SimulaMath Documentation

Release 1.1.beta1

SimulaMath Developers

Sep 30, 2021

Table of contents

Chapter 1

Introduction

SimulaMath is a scientific computing software, dedicated to learning, teaching and research in mathematics. It is developed with the Python language with an emphasis on simplicity (ease of use), through a graphical user interface (GUI). It covers many areas of mathematics including linear algebra, calculus, number theory, descriptive statistics, inferential statistics, probability distributions, 2D and 3D graphics, multivariate polynomials and Groebner bases, elliptic curves, linear codes, and finite fields. It runs on Windows, Mac OSX and many Linux platforms.

1.1 Objectives

SimulaMath is designed to facilitate teaching, learning and research in mathematics from middle school to high school, and to encourage the use of new technologies on education.

1.2 Strong points of SimulaMath

- **Simplicity**: SimulaMath has a very intuitive interface, which allows you to generate results, and create quality graphics (2D and 3D) with very minimal effort.

Note: Golden rule in SimulaMath : the input and output data must converge to the mathematical syntax (notation).

- **Python**: Python is a high level programming language, easy to learn, dynamically typed and maintained by a very large community. It has become the choice of most data scientists today. SimulaMath is developed with Python and its programming language is derived from it like SageMath software. The objective was not to reinvent the wheel by creating a new programming language as is the case with most of scientific software but to simplify what already exists.

If you know how to program in Python, you can use most of its functions and scientific packages like **Numpy**, **Scipy**, **Sympy**, **Pandas** etc. for both clickable and programming interface.

- **Programming**: A new language derived from Python is introduced on SimulaMath. It means that 99% of Python valid code are also valid on SimulaMath. New syntax which is very closed to mathematical notation is added.
- **Two type of interfaces**: SimulaMath has two kind of interfaces : the clickable interface and programming interface. The clickable interface allows to get high results without knowing necessarily how to program and the programming interface is for everyone (programmers and those who want to learn programming).
- **Publication quality graphs**: SimulaMath has a very powerful and intuitive interface for two and three dimensional graphics. You can save your graphics in a range of formats : PNG, PDF, PGF, JPEG, SVG, etc.
- **Multi-platform**: SimulaMath runs on Windows, Mac OS X and many Linux (e.g. Ubuntu 16+) distributions.
- **Documentation**: the documentation is available in HTML and in PDF.
- **Multi-areas**: SimulaMath is not designed only for a specific area of mathematics. One can do Calculus, linear algebra, statistics, probability, elliptic curves, linear codes etc.

1.3 About the Developers

The first version of SimulaMath (version *1.0*), published at 2019, was designed and developed by Michel Seck, PhD in algebra and cryptography at the University Cheikh Anta Diop of Dakar (Senegal).

Since 2020, an international team (see the web page of the team) has joined the project. Thanks to this team, many new functionalities were added to this release.

1.4 What's new in SimulaMath `1.1`

- Programming: SimulaMath has a vera simple and powerful programming language derived from Python.
- Inferential statistics: Estimation by confident interval and hypothesis testing were added.
- Geometry on the plane: for 2d graphics, you can now add texts, images, and various 2D geometric objects: Points, Lines, Rays, Segments, Circles, Arcs, Polygons, Parallel lines, Perpendicular lines, Vectors, Angles, Angle Bisector, Ellipsis, Parabolas, Hyperbolas, Rotation, Homothety, Translation, Reflect about a point and a line, Areas, Barycenter, etc.
- Two languages: SimulaMath is now available in English and French.

- Choice of a level : You can choose between three levels: middle school, secondary school and high school.

Chapter 2

Installation

SimulaMath software can be installed on Windows (32 and 64 bits), Mac OS X (64 bits) and most of Linux distributions.

2.1 Under Windows

To install SimulaMath on Windows, simply run the installation program (**SimulaMath-vXXX-Windows-x64-XXX.exe** if you have a 64 bits version of Windows or **SimulaMath-vXXX-Windows-x86-XXX.exe** if you have a 32-bit version) by double-clicking on it in the Windows Explorer. The automatic installation program starts and guides you through the installation process; simply follow the instructions on the screen. SimulaMath will be install for all users (if you are the administrator), or for you only (if you are not). This allows you to install SimulaMath without any administrator rights.

2.2 Under Mac OS X

To install SimulaMath on an Apple Macbook, double-click on the file **SimulaMath-vXXX-macosx-XXX.pkg**, the installation process will begin. Then follow the instructions until your installation is complete.

SimulaMath is not a signed Apple application. Therefore, Gatekeeper (if you have OSX Mountain Lion or older) may complain about this. This is normal. If the installation is blocked, click on the Open Anyway button in the General pane of Security & Privacy preferences. This button is available for about an hour after you try to open SimulaMath (for more details see Mac Guide).

2.3 Under Linux

To install SimulaMath on Linux, copy (or download) the file **SimulaMath-vXXX-linux-XXX.run** on your computer, then open your terminal and place move to the folder that contains **SimulaMath-vXXX-linux-XXX.run** file. Usually this file will not have the required permissions to run normally. To give the file execution permissions , do one of the following:

- On the command line, type **chmod +x SimulaMath-vXXX-linux-XXX.run**.
- In the file manager, right-click on the file **SimulaMath-vXXX-linux-XXX.run**, select "Permissions", then check the box "Allow the file to run as a program".

Finally, run **./SimulaMath-vXXX-linux-XXX.run** in your terminal. You may be asked your password. If so, simply enter it and validate.

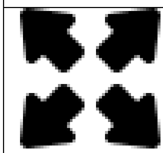
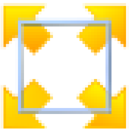
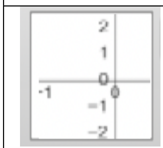
Chapter 3

User interface

3.1 Menus

List of menus	List of sub-menus
Home page	Home page: list of available areas in SimulaMath
Calculus & Algebra	• Solving equations: For solving equations, inequations, systems of equations, differential equations, recurring sequences. • Linear algebra: Operations over matrices, reduction of matrices, etc. • Operations on numbers: Basic operations, complex numbers, etc. • Operations on functions and sequences: Calculation of derivatives, limits, integrals, numerical sequences, etc. • Operations on finite fields and polynomials mod p. • Operations on Groebner bases and Multivariate polynomials: Determination of a groebner basis (reduced or not), calculation of S-polynomial, ideals, normal form etc.
2D & 3D graphics	• 2D graphics: Functions of one variable x, parametric functions, implicit functions, etc • 3D graphics: Functions of two variables x, y and parametric functions. • Diagrams: 2D and 3D bar charts, 2D and 3D histograms, 2D curves, 2D scatterplots, pie charts, box plot, Violin plot, Displot etc.
Probability	• Probability calculation: Discrete and continuous probability distributions. • Approximation of laws: Convergence in distributions.
Statistics	• Univariate descriptive statistics: Characteristics with central tendencies, characteristics of dispersions and shapes. • Bivariate descriptive statistics: Contingency chi-square, covariance, Stchuprow's T, etc. • Inferential statistics : Estimation by confident interval and hypothesis testing. • Spreadsheet: Open and save files in excel, csv and json formats; operations on a table.
Cryptography	• Euclidean lattices: Simulation of a Euclidean lattices in *2* dimension. • Coding Theory: Linear codes, binary Hamming codes, operations on linear codes. • Elliptic curves: Weierstrass short form, Montgomery

3.2 Toolbar

Image	For	Description
	Area 2D and 3D	Save the figure in png, pgf, pdf, eps, jpeg, ep, etc
	Area 2D	Go back
	Area 2D	Redo
	Area 2D	Move figure
	Area 2D	zoom on the figure
	Area 2D	Home
	Area 2D	position and display of the axes of the figure (axes that intersect in zero, axes at the ends, without axes etc.)
	2D and 3D areas	more parameters (settings, grid, axes, orthonormal marker, theme of the graph, adding images, points, segments, circles, annotations, texts, etc.)
	All pages	Help

Chapter 4

Clickable Interface

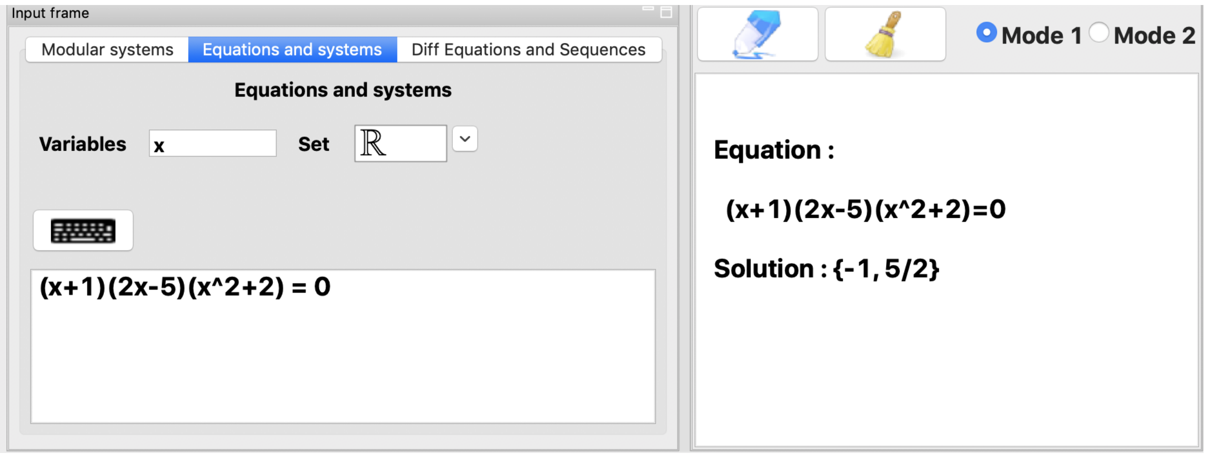
4.1 Solving Equations, Inequations and Systems.

4.1.1 Equations and Inequations

To solve an equation (or an inequation) with **SimulaMath**:

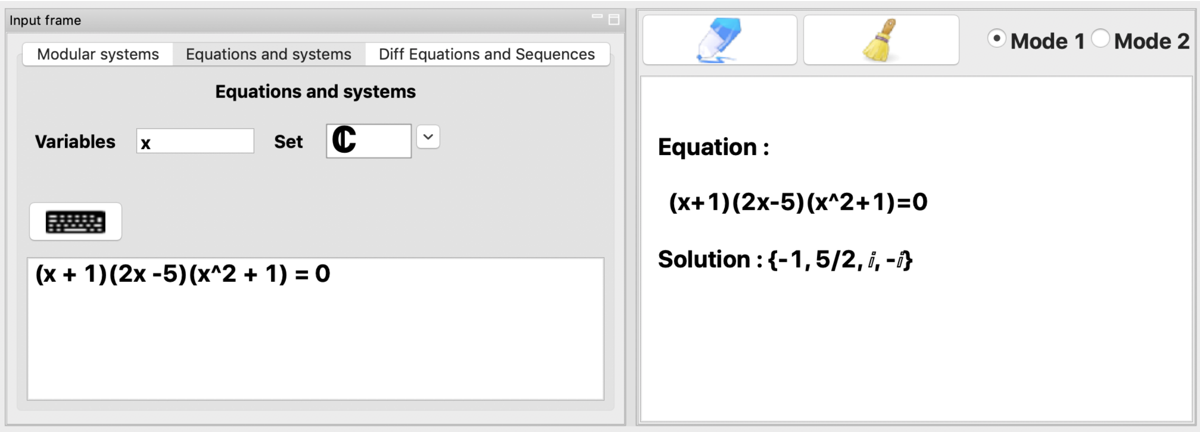
- Enter the equation (or inequation) in the left panel,
- Specify the variable(s) in the variables area,
- Then click on the display button.
- The solution to the equation $(x+1)(2x-5)(x^2+1) = 0$ over $\mathbb{R}$.

Note: You must specify the variables in the variables area



Note: You can also solve equations in a given set.

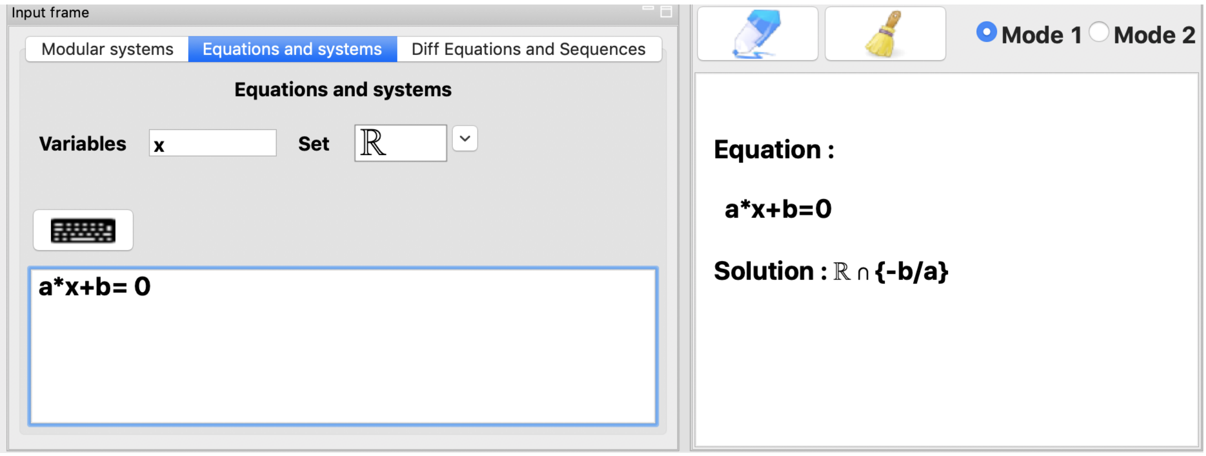
- Solving the equation $(x+1)(2x-5)(x^2+1) = 0$ over $\mathbb{C}$



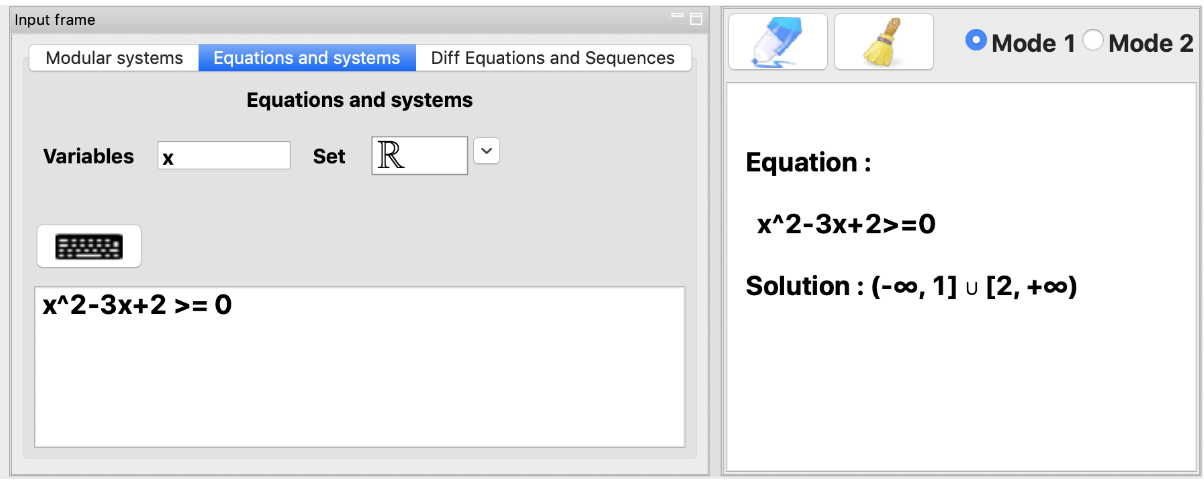
- Resolution with parameters

Note: If there are variables in the equation that are not defined in the variables area, then they are considered as parameters.

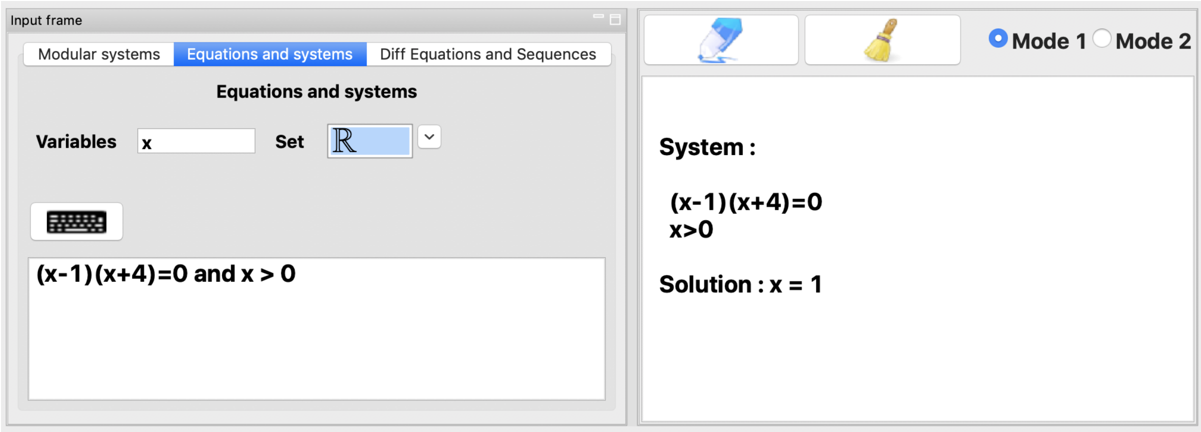
Let us solve the equation $ax + b = 0$ when $x \in \mathbb{R}$.



- The solution of the inequation $x^2 - 3x + 2 \geq 0$ in $\mathbb{R}$



- The resolution in $\mathbb{R}$ of $(x - 1)(x + 4) = 0$ and $x > 0$.



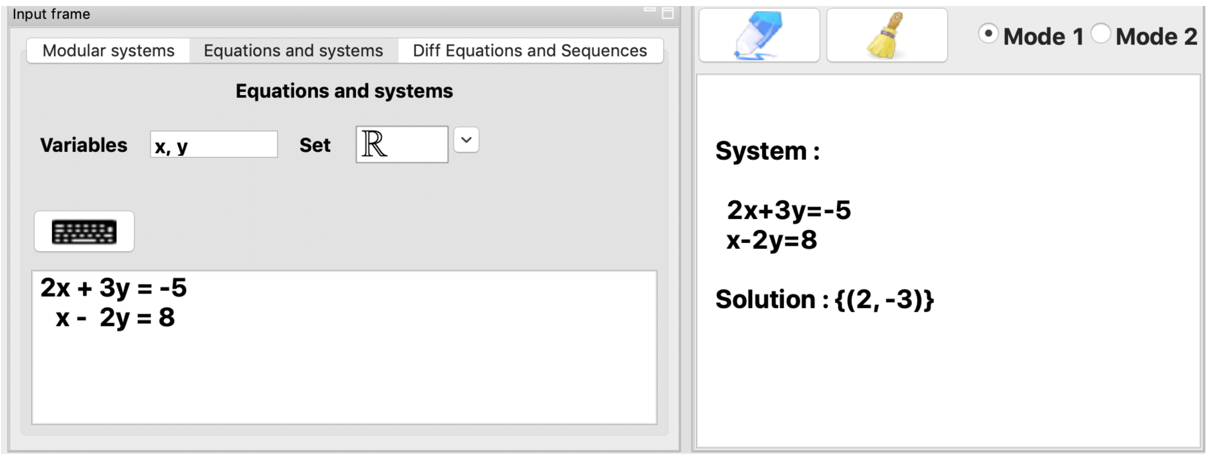
4.1.2 Linear and non-linear systems

To enter a system (or an inequation),

1. First enter the first equation (or inequation) and then press the **ENTER** key;
2. Then enter the second equation (or inequation) and press **ENTER**;
3. And so on until the last equation (or inequation).

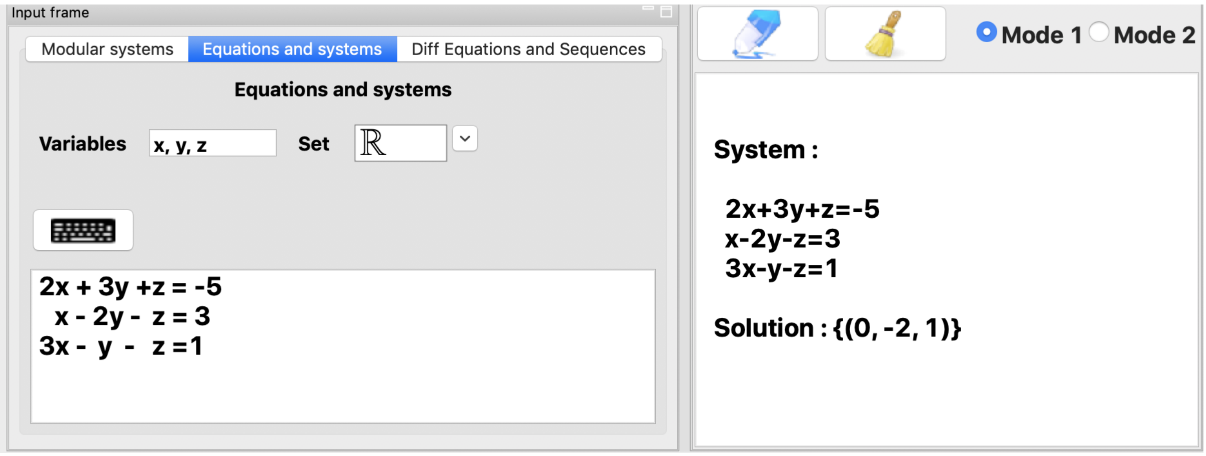
- The solution in $\mathbb{R}^2$ of the system:

$$\begin{cases} 2x + 3y &= -5 \\ x - 2y &= 8 \end{cases}$$



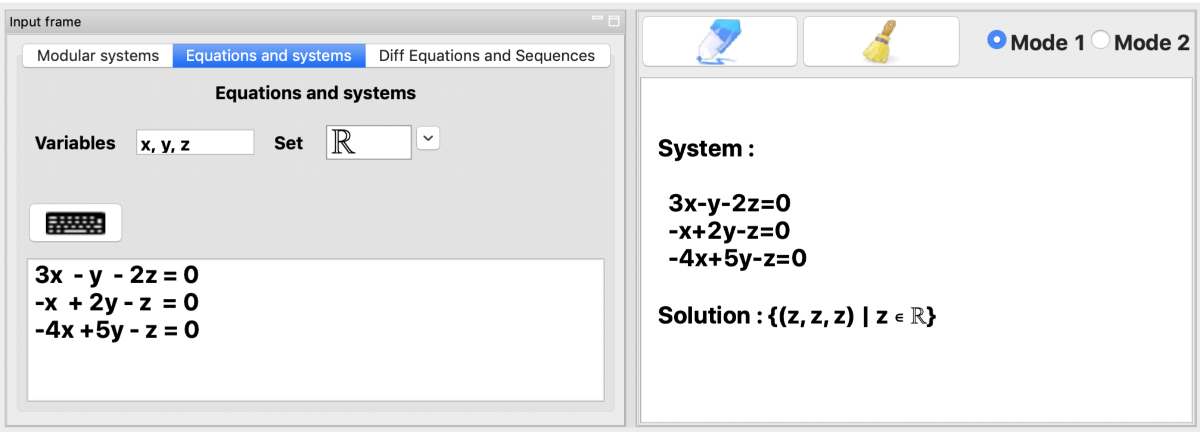
- The resolution in $\mathbb{R}^3$ of the system:

$$\begin{cases} 2x + 3y + z &= -5 \\ x - 2y - z &= 3 \\ 3x - y - z &= 1 \end{cases}$$



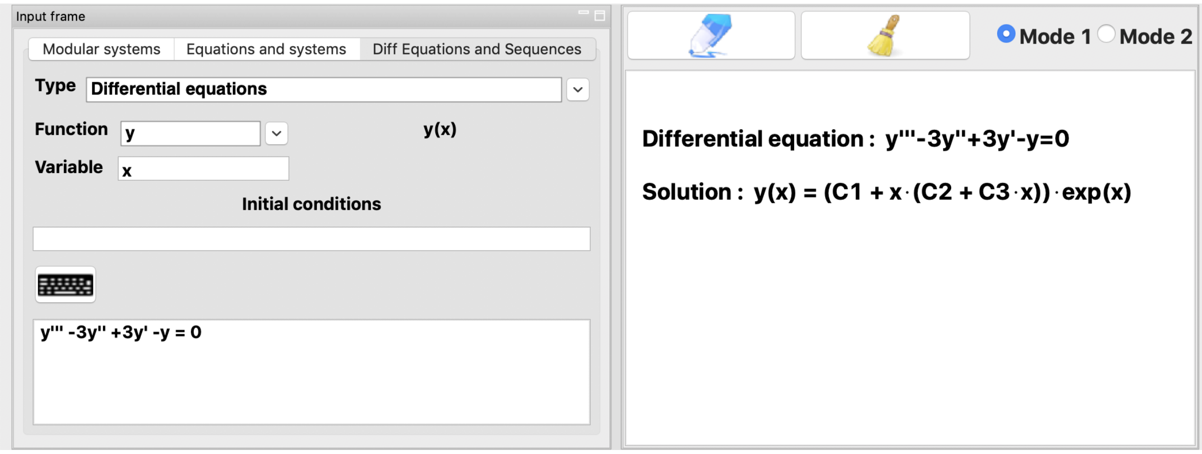
- The resolution in $\mathbb{R}^3$ of the system:

$$\begin{cases} 3x - y - 2z &= 0 \\ x + 2y - z &= 0 \\ -4x + 5y - z &= 0 \end{cases}$$

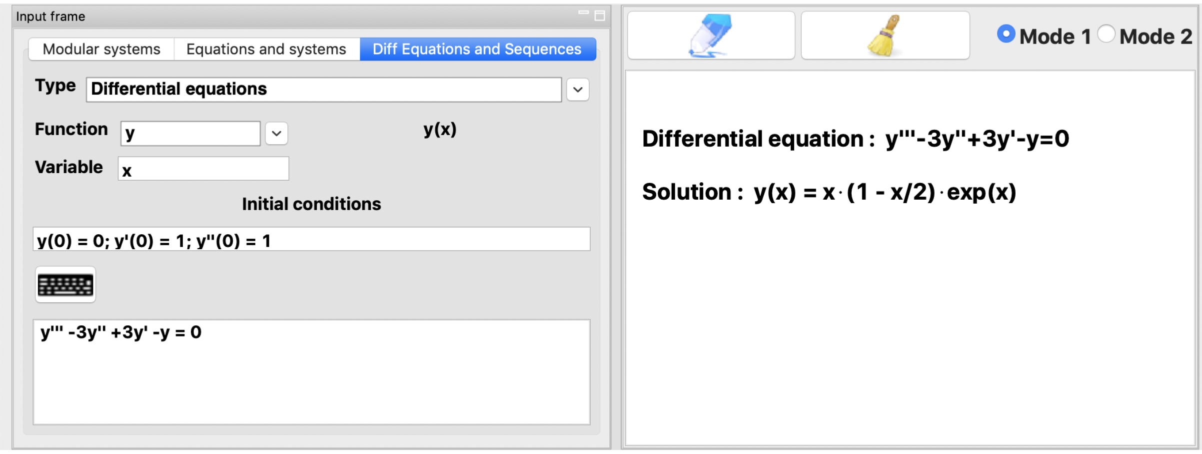


4.1.3 Differential Equations

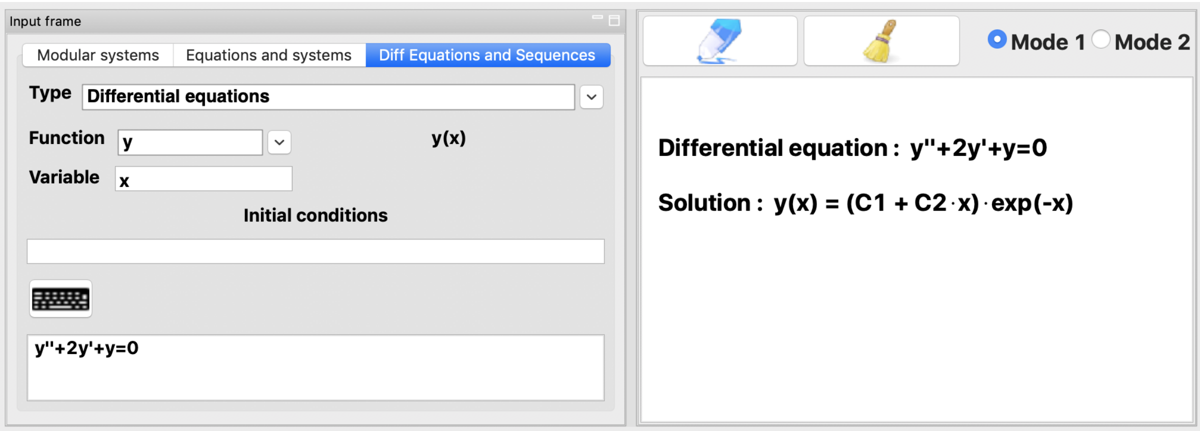
- The resolution of the differential equation $y''' - 3y'' + 3y' - y = 0$



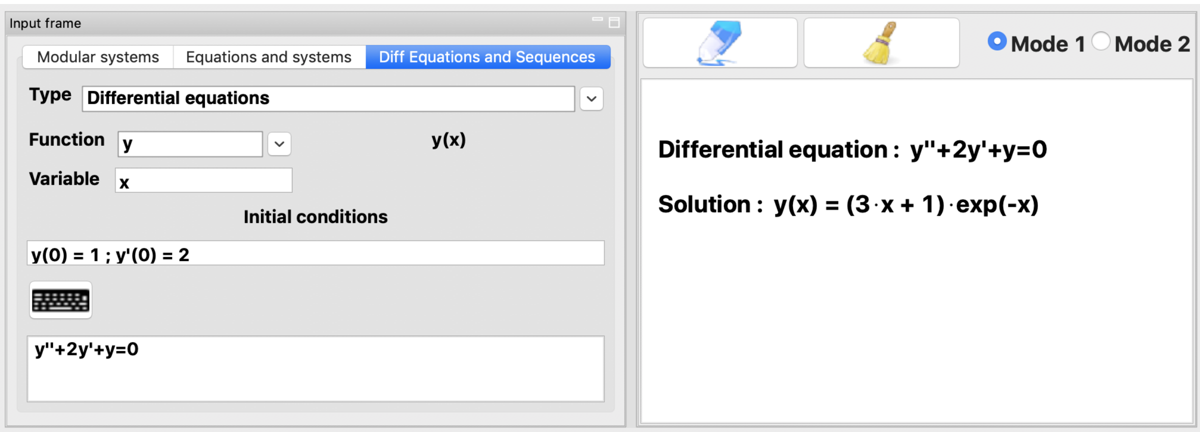
- The resolution of the differential equation $y''' - 3y'' + 3y' - y = 0$ with the initial conditions $y(0) = 0; y'(0) = 1$ and $y''(0) = 1$.



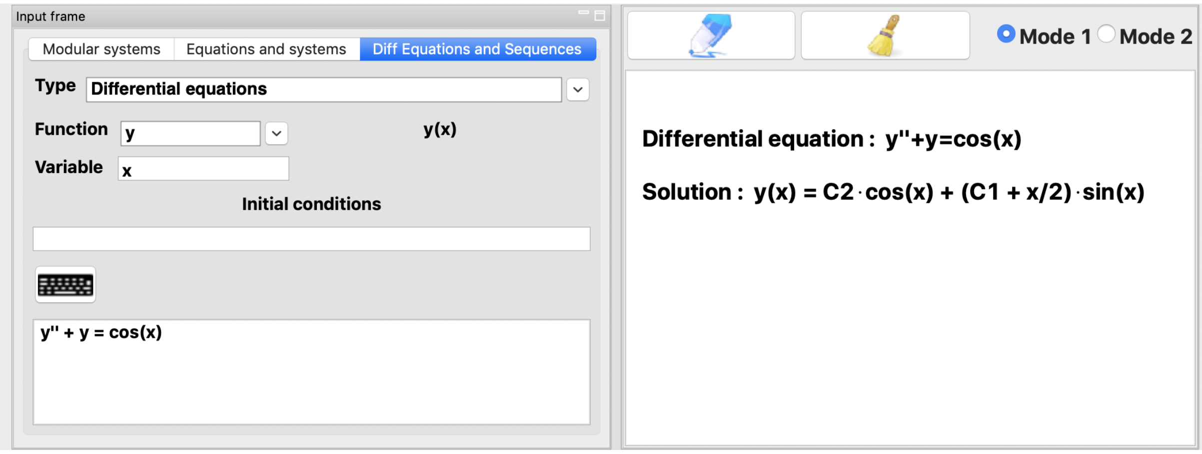
- The resolution of the differential equation $y'' + 2y' + y = 0$.



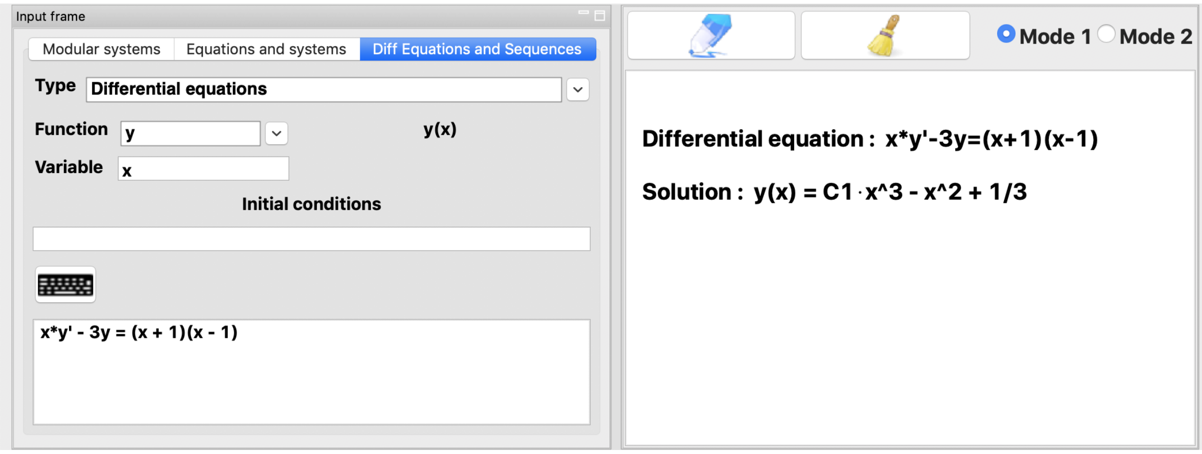
- The solution of the differential equation $y'' + 2y' + y = 0$ with the initial conditions $y(0) = 1$ and $y'(0) = 2$.



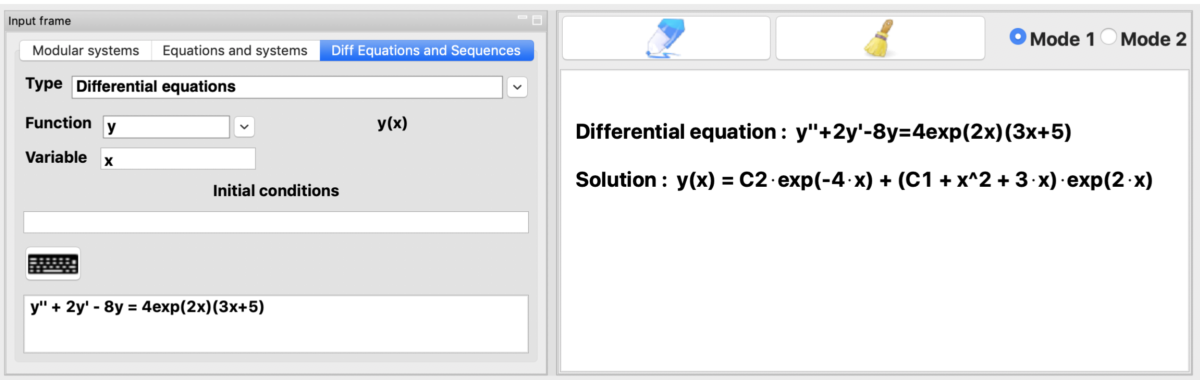
- The solution of the differential equation $y'' + y = \cos(x)$



- The solution of the differential equation $xy' - 3y = (x+1)(x-3)$



- Solving the differential equation $y'' + 2y' - 8y = 4exp(2x)(3x+5)$

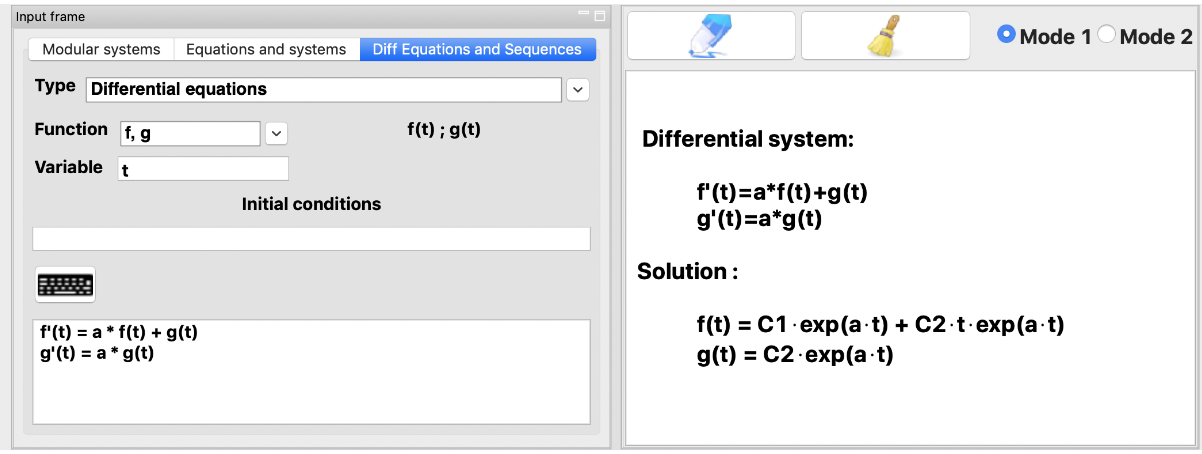


4.1.4 Differential Systems

- The resolution of the differential system

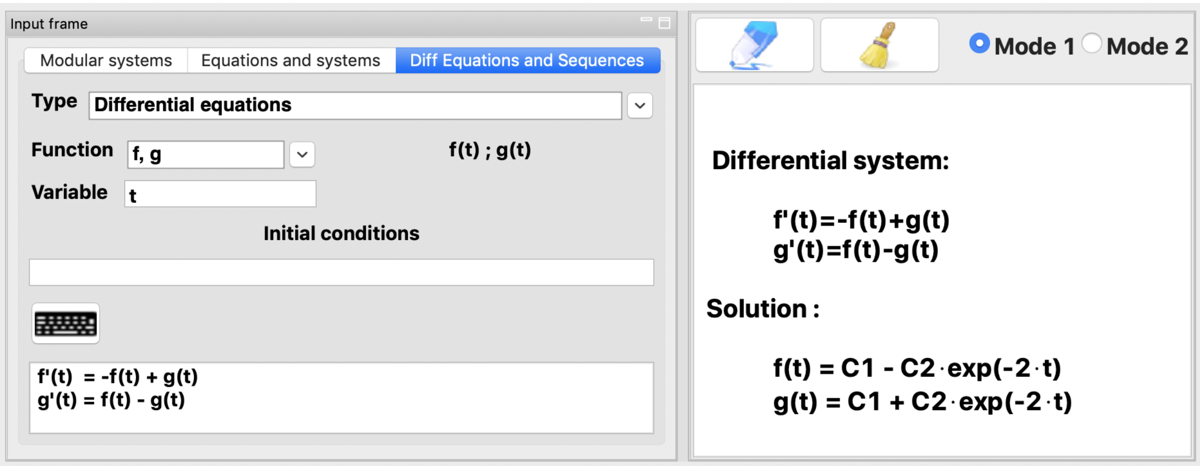
$$\begin{cases} f'(t) &= af(t) + g(t) \\ g'(t) &= ag(t) \end{cases}$$

with $a \in \mathbb{R}$.



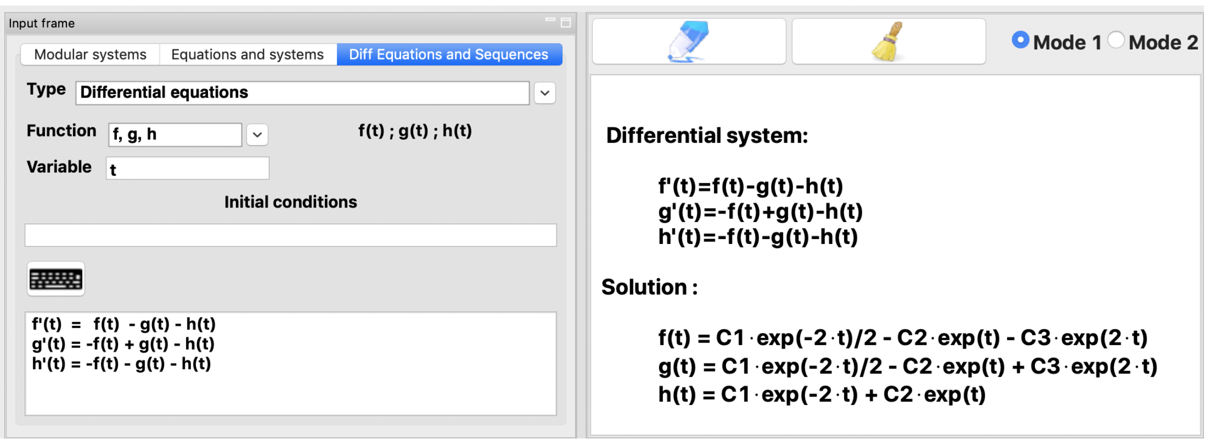
- The resolution of the differential system

$$\begin{cases} f'(t) &= -f(t) + g(t) \\ g'(t) &= f(t) - g(t) \end{cases}$$



- The resolution of the differential system

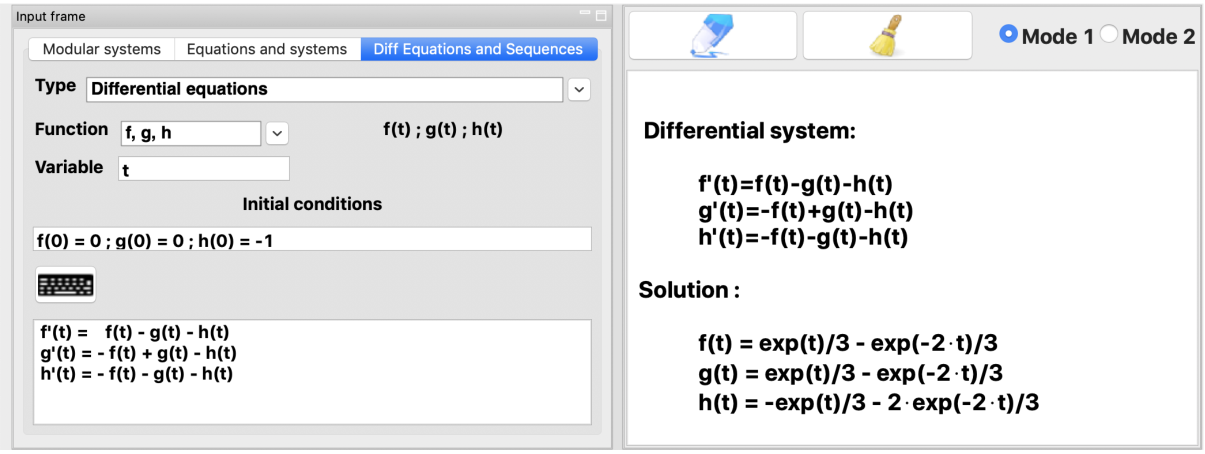
$$\begin{cases} f'(t) &= f(t) - g(t) - h(t) \\ g'(t) &= -f(t) + g(t) - h(t) \\ h'(t) &= -f(t) - g(t) + h(t) \end{cases}$$



- The resolution of the differential system :

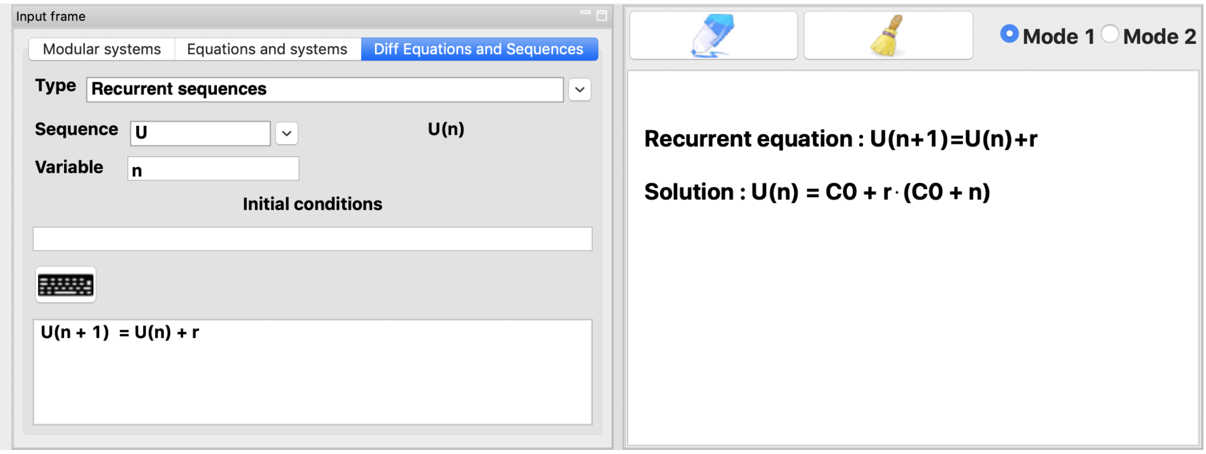
$$\begin{cases} f'(t) & = f(t) - g(t) - h(t) \\ g'(t) & = -f(t) + g(t) - h(t) \\ h'(t) & = -f(t) - g(t) + h(t) \end{cases}$$

with initial conditions $f(0) = 0; g(0) = 1$ and $h(0) = -1$.

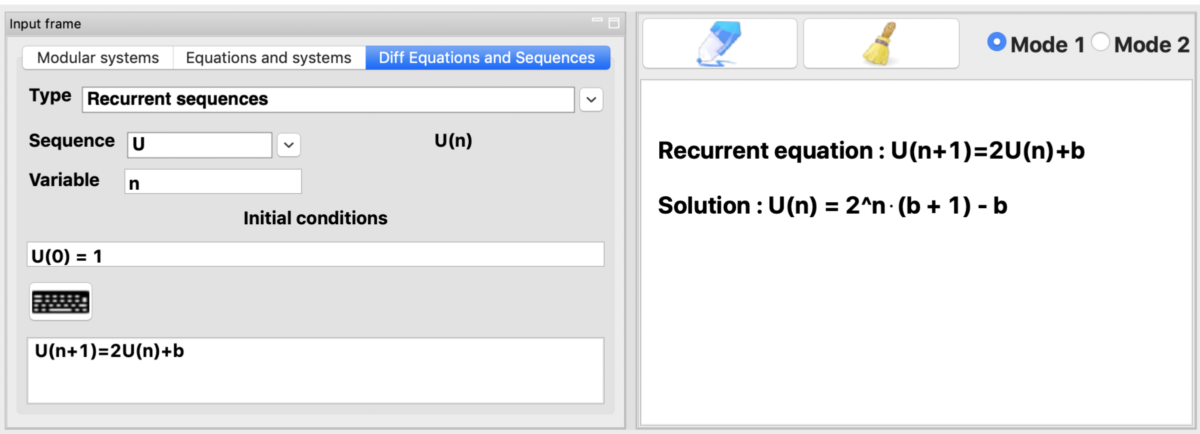


4.1.5 Recurrent Sequences

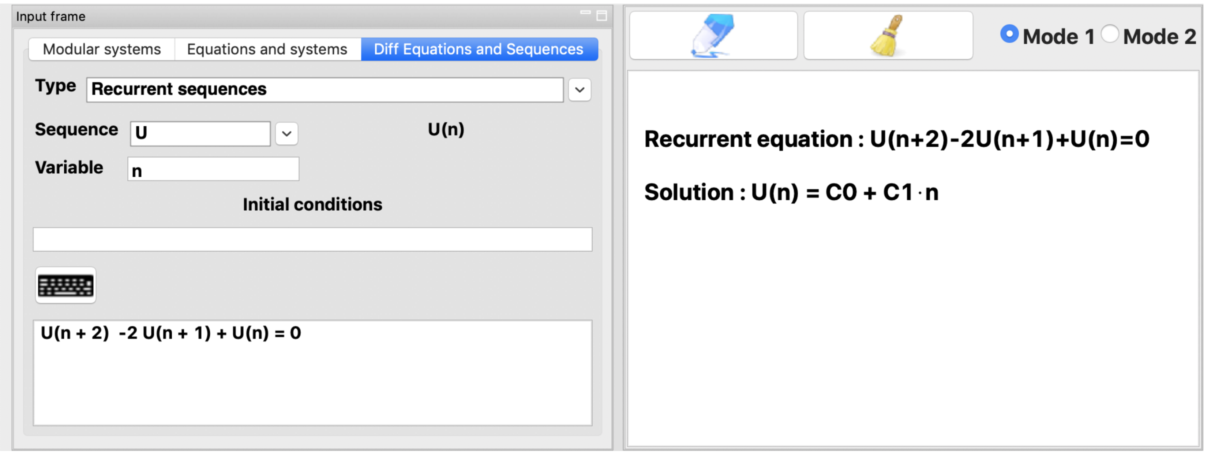
- The solution of the recurrent equation $U(n+1) = U(n) + r$ with $r \in \mathbb{R}$.



- The solution of the recurrent equation $U(n+1) = 2U(n) + b$ with $b \in \mathbb{R}$ with $U(0) = 1$.



- The solution of the recurrent equation $U(n+2) - 2U(n+1) + U(n) = 0$.

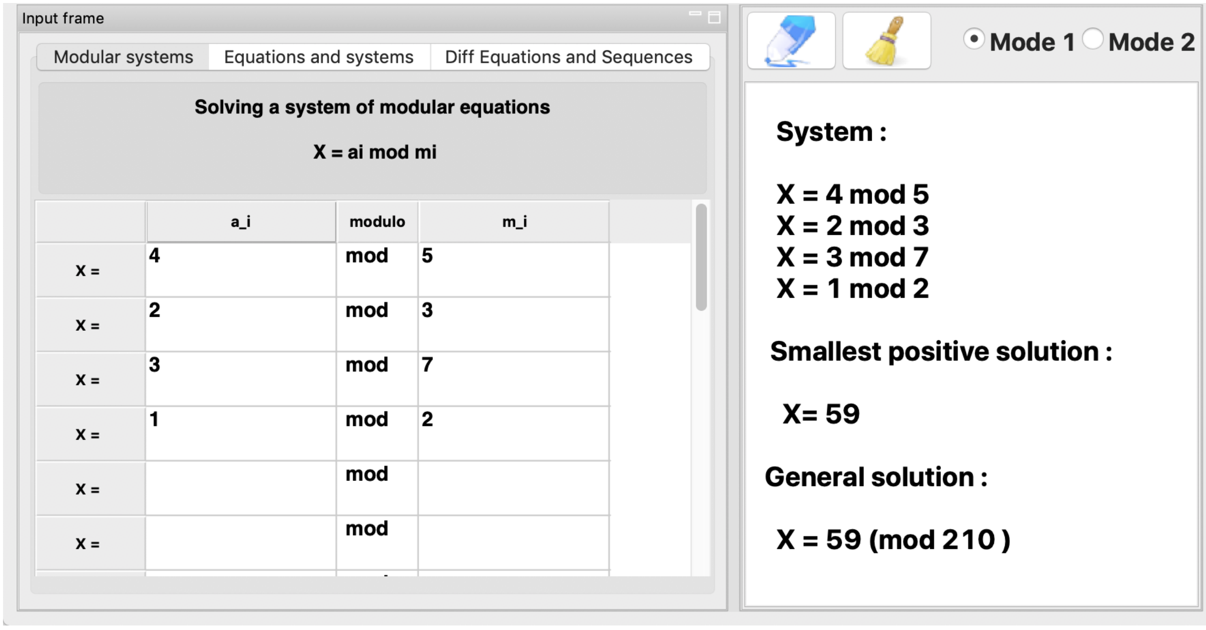


4.1.6 Modular Systems

The solution of some modular systems can be done by using the Chinese remainder theorem.

- Example: solving the modular system

$$\begin{cases} x & \equiv 4 \bmod 5 \\ x & \equiv 2 \bmod 3 \\ x & \equiv 3 \bmod 7 \\ x & \equiv 1 \bmod 2 \end{cases}$$



4.2 Univariate descriptive statistics

For univariate descriptive statistics, the following characteristics can be determined

- the mean
- the quadratic mean
- the geometric mean
- the harmonic mean
- the variance
- the empirical variance
- the moment of order alpha
- the standard deviation
- the empirical standard deviation
- the mode (and the modal class in the case of a continuous characteristic)
- the median (and the median class in the case of a continuous characteristic)
- the quartiles Q1 and Q3
- the absolute mean deviation
- the median absolute range
- the inter-quartile range
- the coefficient of variation
- the coefficient of skewness
- Fisher's coefficient of skewness
- Yule's coefficient of skewness
- Pearson's coefficient of skewness
- Pearson's coefficient of kurtosis
- Fisher's kurtosis coefficient

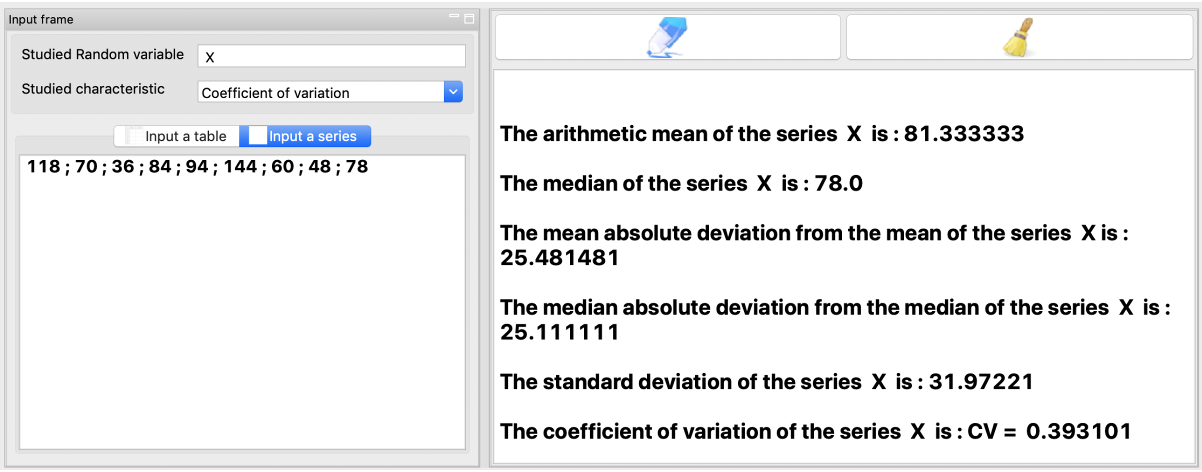
Data entry is very simple as shown in the examples below.

4.2.1 Discrete Variables

Example 1: Statistical series

The following series represents the area (in m^2) of the nine apartments in a residence: 118 ; 70 ; 36 ; 84 ; 94 ; 144 ; 60 ; 48 ; 78

1. Determine the arithmetic mean and median of this distribution.
2. Calculate the following dispersion characteristics: mean absolute deviation from the mean and median, the standard deviation and the coefficient of variation.

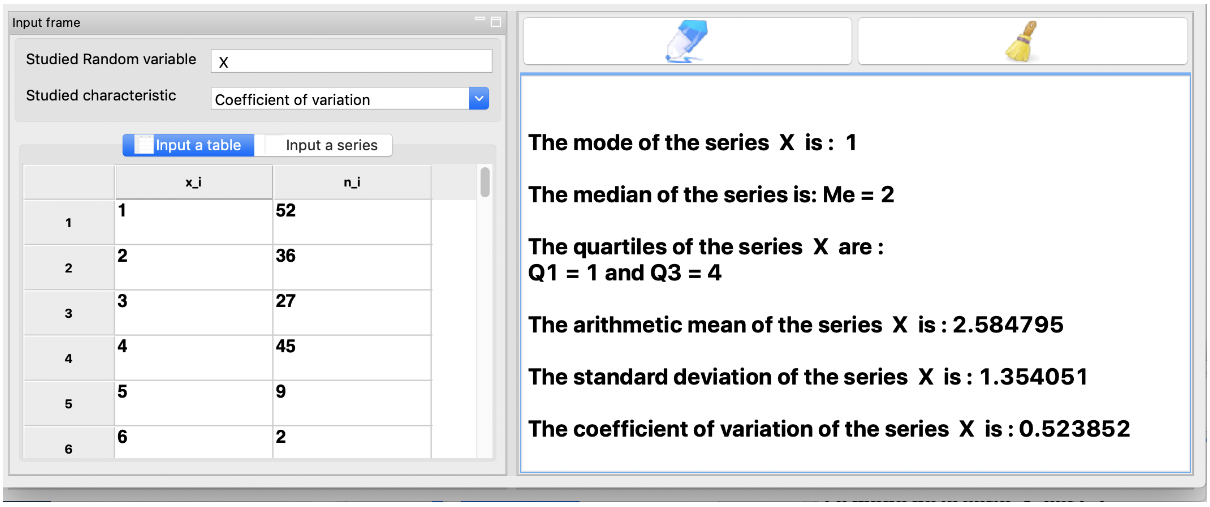


Example 2: Statistical series in table form.

In a bookstore, 180 authors have been divided according to the number of textbooks they have written.

x_i	1	2	3	4	5	6	7
n_i	52	36	27	45	9	2	9

1. Determine the mode, median and quartiles Q_1 and Q_3.
2. Calculate the arithmetic mean, standard deviation and coefficient of variation of this series.



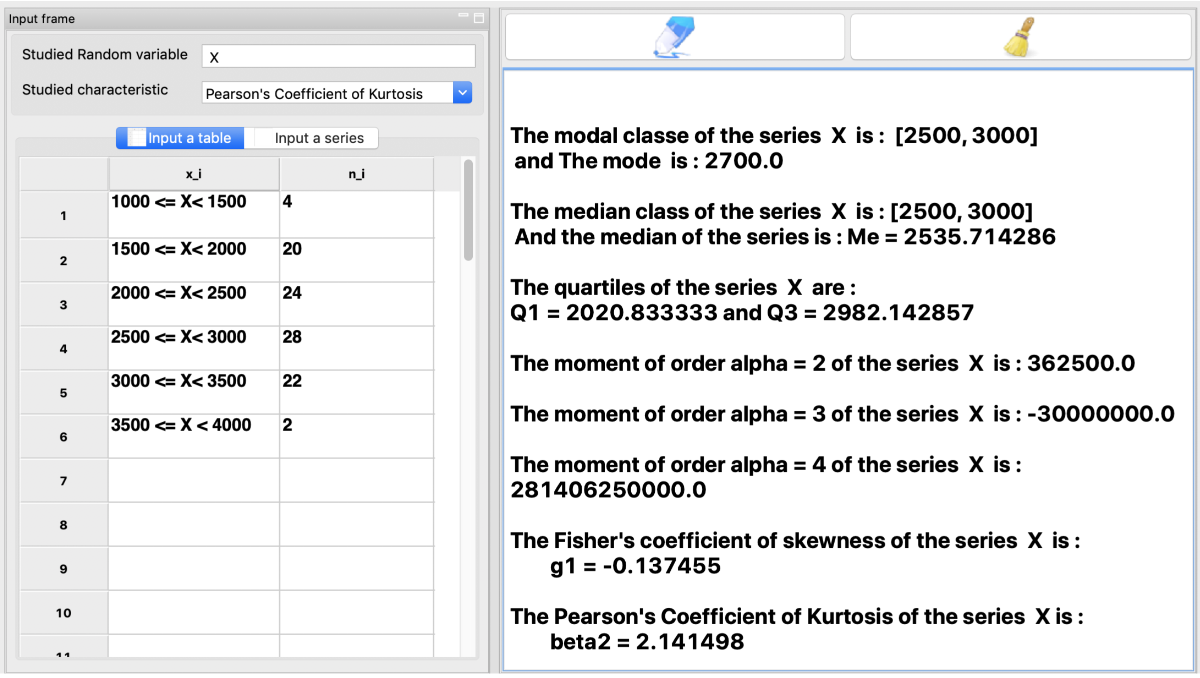
4.2.2 Continuous Variables

Example 3: Data grouped in classes of equal magnitude

The table below gives the distribution of the number of orders as a function of the amount of orders X, for the last six months of GIE LIGGEEY.

X	$1000 \leq X < 1500$	$1500 \leq X < 2000$	$2000 \leq X < 2500$	$2500 \leq X < 3000$	$3000 \leq X < 3500$	$3500 \leq X < 4000$
Values	4	20	24	28	22	2

1. Determine the modal class, mode, median and quartiles Q_1 and Q_3.
2. Compute the centered moments of order $2, 3$ and 4 of this distribution.
3. Calculate the Fisher skewness coefficient and the Pearson kurtosis coefficient.

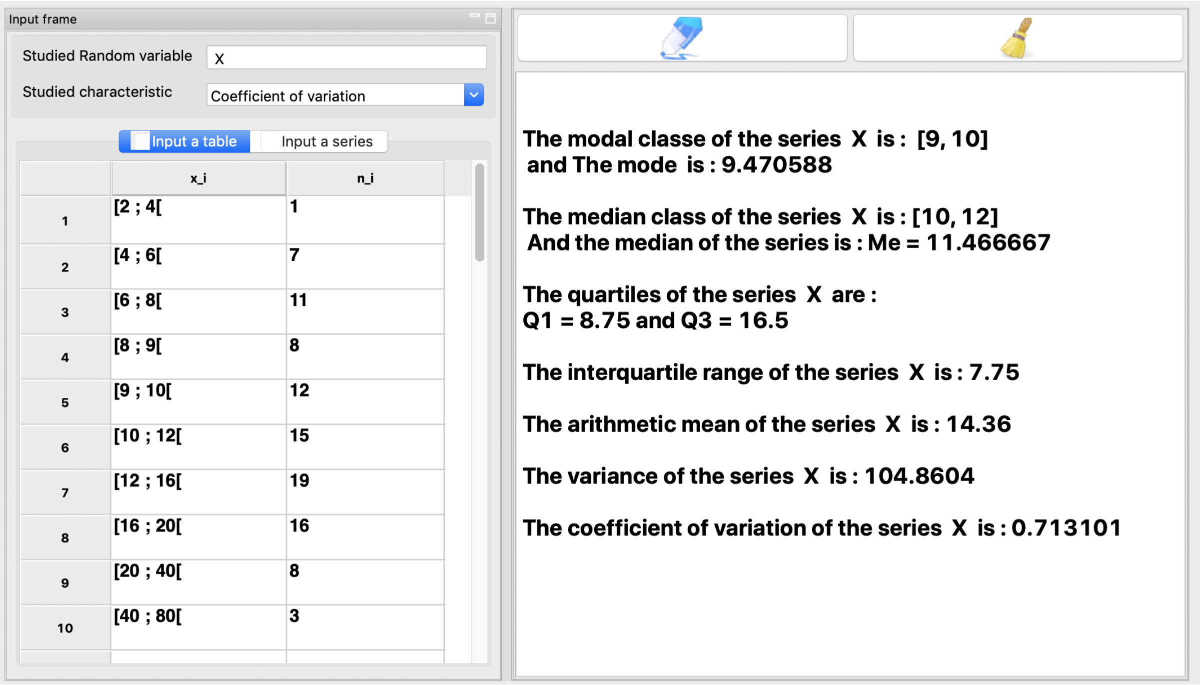


Example 4: Data grouped into classes of unequal magnitude

The table below provides the percentage distribution of a municipality's inhabitants according to the annual amount of their local taxes (in thousands of dollars)nof their local taxes (in thousands of francs).

Classes	[2 ; 4[	[4 ; 6[	[8 ; 9[	[9 ; 10[	[10 ; 12[	[12 ; 16[	[16 ; 20[	[20 ; 40[	[40 ; 60[	[60 ; 80[
Values	1	7	11	8	12	15	19	16	8	3

1. Determine the modal class, mode, median and quartiles Q_1 and Q_3.
2. Calculate the arithmetic mean of this series, the interquartile range, the variance and the coefficient of variation.



4.3 Bivariate descriptive statistics

For bivariate descriptive statistics, the following characteristics can be determined

- the covariance
- the linear correlation coefficient
- the coefficient of determination
- the chi-square distance
- Cramer's phi square
- Tschuprow's T
- the regression line

Data entry is very simple as shown in the examples below.

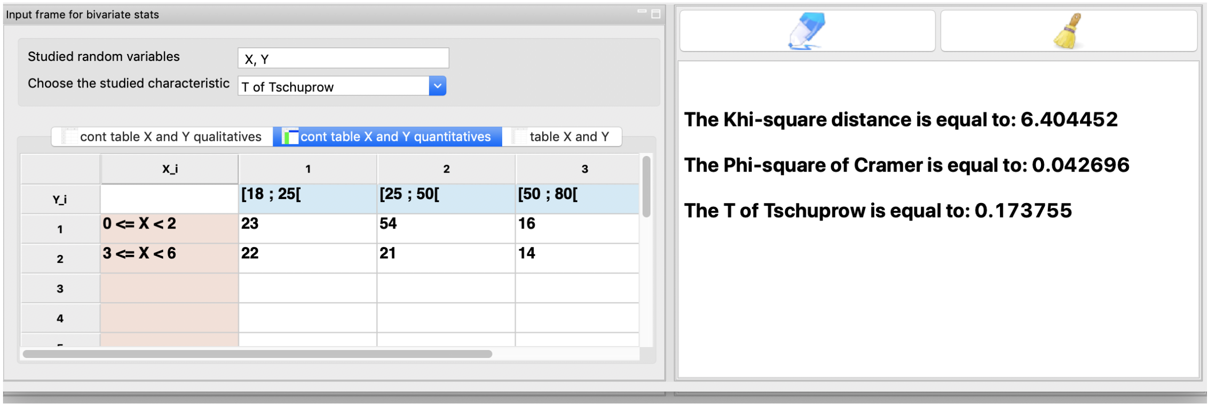
4.3.1 Contingency table

Example 1 : Contingency table with X and Y **quantitative**

An insurance company has carried out a sample survey from its customer file to know the distribution of the number of road accidents **(X)** according to the age of the insured **(Y)**. The result of this survey is given by the following table.

		Age (in years)	
Number of accidents	[18 ; 25[	[25 ; 50[	[50 ; 80[
from 0 to 2	23	54	16
from 3 to 6	22	21	14

1. Calculate the chi-square of contingency.
2. Deduce the values of Cramer's Φ^2 and Tschuprow's T.

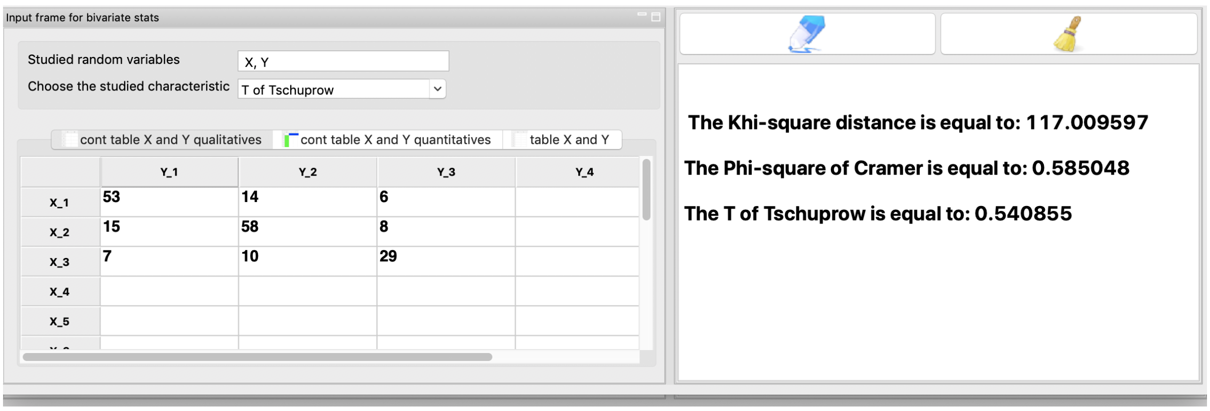


Example 2 : Contingency table with X and Y **qualitatives**

In a sample of 200 randomly selected households, the average propensity to save **(variable Y)** as a function of disposable income **(variable X)**. For the **X** variable, we distinguished 3 classes (low income, intermediate income, high income). Similarly, savings rates were classified into 3 levels (low rates, intermediate rates, high rates). The results are presented in the contingency table :

	$Y_1 =$ low rates	$Y_2 =$ intermediate rates	$Y_3 =$ high rates
$X_1 =$ low income	53	14	6
$X_3 =$ middle income	15	58	8
$X_3 =$ high income	7	10	29

1. Compute the Chi-square of contingency.
2. Deduce the values of Cramer's Φ^2 and Tschuprow's T.

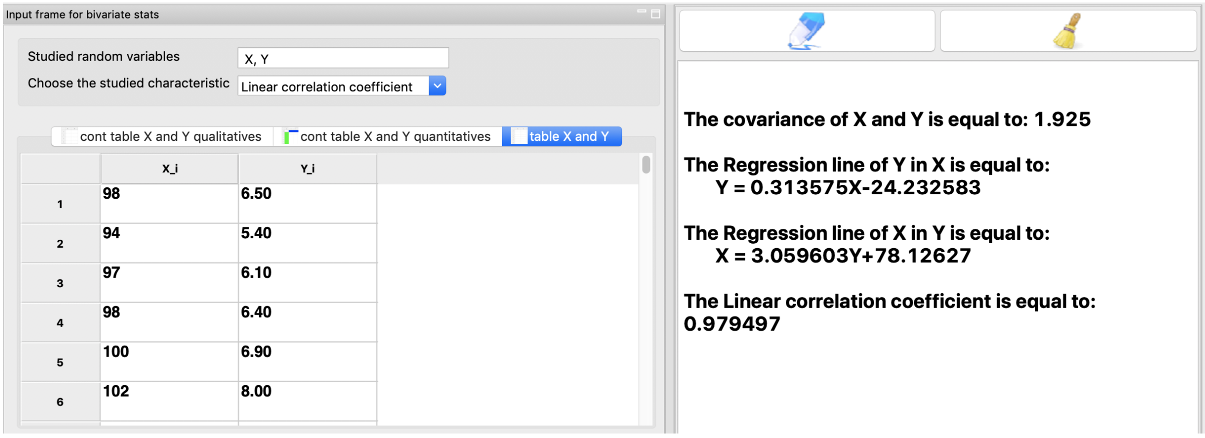


4.3.2 Simple table

The stock corporation R increases its capital. The stock market price of the share (X) has been raised for 6 days of the share (X) and of the subscription right (Y).

X	98	94	97	98	100	102
Y	6.50	5.40	6.10	6.40	6.90	8.00

1. Compute the covariance *Cov(X, Y)*.
2. Establish the equations of the regression line of Y with respect to X.
3. Calculate the linear correlation coefficient between the variables X and Y.



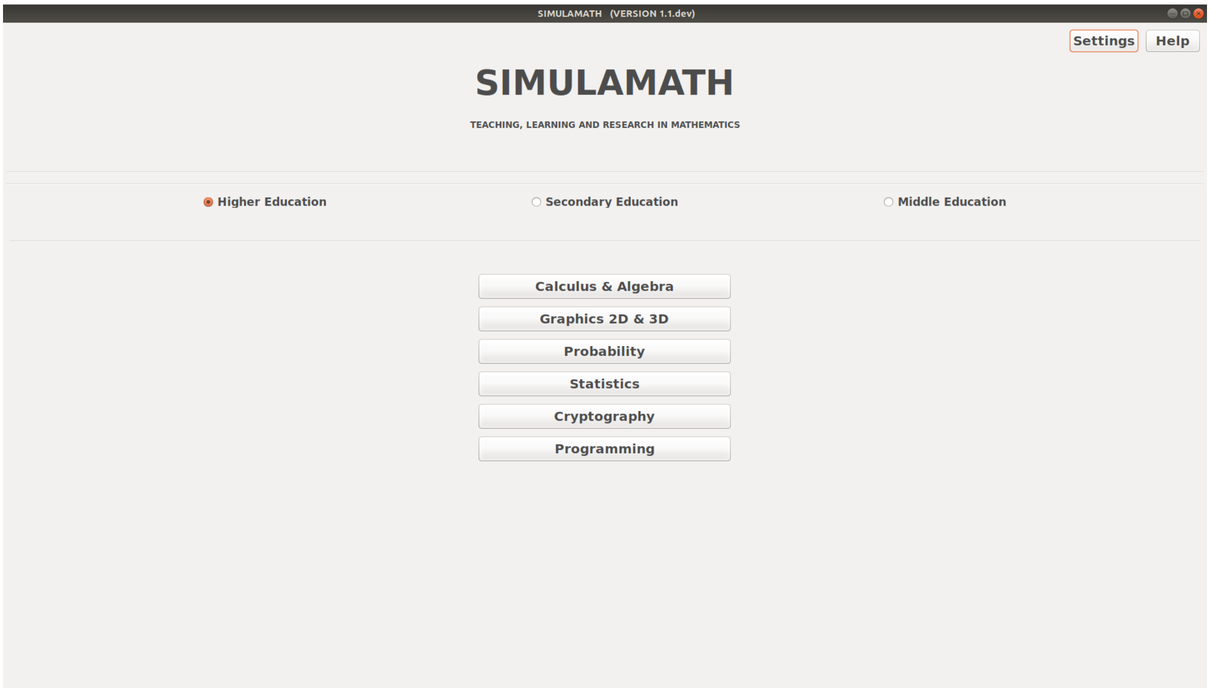
4.4 Inferential statistics

Inferential statistics consist in making inferences (generalizations) about the population based on information from sample(s). Simulamath offers you the implementation of hypothesis testing and confidence interval estimation.

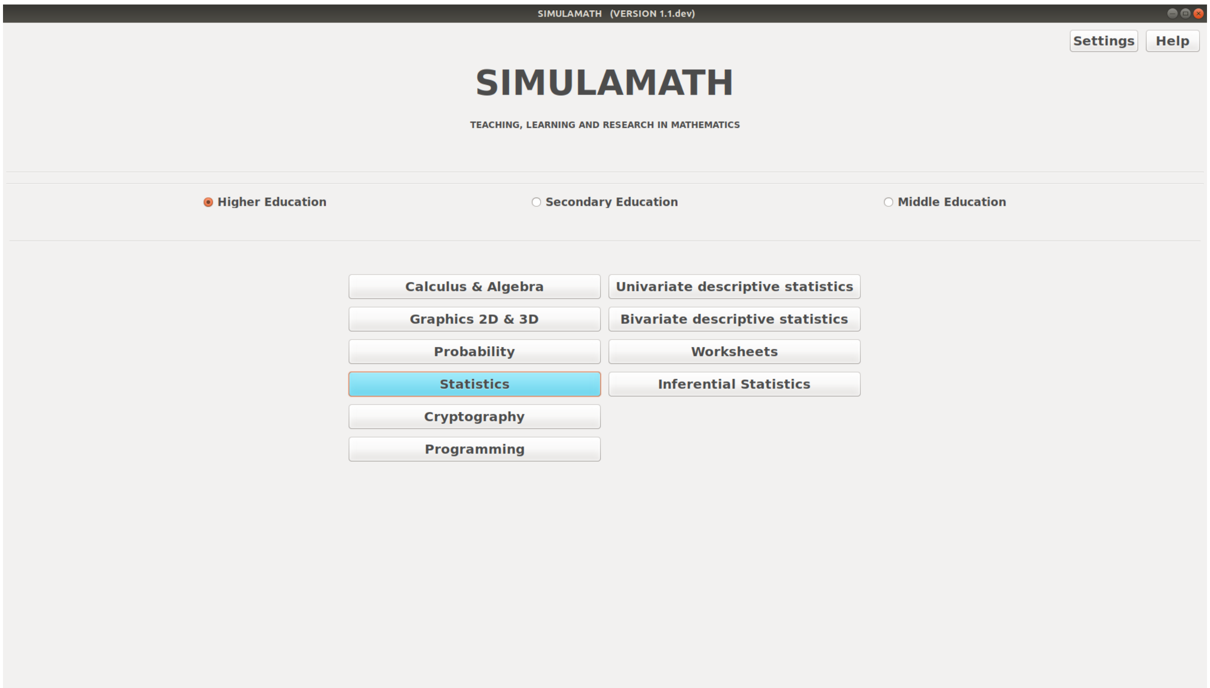
The examples in this section comes from the book *Elementary Statistics, A step by step approach, 8th Edition*, by Professor Allan G. Bluman

4.4.1 How to get to Inferential statistics' section in Simulamath

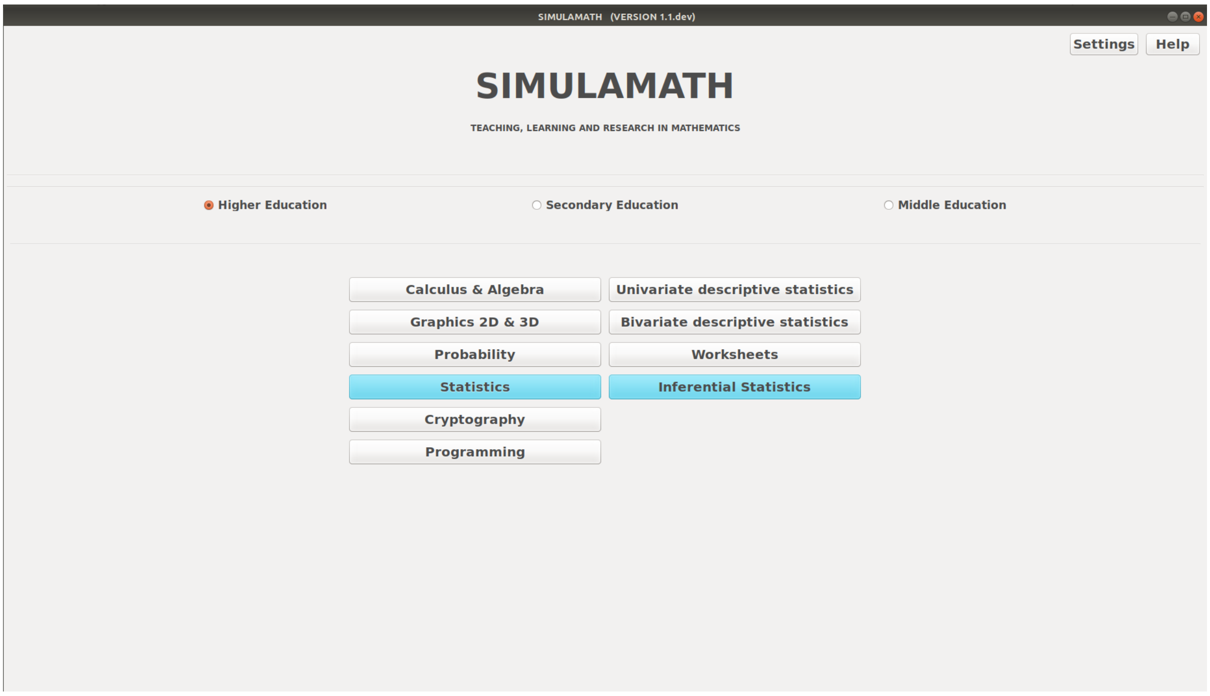
In the image below is the **home page** of Simulamath



Click on **Statistics** (just posing the cusor without clicking is also enough)



Then you can choose/click your theme of interest, here **Inferential Statistics**.



4.4.2 Confidence Intervals Estimation

Z-Estimation for Mean

The formula to get the interval confidence for Z-Estimation for Mean is:

$$\bar{X} - z_{\alpha/2}\left(\frac{\sigma}{\sqrt{n}}\right) < \mu < \bar{X} + z_{\alpha/2}\left(\frac{\sigma}{\sqrt{n}}\right)$$

Example:

A survey of 30 emergency room patients found that the average waiting time for treatment was 174.3 minutes. Assuming that the population standard deviation is 46.5 minutes, find the best point estimate of the population mean and the 99% confidence of the population mean.

Solution:

The best point estimate is 174.3 minutes. The 99% confidence is interval is

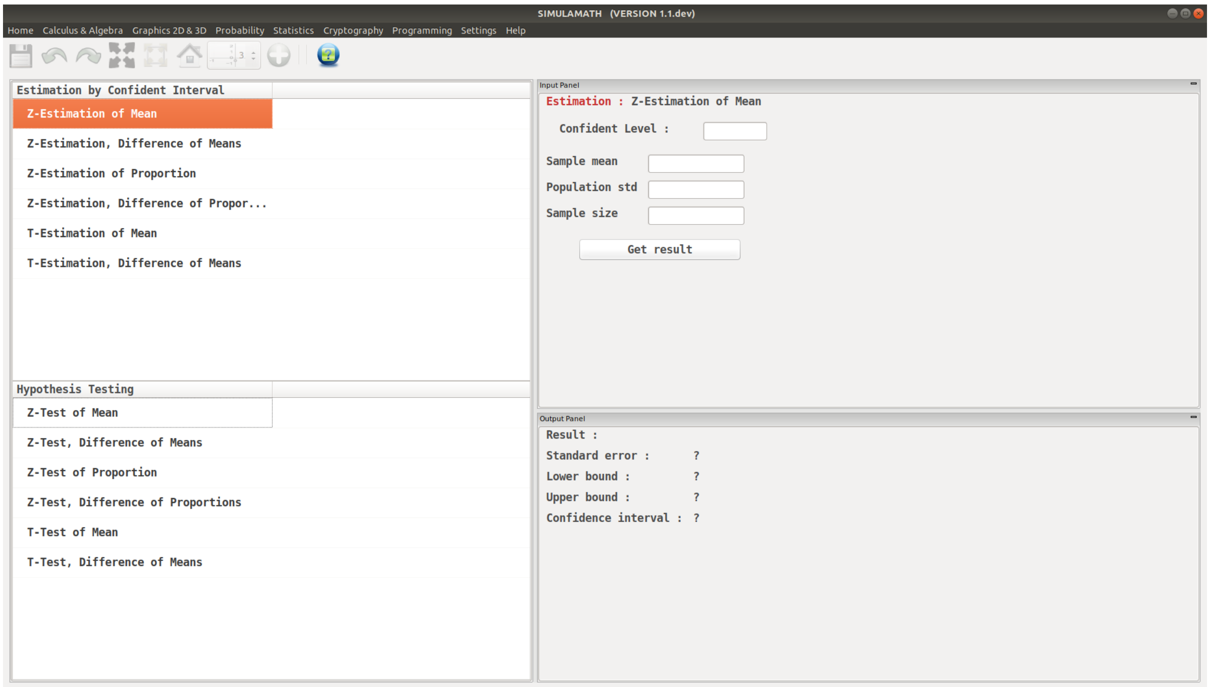
$$174.3 - 2.58\left(\frac{46.5}{\sqrt{30}}\right) < \mu < 174.3 + 2.58\left(\frac{46.5}{\sqrt{30}}\right)$$

$$152.4 < \mu < 196.2$$

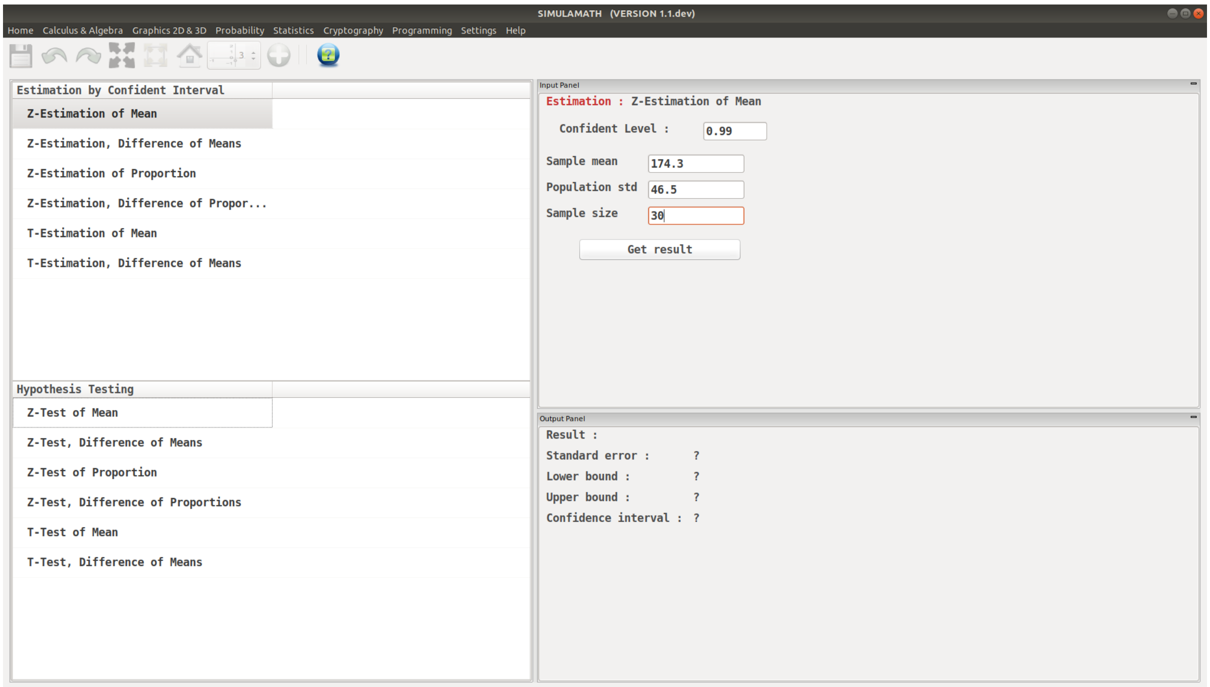
Hence, one can be 99% confident that the mean waiting time for emergency room treatment is between 152.4 and 196.2 minutes.

Z-Estimation for Mean in Simulamath

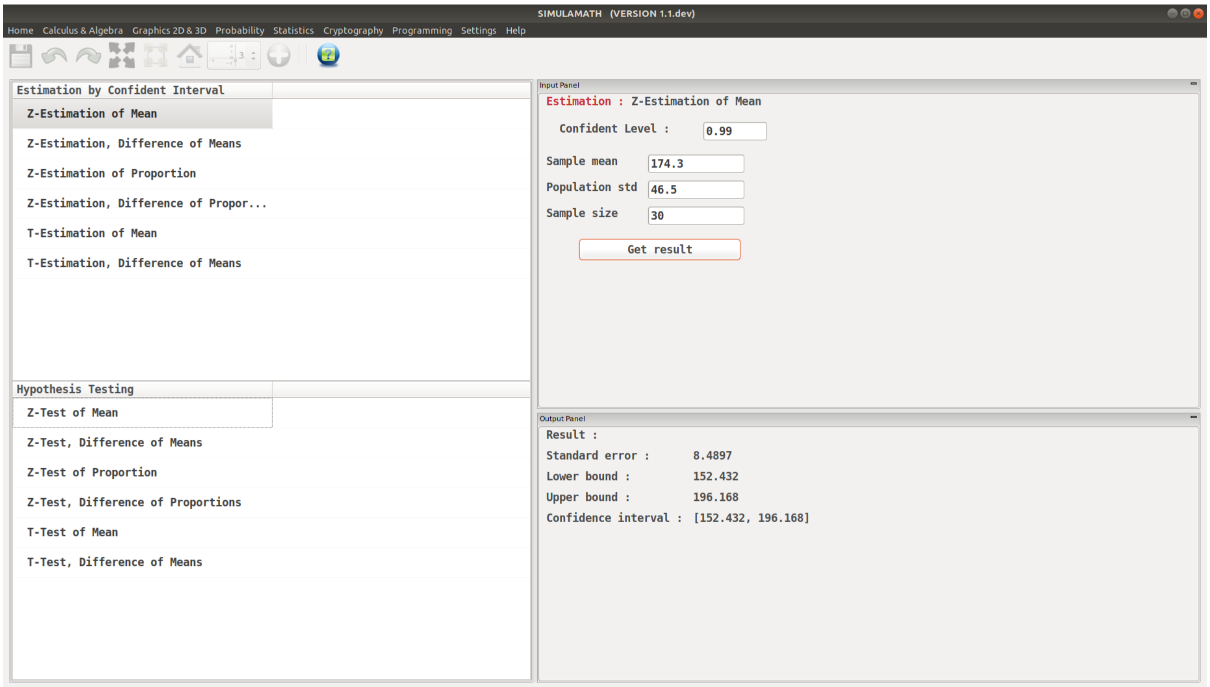
Choose the type of test/estimation you want to compute in the panels on the left hand side, here **Z-Estimation for Mean**



Enter the variables (Confidence level, sample mean, standard deviation of the population, sample size) in the top panel on the right hand side.



Click on **Get result** located below inside the same panel. Voila, you have your results in the panel down on the right hand side.



Z-Estimation, Difference of Means

The formula to get the interval confidence for Z-Estimation, Difference of Means is:

$$(\bar{X}_1 - \bar{X}_2) - z_{\alpha/2}\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}} < \mu_1 - \mu_2 < (\bar{X}_1 - \bar{X}_2) + z_{\alpha/2}\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$$

Example:

A survey found that the average hotel room rate in New Orleans is \$88.42 and the average room rate in Phoenix is \$80.61. Assume that the data were obtained from two samples of 50 hotels each and that the standard deviations of the populations are \$5.62 and \$4.83, respectively. At $\alpha = 0.05$, can it be concluded that there is a significant difference in the rates?

Solution:

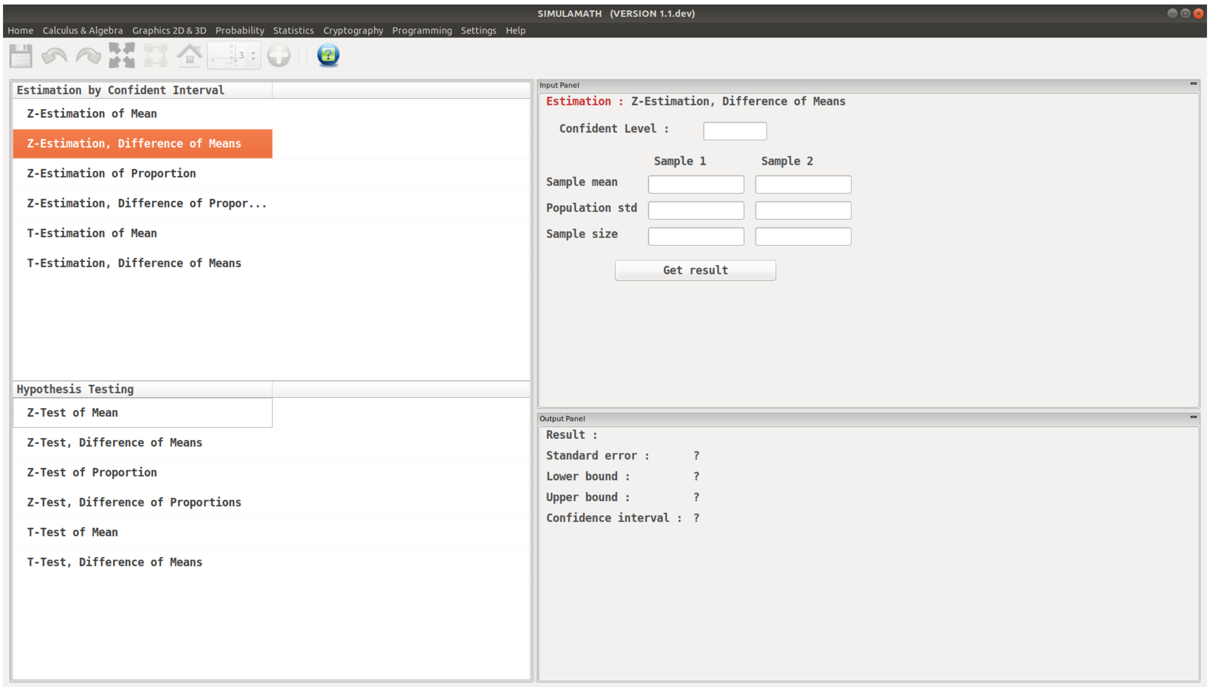
$$88.42 - 80.61) - 1.96\sqrt{\frac{5.62^2}{50} + \frac{4.83^2}{50}} < \mu_1 - \mu_2 < (88.42 - 80.61) + 1.96\sqrt{\frac{5.62^2}{50} + \frac{4.83^2}{50}}$$

$$5.76 < \mu_1 - \mu_2 < 9.86$$

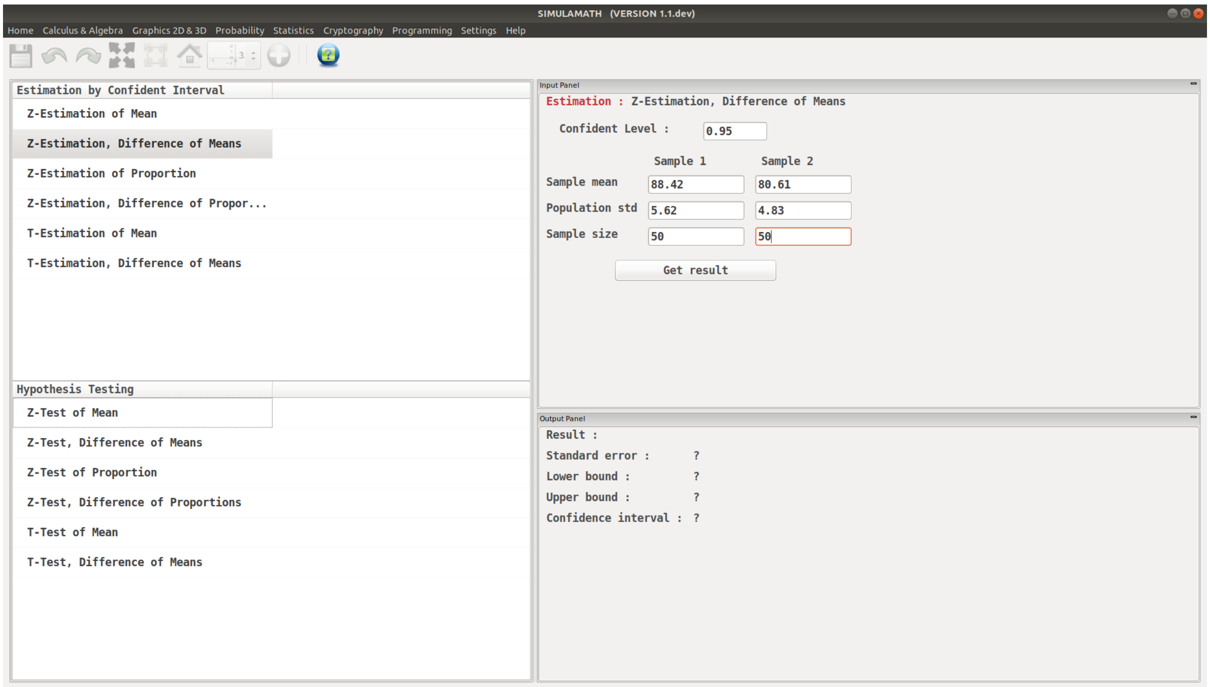
Summarize the results. There is enough evidence to support the claim that the means are not equal. Hence, there is a significant difference in the rates.

Z-Estimation, Difference of Means in Simulamath

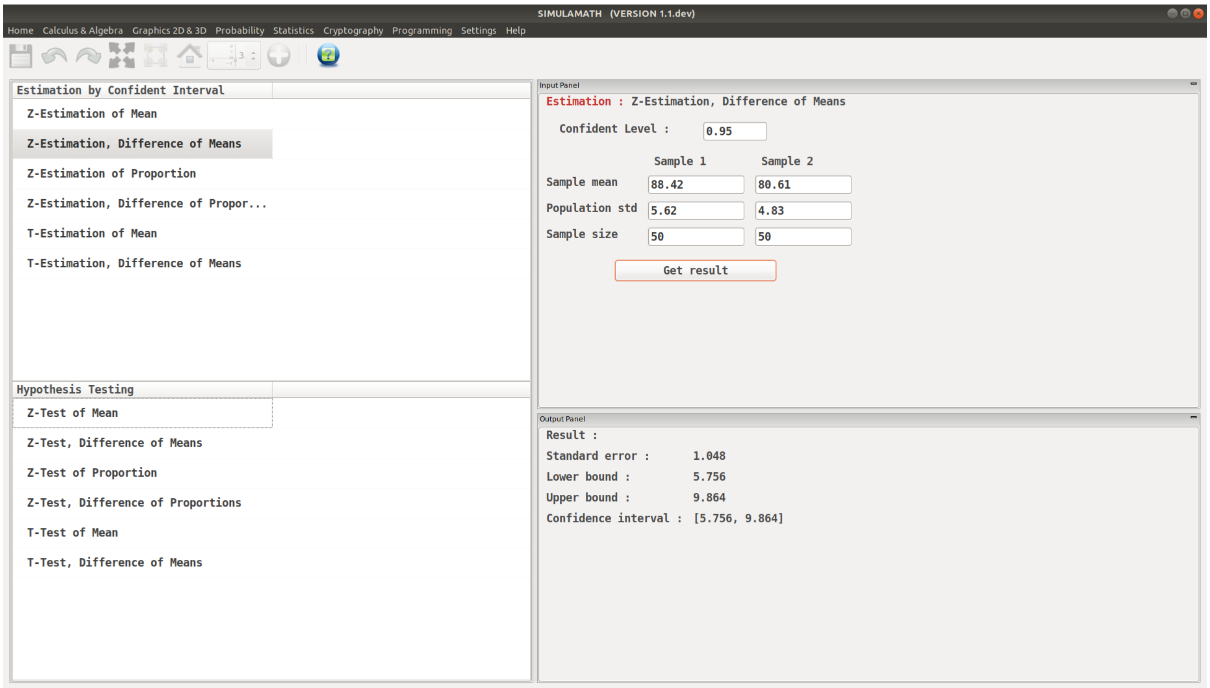
Choose the type of test/estimation you want to compute in the panels on the left hand side, here **Z-Estimation, Difference of Means**



Enter the variables (Confidence level, sample mean, standard deviation of the population, sample size) for each sample in the top panel on the right hand side.



Click on **Get result** located below inside the same panel. Voila, you have your results in the panel down on the right hand side.



Z-Estimation for Proportion

The formula to get the interval confidence for Z-Estimation for Proportion is:

$$\hat{p} - z_{\alpha/2}\sqrt{\frac{\hat{p}\hat{q}}{n}} < p < \hat{p} + z_{\alpha/2}\sqrt{\frac{\hat{p}\hat{q}}{n}}$$

With $\quad \hat{p} = \dfrac{X}{n} \qquad \hat{q} = 1 - p$

Assumptions for Testing a Proportion

1. The sample is a random sample.
2. The conditions for a binomial experiment are satisfied.
3. $n_p \geq 5$ and $n_q \geq 5$.

Example:

A survey conducted by Sallie Mae and Gallup of 1404 respondents found that 323 students paid for their education by student loans. Find the 90% confidence of the true proportion of students who paid for their education by student loans.

Solution:

Since $\alpha = 1 - 0.90 = 0.10$ and $z_{\alpha/2} = 1.65$

Replacing it in the above formula we have

$$0.23 - 1.65\sqrt{\frac{0.23 * 0.77}{1404}} < p < 0.23 + \sqrt{\frac{0.23 * 0.77}{1404}},$$

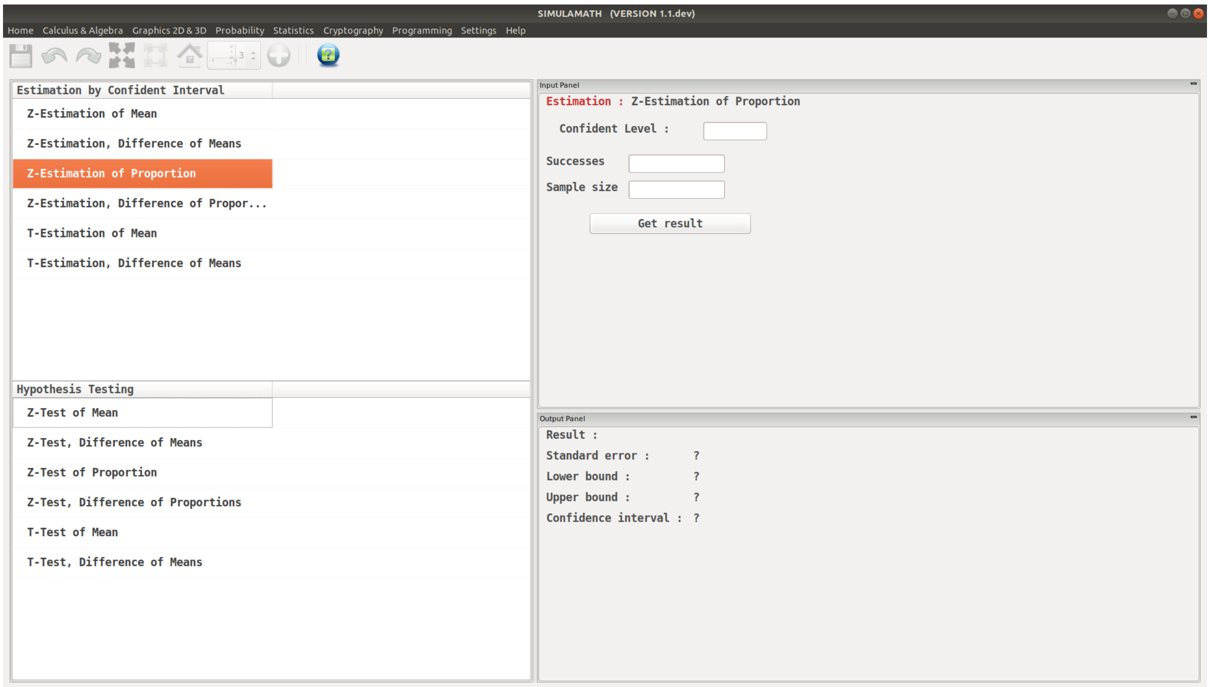
With $\hat{p} = \dfrac{323}{1404} = 0.23$ and $\hat{q} = 1 - p = 0.77$

Hence

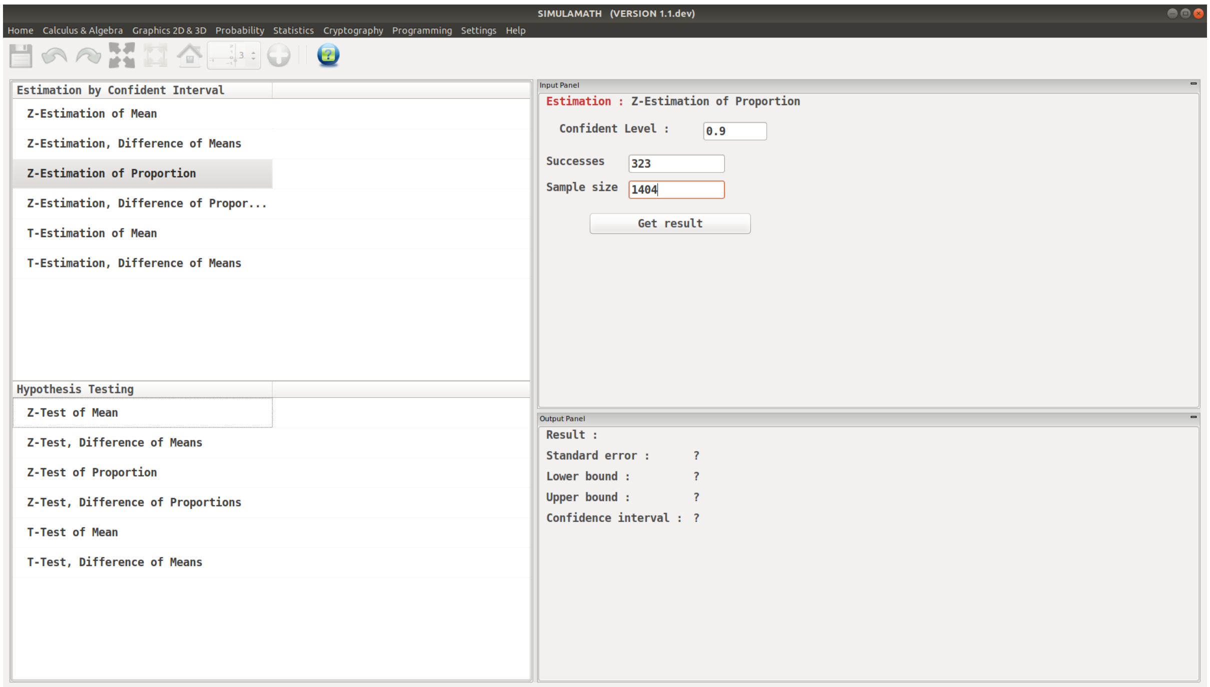
$$0.211 < p < 0.249$$

Z-Estimation for Proportion in Simulamath

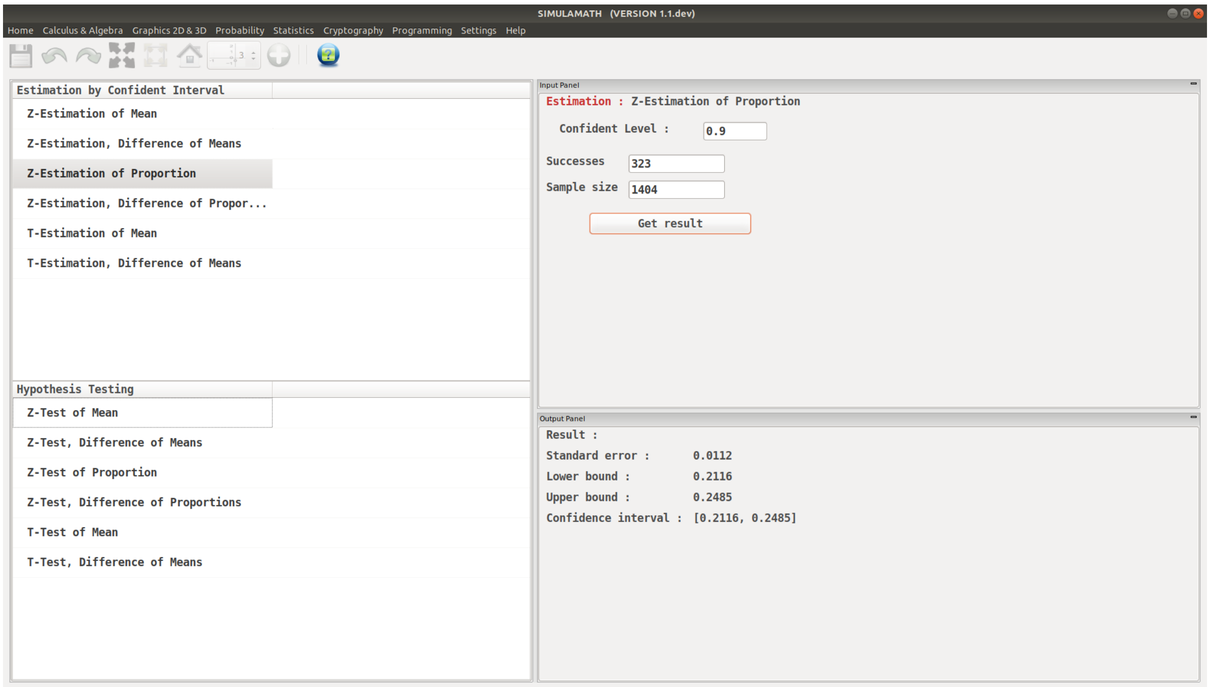
Choose the type of test/estimation you want to compute in the panels on the left hand side, here **Z-Estimation for Proportion**



Enter the variables (Confidence level, Success, Sample size) in the top panel on the right hand side.



Click on **Get result** located below inside the same panel. Voila, you have your results in the panel down on the right hand side.



Z-Estimation, Difference of Proportions

The formula to get the interval confidence for Z-Estimation, Difference of Proportions is:

$$(\hat{p_1} - \hat{p_2}) - z_{\alpha/2}\sqrt{\frac{\hat{p_1}\hat{q_1}}{n_1} + \frac{\hat{p_1}\hat{q_1}}{n_2}} < p_1 - p_2 < (\hat{p_1} - \hat{p_2}) + z_{\alpha/2}\sqrt{\frac{\hat{p_1}\hat{q_1}}{n_1} + \frac{\hat{p_1}\hat{q_1}}{n_2}}$$

Example:

Researchers found that 12 out of 34 small nursing homes had a resident vaccination rate of less than 80%, while 17 out of 24 large nursing homes had a vaccination rate of less than 80%. At a $\alpha = 0.05$, test the claim that there is no difference in the proportions of the small and large nursing homes with a resident vaccination rate of less than 80%.

Solution:

Replacing in the above formula we get

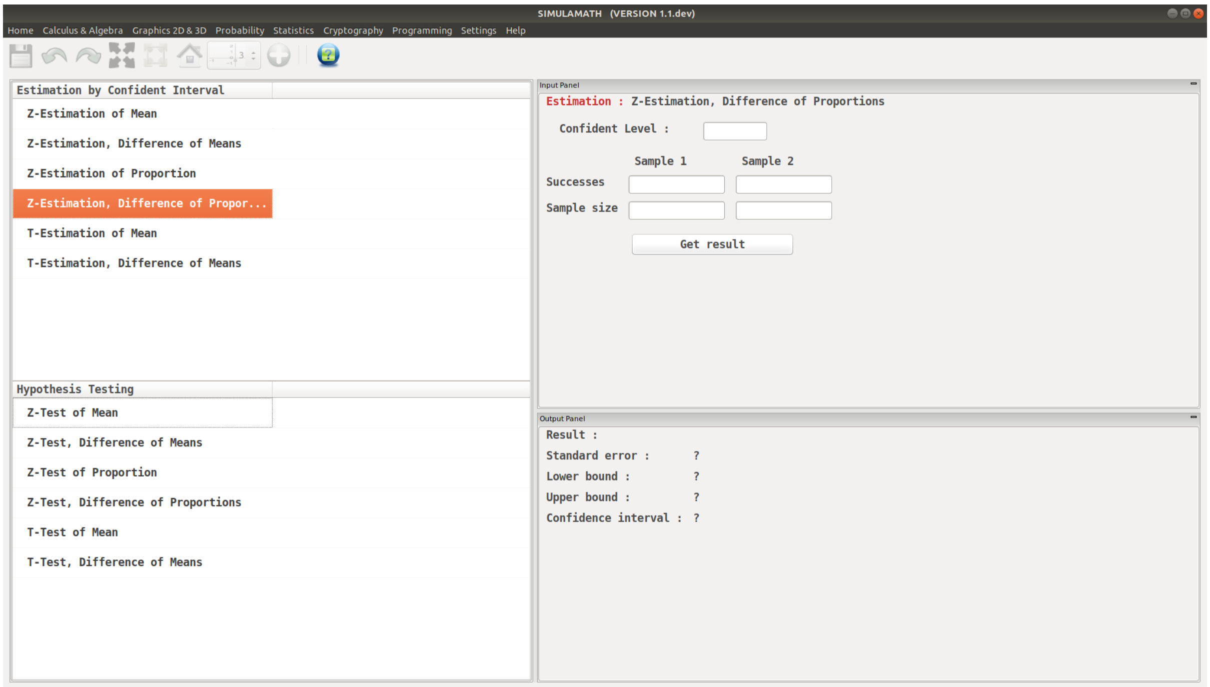
$$(0.35 - 0.71) - 1.96\sqrt{\frac{0.35 * 0.65}{34} + \frac{0.71 * 0.29}{24}} < p_1 - p_2 < (0.35 - 0.71) + 1.96\sqrt{\frac{0.35 * 0.65}{34} + \frac{0.71 * (}{24}$$

$$-0.602 < p_1 - p_2 < -0.118$$

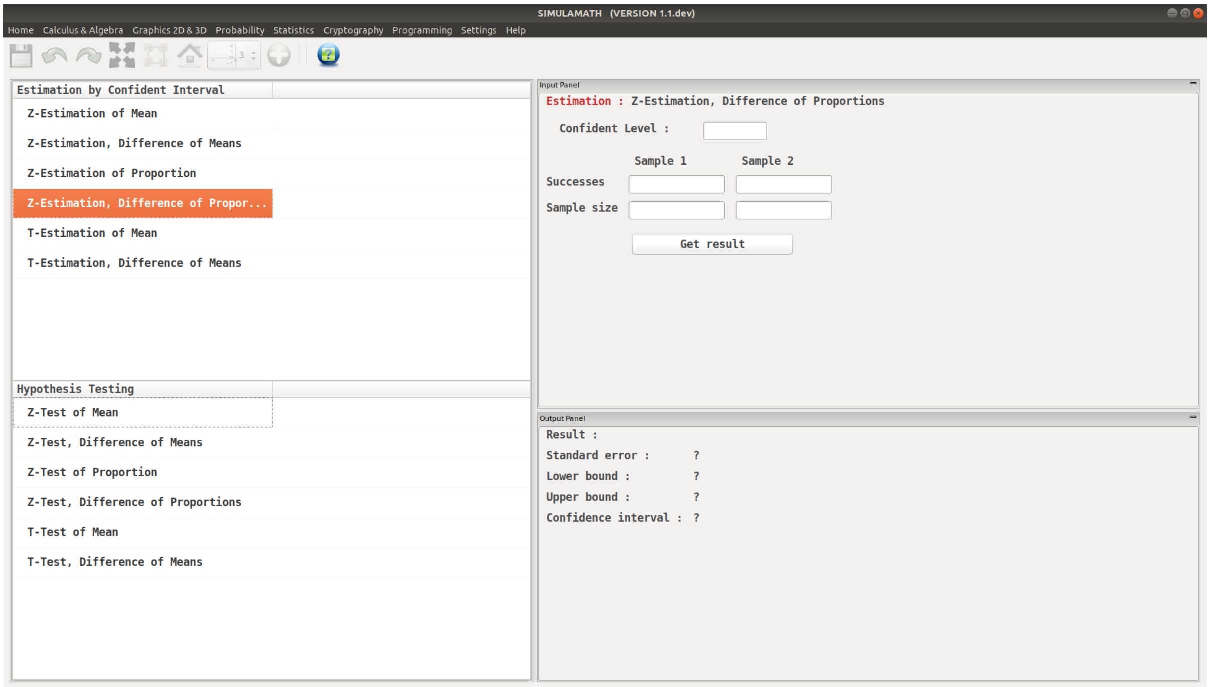
Since 0 is not contained in the interval, the decision is to reject the null hypothesis $H_0 : p_1 = p_2$.

Z-Estimation, Difference of Proportions in Simulamath

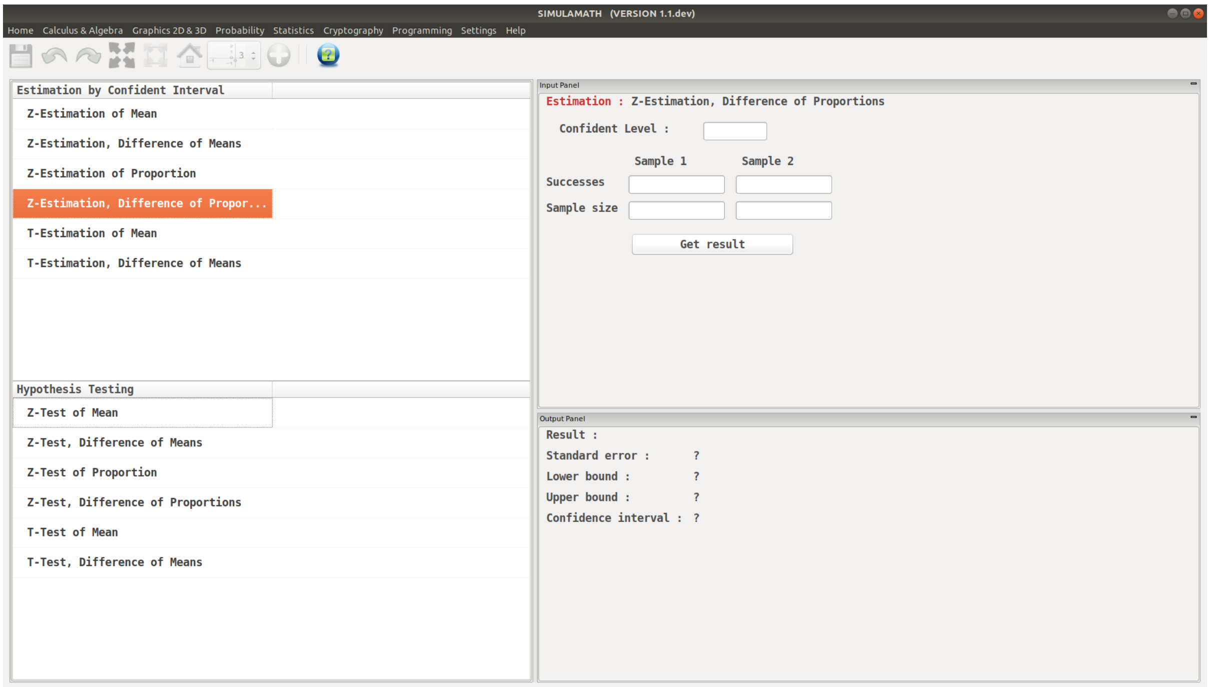
Choose the type of test/estimation you want to compute in the panels on the left hand side, here **Z-Estimation, Difference of Proportions**



Enter the variables (Confidence level, Success, Sample size) for each sample in the top panel on the right hand side.



Click on **Get result** located below inside the same panel. Voila, you have your results in the panel down on the right hand side.



T-Estimation for Mean

The formula to get the interval confidence for T-Estimation for Mean is:

$$\bar{X} - t_{\alpha/2}\left(\frac{s}{\sqrt{n}}\right) < \mu < \bar{X} + t_{\alpha/2}\left(\frac{s}{\sqrt{n}}\right)$$

Assumptions for finding a Confidence interval for a Mean when σ is Unknown

1. The sample is a random sample.
2. Either $n \geq 30$ or the population is normally distributed if math:*n < 30*.

Example:

Ten randomly selected people were asked how long they slept at night. The mean time was 7.1 hours, and the standard deviation was 0.78 hour. Find the 95% confidence interval of the mean time. Assume the variable is normally distributed. Solution:

Since σ is unknown and s must replace it, the t distribution (Table F) must be used for the confidence interval. Hence, with 9 degrees of freedom $t_{\alpha/2} = 2.262$. The 95% confidence interval can be found by substituting in the above formula.

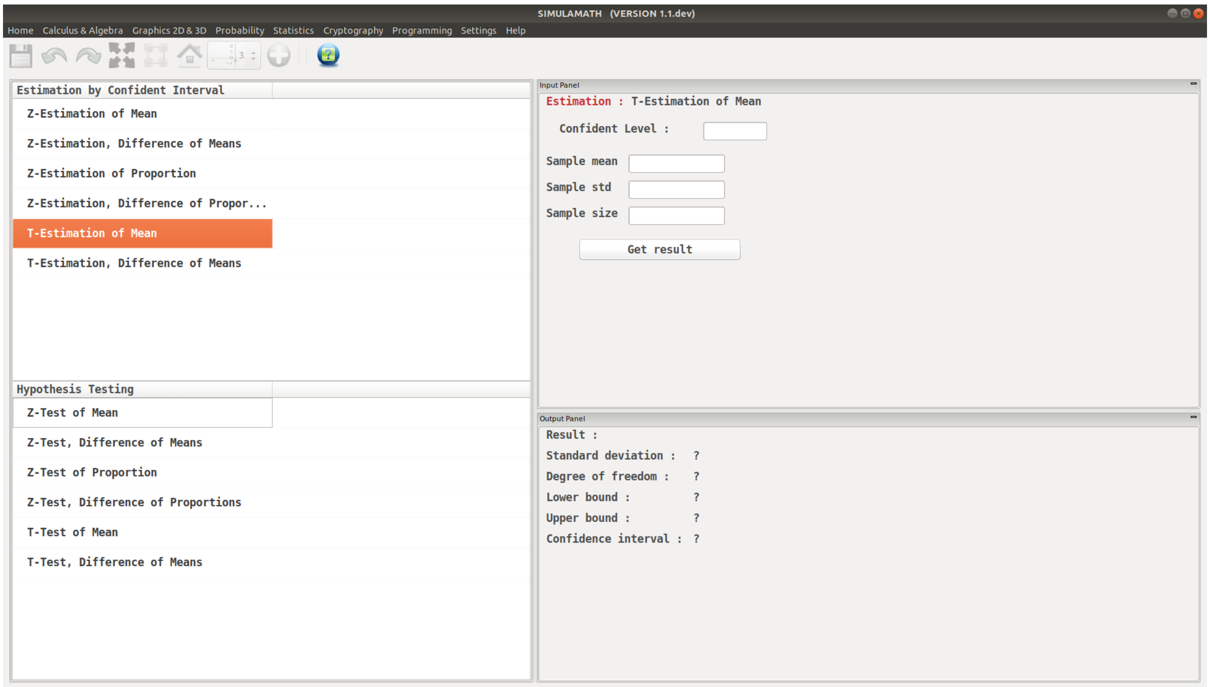
$$7.1 - 2.262\left(\frac{0.78}{\sqrt{10}}\right) < \mu < 7.1 + 2.262\left(\frac{0.78}{\sqrt{10}}\right)$$

$$6.54 < \mu < 7.66$$

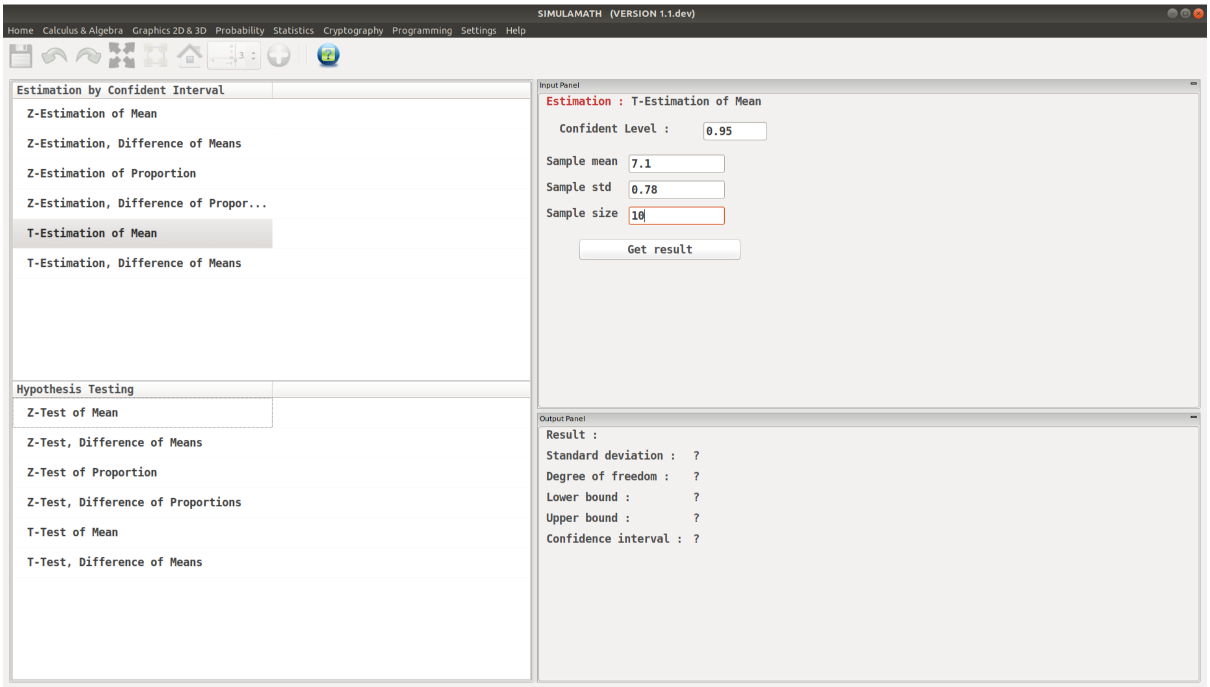
Therefore, one can be 95% confident that the population mean is between 6.54 and 7.66 hours.

T-Estimation for Mean in Simulamath

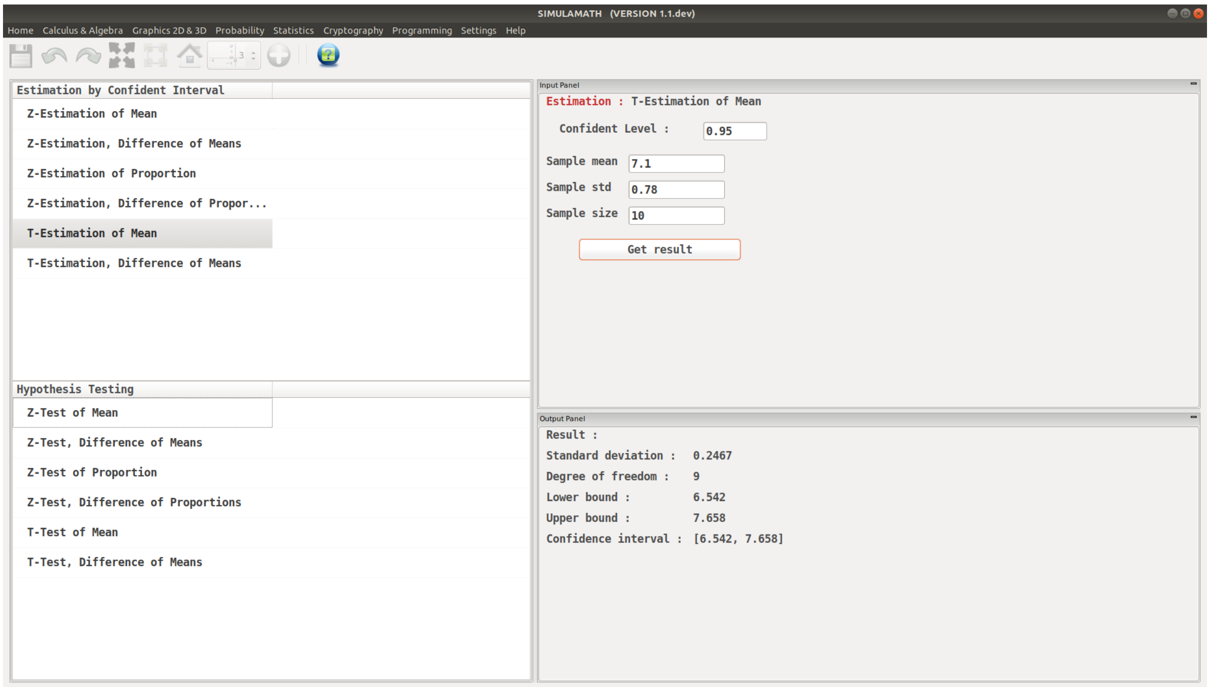
Choose the type of test/estimation you want to compute in the panels on the left hand side, here **T-Estimation for Mean**



Enter the variables (Confidence level, sample mean, standard deviation of the sample, sample size) in the top panel on the right hand side.



Click on **Get result** located below inside the same panel. Voila, you have your results in the panel down on the right hand side.



T-Estimation, Difference of Means

The formula to get the interval confidence for T-Estimation, Difference of Means is:

$$(\bar{X}_1 - \bar{X}_2) - t_{\alpha/2}\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} < \mu_1 - \mu_2 < (\bar{X}_1 - \bar{X}_2) + t_{\alpha/2}\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

Example:

The average size of a farm in Indiana County, Pennsylvania, is 191 acres. The average size of a farm in Greene County, Pennsylvania, is 199 acres. Assume the data were obtained from two samples with standard deviations of 38 and 12 acres, respectively, and sample sizes of 8 and 10, respectively. Can it be concluded at a $\alpha = 0.05$ that the average size of the farms in the two counties is different? Assume the populations are normally distributed. Solution:

Replacing in the above formula

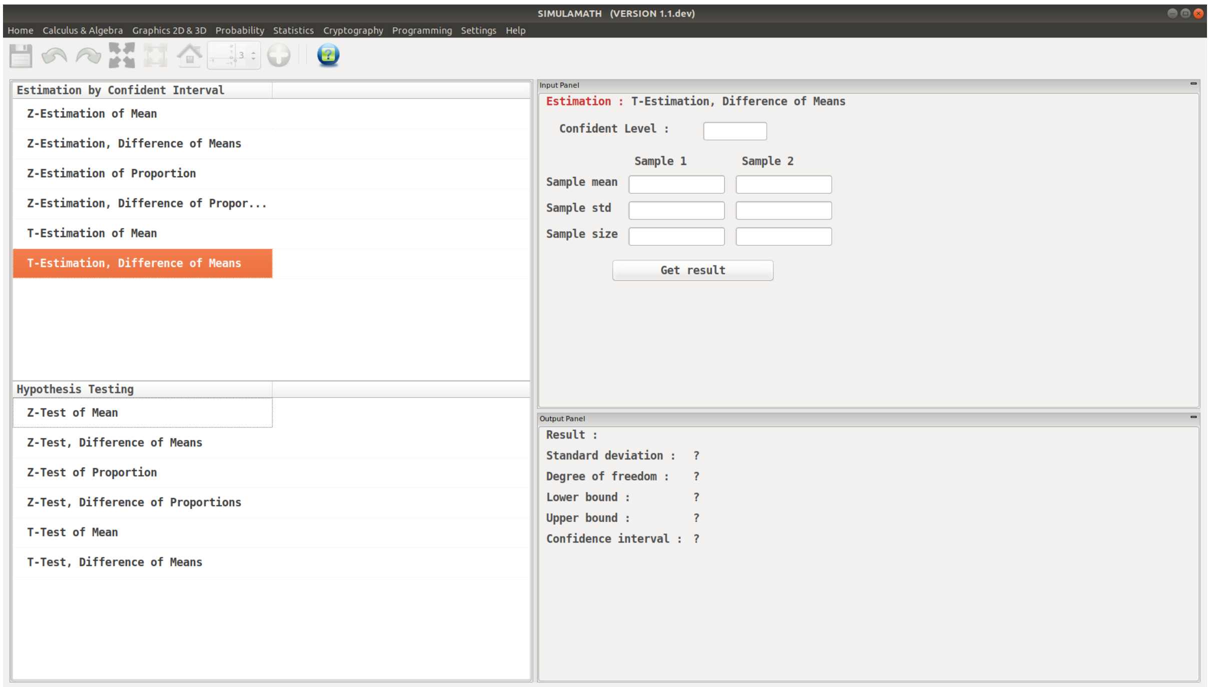
$$(191 - 199) - 2.365\sqrt{\frac{38^2}{8} + \frac{12^2}{10}} < \mu_1 - \mu_2 < (191 - 199) + 2.365\sqrt{\frac{38^2}{8} + \frac{12^2}{10}}$$

$$-41.02 < \mu_1 - \mu_2 < 25.02$$

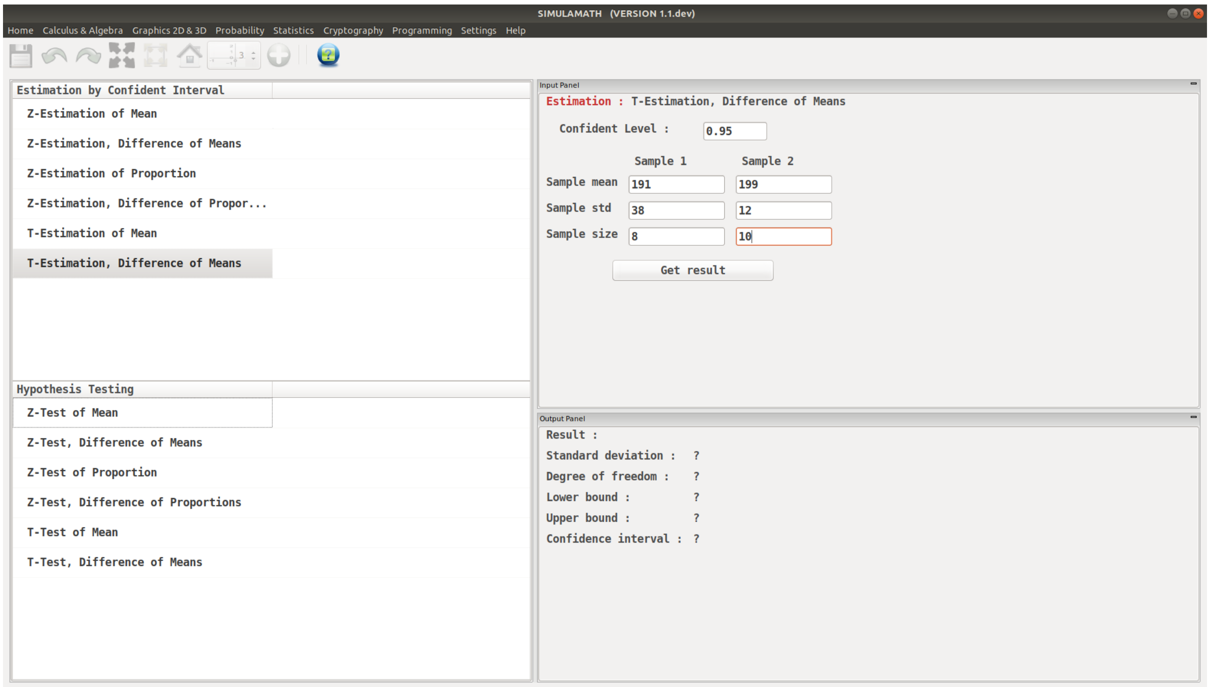
Since 0 is contained in the interval, the decision is to not reject the null hypothesis $H_0 : \mu_1 = \mu_2$.

T-Estimation, Difference of Means in Simulamath

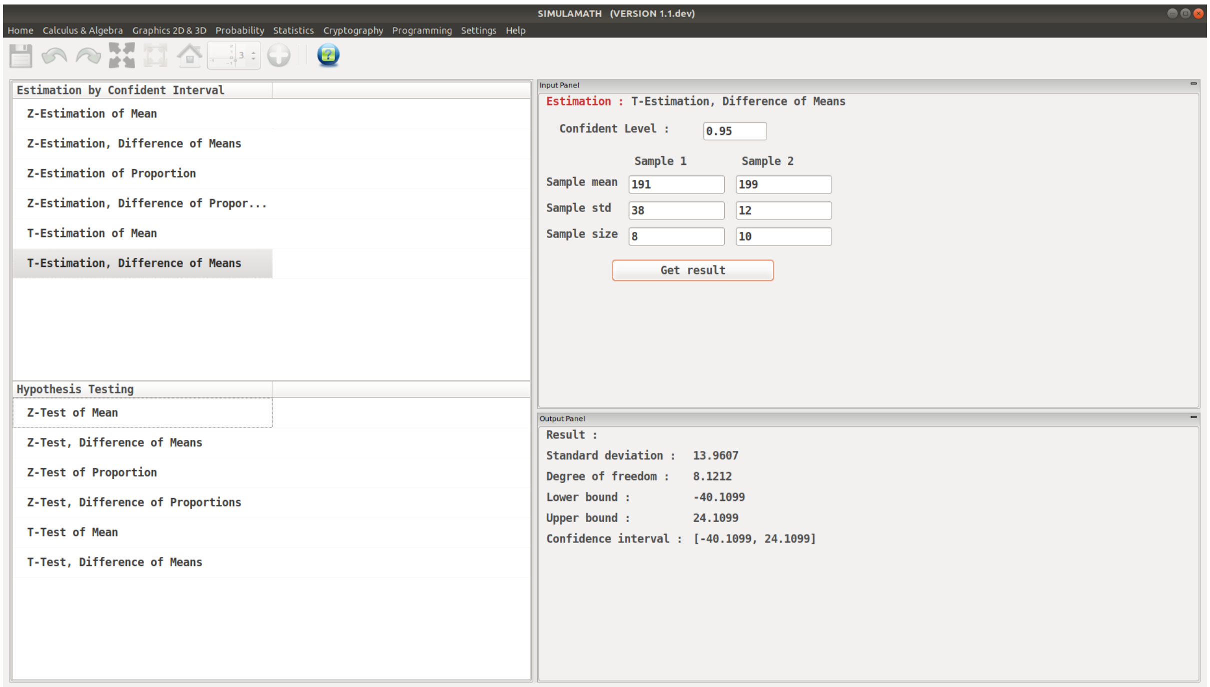
Choose the type of test/estimation you want to compute in the panels on the left hand side, here **T-Estimation, Difference of Means**



Enter the variables (Confidence level, sample mean, standard deviation of the sample, sample size) for each sample in the top panel on the right hand side.



Click on **Get result** located below inside the same panel. Voila, you have your results in the panel down on the right hand side.



4.4.3 Hypothesis Testing

Z-Test for a Mean

The formula for hypothesis testing for Z-Test for a Mean is:

$$z = \frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{n}}}$$

Assumptions for the z-Test for a Mean when σ is known

1. The sample is a random sample.
2. Either $n \geq 30$ or the population is normally distributed if $n < 30$.

Example:

A researcher wishes to see if the mean number of days that a basic, low-price, small automobile sits on a dealer's lot is 29. A sample of 30 automobile dealers has a mean of 30.1 days for basic, low-price, small automobiles. At $\alpha = 0.05$, test the claim that the mean time is greater than 29 days. The standard deviation of the population is 3.8 days.

Solution:

Step 1: State the hypotheses and identify the claim.

$$H_0 : \mu = 29 \quad \text{and} \quad H_1 : \mu > 29 (\text{claim})$$

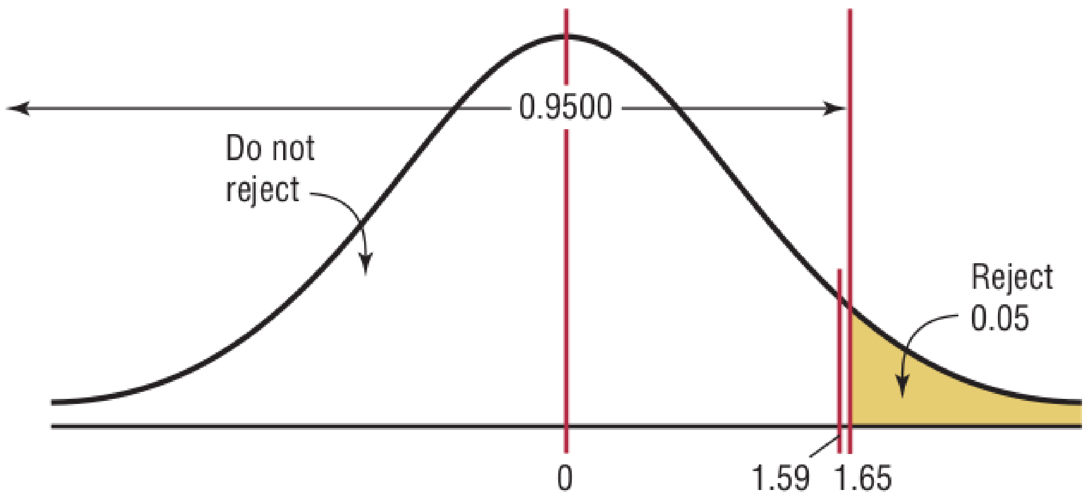
Step 2: Find the critical value. Since $\alpha = 0.05$ and the test is a right-tailed test, the critical value is $z = +1.65$.

Step 3: Compute the test value.

$$z = \frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{n}}}$$

$$z = \frac{30.1 - 29}{\frac{3.8}{\sqrt{3}}} = 1.59$$

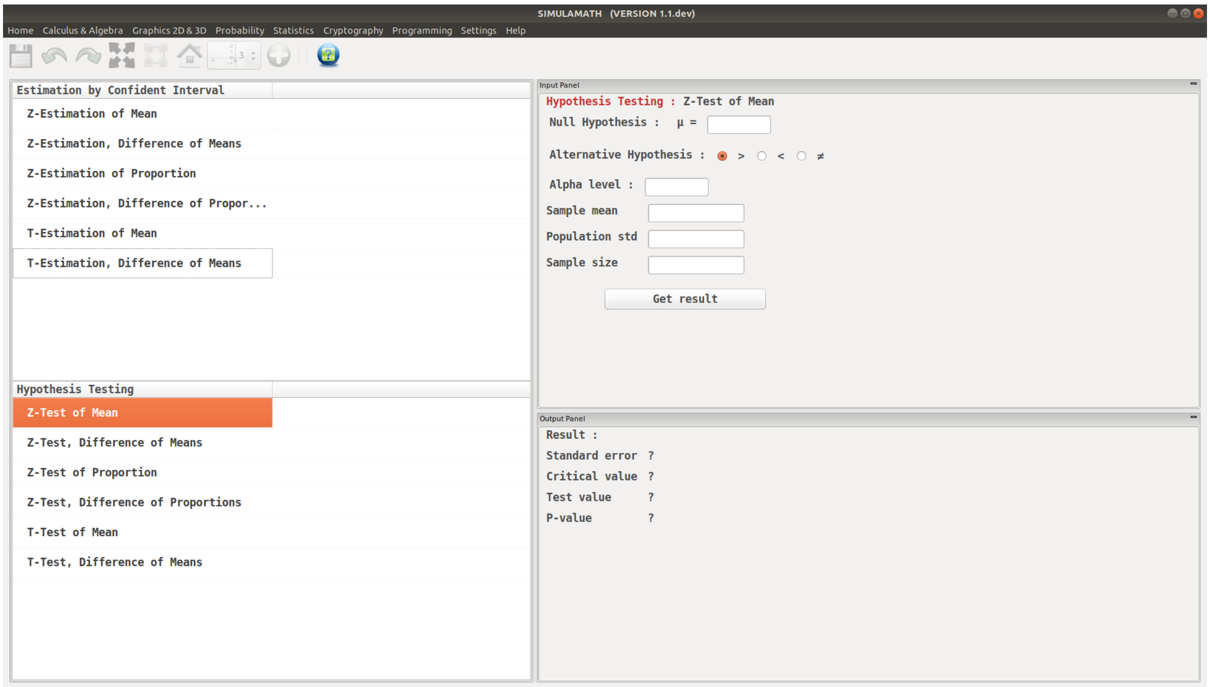
Step 4: Make the decision. Since the test value, +1.59, is less than the critical value, +1.65, and is not in the critical region, the decision is to not reject the null hypothesis.



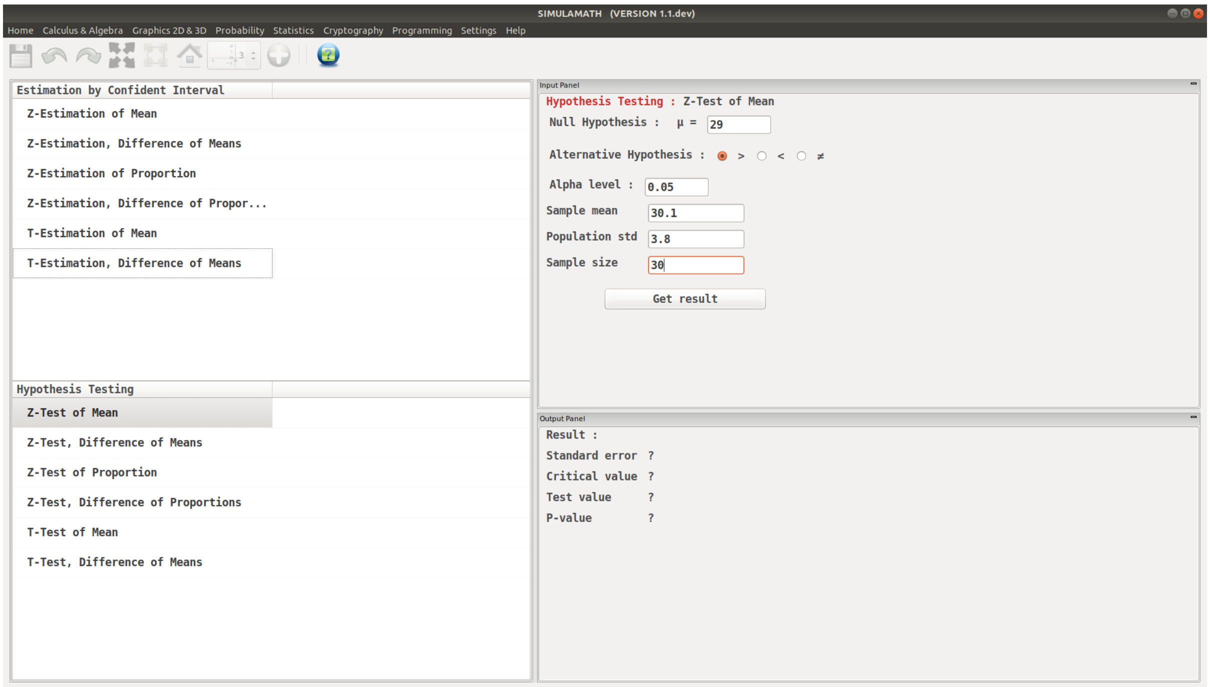
Step 5: Summarize the results. There is not enough evidence to support the claim that the mean time is greater than 29 days.

Z-Test for a Mean in Simulamath

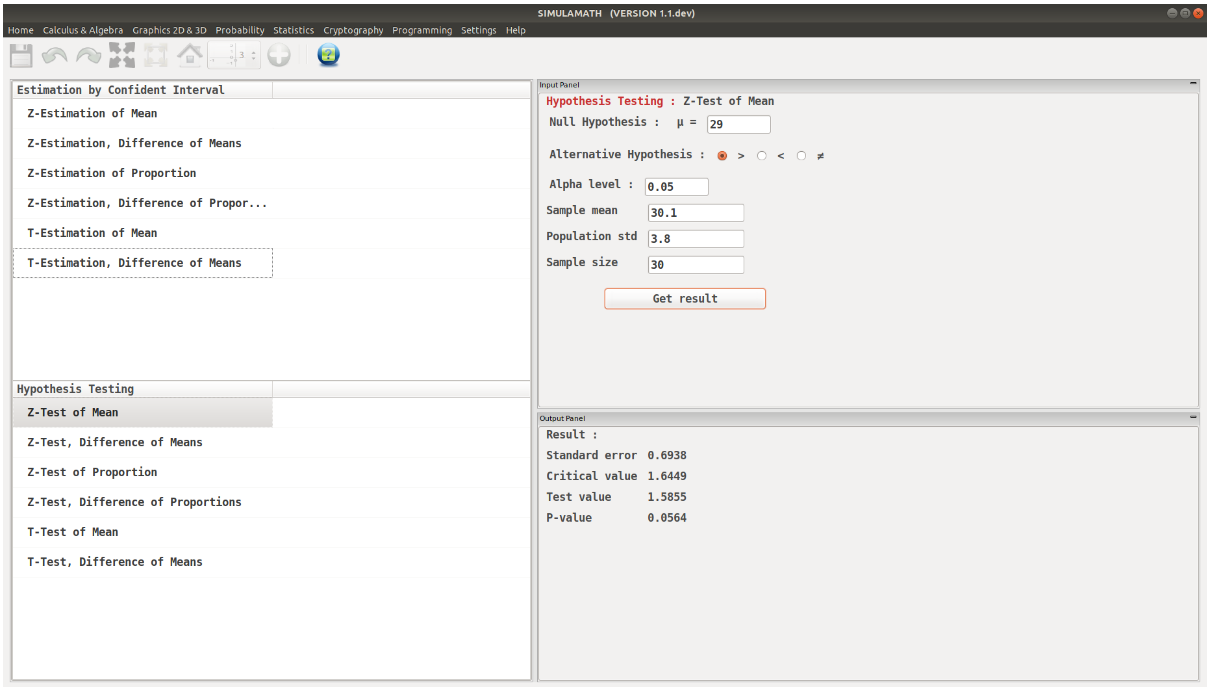
Choose the type of test/estimation you want to compute in the panels on the left hand side, here **Z-Estimation of Mean**



Enter the variables (Null Hypothesis, Alternative Hypothesis, Alpha value, Sample mean, Population standard deviation, Sample size) in the top panel on the right hand side.



Click on **Get result** located below inside the same panel. Voila, you have your results in the panel down on the right hand side.



Z-Test, Difference of Means

The formula for hypothesis testing for Z-Test, Difference of Means is:

$$z = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_1)}{\sqrt{\dfrac{\sigma_1^2}{n_1} + \dfrac{\sigma_2^2}{n_2}}}$$

$$\text{Test value} = \frac{(\text{observed value}) - (\text{expected value})}{\text{standard error}}$$

Example:

A survey found that the average hotel room rate in New Orleans is \$88.42 and the average room rate in Phoenix is \$80.61. Assume that the data were obtained from two samples of 50 hotels each and that the standard deviations of the populations are \$5.62 and \$4.83, respectively. At $\alpha = 0.05$, can it be concluded that there is a significant difference in the rates?

Solution:

Step 1: State the hypotheses and identify the claim.

$$H_0 : \mu_1 = \mu_2 \quad \text{and} \quad H_1 : \mu_1 \neq \mu_2 (\text{claim})$$

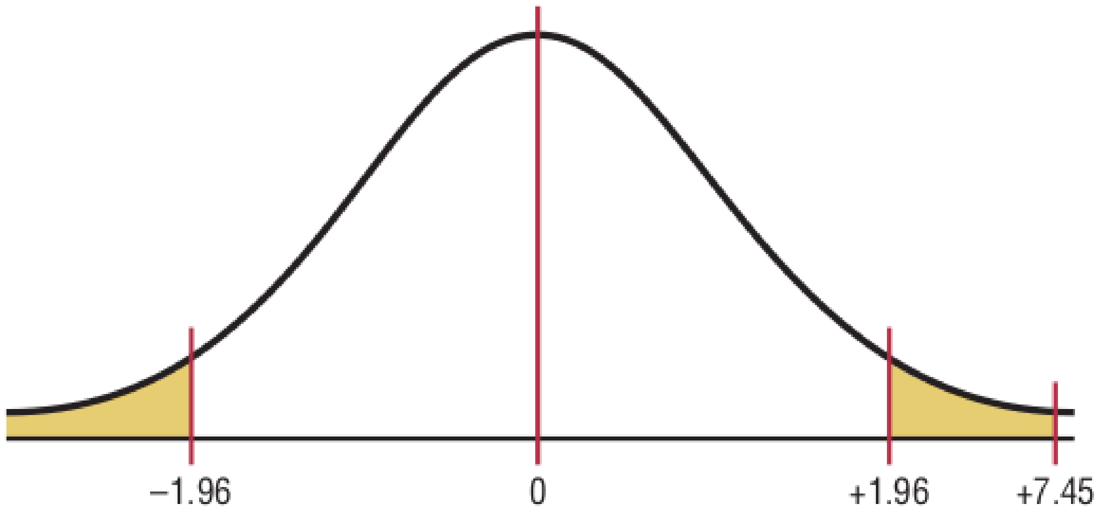
Step 2: Find the critical values. Since $\alpha = 0.05$, the critical values are +1.96 and -1.96.

Step 3: Compute the test value.

$$z = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_1)}{\sqrt{\dfrac{\sigma_1^2}{n_1} + \dfrac{\sigma_2^2}{n_2}}}$$

$$z = \frac{(88.42 - 80.61) - 0}{\sqrt{\frac{5.62^2}{50} + \frac{4.83^2}{50}}} = 7.45$$

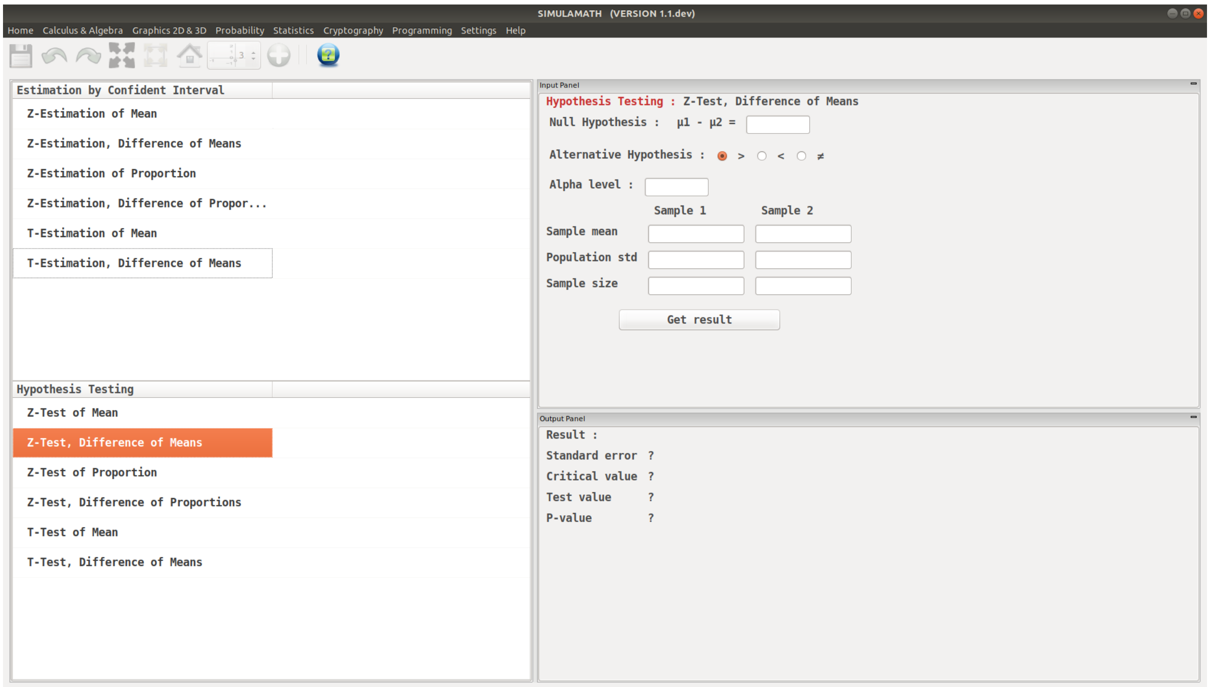
Step 4: Make the decision. Reject the null hypothesis at $\alpha = 0.05$, since 7.45 > 1.96.



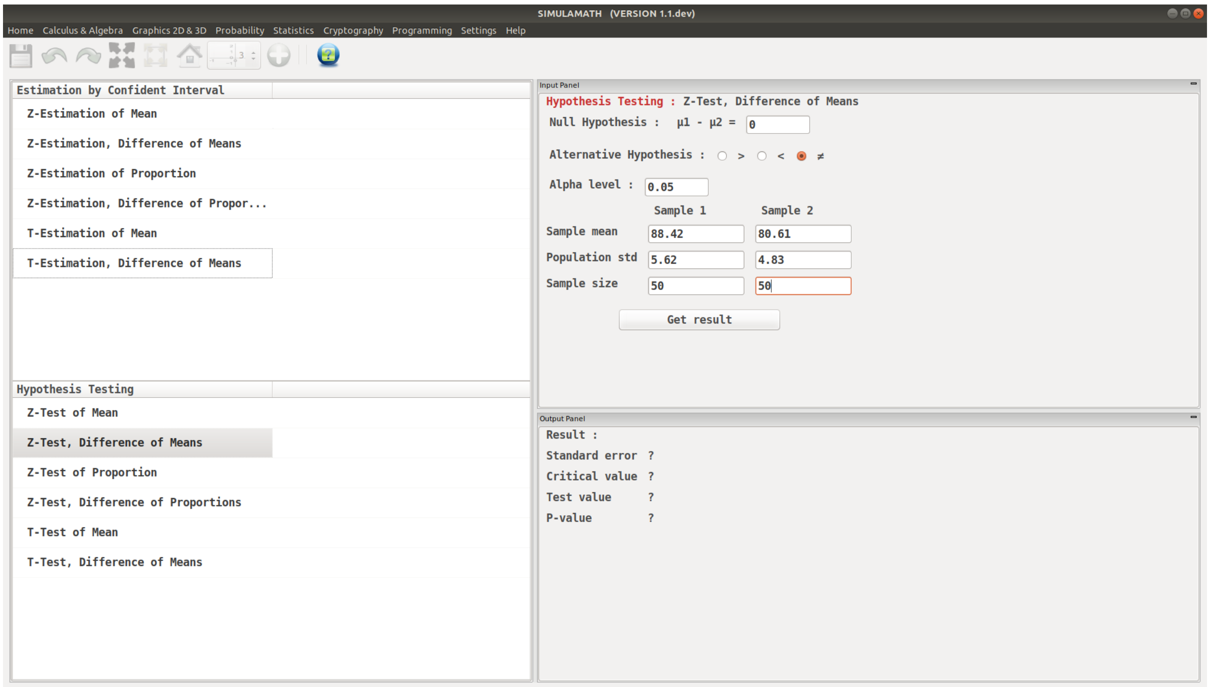
Step 5: Summarize the results. There is enough evidence to support the claim that the means are not equal. Hence, there is a significant difference in the rates.

Z-Test, Difference of Means in Simulamath

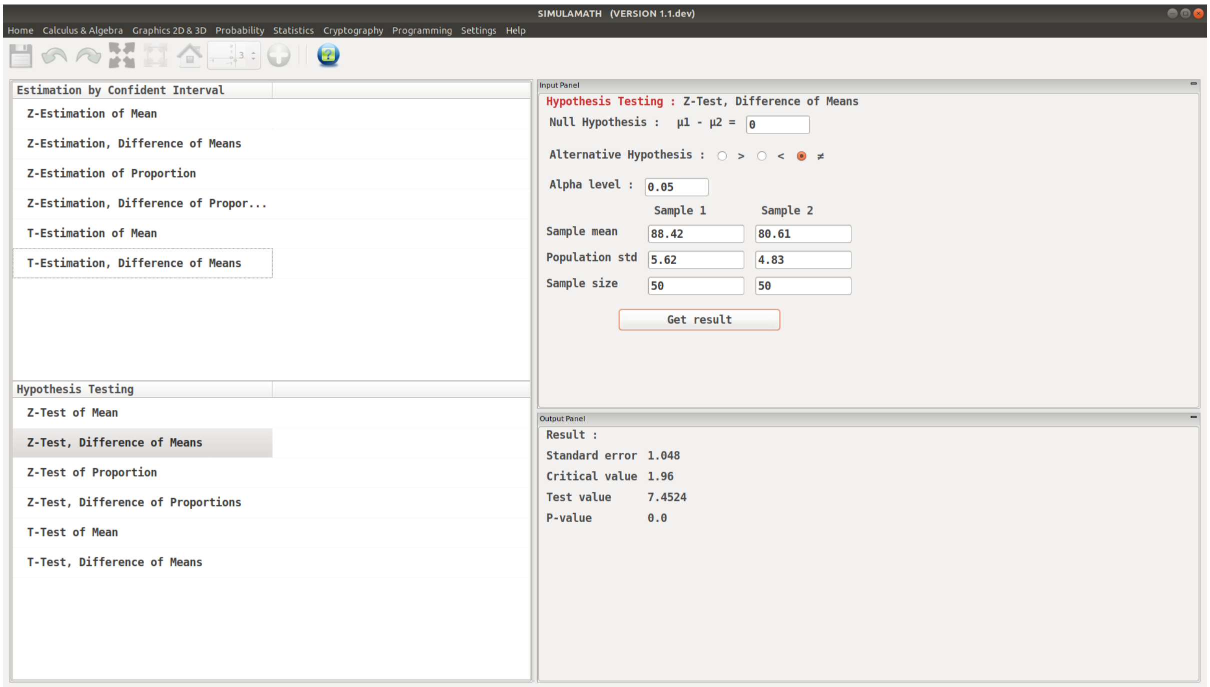
Choose the type of test/estimation you want to compute in the panels on the left hand side, here **Z-Estimation, Difference of Means**



Enter the variables (Null Hypothesis, Alternative Hypothesis, Alpha value, Sample mean, Population standard deviation, Sample size) for each sample in the top panel on the right hand side.



Click on **Get result** located below inside the same panel. Voila, you have your results in the panel down on the right hand side.



Z-Test for Proportion

The formula for hypothesis testing for Z-Test for Proportion is:

$$z = \frac{\hat{p} - p}{\sqrt{pq/n}}$$

Assumptions for Testing a Proportion

1. The sample is a random sample.
2. The conditions for a binomial experiment are satisfied.
3. $np \geq 5$ and $nq \geq 5$.

Example:

A dietitian claims that 60% of people are trying to avoid trans fats in their diets. She randomly selected 200 people and found that 128 people stated that they were trying to avoid trans fats in their diets. At $\alpha = 0.05$, is there enough evidence to reject the dietitian's claim?

Solution:

Step 1: State the hypothesis and identify the claim.

$$H_0 : p = 0.60 \quad \text{(claim)} \quad \text{et} \quad H_1 : p \neq 0.60$$

Step 2: Find the critical values. Since $\alpha = 0.05$ and the test value is two-tailed, the critical values are +1.96 and -1.96.

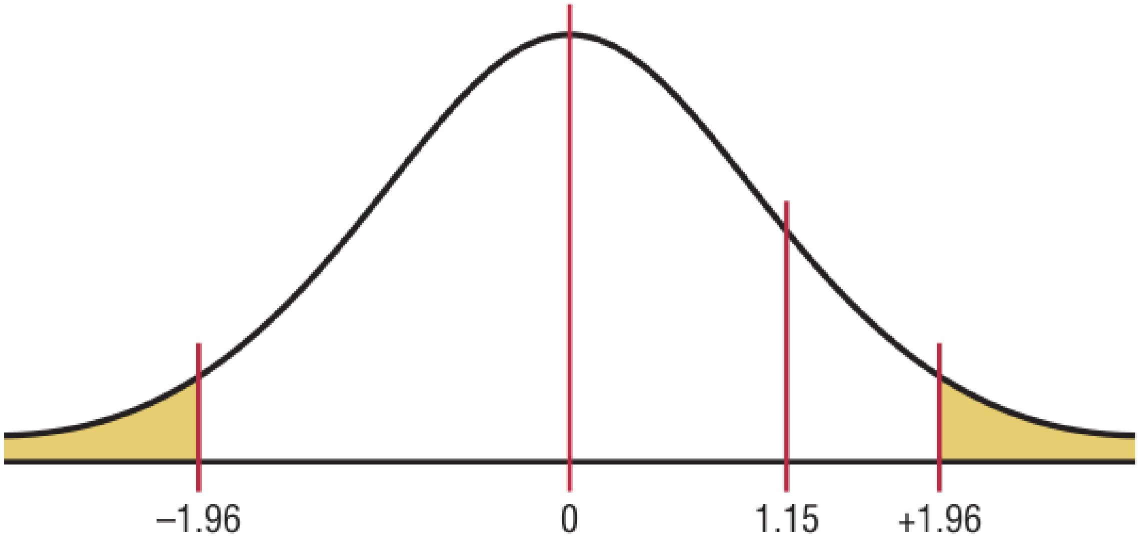
Step 3: Compute the test value. First, it is necessary to find $\hat{p}$.

$$\hat{p} = \frac{X}{n} = \frac{128}{200} = 0.64 \qquad p = 0.60 \quad \text{and then} \quad q = 1 - p = 0.40$$

Therefore

$$z = \frac{0.64 - 0.60}{\sqrt{0.60 * 0.40/200}} = 1.15$$

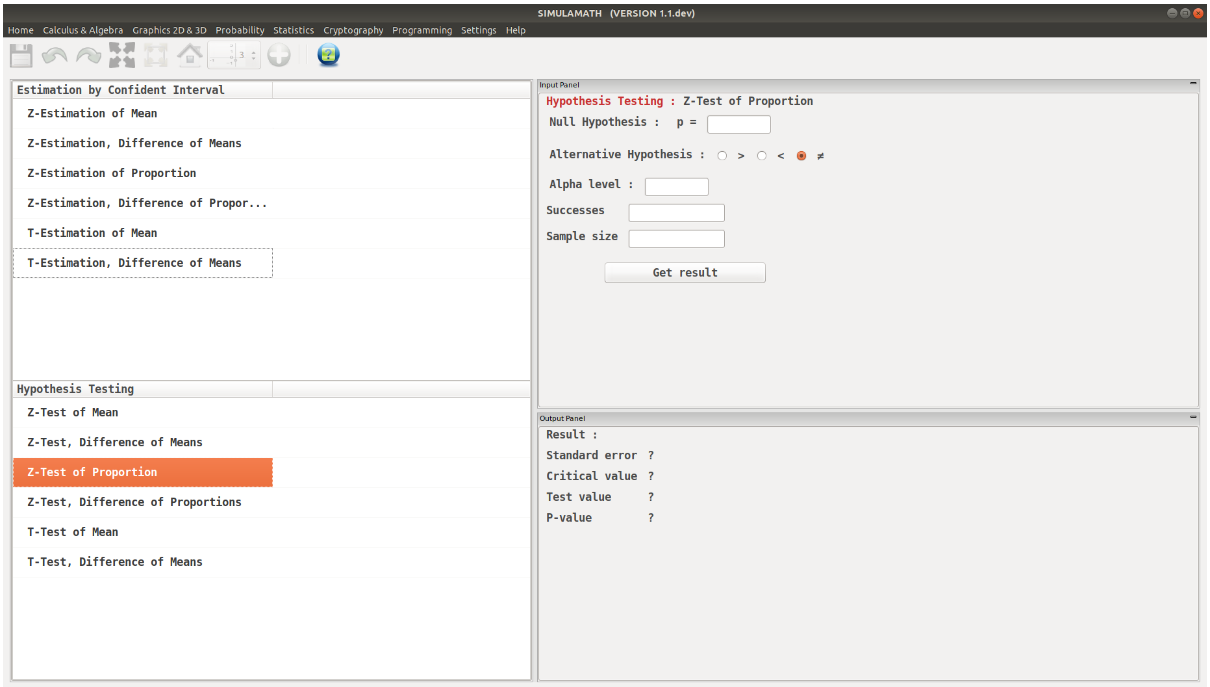
Step 4: Make the decision. Do not reject the null hypothesis since the test value falls outside the critical region, as shown in the figure below



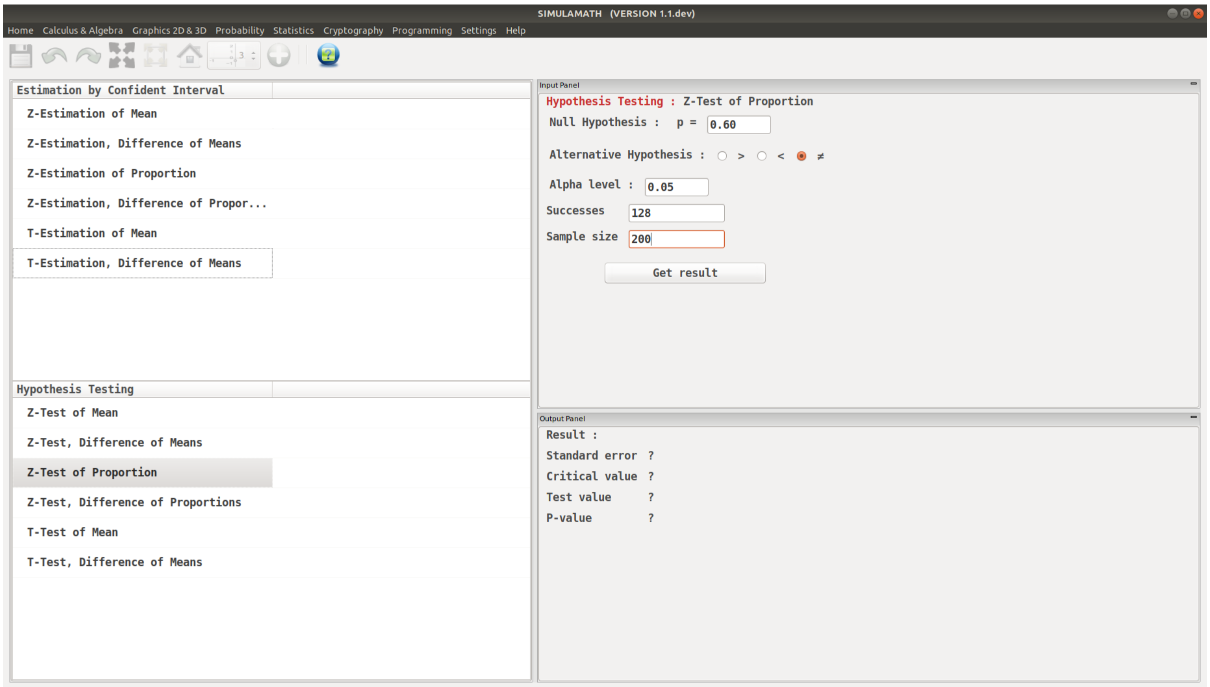
Step 5: Summarize the results. There is not enough evidence to reject the claim that 60% of people are trying to avoid trans fats in their diets.

Z-Test for Proportion in Simulamath

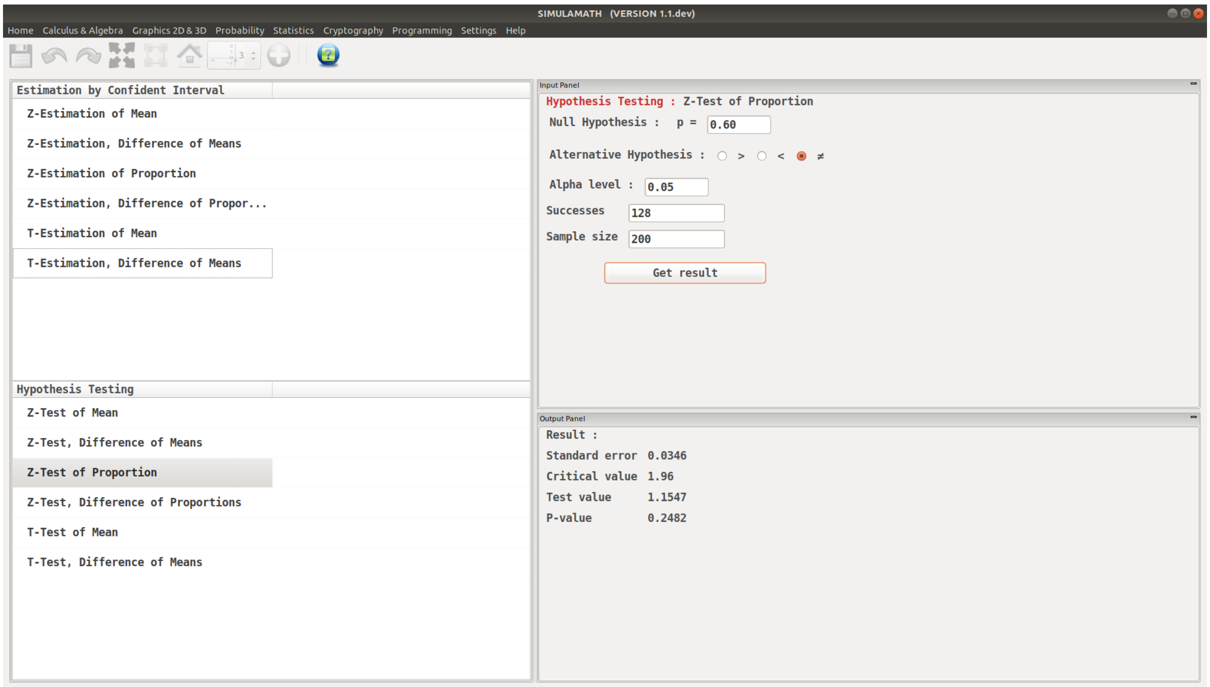
Choose the type of test/estimation you want to compute in the panels on the left hand side, here **Z-Test for Proportion**



Enter the variables (Null Hypothesis, Alternative Hypothesis, Alpha value, Success, Sample size) in the top panel on the right hand side.



Click on **Get result** located below inside the same panel. Voila, you have your results in the panel down on the right hand side.



Z-Test, Difference of Proportions

The formula for hypothesis testing for Z-Test, Difference of Proportions is:

$$z = \frac{(\hat{p_1} - \hat{p_2}) - (p_1 - p_1)}{\sqrt{\bar{p}\bar{q}(\frac{1}{n_1} + \frac{1}{n_2})}}$$

where

$$\bar{p} = \frac{X_1 + X_2}{n_1 + n_2} \qquad \hat{p_1} = \frac{X_1}{n_1}$$

and

$$\bar{q} = 1 - \bar{p} \qquad \hat{p_2} = \frac{X_2}{n_2}$$

This formula is based on the general format of

$$\text{Test value} = \frac{(\text{observed value}) - (\text{expected value})}{\text{standard error}}$$

Assumptions for the z-Test for Two Proportions

1. The samples must be random samples.
2. The sample data are independent of one another.
3. For both samples $np \geq 5$ and $nq \geq 5$.

Example:

Researchers found that 12 out of 34 small nursing homes had a resident vaccination rate of less than 80%, while 17 out of 24 large nursing homes had a vaccination rate of less than 80%. At $\alpha = 0.05$, test the claim that there is no difference in the proportions of the small and large nursing homes with a resident vaccination rate of less than 80%.

Solution:

Let $\hat{p_1}$ be the proportion of the small nursing homes with a vaccination rate of less than 80% and $\hat{p_2}$ be the proportion of the large nursing homes with a vaccination rate of less than 80%. Then

$$\bar{p} = \frac{X_1 + X_2}{n_1 + n_2} = \frac{12 + 17}{34 + 24} = 0.5 \qquad \hat{p_1} = \frac{X_1}{n_1} = \frac{12}{34} = 0.35$$

$$\bar{q} = 1 - \bar{p} = 1 - 0.5 = 0.5 \qquad \hat{p_2} = \frac{X_2}{n_2} = \frac{17}{24} = 0.71$$

Step 1: State the hypotheses and identify the claim.

$$H_0 : p_1 = p_2 \quad \text{(claim)} \quad \text{and} \quad H_1 : p_1 \neq p_2$$

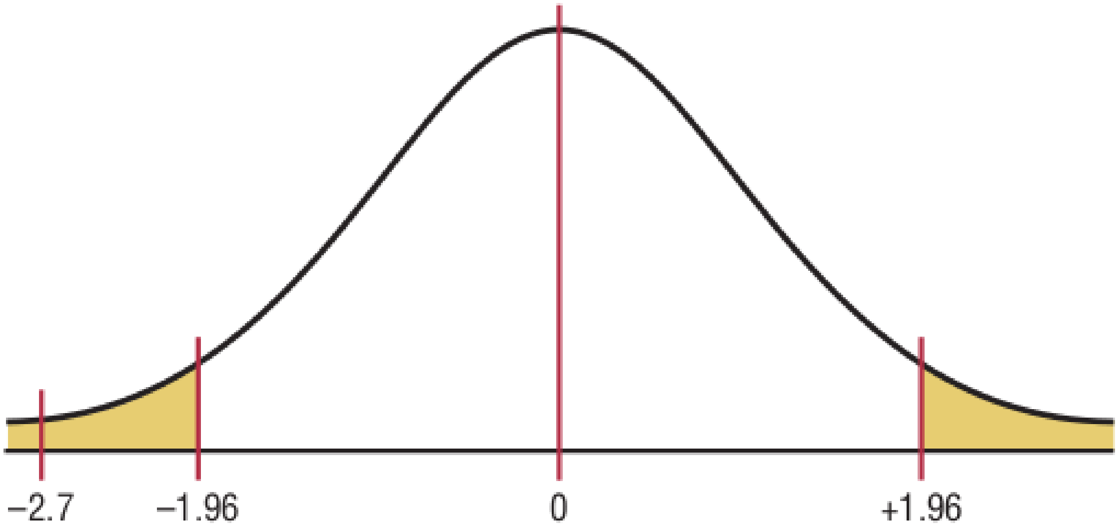
Step 2: Find the critical values. Since $\alpha = 0.05$, the critical values are +1.96 and -1.96.

Step 3: Compute the test value.

$$z = \frac{(\hat{p_1} - \hat{p_2}) - (p_1 - p_1)}{\sqrt{\bar{p}\bar{q}(\frac{1}{n_1} + \frac{1}{n_2})}}$$

$$z = \frac{(0.35 - 0.75) - 0}{\sqrt{0.5 * 0.5(\frac{1}{34} + \frac{1}{24})}} = -2.7$$

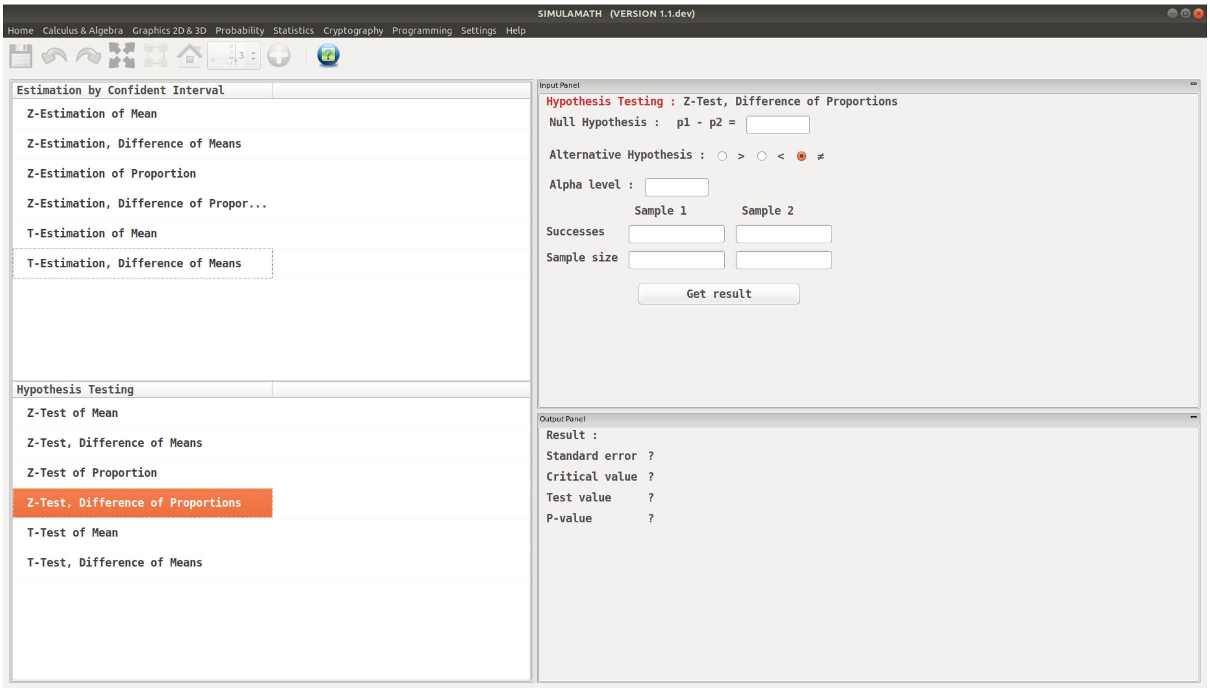
Step 4: Make the decision. Reject the null hypothesis, since -2.7 < -1.96. see figure



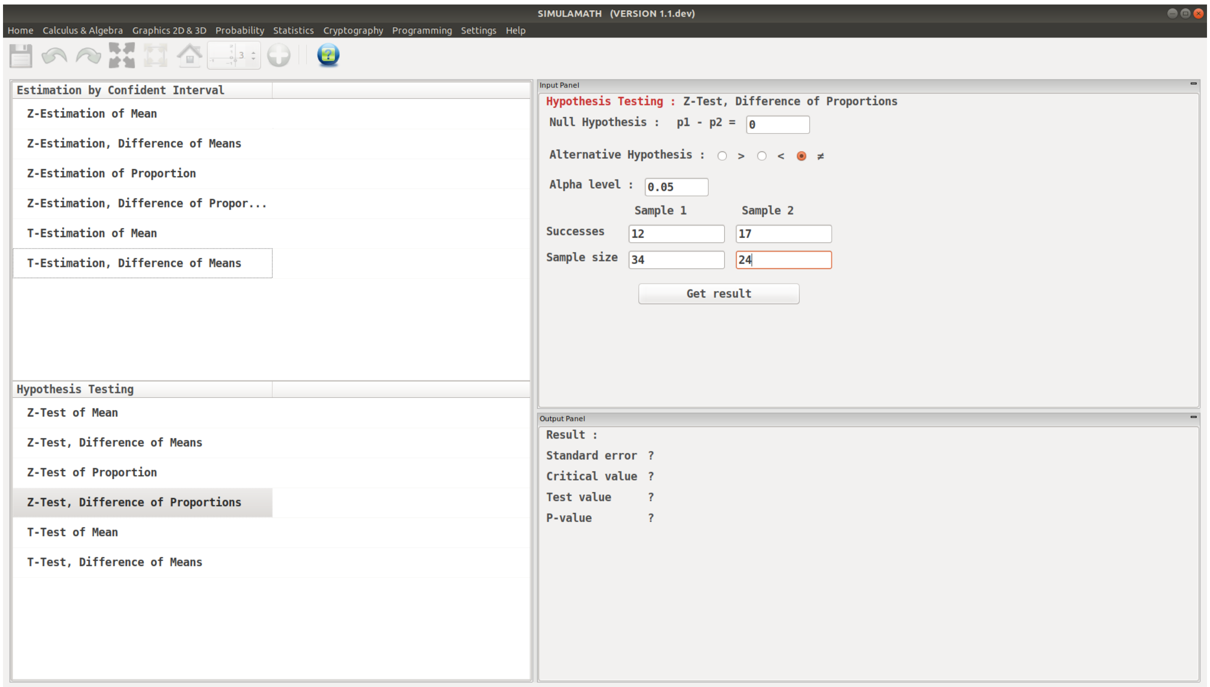
Step 5: Summarize the results. There is enough evidence to reject the claim that there is no difference in the proportions of small and large nursing homes with a resident vaccination rate of less than 80%.

Z-Test, Difference of Proportions in Simulamath

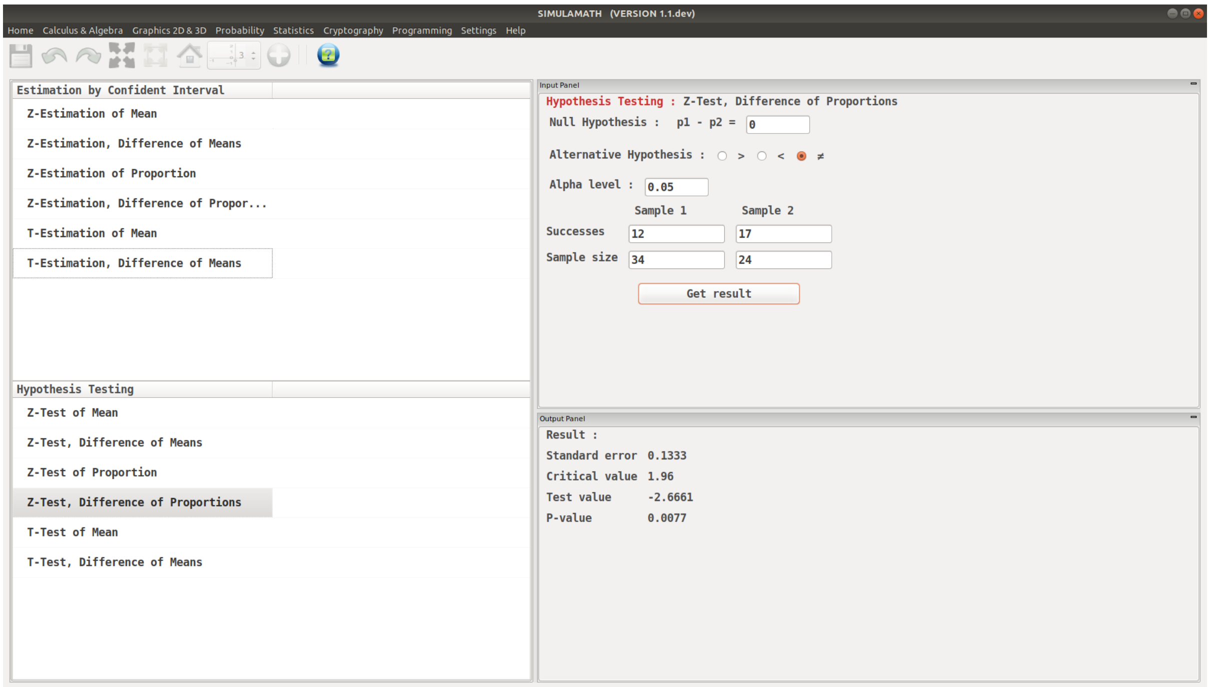
Choose the type of test/estimation you want to compute in the panels on the left hand side, here **Z-Estimation, Difference of Proportions**



Enter the variables (Null Hypothesis, Alternative Hypothesis, Alpha value, Success, Sample size) for each sample in the top panel on the right hand side.



Click on **Get result** located below inside the same panel. Voila, you have your results in the panel down on the right hand side.



T-Test for a Mean

The formula for hypothesis testing for T-Test for a Mean is:

$$t = \frac{\bar{X} - \mu}{s/\sqrt{n}}$$

This formula is based on the general format of

$$\text{Test value} = \frac{(\text{observed value}) - (\text{expected value})}{\text{standard error}}$$

Assumptions for the t-Test for a Mean When σ is unknown

1. The sample is a random sample.
2. Either $n \geq 30$ or the population is normally distributed if $n < 30$.

Example:

A medical investigation claims that the average number of infections per week at a hospital in southwestern Pennsylvania is 16.3. A random sample of 10 weeks had a mean number of 17.7 infections. The sample standard deviation is 1.8. Is there enough evidence to reject the investigator's claim at $\alpha = 0.05$? Solution:

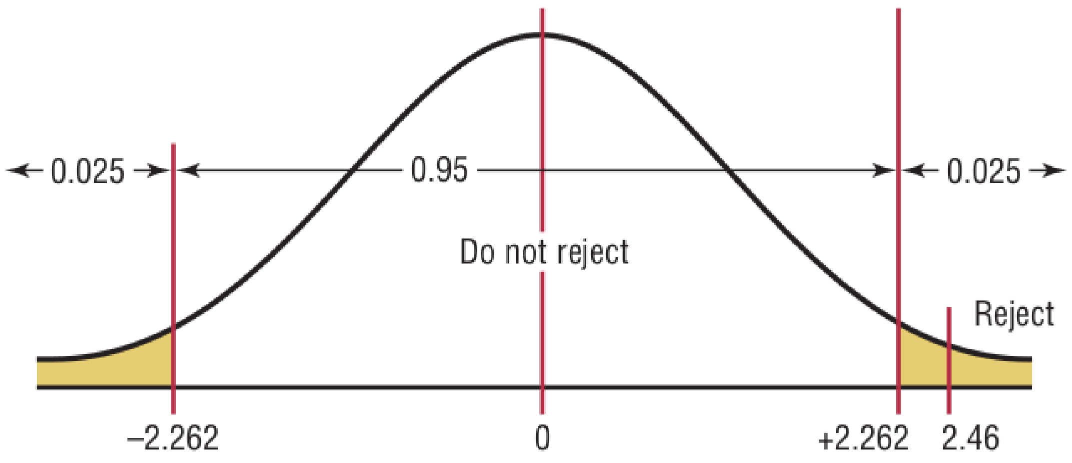
Step 1: $H_0 : \mu = 16.3 \quad \text{(claim)} \quad \text{and} \quad H_1 : \mu \neq 16.3.$

Step 2: The critical values are +2.262 and -2.262 for $\alpha = 0, 05$ and $d.f = 9$.

Step 3: The test value is

$$t = \frac{\bar{X} - \mu}{s/\sqrt{n}} = \frac{17.7 - 16.3}{1.8/\sqrt{10}} = 2.46$$

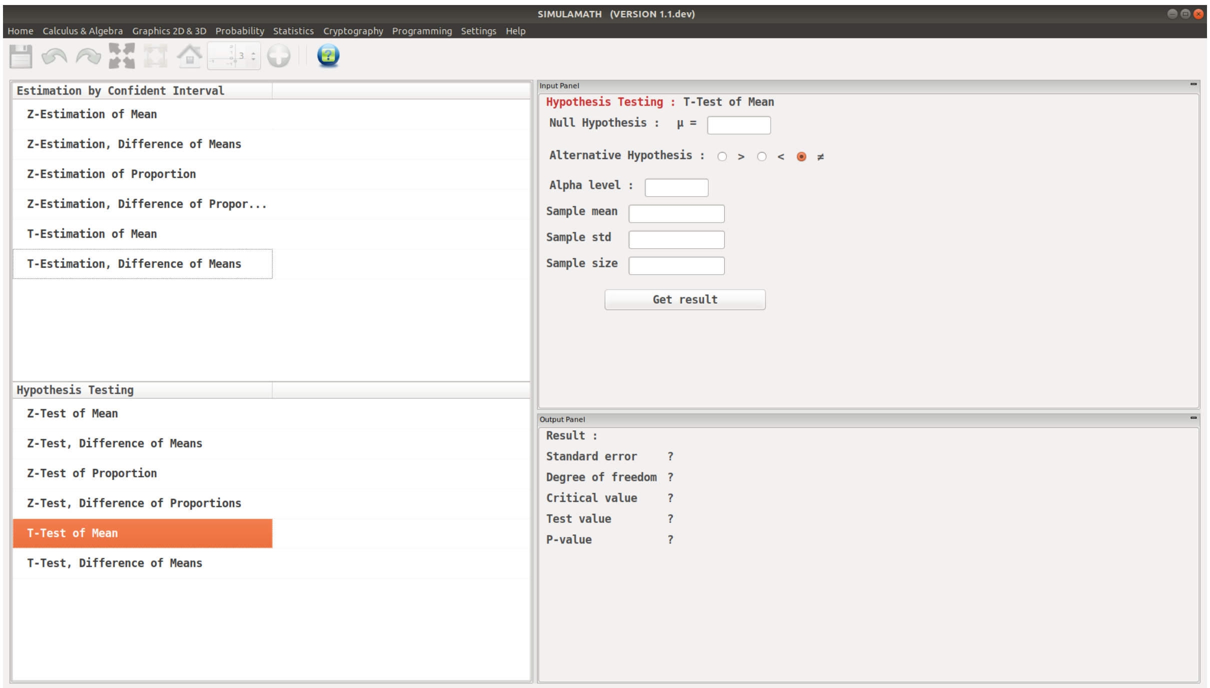
Step 4: Reject the null hypothesis since $2.46 > 2.262$.



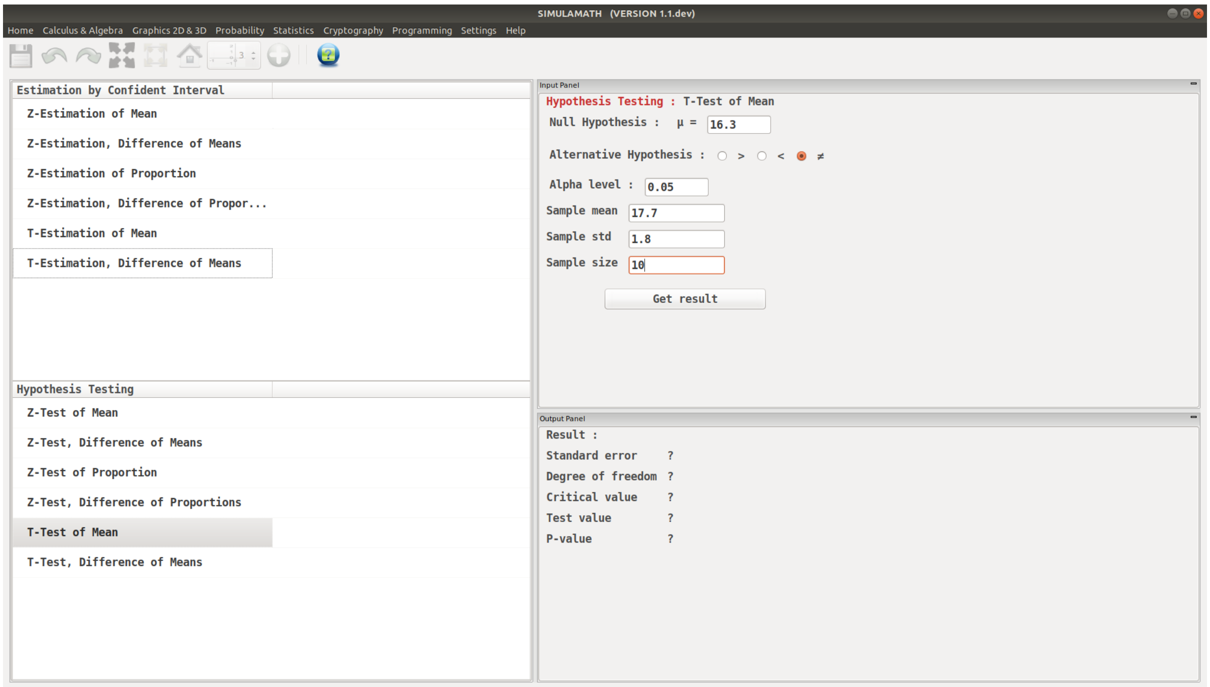
Step 5: There is enough evidence to reject the claim that the average number of infections is 16.3.

T-Test for a Mean in Simulamath

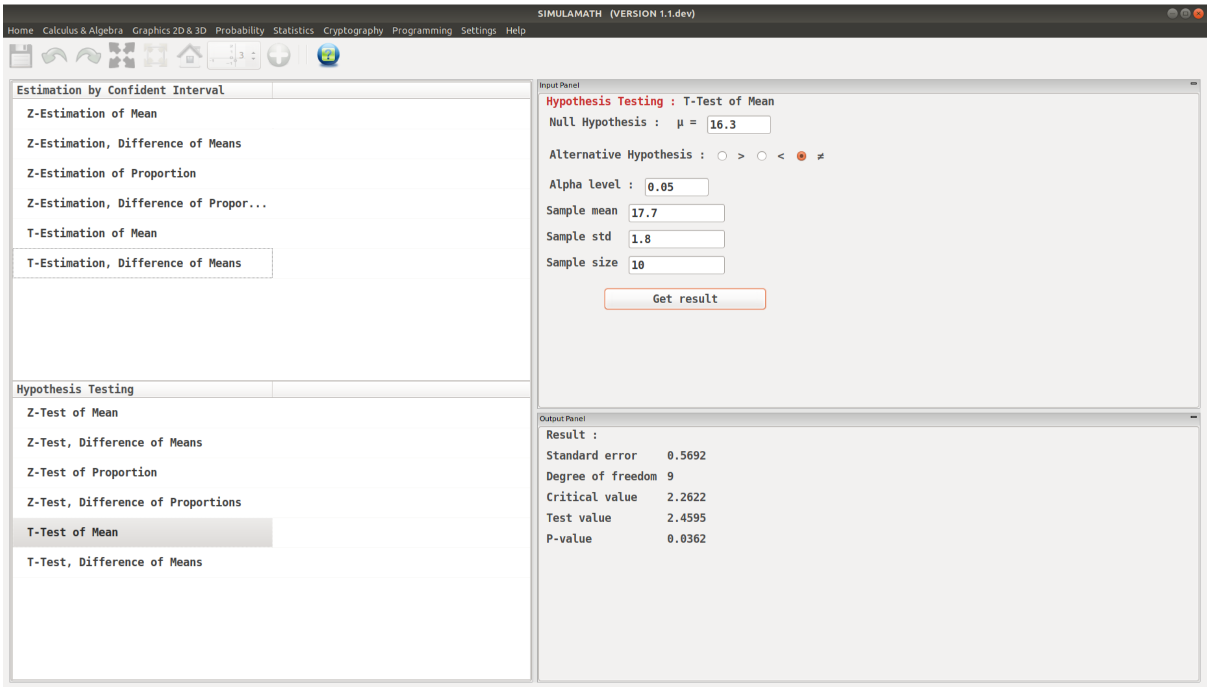
Choose the type of test/estimation you want to compute in the panels on the left hand side, here **T-Estimation of Mean**



Enter the variables (Null Hypothesis, Alternative Hypothesis, Alpha value, Sample mean, Standard deviation of the population, Sample size) in the top panel on the right hand side.



Click on **Get result** located below inside the same panel. Voila, you have your results in the panel down on the right hand side.



T-Test, Difference of Means

The formula for hypothesis testing for T-Test, Difference of Means is:

$$t = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_1)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

This formula is based on the general format of

$$\text{Test value} = \frac{(\text{observed value}) - (\text{expected value})}{\text{standard error}}$$

Assumptions for the t-Test for two independent Means when σ_1 and σ_2 are unknown

1. The samples are random samples.
2. The sample data are independent of one another.
3. When the sample sizes are less than 30, the populations must be normally or approximately normally distributed.

Example:

The average size of a farm in Indiana County, Pennsylvania, is 191 acres. The average size of a farm in Greene County, Pennsylvania, is 199 acres. Assume the data were obtained from two samples with standard deviations of 38 and 12 acres, respectively, and sample sizes of 8 and 10, respectively. Can it be concluded at $\alpha = 0.05$ that the average size of the farms in the two counties is different? Assume the populations are normally distributed.

Solution:

Etape 1: State the hypotheses and identify the claim for the means.

$$H_0 : \mu_1 = \mu_2 \quad \text{and} \quad H_1 : \mu_1 \neq \mu_2 \quad \text{(claim)}$$

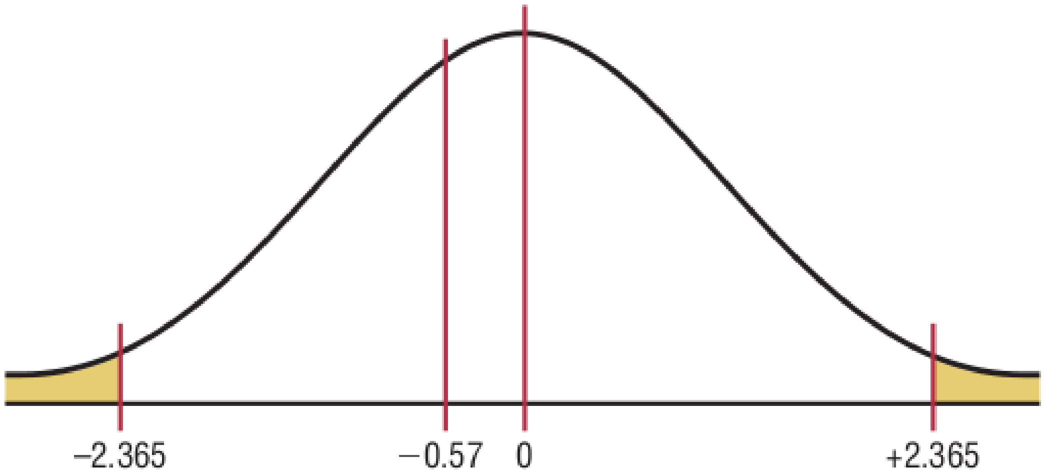
Etape 2: Find the critical values. Since the test is two-tailed, since $\alpha = 0.05$, and since the variances are unequal, the degrees of freedom are the smaller of $n_1 - 1$ or $n2 - 1$. In this case, the degrees of freedom are $8 - 1 = 7$. Hence, from Table F, the critical values are +2.365 and -2.365.

Etape 3: Compute the test value. Since the variances are unequal, use the first formula.

$$t = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_1)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

$$t = \frac{(191 - 199) - 0}{\sqrt{\frac{38^2}{8} + \frac{12^2}{10}}} = -0.57$$

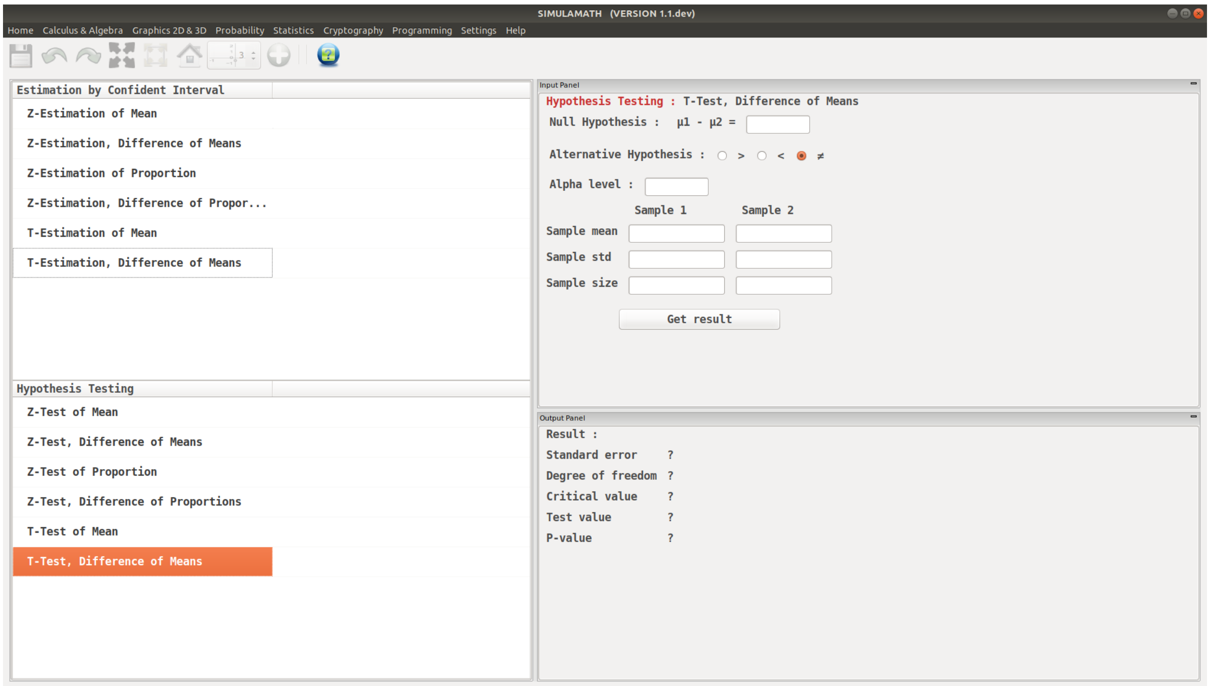
Stape 4: Make the decision. Do not reject the null hypothesis, since $-0.57 > -2.365$



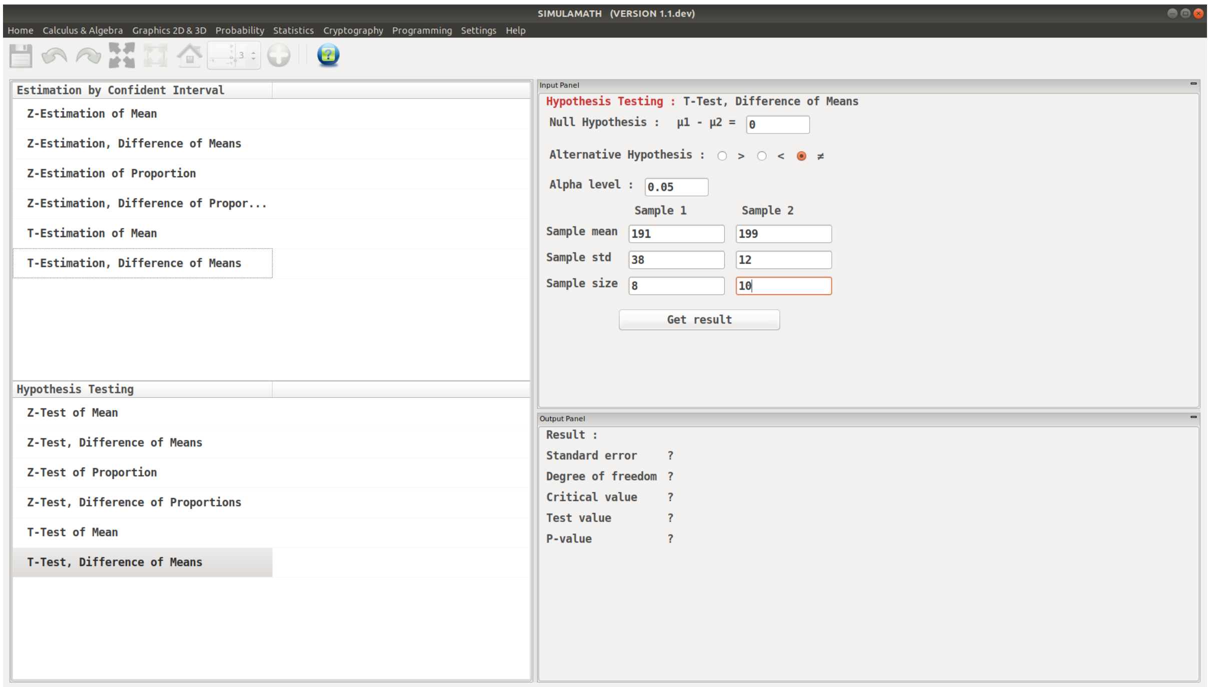
Stape 5: Summarize the results. There is not enough evidence to support the claim that the average size of the farms is different.

T-Test, Difference of Means in Simulamath

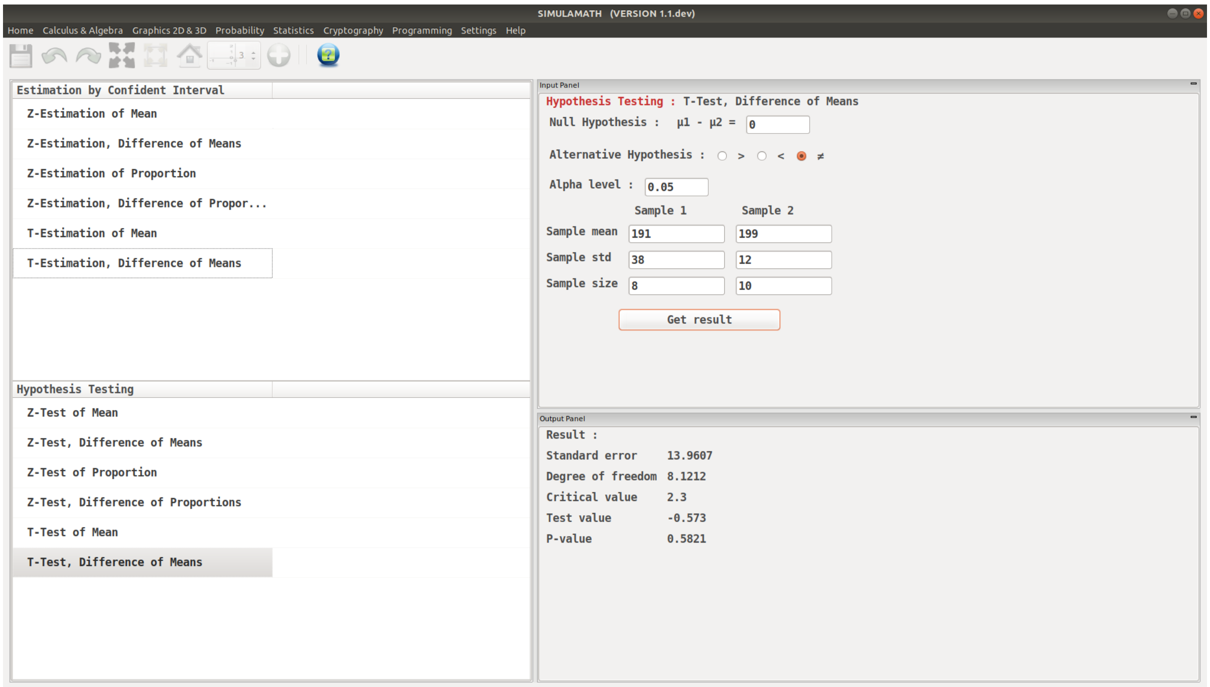
Choose the type of test/estimation you want to compute in the panels on the left hand side, here **T-Estimation, Difference of Means**



Enter the variables (Null Hypothesis, Alternative Hypothesis, Alpha value, Sample mean, Standard deviation of the population, Sample size) for each sample in the top panel on the right hand side.



Click on **Get result** located below inside the same panel. Voila, you have your results in the panel down on the right hand side.



4.5 Graphics in 2D

SimulaMath has a rich area for graphics 2D. You can do the following :

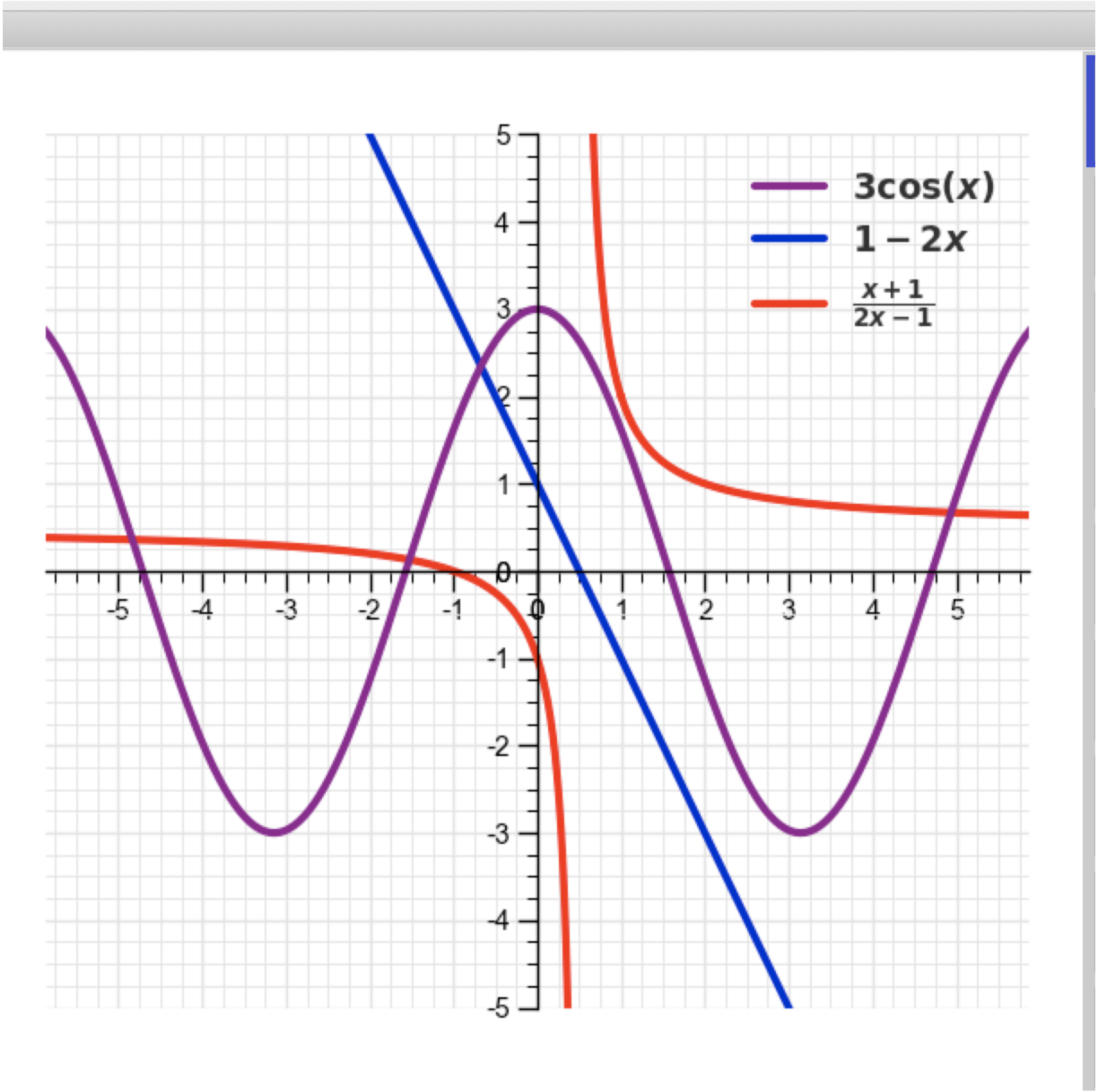
- graph of a function given an expression $f(x)$
- graph of a function given by an implicit equation $f(x, y) = 0$
- area given by an implicit equation $f(x, y) > 0$, $f(x, y) >= 0$, $f(x, y) < 0$ and $f(x, y) <= 0$.
- graph of a parametric function $x(t), y(t)$.
- geometric construction for about 100 objects: line, ray, segment, cercle, polygon, . . .
- choose different types of themes
- customize the theme of your graphics

Note: you can also plot your graph by using your own Simula code.

4.5.1 Functions f(x)

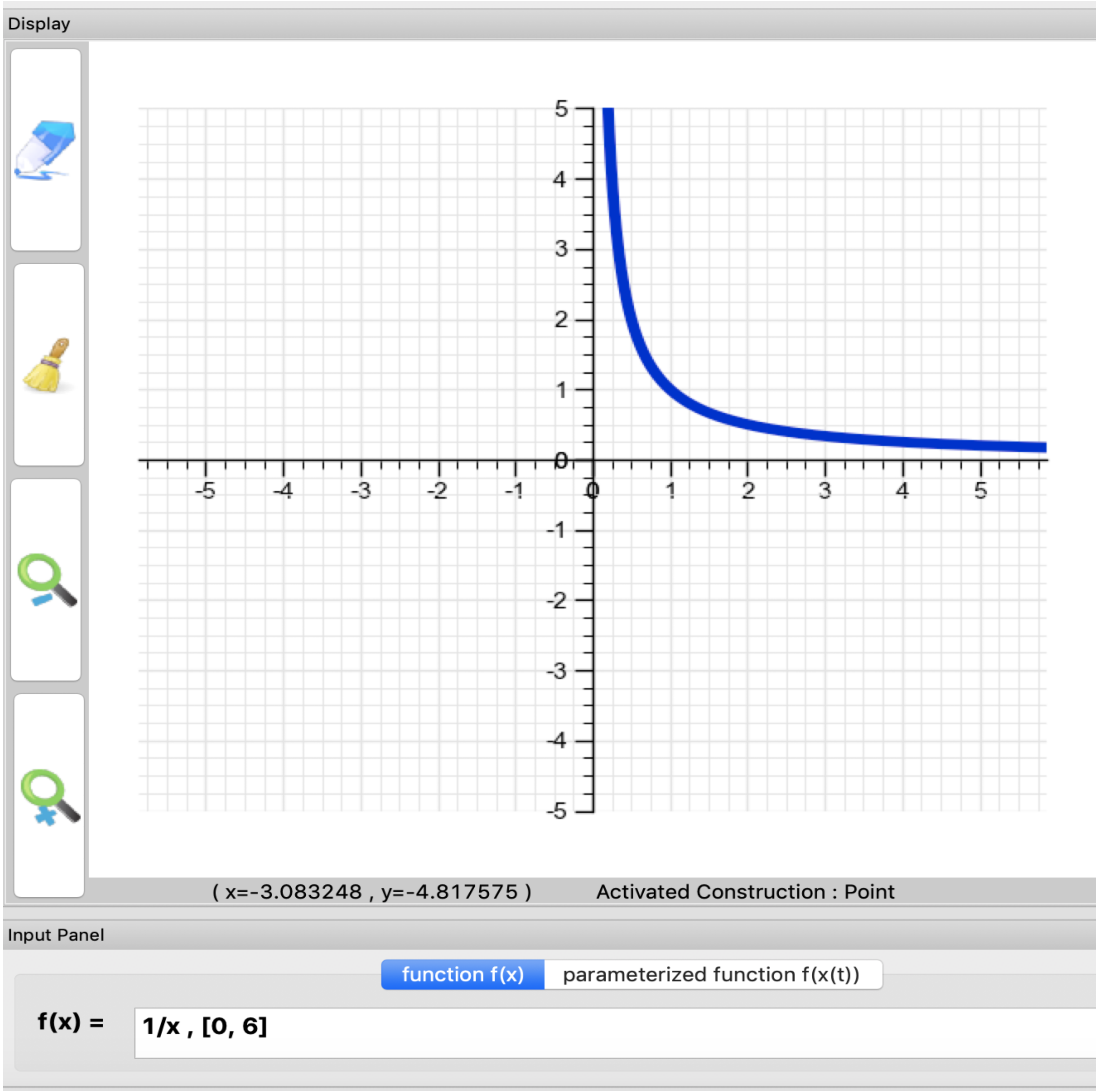
Let us plot the following graphs:

- $3 \cos(x)$
- $1 - 2x$
- $\frac{x+1}{2x-1}$



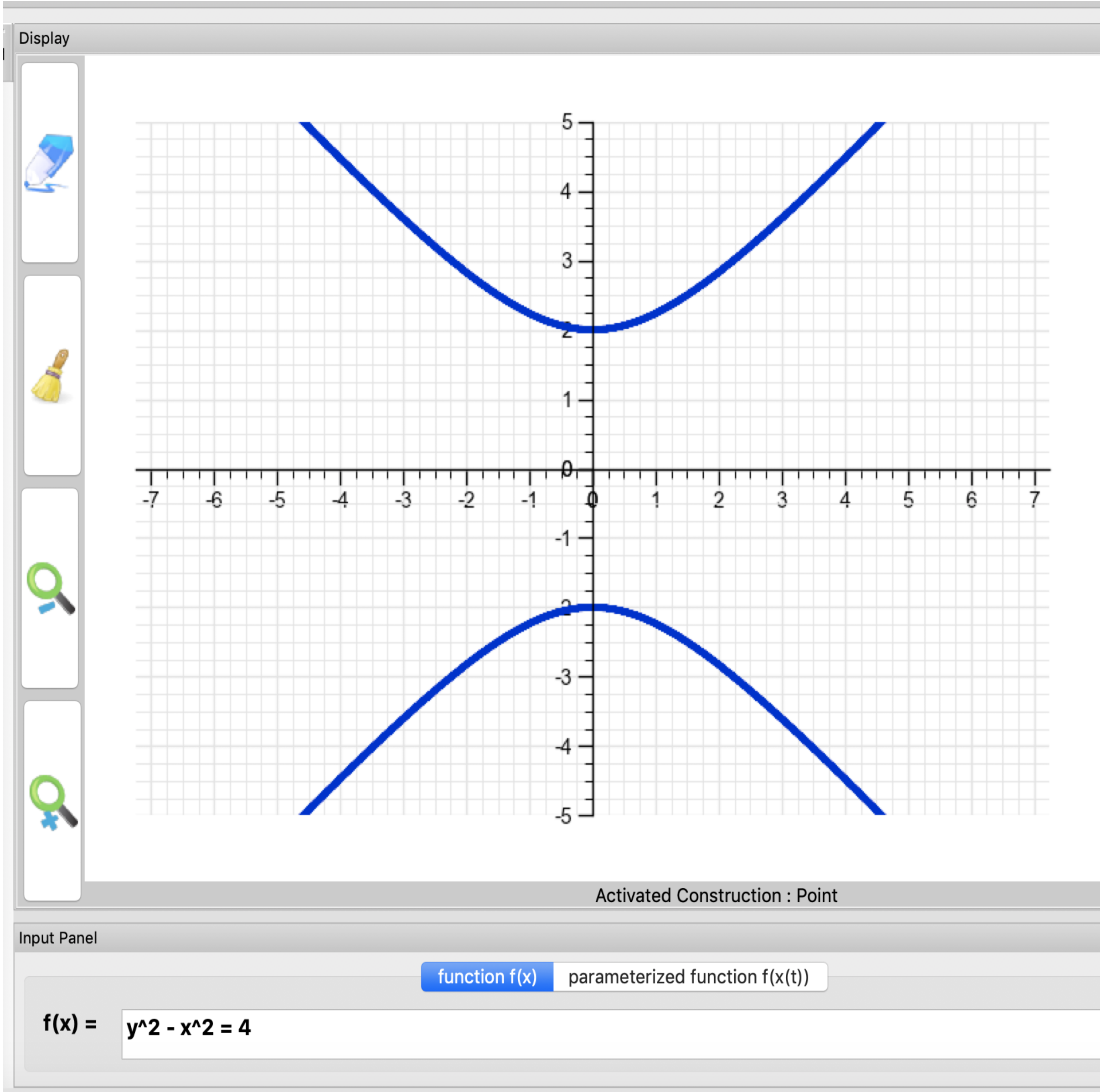
Simula allows you to plot a function on a specific interval. To do this, you must first enter the expression of the function followed by a comma and then the interval.

Let us plot the graph of $\frac{1}{x}$ on the interval $[0, 6]$.

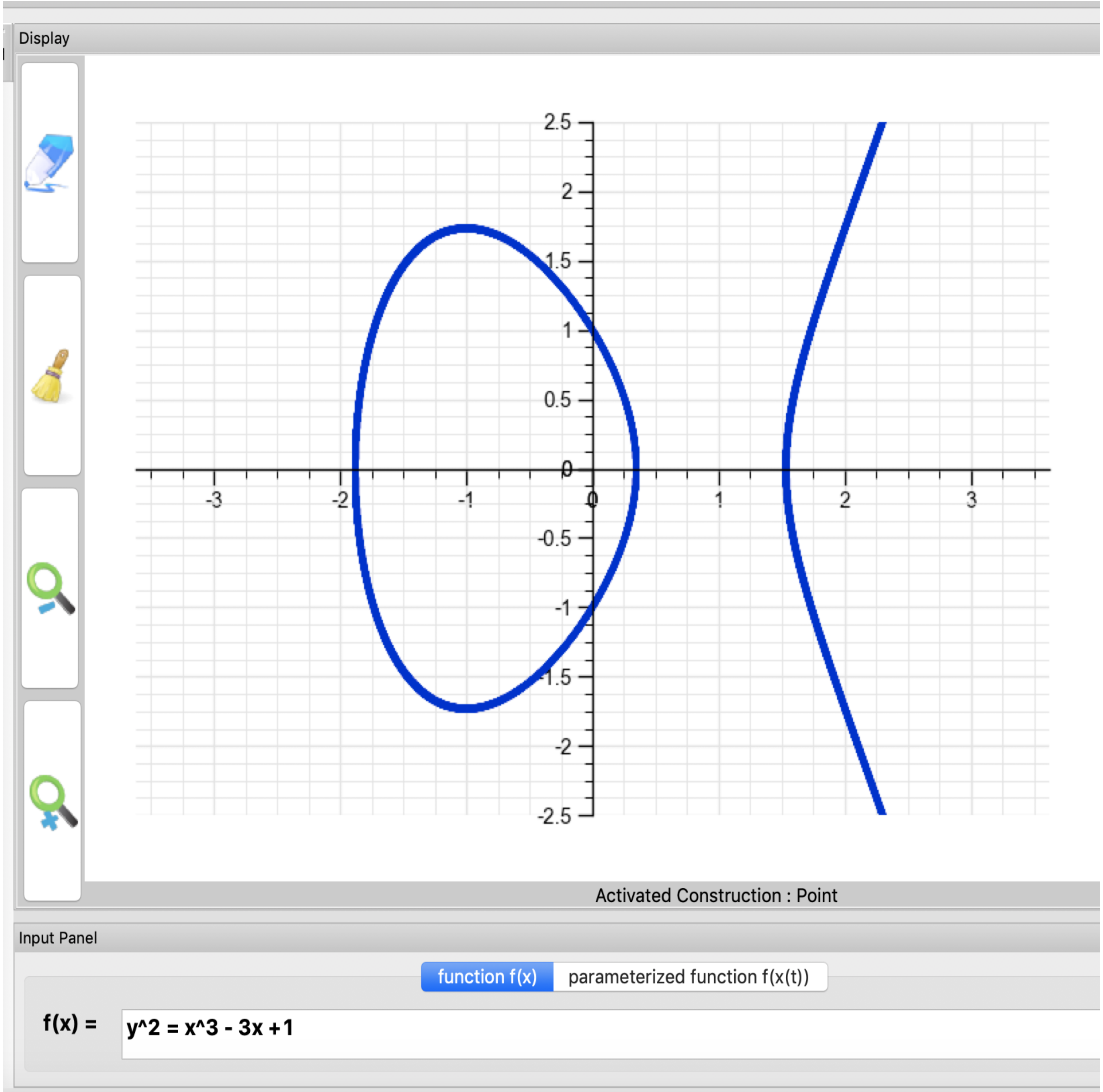


4.5.2 Implicit plots

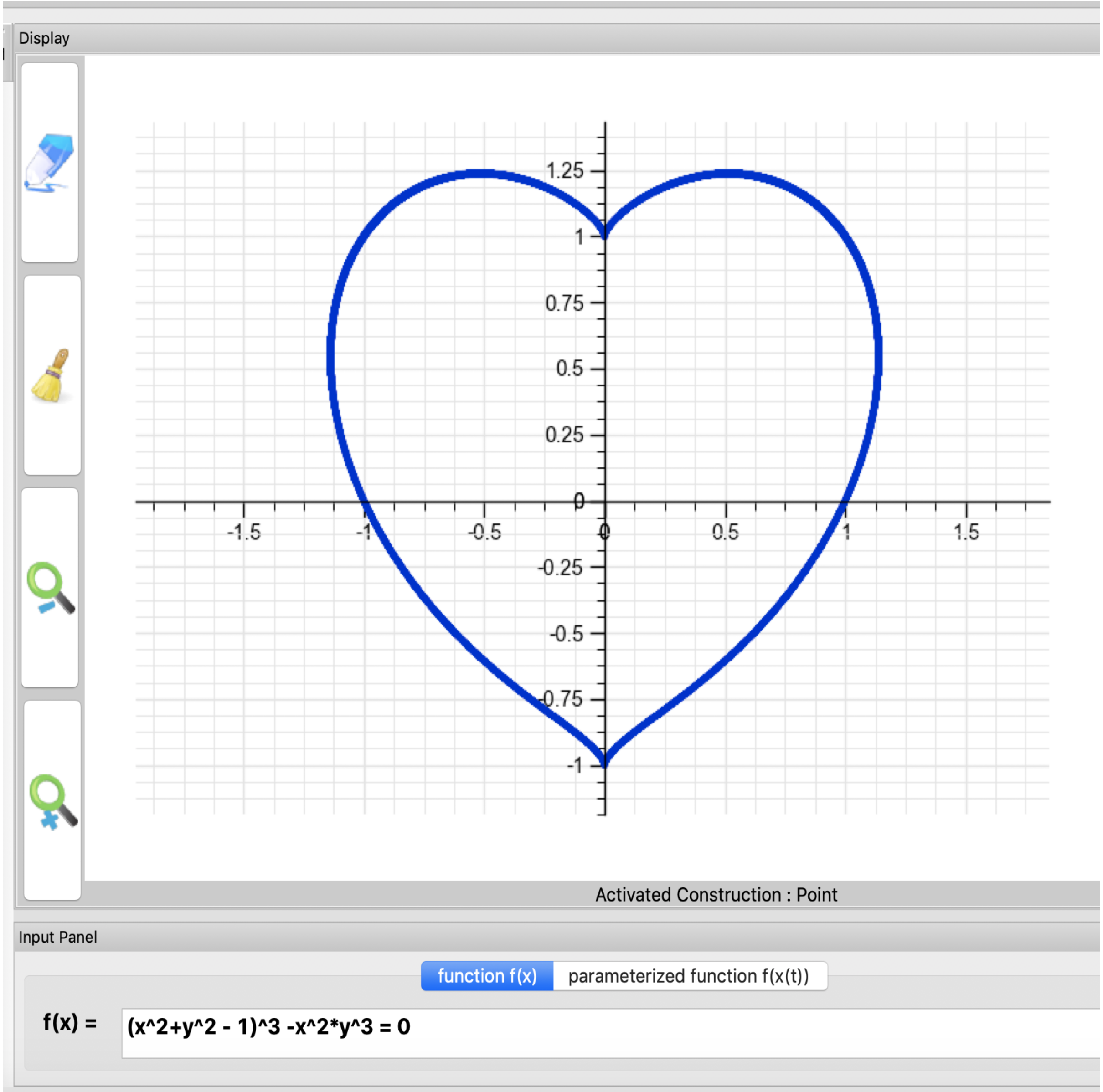
- Graph of $y^2 - x^2 = 4$



- Graph of $y^2 = x^3 - 3x + 1 = 0$

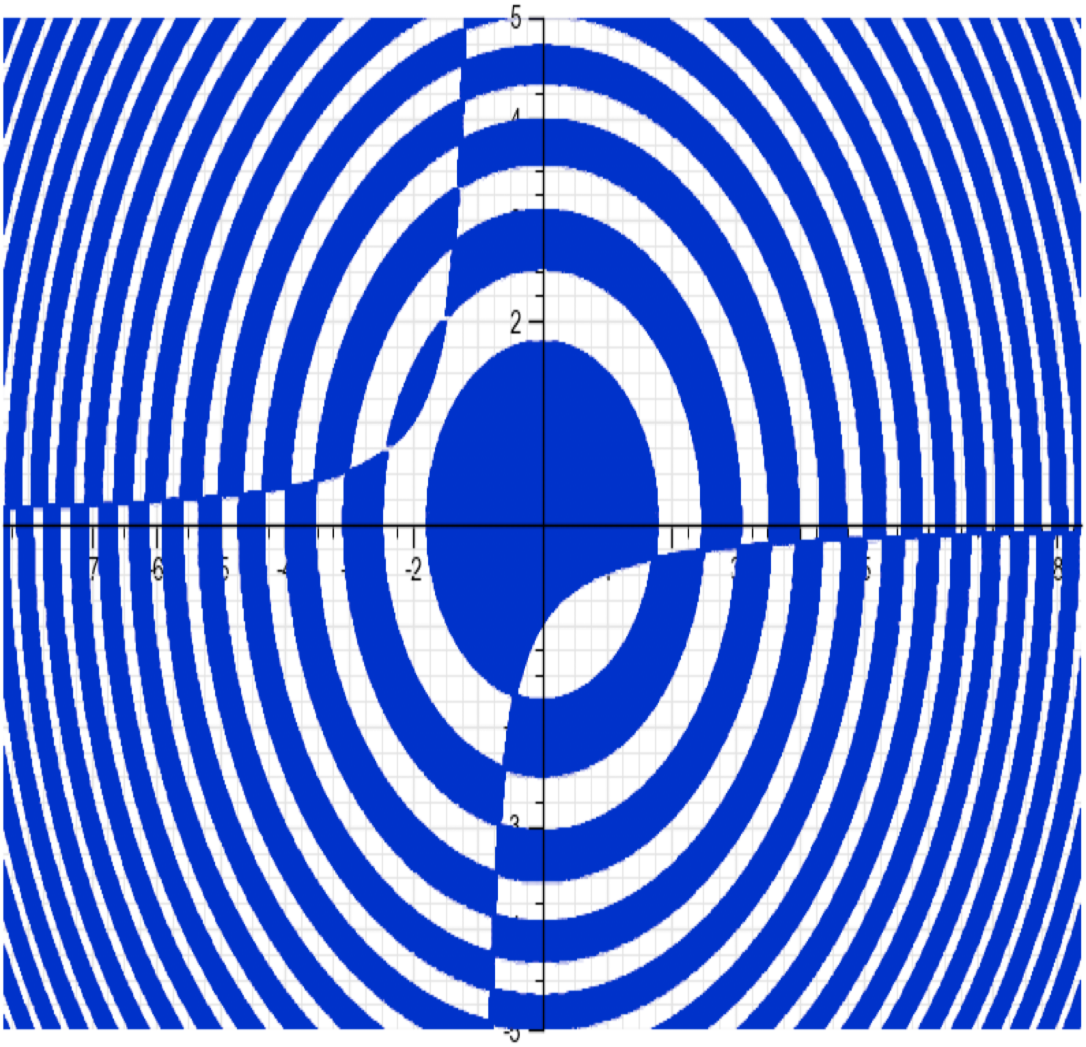


- Graph of $(x^2 + y^2 - 1)^3 - x^2y^3 = 0$



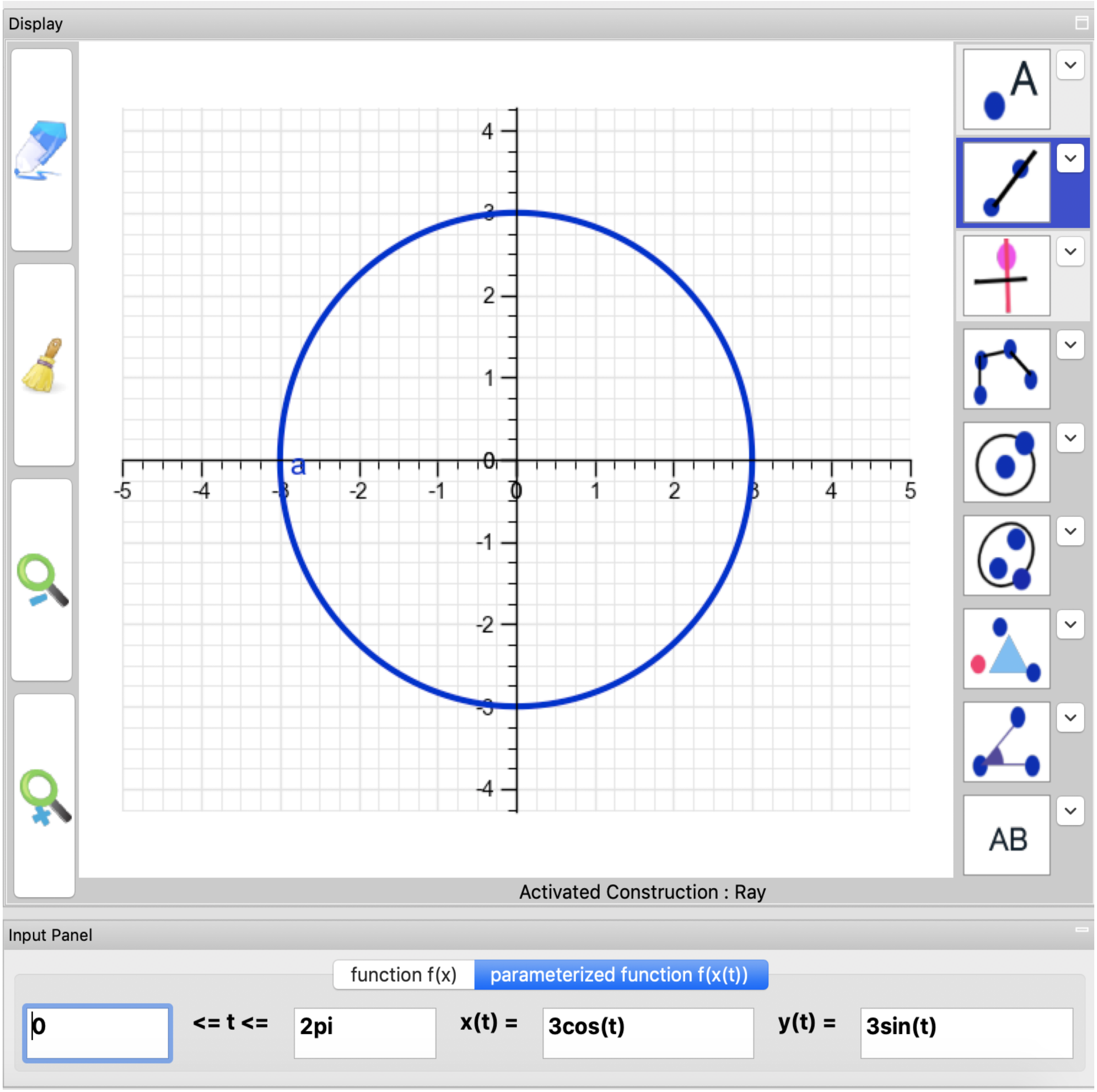
Graph of

$$\frac{sin(x^2+y^2)}{1+y+xy} > 0$$

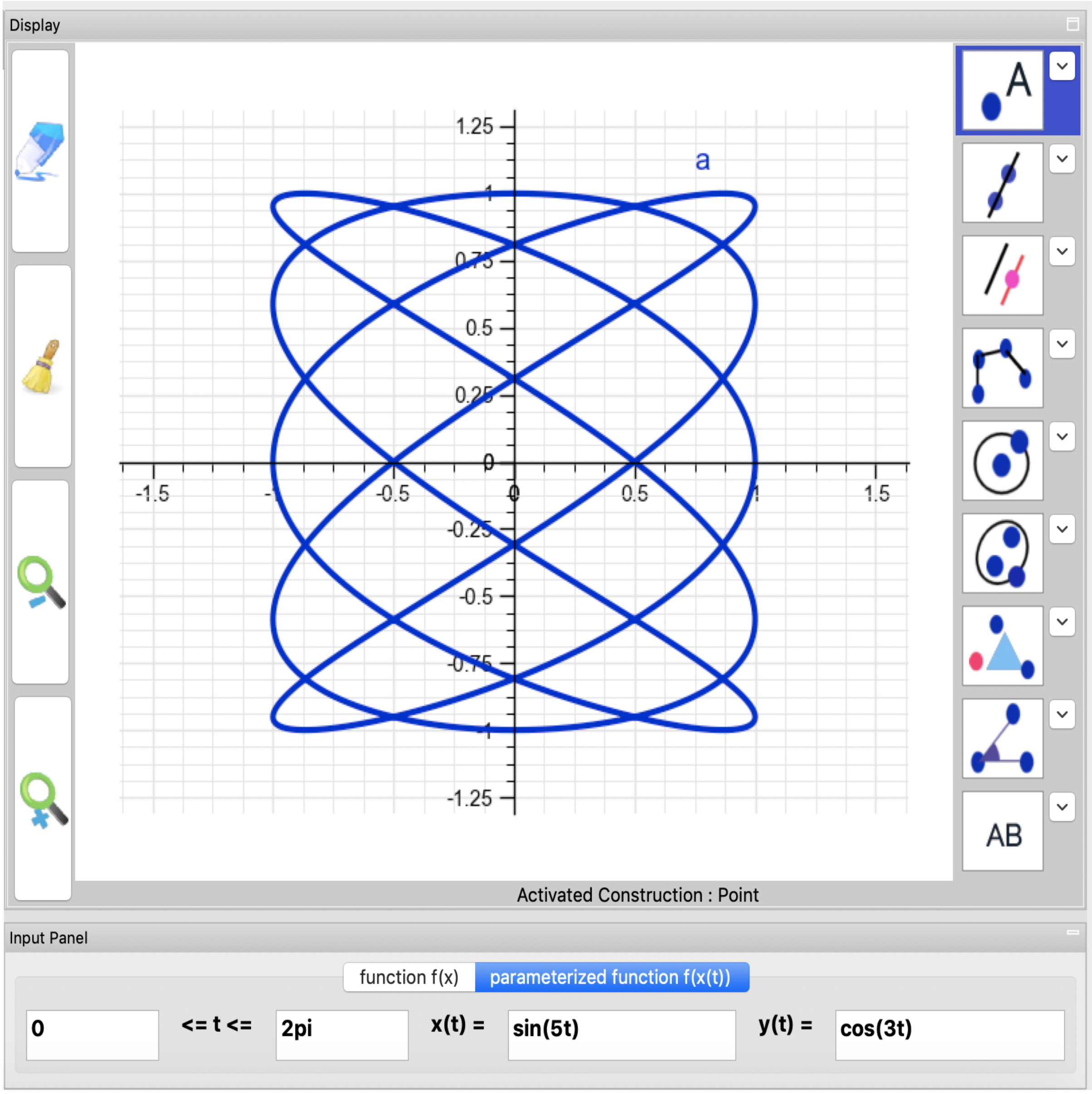


4.5.3 Parametric functions

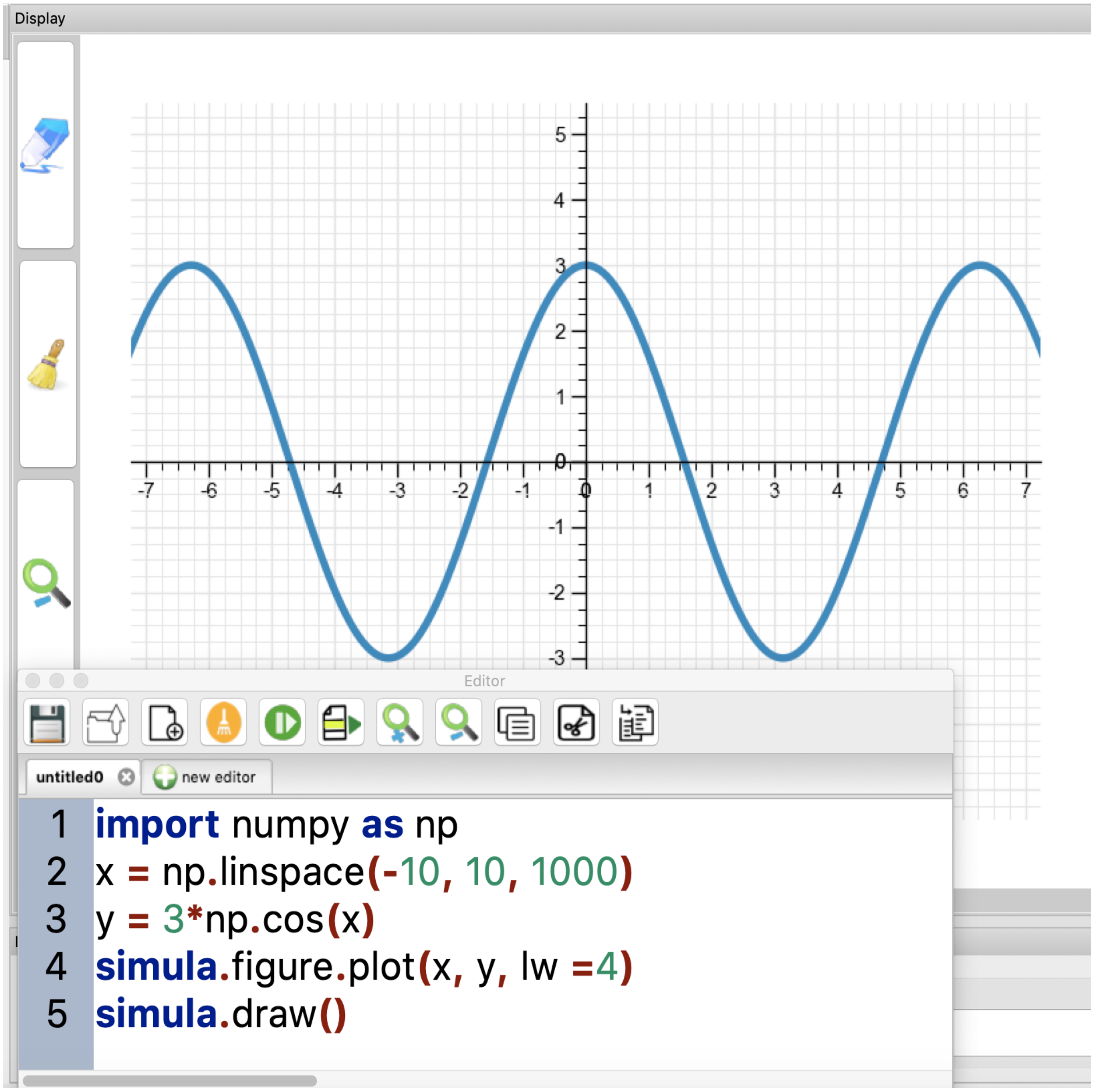
- let us plot the parametric function $x(t) = 3\cos(t); y(t) = 3\sin(t)$ where $t \in [0, 2\pi]$.



- Lissajous figures : $x(t) = \sin(5t); y(t) = \cos(3t)$ where $t \in [0, 2\pi]$.

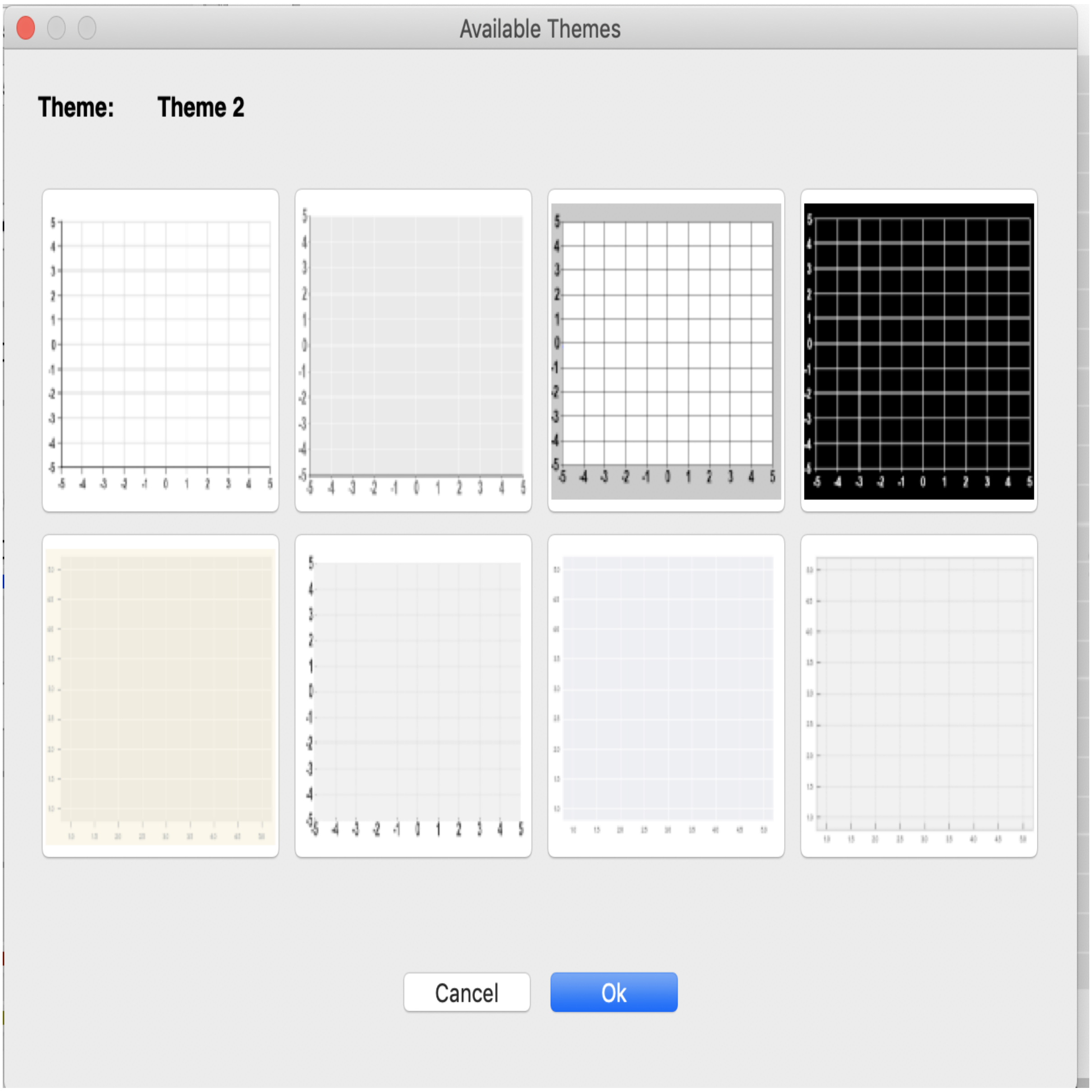


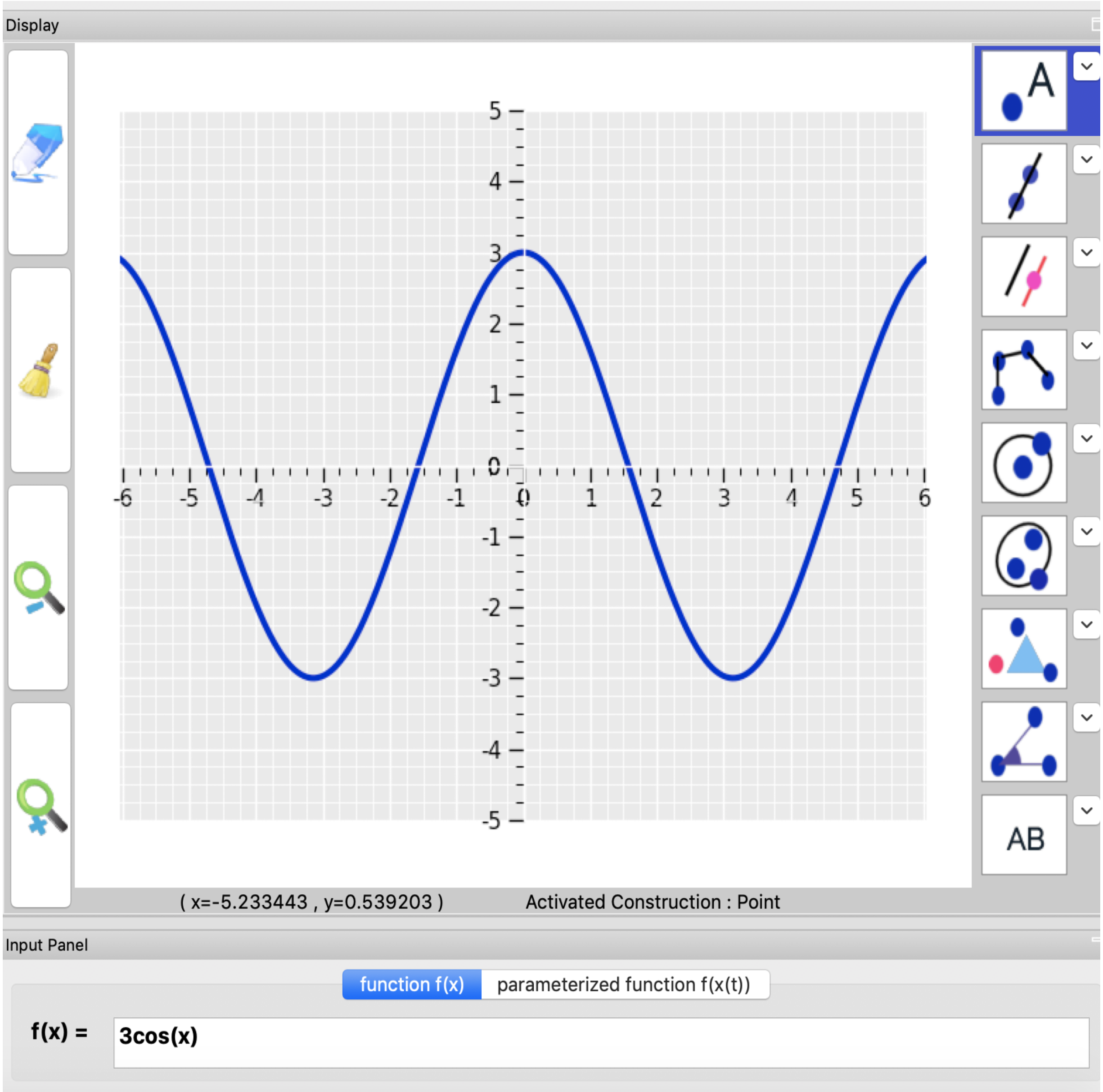
4.5.4 Graphics 2D & Programming



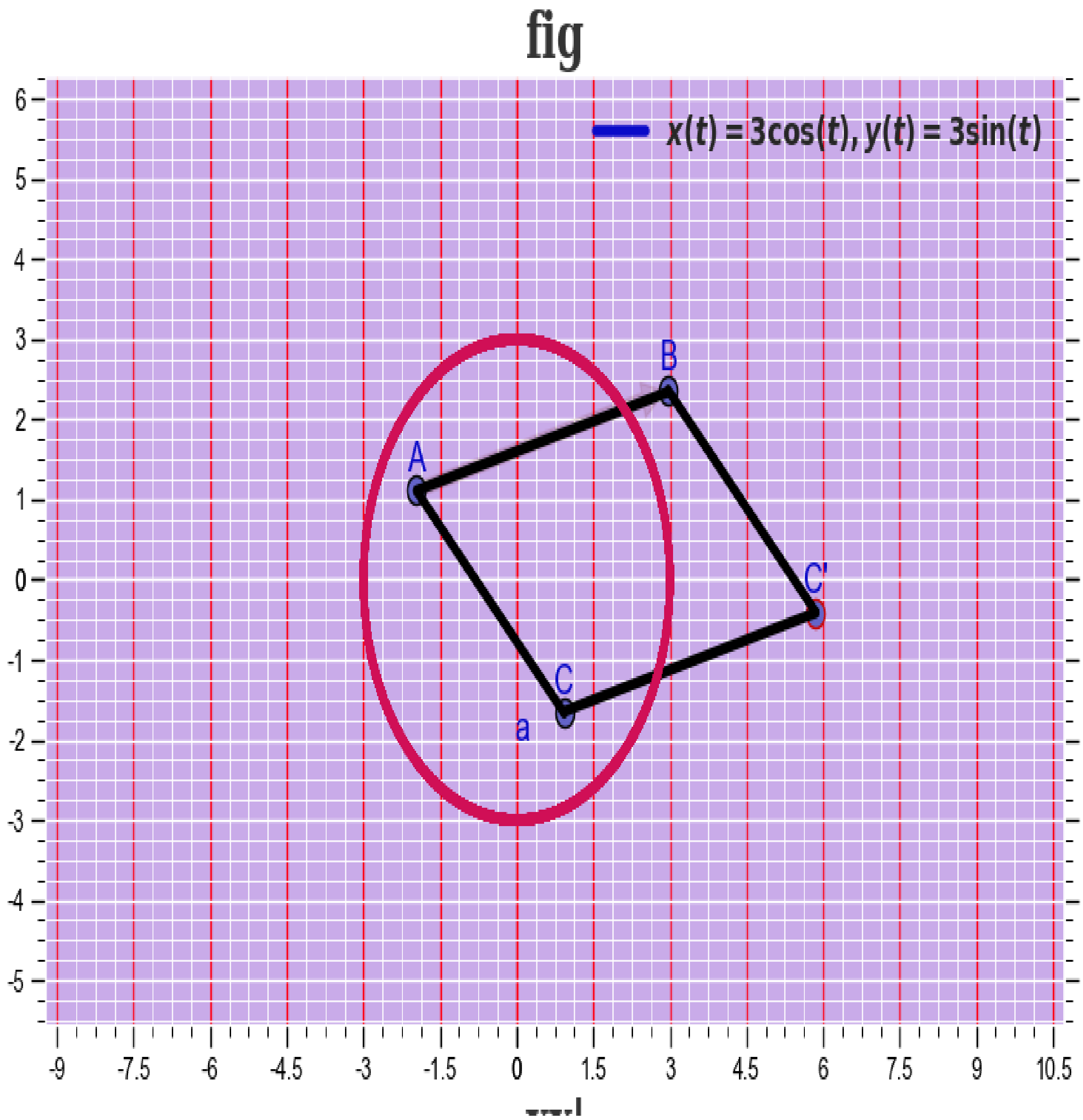
4.5.5 Themes of your graphics

You can choose between 8 customized themes.


Available Themes
Theme: Theme 2
Cancel
Ok



You can also customized your own theme.



4.5.6 Geometry objects in 2D

You can construct about 100 geometric objects:

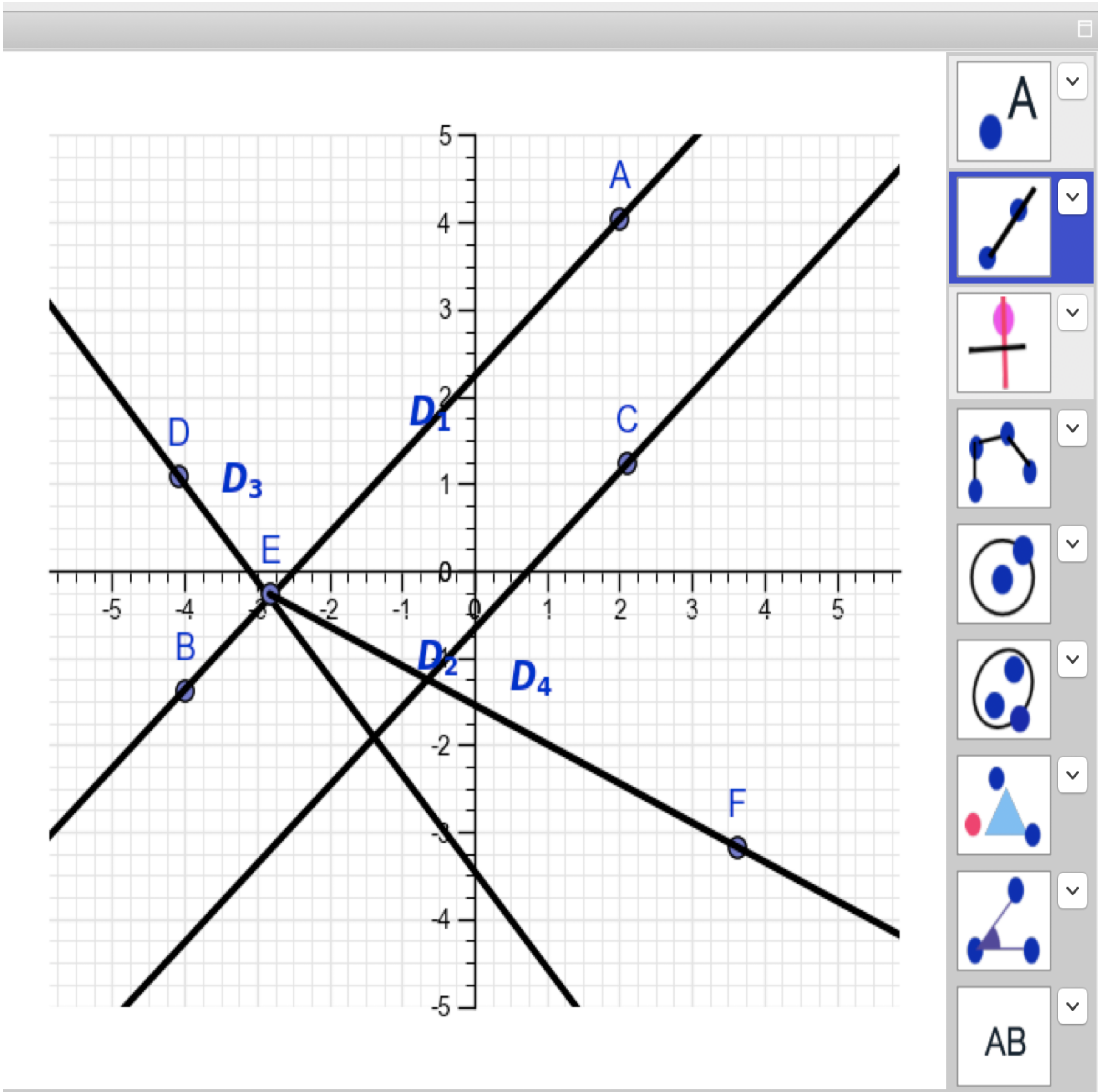
- Points
- Lines,
- Rays,
- Segments,
- Circles,
- Semi-circles,

- Arcs
- Sectors
- Polygons,
- Parallel lines,
- Perpendicular lines,
- Vectors,
- Angles,
- Angle Bisector,
- Ellipsis,
- Parabolas,
- Hyperbolas,
- Rotation,
- Homothety,
- Translation,
- Reflect about a point and a line,
- Areas,
- Barycenter
- Texts,
- Images,
- etc.

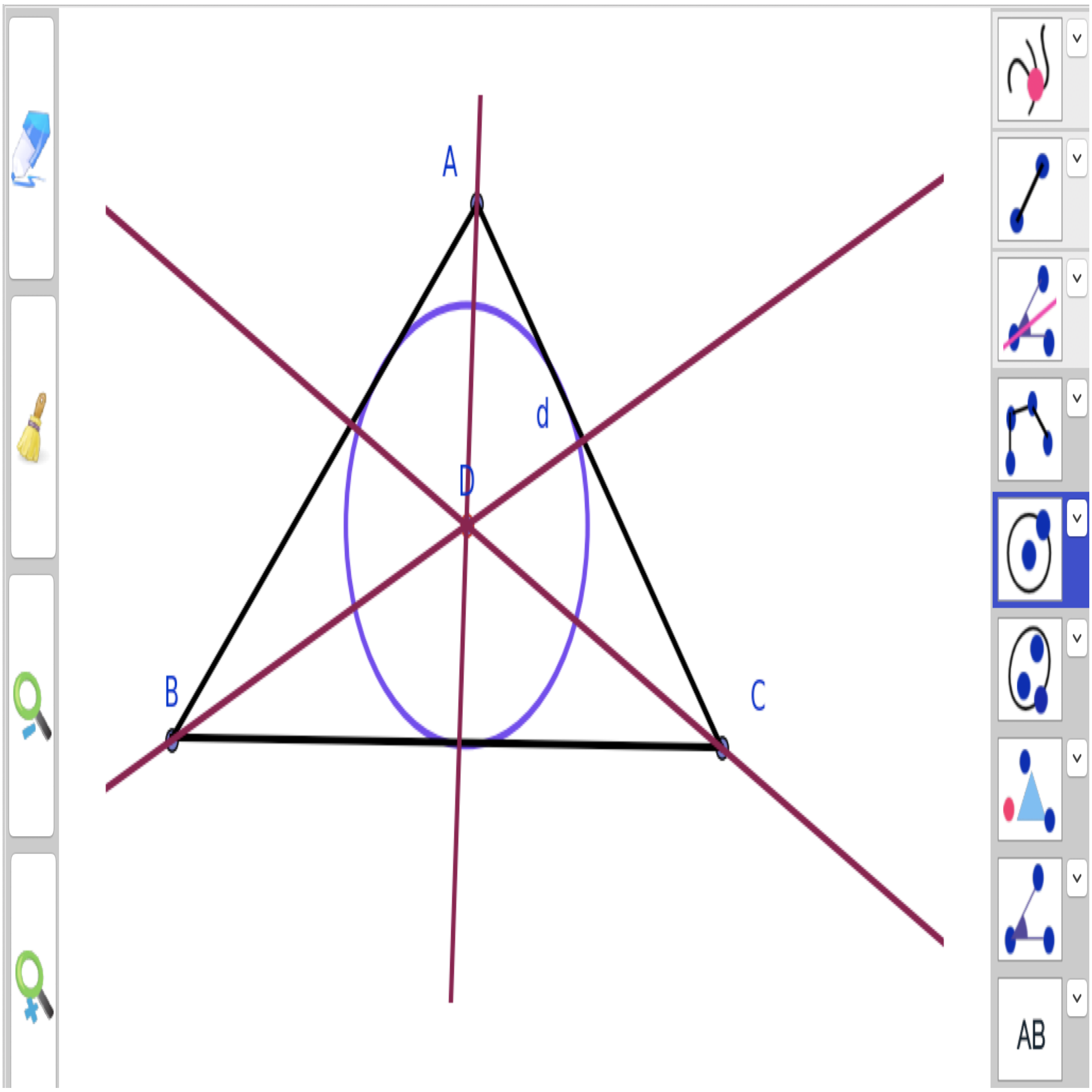
Note:

- All constructions of geometric objects start and end by a double-click.
- For a polygone or barycenter, double-click on the first point, and then a click on each intermediate point, and finally double-clic on the last point.

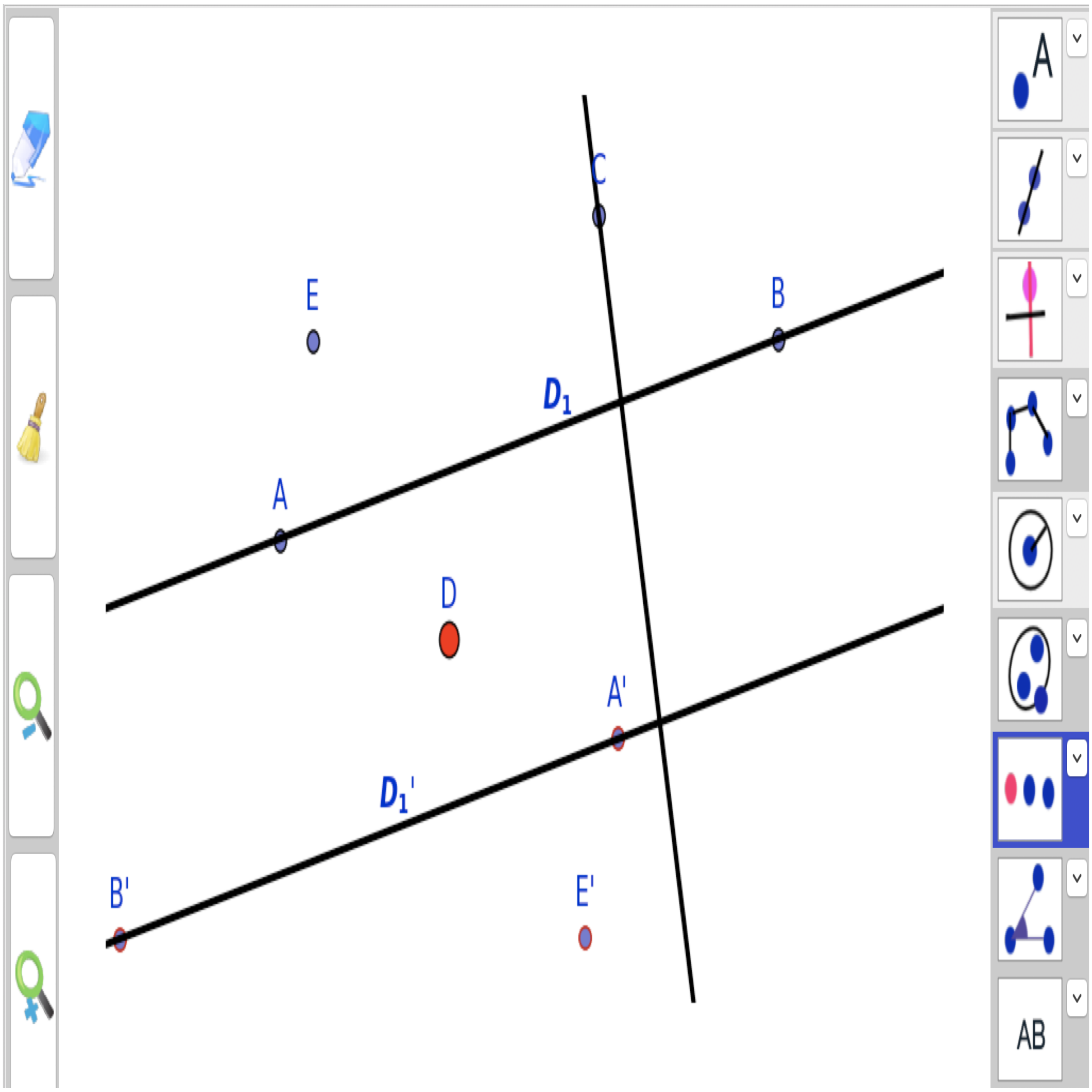
Lines, Segments and Rays



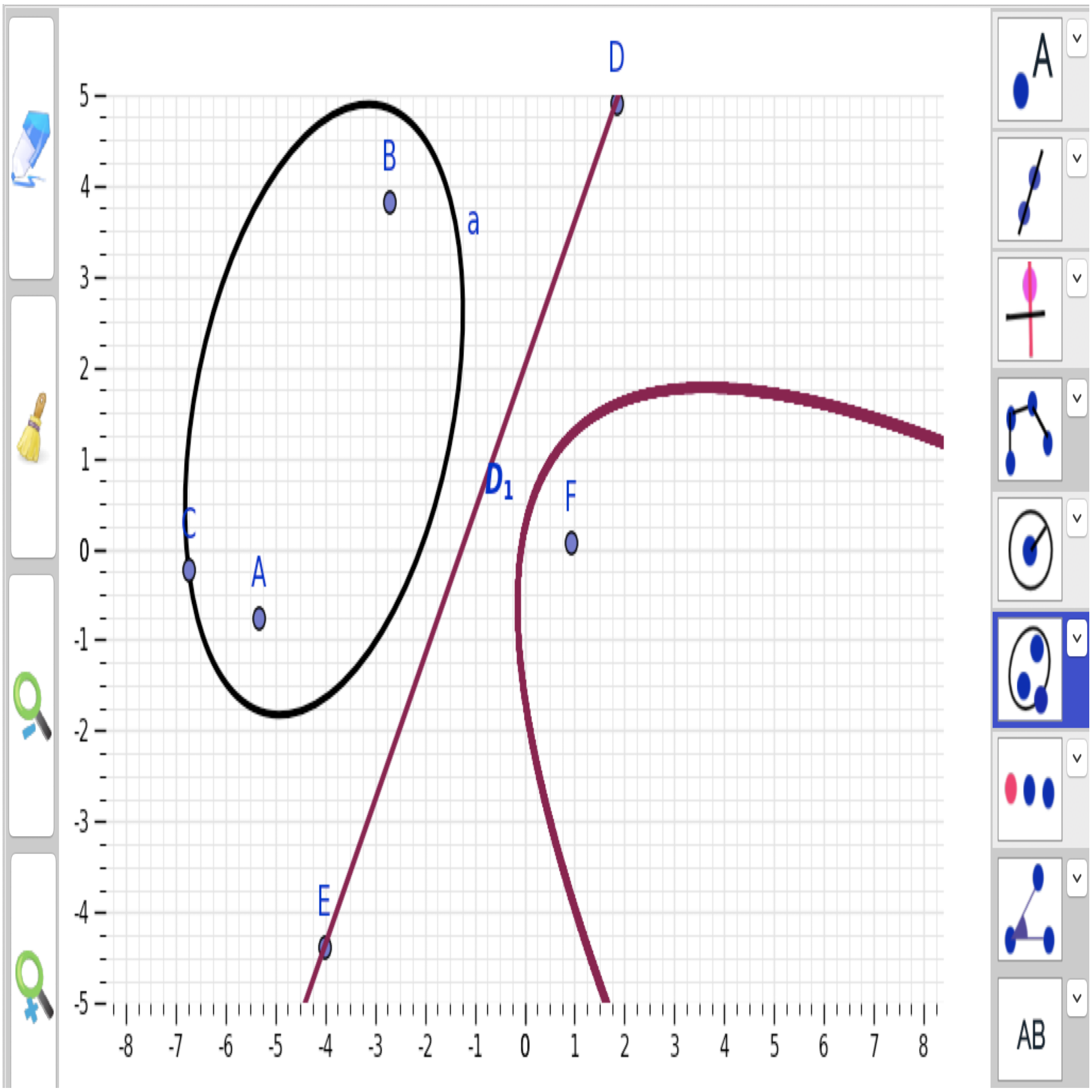
Triangles and Circles



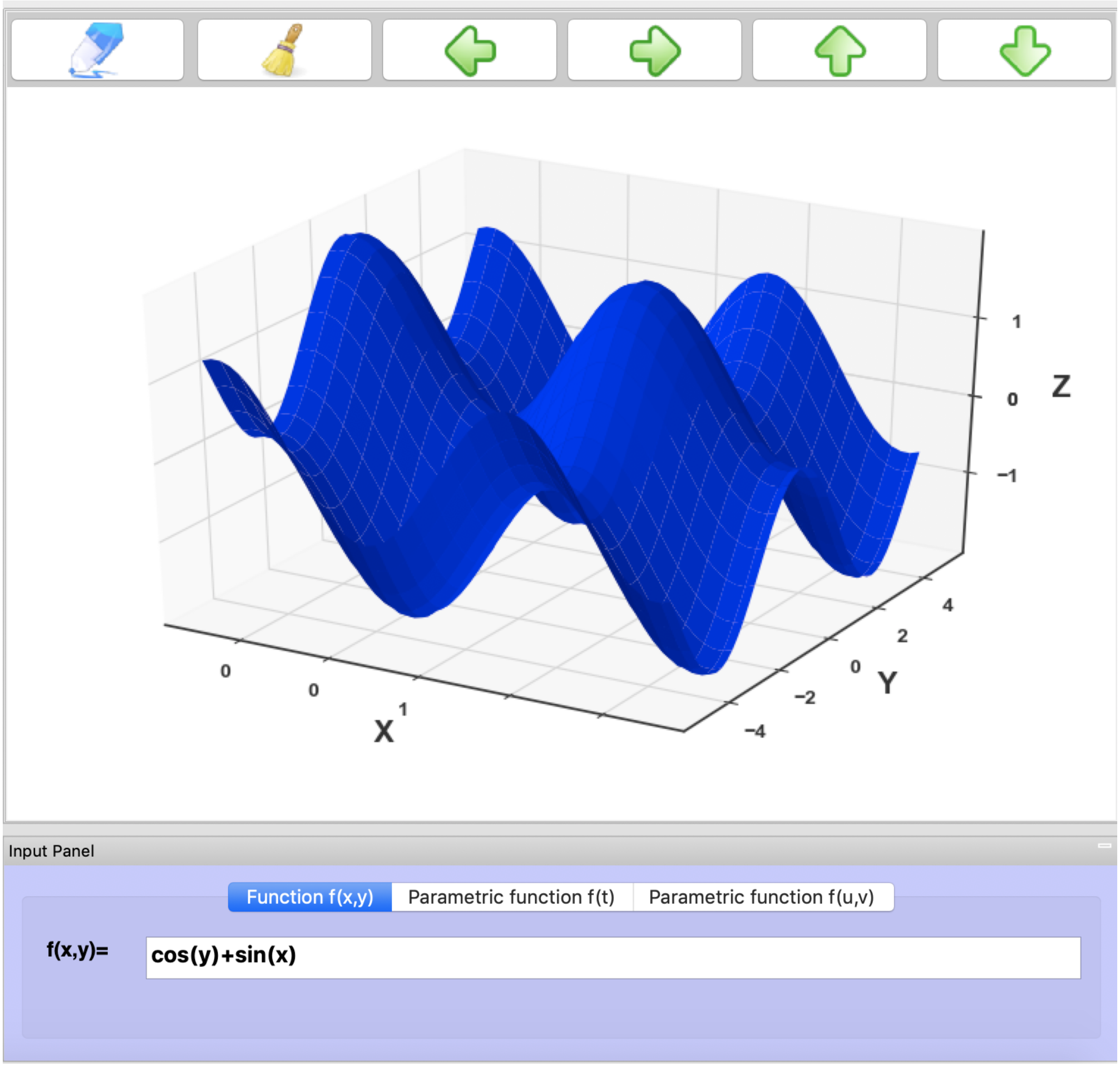
Transformations : Reflects, Rotations, translations and Homothety



Ellipsis, Parabolas and Hyperbolas

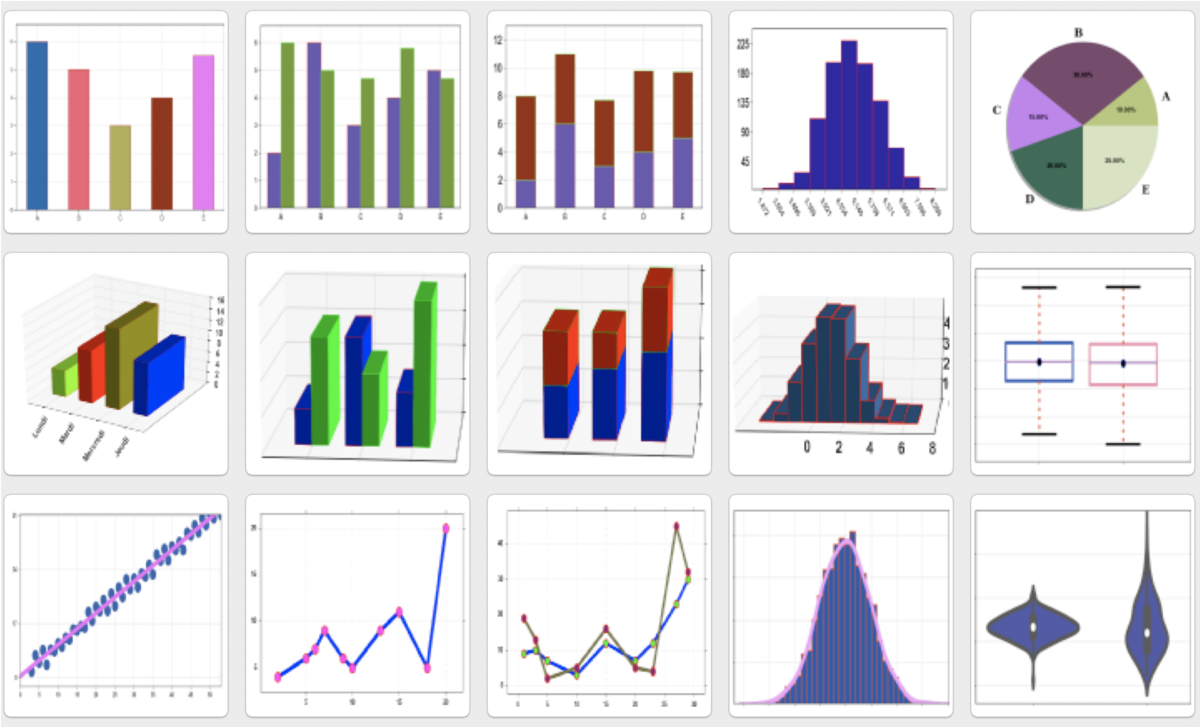

D
B
a
D1
F
C
A
E
AB

4.6 Graphics in 3D



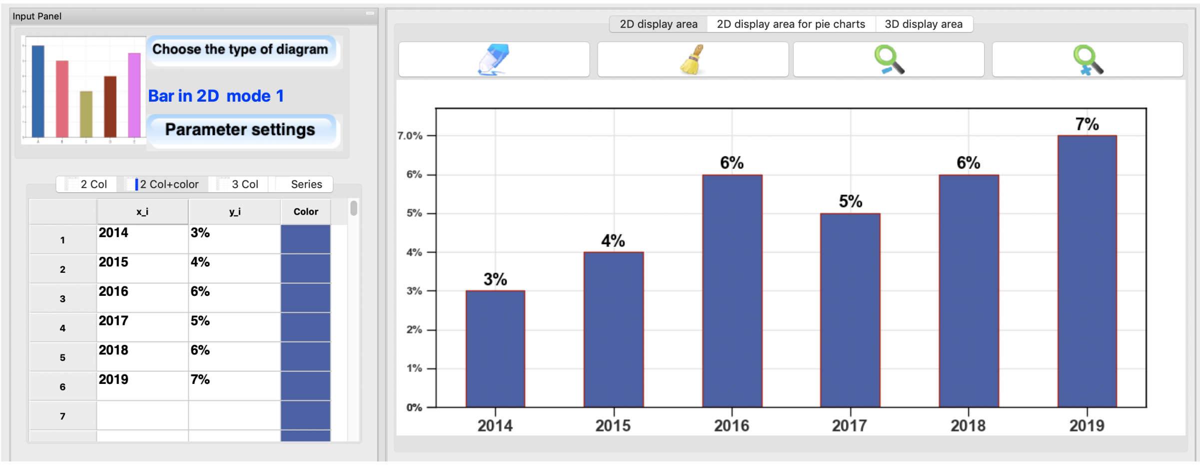
4.7 Diagrams

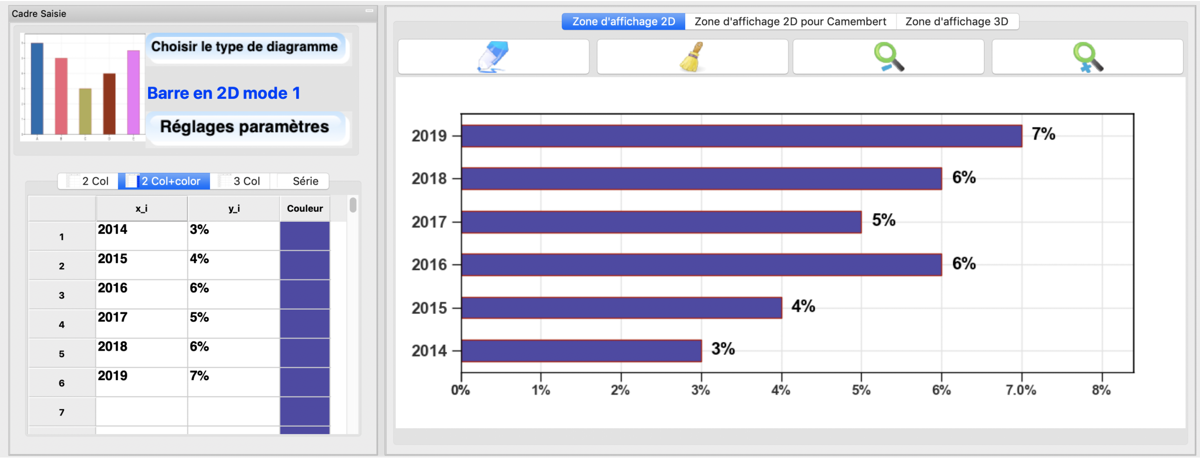
To draw a diagram, first choose the type of diagram: there are `15` of them.



4.7.1 Bar charts

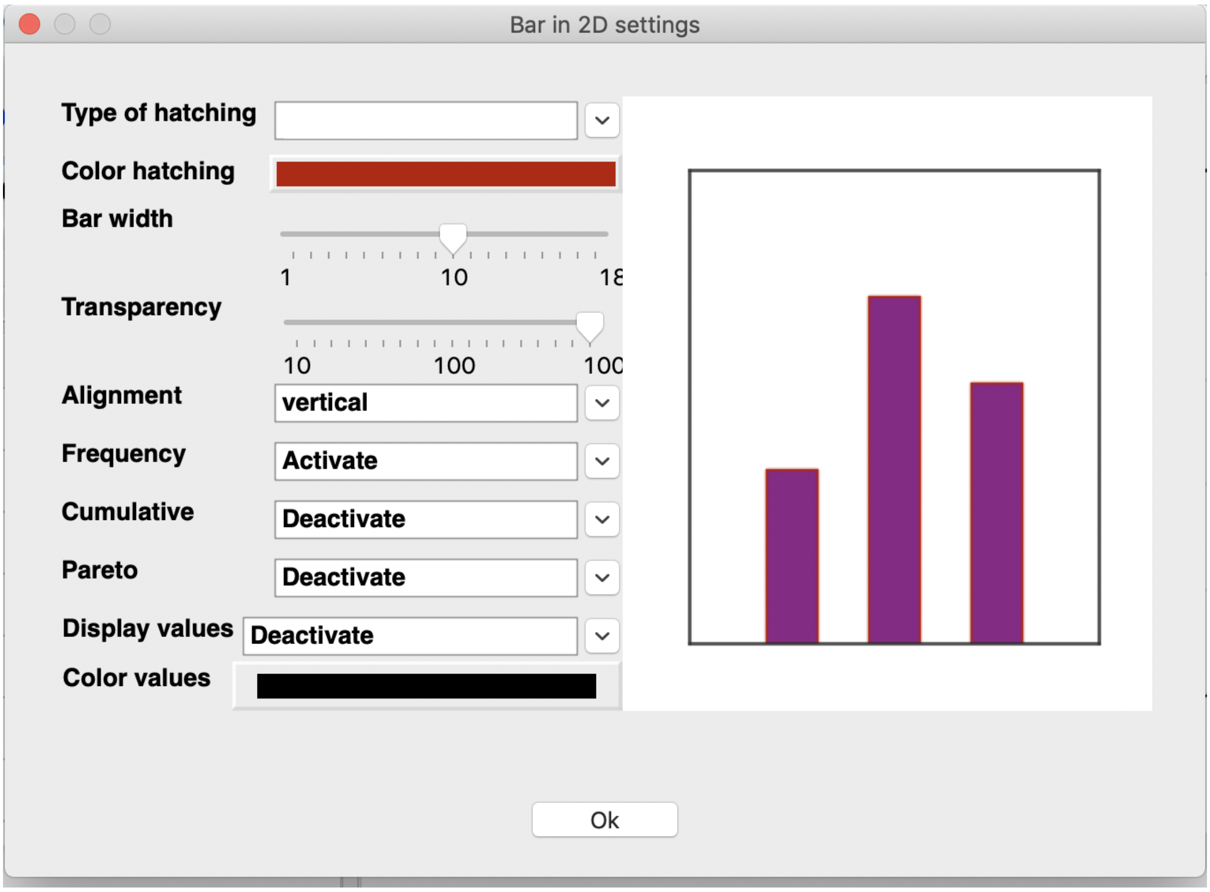
Two-dimensional bar charts

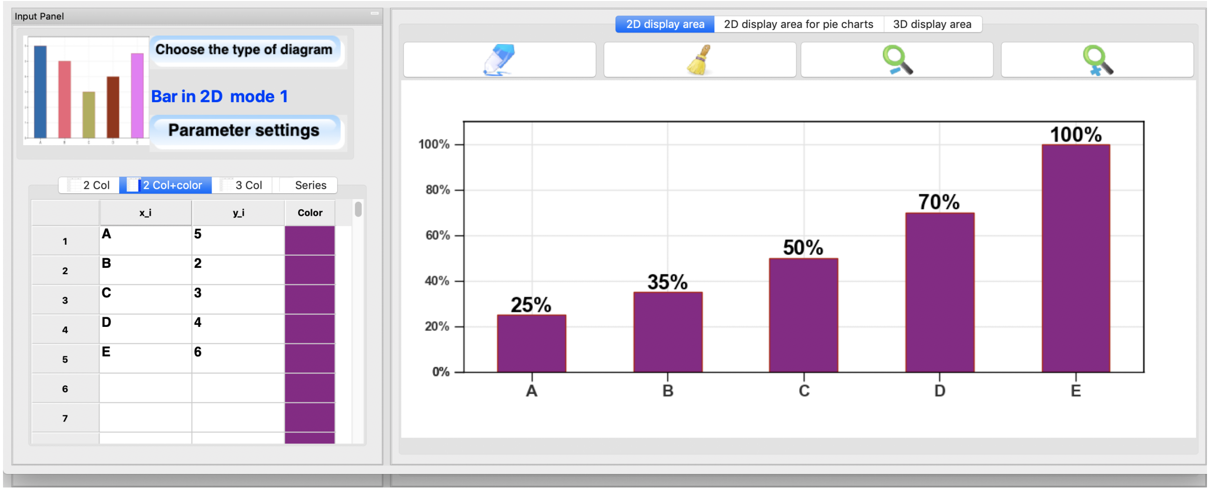




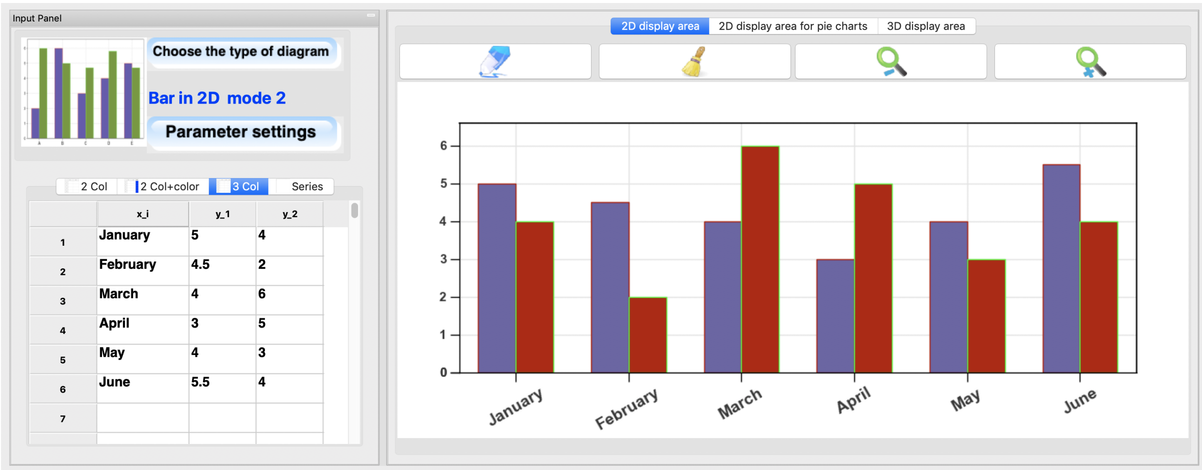
Note: In addition to the bar graph of frequencies, you can also draw the bar graph of the relative frequencies and cumulative frequencies (ascending or decreasing order). You can also draw the Pareto chart.

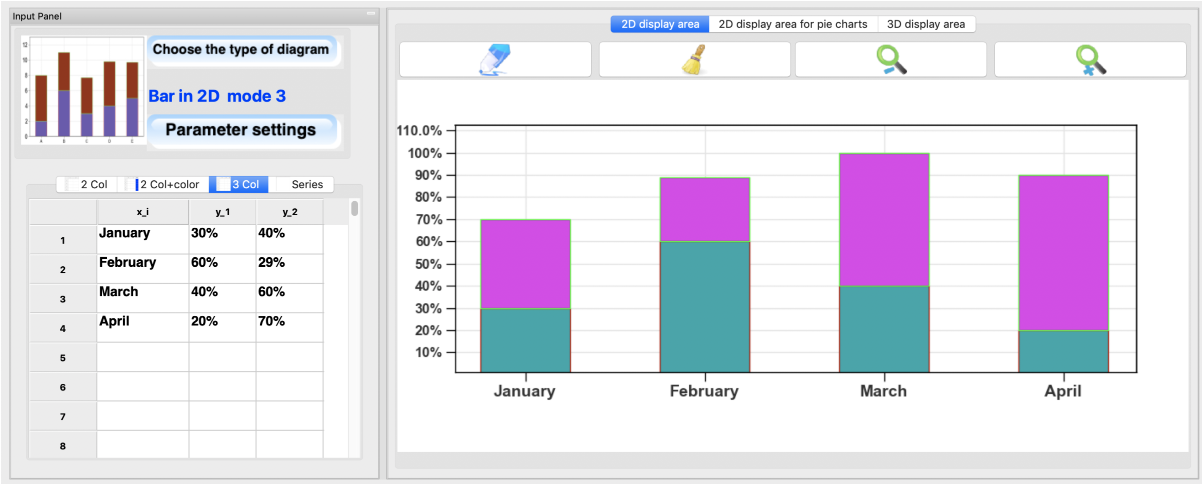
To draw the bar chart of the cumulative frequencies (increasing order), first click on the *Settings Parameters* button, the settings panel will open, for frequencies select *Enable* and for cumulative select *Increasing order*.



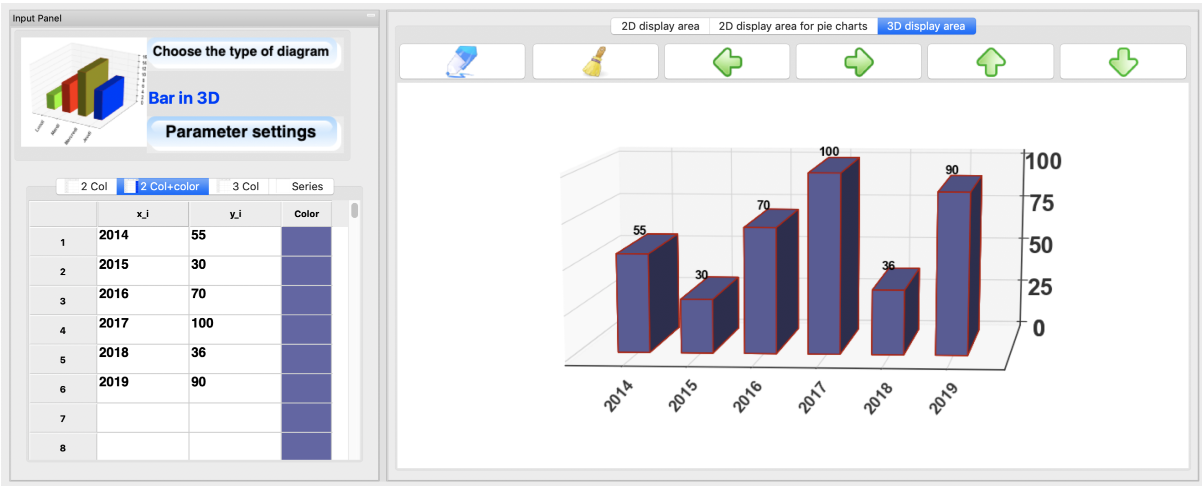


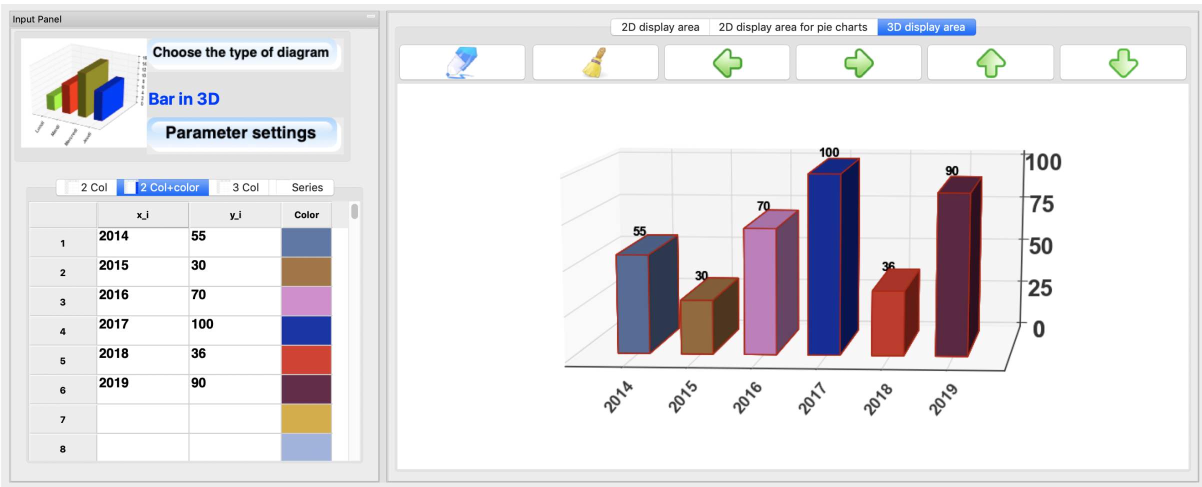
Note: You can also draw two diagrams in parallel either by juxtaposing them or by aligning them.

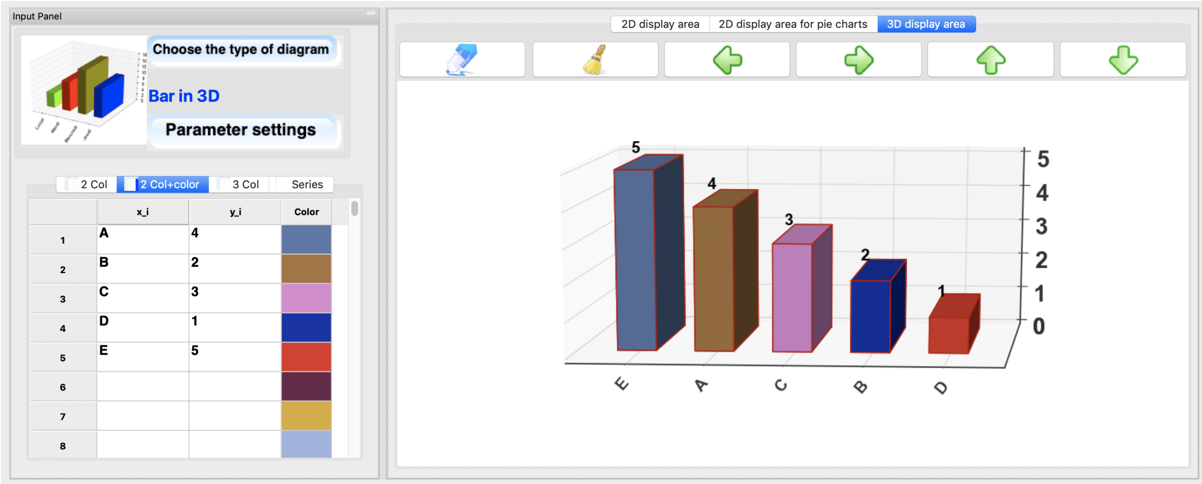




Three-dimensional bar charts





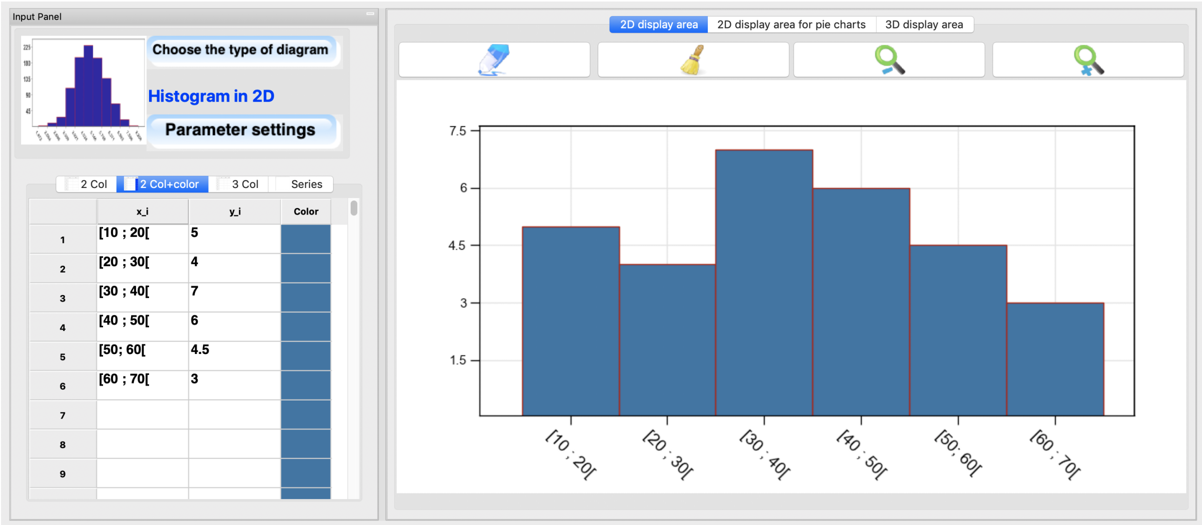

Input Panel
Choose the type of diagram
Bar in 3D
Parameter settings
2 Col
2 Col+color
3 Col
Series
2D display area
2D display area for pie charts
3D display area

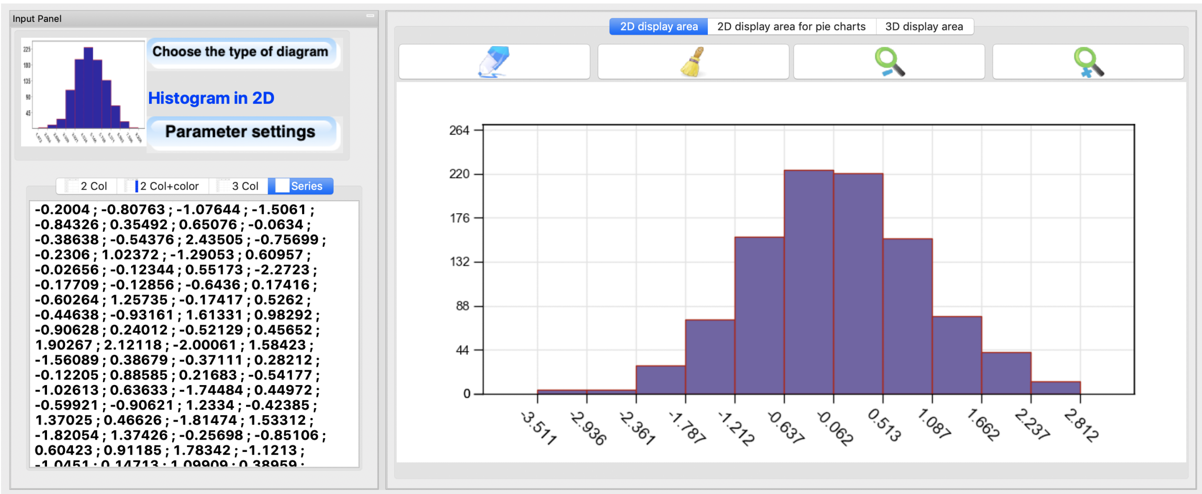

Input Panel
Choose the type of diagram
Bar in 3D mode 2
Parameter settings
2 Col
2 Col+color
3 Col
Series
2D display area
2D display area for pie charts
3D display area


Input Panel
Choose the type of diagram
Bar in 3D mode 3
Parameter settings
2 Col
2 Col+color
3 Col
Series
2D display area
2D display area for pie charts
3D display area

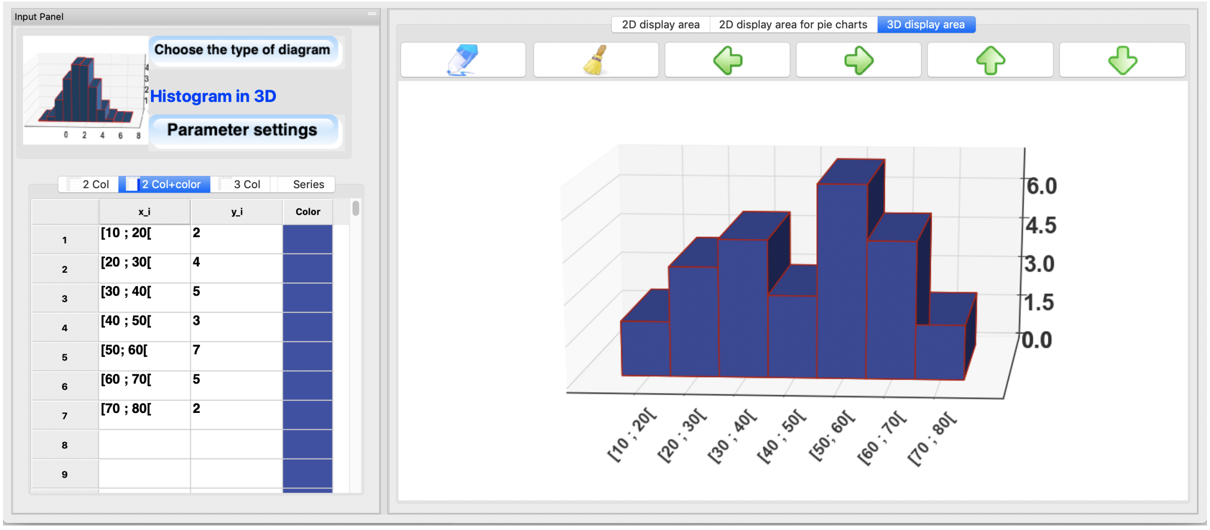
4.7.2 Histograms

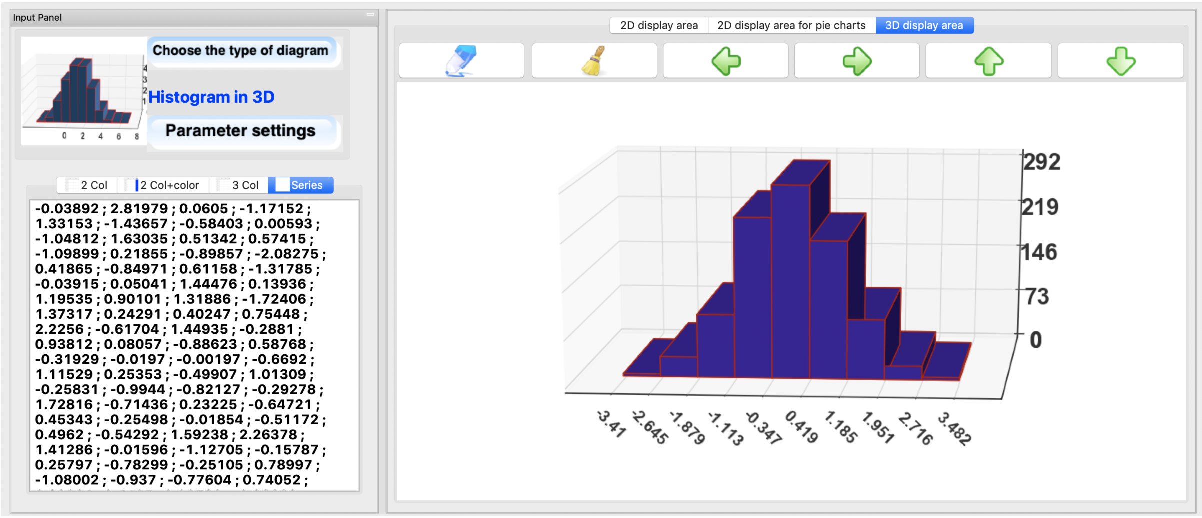
Two-dimensional histogram



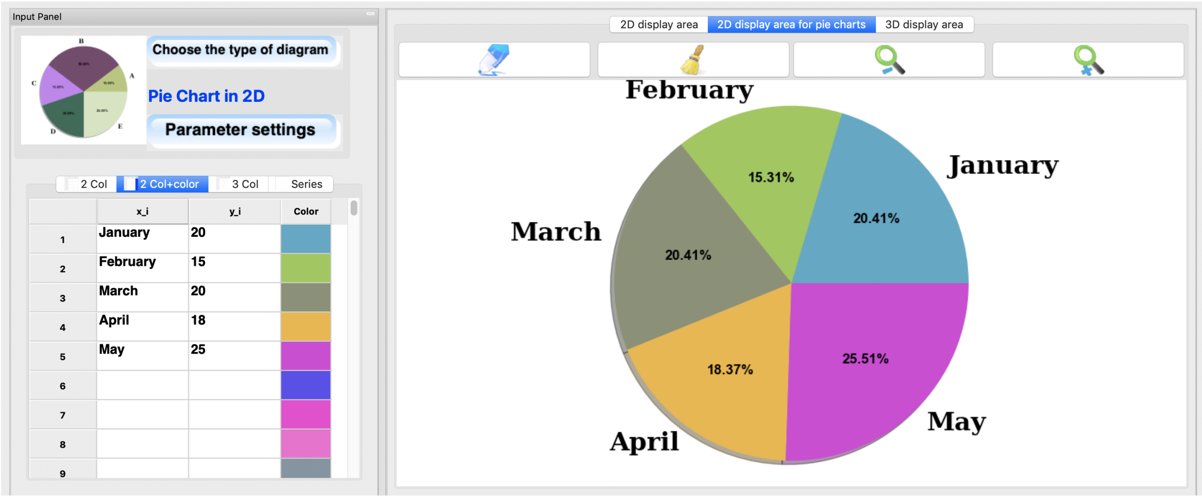


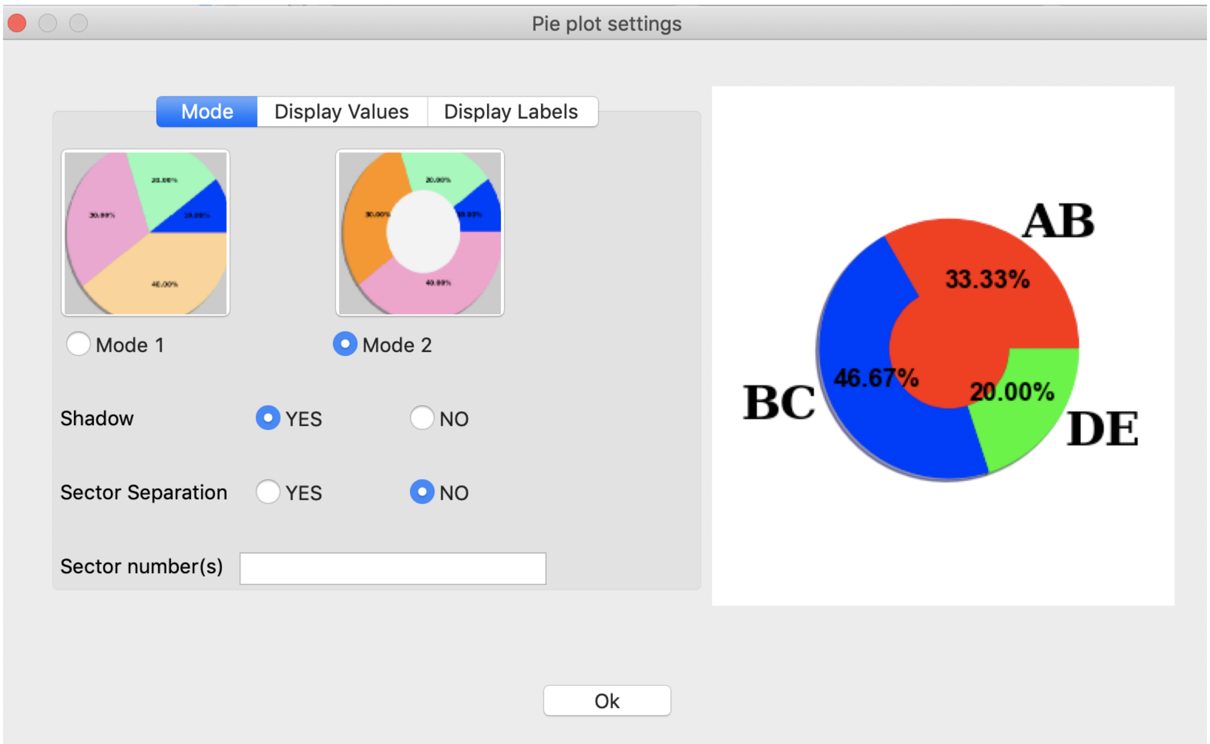
Three-dimensional histogram

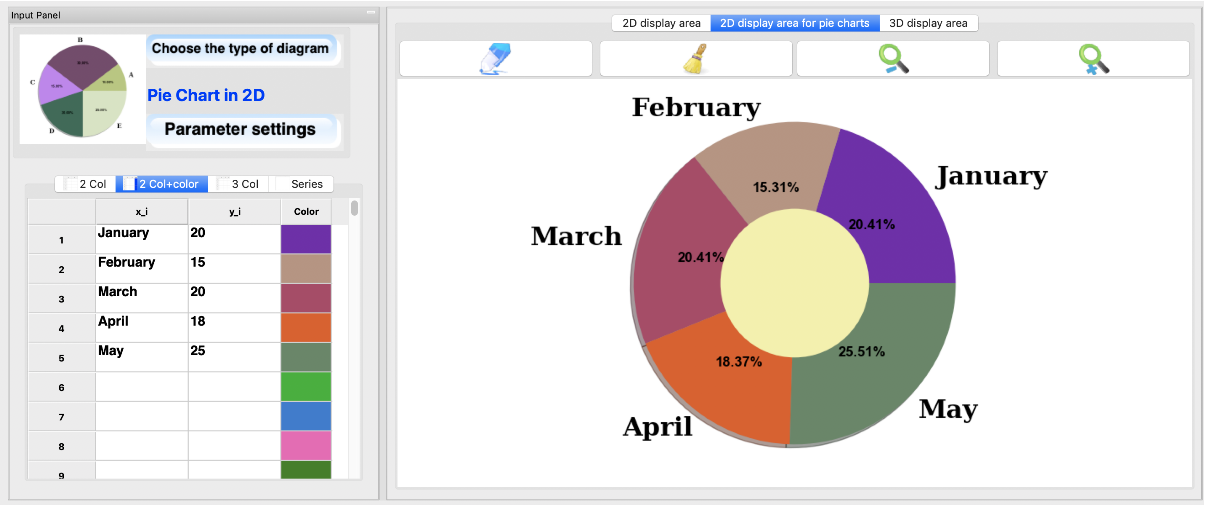


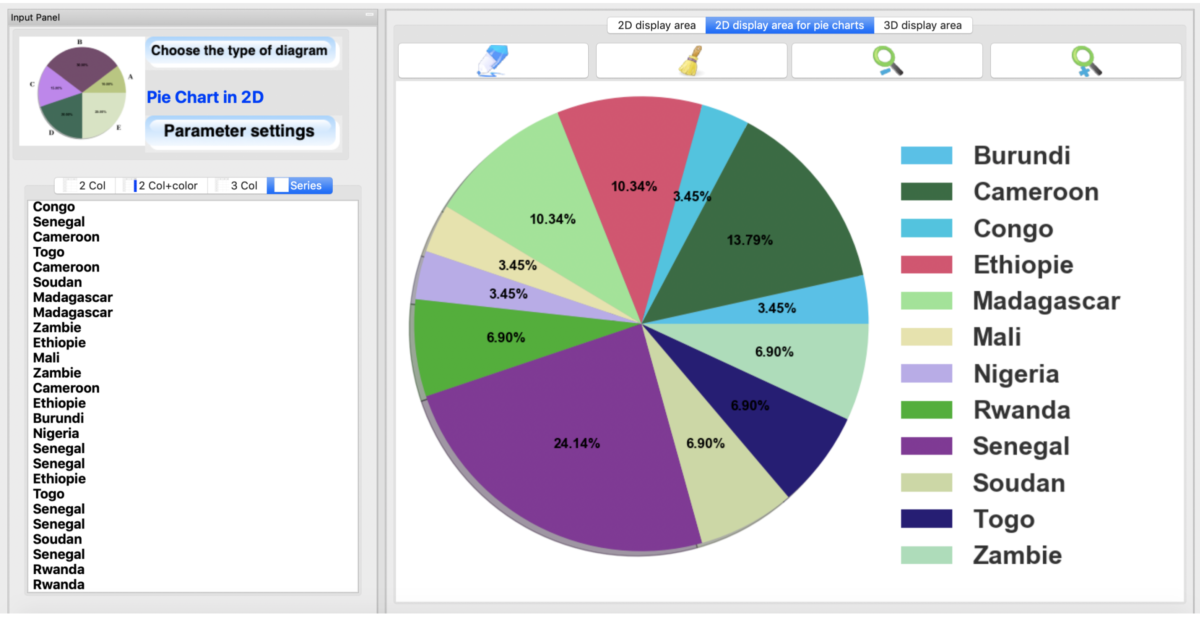


4.7.3 Pie charts







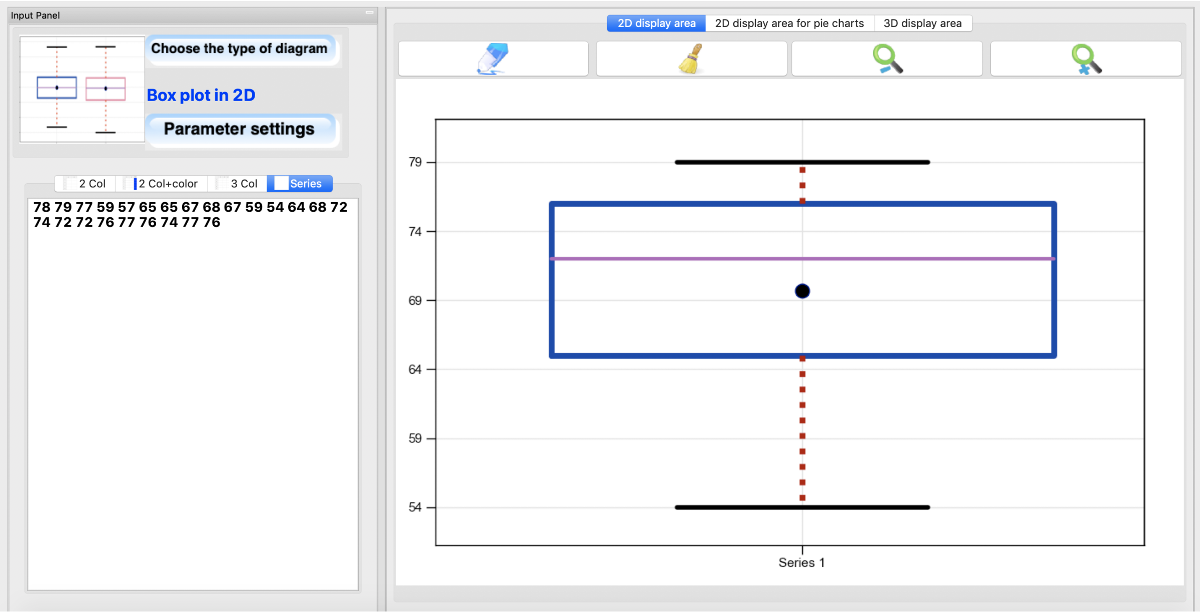


4.7.4 Box diagram

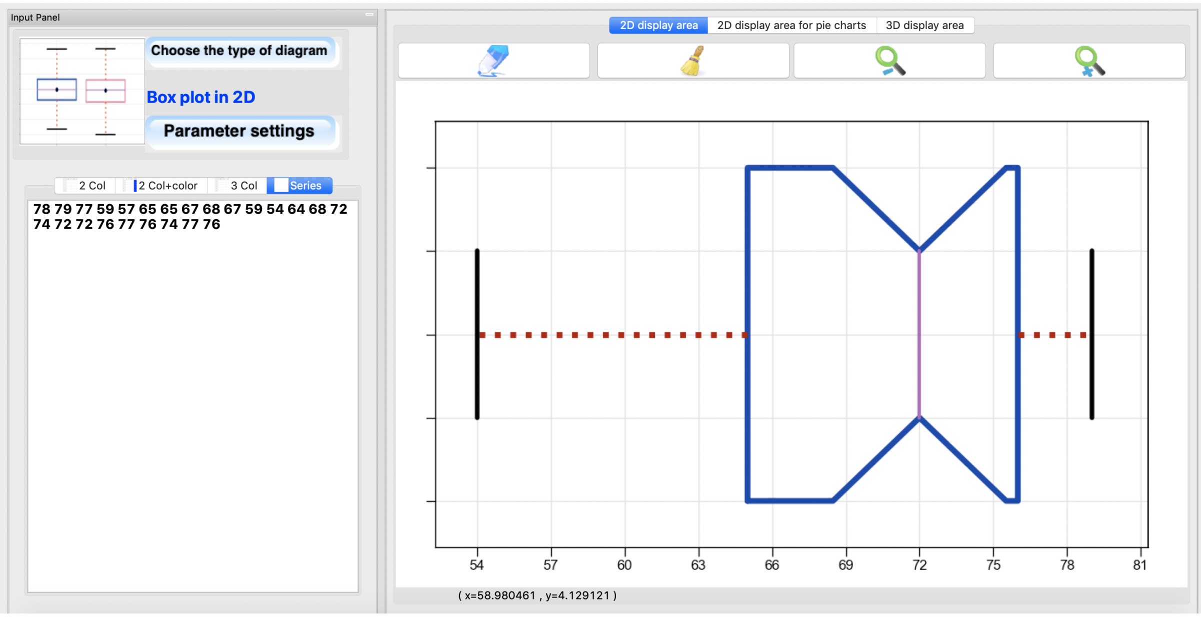
The grades of 24 students in a class were recorded on a 100-point exam.

78 79 77 59 57 65 65 67 68 67 59 54 64 68 72 74 72 72 76 77 76 74 77 76

Let’s draw the whisker box for this statistical series.



We can also change the alignment and properties of the box.



4.7.5 Scatter plot

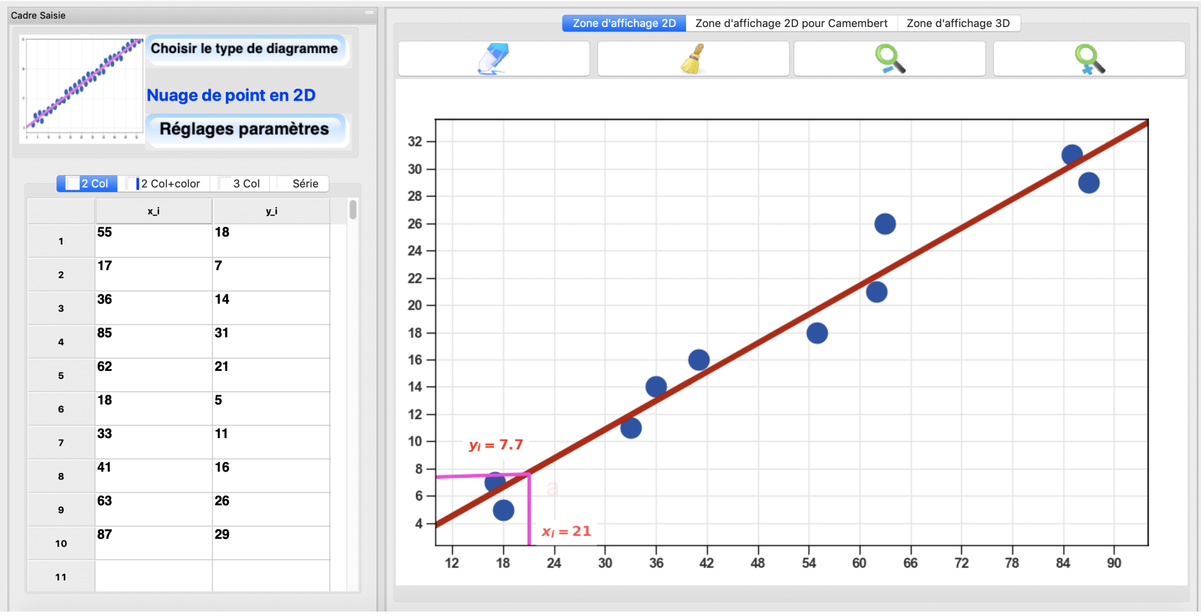
We give the following pairs of observations

x_i	55	17	36	85	62	18	33	41	63	87
y_i	18	7	14	31	21	5	11	16	26	29

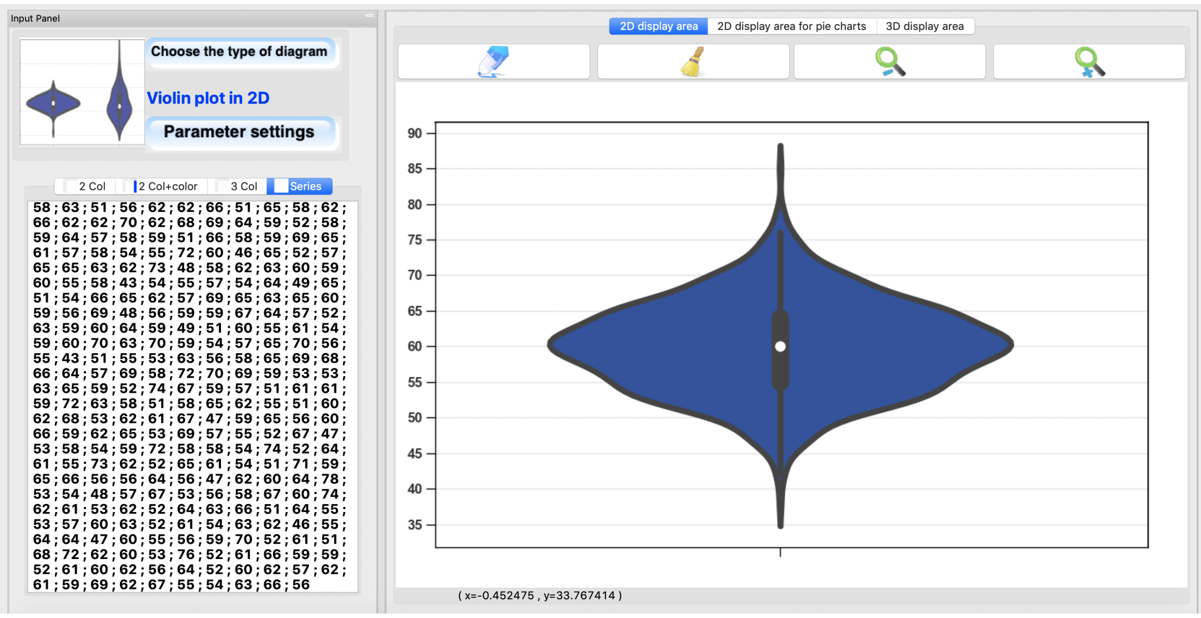
a. Draw the scatter plot of the pairs (x_i, y_i).

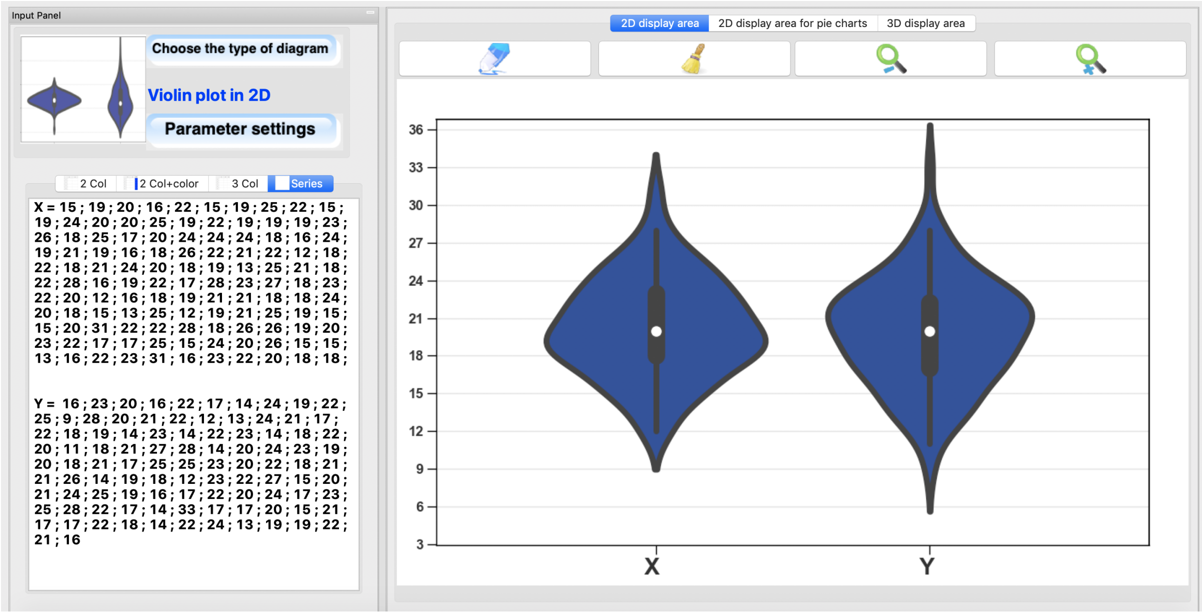
b. Determine the regression line for these observations.

c. What is a plausible estimate of y at $x_i = 21$.

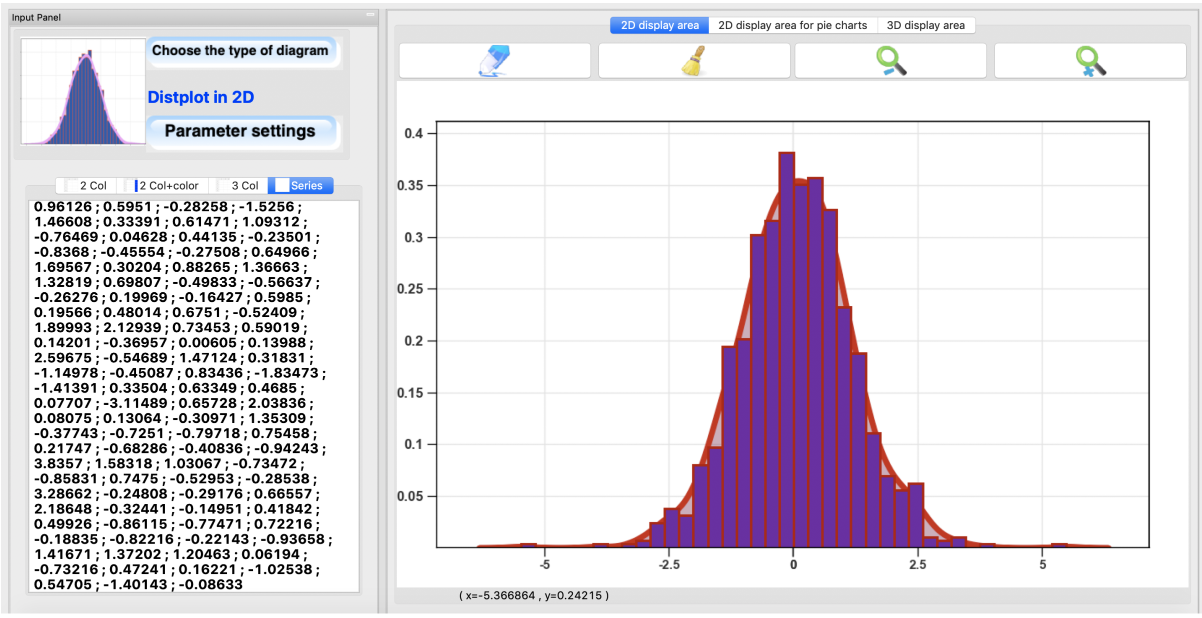


4.7.6 Violin plot



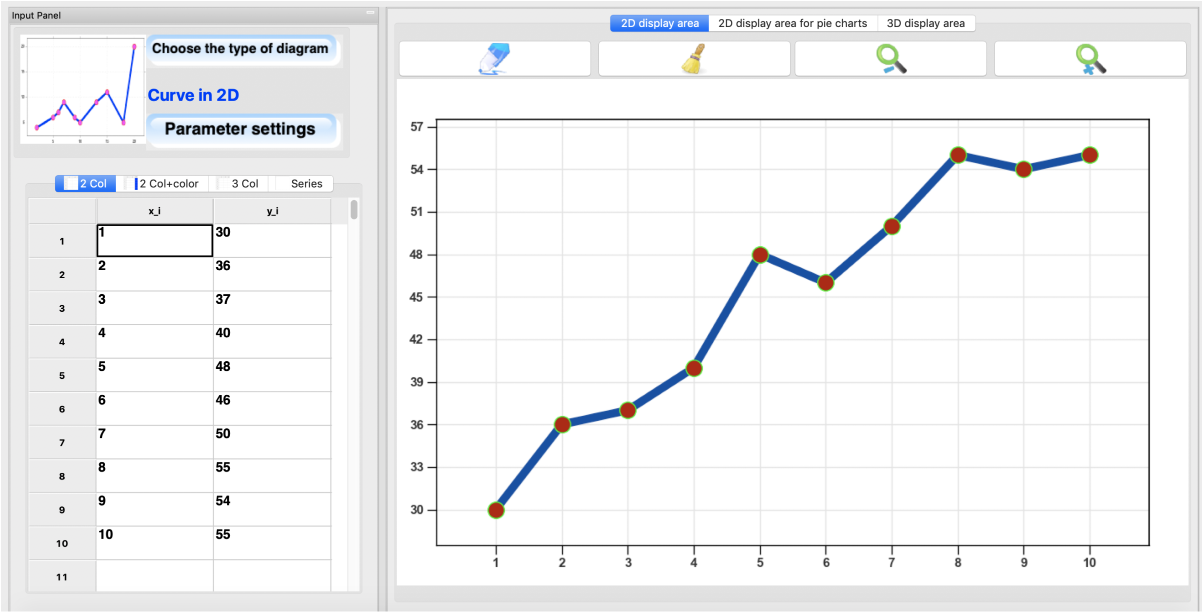


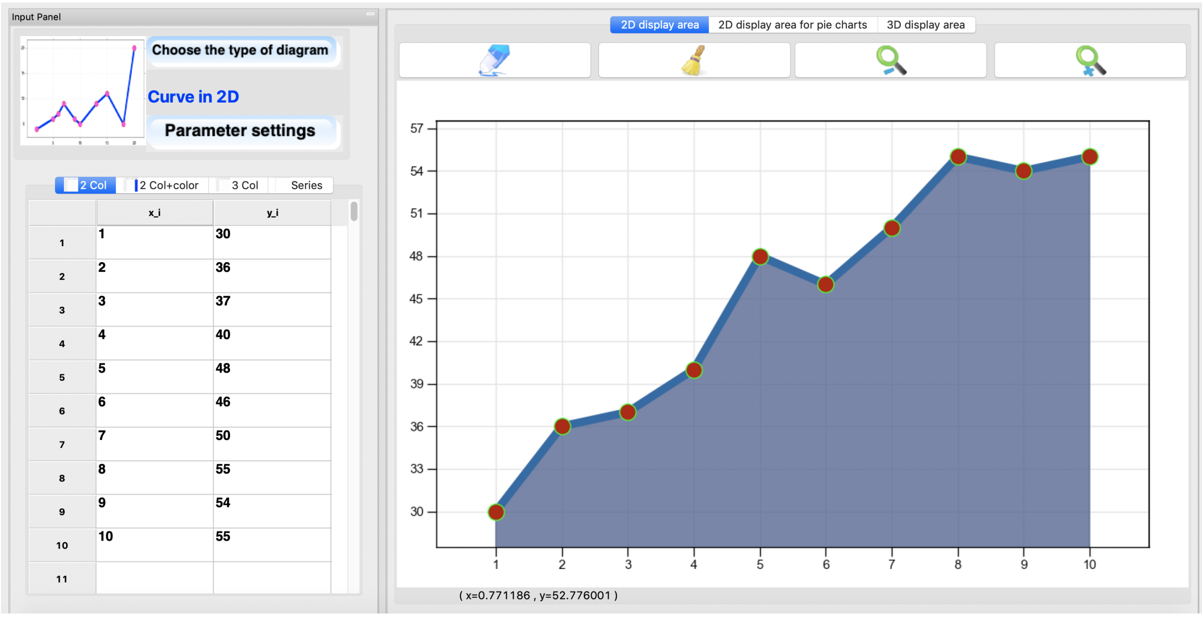
4.7.7 Distplot



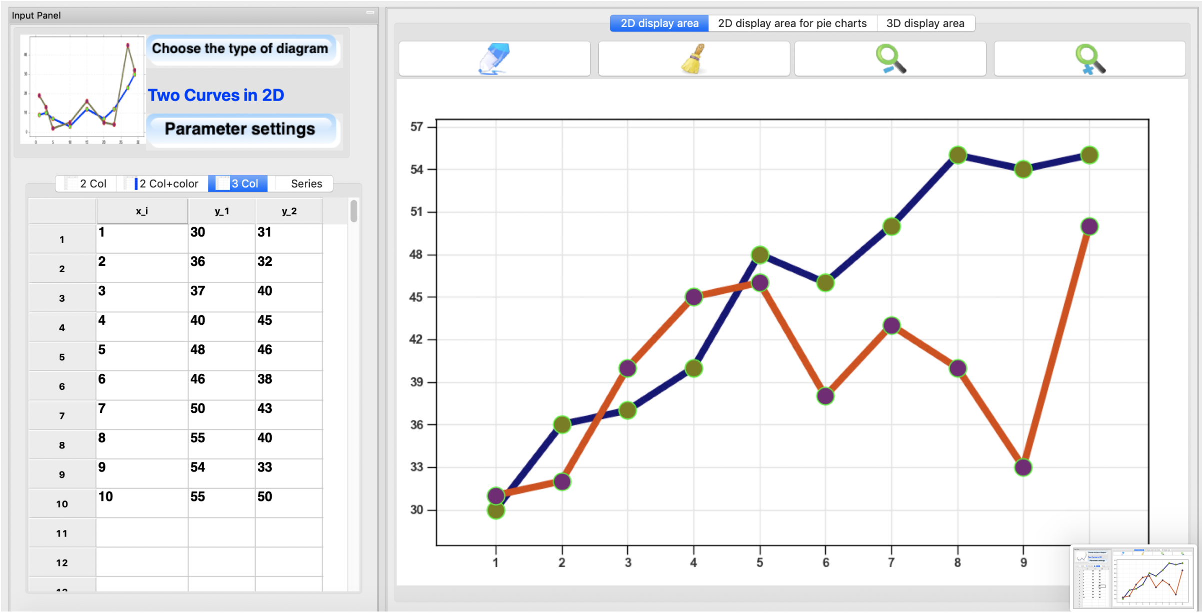
4.7.8 Curves

A curve



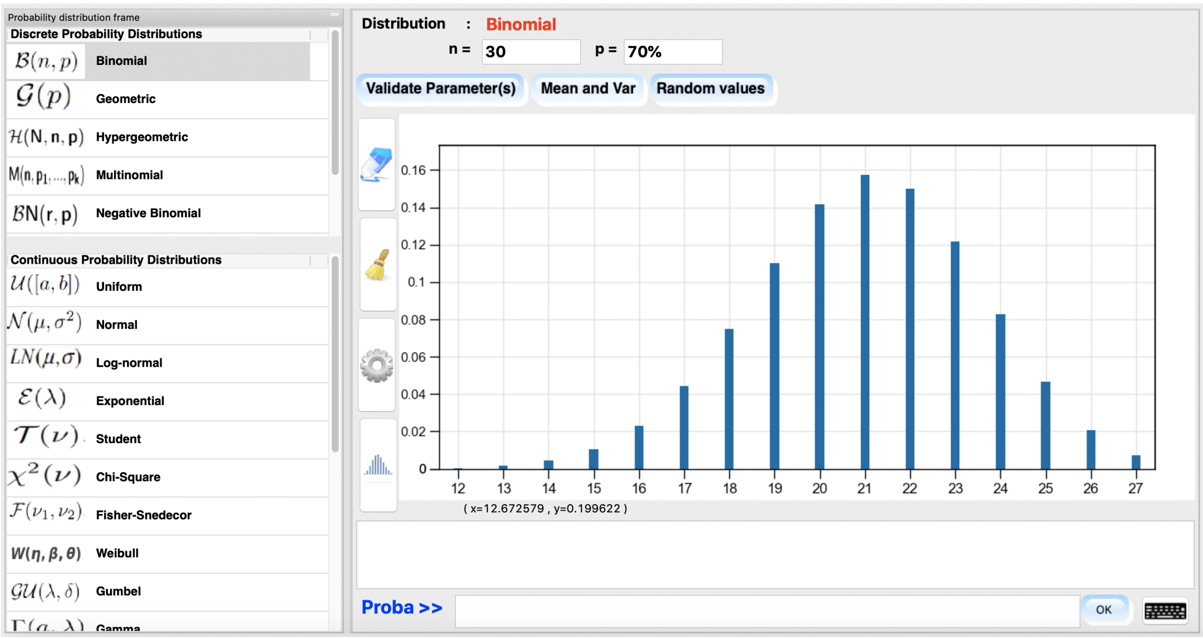


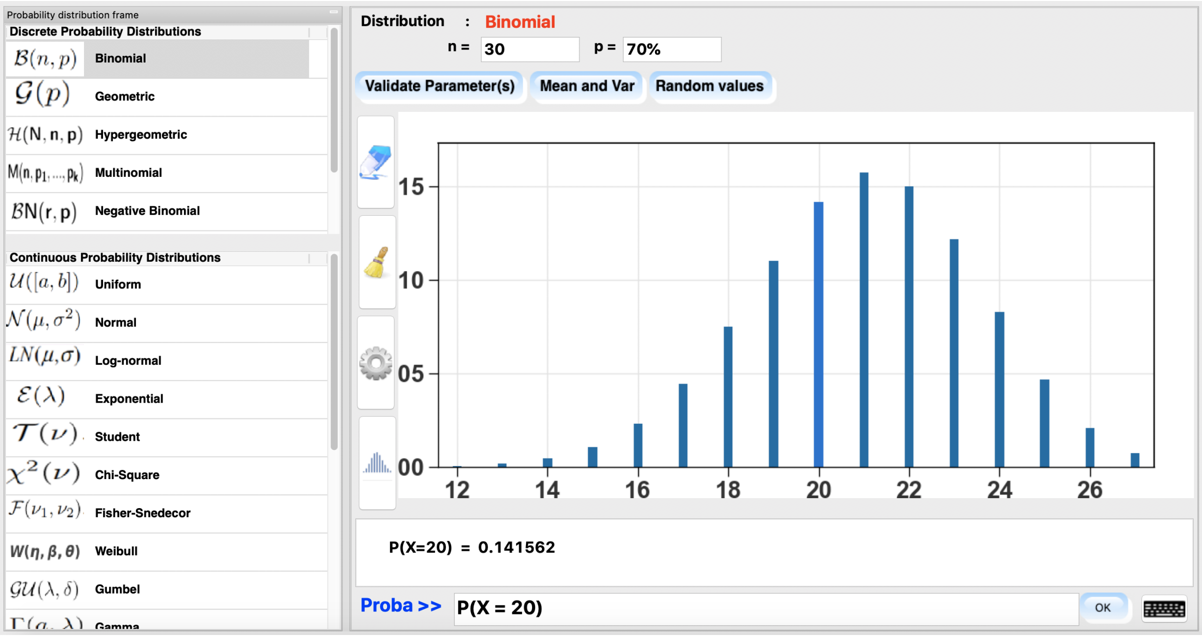
Two curves simultaneously

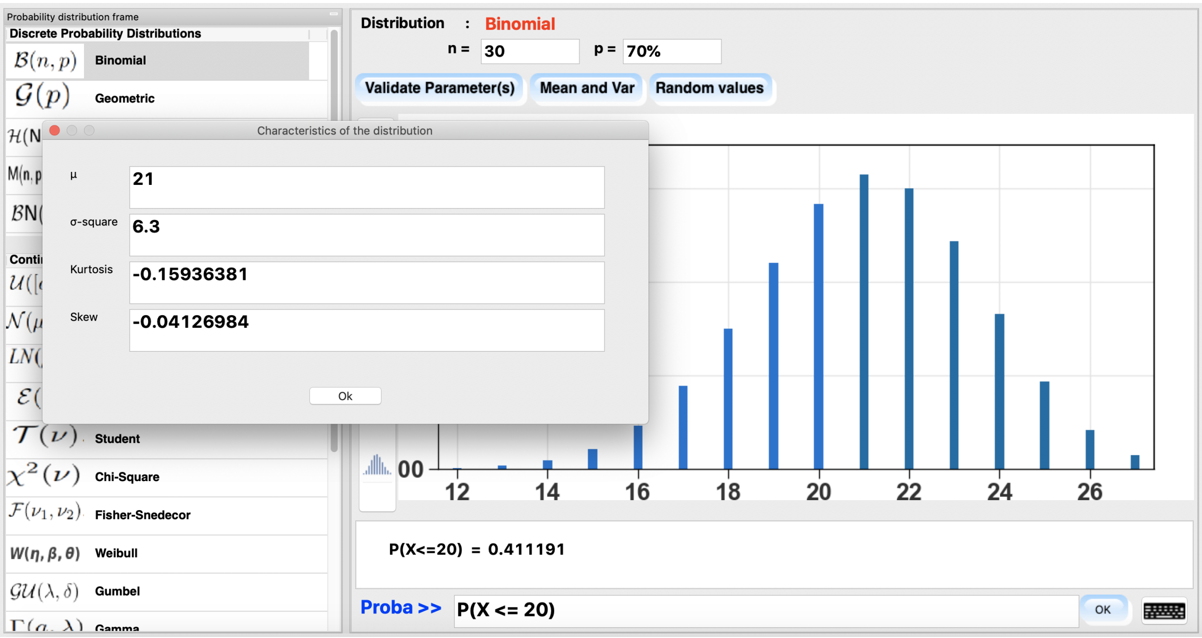


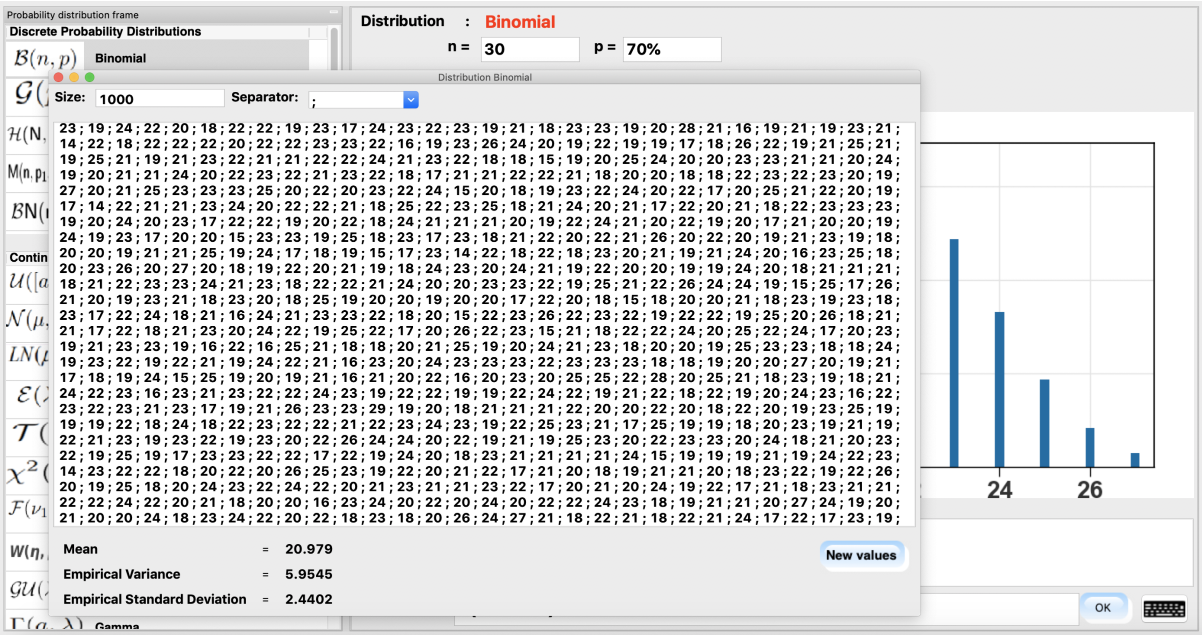
4.8 Probability

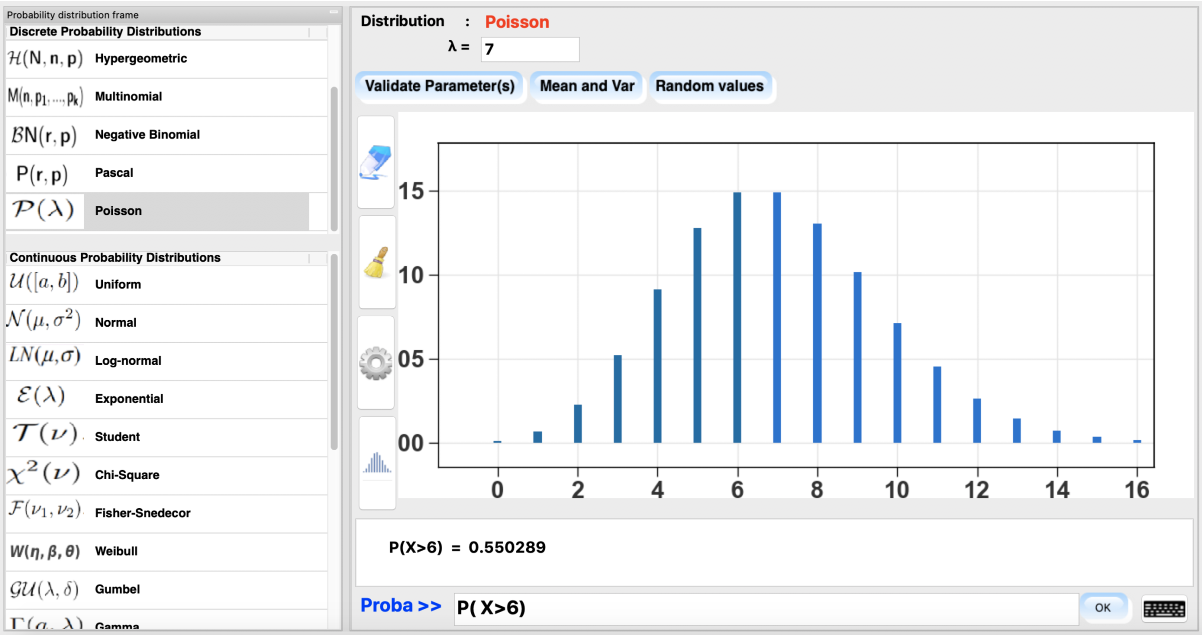
4.8.1 Discrete distributions




Probability distribution frame
Discrete Probability Distributions
Binomial
Geometric
Hypergeometric
Multinomial
Negative Binomial
Continuous Probability Distributions
Uniform
Normal
Log-normal
Exponential
Student
Chi-Square
Fisher-Snedecor
Weibull
Gumbel
Gamma
Distribution : Binomial
n = 30
p = 70%
Validate Parameter(s)
Mean and Var
Random values
P(X=20) = 0.141562
Proba >>
P(X = 20)
OK

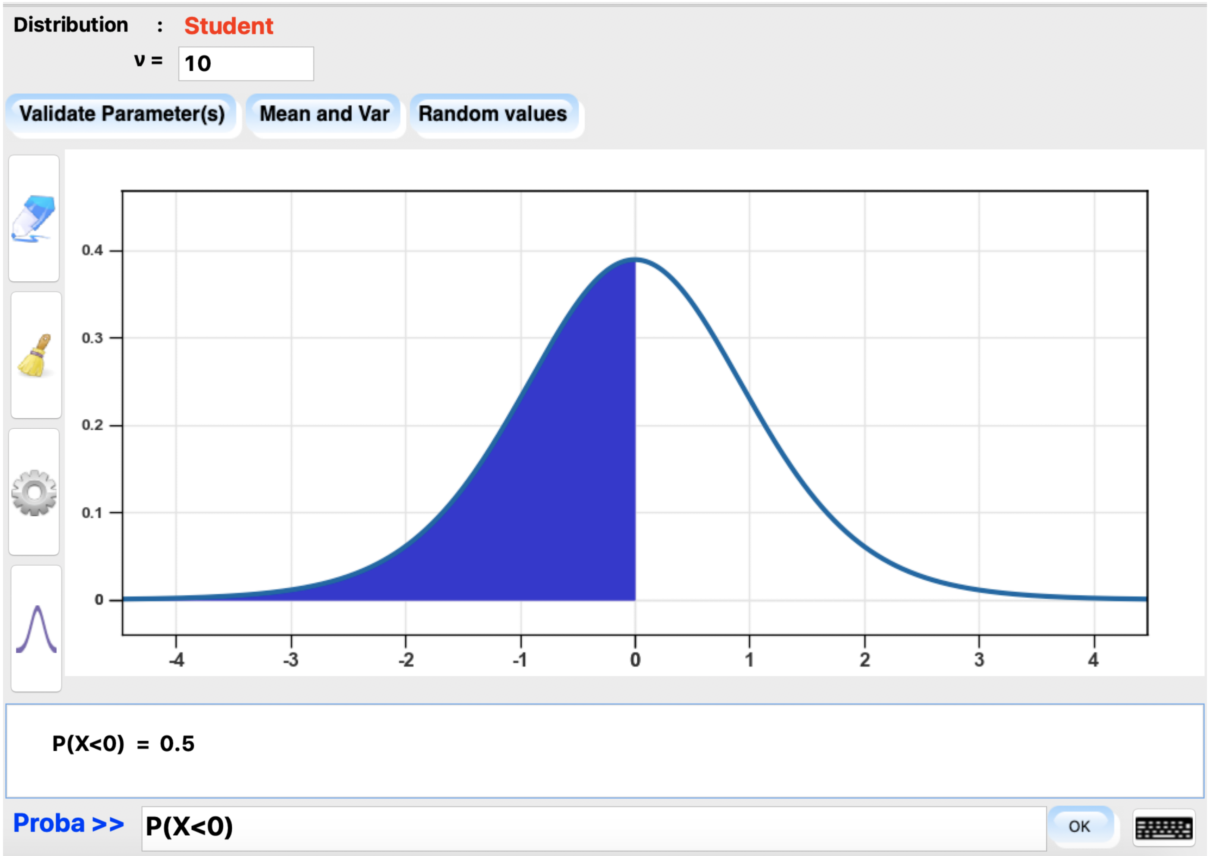

Probability distribution frame
Discrete Probability Distributions
Binomial
Geometric
Characteristics of the distribution
μ
21
σ-square
6.3
Kurtosis
-0.15936381
Skew
-0.04126984
Ok
Student
Chi-Square
Fisher-Snedecor
Weibull
Gumbel
Gamma
Distribution : Binomial
n = 30
p = 70%
Validate Parameter(s)
Mean and Var
Random values
P(X<=20) = 0.411191
Proba >>
P(X <= 20)
OK

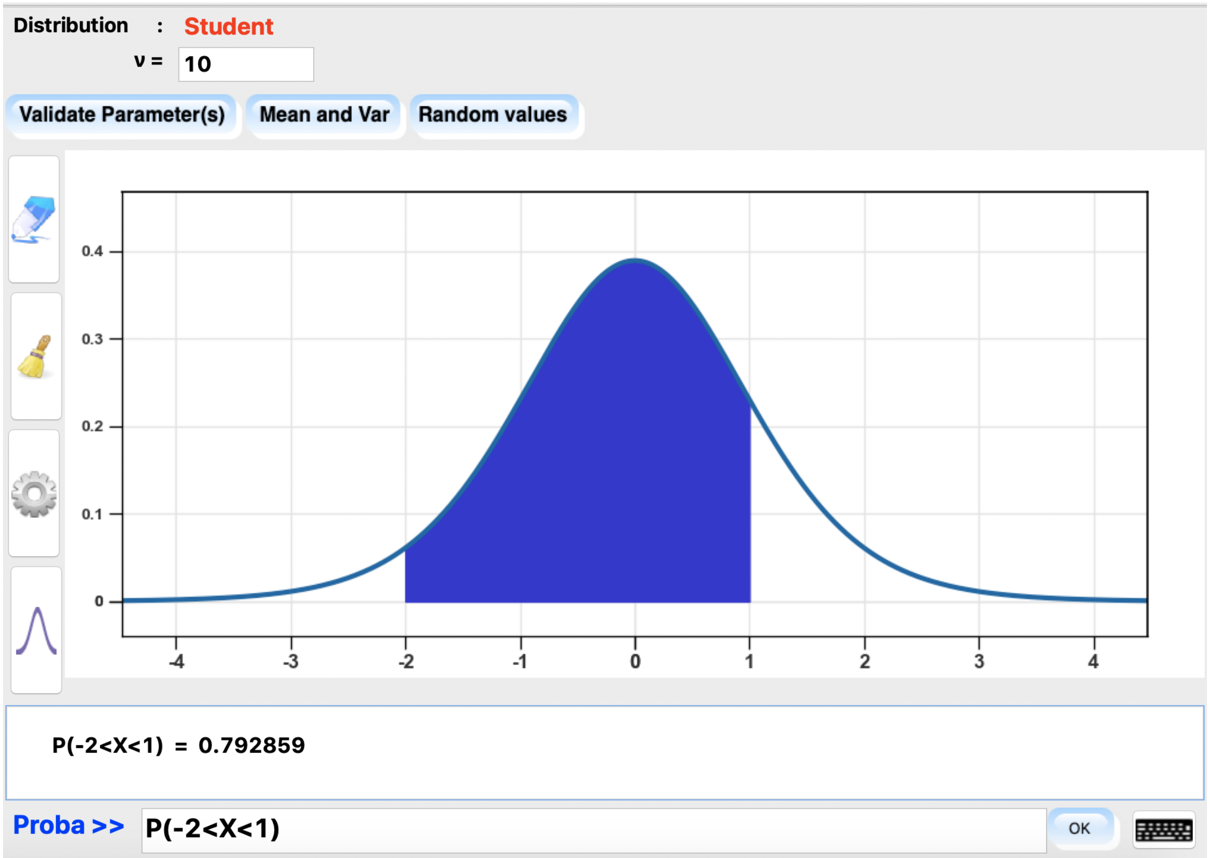


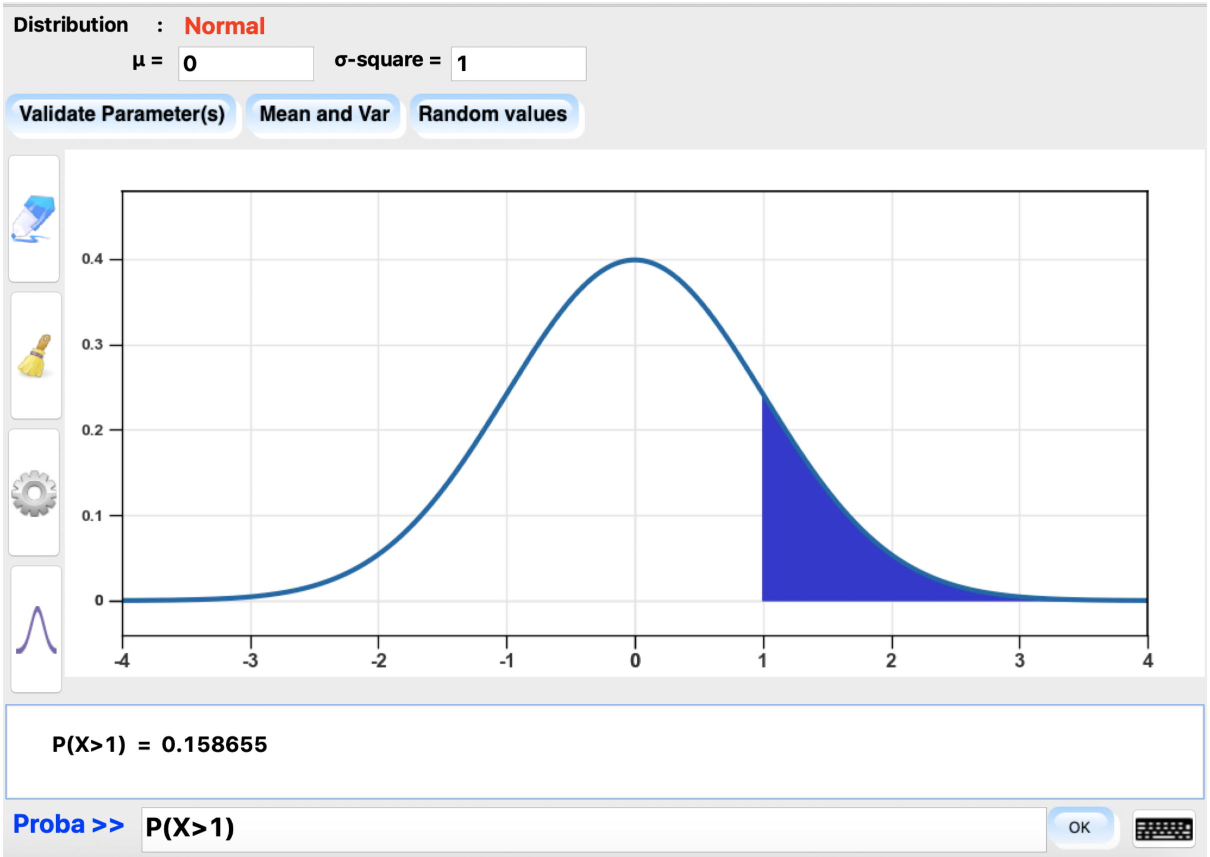


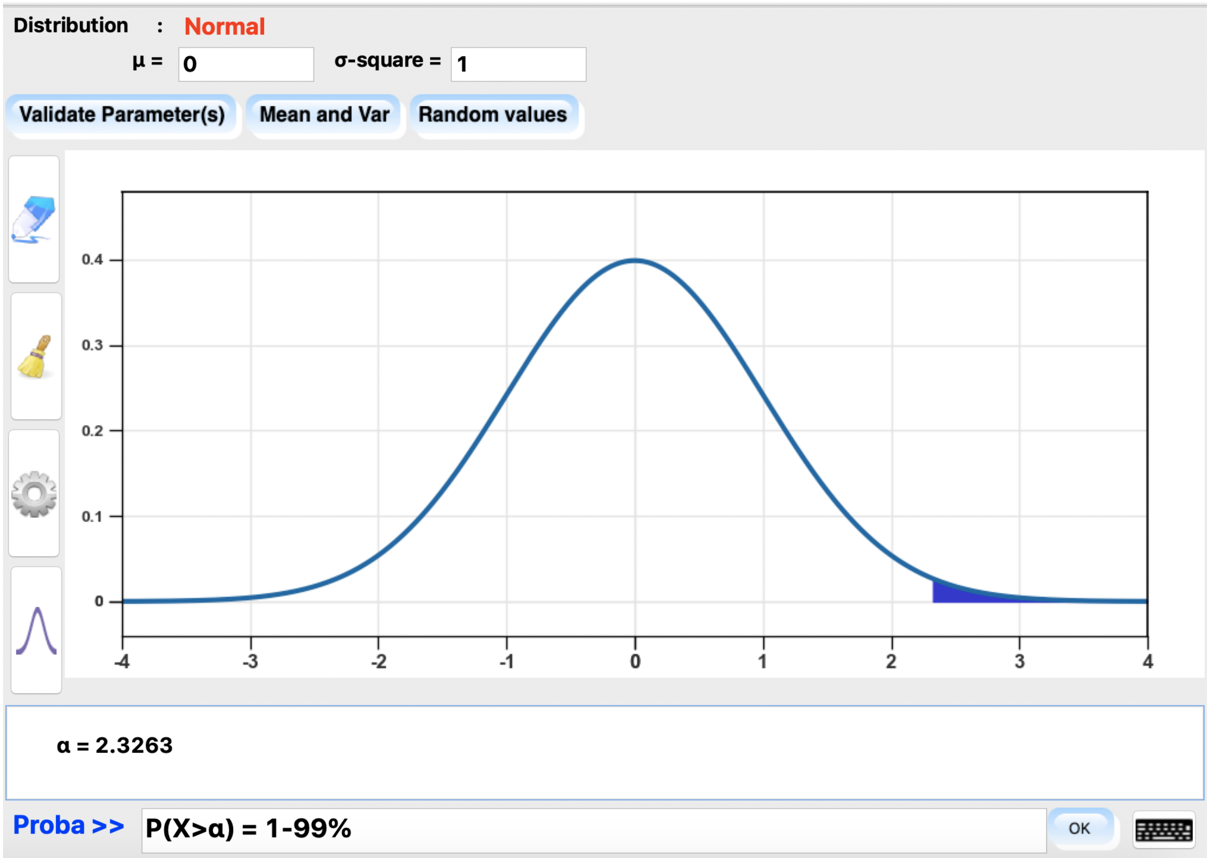
4.8.2 The continuous distributions

Probability calculation

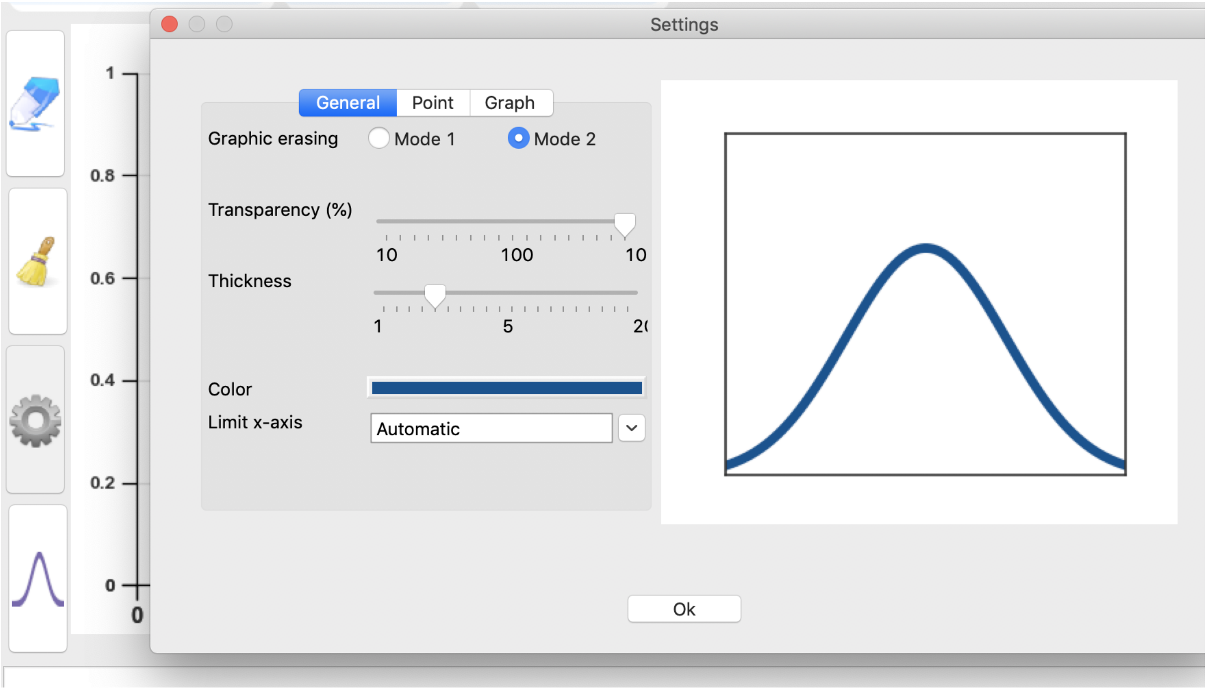


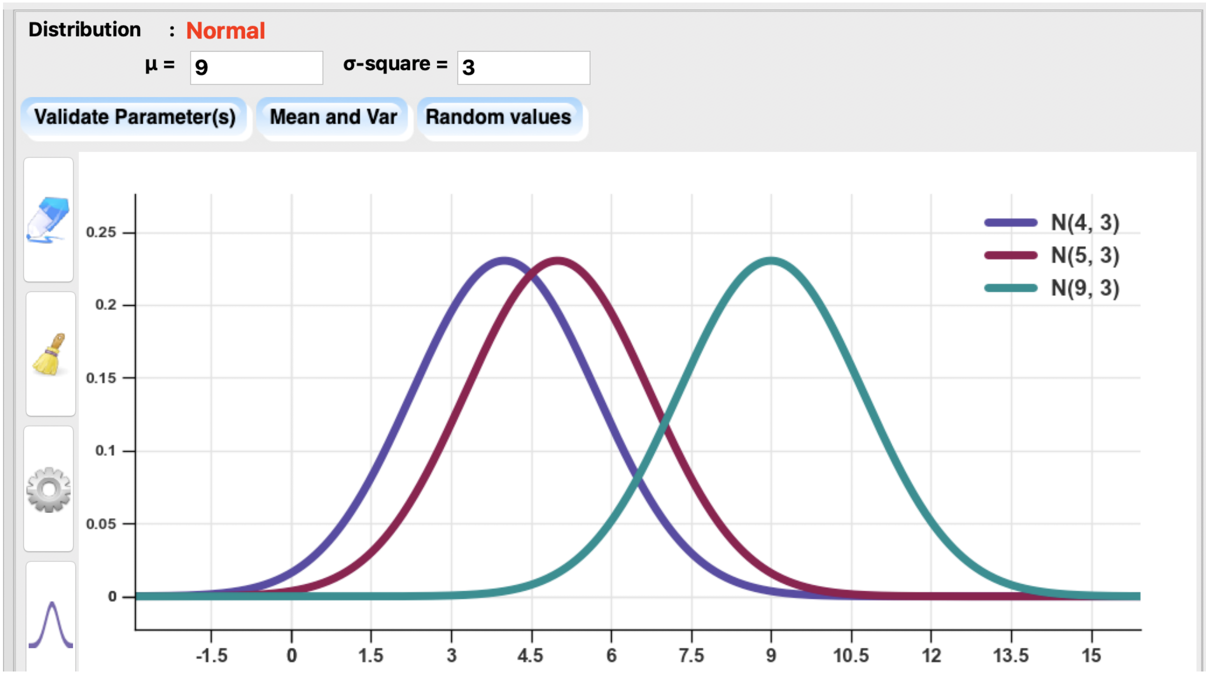


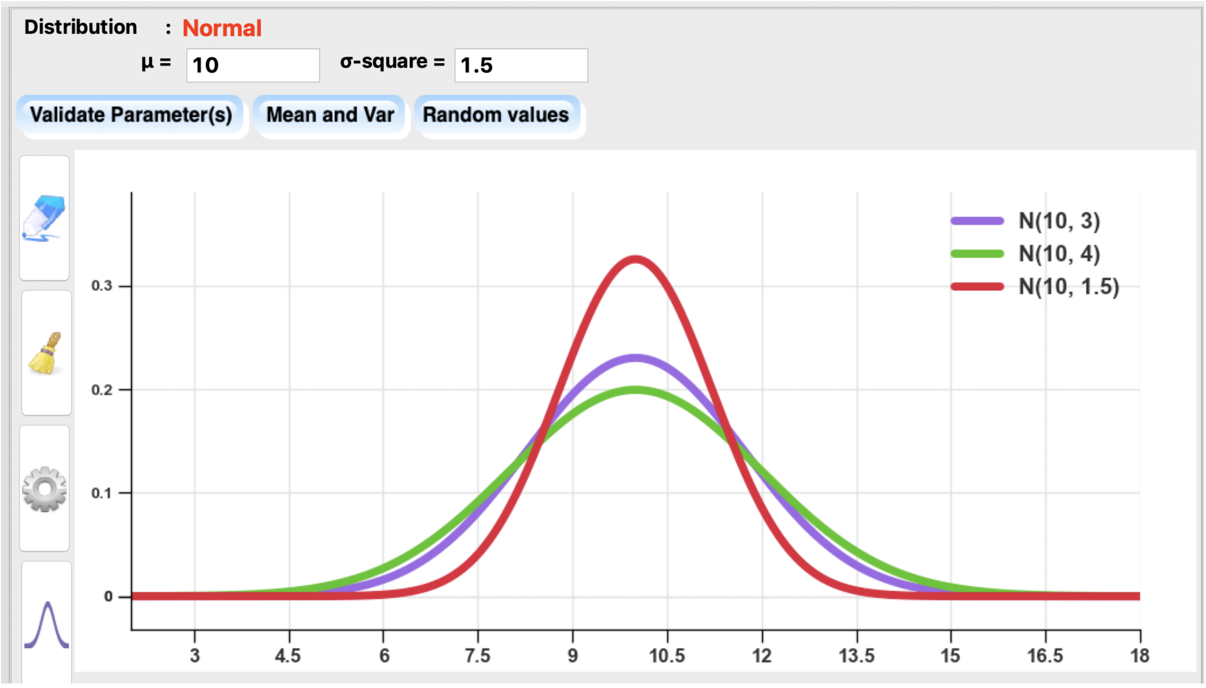

Distribution : Normal
μ = 0
σ-square = 1
Validate Parameter(s)
Mean and Var
Random values
P(X>1) = 0.158655
Proba >>
P(X>1)
OK

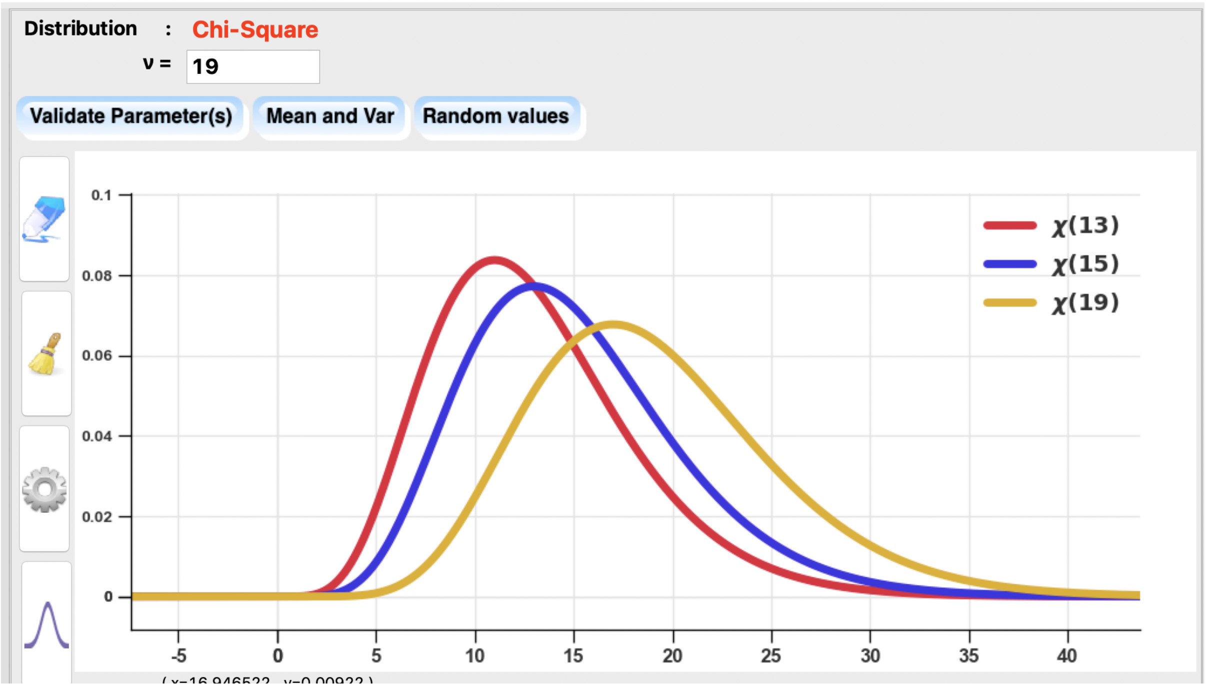

Distribution : Normal
μ = 0
σ-square = 1
Validate Parameter(s)
Mean and Var
Random values
α = 2.3263
Proba >>
P(X>α) = 1-99%
OK

Simulation of probability distributions with different parameters



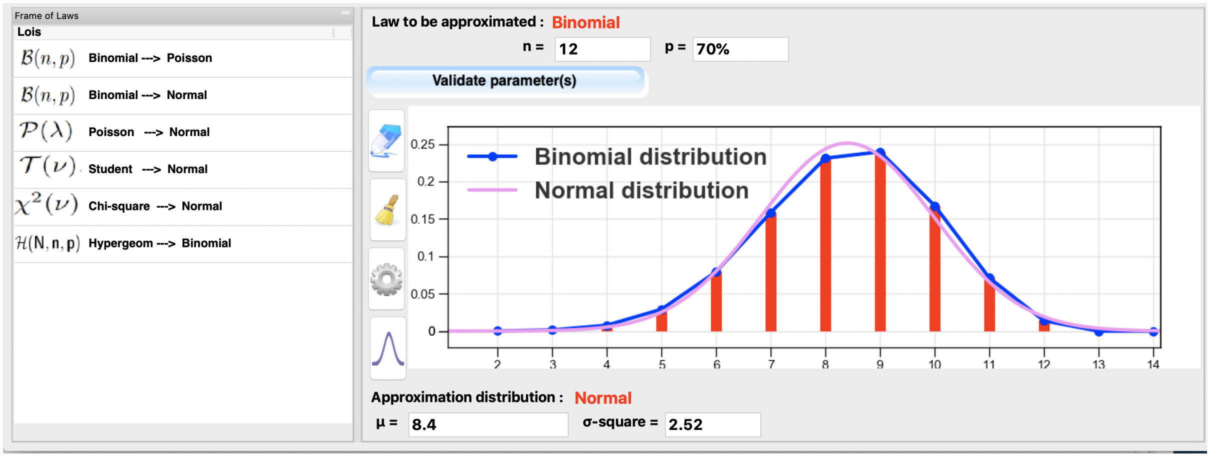


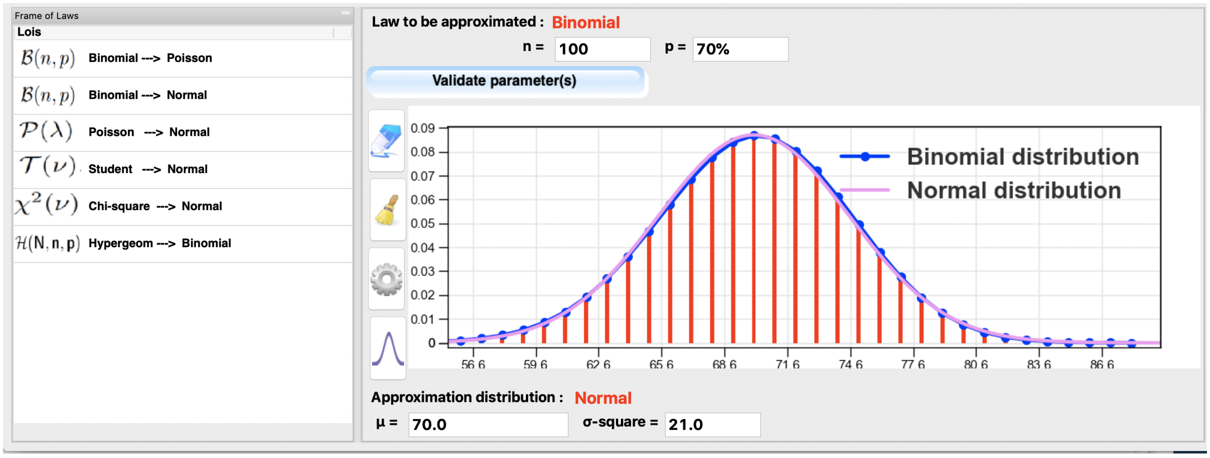

Distribution : Normal
μ = 10
σ-square = 1.5
Validate Parameter(s)
Mean and Var
Random values
N(10, 3)
N(10, 4)
N(10, 1.5)
0.3
0.2
0.1
0
3
4.5
6
7.5
9
10.5
12
13.5
15
16.5
18


Distribution : Chi-Square
ν = 19
Validate Parameter(s)
Mean and Var
Random values
χ(13)
χ(15)
χ(19)
0.1
0.08
0.06
0.04
0.02
0
-5
0
5
10
15
20
25
30
35
40

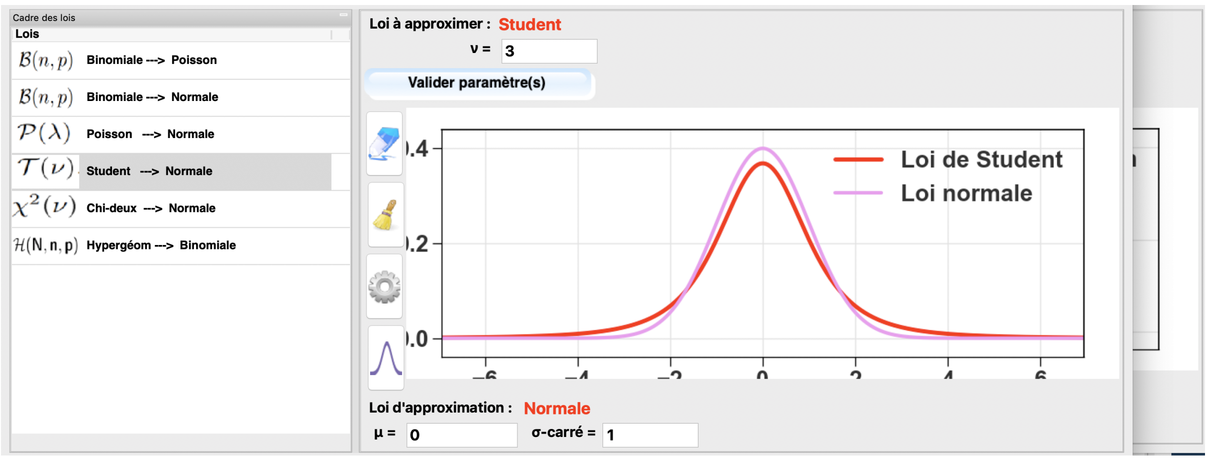
4.9 Approximation of probability distributions

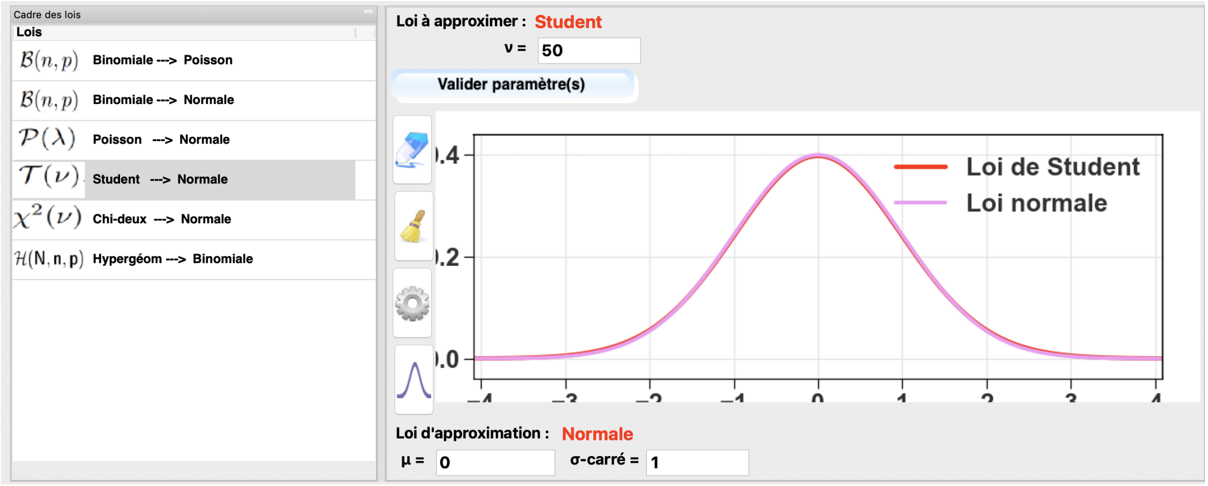
4.9.1 Approximation of the binomial distribution





4.9.2 Approximation of the Student's distribution





4.9.3 Approximation of other probability distributions

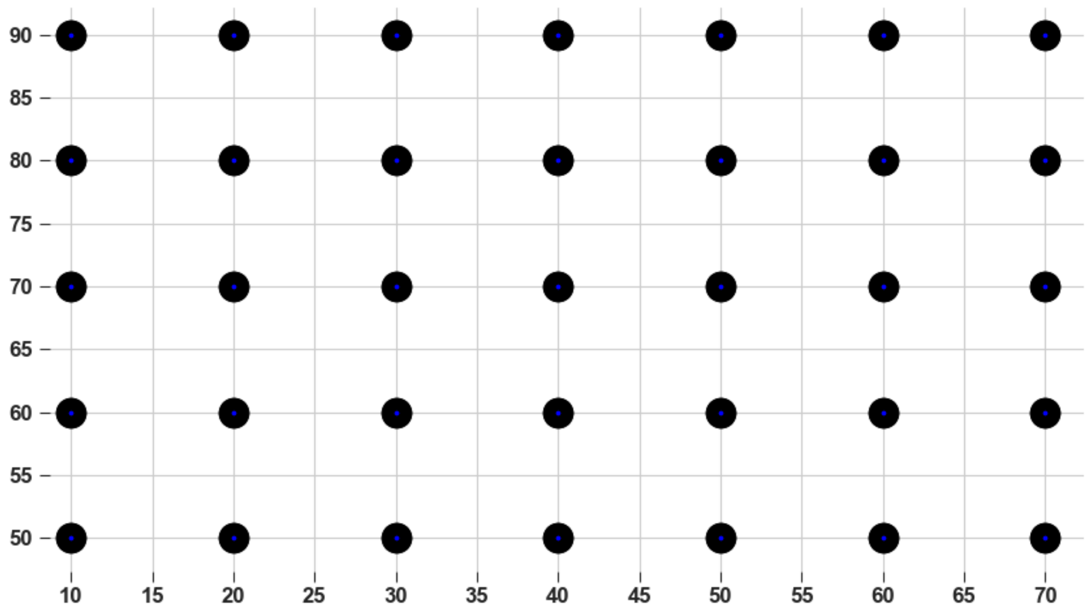
With **SimulaMath**, it is also possible to simulate under which condition(s) one can approximate :

- the *binomial* distribution by a *Poisson* distribution,
- the *Khi-deux* distribution by a *normal* distribution,
- the *Poisson* distribution by a *normal* distribution,
- the *Hypergeometric* distribution by a *binomial* distribution.

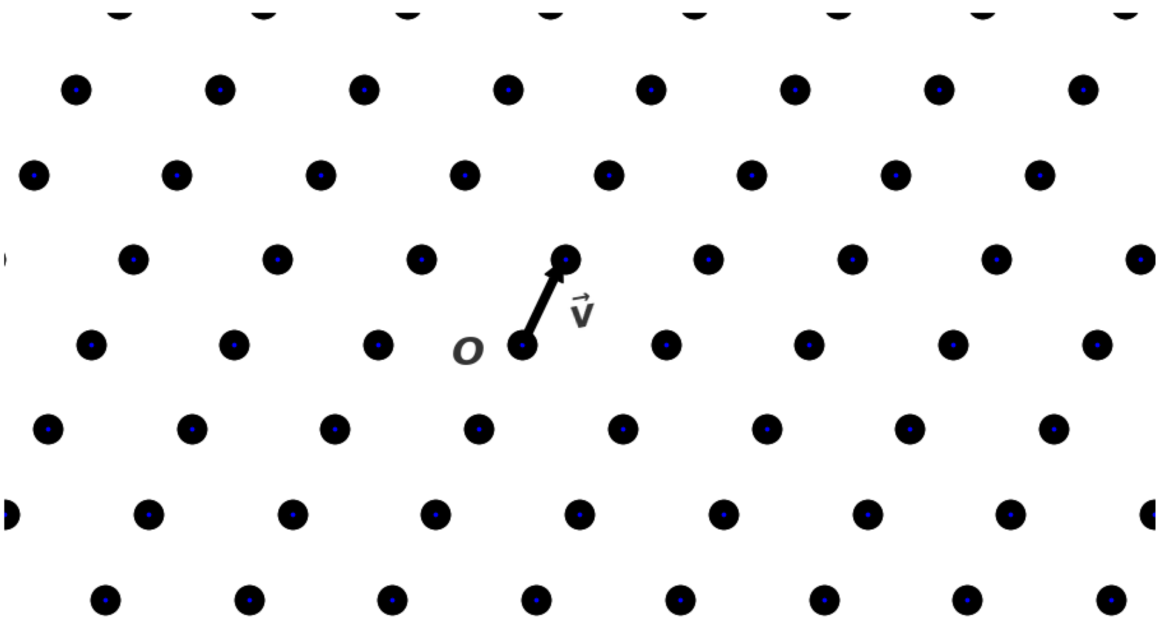
4.10 Euclidean Lattices

The simulation of difficult problems on lattices

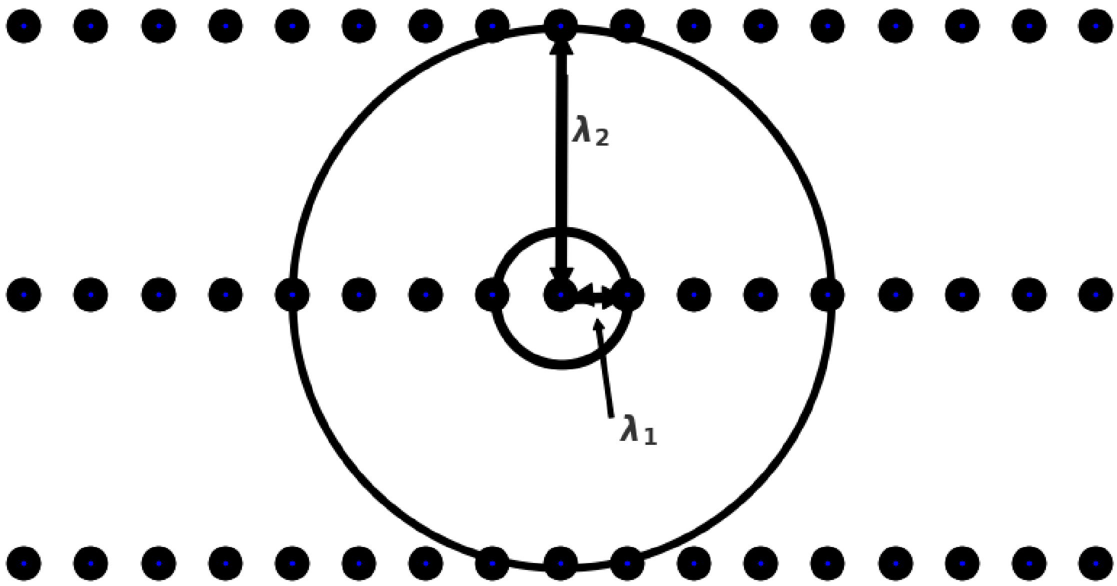
4.10.1 Integer Lattice



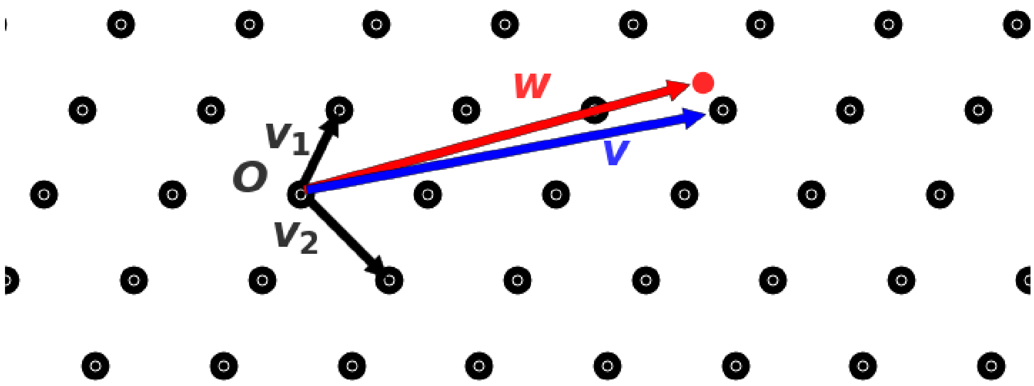
4.10.2 SVP (Shortest Vector Problem)



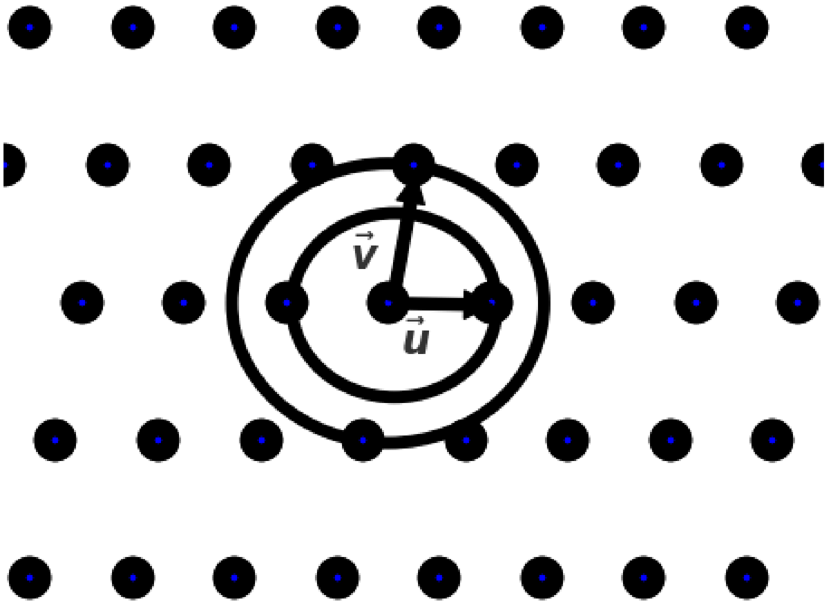
4.10.3 uSVP (Unique Shortest Vector Problem)



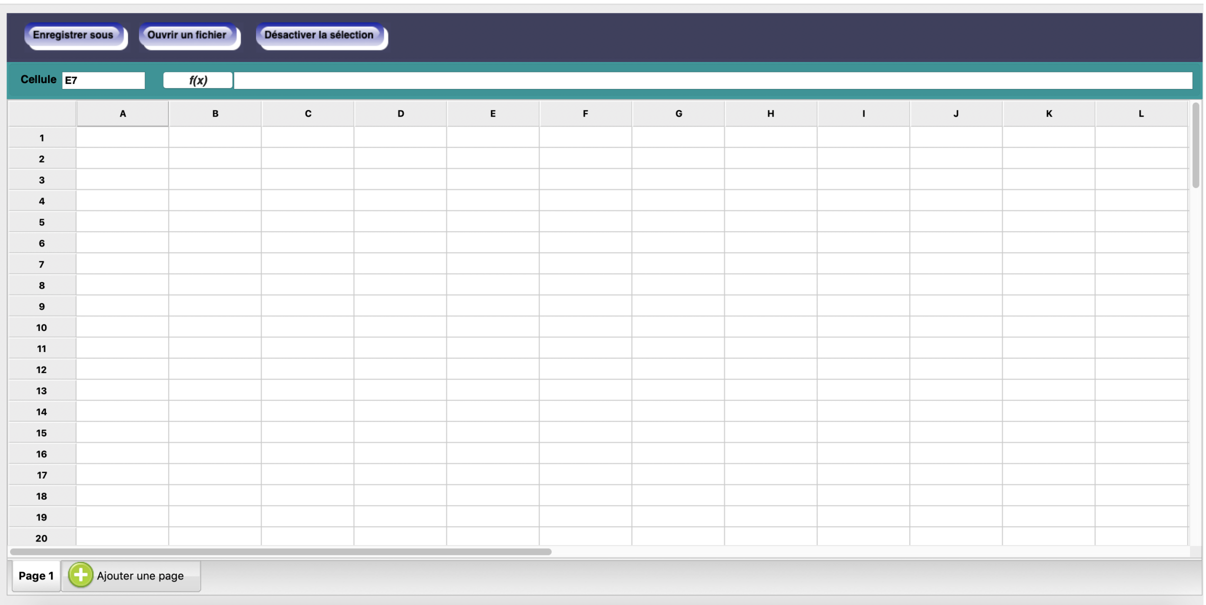
4.10.4 CVP (Closest Vector Problem)



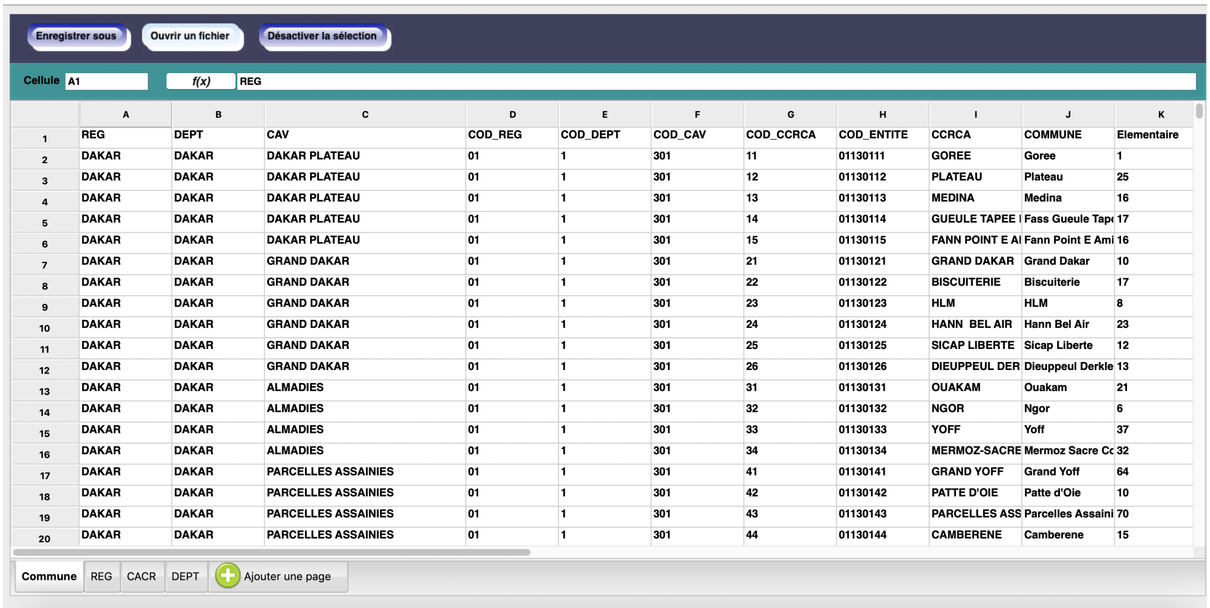
4.10.5 Successive minima

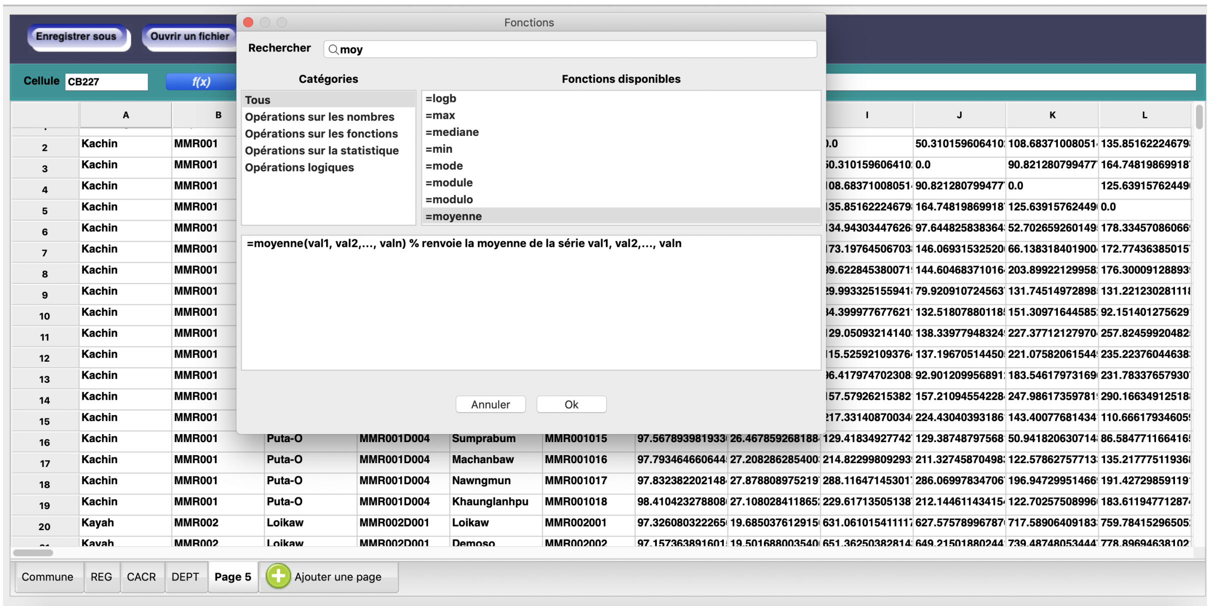


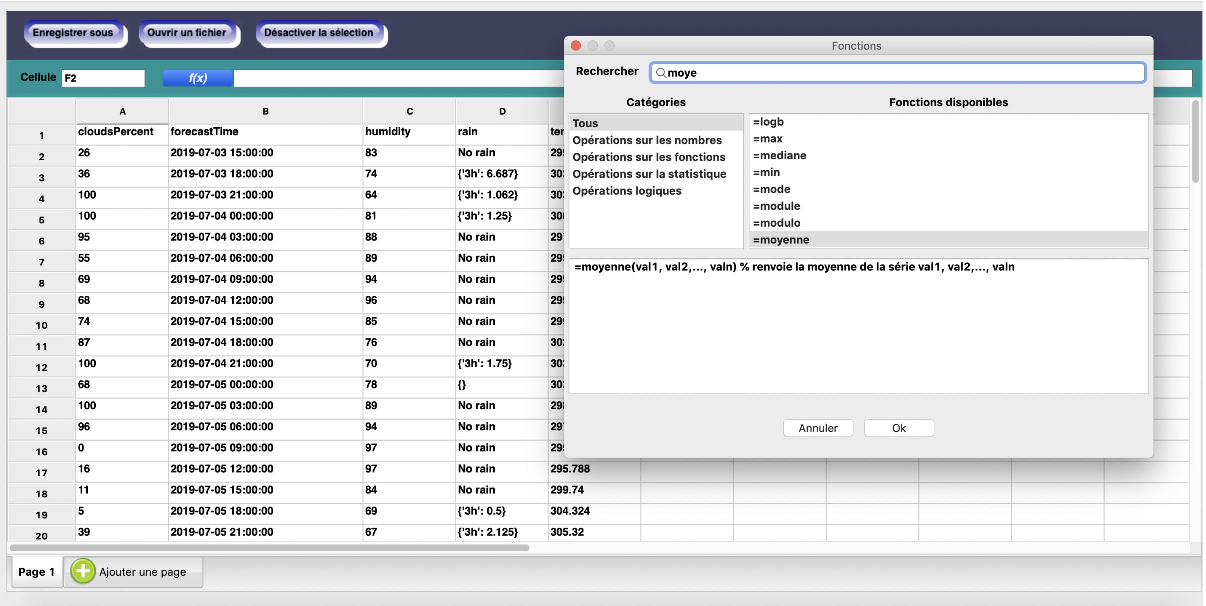
4.11 Spreadsheet

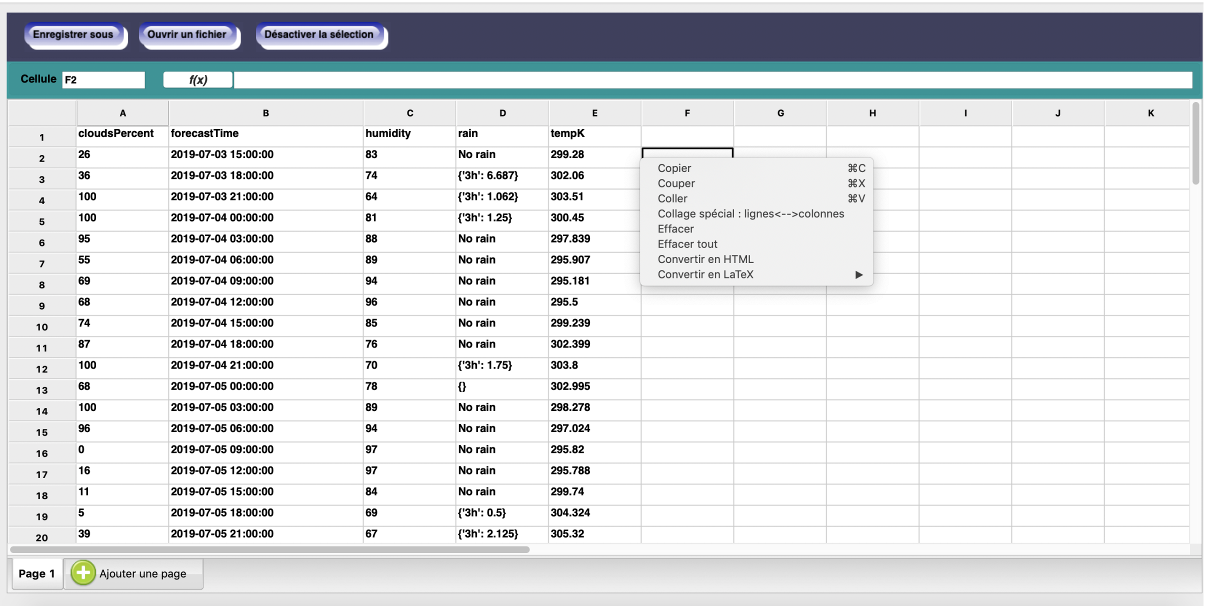


4.11.1 Excel or CSV files

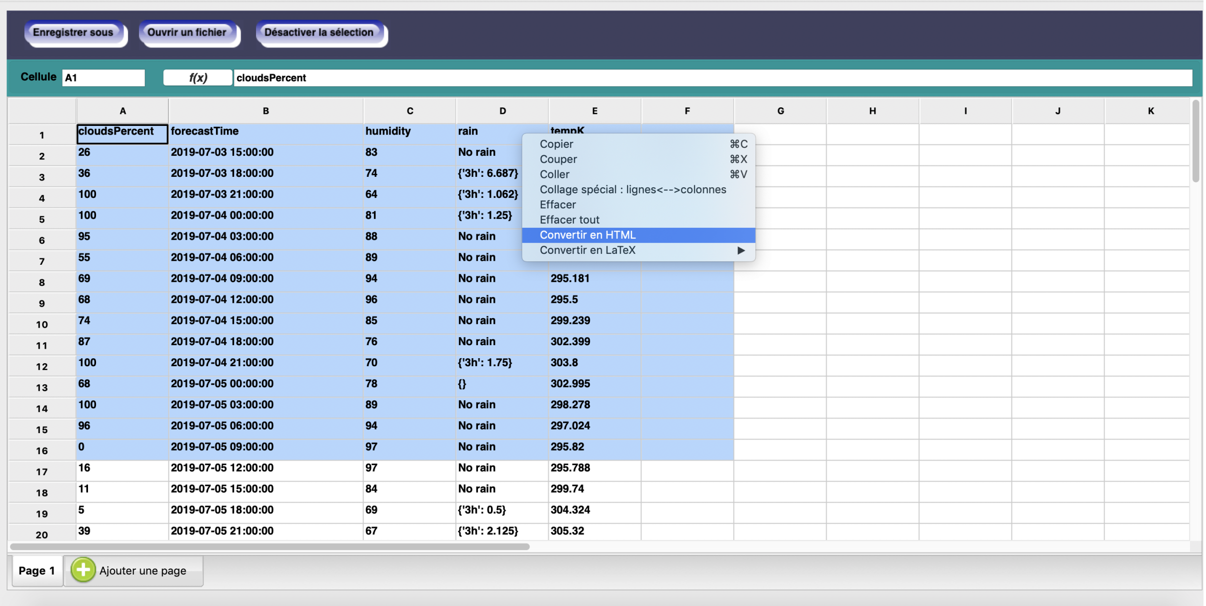


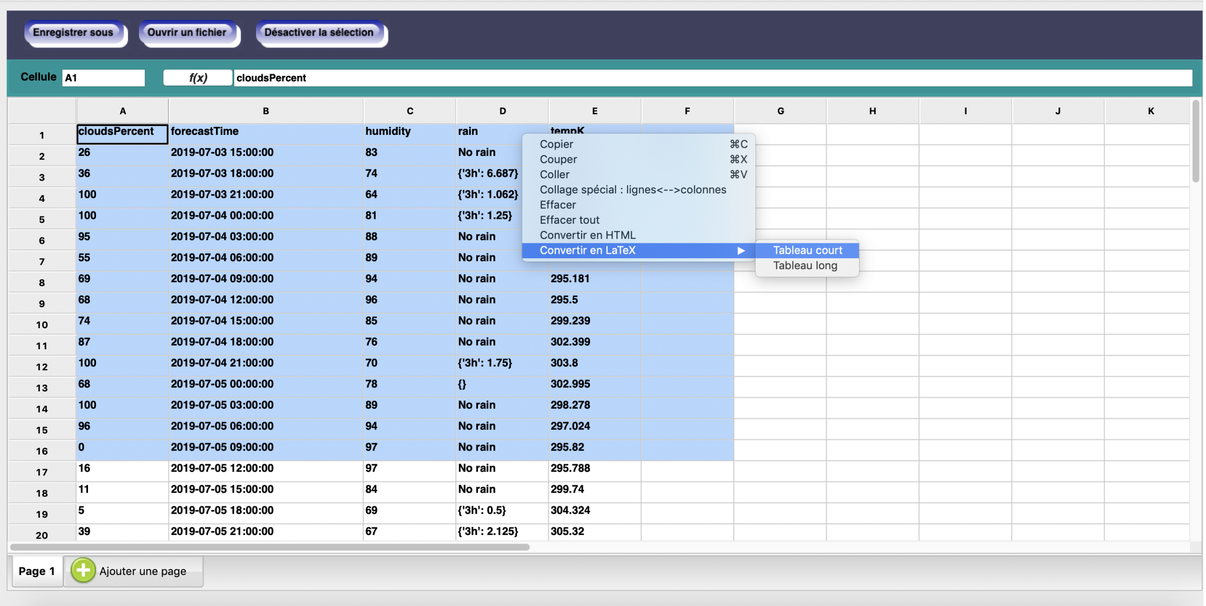


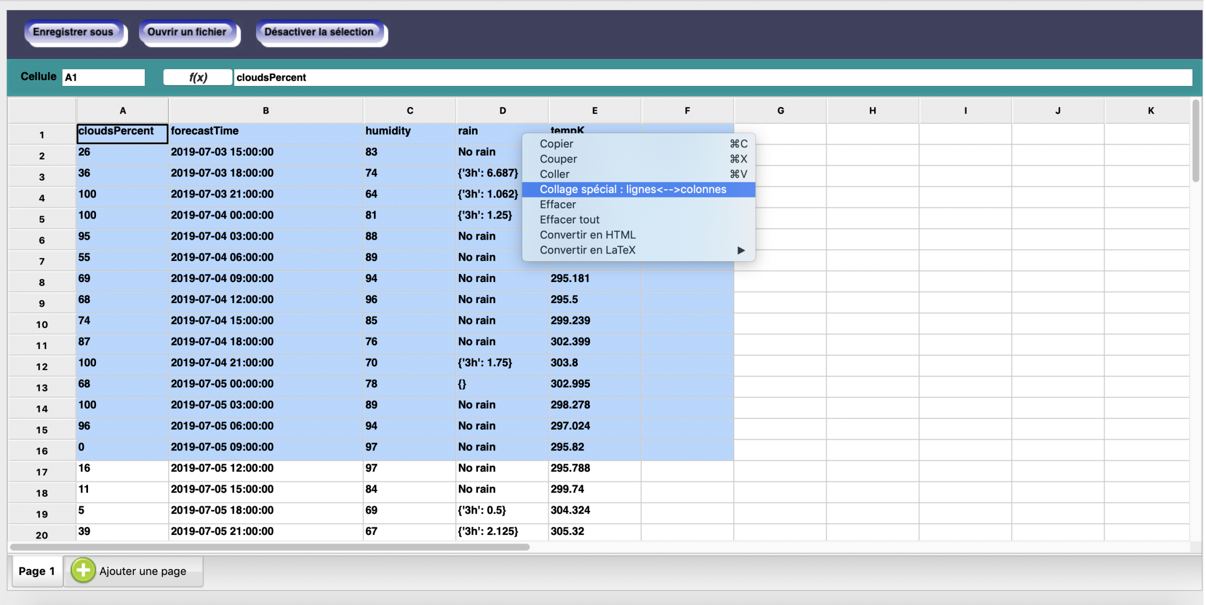

Enregistrer sous
Ouvrir un fichier
Désactiver la sélection
Cellule F2
f(x)
Fonctions
Rechercher
moye
Catégories
Fonctions disponibles
Tous
Opérations sur les nombres
Opérations sur les fonctions
Opérations sur la statistique
Opérations logiques
=logb
=max
=mediane
=min
=mode
=module
=modulo
=moyenne
=moyenne(val1, val2,..., valn) % renvoie la moyenne de la série val1, val2,..., valn
Annuler
Ok
Page 1
Ajouter une page


Enregistrer sous
Ouvrir un fichier
Désactiver la sélection
Cellule F2
f(x)
Copier ⌘C
Couper ⌘X
Coller ⌘V
Collage spécial : lignes<-->colonnes
Effacer
Effacer tout
Convertir en HTML
Convertir en LaTeX
Page 1
Ajouter une page

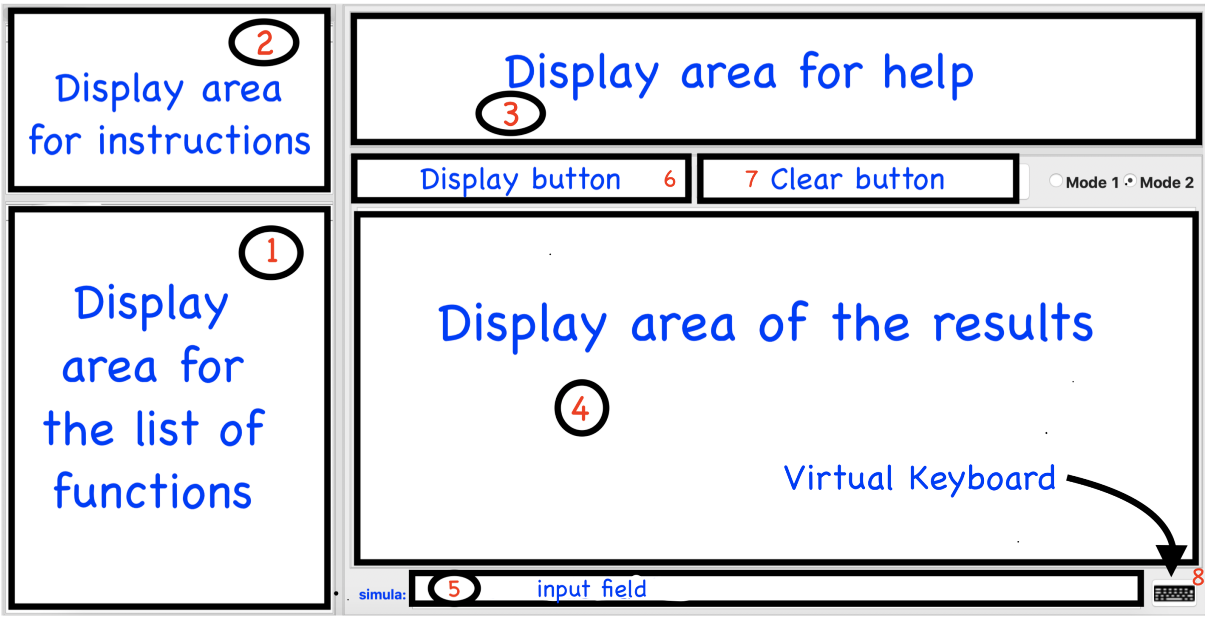
4.11.2 Operations on the spreadsheet







4.12 Computations



You can do some operations in different areas of mathematics without some background on programming.

- Operations over linear algebra
- Operations over functions and sequences
- Operations over finite fields and polynomials mod p
- Operations over groebner bases and multivariate polynomials
- Operations over linear codes
- Operations over classical cryptossytems

Chapter 5

Programming interface

5.1 Introduction

SimulaMath module is built on top of the scientific Python packages like Numpy, Scipy, Sympy, and Mpmath.

5.1.1 Special Simula Syntax

- **Fractions**: on simula, the result of the division of two Integers is a fraction. But on Python, it is a `float` number.

```
simula : 2/3
2/3
simula : 4/10
2/5
simula : 1/2 + 1/5 + 3/5
13/10
simula : int(1)/int(2)
0.5
```

- **Multiplication**: the multiplication under simula is "*" as in Python.

```
simula : x = 2; x
2
simula : 3*x
6
```

There are some special cases of multiplication.

When a number is followed by a variable, it means multiplication.

```
simula : x = 2; x
2
```

(continues on next page)

(continued from previous page)

```
simula : 3x
6
simula : 5x -2
8
```

When a number is followed by an open parenthesis "(", it means multiplication.

```
simula : x = 2; x
2
simula : 3(x+2)
12
simula : 5(2x-1)
15
```

When a closed parenthesis ")" is followed by an open parenthesis "(", it means multiplication.

```
simula : x = 3; x
3
simula : (x-1)(x+2)
10
simula : 5(2x-1)(x-1)
50
```

- **Power**: the power under simula is "^" or "**" as in Python.

```
simula : 2^3
8
simula : x = 3; 2x^2
18
simula : (x - 1)(2x^3 -10)
88
```

Remark : The symbol "^" means bitwise XOR in Python, but on simula, the equivalent operator is "^^".

EXAMPLE:

```
simula : bin(0b100101 ^^ 0b001010)
'0b101111'
```

- **Factorial** : A number followed by "!" symbol means factorial.

```
simula : 3!
6
simula : 3! == 6
True
simula : 3! != 6
False
```

(continues on next page)

(continued from previous page)

```
simula : 6!/4!
30
```

- **Special Sequences** : **[a, b, . . . , n], (a, b, . . . , n) or {a, b, . . . , n}**.

```
simula : [1, 3, ..., 11]
[1, 3, 5, 7, 9, 11]
simula : {1, 3, ..., 11}
{1, 3, 5, 7, 9, 11}
simula : (1, 3, ..., 11)
(1, 3, 5, 7, 9, 11)
simula : [10, 20, ..., 100]
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

- **Symbolic variables**:

Symbolic Variables

`simula.api.symbols.`**`var`**(*names*, *domain=None*, *parity=None*, ***, *globals=True*, ***kwargs*)

Create symbols and inject them into the global namespace.

Valid *kwargs*:

- commutative : True or False

EXAMPLES:

```
simula : var('x')
x
simula : x
x
simula : var('x, y', "RR")   # x and y are real numbers
(x, y)
simula : x.is_real and y.is_real
True
simula : var('z', "RR*+")   # z is a positive real number
z
simula : z > 0
True
simula : z < 0
False
simula: z > 4
z > 4
simula : n = var('n', "NN"); n   # n is a non-negative integer␣
↪number
n
simula : n >= 0
True

simula : var('x, y2, ab')
```

(continues on next page)

(continued from previous page)

```
(x, y2, ab)
simula : y2
y2
simula : var(('a', 'b', 'c'))
(a, b, c)
simula : var(['a', 'b', 'c'])
[a, b, c]
simula : var({'a', 'b', 'c'})
{a, b, c}
simula : var('x:z')
(x, y, z)
simula : var('x1:4')
(x1, x2, x3)
simula : xa, yb = var('x((a:b))')
simula : xa
x(a)
```

Parameters **globals** (*bool*) –

- **Functions** : You can define a function easily on simula like in mathematics.

```
simula : x = var('x')
simula : f(x) = x^2-2x-2; f
Function defined by x |--> x^2 - 2x - 2
simula : f(2)
-2
simula : f(2x-1)
-4x + (2x - 1)^2
simula : y = var('y')
simula : g(x, y) = x - y + 1; g
Function defined by (x, y) |--> x - y + 1
simula : g(x, x)
1
```

- **Complex numbers**: The imaginary unit is represented by **I**.

```
simula : 3-5I
3-5I
simula : conjugate(3-5I)
3 + 5I
simula : real_part(3-5I)
3
simula : im_part(3-5I)
-5
```

Python complex numbers are compatible with Simula complex numbers.

```
simula : 2-5j
3-5I
```

(continues on next page)

(continued from previous page)

```
simula : real_part(2-5j)
3
```

- **Polynomial ring** : You can define a polynomial ring like in Sage.

```
simula : R.<x, y, z> = QQ[]
simula : R
Multivariate Polynomial Ring in x, y, z over QQ with␣
↪deglex order
simula : (x^2-1).factor()
(x - 1)*(x + 1)
simula : F.<w> = GF(3)[]; F
Univariate Polynomial Ring in w over GF(3) with deglex␣
↪order
simula : 5w^4+10w^2-2
2w^4 + w^2 - 2
```

- **Finite Fields** : You can define a finite field like in Sage.

```
simula : G.<a> = GF(9); G
Finite Field of 9 elements defined by the quotient of F_
↪3[a] by the ideal <a^2 + 2a + 2>
simula : a^2
a + 1
simula : 7a^3
2a + 1
simula : 1/a
a + 2
```

- **Binary, Octal and Hexadecimal**:
  - **Python Binary, Octal and Hexadecimal** :

```
simula : 0b1110
14
simula : bin(14)
'0b1110'
simula : oct(100)
'0o144'
simula : hex(1000)
'0x3e8'
```

  - **Simula Binary, Octal and Hexadecimal** :

```
simula : Bin(14)
0b1110
simula : A = Bin(111); A
0b1101111
simula : A.to_list()
[1, 1, 0, 1, 1, 1, 1]
```

(continues on next page)

(continued from previous page)

```
simula : A.to_list(10)
[0, 0, 0, 1, 1, 0, 1, 1, 1, 1]
simula : Bin(14) + Bin(17)
0b11111
simula : Bin(14) + Bin(17) == Bin(31)
True
simula : Bin(bin(14))
0b1110
simula : Oct(1000)
0o1750
simula : Hex(1000)
0x3e8
simula : Hex(100) + Hex(120) == Hex(220)
True
```

5.1.2 Simula Syntaxe as Python

Since SimulaMath language is based on Python, 99% of Python valid code work also on SimulaMath.

- Float numbers:

```
simula : 7.8
7.8
simula : 6.
6.0
simula : .5
0.5
```

- Exponents:

```
simula : 2e3
2000.0
simula : 3e-4
0.0003
simula : 3e+4
30000.0
```

- Lists:

```
simula : seq = [1,2,3,4,5]; print(seq)
[1, 2, 3, 4, 5]
simula : seq[0]
1
simula : seq[:2]
[1, 2]
simula : seq[-2:]
[4, 5]
```

Comprehension of list

```
simula : [i^2 for i in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
simula : x = var('x'); f(x) = x^2-4x-1
simula : [f(i) for i in range(15)]
[-1, -4, -5, -4, -1, 4, 11, 20, 31, 44, 59, 76, 95, 116,
↪ 139]
```

- Tuples:

```
simula : seq2 = (1,2,3,4,5); print(seq2)
(1, 2, 3, 4, 5)
simula : seq[-1]
5
simula : seq[:2]
[1, 2]
simula : seq[-2:]
[4, 5]
```

- Sets:

```
simula : A = {1,2,3,4,5, 10, 15}; print(A)
{1, 2, 3, 4, 5, 10, 15}
simula : len(A)
7
simula : B = {-2, 4}; B
{4, -2}
simula : A | B
{1, 2, 3, 4, 5, 10, 15, -2}
simula : A & B
{4}
```

Comprehension of Set

```
simula : {i^2 for i in range(10)}
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
simula : x = var('x'); f(x) = x^2-4x-1
simula : {f(i) for i in range(15)}
{4, 59, 11, 44, 76, 139, 20, 116, 95, -5, -4, -1, 31}
```

- Strings:

```
simula : word = "SimulaMath"; word
'SimulaMath'
simula : word.upper()
'SIMULAMATH'
simula : word.isalpha()
True
simula : word[2:]
'mulaMath'
```

(continues on next page)

(continued from previous page)

```
simula : "Simula" "Math"
'SimulaMath'
simula : a, b = 2, 8
simula : "We get a = {} and b = {}".format(a, b)
'We get a = 2 and b = 8'
simula : f"We get a = {a} and b = {b}"
'We get a = 2 and b = 8'
simula : f"We get a = {2a} and b = {b^2}"
'We get a = 4 and b = 64'
```

- Dictionaries:

```
simula : dico = {'A': 0, "B": 1, 3: (1,2,3)};␣
↪print(dico)
{'A': 0, 'B': 1, 3: (1, 2, 3)}
simula : list(dico.keys())
['A', 'B', 3]
simula : list(dico.values())
[0, 1, (1, 2, 3)]
simula : del dico['A']; dico
{'B': 1, 3: (1, 2, 3)}
simula : dico["S"] = "SimulaMath"; dico
{'B': 1, 3: (1, 2, 3), 'S': 'SimulaMath'}
```

Comprehension of Dictionary

```
simula : {i : i^2 for i in range(10)}
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8:␣
↪64, 9: 81}
simula : x = var('x'); g(x) = 3x-1
simula : {2m: g(m) for m in range(15)}
{0: -1, 2: 2, 4: 5, 6: 8, 8: 11, 10: 14}
```

Note that conditions, loops (for loop, while loop) and functions syntax on SimulaMath and Python are the same.

- Conditions

```
simula : N = 194
simula : if N % 7 == 0:
.......:     print(f"{N} is a multiple of 7")
.......: else:
.......:      print(f"{N} is not a multiple of 7")
.......:
194 is not a multiple of 7
```

- **Loops**

```
simula : for i in range(9):
.......:     print(2i)
```

(continues on next page)

(continued from previous page)

```
0
2
4
6
8
10
12
14
16
simula : for elt in [0, 5, ..., 30]:
.......:     print(elt)
0
5
10
15
20
25
30
```

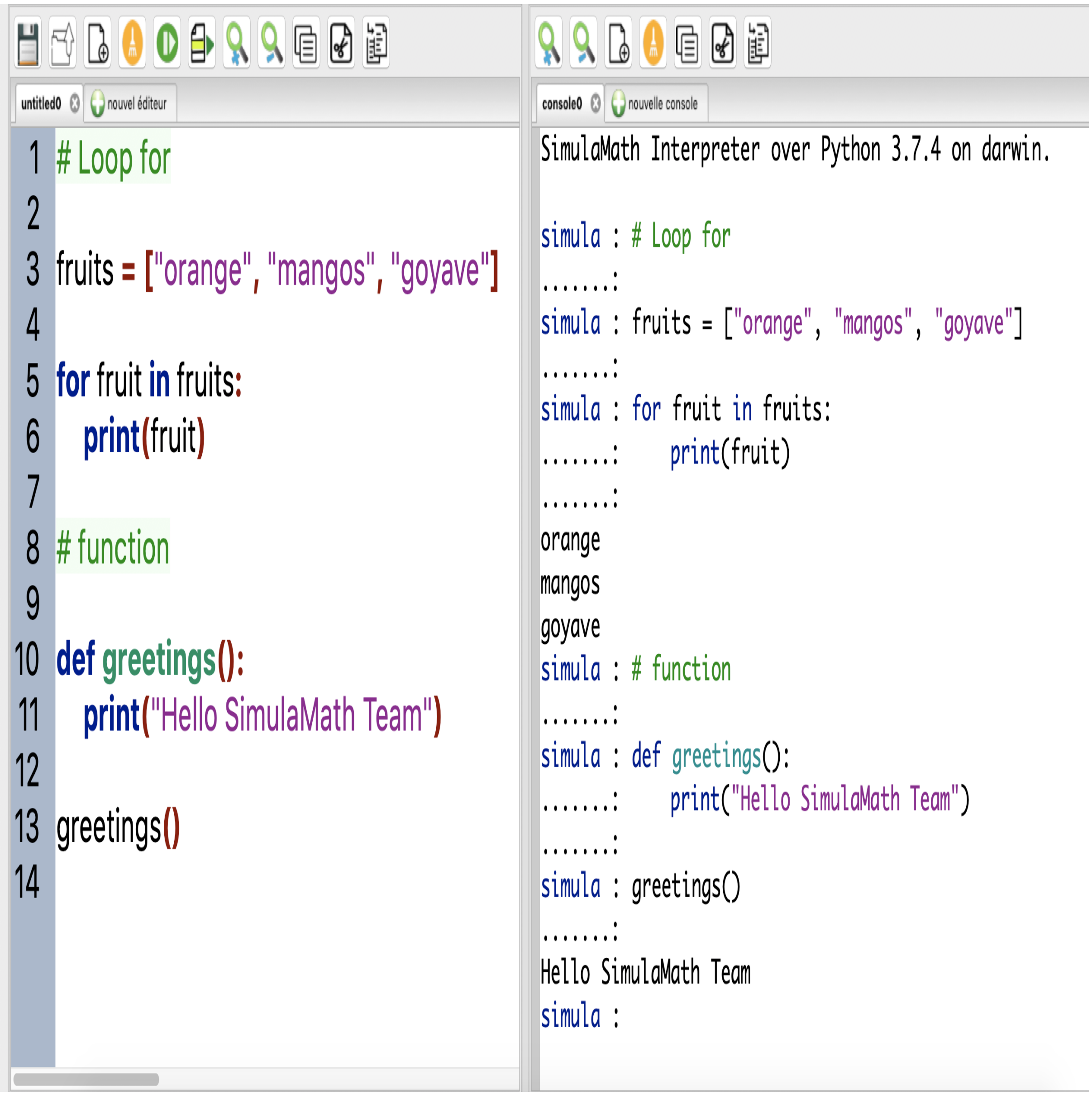
- **Functions**

```
simula : def mean(L):
.......:     return sum(L)/len(L)
.......:
simula : mean([1,2,3,4,5])
3
simula : mean([3,4])
7/2
```

For more details on Python syntax, see the Python Doc

5.1.3 SimulaMath Editor

SimulaMath has a basic editor which allow you to save and load files with extension **.sim** and **.py**.



5.2 Linear Algebra

5.2.1 Matrices

Operations over matrices

`simula.api.linalg.matrices.In(n)`

Returns an identity matrix of order n.

Parameters **n** – an integer

```
simula : identity_matrix(2)
Matrix([
[1, 0],
```

(continues on next page)

(continued from previous page)

```
[0, 1]])
simula : identity_matrix(3)
Matrix([
[1, 0, 0],
[0, 1, 0],
[0, 0, 1]])
```

class simula.api.linalg.matrices.**Matrix**(**args*, ***kwargs*)

Bases: `sympy.matrices.dense.MutableDenseMatrix`, `simula.api.structure.simula_object.SimulaObject`

Base class for Matrices.

EXAMPLES:

```
simula : A = matrix([[-2, 1, 4], [-5, 2, 5], [-1, 1, 3]]); A
Matrix([
[-2, 1, 4],
[-5, 2, 5],
[-1, 1, 3]])
simula : A.det()
-4
simula : A.rank()
3
simula : A.trace()
3
simula : A.transpose()
Matrix([
[-2, -5, -1],
[ 1,  2,  1],
[ 4,  5,  3]])
simula : A^3
Matrix([
[-19, 12, 27],
[-15,  8, 15],
[-18, 12, 26]])
simula : A.charpoly('x')
PurePoly(x^3 - 3x^2 + 4, x, domain='ZZ')
simula : A.eigenvals()
{-1: 1, 2: 2}
simula : A.cofactor_matrix()
Matrix([
[ 1,  10, -3],
[ 1,  -2,  1],
[-3, -10,  1]])
simula : B = matrix([[1, 2], [3, 4]]); B
Matrix([
[1, 2],
[3, 4]])
simula : C = matrix([[5, 6, 7], [8, 9, 10]]); C
```

(continues on next page)

(continued from previous page)

```
Matrix([
[5, 6,  7],
[8, 9, 10]])
simula : matrix([B, C])
Matrix([
[1, 2, 5, 6,  7],
[3, 4, 8, 9, 10]])
simula : matrix([[B, C], [B, C]])
[1, 2, 5, 6,  7],
[3, 4, 8, 9, 10],
[1, 2, 5, 6,  7],
[3, 4, 8, 9, 10]])
```

algebraic_multiplicity(*lamda*)

Returns the algebraic multiplicity of `lamda`.

EXAMPLES:

```
simula : A = matrix([[-2, 1, 4], [-5, 2, 5], [-1, 1, 3]]); A
Matrix([
[-2, 1, 4],
[-5, 2, 5],
[-1, 1, 3]])
simula : A.eigenvalues()
{-1: 1, 2: 2}
simula : A.algebraic_multiplicity(2)
2
```

static circulant(*v*, *shift=None*)

Returns a circulant matrix.

See also `circulant_matrix`

dunford_decomposition()

Returns the Dunford decomposition of `self`.

EXAMPLES:

```
simula : A = matrix([[-2, 1, 4], [-5, 2, 5], [-1, 1, 3]]); A
Matrix([
[-2, 1, 4],
[-5, 2, 5],
[-1, 1, 3]])
simula : B, N = A.dunford_decomposition(); B, N
(Matrix([
[-1/3, 0, 7/3],
[  -5, 2,   5],
[ 2/3, 0, 4/3]]), Matrix([
[-5/3, 1, 5/3],
[   0, 0,   0],
[-5/3, 1, 5/3]]))
```

(continues on next page)

(continued from previous page)

```
simula : B*N == N*B
True
simula : B.is_diagonalizable()
True
simula : N.is_nilpotent()
True
simula : B+N
Matrix([
[-2, 1, 4],
[-5, 2, 5],
[-1, 1, 3]])
```

eigenvalues(**args*, ***kwargs*)
Returns the eigenvalues of `self`.

EXAMPLES:

```
simula : A = matrix([[-2, 1, 4], [-5, 2, 5], [-1, 1, 3]]); A
Matrix([
[-2, 1, 4],
[-5, 2, 5],
[-1, 1, 3]])
simula : A.eigenvalues()
{-1: 1, 2: 2}
```

eigenvectors_left(***kwargs*)
Returns the left eigenvectors of `self`.

EXAMPLES:

```
simula : A = matrix([[-2, 1, 4], [-5, 2, 5], [-1, 1, 3]]); A
Matrix([
[-2, 1, 4],
[-5, 2, 5],
[-1, 1, 3]])
simula : A.eigenvectors_left()
[(-1, 1, [Matrix([[-1, 0, 1]])]), (2, 2, [Matrix([[-1, 3/5,␣
↪1]])])]
```

eigenvectors_right(***kwargs*)
Returns the right eigenvectors of `self`.

EXAMPLES:

```
simula : A = matrix([[-2, 1, 4], [-5, 2, 5], [-1, 1, 3]]); A
Matrix([
[-2, 1, 4],
[-5, 2, 5],
[-1, 1, 3]])
simula : A.eigenvectors_right()
```

(continues on next page)

(continued from previous page)

```
[(-1, 1, [Matrix([
[ -7/2],
[-15/2],
[    1]])]), (2, 2, [Matrix([
[1],
[0],
[1]])])]
```

geometric_multiplicity(*lamda*)

Returns the geometric multiplicity of `lamda`.

EXAMPLES:

```
simula : A = matrix([[-2, 1, 4], [-5, 2, 5], [-1, 1, 3]]); A
Matrix([
[-2, 1, 4],
[-5, 2, 5],
[-1, 1, 3]])
simula : A.eigenvalues()
{-1: 1, 2: 2}
simula : A.geometric_multiplicity(2)
1
```

linear_map(*domain=None*, *codomain=None*)

Returns the linear map associated to `self`.

Parameters **domain** – (optional) a field or vector space

:param codomain : (optional) a field or vector space

EXAMPLES:

```
simula : A = matrix([[-2, 1, 4], [-5, 2, 5], [-1, 1, 3]]); A
Matrix([
[-2, 1, 4],
[-5, 2, 5],
[-1, 1, 3]])
simula : f = A.linear_map(RR, RR)
simula : f
Linear map from RR^3 --> RR^3 defined by (x1, x2, x3) |-->␣
↪(-2x1 + x2 + 4x3, -5x1 + 2x2 + 5x3, -x1 + x2 + 3x3)
simula : f(1, 0, 0)
(-2, -5, -1)
```

reverse_cols()

Returns a matrix when its columns are the reversed columns of `self`.

EXAMPLES:

```
simula : A = matrix([[-2, 1, 4], [-5, 2, 5], [-1, 1, 3]]); A
Matrix([
```

(continues on next page)

(continued from previous page)

```
[-2, 1, 4],
[-5, 2, 5],
[-1, 1, 3]])
simula : A.reverse_cols()
Matrix([
[-1, 1, 3],
[-5, 2, 5],
[-2, 1, 4]])
```

reverse_rows()

Returns a matrix when its rows are the reversed rows of `self`.

EXAMPLES:

```
simula : A = matrix([[-2, 1, 4], [-5, 2, 5], [-1, 1, 3]]); A
Matrix([
[-2, 1, 4],
[-5, 2, 5],
[-1, 1, 3]])
simula : A.reverse_rows()
Matrix([
[4, 1, -2],
[5, 2, -5],
[3, 1, -1]])
```

rref_mod(*gf*)

Returns the row reduced echelon form in finite field GF(q).

Parameters **gf** – a finite field GF(q)

EXAMPLES:

```
simula : A = matrix([[2, 1, 0, 0, 1, 1, -1], [1, 1, 0, 1, 0,
↪ -1, 0], [2, -1, 1, 1, 0, 1, -1]]); A
Matrix([
[2,  1, 0, 0, 1,  1, -1],
[1,  1, 0, 1, 0, -1,  0],
[2, -1, 1, 1, 0,  1, -1]])
simula : A.rref_mod(GF(3))
Matrix([
[1, 0, 0, 2, 1, 2, 2],
[0, 1, 0, 2, 2, 0, 1],
[0, 0, 1, 2, 0, 0, 2]])
simula : A.rref_mod(GF(5))
Matrix([
[1, 0, 0, 4, 1, 2, 4],
[0, 1, 0, 2, 4, 2, 1],
[0, 0, 1, 0, 2, 4, 2]])
```

spectral_radius()

Returns the spectral radius of `self`.

EXAMPLES:

```
simula : A = matrix([[-2, 1, 4], [-5, 2, 5], [-1, 1, 3]]); A
Matrix([
[-2, 1, 4],
[-5, 2, 5],
[-1, 1, 3]])
simula : A.spectral_radius()
2
```

spectrum()

Returns the spectrum of `self`.

EXAMPLES:

```
simula : A = matrix([[-2, 1, 4], [-5, 2, 5], [-1, 1, 3]]); A
Matrix([
[-2, 1, 4],
[-5, 2, 5],
[-1, 1, 3]])
simula : A.spectrum()
{2, -1}
```

`simula.api.linalg.matrices.`**block_matrix**

alias of `simula.api.linalg.matrices.BlockMatrix`

`simula.api.linalg.matrices.`**circulant_matrix**(*v*, *shift=None*)

Returns a circulant matrix.

Parameters

- **v** – a vector
- **shift** – (optional) the number of rows. If it is not `None`, it should be less or equal than the length of `v`

EXAMPLES:

```
simula : v = [1,2,3,4,5]
simula : circulant_matrix(v)
Matrix([
[1, 2, 3, 4, 5],
[5, 1, 2, 3, 4],
[4, 5, 1, 2, 3],
[3, 4, 5, 1, 2],
[2, 3, 4, 5, 1]])
simula : circulant_matrix(v, shift=3)
Matrix([
[1, 2, 3, 4, 5],
[5, 1, 2, 3, 4],
[4, 5, 1, 2, 3]])
```

`simula.api.linalg.matrices.`**companion_matrix**(*poly*, *format='right'*)

Returns a companion matrix associated to a polynomial for a given format.

Parameters

- **poly** – a polynomial
- **format** – a string ("right", "left", "top", "bottom"), default is "right"

EXAMPLES:

```
simula : p = x^5-x^4-3x^3-x^2+5x-6
simula : companion_matrix(p)
Matrix([
[0, 0, 0, 0,  6],
[1, 0, 0, 0, -5],
[0, 1, 0, 0,  1],
[0, 0, 1, 0,  3],
[0, 0, 0, 1,  1]])
simula : companion_matrix(p, format='left')
Matrix([
[ 1, 1, 0, 0, 0],
[ 3, 0, 1, 0, 0],
[ 1, 0, 0, 1, 0],
[-5, 0, 0, 0, 1],
[ 6, 0, 0, 0, 0]])
simula : companion_matrix(p, format='top')
Matrix([
[1, 3, 1, -5, 6],
[1, 0, 0,  0, 0],
[0, 1, 0,  0, 0],
[0, 0, 1,  0, 0],
[0, 0, 0,  1, 0]])
simula : companion_matrix(p, format='bottom')
Matrix([
[0,  1, 0, 0, 0],
[0,  0, 1, 0, 0],
[0,  0, 0, 1, 0],
[0,  0, 0, 0, 1],
[6, -5, 1, 3, 1]])
```

simula.api.linalg.matrices.**diag**(**args*)

Returns a diagonal matrix.

EXAMPLES:

```
simula : diagonal_matrix(1, 2, 3)
Matrix([
[1, 0, 0],
[0, 2, 0],
[0, 0, 3]])
simula : diagonal_matrix(-2, 2, 1, 1)
Matrix([
[-2, 0, 0, 0],
```

(continues on next page)

(continued from previous page)

```
[ 0, 2, 0, 0],
[ 0, 0, 1, 0],
[ 0, 0, 0, 1]])
```

simula.api.linalg.matrices.**diagonal_matrix**(**args*)

Returns a diagonal matrix.

EXAMPLES:

```
simula : diagonal_matrix(1, 2, 3)
Matrix([
[1, 0, 0],
[0, 2, 0],
[0, 0, 3]])
simula : diagonal_matrix(-2, 2, 1, 1)
Matrix([
[-2, 0, 0, 0],
[ 0, 2, 0, 0],
[ 0, 0, 1, 0],
[ 0, 0, 0, 1]])
```

simula.api.linalg.matrices.**hilbert_matrix**(*n*)

Return a Hilbert matrix of the given dimension.

H_ij = 1/(i + j - 1) for i, j = 1, . . . , n

Parameters **n** – an integer

EXAMPLES:

```
simula : hilbert_matrix(2)
Matrix([
[  1, 1/2],
[1/2, 1/3]])
simula : hilbert_matrix(3)
Matrix([
[  1, 1/2, 1/3],
[1/2, 1/3, 1/4],
[1/3, 1/4, 1/5]])
simula : hilbert_matrix(4)
Matrix([
[  1, 1/2, 1/3, 1/4],
[1/2, 1/3, 1/4, 1/5],
[1/3, 1/4, 1/5, 1/6],
[1/4, 1/5, 1/6, 1/7]])
```

simula.api.linalg.matrices.**identity_matrix**(*n*)

Returns an identity matrix of order n.

Parameters **n** – an integer

```
simula : identity_matrix(2)
Matrix([
[1, 0],
[0, 1]])
simula : identity_matrix(3)
Matrix([
[1, 0, 0],
[0, 1, 0],
[0, 0, 1]])
```

`simula.api.linalg.matrices.`**`jordan_cell`**`(`*lamda*, *dim*`)`

Returns a jordan block of dimension `dim` associated to the eigenvalue `lamda`.

Param an eigenvalue

Parameters **`dim`** – (an integer) the number of rows and columns

EXAMPLES:

```
simula : jordan_cell(2, 3)
Matrix([
[2, 1, 0],
[0, 2, 1],
[0, 0, 2]])
simula : jordan_cell(-3, 4)
Matrix([
[-3,  1,  0,  0],
[ 0, -3,  1,  0],
[ 0,  0, -3,  1],
[ 0,  0,  0, -3]])
```

`simula.api.linalg.matrices.`**`linear_system_to_matrix`**`(`*expr*, *symbols=None*`)`

Returns the two matrices associated to a linear system.

Parameters

- **`expr`** – an expression
- **`symbols`** – list of symbols

Return type (<class 'simula.api.linalg.matrices.Matrix'>, <class 'simula.api.linalg.matrices.Matrix'>)

EXAMPLES

```
simula : x, y , z = var('x,y,z')
simula : A, b = linear_system_to_matrix([x+y-z-1, 2x-3y-z-7,␣
↪2x+z-5], (x,y,z))
simula : A
Matrix([
[1,  1, -1],
[2, -3, -1],
```

(continues on next page)

(continued from previous page)

```
[2,  0,  1]])
simula : b
Matrix([
[1],
[7],
[5]])
```

simula.api.linalg.matrices.**matrix**
alias of *simula.api.linalg.matrices.Matrix*

simula.api.linalg.matrices.**ones**(*rows*, *cols=None*)
Returns a matrix when all its coefficients are equal to one.

Parameters

- **rows** – (an integer) the number or rows
- **cols** – (optional) the number or columns

EXAMPLES:

```
simula : ones_matrix(2, 4)
Matrix([
[1, 1, 1, 1],
[1, 1, 1, 1]])
simula : ones_matrix(3)
Matrix([
[1, 1, 1],
[1, 1, 1],
[1, 1, 1]])
```

simula.api.linalg.matrices.**ones_matrix**(*rows*, *cols=None*)
Returns a matrix when all its coefficients are equal to one.

Parameters

- **rows** – (an integer) the number or rows
- **cols** – (optional) the number or columns

EXAMPLES:

```
simula : ones_matrix(2, 4)
Matrix([
[1, 1, 1, 1],
[1, 1, 1, 1]])
simula : ones_matrix(3)
Matrix([
[1, 1, 1],
[1, 1, 1],
[1, 1, 1]])
```

simula.api.linalg.matrices.**zero_matrix**(*rows*, *cols=None*)

Returns a zero matrix with rows `rows` and cols `cols`. If `cols` is not given it is equal to the `rows`.

Parameters

- **rows** – (an integer) the number or rows
- **cols** – (optional) the number or columns

EXAMPLES:

```
simula : zero_matrix(2, 3)
Matrix([
[0, 0, 0],
[0, 0, 0]])
simula : zero_matrix(3)
Matrix([
[0, 0, 0],
[0, 0, 0],
[0, 0, 0]])
```

`simula.api.linalg.matrices.`**`zeros`**(*rows*, *cols=None*)

Returns a zero matrix with rows `rows` and cols `cols`. If `cols` is not given it is equal to the `rows`.

Parameters

- **rows** – (an integer) the number or rows
- **cols** – (optional) the number or columns

EXAMPLES:

```
simula : zero_matrix(2, 3)
Matrix([
[0, 0, 0],
[0, 0, 0]])
simula : zero_matrix(3)
Matrix([
[0, 0, 0],
[0, 0, 0],
[0, 0, 0]])
```

5.2.2 Vector Spaces

Vectors Spaces

Classes

- VectorSpace K^n
- SubSpace
- Vect

- MatrixSpace
- vector

class simula.api.linalg.vector_space.**MatrixSpace**(*field*, *rows=None*, *cols=None*)

Bases: *simula.api.linalg.vector_space.VectorSpace*, abc.ABC

Representation of Matrix spaces

Parameters

- **field** – a field
- **rows** – the number of rows
- **cols** – the number of columns

EXAMPLES:

```
simula : M = MatrixSpace(QQ, 3, 2)
Matrix Space of dimension 3 x 2 over Rational Numbers
simula : M.canonical_basis()
[Matrix([
[1, 0],
[0, 0],
[0, 0]]), Matrix([
[0, 1],
[0, 0],
[0, 0]]), Matrix([
[0, 0],
[1, 0],
[0, 0]]), Matrix([
[0, 0],
[0, 1],
[0, 0]]), Matrix([
[0, 0],
[0, 0],
[1, 0]]), Matrix([
[0, 0],
[0, 0],
[0, 1]])]
simula : A = matrix([[1, 2.0], [0.5, 2.5], [0.3, 1]]) ; A
Matrix([
[  1, 2.0],
[0.5, 2.5],
[0.3,   1]])
simula : M(A)
Matrix([
[   1,   2],
[ 1/2, 5/2],
[3/10,   1]])
```

are_linearly_independent (*family*)
Tests if the vectors in``family`` are linearly independent in `self`.

Parameters **family** (*Union[Iterable, Sized]*) –

canonical_basis ()
Returns a canonical basis of `self`.

change_field (*field*)
Returns a matrix space of same dimension of `self` for the new `field`.

get_a_basis ()
Returns a basis of `self`.

get_component_in_basis (*v*, *family=None*)
Returns the components of the vector `v` in `family`.

is_basis (*family*)
Tests if `family` is a basis of `self`.

Parameters **family** (*Union[Iterable, Sized]*) –

linear_combination (*family*, *coeffs*)
Returns a linear combination of the vectors in `family` by the coefficients in `self`.

random_element (*a=None*, *b=None*)
Returns a random matrix in `self`.

class simula.api.linalg.vector_space.**SubSpace** (*family=None*, *domain=None*, *name=''*)
Bases: *simula.api.linalg.vector_space.VectorSpace*, `abc.ABC`

Representation of a subspace

INPUT:

Parameters

- **family** – a set of vectors
- **domain** – a vector space (optional)
- **name** – the name of `self` (optional)

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : F = SubSpace({(1, 2, 0), (1, 1, 2)}, V)
Subspace of QQ^3 generated by the family {(1, 2, 0), (1, 1, 2)}
simula : F == V
False
simula : W = SubSpace({(1, 2, 0), (1, 1, 2), (1, 0, 0)}, V)
Subspace of QQ^3 generated by the family {(1, 0, 0), (1, 2, 0),␣
↪(1, 1, 2)}
simula : W == V
```

get_a_basis()
Returns a basis of `self`.

get_component_in_basis(*v*, *family=None*)
Returns the component of `v` (if it exists) for the `family`.

class `simula.api.linalg.vector_space.`**Vect**(*family=None*, *domain=None*, *transpose=False*)
Bases: *simula.api.linalg.vector_space.SubSpace*, `abc.ABC`

Representation of (an abstract) subspace.

```
simula : V = QQ^3; V
Vector Space of dimension 3 over Rational Numbers
simula : F = Vect({(1,0,0), (1, 1, 1)}, V); F
Vect({(1, 0, 0), (1, 1, 1)})
simula : W = Vect({(1, 0, 0), (1, 1, 1)}); W
Vect({(1, 0, 0), (1, 1, 1)})
simula : W.dimension()
2
```

class `simula.api.linalg.vector_space.`**VectorSpace**(*field*, *dim=1*)
Bases: `sympy.polys.domains.domain.Domain`, `simula.api.structure.simula_object.SimulaObject`, `abc.ABC`

are_linearly_dependent(*family*)
Tests if the vectors in `family` are linearly dependent in ``self.`

Parameters **family** (*Iterable*) – a set of vectors

Returns a boolean

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.are_linearly_dependent({(1, 1, 1), (2, 1, 0), (3,
↪ 2, 1)})
True
```

are_linearly_independent(*family*)
Tests if the vectors in `family` are linearly independent in ``self.`

Parameters **family** (*Union[Iterable, Sized]*) – a set of vectors

Returns a boolean

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
```

(continues on next page)

(continued from previous page)

```
simula : V.are_linearly_independent({(1, 1, 1), (2, 1, 0),␣
↪(3, 2, 1)})
False
```

canonical_basis()

Returns the canonical basis of `self`.

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.canonical_basis()
{(1, 0, 0), (0, 1, 0), (0, 0, 1)}
```

cardinality()

Returns the cardinality of `self`

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.cardinality()
oo
simula : V2 = VectorSpace(GF(5), 3); V2
Vector Space of dimension 3 over GF(5)
simula : V2.cardinality()
125
```

change_field(*field*)

Change the base field of `self`.

Parameters **field** – a field (QQ, RR, CC or GF(q))

Returns a vector space

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.change_field(RR)
Vector Space of dimension 3 over Real Numbers with 53 bits␣
↪of precision
```

contains(*family*)

Tests if `family` is included in ``self.`

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.include({(1, 1, 0), (2, 1, 0), (1/2, 0, 1)})
True
```

Parameters **family** (*Iterable*) –

dim()

The dimension of `self`.

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.dimension()
3
```

dimension()

The dimension of `self`.

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.dimension()
3
```

get_a_basis()

Returns a basis of `self`.

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.get_a_basis()
{(1, 0, 0), (0, 1, 0), (0, 0, 1)}
```

get_component_in_basis(*v*, *family=None*)

Returns the component of `v` (if it exists) for the `family`.

Parameters

- **v** – a vector
- **family** – a list vectors

Returns a tuple of scalars

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.get_component_in_basis((1,2,3), [(1, 1, -2), (2,␣
↪0, 1), (0, 0, 1)])
(2, -1/2, 15/2)
```

include(*family*)

Tests if `family` is included in ``self`.

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.include({(1, 1, 0), (2, 1, 0), (1/2, 0, 1)})
True
```

Parameters **family** (*Iterable*) –

is_basis (*family*)

Tests if family` is a basis of ``self.

Parameters **family** (*Union[Iterable, Sized]*) – a set of vectors

Returns a boolean

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.is_basis({(1, 1, 1), (2, 1, 0), (3, 2, 1)})
False
```

is_finite ()

Tests if self is finite.

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.is_finite()
False
simula : V2 = VectorSpace(GF(5), 3); V2
Vector Space of dimension 3 over GF(5)
simula : V2.is_finite()
True
```

is_generators (*family*)

Tests if family` generate ``self.

Parameters **family** – a set of vectors

Returns a boolean

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.is_generators({(0, 0, 1), (1, 1, 1), (1, 0, -1)})
True
```

is_infinite ()

Tests if self is infinite.

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.is_infinite()
True
simula : V2 = VectorSpace(GF(5), 3); V2
Vector Space of dimension 3 over GF(5)
simula : V2.is_infinite()
False
```

is_subspace (*domain*)
Tests if `domain` is a subspace of `self`.

Parameters **domain** – a vector space

Returns a boolean True or False

linear_combination (*family*, *coeffs*)
Returns a linear combination of `family` by the coefficients `coeffs`.

Parameters

- **family** – a list of vectors
- **coeffs** – a list of scalars

Returns a vector

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.linear_combination([(1, 1, -2), (2, 0, 1)], [-2,␣
↪3])
(4, -2, 7)
```

matrix_change_basis (*basis1*, *basis2*)
Returns subspace of `self` generated by `family`.

Parameters

- **basis1** (*Iterable*) – a basis of `self`
- **basis2** (*Iterable*) – a basis of `self`

Returns a matrix

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.matrix_change_basis([(1, 0, 0), (1, -1, 0), (0,␣
↪2, 2)], [(1,0,0), (0,1,0), (0, 0, 1)])
Matrix([
[1,  1,  -1],
```

(continues on next page)

(continued from previous page)

```
[0, -1,    1],
[0,   0, 1/2]])
```

subspace (*family*)

Returns subspace of `self` generated by `family`.

Parameters **family** (*Iterable*) – a set of vectors

Returns a vector space

EXAMPLES:

```
simula : V = VectorSpace(QQ, 3); V
Vector Space of dimension 3 over Rational Numbers
simula : V.subspace({(1, 1, -2), (2, 0, 1)})
Subspace of QQ^3 generated by the family {(1, 1, -2), (2, 0,
↪ 1)}
```

`simula.api.linalg.vector_space.`**gramSchmidt**(**args*, *orthonormal=False*)

GramShmidt orthogonalisation

class `simula.api.linalg.vector_space.`**vector**(*v*, ***, *column=False*)

Bases: `object`

Representation of a vector.

EXAMPLES:

```
simula : v = vector((1, -2, 3))
simula : v
(1, -2, 3)
simula : 5v
(5, -10, 15)
simula : v in QQ^3
True
simula : v.T
[ 1]
[-2]
[ 3]
simula : v * v.T
14
simula :  A = matrix([[1, 2, 0], [-1, 2, 3], [0, -1, 2]]); A
Matrix([
[ 1,  2, 0],
[-1,  2, 3],
[ 0, -1, 2]])
simula : v * A
(3, -5, 0)
simula : A * v.T
[-3]
[ 4]
```

(continues on next page)

(continued from previous page)

```
[ 8]
simula : w = vector((2, 2, 5), column=True); w
[2]
[2]
[5]
simula : v - 3w.T
(-5, -8, -12)
```

5.2.3 Linear Maps

Operations over linear maps

class simula.api.linalg.linear_map.**LinearMap**(*symbols=None*, *expr=None*, *, *domain=None*, *codomain=None*, *matrix=None*, *basis1=None*, *basis2=None*, *check=True*)

Bases: *simula.api.calculus.functions.Function*

Representation of linear maps.

Parameters

- **symbols** – a tuple of symbols
- **expr** – an expression or a tuple of expression
- **domain** – (optional) a field or a vector space
- **codomain** – (optional) a field or a vector space
- **matrix** – (optional) a matrix
- **basis1** – a basis of the domain
- **basis2** – a basis of the codomain
- **check** – a boolean

One can define a linear map by two methods

1) First method : you know the expression of the linear map

```
simula : (x, y, z) = var("x,y,z")
simula : f = linear_map((x, y, z), (2x-y-z, x-y, -x+z)); f
Linear map from QQ^3 --> QQ^3 defined by (x, y, z) |--> (2x - y␣
↪- z, x - y, -x + z)
simula : f(1, 2, -3)
(3, -1, -4)
```

(continues on next page)

(continued from previous page)

```
simula : f.kernel()
Subspace of QQ^3 generated by the family {(1, 1, 1)}
simula : f.image()
Subspace of QQ^3 generated by the family {(2, 1, -1), (-1, -1,␣
↪0)}
simula : f.is_diagonalizable()
True
simula : f.eigenvals()
{1 - sqrt(2): 1, 1 + sqrt(2): 1, 0: 1}
simula : f.is_one_to_one()
False
```

2) **Second method** [you know the matrix associated to the linear map in some bases (if no basis] is specified they are equal to canonical bases).

```
simula : A = matrix([[-1, 1, 1], [1, -1, 1], [1, 1, -1]]); A
Matrix([
[-1,  1,  1],
[ 1, -1,  1],
[ 1,  1, -1]])
simula : g = linear_map(matrix=A, domain=RR^3, codomain=RR^3); g
g = linear_map(matrix=A, domain=RR^3, codomain=RR^3); g
Linear map from RR^3 --> RR^3 defined by (x1, x2, x3) |--> (-x1␣
↪+ x2 + x3, x1 - x2 + x3, x1 + x2 - x3)
simula : g(0, 1, 1)
(2, 0, 0)
simula : g.spectrum()
{1, -2}
simula : g.image()
Subspace of RR^3 generated by the family {(1, 1, -1), (1, -1,␣
↪1), (-1, 1, 1)}
simula : g.is_endomorphism()
True
simula : g.is_surjective()
True
```

det()
Returns the determinant of the linear map.

eigenvals()
Returns the eigen values of `self`.

eigenvects()
Returns the eigen vectors of `self`.

get_matrix(*basis1=None*, *basis2=None*)
Return the matrix associated to the linear map with respect to `basis1` and `basis2`.

im()

Returns the image of `self`.

image()
Returns the image of `self`.

is_diagonalizable()
Tests if `self` is diagonalizable.

is_endomorphism()
Tests if `self` is an endomorphism.

is_idempotent()
Tests if `self` is idempotent.

is_injective()
Tests if `self` is an one-to-one.

is_isomorphism()
Tests if `self` is an isomorphism.

is_nilpotent()
Tests if `self` is nilpotent.

is_one_to_one()
Tests if `self` is an one-to-one.

is_surjective()
Tests if `self` is an surjective.

is_zero()
Tests if `self` is null.

ker()
Returns the kernel of `self`.

kernel()
Returns the kernel of `self`.

nullity()
Returns the nullity of the kernel of the linear map.

rank()
Returns the rank of the linear map.

spectrum()
Returns the spectrum of `self` i.e the set of eigen values of `self`.

trace()
Returns the trace of the linear map.

simula.api.linalg.linear_map.**image**(*expr*)
Image of a linear map or a matrix.

simula.api.linalg.linear_map.**ker**(*expr*)
Kernel of a linear map or a matrix.

`simula.api.linalg.linear_map.`**`kernel`**`(`*expr*`)`
Kernel of a linear map or a matrix.

`simula.api.linalg.linear_map.`**`linear_map`**
alias of *`simula.api.linalg.linear_map.LinearMap`*

`simula.api.linalg.linear_map.`**`linear_transformation`**
alias of *`simula.api.linalg.linear_map.LinearMap`*

5.3 Number Theory

5.3.1 General Functions

`simula.api.ntheory.functions.`**`Abs`**`(`*f*`)`
Returns the absolute value of `f` i. e |f|.

EXAMPLES:

```
simula :  Abs(2-pi)
-2 + pi
simula : Abs(x)
Abs(x)
```

`class` `simula.api.ntheory.functions.`**`Integer`**`(`*i*`)`

`class` `simula.api.ntheory.functions.`**`IntegerFactorization`**`(`*i*`)`
Representation of the prime decomposition of an integer.

`simula.api.ntheory.functions.`**`Max`**`(`**args*`)`
Returns the maximum of `args`.

EXAMPLES:

```
simula : Max(4, 8, 9, 5)
9
simula : Max([4, 8, 9, 5, 20])
20
```

`simula.api.ntheory.functions.`**`Min`**`(`**args*`)`
Returns the minimun of `args`.

EXAMPLES:

```
simula : Min(4, 8, 9, 5)
4
simula : Min([4, 8, 9, 2, 5, 20])
2
```

`simula.api.ntheory.functions.`**`Mod`**`(`*g, f*`)`
Represents a modulo operation on symbolic expressions i. e `g` modulo `f`.

`simula.api.ntheory.functions.`**`N`**`(`*x*, ***, *precision=15*`)`

Returns a numerical approximation of `x` for a given precision `precision` (default : 15).

EXAMPLES :

```
simula : numerical_approx(pi)
3.141592653589793
simula : numerical_approx(pi, precision=20)
3.1415926535897932385
```

`simula.api.ntheory.functions.`**`beta`**`(`*a*, *b*`)`

Returns Beta(a, b).

EXAMPLES:

```
simula :  beta(3, 1)
1/3
simula : numerical_approx(beta(2, 2))
0.1666666666666667
```

`simula.api.ntheory.functions.`**`binomial`**`(`*n*, *k*`)`

Returns the binomial coefficient $\dfrac{n!}{(n-k)!\times k!}$.

`simula.api.ntheory.functions.`**`ceil`**`(`*a*`)`

Returns the smallest integer value not less than `a`.

`simula.api.ntheory.functions.`**`denominator`**`(`*expr*`)`

Returns the denominator of `expr`.

EXAMPLES:

```
simula : f = 2/(sqrt(3)-1); f
2
simula : denominator(f)
-1 + sqrt(3)
simula : denominator(3pi/5)
5
```

`simula.api.ntheory.functions.`**`ellipsis_range`**`(`*a*, *b*, *c=None*`)`

Returns `[a, b, a + (b-a), ..., c]` if `c` is not `None` otherwise `[a, a+1, ..., b]`.

EXAMPLES:

```
simula : ellipsis_range(2, 5)
[2, 3, 4, 5]
simula : ellipsis_range(2, 3, 10)
[2, 3, 4, 5, 6, 7, 8, 9, 10]
simula : ellipsis_range(1, 1.5, 5)
[1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]
```

simula.api.ntheory.functions.**euler_phi**(*n*)

Returns the image of the Euler totient function at *n*.

EXAMPLES:

```
simula : euler_phi(3)
2
simula : euler_phi(10)
4
simula : euler_phi(15)
8
```

simula.api.ntheory.functions.**evalf**(*x*, ***, *precision=15*)

Returns a numerical approximation of `x` for a given precision `precision` (default : 15).

EXAMPLES :

```
simula : numerical_approx(pi)
3.141592653589793
simula : numerical_approx(pi, precision=20)
3.14159265358979323846
```

simula.api.ntheory.functions.**factorial**(*n*)

Returns the factorial of `n`.

Parameters **n** – a nonnegative integer

EXAMPLES:

```
simula : factorial(4)
24
simula : factorial(5)
120
```

simula.api.ntheory.functions.**floor**(*a*)

Returns the largest integer value not greater than `a`.

simula.api.ntheory.functions.**fraction**(*a*, *b=1*)

Returns `a/b`.

EXAMPLES:

```
simula : fraction(3,5)
3/5
```

simula.api.ntheory.functions.**gamma**(*n*)

Returns gamma(n).

EXAMPLES:

```
simula : gamma(2)
1
```

(continues on next page)

(continued from previous page)

```
simula : gamma(6)
120
simula : gamma(2.5)
1.32934038817914
```

simula.api.ntheory.functions.**integer_decomposition**(*n*)

Returns the prime decomposition of the integer `n`.

simula.api.ntheory.functions.**inverse_mod**(*a, n*)

Returns the inverse of `a` mod `n` provided that `a` is prime with `n`.

EXAMPLES:

```
simula : inverse_mod(2, 5)
3
simula : inverse_mod(4, 7)
2
```

simula.api.ntheory.functions.**is_prime**(*n*)

Tests if `n` is a prime number.

Parameters **n** – a nonnegative integer

EXAMPLES:

```
simula : is_prime(5)
True
simula : is_prime(201)
False
```

simula.api.ntheory.functions.**is_primitive_root**(*a, p*)

Tests if `a` is a primitive root of `p`.

Parameters

- **a** – an integer
- **p** – a prime number.

EXAMPLES:

```
simula : is_primitive_root(2, 5)
True
simula : is_primitive_root(3, 11)
False
```

simula.api.ntheory.functions.**is_quad_residue**(*a, p*)

Tests if `a` modulo `p` is in the set of squares mod `` p` ``.

Parameters

- **a** – an integer
- **p** – a prime number.

EXAMPLES:

```
simula : is_quad_residue(2, 5)
False
simula : is_quad_residue(3, 11)
True
```

simula.api.ntheory.functions.**jacobi_symbol**(*m*, *n*)

Returns the Jacobi symbol `(m / n)`.

Parameters

- **m** – an integer
- **n** – an odd positive integer

EXAMPLES:

```
simula : jacobi_symbol(7, 15)
-1
simula : jacobi_symbol(2, 33)
1
```

simula.api.ntheory.functions.**legendre_symbol**(*a*, *p*)

Returns the Legendre symbol `(m / n)`.

Parameters

- **m** – an integer
- **n** – an odd prime number

EXAMPLES:

```
simula : legendre_symbol(7, 11)
-1
simula : legendre_symbol(3, 37)
1
```

simula.api.ntheory.functions.**list_divisors**(*n*, *, *proper=False*)

Return all divisors of `n` sorted from `1` to `n`.

Parameters

- **n** – an integer
- **proper** – (a boolean, default `False`) specify if `n` is included to the list o divisors or not.

EXAMPLES:

```
simula : list_divisors(10)
[1, 2, 5, 10]
simula : list_divisors(10, proper=True)
[1, 2, 5]
```

`simula.api.ntheory.functions.`**`loggamma`**`(`*n*`)`

Returns log(gamma(n)).

EXAMPLES:

```
simula : loggamma(1)
0
simula : loggamma(2)
log(2)
```

`simula.api.ntheory.functions.`**`mobius`**`(`*n*`)`

Returns `mobius(n)` which maps natural number to {-1, 0, 1}.

Parameters **n** – a positive integer

EXAMPLES:

```
simula : mobius(2)
-1
simula : mobius(15)
1
```

`simula.api.ntheory.functions.`**`multiplicity`**`(`*m*, *n*`)`

Returns the greatest integer `k` such that `m^k` divides `n`.

Parameters

- **m** – an integer
- **n** – an integer

EXAMPLES:

```
simula : multiplicity(10, 100)
2
simula : multiplicity(3, 36)
2
```

`simula.api.ntheory.functions.`**`n`**`(`*x*, *, *precision=15*`)`

Returns a numerical approximation of `x` for a given precision `precision` (default : 15).

EXAMPLES :

```
simula : numerical_approx(pi)
3.141592653589793
simula : numerical_approx(pi, precision=20)
3.1415926535897932385
```

`simula.api.ntheory.functions.`**`nAk`**`(`*n*, *k*`)`

Returns the `k`-arrangement in `n` i.e. $\frac{n!}{(n-k)!}$.

`simula.api.ntheory.functions.`**`nCk`**(*n*, *k*)

Returns the binomial coefficient $\frac{n!}{(n-k)! \times k!}$.

`simula.api.ntheory.functions.`**`next_prime`**(*n*)

Returns the i-th prime number greater than `n`.

Parameters **n** – an integer

EXAMPLES:

```
simula : next_prime(4)
5
simula : next_prime(23)
29
```

`simula.api.ntheory.functions.`**`nthroot_mod`**(*a*, *n*, *p*, ***, *all_roots=False*)

Returns the solutions to `x^n = a mod p`.

Parameters

- **a** – an integer
- **n** – a positive integer
- **p** – a positive integer
- **all_roots** – (default `False`) if `False` returns the smallest root, else the list of roots

EXAMPLES:

```
simula : nthroot_mod(1, 2, 7)
1
simula : nthroot_mod(1, 2, 7, all_roots=True)
[1, 6]
```

`simula.api.ntheory.functions.`**`number_divisors`**(*n*, ***, *proper=False*)

Return the number of divisors of `n`.

Parameters

- **n** – an integer
- **proper** – (default `False`) If `True` then the divisor of `n` will not be counted

EXAMPLES:

```
simula : number_divisors(10)
4
simula : number_divisors(10, proper=True)
3
```

`simula.api.ntheory.functions.`**`numerator`**(*expr*)

Returns the numerator of `expr`.

EXAMPLES:

```
simula : f = 2/(sqrt(3)-1); f
2/(-1 + sqrt(3))
simula : numerator(f)
2
simula : numerator(3pi/5)
3pi
```

`simula.api.ntheory.functions.`**`numerical_approx`**(*x*, ***, *precision=15*)

Returns a numerical approximation of `x` for a given precision `precision` (default : 15).

EXAMPLES :

```
simula : numerical_approx(pi)
3.141592653589793
simula : numerical_approx(pi, precision=20)
3.14159265358979323846
```

`simula.api.ntheory.functions.`**`order_modulo`**(*a*, *n*)

Returns the order of `a` modulo `n`.

Parameters

- **a** – an integer
- **n** – an integer relatively prime to `a`

EXAMPLES:

```
simula : order_modulo(2, 9)
6
simula : order_modulo(3, 11)
5
```

`simula.api.ntheory.functions.`**`perfect_power`**(*n*)

Returns `(a, e)` such that $n = a^e$ if `n` is a perfect power with `e > 1`, else `False`.

Parameters **n** – an integer

EXAMPLES:

```
simula : perfect_power(6)
False
simula : perfect_power(100)
(10, 2)
simula : perfect_power(16807)
(7, 5)
```

`simula.api.ntheory.functions.`**`power_mod`**(*x*, *a*, *n*)

Returns the power `x^a mod n`.

EXAMPLES:

```
simula : power_mod(100, 10000, 11)
1
simula : power_mod(99, 99876655, 13)
5
```

simula.api.ntheory.functions.**previous_prime**(*n*)

Returns the i-th prime number less than `n`.

Parameters **n** – an integer

EXAMPLES:

```
simula : previous_prime(4)
3
simula : previous_prime(23)
19
```

simula.api.ntheory.functions.**prime_factors**(*n*)

Returns a sorted list of `n`'s prime factors, ignoring multiplicity.

Parameters **n** – an integer

EXAMPLES:

```
simula : prime_factors(6)
[2, 3]
simula : prime_factors(20)
[2, 5]
```

simula.api.ntheory.functions.**prime_pi**(*n*)

Returns `pi(n)` the number of prime numbers less than or equal to `n`.

Parameters **n** – an integer

EXAMPLES:

```
simula : prime_pi(4)
2
simula : prime_pi(20)
8
```

simula.api.ntheory.functions.**prime_position**(*n*)

Returns the n-th prime number.

Parameters **n** – a positive integer

EXAMPLES:

```
simula : prime_position(1)
2
simula : prime_position(2)
3
```

(continues on next page)

(continued from previous page)

```
simula : prime_position(10)
29
```

simula.api.ntheory.functions.**prime_range**(*a*, *b*)
Returns the list of primes between a and b.

simula.api.ntheory.functions.**primes**(*start*, *end*)
Generators of primes between start and end (both included).

EXAMPLES:

```
simula : 11 in primes(10, 40)
True
simula : for i in primes(10, 40):
.......:     print(i)
11
13
17
19
23
29
31
37
```

simula.api.ntheory.functions.**primitive_root**(*n*)
Returns the smallest primitive root modulo n or Raise an exception valueError.

EXAMPLES:

```
simula : primitive_root(7)
3
simula : primitive_root(10)
3
simula : primitive_root(29)
2
```

simula.api.ntheory.functions.**primitive_root_mod**(*n*)
Returns the smallest primitive root or None.

Parameters n – a positive integer

EXAMPLES:

```
simula : primitive_root_mod(4)
3
simula : primitive_root_mod(20)
simula : primitive_root_mod(19)
2
```

simula.api.ntheory.functions.**quadratic_residues**(*n*)
Returns the set of quadratic residues mod n.

Parameters **n** – a positive integer

EXAMPLES:

```
simula : quadratic_residues(4)
{0, 1}
simula : quadratic_residues(20)
{0, 1, 4, 5, 9, 16}
```

`simula.api.ntheory.functions.`**`randint`**(*a*, *b=None*)

Returns a random integer between `a` and `b` if `b` is not `None` otherwise between `` 0 and ``a`.

Parameters **b** (*Optional[int]*) –

`simula.api.ntheory.functions.`**`random_prime`**(*a*, *b=None*)

Returns randomly a prime number between `a` and `b`.

Parameters **b** (*Optional[int]*) –

`simula.api.ntheory.functions.`**`random_prime_size`**(*size*)

Returns randomly a prime of size `size` bits.

Parameters **size** (*int*) –

`simula.api.ntheory.functions.`**`rationalize_denominator`**(*expr*)

Rationalizes the denominator of `expr`.

EXAMPLES:

```
simula : f = 2/(sqrt(3)-1); f
2/(-1 + sqrt(3))
simula : rationalize_denominator(f)
1 + sqrt(3)
simula : rationalize_denominator(1/(5-sqrt(11)))
(sqrt(11) + 5)/14
```

Parameters **expr** (*sympy.core.expr.Expr*) –

`simula.api.ntheory.functions.`**`sign`**(*x*)

Returns the sign of `x`.

EXAMPLES:

```
simula :  sign(2-pi)
-1
simula : sign(2-sqrt(3))
1
```

`simula.api.ntheory.functions.`**`sqrt_mod`**(*a*, *n*, *all_roots=False*)

Returns a root of $x^2 = a \mod n$ or `None`.

Parameters

- **a** – an integer

- **n** – a positive integer
- **all_roots** – (default `False`) if `True` the list of roots is returned or `None`

EXAMPLES:

```
simula : sqrt_mod(4, 7)
2
simula : sqrt_mod(4, 7, all_roots=True)
[2, 5]
simula : sqrt_mod(5, 11, all_roots=True)
[4, 7]
simula : sqrt_mod(5, 10, all_roots=True)
[5]
```

`simula.api.ntheory.functions.`**`srange`**`(`*start*, *stop=None*, *step=1*`)`

Returns all integers from `start` to `stop` provided that `stop` is not `None` otherwise returns all integers from `0` to `start` for a given *step*` (default is `1`).

EXAMPLES:

```
simula : srange(2, 9)
[2, 3, 4, 5, 6, 7, 8]
simula : srange(9, 0, -1)
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

5.3.2 Complex Numbers

Functions acting on Complex numbers

`simula.api.ntheory.complexe.`**`argument`**`(`*z*`)`

return the argument of `z`.

Parameters **z** – a complex number

EXAMPLES:

```
simula : argument(1-I)
-pi/4
simula : argument(I)
pi/2
```

`simula.api.ntheory.complexe.`**`complex_alg_form`**`(`*z*`)`

return the algebraic form of `z`.

Parameters **z** – a complex number

EXAMPLES:

```
simula : z = 1/(1 - I); z
(1 + I)/2
```

(continues on next page)

(continued from previous page)

```
simula : complex_alg_form(z)
1/2 + I/2
```

simula.api.ntheory.complexe.**complex_exp_form**(*z*)
return the exponential form of `z`.

Parameters **z** – a complex number

EXAMPLES:

```
simula : complex_exp_form(1 - I)
'sqrt(2)*exp(-I*pi/4)'
simula : complex_exp_form(-I)
'exp(-I*pi/2)'
```

simula.api.ntheory.complexe.**complex_trig_form**(*z*)
return the trigonometric form of `z`.

Parameters **z** – a complex number

EXAMPLES:

```
simula : complex_trig_form(1 - I)
'sqrt(2)(cos(-pi/4)+I*sin(-pi/4))'
simula : complex_trig_form(-I)
'cos(-pi/2)+I*sin(-pi/2)'
```

simula.api.ntheory.complexe.**complexe**(*a*, *b=0*)
return the complex number `a + bI`.

Parameters

- **a** (*int*) – an real number
- **b** – an real number

EXAMPLES:

```
simula : complex(2, 6)
2 + 6I
simula : complex(1, 1)
1 + I
```

simula.api.ntheory.complexe.**conjugate**(*z*)
return the conjugate of `z`.

Parameters **z** – a complex number

EXAMPLES:

```
simula : conjugate(1-I)
1 + I
simula : conjugate(1-I)(6I)
-6I
```

simula.api.ntheory.complexe.**im_part**(*z*)

return the imaginary part of z.

Parameters **z** – a complex number

EXAMPLES:

```
simula : im_part(1-I)
-1
simula : im_part(6I)
6
```

simula.api.ntheory.complexe.**imag_part**(*z*)

return the imaginary part of z.

Parameters **z** – a complex number

EXAMPLES:

```
simula : im_part(1-I)
-1
simula : im_part(6I)
6
```

simula.api.ntheory.complexe.**module**(*z*)

return the module of z i.e. |z|.

Parameters **z** – a complex number

EXAMPLES:

```
simula : module(1-I)
sqrt(2)
simula : module(-I)
1
```

simula.api.ntheory.complexe.**real_part**(*z*)

return the real part of z.

Parameters **z** – a complex number

EXAMPLES:

```
simula : real_part(1-I)
1
simula : real_part(6I)
0
```

5.4 Calculus

5.4.1 General Functions

Calculus functions.

class `simula.api.calculus.functions.`**`Function`**(*var*, *expression=None*)

Definition of a mathematical function.

EXAMPLES:

```
simula : x = var('x')
simula : f = function(x, x^2-2x+1); f
Function defined by x |--> x^2 - 2x + 1
simula : f(2x)
4x^2 - 4x + 1
simula : 3f
Function defined by x |--> 3x^2 - 6x + 3
simula : factor(f(x))
(x - 1)^2
simula : g = function("g"); g
X |--> g(X)
# We can simplify the definition of a function
simula : x, y = var('x, y')
simula : h(x, y) = x^2-y^2-x*y-2; h
Function defined by (x, y) |--> x^2 - x*y - y^2 - 2
simula : h(1,-2)
-3
simula : h(10, 0)
98
simula : diff(h(x, y), x)
2x - y
```

`as_expr`()

Returns the expression of `self`.

`compose`(*g*, **args*)

Composition function of f and g i.e x –> f o g (x)

Parameters **g** – Function

`critical_points`(*field=RR*)

Returns the critical points of `self`.

`critical_points_ambigus`(*field=RR*)

Returns the critical points ambigus of `self`.

`gradient`()

Returns the gradient function of `self`.

`hessian`()

Returns the hessian function of `self`.

hessian_matrix()
Returns the hessian matrix of `self`.

jacobian()
Returns the jacobian function of `self`.

jacobian_matrix()
Returns the jacobian matrix of `self`.

local_extrema(*field=RR*)
Returns the local extrema of `self`.

local_maxima(*field=RR*)
Returns the local maxima of `self`.

local_minima(*field=RR*)
Returns the local minima of `self`.

saddle_points(*field=RR*)
Returns the saddle points of `self`.

class simula.api.calculus.functions.**FunctionPiecewise**(*var*, **expr*)

Representation of a piecewise function.

EXAMPLES:

```
simula : x = var("x")
simula : f = FunctionPiecewise(x, (2x-1, x<0), (3x, True)); f
x |--> 2x - 1 if x < 0  and  3x if True
simula : f(1)
3
simula : f(3)
27
simula : f(-2)
-5
simula : n = var('n', "NN")
simula : f(n)
3n
simula : y = var('y', "ZZ*-")
simula : f(y)
2y - 1
```

class simula.api.calculus.functions.**O**(*expr*, **args*, ***kwargs*)
Big O notation.

EXAMPLES:

```
simula : O(x + x^2)
O(x)
simula : O(x^3-x^2)
O(x^2)
simula : O(5x^5)
O(x^5)
```

(continues on next page)

(continued from previous page)

```
simula : O(x^5-x-4)
O(1)
```

`simula.api.calculus.functions.`**`canonical_form`**(*expr*, *var=None*)
Returns the canonical form of a second degree equation.

`simula.api.calculus.functions.`**`coefficients`**(*expr*, **args*)
Returns the coefficients of a polynomial.

`simula.api.calculus.functions.`**`cyclotomic_polynomial`**(*n*, *var=None*)
Returns the cyclotomic polynomial of order `n`.

Parameters

- **n** – an integer
- **var** – (optional) a variable

`simula.api.calculus.functions.`**`degree`**(*expr*, **args*)
Returns the degree of a polynomial.

`simula.api.calculus.functions.`**`derivative`**(*f*, *var=None*, **args*)
Returns the derivative of `f` with respect to `var`.

`simula.api.calculus.functions.`**`derivative_number`**(*func*, *var*, *point=None*)
Returns the derivative number of `func` at point `point`.

`simula.api.calculus.functions.`**`diff`**(*f*, *var*, **args*, ***kwargs*)
Returns the differentiation of `f` with respect to symbols.

`simula.api.calculus.functions.`**`discriminant`**(*f*, **args*, ***kwargs*)
Returns the discriminant of``f``.

`simula.api.calculus.functions.`**`div`**(*f*, *g*, **args*, ***kwargs*)
Returns the quotient and remainder of the division of `f` by `g`.

`simula.api.calculus.functions.`**`exp`**(*x*)
Returns the exponential of `x` i.e. e^x.

`simula.api.calculus.functions.`**`expand`**(*f*, ***kwargs*)
Returns the expansion of `f`.

`simula.api.calculus.functions.`**`expand_trig`**(*f*, **args*)
Returns the trigonometric expansion of `f`.

`simula.api.calculus.functions.`**`factor`**(*expr*, **args*, ***kwargs*)
Factors an expression `f`.

`simula.api.calculus.functions.`**`function`**
alias of *`simula.api.calculus.functions.Function`*

`simula.api.calculus.functions.`**`function_composition`**(*f*, *g*, **args*)
Composite function of f and g i.e x—-> f o g.

Parameters

- **f** (simula.api.calculus.functions.Function) – Function
- **g** (*(<class 'simula.api.calculus.functions.Function'>, <class 'sympy.core.expr.Expr'>)*) – Function

simula.api.calculus.functions.**gcd**(*f*, *g*, ***kwargs*)
Returns the gcd of `f` and `g`.

simula.api.calculus.functions.**gcdex**(*f*, *g*, ***kwargs*)
Extended Euclidean algorithm of `f` and `g`.

simula.api.calculus.functions.**homogenize**(*f*, *var*, *, *symbols=None*)
Returns the homogenization of `f` with respect to `var`.

simula.api.calculus.functions.**inflection_point**(*func*, *var=None*)
Returns Points d'inflexion de la fonction "func"

simula.api.calculus.functions.**integrate**(*f*, **symbols*, ***kwargs*)
Integrates `f` with respect to `symbols`.

simula.api.calculus.functions.**lcm**(*f*, *g*, ***kwargs*)
Returns the lcm of `f` and `g`.

simula.api.calculus.functions.**limit**(*f*, *x*, *x0*, *, *dir='+-'*)
Returns the limit of `f` when `x --> x0`.

simula.api.calculus.functions.**limit_left**(*f*, *x*, *x0*)
Returns the limit from the left of `f` when `x --> x0`.

simula.api.calculus.functions.**limit_right**(*f*, *x*, *x0*)
Returns the limit from the right of `f` when `x --> x0`.

simula.api.calculus.functions.**ln**(*a*, **args*)
Returns the natural logarithm $\ln(a)$ or $\log(a)$.

simula.api.calculus.functions.**log**(*a*, **args*)
Returns the natural logarithm $\ln(a)$ or $\log(a)$.

simula.api.calculus.functions.**logb**(*a*, *b*)
Returns the logarithm of `a` in base `b`.

simula.api.calculus.functions.**partial**(*f*, *var*, **args*, ***kwargs*)
Returns the differentiation of `f` with respect to symbols.

simula.api.calculus.functions.**poly**(*expr*, **args*, ***kwargs*)
Returns `expr` as a polynomial.

simula.api.calculus.functions.**primitive**(*f*, *var=None*)
Returns the primitive of `f`.

simula.api.calculus.functions.**product**(*expr*, **symbols*, ***kwargs*)
Computes the product of `expr` with respect to `symbols`.

`simula.api.calculus.functions.`**`real_roots`**(*func*, *var=None*)
Returns the real roots of `func`.

`simula.api.calculus.functions.`**`roots`**(*f*, **args*, ***kwargs*)
Returns the complex roots of `f`.

`simula.api.calculus.functions.`**`series`**(*f*, *x=None*, *x0=0*, *n=6*, ***, *dir='+'*)
Returns the series expansion of `f` of order `n` around point `x = x0`.

`simula.api.calculus.functions.`**`simplify`**(*f*, ***kwargs*)
Reduces `f`.

`simula.api.calculus.functions.`**`sqrt`**(*expr*)
Returns the square root of `f`.

`simula.api.calculus.functions.`**`summation`**(*expr*, **symbols*, ***kwargs*)
Computes the summation of `expr` with respect to `symbols`.

`simula.api.calculus.functions.`**`taylor_polynomial`**(*f*, *x=None*, *x0=0*, *n=6*, ***, *dir='+'*)
Returns the taylor polynomial of `f` of order `n` around point $x = x0$.

`simula.api.calculus.functions.`**`trigsimp`**(*f*, ***kwargs*)
Reduces the trigonometric expression `f`.

`simula.api.calculus.functions.`**`trunc`**(*f*, *n*, **gens*, ***args*)
Reduces `f` modulo a constant `n`.

5.4.2 Sequences

Representation of sequences :

- Sequence
- ArithmeticSequence
- GeometricSequence
- ArithmeticoGeometricSequence

class `simula.api.calculus.sequense.`**`ArithmeticGeometricSequence`**(*a*, *b*, ***, *ics=None*)
Representation of a arithmetic-geometric sequence `U(n+1) = aU(n) + b`.

EXAMPLES:

```
simula :  n = var('n')
simula : U = ArithmeticGeometricSequence(1, 3, ics=(1, 5)); U
n |--> 3n + 2
simula : U(n+1) == U(n) + 3
True
```

(continues on next page)

(continued from previous page)

```
simula : U(1)
5
```

class simula.api.calculus.sequense.**ArithmeticSequence**(*terms=None*, *, *r=None*)

Representation of an arithmetic sequence `U(n+1) = U(n) + r`.

EXAMPLES:

```
simula :  n = var('n')
simula : U = ArithmeticSequence((0, 1), r=5)
simula : U
n |--> 5n + 1
simula : U(n+1) - U(n)
5
simula : V = ArithmeticSequence(((0, 1), (2, 10))); V
n |--> 9n/2 + 1
simula : V(2)
10
```

class simula.api.calculus.sequense.**GeometricSequence**(*terms=None*, *q=None*)

Representation of a geometric sequence `U(n+1) = qU(n)`.

EXAMPLES:

```
simula :  n = var('n')
simula : U = GeometricSequence((1, 3), q=5);
n |--> 3*5^n/5
simula : simplify(U(n+1) / U(n))
5
simula : V = GeometricSequence(((1, 3), (2, 10))); V
n |--> 10^(n - 1)*3^(2 - n)
simula : simplify(V(n+1) / V(n))
10/3
```

class simula.api.calculus.sequense.**Sequence**(*var*, *expression=None*)

Representation of a sequence.

EXAMPLES:

```
simula :  n = var('n')
simula : Un = sequence(n, n^3-3n-1)
simula : Un
n |--> n^3 - 3n - 1
simula : Un(1), Un(2)
(-3, 1)
simula : Un.is_convergente()
False
```

(continues on next page)

(continued from previous page)

```
simula : Un.limit()
oo
```

is_convergente()

Tests if `self` is convergent.

EXAMPLES:

```
simula :  n = var('n')
simula : Vn = sequence(n, (n-1)/(2n^2-1))
simula : Vn
n |--> (n - 1)/(2n^2 - 1)
simula : Vn.is_convergente()
True
```

limit()

Returns the limit of `self`.

EXAMPLES:

```
simula :  n = var('n')
simula : Vn = sequence(n, (n-1)/(2n^2-1))
simula : Vn
n |--> (n - 1)/(2n^2 - 1)
simula : Un.limit()
oo
```

`simula.api.calculus.sequense.`**limit_sequence**(*Un*, **args*)

Returns the limit of `Un` at infinity.

`simula.api.calculus.sequense.`**sequence**

alias of *simula.api.calculus.sequense.Sequence*

5.5 Finite Fields

`simula.api.finite_field.finite_field.`**GF**

alias of *simula.api.finite_field.finite_field.FiniteField*

class `simula.api.finite_field.finite_field.`**FiniteField**(*q*, *gen=None*, *ideal=None*, ***kwargs*)

Finite field operations.

Parameters

- **q** – a prime power p^n
- **gen** – (optional) a generator of `self`.

- **ideal** – (optional) an irreducible polynomial which is used to construct `self`
- **kwargs** –

EXAMPLES:

```
simula : G.<a> = GF(9); G
Finite Field of 9 elements defined by the quotient of F_3[a] by␣
↪the ideal <a^2 + 2a + 2>
simula : {0, 1, 2, a + 1, 2a, a, 2a + 2, a + 2, 2a + 1}
simula : G(a^3-a^2-a+2)
2
simula : f = G(a+1); f
a + 1
simula : f^-1
2a + 2
simula : (2a + 2) * f
1
simula : a*f
2a + 1
```

cardinality()
Returns the cardinality of `self`.

characteristic()
Returns the characteristic of `self`.

exponential(*prim_elt=None*)
Returns an exponential representation of `self`.

exquo(*poly1*, *poly2*)
Exact quotient of `poly1` and `poly2`

from_ComplexField(*a*, *K0*)
Convert a complex element to `dtype`.

from_FF_gmpy(*a*, *K0=None*)
Convert `ModularInteger(mpz)` to `dtype`.

from_FF_python(*a*, *K0=None*)
Convert `ModularInteger(int)` to `dtype`.

from_QQ_gmpy(*a*, *K0=None*)
Convert GMPY's `mpq` to `dtype`.

from_QQ_python(*a*, *K0=None*)
Convert Python's `Fraction` to `dtype`.

from_Rational(*a*, *K0=None*)
Convert Python's `Fraction` to `dtype`.

from_RealField(*a*, *K0*)
Convert mpmath's `mpf` to `dtype`.

from_ZZ_gmpy (*a*, *K0=None*)
Convert GMPY's `mpz` to `dtype`.

from_ZZ_python (*a*, *K0=None*)
Convert Python's `int` to `dtype`.

from_sympy (*a*)
Convert SymPy's Element to GF element

gen ()
Returns the generator of `self`.

get_elements ()
Returns all elements of `self`.

get_field ()
Returns a field associated with `self`.

get_prime_field ()
Returns all elements of the prime sub-field of `self`.

get_primitive_element ()
Get a primitive element.

inv (*pol*)
Returns the inverse of `pol` if it exists.

inverse (*pol*)
Returns the inverse of `pol` if it exists.

is_nth_power (*pol*, *n*)
Tests if `pol` is a power of `n` in `self`.

is_prime_field ()
Tests if `self` is a prime field.

is_square (*pol*)
Tests if `pol` is a square in `self`.

modulus ()
Returns the modulus polynomial of `self`.

objgen ()
Returns `self` and the generator of `self`.

order (*pol=None*)
Returns the order of `self` if `pol=None` otherwise returns the order of `pol`.

prime_subfield ()
Returns a field associated with `self`.

primitive_elements ()
Returns all primitive elements of `self`.

quadratic_character (*pol*)
Returns $\chi(pol)$ which is equal to `1` if `pol` is a nonzero square in `self`, `-1` if `pol` is not a square in `self` and `0` otherwise.

quo (*poly1*, *poly2*)
Quotient of `poly1` and `poly2`

random_element ()
Returns a random element in `self`.

sqrt (*a*)
Returns the square root of `a` if it exists.

sub_group_generatedby (*pol*)
Returns the sub group generated by `self`.

to_sympy (*elt*)
Convert `a` to a sympy object.

5.6 Statistics

5.6.1 Statistical Series

Statistic characteristics of Series

class simula.api.stats.series.**StatisticsSeries** (*series=None*)
Statistic functions of series.

classmethod all_deciles (*series=None*)
Returns the deciles of a statistic series `series`.

classmethod arithmetic_mean (*series=None*)
Arithmetic mean of a statistic series `series`.

classmethod coefficient_of_dispersion (*series=None*)
Coefficient of dispersion of a statistic series `series`.

classmethod coefficient_of_variation (*series=None*)
Coefficient of variation of a statistic series `series`.

classmethod deciles (*series=None*)
Returns the deciles `d1` and `d2` of a statistic series `series`.

classmethod geometric_mean (*series=None*)
Geometric mean of a statistic series `series`.

classmethod harmonic_mean (*series=None*)
Harmonic mean of a statistic series `series`.

classmethod interquartile_range (*series=None*)
Interquartile range of a statistic series `series`.

classmethod interval_interquartile (*series=None*)
Interval interquartile of a statistic series `series`.

classmethod kurtosis (*series=None*)
Kurtosis coefficient of Pearson of a statistic series `series`.

classmethod kurtosis_coefficient_fisher(*series=None*)
Kurtosis coefficient of Fisher of a statistic series `series`.

classmethod mad_from_median(*series=None*)
Mean absolute deviation from median of a statistic series `series`.

classmethod mean(*series=None*)
Arithmetic mean of a statistic series `series`.

classmethod mean_absolute_deviation(*series=None*)
Mean absolute deviation of a statistic series `series`.

classmethod median(*series=None*)
Median of a statistic series `series`.

classmethod mode(*series=None*)
Mode of a statistic series `series`.

classmethod moment_order_alpha(*series*, *alpha=2*)
Moment of order `alpha` of a statistic series `series`.

classmethod quadratic_mean(*series=None*)
Quadratic mean of a statistic series `series`.

classmethod quantile(*series*, *alpha*)
Quantile of order `alpha` of a statistic series `series`.

classmethod quartiles(*series=None*)
Quartiles *Q1, Q2* and *Q3* of a statistic series `series`.

classmethod sample_std(*series=None*)
Sample standard deviation of a statistic series `series`.

classmethod sample_variance(*series=None*)
Sample variance of a statistic series `series`.

classmethod skewness(*series=None*)
Skewness coefficient of Fisher of a statistic series `series`.

classmethod skewness_coefficient_of_pearson(*series=None*)
Skewness coefficient of Pearson of a statistic series `series`.

classmethod skewness_coefficient_of_yule(*series=None*)
Skewness coefficient of Yule of a statistic series `series`.

classmethod standard_deviation(*series=None*)
Standard deviation of a statistic series `series`.

classmethod var(*series=None*)
Variance of a statistic series `series`.

5.6.2 Statistics for Grouped Datas

Statistic characteristics of grouped data.

class simula.api.stats.tabular.**StatisticsGroupedDatas**(*values=None, frequencies=None*)

Statistic functions of grouped data.

classmethod arithmetic_mean(*values=None, frequencies=None*)
Arithmetic mean for grouped data.

classmethod class_median(*values=None, frequencies=None*)
Class median for grouped data.

classmethod coefficient_of_dispersion(*values=None, frequencies=None*)
Coefficient of dispersion for grouped data.

classmethod coefficient_of_variation(*values=None, frequencies=None*)
Coefficient of variation for grouped data.

classmethod geometric_mean(*values=None, frequencies=None*)
Geometric mean for grouped data.

classmethod harmonic_mean(*values=None, frequencies=None*)
Harmonic mean for grouped data.

classmethod interquartile_range(*values=None, frequencies=None*)
Interquartile range for grouped data.

classmethod interval_interquartile(*values=None, frequencies=None*)
Interval interquartile for grouped data.

classmethod kurtosis(*values=None, frequencies=None*)
Kurtosis coefficient of Pearson for grouped data.

classmethod kurtosis_coefficient_fisher(*values=None, frequencies=None*)
Kurtosis coefficient of Fisher for grouped data.

classmethod mad_from_median(*values=None, frequencies=None*)
Mean absolute deviation from median for grouped data.

classmethod mean(*values=None, frequencies=None*)
Arithmetic mean for grouped data.

classmethod mean_absolute_deviation(*values=None, frequencies=None*)
Mean absolute deviation for grouped data.

classmethod median(*values=None, frequencies=None*)
Median for grouped data.

classmethod mode(*values=None*, *frequencies=None*)
Mode(s) for grouped data.

classmethod moment_order_alpha(*values*, *frequencies*, *alpha=2*)
Moment of order `alpha` for grouped data.

classmethod quadratic_mean(*values=None*, *frequencies=None*)
Quadratic mean for grouped data.

classmethod quartiles(*values=None*, *frequencies=None*)
Quartiles *Q1* and *Q3* for grouped data.

classmethod sample_std(*values=None*, *frequencies=None*)
Sample Standard deviation for grouped data.

classmethod sample_variance(*values=None*, *frequencies=None*)
Sample variance for grouped data.

classmethod skewness(*values=None*, *frequencies=None*)
Skewness coefficient of Fisher for grouped data.

classmethod skewness_coefficient_of_pearson(*values=None*, *frequencies=None*)
Skewness coefficient of Pearson for grouped data.

classmethod skewness_coefficient_of_yule(*values=None*, *frequencies=None*)
Skewness coefficient of Yule for grouped data.

classmethod standard_deviation(*values=None*, *frequencies=None*)
Standard deviation for grouped data.

classmethod var(*values=None*, *frequencies=None*)
Variance for grouped data.

5.7 Number in Base B

Conversion of number from one base to another base

- NumberBaseB
- Bin
- oct
- Hex

`simula.api.base.`**Bin**
alias of *simula.api.base.Binary*

class `simula.api.base.`**Binary**(*number*)
Converts a number from one base into binary.

Parameters **number** – a number

EXAMPLES:

```
simula : a = Bin(34); a
0b100010
simula : a + a
0b1000100
simula : Bin(4) + Bin(10)
0b1110
simula : Bin(14)
0b1110
simula : Bin(4) + Bin(10) == Bin(14)
True
```

`simula.api.base.`**`Hex`**

alias of *`simula.api.base.Hexadicimal`*

`class` `simula.api.base.`**`Hexadicimal`**(*number*)

Converts a number from one base into hexadecimal base.

Parameters **`number`** – a number

EXAMPLES:

```
simula : a = Hex(1000); a
0x3e8
simula : a + a
0x7d0
simula : Hex(400) + Hex(1000)
0x578
simula : Hex(5099)
0x13eb
```

`class` `simula.api.base.`**`NumberBaseB`**(*number*, *base=2*)

Representation of number in some basis.

Parameters

- **`number`** – an integer
- **`base`** – (an integer) the basis

EXAMPLES:

```
simula : a = NumberBaseB(16, 2); a
[1, 0, 0, 0, 0]
simula : NumberBaseB(168, 8)
[2, 5, 0]
```

`to_list`(*length=None*)

Returns a list of size `length` of the representation of `self`.

Parameters **`length`** – (optional) the size of representation of `self`

EXAMPLES:

simula :

`simula.api.base.`**`Oct`**
alias of *`simula.api.base.Octal`*

`class` `simula.api.base.`**`Octal`**(*number*)
Converts a number from one base into octal base.

Parameters **`number`** – a number

EXAMPLES:

```
simula : a = Oct(100); a
0o144
simula : a + a
0o310
simula : Oct(40) + Oct(60)
0o144
simula : 2Oct(14)
28
```

`simula.api.base.`**`int_to_base_b`**(*number*, *base=2*, *length=None*)
Converts an integer from one base into another base for a given length.

Parameters

- **`number`** – (an integer) the number to write in some basis
- **`base`** – (an integer) the basis
- **`length`** – (optional) the length of the new vector

5.8 Cryptography

5.8.1 Classic Cryptosystems

Classical Encryption and Decryption Algorithms

- ShiftCryptosystem
- AffineCryptosystem
- PermutationCryptosystem
- SubstitutionCryptosystem
- VernamCryptosystem
- VigenereCryptosystem
- HillCryptosystem

class simula.api.crypto.classic.**AffineCryptosystem**(*alphabet=('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'), block_length=1*)

Affine Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = (5, 11)
simula : C = AffineCryptosystem.encipher(M, k); C
'CDIXFVSFC'
simula : AffineCryptosystem.decipher(C, k)
'TOPSECRET'
```

classmethod decipher(*cipher, key*)

Decryption algorithm for Affine Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = (5, 11)
simula : C = AffineCryptosystem.encipher(M, k); C
'CDIXFVSFC'
simula : AffineCryptosystem.decipher(C, k)
'TOPSECRET'
```

Parameters

- **cipher** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[str, int]*) –

Return type str

classmethod encipher(*message, key*)

Encryption algorithm for Affine Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = (5, 11)
simula : C = AffineCryptosystem.encipher(M, k); C
'CDIXFVSFC'
```

Parameters

- **message** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[str, int]*) –

Return type str

class simula.api.crypto.classic.**HillCryptosystem**(*alphabet=('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'), block_length=1*)

Hill Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = matrix([[11, 25, 25], [12, 16, 3], [11, 16, 14]])
simula : C = HillCryptosystem.encipher(M, k); C
'WDZIAWCNB'
simula : HillCryptosystem.decipher(C, k)
'TOPSECRET'
```

classmethod decipher(*cipher, key*)

Decryption algorithm or Hill Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = matrix([[11, 25, 25], [12, 16, 3], [11, 16,␣
↪14]])
simula : C = HillCryptosystem.encipher(M, k); C
'WDZIAWCNB'
simula : HillCryptosystem.decipher(C, k)
'TOPSECRET'
```

Parameters

- **cipher** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[str, int]*) –

Return type str

classmethod encipher(*message*, *key*)

Encryption algorithm or Hill Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = matrix([[11, 25, 25], [12, 16, 3], [11, 16,␣
↪14]])
simula : C = HillCryptosystem.encipher(M, k); C
'WDZIAWCNB'
```

Parameters

- **message** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[str, int]*) –

Return type str

class simula.api.crypto.classic.**PermutationCryptosystem**(*alphabet=('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'), block_length=1*)

Permutation Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = {0: 2, 1: 0, 2: 1}
simula : C = PermutationCryptosystem.encipher(M, k); C
'PTOCSETRE'
simula : PermutationCryptosystem.decipher(C, k)
'TOPSECRET'
simula : k2 = {0: 2, 1: 5, 2: 4, 3: 1, 4: 6, 5: 3, 6: 8, 7: 0,␣
↪8: 7}
simula : C2 = PermutationCryptosystem.encipher(M, k2); C2
'PCEORSTTE'
```

classmethod decipher(*cipher*, *key*)

Decryption algorithm for Permutation Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = {0: 2, 1: 0, 2: 1}
simula : C = PermutationCryptosystem.encipher(M, k); C
'PTOCSETRE'
simula : PermutationCryptosystem.decipher(C, k)
'TOPSECRET'
```

Parameters

- **cipher** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*dict*) –

Return type str

classmethod encipher(*message*, *key*)

Encryption algorithm for Permutation Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = {0: 2, 1: 0, 2: 1}
simula : C = PermutationCryptosystem.encipher(M, k); C
'PTOCSETRE'
```

Parameters

- **message** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*dict*) –

Return type str

class simula.api.crypto.classic.**ShiftCryptosystem**(*alphabet=('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'), block_length=1*)

Shift Cryptosystem.

EXAMPLES:

```
simula : M, k = "TOPSECRET", 7
simula : C = ShiftCryptosystem.encipher(M, k); C
'AVWZLJYLA'
simula : ShiftCryptosystem.decipher(C, k)
'TOPSECRET'
```

classmethod decipher(*cipher, key*)

Decryption algorithm for Shift Cryptosystem.

EXAMPLES:

```
simula : M, k = "TOPSECRET", 7
simula : C = ShiftCryptosystem.encipher(M, k); C
'AVWZLJYLA'
simula : ShiftCryptosystem.decipher(C, k)
'TOPSECRET'
```

Parameters

- **cipher** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[str, int]*) –

Return type str

classmethod encipher(*message, key*)

Encryption algorithm for Shift Cryptosystem.

EXAMPLES:

```
simula : M, k = "TOPSECRET", 7
simula : C = ShiftCryptosystem.encipher(M, k); C
'AVWZLJYLA'
```

Parameters

- **message** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[str, int]*) –

Return type str

classmethod keygen()
Returns randomly a key for the Shift Cryptosystem.

class simula.api.crypto.classic.**SubstitutionCryptosystem**(*alphabet=('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'), block_length=1*)

Substitution Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = {0: 20, 1: 24, 2: 12, 3: 7, 4: 22, 5: 0, 6: 1, 7:␣
↪16, 8: 6, 9: 9, 10: 17, 11: 15,
 12: 4, 13: 18, 14: 23, 15: 8, 16: 19, 17: 13, 18: 2, 19: 3,␣
↪20: 21, 21: 11, 22: 5, 23: 10, 24: 25, 25: 14}
simula : C = SubstitutionCryptosystem.encipher(M, k); C
'DXICWMNWD'
simula : SubstitutionCryptosystem.decipher(C, k)
'TOPSECRET'
```

classmethod decipher (*cipher*, *key*)

Decryption algorithm for Substitution Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = {0: 20, 1: 24, 2: 12, 3: 7, 4: 22, 5: 0, 6: 1,␣
↪7: 16, 8: 6, 9: 9, 10: 17, 11: 15,
12: 4, 13: 18, 14: 23, 15: 8, 16: 19, 17: 13, 18: 2, 19: 3,␣
↪20: 21, 21: 11, 22: 5, 23: 10, 24: 25, 25: 14}
simula : C = SubstitutionCryptosystem.encipher(M, k); C
'DXICWMNWD'
simula : SubstitutionCryptosystem.decipher(C, k)
'TOPSECRET'
```

Parameters

- **cipher** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*dict*) –

Return type str

classmethod encipher (*message*, *key*)

Encryption algorithm for Substitution Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = {0: 20, 1: 24, 2: 12, 3: 7, 4: 22, 5: 0, 6: 1,␣
↪7: 16, 8: 6, 9: 9, 10: 17, 11: 15,
12: 4, 13: 18, 14: 23, 15: 8, 16: 19, 17: 13, 18: 2, 19: 3,␣
↪20: 21, 21: 11, 22: 5, 23: 10, 24: 25, 25: 14}
simula : C = SubstitutionCryptosystem.encipher(M, k); C
'DXICWMNWD'
```

Parameters

- **message** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*dict*) –

Return type str

class simula.api.crypto.classic.**VernamCryptosystem**(*alphabet=('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'), block_length=1*)

Vernam Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = (23, 13, 25, 22, 2, 16, 9, 11, 7)
simula : C = VernamCryptosystem.encipher(M, k); C
'QBOOGSAPA'
simula : VernamCryptosystem.decipher(C, k)
'TOPSECRET'
```

classmethod decipher(*cipher, key*)

Decryption algorithm for Vernam Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = (23, 13, 25, 22, 2, 16, 9, 11, 7)
simula : C = VernamCryptosystem.encipher(M, k); C
'QBOOGSAPA'
simula : VernamCryptosystem.decipher(C, k)
'TOPSECRET'
```

Parameters

- **cipher** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[Tuple[str], Tuple[int]]*) –

Return type str

classmethod encipher(*message, key*)

Encryption algorithm for Vernam Cryptosystem.

EXAMPLES:

```
simula : M = "TOPSECRET"
simula : k = (23, 13, 25, 22, 2, 16, 9, 11, 7)
simula : C = VernamCryptosystem.encipher(M, k); C
'QBOOGSAPA'
```

Parameters

- **message** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[Tuple[str], Tuple[int]]*) –

Return type str

class simula.api.crypto.classic.**VigenereCryptosystem**(*alphabet=('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z')*, *block_length=1*)

Vigenere Cryptosystem.

EXAMPLES:

```
simula : M, k = "TOPSECRET", (12, 16, 16)
simula : C = VigenereCryptosystem.encipher(M, k); C
''FEFEUSDUJ'
simula : VigenereCryptosystem.decipher(C, k)
'TOPSECRET'
```

classmethod decipher(*cipher*, *key*)

Decryption algorithm for Vigenere Cryptosystem.

EXAMPLES:

```
simula : M, k = "TOPSECRET", (12, 16, 16)
simula : C = VigenereCryptosystem.encipher(M, k); C
'FEFEUSDUJ'
simula : VigenereCryptosystem.decipher(C, k)
'TOPSECRET'
```

Parameters

- **cipher** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[Tuple[str], Tuple[int]]*) –

Return type str

classmethod encipher(*message*, *key*)

Encryption algorithm for Vigenere Cryptosystem.

EXAMPLES:

```
simula : M, k = "TOPSECRET", (12, 16, 16)
simula : C = VigenereCryptosystem.encipher(M, k); C
'FEFEUSDUJ'
```

Parameters

- **message** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[Tuple[str], Tuple[int]]*) –

Return type str

simula.api.crypto.classic.**ascii_letters**()

Returns the ASCII letters.

simula.api.crypto.classic.**clean_text**(*message*, *alphabet=None*)

Transforms or deletes all non-ascii letters.

Parameters **message** (*str*) –

Return type str

5.8.2 Asymmetric Schemes

Public key encryption and Signature schemes.

- RSA (encryption and signature)
- ElGamal (encryption and signature)
- DSA (Digital Signature Algorithm)

class simula.api.crypto.asymmetric.**DSA**

Digital Signature Algorithm (DSA)

See https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

classmethod keygen(*N=None*, *L=None*)

Returns a keypair `sk = (p, q, g, x)` and ``pk = (p, q, g, y)` when `p` and `q` have size respectively `L` and `N`.

Parameters

- **N** (*Optional[int]*) –
- **L** (*Optional[int]*) –

classmethod signing(*message*, *sk*, *hash_func=None*)

DSA signing algorithm.

Parameters

- **message** (*int*) – an integer
- **sk** (*Tuple[int, int, int, int]*) – the private key `sk = (p, q, g, x)`
- **hash_func** – (optional) message digest

Return type Tuple[int, int]

classmethod verifier(*sign*, *pk*, *message*, *hash_func=None*)

DSA Verification algorithm.

Parameters

- **sign** (*Tuple[int, int]*) – an integer
- **pk** (*Tuple[int, int, int, int]*) – the public key `pk = (p, q, g, y)`
- **message** (*int*) – an integer
- **hash_func** – (optional) message digest

Return type bool

class simula.api.crypto.asymmetric.**ECDSA**

Elliptic Curve Digital Signature Algorithm (ECDSA)

classmethod keygen(**, *size=None*, *p=None*, *ec_coeffs=None*)

Returns a keypair `sk = (q, E, G, x)` and ``pk = (q, E, G, Y)`` when `p` has size `size`.

Parameters

- **size** (*Optional[int]*) –
- **p** (*Optional[int]*) –

classmethod signing(*message*, *sk*, *hash_func=None*)

ECDSA signing algorithm.

Parameters

- **message** (*int*) – an integer
- **sk** (*Tuple[int, simula.api.hecc.weirstrass.EllipticCurve, Union[simula.api.hecc.curve.EllipticCurvePoint, Tuple], int]*) – the private key `sk = (q, E, G, x)`
- **hash_func** – (optional) message digest

Return type Tuple[int, int]

classmethod verifier(*sign*, *pk*, *message*, *hash_func=None*)

ECDSA Verification algorithm.

Parameters

- **sign** (*Tuple[int, int]*) – an integer
- **pk** (*Tuple[int, simula.api.hecc.weirstrass.EllipticCurve, Union[simula.api.hecc.curve.EllipticCurvePoint, Tuple], Union[simula.api.hecc.curve.EllipticCurvePoint, Tuple]]*) – the public key `pk = (q, E, G, Y)`
- **message** (*int*) – an integer
- **hash_func** – (optional) message digest

Return type bool

class `simula.api.crypto.asymmetric.`**ElGamal**

El Gamal scheme : Encryption and Signature.

EXAMPLES:

```
simula : scheme = ElGamal()
simula : p, a = 11, 7
simula : p, g, ga = 11, 2, 7
simula : sk, pk = (p, a), (p, g, ga)
simula : c = scheme.encipher(30, pk); c
(7, 3)
simula : scheme.decipher(c, sk)
3
simula : ElGamal.decipher(ElGamal.encipher(5, pk), sk) == 5
True
```

classmethod decipher (*cipher*, *sk*)

El Gamal decipher algorithm.

Parameters

- **cipher** (*int*) – an integer
- **sk** (*int*) – the private key `sk = (p, a)`

Return type int

classmethod encipher (*message*, *pk*)

El Gamal encipher algorithm.

Parameters

- **message** (*int*) – an integer
- **pk** (*int*) – the public key *pk = (p, g, ga)*`

Return type Tuple[int, int]

classmethod keygen (*size=None*, *sign=False*)

Returns a keypair `sk = (p, a) or (p, g, a)` and `pk = (p, g, ga)` when `p` has size `size`.

Parameters **size** (*Optional[int]*) –

classmethod signing(*message*, *sk*, *hash_func=None*)
El Gamal signing algorithm.

Parameters

- **message** (*int*) – an integer
- **sk** (*Tuple[int, int, int]*) – the private key `sk = (p, g, a)`
- **hash_func** – (optional) message digest

Return type Tuple[int, int]

classmethod verifier(*sign*, *pk*, *message*, *hash_func=None*)
El Gamal Verification algorithm.

Parameters

- **sign** (*Tuple[int, int]*) – an integer
- **pk** (*Tuple[int, int, int]*) – the public key `pk = (p, g, ga)`
- **message** (*int*) – an integer
- **hash_func** – (optional) message digest

Return type bool

class simula.api.crypto.asymmetric.**RSA**
RSA scheme : Encryption and Signature.

EXAMPLES:

```
simula : scheme = RSA()
simula : p, q, d = 11, 13, 107
simula : N, e = 143, 83
simula : sk, pk = (p, q, d), (N, e)
simula : c = scheme.encipher(30, pk); c
127
simula : scheme.decipher(c, sk)
30
simula : RSA.decipher(RSA.encipher(58, pk), sk) == 58
True
```

classmethod decipher(*cipher*, *sk*)
RSA decipher algorithm.

Parameters

- **cipher** (*int*) – an integer
- **sk** (*Union[Tuple[int, int], Tuple[int, int, int]]*) – the private key `sk = (p, q, d)`

Return type int

classmethod encipher(*message*, *pk*)
RSA encipher algorithm.

Parameters

- **message** (*int*) – an integer
- **pk** (*Tuple[int, int]*) – the public key *pk = (N, e)*`

Return type int

classmethod keygen(*size=None*)
Returns a keypair `sk = (p, q, d)` and `pk = (N, e)` when `p` and `q` have size `size`.

Parameters **size** (*Optional[int]*) –

classmethod signing(*message*, *sk*, *hash_func=None*)
RSA signing algorithm.

Parameters

- **message** (*int*) – an integer
- **sk** (*Union[Tuple[int, int], Tuple[int, int, int]]*) – the private key `sk = (p, q, d)`
- **hash_func** – (optional) message digest

Return type int

classmethod verifier(*sign*, *pk*, *message*, *hash_func=None*)
RSA Verification algorithm.

Parameters

- **sign** (*int*) – an integer
- **pk** (*Tuple[int, int]*) – the public key `pk = (N, e)`
- **message** (*int*) – an integer
- **hash_func** – (optional) message digest

Return type int

5.8.3 Schemes based on Elliptic Curves

class simula.api.crypto.ecc.**ECDSA**
Elliptic Curve Digital Signature Algorithm (ECDSA)

classmethod keygen(**, size=None, p=None, ec_coeffs=None*)
Returns a keypair `sk = (q, E, G, x)` and ``pk = (q, E, G, Y)` when `p` has size `size`.

Parameters

- **size** (*Optional[int]*) –

- **p** (*Optional[int]*) –

classmethod signing(*message*, *sk*, *hash_func=None*)
ECDSA signing algorithm.

Parameters

- **message** (*int*) – an integer
- **sk** (*Tuple[int, simula.api.hecc.weirstrass.EllipticCurve, Union[simula.api.hecc.curve.EllipticCurvePoint, Tuple], int]*) – the private key `sk = (q, E, G, x)`
- **hash_func** – (optional) message digest

Return type Tuple[int, int]

classmethod verifier(*sign*, *pk*, *message*, *hash_func=None*)
ECDSA Verification algorithm.

Parameters

- **sign** (*Tuple[int, int]*) – an integer
- **pk** (*Tuple[int, simula.api.hecc.weirstrass.EllipticCurve, Union[simula.api.hecc.curve.EllipticCurvePoint, Tuple], Union[simula.api.hecc.curve.EllipticCurvePoint, Tuple]]*) – the public key `pk = (q, E, G, Y)`
- **message** (*int*) – an integer
- **hash_func** – (optional) message digest

Return type bool

5.9 Coding Theory

5.9.1 Linear Codes

Linear Codes

class simula.api.coding.linear_code.**LinearCode**(*field=GF(2)*, *gen_matrix=None*, *check_matrix=None*)
Representation of linear codes.

EXAMPLES:

```
simula : G = matrix([ [1,1,1,0,1,1], [0,1,0,0,1,1], [1,0,1,1,0,
↪1], [0,1,1,1,0,1] ])
simula : G
```

(continues on next page)

(continued from previous page)

```
Matrix([
[1, 1, 1, 0, 1, 1],
[0, 1, 0, 0, 1, 1],
[1, 0, 1, 1, 0, 1],
[0, 1, 1, 1, 0, 1]])
simula : C = LinearCode(GF(2), G); C
Linear code over GF(2) of generator matrix
Matrix([
[1, 1, 1, 0, 1, 1],
[0, 1, 0, 0, 1, 1],
[1, 0, 1, 1, 0, 1],
[0, 1, 1, 1, 0, 1]])
simula : C.dimension()
4
simula : C.minimum_distance()
2
simula : C.correction_capacity()
0
simula : C.parity_check_matrix()
Matrix([
[1, 1, 1, 0, 1, 0],
[1, 1, 1, 1, 0, 1]])
simula : C.all_codewords()
[(0, 0, 0, 0, 0, 0), (0, 1, 1, 1, 0, 1), (1, 0, 1, 1, 0, 1), (1,
↪ 1, 0, 0, 0, 0), (0, 1, 0, 0, 1, 1),
 (0, 0, 1, 1, 1, 0), (1, 1, 1, 1, 1, 0), (1, 0, 0, 0, 1, 1), (1,
↪ 1, 1, 0, 1, 1), (1, 0, 0, 1, 1, 0),
  (0, 1, 0, 1, 1, 0), (0, 0, 1, 0, 1, 1), (1, 0, 1, 0, 0, 0),␣
↪(1, 1, 0, 1, 0, 1), (0, 0, 0, 1, 0, 1),
  (0, 1, 1, 0, 0, 0)]
```

control_matrix()

Returns a parity check matrix of `self`.

Return type *simula.api.linalg.matrices.Matrix*

correction_capacity()

Returns the error correction capacity of `self`.

dimension()

Returns the dimension of `self`.

dual_code()

Returns the dual code of `self`.

encode(*m*)

Returns the encoding of the vector `m`.

generator_matrix()

Returns a generator matrix of the linear code `self`.

Return type *simula.api.linalg.matrices.Matrix*

is_codeword(*w*)
Returns True if w is a codeword of self and False otherwise.

INPUT:

- w – a word

property k
Returns the dimension of self.

length()
Returns the length of self.

minimum_distance()
Returns the minimum distance of self.

property n
Returns the length of self.

number_of_codewords()
Returns the number of codewords (cardinality) of self.

parity_check_matrix()
Returns a parity check matrix of self.

Return type *simula.api.linalg.matrices.Matrix*

syndrome(*w*)
Returns the syndrome of the word w.

INPUT:

- w – a word

5.9.2 Hamming Codes

Hamming codes

class simula.api.coding.hamming_code.**HammingCode**(*field*, *r=3*)
Bases: *simula.api.coding.linear_code.LinearCode*

Representation of a hamming code.

EXAMPLES:

```
simula : C = HammingCode(GF(2), r=3); C
Hamming Code defined over GF(2) of parity check matrix
Matrix([
[0, 0, 0, 1, 1, 1, 1],
[0, 1, 1, 0, 0, 1, 1],
[1, 0, 1, 0, 1, 0, 1]])
simula : C.generator_matrix()
Matrix([
[1, 0, 0, 0, 0, 1, 1],
[0, 1, 0, 0, 1, 0, 1],
```

(continues on next page)

(continued from previous page)

```
[0, 0, 1, 0, 1, 1, 0],
[0, 0, 0, 1, 1, 1, 1]])
simula : C.dimension()
4
simula : C.correction_capacity()
1
```

dimension()
Returns the dimension of `self`.

length()
Returns the length of `self`.

minimum_distance()
Returns the minimum distance of `self`.

5.9.3 Cyclic Codes

Cyclic Codes

class simula.api.coding.cyclic_code.**CyclicCode**(*length=None*, *gen_poly=None*, *check_poly=None*, *code=None*)

Bases: *simula.api.coding.linear_code.LinearCode*

Representation of a cyclic code.

There are two different ways to create a new CyclicCode, either by providing:

- the generator polynomial and the length (1) or
- the check polynomial and the length (2).

Parameters

- **gen_poly** – (default: `None`) the generator polynomial of `self`. That is, the highest-degree monic polynomial which divides every polynomial representation of a codeword in `self`.
- **check_poly** – (default: `None`) the check polynomial of `self`.
- **length** – (default: `None`) the length of `self`. It has to be bigger than the degree of `gen_poly`.

EXAMPLES:

```
simula: R.<x> = GF(2)[]
simula: g = x^3 + x + 1
simula: C = CyclicCode(gen_poly=g, length=7)
simula : C
```

(continues on next page)

(continued from previous page)

```
Linear code over GF(2) of generator matrix
Matrix([
[1, 1, 0, 1, 0, 0, 0],
[0, 1, 1, 0, 1, 0, 0],
[0, 0, 1, 1, 0, 1, 0],
[0, 0, 0, 1, 1, 0, 1]])
simula: h = C.check_polynomial(); h
x^4 + x^2 + x + 1
simula: C2 = CyclicCode(check_poly=h, length=7)
simula : C2
Linear code over GF(2) of parity check matrix
Matrix([
[1, 0, 1, 1, 1, 0, 0],
[0, 1, 0, 1, 1, 1, 0],
[0, 0, 1, 0, 1, 1, 1]])
simula : C2.generator_polynomial()
x^3 + x + 1
```

check_polynomial()

Returns the check polynomial of `self`.

EXAMPLES:

```
simula: R.<x> = GF(2)[]
simula: g = x^3 + x + 1
simula: C = CyclicCode(gen_poly=g, length=7)
simula: C.check_polynomial()
x^4 + x^2 + x + 1
```

generator_polynomial()

Returns the generator polynomial of `self`.

EXAMPLES:

```
simula: R.<x> = GF(2)[]
simula: g = x^3 + x + 1
simula: C = CyclicCode(gen_poly=g, length=7)
simula: C.generator_polynomial()
x^3 + x + 1
```

is_codeword(*w*)

Returns `True` if `w` is a codeword of `self` and `False` otherwise.

Parameters **w** – a word

length()

Returns the length of `self`.

parity_check_matrix()

Returns the parity check matrix of `self`.

EXAMPLES:

```
simula: R.<x> = GF(2)[]
simula: g = x^3 + x + 1
simula: C = CyclicCode(gen_poly=g, length=7)
simula: C.parity_check_matrix()
Matrix([
[1, 0, 1, 1, 1, 0, 0],
0, 1, 0, 1, 1, 1, 0],
[0, 0, 1, 0, 1, 1, 1]])
```

Return type *simula.api.linalg.matrices.Matrix*

syndrome (*w*, *poly=False*)

Returns the syndrome of the word `w` in the form of a polynomial or a vector.

Parameters

- **w** – a word
- **poly** – (default: `False`) if `True` the syndrome is returned as a polynomial

5.10 Polynomials ring

5.10.1 Multivariate Polynomials ring

Operations over polynomial rings.

class simula.api.polyring.polyring.**PolynomialRing** (*domain*, *symbols=None*, *order=DegreeLexicographicOrder()*, ***kwargs*)

Multivariate polynomial ring.

INPUT:

Parameters

- **domain** – a domain (eg. QQ, RR, CC, ZZ, GF(p))
- **symbols** – a sequence of symbols
- **order** – (default 'deglex') a monomial ordering e.g. 'lex', 'deglex', 'degrevlex'
- **kwargs** –

EXAMPLES:

```
simula : R = PolynomialRing(QQ, "x, y, z", order="lex")
simula : x, y ,z = R.gens

These two lines are equivalent to the following code:

simula : R.<x, y ,z> = PolynomialRing(QQ, "x, y, z", order="lex
↪")

By default the monomial ordering is "deglex", if don't need to␣
↪change it, we can simplify again the notation.

simula : R.<x, y ,z> = QQ[]
simula : p1 = x^3*y-x*y^2-x-z; p1
x^3*y - x*y^2 - x - z
simula : p1.lcm(x-y-y*z)
x^4*y - x^3*y^2*z - x^3*y^2 - x^2*y^2 - x^2 + x*y^3*z + x*y^3 +␣
↪x*y*z + x*y - x*z + y*z^2 + y*z
simula : R.make_monic(6x^4-3x-1)
x^4 - 1/2x - 1/6
simula : I = R.ideal([x*y^2-y-z, x^2*z-y*x]); I
ideal generated by [x*y^2 - y - z, x^2*z - x*y] of Polynomial␣
↪ring in x, y, z over QQ with deglex order
```

add (*pol1*, *pol2*)
Returns `pol1 + pol2` in `self`.

change_ring (*domain=None*, *symbols=None*, *order=None*)
Returns a new polynomial ring with the new given domain `domain`.

characteristic ()
Returns the characteristic `self`.

cyclotomic_polynomial (*n*)
Returns the n-th cyclotomic polynomial.

EXAMPLES:

```
simula : R.<x> = GF(5)[]
simula : R.cyclotomic_polynomial(3)
x^2 + x + 1
simula : R.cyclotomic_polynomial(6)
x^2 + 4x + 1
```

div (*pol1*, *pol2*)
Returns the quotient and the remainder of the division of `pol1` by `pol2` in `self`.

factor (*pol*)
Returns the factorisation of the polynomial `pol`.

gcd (*pol1*, *pol2*)
Returns the gcd of `pol1` and `pol2`.

gcdex (*pol1*, *pol2*)
Returns the extended gcd of `pol1` and `pol2`.

ideal (*F*)
Returns the ideal in `self` generated by `F`.

is_exact ()
Tests if `self` is an exact domain.

is_field ()
Tests if `self` is a field.

is_irreducible (*pol*)
Tests if `pol` is an irreducible polynomial.

lcm (*pol1*, *pol2*)
Returns the lcm of `pol1` and `pol2`.

make_monic (*pol*)
Makes monic the polynomial `pol`.

monic (*pol*)
Makes monic the polynomial `pol`.

mul (*pol1*, *pol2*)
Returns `pol1 * pol2` in `self`.

objgen ()
Returns `self` and its generators.

EXAMPLES:

```
simula : ring = PolynomialRing(QQ, "x, y, z", order="lex");␣
↪ring
simula : R, gens = ring.objgen()
simula : R
Multivariate Polynomial Ring in x, y, z over QQ with lex␣
↪order
simula : gens
(x, y, z)
```

pow (*pol*, *n*)
Returns `pol1^n` in `self`.

primitive_polynomials (*deg*)
Returns the primitive polynomials of degree `deg` if `self` is a finite polynomial ring.

EXAMPLES:

```
simula : R.<x> = GF(5)[]
simula : R.primitive_polynomials(3)
{x^2 + x + 2, x^2 + 4x + 2, x^2 + 3x + 3, x^2 + 2x + 3}
```

quo (*pol1*, *pol2*)
Returns the quotient of the division of `pol1` by `pol2` in `self`.

random_irreducible (*n*)
Returns a random irreducible polynomial of degree `n`.

rem (*pol1*, *pol2*)
Returns the remainder of the division of `pol1` by `pol2` in `self`.

roots (*f*)
Returns the roots of the polynomial `n`.

sub (*pol1*, *pol2*)
Returns `pol1 - pol2` in `self`.

univariate_ring (*x*)
Returns a univariate ring in `x` which has the same domain as `self`.

5.10.2 Groeber Bases

Operations over Groebner Bases.

class simula.api.polyring.groebner.**Ideal** (*F*, *symbols=None*, *domain=None*, *order=None*, ***, *ring=None*)

Ideal generated by a set of polynomials `F`.

Parameters

- **F** – a list of polynomials
- **symbols** – (optional) list of variables
- **domain** – (optional) a domain e.g. QQ, RR, ZZ
- **order** – (optional) a monomial ordering e.g. 'lex', 'deglex', 'degrevlex'
- **ring** – (optional) a polynomial ring.

EXEMPLES:

```
simula : R.<x, y, z> = QQ[]
simula : R
Multivariate Polynomial Ring in x, y, z over QQ with deglex␣
↪order
simula : I = ideal([x^2*y-z, x*y-1]); I
ideal generated by [x^2*y - z, x*y - 1] of Polynomial ring in x,
↪ y, z over QQ with deglex order
simula : J = (x^2*y-z, x*y-1) * R; J
ideal generated by [x^2*y
simula : I == J
True
simula : J.groebner_basis()
```

(continues on next page)

(continued from previous page)

```
[y*z - 1, x - z]
simula : J.buchberger()
[x^2*y - z, x*y - 1, x - z, y*z - 1]
simula : J.homogenize('h')
ideal generated by [x^2*y - z*h^2, x*y - h^2] of Polynomial␣
↪ring in x, y, z, h over QQ with deglex order
simula : J.reduce(x-y)
-y + z
simula : J.reduce(x^2*y-z + 2x*y-2)
0
```

basis()
Returns the basis of `self`.

basis_as_expr()
Returns the basis of `self` as an expression.

basis_is_groebner()
Tests if the given basis is a groebner basis of `self`.

buchberger()
Returns a groebner basis of `self` using a toy Buchberger algorithm.

change_ring(*new_ring*)
Returns a new ideal with the new polynomial ring.

groebner_basis()
Returns a reduced groebner basis of `self`.

groebner_basis_f5()
Returns a reduced groebner basis of `self` using the F5 algorithm.

homogenize(*var=None*)
Returns the ideal generated by the homogeneous polynomials of the basis of `self`.

is_homogeneous()
Tests if the polynomials in the basis of `self` are homogeneous.

is_in_radical_ideal(*f*)
Tests if `f` is in radical of `self`.

leading_ideal()
Returns the leading ideal of `self`.

normal_form(*f, greobner=False*)
Returns the normal form of `f` with respect to the basis of `self`.

reduce(*f*)
Reduces `f` with respect to the basis of `self`.

weak_normal_form(*f, greobner=False*)
Returns the weak normal form of `f` with respect to the basis of `self`.

`simula.api.polyring.groebner.`**`LC`**(*f*, *symbols=None*, ***kwargs*)

Returns the leading coefficient of `f`.

`simula.api.polyring.groebner.`**`LM`**(*f*, *symbols=None*, ***kwargs*)

Returns the leading monomial of `f`.

`simula.api.polyring.groebner.`**`LT`**(*f*, *symbols=None*, ***kwargs*)

Returns the leading term of `f`.

`simula.api.polyring.groebner.`**`groebner_basis`**(*G*, *symbols=None*, *domain=None*, *order=None*)

Returns a reduced groebner basis of the ideal generated by ``G`.

`simula.api.polyring.groebner.`**`groebner_f5`**(*G*, *symbols=None*, *domain=None*, *order=None*)

Returns a reduce groebner basis using the F5 algorithm.

`simula.api.polyring.groebner.`**`ideal`**

alias of *`simula.api.polyring.groebner.Ideal`*

`simula.api.polyring.groebner.`**`leading_coefficient`**(*f*, *symbols=None*, ***kwargs*)

Returns the leading coefficient of `f`.

`simula.api.polyring.groebner.`**`leading_ideal`**(*I*)

Returns the leading ideal of `I`.

Parameters **I** (`simula.api.polyring.groebner.Ideal`) –

`simula.api.polyring.groebner.`**`leading_monom`**(*f*, *symbols=None*, ***kwargs*)

Returns the leading monomial of `f`.

`simula.api.polyring.groebner.`**`leading_term`**(*f*, *symbols=None*, ***kwargs*)

Returns the leading term of `f`.

`simula.api.polyring.groebner.`**`normal_form`**(*f*, *G*, *symbols=None*, *domain=Rational Numbers*, *order=DegreeLexicographicOrder()*)

Returns the normal form of `f` in `G`.

`simula.api.polyring.groebner.`**`spoly`**(*f*, *g*, *symbols=None*, *domain=Rational Numbers*, *order=DegreeLexicographicOrder()*)

Returns the S-polynomial of `f` and `g` .

`simula.api.polyring.groebner.`**`weak_normal_form`**(*f*, *G*, *symbols=None*, *domain=Rational Numbers*, *order=DegreeLexicographicOrder()*)

Returns the weak normal form of `f` in `G`

5.11 Elliptic Curves

5.11.1 Curves

Implementation of Elliptic curves over Finite Fields.

class simula.api.hecc.curve.**EllipticCurveObject**(*domain*, *projective=False*)

Main class of any Elliptic curve.

add(*p1*, *p2*)
Addition of *p1* and *p2*.

base_ring()
Returns the base ring : the domain.

cardinality()
Returns the order of `self`.

get_point_at_infinity()
Returns the point at infinity of `self`.

static is_irreducible()
Tests if `self` is irreducible.

static is_order_finite()
Returns *True* of the number of points of `self` is **finite** and *False* otherwise.

is_ordinary()
Tests if `self` is an ordinary elliptic curve.

static is_singular()
Tests if `self` is singular.

static is_smooth()
Tests if `self` is smooth.

is_supersingular()
Tests if `self` is a supersingular elliptic curve.

multiply_by_scalar(*P*, *k=2*)
Scalar multiplication $kP = P + P + \ldots + P$ k times.

order()
Returns the order of `self`.

random_element()
Returns a random point of `self`.

random_point()
Returns a random point of `self`.

rational_points()
Returns the rational points of `self`.

trace_of_frobenius()
Returns the trace of Frobenius of `self`.

class simula.api.hecc.curve.**EllipticCurvePoint**(*curve*, *x=None*, *y=None*, *z=None*, *, *projective=False*, *check=True*)
Point of an elliptic curve.

cardinality()
Returns the order of `self`.

get_generated_sub_group()
Returns the additive sub-group generated by `self`.

is_point()
Tests if `self` is a point.

is_point_at_infinity()
Tests if `self` is the point at infinity.

opposite()
Returns the opposite point of `self`.

order()
Returns the order of `self`.

xy()
returns th (x, y) coordinates.

class simula.api.hecc.curve.**GroupGeneratedBy**(*point*)
Additive-Sub group of an elliptic curve generated by a point.

all_group_points()
Returns all rational points of `self`.

is_point(*Q*)
Tests if *Q* is a point of `self`.

order()
Returns the order of `self`.

random_point()
Returns a random point of `self`.

rational_points()
Returns all rational points of `self`.

5.11.2 Weierstrass Curves

Implementation of Elliptic curves over Finite Fields.

- Elliptic curves defined by a short Weierstrass equation
- Elliptic curves defined by a long Weierstrass equation

simula.api.hecc.weirstrass.**EllipticCurve**(*domain*, **coeffs*, *projective=False*)

Returns an elliptic curve over a einite field.

Parameters

- **domain** (simula.api.finite_field.finite_field.FiniteField) – a finite field of size *p^n*.
- **coeffs** (*Union[Sized, simula.api.finite_field.finite_field.ElementFiniteField, Iterable]*) – the list of coefficients. The size should be either 2 (for a short Weierstrass equation $y^2 = x^3 + ax + b$) or 5 (for a long Weierstrass equation $y^2 + a_3xy + a_1y = x^3 + a_2x^2 + a_4x + a_6$).
- **projective** – (a boolean) if True the equation and rational points will be printed in projective form.

EXAMPLES:

```
simula : E = EllipticCurve(GF(11), [1, 5]); E
Elliptic curve defined by : y^2 = x^3 + x + 5 over GF(11)
simula : E.rational_points()
[(0, 4), (0, 7), (2, 2), (2, 9), (5, 5), (5, 6), (7, 5), (7, 6),
↪ (10, 5), (10, 6), P_oo]
simula : E.projective = True
simula : E
Elliptic curve defined by : Y^2*Z = X^3 + X*Z^2 + 5Z^3 over␣
↪GF(11)
simula : E.rational_points()
[(0 : 1 : 0), (0 : 4 : 1), (0 : 7 : 1), (2 : 2 : 1), (2 : 9 :␣
↪1), (5 : 5 : 1), (5 : 6 : 1),
 (7 : 5 : 1), (7 : 6 : 1), (10 : 5 : 1), (10 : 6 : 1)]
simula : E2 = EllipticCurve(GF(7), [1, 0, 1, -3, 2]); E2
Elliptic curve defined by : y^2 + y*x + y = x^3 - 3x + 2 over␣
↪GF(7)
simula : E2.a_invariants()
(1, 0, 1, -3, 2)
simula : E2.b_invariants()
(1, 2, 2, 3)
simula : E2.order()
11
simula : E3 = E2.short_weierstrass_model(); E3
Elliptic curve defined by : y^2 = x^3 + 2x + 6 over GF(7)
simula : E3.order()
```

(continues on next page)

(continued from previous page)

```
11
simula : E3.rational_points()
[(1, 3), (1, 4), (2, 2), (2, 5), (3, 2), (3, 5), (4, 1), (4, 6),
↪ (5, 1), (5, 6), P_oo]
simula : P = E3(1, 4); P
(1, 4)
simula : 7P
(3, 5)
simula : Q = E3(3, 2); Q
(3, 2)
simula : P-Q
(5, 1)
simula : P.order()
11
simula : 11P
P_oo
simula : P.projective = True
simula : 11P
(0 : 1 : 0)
```

class simula.api.hecc.weirstrass.**WeierstrassCurve**(*domain*, *projective=False*)

Bases: *simula.api.hecc.curve.EllipticCurveObject*

b_invariants()
: Returns the b-invariant of `self`.

c_invariants()
: Returns the c-invariant of `self`.

discriminant()
: Returns the discriminant of `self`.

j_invariant()
: Returns the j-invariant of `self`.

order()
: Returns the order of `self` i.e the number of elements of `self`.

5.11.3 Montgomery Curves

Implementation of Montgomery Curves.

class simula.api.hecc.montgomery.**MontgomeryCurve**(*domain*, *b=None*, *a=None*, ***, *projective=False*)

Bases: *simula.api.hecc.curve.EllipticCurveObject*

Elliptic curve defined by a Montgomery curve in the form $by^2 = x^3 + ax^2 + x$ over a finite field.

Parameters

- **domain** – a finite field
- **b** – a non-square in the domain
- **a** – an element of the domain
- **projective** – (a boolean) if `True` the equation and rational points will be printed in projective form.

```
simula : E = MontgomeryCurve(GF(11), 2, 5); E
Elliptic curve in Montgomery form defined by : 2y^2 = x^3 + 5x^
↪2 + x over GF(11)
simula : E.rational_points()
[(0, 0), (1, 3), (1, 8), (2, 2), (2, 9), (5, 1), (5, 10), (6,␣
↪5), (6, 6), (9, 4), (9, 7), P_oo]
simula : E.order()
12
simula : E2 = E.short_weierstrass_model(); E2
Elliptic curve defined by : y^2 = x^3 + 7 over GF(11)
simula : E2.order()
12
```

add_distinct_points(*p1*, *p2*)
Addition of `p1` and `p2` with `p1 != p2 != POINT_INFINI` and `p1 != -p2`.

doubling(*P*)
Doubling of point `P`.

is_point(*Q*)
Tests if `Q` is a point of `self`.

j_invariant()
Returns the j-invariant of `self`.

short_weierstrass_model()
Returns an elliptic curve defined by a short Weierstrass equation $y^2 = x^3 + ax + b$ which is birationaly equivalent to `self`.

Chapter 6

Licences

6.1 SimulaMath

TERMES DU CONTRAT DE LICENCE DU LOGICIEL SIMULAMATH

Les présents termes du contrat de licence constituent un contrat entre l'auteur du présent logiciel et vous.

SI VOUS VOUS CONFORMEZ AUX PRÉSENTS TERMES DU CONTRAT DE LICENCE, VOUS AVEZ LES DROITS CI-DESSOUS.

1. INSTALLATION ET DROITS D'UTILISATION:

    a. Stipulations générales. Vous êtes autorisé à utiliser un nombre quelconque de copies du logiciel pour faire des calculs, des graphiques et des simulations.

    b. Copie de sauvegarde. Vous êtes autorisé à effectuer une ou plusieurs copies de sauvegarde du logiciel afin de le réinstaller.

2. EXCLUSIONS DE GARANTIE: VOUS ASSUMEZ TOUS LES RISQUES LIÉS À SON UTILISATION. SIMULAMATH TEAM N'ACCORDE AUCUNE GARANTIE OU CONDITION EXPRESSE. SIMULAMATH TEAM EXCLUT LES GARANTIES IMPLICITES DE QUALITÉ, D'ADÉQUATION À UN USAGE PARTICULIER ET D'ABSENCE DE VIOLATION.

3. COMMENTAIRES: Si vous faites part de vos commentaires concernant le logiciel à SimulaMath Team, vous concédez gracieusement à SimulaMath Team le droit de les utiliser et de les partager. Vous ne donnerez pas d'informations faisant l'objet d'une licence qui impose à SimulaMath Team de concéder sous licence son logiciel ou sa documentation à des tiers du fait que nous y incluons vos commentaires. Ces droits survivent au présent contrat.

4. CHAMP D'APPLICATION DE LA LICENCE: Le présent contrat vous confère certains droits d'utilisation du logiciel. SimulaMath Team se réserve tous les autres droits. Sauf si la réglementation applicable vous confère d'autres droits, nonobstant la présente limitation, vous n'êtes autorisé à utiliser le logiciel qu'en conformité avec les termes du présent contrat.

À cette fin, vous devez vous conformer aux restrictions techniques contenues dans le logiciel qui vous permettent de l'utiliser uniquement d'une certaine façon. Vous n'êtes pas autorisé à :

- reconstituer la logique du logiciel, le decompiler ou le désassembler, ou tenter de quelque autre manière de dériver le code source du logiciel, sauf et uniquement si cela est autorisé par la loi.
- utiliser le logiciel d'une manière contraire à la législation.

6.2 Third-Party Licensing

SimulaMath doit son existence à Python et à plusieurs excellents logiciels libres. Cette section se trouve dans la documentation afin de se conformer aux exigences de licence de ces modules.

6.2.1 Python

PSF LICENSE AGREEMENT FOR PYTHON 3.7.4

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 3.7.4 software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 3.7.4 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001-2019 Python Software Foundation; All Rights Reserved" are retained in Python 3.7.4 alone or in any derivative version prepared by Licensee.
3. In the event Licensee prepares a derivative work that is based on or incorporates Python 3.7.4 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 3.7.4.
4. PSF is making Python 3.7.4 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 3.7.4 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.7.4 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.7.4, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python 3.7.4, Licensee agrees to be bound by the terms and conditions of this License Agreement.

6.2.2 Others

Matplotlib

License agreement for matplotlib 3.1.1 1. This LICENSE AGREEMENT is between the Matplotlib Development Team ("MDT"), and the Individual or Organization ("Licensee") accessing and otherwise using matplotlib software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, MDT hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use matplotlib 3.1.1 alone or in any derivative version, provided, however, that MDT's License Agreement and MDT's notice of copyright, i.e., "Copyright (c) 2012-2013 Matplotlib Development Team; All Rights Reserved" are retained in matplotlib 3.1.1 alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates matplotlib 3.1.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to matplotlib 3.1.1.

4. MDT is making matplotlib 3.1.1 available to Licensee on an "AS IS" basis. MDT MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, MDT MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF MATPLOTLIB 3.1.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. MDT SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF MATPLOTLIB 3.1.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING MATPLOTLIB 3.1.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between MDT and Licensee. This License Agreement does

not grant permission to use MDT trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using matplotlib 3.1.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

Numpy

Scipy

FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Sympy

Copyright (c) 2006-2019 SymPy Development Team All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: a. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. b. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. c. Neither the name of SymPy nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Pandas

BSD 3-Clause License

Copyright (c) 2008-2012, AQR Capital Management, LLC, Lambda Foundry, Inc. and PyData Development Team All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written

permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

WxWidgets and WxPython

`Preambule`

The licencing of the wxWidgets library is intended to protect the wxWidgets library, its developers, and its users, so that the considerable investment it represents is not abused.

Under the terms of the original wxWidgets licences, you as a user are not obliged to distribute wxWidgets source code with your products, if you distribute these products in binary form. However, you are prevented from restricting use of the library in source code form, or denying others the rights to use or distribute wxWidgets library source code in the way intended.

The wxWindows Library License establishes the copyright for the code and related material, and it gives you legal permission to copy, distribute and/or modify the library. It also asserts that no warranty is given by the authors for this or derived code.

The core distribution of the wxWidgets library contains files under two different licences:

- Most files are distributed under the GNU Library General Public License, version 2, with the special exception that you may create and distribute object code versions built from the source code or modified versions of it (even if these modified versions include code under a different licence), and distribute such binaries under your own terms.
- Most core wxWidgets manuals are made available under the “wxWindows Free Documentation License”, which allows you to distribute modified versions of the manuals, such as versions documenting any modifications made by you in your version of the library. However, you may not restrict any third party from reincorporating your changes into the original manuals.

`wxWindows Library Licence`

wxWindows Library Licence, Version 3.1

Copyright (c) 1998-2005 Julian Smart, Robert Roebling et al

Everyone is permitted to copy and distribute verbatim copies of this licence document, but changing it is not allowed.

WXWINDOWS LIBRARY LICENCE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public Licence as published by the Free Software Foundation; either version 2 of the Licence, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public Licence for more details.

You should have received a copy of the GNU Library General Public Licence along with this software, usually in a file named COPYING.LIB. If not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

EXCEPTION NOTICE

1. As a special exception, the copyright holders of this library give permission for additional uses of the text contained in this release of the library as licenced under the wxWindows Library Licence, applying either version 3.1 of the Licence, or (at your option) any later version of the Licence as published by the copyright holders of version 3.1 of the Licence document.

2. The exception is that you may use, copy, link, modify and distribute under your own terms, binary object code versions of works based on the Library.

3. If you copy code from files distributed under the terms of the GNU General Public Licence or the GNU Library General Public Licence into a copy of this library, as this licence permits, the exception does not apply to the code that you add in this way. To avoid misleading anyone as to the status of such modified files, you must delete this exception notice from such code and/or adjust the licensing conditions notice accordingly.

4. If you write modifications of your own for this library, it is your choice whether to permit this exception to apply to your modifications. If you do not wish that, you must delete the exception notice from such code and/or adjust the licensing conditions notice accordingly.

Seaborn

Copyright (c) 2012-2019, Michael L. Waskom All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Chapter 7

Indexes and tables

- genindex
- search

Python Module Index

S

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z