Calcul formel

Vidéo ■ partie 1. Premiers pas avec Sage
Vidéo ■ partie 2. Structures de contrôle avec Sage
Vidéo ■ partie 3. Suites récurrentes et preuves formelles
Vidéo ■ partie 4. Suites récurrentes et visualisation
Vidéo ■ partie 5. Algèbre linéaire
Vidéo ■ partie 6a. Courbes et surfaces
Vidéo ■ partie 6b. Courbes et surfaces
Vidéo ■ partie 7. Calculs d'intégrales
Vidéo ■ partie 8. Polynômes
Vidéo ■ partie 9. Équations différentielles

1. Premiers pas avec Sage

Le calcul formel est un domaine charnière entre mathématiques et informatique. L'objectif de ce cours est d'obtenir des algorithmes efficaces afin de manipuler des objets mathématiques abstraits tels que les fonctions, les polynômes, les matrices, etc. À notre niveau, le calcul formel ou symbolique sera synonyme de «mathématiques effectives et efficaces».

1.1. Hello world !

Servons-nous d'abord de Sage comme d'une calculatrice :

Travaux pratiques 1.

1. Calculer $1+2\times 3^4$.
2. Calculer $\frac{22}{33}$.
3. Calculer $\cos(\frac{\pi}{6})$.

Voilà les résultats :

Code 1 (*hello-world.sage*)**.**

```
sage: 1+2*3^4
163
sage: 22/33
2/3
sage: cos(pi/6)
1/2*sqrt(3)
```

On retient que :

- Sage connaît les opérations classiques $+$, $-$, $\times$, $/$. La puissance a^b s'écrit `a^b` ou `a**b`.
- Sage fait des calculs exacts, il simplifie $\frac{22}{33}$ en $\frac{2}{3}$, contrairement à une banale calculatrice qui afficherait $0.6666\ldots$
- Sage manipule les fonctions et les constantes usuelles : par exemple $\cos\left(\frac{\pi}{6}\right)=\frac{\sqrt{3}}{2}$.

Dans toute la suite, on omettra l'invite de commande « `sage:` ». En fin de section vous trouverez davantage d'informations sur la mise en place de Sage.

1.2. Calcul formel

L'utilisation d'un système de calcul symbolique conduit le mathématicien à un type de démarche auquel il n'est traditionnellement pas habitué : l'expérimentation !

1. Je me pose des questions.
2. J'écris un algorithme pour tenter d'y répondre.
3. Je trouve une conjecture sur la base des résultats expérimentaux.
4. Je prouve la conjecture.

Travaux pratiques 2.

1. Que vaut le nombre complexe $(1-\mathrm{i})^k$, pour $k \geqslant 0$?
2. Les nombres de la forme $F_n = 2^{(2^n)} + 1$ sont-ils tous premiers ?

Nous pouvons faire des calculs avec les nombres complexes. En posant $z = 1-\mathrm{i}$, on calcule successivement la partie réelle (par la commande `real_part`), la partie imaginaire (`imag_part`), le module (`abs`) et l'argument (`arg`) de $z^0, z^1, z^2, \ldots$.

Code 2 (*motiv-calcul-formel.sage (1)*).

```
z = 1-I
for k in range(10):
    print k, z^k, real_part(z^k), imag_part(z^k), abs(z^k), arg(z^k)
```

```
0 1          5 4*I - 4      10 -32*I
1 -I + 1     6 8*I          11 -32*I - 32
2 -2*I       7 8*I + 8      12 -64
3 -2*I - 2   8 16           13 64*I - 64
4 -4         9 -16*I + 16   14 128*I
```

$$
\begin{aligned}
z^{4\ell} &= (-1)^\ell 2^{2\ell} \\
z^{4\ell+1} &= (-1)^\ell 2^{2\ell}(1-\mathrm{i}) \\
z^{4\ell+2} &= (-1)^\ell 2^{2\ell+1}(-\mathrm{i}) \\
z^{4\ell+3} &= (-1)^\ell 2^{2\ell+1}(-1-\mathrm{i})
\end{aligned}
$$

On remarque expérimentalement une structure assez simple. Par exemple pour passer de $z_k = (1+\mathrm{i})^k$ à $z_{k+1} = (1+\mathrm{i})^{k+1}$ le module est multiplié par $\sqrt{2}$ alors que l'argument change de $-\frac{\pi}{4}$. On en conjecture $z_{k+1} = \sqrt{2}e^{-\frac{\mathrm{i}\pi}{4}} z_k$. Comme $z_0 = (1-\mathrm{i})^0 = 1$ alors on conjecture $z_k = \sqrt{2}^k e^{-\frac{k\mathrm{i}\pi}{4}}$.
Passons à la preuve : en écrivant sous la forme module-argument $z = \sqrt{2}e^{-\frac{\mathrm{i}\pi}{4}}$, on obtient bien que $z^k = \sqrt{2}^k e^{-\frac{k\mathrm{i}\pi}{4}}$. On pourrait aussi obtenir une expression simple en discutant selon les valeurs de k modulo 8.

Le calcul formel permet aussi d'éviter de passer des années à essayer de montrer un résultat qui s'avère faux au final. Pierre de Fermat pensait que tous les nombres de la forme $F_n = 2^{2^n} + 1$ étaient premiers. Cent ans plus tard, Euler calcule que $F_5 = 4\,294\,967\,297 = 641 \times 6\,700\,417$ n'est pas un nombre premier.

Code 3 (*motiv-calcul-formel.sage (2)*).

```
for n in range(8):
    print n, factor(2^(2^n)+1)
```

1.3. Calcul formel vs calcul numérique

Même si Sage est conçu pour le calcul symbolique, il sait aussi effectuer des calculs numériques. Bien que non exact, le calcul numérique possède plusieurs avantages : il est plus rapide, souvent suffisant pour les applications pratiques et peut aussi être plus lisible.

Travaux pratiques 3.
Quelle est la limite de la suite définie par récurrence :

$$u_0 = 1 \qquad u_{n+1} = \sqrt{1+u_n} \ \text{ pour } n \geqslant 0 \quad ?$$

Si on calcule formellement les premiers termes on trouve :

$$u_0 = 1 \qquad u_1 = \sqrt{2} \qquad u_2 = \sqrt{1+\sqrt{2}} \qquad u_3 = \sqrt{1+\sqrt{1+\sqrt{2}}} \qquad u_4 = \sqrt{1+\sqrt{1+\sqrt{1+\sqrt{2}}}}$$

Ce qui n'éclaire pas vraiment sur le comportement de la suite.

Code 4 (*calcul-numerique.sage*).

```
u = 1
for i in range(10):
    u = sqrt(1 + u)
    print(u)
    print(numerical_approx(u))
```

Par contre au vu des approximations :

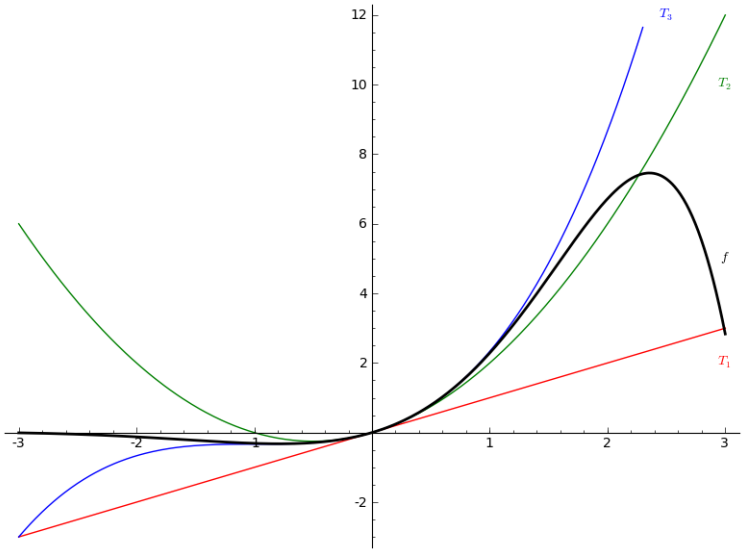
$$\begin{aligned} u_0 &= 1 \\ u_1 &= 1.4142\ldots \\ u_2 &= 1.5537\ldots \\ u_3 &= 1.5980\ldots \\ u_4 &= 1.6118\ldots \\ u_5 &= 1.6161\ldots \end{aligned}$$

on peut émettre plusieurs conjectures : la suite est croissante, elle est majorée par 2 et converge. En poussant les calculs, une approximation de la limite est $1.618033\ldots$ Les plus observateurs auront reconnu une approximation du nombre d'or $\phi = \frac{1+\sqrt{5}}{2}$, solution positive de l'équation $x^2 - x - 1$. On renvoie à un cours d'analyse pour la preuve que la suite (u_n) converge effectivement vers ϕ.

1.4. Graphiques

La production de graphiques est un formidable outil qui permet de mettre en lumière des phénomènes complexes.

Travaux pratiques 4.
Soit la fonction $f(x) = \sin(x) \cdot \exp(x)$. Calculer les premiers polynômes de Taylor associés aux développements limités de f en 0. Tracer leurs graphes. Quelles propriétés des développements limités cela met-il en évidence ?



- Après avoir défini la fonction f par `f = sin(x)*exp(x)`, la commande `taylor(f,x,0,n)` renvoie le DL de f en $x = 0$ à l'ordre n, par exemple ici $f(x) = x + x^2 + \frac{1}{3}x^3 + \epsilon(x)x^3$, donc $T_1(x) = x$, $T_2(x) = x + x^2$, $T_3(x) = x + x^2 + \frac{1}{3}x^3$.
- La commande `plot(f,(a,b))` trace le graphe de f sur l'intervalle $[a, b]$.

Il est perceptible que :

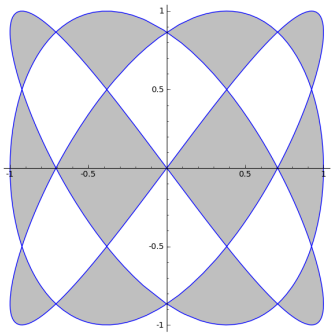
- Les polynômes de Taylor sont une bonne approximation de la fonction au voisinage d'un point.

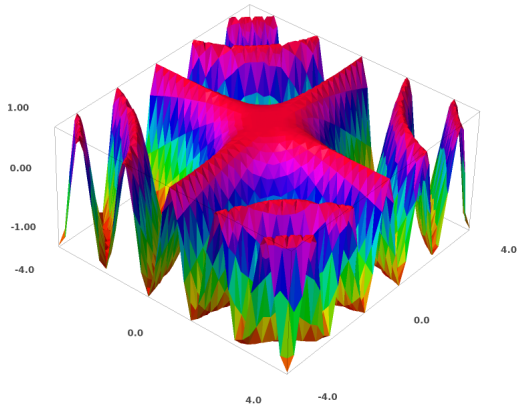
- Plus l'ordre du DL est élevé, meilleure est l'approximation.
- L'approximation est seulement *locale* : loin du point considéré (ici l'origine) les polynômes de Taylor n'approchent plus du tout la fonction.

Il est possible de tracer une grande variété de graphiques. Voici par exemple la courbe de Lissajous d'équation $t \mapsto \big(\cos(3t), \sin(4t)\big)$ et le graphe de la fonction de deux variables définie par $f(x,y) = \cos(xy)$. Les commandes sont :

- `parametric_plot((cos(3*t),sin(4*t)),(t,0,2*pi))`
- `plot3d(cos(x*y),(x,-4,4),(y,-4,4))`

Attention ! Il faut avoir au préalable définir les variables utilisées : `var('t')` et `var('x,y')`. (En fait, seule la variable x est définie par défaut dans Sage.)





1.5. Le calcul formel peut-il tout faire ?

Le calcul formel ne résout malheureusement pas tous les problèmes de mathématique d'un coup de baguette magique !

Travaux pratiques 5.

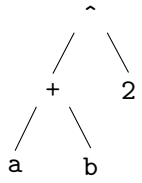
1. Pouvez-vous calculer les solutions réelles de $x^k - x - 1 = 0$ pour les entiers $k \geqslant 2$?
2. Est-ce que Sage sait que toutes ces expressions sont nulles ?

$$2^{10} - 1024 \qquad (x+1)^2 - x^2 - 2x - 1 \qquad \sin^2(x) + \cos^2(x) - 1$$

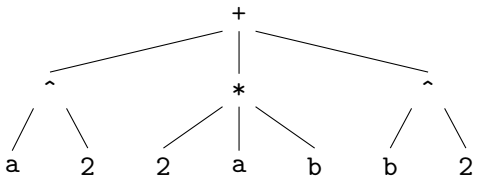
1. La première limitation est propre aux mathématiques : on ne peut pas trouver une écriture explicite des solutions de toutes les équations. Pour $x^2 - x - 1 = 0$ à l'aide de la commande `solve(x^2-x-1==0,x)` on trouve bien les deux solutions $\frac{1+\sqrt{5}}{2}$ et $\frac{1-\sqrt{5}}{2}$. Par contre `solve(x^5-x-1==0,x)` ne renvoie pas les solutions, mais il renvoie l'équation qui définit les solutions (ce qui ne nous avance guère). Ce n'est pas ici un problème de Sage. En effet, il n'est mathématiquement pas possible d'exprimer la solution réelle de $x^5 - x - 1 = 0$ à l'aide de racines ($\sqrt{}, \sqrt{}, \sqrt[3]{}, \sqrt[4]{}, \ldots$). C'est seulement possible jusqu'au degré 4. Par contre on obtient une approximation de la solution réelle par la commande `find_root(x^5-x-1==0,-1,2)` en précisant que l'on cherche la solution sur l'intervalle $[-1, 2]$.
2. (a) Sans problème `2^10-1024` renvoie 0.
   (b) Il est nécessaire de développer pour trouver 0 : `expand((x+1)^2-x^2-2*x-1)`.
   (c) Il faut explicitement préciser de simplifier l'expression trigonométrique : d'abord `f = sin(x)^2+cos(x)^2 - 1` puis on demande de simplifier `f.simplify_trig()` pour obtenir 0.

Il n'est pas du tout évident pour un ordinateur de reconnaître les identités comme $(a+b)^2 = a^2 + 2ab + b^2$ ou bien $\cos^2(x) + \sin^2(x) = 1$. Souvenez-vous d'ailleurs qu'il faut plusieurs années d'apprentissage pour les assimiler. Lorsqu'il y a plusieurs écritures d'une même expression, il n'est pas non plus évident pour l'ordinateur de savoir quelle forme est la plus adaptée à l'utilisateur. Par exemple, selon le contexte, les trois écritures sont utiles : $(a-b)^3 = (a-b)(a^2 - 2ab + b^2) = a^3 - 3a^2b + 3a^2b - b^3$. Il faudra donc «guider» le logiciel de calcul formel avec les fonctions `expand`, `factor`, `simplify`...

Remarque.
Pour avoir une idée de la difficulté à identifier deux expressions, voici une représentation de la façon dont les expressions $(a+b)^2$ et $a^2+2ab+b^2$ sont stockées dans le logiciel.



$(a+b)^2$



$a^2+2ab+b^2$

1.6. Un peu plus sur Sage

Ce cours n'a pas pour but d'être un manuel d'utilisation du logiciel Sage. Vous trouverez sur internet des tutoriels pour démarrer et pour un usage avancé :

Site officiel de Sage

Il existe aussi un livre gratuit :

Calcul mathématique avec Sage

Une façon simple d'obtenir de l'aide pour une commande Sage est d'utiliser le point d'interrogation : `ln?` (ou bien `help(ln)`). Vous y apprendrez que la fonction `ln` est le logarithme naturel.

Il y a deux façons d'utiliser Sage :

- *En ligne de commande* : vous obtenez une fenêtre avec l'invite `sage` : puis vous tapez vos commandes (en n'oubliant pas de faire une copie de votre travail dans un fichier texte du type `mon_programme.sage`).
- *Dans votre navigateur* : à partir de l'invite `sage:` vous tapez `notebook()` pour obtenir une interface complète et conviviale.

Voici une liste de fonctions usuelles :

`abs(x)`	$\lvert x\rvert$
`x^n` ou `x**n`	x^n
`sqrt(x)`	$\sqrt{x}$
`exp(x)`	$\exp x$
`ln(x)` ou `log(x)`	$\ln x$ logarithme népérien
`log(x,10)`	$\log x$ logarithme décimal
`cos(x), sin(x), tan(x)`	$\cos x$, $\sin x$, $\tan x$ en radians
`arccos(x), arcsin(x), arctan(x)`	$\arccos x$, $\arcsin x$, $\arctan x$ en radians
`floor(x)`	partie entière $E(x)$: plus grand entier $n \leqslant x$ (*floor* = plancher)
`ceil(x)`	plus petit entier $n \geqslant x$ (*ceil* = plafond)

Il existe des fonctions spécifiques qui manipulent les entiers, les vecteurs, les matrices, les polynômes, les fonctions mathématiques... Nous les découvrions au fil des chapitres.

La syntaxe de Sage est celle du langage Python. Il n'est pas nécessaire de connaître ce langage, la syntaxe sera introduite au fur et à mesure. Cependant vous pourrez étudier avec profit le chapitre «Algorithmes et mathématiques». Pour l'instant voici ce que l'on retient de l'exemple

Code 5 (*motiv-calcul-formel.sage (2)*).
```
for n in range(8):
    print n, factor(2^(2^n)+1)
```

- Une boucle `for n in range(N)` : l'indice n parcourt les entiers de 0 à $N-1$.
- Le bloc d'instructions suivant `print n, factor(2^(2^n)+1)` est donc exécuté successivement pour $n = 0, 1, \ldots, N-1$.
- Les espaces en début de ligne (*l'indentation*) sont essentielles car elles délimitent le début et la fin d'un bloc d'instructions.

2. Structures de contrôle avec Sage

2.1. Boucles

Boucle `for` (pour)

Pour faire varier un élément dans un ensemble on utilise l'instruction "`for x in ensemble:`". Le bloc d'instructions suivant sera successivement exécuté pour toutes les valeurs de x.

Code 6 (*structures.sage (1)*).
```
for x in ensemble:
    première ligne de la boucle
    deuxième ligne de la boucle
    ...
    dernière ligne de la boucle
instructions suivantes
```

Notez encore une fois que le bloc d'instructions exécuté est délimité par les espaces en début de ligne. Un exemple fréquent est "`for k in range(n):`" qui fait varier k de 0 à $n-1$.

Liste `range` (intervalle)

En fait `range(n)` renvoie la liste des n premiers entiers : `[0,1,2,...,n-1]`
Plus généralement `range(a,b)` renvoie la liste `[a,a+1,...,b-1]`. Alors que `range(a,b,c)` effectue une saut de c termes, par exemple `range(0,101,5)` renvoie la liste `[0,5,10,15,...,100]`.
Une autre façon légèrement différente pour définir des listes d'entiers est l'instruction `[a..b]` qui renvoie la liste des entiers k tels que $a \leqslant k \leqslant b$. Par exemple après l'instruction `for k in [-7..12]:` l'entier k va prendre successivement les valeurs -7, -6, -5, ..., -1, 0, $+1$,... jusqu'à $+12$.
La boucle `for` permet plus généralement de parcourir n'importe quelle liste, par exemple `for x in [0.13,0.31,0.53,0.98]:` fait prendre à x les 4 valeurs de $\{0.13, 0.31, 0.53, 0.98\}$.

Boucle `while` (tant que)

Code 7 (*structures.sage (2)*).
```
while condition:
    première ligne de la boucle
    deuxième ligne de la boucle
    ...
    dernière ligne de la boucle
instructions suivantes
```

La boucle `while` exécute le bloc d'instructions, tant que la condition est vérifiée. Lorsque la condition n'est plus vérifiée, on passe aux instructions suivantes.
Voici le calcul de la racine carrée entière d'un entier n :

Code 8 (*structures.sage (3)*).
```
n = 123456789      # l'entier dont on cherche la racine carrée
k = 1              # le premier candidat
```

```
while k*k <= n:    # tant que le carré de k ne dépasse pas n
    k = k+1        # on passe au candidat suivant
print(k-1)         # la racine cherchée
```

Lorsque la recherche est terminée, k est le plus petit entier dont le carré dépasse n, par conséquent la racine carrée entière de n est $k-1$.
Notez qu'utiliser une boucle « tant que » comporte des risques, en effet il faut toujours s'assurer que la boucle se termine. Voici quelques instructions utiles pour les boucles : `break` termine immédiatement la boucle alors que `continue` passe directement à l'itération suivante (sans exécuter le reste du bloc).

Test `if... else` (si... sinon)

Code 9 (*structures.sage (4)*)**.**
```
if condition:
    première ligne d'instruction
␣␣␣␣...
␣␣␣␣dernière␣ligne␣d'instruction
else:
    autres instructions
```

Si la condition est vérifiée, c'est le premier bloc d'instructions qui est exécuté, si la condition n'est pas vérifiée, c'est le second bloc. On passe ensuite aux instructions suivantes.

2.2. Booléens et conditions

Booléens

Une ***expression booléenne*** est une expression qui peut seulement prendre les valeurs « Vrai » ou « Faux » et qui sont codées par `True` ou `False`. Les expressions booléennes s'obtiennent principalement par des conditions.

Quelques conditions

Voici une liste de conditions :
- `a < b` : teste l'inégalité stricte $a < b$,
- `a > b` : teste l'inégalité stricte $a > b$,
- `a <= b` : teste l'inégalité large $a \leqslant b$,
- `a >= b` : teste l'inégalité large $a \geqslant b$,
- `a == b` : teste l'égalité $a = b$,
- `a <> b` (ou `a != b`) : teste la non égalité $a \neq b$.
- `a in B` : teste si l'élément a appartient à B.

Remarque. 1. Une condition prend la valeur `True` si elle est vraie et `False` sinon. Par exemple `x == 2` renvoie `True` si x vaut 2 et `False` sinon. La valeur de x n'est pas modifiée pas le test.

2. Il ne faut surtout pas confondre le test d'égalité `x == 2` avec l'affectation `x = 2` (après cette dernière instruction x vaut 2).

3. On peut combiner des conditions avec les opérateurs `and`, `or`, `not`. Par exemple : `(n>0) and (not (is_prime(n)))` est vraie si et seulement si n est strictement positif et non premier.

Travaux pratiques 6.

1. Pour deux assertions logiques P, Q écrire l'assertion $P \implies Q$.
2. Une ***tautologie*** est une assertion vraie quelles que soient les valeurs des paramètres, par exemple (P ou (non P)) est vraie que l'assertion P soit vraie ou fausse. Montrer que l'assertion suivante est une tautologie :

$$\text{non}\left(\left[\text{non } (P \text{ et } Q) \text{ et } (Q \text{ ou } R)\right] \text{ et } \left[P \text{ et (non } R)\right]\right)$$

Il faut se souvenir du cours de logique qui définit l'assertion « $P \implies Q$ » comme étant « non (P) ou Q ». Il suffit donc de renvoyer : `not(P) or Q`.
Pour l'examen de la tautologie, on teste toutes les possibilités et on vérifie que le résultat est vrai dans chaque cas !

Code 10 (*structures.sage (10)*).
```
for P in {True, False}:
    for Q in {True, False}:
        for R in {True, False}:
            print(not( (not(P and Q) and (Q or R)) and ( P and (not R) ) ))
```

2.3. Fonctions informatiques

Une fonction informatique prend en entrée un ou plusieurs paramètres et renvoie un ou plusieurs résultats.

Code 11 (*structures.sage (5)*).
```
def mafonction (mesvariables):
    première ligne d'instruction
␣␣␣␣...
␣␣␣␣dernière␣ligne␣d'instruction
    return monresultat
```

Par exemple voici comment définir une fonction valeur absolue.

Code 12 (*structures.sage (6)*).
```
def valeur_absolue(x):
    if x >= 0:
        return x
    else:
        return -x
```

Par exemple `valeur_absolue(-3)` renvoie +3.
Il est possible d'utiliser cette fonction dans le reste du programme ou dans une autre fonction. Noter aussi qu'une fonction peut prendre plusieurs paramètres.
Par exemple que fait la fonction suivante ? Quel nom aurait été plus approprié ?

Code 13 (*structures.sage (7)*).
```
def ma_fonction(x,y):
    resultat = (x+y+valeur_absolue(x-y))/2
    return resultat
```

Travaux pratiques 7.
On considère trois réels distincts a, b, c et on définit le polynôme

$$P(X) = \frac{(X-a)(X-b)}{(c-a)(c-b)} + \frac{(X-b)(X-c)}{(a-b)(a-c)} + \frac{(X-a)(X-c)}{(b-a)(b-c)} - 1$$

1. Définir trois variables par `var('a,b,c')`. Définir une fonction `polynome(x)` qui renvoie $P(x)$.
2. Calculer $P(a)$, $P(b)$, $P(c)$.
3. Comment un polynôme de degré 2 pourrait avoir 3 racines distinctes ? Expliquez ! (Vous pourrez appliquer la méthode `full_simplify()` à votre fonction.)

2.4. Variable locale/variable globale

Il faut bien faire la distinction entre les variables locales et les variables globales.
Par exemple, qu'affiche l'instruction `print(x)` dans ce petit programme ?

Code 14 (*structures.sage (9)*).
```
x = 2
def incremente(x):
    x = x+1
    return x
incremente(x)
print(x)
```

Il faut bien comprendre que tous les `x` de la fonction `incremente` représentent une variable locale qui n'existe que dans cette fonction et disparaît en dehors. On aurait tout aussi bien pu définir `def incremente(y): y = y+1 return y` ou bien utiliser la variable `truc` ou `zut`. Par contre les `x` des lignes `x=2`, `incremente(x)`, `print(x)` correspondent à une variable globale. Une variable globale existe partout (sauf dans les fonctions où une variable locale porte le même nom !).
En mathématique, on parle plutôt de variable muette au lieu de variable locale, par exemple dans

$$\sum_{k=0}^{n} k^2$$

k est une variable muette, on aurait pu tout aussi bien la nommer i ou x. Par contre il faut au préalable avoir défini n (comme une variable globale), par exemple « $n = 7$ » ou « fixons un entier n »...

2.5. Conjecture de Syracuse

Mettez immédiatement en pratique les structures de contrôle avec le travail suivant.

Travaux pratiques 8.
La ***suite de Syracuse*** de terme initial $u_0 \in \mathbb{N}^*$ est définie par récurrence pour $n \geqslant 0$ par :

$$u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$$

1. Écrire une fonction qui, à partir de u_0 et de n, calcule le terme u_n de la suite de Syracuse.
2. Vérifier pour différentes valeurs du terme initial u_0, que la suite de Syracuse atteint la valeur 1 au bout d'un certain rang (puis devient périodique : $..., 1, 4, 2, 1, 4, 2, ...$). Écrire une fonction qui pour un certain choix de u_0 renvoie le premier rang n tel que $u_n = 1$.

Personne ne sait prouver que pour toutes les valeurs du terme initial u_0, la suite de Syracuse atteint toujours la valeur 1.
Voici le code pour calculer le n-ème terme de la suite.

Code 15 (*suite-syracuse.sage (1)*).
```
def syracuse(u0,n):
    u = u0
    for k in range(n):
        if u%2 == 0:
            u = u//2
        else:
            u = 3*u+1
    return u
```

Quelques remarques : on a utilisé une boucle `for` pour calculer les n premiers termes de la suite et aussi un test `if...else`.

L'instruction `u%2` renvoie le reste de la division de u par 2. Le booléen `u%2 == 0` teste donc la parité de u. L'instruction `u//2` renvoie le quotient de la division euclidienne de u par 2.

Remarque.
Si $a \in \mathbb{Z}$ et $n \in \mathbb{N}^*$ alors

- `a%n` est le ***reste*** de la division euclidienne de a par n. C'est donc l'entier r tel que $r \equiv a \pmod{n}$ et $0 \leqslant r < n$.
- `a//n` est le ***quotient*** de la division euclidienne de a par n. C'est donc l'entier q tel que $a = nq + r$ et $0 \leqslant r < n$.

Il ne faut pas confondre la division de nombres réels : `a/b` et le quotient de la division euclidienne de deux entiers : `a//b`. Par exemple `7/2` est la fraction $\frac{7}{2}$ dont la valeur numérique est $3,5$ alors que `7//2` renvoie 3. On a `7%2` qui vaut 1. On a bien $7 = 2 \times 3 + 1$.
Pour notre cas `u%2` renvoie le reste de la division euclidienne de u par 2. Donc cela vaut 0 si u est pair et 1 si u est impair. Ainsi `u%2 == 0` teste si u est pair ou impair. Ensuite `u = u//2` divise u par 2 (en fait cette opération n'est effectuée que pour u pair).

Pour calculer à partir de quel rang on obtient 1, on utilise une boucle `while` dont les instructions sont exécutées tant que $u \neq 1$. Donc par construction quand la boucle se termine c'est que $u = 1$.

Code 16 (*suite-syracuse.sage (2)*)**.**
```
def vol_syracuse(u0):
    u = u0
    n = 0
    while u <> 1:
        n = n+1
        if u%2 == 0:
            u = u//2
        else:
            u = 3*u+1
    return n
```

3. Suites récurrentes et preuves formelles

Commençons par un exemple. Après avoir déclaré les deux variables a et b par `var('a,b')` alors la commande

`expand((a+b)^2-a^2-2*a*b-b^2)`

renvoie 0. Ce simple calcul doit être considéré comme une preuve par ordinateur de l'identité $(a+b)^2 = a^2 + 2ab + b^2$, sans qu'il y ait obligation de vérification. Cette « preuve » est fondée sur la confiance en `Sage` qui sait manipuler les expressions algébriques. Nous allons utiliser la puissance du calcul formel, nous seulement pour faire des expérimentations, mais aussi pour effectuer des preuves (presque) à notre place.

Rappelons la démarche que l'on adopte :

1. Je me pose des questions.
2. J'écris un algorithme pour tenter d'y répondre.
3. Je trouve une conjecture sur la base des résultats expérimentaux.
4. Je prouve la conjecture.

3.1. La suite de Fibonacci

La suite de Fibonacci est définie par les relations suivantes :

$$F_0 = 0 \qquad F_1 = 1 \qquad F_n = F_{n-1} + F_{n-2} \quad \text{pour } n \geqslant 2.$$

Travaux pratiques 9.

1. Calculer les 10 premiers termes de la suite de Fibonacci.
2. **Recherche d'une conjecture.** On cherche s'il peut exister des constantes $\phi, \psi, c \in \mathbb{R}$ telles que

$$F_n = c\left(\phi^n - \psi^n\right)$$

Nous allons calculer les constantes qui conviendraient.

(a) On pose $G_n = \phi^n$. Quelle équation doit satisfaire ϕ afin que $G_n = G_{n-1} + G_{n-2}$?

(b) Résoudre cette équation.

(c) Calculer c, ϕ, ψ.

3. **Preuve.** On note $H_n = \frac{1}{\sqrt{5}}\left(\phi^n - \psi^n\right)$ où $\phi = \frac{1+\sqrt{5}}{2}$ et $\psi = \frac{1-\sqrt{5}}{2}$.

(a) Vérifier que $F_n = H_n$ pour les premières valeurs de n.

(b) Montrer à l'aide du calcul formel que $H_n = H_{n-1} + H_{n-2}$. Conclure.

1. Voici une fonction qui calcule le terme de rang n de la suite de Fibonacci en appliquant la formule de récurrence.

Code 17 (*fibonacci.sage (1)*)**.**
```
def fibonacci(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    F_n_2 = 0
    F_n_1 = 1
    for k in range(n-1):
        F_n = F_n_1 + F_n_2
        F_n_2 = F_n_1
        F_n_1 = F_n
    return F_n
```

On affiche les valeurs de F_0 à F_9 :

Code 18 (*fibonacci.sage (2)*)**.**
```
for n in range(10):
    print fibonacci(n)
```

Ce qui donne :

0 1 1 2 3 5 8 13 21 34

2. (a) On pose $G_n = \phi^n$ et on suppose que $G_n = G_{n-1} + G_{n-2}$. C'est-à-dire que l'on veut résoudre $\phi^n = \phi^{n-1} + \phi^{n-2}$. Notons (E) cette équation $X^n = X^{n-1} + X^{n-2}$, ce qui équivaut à l'équation $X^{n-2}\left(X^2 - X - 1\right) = 0$. On écarte la solution triviale $X = 0$, pour obtenir l'équation $X^2 - X - 1 = 0$.

(b) On profite de Sage pour calculer les deux racines de cette dernière équation.

Code 19 (*fibonacci.sage (3)*)**.**
```
solutions = solve(x^2-x-1==0,x)
print(solutions)

var('phi,psi')
phi = solutions[1].rhs()
psi = solutions[0].rhs()
print 'phi␣:␣',phi,'␣␣␣psi␣:␣',psi
```

Les solutions sont fournies sous la forme d'une liste que l'on appelle `solutions`. On récupère chaque solution avec `solutions[0]` et `solutions[1]`. Par exemple `solutions[1]` renvoie `x == 1/2*sqrt(5) + 1/2`. Pour récupérer la partie droite de cette solution et la nommer ϕ, on utilise la fonction `rhs()` (pour *right hand side*) qui renvoie la partie droite d'une équation (de même `lhs()`, pour *left hand side* renverrait la partie gauche d'une équation).

On obtient donc les deux racines

$$\phi = \frac{1+\sqrt{5}}{2} \qquad \text{et} \qquad \psi = \frac{1-\sqrt{5}}{2}.$$

ϕ^n et ψ^n vérifient la même relation de récurrence que la suite de Fibonacci mais bien sûr les valeurs prises ne sont pas entières.

(c) Comme $F_1 = 1$ alors $c(\phi - \psi) = 1$, ce qui donne $c = \frac{1}{\sqrt{5}}$.

La conséquence de notre analyse est que *si* une formule $F_n = c(\phi^n - \psi^n)$ existe alors les constantes sont nécessairement $c = \frac{1}{\sqrt{5}}$, $\phi = \frac{1+\sqrt{5}}{2}$, $\psi = \frac{1-\sqrt{5}}{2}$.

3. Dans cette question, nous allons maintenant montrer qu'avec ces constantes on a effectivement

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\tfrac{1+\sqrt{5}}{2}\right)^n - \left(\tfrac{1-\sqrt{5}}{2}\right)^n\right).$$

(a) On définit une nouvelle fonction `conjec` qui correspond à H_n, la conjecture que l'on souhaite montrer. On vérifie que l'on a l'égalité souhaitée pour les premières valeurs de n.

Code 20 (*fibonacci.sage (4)*).

```
def conjec(n):
    return 1/sqrt(5)*(phi^n-psi^n)

for n in range(10):
    valeur = fibonacci(n)-conjec(n)
    print expand(valeur)
```

(b) Nous allons montrer que $F_n = H_n$ pour tout $n \geqslant 0$ en s'aidant du calcul formel.

- **Initialisation.** Il est clair que $F_0 = H_0 = 0$ et $F_1 = H_1 = 1$. On peut même demander à l'ordinateur de le faire, en effet `fibonacci(0)-conjec(0)` et `fibonacci(1)-conjec(1)` renvoient tous les deux 0.
- **Relation de récurrence.** Nous allons montrer que H_n vérifie exactement la même relation de récurrence que les nombres de Fibonacci. Encore une fois on peut le faire à la main, ou bien demander à l'ordinateur de le faire pour nous :

  Code 21 (*fibonacci.sage (5)*).

  ```
  var('n')
  test = conjec(n)-conjec(n-1)-conjec(n-2)
  print(test.simplify_radical())
  ```

  Le résultat est 0. Comme le calcul est valable pour la variable formelle n, il est vrai quel que soit $n \geqslant 0$. Ainsi $H_n = H_{n-1} + H_{n-2}$. Il a tout de même fallu préciser à Sage de simplifier les racines carrées avec la commande `simplify_radical()`.
- **Conclusion.** Les suites (F_n) et (H_n) ont les mêmes termes initiaux et vérifient la même relation de récurrence. Elles ont donc les mêmes termes pour chaque rang : pour tout $n \geqslant 0$, $F_n = H_n$.

3.2. L'identité de Cassini

A votre tour maintenant d'expérimenter, conjecturer et prouver *l'identité de Cassini*.

Travaux pratiques 10.
Calculer, pour différentes valeurs de n, la quantité

$$F_{n+1}F_{n-1} - F_n^2.$$

Que constatez-vous ? Proposez une formule générale. Prouvez cette formule qui s'appelle l'identité de Cassini.

Notons pour $n \geqslant 1$:

$$C_n = F_{n+1}F_{n-1} - F_n^2.$$

Après quelques tests, on conjecture la formule très simple $C_n = (-1)^n$.
Voici deux preuves : une preuve utilisant le calcul formel et une autre à la main.

Première preuve – À l'aide du calcul formel.
On utilise la formule

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

et on demande simplement à l'ordinateur de vérifier l'identité !

Code 22 (*cassini.sage (1)*).
```
def fibonacci_bis(n):
    return 1/sqrt(5)*( ((1+sqrt(5))/2)^n - ((1-sqrt(5))/2)^n )
```

Code 23 (*cassini.sage (2)*).
```
var('n')
cassini = fibonacci_bis(n+1)*fibonacci_bis(n-1)-fibonacci_bis(n)^2
print(cassini.simplify_radical())
```

L'ordinateur effectue (presque) tous les calculs pour nous. Le seul problème est que Sage échoue de peu à simplifier l'expression contenant des racines carrées (peut-être les versions futures feront mieux ?). En effet, après simplification on obtient :

$$C_n = (-1)^n \frac{(\sqrt{5}-1)^n(\sqrt{5}+1)^n}{2^{2n}}.$$

Il ne reste qu'à conclure à la main. En utilisant l'identité remarquable $(a-b)(a+b) = a^2 - b^2$, on obtient $(\sqrt{5}-1)(\sqrt{5}+1) = (\sqrt{5})^2 - 1 = 4$ et donc $C_n = (-1)^n \frac{4^n}{2^{2n}} = (-1)^n$.

Seconde preuve – Par récurrence.
On souhaite montrer que l'assertion

$$(\mathscr{H}_n) \qquad F_n^2 = F_{n+1}F_{n-1} - (-1)^n$$

est vrai pour tout $n \geqslant 1$.

- Pour $n = 1$, $F_1^2 = 1^2 = 1 \times 0 - (-1)^1 = F_2 \times F_0 - (-1)^1$. $\mathscr{H}_1$ est donc vraie.
- Fixons $n \geqslant 1$ et supposons $\mathscr{H}_n$ vraie. Montrons $\mathscr{H}_{n+1}$ est vraie. On a :

  $$F_{n+1}^2 = F_{n+1} \times (F_n + F_{n-1}) = F_{n+1}F_n + F_{n+1}F_{n-1}.$$

  Par l'hypothèse $\mathscr{H}_n$, $F_{n+1}F_{n-1} = F_n^2 + (-1)^n$. Donc

  $$F_{n+1}^2 = F_{n+1}F_n + F_n^2 + (-1)^n = (F_{n+1} + F_n)F_n - (-1)^{n+1} = F_{n+2}F_n - (-1)^{n+1}.$$

  Donc $\mathscr{H}_{n+1}$ est vraie.
- Conclusion, par le principe de récurrence, pour tout $n \geqslant 1$, l'identité de Cassini est vérifiée.

3.3. Une autre suite récurrente double

Travaux pratiques 11.
Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par

$$u_0 = \frac{3}{2}, \qquad u_1 = \frac{5}{3} \qquad \text{et} \qquad u_n = 1 + 4\frac{u_{n-2} - 1}{u_{n-1}u_{n-2}} \quad \text{pour } n \geqslant 2.$$

Calculer les premiers termes de cette suite. Émettre une conjecture. La prouver par récurrence.
Indication. Les puissances de 2 seront utiles...

Commençons par la partie expérimentale, le calcul des premières valeurs de la suite (u_n) :

Code 24 (*suite-recurrente.sage (1)*).
```
def masuite(n):
    if n == 0:
        return 3/2
    if n == 1:
        return 5/3
    u_n_2, u_n_1 = 3/2, 5/3
```

```
    for k in range(n-1):
        u_n = 1+4*(u_n_2-1)/(u_n_1*u_n_2)
        u_n_2, u_n_1 = u_n_1, u_n
    return u_n
```

(Notez l'utilisation de la double affectation du type `x, y = newx, newy`. C'est très pratique comme par exemple dans `a, b = a+b, a-b` et évite l'introduction d'une variable auxiliaire.)
Cela permet de calculer les premiers termes de la suite (u_n) :

$$\frac{3}{2} \quad \frac{5}{3} \quad \frac{9}{5} \quad \frac{17}{9} \quad \frac{33}{17} \quad \frac{65}{33} \quad \dots$$

Au vu des premiers termes, on a envie de dire que notre suite (u_n) coïncide avec la suite (v_n) définie par :

$$v_n = \frac{2^{n+1}+1}{2^n+1}.$$

On définit donc une fonction qui renvoie la valeur de v_n en fonction de n.

Code 25 (*suite-recurrente.sage (2)*)**.**
```
def conjec(n):
    return (1+2^(n+1))/(1+2^n)
```

Pour renforcer notre conjecture, on vérifie que (u_n) et (v_n) sont égales pour les 20 premiers termes :

Code 26 (*suite-recurrente.sage (3)*)**.**
```
for n in range(20):
    print n,'␣',masuite(n),'␣',conjec(n)
```

Il est à noter que l'on peut calculer u_n pour des valeurs aussi grandes que l'on veut de n mais que l'on n'a pas une formule directe pour calculer u_n en fonction de n. L'intérêt de (v_n) est de fournir une telle formule.

Voici maintenant une preuve assistée par l'ordinateur. Notre conjecture à démontrer est : pour tout $n \geqslant 0$, $v_n = u_n$. Pour cela, on va prouver que v_n et u_n coïncident sur les deux premiers termes et vérifient la même relation de récurrence. Ce seront donc deux suites égales.

- **Initialisation.** Vérifions que $v_0 = u_0$ et $v_1 = u_1$. Il suffit de vérifier que `conjec(0)-3/2` et `conjec(1)-5/3` renvoient tous les deux 0.
- **Relation de récurrence.** Par définition (u_n) vérifie la relation de récurrence $u_n = 1 + 4\frac{u_{n-2}-1}{u_{n-1}u_{n-2}}$. Définissons une fonction correspondant à cette relation : à partir d'une valeur x (correspond à u_{n-1}) et y (correspondant à u_{n-2}) on renvoie $1 + 4\frac{y-1}{xy}$.

  Code 27 (*suite-recurrente.sage (4)*)**.**
  ```
  def recur(u_n_1,u_n_2):
     return 1+4*(u_n_2-1)/(u_n_1*u_n_2)
  ```

  On demande maintenant à l'ordinateur de vérifier que la suite (v_n) vérifie aussi cette relation de récurrence :

  Code 28 (*suite-recurrente.sage (5)*)**.**
  ```
  var('n')
  test = conjec(n)-recur(conjec(n-1),conjec(n-2))
  print(test.simplify_rational())
  ```

  Le résultat obtenu est 0. Donc v_n vérifie pour tout n la relation $v_n = 1 + 4\frac{v_{n-2}-1}{v_{n-1}v_{n-2}}$.
- **Conclusion.** Les suites (v_n) et (u_n) sont égales aux rangs 0 et 1 et vérifient la même relation de récurrence pour $n \geqslant 2$. Ainsi $u_n = v_n$, pour tout $n \in \mathbb{N}$. Les suites (u_n) et (v_n) sont donc égales.

Notez qu'il a fallu guider `Sage` pour simplifier notre résultat à l'aide de la fonction `simplify_rational` qui simplifie les fractions rationnelles.

Remarque.
Quelques commentaires sur les dernières lignes de code.

- La commande `var('n')` est essentielle. Elle permet de réinitialiser la variable n en une variable formelle. Souvenez-vous qu'auparavant n valait 19. Après l'instruction `var('n')` la variable n redevient une indéterminée `'n'` sans valeur spécifique.
- Pour bien comprendre ce point, supposons que vous souhaitiez vérifier que $(n+2)^3 = n^3 + 6n^2 + 12n + 8$. Vous pouvez-bien sûr le vérifier par exemple pour les vingts premiers rang $n = 0, 1, 2, \ldots, 19$. Mais en définissant la variable formelle `var('n')` et à l'aide de la commande `expand((n+2)^3-n^3-6*n^2-12*n-8))` le logiciel de calcul formel «prouve» directement que l'égalité est vérifiée pour tout n. Si vous oubliez de réinitialiser la variable n en une variable formelle, vous ne l'aurez vérifiée que pour l'unique valeur $n = 19$!

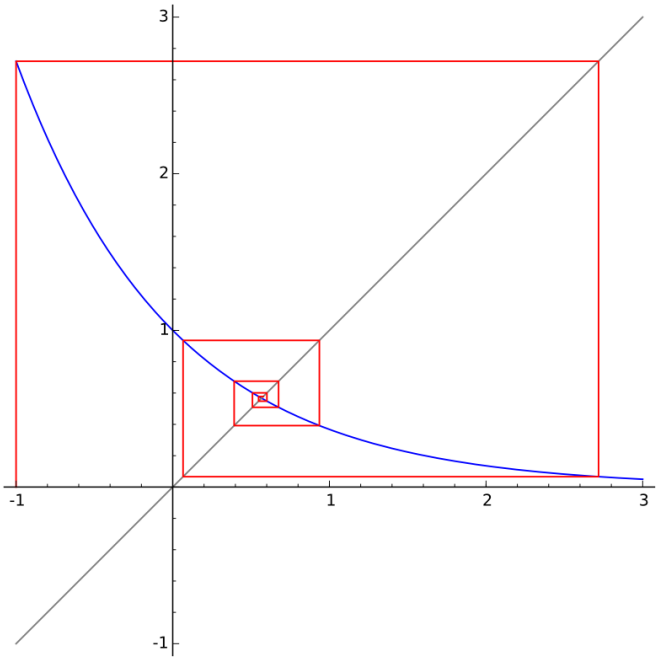
4. Suites récurrentes et visualisation

4.1. Visualiser une suite récurrente

Travaux pratiques 12.
Fixons $a \in \mathbb{R}$. Définissons une suite $(u_n)_{n\in\mathbb{N}}$ par récurrence :

$$u_0 = a \qquad \text{et} \qquad u_{n+1} = \exp(-u_n) \quad \text{pour } n \geqslant 0.$$

1. Calculer les premiers valeurs de la suite pour $a = -1$. Écrire une fonction qui renvoie ces premières valeurs sous la forme d'une liste.
2. Sur un même graphique tracer le graphe de la fonction de récurrence $f(x) = \exp(-x)$, la bissectrice $(y = x)$ et la trace de la suite récurrente, c'est-à-dire la ligne brisée joignant les points $\big(u_k, f(u_k)\big)$, (u_{k+1}, u_{k+1}) et $\big(u_{k+1}, f(u_{k+1})\big)$.
3. Émettre plusieurs conjectures : la suite (ou certaines sous-suites) sont-elles croissantes ou décroissantes ? Majorées, minorées ? Convergentes ?
4. Prouver vos conjectures.



1. On définit la fonction $f(x) = \exp(-x)$ par la commande `f(x) = exp(-x)`.

Code 29 (*suites-visual.sage (1)*).

```
def liste_suite(f,terme_init,n):
    maliste = []
```

```
    x = terme_init
    for k in range(n):
        maliste.append(x)
        x = f(x)
    return maliste
```

Par exemple la commande `liste_suite(f,-1,4)` calcule, pour $a=-1$, les 4 premiers termes de la suite (u_n) ; on obtient la liste :

`[-1, e, e^(-e), e^(-e^(-e))]`

correspondant aux termes : $u_0=-1$, $u_1=e$, $u_2=e^{-e}$ et $u_3=e^{-e^{-e}}$, où l'on note $e=\exp(1)$.

2. Nous allons maintenant calculer une liste de points. On démarre du point initial $(u_0,0)$, puis pour chaque rang on calcule deux points : $\big(u_k,f(u_k)\big)$ et $\big(u_{k+1},u_{k+1}\big)$. Notez que chaque élément de la liste est ici un couple (x,y) de coordonnées.

Code 30 (*suites-visual.sage (2)*).

```
def liste_points(f,terme_init,n):
    u = liste_suite(f,terme_init,n)
    mespoints = [ (u[0],0) ]
    for k in range(n-1):
        mespoints.append( (u[k],u[k+1]) )
        mespoints.append( (u[k+1],u[k+1]) )
    return mespoints
```

Par exemple la commande `liste_points(f,-1,3)` calcule, pour $a=-1$, le point initial $(-1,0)$ et les 3 premiers points de la visualisation de la suite (u_n) ; on obtient la liste :

`[(-1, 0), (-1, e), (e, e), (e, e^(-e)), (e^(-e), e^(-e))]`

Il ne reste plus qu'à tracer le graphe de la fonction et construire la suite en traçant les segments reliant les points.

Code 31 (*suites-visual.sage (3)*).

```
def dessine_suite(f,terme_init,n):
    mespoints = liste_points(f,terme_init,n)
    G = plot(f,(x,-1,3))      # La fonction
    G = G + plot(x,(x,-1,3))  # La droite (y=x)
    G = G + line(mespoints)   # La suite
    G.show()
```

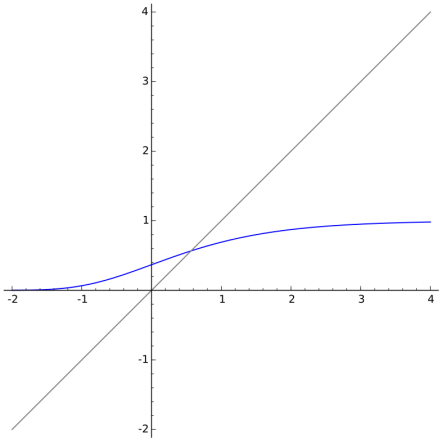
Par exemple la figure illustrant ce tp est construite par la commande `dessine_suite(f,-1,10)`.

3. (et 4.) Passons à la partie mathématique. Pour simplifier l'étude on va supposer $a=-1$. Donc $u_0=-1$ et $u_1=f(u_0)=\exp(1)=e$.
   (a) La fonction f définie par $f(x)=\exp(-x)$ est décroissante. Donc la fonction g définie par $g(x)=f\big(f(x)\big)$ est croissante.
   (b) La suite $(u_{2n})_{n\in\mathbb{N}}$ des termes de rangs pairs est croissante. En effet : $u_2=e^{-e}\geqslant -1=u_0$. Alors $u_4=f\circ f(u_2)\geqslant f\circ f(u_0)=u_2$ car $f\circ f=g$ est croissante. Puis par récurrence $u_6=f\circ f(u_4)\geqslant f\circ f(u_2)=u_4$... La suite (u_{2n}) est croissante.
   (c) Comme $u_2\geqslant u_0$ et que f est décroissante alors $u_3\leqslant u_1$. On démontre alors de la même façon que la suite $(u_{2n+1})_{n\in\mathbb{N}}$ des termes de rangs impairs est décroissante.
   (d) Enfin, toujours par la même méthode et en partant de $u_0\leqslant u_1$, on prouve que $u_{2n}\leqslant u_{2n+1}$.
   (e) La situation est donc la suivante :

$$u_0\leqslant u_2\leqslant\cdots\leqslant u_{2n}\leqslant\cdots\leqslant u_{2n+1}\leqslant\cdots\leqslant u_3\leqslant u_1$$

   La suite (u_{2n}) est croissante et majorée par u_1, donc elle converge. Notons ℓ sa limite.
   La suite (u_{2n+1}) est décroissante et minorée par u_0, donc elle converge. Notons ℓ' sa limite.

(f) Par les théorèmes usuels d'analyse, la suite (u_{2n}) converge vers un point fixe de la fonction $g = f \circ f$, c'est-à-dire une valeur x_0 vérifiant $f \circ f(x_0) = x_0$. Une étude de la fonction g (ou plus précisément de $g(x)-x$) montrerait que g n'a qu'un seul point fixe. Voici le graphe de g.



Cela prouve en particulier que $\ell = \ell'$.

(g) Par ailleurs la fonction f admet un unique point fixe x_0. Mais comme $f(x_0) = x_0$ alors l'égalité $f\big(f(x_0)\big) = f\big(x_0\big) = x_0$ prouve que x_0 est aussi le point fixe de g.

(h) Conclusion : la suite (u_{2n}) et la suite (u_{2n+1}) convergent vers $x_0 = \ell = \ell'$. Ainsi la suite (u_n) converge vers x_0 le point fixe de f.

(i) Il n'y a pas d'expression simple de la solution x_0 de l'équation $f(x) = x$. La commande `solve(f(x)==x,x)` renvoie seulement l'équation. Par contre, on obtient une valeur numérique approchée par `find_root(f(x)==x,0,1)`. On trouve $x_0 = 0,56714329041\ldots$

4.2. Listes

Une ***liste*** est ce qui ressemble le plus à une suite mathématique. Une liste est une suite de nombres, de points... ou même de listes !

- Une liste est présentée entre crochets : `mesprems = [2,3,5,7,11,13]`. On accède aux valeurs par `mesprems[i]` ; `mesprems[0]` vaut 2, `mesprems[1]` vaut 3...
- On parcourt les éléments d'une liste avec `for p in mesprems:`, p vaut alors successivement 2, 3, 5, 7...
- Une première façon de créer une liste est de partir de la liste vide, qui s'écrit `[]`, puis d'ajouter un à un des éléments à l'aide de la méthode `append`. Par exemple `mespoints=[]`, puis `mespoints.append((2,3))`, `mespoints.append((7,-1))`. La liste `mespoints` contient alors deux éléments (ici deux points) `[ (2,3), (7,-1) ]`.

Travaux pratiques 13.
Pour n un entier fixé. Composer les listes suivantes :

1. la liste des entiers de 0 à $n-1$,
2. la liste des entiers premiers strictement inférieurs à n,
3. la liste des $2p+1$, pour les premiers p strictement inférieurs à n,
4. les 10 premiers éléments de la liste précédente,
5. la liste de $p_i + i$, où $(p_i)_{i\geqslant 0}$ sont les nombres premiers strictement inférieurs à n ($p_0 = 2$, $p_1 = 3$,...),
6. la liste de 1 et 0 selon que le rang $0 \leqslant k < n$ soit premier ou pas.

Une utilisation intelligente permet un code très court et très lisible. Il faut potasser le manuel afin de manipuler les listes avec aisance.

```
1. entiers = range(n)
2. premiers = [k for k in range(n) if is_prime(k)]
3. doubles = [2*p+1 for p in premiers]
4. debut = doubles[0:10]
5. premiersplusrang = [premiers[i]+i for i in range(len(premiers))]
```

6. `binaire = [zeroun(k) for k in range(n)]` où `zeroun(k)` est une fonction à définir qui renvoie 1 ou 0 selon que k est premier ou pas.

On peut aussi trier les listes, les fusionner...

4.3. Suites et chaos

Travaux pratiques 14.
On considère la fonction f définie sur $x \in [0,1]$ par

$$f(x) = rx(1-x) \qquad \text{où} \qquad 0 \leqslant r \leqslant 4.$$

Pour r fixé et $u_0 \in [0,1]$ fixé, on définit une suite (u_n) par

$$u_{n+1} = f(u_n).$$

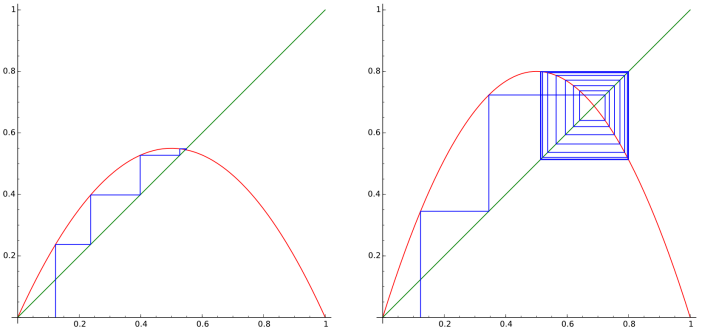
1. Pour différentes valeurs de r et u_0 fixées, tracer sur un même graphique le graphe de f, la droite d'équation $(y = x)$ et la suite (u_n). Essayez de conjecturer le comportement de la suite pour $0 < r \leqslant 3$, $3 < r \leqslant 1+\sqrt{6}$, $1+\sqrt{6} < r < 4$ et $r = 4$.
2. Pour u_0 fixé et $M < N$ grands (par exemple $M = 100$, $N = 200$) tracer le ***diagramme de bifurcation*** de f, c'est-à-dire l'ensemble des points

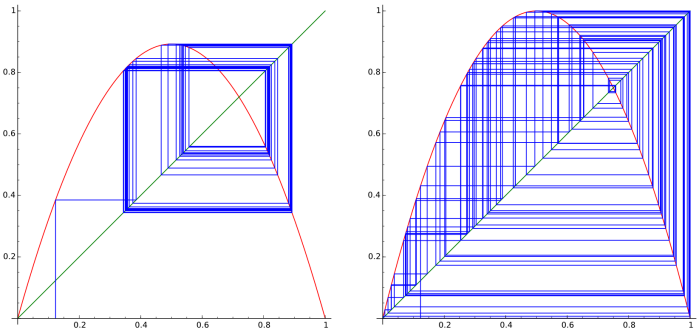
$$(u_i, r) \qquad \text{pour} \qquad M \leqslant i \leqslant N \text{ et } 0 \leqslant r \leqslant 4$$

3. (a) Montrer que les points fixes de f sont 0 et $\frac{r-1}{r}$.
   (b) Montrer que le point fixe 0 est répulsif (c'est-à-dire $|f'(0)| > 1$) dès que $r > 1$.
   (c) Montrer que le point fixe $\frac{r-1}{r}$ est attractif (c'est-à-dire $|f'(\frac{r-1}{r})| < 1$) si et seulement si $1 < r < 3$.
4. **Cas $1 < r \leqslant 3$. Pour $0 < u_0 < 1$ la suite (u_n) converge vers $\frac{r-1}{r}$.**
   On va prouver ce résultat seulement dans le cas particulier $r = 2$:
   (a) Montrer que $u_{n+1} - \frac{1}{2} = -2\left(u_n - \frac{1}{2}\right)^2$.
   (b) Montrer que si $|u_0 - \frac{1}{2}| < k < \frac{1}{2}$ alors $|u_n - \frac{1}{2}| < \frac{1}{2}(2k)^{2^n}$.
   (c) Conclure.
5. **Cas $3 < r < 1+\sqrt{6}$. La suite (u_n) possède un cycle attracteur de période 2.**
   (a) Déterminer les points fixes ℓ_1 et ℓ_2 de $f \circ f$ qui ne sont pas des points fixes de f.
   (b) Montrer que $f(\ell_1) = \ell_2$ et $f(\ell_2) = \ell_1$.
   (c) À l'aide du graphe de $f \circ f$, vérifier graphiquement sur un exemple, que les suites (u_{2n}) et (u_{2n+1}) sont croissantes ou décroissantes à partir d'un certain rang et convergent, l'une vers ℓ_1, l'autre vers ℓ_2.
6. **Cas $1+\sqrt{6} < r < 3,5699456\ldots$ La suite (u_n) possède un cycle attracteur de période** $4, 8, 16\ldots$
   Trouver de tels exemples.
7. **À partir de $r > 3,5699456\ldots$ la suite (u_n) devient chaotique.**

Le tp suivant sera consacré au cas $r = 4$.

1. Voici le comportement de différentes suites définies par le même $u_0 = 0,123$ et pour r valant successivement $r = 2,2$, $r = 3,2$, $r = 3,57$ et $r = 4$.



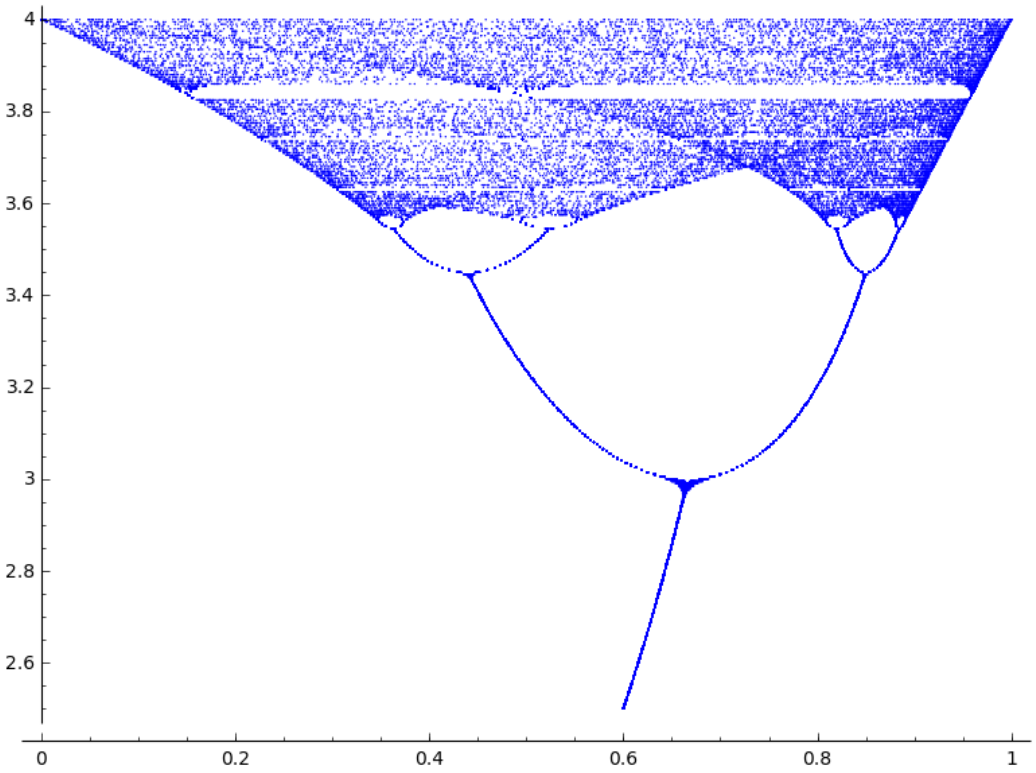


On voit d'abord une limite, puis la suite semble avoir « deux limites » c'est-à-dire deux valeurs d'adhérence, puis 4 valeurs d'adhérence. Pour $r = 4$ la situation semble chaotique.

2. Voici le code et le résultat de l'exécution `bifurcation(f,0.102)`, où `f(x,r) = r*x*(1-x)`.

Code 32 (*suites-chaos.sage (1)*).

```
def bifurcation(F,terme_init):
    Nmin = 100              # On oublie Nmin premiers termes
    Nmax = 200              # On conserve les termes entre Nmin et Nmax
    epsilon = 0.005         # On fait varier r de epsilon à chaque pas
    r = 2.5                 # r initial
    mespoints = []
    while r <= 4.0:
        u = liste_suite(F(r=r),terme_init,Nmax)  # On calcule la suite
        for k in range(Nmin,Nmax):
            mespoints = mespoints + [(u[k],r)]
        r = r + epsilon
    G = points(mespoints)
    G.show()
```



3. Voici le code pour les trois questions.

Code 33 (*suites-chaos.sage (2)*).
```
var('x,r')
f(x,r) = r*x*(1-x)
pts_fixes = solve(f==x,x)      # (a) Points fixes
ff = diff(f,x)                 # Dérivée
ff(x=0)                        # (b) f'(0)
solve(abs(ff(x=(r-1)/r))<1,r) # (c) Condition point attractif
```

(a) Les deux points fixes fournis dans la liste `pts_fixes` sont 0 et $\frac{r-1}{r}$.

(b) On calcule la dérivée `ff`, on l'évalue en 0, on trouve $f'(0) = r$. Ainsi si $r > 1$ alors $|f'(0)| > 1$ et le point 0 est répulsif.

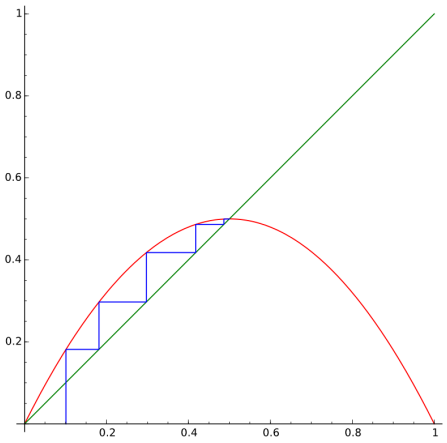
(c) Par contre lorsque l'on résout l'inéquation $|f'(\frac{r-1}{r})| < 1$, la machine renvoie les conditions $r > 1$ et $r < 3$. Ainsi pour $1 < r < 3$, le point fixe $\frac{r-1}{r}$ est attractif.

4. On pose d'abord `r = 2`, alors le point fixe est $\frac{r-1}{r} = \frac{1}{2}$.

(a) Après simplification `(r*u*(1-u) - 1/2) + 2*(u-1/2)^2` vaut 0. Autrement dit $u_{n+1} - \frac{1}{2} = -2(u_n - \frac{1}{2})^2$, quelque soit u_n.

(b) C'est une simple récurrence :
$$\left|u_{n+1} - \frac{1}{2}\right| = 2\left(u_n - \frac{1}{2}\right)^2 < 2\left(\frac{1}{2}(2k)^{2^n}\right)^2 = \frac{1}{2}(2k)^{2^{n+1}}.$$

(c) Ainsi pour $u_0 \neq 0, 1$, il existe k tel que $|u_0 - \frac{1}{2}| < k < \frac{1}{2}$ et la suite (u_n) tend (très très) vite vers le point fixe $\frac{1}{2}$. Avec $r = 2$, en quelques itérations le terme de la suite n'est plus discernable de la limite $\frac{1}{2}$:



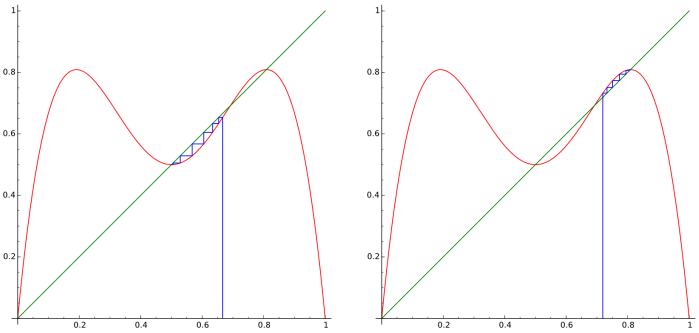
5. Voici le code pour les deux premières questions :

Code 34 (*suites-chaos.sage (3)*).
```
var('x,r')
f(x,r) = r*x*(1-x)              # f
g = f(x=f(x,r))                 # g(x) = f(f(x))
pts_doubles = solve(g==x,x)     # (a) Points fixes de g
# Les deux nouveaux points fixes de g :
ell1 = pts_doubles[0].rhs()
ell2 = pts_doubles[1].rhs()
eq = f(x=ell1)-ell2             # (b) l'image de ell1 est-elle ell2 ?
eq.full_simplify()              # Oui !
```

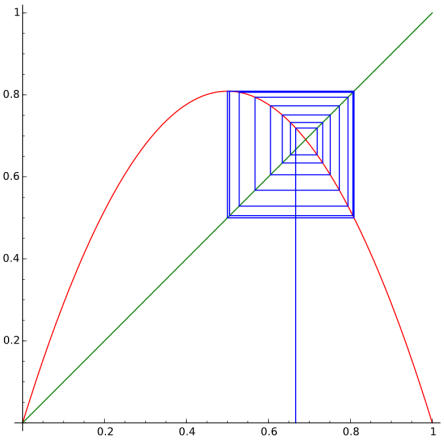
(a) On calcule les points fixes de $g = f \circ f$. Il y en a quatre, mais deux d'entre eux sont les points fixes de f. Les deux nouveaux sont :
$$\ell_1 = \frac{1}{2}\frac{r+1-\sqrt{r^2-2r-3}}{r} \qquad \ell_2 = \frac{1}{2}\frac{r+1+\sqrt{r^2-2r-3}}{r}$$

(b) Bien sûr ℓ_1 et ℓ_2 ne sont pas des points fixes pour f. Par contre, on vérifie que $f(\ell_1) = \ell_2$ et $f(\ell_2) = \ell_1$.

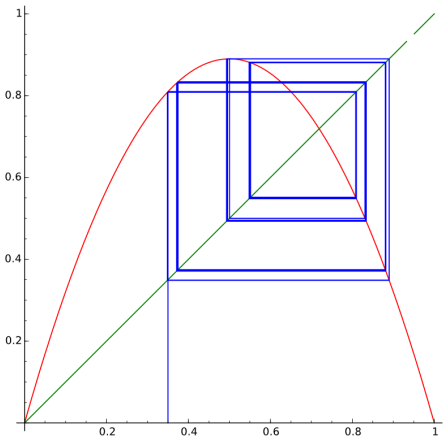
(c) Voici les dessins pour $r = 1+\sqrt{5}$ et $u_0 = \frac{2}{3}$. La suite (u_{2n}), qui est vue comme suite récurrente de fonction g, est décroissante vers ℓ_1 (figure de gauche). La suite (u_{2n+1}), vue comme suite récurrente de fonction g, est croissante vers ℓ_2 (figure de droite).



La suite (u_n), comme suite récurrente de fonction f, possède le cycle (ℓ_1, ℓ_2) comme cycle attracteur :



6. Voici un exemple de cycle de longueur 8 ($r = 3,56$, $u_0 = 0,35$) :



7. Le cas $r = 4$ sera étudié dans le tp suivant.

Travaux pratiques 15.

On s'intéresse au cas $r = 4$. On rappelle que

$$f(x) = rx(1-x) \qquad u_0 \in [0,1] \qquad \text{et} \qquad u_{n+1} = f(u_n) \quad (n \geqslant 0).$$

1. (a) Montrer que $\sin^2(2t) = 4\sin^2 t \cdot (1-\sin^2 t)$.
   (b) Montrer que pour $u_0 \in [0,1]$, il existe un unique $t \in [0, \frac{\pi}{2}]$ tel que $u_0 = \sin^2 t$.
   (c) En déduire $u_n = \sin^2(2^n t)$.
2. Pour $t_k = \frac{2\pi}{2^k+1}$ et $u_0 = \sin^2 t_k$, montrer que la suite (u_n) est périodique de période k.
3. **La suite est instable.**

(a) Pour $k=3$, on pose $t_3=\frac{2\pi}{9}$, $u_0=\sin^2 t_3$. Calculer une valeur exacte (puis approchée) de u_n, pour tout n.

(b) Partant d'une valeur approchée $\tilde{u}_0$ de u_0, que donne la suite des approximations $(\tilde{u}_n)$ définie par la relation de récurrence ?

4. **La suite est partout dense.**

On va construire $u_0\in[0,1]$ tel que, tout $x\in[0,1]$ peut être approché d'aussi près que l'on veut par un terme u_n de notre suite :

$$\forall x\in[0,1]\quad\forall\epsilon>0\quad\exists n\in\mathbb{N}\qquad \left|x-u_n\right|<\epsilon.$$

- Soit σ la constante binaire de Champernowne, formée de la juxtaposition des entiers en écriture binaire à 1, 2, 3,... chiffres 0, 1, 00, 01, 10,... dont l'écriture binaire est $\sigma=0,0100011011\ldots$
- Soit $u_0=\sin^2(\pi\sigma)$.
- Soit $x\in[0,1[$. Ce réel peut s'écrire $x=\sin^2(\pi t)$ avec $t\in[0,1[$ qui admet une écriture binaire $t=0,a_1a_2\ldots a_p\ldots$.

Pour $\epsilon>0$ fixé, montrer qu'il existe n tel que $\left|x-u_n\right|<\epsilon$.

1. (a) On pose `eq = sin(2*t)^2 - 4*sin(t)^2*(1-sin(t)^2)` et `eq.full_simplify()` renvoie 0. Donc $\sin^2(2t)=4\sin^2 t\cdot(1-\sin^2 t)$, pour tout $t\in\mathbb{R}$.

(b) Soit $h(t)=\sin^2 t$. On définit la fonction `h = sin(t)^2`, sa dérivée `hh = diff(h,t)` dont on cherche les zéros par `solve(hh==0,t)`. La dérivée ne s'annulant qu'en $t=0$ et $t=\frac{\pi}{2}$. La fonction h' est strictement monotone sur $[0,\frac{\pi}{2}]$. Comme $h(0)=0$ et $h(\frac{\pi}{2})=1$, pour chaque $u_0\in[0,1]$, il existe un unique $t\in[0,\frac{\pi}{2}]$ tel que $u_0=h(t)$.

(c) Pour $u_0=\sin^2 t$, on a

$$u_1=f(u_0)=ru_0(1-u_0)=4\sin^2 t\cdot(1-\sin^2 t)=\sin^2(2t).$$

On montre de même par récurrence que $u_n=\sin^2(2^n t)$.

2. Le code suivant calcule u_k-u_0. Sage sait calculer que cela vaut 0, pour k prenant la valeur 0, 1, 2,... mais pas lorsque k est une variable. Le calcul à la main est pourtant très simple !

Code 35 (*suites-chaos.sage (4)*)**.**

```
t = 2*pi/(2^k+1)
u_0 = sin(t)^2
u_k = sin(2^k*t-2*pi)^2
zero = u_k-u_0
zero.simplify_trig()
```

3. **La suite est instable.**

(a) Pour $k=3$ et $t_3=\frac{2\pi}{9}$, la suite (u_k) est périodique de période 3. Donc il suffit de calculer u_0,u_1,u_2. Par exemple :

$$u_{3p}=u_0=\sin^2\left(\frac{2\pi}{9}\right)\simeq 0{,}413\,175\,911\,1\ldots$$

(b) Partons de la valeur approchée de u_0 avec 10 décimales exactes : $\tilde{u}_0=0{,}413\,175\,911\,1$ et calculons les premiers termes de la suite. On en extrait :

$$\begin{aligned}
\tilde{u}_0 &\simeq 0{,}413\,175\,911\,100\\
\tilde{u}_3 &\simeq 0{,}413\,175\,911\,698\\
\tilde{u}_6 &\simeq 0{,}413\,175\,906\,908\\
\tilde{u}_9 &\simeq 0{,}413\,175\,945\,232\\
\tilde{u}_{12} &\simeq 0{,}413\,175\,638\,640\\
\tilde{u}_{15} &\simeq 0{,}413\,178\,091\,373\\
\tilde{u}_{18} &\simeq 0{,}413\,158\,469\,568\\
\tilde{u}_{21} &\simeq 0{,}413\,315\,447\,870\\
\tilde{u}_{24} &\simeq 0{,}412\,059\,869\,464\\
\tilde{u}_{27} &\simeq 0{,}422\,119\,825\,829\\
\tilde{u}_{30} &\simeq 0{,}342\,897\,499\,745\\
\tilde{u}_{33} &\simeq 0{,}916\,955\,513\,784\\
\tilde{u}_{36} &\simeq 0{,}517\,632\,311\,613\\
\tilde{u}_{39} &\simeq 0{,}019\,774\,022\,431
\end{aligned}$$

Si l'approximation de départ et tous les calculs étaient exacts, on devrait obtenir u_0 à chaque ligne. On constate que l'erreur augmente terme après terme, et après $\tilde{u}_{30}$, le comportement de $\tilde{u}_n$ n'a plus rien à voir avec u_n. L'explication vient de la formule $u_n = \sin^2(2^n t)$, une erreur sur t, même infime au départ, devient grande par multiplication par 2^n.

4. **La suite est partout dense.**

   La construction est assez jolie mais délicate. (Il ne faut pas avoir peur de l'écriture en base 2 qui n'est pas ici le point clé. Vous pouvez penser en base 10 pour mieux comprendre.)

   Fixons $\epsilon > 0$. Soit un entier p tel que $\frac{\pi}{2^p} < \epsilon$. L'écriture binaire de t étant $t = 0,a_1a_2\ldots a_p\ldots$ la séquence $a_1a_2\ldots a_p$ apparaît quelque part dans la constante de Champernowne. Plus précisément il existe un entier n tel que $2^n\sigma = \ldots b_2b_1b_0,a_1a_2\ldots a_p\ldots$ (on a fait apparaître la séquence juste après la virgule par décalage de la virgule). On va pouvoir oublier la partie entière $m = \ldots b_2b_1b_0 \in \mathbb{N}$ et on note $\tilde{\sigma} = 0,a_1a_2\ldots a_p\ldots$ la partie fractionnaire de σ, de sorte que $2^n\sigma = m + \tilde{\sigma}$.

$$\begin{aligned}
|x - u_n| &= \left|\sin^2(\pi t) - \sin^2(2^n\pi\sigma)\right|\\
&= \left|\sin^2(\pi t) - \sin^2(2^n\pi m + \pi\tilde{\sigma})\right|\\
&= \left|\sin^2(\pi t) - \sin^2(\pi\tilde{\sigma})\right| \qquad \text{car } \sin^2 \text{ est } \pi\text{-périodique}\\
&\leqslant |\pi t - \pi\tilde{\sigma}| \qquad \text{par le théorème des accroissements finis}\\
&\leqslant \pi\frac{1}{2^p} \qquad \text{car } t \text{ et } \tilde{\sigma} \text{ ont les mêmes } p \text{ premières décimales}\\
&< \epsilon
\end{aligned}$$

   Conclusion : n'importe quel x est approché d'aussi près que l'on veut par la suite.

Pour en savoir plus :

- *La suite logistique et le chaos*, Daniel Perrin.
- *Étude d'une suite récurrente*, Jean-Michel Ferrard.

5. Algèbre linéaire

L'algèbre linéaire est la partie des mathématiques qui s'occupe des matrices et des structures vectorielles. Beaucoup de problèmes se ramènent ou s'approchent par des problèmes linéaires, pour lesquels il existe souvent des solutions efficaces.

5.1. Opérations de base

Travaux pratiques 16.
Soient les matrices :

$$A=\begin{pmatrix}1&2&3\\-1&0&1\\0&1&0\end{pmatrix} \qquad B=\begin{pmatrix}2&-1&0\\-1&0&1\end{pmatrix} \qquad u=\begin{pmatrix}1\\x\\x^2\end{pmatrix} \qquad v=\begin{pmatrix}1\\0\\1\end{pmatrix}$$

1. Calculer tous les produits possibles à partir de A, B, u, v.
2. Calculer $(A-I)^7$ et en extraire le coefficient en position $(2,3)$.
3. Calculer A^{-1}. Calculer la trace de A^{-1} (c'est-à-dire la somme des coefficients sur la diagonale).

1. On définit une matrice n lignes et p colonnes par la commande

   `matrix(n,p,[ [ligne1],[ligne2],... ])`

   Pour nous cela donne :

   Code 36 (*alglin.sage (1)*).
   ```
   A = matrix(3,3,[[1,2,3],[-1,0,1],[0,1,0]])
   B = matrix(2,3,[[2,-1,0],[-1,0,1]])
   I = identity_matrix(3)
   u = matrix(3,1,[1,x,x^2])
   v = matrix(3,1,[1,0,1])
   ```

   On multiplie deux matrices par `A*B`. Cette opération est possible seulement si le nombre de colonnes de A est égal au nombre de lignes de B. Les produits possibles sont donc ici : $B\times A$, $A\times u$, $A\times v$, $B\times u$, $B\times v$.

2. Ayant défini la matrice identité I (par `I = identity_matrix(3)`), on calcule $M=(A-I)^7$ par `(A-I)^7`, ce qui renvoie

   $$(A-I)^7=\begin{pmatrix}-16&-46&11\\-25&-73&153\\32&57&-137\end{pmatrix}.$$

   Le coefficient en position $(2,3)$ d'une matrice M est celui sur la deuxième ligne et la troisième colonne. Donc ici c'est 153. Une matrice A se comporte en Sage comme une liste à double entrée et on accède aux coefficients par la commande `A[i,j]` (`i` pour les lignes, `j` pour les colonnes). Attention ! Les listes étant indexées à partir du rang 0, les indices `i`, `j` partent de 0. Le coefficient en position $(2,3)$ s'obtient donc par la commande :

   `((A-I)^7)[1,2]`

   Faites bien attention au décalage des deux indices.

3. L'inverse d'une matrice `A` se calcule par `A^-1` ou `A.inverse()`. Ce qui donne

   $$A^{-1}=\begin{pmatrix}\frac14&-\frac34&-\frac12\\0&0&1\\\frac14&\frac14&-\frac12\end{pmatrix}.$$

   La trace se calcule par une simple somme des éléments sur la diagonale principale :

   `sum( (A^-1)[i,i] for i in range(3) )`

   et vaut ici $-\frac14$.

Remarque. • Noter encore une fois que pour une matrice de taille n et pour Sage les indices varient de 0 à $n-1$.
- On peut préciser le corps sur lequel on travaille, pour nous ce sera le plus souvent $K=\mathbb{Q}$. Par exemple : `A = matrix(QQ, [[1,2],[3,4]])`.
- Avec Sage on peut aussi définir des vecteurs par `vector([x1,x2,...,xn])`. Un vecteur peut représenter soit un vecteur ligne, soit un vecteur colonne. On peut multiplier une matrice par un vecteur (à gauche ou à droite).
- Si on définit un vecteur, par exemple `v = vector([1,2,3,4])` alors `L = matrix(v)` renvoie la matrice ligne correspondante. `C = L.transpose()` renvoie la matrice colonne correspondante.

Travaux pratiques 17.
Pour une matrice $A \in M_n(\mathbb{K})$ inversible et des vecteurs colonnes u, v à n lignes on définit $\alpha \in \mathbb{K}$ par :

$$\alpha = 1 + v^T A^{-1} u.$$

La formule de Sherman-Morrison affirme que si $\alpha \neq 0$ alors

$$\left(A + uv^T\right)^{-1} = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{\alpha}$$

Prouver par le calcul formel cette formule pour les matrices de taille 2×2 et 3×3.

Connaissant l'inverse de A, cette formule permet de calculer à moindre coût l'inverse d'une déformation de A par une matrice de rang 1. Le code ne pose pas de problème.

Code 37 (*alglin.sage (2)*).
```
var('a,b,c,d,x,y,xx,yy')
A = matrix(2,2,[[a,b],[c,d]])
u = matrix(2,1,[x,y])
v = matrix(2,1,[xx,yy])

alpha_matrice = 1 + v.transpose() * (A^-1) * u
alpha_reel = alpha_matrice[0,0]
B = ( A + u * v.transpose() )^-1
BB  = A^-1 - 1/alpha_reel*( A^-1 * u * v.transpose() * A^-1 )
C = B - BB

for i in range(A.nrows()):
    for j in range(A.ncols()):
        print( "indices␣:",i,j,"coeff␣:", (C[i,j]).full_simplify() )
```

- On commence par définir les coefficients qui seront nos variables (ici pour $n = 2$).
- On calcule α par la formule $\alpha = 1 + v^T A^{-1} u$, qui est une matrice de taille 1×1 (`alpha_matrice`), identifiée à un réel (`alpha_reel`).
- On calcule ensuite les termes de gauche (`B`) et droite (`BB`) de la formule à prouver. Pour la matrice `C = B - BB`, on vérifie (après simplification) que tous ses coefficients sont nuls. Ce qui prouve l'égalité cherchée.
- On pourrait également obtenir directement le résultat en demandant à Sage : `C.simplify_rational()==matrix(2,2,0)`.

Pour une preuve à la main et en toute dimension, multiplier $\left(A + uv^T\right)$ par $A^{-1} - \frac{A^{-1}uv^T A^{-1}}{\alpha}$.

5.2. Réduction de Gauss, calcul de l'inverse

Le but de cette section est est de mettre en œuvre la méthode de Gauss. Cette méthode est valable pour des systèmes linéaires (ou des matrices) de taille quelconque, mais ici on se limite au calcul de l'inverse d'une matrice (qui est obligatoirement carrée). N'hésitez pas à d'abord relire votre cours sur la méthode de Gauss.

Travaux pratiques 18.

1. Réaliser trois fonctions qui transforment une matrice en fonction des opérations élémentaires sur les lignes :
   - $L_i \leftarrow cL_i$ avec $c \neq 0$: on multiplie une ligne par un réel ;
   - $L_i \leftarrow L_i + cL_j$: on ajoute à la ligne L_i un multiple d'une autre ligne L_j ;
   - $L_i \leftrightarrow L_j$: on échange deux lignes.
2. À l'aide de ces transformations élémentaires, transformer par la méthode de Gauss une matrice inversible en une matrice échelonnée (c'est-à-dire ici, en partant d'une matrice carrée, obtenir une matrice triangulaire supérieure).
3. Toujours à l'aide de ces transformations et de la méthode de Gauss, transformer cette matrice échelonnée en une matrice échelonnée et réduite (c'est-à-dire ici, en partant d'une matrice inversible, obtenir l'identité).
4. La méthode de Gauss permet de calculer l'inverse d'une matrice A :
   - Partant de A, on pose $B = I$ la matrice identité.

- Par la méthode de Gauss et des opérations élémentaires on transforme A en la matrice I.
- On applique exactement les mêmes transformations à la matrice B.
- Lorsque A s'est transformée en I alors B s'est transformée en A^{-1}.

Modifier légèrement vos deux fonctions (issues des questions 2. et 3.) afin de calculer l'inverse d'une matrice.

Voici une matrice A, sa forme échelonnée, sa forme échelonnée réduite (l'identité), et son inverse :

$$A=\begin{pmatrix}1&2&3\\-1&0&1\\0&1&0\end{pmatrix}\quad\begin{pmatrix}1&2&3\\0&2&4\\0&0&-2\end{pmatrix}\quad\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\quad A^{-1}=\begin{pmatrix}\frac{1}{4}&-\frac{3}{4}&-\frac{1}{2}\\0&0&1\\\frac{1}{4}&\frac{1}{4}&-\frac{1}{2}\end{pmatrix}$$

1. Voici la fonction pour la deuxième opération : $L_i \leftarrow L_i + cL_j$.

Code 38 (*gauss.sage (1)*).
```
def op2(A,i,j,c):
    A[i,:] = A[i,:] + c*A[j,:]
    return A
```

`A[i,:]` correspond à la ligne `i` (les deux points signifiant de prendre tous les indices des colonnes). Lorsque l'on execute cette fonction sur une matrice A cela modifie la matrice (voir la remarque plus bas). Pour éviter cela on commence par faire une copie de A par `AA = copy(A)`, puis on exécute la fonction sur cette copie, avec par exemple `op2(AA,0,2,-1)` pour l'opération $L_0 \leftarrow L_0 - L_2$.

Les autres opérations `op1(A,i,c)`, `op3(A,i,j)` se définissent de manière analogue.

2. Il s'agit de traduire la première partie de la méthode de Gauss. Pour chaque colonne p en partant de la plus à gauche :
   (a) on cherche un coefficient non nul dans cette colonne ;
   (b) on place ce coefficient en position de pivot (p,p), en permutant deux lignes par la troisième opération élémentaire ;
   (c) on annule, un par un, les coefficients qui sont sous ce pivot, par des opérations élémentaires : $L_i \leftarrow L_i + cL_p$ où $c = -\frac{a_{i,p}}{a_{p,p}}$.

Code 39 (*gauss.sage (2)*).
```
def echelonne(A):
    n = A.ncols()  # taille de la matrice
    for p in range(n):  # pivot en A[p,p]
        # a. On cherche un coeff non nul
        i = p
        while i < n and A[i,p] == 0:
            i = i+1
        if i >= n :
            return matrix(n,n) # stoppe ici car pas inversible
        # b. On place la ligne avec coeff non nul en position de pivot
        A = op3(A,p,i)
        # c. On soustrait la ligne du pivot aux lignes en dessous
        for i in range(p+1,n):
            c = -A[i,p]/A[p,p]
            A = op2(A,i,p,c)
    return A
```

3. Pour passer à une forme réduite, on parcourt les colonnes en partant de la droite :
   (a) on commence par multiplier la ligne du pivot pour avoir un pivot valant 1 ;
   (b) on annule, un par un, les coefficients qui sont au-dessus de ce pivot.

Code 40 (*gauss.sage (3)*).
```
def reduite(A):
    n = A.ncols()
    for p in range(n-1,-1,-1):
        # a. On divise pour avoir un pivot valant 1
        c = 1/A[p,p]
        A = op1(A,p,c)
        # b. On élimine les coefficients au-dessus du pivot
        for i in range(p-1,-1,-1):
            c = -A[i,p]
            A = op2(A,i,p,c)
    return A
```

4. Pour calculer l'inverse, on définit deux nouvelles fonctions `echelonne_bis(A,B)` et `reduite_bis(A,B)` qui renvoient chacune, deux matrices modifiées de A et B. À chaque fois que l'on effectue une opération sur A, on effectue la même opération sur B. Par exemple, la dernière boucle de `echelonne_bis` contient la ligne `A = op2(A,i,p,c)` et la ligne `B = op2(B,i,p,c)`. C'est bien le même coefficient $c = -\frac{a_{i,p}}{a_{p,p}}$ pour les deux opérations. On obtient alors l'inverse comme ceci :

Code 41 (*gauss.sage (4)*).
```
def inverse_matrice(A):
    n = A.ncols()
    B = identity_matrix(QQ,n)
    A,B = echelonne_bis(A,B)
    A,B = reduite_bis(A,B)
    return B
```

Remarque.
Si on pose `A = matrix( [[1,2],[3,4]] )`, puis `B = A`, puis que l'on modifie la matrice A par `A[0,0]= 7`. Alors la matrice `B` est aussi modifée !
Que se passe-t-il ?

- `A` ne contient pas les coefficients de la matrice, mais l'adresse où sont stockés ces coefficients (c'est plus léger de manipuler l'adresse d'une matrice que toute la matrice).
  Comme `A` et `B` pointent vers la même zone qui a été modifiée, les deux matrices `A` et `B` sont modifiées.
- Pour pouvoir définir une copie de `A`, avec une nouvelle adresse mais les mêmes coefficients, on écrit `B = copy(A)`. Les modifications sur `A` ne changeront plus `B`.

5.3. Application linéaire, image, noyau

Travaux pratiques 19.
Soit $f : \mathbb{R}^3 \to \mathbb{R}^4$ l'application linéaire définie par

$$f : \begin{pmatrix} x \\ y \\ z \end{pmatrix} \longmapsto \begin{pmatrix} x+2z \\ 5x+2y+2z \\ 2x+y \\ x+y-2z \end{pmatrix}.$$

1. Expliciter la matrice A de f (dans les bases canoniques). Comment s'écrit alors l'application f ?
2. Calculer une base du noyau de f. L'application linéaire f est-elle injective ?
3. Calculer une base de l'image de f. L'application linéaire f est-elle surjective ?

1. La matrice associée est
$$A = \begin{pmatrix} 1 & 0 & 2 \\ 5 & 2 & 2 \\ 2 & 1 & 0 \\ 1 & 1 & -2 \end{pmatrix}.$$
L'application linéaire f étant alors définie par $X \mapsto Y = AX$. Voici l'implémentation avec un exemple de calcul.

Code 42 (*matlin.sage (1)*).
```
K = QQ
A = matrix(K,4,3,[[1,0,2],[5,2,2],[2,1,0],[1,1,-2]])
X = vector(K,[2,3,4])
Y = A*X
```

2. et 3. Le noyau s'obtient par la commande `right_kernel()` et l'image par la commande `column_space()` car l'image est exactement le sous-espace vectoriel engendré par les vecteurs colonnes de la matrice.

Code 43 (*matlin.sage (2)*).
```
Ker = A.right_kernel()
Ker.basis()
Im = A.column_space()
Im.basis()
```

- Le noyau est un espace vectoriel de dimension 1 dans $\mathbb{R}^3$ engendré par $\begin{pmatrix} 2 \\ -4 \\ -1 \end{pmatrix}$. Le noyau n'étant pas trivial, l'application f n'est pas injective.
- L'image est un espace vectoriel de dimension 2 dans $\mathbb{R}^4$, dont une base est :
$$\left(\begin{pmatrix} 1 \\ 1 \\ 0 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ 2 \\ 1 \\ 1 \end{pmatrix} \right).$$
L'image n'étant pas tout $\mathbb{R}^4$, l'application f n'est pas non plus surjective.
- On peut tester si un vecteur donné est dans un sous-espace. Si on pose par exemple `X = vector(K,[1,5,2,1])` alors le test « `X in Im` » renvoie vrai. Ce vecteur est donc bien un élément de l'image de f.

Attention ! Il ne faut pas utiliser directement les méthodes `kernel()` et `image()` car Sage préfère travailler avec les vecteurs lignes et donc ces méthodes calculent le *noyau à gauche* (c'est-à-dire l'ensemble des X tels que $XA = 0$) et l'*image à gauche* (c'est-à-dire l'ensemble des $Y = XA$). Si vous souhaitez utiliser ces méthodes pour calculer le noyau et l'image avec notre convention habituelle (c'est-à-dire à droite) il faut le faire sur la transposée de A :

Code 44 (*matlin.sage (3)*).
```
AA = A.transpose()
Ker = AA.kernel()
Ker.basis()
Im = AA.image()
Im.basis()
```

5.4. Méthodes des moindres carrés

Si on veut résoudre un système linéaire avec plus d'équations que d'inconnues alors il n'y a pas de solution en général. On aimerait pourtant parfois trouver ce qui s'approche le plus d'une solution. Formalisons ceci : soit $A \in M_{n,p}(\mathbb{R})$ avec $n \geqslant p$ une matrice à coefficients réels, X un vecteur inconnu de taille p et B un vecteur de taille n. Il n'y a en général pas de solution X au système $AX = B$, mais on aimerait que $AX - B$ soit le plus proche du vecteur nul. Ainsi, une

solution des moindres carrés est un vecteur X tel que $\|AX-B\|$ soit minimale, où la norme considérée est la norme euclidienne. Cette solution des moindres carrées est donnée par la formule :

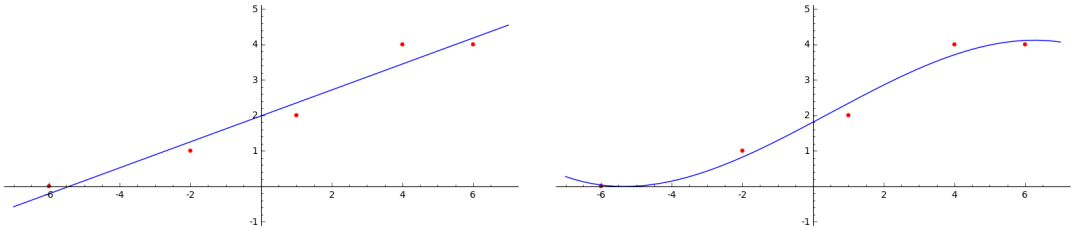
$$X = (A^TA)^{-1}A^TB \qquad (\dagger)$$

(si $n \geqslant p$ et A est de rang maximal p, ce qu'on suppose, alors on peut montrer que la matrice A^TA est inversible).

Travaux pratiques 20.

1. **Régression linéaire.** On considère les points $\{(-6,0),(-2,1),(1,2),(4,4),(6,4)\}$. Trouver la droite qui approche au mieux ces points (au sens des moindres carrés).
   Indications. On pose $f(x) = a + bx$. On aimerait trouver $a, b \in \mathbb{R}$ tels que $f(x_i) = y_i$ pour tous les points (x_i, y_i) donnés. Transformer le problème en un problème des moindres carrés : quel est le vecteur inconnu X ? quelle est la matrice A? quel est le second membre B ?
2. **Interpolation polynomiale.** Pour ces mêmes points, quel polynôme de degré 3 approche au mieux ces points (au sens des moindres carrés).
   Indication. Cette fois $f(x) = a + bx + cx^2 + dx^3$.

Voici la droite demandée (à gauche) ainsi que le polynôme de degré 3 (à droite).



1. Bien sûr les points ne sont pas alignés, donc il n'existe pas de droite passant par tous ces points. On cherche une droite d'équation $y = a + bx$ qui minimise (le carré de) la distance entre les points et la droite. Posons $f(x) = a + bx$.
   Alors pour nos n points (x_i, y_i) donnés, on voudrait $f(x_i) = y_i$.
   $$\begin{cases} f(x_1) = y_1 \\ f(x_2) = y_2 \\ \vdots \quad \vdots \\ f(x_n) = y_n \end{cases} \iff \begin{cases} a + bx_1 = y_1 \\ a + bx_2 = y_2 \\ \vdots \quad \vdots \\ a + bx_n = y_n \end{cases} \iff \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \iff AX = B.$$
   On résout (de façon non exacte) notre problème $AX = B$, par la formule des moindres carrés $X = (A^TA)^{-1}A^TB$. Comme $X = \binom{a}{b}$, on obtient l'équation $y = a + bxb$ de la droite des moindres carrés (voir la figure ci-dessus).
   La fonction `moindres_carres` résout le problème général des moindres carrés. Il reste ensuite à définir la matrice A et le vecteur B en fonction de notre liste de points afin de trouver X. Le code est le suivant :

Code 45 (*moindres_carres.sage (1)*).
```
def moindres_carres(A,B):
    return (A.transpose()*A)^-1 * A.transpose() * B

points = [(-6,0),(-2,1),(1,2),(4,4),(6,4)]

A = matrix([ [1,p[0]] for p in points ])
B = vector(p[1] for p in points)
X = moindres_carres(A,B)
```

2. L'idée est la même, mais cette fois les inconnues sont les coefficients de la fonction $f(x) = a + bx + cx^2 + dx^3$:
   $$\begin{cases} f(x_1) = y_1 \\ f(x_2) = y_2 \\ \vdots \quad \vdots \\ f(x_n) = y_n \end{cases} \iff \begin{cases} a + bx_1 + cx_1^2 + dx_1^3 = y_1 \\ a + bx_2 + cx_2^2 + dx_2^3 = y_2 \\ \vdots \quad \vdots \\ a + bx_n + cx_n^2 + dx_n^3 = y_n \end{cases}$$

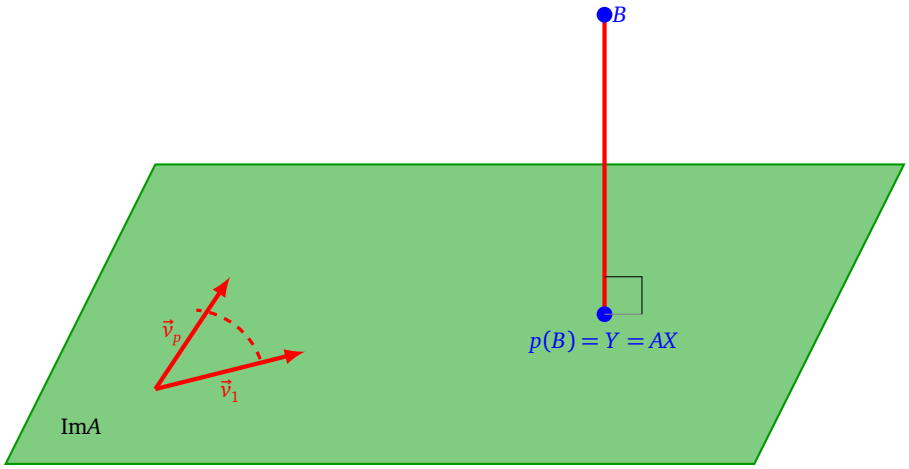
$$\iff \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \iff AX = B$$

Voici le code pour les mêmes points et une interpolation polynomiale de degré d :

Code 46 (*moindres_carres.sage (2)*).

```
d = 3   # degré
A = matrix([ [p[0]^k for k in range(d+1)]  for p in points ])
B = vector(p[1] for p in points)
X = moindres_carres(A,B)
```

Pour conclure, voici la preuve de la formule (†) des moindres carrés. Caractérisons géométriquement la condition « $\|AX - B\|$ minimale ». On note $\mathrm{Im}A$ l'image de A, c'est-à-dire le sous-espace vectoriel engendré par les vecteurs colonnes de la matrice A. Notons $p : \mathbb{R}^n \to \mathbb{R}^n$ la projection orthogonale sur $\mathrm{Im}A$. Enfin, notons $Y = p(B)$ le projeté orthogonal de B sur l'image de A.



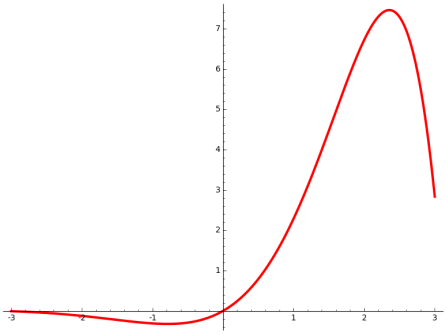
Comme Y est dans l'image de A alors il existe X tel que $AX = Y$ et X est notre «solution» cherchée. En effet, c'est l'une des caractérisations du projeté orthogonal : $Y = p(B)$ est le point de $\mathrm{Im}A$ qui minimise la distance entre B et un point de $\mathrm{Im}A$.

Que Y soit le projeté orthogonal de B sur $\mathrm{Im}A$ signifie que le vecteur $B - Y$ est orthogonal à tout vecteur de $\mathrm{Im}A$:

$$\begin{aligned}
& \langle AV \mid B - Y \rangle = 0 \quad \forall V \in \mathbb{R}^p \\
\iff\ & \langle AV \mid B - AX \rangle = 0 \quad \forall V \in \mathbb{R}^p \quad \text{pour } Y = AX \\
\iff\ & (AV)^T \cdot (B - AX) = 0 \quad \forall V \in \mathbb{R}^p \quad \text{car } \langle u \mid v \rangle = u^T \cdot v \\
\iff\ & V^T A^T (B - AX) = 0 \quad \forall V \in \mathbb{R}^p \\
\iff\ & A^T (B - AX) = 0 \\
\iff\ & A^T AX = A^T B \\
\iff\ & X = (A^T A)^{-1} A^T B
\end{aligned}$$

6. Courbes et surfaces

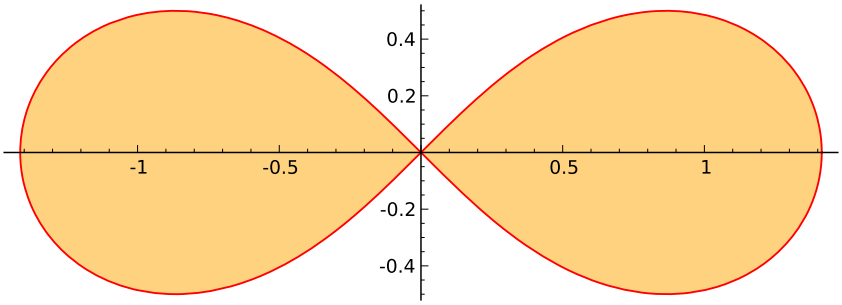
La visualisation est une étape importante dans l'élaboration des preuves en mathématiques. L'avènement des logiciels de calcul formel possédant une interface graphique évoluée a rendu cette phase attrayante. Nous allons explorer les possibilités graphiques offertes par Sage. Nous avons déjà vu comment tracer les graphes de fonctions avec la commande `plot` (voir « Premiers pas avec Sage »), par exemple `plot(sin(x)*exp(x),(x,-3,3))` trace le graphe de la fonction $f(x) = \sin(x)\exp(x)$ sur l'intervalle $[-3, 3]$.



En plus des graphes de fonctions Sage sait tracer des courbes et des surfaces par d'autres méthodes.

6.1. Courbes paramétrées

La commande `parametric_plot((f(t), g(t)), (t, a, b))` permet de tracer la courbe paramétrée plane donnée par les points de coordonnées $(f(t), g(t))$ lorsque le paramètre t varie dans l'intervalle $[a, b]$. Commençons par la lemniscate de Bernoulli :

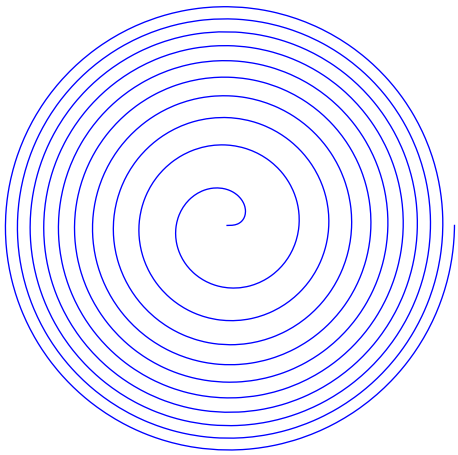


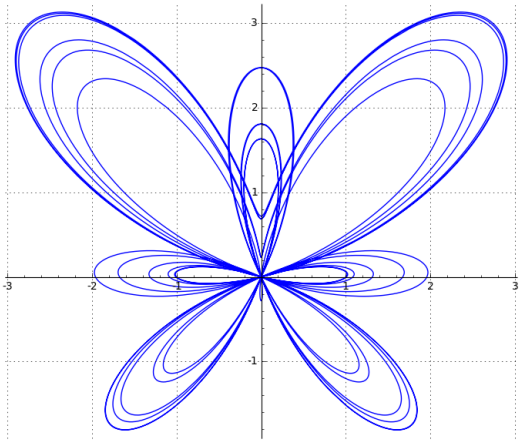
Code 47 (*lemniscate-bernoulli.sage*)**.**

```
var('t')
x = sqrt(2)*cos(t)/(1+sin(t)^2)
y = sqrt(2)*cos(t)*sin(t)/(1+sin(t)^2)
G = parametric_plot((x, y), (t, 0, 2*pi))
G.show()
```

Travaux pratiques 21.

1. Tracer la spirale de Fermat d'équation
$$\begin{cases} x(t) = \sqrt{t}\cos t \\ y(t) = \sqrt{t}\sin t \end{cases} \qquad t \in \mathbb{R}_+.$$
2. Tracer la courbe du papillon
$$\begin{cases} x(t) = \sin(t)\left(\exp(\cos(t)) - 2\cos(4t) - \sin^5\left(\frac{t}{12}\right)\right) \\ y(t) = \cos(t)\left(\exp(\cos(t)) - 2\cos(4t) - \sin^5\left(\frac{t}{12}\right)\right) \end{cases} \qquad t \in \mathbb{R}.$$



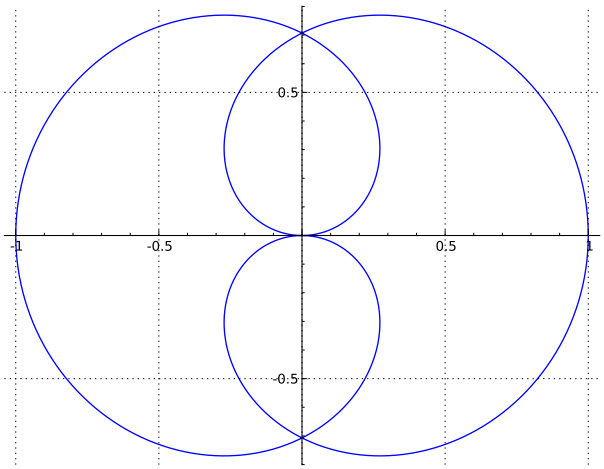


Les commandes de tracé possèdent de nombreuses options, qu'on pourra découvrir grâce à la commande `help(parametric_plot)`. Par exemple :

- Imposer un repère orthonormé : `aspect_ratio = 1`
- Nombre de points pour le tracé : `plot_points = 500`
- Ne pas afficher les axes : `axes = False`
- Afficher une grille : `gridlines = True`
- Remplir l'intérieur : `fill = True`
- Couleur du trait : `color = 'red'`, couleur du remplissage : `fillcolor = 'orange'`

6.2. Courbes en coordonnées polaires

Les courbes en coordonnées polaires sont tracées grâce à la commande `polar_plot(r(t),(t,a,b))`, qui produira l'ensemble des points de coordonnées polaires $[r(t) : t]$ pour t variant dans l'intervalle $[a, b]$. Voici le folium de Dürer d'équation $r(t) = \sin \frac{t}{2}$, $t \in \mathbb{R}$.



Code 48 (*durer-folium.sage*)**.**

```
var('t')
G = polar_plot(sin(t/2),(t,0,4*pi))
G.show()
```

Travaux pratiques 22.

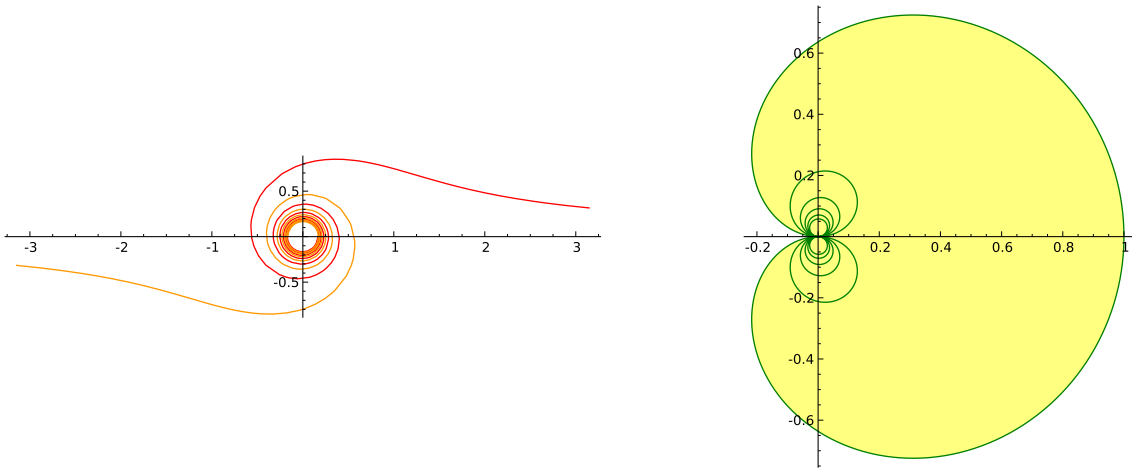
1. Tracer la courbe du Lituus d'équation polaire

$$r(t)^2 = \frac{1}{t} \qquad t \in \mathbb{R}^*.$$

2. Tracer la cochléoïde d'équation polaire

$$r(t) = \frac{\sin t}{t} \qquad t \in \mathbb{R}^*.$$

Indications : on peut définir deux graphes `G1`, `G2` pour chacun des intervalles de définition. On superpose les graphes avec `G = G1 + G2`, puis on les affiche avec `G.show()`.



6.3. Courbes définies par une équation

Une courbe peut avoir plusieurs types de représentations, comme par exemple un cercle d'équation paramétrique $(\cos t, \sin t)$ ou d'équation implicite $x^2 + y^2 = 1$. La commande

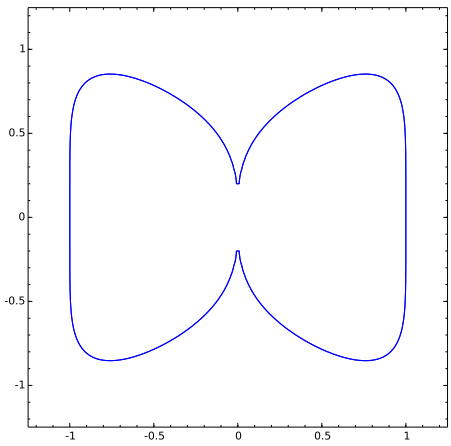
```
implicit_plot(f(x, y), (x, a, b), (y, c, d))
```

permet de tracer l'ensemble des couples (x, y) dans $[a, b] \times [c, d]$ qui sont solutions de l'équation $f(x, y) = 0$. Voici une autre courbe de papillon, cette fois algébrique.

Code 49 (*butterfly-curve-algebraic.sage*).

```
f(x,y) = x^6 + y^6 - x^2
G = implicit_plot(f, (x, -1.2, 1.2), (y, -1.2, 1.2))
G.show()
G.save('butterfly_curve_algebraic.png')
```

Noter comment il est possible de sauvegarder une image du tracé dans un fichier externe.



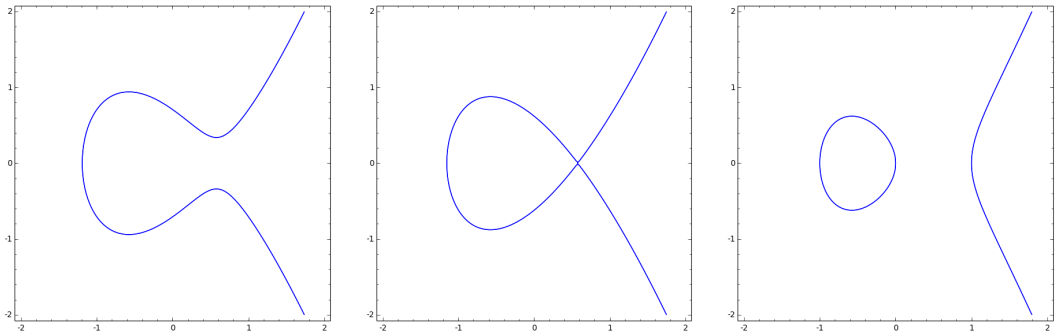
Travaux pratiques 23.

1. Définir une fonction qui renvoie la courbe définie par

$$y^2 - x^3 + x + c = 0$$

en fonction du paramètre $c \in \mathbb{R}$.

2. À l'aide de la commande `animate`, réaliser une animation qui affiche l'évolution de la courbe pour $c \in [-1,1]$.



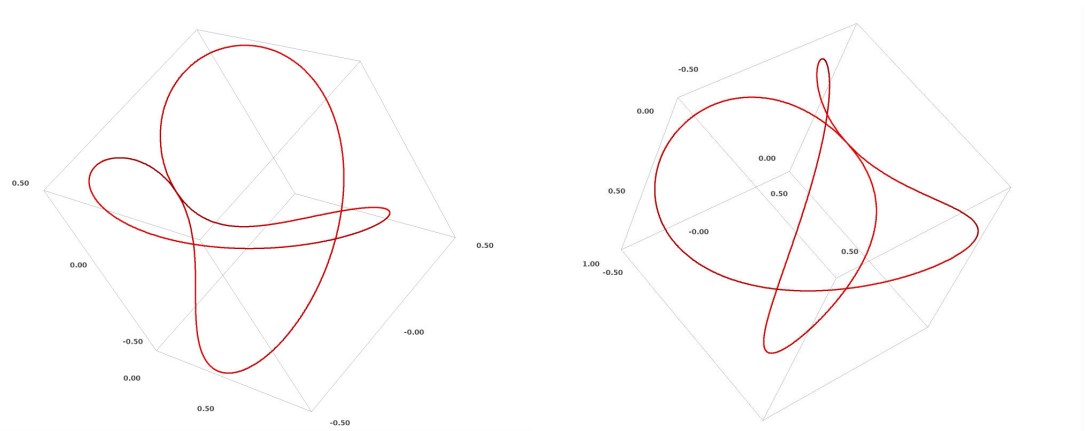
6.4. Courbes de l'espace

La commande `parametric_plot((x, y, z), (t, a, b))`, (ou bien `parametric_plot3d()`) analogue à celle de la dimension 2, trace la courbe paramétrée de l'espace donnée par les points de coordonnées $\big(x(t), y(t), z(t)\big)$ lorsque le paramètre t varie dans l'intervalle $[a,b]$.

Travaux pratiques 24.
Tracer la courbe d'Archytas d'équation paramétrique :

$$\begin{cases} x(t) = \cos^2 t \\ y(t) = \cos t \sin t \\ z(t) = \pm\sqrt{\cos t(1-\cos t)} \end{cases} \qquad t \in \Big[-\tfrac{\pi}{2}, +\tfrac{\pi}{2}\Big].$$

Vous obtiendrez une figure qu'il est possible d'orienter dynamiquement avec la souris. Voici quelques-unes de ces vues.



6.5. Surfaces

Découvrez, à l'aide de l'énoncé suivant, les différentes méthodes pour tracer des surfaces avec `Sage`.

Travaux pratiques 25.

1. Tracer le graphe de la fonction $f(x,y) = x^2y^2 - (x^2+y^2)^3$ définie sur $(x,y) \in [-1,1] \times [-1,1]$. Utiliser la fonction `plot3d`.
2. Tracer la surface d'Enneper définie par l'équation

$$\left(\frac{y^2-x^2}{2z} + \frac{2}{9}z^2 + \frac{2}{3}\right)^3 - 6\left(\frac{y^2-x^2}{4z} - \frac{1}{4}\big(x^2+y^2+\frac{8}{9}z^2\big) + \frac{2}{9}\right)^2 = 0.$$

Utiliser la fonction `implicit_plot3d`.

3. Tracer la nappe paramétrée définie par :

$$\begin{cases} x(s,t) = t^2 \cos s \\ y(s,t) = s^2 \sin t \\ z(s,t) = \cos t + \sin t \end{cases} \qquad s \in [0,1], \quad t \in [-\pi, +\pi].$$

   Utiliser la fonction `parametric_plot3d`.

4. Tracer la surface paramétrée définie en coordonnées cylindriques par :

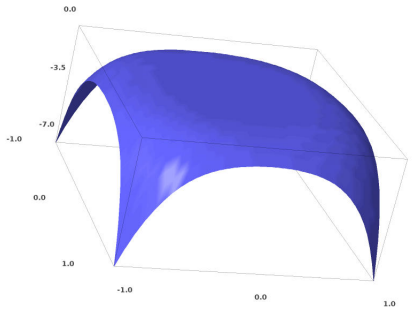
$$r(\theta, z) = z^3 \cos\theta \qquad \theta \in [0, 2\pi], \quad z \in [0, 2].$$

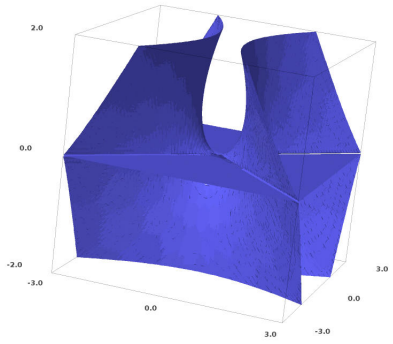
   Utiliser la fonction `cylindrical_plot3d`.

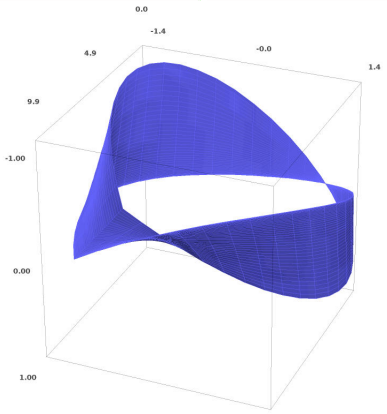
5. Tracer la surface paramétrée définie en coordonnées sphériques par

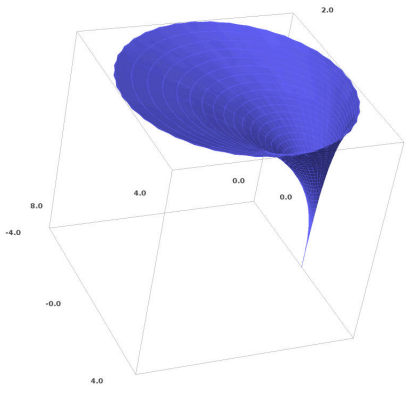
$$r(\theta, \phi) = \theta \sin(2\phi) \qquad \theta \in [-1, 2\pi], \quad \phi \in [0, \pi].$$

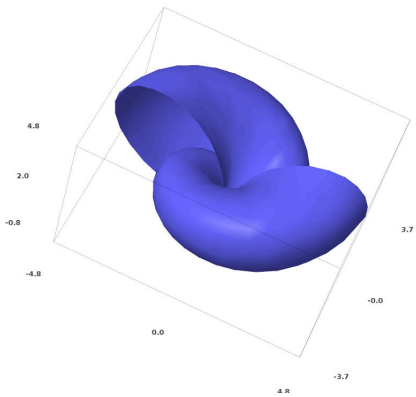
   Utiliser la fonction `spherical_plot3d`.









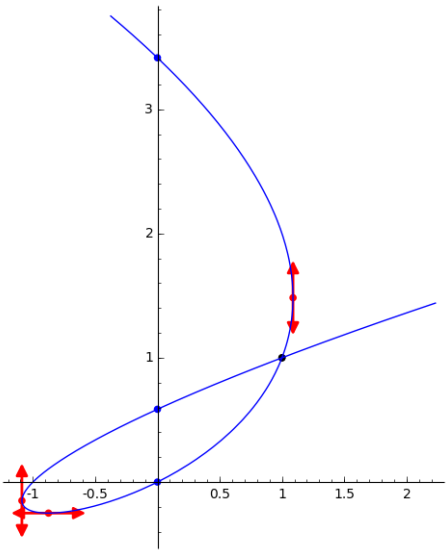


6.6. Étude d'une courbe paramétrée

Travaux pratiques 26.
Étudier en détail la courbe paramétrée du plan définie par

$$\begin{cases} x(t) = t^3 - 2t \\ y(t) = t^2 - t \end{cases} \qquad t \in \mathbb{R}.$$

1. Tracer la courbe.
2. Trouver les points d'intersection de la courbe avec l'axe des ordonnées.
3. Trouver les points en lesquels la tangente est verticale, puis horizontale.
4. Trouver les coordonnées des points doubles.



1. La courbe a l'allure d'une boucle.
2. On définit `x = t^3-2*t` et `y = t^2-t`. On obtient les valeurs de t correspondant aux points d'intersection de la courbe avec l'axe $(x = 0)$ en résolvant l'équation $x(t) = 0$: `solve(x==0,t)`. On obtient trois solutions $t \in \{-\sqrt{2}, 0, +\sqrt{2}\}$ correspondant aux trois points $\{(0, 2+\sqrt{2}), (0,0), (0, 2-\sqrt{2})\}$
3. On définit les fonctions dérivées $x'(t)$ et $y'(t)$ par `xx = diff(x,t)` et `yy = diff(y,t)`, les valeurs de t pour lesquelles la tangente à la courbe est verticale s'obtiennent en résolvant l'équation $x'(t) = 0$, ce qui s'écrit `solve(xx==0,t)`.
4. Trouver les points doubles est souvent un calcul délicat (même pour un logiciel de calcul formel). Il s'agit ici de résoudre le système d'équations polynomiales :

$$\begin{cases} x(s) = x(t) \\ y(s) = y(t) \end{cases} \qquad s, t \in \mathbb{R}.$$

Mais il faut exclure la solution évidente $t = s$. Algébriquement cela signifie que $(s-t)$ divise les polynômes de deux variables $x(s)-x(t)$ et $y(s)-y(t)$. Autrement dit, il s'agit de résoudre :

$$\begin{cases} \dfrac{x(s)-x(t)}{s-t} = 0 \\ \dfrac{y(s)-y(t)}{s-t} = 0 \end{cases} \qquad s, t \in \mathbb{R}.$$

Le code suivant fournit la solution $(s,t) = \left(\frac{1-\sqrt{5}}{2}, \frac{1+\sqrt{5}}{2}\right)$, il y a un unique point double ayant pour coordonnées $(1,1)$.

Code 50 (*courbe.sage*).

```
var('s,t')
x = t^3-2*t
y = t^2-t
```

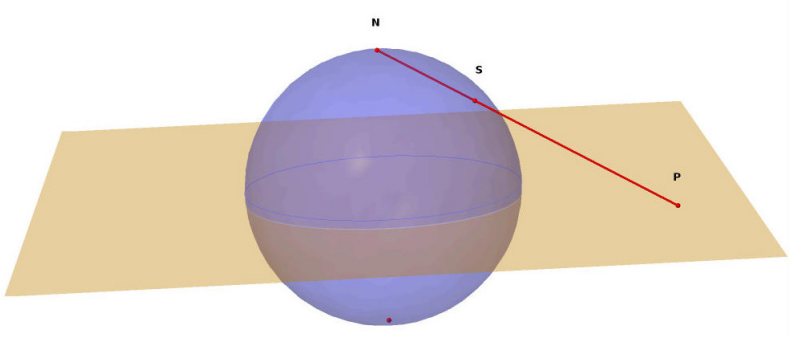
```
eqx = x.subs(t=s) - x                    # Equation x(s)-x(t)
eqxs = (eqx/(s-t)).simplify_rational()  # (x(s)-x(t))/(s-t)
eqy = y.subs(t=s) - y                    # Equation y(s)-y(t)
eqys = (eqy/(s-t)).simplify_rational()  # (y(s)-y(t))/(s-t)
points_double = solve([eqxs==0,eqys==0], s,t)   # Solutions
```

6.7. La projection stéréographique

Travaux pratiques 27.
Soit $\mathscr{S}$ la sphère centrée à l'origine de $\mathbb{R}^3$ et de rayon 1. On note $N=(0,0,1)$ le pôle Nord. Soit $\mathscr{P}$ le plan équatorial d'équation $(z=0)$. La ***projection stéréographique*** est l'application $\Phi : \mathscr{S}\setminus\{N\}\to\mathscr{P}$ qui à un point S de la sphère associe le point $P=(NS)\cap\mathscr{P}$ défini par l'intersection de la droite (NS) avec le plan $\mathscr{P}$.

1. En écrivant la relation $\overrightarrow{NP}=k\overrightarrow{NS}$ et sachant que $P\in\mathscr{P}$, trouver k et en déduire P.
   Vérifier ainsi que l'application Φ est définie par :
   $$\Phi : \mathscr{S}\setminus\{N\}\to\mathscr{P} \qquad \Phi(x,y,z)=\left(\frac{x}{1-z},\frac{y}{1-z}\right)$$
   Définir la fonction correspondante avec Sage.
2. Définir et vérifier que l'application inverse est :
   $$\Psi : \mathscr{P}\to\mathscr{S}\setminus\{N\} \qquad \Psi(X,Y)=\left(\frac{2X}{\rho},\frac{2Y}{\rho},1-\frac{2}{\rho}\right) \quad \text{avec } \rho=1+X^2+Y^2$$
3. Écrire une fonction qui dessine une courbe paramétrée $\mathscr{C}' : \big(x(t),y(t)\big)$, $t\in[a,b]$ du plan $\mathscr{P}$, et qui dessine aussi l'image inverse de la courbe sur la sphère, $\mathscr{C}=\Psi(\mathscr{C}')$.
4. Vérifier graphiquement deux propriétés fondamentales de la projection stéréographique :
   (a) « La projection stéréographique envoie les cercles de la sphère sur des cercles ou des droites du plan. »
   (b) « La projection stéréographique préserve les angles. » En particulier, deux courbes qui se coupent à angle droit, s'envoient sur deux courbes qui se coupent à angle droit.
5. Soit $\mathscr{C}'$ la spirale logarithmique d'équation $\big(e^t\cos t, e^t\sin t\big)$, $t\in\mathbb{R}$. Tracer la loxodromie de la sphère qui est $\mathscr{C}=\Psi(\mathscr{C}')$.
6. Soit la courbe $\mathscr{C}\subset\mathscr{S}$ définie par $\frac{1}{13t^2-6t+2}\big(4t,-6t+2,13t^2-6t\big)$. Montrer que son image $\mathscr{C}'=\Phi(\mathscr{C})\subset\mathscr{P}$ est une droite, dont vous déterminerez une équation.

1.



Le vecteur $\overrightarrow{NP}$ est colinéaire au vecteur $\overrightarrow{NS}$, donc il existe $k\in\mathbb{R}$ tel que $\overrightarrow{NP}=k\overrightarrow{NS}$. Mais en plus on veut que $P\in\mathscr{P}$, c'est-à-dire $z_P=0$. Ce qui permet de trouver k et d'en déduire $P=N+k\overrightarrow{NS}$. On laisse Sage faire les calculs :

Code 51 (*stereographic.sage (1)*).
```
var('x,y,z')              # Variables de l'espace
var('X,Y')                # Variables du plan
```

```
var('k')
N = vector([0,0,1])    # Les points N, S, P comme des vecteurs
S = vector([x,y,z])
P = vector([X,Y,0])
V = (P-N)-k*(S-N)      # Le vecteur NP - k NS
eq = V[2]              # On veut la troisième coordonnée nulle
sol = solve(eq==0,k)   # Equation en k pour cette condition
k = sol[0].rhs()       # Solution k
P = N + k*(S-N)        # Le point P
```

On obtient $k = \frac{1}{1-z}$, et alors si $S = (x,y,z)$, $P = \left(\frac{x}{1-z}, \frac{y}{1-z}, 0\right)$. En identifiant les points du plan à $\mathbb{R}^2$, on a ainsi prouvé $\Phi(x,y,z) = \left(\frac{x}{1-z}, \frac{y}{1-z}\right)$. D'où la définition de la fonction Φ :

Code 52 (*stereographic.sage (2)*)**.**
```
def stereo(x,y,z):
    X = x/(1-z)
    Y = y/(1-z)
    return X,Y
```

2. Voici la fonction Ψ :

Code 53 (*stereographic.sage (3)*)**.**
```
def inverse_stereo(X,Y):
    r = 1+X^2+Y^2
    x = 2*X/r
    y = 2*Y/r
    z = 1-2/r
    return x,y,z
```

On profite du fait que l'on nous donne le candidat, pour se contenter de vérifier que c'est bien la bijection réciproque, c'est-à-dire $\Phi\big(\Psi(X,Y)\big) = (X,Y)$ pour tout $(X,Y) \in \mathbb{R}^2$. La composition s'écrit

Code 54 (*stereographic.sage (4)*)**.**
```
newx,newy,newz = inverse_stereo(X,Y)
newX,newY = stereo(newx,newy,newz)
```

et comme on obtient `newX = X` et `newY = Y`, cela prouve le résultat.
Il est possible de composer les fonctions de plusieurs variables, mais il faut rajouter une « * » devant la fonction à composer :

```
stereo(*inverse_stereo(X,Y)))
```

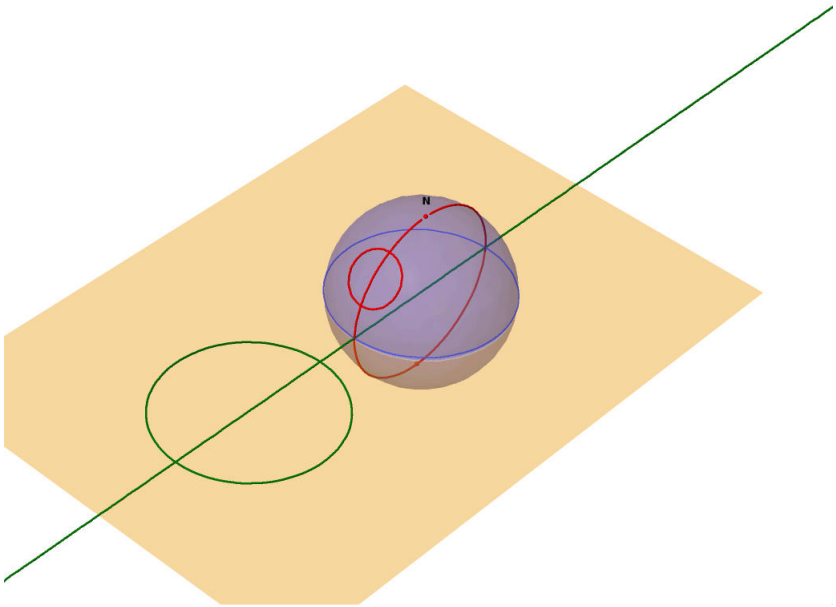
cela renvoie `X,Y` ce qui prouve bien $\Phi\big(\Psi(X,Y)\big) = (X,Y)$. (L'opérateur « * » devant une liste permet de la décompacter, par exemple `*(2,5,7)` s'interprète comme `2,5,7` qui peut alors être passé en argument à une fonction.)
Pour prouver $\Psi\big(\Phi(x,y,z)\big) = (x,y,z)$, il faudrait se souvenir que $(x,y,z) \in \mathscr{S}$, donc $x^2+y^2+z^2 = 1$.

3. Voici comment tracer les courbes. La courbe du plan est tracée comme une ligne polygonale, avec un pas assez petit. Ensuite on calcule l'image par Ψ de chacun de ces points.

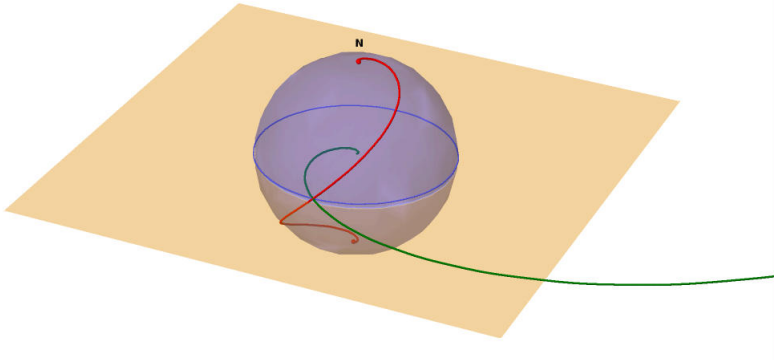
Code 55 (*stereographic.sage (5)*)**.**
```
def courbes(X,Y,a,b):
    XYtab = [ [X.subs(t=myt),Y.subs(t=myt),0] for myt in srange(a,b,0.1) ]
    xyztab = [ inverse_stereo(coord[0],coord[1]) for coord in XYtab ]
    G = sphere((0,0,0),1)
    G = G + line(XYtab)
```

```
    G = G + line(xyztab)
    return G
```

Par exemple `G = courbes(t^3,t^2,-2,2)`, trace la courbe du plan d'équation paramétrique $\left(t^3, t^2\right)$, $t \in [-2,2]$, ainsi que son image par la projection stéréographique inverse.
Voici le cas d'un cercle et d'une droite du plan qui se coupent à angle droit, de même que les cercles de la sphères.



4. Voici une loxodromie de la sphère, courbe utilisée par les navigateurs, car elle correspond à une navigation à cap constant.



5. Pour x, y, z définis par la paramétrisation, on calcule l'image par la projection stéréographique avec `X,Y = stereo(x,y,z)`, on n'oublie pas de simplifier à l'aide de `full_simplify()`. Cela donne une paramétrisation de l'image, $x(t) = 2t$, $y(t) = -3t + 1$, qui est bien l'équation d'une droite.

7. Calculs d'intégrales

7.1. Sage comme une super-calculatrice

Travaux pratiques 28.
Calculer à la main les primitives des fonctions suivantes. Vérifier vos résultats à l'ordinateur.

1. $f_1(x) = x^4 + \frac{1}{x^2} + \exp(x) + \cos(x)$
2. $f_2(x) = x\sin(x^2)$
3. $f_3(x) = \frac{\alpha}{\sqrt{1-x^2}} + \frac{\beta}{1+x^2} + \frac{\gamma}{1+x}$
4. $f_4(x) = \frac{x^4}{x^2-1}$
5. $f_5(x) = x^n \ln x$ pour tout $n \geqslant 0$

Pour une fonction f, par exemple `f(x) = x*sin(x^2)` donnée, la commande `integral(f(x),x)` renvoie une primitive de f. Le résultat obtenu ici est `-1/2*cos(x^2)`.
Quelques remarques :

- La machine renvoie une primitive. Pour avoir l'ensemble des primitives, il faut bien entendu ajouter une constante.
- La fonction `log` est la fonction logarithme népérien usuel, habituellement notée ln.
- `integral(1/x,x)` renvoie `log(x)` alors que $\int \frac{dx}{x} = \ln|x|$. Sage omet les valeurs absolues, ce qui ne l'empêche pas de faire des calculs exacts même pour les valeurs négatives. (En fait Sage travaille avec le logarithme complexe.).
- Souvent, pour intégrer des fractions rationnelles, on commence par écrire la décomposition en éléments simples. C'est possible avec la commande `partial_fraction`. Par exemple pour `f = x^4/(x^2-1)` la commande `f.partial_fraction(x)` renvoie la décomposition sur $\mathbb{Q}$: $f(x) = x^2 + 1 - \frac{1}{2}\frac{1}{x+1} + \frac{1}{2}\frac{1}{x-1}$.

Travaux pratiques 29.
Calculer à la main et à l'ordinateur les intégrales suivantes.

1. $I_1 = \displaystyle\int_1^3 x^2 + \sqrt{x} + \frac{1}{x} + \sqrt[3]{x}\,dx$
2. $I_2 = \displaystyle\int_0^{\frac{\pi}{2}} \sin x(1+\cos x)^4\,dx$
3. $I_3 = \displaystyle\int_0^{\frac{\pi}{6}} \sin^3 x\,dx$
4. $I_4 = \displaystyle\int_0^{\sqrt{3}} \frac{3x^3 - 2x^2 - 5x - 6}{(x+1)^2(x^2+1)}\,dx$

Par exemple `integral(sin(x)^3,x,0,pi/6)` renvoie $\frac{2}{3} - \frac{3\sqrt{3}}{8}$.

7.2. Intégration par parties

Travaux pratiques 30.
Pour chaque $n \geqslant 0$, nous allons déterminer une primitive de la fonction

$$f_n(x) = x^n \exp(x).$$

1. Sage connaît-il directement la formule ?
2. Calculer une primitive F_n de f_n pour les premières valeurs de n.
3. (a) Émettre une conjecture reliant F_n et F_{n-1}.
   (b) Prouver cette conjecture.
   (c) En déduire une fonction récursive qui calcule F_n.
4. (a) Émettre une conjecture pour une formule directe de F_n.
   (b) Prouver cette conjecture.

(c) En déduire une expression qui calcule F_n.

1. La commande `integral(x^n*exp(x),x)` renvoie le résultat `(-1)^n*gamma(n + 1, -x)`. Nous ne sommes pas tellement avancés ! Sage renvoie une *fonction spéciale* Γ que l'on ne connaît pas. En plus (à l'heure actuelle, pour la version 6.6), dériver la primitive avec Sage ne redonne pas la même expression que f_n.
2. Par contre, il n'y a aucun problème pour calculer les primitives F_n pour les premières valeurs de n. On obtient :
$$\begin{aligned} F_0(x) &= \exp(x) \\ F_1(x) &= (x-1)\exp(x) \\ F_2(x) &= (x^2-2x+2)\exp(x) \\ F_3(x) &= (x^3-3x^2+6x-6)\exp(x) \\ F_4(x) &= (x^4-4x^3+12x^2-24x+24)\exp(x) \\ F_5(x) &= (x^5-5x^4+20x^3-60x^2+120x-120)\exp(x) \end{aligned}$$
3. (a) Au vu des premiers termes, on conjecture $F_n(x) = x^n\exp(x) - nF_{n-1}(x)$. On vérifie facilement sur les premiers termes que cette conjecture est vraie.

   Sage ne sait pas (encore) vérifier formellement cette identité (avec n une variable formelle). La commande suivante échoue à trouver 0 :
   ```
   integral(x^n*exp(x),x) - x^n*exp(x) + n*integral(x^(n-1)*exp(x),x)
   ```
   (b) Nous prouverons la validité de cette formule en faisant une intégration par parties (on pose par exemple $u = x^n$, $v' = \exp(x)$ et donc $u' = nx^{n-1}$, $v = \exp(x)$) :
$$F_n(x) = \int x^n\exp(x)\,\mathrm{d}x = \big[x^n\exp(x)\big] - \int nx^{n-1}\exp(x)\,\mathrm{d}x = x^n\exp(x) - nF_{n-1}(x).$$
   (c) Voici l'algorithme récursif pour le calcul de F_n utilisant la relation de récurrence.

Code 56 (*integrales-ipp (1)*).
```
def FF(n):
    if n==0:
        return exp(x)
    else:
        return x^n*exp(x) - n*FF(n-1)
```

4. Avec un peu de réflexion, on conjecture la formule
$$F_n(x) = \exp x \sum_{k=0}^{n} (-1)^{n-k}\frac{n!}{k!}x^k$$
   où $\frac{n!}{k!} = n(n-1)(n-2)\cdots(k+1)$.
   Pour la preuve, on pose $G_n(x) = \exp(x)\sum_{k=0}^{n}(-1)^{n-k}\frac{n!}{k!}x^k$. On a
$$G_n(x) = \exp(x)\sum_{k=0}^{n-1}(-1)^{n-k}\frac{n!}{k!}x^k + \exp(x)x^n = -nG_{n-1}(x) + \exp(x)x^n.$$
   Ainsi G_n vérifie la même relation de récurrence que F_n. De plus $G_0(x) = \exp x = F_0(x)$. Donc pour tout n, $G_n = F_n$.
   On peut donc ainsi calculer directement notre intégrale par la formule :
   ```
   exp(x)*sum((-1)^(n-k)*x^k*factorial(n)/factorial(k),k,0,n)
   ```

7.3. Changement de variable

Soit f une fonction définie sur un intervalle I et $\varphi : J \to I$ une bijection de classe $\mathscr{C}^1$. La formule de changement de variable pour les primitives est :
$$\int f(x)\,\mathrm{d}x = \int f\big(\varphi(u)\big)\cdot\varphi'(u)\,\mathrm{d}u.$$
La formule de changement de variable pour les intégrales, pour tout $a, b \in I$, est :
$$\int_a^b f(x)\,\mathrm{d}x = \int_{\varphi^{-1}(a)}^{\varphi^{-1}(b)} f\big(\varphi(u)\big)\cdot\varphi'(u)\,\mathrm{d}u.$$

Travaux pratiques 31.

1. Calcul de la primitive $\int \sqrt{1-x^2}\,\mathrm{d}x$.
   (a) Poser $f(x)=\sqrt{1-x^2}$ et le changement de variable $x=\sin u$, c'est-à-dire on pose $\varphi(u)=\sin u$.
   (b) Calculer $f\big(\varphi(u)\big)$ et $\varphi'(u)$.
   (c) Calculer la primitive de $g(u)=f\big(\varphi(u)\big)\cdot\varphi'(u)$.
   (d) En déduire une primitive de $f(x)=\sqrt{1-x^2}$.
2. Calcul de la primitive $\displaystyle\int \frac{\mathrm{d}x}{1+\left(\frac{x+1}{x}\right)^{1/3}}$.
   (a) Poser le changement de variable défini par l'équation $u^3=\frac{x+1}{x}$.
   (b) En déduire le changement de variable $\varphi(u)$ et $\varphi^{-1}(x)$.
3. Écrire une fonction `integrale_chgtvar(f,eqn)` qui calcule une primitive de $f(x)$ par le changement de variable défini par l'équation `eqn` reliant u et x.
   En déduire $\displaystyle\int (\cos x+1)^n\cdot\sin x\,\mathrm{d}x$, en posant $u=\cos x+1$.
4. Même travail avec `integrale_chgtvar_bornes(f,a,b,eqn)` qui calcule l'intégrale de $f(x)$ entre a et b par changement de variable.
   En déduire $\displaystyle\int_0^{\frac{\pi}{4}} \frac{\mathrm{d}x}{2+\sin^2 x}$, en posant $u=\tan x$.

1. (a) La fonction est $f(x)=\sqrt{1-x^2}$ et on pose la fonction $\varphi(u)=\sin u$, c'est-à-dire que l'on espère que le changement de variable $x=\sin u$ va simplifier le calcul de l'intégrale :
   `f = sqrt(1-x^2)` `phi = sin(u)`
   (b) On obtient $f\big(\varphi(u)\big)$ en substituant la variable x par $\sin u$: cela donne $f\big(\varphi(u)\big)=\sqrt{1-\sin^2 u}=\sqrt{\cos^2 u}=|\cos u|$. Ce qui s'obtient par la commande `frondphi = f(x=phi)` (puis en simplifiant). (Notez que Sage « oublie » les valeurs absolues, car il calcule en fait la racine carrée complexe et pas réelle. Ce sera heureusement sans conséquence pour la suite. En effet pour que $f(x)$ soit définie, il faut $x\in[-1,1]$, comme $x=\sin u$ alors $u\in[-\frac{\pi}{2},\frac{\pi}{2}]$ et donc $\cos u\geqslant 0$.) Enfin $\varphi'(u)=\cos u$ s'obtient par `dphi = diff(phi,u)`.
   (c) On pose $g(u)=f\big(\varphi(u)\big)\cdot\varphi'(u)=\cos^2 u=\frac{1+\cos(2u)}{2}$. Donc $\int g(u)\,\mathrm{d}u=\frac{u}{2}+\frac{\sin(2u)}{4}$.
   (d) Pour obtenir une primitive de f, on utilise la formule de changement de variable :
   $$\int \sqrt{1-x^2}\,\mathrm{d}x=\int f(x)\,\mathrm{d}x=\int g(u)\,\mathrm{d}u=\frac{u}{2}+\frac{\sin(2u)}{4}.$$
   Mais il ne faut pas oublier de revenir à la variable x, Comme $x=\sin u$ alors $u=\arcsin x$, donc
   $$\int \sqrt{1-x^2}\,\mathrm{d}x=\frac{\arcsin x}{2}+\frac{\sin(2\arcsin x)}{4}.$$
   Les commandes sont donc `G = integral(g,u)` puis `F = G(u = arcsin(x))` donne la primitive recherchée. Après simplification de $\sin(2\arcsin x)$, on obtient :
   $$\int \sqrt{1-x^2}\,\mathrm{d}x=\frac{\arcsin x}{2}+\frac{x}{2}\sqrt{1-x^2}.$$
2. Pour calculer la primitive $\int \frac{\mathrm{d}x}{1+\left(\frac{x+1}{x}\right)^{1/3}}$, l'énoncé nous recommande le changement de variable $u^3=\frac{x+1}{x}$. La seule difficulté supplémentaire est qu'il faut exprimer x en fonction de u et aussi u en fonction de x. Pour cela on définit l'équation `u^3 == (x+1)/x` reliant u et x : `eqn = u^3 == (x+1)/x`. On peut maintenant résoudre l'équation afin d'obtenir $\phi(u)$ (x en fonction de u) : `phi = solve(eqn,x)[0].rhs()` et $\phi^{-1}(x)$ (u en fonction de x) : `phi_inv = solve(eqn,u)[0].rhs()`. Le reste se déroule comme précédemment.
3. On automatise la méthode précédente :

Code 57 (*integrales-chgtvar (1)*).

```
def integrale_chgtvar(f,eqn):
    phi = solve(eqn,x)[0].rhs()      # x en fonction de u : fonction phi(u)=x
```

```
    phi_inv = solve(eqn,u)[0].rhs() # u en fonction de x : inverse de phi : phi_inv(x)=u
    frondphi = f(x=phi)             # la fonction f(phi(u))
    dphi = diff(phi,u)              # sa dérivée
    g = frondphi*dphi               # la nouvelle fonction g(u)
    g = g.simplify_full()
    G = integral(g,u)               # g doit être plus facile à intégrer
    F = G(u = phi_inv)              # on revient à la variable x
    F = F.simplify_full()
    return F
```

Ce qui s'utilise ainsi :

Code 58 (*integrales-chgtvar (2)*).
```
f = (cos(x)+1)^n*sin(x)
eqn = u == cos(x)+1
integrale_chgtvar(f,eqn)
```

Et montre que

$$\int (\cos x+1)^n \cdot \sin x \,\mathrm{d}x = -\frac{1}{n+1}(\cos x+1)^{n+1}.$$

4. Pour la formule de changement de variable des intégrales, on adapte la procédure précédente en calculant l'intégrale $\int_{\varphi^{-1}(a)}^{\varphi^{-1}(b)} g(u)\,\mathrm{d}u$.
Pour cela on calcule $\varphi^{-1}(a)$ (par `a_inv = phi_inv(x=a)`) et $\varphi^{-1}(b)$ (par `b_inv = phi_inv(x=b)`).
Pour calculer $\int_0^{\frac{\pi}{4}} \frac{\mathrm{d}x}{2+\sin^2 x}$, on pose $u = \tan x$. Donc $x = \phi(u) = \arctan u$ et $u = \varphi^{-1}(x) = \tan x$. Nous avons $\phi'(u) = \frac{1}{1+u^2}$. D'autre part, comme $a = 0$ alors $\varphi^{-1}(a) = \tan 0 = 0$ et comme $b = \frac{\pi}{4}$ alors $\varphi^{-1}(b) = \tan\frac{\pi}{4} = 1$.
Ainsi la formule de changement de variable :

$$\int_a^b f(x)\,\mathrm{d}x = \int_{\varphi^{-1}(a)}^{\varphi^{-1}(b)} f\big(\varphi(u)\big)\cdot\varphi'(u)\,\mathrm{d}u$$

devient

$$I = \int_0^{\frac{\pi}{4}} \frac{1}{2+\sin^2 x}\,\mathrm{d}x = \int_0^1 \frac{1}{2+\sin^2(\arctan u)}\frac{1}{1+u^2}\,\mathrm{d}u.$$

Mais $\sin^2(\arctan u) = \frac{u^2}{1+u^2}$ donc

$$I = \int_0^1 \frac{1}{2+\frac{u^2}{1+u^2}}\frac{1}{1+u^2}\,\mathrm{d}u = \int_0^1 \frac{\mathrm{d}u}{2+3u^2}\,\mathrm{d}u = \left[\frac{1}{\sqrt{6}}\arctan\left(\frac{\sqrt{6}}{2}\right)\right]_0^1 = \frac{1}{\sqrt{6}}\arctan\left(\frac{\sqrt{6}}{2}\right).$$

Ce que l'ordinateur confirme !

8. Polynômes

8.1. Manipuler les polynômes

Travaux pratiques 32.
Soient $P(X) = X^4 - 3X^2 - 1$, $Q(X) = (X+1)^4$ des polynômes de $\mathbb{Q}[X]$. Après avoir déclaré l'anneau des polynômes $\mathbb{Q}[X]$ par `R.<X> = QQ[]` (où `QQ` désigne le corps des rationnels), répondre aux questions suivantes :

1. Est-ce que $\deg(P\cdot Q) = \deg P + \deg Q$?
2. Est-ce que $\deg(P-Q) = \max(\deg P, \deg Q)$?
3. Développer Q. Quel est le coefficient devant X^3 ?
4. Quel est le quotient de la division euclidienne de P par $(X+1)^2$? Et le reste ?
5. Quelles sont les racines de Q ? Et celles de P ?

1. (et 2.) La commande `degree()` permet d'obtenir le degré ; ainsi, après avoir déclaré l'anneau de polynômes $\mathbb{Q}[X]$ et avoir défini les deux polynômes P et Q, on teste les égalités sur les degrés.

Code 59 (*intro-polynome.sage (1)*).
```
R.<X> = QQ[]
P = X^4-3*X^2-1
Q = (X+1)^4
(P*Q).degree() == P.degree() + Q.degree()
(P-Q).degree() == max(P.degree(),Q.degree())
```

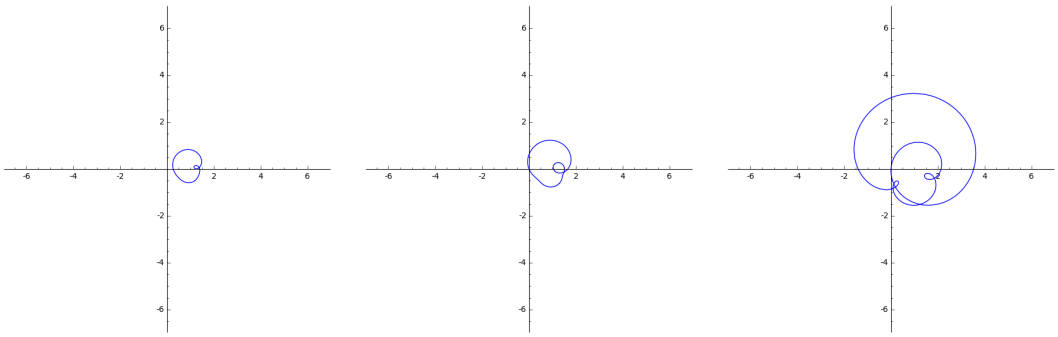
   La première égalité est vraie, par contre la seconde est fausse (il n'y a pas toujours égalité des degrés). Les énoncés vrais en toute généralité pour des polynômes non nuls sont :

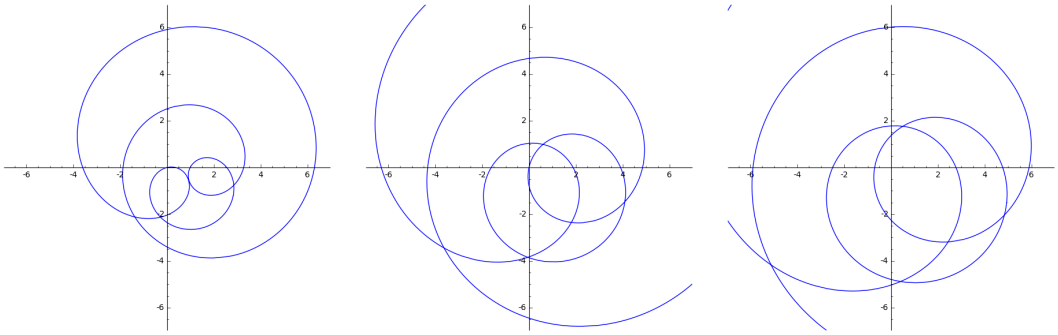
$$\deg(P\cdot Q)=\deg P+\deg Q \quad \text{et} \quad \deg(P+Q)\leqslant \max(\deg P,\deg Q)$$

3. On développe un polynôme par `expand(Q)`. On récupère les coefficients comme si le polynôme était une liste : `Q[k]` renvoie le coefficient a_k devant X^k.
4. On obtient le quotient et le reste comme avec les entiers. Le quotient `P // (X+1)^2` vaut X^2-2X. Le reste `P % (X+1)^2` vaut $2X-1$.
5. La question est ambigüe ! Il faut préciser dans quel ensemble on cherche les racines. Est-ce dans $\mathbb{Q}$, $\mathbb{R}$ ou $\mathbb{C}$? Souhaitons-nous une racine exacte ou approchée ?
   - `P.roots()` : renvoie les racines du corps de base (ici $\mathbb{Q}$). Renvoie ici une liste vide car P n'a pas de racines rationnelles.
   - `Q.roots()` renvoie `[(-1, 4)]` car -1 est une racine de multiplicité 4.
   - `P.roots(QQbar)` : racines exactes dans $\mathbb{C}$ (pour un polynôme à coefficients dans $\mathbb{Q}$). Ici renvoie deux racines réelles et deux racines imaginaires pures : `-1.817354021023971?`, `1.817354021023971?`, `-0.5502505227003375?*I`, `0.5502505227003375?*I`. Le point d'interrogation en fin d'écriture signifie que Sage calcule avec les valeurs exactes, mais n'affiche que les premières décimales.
   - `P.roots(RR)` : racines réelles *approchées* : `-1.81735402102397`, `1.81735402102397`. On quitte donc la résolution formelle pour une résolution numérique approchée.
   - `P.roots(CC)` : racines complexes *approchées*.

Travaux pratiques 33.
- Pour un polynôme $P\in\mathbb{C}[X]$ et un réel $r>0$, tracer l'image par P du cercle centré à l'origine de $\mathbb{C}$ et de rayon r.
- Faites varier r (ou mieux faites une animation). En quoi cela illustre-t-il le théorème de d'Alembert-Gauss ?
- Application à $P(X)=X^4-X^3+X^2-\mathrm{i}X+1\in\mathbb{C}[X]$.

Voici quelques images de l'animation pour des valeurs de r valant successivement : $r_0=0.5$, $r_1=0.6176\ldots$, $r_2=0.9534\ldots$, $r_3=1.2082\ldots$, $r_4=1.4055\ldots$ et $r_5=1.5$.





Quand la courbe $\mathscr{C}_r$ passe-t-elle par l'origine ? La courbe passe par l'origine s'il existe un nombre complexe re^{it} tel que $P(re^{it}) = 0$ autrement dit lorsque P admet une racine de module r. D'après le théorème de d'Alembert-Gauss un polynôme de degré n a au plus n racines. Alors il y a au plus n valeurs $r_1, \dots, r_n$ pour lesquelles $\mathscr{C}_{r_i}$ passe par l'origine. Pour notre exemple de degré 4, il y a 4 racines, qui conduisent ici à 4 modules distincts, $r_1, \dots, r_4$.

Voici comment procéder :

- On commence par définir l'anneau $\mathbb{C}[X]$ par `R.<X> = CC[]`. Les calculs seront donc des calculs approchés avec des nombres complexes. Notre polynôme P est défini par `P = X^4-X^3+X^2-I*X+1`.
- Le cercle de rayon r est l'ensemble des complexes de la forme re^{it} pour $t \in [0, 2\pi]$. La courbe $\mathscr{C}_r$ cherchée est donc l'ensemble
$$\mathscr{C}_r = \left\{P(re^{it}) \mid t \in [0, 2\pi]\right\}.$$
Voici la fonction qui renvoie ce tracé :

Code 60 (*intro-polynome.sage (2)*).
```
def plot_image_cercle(P,r):
    var('t')
    zreal = P(r*exp(I*t)).real()
    zimag = P(r*exp(I*t)).imag()
    G = parametric_plot( (zreal,zimag), (t,0,2*pi) )
    return G
```

- Une animation est une juxtaposition d'images. On la définit par :
`A = animate( [plot_image_cercle(P,r) for r in srange(0.5,1.5,0.05)] )`
puis on l'affiche par `A.show()`.

Remarque.
Par défaut Sage fait les calculs dans un ensemble appelé SR (*Symbolic Ring*). Dans cet ensemble vous pouvez définir des expressions polynomiales et en trouver les racines. L'accès aux fonctions spécifiques aux polynômes (comme le degré, les coefficients,...) est un peu différent. Voici comment déclarer un polynôme, récupérer son degré ainsi que le coefficient devant X^2 :

Code 61 (*intro-polynome.sage (3)*).
```
var('X')
P = X^4-3*X^2-1
P.degree(X)
P.coefficient(X,2)
```

8.2. Algorithme de Horner

L'algorithme de Horner permet d'évaluer rapidement un polynôme P en une valeur α. Voyons comment il fonctionne à la main avec l'exemple de $P(X) = X^5 + X^4 - 5X^3 - 3X - 2$ et $\alpha = 2$.
Sur la première ligne du tableau, on écrit les coefficients de P, $a_n, a_{n-1}, \dots, a_1, a_0$. On reporte en troisième ligne première colonne, a_n, le coefficient dominant de P. On multiplie ce nombre par α, on l'écrit en deuxième ligne, deuxième colonne, puis on ajoute a_{n-1} que l'on écrit en troisième ligne, deuxième colonne. Et on recommence. À

chaque étape, on multiplie le dernier nombre obtenu par α et on lui ajoute un coefficient de P. Le dernier nombre obtenu est $P(\alpha)$.

a_k	1	1	−5	0	−3	−2
$\alpha = 2$		2	6	2	4	2
b_k	1	3	1	2	1	0

Ici le dernier nombre est 0, donc $P(2) = 0$. Conclusion : 2 est racine de P.
Avec le même polynôme et $\alpha = -1$ le tableau se complète ainsi :

a_k	1	1	−5	0	−3	−2
$\alpha = -1$		−1	0	5	−5	8
b_k	1	0	−5	5	−8	6

ce qui implique $P(-1) = 6$.
Le travail à faire suivant formalise cette procédure et montre que l'algorithme de Horner permet des calculs efficaces avec les polynômes.

Travaux pratiques 34.
On fixe un corps K, et $\alpha \in K$. Soit $P(X) = a_n X^n + a_{n-1}X^{n-1} + \cdots + a_k X^k + \cdots + a_1 X + a_0 \in K[X]$.
On pourra prendre pour les applications $P(X) = X^5 + X^4 - 5X^3 - 3X - 2$ et $\alpha = 2$, puis $\alpha = -1$.

1. Compter et comparer le nombre de multiplications nécessaires dans K pour calculer $P(\alpha)$, par :
   (a) le calcul direct :
   $$P(\alpha) = a_n \alpha^n + a_{n-1}\alpha^{n-1} + \cdots + a_k \alpha^k + \cdots + a_1 \alpha + a_0,$$
   (b) l'algorithme de Horner :
   $$P(\alpha) = \Big(\big((a_n \alpha + a_{n-1})\alpha + a_{n-2}\big)\alpha + \cdots + a_1\Big)\alpha + a_0.$$
   (c) Écrire une fonction qui calcule $P(\alpha)$ par l'algorithme de Horner.
2. On formalise le calcul précédent en définissant la suite :
   $$b_n = a_n \qquad \text{puis pour} \qquad n-1 \geqslant k \geqslant 0 \ : \ b_k = \alpha b_{k+1} + a_k.$$
   (a) Montrer que dans la division euclidienne de $P(X)$ par $X - \alpha$ qui s'écrit :
   $$P(X) = (X - \alpha)Q(X) + P(\alpha)$$
   le reste $P(\alpha)$ est égal à b_0 et le quotient $Q(X)$ est égal au polynôme $b_n X^{n-1} + \cdots + b_2 X + b_1$ dont les coefficients sont les b_k pour $k \geqslant 1$.
   Indications. Pour ce faire développer $(X - \alpha)Q(X)$ et retrouver pour les coefficients de Q la même relation de récurrence que celle définissant la suite des b_k.
   (b) Modifier votre fonction afin qu'elle calcule la suite $(b_n, b_{n-1}, \ldots, b_1, b_0)$ et qu'elle renvoie le quotient Q et le reste $b_0 = P(\alpha)$.
   (c) Écrire une fonction qui calcule l'expression d'un polynôme $P(X)$ dans la base des $(X - \alpha)^k$, c'est-à-dire calculer les $c_k \in K$ tels que :
   $$P(X) = c_n(X - \alpha)^n + c_{n-1}(X - \alpha)^{n-1} + \cdots + c_1(X - \alpha) + c_0.$$

1. (a) Le calcul d'un monôme $a_k \cdot \alpha^k$ de degré k nécessite k multiplications, donc pour calculer $P(\alpha)$ il faut $n + (n-1) + \cdots + k + \cdots + 1 + 0 = \frac{n(n+1)}{2}$ multiplications (et n additions).

(b) La méthode de Horner
$$P(\alpha) = \Big(\big((a_n\alpha + a_{n-1})\alpha + a_{n-2}\big)\alpha + \cdots + a_1\Big)\alpha + a_0.$$
nécessite seulement n multiplications (et n additions). On passe d'un coût d'ordre $\frac{1}{2}n^2$ à un coût d'ordre n ; le gain est donc énorme.

(c) Pour l'implémentation, on note que la formule de récurrence se fait pour des indices k allant en décroissant.

Code 62 (*horner.sage (1)*)**.**
```
def eval_horner(P,alpha):
    n = P.degree()
    val = P[n]
    for k in range(n-1,-1,-1):
        val = alpha*val + P[k]
    return val
```

2. (a) On note la division euclidienne de $P(X)$ par $X-\alpha$ sous la forme $P(X) = (X-\alpha)Q(X) + R$. Le polynôme Q est de degré $n-1$ et on le note $Q(X) = b'_nX^{n-1} + \cdots + b'_kX^{k-1} + \cdots + b'_2X + b'_1$ (on veut montrer $b'_k = b_k$). Le reste R est de degré strictement inférieur à $\deg(X-\alpha) = 1$ donc R est un polynôme constant. En évaluant l'égalité de la division euclidienne en α, on a immédiatement $P(\alpha) = R$, que l'on note pour l'instant b'_0.
L'égalité $P(X) = (X-\alpha)Q(X) + R$ s'écrit :
$$P(X) = (X-\alpha)\big(b'_nX^{n-1} + \cdots + b'_kX^{k-1} + \cdots + b'_2X + b'_1\big) + b'_0.$$
On développe le terme de droite et on regroupe les monômes de même degré :
$$P(X) = b'_nX^n + (b'_{n-1} - \alpha b'_n)X^{n-1} + \cdots + (b'_k - \alpha b'_{k+1})X^k + \cdots + (b'_1 - \alpha b'_2)X + (b'_0 - \alpha b'_1)$$
On identifie coefficient par coefficient :
$$\left\{\begin{array}{rcl} b'_n & = & a_n \\ b'_{n-1} - \alpha b'_n & = & a_{n-1} \\ & \cdots & \\ b'_k - \alpha b'_{k+1} & = & a_k \\ & \cdots & \\ b'_0 - \alpha b'_1 & = & a_0 \end{array}\right. \quad \text{donc} \quad \left\{\begin{array}{rcl} b'_n & = & a_n \\ b_{n-1'} & = & \alpha b'_n + a_{n-1} \\ & \cdots & \\ b'_k & = & \alpha b'_{k+1} + a_k \\ & \cdots & \\ b'_0 & = & \alpha b'_1 + a_0 \end{array}\right.$$
Les suites (b'_k) et (b_k) sont définies par le même terme initial a_n et la même relation de récurrence, elles sont donc égales.

(b) On modifie légèrement le code précédent. La suite $(b_n, b_{n-1}, \ldots, b_1)$ donne les coefficients de Q (attention au décalage) et b_0 donne le reste qui n'est autre que $P(\alpha)$.

Code 63 (*horner.sage (2)*)**.**
```
def division_horner(P,alpha):
    n = P.degree()
    if n <= 0: return 0,P
    b = [None] * (n+1)     # Liste triviale de longueur n+1
    b[n] = P[n]
    for k in range(n-1,-1,-1):
        b[k] = alpha*b[k+1] + P[k]
    Q = sum( b[k+1]*X^k for k in range(0,n) )
    R = b[0]
    return Q,R
```

- Pour notre polynôme $P(X) = X^5 + X^4 - 5X^3 - 3X - 2$ et $\alpha = 2$, l'appel de la fonction définie ci-dessus : `division_horner(P,alpha)` renvoie un reste R nul et un quotient $Q(X) = X^4 + 3X^3 + X^2 + 2X + 1$. Ce qui signifie que $X-2$ divise $P(X)$, ou encore $P(2) = 0$.
- Pour ce même polynôme et $\alpha = -1$, la commande `division_horner(P,alpha)` renvoie $Q(X) = X^4 - 5X^2 + 5X - 8$ et un reste $R = 6$.

(c) On obtient les c_k comme restes de divisions euclidiennes successives par $X-\alpha$. On commence par la division euclidienne de $P(X)$ par $X-\alpha$. On a vu comment calculer le quotient Q_1 et le reste $b_0 = P(\alpha)$. En évaluant l'expression $P(X) = c_n(X-\alpha)^n + \cdots + c_1(X-\alpha) + c_0$ en α on voit que l'on a aussi $c_0 = P(\alpha)$ donc c_0 est le reste de la première division par $X-\alpha$.
On a donc $P(X) = (X-\alpha)Q_1(X) + c_0$. On recommence en divisant Q_1 par $X-\alpha$, on obtient un quotient Q_2 et un reste qui va être c_1. Donc $P(X) = (X-\alpha)\big((X-\alpha)Q_2 + c_1\big) + c_0$.
On continue ainsi de suite jusqu'à ce que le quotient soit nul (au bout de $n+1$ étapes), on a alors

$$P(X) = \Big(\big((c_n(X-\alpha) + c_{n-1})(X-\alpha) + c_{n-2}\big)(X-\alpha) + \cdots c_1\Big)(X-\alpha) + c_0$$

soit en développant :

$$P(X) = c_n(X-\alpha)^n + \cdots + c_1(X-\alpha) + c_0$$

Le code suivant renvoie les coefficients $(c_0, c_1, \ldots, c_n)$.

Code 64 (*horner.sage (3)*).

```
def developpe_horner(P,alpha):
    Q = P
    coeff = []
    while Q != 0:
        Q,R = division_horner(Q,alpha)
        coeff.append(R)
    return coeff
```

- Pour $P(X) = X^5 + X^4 - 5X^3 - 3X - 2$ et $\alpha = 2$, la commande `developpe_horner(P,alpha)` renvoie la liste `[0, 49, 74, 43, 11, 1]`, ce qui signifie :
$$P(X) = 1\cdot(X-2)^5 + 11\cdot(X-2)^4 + 43\cdot(X-2)^3 + 74\cdot(X-2)^2 + 49\cdot(X-2).$$
- Pour le même polynôme et $\alpha = -1$, la fonction renvoie `[6, -17, 11, 1, -4, 1]`, donc
$$P(X) = 1\cdot(X+1)^5 - 4\cdot(X+1)^4 + 1\cdot(X+1)^3 + 11\cdot(X+1)^2 - 17\cdot(X-2) + 6.$$

8.3. Interpolation de Lagrange

Théorème 1 (Interpolation de Lagrange).
Soient (x_i, y_i), $i = 0, \ldots, n$, une suite de $n+1$ points, d'abscisses deux à deux distinctes. Il existe un unique polynôme P de degré inférieur ou égal à n tel que

$$P(x_i) = y_i \quad \textit{pour } i = 0, \ldots, n.$$

En particulier, pour toute fonction f continue sur un intervalle contenant $x_0, \ldots, x_n$, il existe un unique polynôme P de degré inférieur ou égal à n tel que

$$P(x_i) = f(x_i) \quad \text{pour } i = 0, \ldots, n.$$

Travaux pratiques 35.

1. Montrer l'unicité du polynôme P dans le théorème d'interpolation de Lagrange.
2. Pour l'existence, étant donnés $x_0, \ldots, x_n$, on définit les polynômes de Lagrange $L_0, L_1, \ldots, L_n$:
$$L_i(X) = \prod_{j \neq i} \frac{X - x_j}{x_i - x_j}$$
Montrer que le polynôme :
$$P(X) = \sum_{i=0}^{n} y_i L_i(X)$$
répond au problème : $P(x_i) = y_i$ $(i = 0, \ldots, n)$ et $\deg P \leqslant n$.
3. Écrire une fonction qui, étant donnée une liste de points, renvoie le polynôme d'interpolation P.
4. Interpoler la fonction définie par $f(x) = \sin(2\pi x)e^{-x}$ sur l'intervalle $[0, 2]$, avec une subdivision régulière de $n+1$ points. Tracer les graphes de la fonction f et des polynômes d'interpolation correspondant à différentes valeurs de n.

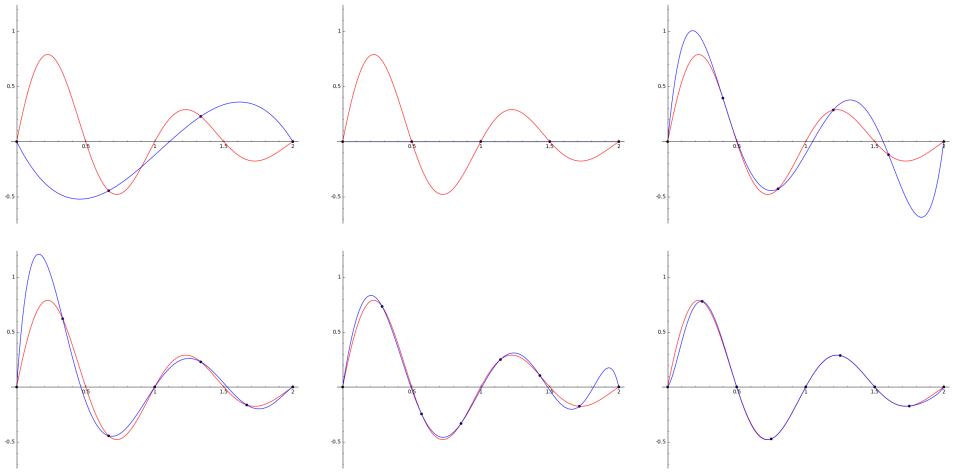
5. Faire le même travail avec la fonction définie par $f(x) = \frac{1}{1+8x^2}$ sur l'intervalle $[-1,1]$, avec une subdivision régulière de $n+1$ points. Quel problème apparaît ? C'est le *phénomène de Runge*.

1. Supposons que P et Q soient deux polynômes de degré $\leqslant n$ vérifiant tous les deux $P(x_i) = y_i$. Alors le polynôme $P-Q$ vérifie $(P-Q)(x_i) = 0$, $i = 0,\dots,n$. Ainsi $P-Q$ est un polynôme de degré inférieur à n ayant $n+1$ racines. D'après le théorème de d'Alembert-Gauss, c'est nécessairement le polynôme nul. Ainsi $P-Q=0$, donc $P=Q$.
2. Les polynômes L_i sont de degré exactement n et sont définis de sorte que
$$L_i(x_i) = 1 \quad \text{et} \quad L_i(x_j) = 0 \quad \text{pour } j \neq i.$$
Il est alors clair que $P(X) = \sum_{j=0}^{n} y_j L_j(X)$ est aussi de degré inférieur à n et vérifie pour $i = 0,\dots,n$:
$$P(x_i) = \sum_{j=0}^{n} y_j L_j(x_i) = y_i L_i(x_i) = y_i.$$
3. On commence par définir :
   - `R.<X> = RR[]` : l'anneau de polynômes (les coefficients sont des réels approchés) ;
   - `f = sin(2*pi*x)*exp(-x)` : la fonction ;
   - `liste_points = [ (2*i/n, f(x=2*i/n)) for i in range(n+1) ]` : une liste de points du graphe de f.

   La fonction suivante renvoie le polynôme interpolateur d'une liste de points.

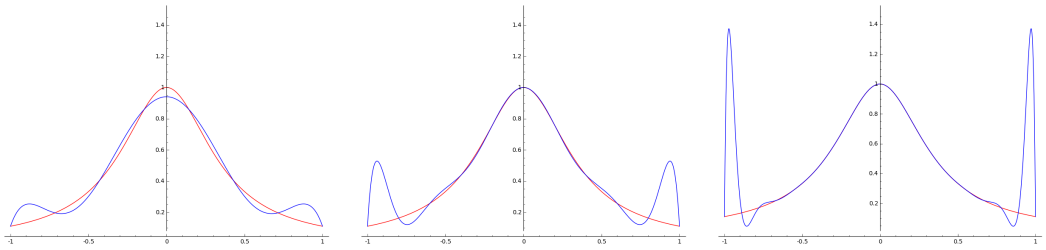
Code 65 (*interpolation.sage*).
```
def interpolation_lagrange(liste_points):
    n = len(liste_points)-1
    allx = [p[0] for p in liste_points] # Abscisses
    ally = [p[1] for p in liste_points] # Ordonnées
    liste_lagrange = []
    for i in range(n+1):
        A = prod(X-x for x in allx if x != allx[i])          # Numérateur
        B = prod(allx[i]-x for x in allx if x != allx[i]) # Dénominateur
        L = A/B                                          # Polynôme de Lagrange
        liste_lagrange.append(L)
    # Le polynôme interpolateur :
    lagrange = sum( liste_lagrange[i]*ally[i] for i in range(n+1) )
    return lagrange
```

4. Voici les tracés (en rouge la fonction, en bleu le polynôme) pour l'interpolation de $f(x) = \sin(2\pi x)e^{-x}$ avec $n = 3,4,5,6,7,8$. Pour $n=4$, le polynôme est le polynôme nul. À partir de $n=8$, il devient difficile de distinguer le graphe de la fonction, du graphe du polynôme.



5. Voici l'interpolation de $f(x) = \frac{1}{1+8x^2}$ pour $n = 7, 10, 18$. La zone centrale de la courbe est bien interpolée mais autour de -1 et $+1$, des «bosses» apparaissent. Ces bosses deviennent de plus en plus grandes en se décalant vers

les extrémités. Il y a convergence simple mais pas convergence uniforme. Pour éviter ce phénomène, il faudrait mieux choisir les x_i.



9. Équations différentielles

9.1. Équations différentielles $y' = f(x,y)$

La commande `desolve` permet de résoudre formellement des équations différentielles. Consultez la documentation, accessible par `help(desolve)`.

Travaux pratiques 36.

1. (a) Résoudre l'équation différentielle :
$$y' = y + x + 1.$$
Il s'agit donc de trouver toutes les fonctions $x \mapsto y(x)$ dérivables, telles que $y'(x) = y(x) + x + 1$, pour tout $x \in \mathbb{R}$.

   (b) Résoudre l'équation différentielle avec condition initiale :
$$y' = y + x + 1 \quad \text{et} \quad y(0) = 1.$$

2. (a) Résoudre sur $\mathbb{R}_+^*$ l'équation différentielle :
$$x^2 y' = (x-1)y.$$

   (b) Trouver la solution vérifiant $y(1) = 2$.

   (c) Peut-on trouver une solution sur $\mathbb{R}$?

3. Calculer la ***fonction logistique*** $y(x)$, solution de l'équation différentielle
$$y' = y(1-y) - 1.$$

1.

Code 66 (*equadiff-intro.sage (1)*).
```
var('x')
y = function('y',x)
yy = diff(y,x)
desolve(yy == y+x+1, y)
desolve(yy == y+x+1, y, ics=[0,1])
```

(a) On commence par déclarer la variable x. On définit une fonction $x \mapsto y(x)$ ainsi que sa dérivée $y'(x)$ par `diff(y,x)`. On prendra ici la convention de noter y' par `yy`. L'équation différentielle $y' = y + x + 1$ s'écrit donc `yy == y+x+1`. On lance la résolution de l'équation différentielle par la fonction `desolve`, avec pour argument l'équation différentielle à résoudre, ainsi que la fonction inconnue, qui est ici y. Sage renvoie :
```
-((x + 1)*e^(-x) - _C + e^(-x))*e^x
```
`_C` désigne une constante. Après simplification évidente, les solutions sont les :
$$y(x) = -x - 2 + C\exp(x) \quad \text{où} \quad C \in \mathbb{R}.$$

(b) Il est possible de préciser une condition initiale grâce à l'argument optionnel `ics` (voir la dernière ligne du code ci-dessus), ici on spécifie donc que l'on cherche la solution vérifiant $y(0) = 1$. L'unique solution est alors : $y(x) = -x - 2 + 3\exp(x)$, c'est-à-dire $C = 3$.

2.

Code 67 (*equadiff-intro.sage (2)*).
```
equadiff = x^2*yy == (x-1)*y
assume(x>0)
desolve(equadiff, y)
desolve(equadiff, y, ics=[1,2])
forget()
assume(x<0)
desolve(equadiff, y)
```

(a) En se restreignant à $\mathbb{R}_+^*$ ($x>0$), on trouve $y(x)=Cx\exp(\frac{1}{x})$, $C\in\mathbb{R}$.

(b) La solution vérifiant $y(1)=2$ est $y(x)=2x\exp(\frac{1}{x}-1)$, c'est-à-dire $C=2\exp(-\frac{1}{e})$.

(c) Sur $\mathbb{R}_-^*$ ($x<0$), les solutions sont aussi de la forme $y(x)=Cx\exp(\frac{1}{x})$.
La seule solution définie sur $\mathbb{R}$ est donc la fonction nulle $y(x)=0$ (et $C=0$). En effet si $C\neq 0$ il n'y a pas de limite finie en 0.

3.

Code 68 (*equadiff-intro.sage (3)*).
```
equadiff = yy == y*(1-y) - 1
sol_impl = desolve(equadiff, y)
sol = solve(sol_impl,y)
```

Sage ne résout pas directement cette équation différentielle. En effet, il renvoie une équation vérifiée par y :

$$-\frac{2}{3}\sqrt{3}\arctan\left(\frac{1}{3}\sqrt{3}\big(2y(x)-1\big)\right)=C+x. \tag{1}$$

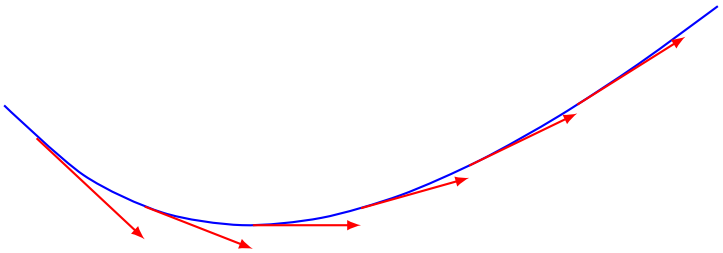
On demande alors à Sage de résoudre cette équation, pour obtenir :

$$y(x)=-\frac{\sqrt{3}}{2}\tan\left(\frac{\sqrt{3}}{2}C+\frac{\sqrt{3}}{2}x\right)+\frac{1}{2}. \tag{2}$$

L'équation (1) a l'avantage d'être valide quel que soit x, alors que pour l'équation (2) il faudrait préciser l'intervalle de définition.

9.2. Interprétation géométrique

- Les ***courbes intégrales*** d'une équation différentielle $y'=f(x,y)$ sont les graphes des solutions de cette équation.
- À chaque point d'une courbe intégrale, on associe un vecteur tangent. Si $y(x)$ est une solution de l'équation différentielle alors au point $(x,y(x))$ la tangente est portée par le vecteur $(x',y'(x))$. D'une part $x'=1$ et d'autre part comme y est solution de l'équation différentielle alors $y'(x)=f(x,y)$. Ainsi un ***vecteur tangent*** en (x,y) est $\big(1,f(x,y)\big)$.



- Le ***champ de vecteurs*** associé à l'équation différentielle $y'=f(x,y)$ est, en chaque point (x,y) du plan, le vecteur $\big(1,f(x,y)\big)$.
- Voici ce qui signifie géométriquement « résoudre » une équation différentielle. À partir d'un champ de vecteur, c'est-à-dire la donnée d'un vecteur attaché à chaque point du plan, il s'agit de trouver une courbe intégrale, c'est-à-dire une courbe qui en tout point est tangente aux vecteurs.
- Comme on ne s'occupera pas de la norme des vecteurs, la donnée d'un champ de vecteur revient ici à associer à chaque point, la pente de la tangente au graphe d'une solution passant par ce point.

Travaux pratiques 37.
Nous allons étudier graphiquement l'équation différentielle

$$y' = -xy.$$

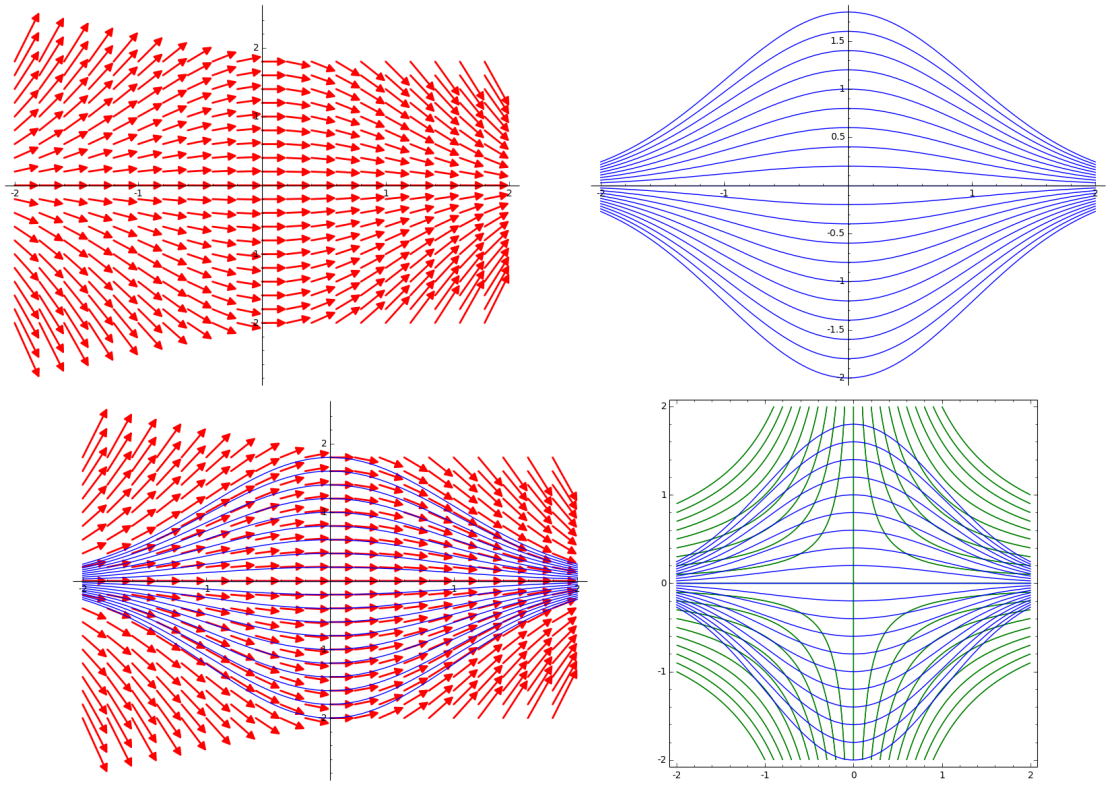
1. Représenter le champ de vecteurs associé à cette équation différentielle.
2. (a) Résoudre l'équation.
   (b) Tracer, sur un même graphe, les courbes intégrales correspondant aux solutions définies par $y(0) = k$, pour différentes valeurs de $k \in \mathbb{R}$.
   (c) Que remarquez-vous ? Quel théorème cela met-il en évidence ?
3. Tracer quelques isoclines de cette équation différentielle.

Les ***isoclines*** de l'équation différentielle $y' = f(x, y)$ sont les courbes sur lesquelles la tangente d'une courbe intégrale a une pente donnée c, c'est-à-dire qu'une isocline est un ensemble

$$\{(x, y) \in \mathbb{R}^2 \mid f(x, y) = c\}.$$

À ne pas confondre avec les solutions ! En chaque point d'une isocline, la solution passant par ce point croise cette isocline avec une pente c.
Voici quelques graphes :

- Le champ de vecteurs (en rouge).
- Les courbes intégrales (en bleu).
- Le champ de vecteurs superposé aux courbes intégrales : les vecteurs sont bien tangents aux courbes intégrales.
- Les isoclines (en vert). Pour une isocline fixée, vérifier que chaque intersection avec les courbes intégrales se fait en des points où la tangente a toujours la même pente.



1. Voici comment tracer le champ de vecteurs associé à $y' = f(x, y)$.

Code 69 (*equadiff-courbe.sage (1)*).

```
def champ_vecteurs(f,xmin,xmax,ymin,ymax,delta):
    G = Graphics()
    for i in srange(xmin, xmax, delta):
        for j in srange(ymin, ymax, delta):
            pt = vector([i,j])
```

```
            v = vector([1,f(u=i,v=j)])
            G = G + arrow(pt,pt+v)
    return G
```

- Les variables de la fonction f sont ici u et v pour ne pas interférer avec la variable x et la fonction y.
- N'hésitez pas à renormaliser les vecteurs v, (par exemple en $v/10$) pour plus de lisibilité.
- En fait la fonction `plot_vector_field`, prédéfinie dans Sage, trace aussi les champs de vecteurs.

2. (a) Les solutions de l'équation sont les

$$y(x) = C\exp\left(-\frac{x^2}{2}\right) \quad \text{où} \quad C \in \mathbb{R}.$$

(b) Le code suivant résout l'équation différentielle et trace la solution avec la condition initiale $y(0) = k$, pour k variant de `ymin` à `ymax` avec un pas de `delta`.

Code 70 (*equadiff-courbe.sage (2)*).

```
def courbes_integrales(equadiff,a,b,kmin,kmax,delta):
    G = Graphics()
    for k in srange(kmin, kmax, delta):
        sol = desolve(equadiff, y, ics=[0,k])
        G = G + plot(sol, (x, a, b))
    return G
```

(c) Les courbes intégrales forment une partition de l'espace : par un point il passe exactement une courbe intégrale. C'est une conséquence du théorème d'existence et d'unicité de Cauchy-Lipschitz.

3. Les isoclines sont définies par l'équation $f(x,y) = c$. Elles se tracent par la commande

```
implicit_plot(f-c, (x,xmin,xmax), (y,ymin, ymax))
```

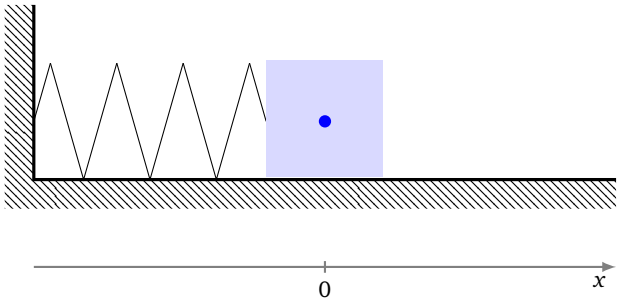
Ici les isoclines ont pour équation $-xy = c$, ce sont donc des hyperboles.

9.3. Équations différentielles du second ordre

Travaux pratiques 38.
L'équation différentielle

$$x''(t) + f\,x'(t) + \frac{k}{m}x(t) = 0$$

correspond au mouvement d'une masse m attachée à un ressort de constante $k > 0$ et des frottements $f \geqslant 0$.



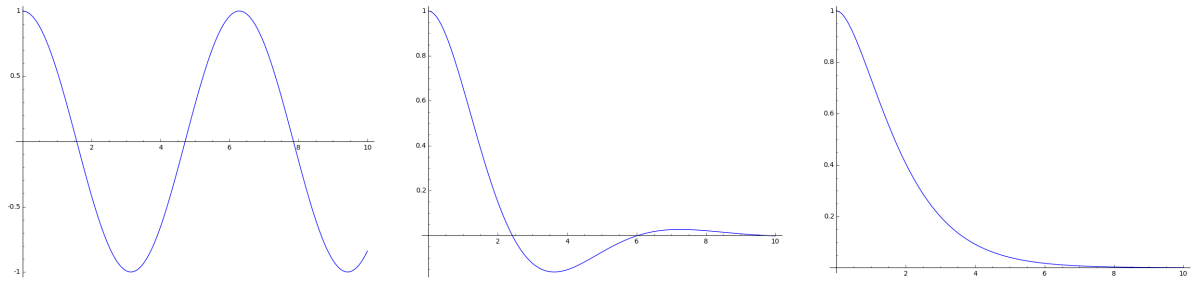
Résoudre et tracer les solutions pour différentes valeurs de $f \geqslant 0$ (prendre $k = 1$, $m = 1$), avec les conditions initiales :

$$x(0) = 1 \quad \text{et} \quad x'(0) = 0.$$

Pour une valeur de f, l'équation se résout par :

```
desolve(diff(x,t,2) + f*diff(x,t) + k/m*x(t) == 0, x, ics=[0,1,0])
```

Voici les solutions pour $f = 0$, $f = 1$, $f = 2$:



- Cas $f = 0$. Il n'y a pas de frottement. L'équation différentielle s'écrit $x'' + \frac{k}{m}x = 0$. Pour $k = 1$, $m = 1$ alors l'équation est juste $x'' + x = 0$ dont les solutions sont de la forme
$$x(t) = \lambda \cos t + \mu \sin t \qquad \lambda, \mu \in \mathbb{R}.$$
Avec les conditions initiales $x(0) = 1$ et $x'(0) = 0$, l'unique solution est
$$x(t) = \cos t.$$
Il s'agit donc d'un mouvement périodique.
- Cas $0 < f < 2$. Les frottements sont faibles. Par exemple pour $f = 1$ la solution est
$$\left(\frac{\sqrt{3}}{3}\sin\left(\frac{\sqrt{3}}{2}t\right) + \cos\left(\frac{\sqrt{3}}{2}t\right)\right)e^{-\frac{1}{2}t}.$$
Il s'agit d'un mouvement amorti oscillant autour de la position d'origine $x = 0$.
- Cas $f \geqslant 2$. Les frottements sont forts. Par exemple pour $f = 2$ la solution est
$$x(t) = (t+1)e^{-t}.$$
Il n'y a plus d'oscillations autour de la position d'origine.

Auteurs du chapitre

- Arnaud Bodin, Niels Borne, François Recher
- D'après un cours de calcul formel de Saïd Belmehdi, Jean-Paul Chehab, Bernard Germain-Bonne, François Recher donné à l'université Lille 1.
- Relu par Marc Mezzarobba et Paul Zimmermann.