Bachelor of Ecole Polytechnique
Computational Mathematics, year 2, semester 1
Lecturer: Lucas Gerin [(send mail) (mailto:lucas.gerin@polytechnique.edu)](mailto:lucas.gerin@polytechnique.edu)

Symbolic computation 3: Linear recurrences

Table of contents


```python
# execute this part to modify the css style
from IPython.core.display import HTML
def css_styling():
    styles = open("./style/custom2.css").read()
    return HTML(styles)
css_styling()
```

```python
## loading python libraries

# necessary to display plots inline:
%matplotlib inline

# load the libraries
import matplotlib.pyplot as plt # 2D plotting library
import numpy as np              # package for scientific computing
from pylab import *

from math import *              # package for mathematics (pi, arctan, sqrt, factorial
import sympy as sympy             # package for symbolic computation
from sympy import *
```

The aim of this Lab session is to use SymPy to solve automatically some simple recurrences. We first solve a particular case "by hand" and then use the SymPy function `rsolve`.

Warm-up

Exercise 1. Solving a recurrence (almost) by hand

Do it yourself.
We consider the sequence defined by

$$\begin{cases} u_0 & = 1, \\ u_n & = 2u_{n-1} + 3n^2. \qquad (\forall n \geq 1). \end{cases} \qquad (\star)$$

1. Use `SymPy` to find α, a, b, c such that for every $0 \leq n \leq 3$
$$u_n = \alpha 2^n + an^2 + bn + c.$$
To solve a system of equations with `SymPy` *with unknowns x, y, write for example*

```
solve([x-y-2,3*y+x],[x,y])
```

2. Prove with `SymPy` that your formula is true for every $n \geq 0$.

```
# ------------- Question 1 --------------
```

Answers.
1.

```
#-------------- Question 2 ----------------
```

Answers.
2.

Solving recurrences with SymPy: `rsolve`

The function `rsolve`

We will use Sympy to obtain explicit formulas for some sequences defined by linear recurrences. More precisely, we will see how to obtain an explicit formula for u_n in two cases:

1. **Linear recurrence of order one:**: this is a sequence $(u_n)_{n\geq 0}$ is defined by
$$\begin{cases} u_0 & = a, \\ u_n & = \alpha u_{n-1} + f(n), \qquad (n \geq 1), \end{cases}$$
where a, α are some given constants and f is an arbitrary function.
2. **Linear recurrence of order two:**: this is a sequence $(u_n)_{n\geq 0}$ is defined by
$$\begin{cases} u_0 & = a, \\ u_1 & = b, \\ u_n & = \alpha u_{n-1} + \beta u_{n-2} + f(n), \qquad (n \geq 2), \end{cases}$$

where $a, , b, \alpha, \beta$ are some given constants and f is an arbitrary function.

Some known examples fit in this settings:

1. Geometric sequences: $u_0 = a, u_n = ru_{n-1}$.
2. Arithmetic sequences: $u_0 = a, u_n = u_{n-1} + r$.
3. The Fibonacci sequence: $F_1 = 1, F_2 = 1, F_n = F_{n-1} + F_{n-2}$.

The following script shows how to solve the two recurrences

$$u_0 = 5, \qquad u_n = 3u_{n-1},$$
$$v_0 = 1, \qquad v_n = 2v_{n-1} + n,$$

using function `rsolve` .

```
# First example: a geometric sequence
u = Function('u')                  # declares the name of the sequence
n = symbols('n',integer=True)      # declares the variable
f = u(n)-3*u(n-1)                  # the expression which is to be zero
ExplicitFormula1=rsolve(f,u(n),
                        {u(0):5}) # initial condition

print('The formula for u(n) is '+str(ExplicitFormula1)+'')

# Second example with a non-linear term
print('-------')
v = Function('v')                  # declares the name of the sequence
n = symbols('n',integer=True)      # declares the variable
f = v(n)-2*v(n-1)-n            # the expression which is to be zero
ExplicitFormula2=rsolve(f,v(n),
                        {v(0):1}) # initial condition

print('The formula for v(n) is '+str(ExplicitFormula2)+'')
```

Exercise 2. A first example with `rsolve`

Do it yourself.

1. Use `SymPy` to solve the recurrence of the Fibonacci sequence and find an explicit formula for F_n.
   (To set up two initial conditions you must write `{u(1):1,u(2):1}` .)
2. **(Theory)** Use the formula obtained at Question 1 to prove that

$$\lim_{n\to+\infty} \frac{F_n}{F_{n-1}} = \varphi = \frac{1+\sqrt{5}}{2}.$$

```
# Question 1
```

Answers.

1.
2.

Remark.
The output of `rsolve` is an *expression* which depends on the symbolic variable `n`. If we want to evaluate this expression (for instance for $n = 10$) we must write:

```
Value=ExplicitFormula.subs(n,10)
print('10th Fibonacci number = '+str(Value))
print('After simplification : '+str(simplify(Value)))
```

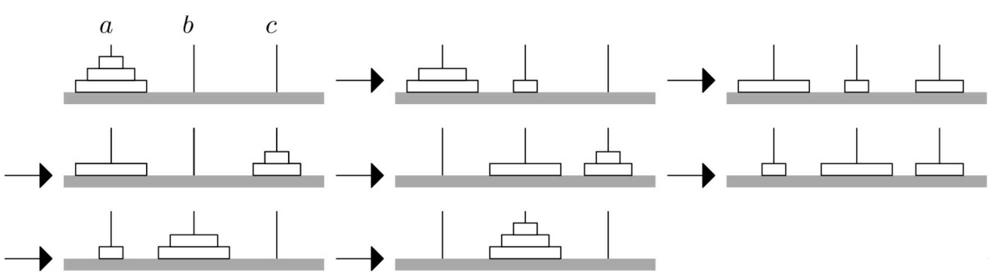
Application: the Towers of Hanoi

(This remainder of this session is devoted to the study of the Towers of Hanoi, which you have studied in "Discrete Mathematics" (Bachelor 1).)

Exercise 3. The puzzle with three rods

Let us recall this mathematical puzzle.
We are given a stack of n disks arranged from largest on the bottom to smallest on top placed on a rod a, together with two empty rods b, c. The Tower of Hanoi puzzle asks for the minimum number of moves required to move the entire stack, one disk at a time, from rod a to another (b or c). A move is allowed only if it moves a smaller disk on top of a larger one.

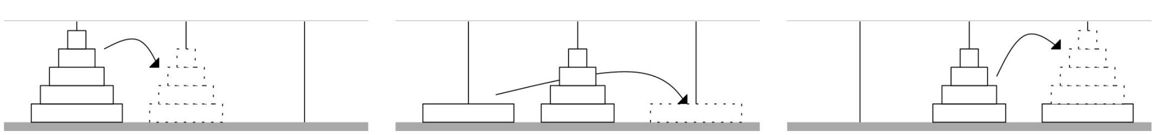
Here is an example which shows that the Tower of Hanoi with $n = 3$ disks is solvable in 7 moves:



More generally we use the following recursive strategy to solve the Tower of Hanoi with n disks:

- Assume that you have a strategy for $n - 1$ disks;

1. Move the $n - 1$ smallest disks from rod a to rod b with your strategy
2. Move disk n from a to c
3. Move the $n - 1$ smallest disks from rod b to rod c with your strategy



As there is an obvious strategy for $n = 1$ this recursive algorithm allows to solve the Hanoi of Tower for any n.

Let H_n be the sequence defined by

$$H_n = \text{Number of moves for solving Hanoi with } n \text{ disks, with the recursive strategy.}$$

Of course $H_1 = 1$, the above figure gives $H_3 = 7$.

The aim of the session is to study the sequence (H_n) and eventually compare to several

Do it yourself. Find a recursive formula for the sequence H_n: write H_n as a function of H_{n-1}.

Answers.

Do it yourself.

1. Write a recursive function `Hanoi(n)` which returns the value of H_n.
2. Write a small script which returns the 15 first values of H_n. Can you guess a formula?

```
def Hanoi(n):
    # input: positive integer n
    # output: number of moves needed to solve Hanoi with n disks
```

Answers.

Do it yourself.

1. Use `rsolve` to find an explicit formula for the sequence H_n.
2. Compare with your own function `Hanoi(n)` and with your guess.

The output should be like

```
 Sympy says the formula for H(n) is ...
Sympy says the first values for H(n) are [1, 3, 7, ...]
Hanoi(n) says the first values for H(n) are [1, 3, 7, ...]
```

```

```

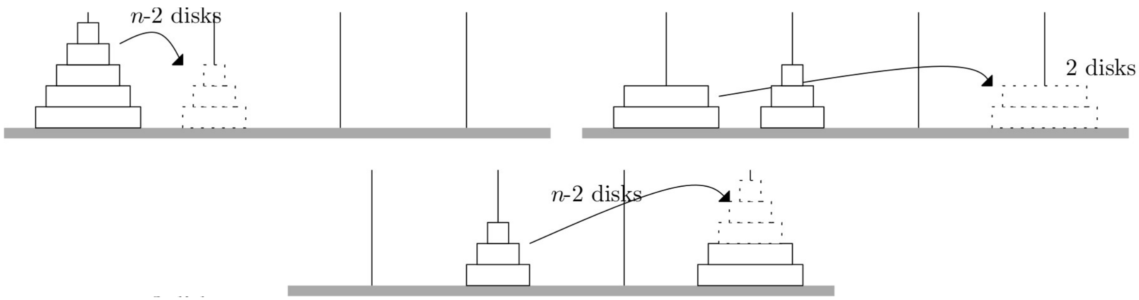
Exercise 4. The tower of Hanoi with four rods

We now consider the towers of Hanoi with four rods a, b, c, d. In that case, formulas are more difficult to guess and we will really need `rsolve` to obtain exact formulas and asymptotics.

The recursive algorithm for fours rods

The recursive strategy is as follows.

- If $n = 1$ or $n = 2$, use the algorithm of Exercise 4 to move the disk(s) to peg d (it takes 1 move for $n = 1$, 3 moves for $n = 2$).
- If $n \geq 3$, use the following recursion:
  - Use the fours rods a, b, c, d to move the $n - 2$ smallest disks from $a \to b$.
  - Use rods a, c, d to move the two largest disks $a \to d$.
  - Use the fours rods a, b, c, d to move the $n - 2$ smallest disks from $b \to d$.



Let J_n be the sequence defined by

$$J_n = \text{ Number of moves for solving Hanoi with } n \text{ disks and 4 rods.}$$

As before, $J_1 = 1$, $J_2 = 3$.

Do it yourself. Find a recursive formula for the sequence J_n.

Answers.

Do it yourself.

1. Write a recursive function `HanoiFour(n)` which returns the value of J_n.
2. Write a small script which returns the ten first values of J_n. Can you guess a formula? (You can ask [https://oeis.org/ (https://oeis.org/)](https://oeis.org/).)

```
def HanoiFour(n):
```

Answers.

Do it yourself.

1. Use `rsolve` to find an explicit formula for the sequence J_n.
2. Compare with your own function `HanoiFour(n)` .

Answers.

Do it yourself. **Theory**

1. According to the formula found by `SymPy`, what happens when $n \to +\infty$ for the sequence $\frac{J_n}{\sqrt{2}^n}$?
2. Using the formula found by `SymPy`, prove that $\frac{J_n}{H_n} \to 0$.

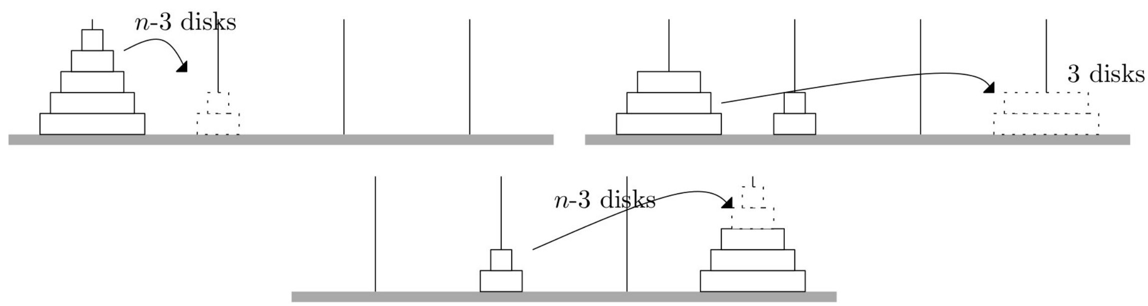
Answers.

1.
2.

Exercise 5. The recursive algorithm for fours rods: a better variant?

We define a variant of the previous strategy:

- If $n = 1$ or $n = 2$ or $n = 3$, use the previous algorithm with 4 rods to move the disk(s) to peg d (it takes 1 move for $n = 1$, 3 moves for $n = 2$, 5 moves for $n = 3$).
- If $n \geq 3$, use the following recursion:
  - Use the fours rods a, b, c, d to move the $n - 3$ smallest disks from $a \to b$.
  - Use rods a, c, d to move the three largest disks $a \to d$.
  - Use the fours rods a, b, c, d to move the $n - 3$ smallest disks from $b \to d$.



Let K_n be the sequence defined by

$$K_n = \text{ Number of moves for solving Hanoi with } n \text{ disks and 4 rods with that strategy.}$$

We have, $K_1 = 1$, $K_2 = 3$, $K_3 = 5$.

Do it yourself.

1. Find a recursive formula for the sequence K_n. *(Do not try to solve the recurrence with* `SymPy` *, it does not seem to work.)*

2. Write a function `HanoiFourBis(n)` which returns the value of K_n.
3. Plot on the same figure the first values of K_n and J_n. Which algorithm looks faster?

Answers.

```
def HanoiFourBis(n):
```

Two-dimensional recurrence: matrices with SymPy

Exercise 6. A double linear recurrence

In Lab 2 we proved that if a_n, b_n are integers defined by

$$a_n + b_n\sqrt{2} = (1+\sqrt{2})^n,$$

then

$$\begin{pmatrix} a_n \\ b_n \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}^{n-1} \times \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Do it yourself.

1. Use `SymPy` to find an explicit formula for a_n.

   (In `SymPy` matrices are defined by `Matrix([[a,b],[c,d]])` .)
2. Use the formula obtained at Question 1 to find real numbers c, R such that

$$a_n \overset{n\to+\infty}{\sim} cR^n.$$

Answers.

1.
2.