud. Prenons un moment pour l'expérimenter et voir comment elle fonctionne. Notez que lorsque vous cliquez et faites glisser le curseur, vous devez relâcher le bouton de la souris (ou du pavé tactile) pour que la fonction interactive réagisse.

Pour que vous puissiez voir à quoi cela ressemblera, j'ai mis une capture d'écran de l'interaction dans la figure 3.

Étape 4 : Insérer dans un modèle web. À cette étape, je copie-colle ma fonction interactive depuis Sage Cloud dans un fichier HTML, créant ainsi une page web interactive rudimentaire. Vous pouvez utiliser le modèle HTML de votre choix pour votre page web interactive ; j'en ai toutefois mis un à votre disposition. Ce modèle se trouve sous le nom « generictemplate.html » dans l'un des répertoires de projet sur Sage Cloud pour ce chapitre. Il est également présenté à la figure 5.

Machine Translated by Google

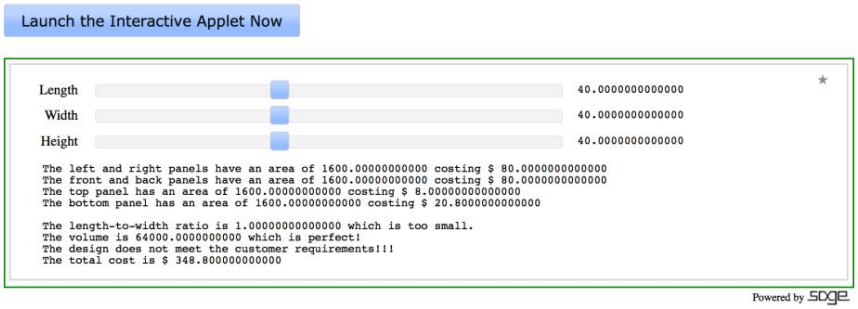


Figure 3. Capture d'écran de l'Aquarium Interact

Je suis sûr que le lecteur sait copier-coller, mais il arrive que des erreurs mineures surviennent. C'est pourquoi, au risque de paraître puéril, je vais maintenant détailler les étapes très précisément. Ne vous sentez pas offensé.

Les étapes exactes pour copier-coller dans le modèle

(1) J'ai mis en évidence la fonction interactive de l'étape 3 dans Sage Cloud.
(2) J'ai donné la commande « copier » à mon navigateur. (Comme j'utilise un Mac, il s'agit de « commande-C », mais ce serait différent si vous utilisez UNIX, LINUX ou Windows.)
(3) J'ai repéré l'emplacement « Vous devez copier-coller votre fonction interactive Sage ici » dans le fichier HTML du modèle générique et je l'ai surligné, y compris les lignes marquées du signe #. (Pour modifier les fichiers HTML, j'utilise TextWrangler version 3.5.3, mais vous pouvez utiliser ce que vous voulez, même Sage Cloud.)
(4) J'ai donné la commande « coller » à mon éditeur HTML.
(5) J'ai enregistré le fichier à l'aide de la commande « Enregistrer sous » sous un nouveau nom de fichier, dans ce cas, Stage4.html.
(6) J'ai changé « YourTitleGoesHere, pour la première fois » en « Exploration de la ligne tangente ».
(7) J'ai changé « YourTitleGoesHere, pour la deuxième fois » en « Explorer-la ligne tangente.
(8) J'ai changé « VotreNomValci » en « Prof. Gregory V. Bard ».
(9) J'ai changé « PutTheDateHere » en « 28 décembre 2013 ».

Vous pouvez maintenant visualiser le fichier HTML dans votre navigateur. Comme vous pouvez le constater, il est très simple : il ne s'agit que d'un squelette. Cliquez sur le gros bouton bleu « Lancer l'applet interactif maintenant » pour afficher notre interface interactive. Prenez un moment pour l'admirer. Nous sommes passés d'un simple concept à une véritable page web interactive !

Machine Translated by Google

Étape 5 : Élaborer la page web. À ce stade,

vous pouvez modifier la page web comme n'importe quelle autre page web. Dans le modèle, j'ai prévu des espaces pour « Aperçu », « Instructions » et « Discussion ». Bien sûr, vous pouvez ajouter autant de titres que vous le souhaitez. De même, vous pouvez insérer autant de paragraphes que vous le souhaitez sous chaque titre.

La page web interactive finale, fruit de l'étape 5, est disponible sur mon site web. Rendez-vous sur gregorybard.com et cliquez sur « Pages web interactives pour l'apprentissage des mathématiques ». Vous trouverez l'interaction sous le titre « Exploration de la tangente ».

6.3. Un défi au lecteur

Vous avez maintenant compris comment créer une page web interactive dans Sage. Je vous encourage vivement, voire vous supplie, à prendre un moment pour l'essayer vous-même. C'est le meilleur moyen de vérifier votre compréhension ou d'identifier d'éventuelles lacunes.

Vous pouvez choisir la tâche qui vous convient, mais je recommande la suivante : vous souhaitez peut-être enseigner les ondes sinusoïdales telles qu'elles se produisent dans la lumière, l'électricité, le son, etc. Par exemple, vous souhaitez peut-être expliquer à l'élève le rôle de l'amplitude et de la vitesse angulaire (radians par seconde).
En résumé, imaginez que vous souhaitez créer une interaction qui représentera graphiquement

$$f(t) = A \sin(\omega t)$$

où A et ω sont contrôlés par des curseurs.

Pour le tracé, je recommande d'utiliser $-10 < x < 10$ et $-5 < y < 5$. Pour l'amplitude, bien que l'on puisse voir des avantages à forcer l'amplitude à être positive, je pense qu'il serait intéressant pour les élèves d'apprendre l'effet de $A \leq 0$, donc je recommande $-5 \leq A \leq 5$ volts. Pour ω, il peut être difficile d'expliquer ce que $\omega = 0$ représente, donc $0{,}5 \leq \omega \leq 5$ est peut-être un bon choix. J'avais A dans une granularité où chaque pas était de 1/4 de volt et ω dans une granularité de 1/2 radian par seconde.

Essayez et, si vous le souhaitez, comparez votre travail avec mes fichiers. Je vous recommande de terminer les cinq étapes avant de comparer votre travail au mien. Cependant, si vous êtes bloqué, n'hésitez pas à jeter un œil à mon code plus tôt dans le processus.

6.4. L'interaction optimale avec l'aquarium

L'objectif principal est de présenter à un étudiant une question au début d'un cours (ou d'une unité d'un cours) qu'il (1) sera capable de comprendre

à ce moment-là, (2) qu'il ne pourra pas résoudre à ce moment-
là, (3) qu'il voudra peut-être résoudre, et

Machine Translated by Google

(4) qu'ils seront capables de résoudre après avoir terminé ce cours (ou (terminer cette unité du cours).

Dans ce cas, le cours est Calcul I et l'unité est l'optimisation. Nous présentons ici un problème nécessitant les dimensions nécessaires à la conception d'un aquarium. L'objectif est de construire un aquarium de 64 000 pouces cubes, avec un rapport longueur/largeur de 2,0, mais à un coût minimal. Le coût du dessus de l'aquarium est de 0,5 centime par pouce carré, des côtés de 5 centimes par pouce carré et du fond de 1,3 centime par pouce carré. La conception de l'aquarium2 doit respecter les deux exigences de volume et de rapport longueur/largeur (à 1 % près), tout en minimisant les coûts.

Bien sûr, ce problème est facile à résoudre avec le calcul différentiel et intégral, mais l'objectif est de montrer qu'il est difficile à résoudre sans lui, par la seule intuition. Imaginons une classe remplie d'élèves, dont la plupart ne seraient pas capables de répondre simultanément aux exigences de volume et de rapport longueur/largeur. Parmi ceux qui ont satisfait à ces deux exigences, la plupart s'arrêteront là, ignorant la minimisation des coûts. On peut alors demander aux élèves quels coûts ils ont réussi à obtenir, et il apparaîtra clairement que ce coût n'est pas le même pour tous. Ensuite, le professeur pourrait énoncer la solution optimale afin que beaucoup d'élèves constatent qu'elle est effectivement moins chère que la leur, mais peut-être pas de beaucoup. Enfin, le professeur devrait résoudre le problème avec des notions explicites d'algèbre et de calcul, plusieurs leçons plus tard, une fois le module d'optimisation terminé.

6.5. Sélecteurs et cases à cocher

Vous trouverez ci-dessous une interaction très simple permettant de représenter graphiquement quelques polynômes. Je l'ai écrite rapidement pour vous montrer comment créer des menus déroulants ou des cases à cocher pour vos interactions dans Sage. Voici le code :

```
@interagir
def polynomial_grapher( mypoly =
                selector( [ x^2-4, x^3-x, x^2+1 ], label="Polynôme : " ), show_grid = False ): if
                (show_grid == True):

            P = plot( mypoly, -3, 3, ymin = -5, ymax = 5, gridlines="minor" )

        autre:
            P = tracé (monpoly, -3, 3, ymin = -5, ymax = 5)

        montrer(P)
```

Nous voyons deux nouvelles commandes. La première, selector, crée un menu déroulant avec les choix correspondant aux trois polynômes indiqués dans le code. L'étiquette

---

2Pour une classe avec des élèves d'âge moyen qui « retournent à l'école », le même problème peut être posé sous la forme de la conception d'un entrepôt de fabrication bon marché avec des coûts pour le sol, le toit et les murs.

Machine Translated by Google

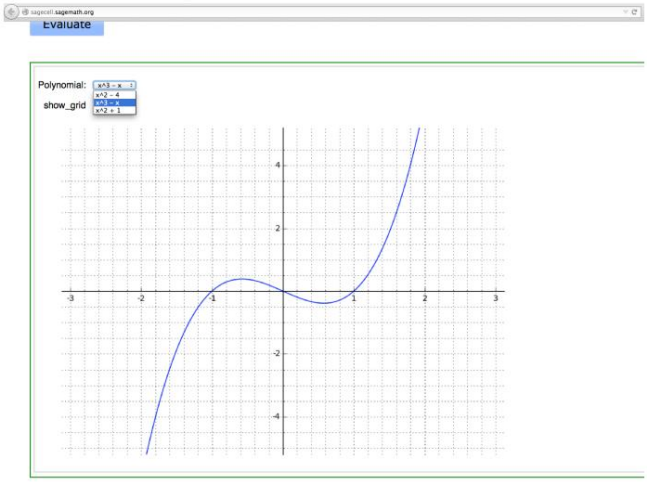


Figure 4. Capture d'écran de l'applet Polynomial Grapher

Le mot « Polynomial : » doit être utilisé pour que l'utilisateur comprenne l'objectif du menu déroulant. (Je suppose que dans ce cas, c'était évident, mais peut-être que dans d'autres, ce n'est pas évident.)

Deuxièmement, nous avons un paramètre facultatif afficher la grille qui est défini par défaut sur Faux, tout comme nous avions défini par défaut le paramètre verbose sur Faux à la page 244. Chaque fois que vous avez un paramètre facultatif dans une interaction Sage qui peut être Vrai ou Faux, Sage sait que vous voulez avoir une case qui peut être cochée ou décochée. Le tracé du polynôme sélectionné sera tracé avec ou sans quadrillage, selon que « show grid » est défini sur « Vrai » ou « Faux ». Le reste du code n'est pas nouveau pour nous.

La figure 4 présente une capture d'écran de cette applet, mais notez que la case à cocher a été éclipsée par le menu déroulant.

6.6. L'interaction intégrale définie

À première vue, cette interaction paraît très simple. Nous souhaitons expliquer la signification de l'intégrale définie et avons choisi une fonction simple : $y = x^2 - 1$. Les bornes d'intégration sont initialement $-1{,}5 < x < 1$, mais l'utilisateur peut modifier l'une ou l'autre de ces bornes. La zone comprise dans l'intégrale définie est grisée, afin d'expliquer à l'élève que l'on ombrage de la courbe vers l'axe des x.

Grâce à la technique de « tracé d'intégrales » décrite à la section 3.1.4, page 95, nous pouvons créer un tel tracé dans Sage. À première vue, cette interaction semble donc simple à créer.

Il s'avère que ce n'est pas tout à fait vrai. Il faut d'abord traiter le cas où l'étudiant intervertit les bornes inférieure et supérieure. Cela entraînera

Machine Translated by Google

zone (signée) que l'applet calcule comme ayant le signe opposé, car

$$\int_a^b f(x)\,dx = -\int_b^a f(x)\,dx$$

Deuxièmement, Sage réagit mal si la zone à ombrager est nulle. Autrement dit, si a = b, les commandes de remplissage utilisées par la technique de tracé intégral sont perturbées et génèrent un message d'erreur. Je modifie donc artificiellement la largeur pour qu'elle soit 10−12 supérieure à la normale. Par conséquent, la largeur n'est jamais nulle et la plainte est supprimée.

Finalement, j'ai décidé d'ajouter une grande flèche. Si les bornes sont dans l'ordre normal, de gauche à droite, je dessine une flèche verte dans cette direction ; si les bornes sont « inverses », ou de droite à gauche, je dessine une flèche rouge pointant de droite à gauche. Cela permettra d'expliquer aux élèves ce qui se passe si l'on intervertit les bornes d'intégration.

Comme vous pouvez le constater, certains de ces changements sont quelque peu inattendus. J'ai donc jugé judicieux d'inclure l'intégralité du code (pour toutes les étapes intermédiaires) afin que vous puissiez suivre l'évolution du projet.

Machine Translated by Google


```
<!DOCTYPE HTML>
<html>
    <tête>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width"> <title>blah
        VotreTitreValci, pour la première fois</title> <script src="https://sagecell.sagemath.org/
        static/jquery.min.js"></script> <script src="https://sagecell.sagemath.org/embedded_sagecell.js"></script> <script>
        $(function () { // Transformer *n'importe quel* div avec la classe 'compute' en cellule Sage


sagecell.makeSagecell({inputLocation: 'div.compute', template:

                                        sagecell.templates.minimal,
                                                evalButtonText: 'Lancer l'applet interactive maintenant'});
});
        </script> </
    head>
    <body style="width: 1000px;">

<! vous pouvez commencer à modifier en dessous de ce point!>
<! vous pouvez commencer à modifier en dessous de ce point!>
<! vous pouvez commencer à modifier en dessous de ce point!>

    <h1>blah YourTitleGoesHere, pour la deuxième fois</h1>

<p>Une applet interactive propulsée par Sage et MathJax.</p> <p>(Par
YourNameGoesHere)</p>

<hr>

<h2>Aperçu</h2> <p>bla
bla bla</p> <p>bla bla bla</p>
<p>bla bla bla</p>


<h2>Instructions</h2> <p>bla
bla bla</p> <p>bla bla bla</p>


<div class="compute"> <script
type="text/x-sage">

##
## Vous devez copier-coller votre fonction interactive Sage ici. ##


</script> </
div>

<h2>Discussion</h2> <p>bla
bla bla</p> <p>bla bla bla</p>
<p>bla bla bla</p>


<hr>
Dernière modification le PutTheDateHere.
</body> </
html>
```

Figure 5. Le modèle HTML suggéré pour une page Web interactive, basé sur les travaux du professeur Jason Grout.

Machine Translated by Google

Annexe A

Que faire en cas de frustration !

Les ordinateurs sont impitoyables en matière de syntaxe. Pour les utilisateurs peu habitués à cette rigueur, cela peut être très frustrant. Voici une liste de conseils à suivre lorsque Sage rejette votre code pour des raisons que vous ne pouvez pas déterminer. J'espère que ces 16 questions couvriront la plupart des cas.

1 : Faites-vous de la multiplication implicite ?

Ce cas particulier est mieux expliqué par un exemple :

```
résoudre( x^3 - 5x^2 + 6x == 0, x )
```

n'est pas correct. Vous devez plutôt saisir

```
résoudre( x^3 - 5*x^2 + 6*x == 0, x )
```

Autrement dit, Sage a besoin de l'astérisque ou du symbole de multiplication entre les coefficients 5 et 6, ainsi que des termes auxquels ils sont rattachés : x et x. Pour une raison inconnue, les fonctions trigonométriques sont source de confusion pour certains utilisateurs. Sage rejettera

```
f(x) = 5cos( 6x )
```

mais acceptera f(x)

```
= 5*cos( 6*x )
```

Cependant, pour les utilisateurs qui trouvent cela extrêmement frustrant, il existe une solution à cette exigence. Si vous placez la ligne suivante

```
implicit_multiplication(True) au début d'une
```

session Sage Cloud, vous serez alors en « mode multiplication implicite ».

Considérez le code suivant :

```
implicit_multiplication(True) g=(12+3x)/
(-12+2x) plot( g, 2, 8,
ymin=-30, ymax=30 )
```

Machine Translated by Google

À ce stade précis (18 juillet 2014), cela fonctionnera dans Sage Cloud, mais pas dans Sage Cell Server. Notez que la définition de g ci-dessus ne comporte pas d'astérisque entre le 3 et le premier x, ni entre le 2 et le deuxième x. Comme nous sommes en « mode multiplication implicite », ces deux astérisques manquants sont tolérés.

2 : Avez-vous oublié de déclarer une variable ?

La déclaration des variables peut être déroutante pour ceux qui n'ont pas beaucoup programmé avant d'apprendre Sage. Dans des langages comme C, C++ et Java, toutes les variables doivent être déclarées, sans exception. Dans Sage, la variable x est pré-déclarée. Toutes les autres variables doivent être déclarées, à l'exception des constantes.

Par exemple, si j'ai besoin de x, y et z pour résoudre un problème, mais que j'ai aussi une constante g = 9,82, je dois déclarer y et z avec var("y z"). La variable g est déclarée implicitement lorsque j'utilise la commande g=9,82. La variable x est toujours prédéclarée.

Pour des raisons stylistiques, certains utilisateurs préfèrent déclarer x de toute façon. Ils tapent donc var("xy z"). Dans Sage, la déclaration redondante de x n'a aucun impact. Cependant, visuellement, il existe une différence : x est traité de manière plus égalitaire entre x, y et z.

3 : Votre connexion Internet est-elle en panne ?

Souvent, si Sage cesse soudainement de répondre, je me rends compte que c'est parce que ma connexion Internet a été coupée. Cela peut être dû à un problème de connexion sans fil, ou simplement à une interruption de service momentanée. Il est peut-être absurde d'inclure ce problème dans cette liste, mais il est important de le mentionner.
Étant donné que Sage fonctionne via Internet, si votre connexion tombe en panne, vous devez restaurer votre connexion pour continuer à utiliser Sage.

π : Recevez-vous un message d'erreur énorme ?

Dans Sage, vous recevrez parfois un message d'erreur volumineux, d'environ 40 lignes. Cela peut être intimidant, voire décourageant, même pour les utilisateurs expérimentés.

Pour comprendre les messages d'erreur de Sage, il est essentiel de regarder la dernière ligne ou les deux dernières. C'est tout ce que vous devez savoir sur ce point. Cependant, je vais maintenant expliquer pourquoi c'est le cas dans les deux paragraphes suivants, qui ne sont destinés qu'aux curieux.

Lorsque vous exécutez une commande, votre code appelle Sage, qui appelle probablement un autre composant de Sage. Un troisième appel peut être effectué vers une partie de Sage, puis le flux est envoyé à un outil comme Maxima, GAP, R ou Singular, qui constitue l'un des éléments constitutifs de Sage. Ce transfert peut impliquer neuf appels de fonction supplémentaires. Les lignes de code en cours d'exécution au moment de l'erreur sont affichées, les niveaux les plus profonds étant indiqués en premier, puis les niveaux les plus proches de la surface (l'interface utilisateur).

Machine Translated by Google

Il est fort probable que l'erreur se situe dans votre code (les dernières lignes imprimées) ou peut-être dans Sage (les lignes suivantes). Les packages dans Sage (par exemple) Maxima, Singular, GAP, etc.) sont très bien testés et débogués ; ce sont les deux tiers supérieurs du désordre des lignes. C'est pourquoi il est conseillé de consulter les dernières lignes pour obtenir des informations et identifier le problème. La dernière ligne est souvent la plus instructive.

4 : Le serveur cellulaire Sage expire-t-il ?

Imaginons que vous travaillez sur le serveur cellulaire Sage et que vous vous levez pour déjeuner. Vous revenez ensuite au bout d'environ une heure et quart.
Il est probable que votre « session » ait expiré.

Pour recommencer sans perdre ce que vous avez tapé dans la cellule, procédez comme suit : • Mettez votre code en

surbrillance • Donnez à votre
navigateur Web la commande « Copier » • Rechargez la page Web de
Sage Cell Server • Donnez à votre navigateur Web la
commande « Coller » • Cliquez sur Évaluer

En fait, je le fais si souvent que c'est devenu une seconde nature. Fréquemment, peut-être une fois toutes les vingt minutes, j'exécute ces cinq étapes pour m'assurer que ma session Cell Server reste active. Par ailleurs, si le code sur lequel vous travaillez fait plus de dix lignes, vous devriez peut-être travailler dans Sage Cloud, afin que votre travail soit plus pérenne et accessible dans les mois ou les années à venir en cas de besoin.

5 : Êtes-vous dans un café ou un hôtel ?

Le type de Wi-Fi public d'un café ou d'un hôtel peut entraîner des problèmes dans Sage, mais ils sont faciles à résoudre. Tout d'abord, les routeurs d'un hôtel mettent souvent en cache les pages de tous leurs utilisateurs. Par exemple, il est probable que de nombreux clients consultent la météo quotidienne ou des sites d'actualités célèbres. Pourquoi télécharger ces pages pour chaque utilisateur encore et encore, alors qu'il est possible de les télécharger une fois et de les mettre en cache localement ? Le plus simple est d'utiliser https plutôt que http. Avec Sage Cloud, c'est le protocole par défaut. Avec Sage Cell Server, veillez à saisir ou à ajouter à vos favoris :

https://sagecell.sagemath.org/

J'insiste encore une fois sur le protocole https. L'autre problème est très différent. De nombreux cafés exigent une réauthentification toutes les 10 ou 20 minutes. Cela rend l'utilisation d'Internet sur de longues périodes peu attrayante.

De cette façon, les étudiants effectuant des travaux universitaires ne monopoliseront pas une place pendant deux heures.

Habituellement, ces ré-authentifications n'interagissent pas bien avec Sage, et vous ne voyez donc même pas d'écran vous demandant de ressaisir le code Wi-Fi.

Ouvrez simplement une nouvelle fenêtre de navigateur, saisissez l'URL de n'importe quel site Web grand et célèbre.

Machine Translated by Google

Vous accéderez alors au site que vous n'avez pas visité lors de cette session web et la page de réauthentification s'affichera. Une fois réauthentifié avec le code requis, vous pourrez à nouveau utiliser Sage normalement.

6 : Vos parenthèses ne correspondent-elles pas ?

Dans Sage, tout comme en mathématiques, vous devez avoir un « (» pour chaque «) » et un « [» pour chaque «]. » C'est absolument obligatoire.

Voici une vieille astuce qu'on m'a apprise au collège pour travailler les mathématiques sur papier. Elle fonctionne aussi très bien sur ordinateur.
Pour vérifier si vous avez mal interprété les parenthèses, commencez à lire de gauche à droite d'une ligne. Commencez par zéro. Chaque fois que vous voyez « (», comptez de un en augmentant, et chaque fois que vous voyez «) », comptez de un en diminuant. Le nombre que vous avez en tête ne doit jamais devenir négatif et doit être nul à la fin de la ligne. Si l'une de ces conditions n'est pas respectée, il y a un problème.

7 : Avez-vous mal orthographié une commande ?

Parfois, l'orthographe d'une commande est floue. Par exemple, s'agit-il de A.inverse(), A.invert() ou A.find inverse() ? S'agit-il de find root ou find roots ? Ces distinctions sont difficiles à retenir, mais il est essentiel de les faire correctement.

La solution à ce problème est de saisir d'abord une partie de la commande, puis d'appuyer sur la touche Tab. Si vous ne trouvez qu'une seule commande, Sage la complètera automatiquement. Si plusieurs commandes correspondent, une petite liste déroulante s'affiche. Nous avons abordé ce sujet en page 4.

8 : Les paramètres d'une commande sont-ils dans le bon ordre ?

Une autre mauvaise surprise peut survenir lorsque les paramètres d'une commande sont dans le mauvais ordre. Par exemple, la commande taylor pour le calcul d'un polynôme de Taylor et la commande slider pour la création d'applets (pages web interactives) ont quatre paramètres. Par conséquent, lorsque j'essaie de deviner l'ordre de ces paramètres, je me trompe souvent.

La solution consiste à saisir la commande sur une ligne distincte, puis à la suivre de l'opérateur « ? ». Appuyez ensuite sur Tab. La docstring (ou fichier d'aide) de cette commande s'affiche, avec sa syntaxe, y compris les paramètres. Nous avons vu cela pour la première fois page 47.

9 : Les sauts de ligne ne sont-ils pas bien placés ?

Il arrive que Python et Sage insistent simplement pour que certains passages ne contiennent pas de saut de ligne. Un autre problème est que la largeur de cette page imprimée, en caractères, est bien plus petite que celle d'une cellule Sage Cell Server ou de l'écran de Sage Cloud. Par conséquent, il m'arrive parfois de devoir couper la ligne dans ce livre pour que le code tienne sur la page, mais cela n'est pas toujours logique à l'écran.

Machine Translated by Google

Mon seul conseil est d'expérimenter et d'être patient pendant que vous essayez différentes possibilités.

~~3~~ : Y a-t-il un problème d'indentation ? 9 4

Il s'agit presque du même problème que ci-dessus. En Python, l'indentation est très importante. Si vous utilisez Sage sans utiliser de structures de programmation comme les boucles for et while, vous n'avez peut-être pas à vous soucier de l'indentation. En revanche, si vous programmez en Python avec Sage, comme nous l'avons vu au chapitre 5, vous devez faire attention à l'indentation. L'indentation est importante, car elle indique quelles commandes sont subordonnées à quelles structures de contrôle de flux. La relecture du chapitre 5 pourrait vous aider.

Malheureusement, lorsque je copie-colle du code de la version électronique de ce livre dans Sage Cell Server, je constate souvent que l'indentation est détruite ou abîmée. La solution consiste à revenir en arrière sur toute l'indentation, puis à la réindenter ligne par ligne avec la touche Tabulation de votre clavier. Cela prendra beaucoup moins de temps qu'on ne le pense.

10 : Il vous manque un côlon ?

Poursuivant sur le thème des bizarreries de Python qui frustrent les étudiants, si vous programmez en Python avec Sage, comme nous l'avons vu au chapitre 5, pensez à placer deux points après les commandes subordonnées. Il s'agit notamment des boucles for, while, des définitions de sous-routines et de la construction if-then-else. Voir le chapitre 5 pour des exemples.

11 : Avez-vous des virgules dans les grands nombres ?

Les grands nombres écrits sont souvent séparés par des virgules. Par exemple : • Notation américaine/britannique :

1 234 567,89 • Notation européenne continentale :
1 234 567,89 • Notation Sage : 1 234 567,89

Comme vous pouvez le constater, vous ne devez pas utiliser de virgules pour séparer les milliers, les millions et les milliards lors de la saisie de nombres dans Sage. C'est absolument interdit.

12 : Http:// contre Https:// ?

J'en ai parlé au point 5, mais je tiens à le répéter. Avec Sage Cell Server, il est bien plus judicieux d'utiliser https au début de l'URL plutôt que http. Ainsi, votre trafic Internet avec le serveur est chiffré et les routeurs intermédiaires ne peuvent pas l'interférer. Le risque n'est pas tant que vos calculs soient espionnés. En revanche, les routeurs intermédiaires pourraient tenter de mettre en cache le trafic http, une pratique pertinente pour presque tous les types de sites web, à l'exception des logiciels mathématiques.

Machine Translated by Google

13 : Plusieurs lignes de sortie dans le serveur de cellules ?

Imaginons un professeur d'algèbre. Il tape factor(x^2 - 5*x + 6).

dans le serveur de cellules Sage, obtenant le résultat habituel. Ensuite, pour montrer un modèle aux étudiants, l'instructeur saisit les quatre lignes suivantes dans la cellule :

```
factor( x^2 - 5*x + 6 ) factor( x^2
- 6*x + 8 ) factor( x^2 - 7*x + 10 )
factor( x^2 - 8*x + 12 )
```

Seule la factorisation du dernier polynôme est indiquée ! Pourquoi ? Parce que Sage n'affiche que la dernière ligne d'un calcul Sage Cell.
Au lieu de cela, il faut taper

```
print factor( x^2 - 5*x + 6 ) print factor( x^2
- 6*x + 8 ) print factor( x^2 - 7*x + 10 )
print factor( x^2 - 8*x + 12 ) puis les quatre
```

lignes de sortie appropriées sont données.

14 : Toujours pas de travail ?

Si cette liste de seize conseils n'a pas résolu votre problème, vous trouverez une aide précieuse sur le forum suivant. Cependant, comme il est géré par des experts Sage entièrement bénévoles, il faudra peut-être patienter quelques heures, voire quelques jours, avant d'obtenir une réponse. Il est certainement utile (et plus rapide) de consulter les anciens messages avant d'en publier un nouveau.
La question que vous vous apprêtez à poser a probablement déjà été posée. Dans l'ensemble, c'est un très bon

service. http://ask.sagemath.org

Machine Translated by Google

Annexe B

Transition vers Sage Cloud

Le chapitre 1 de ce livre suppose que vous utilisez Sage Cell Server. C'est idéal pour les tâches de petite et moyenne envergure. Cependant, vous souhaiterez probablement migrer vers Sage Cloud à terme. En général, si vous écrivez du code de plus de 20 lignes, il est impératif de migrer vers Sage Cloud. Si vous écrivez du code de 10 à 20 lignes, il est préférable de migrer vers Sage Cloud. Sage Cell Server est conçu pour les tâches simples.

B.1. Qu'est-ce que Sage Cloud ?

Le cloud computing consiste à combiner des dizaines, voire des centaines d'ordinateurs répartis dans le monde entier pour offrir aux utilisateurs un supercalculateur virtuel unique. Les avantages du cloud computing sont nombreux :

Fiabilité des données : Lorsque vous utilisez Sage Cloud, toutes vos feuilles de calcul Sage sont stockées dans votre compte. Vous disposez de plusieurs projets, et vous pouvez donc ajouter autant de feuilles de calcul Sage[1] que vous le souhaitez dans chaque projet. Les données sont stockées sur plusieurs machines dans le cloud, réparties dans le monde entier. Il n'y a donc aucun risque de catastrophe susceptible de détruire simultanément toutes les copies existantes.

Comme vous pouvez le constater, c'est beaucoup plus sûr que de conserver tous vos fichiers Sage sur un seul ordinateur portable.

Disponibilité des données : Toutes vos données sont disponibles, à tout moment, depuis n'importe quel ordinateur dans le monde disposant d'une connexion Internet.

Équilibrage de charge : Auparavant, de nombreux serveurs Sage Notebook étaient répartis dans le monde. Cependant, aux heures de pointe, il peut arriver qu'un serveur soit occupé. En théorie, s'il est 16 h dans le Wisconsin, il est 5 h à Pékin ; vous auriez donc peut-être plus de chances de vous connecter à une machine située à Pékin. Cependant, il vous faudrait connaître l'emplacement de ce serveur (et son URL). Ainsi, tous vos fichiers seraient conservés sur votre machine habituelle, par exemple dans le Wisconsin.

1Vous pouvez également organiser les feuilles de calcul appartenant aux projets dans des répertoires et sous-répertoires, si vous en avez envie.

Machine Translated by Google

Vous devrez déplacer ces fichiers vous-même. Tout cela serait fastidieux. Le cloud computing est exactement le contraire : si un serveur est occupé, votre travail de calcul sera automatiquement transféré ailleurs, sans que vous ne le sachiez.

Simplicité : Tous les serveurs étant administrés par la communauté Sage Cloud, les petites universités d'arts libéraux n'ont pas à se soucier de l'achat d'un serveur, de sa maintenance ou de la mise à jour du logiciel. Tous les logiciels sont constamment à jour, et aucune démarche particulière ni aucun achat de matériel supplémentaire ne sont nécessaires.

Économies : Plusieurs raisons expliquent la rentabilité exceptionnelle du cloud computing. En général, les ressources informatiques ne sont pas utilisées pendant les heures de sommeil des étudiants et des mathématiciens, soit de 3 h à 9 h. Maintenant que toutes les machines cloud du monde entier desservent le monde entier en permanence, les fuseaux horaires n'ont plus d'importance. Le problème avec l'ancienne façon de faire, où les grandes universités avaient chacune leur propre serveur, est qu'il est impossible d'acheter 0,1 serveur. Le coût le plus bas serait d'acheter un seul serveur, mais une petite université d'arts libéraux n'a peut-être besoin que de 0,05 serveur de puissance de calcul. L'agrégation des services atténue cet effet, appelé « coût d'indivisibilité » en économie mathématique.

Collaboration : Sage Cloud dispose de nombreuses fonctionnalités collaboratives, telles que la possibilité de rendre des projets publics/privés, visibles par d'autres utilisateurs spécifiques et des outils de chat en ligne.

Point de contrôle : sauf si vous supprimez un fichier, Sage conserve non seulement tous vos fichiers indéfiniment, mais prend également des instantanés et conserve toutes les versions précédentes de tous les fichiers. Ainsi, en cas de problème, vous pouvez revenir à une ancienne version.

Enfin, le plan final prévoit qu'environ 1 % des utilisateurs les plus gourmands en ressources informatiques devraient payer pour leur temps de calcul sur Sage Cloud, mais seulement une somme modique, et que le reste du service serait donc mis à la disposition de l'utilisateur moyen gratuitement.

Le seul inconvénient est la nécessité de créer un compte. Cela prend moins d'une minute, mais en comparaison, vous n'avez pas besoin de créer de compte pour Sage Cell Server.

B.2. Premiers pas dans le cloud

La structure générale de votre compte Sage Cloud est divisée en projets. Le partage entre utilisateurs s'effectue au niveau du projet. Vous pouvez intégrer des feuilles de calcul Sage à vos projets, ou les diviser en répertoires et sous-répertoires si vous le souhaitez.

Une feuille de calcul Sage est comparable à une séquence de cellules géantes du serveur Sage Cell. Vous pouvez saisir quelques lignes de code, puis appuyer sur Ctrl-Entrée (ce qui revient à appuyer sur « Évaluer » dans le serveur Sage Cell).

Machine Translated by Google

Appuyez sur Ctrl-Entrée pour tracer une ligne horizontale. Tout ce qui se trouve depuis la ligne horizontale précédente (ou le haut de la feuille) est alors évalué comme s'il s'agissait d'une cellule Sage sur le serveur de cellules Sage.

Étant donné que les feuilles de calcul Sage sont des fichiers, vous pouvez en avoir plusieurs, copier les renommer, les déplacer et les supprimer.

Ceci est mieux expliqué en vidéo. Les premières minutes de cette vidéo présentent Sage en général. Une démonstration de Sage Cloud est ensuite proposée. Cette vidéo a été réalisée par le professeur William Stein, fondateur et architecte en chef de Sage, et auteur de Sage Cloud. https://www.youtube.com/watch?v=6UGvrVpP89Q

B.3. Autres fonctionnalités du Cloud

Cloud propose également d'autres fonctionnalités, notamment l'utilisation de LATEX, Python, R et le chat en ligne. Nous n'entrerons pas dans les détails ici, mais vous trouverez plus d'informations à l'adresse suivante : http://

www.sagemath.org/help.html

Machine Translated by Google

Machine Translated by Google

Annexe C

Autres ressources pour Sage

Il existe de nombreuses ressources pour Sage. En voici quelques-unes des plus importantes :

Liste principale des ressources d'aide : À l'adresse suivante, vous trouverez un répertoire de tous les éléments pouvant figurer sur cette liste. Cela comprend des tutoriels, des guides de démarrage rapide, des documents de référence, des livres, des visites web et des vidéos. Un nombre limité, mais croissant, de ressources sont rédigées dans d'autres langues que l'anglais. http://www.sagemath.org/help.html

Tutoriels thématiques : Ce site web héberge un grand nombre de tutoriels thématiques Sage. Ces tutoriels portent sur des domaines mathématiques courants, tels que le calcul différentiel et intégral, la théorie des groupes, etc. « Programmation linéaire » ou « Combinatoire algébrique ». http://www.sagemath.org/doc/thematic_tutorials/

Forum de conseils : Ce site web est un espace où chacun peut poser une question sur Sage, et un expert Sage y répondra rapidement. L'activité y est dynamique, car Sage est une communauté importante. http://ask.sagemath.org

Présentation des fonctionnalités : Si vous souhaitez présenter Sage à vos amis, parents ou non-mathématiciens, deux présentations d'une seule page sont disponibles. L'une d'elles est un peu plus axée sur les mathématiques : http://www.sagemath.org/tour-quickstart.html

L'autre concerne les magnifiques graphiques que Sage peut créer : http://www.sagemath.org/tour-graphics.html

Exploration des mathématiques avec Sage : Ce site web/eBook de Paul Lutus traite également de Sage et explique parfaitement la culture open source du développement Sage. Il présente également des problèmes d'application très intéressants. N'oubliez pas de consulter le menu déroulant en haut de votre écran pour découvrir toutes les sections.
Sinon, vous ne pourrez pas naviguer au-delà de l'introduction. http://arachnoid.com/sage/index.html

Machine Translated by Google

Visite Web : Destinée aux professeurs de mathématiques, la visite Web suivante peut être très informative sur certaines des fonctionnalités avancées de Sage, ainsi que sur ses fonctionnalités de base. http://

www.sagemath.org/doc/a_tour_of_sage/

Cartes de référence rapide : Plusieurs utilisateurs de Sage ont créé des cartes de référence rapide pour les aider à apprendre les commandes et à les avoir à portée de main lors de
l'utilisation de Sage : http://wiki.sagemath.org/quickref

Le manuel : Le manuel de référence officiel de Sage est disponible à l'adresse http://www.sagemath.org/doc/reference/, mais ce n'est
pas pour les débutants ! Attention, il s'agit de milliers de pages. C'est comme une encyclopédie : on ne le lit pas d'un bout à l'autre, on y cherche ce dont on a besoin.

Tutoriel Web officiel : il s'agit du tutoriel officiel, destiné davantage aux étudiants diplômés, aux étudiants en mathématiques de dernière année et aux professeurs. http://www.sagemath.org/doc/tutorial/

Tutoriel Web : Le tutoriel Web suivant de Mike O'Sullivan et David Monarres aborde certains sujets que nous n'avons pas abordés dans ce livre.

Il s'agit notamment de l'algèbre abstraite, de la théorie du codage et de l'écriture de projets de programmation plus
importants dans Sage. http://www-rohan.sdsu.edu/~mosulliv/sagetutorial/index.html

Tutoriel numérique : si vous effectuez des cours ou des recherches en nu-informatique numérique, le tutoriel suivant est pour vous : http://
www.sagemath.org/doc/numerical_sage/

Tutoriels pour les enseignants : Comme le décrit le site web lui-même : « Il s'agit d'une série de tutoriels développés pour les ateliers de la Mathematical Association of America intitulés « Sage : Utilisation de logiciels mathématiques open source avec des étudiants de premier cycle » durant les étés 1 2010-2012. Le public initial de ces tutoriels était composé d'enseignants de mathématiques d'établissements de premier cycle ayant peu ou pas d'expérience en programmation ou en logiciels mathématiques. L'expérience informatique requise étant relativement minimale, ces tutoriels devraient être utiles à toute personne arrivant à Sage avec une expérience
limitée. » http://www.sagemath.org/doc/prep/index.html

Tutoriels de démarrage rapide : Ils sont adaptés à des cours de mathématiques spécifiques de licence et sont conçus pour être compréhensibles par un étudiant qui vient de terminer ce cours. De plus, ils sont tous très courts. Nombre d'entre eux tiennent sur une seule page. Ils sont identiques.

1Financement fourni par la National Science Foundation dans le cadre de la subvention DUE 0817071.

Machine Translated by Google

URL comme le tutoriel ci-dessus, mais ils sont répertoriés au bas de la page en tant qu'annexes.

http://www.sagemath.org/doc/prep/index.html

Livres utilisant Sage : Au moment de la publication de cet ouvrage, 24 ouvrages répertoriés sur le site web de Sage utilisent Sage pour leurs exemples. Bien sûr, certains traitent de sujets de recherche exotiques destinés aux doctorants en mathématiques. Cependant, plusieurs d'entre eux conviennent parfaitement aux étudiants de licence. Voici l'URL de la collection complète : http://www.sagemath.org/library-publications.html#books

Mini-cours vidéo en ligne sur Sage : Un ami, Travis Scrimshaw, a réalisé une série de quatre vidéos qui abordent en détail Sage. Ces vidéos ont été réalisées en juillet 2013 et sont donc très actuelles. Elles totalisent 4 heures et 55 minutes. • Partie 1 : https://www.youtube.com/watch?v=FXbHLWtY7s4 • Partie 2 : https://www.youtube.com/watch?

v=9fPf1TQgFIY • Partie 3 : https://www.youtube.com/watch?v=cbBkH8Sgryw • Partie 4 : https://www.youtube.com/watch?v=cyfv9vl6s9M

Machine Translated by Google

Machine Translated by Google

Annexe D

Systèmes linéaires avec Une infinité de solutions

Cette section devrait plutôt figurer dans un manuel d'algèbre linéaire, plutôt que dans un guide Sage. Cependant, je constate que les étudiants se trompent souvent sur ce point, pendant et bien après le cours d'algèbre linéaire, ainsi que dans les cours de niveau inférieur où ces situations peuvent se produire. Par conséquent, voici une discussion complète sur la gestion des systèmes d'équations linéaires à solutions infinies. Ces systèmes sont assez rares, mais souvent utilisés dans les applications. Ce sujet n'est pas très difficile, plutôt approfondi et intéressant à explorer.

D.1. L'exemple d'ouverture

Le système linéaire d'équations donné par

$$\begin{aligned} 20w + 40x + 2a + 86z &= 154 \\ 4w + 8x + 5a + 31z &= 17 \\ w + 2x + 3y + 13z &= -1 \\ -8w - 16x - 4y - 44z &= -52 \end{aligned}$$

a la matrice donnée ci-dessous :

$$A = \begin{matrix} 20 & 40 & 2 & 86 & 154 \\ 4 & 8 & 5 & 31 & 17 \\ 1 & 2 & 3 & 13 & -1 \\ -8 & -16 & -4 & -44 & -52 \end{matrix}$$

Il s'avère que la matrice A a comme RREF :

$$A.\text{rref}() = \begin{matrix} 1 & 2 & 0 & 4 & 8 & 0 & 0 & 1 \\ 3 & -3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Machine Translated by Google

Nous constatons ici la présence de lignes de zéros, et l'absence de la fameuse « diagonale principale » de uns. La matrice identité 4 × 4 est manquante. Nous sommes donc en alerte quant à la possibilité d'un nombre infini de solutions.

Il existe une méthode systématique pour résoudre ce problème. Premièrement, nous allons identifier les éléments spécifiques de la matrice appelés « pivots » ou « ancres ». Ensuite, nous classerons chaque variable comme libre ou déterminée. Troisièmement, nous réécrirons le système d'équations en plaçant toutes les variables déterminées d'un côté des signes égal, et toutes les variables libres ainsi que les constantes de l'autre côté. Une fois cette étape terminée, nous aurons identifié chacune des infinies solutions de ce système d'équations.

Nous appellerons l'entrée non nulle la plus à gauche de chaque ligne « ancre » ou « pivot ». La ligne 1 a comme pivot A11 et la ligne 2 comme pivot A23.
Comme les lignes 3 et 4 ne contiennent aucune valeur non nulle (à proprement parler, car elles sont toutes nulles), elles ne contiennent pas de pivots. Souvent, lorsque je travaille avec un crayon et du papier, j'entoure ou souligne les valeurs pivots d'une matrice. Dans ce cas, j'entoure ou souligne l'entrée de la première ligne, première colonne (A11) et celle de la deuxième ligne, troisième colonne (A23).

Comme nous l'avons vu, chaque colonne d'une matrice représente une variable, à l'exception de la colonne la plus à droite, qui représente les constantes. Bien entendu, chaque colonne contient ou n'en contient pas. Toute colonne (sauf la colonne la plus à droite) est dite « efficace » et sa variable « déterminée » si elle possède un pivot. De même, toute colonne (sauf la colonne la plus à droite) est dite « défectueuse » et sa variable « libre » si elle n'a pas de pivot.

La règle du paragraphe précédent peut paraître arbitraire et quelque peu déroutante, mais appliquons-la une fois et voyons ce qu'elle nous apporte. Nous allons procéder colonne par colonne.

- Dans la colonne 1, nous avons un pivot. Par conséquent, la colonne 1 est effective et w est une variable déterminée.
- Dans la colonne 2, nous n'avons pas de pivot. Par conséquent, la colonne 2 est défectueux et x est une variable libre.
- Dans la colonne 3, nous avons un pivot. Par conséquent, la colonne 3 est effective et y est une variable déterminée.
- Dans la colonne 4, il n'y a pas de pivot. Par conséquent, la colonne 4 est défectueuse et z est une variable libre.

Vous voyez, ce n'était pas si mal, non ? Nous allons maintenant prendre le RREF, traduire les lignes (non nulles) en algèbre, puis déplacer les constantes et les variables libres à droite du signe égal, en ne laissant que les variables déterminées à gauche du signe égal. Commençons par le littéral ou non interprété.

1Pour les plus curieux : la terminologie « effectif/défectueux » n'est pas utilisée dans la colonne la plus à droite, car celle-ci ne représente aucune variable. Il est donc inutile de discuter si la variable de cette colonne est libre ou déterminée.

Machine Translated by Google

la traduction du RREF en algèbre serait

$$w + 2x + 0y + 4z = 8$$
$$0w + 0x + 1y + 3z = -3$$

Maintenant, déplaçons les variables libres (x et z) vers le côté droit de la signes égaux, pour joindre les constantes. On obtient

$$w = 8 - 2x - 4z$$
$$y = -3 - 3z$$

cependant, je préfère parfois écrire

$$w = 8 - 2x - 4z$$
$$x \text{ est gratuit}$$
$$y = -3 - 3z$$
$$z \text{ est libre}$$

Ce qui précède signifie que vous pouvez choisir les valeurs que vous souhaitez pour x et z, mais une fois que vous l'avez fait, w et y sont déterminés par les choix que vous avez faits.

Faisons sept solutions spécifiques, en choisissant sept paires de valeurs pour x et z. On obtient :

(1) Si $x = 0$ et $z = 0$ alors $w = 8$ et $y = -3$.
(2) Si $x = 1$ et $z = 0$ alors $w = 6$ et $y = -3$.
(3) Si $x = 0$ et $z = 1$ alors $w = 4$ et $y = -6$.
(4) Si $x = 1$ et $z = 1$ alors $w = 2$ et $y = -6$.
(5) Si $x = 2$ et $z = 3$ alors $w = -8$ et $y = -12$.
(6) Si $x = -4$ et $z = -6$ alors $w = 40$ et $y = 15$.
(7) Si $x = \pi$ et $z = \sqrt{2}$ alors $w = 8 - 2\pi - 4\sqrt{2}$ et $y = -3 - 3\sqrt{2}$.

Prenez un moment pour vérifier que chacune de ces affectations de variables satisfait bien chacune des équations données. Effectuez cette opération manuellement pour les cinq premières. Nous utiliserons Sage pour vérifier les deux dernières entrées. Pour la sixième entrée de notre liste, nous saisissons

```
dans=40
x=-4
y=15
z=-6
imprimer 20*w + 40*x + 2*y + 86*z imprimer
4*w + 8*x + 5*y + 31*z imprimer w + 2*x +
3*y + 13*z imprimer -8*w - 16*x - 4*y -
44*z
```

Nous voyons exactement ce que nous voulions, à savoir les côtés droits de nos équations originales : 154, 17, −1 et −52.

De même, nous pouvons simplement revenir en arrière sur ces quatre paramètres de variables et fournir les paramètres de chaque solution spécifique de la liste ci-dessus. Plus précisément, pour la septième entrée de la liste ci-dessus, nous saisirions

Machine Translated by Google


```
w=8-2*pi-4*sqrt(2) x=pi

y=-3-3*sqrt(2)
z=sqrt(2)
imprimer 20*w + 40*x + 2*y + 86*z
imprimer 4*w + 8*x + 5*y + 31*z
imprimer w + 2*w +-1 3*x 16*x −
4*y − 44*z
```

Comme vous pouvez le voir, nous obtenons la sortie souhaitée : 154, 17, −1 et −52.

D.2. Un autre exemple

Imaginons qu'on vous demande de trouver toutes les solutions du système d'équations linéaires

$$3w - 3y + 6z = 51 \quad x + 4y + 5z = 19 \quad -w + x + 5y + 3z = 2 \quad 2w + 5x + 18y + 29z = 129$$

Nous commencerions par convertir cela en une matrice,

$$\begin{matrix} 3 & 0 & -3 & 6 & 51 \\ 0 & 1 & 4 & 5 & 19 \\ -1 & 1 & 5 & 3 & 2 \\ 2 & 5 & 18 & 29 & 129 \end{matrix}$$

qui a comme RREF, la matrice

$$\begin{matrix} 1 & 0 & -1 & 2 & 17 \\ 0 & 1 & 4 & 5 & 19 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Premièrement, nous identifions les pivots. Comme vous pouvez le constater, je les ai marqués d'un s. Deuxièmement, nous devons classer les variables comme « libres » ou « déterminées ». Dans ce cas, puisque w et x ont des pivots, ce sont des variables déterminées ; de même, y et z n'ont pas de pivot, ce sont donc des variables libres. Nous écrivons maintenant la traduction littérale non interprétée de la matrice RREF en algèbre, en ignorant les lignes entièrement nulles.

$$w - y + 2z = 17 \quad x + 4y + 5z = 19$$

Machine Translated by Google

Enfin, nous déplaçons les variables libres (y et z) à droite du signe égal, et obtenons

$$w = 17 + y - 2z$$
$$x = 19 - 4y - 5z \text{ y est}$$
$$\text{libre } z \text{ est libre}$$

Peut-être que si on nous demandait de trouver quatre solutions distinctes, nous pourrions écrire : (1) Si y = 0 et z = 0 alors w = 17 et x = 19.
(2) Si y = 0 et z = 1 alors w = 15 et x = 14.
(3) Si y = 1 et z = 0 alors w = 18 et x = 15.
(4) Si y = 1 et z = 1 alors w = 16 et x = 10.

Un peu plus de terminologie Nous avons utilisé les termes « variable libre » et « variable déterminée », ainsi que « colonne élémentaire » et « colonne dégénérée ». Le terme système dégénéré désigne2 tout système d'équations linéaires où la solution n'est pas unique, qu'il n'ait pas de solution ou qu'il ait une infinité de solutions.

Pour distinguer les systèmes d'équations linéaires à infinité de variables, nous utiliserons le concept de « degrés de liberté ». Le nombre de degrés de liberté correspond au nombre de variables libres. Ici, les deux systèmes que nous avons vus jusqu'ici ont deux degrés de liberté. Nous en verrons bientôt plusieurs qui n'en ont qu'un seul. Il existe également des systèmes à trois degrés de liberté ou plus (c'est-à-dire trois variables libres), mais ils semblent très rares dans les applications et ne sont donc pas abordés ici. De plus, une solution où toutes les variables sont déterminées et aucune variable n'est libre (solution unique) est dite à zéro degré de liberté. On n'utilise généralement pas le terme « degrés de liberté » pour désigner un système sans solution.

Le nombre de degrés de liberté d'un système d'équations est également appelé nullité à droite de la matrice qui le décrit. Le nombre de variables déterminées est appelé rang de la matrice. Le rang de la matrice est évoqué à de nombreuses reprises dans l'étude de l'algèbre matricielle.

Relevez le défi : Voici quatre exemples simples pour vous mettre au défi et vérifier votre compréhension. Les dernières équations avec le mot « libre » sont présentées à la fin de cette annexe. Vous devriez également générer 3 ou 4 équations spécifiques.

2Le terme « dégénéré » est, à l'époque moderne, extrêmement grossier. Pourtant, à une certaine époque, il devait être poli, ou du moins socialement acceptable. Ce terme était utilisé au troisième quart du XXe siècle pour décrire toute personne s'écartant significativement des normes sociales. Ici, un système d'équations linéaires « normal » est un système à solution unique. Tout écart par rapport à ce comportement attendu est qualifié de « dégénéré ».

Machine Translated by Google

solutions et voir si elles fonctionnent réellement. Nous fournirons une solution spécifique où toutes les variables libres sont définies à 1.

M1 =
1 2 −1 0 1
3 6 −3 2 7
2 4 −2 −1 0
7 14 −7 6 19

M2 =
1 3 0 0 | 1
3 9 2 0 | 7
2 6 −1 −4 −32
7 21 6 5 59

M3 =
4 20 −24 16
7 35 −42 28 1 5
−6 4

C2 =
1 2 3 7 4 | 5
6 16
7 8 9 25

Au fait, c'est le même C2 que nous avons vu à la page 29. À ce moment-là, nous avons calculé que C2.rref() est égal à

[1 0 -1 -1] [0 1 2 4]
[0 0 0 0]

et puis nous nous sommes arrêtés et avons dit « il y a une infinité de solutions. » Cependant, j'aimerais maintenant que vous produisiez des équations avec des variables libres et déterminées comme nous l'avons fait tout au long de cette annexe.

D.3. Explorer plus en profondeur

À ce stade, la majeure partie du sujet vous est familière, mais les points mineurs suivants vous donneront un aperçu plus approfondi.

D.3.1. Un réexamen intéressant des solutions uniques

Revenons brièvement à B de la section 1.5.3, « Matrices et Sage, première partie » à la page 23. La matrice originale et son RREF sont

B =
−1 2 1 −5 6 0 0 −1 | 1
−5 1 3 3 −1 0 −1 5
−1 0 −1

devient

1 0 0 0 −107/7 0 1 0 0
−12/7 54/7 0 0 1 0 19/7 0
0 0 1

J'ai de nouveau marqué les pivots avec un s. Comme vous pouvez le constater, toutes les colonnes sont élémentaires et aucune n'est dégénérée. Toutes les variables sont déterminées et aucune n'est libre. De ce fait, nous avons une seule et unique solution :

dans = −107/7 x = −12/7 y = 54/7 avec = 19/7

Machine Translated by Google

D.3.2. Deux équations et quatre inconnues Ce serait un peu étrange, mais quelqu'un pourrait vous demander de produire des solutions aux deux équations suivantes.

$$11sem + 88x - 11y + 13z = 47 \quad 17sem + 136x - 17y + 19z = 65$$

Cela paraît étrange car, bien que nous ayons 4 inconnues, nous n'avons que 2 variables. Il semble que nous ayons « trop peu » d'équations et que le problème ne soit pas suffisamment renseigné pour le résoudre. En réalité, cette intuition est juste. Il existe un théorème : lorsqu'il y a moins d'équations que d'inconnues, soit il n'y a pas de solution, soit il y a une infinité de solutions, il est impossible d'obtenir une solution unique. Nous ne pouvons pas démontrer ce théorème de manière générale ici, mais nous pouvons explorer cet exemple précis.

Premièrement, il est extrêmement simple de convertir ce système d'équations dans une matrice, bien qu'elle semble un peu étrange puisqu'il s'agit d'une matrice 2 × 5.

$$\left(\begin{array}{ccccc|c} 11 & 88 & -11 & 13 & 47 & \\ 17 & 136 & -17 & 19 & 65 & \end{array}\right)$$

Nous pouvons demander à Sage de trouver le RREF, puis nous pouvons identifier les pivots. Le RREF est donné ci-dessous et les pivots ont été marqués avec s.

$$\left(\begin{array}{ccccc|c} 1 & 8 & -1 & 0 & -4 & \\ 0 & 0 & 0 & 1 & & 7 \end{array}\right)$$

Maintenant, en utilisant nos méthodes standard, nous terminons avec :

$$w = -4 - 8x + yx \text{ est libre}$$

$$y \text{ est libre}$$

$$\text{avec} = 7$$

Qu'est-ce que cela signifie ? Cela signifie que vous pouvez choisir n'importe quel x et n'importe quel y. Cependant, une fois ce choix effectué, w est déterminé par la formule w = −4 − 8x + y. Contrairement à tous ces changements dans w, x et y, z est très stéréotypé et définitivement verrouillé à z = 7. Il est intéressant de noter que nous avons deux degrés de liberté, mais simultanément, il n'y a aucune ligne de zéros dans le RREF.

D.3.3. Deux équations et quatre inconnues, mais aucune solution : Le lecteur curieux pourrait être intéressé par un système d'équations à deux équations et quatre variables, mais sans solution. En voici un exemple évident :

$$w + x + y + z = 1 \quad w + x + y + z = 2$$

Machine Translated by Google

De toute évidence, il n'y a aucun moyen pour que w + x + y + z soient à la fois égaux à 1 et égaux à 2 en même temps.

D.3.4. Quatre équations et trois inconnues Considérons le système de 4 équations et 3 inconnues donné ci-dessous :

$$\begin{aligned} x + 2y + 3z &= 31 \\ 4x + 5y + 6z &= 73 \\ 7x + 8y - z &= 35 \\ 5x + 7y + 9z &= 104 \end{aligned}$$

Cela produit la matrice

$$G = \left[\begin{array}{ccc|c} 1 & 2 & 3 & 31 \\ 4 & 5 & 6 & 73 \\ 7 & 8 & -1 & 35 \\ 5 & 7 & 9 & 104 \end{array}\right]$$

avec le RREF

$$G.rref = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

Comme vous pouvez le voir, il y a trois pivots, que j'ai marqués d'un s. Par conséquent, chacune des variables est déterminée et aucune d'entre elles n'est libre. Enfin, nous concluons qu'il existe une solution unique, avec x = 5, y = 1 et z = 8. Notez qu'il existe une solution unique, malgré la présence d'une ligne de zéros.

D.4. Idées fausses

J'aimerais maintenant aborder brièvement quelques idées fausses que les étudiants ont souvent sur ce sujet de l'infinité de solutions à un système linéaire d'équations.

Idée fausse majeure : Les élèves imaginent souvent qu'une ligne de zéros donne lieu à une infinité de solutions. Or, nous avons vu à la section D.3.2 un cas où une matrice ne contenait aucune ligne de zéros, mais une infinité de solutions. À la section D.3.4, nous avons vu un cas où la matrice contenait une ligne de zéros, mais une seule solution. Que nous apprend donc une matrice dont l'une des lignes est entièrement composée de zéros ?

Si vous avez le même nombre de variables que d'équations, vous pouvez utiliser la règle empirique de la section 1.5.8, page 28. Sinon, la ligne de zéros ne vous dit absolument rien. Vous devez donc identifier les pivots, puis déterminer les variables libres et celles qui sont déterminées, comme nous l'avons fait tout au long de cette annexe.

Machine Translated by Google

Une idée fausse moins courante : les élèves

imaginent parfois qu'une colonne est efficace si elle contient zéro partout, sauf une entrée contenant un 1. Bien que cela soit presque toujours vrai, voici un contre-exemple qui démontre que cette croyance est fausse. Prenons l'exemple de la matrice.

$$Q = \begin{array}{c|c} 1 \quad 0\,1\,3\,0 & 1 \\ 0\,2\,0\,0\,0\,0 & \end{array}$$

Comme vous pouvez le constater, la colonne correspondant à chaque variable est nulle partout, sauf pour une seule. Un élève ayant cette idée fausse imaginerait que chaque variable est déterminée et qu'aucune n'est libre. Cet élève confus dirait qu'il n'existe qu'une seule solution : x = 3, y = 2 et z = 0. Or, c'est clairement faux.

J'attire particulièrement votre attention sur la troisième colonne, où il n'y a qu'un seul 1 et où toutes les entrées sont nulles. Cette colonne semble élémentaire, mais comme ce 1 isolé n'est pas un pivot, elle n'est pas élémentaire, mais plutôt dégénérée. On voit ici l'intérêt de marquer les pivots avec s. Les pivots de cette matrice sont Q11 et Q22, mais Q13 n'est pas un pivot. Il est clair que la troisième colonne n'a pas de pivot. Par conséquent, z est libre. On devrait écrire

$$x = 3 - z$$
$$y = 2$$
$$z \text{ est libre}$$

pour décrire l'ensemble infini de solutions.

D.5. Définitions formelles de REF et RREF

Pour les plus curieux, les définitions formelles de pivot, REF et RREF sont données ci-dessous.

- Une entrée d'une matrice est un pivot si et seulement si elle est la plus à gauche entrée non nulle dans sa ligne.
- Par conséquent, par définition, il n'y a jamais d'entrées non nulles à gauche d'un pivot. • Si une matrice est telle qu'il n'y a pas d'entrées non nulles à gauche ou en dessous d'un pivot, alors cette matrice est dite sous forme d'échelon de ligne ou REF.

- Si une matrice est telle qu'il n'y a aucune entrée non nulle à gauche, en dessous ou au-dessus d'un pivot, alors cette matrice est dite sous forme réduite d'échelons de lignes ou RREF. • Parfois, une matrice est telle qu'il n'y a aucune entrée non nulle à gauche, en dessous, au-dessus ou même à droite d'un pivot.

Machine Translated by Google

– Si le système est défini avec exactitude, on parle de matrice de permutation. Lors de la résolution d'un système d'équations avec une matrice de permutation, on constate que les inconnues sont simplement les constantes, mais probablement dans un ordre différent. Les matrices de permutation apparaissent dans plusieurs branches de l'algèbre linéaire, pure et appliquée.
– Si le système est sur-défini ou sous-défini, on parle alors de matrice de tours en raison de certains problèmes intéressants liés aux mathématiques et aux échecs, mais ils apparaissent également dans les problèmes de combinatoire.

D.6. Notations alternatives

Il existe une notation plus sophistiquée pour explorer ces espaces à solutions infinies. Si vous avez déjà étudié les vecteurs, cette section pourrait vous intéresser. Cependant, si vous n'êtes pas très à l'aise avec les vecteurs, il est préférable d'arrêter la lecture de cette annexe. Vous pouvez également passer à la section D.7, « Interprétations géométriques en 3 dimensions », page 316.

Considérons le premier exemple de cette annexe, mais en écrivant x = x au lieu de « x est libre » et z = z au lieu de « z est libre ». Nous obtenons :

w = 8 − 2x − 4z	Dans				−2		−4
x = x	x						
y = −3 − 3z	et		8 0		1		0
avec = avec	Avec	=	−3 0	+ x	0 0	+ de	−3 1

Considérons maintenant le deuxième exemple de cette annexe, mais en écrivant y = y au lieu de « y est libre », et z = z au lieu de « z est libre ». On obtient :

w = 17 + y − 2z x =	Dans		17				−2
19 − 4y − 5z	x		19		1 −4		−5
y = y	et		0				0
avec = avec	Avec	=	0	+ y	1 0	+ de	1

Pour être sûr que vous l'avez, nous vous proposons deux autres défis :

5. Convertissez M3 de la page 312 dans cette notation vectorielle.
6. Convertissez C2 de la page 312 dans cette notation vectorielle.

D.7. Interprétations géométriques en 3 dimensions

Il existe des moyens remarquablement utiles de visualiser géométriquement ce type de problèmes. Tout comme 2x+3y = 6 représente une droite dans un plan, x+2y+3z = 6 représente un plan flottant dans l'espace. Lorsque vous coupez deux droites (ou résolvez un système de deux équations linéaires à deux inconnues), vous calculez généralement le point d'intersection des deux droites. Cependant, il existe des « cas mineurs ». L'un d'eux est lorsque les deux droites sont parallèles, et donc jamais

Machine Translated by Google

se croisent. L'autre cas mineur est celui où les deux droites sont en fait la même, et il existe un nombre infini de solutions.

Lorsqu'on a un système de trois équations linéaires et trois variables, on parle d'intersection de trois plans dans l'espace ordinaire. Généralement, ces trois plans se coupent en un point commun, mais il existe aussi des cas mineurs.
Cette fois, il s'agit de 4 cas mineurs.

- La première est qu'il est possible que les trois plans n'aient jamais de point commun : un peu comme deux murs opposés d'une pièce rectangulaire et le toit de cette pièce. Puisque les deux murs opposés sont parallèles, ils n'ont jamais de point commun.
- Le deuxième cas est celui où les trois plans ont une droite en commun. Imaginez un livre à couverture rigide, peu paginé, partiellement ouvert, avec des pages très rigides. Les trois plans peuvent être n'importe laquelle des pages très rigides, ou l'une des couvertures. De plus, ces plans ont en commun le dos du livre (qui est une droite). • Le troisième cas, ou cas vraiment rare pour trois variables, est celui où les trois plans sont en fait le même plan, et où tout point de ce plan résout les trois équations. Cependant, cela ne signifie pas que chaque point de l'espace tridimensionnel soit une solution. Il faut plutôt qu'un point soit dans le plan pour être une solution.
- Un quatrième cas (sans sens) est la matrice suivante :

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$

où chacune des trois équations est en fait l'équation triviale :

$$0x + 0y + 0z = 0$$

et donc chaque point dans l'espace tridimensionnel satisfait ces « équations ». L'espace solution est notre espace tridimensionnel entier-
verset.

D.7.1. Visualisation de ces cas

Avec Sage, chacune de ces images peut être dessinée. Cependant, s'agissant d'images 3D, elles ne sont visibles que sur un écran où la couleur est présente, où la souris permet de déplacer les images pour les visualiser sous différents angles et où les plans peuvent être rendus transparents. Cependant, je ne peux pas insérer de telles images significatives sur la page imprimée d'un livre noir et blanc.

J'ai donc créé une applet pour vous. Cette applet, ou page web interactive, s'intitule « Visualiser une infinité de solutions dans un système linéaire de 3 équations et 3 inconnues ». Vous la trouverez sur ma page web www.gregorybard.com, en cliquant sur « Pages web interactives pour l'enseignement des mathématiques ». Trois exemples y sont donnés. Pour le cas « zéro degré de liberté », nous utilisons la matrice A de la page 19 ; pour le cas « un degré de liberté ».

Machine Translated by Google

cas, nous utilisons la matrice C2 qui apparaît à la page 29 et à la page 312 ; pour le cas « deux degrés de liberté », nous utilisons la matrice M3 de la page 312.

D.7.2. Interprétations géométriques en dimensions supérieures. Les problèmes à quatre variables ou plus nécessitent les notions d'espaces de grande dimension et d'hyperplans. Par exemple, avec deux équations à quatre inconnues dans le RREF, la solution se révèle souvent être un plan flottant dans un espace à quatre dimensions. Avec trois équations à quatre inconnues, la solution est souvent un hyperplan à trois dimensions dans cet espace à quatre dimensions. L'hyperplan a la même géométrie que l'univers dans lequel nous vivons. Tout cela semble très compliqué, mais cela peut empirer, en réalité.

Avec 5 inconnues et 2 équations dans le RREF, les solutions sont souvent un sous-espace à 3 dimensions flottant dans un espace à 5 dimensions. (Cela représente trois degrés de liberté, soit dit en passant.) On peut imaginer un univers plus vaste et plus complexe que le nôtre, mais dont un minuscule sous-espace représente notre « univers quotidien ». Ce minuscule sous-espace constitue l'ensemble des solutions.

Après de nombreux cours, les mathématiciens peuvent apprendre à comprendre ces idées de manière formelle, au sens de l'algèbre. Cependant, très peu de personnes sont réellement capables de visualiser ces objets. La grande majorité d'entre nous, moi y compris, en est incapable ; pour beaucoup et moi-même, la visualisation s'arrête à trois dimensions. Par conséquent, nous n'explorerons pas davantage le point de vue géométrique en hautes dimensions.

D.8. Notation des variables muettes

De nombreux manuels scolaires insistent pour que les variables libres soient réécrites comme s et t lorsqu'il y a deux degrés de liberté, et seulement t lorsqu'il y a un degré de liberté. Ce choix est extrêmement étrange, car il semble brouiller la frontière entre variables libres et variables déterminées. (Par souci d'exhaustivité, lorsqu'il y a trois degrés de liberté, les variables libres seraient remplacées par r, s et t.)

Si nous nous sentions obligés de suivre cette notation, nous aurions une étape supplémentaire très simple (mais superflue) lors du traitement du premier exemple de cette annexe. À savoir, nous apporterions la modification suivante :

w = 8 − 2x − 4z	w = 8 − 2s − 4t
x = x	x = s
y = −3 − 3z	y = −3 − 3t z = t
avec = avec	

et de même, pour le deuxième exemple de cette annexe,

w = 17 + y − 2z x = 19 −	w = 17 + s − 2t x =
4y − 5z y = y	19 − 4s − 5t
	y = sz = t
avec = avec	

Machine Translated by Google

À première vue, ce changement constitue une étape supplémentaire, mais semble n'avoir aucun effet. Cependant, cette notation n'est pas totalement dénuée de mérite. Par exemple, un système à un degré de liberté contient at, qui est toujours la variable libre. De même, avec deux degrés de liberté, on trouvera un s et un t.
Il n'y a donc aucun doute quant aux variables libres et à leur nombre. Cependant, on peut en dire autant de la notation que j'ai utilisée tout au long de l'annexe, en utilisant le mot « libre ». De plus, il semble dommage de transformer un problème à 4 variables en un problème à 6 variables ; les problèmes à 4 variables sont déjà assez difficiles. Enfin, t est souvent confondu avec +, et de même s avec 5. Tout bien considéré, je pense qu'il est préférable d'éviter d'introduire de nouvelles variables.

Cependant, si votre instructeur vous oblige à utiliser t, sachez qu'une majuscule
Il est plus facile de distinguer un T d'un signe + qu'un t minuscule.

D.9. Solutions aux défis

(1) Pour M1 , la solution est

$$w = 1 - 2x + y$$
$$x \text{ est gratuit}$$
$$y \text{ est libre } z = 2$$

Un exemple de solution serait w = 0, x = 1, y = 1, z = 2.

(2) Pour M2 , la solution est

$$w = 1 - 3x$$
$$x \text{ est gratuit}$$
$$y = 2$$
$$\text{avec} = 8$$

Un exemple de solution serait w = −2, x = 1, y = 2, z = 8.

(3) Pour M3 , la solution est

$$x = 4 - 5y + 6z \text{ y est}$$
$$\text{libre z est libre}$$

Un exemple de solution serait x = 5, y = 1, z = 1.

(4) Pour C2 , la solution est

$$x = -1 + z$$
$$y = 4 - 2z$$
$$z \text{ est libre}$$

Un exemple de solution serait x = 1, y = −2, z = 1.

Machine Translated by Google

(5) Ici, nous transformons la solution pour M3 en notation vectorielle.

$$\begin{aligned} x &= 4 - 5y + 6z \\ y &= y \\ \text{avec} &= \text{avec} \end{aligned} \qquad \begin{matrix} x \\ \text{et} \\ \text{Avec} \end{matrix} = \begin{matrix} 4 \\ 0 \\ 0 \end{matrix} + \text{et} \begin{matrix} -5 \\ 1 \\ 0 \end{matrix} + \text{de} \begin{matrix} 6 \\ 0 \\ 1 \end{matrix}$$

(6) Ici, nous transformons la solution pour C2 en notation vectorielle.

$$\begin{aligned} x &= -1 + zy = 4 - \\ 2z \\ \text{avec} &= \text{avec} \end{aligned} \qquad \begin{matrix} x \\ \text{et} \\ \text{Avec} \end{matrix} = \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} + \text{de} \begin{matrix} 1 \\ -2 \\ 1 \end{matrix}$$

Machine Translated by Google

Annexe E

Installation de Sage sur votre Ordinateur personnel

Ma réponse habituelle lorsque les gens me demandent comment installer Sage sur leur ordinateur personnel est : « Êtes-vous sûr de vouloir faire cela ? »

Arguments contre une installation locale : L'un des

atouts de Sage réside dans sa capacité à mettre à disposition des logiciels mathématiques de pointe à toute personne ayant accès à Internet. Depuis n'importe quel navigateur web, vous pouvez utiliser Sage Cell Server pour les petits calculs, et Sage Cloud pour les calculs plus importants.

Conserver votre travail sur Sage Cloud présente de nombreux avantages par rapport à une installation locale. Par exemple, en cas de perte ou de vol de votre ordinateur portable, vous ne perdez pas vos fichiers. Si vous devez travailler en déplacement, vous pouvez utiliser n'importe quel ordinateur connecté à Internet. Partager vos projets avec d'autres personnes dans Sage Cloud est très simple, en leur donnant un accès électronique. Si vous devez effectuer des calculs pendant plusieurs heures, vous n'avez pas à monopoliser votre ordinateur portable. Tous les logiciels seront toujours à jour, sans effort supplémentaire de la part de l'utilisateur. Personnellement, je n'ai jamais installé Sage en local.

En classe, ce point est particulièrement important. Si un enseignant demande à un groupe de 40 étudiants d'effectuer une installation locale, il devra gérer la diversité des systèmes d'exploitation, des machines et l'inexpérience des étudiants. Nombre d'entre eux maîtrisent peut-être assez bien l'informatique, mais les 2 ou 3 moins bons d'entre eux le sont probablement beaucoup moins. À mesure que les ordinateurs deviennent de plus en plus faciles à utiliser, le niveau de compétence de l'utilisateur moyen tend à diminuer. Par exemple, le concept d'arborescence et de sous-répertoires est désormais déroutant et difficile à comprendre pour les étudiants au début. Les étudiants sont souvent extrêmement intimidés et hostiles à la « ligne de commande », ou, au mieux, la connaissent mal.

Souvent, les étudiants qui ont une dent contre un instructeur (pour avoir des normes élevées ou des examens difficiles) se retrouvent incapables de se plaindre auprès de

Machine Translated by Google

Les directeurs de département et les doyens peuvent ainsi invoquer la difficulté d'utiliser un logiciel obligatoire comme motif de réclamation. Par exemple, « Mais, doyen XYZ, vous ne comprenez pas, je n'ai même pas réussi à installer le logiciel ! »

Raisons en faveur d'une installation locale :

À bien y réfléchir, une installation locale présente également des avantages. Si l'accès internet haut débit est omniprésent dans la majeure partie de l'Amérique du Nord, il l'est moins dans les zones rurales de Chine, en Afrique subsaharienne et (d'après ce qu'on m'a dit) dans certaines régions du Mississippi. L'un des principaux inconvénients de Sage Cell Server et de Sage Cloud est la nécessité d'un accès internet. Plusieurs membres de la communauté Sage sont impliqués dans l'Institut africain des sciences mathématiques, une communauté attirée par Sage en raison de son caractère gratuit. De plus, certains parents aux États-Unis refusent tout accès internet à leurs enfants, pour des raisons religieuses. Cela pourrait être un facteur pour certains enseignants du secondaire. L'un de mes anciens assistants de recherche de premier cycle a grandi dans un tel foyer.

L'avantage, c'est que même si l'installation de Sage sur votre machine demande beaucoup de temps et d'efforts, elle sera beaucoup plus rapide. Une grande partie du temps d'attente lors de l'utilisation de Sage est due au temps de transit entre votre ordinateur et les serveurs. Pour les connexions internet lentes, c'est un facteur important.

Instructions pour Mac :

Si vous utilisez un Mac et que vous avez OS X, vous devez accéder à cette URL :

http://www.sagemath.org/download-mac.html Des instructions supplémentaires sont disponibles à

l'adresse : http://www.sagemath.org/mirror/osx/README.txt

Le LiveCD Sage : il s'agit

d'un CD bootable permettant d'exécuter directement Sage. Vous le trouverez à l'adresse suivante : http://

www.sagemath.org/download-livecd.html

Instructions pour les utilisateurs Unix et Linux :

Vous pouvez trouver des binaires pour une installation Unix ou Linux à cette URL :

http://www.sagemath.org/download-linux.html Cependant, si vous êtes un développeur expérimenté, vous pourriez également vouloir construire le système entier de A à Z. C'est tout l'intérêt des logiciels open source, après tout. Sage étant un système vaste et complexe, cela peut prendre plus de temps que prévu. Les utilisateurs expérimentés et audacieux devraient consulter cette URL : http://www.sagemath.org/download-source.html

Machine Translated by Google

Instructions pour Microsoft Windows :

Le téléchargement pour les utilisateurs Windows est disponible à l'adresse suivante : http://www.sagemath.org/download-windows.html. Cependant, je recommande vivement les instructions disponibles à l'adresse : http://wiki.sagemath.org/SageAppliance. Vous installerez ensuite VirtualBox, une machine virtuelle qui encapsulera Sage. Ce processus est hautement testé et sécurisé.

On me dit qu'il n'y a presque aucun décalage de performances par rapport aux installations Linux et Mac.

Machine Translated by Google

Machine Translated by Google

Annexe F

Index des commandes par Nom et par section

Nous avons ici un index complet de toutes les commandes de ce livre. Nous les présentons d'abord par thème (par section). Ensuite, par ordre alphabétique (par nom). Un grand merci à Thomas Suiter, qui a compilé cet index.

Machine Translated by Google

Section	Commande	Description
1.2	carré	[illegible]tion
1.2	approximation	
1.2	_	
1.2	log	[illegible]entielles,
1.2		[illegible]ne,
1.3		[illegible]métrique
1.3	[illegible]	[illegible]métrique
	tanné	[illegible]métrique
	csc	[illegible]métrique
1.3	seconde	[illegible]métrique
	[illegible]ne	[illegible]métrique
		[illegible]métrique
1.3	arccos	[illegible]métrique
1.3	arctan	[illegible]métrique
1.3	arccsc	[illegible]métrique
1.3	arcsec	[illegible]métrique
	Arccot	[illegible]métrique
1.3	sel	[illegible] »
1.3	acos	[illegible] »
	[illegible]	[illegible] »
	acsc	[illegible] »
1.3	asec	[illegible] »
	[illegible]elle	[illegible] »
		[illegible]nées
1.4		[illegible]

Machine Translated by Google

1.4	spectacle	[illegible]er/
1.4.2		[illegible]sition
1.5.3	matrice	[illegible]s
1.5.3	rref	[illegible]née
	développer	[illegible]ions
		[illegible]
1,7	[illegible]	[illegible]née
		[illegible]
		[illegible]
1.8.1		[illegible]ques
	notre	[illegible]ter
1.9	[illegible]	[illegible]ximation
1.10	–	[illegible]tion
		commande
1.11	diff	[illegible]s
1.11	dérivé	[illegible]onyme
1.12	intégrale	[illegible]s
1.12		[illegible]rale
1.12	–	[illegible]ment,
		[illegible]
1.12	partielle –	[illegible]ée
1.12		[illegible]me
		Distribution
1.12	supposer	[illegible]eur

Machine Translated by Google

2.4.3	[illegible]rielle	[illegible]
		[illegible]ent
3.1.1	[illegible]settes	[illegible]ent
3.1.3	flèche	
3.1.3	texte	
3.3	[illegible]me	
3.4		
3,5		[illegible]les
		[illegible]tion
		[illegible]ment
	[illegible]rique	[illegible]tique
		[illegible]s
		[illegible]s
		[illegible]atique
		[illegible]r)
3.8		[illegible]ées
3.8.2	[illegible]	[illegible]t
		[illegible]le
3.9.2		[illegible]ie
		[illegible]ctivement
3.9.2	[illegible]	[illegible]ement
		nombre

Machine Translated by Google

4.4.2	[illegible] –	[illegible]
4.4.3	–	[illegible]
4.4.3	–	[illegible]
4.4.3	[illegible]	[illegible]
	–	[illegible]
		[illegible]
4.4.5	–	[illegible]
4.4.5	–	[illegible]
4.4.5	–	[illegible]
		Matrice
4.4.5	[illegible]	[illegible]
		Matrice
4.4.6	le	[illegible]
4,5	[illegible]	[illegible]
4,6.2	–	[illegible]
	–	[illegible]
	–	[illegible]
	–	[illegible]
		[illegible]
	–	[illegible]
4.6.3	–	[illegible]
4.6.4		[illegible]
		entier
4.6.4	seulement	[illegible]
4.6.6		[illegible]
4.7	min	[illegible]

Machine Translated by Google

4.7	max	[illegible]
4.7.1	sol	
		[illegible]
4.7.2	plafond	[illegible]ur
	binôme	[illegible]ions
	liste	[illegible]vidé
	Permutations	[illegible]ns
		[illegible]
4.7.3	poisson	[illegible]que
4.7.5		[illegible]que
		[illegible]que
		[illegible]que
4.7.5	[illegible]	[illegible]que
4.7.5	[illegible]	[illegible]que
4.7.5		[illegible]que
4.7.5		[illegible]que
		[illegible]que
	arccoth	[illegible]que
	[illegible]	[illegible]que
		[illegible]que
		[illegible]h »
4.7.5		[illegible]h »
		[illegible]h »
		[illegible]h »
4.7.5	asech	[illegible]h »
4.7.5	[illegible]	[illegible]h »
4.8		[illegible]e

Machine Translated by Google

4.9	[illegible]ment –	[illegible]et
4.10	–	[illegible]dant
4.11	hexagone	
4.11	poubelle	
4.11	[illegible]t	
4.11		
	[illegible]gs	
	–	
	–	[illegible]te
	– – –	[illegible]étique [illegible]ne [illegible]ns
		[illegible]ns [illegible]ns [illegible]ns
	–	[illegible]ns [illegible]ns [illegible]ns
	–	[illegible]ns [illegible]ns [illegible]ns
[illegible]ectique	–	[illegible]ns [illegible]ns

Machine Translated by Google

	– –	[illegible]ns
	–	[illegible]ns [illegible]tion
	somme	
	– – –	[illegible]inearProgram
[illegible]er	– –	
[illegible]e	– –	
	–	
5.2.1	pour [illegible]tion	[illegible]ction [illegible]si [illegible]le
5.5.6	[illegible]mentation	[illegible]vidé [illegible]tions [illegible]ont
5.5.1	[illegible]xécution	[illegible]ion

Machine Translated by Google

5.5.2	autre	[illegible]
5.6.1	plus	[illegible]
5.7.3	casser	[illegible]
	ajouter	[illegible]
		[illegible]
	retirer	[illegible]
	modifications	
	ou	
	5.7.5	
5.7.6	gauche	
6.1	[illegible]	[illegible]
6.1		[illegible]
6,5		[illegible]
	multiplication	[illegible]

Machine Translated by Google

Section	Commande	Description
6.1	@interact	[illegible]
1.4		[illegible]
1.3	acos	[illegible] »
	acosh	[illegible]h »
4.7.5	acot	[illegible] »
4.7.5	acoth	[illegible]h »
	acsc	[illegible]que
	[illegible]nte	
	_	
	arccos	
4.7.5	arccosh	
1.3	arccoth	
4.7.5		
4.7.5	arccsc	
	arccsch	
1.3	arcsec	
	arcsinh	
4.7.5		
[illegible]ations	arctan	
	flèche	
	_ _ _	

Machine Translated by Google

1.3	asec	[illegible]ns
4.7.5	asech	
4.7.5	asinh	
	supposer	
4.7.5	[illegible]er	
3.1.1	–	
4.7.2		
5.6.3	casser	
4.7.1	plafond	
		[illegible]tique
		[illegible]ns
	[illegible] –	[illegible]les
		[illegible]tion
4,3,5	[illegible]	[illegible]rique
	matraque	[illegible]tique
	[illegible]el	[illegible]rique
		[illegible]tique
4,5	–	[illegible]le
4.7.5	csc	[illegible]rique
	csch	[illegible]tique
5.2.1		[illegible]le
1.11	dérivé	[illegible]onyme

Machine Translated by Google

4.2.2	autre	[illegible]
4.4.1	différent	
4.4.6		
	diviseurs	[illegible]ant
	[illegible]née	
	–	
	–	
	–	[illegible]te
		[illegible]
		[illegible]ation
1.12	hériter	[illegible]me
		Distribution
4.6.3	Fouiller –	[illegible]
1.2	développer	[illegible]entielles,
1.7		[illegible]
		parenthèses
1.7	[illegible]ent	[illegible]
2.6.1		[illegible]elle
	–	[illegible]che
4.3.1	–	[illegible]ximation
		[illegible]
		[illegible]ers
	[illegible]ius	[illegible]ction
		[illegible]ns

Machine Translated by Google

		[illegible]ns
4.22.1	[illegible]	
	–	
	–	
	–	[illegible]ns
	–	[illegible]ns
		[illegible]imal
4.4.2	–	
6.4.1		[illegible]otions
	–	[illegible]el
3.4	–	[illegible]se
5.7.3		[illegible]le
4.11	int	[illegible]it
1.12	intégrale	[illegible]es
1.12		[illegible]rale
4.4.4		[illegible]ie
	–	[illegible]mée
	[illegible]	[illegible]té
	–	[illegible]ns
		[illegible]mée
		[illegible]te
2.4.3		[illegible]es
4.6.1	lcm	[illegible]es
4.4.5	[illegible]	[illegible]sle
4.4.5	[illegible]	[illegible]le
		Matrice
4.6.4	seulement	[illegible]

Machine Translated by Google

5.7.6			[illegible]
4.8	limite		
	liste		
4.2.2	[illegible]		
			[illegible]ent
[illegible]	[illegible]erLinearProgram		
	min		
			[illegible]
1.2	N		[illegible]ation
[illegible]	–		
	[illegible]		
			[illegible]eur)
	[illegible]ation		[illegible]
	–		[illegible]de
3.9.2	–		[illegible]de
1.12	–		[illegible]ement,
			[illegible]
5.7.5	octobre		[illegible]t
	ou		[illegible]ions
3.6	paramétrique	–	[illegible]tique

Machine Translated by Google

1.12 4.7.2	partielle – Permutations	[illegible]
1.4 3.8	journal – – –	[illegible] [illegible]nt
	[illegible]ion –	
3.8.4	– –	
		[illegible]ns
	[illegible]nelle – – –	[illegible]ement nombre
	retirer – [illegible]	[illegible] [illegible] [illegible]
		[illegible]sion
4.4.5	[illegible] –	[illegible]ble

Machine Translated by Google

4.4.5	[illegible] –	[illegible] Matrice
5.8.3	rref	[illegible]
	[illegible]	[illegible]
	–	[illegible]
	–	[illegible]
1.10	–	[illegible] commande
1.3	seconde	[illegible]
4.7.5	sélecteur	[illegible]
6,5		[illegible]
	–	[illegible]
	[illegible]	[illegible]
5.7.3	mélanger	[illegible]
4.6.4	sigma	[illegible] entier
3.9.2	signe	[illegible]
		[illegible]
	péché	[illegible]
	né	[illegible]
	[illegible]	[illegible]
	–	[illegible]
		[illegible]
4.4.3	[illegible] –	[illegible]
4.4.3	[illegible] –	[illegible]
1.2		[illegible]

Machine Translated by Google

4.12	sudoku	[redacted]on
4.19	somme	
8.5	somme	[redacted]ment
	–	[redacted]ns
	[redacted]stique	[redacted]ure
	notre	
	vecteur	
	– –	
		[redacted]ns
	–	

Machine Translated by Google