3. Suites récurrentes et preuves formelles

Commençons par un exemple. Après avoir déclaré les deux variables a et b par `var('a,b')`, 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 que nous avons en Sage pour manipuler les expressions algébriques. Nous allons utiliser la puissance du calcul formel, non 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 observés.
4. Je tente de prouver la conjecture.

3.1. La suite de Fibonacci

La suite de Fibonacci est définie par les relations suivantes :

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

Travaux pratiques 1.

1. Calculer les 10 premiers termes de la suite de Fibonacci.
2. **Recherche d'une conjecture.** On cherche s'il peut exister des constantes ϕ, ψ, c réelles telles que
$$F_n = c\big(\phi^n - \psi^n\big).$$
Nous allons calculer ces constantes.
   (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 $H_n = \frac{1}{\sqrt{5}}\big(\phi^n - \psi^n\big)$ 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 1 (*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 2 (*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 3 (*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 avec Sage qu'avec ces constantes on a effectivement

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

   (a) On définit une nouvelle fonction `conjec` qui correspond à l'expression de 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 4 (*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 nous 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 5 (*fibonacci.sage (5)*).
  ```
  var('n')
  test = conjec(n)-conjec(n-1)-conjec(n-2)
  print(test.canonicalize_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 `canonicalize_radical()` (la commande `simplify_radical` ne fait plus partie des nouvelles version de `Sage`).
- **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 2.
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 6 (*cassini.sage (1)*).
```
def fibonacci_bis(n):
    return 1/sqrt(5)*( ((1+sqrt(5))/2)^n - ((1-sqrt(5))/2)^n )
```

Code 7 (*cassini.sage (2)*).
```
var('n')
cassini = fibonacci_bis(n+1)*fibonacci_bis(n-1)-fibonacci_bis(n)^2
```

```
print(cassini.canonicalize_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 facilement $(\sqrt{5}-1)(\sqrt{5}+1) = (\sqrt{5})^2 - 1 = 4$ 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 que $\mathscr{H}_{n+1}$ 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 3.
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 8 (*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.
Cette fonction `masuite` 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 conjecture 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 9 (*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 10 (*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 de n aussi grandes que l'on veut 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, nous allons 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 11 (*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 12 (*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 premières valeurs de 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 les vingt premiers entiers !