SimulaMath Documentation

Version 1.1.beta1

SimulaMath Developers

sept. 30, 2021

Table des matières

Chapitre 1

Introduction

SimulaMath est un logiciel de calcul scientifique, dédié à l'apprentissage, l'enseignement et la recherche en mathématiques. Il est développé avec le langage Python en mettant l'accent sur la simplicité (facilité d'utilisation), à travers une interface utilisateur graphique (GUI). Il couvre de nombreuses domaines des mathématiques, notamment l'algèbre linéaire, l'analyse, la théorie des nombres, la statistique descriptive univariée et bivarée, la statistique inférentielle, les distributions de probabilité, les graphiques 2D et 3D, les polynômes multivariés et les bases de Groebner, les courbes elliptiques, les codes linéaires et les corps finis. Il fonctionne sur Windows, Mac OSX et de nombreuses plateformes Linux.

1.1 Objectifs

SimulaMath est conçu dans le but de faciliter l'enseignement et l'apprentissage et la recherche en mathématiques du collège à l'université et d'encourager l'utilisation des TIC dans l'éducation.

1.2 Points forts de SimulaMath

— **Simplicité** : SimulaMath possède une interface simple et intuitive, qui vous permet de produire des résultats et de créer des graphiques (2D et 3D) de qualité avec un minimum d'effort.

Note : Règle d'or dans SimulaMath : les données d'entrée et de sortie doivent converger vers la syntaxe mathématique (notation).

— **Python** : Python est un langage de programmation de haut niveau, facile à apprendre, dynamiquement typé et maintenu par une très large communauté. Il est devenu le choix de la plupart des data scientists aujourd'hui. SimulaMath est développé avec Python et son langage de programmation en est dérivé comme le logiciel SageMath. L'objectif n'était pas de réinventer la roue en créant un nouveau langage de programmation comme

c'est le cas pour la plupart des logiciels scientifiques mais de simplifier ce qui existe déjà.
Si vous savez programmer en Python, vous pouvez utiliser la plupart de ses fonctions et modules scientifiques comme **Numpy**, **Scipy**, **Sympy**, **Pandas** etc. pour l'interface cliquable et de programmation.

— **Programmation** : Un nouveau langage dérivé de Python est ajouté au logiciel SimulaMath. Cela signifie que 99% d'un code Python valide est également valide sur SimulaMath. Une nouvelle syntaxe très proche de la notation mathématique a également été ajoutée.
— **Deux types d'interfaces** : SimulaMath dispose de deux types d'interfaces : l'interface cliquable et l'interface de programmation. L'interface cliquable permet d'obtenir des résultats de qualité sans nécessairement savoir programmer et l'interface de programmation s'adresse à tous (programmeurs et ceux qui veulent apprendre la programmation).
— **Graphiques de qualité** : SimulaMath possède une interface très puissante et intuitive pour les graphiques en deux et trois dimensions. Vous pouvez enregistrer vos graphiques dans de nombreux formats : PNG, PDF, PGF, JPEG, SVG, etc.
— **Multi-plateforme** : SimulaMath fonctionne sous Windows, Mac OS X et de nombreuses plateformes Linux (par exemple Ubuntu 16+).
— **Documentation** : la documentation est disponible en HTML et en PDF.
— **Multi-domaines** : SimulaMath n'est pas conçu uniquement pour un domaine spécifique des mathématiques. On peut faire du calcul, de l'algèbre linéaire, des statistiques, des probabilités, des courbes elliptiques, des codes linéaires, etc.

1.3 A propos des Développeurs

La première version de SimulaMath (version *1.0*), publiée en 2019, a été conçue et développée par Michel Seck, Docteur en algèbre et cryptographie à l'Université Cheikh Anta Diop de Dakar (Sénégal).

Depuis 2020, une équipe internationale (voir la page web de l'équipe) a rejoint le projet. Grâce à cette équipe, de nombreuses nouvelles fonctionnalités ont été ajoutées à cette version.

1.4 Nouveautés dans SimulaMath `1.1`

— La programmation : SimulaMath dispose d'un langage de programmation très simple et puissant, dérivé de Python.
— Statistique inférentielle : Estimation par intervalle de confiance et les tests d'hypothèse ont été ajoutés.
— Géométrie sur le plan : pour les graphiques 2d, vous pouvez désormais ajouter des textes, des images et divers objets géométriques 2D : Points, Droites, Démi-droites, Segments, Cercles, Arcs, Polygones, Droites parallèles, Droites perpendiculaires, Vecteurs, Angles, Bissectrice, Ellipses, Paraboles, Hyperboles, Rotation, Homothétie, Translation, Réflexion par rapport à un point et une droite, Surfaces, Barycentre, etc.
— Deux langues : SimulaMath est désormais disponible en anglais et en français.

— Choix d'un niveau : Vous pouvez choisir entre trois niveaux : collège, lycée et université.

Chapitre 2

Installation

Le logiciel SimulaMath peut être installé sur Windows (32 et 64 bits) Mac OsX (64 bits) et la plupart des distributions Linux.

2.1 Sous Windows

Pour installer SimulaMath sous Windows, il suffit de lancer le programme d'installation (**SimulaMath-vXXX-Windows-x64-XXX.exe** si vous avez une version 64 bits de Windows ou **SimulaMath-vXXX-Windows-x86-XXX.exe** si vous avez une version 32 bits) en double-cliquant dessus dans l'explorateur Windows. Le programme d'installation automatique démarre et vous guide tout au long du processus d'installation ; suivez simplement les instructions à l'écran. SimulaMath sera installé pour tous les utilisateurs (si vous êtes l'administrateur), ou pour vous seul (si vous ne l'êtes pas). Ceci vous permet d'installer SimulaMath sans aucun droit d'administrateur.

2.2 Sous Mac OS X

Pour installer SimulaMath sur un Apple Macbook, double-cliquez sur le fichier **SimulaMath-vXXX-macosx-XXX.pkg**, le processus d'installation commencera. Suivez ensuite les instructions jusqu'à ce que votre installation soit terminée.

SimulaMath n'est pas une application Apple signée. Par conséquent, Gatekeeper (si vous avez OSX Mountain Lion ou plus ancien) peut se plaindre à ce sujet. Ceci est normal. Si l'installation est bloquée, cliquez sur le bouton Ouvrir quand même dans le volet général des préférences Sécurité et confidentialité. Ce bouton est disponible pendant environ une heure après que vous ayez essayé d'ouvrir SimulaMath (pour plus de détails voir Guide Mac).

2.3 Sous Linux

Pour installer SimulaMath sous Linux, copiez (ou téléchargez) le fichier **SimulaMath-vXXX-linux-XXX.run** sur votre ordinateur, puis ouvrez votre terminal et placez-vous dans le dossier qui contient le fichier **SimulaMath-vXXX-linux-XXX.run**. Habituellement, ce fichier n'a pas les permissions nécessaires pour s'exécuter normalement. Pour donner au fichier les droits d'exécution, faites l'une des choses suivantes :

— Sur la ligne de commande, tapez **chmod +x SimulaMath-vXXX-linux-XXX.run**.

— Dans le gestionnaire de fichiers, faites un clic droit sur le fichier **SimulaMath-vXXX-linux-XXX.run**, sélectionnez « Permissions », puis cochez la case « Autoriser le fichier à s'exécuter en tant que programme ».

Et enfin exécutez **./SimulaMath-vXXX-linux-XXX.run** dans votre terminal. Il se peut que l'on vous demande votre mot de passe. Si c'est le cas, il suffit de le saisir et de valider.

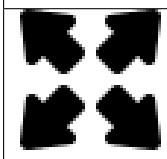
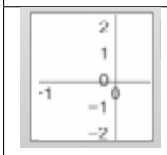
Chapitre 3

Interface utilisateur

3.1 Menus

Liste des menus	Liste des sous-menus
Page d'accueil	Page d'accueil : liste des domaines disponibles dans SimulaMath
Analyse & Algèbre	— Résolution d'équations : pour la résolution d'équations, d'inéquations, de systèmes d'équations, d'équations différentielles, de suites récurrentes. — Algèbre linéaire : opérations sur les matrices, réduction des matrices, etc. — Opérations sur les nombres : opérations de base, nombres complexes, etc. — Opérations sur les fonctions et les suites : calcul des dérivées, limites, intégrales, suites numériques, etc. — Opérations sur les corps finis et les polynômes mod p. — Opérations sur les bases de Groebner et les polynômes multivariés : détermination d'une base de Groebner (réduite ou non), calcul de S-polynômes, idéaux, forme normale ...
Graphiques 2D & 3D	— Graphiques 2D : Fonctions d'une variable x, fonctions paramétriques, fonctions implicites, etc — Graphiques 3D : Fonctions à deux variables x, y et fonctions paramétriques. — Diagrammes : diagrammes en barres 2D et 3D, histogrammes 2D et 3D, courbes 2D, diagrammes de dispersion 2D, camemberts, boîte à moustaches, diagramme de violon, Displot etc.
Probabilités	— Calcul de probabilités : distributions de probabilités discrètes et continues. — Approximation des lois : convergence en lois.
Statistique	— Statistique descriptive univariée : caractéristiques des tendances centrales, caractéristiques des dispersions et des formes. — Statistiques descriptives à deux variables : chi-deux de contingence, covariance, T de Stchuprow, etc. — Statistique inférentielle : estimation par intervalle de confiance et tests d'hypothèses. — Tableur : ouverture et sauvegarde de fichiers aux formats Excel, CSV et JSON, manipulation des données du tableur
Cryptographie	— Réseaux euclidiens : simulation d'un réseau euclidien en

3.2 Barre d'outils

Image	Destination	Description
	Zone 2D et 3D	Enregistrez la figure en png, pgf, pdf, eps, jpeg, ep, etc
	Zone 2D	Retourner
	Zone 2D	Avancer
	Zone 2D	Déplacez la figure
	Zone 2D	zoomer sur la figure
	Zone 2D	Retour Complet
	Zone 2D	position et affichage des axes de la figure (axes qui se coupent en zéro, axes aux extrémités, sans axes, etc.)
	Zone 2D et 3D	plus de paramètres (réglages, grille, axes, marqueur orthonormal, thème du graphique, ajout d'images, de points, de segments, de cercles, d'annotations, de textes, etc.)
	Toutes les pages	Aide

Chapitre 4

Interface cliquable

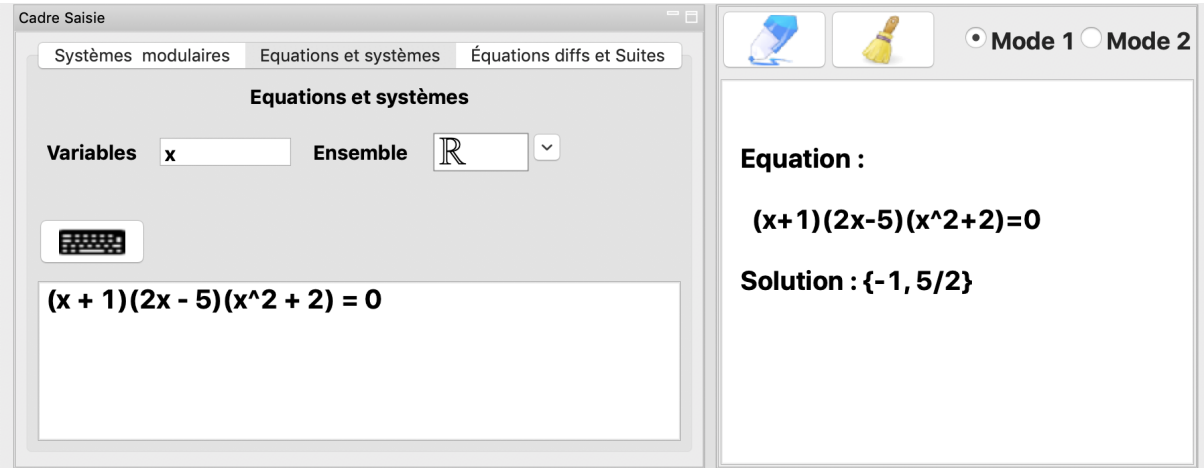
4.1 Résolution d'équations, d'inéquations et de systèmes.

4.1.1 Equations et inéquations

Pour résoudre une équation (ou une inéquation) avec **SimulaMath** :

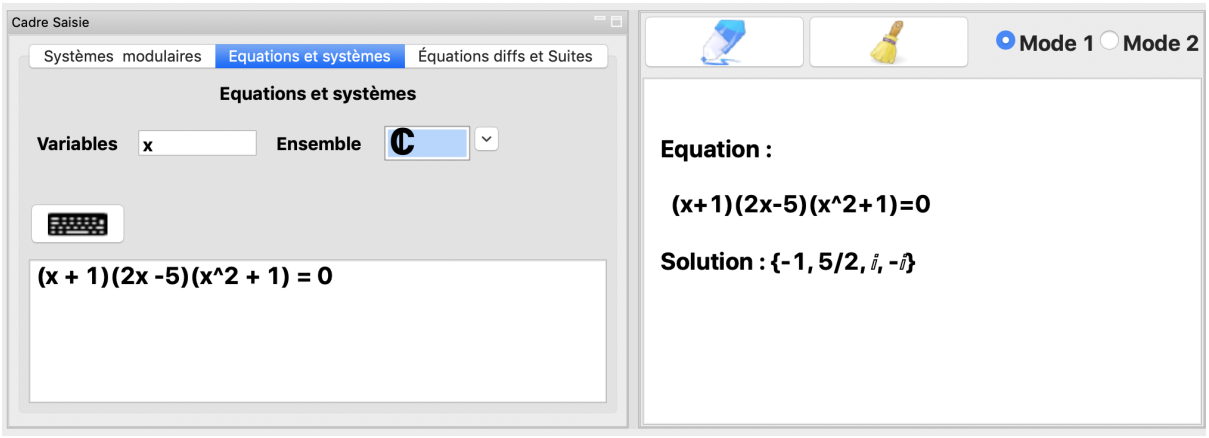
- Saisir l'équation (ou l'inéquation) dans le panneau de gauche,
- Spécifier la ou les variables dans la zone des variables,
- Puis appuyez sur le bouton d'affichage.
- La solution de l'équation $(x+1)(2x-5)(x^2+1)=0$ sur $\mathbb{R}$.

Note : Vous devez spécifier les variables dans la zone des variables



Note : Vous pouvez également résoudre des équations dans un ensemble donné.

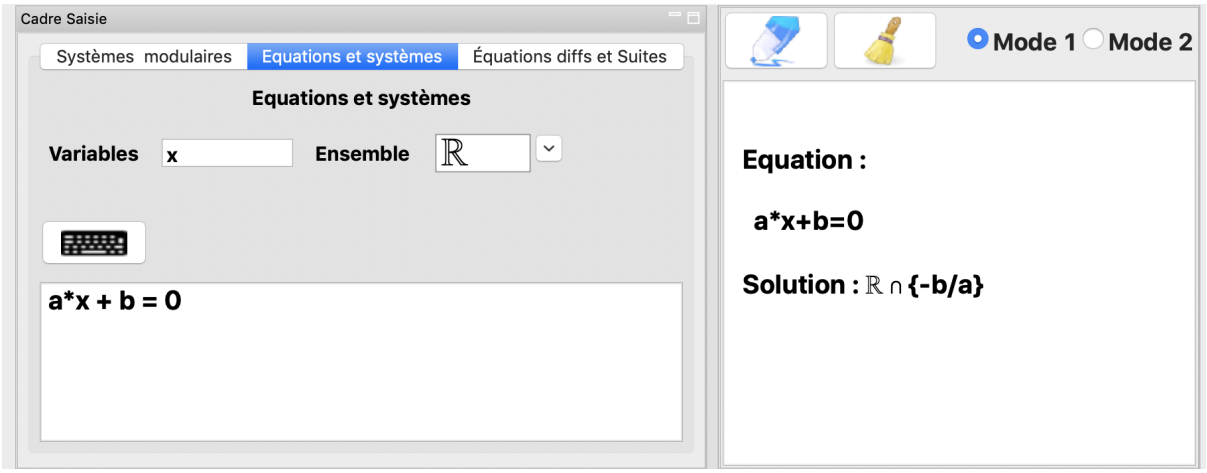
- Résolution de l'équation $(x+1)(2x-5)(x^2+1)=0$ sur $\mathbb{C}$



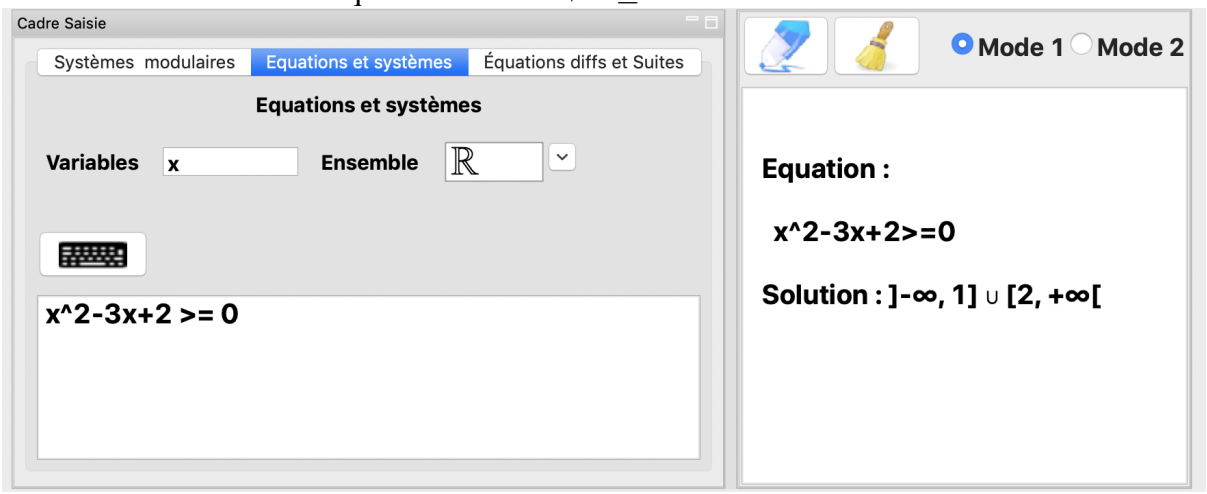
— Résolution avec des paramètres

Note : Si l'équation contient des variables qui ne sont pas définies dans la zone des variables, elles sont considérées comme des paramètres.

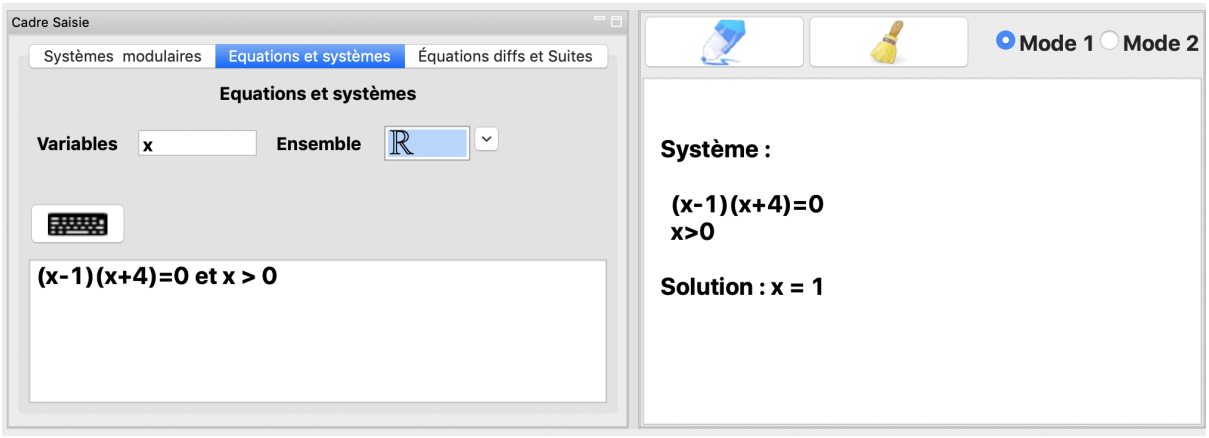
Résolvons l'équation $ax + b = 0$ où $x \in \mathbb{R}$.



— La solution de l'inéquation $x^2 - 3x + 2 \geq 0$ dans $\mathbb{R}$



— La résolution dans $\mathbb{R}$ de $(x - 1)(x + 4) = 0$ et $x > 0$.



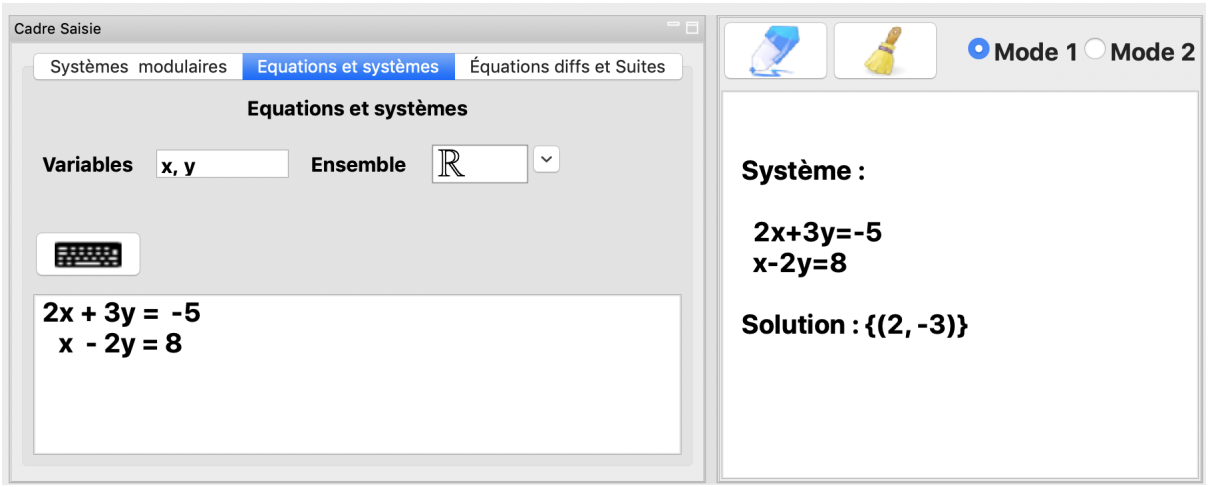
4.1.2 Systèmes linéaires et non linéaires

Pour entrer un système (ou une inéquation),

1. Saisissez d'abord la première équation (ou inéquation), puis appuyez sur la touche **ENTREE**;
2. Saisissez ensuite la deuxième équation (ou inéquation) et appuyez sur **ENTREE**;
3. Et ainsi de suite jusqu'à la dernière équation (ou inéquation).

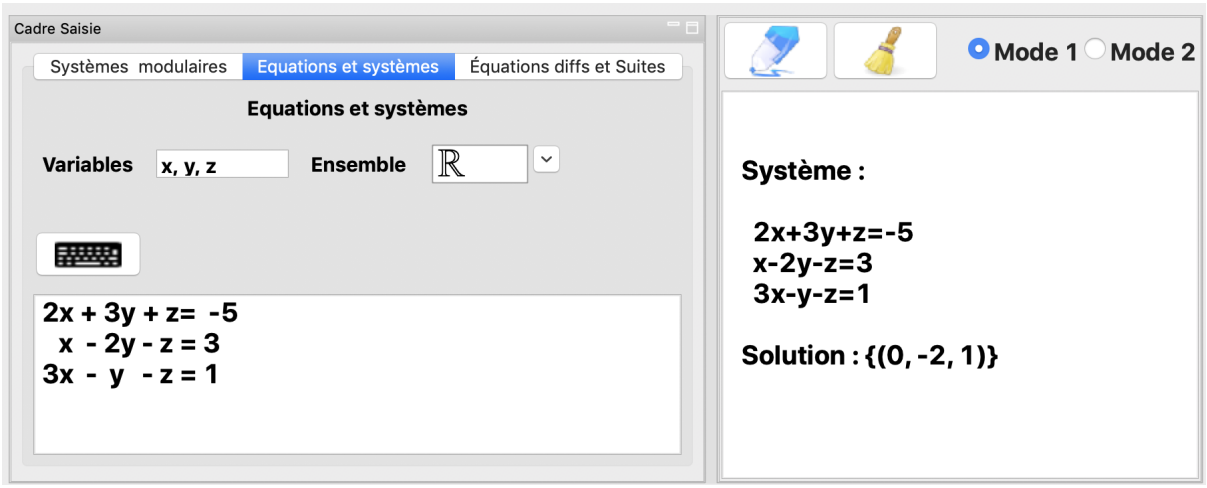
— La solution dans $\mathbb{R}^2$ du système :

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



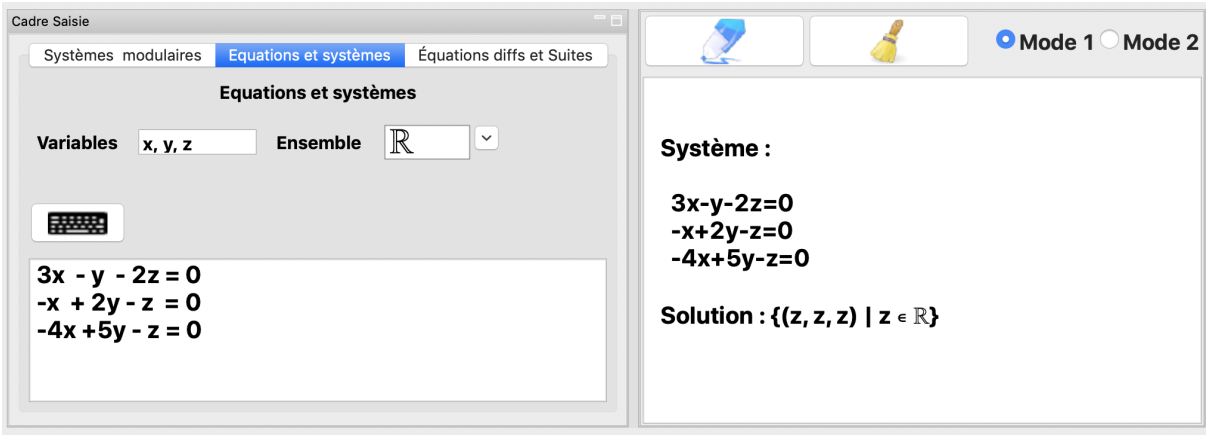
— La résolution sur $\mathbb{R}^3$ du système :

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



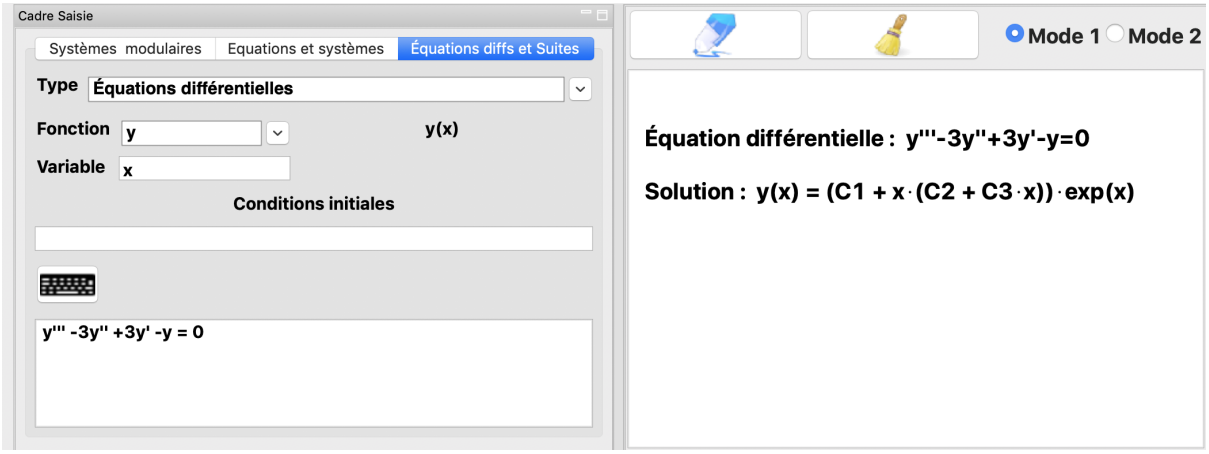
— La résolution sur $\mathbb{R}^3$ du système :

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

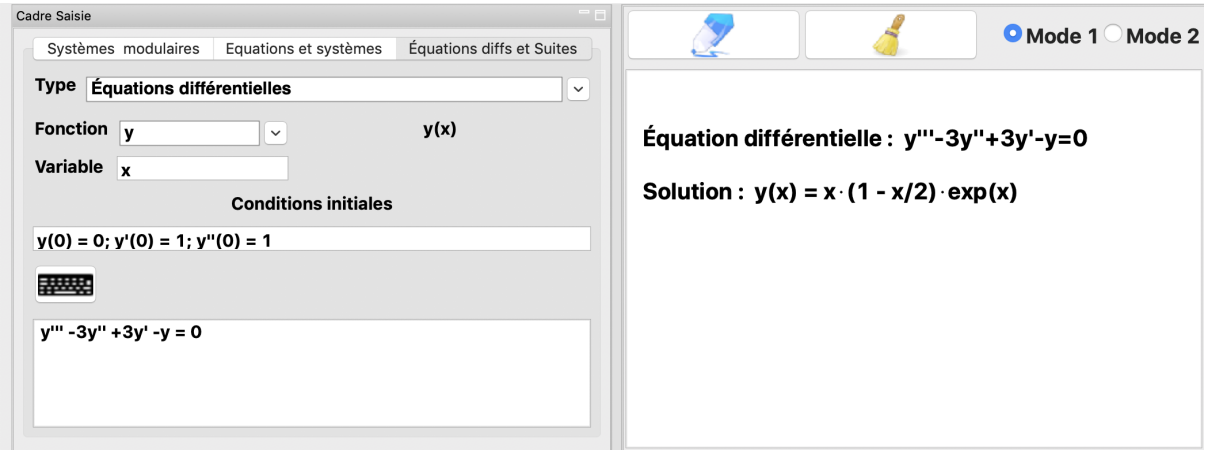


4.1.3 Équations différentielles

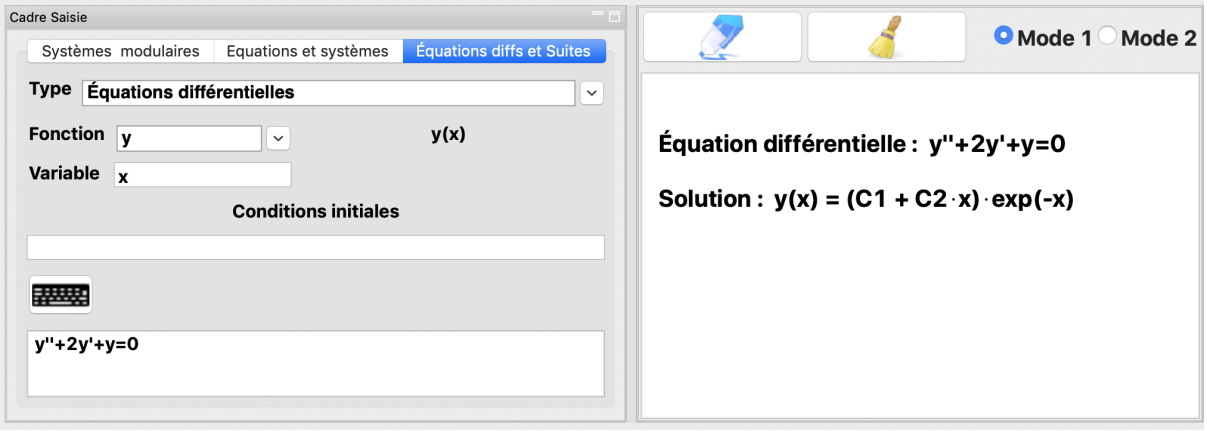
— La résolution de l'équation différentielle $y''' - 3y'' + 3y' - y = 0$



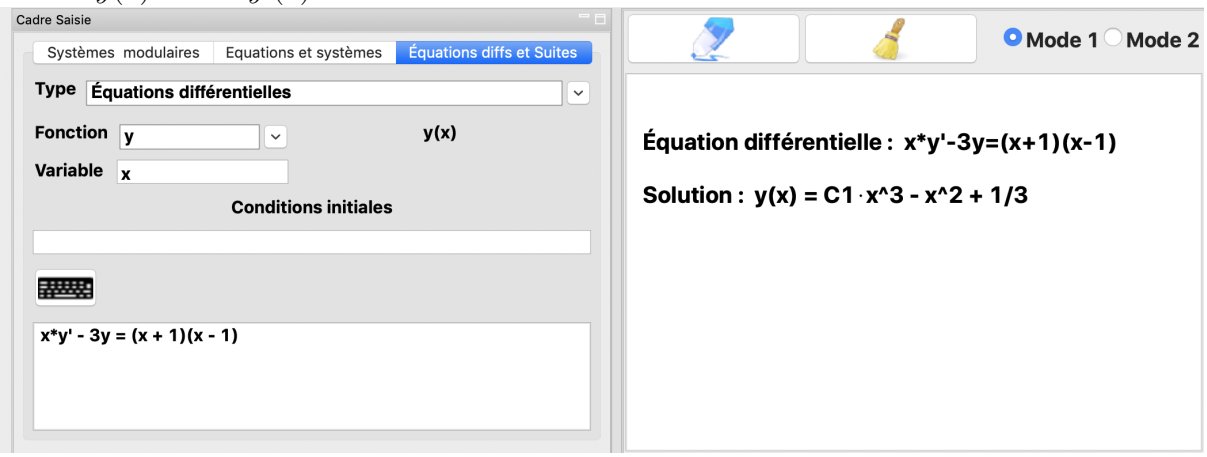
— La résolution de l'équation différentielle $y''' - 3y'' + 3y' - y = 0$ avec les conditions initiales $y(0) = 0; y'(0) = 1$ et $y''(0) = 1$.



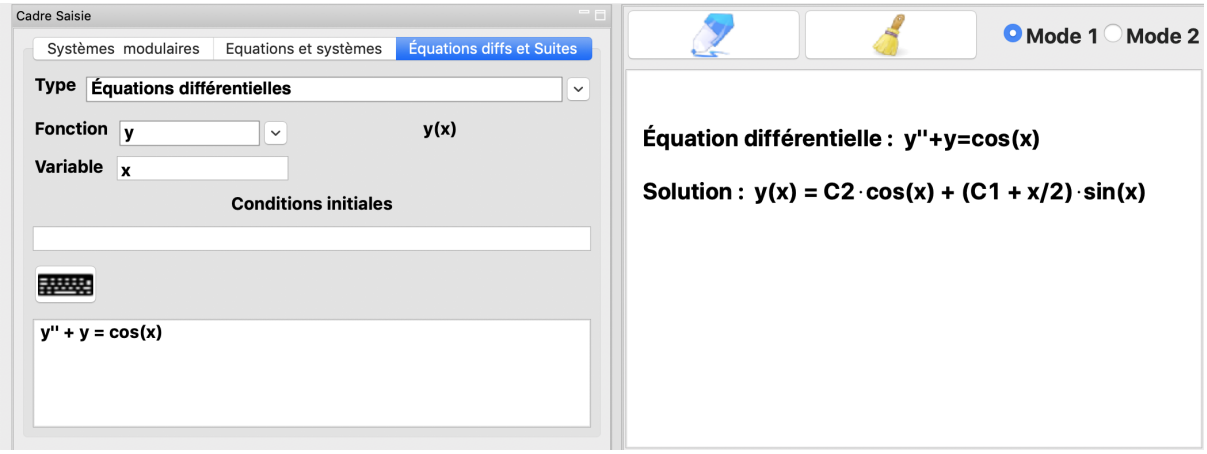
— La résolution de l'équation différentielle $y'' + 2y' + y = 0$.



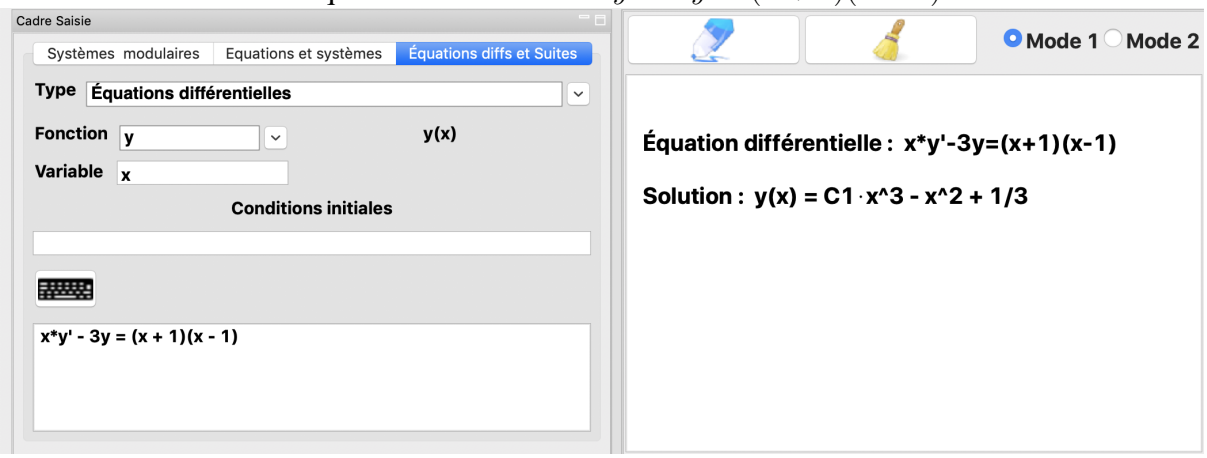
— La solution de l'équation différentielle $y'' + 2y' + y = 0$ avec les conditions initiales $y(0) = 1$ et $y'(0) = 2$.



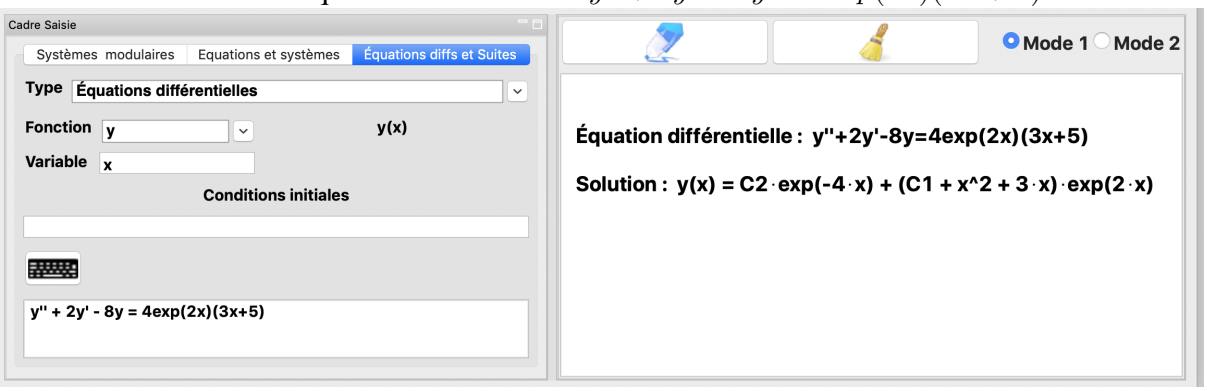
— La solution de l'équation différentielle $y'' + y = \cos(x)$



— La solution de l'équation différentielle $xy' - 3y = (x+1)(x-3)$



— Résolution de l'équation différentielle $y'' + 2y' - 8y = 4exp(2x)(3x+5)$

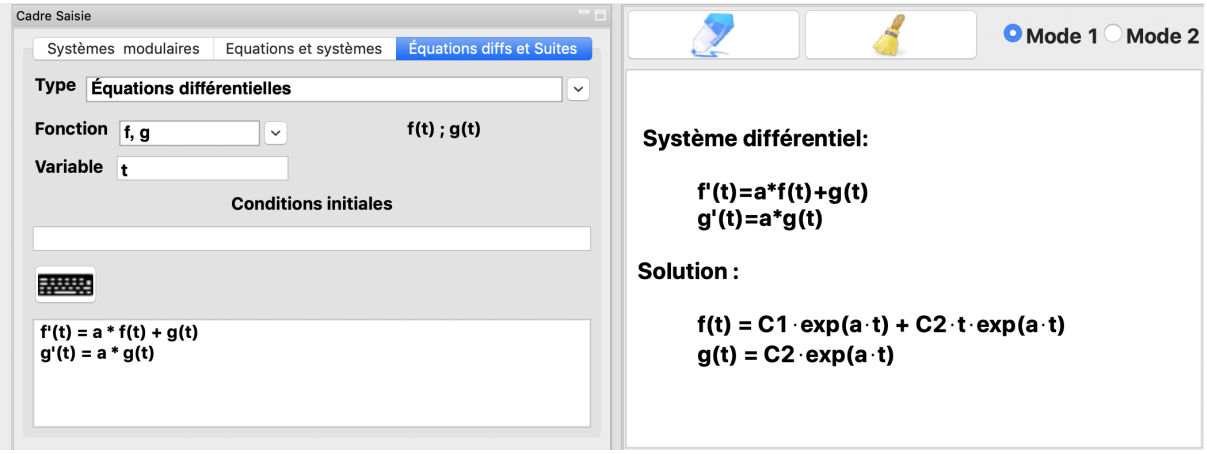


4.1.4 Systèmes différentiels

— La résolution du système différentiel

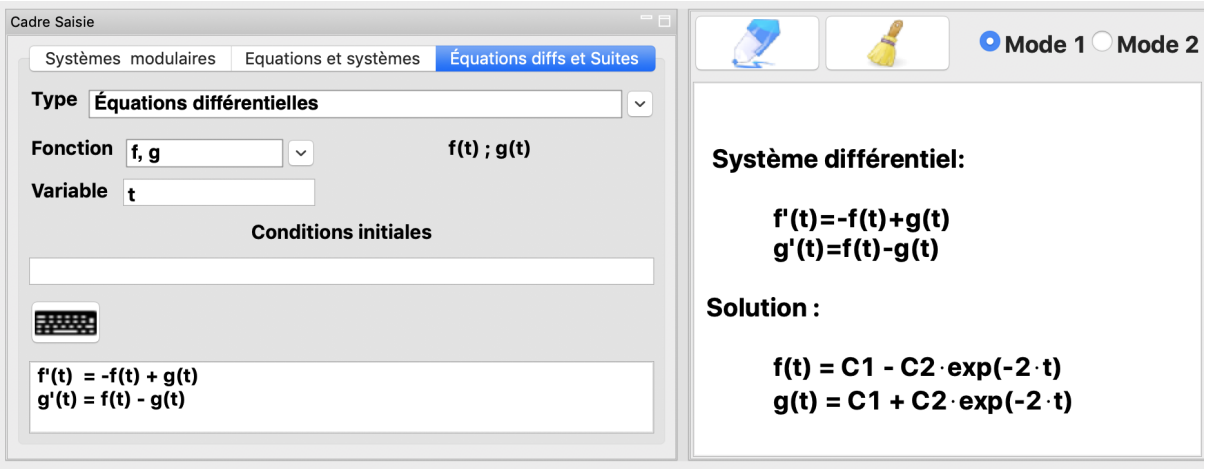
$$\begin{cases} f'(t) &= af(t) + g(t) \\ g'(t) &= ag(t) \end{cases}$$

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



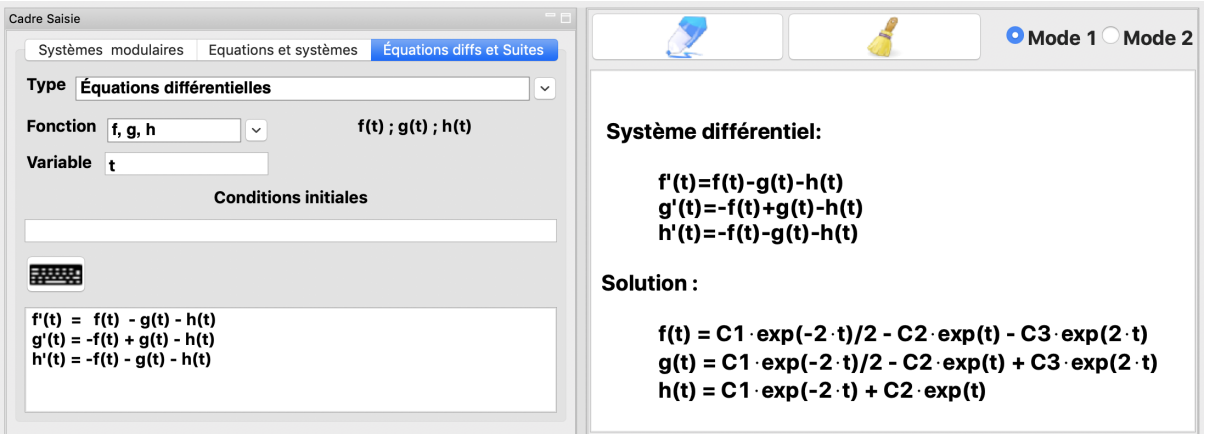
— La résolution du système différentiel

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



— La résolution du système différentiel

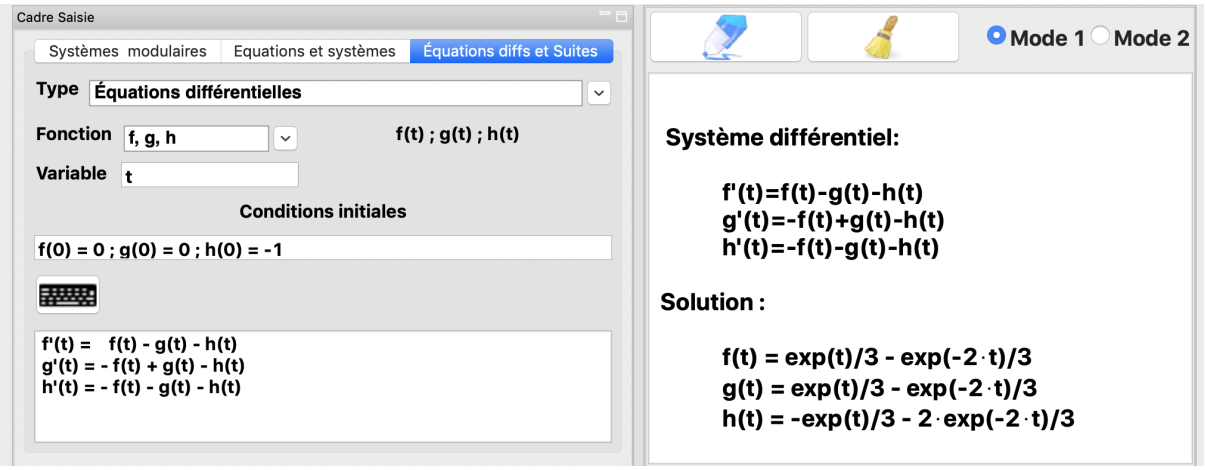
$$\begin{cases} f'(t) &= f(t) - g(t) - h(t) \\ g'(t) &= -f(t) + g(t) - h(t) \\ h'(t) &= -f(t) - g(t) + h(t) \end{cases}$$



— La résolution du système différentiel :

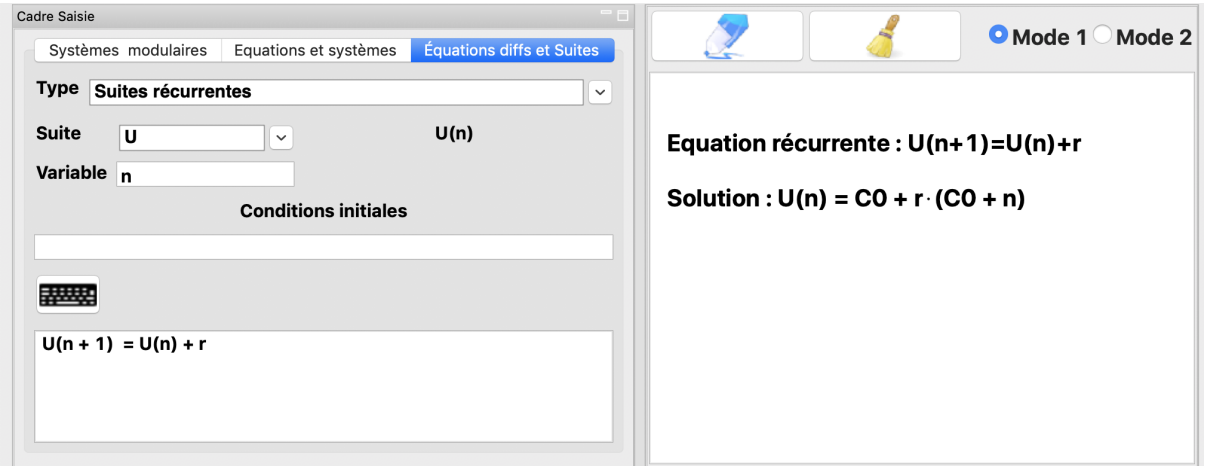
$$\begin{cases} f'(t) &= f(t) - g(t) - h(t) \\ g'(t) &= -f(t) + g(t) - h(t) \\ h'(t) &= -f(t) - g(t) + h(t) \end{cases}$$

avec les conditions initiales $f(0) = 0; g(0) = 1$ et $h(0) = -1$.

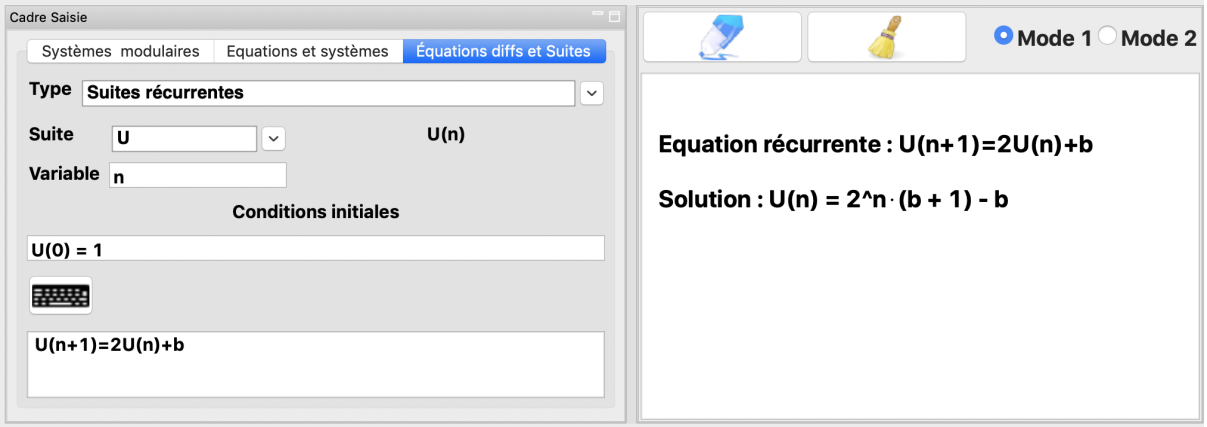


4.1.5 Suites récurrentes linéaires

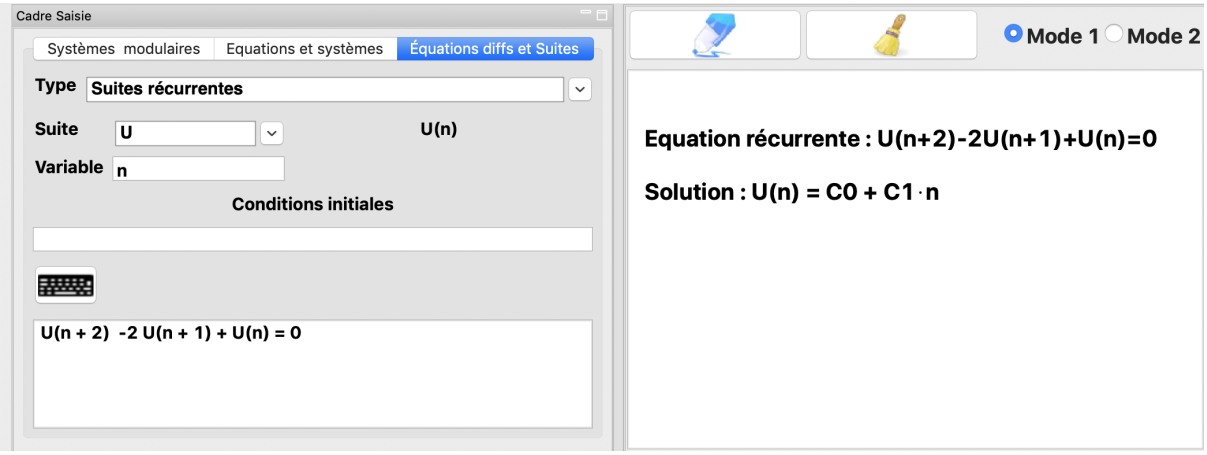
— La solution de l'équation récurrente $U(n+1) = U(n) + r$ avec $r \in \mathbb{R}$.



— La solution de l'équation récurrente $U(n+1) = 2U(n) + b$ avec $b \in \mathbb{R}$ avec $U(0) = 1$.



— La solution de l'équation récurrente $U(n+2) - 2U(n+1) + U(n) = 0$.

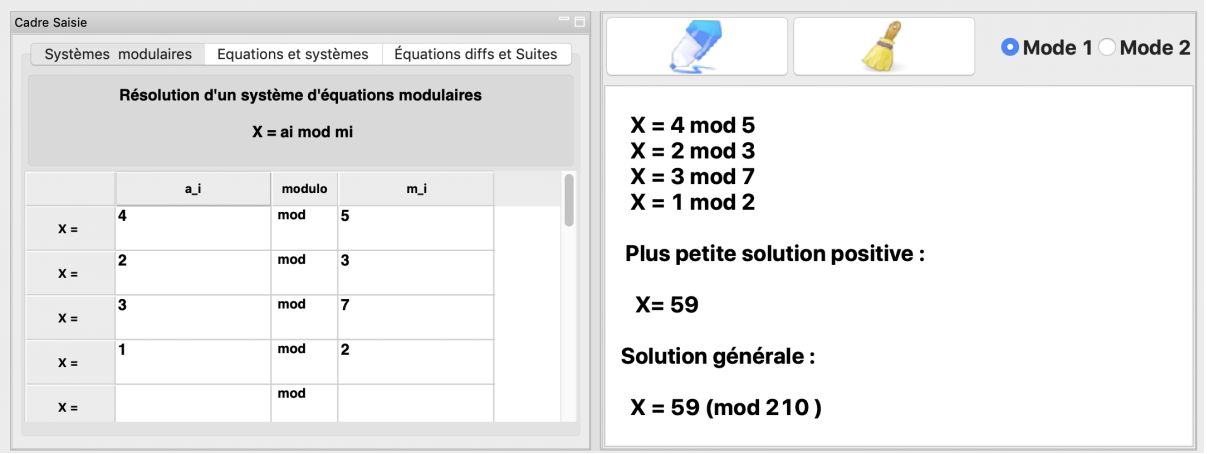


4.1.6 Systèmes modulaires

La résolution de certains systèmes modulaires peut être effectuée en utilisant le théorème du reste chinois.

— Exemple : résoudre le système modulaire

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



4.2 Statistique descriptive univariée

Pour la statistique descriptive univariée, les caractéristiques suivantes peuvent être déterminées

— la moyenne
— la moyenne quadratique
— la moyenne géométrique
— la moyenne harmonique
— la variance
— la variance empirique

— le moment d'ordre alpha
— l'écart type
— l'écart type empirique
— le mode (et la la classe modale dans le cas d'un caractère continu)
— la médiane (et la la classe médiane dans le cas d'un caractère continu)
— les quartiles Q1 et Q3
— l'écart moyen absolu
— l'écart médian absolu
— l'écart inter-quartiles
— le coefficient de variation
— le coefficient de dissymétrie
— Le coefficient d'asymétrie de Fisher
— Le coefficient d'asymétrie de Yule
— Le coefficient d'asymétrie de Pearson
— Le coefficient d'aplatissement de Pearson
— Le coefficient d'aplatissement de Fisher

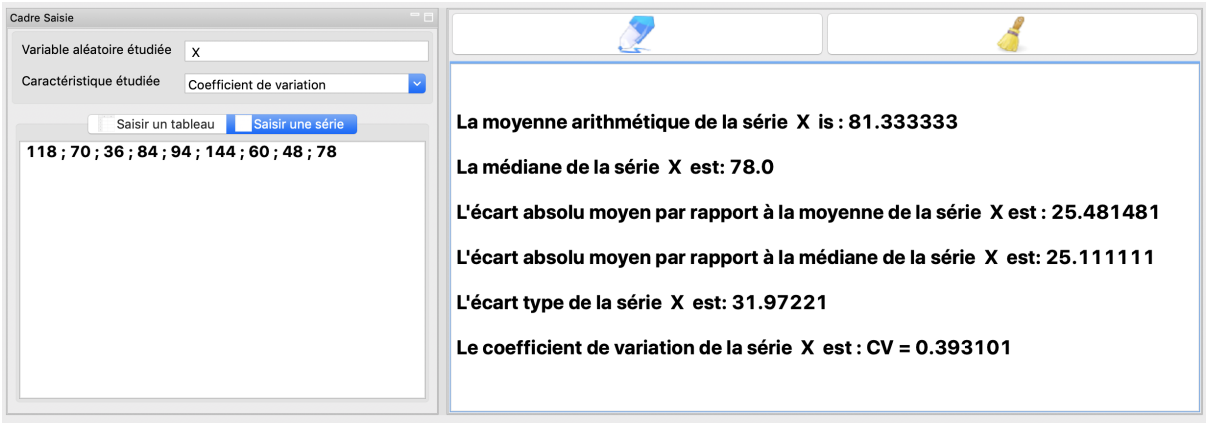
La saisie de données est très simple comme l'illustrent les exemples ci_dessous.

4.2.1 Caractères discrets

Exemple 1 : Série statistique

La série ci-après représente la superficie (en m^2) des neuf appartements d'une résidence : 118 ; 70 ; 36 ; 84 ; 94 ; 144 ; 60 ; 48 ; 78

1. Déterminer la moyenne arithmétique et la médiane de cette distribution.
2. Calculer les caractéristiques de dispersion suivantes : écart absolus moyen par rapport à la moyenne et à la médiane, l'écart type et le coefficient de variation.



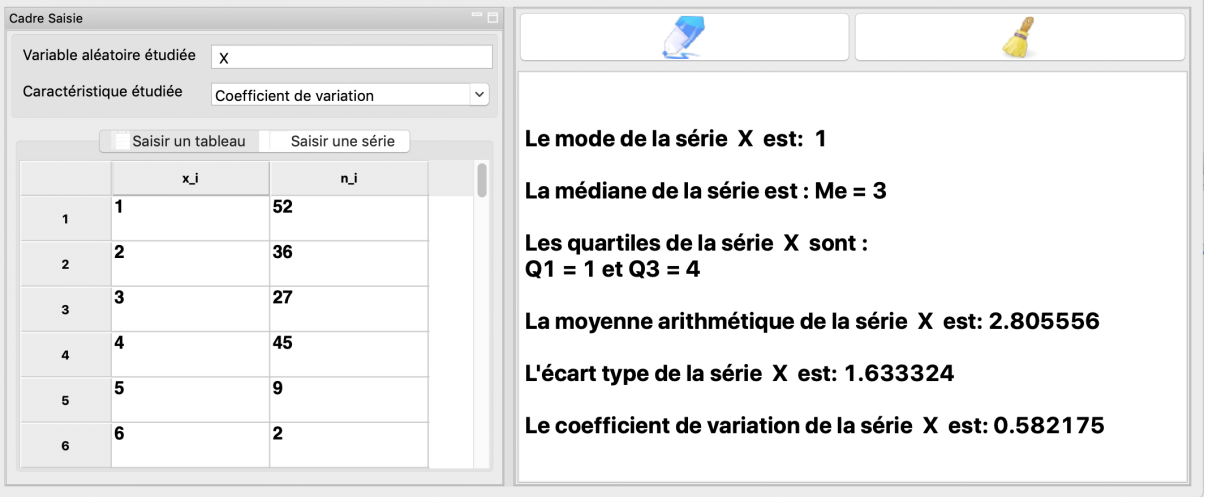
Exemple 2 : Série statistique sous forme de tableau.

Dans une librairie, 180 auteurs ont été répartis suivant le nombre de manuels qu'ils ont écrits.

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

1. Déterminer le mode, la médiane et les quartiles Q_1 et Q_3.

2. Calculer la moyenne arithmétique, l'écart type et le coefficient de variation de cette série.



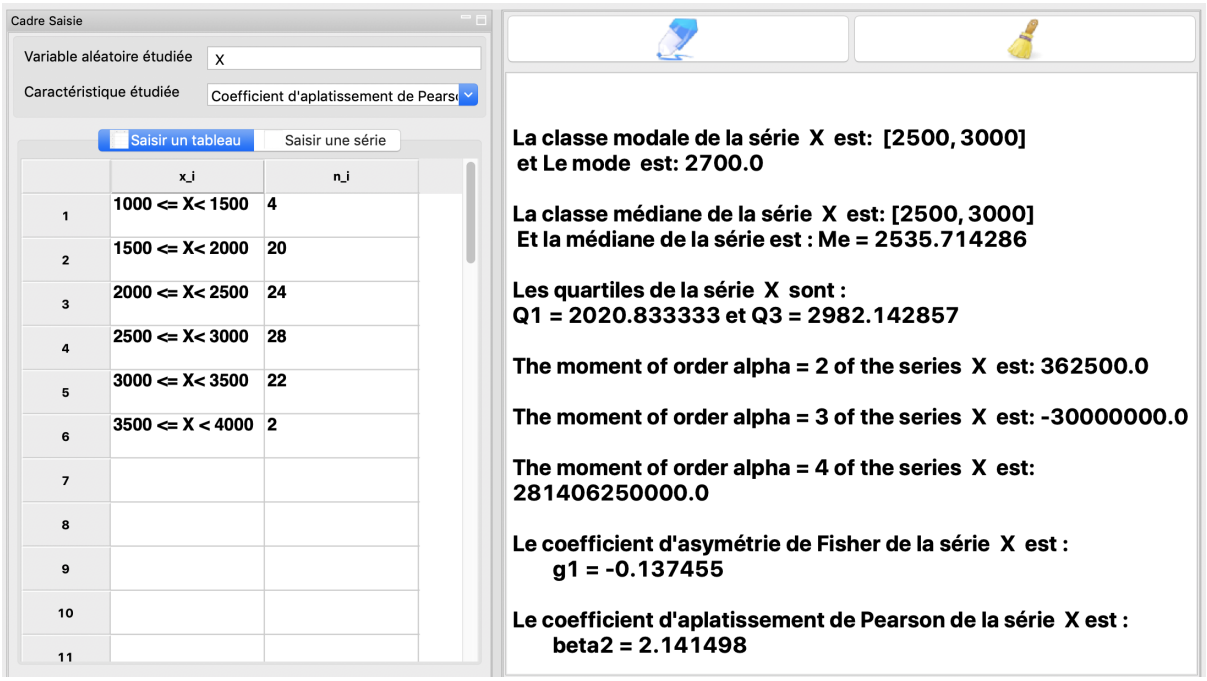
4.2.2 Caractères Continus

Exemple 3 : Données groupées en classes d'amplitudes égales

Le tableau ci-dessous donne la distribution du nombre de commandes en fonction du montant des commandes X, pour les six derniers mois du GIE LIGGEEY.

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

1. Déterminer la classe modale, le mode, la médiane et les quartiles Q_1 et Q_3.
2. Calculer les moments centrés d'ordre $2, 3$ et 4 de cette distribution.
3. Calculer le coefficient d'asymétrie de Fisher et le coefficient d'aplatissement de Pearson.

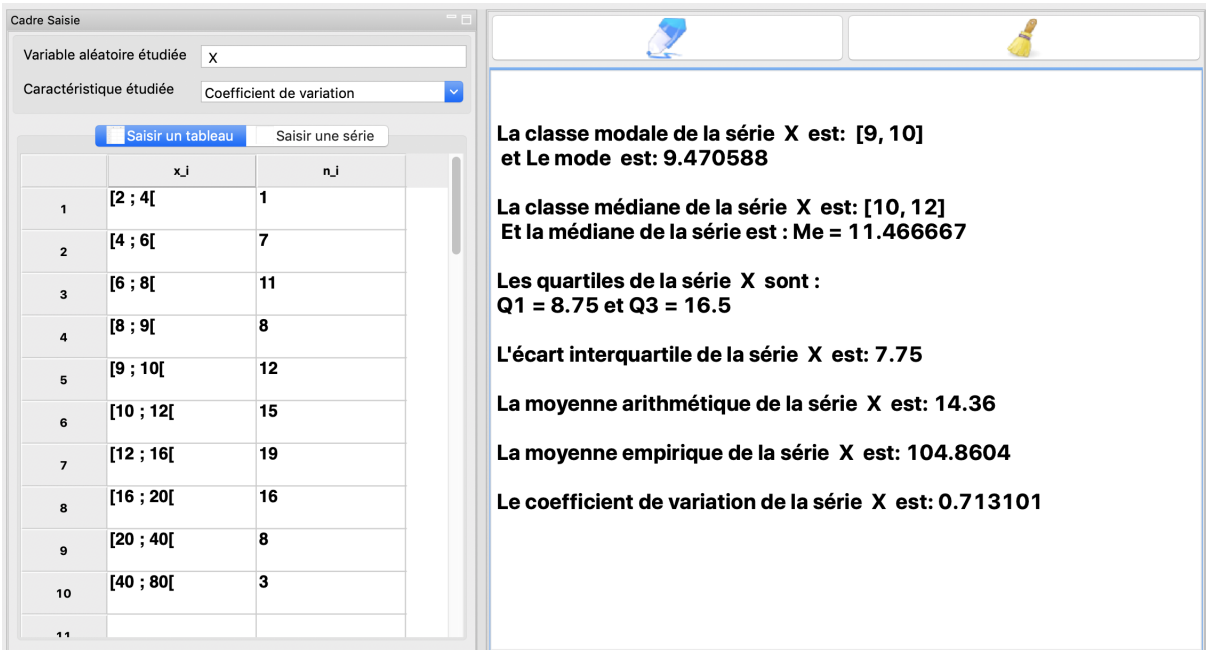


Exemple 4 : Données groupées en classes d'amplitudes inégales

Le tableau ci-après fournit la répartition en pourcentage des habitants d'une commune, selon le montant annuel de leurs impôts locaux (en milliers de francs).

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

1. Déterminer la classe modale, le mode, la médiane et les quartiles Q_1 et Q_3.
2. Calculer la moyenne arithmétique de cette série, l'écart interquartile, la variance et le coefficient de variation.



4.3 Statistique descriptive bivariée

Pour la statistique descriptive à deux variables, on peut déterminer les caractéristiques suivantes.

- la covariance
- le coefficient de corrélation linéaire
- le coefficient de détermination
- la distance du khi-deux
- Phi-carré de Cramer
- T de Tschuprow
- la droite de régression de Y en X

La saisie des données est très simple comme le montrent les exemples ci-dessous.

4.3.1 Tableau de contingence

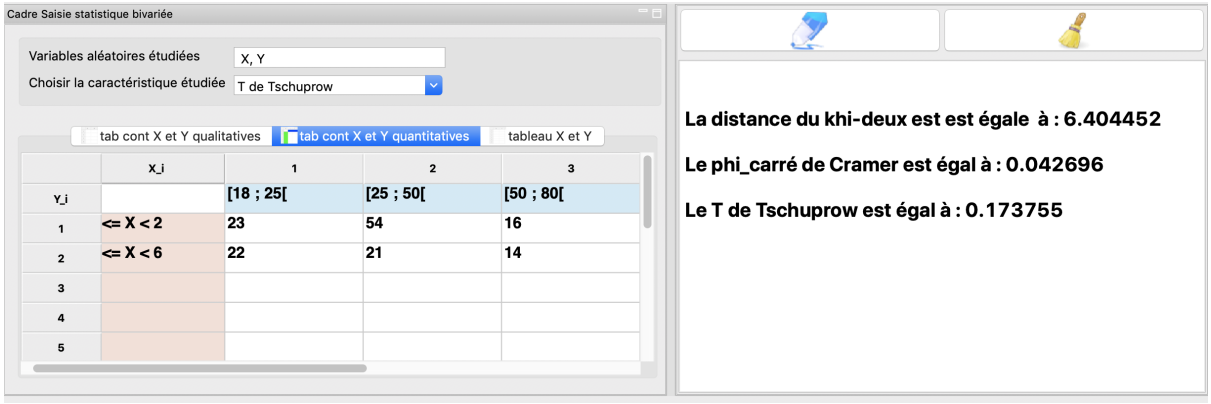
Exemple 1 : Tableau de contingence avec X et Y **quantitatifs**

Une société d'assurance a réalisé à partir de son fichier de clients une enquête par sondage pour connaitre la répartition du nombre d'accidents de la route **(X)** selon l'âge des assurés **(Y)**. Le résultat de cette enquête est donné par le tableau suivant.

		Âge (en années)	
Nombre d'accidents	[18 ; 25[	[25 ; 50[	[50 ; 80[
de 0 à 2	23	54	16
de 3 à 6 ans	22	21	14

1. Calculer le khi-Deux de contingence.

2. En déduire les valeurs de Φ^2 de Cramer et le T de Tschuprow.

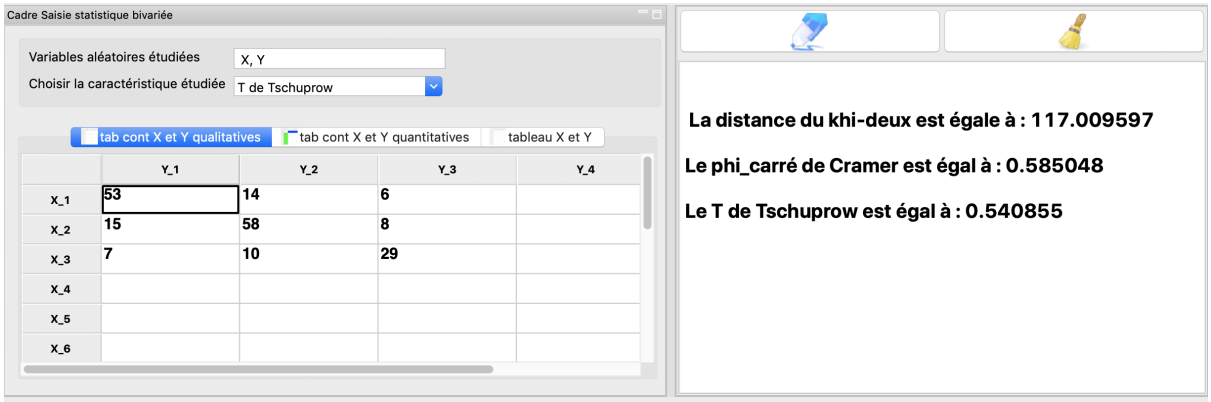


** Exemple 2** : Tableau de contingence avec X et Y **qualitatifs**

Sur un échantillon de 200 ménages choisis au hasard, on a étudié la propension moyenne à épargner **(variable Y)** en fonction de revenu disponible **(variable X)**. Pour la variable **X**, on a distingué 3classes (faibles revenus, intermédiaires, revenus élevés). De même les taux d'épargne ont été classés en 3 niveaux (faibles taux, taux intermédiaires, taux élevés). Les résultats sont présentés dans la table de contingence :

	Y_1 = taux faibles	Y_2 = taux intermédiaires	Y_3 = taux élevés
X_1 = revenus faibles	53	14	6
X_3 = revenus intermédiaires	15	58	8
X_3 = revenus élevés	7	10	29

1. Calculer le khi-deux de contingence.
2. En déduire les valeurs du Φ^2 de Cramer et du T de Tschuprow.

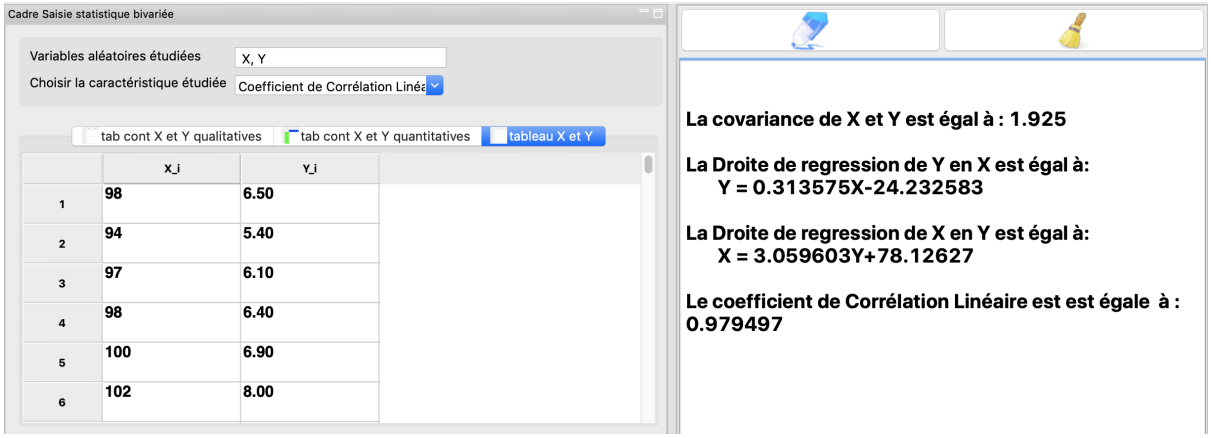


4.3.2 Tableau simple

La société anonyme par action R augmente son capital. On a relevé pendant 6 jours le cours en bourse de l'action (X) et celui du droit de souscription (Y).

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

1. Calculer la covariance *Cov(X, Y)*.
2. Établir les équations de la droite de régression de *Y* en *X*.
3. Calculer le coefficient de corrélation linéaire entre les variables *X* et *Y*.



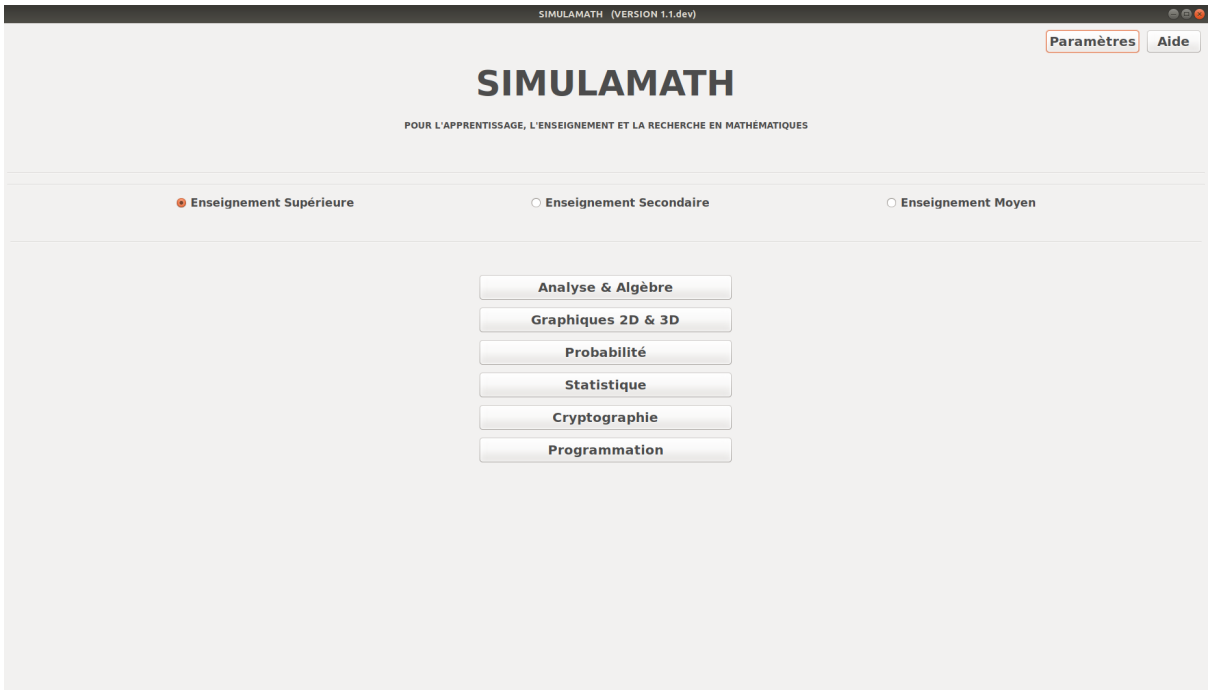
4.4 Statisque inferentielle

La statistique inferentielle consiste a faire des inferences (generalisations) sur la population a partir d' echantillons. Simulamath vous offre l'implementation des tests d'hypothèses et l'estimation par intervalle de confiance.

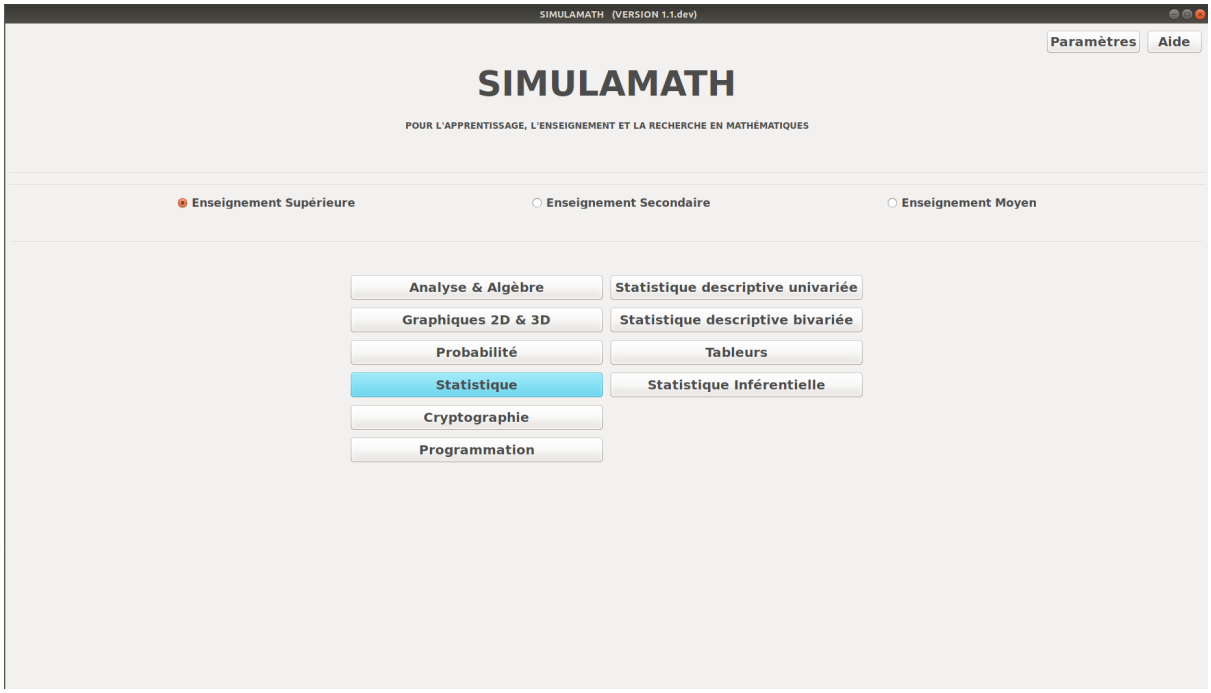
Les exemples dans cette section viennent du live *Elementary Statistics, A step by step approach, 8th Edition*, par Professeur Allan G. Bluman

4.4.1 Comment aller à la section statistique inferentielle

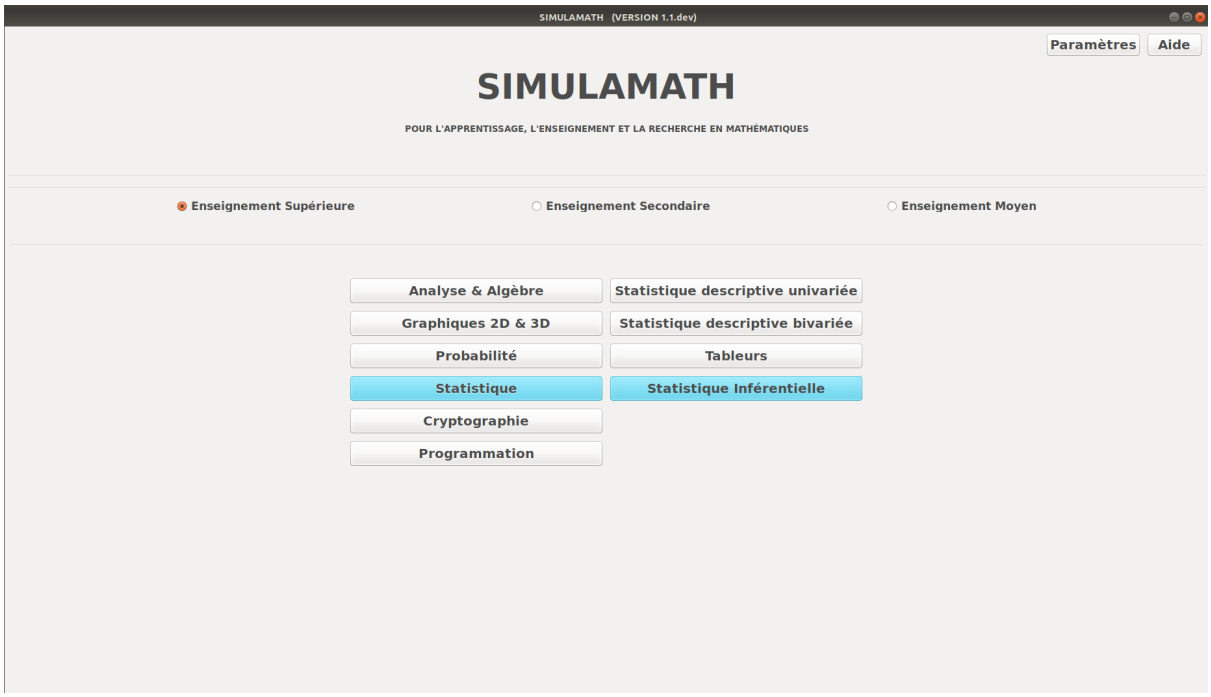
L'image ci-dessous presente la **page d'acceuil de Simulamath**



Cliquer sur **Statistique** (juste poser le curseur sans cliquer est suffisant)



Ensuite vous pourrez choisir/cliquer le thème en statistique qui vous concerne dans notre cas présent, **Statistique inférentielle**.



4.4.2 Estimation par intervalle de confiance

Z-Estimation de moyenne (écart type connu)

La formule pour obtenir l'intervalle de confiance pour Z-Estimation de moyenne est :

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

Exemple :

Une enquête menée auprès de 30 patients aux urgences a révélé que le temps d'attente moyen pour un traitement était d

Solution :

La meilleure estimation ponctuelle de la moyenne est 174,3 minutes. l'intervalle de confiance à 99% de la moyenne de la population est donnée par

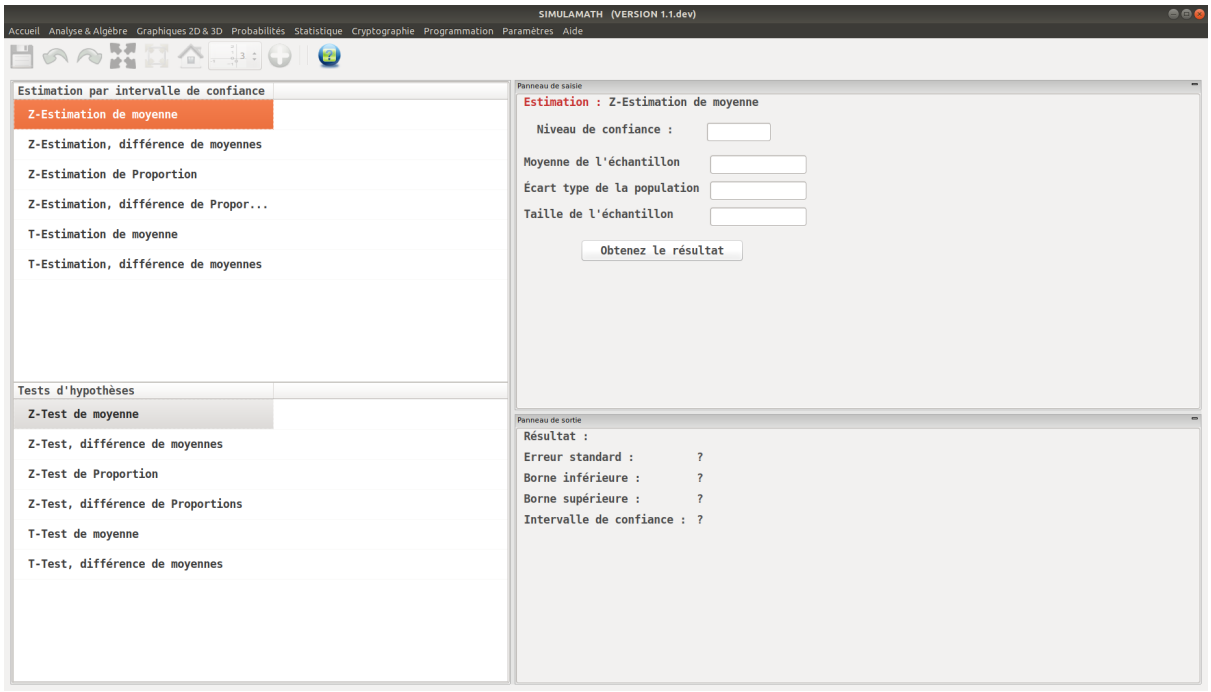
$$174.3 - 2.58\left(\frac{46.5}{\sqrt{30}}\right) < \mu < 174.3 + 2.58\left(\frac{46.5}{\sqrt{30}}\right)$$

$$152.4 < \mu < 196.2$$

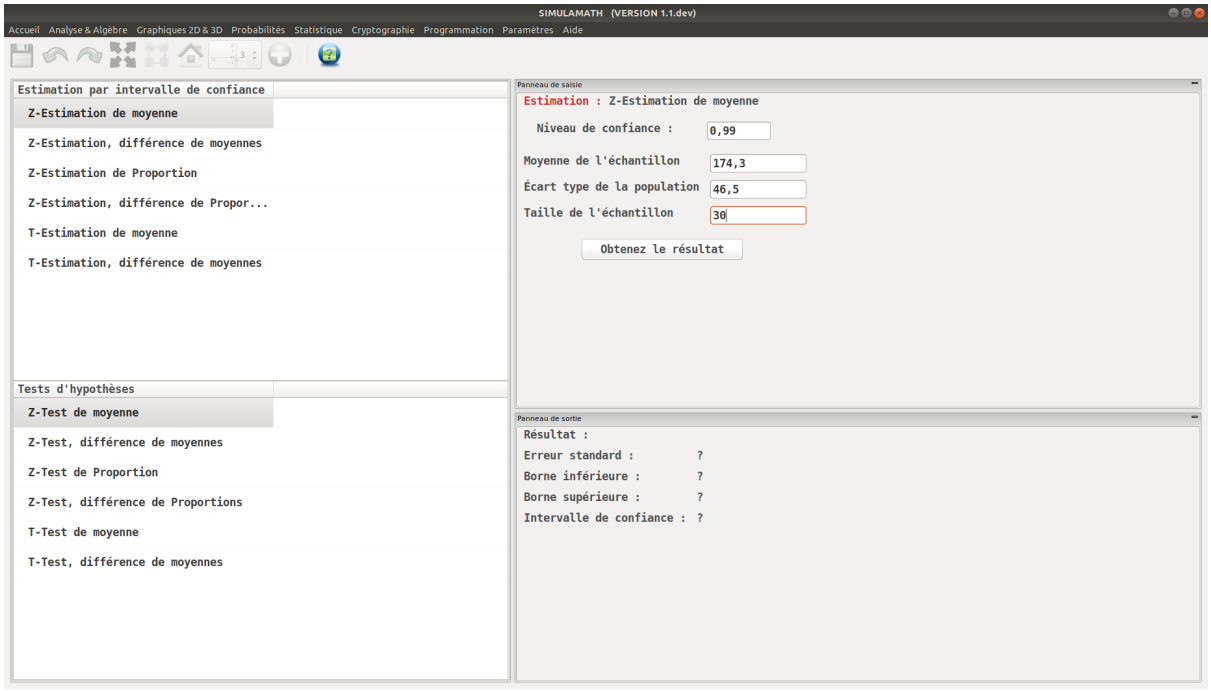
Par conséquent, on peut être sûr à 99% que le temps d'attente moyen pour les urgences le traitement dure entre 152,4 et 196,2 minutes.

Z-Estimation de moyenne dans SimulaMath

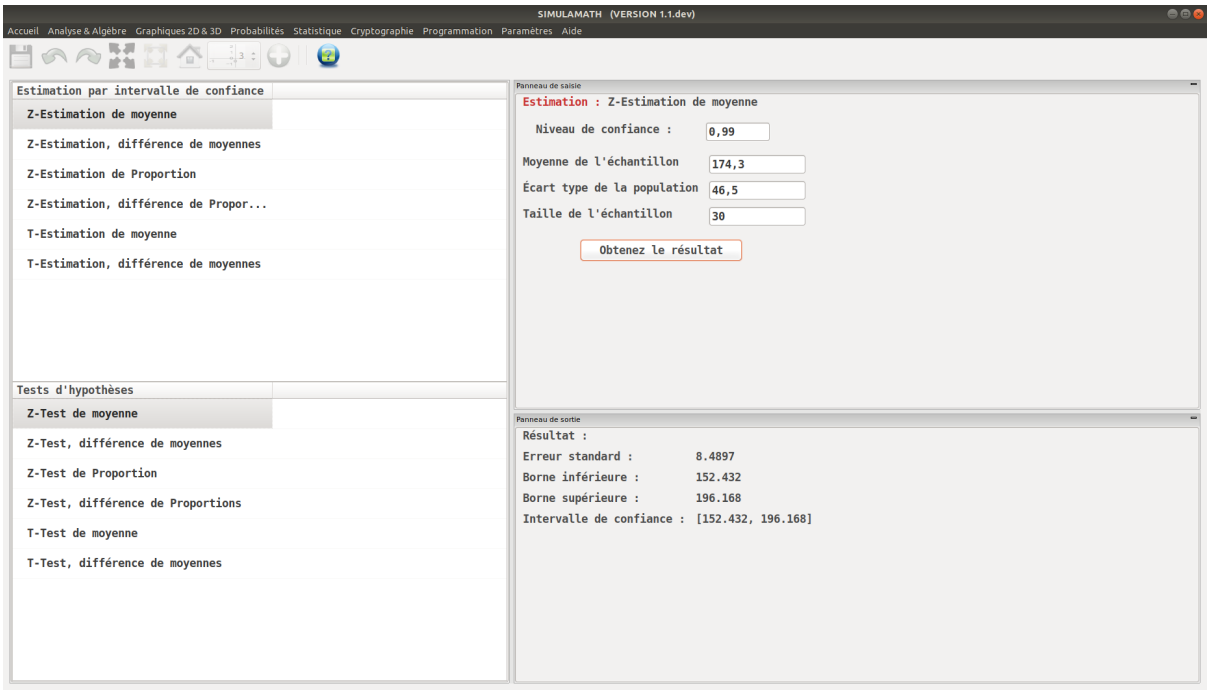
Choisir le type de test/estimation que l'on voudrait evaluer dans les panels à gauche, ici **Z-Estimation de moyenne**



Saisir les variables (Niveau de confiance, Moyenne de l'échantillon , Écart type de la population , Taille de l'échantillon) dans le panel en haut à droite.



Cliquer sur Obtenez le résultat situé juste en dessous de la zone de saisie des varibles. Et Voila, vous avez vous resultats dans le panel de bas à droite.



Z-Estimation, difference de moyenne

La formule pour obtenir l'intervalle de confiance pour Z-Estimation différence de moyenne est :

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

Exemple :

Une enquête a révélé que le tarif moyen d'une chambre d'hôtel à la Nouvelle-Orléans est de 88,42 dollars et le tarif moyen d'une chambre à Phoenix est de 80,61 dollars. Supposons que les données proviennent de deux échantillons de 50 hôtels chacun et que les écarts-types des populations sont respectivement de 5,62 dollars et 4,83 dollars. Avec , peut-on conclure qu'il y a une différence significative dans les taux ? Trouvez l'intervalle de confiance à 95% pour la différence entre les moyennes de ces données.

Solution :

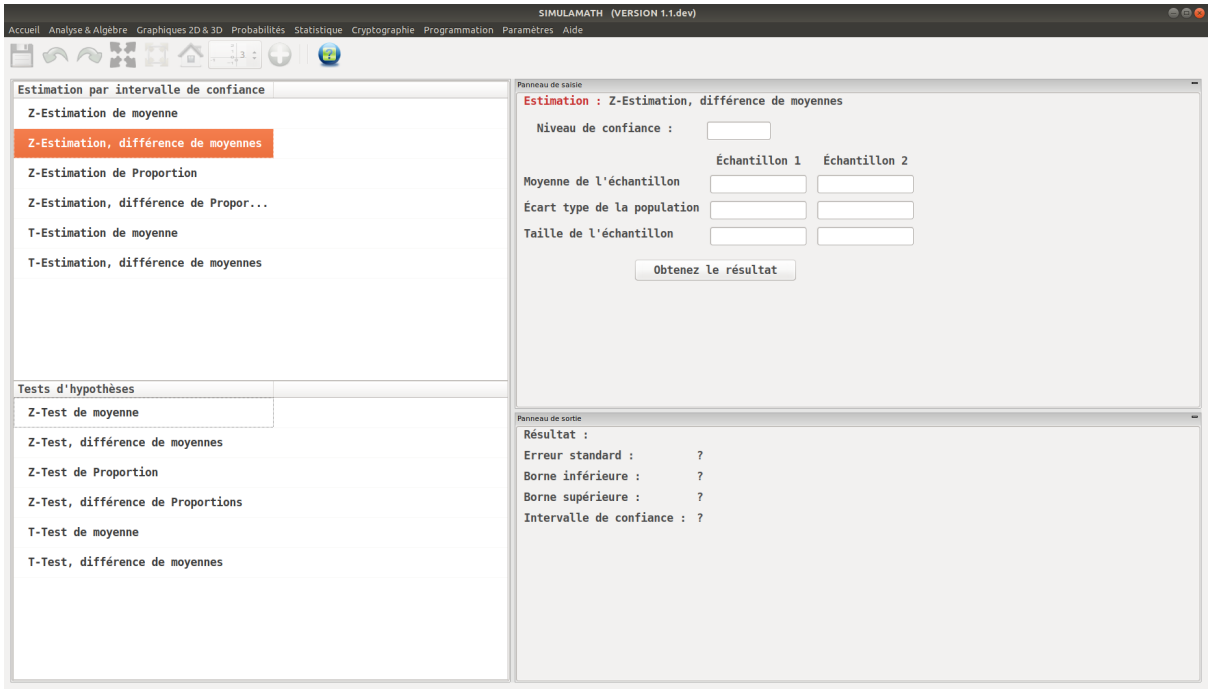
$$88.42 - 80.61) - 1.96\sqrt{\frac{5.62^2}{50} + \frac{4.83^2}{50}} < \mu_1 - \mu_2 < (88.42 - 80.61) + 1.96\sqrt{\frac{5.62^2}{50} + \frac{4.83^2}{50}}$$

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

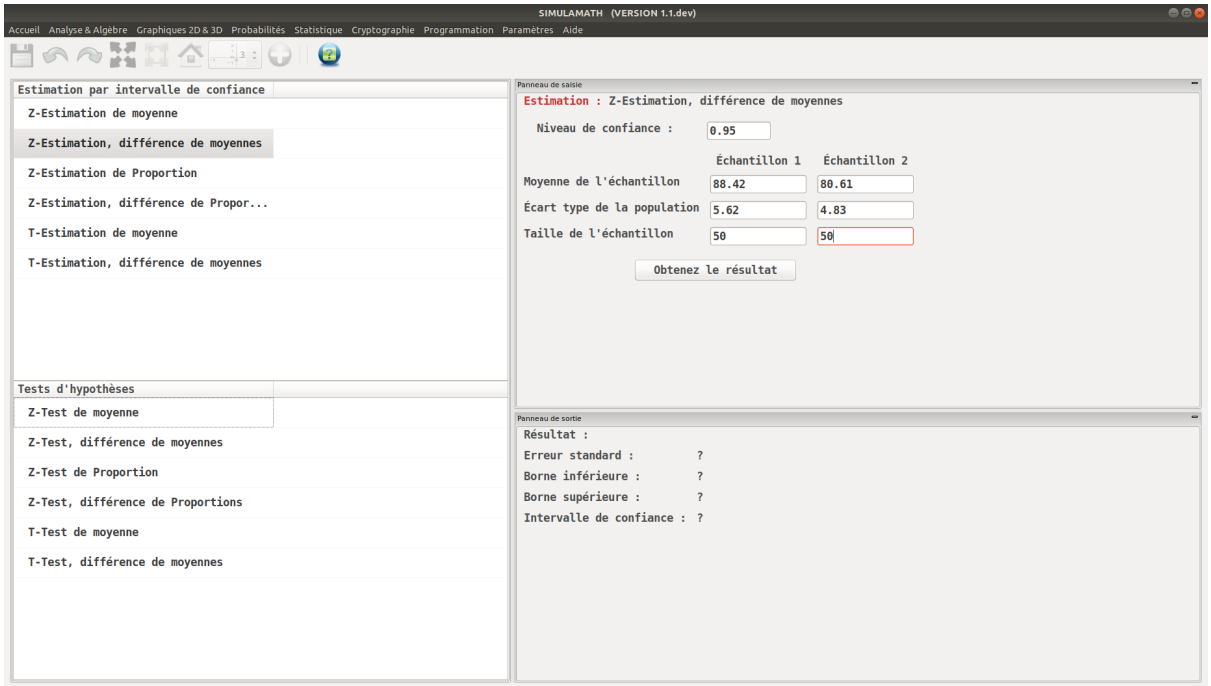
Puisque l'intervalle de confiance ne contient pas zéro, la décision est de rejeter l'hypothèse nulle

Z-Estimation, difference de moyenne dans SimulaMath

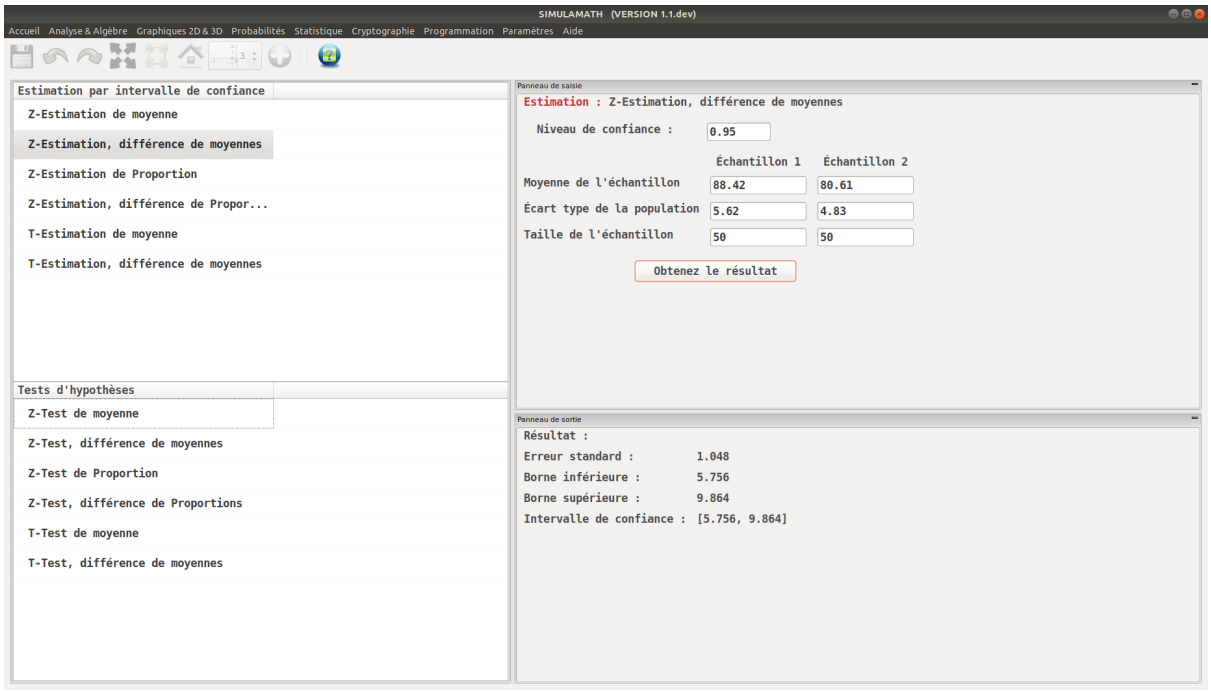
Choisir le type de test/estimation que l'on voudrait evaluer dans les panels à gauche, ici Z-Estimation, difference de moyenne



Saisir les varibles (Niveau de confiance, Moyenne de l'échantillon , Écart type de la population , Taille de l'échantillon) dans le panel en haut à droite pour les deux échantillons (Échantillon 1 et Échantillon 2)



Cliquer sur Obtenez le résultat situé juste en dessous de la zone de saisie des varibles. Et Voila, vous avez vous resultats dans le panel de bas à droite.



Z-Estimation de Proportion

La formule pour obtenir l'intervalle de confiance pour Z-Estimation de Proportion est :

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

Avec $\hat{p} = \frac{X}{n}$ $\hat{q} = 1 - p$

On suppose que pour ce test :

1. On a un échantillon aléatoire.
2. Les conditions pour une expérience binomiale sont remplies.
3. $n_p \geq 5$ et $n_q \geq 5$.

Exemple :

Une enquête menée par Sallie Mae et Gallup auprès de 1404 répondants a révélé que 323 étudiants ont payé leurs études par des prêts étudiants. Trouvez l'intervalle confiance de 90% de la vraie proportion d'étudiants qui ont payé leurs études par des prêts étudiants.

Solution :

Vue que $\alpha = 1 - 0,90 = 0,10$ et $z_{\alpha/2} = 1,65$

En remplaçant dans la formule suivante

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

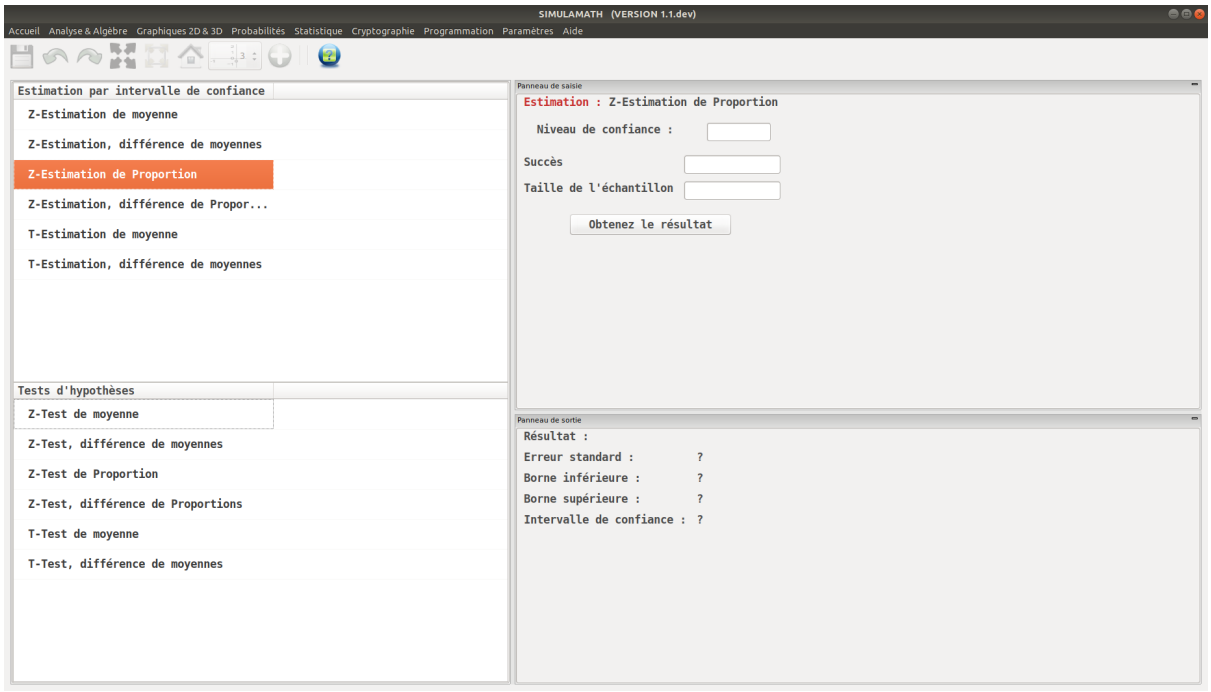
Avec $\hat{p} = \frac{323}{1404} = 0,23$ et $\hat{q} = 1 - p = 0.,77$

Alors

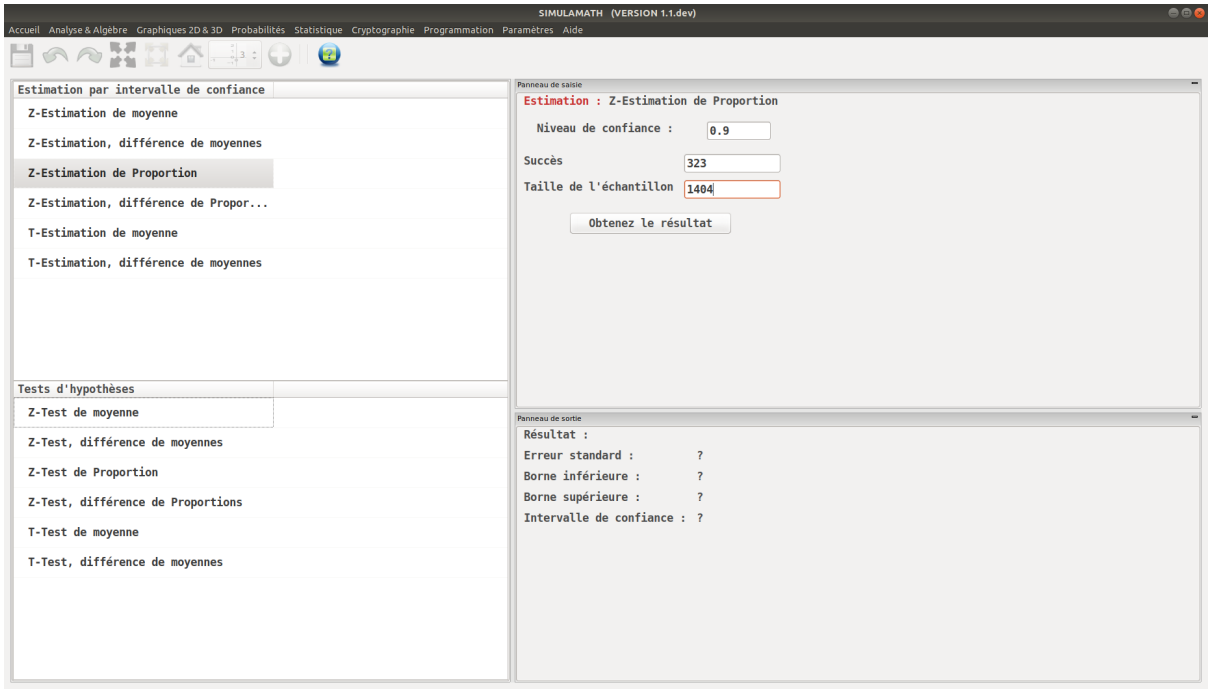
$$0.211 < p < 0.249$$

Z-Estimation de proportion dans SimulaMath

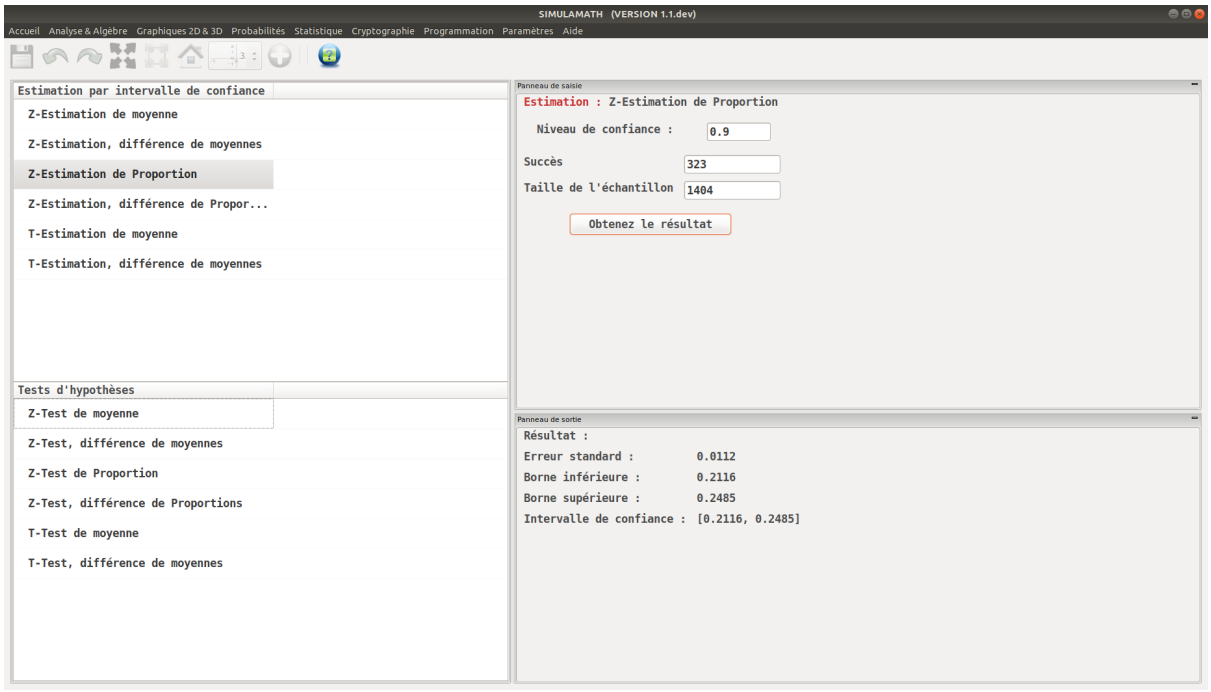
Choisir le type de test/estimation que l'on voudrait evaluer dans les panels à gauche, ici **Z-Estimation de proportion**



Saisir les varibles (Niveau de confiance, nombre de succès, Taille de l'échantillon) dans le panel en haut à droite.



Cliquer sur Obtenez le résultat situé juste en dessous de la zone de saisie des varibles. Et Voila, vous avez vous resultats dans le panel de bas à droite.



Z-Estimation, difference de Proportion

La formule pour obtenir l'intervalle de confiance pour Z-Estimation différence de proportion est :

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

Exemple :

Les chercheurs ont constaté que 12 des 34 petites maisons de soins infirmiers avaient un taux de vaccination des résidents de moins de 80%, tandis que 17 des 24 grandes maisons de soins infirmiers avaient un taux de vaccination de moins de 80%. Avec $\alpha = 0.05$, testez l'affirmation selon laquelle il n'y a pas de différence dans les proportions des petites et grandes maisons de soins infirmiers avec un taux de vaccination des résidents de moins de 80%. Trouvez l'intervalle de confiance à 95% pour ces données.

Solution :

En remplaçant dans la formule suivante

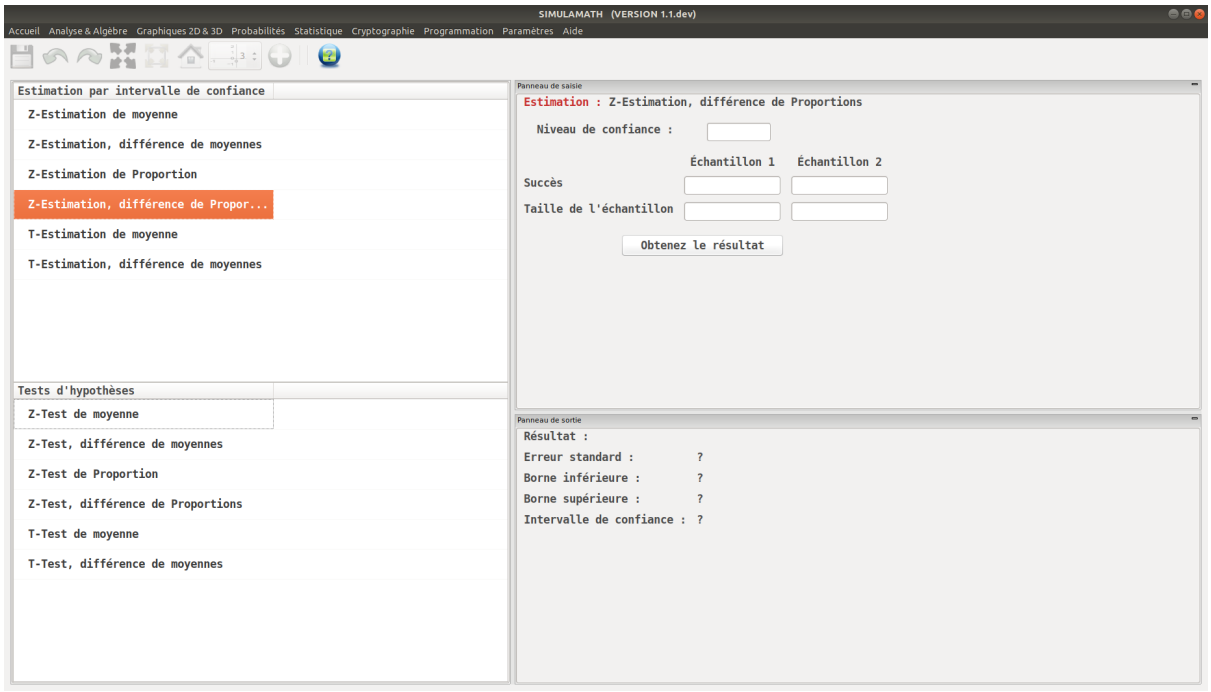
$$(0.35 - 0.71) - 1.96\sqrt{\frac{0.35 * 0.65}{34} + \frac{0.71 * 0.29}{24}} < p_1 - p_2 < (0.35 - 0.71) + 1.96\sqrt{\frac{0.35 * 0.65}{34} + \frac{0.71 * 0}{24}}$$

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

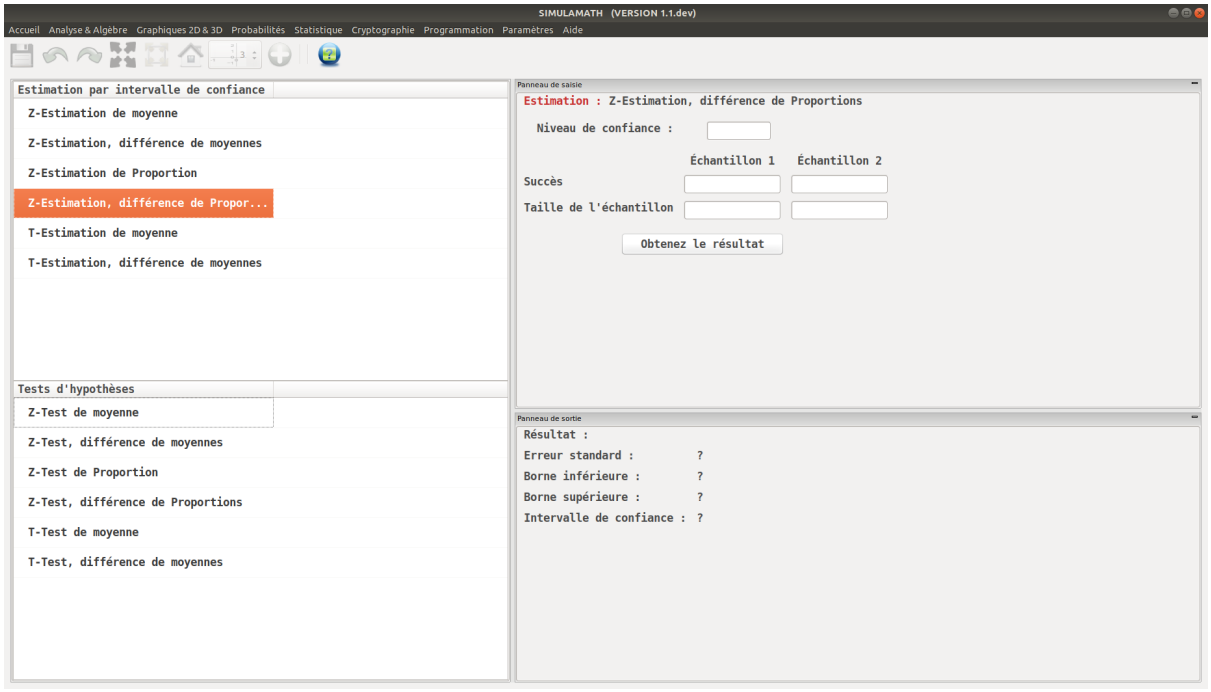
Puisque l'intervalle de confiance ne contient pas zéro, la décision est de rejeter l'hypothèse nulle $H_0 : p_1 = p_2$.

Z-Estimation, difference de Proportion dans SimulaMath

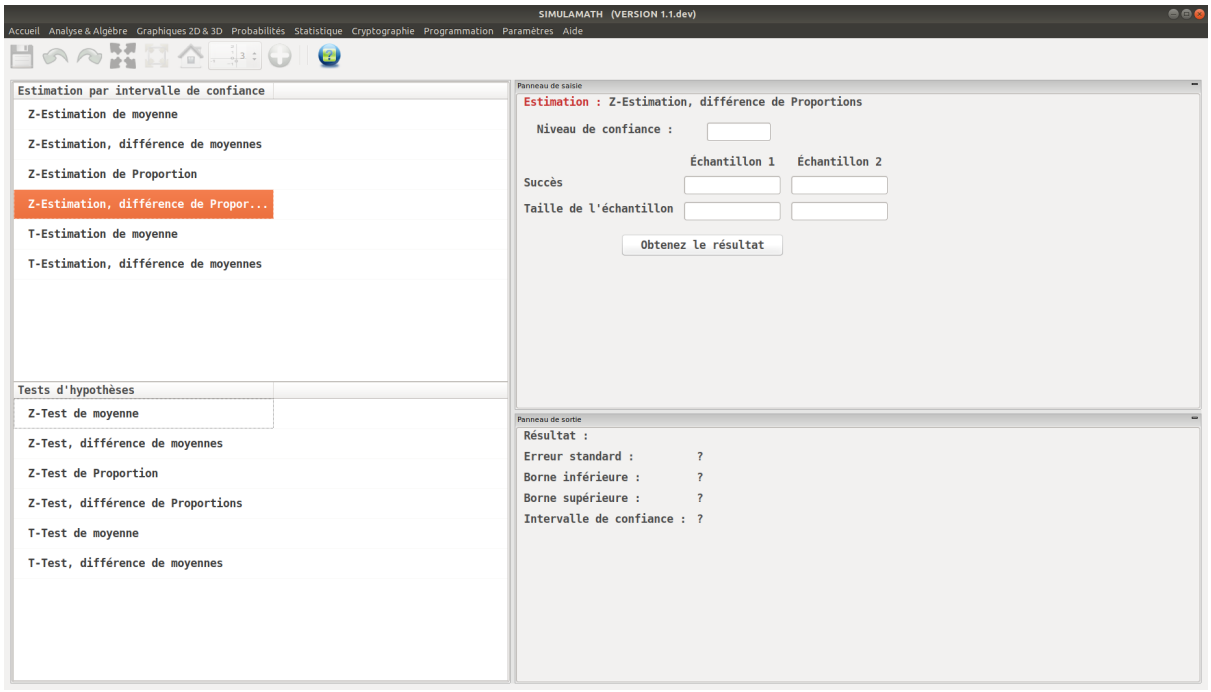
Choisir le type de test/estimation que l'on voudrait evaluer dans les panels à gauche, ici **Z-Estimation, difference de Proportion**



Saisir les varibles (Niveau de confiance, nombre de succès, Taille de l'échantillon) dans le panel en haut à droite.



Cliquer sur Obtenez le résultat situé juste en dessous de la zone de saisie des varibles. Et Voila, vous avez vous resultats dans le panel de bas à droite.



T-Estimation de moyenne (écart type inconnu)

La formule pour obtenir l'intervalle de confiance pour T-Estimation de moyenne est :

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

On suppose que pour ce test :

1. On a un échantillon aléatoire.
2. Soit $n \geq 30$ soit la population est normalement distribuée si math :*n < 30*.

Exemple :

On a demandé à dix personnes sélectionnées au hasard combien de temps elles dormaient la nuit. La durée moyenne était de 7,1 heures et l'écart type de 0,78 heure. Trouvez l'intervalle de confiance à 95% du temps moyen. Supposons que la variable est normalement distribuée.

Puisque σ est inconnu et s que doit le remplacer, la distribution t (table F) doit être utilisée pour l'intervalle de confiance. Donc, avec 9 degrés de liberté $t_{\alpha/2} = 2.262$. L'intervalle de confiance à 95% peut être trouvé en remplaçant dans la formule.

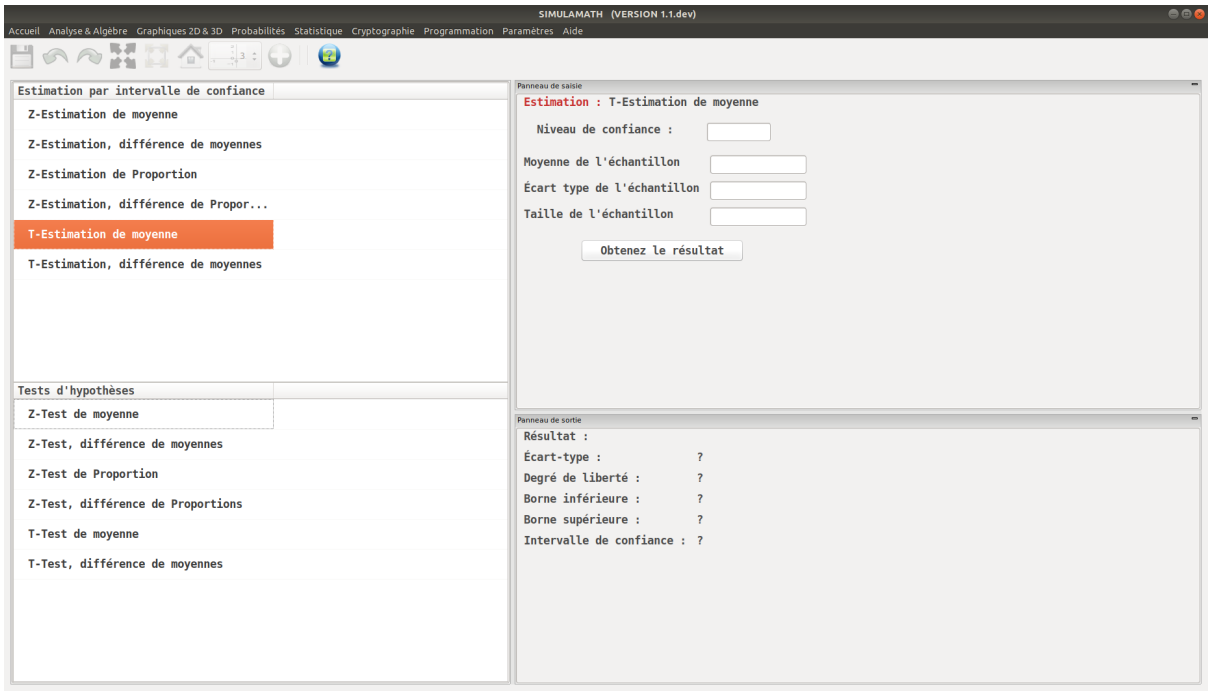
$$7.1 - 2.262\left(\frac{0.78}{\sqrt{10}}\right) < \mu < 7.1 + 2.262\left(\frac{0.78}{\sqrt{10}}\right)$$

$$6.54 < \mu < 7.66$$

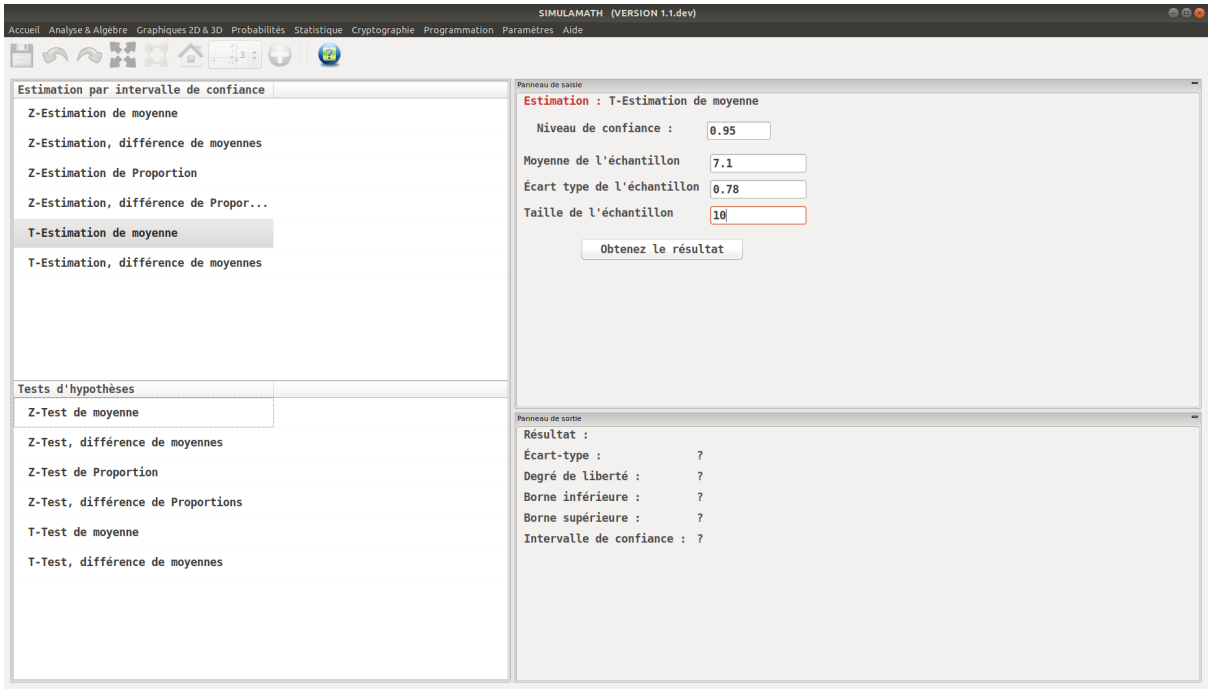
Par conséquent, on peut être sûr à 90% que la moyenne de la population est entre 6,54 et 7,66 heures.

T-Estimation de moyenne dans SimulaMath

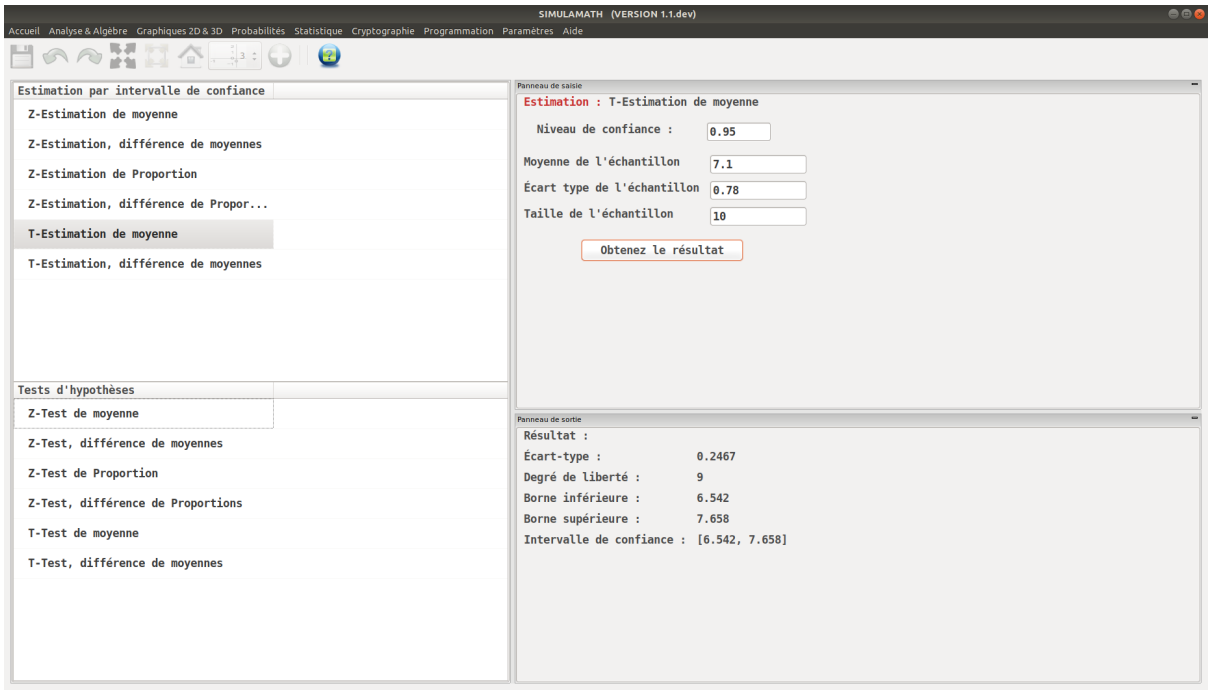
Choisir le type de test/estimation que l'on voudrait evaluer dans les panels à gauche, ici **T-Estimation de moyenne**



Saisir les variables (Niveau de confiance, Moyenne de l'échantillon , Écart type de l'échantillon , Taille de l'échantillon) dans le panel en haut à droite pour les deux échantillons.



Cliquer sur Obtenez le résultat situé juste en dessous de la zone de saisie des varibles. Et Voila, vous avez vous resultats dans le panel de bas à droite.



T-Estimation, difference de moyenne

La formule pour obtenir l'intervalle de confiance pour Z-Estimation différence de moyenne est ;

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

Exemple :

La taille moyenne d'une ferme dans le ville de Thies, au Senegal, est de 191 m^2. La taille moyenne d'une ferme dans la ville de Mbour, au Senegal, est de 199 m^2. Supposons que les données ont été obtenues à partir de deux échantillons avec des écarts types de 38 et 12 m^2, respectivement, et d'échantillon de tailles de 8 et 10, respectivement. Peut-on conclure avec $lpha = 0.05$ que la taille moyenne des exploitations dans les deux villes est différente ? Supposons que les populations soient normalement réparties. Solution :

En remplaçant dans la formule suivante

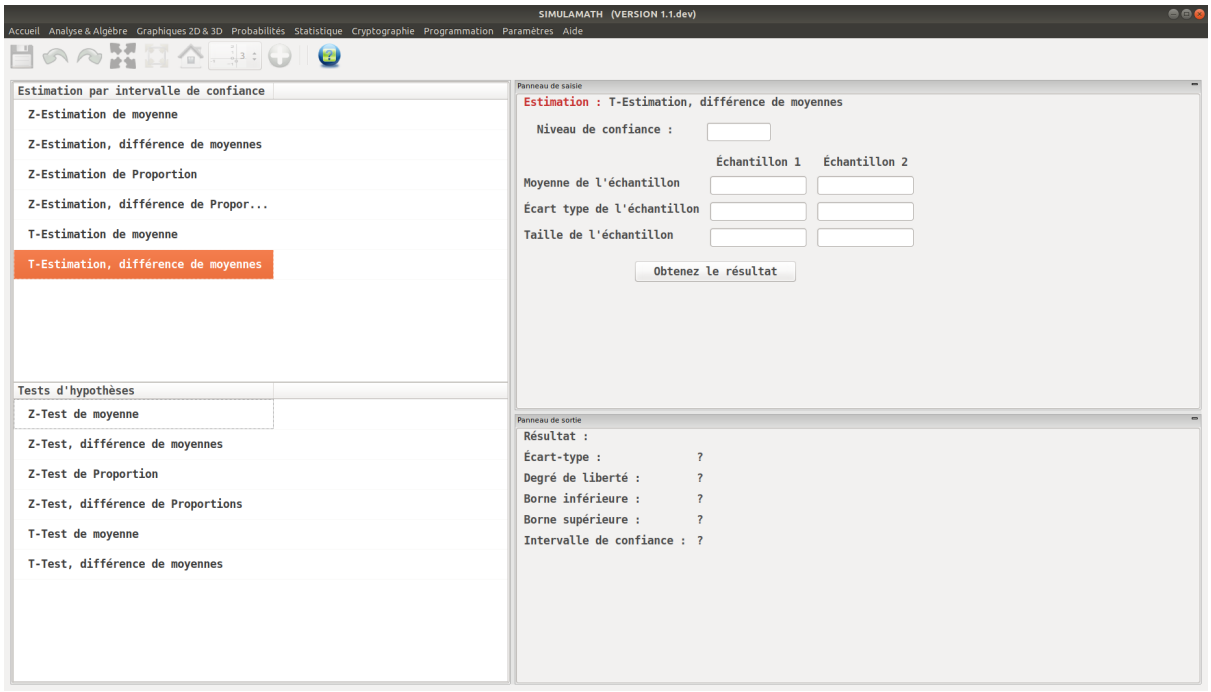
$$(191 - 199) - 2.365\sqrt{\frac{38^2}{8} + \frac{12^2}{10}} < \mu_1 - \mu_2 < (191 - 199) + 2.365\sqrt{\frac{38^2}{8} + \frac{12^2}{10}}$$

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

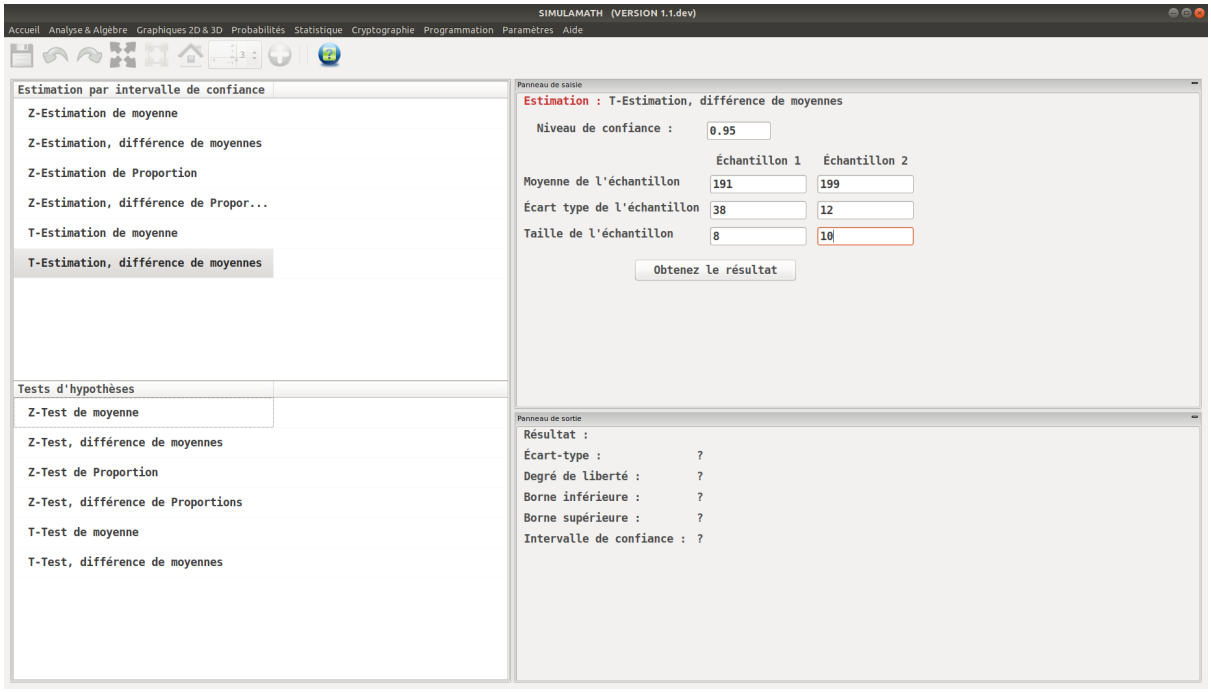
Puisque l'intervalle de confiance contient zéro, la décision est de ne rejeter pas l'hypothèse nulle $H_0 : \mu_1 = \mu_2$.

T-Estimation, Difference of Means in Simulamath

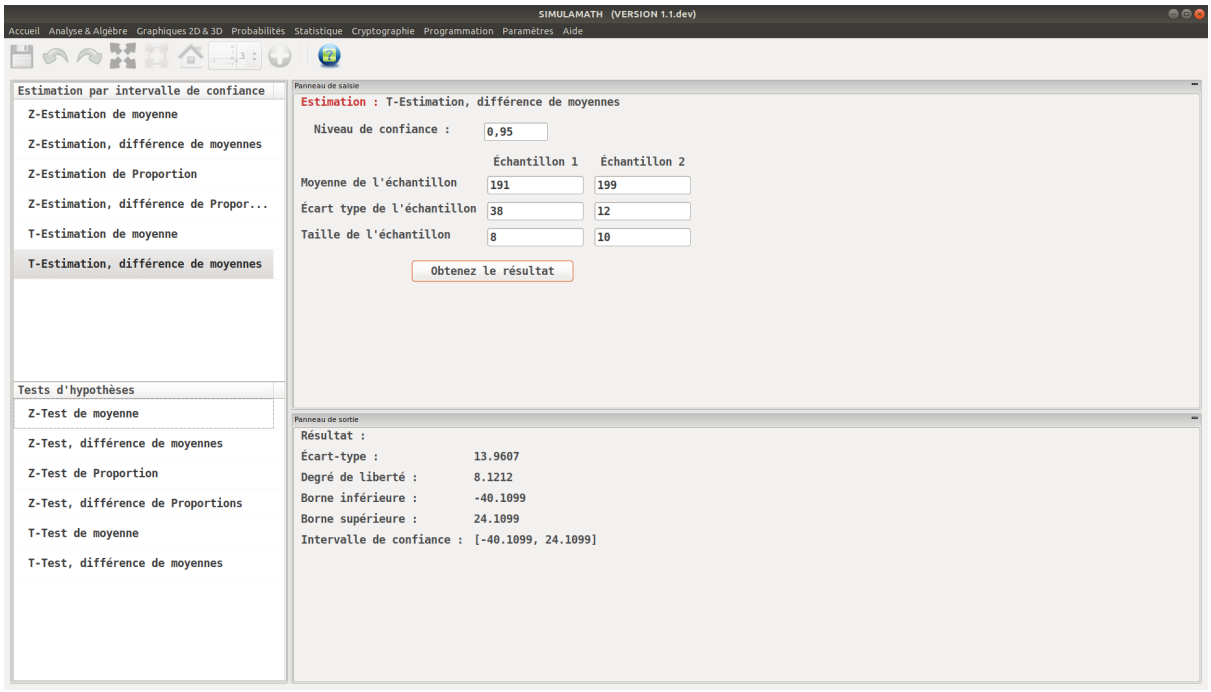
T-Estimation, difference de moyennes dans SimulaMath



Saisir les variables (Niveau de confiance, nombre de succès, Taille de l'échantillon) pour chaque échantillons dans le panel en haut à droite.



Cliquer sur Obtenez le résultat situé juste en dessous de la zone de saisie des varibles. Et Voila, vous avez vous resultats dans le panel de bas à droite.



4.4.3 Test d'hypothèses

Z-Test de moyenne

La formule pour le Z-Test de moyenne est :

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

Hypothèses pour le z-test pour une moyenne lorsque :math :`sigma` est connu

1. On a un échantillon aléatoire.
2. Soit $n \geq 30$ soit la population est normalement distribuée si $n < 30$.

Exemple :

Un chercheur souhaite voir si le nombre moyen de jours pendant lesquels une petite automobile basique à bas prix se trouve sur le terrain d'un concessionnaire est de 29. Un échantillon de 30 concessionnaires automobiles a une moyenne de 30,1 jours pour les petites automobiles basiques à bas prix. Avec $\alpha = 0.05$, testez l'affirmation selon laquelle le temps moyen est supérieur à 29 jours. L'écart type de la population est 3,8 jours.

Solution :

Etape 1 : Enoncer les hypothèses et identifier la revendication des moyennes.

$$H_0 : \mu = 29 \quad \text{et} \quad H_1 : \mu > 29 \text{(revendication)}$$

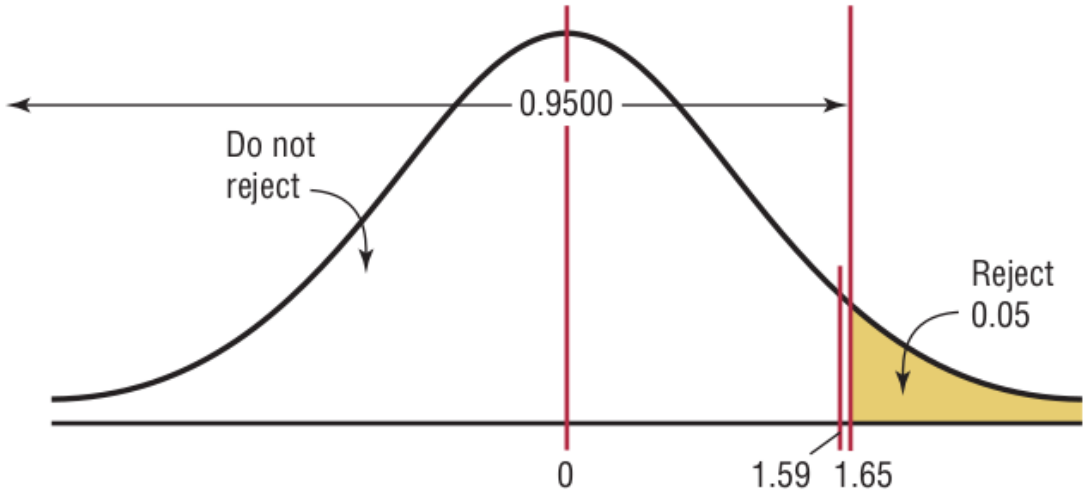
Etape 2 : Trouvez la valeur critique. Étant donné que $\alpha = 0.05$ et que le test est un test unilatéralement, la valeur critique est $z = +1.65$.

Etape 3 : Calculer la valeur de test.

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

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

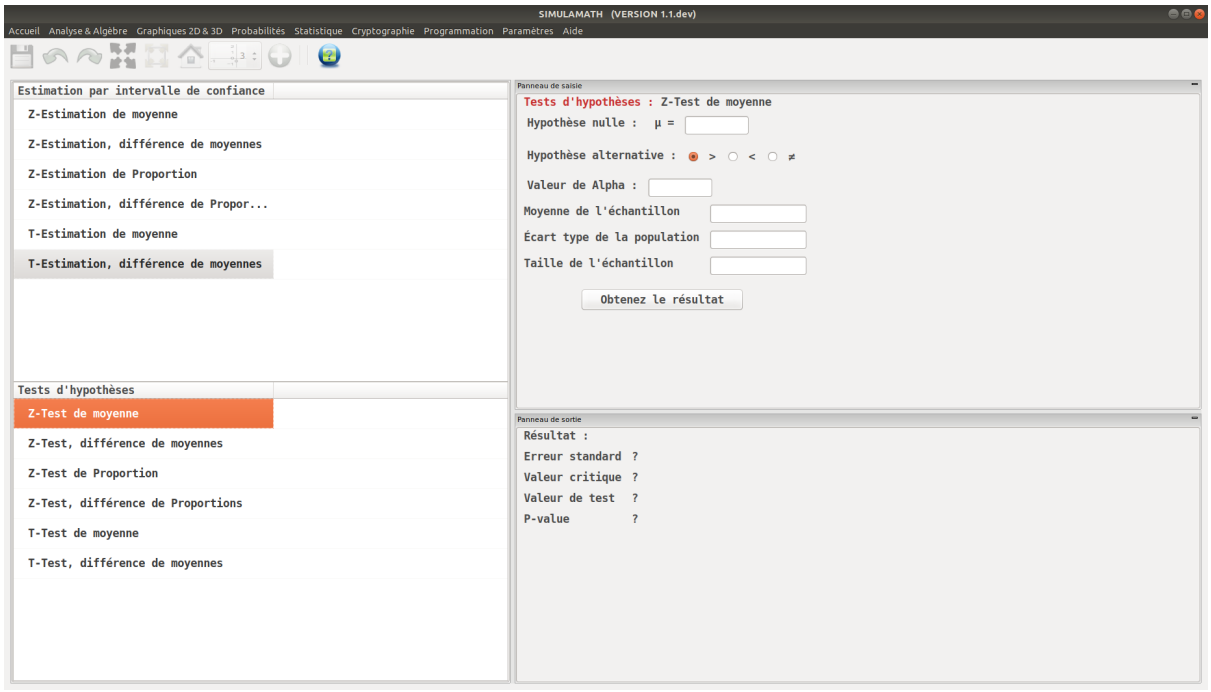
Etape 4 : Decider. Puisque la valeur de test, 1,59, est inférieure à la valeur critique, 1,65, et n'est pas dans la région critique, la décision est de ne pas rejeter l'hypothèse nulle.



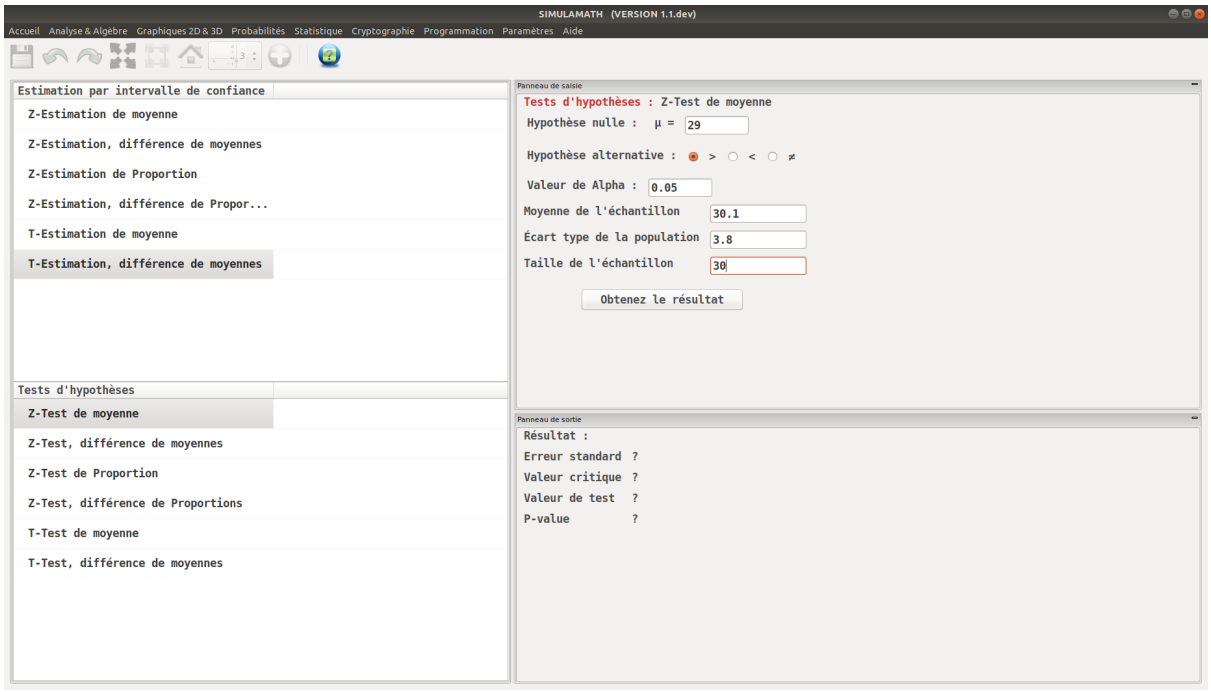
Etape 5 : Résumez les résultats. Il n'y a pas suffisamment de preuves pour étayer l'affirmation selon laquelle le délai moyen est supérieur à 29 jours.

Z-Test de moyenne dans SimulaMath

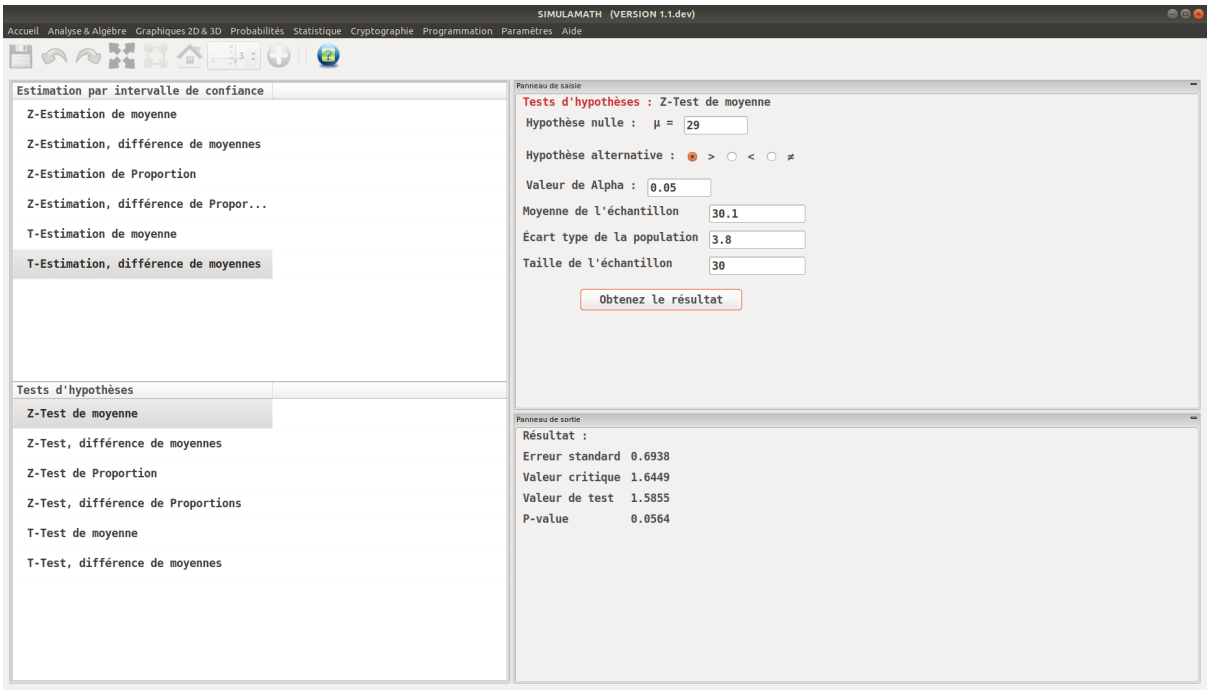
Choisir le type de test/estimation que l'on voudrait evaluer dans les panels à gauche, ici **Z-test de moyenne**



Saisir les varibles (Hypothèse nulle , Hypothèse alternative, Valeur de Alpha , Moyenne de l'échantillon , Écart type de la population , Taille de l'échantillon) dans le panel en haut à droite.



Cliquer sur Obtenez le résultat situé juste en dessous de la zone de saisie des varibles. Et Voila, vous avez vous resultats dans le panel de bas à droite.



Z-Test, difference de moyennes

La formule pour le Z-Test de différence entre deux moyennes de populations est :

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

$$\text{valeur de test} = \frac{(\text{valeur observée}) - (\text{valeur attendue})}{\text{erreur standard}}$$

Exemple :

Une enquête a révélé que le tarif moyen d'une chambre d'hôtel à la Nouvelle-Orléans est de 88,42 dollars et le tarif moyen d'une chambre à Phoenix est de 80,61 dollars. Supposons que les données proviennent de deux échantillons de 50 hôtels chacun et que les écarts-types des populations sont respectivement de 5,62 dollars et 4,83 dollars. Avec , peut-on conclure qu'il y a une différence significative dans les taux ? Trouvez l'intervalle de confiance à 95% pour la différence entre les moyennes de ces données.

Solution :

Etape 1 : Enoncer les hypothèses et identifier la revendication des moyennes.

$$H_0 : \mu_1 = \mu_2 \quad \text{et} \quad H_1 : \mu_1 \neq \mu_2 (\text{revendication})$$

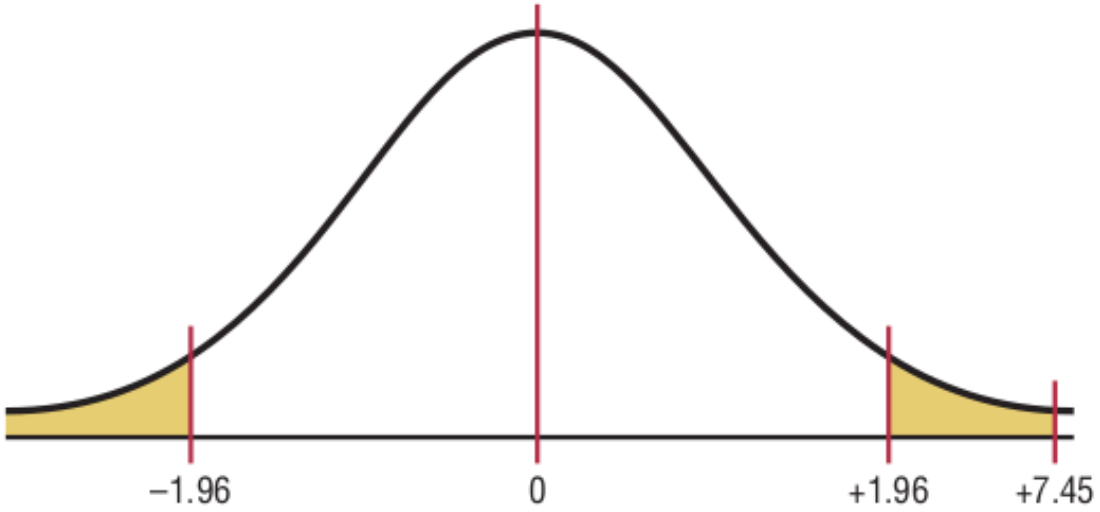
Étape 2 : Trouvez les valeurs critiques. Puisque $lpha = 0.05$, les valeurs critiques sont +1.96 et -1.96.

Etape 3 : Calculer la valeur de test.

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

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

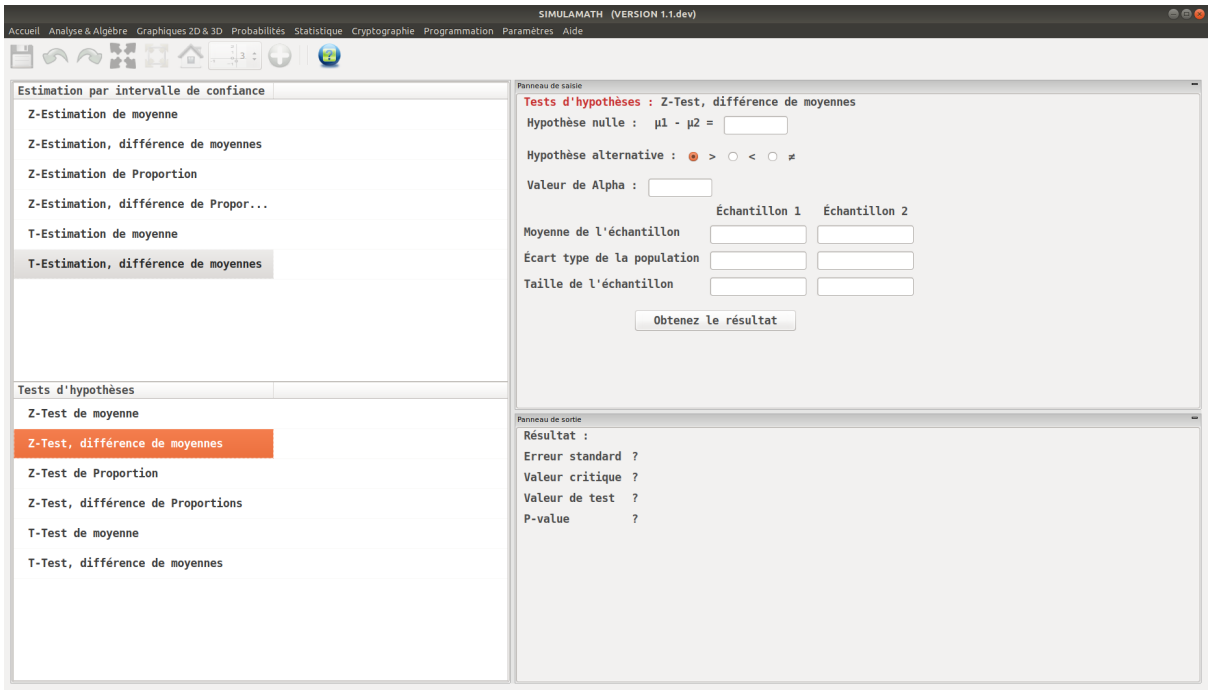
Etape 4 : Décider. Rejetez l'hypothèse nulle à $lpha = 0,05$, puisque 7,45 > 1,96.



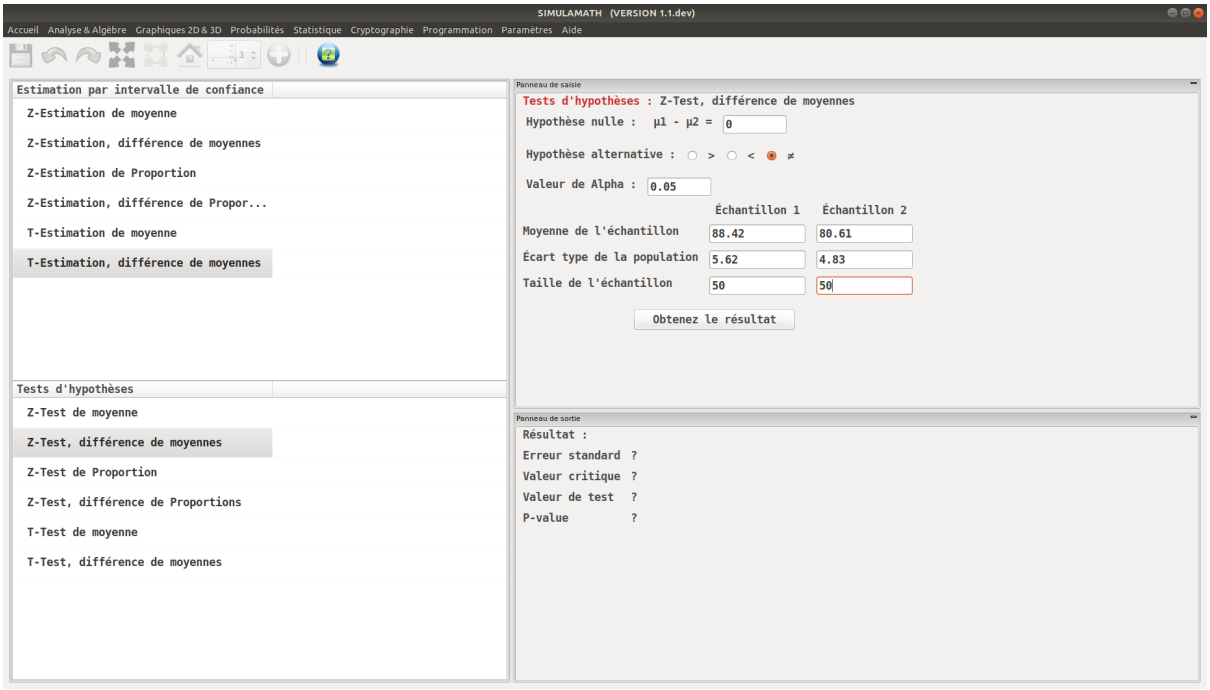
Stape 5 : Résumer le résultat. Il existe suffisamment de preuves pour étayer l'affirmation selon laquelle les moyennes ne sont pas égaux. Par conséquent, il existe une différence significative dans les taux.

Z-Test, difference de moyenne dans SimulaMath

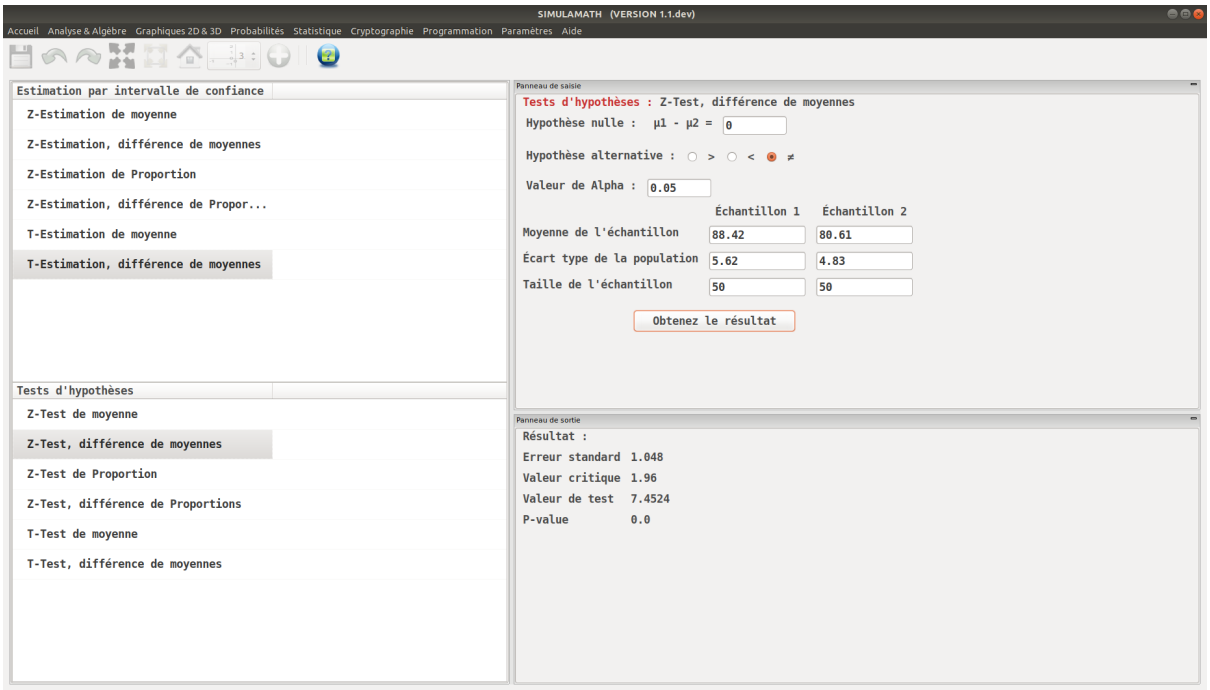
Choisir le type de test/estimation que l'on voudrait evaluer dans les panels à gauche, ici Z-Estimation, difference de moyenne



Saisir les varibles (Hypothèse nulle, Hypothèse alternative, Valeur de Alpha, Moyenne de l'échantillon, Écart type de la population , Taille de l'échantillon) dans le panel en haut à droite pour les deux échantillons.



Cliquer sur Obtenez le résultat situé juste en dessous de la zone de saisie des varibles. Et Voila, vous avez vous resultats dans le panel de bas à droite.



Z-Test de Proportion

La formule pour le Z-Test de proportion est

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

On suppose que pour ce test :

1. On a un échantillon aléatoire.
2. Les conditions pour une expérience binomiale sont remplies.
3. $np \geq 5$ and $nq \geq 5$.

Exemple :

Une diététiste affirme que 60% des gens essaient d'éviter les gras trans dans leur alimentation. Elle a sélectionné au hasard 200 personnes et a constaté que 128 personnes ont déclaré qu'elles essayaient d'éviter les gras trans dans leur alimentation. Avec $\alpha = 0.05$, y a-t-il suffisamment de preuves pour rejeter l'allégation de la diététiste ?

Solution :

Étape 1 : Enoncez l'hypothèse et identifiez la réclamation.

$$H_0 : p = 0,60 \quad \text{(revendication)} \quad \text{et} \quad H_1 : p \neq 0,60$$

Étape 2 : Trouvez les valeurs critiques. Étant donné que $\alpha = 0.05$ et que la valeur de test est bilatérale, les valeurs critiques sont +1,96 et -1,96.

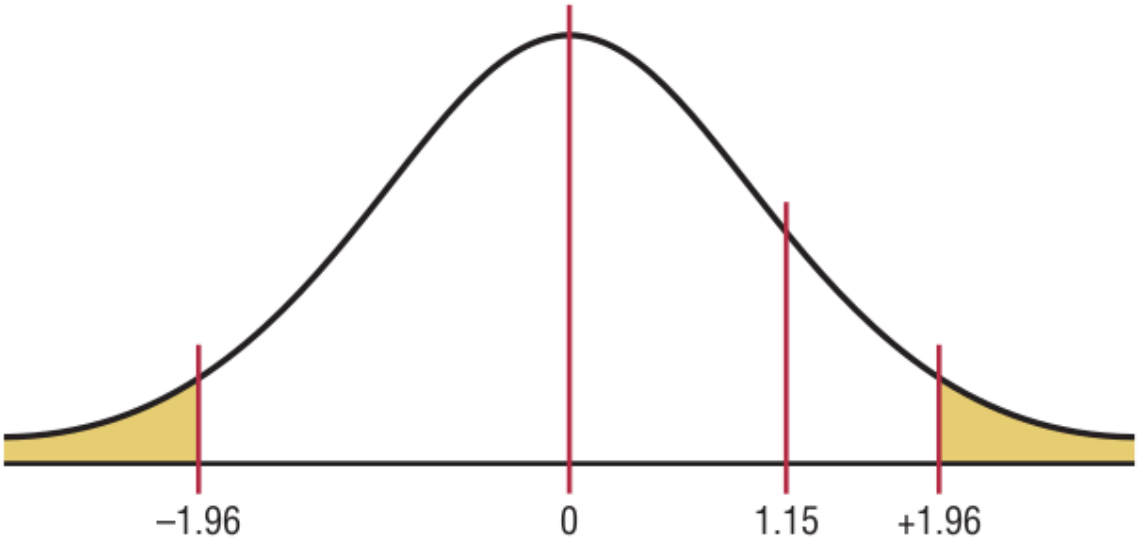
Étape 3 : Calculez la valeur de test. Tout d'abord, il est nécessaire de trouver $\hat{p}$.

$$\hat{p} = \frac{X}{n} = \frac{128}{200} = 0,64 \qquad p = 0,60 \quad \text{et donc} \quad q = 1 - p = 0,40$$

Ainsi

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

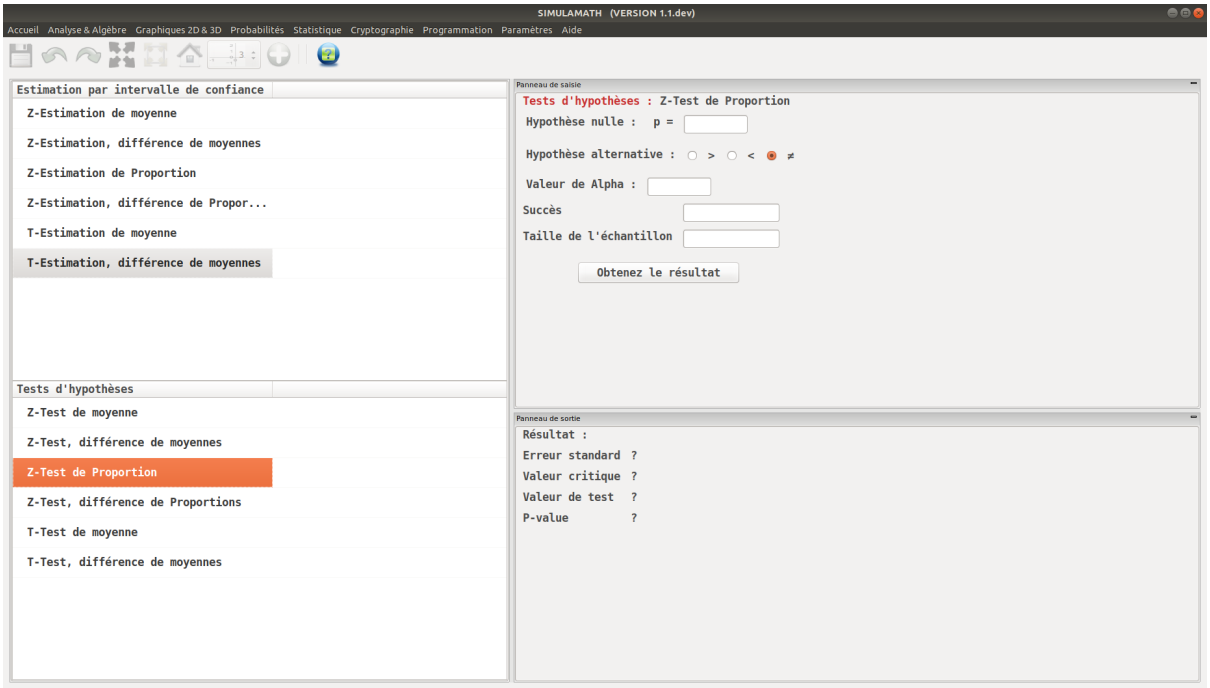
Etape 4 : Décision. Ne rejetez pas l'hypothèse nulle car la valeur de test se situe en dehors de la région critique, comme le montre la figure ci-dessous.



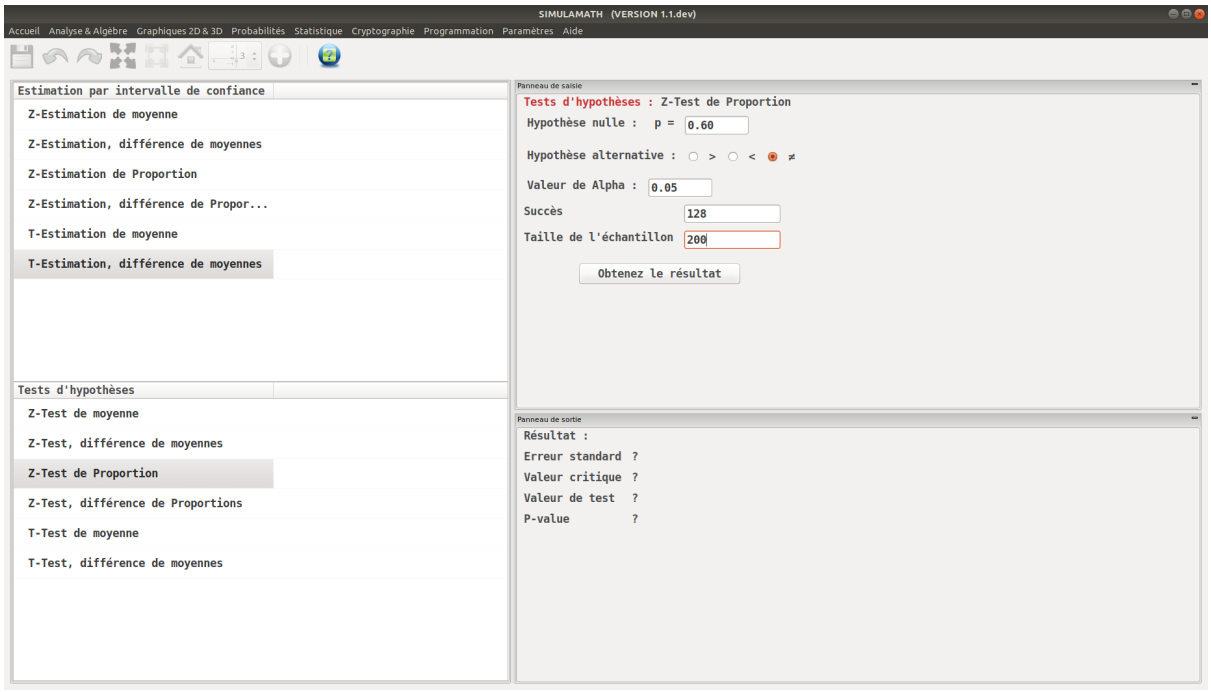
Etape 5 : Il n'y a pas suffisamment de preuves pour rejeter l'affirmation selon laquelle 60% des gens essaient d'éviter les gras trans dans leur alimentation.

Z-Test de proportion dans SimulaMath

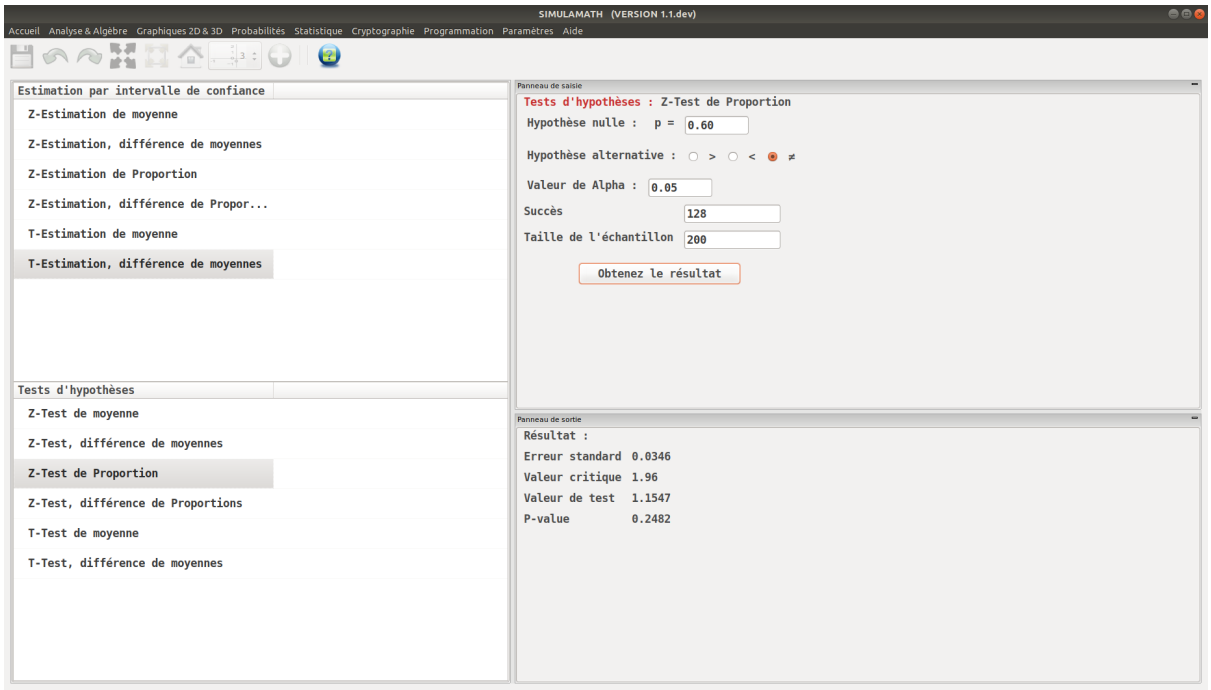
Choisir le type de test/estimation que l'on voudrait evaluer dans les panels à gauche, ici **Z-test proportion**



Saisir les varibles (Hypothèse nulle, Hypothèse alternative, Valeur de Alpha, nombre de succès, Taille de l'échantillon) dans le panel en haut à droite.



Cliquer sur Obtenez le résultat situé juste en dessous de la zone de saisie des varibles. Et Voila, vous avez vous resultats dans le panel de bas à droite.



Z-Test, difference de Proportion

La formule pour le Z-Test de difference entre deux proportions est

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

où

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

et

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

La formule ci-dessus suis le format suivant :

$$\text{valeur de test} = \frac{(\text{valeur observée}) - (\text{valeur attendue})}{\text{erreur standard}}$$

On suppose que pour ce test :

1. On a des échantillons aléatoires
2. Les échantillons sont independants.
3. Pour les deux échantillon $np \geq 5$ and $nq \geq 5$.

Exemple :

Les chercheurs ont constaté que 12 des 34 petites maisons de soins infirmiers avaient un taux de vaccination des résidents de moins de 80%, tandis que 17 des 24 grandes maisons de soins infirmiers avaient un taux de vaccination de moins de 80%. Avec $\alpha = 0.05$, testez l'affirmation selon laquelle il n'y a pas de différence dans les proportions des petites et grandes maisons de soins infirmiers avec un taux de vaccination des résidents de moins de 80%.

Solution :

Soit $\hat{p_1}$ la proportion petites maisons de soins infirmiers avec un taux de vaccination de moins de 80% et $\hat{p_2}$ la proportion des grandes maisons de soins infirmiers avaient un taux de vaccination de moins de 80%. Alors

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

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

Etape 1 : Enoncer les hypothèses et identifier la revendication des moyennes.

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

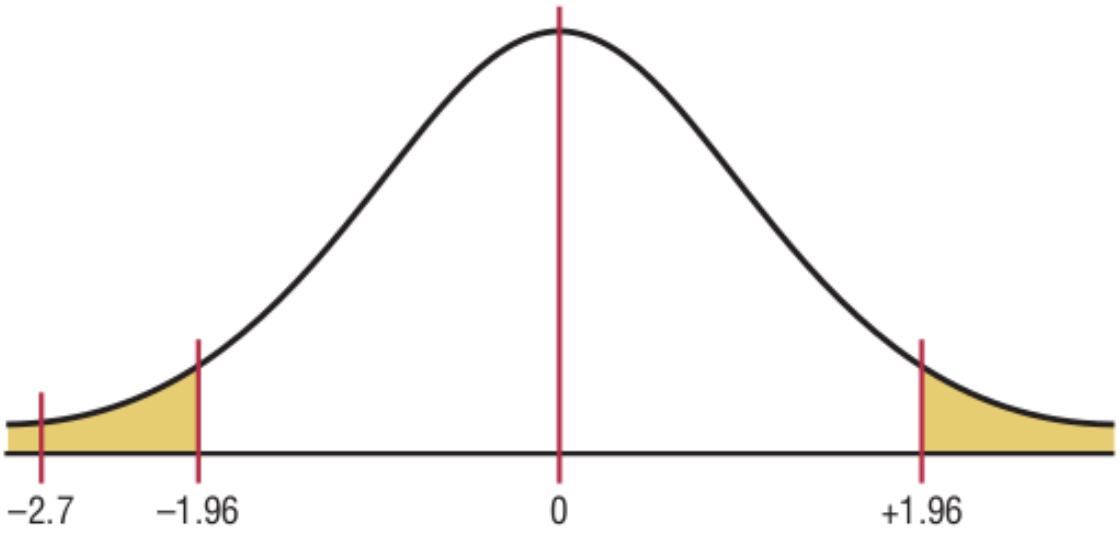
Étape 2 : Trouvez les valeurs critiques. Puisque $lpha = 0.05$, les valeurs critiques sont +1.96 et -1.96.

Etape 3 : Calculer la valeur de test.

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

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

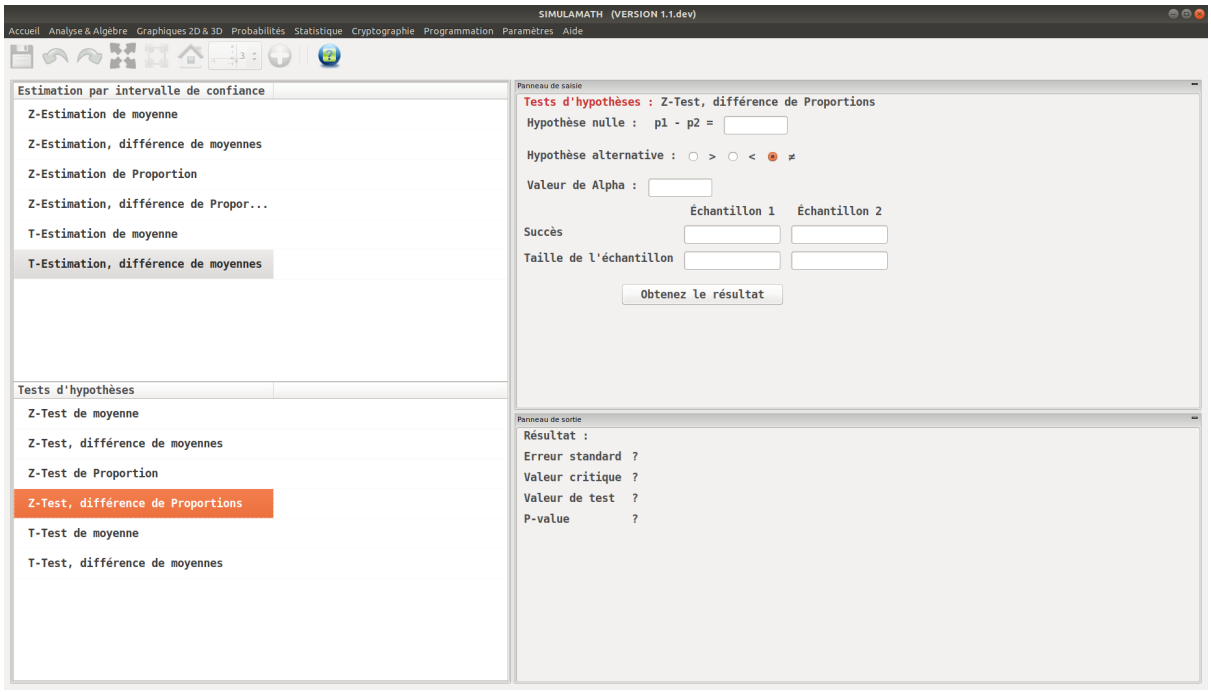
Etape 4 : Decider. Rejeter l'hypothèse nulle, vue que -2,7 < -1,96.



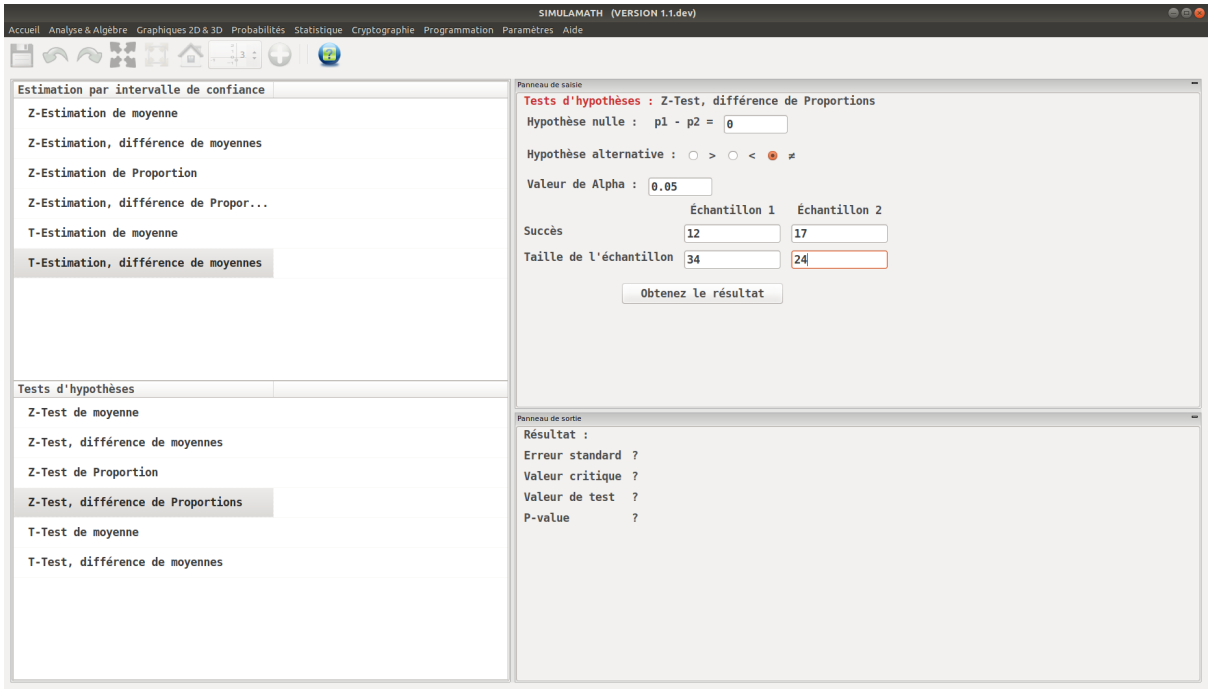
Etape 5 : Resumer le resultat. Il y a suffisamment de preuves pour rejeter l'affirmation selon laquelle il n'y a pas de différence dans les proportions de petites et grandes maisons de soins infirmiers avec un taux de vaccination des résidents inférieur à 80%.

Z-Test, difference de proportion dans SimulaMath

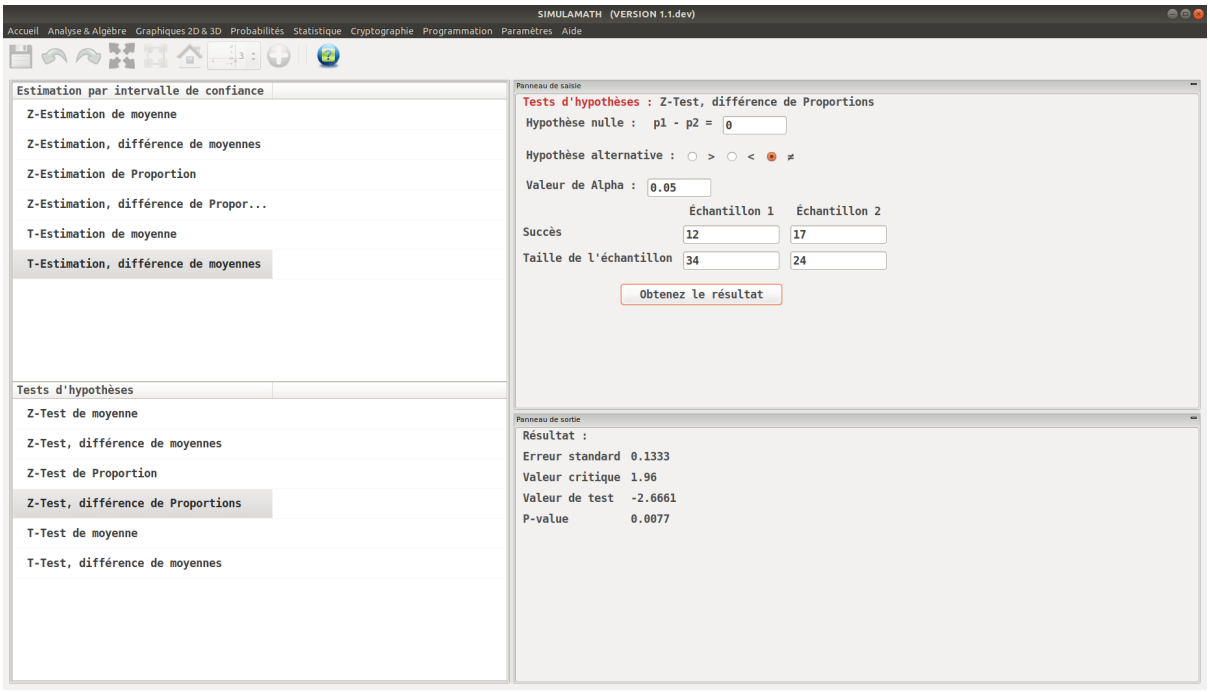
Choisir le type de test/estimation que l'on voudrait evaluer dans les panels à gauche, ici **Z-Estimation, difference de Proportion**



Choisir le type de test/estimation que l'on voudrait evaluer dans les panels à gauche, ici **Z-test, difference proportion**



Cliquer sur Obtenez le résultat situé juste en dessous de la zone de saisie des varibles. Et Voila, vous avez vous resultats dans le panel de bas à droite.



T-Test de moyenne

La formule pour le T-Test de moyenne

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

La formule ci-dessus suis le format suivant :

$$\text{valeur de test} = \frac{(\text{valeur observée}) - (\text{valeur attendue})}{\text{erreur standard}}$$

Ici on assume que :

1. On a un échantillon aléatoire.
2. Soit $n \geq 30$ soit la population est normallement distribuée si $n < 30$.

Exemple :

Une enquête médicale affirme que le nombre moyen d'infections par semaine dans un hôpital du sud-ouest de Wakadan est de 16,3. Un échantillon aléatoire de 10 semaines avait un nombre moyen de 17,7 infections. L'écart type de l'échantillon est de 1,8. Y a-t-il suffisamment de preuves pour rejeter la demande de l'enquêteur à 0,05 ?

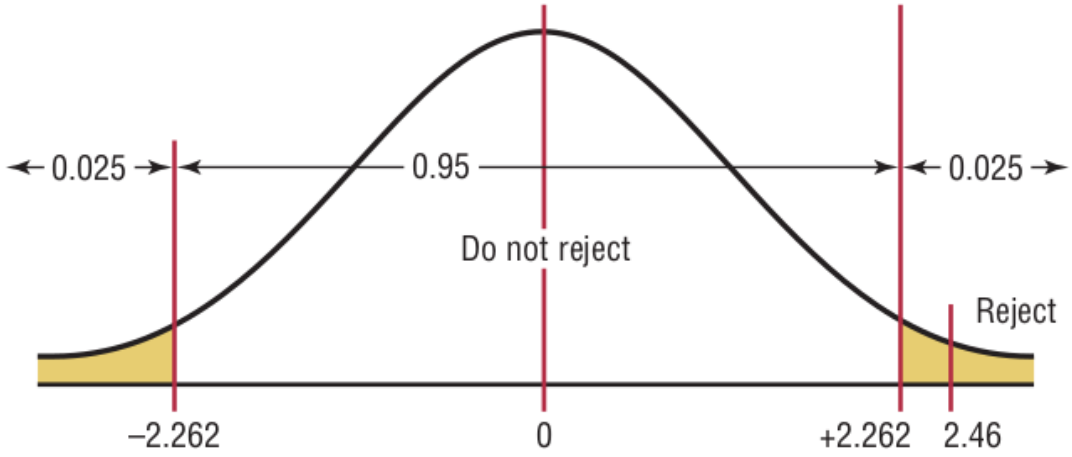
Etape 1 : $H_0 : \mu = 16,3$ (revendication) et $H_1 : \mu \neq 16,3$.

Étape 2 : Les valeurs critiques sont +2.262 et -2.262 pour $\alpha = 0,05$ et $d.f = 9$.

Étape 3 : La valeur de test est

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

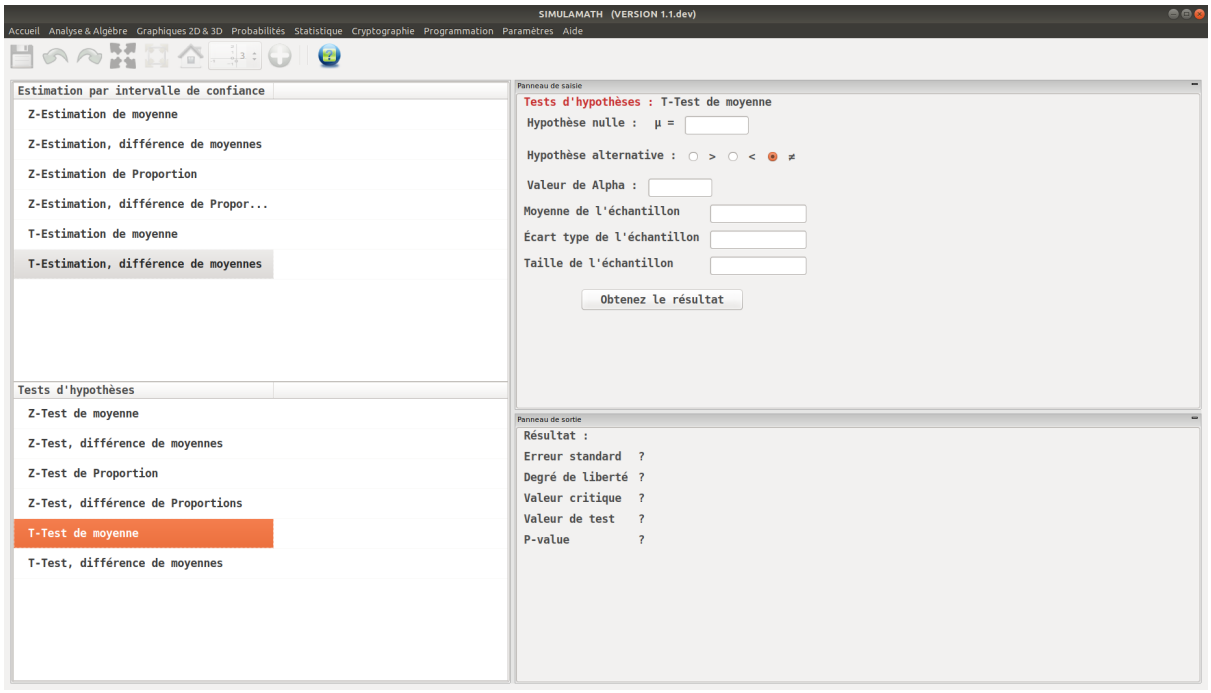
Étape 4 : Rejeter l'hypothèse nulle vue que $2.46 > 2.262$.



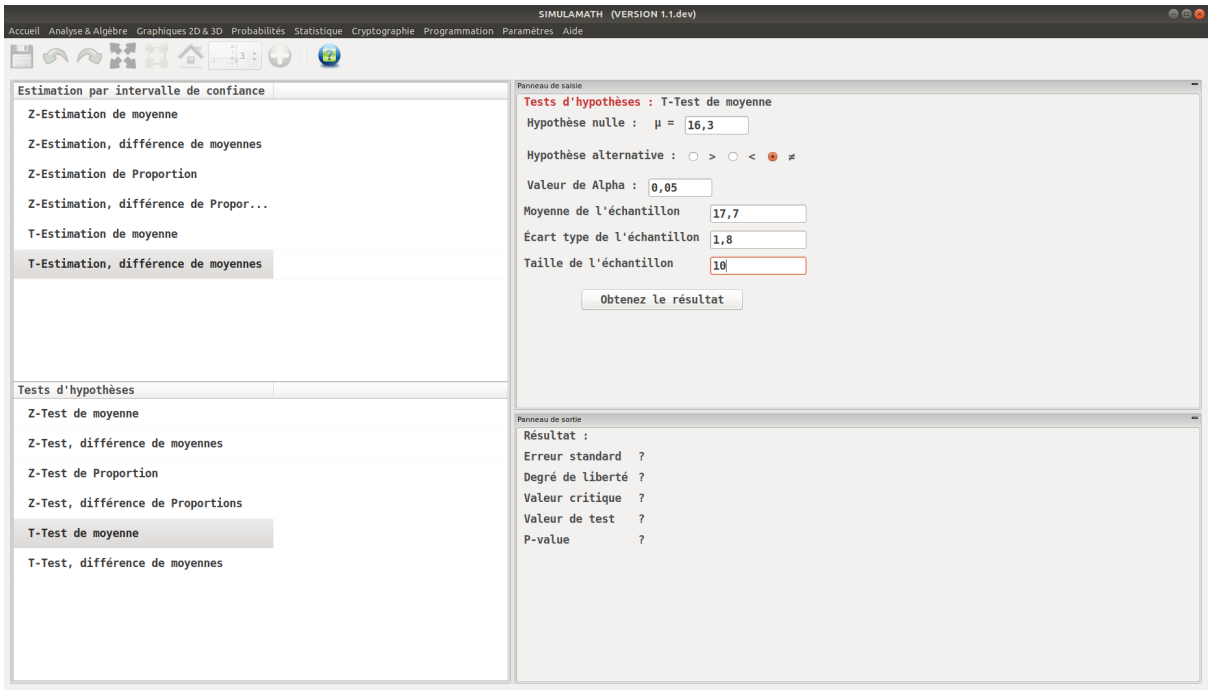
Étape 5 : Il existe suffisamment de preuves pour rejeter l'affirmation selon laquelle le nombre moyen d'infections est de 16,3.

T-Test de moyenne dans SimulaMath

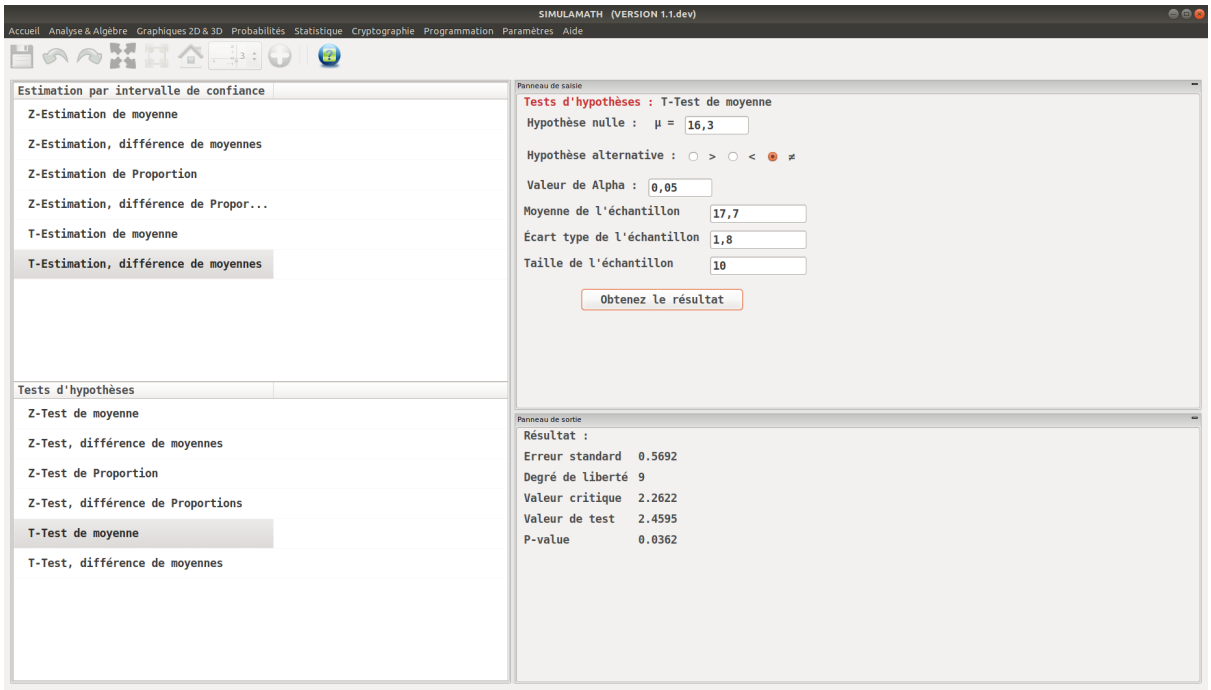
Choisir le type de test/estimation que l'on voudrait evaluer dans les panels à gauche, ici **T-test de moyenne**



Saisir les varibles (Hypothèse nulle, Hypothèse alternative, Valeur de Alpha, Moyenne de l'échantillon, Écart type de l'échantillon, Taille de l'échantillon) dans le panel en haut à droite



Cliquer sur Obtenez le résultat situé juste en dessous de la zone de saisie des varibles. Et Voila, vous avez vous resultats dans le panel de bas à droite.



T-Test, difference de moyennes

La formule pour le T-Test de difference entre deux moyennes d'echantillons independant

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

La formule ci-dessus suis le format suivant :

$$\text{valeur de test} = \frac{(\text{valeur observée}) - (\text{valeur attendue})}{\text{erreur standard}}$$

Ici on assume que les variances ne sont pas egales.

1. Les échantillons sont aléatoires
2. Les échantillons sont independants.
3. Soit $n \geq 30$ soit la population est normalement distribuée si $n < 30$.

Exemple :

La taille moyenne d'une ferme dans le ville de Thies, au Senegale, est de 191 m^2. La taille moyenne d'une ferme dans la ville de Mbour, au Senegale, est de 199 m^2. Supposons que les données ont été obtenues à partir de deux échantillons avec des écarts types de 38 et 12 m^2, respectivement, et d'échantillon de tailles de 8 et 10, respectivement. Peut-on conclure avec $\alpha = 0.05$ que la taille moyenne des exploitations dans les deux villes est différente ? Supposons que les populations soient normalement réparties.

Solution :

Etape 1 : Enoncer les hypothèses et identifier la revendication des moyennes.

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

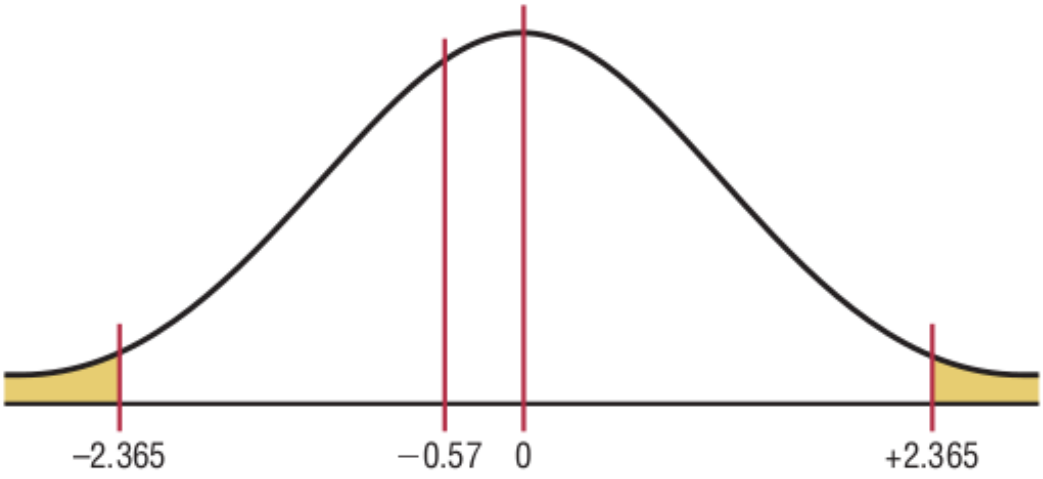
Etape 2 : Trouvez les valeurs critiques. Puisque le test est bilatéral, et $lpha = 0.05$, et que les variances sont inégales, les degrés de liberté sont les plus petits de $n_1 - 1$ ou $n2 - 1$. Dans ce cas, les degrés de liberté sont $8 - 1 = 7$. D'où , d'après la Table F, les valeurs critiques sont 2,365 et -2,365.

Etape 3 : Calculer la valeur du test. Puisque les variances sont inégales, utilisez la première formule.

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

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

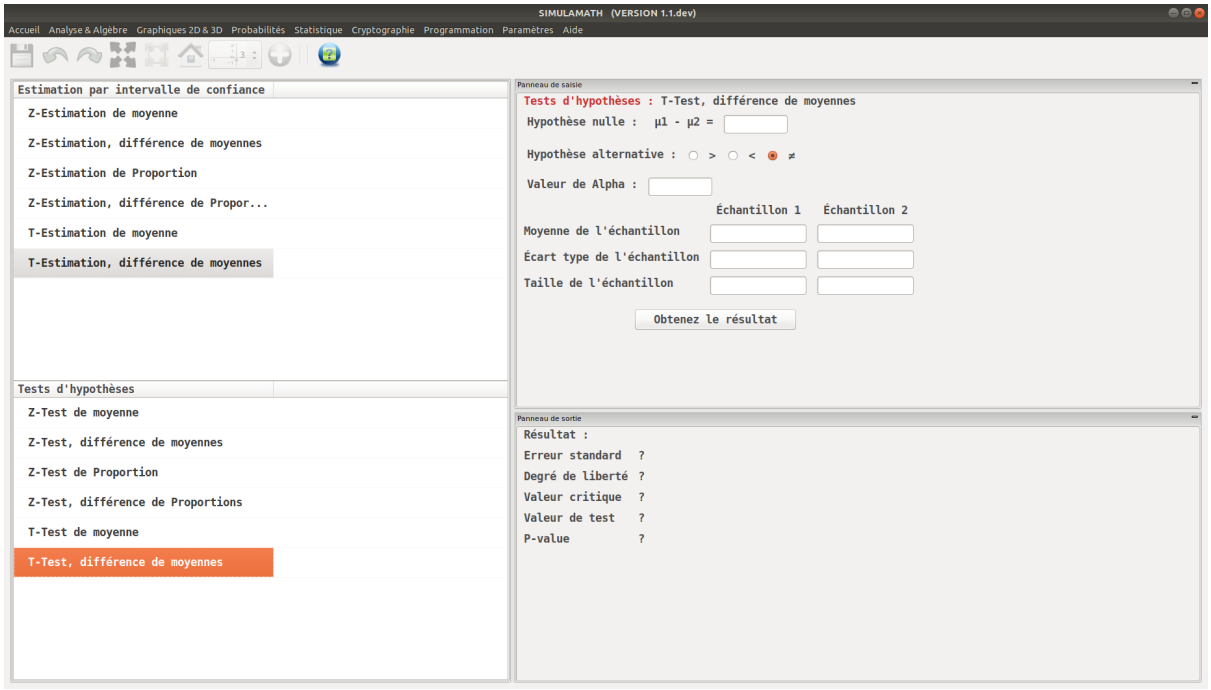
Etape 4 : Décider. Ne pas rejeter l'hypothèse nulle, vue que $-0.57 > -2.365$



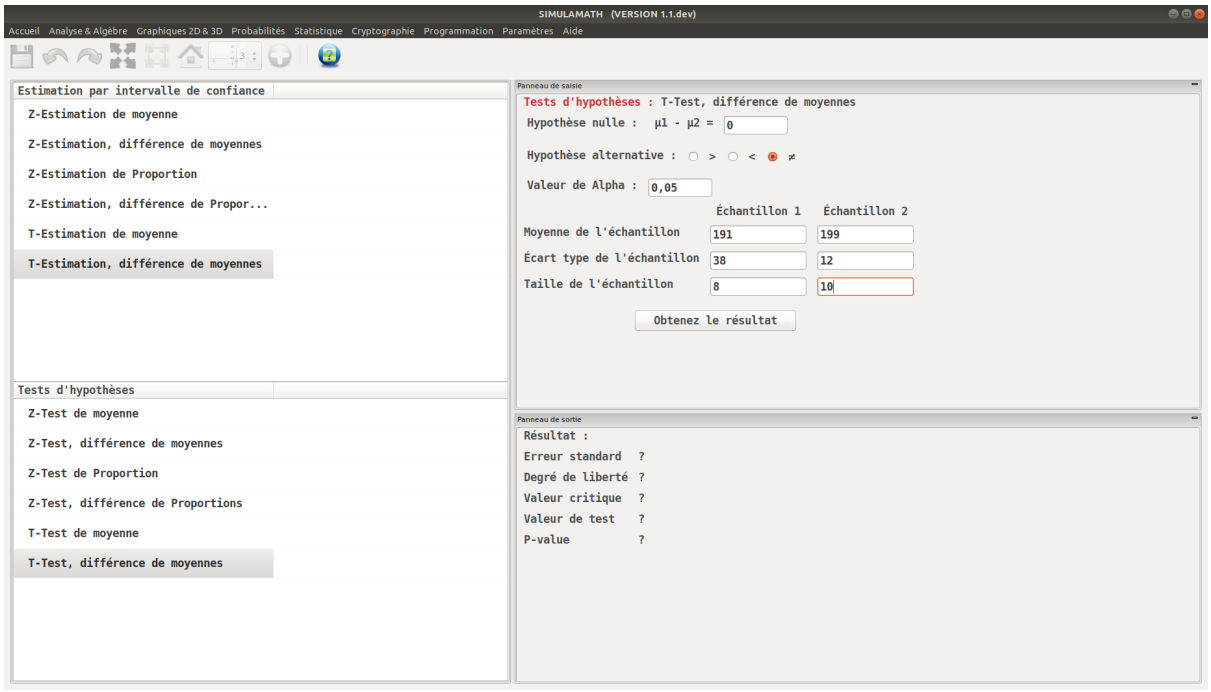
Etape 5 : Résumer le résultat. Il n'y a pas suffisamment de preuves pour étayer l'affirmation selon laquelle la taille moyenne des exploitations est différente

T-Test, difference de moyenne dans SimulaMath

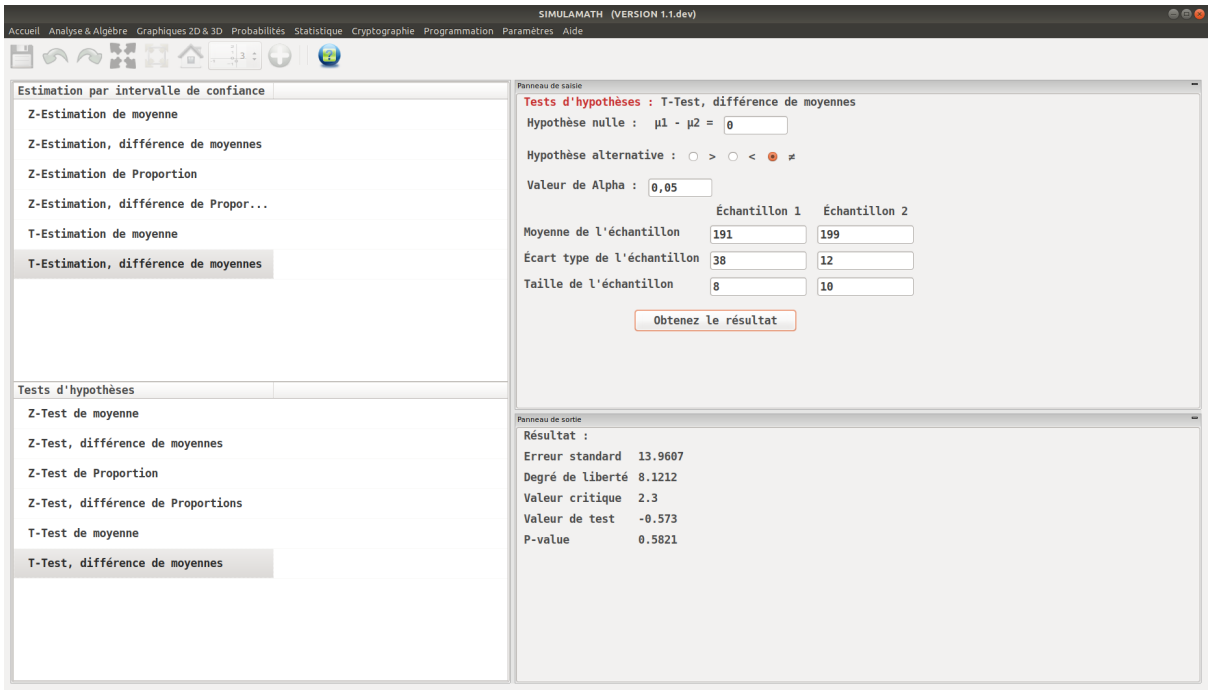
T-Estimation, difference de moyennes dans SimulaMath



Choisir le type de test/estimation que l'on voudrait evaluer dans les panels à gauche, ici T-test, difference de moyenne



Cliquer sur Obtenez le résultat situé juste en dessous de la zone de saisie des varibles. Et Voila, vous avez vous resultats dans le panel de bas à droite.



4.5 Graphiques en 2D

SimulaMath dispose d'une riche zone pour les graphiques 2D. Vous pouvez effectuer les opérations suivantes :

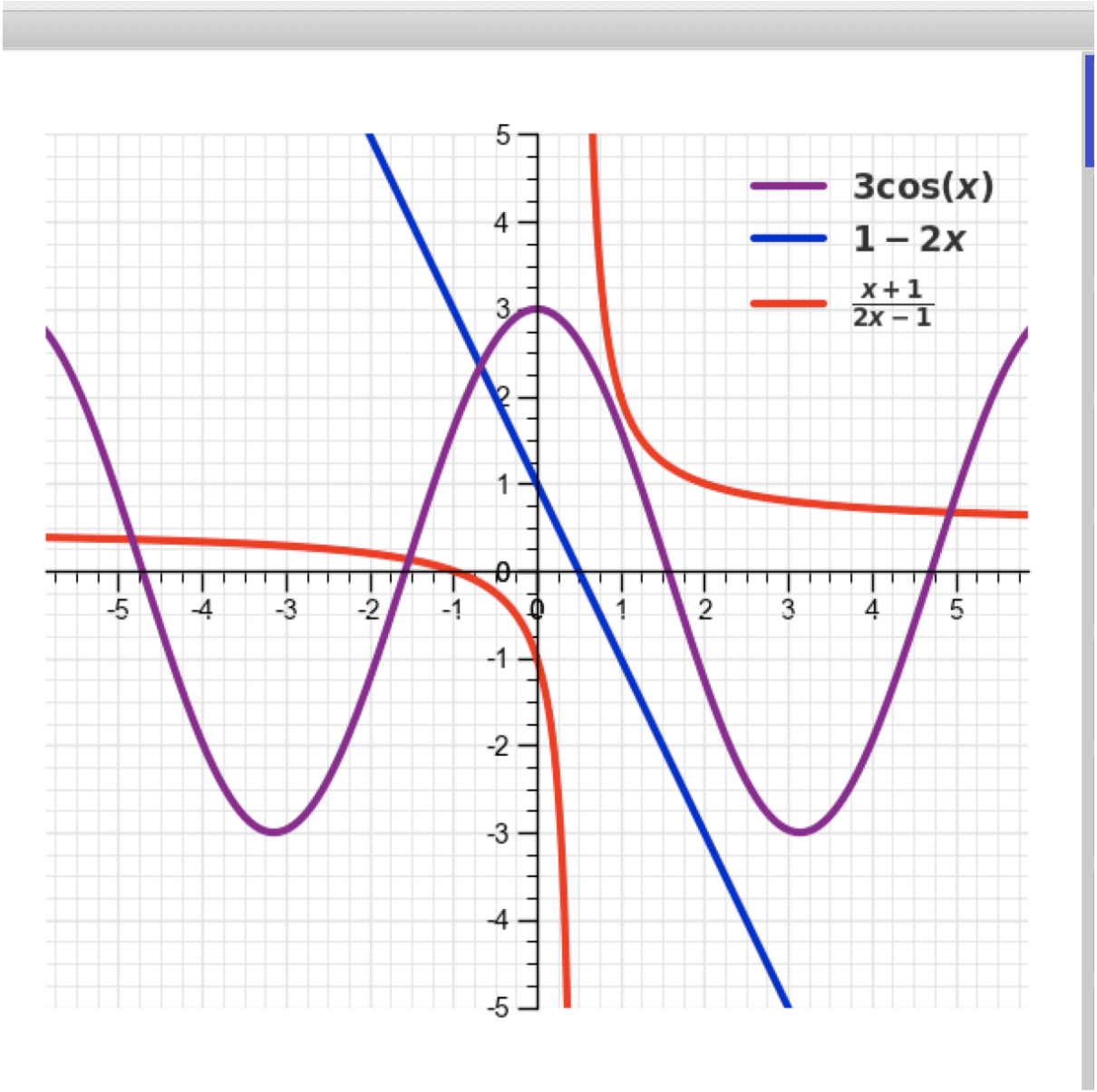
- la représentation graphique d'une fonction à partir d'une expression $f(x)$
- la représentation graphique d'une fonction donnée par une équation implicite : math :*f(x, y) = 0*
- surface donnée par une équation implicite $f(x, y) > 0$, $f(x, y) >= 0$, $f(x, y) < 0$ et $f(x, y) <= 0$.
- la représentation graphique d'une fonction paramétrique $x(t), y(t)$.
- des constructions géométriques pour 100 objets environ : ligne, rayon, segment, cercle, polygone, ...
- choisir différents types de thèmes
- personnaliser le thème de vos graphiques

Note : vous pouvez également tracer votre graphique en utilisant votre propre code Simula.

4.5.1 Fonctions f(x)

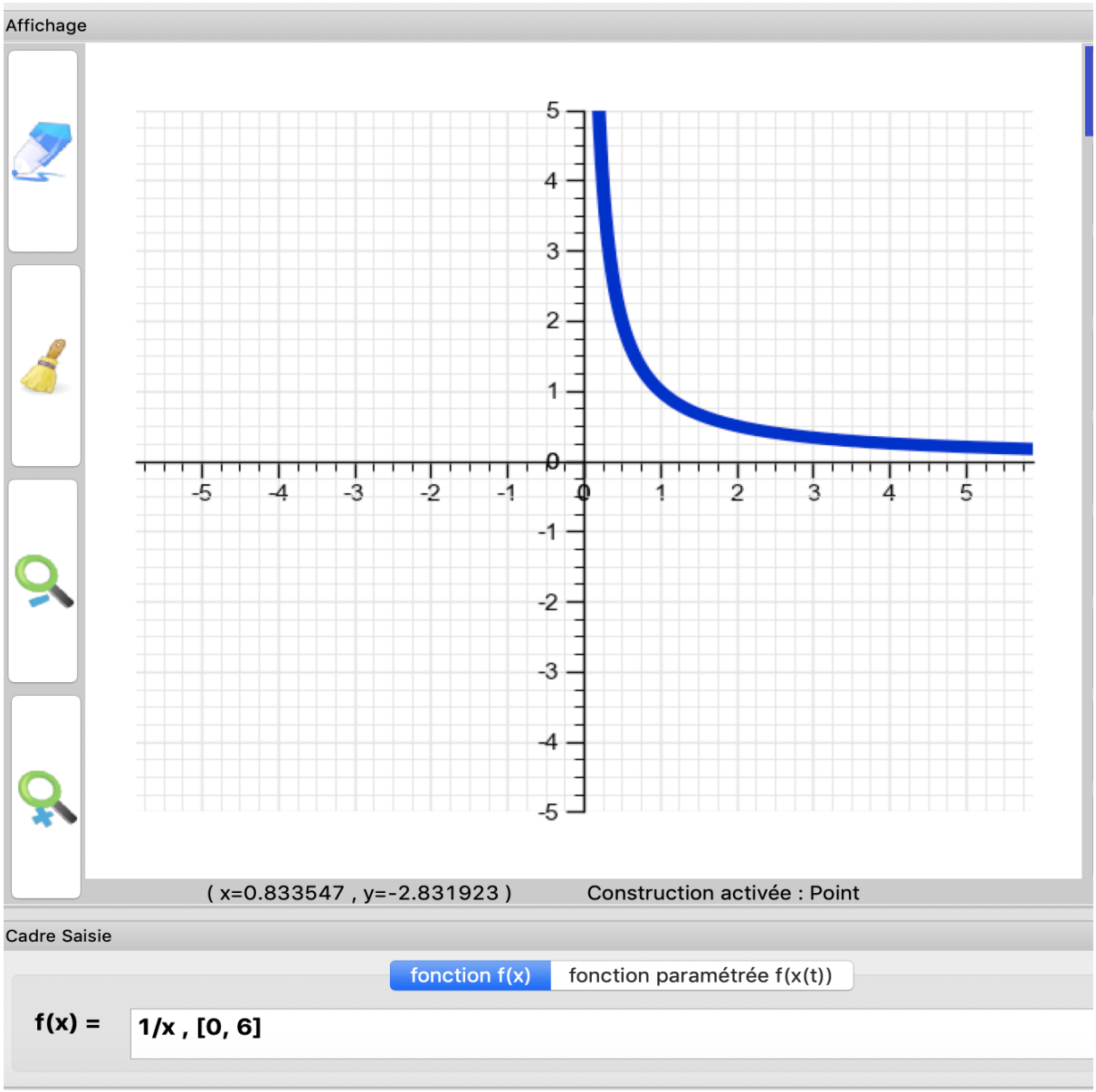
Traçons les graphiques suivants :

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



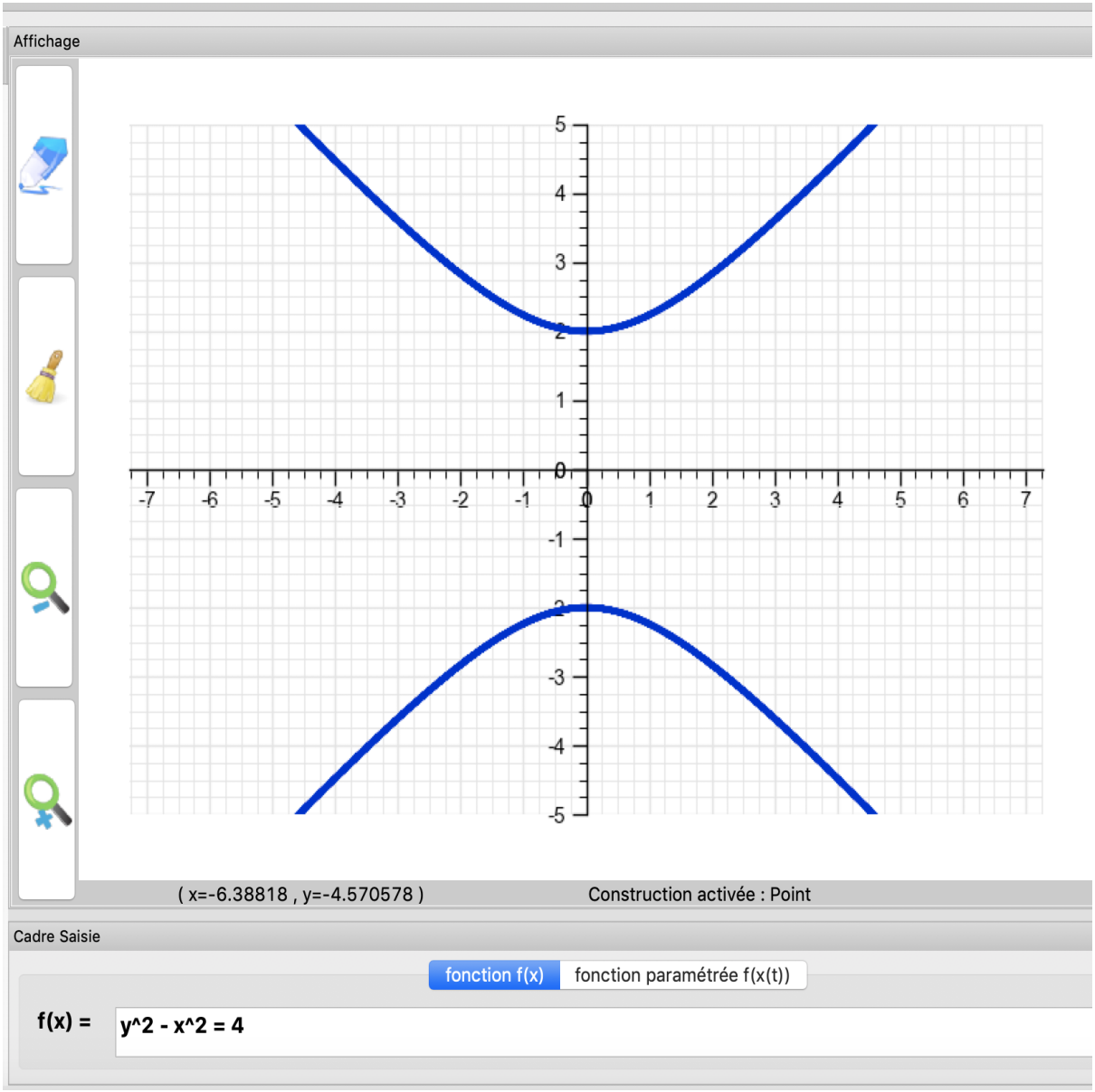
Simula vous permet de tracer une fonction sur un intervalle spécifique. Pour cela, vous devez d'abord saisir l'expression de la fonction suivie d'une virgule puis de l'intervalle.

Traçons le graphe de $\frac{1}{x}$ sur l'intervalle $[0, 6]$.

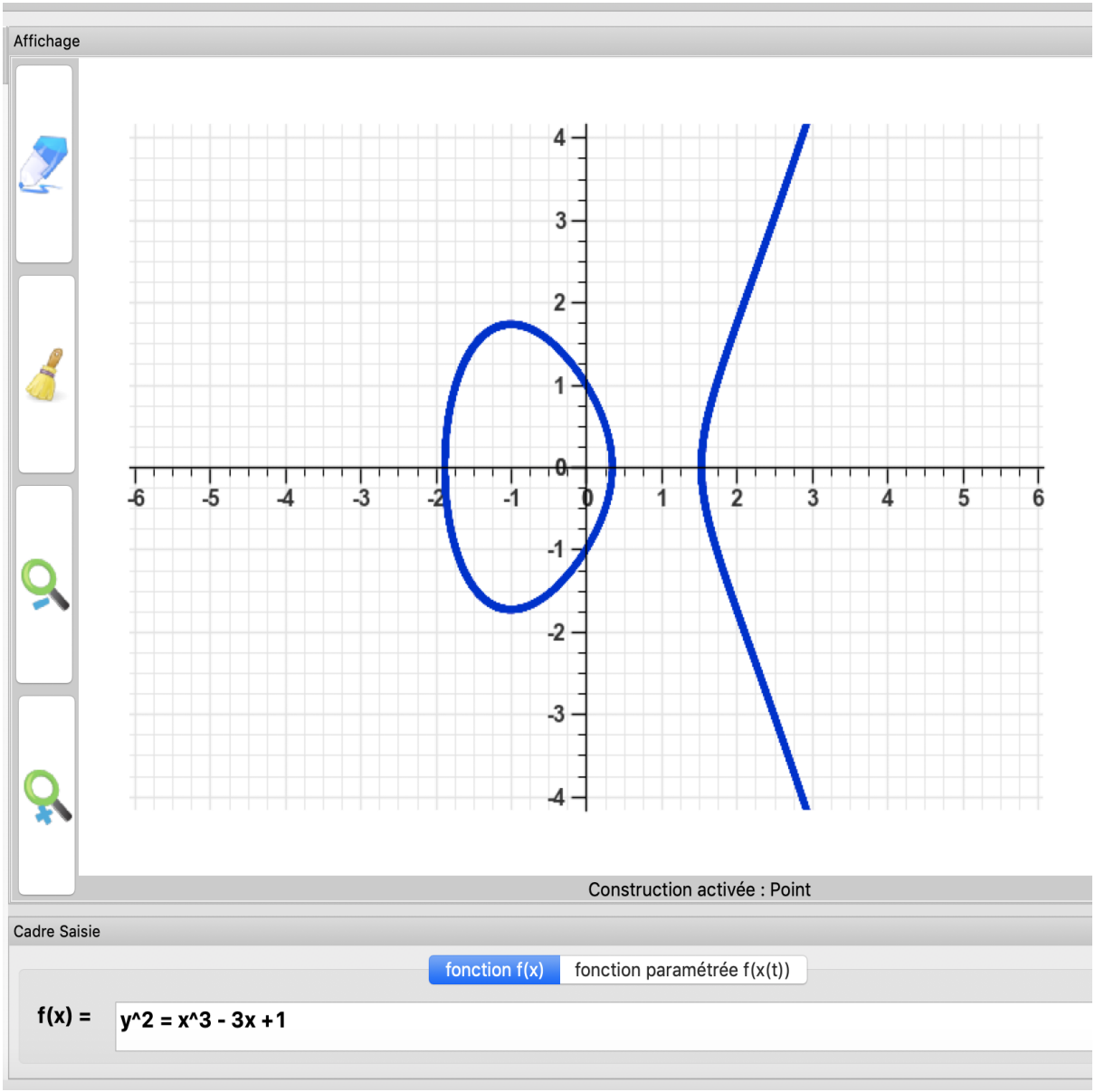


4.5.2 Courbes de fonctions implicites

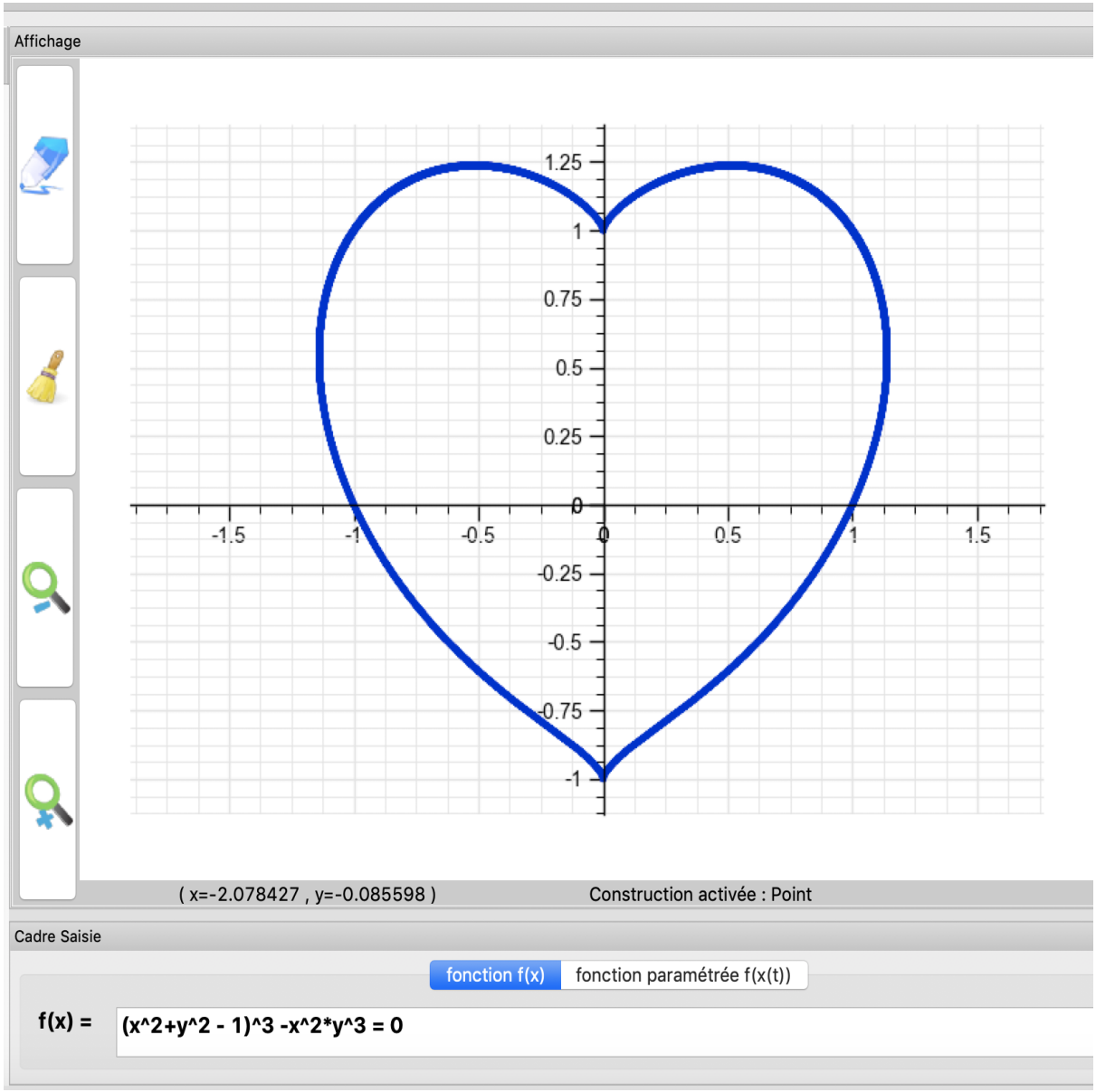
— Graphe de $y^2 - x^2 = 4$



— Graphe de $y^2 = x^3 - 3x + 1 = 0$

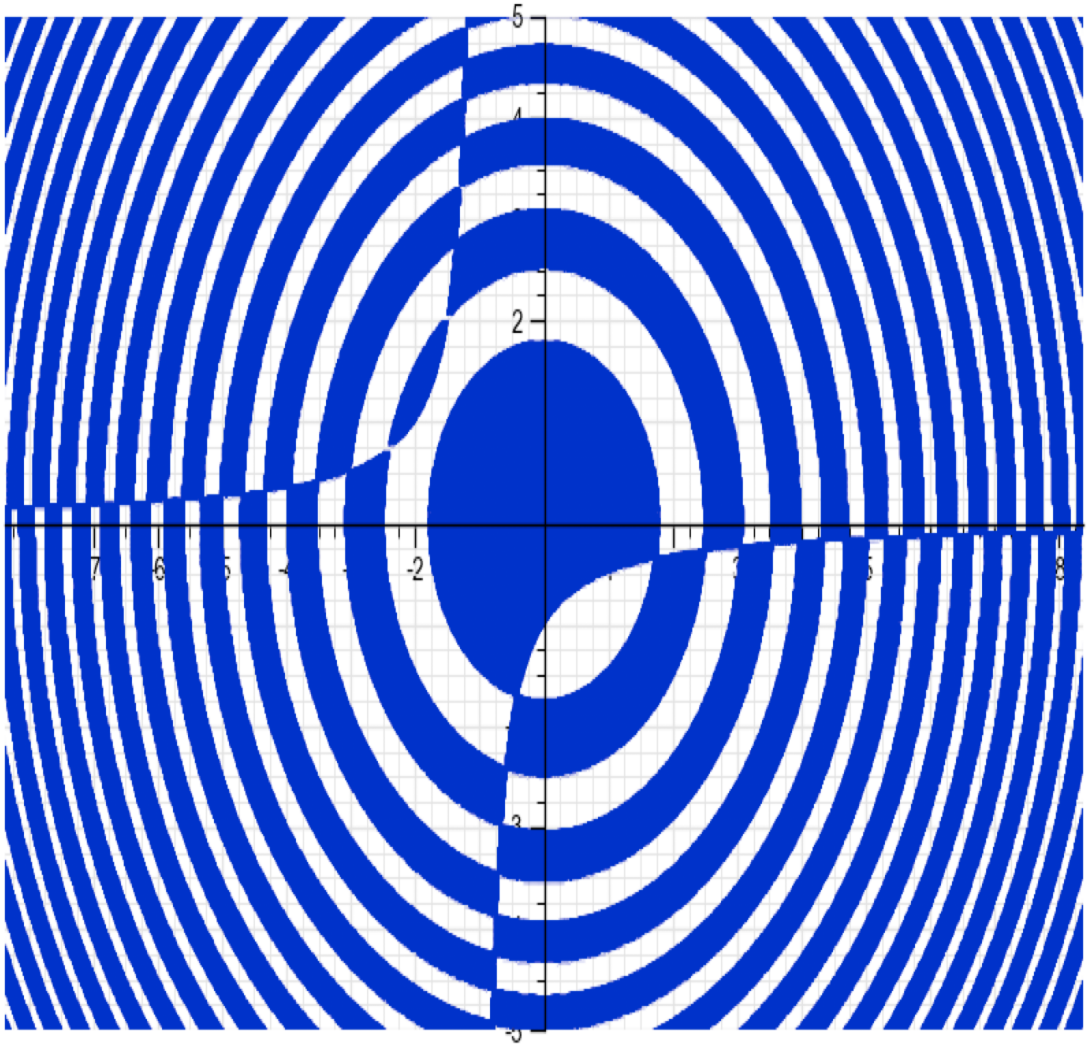


— Graphe de $(x^2 + y^2 - 1)^3 - x^2y^3 = 0$



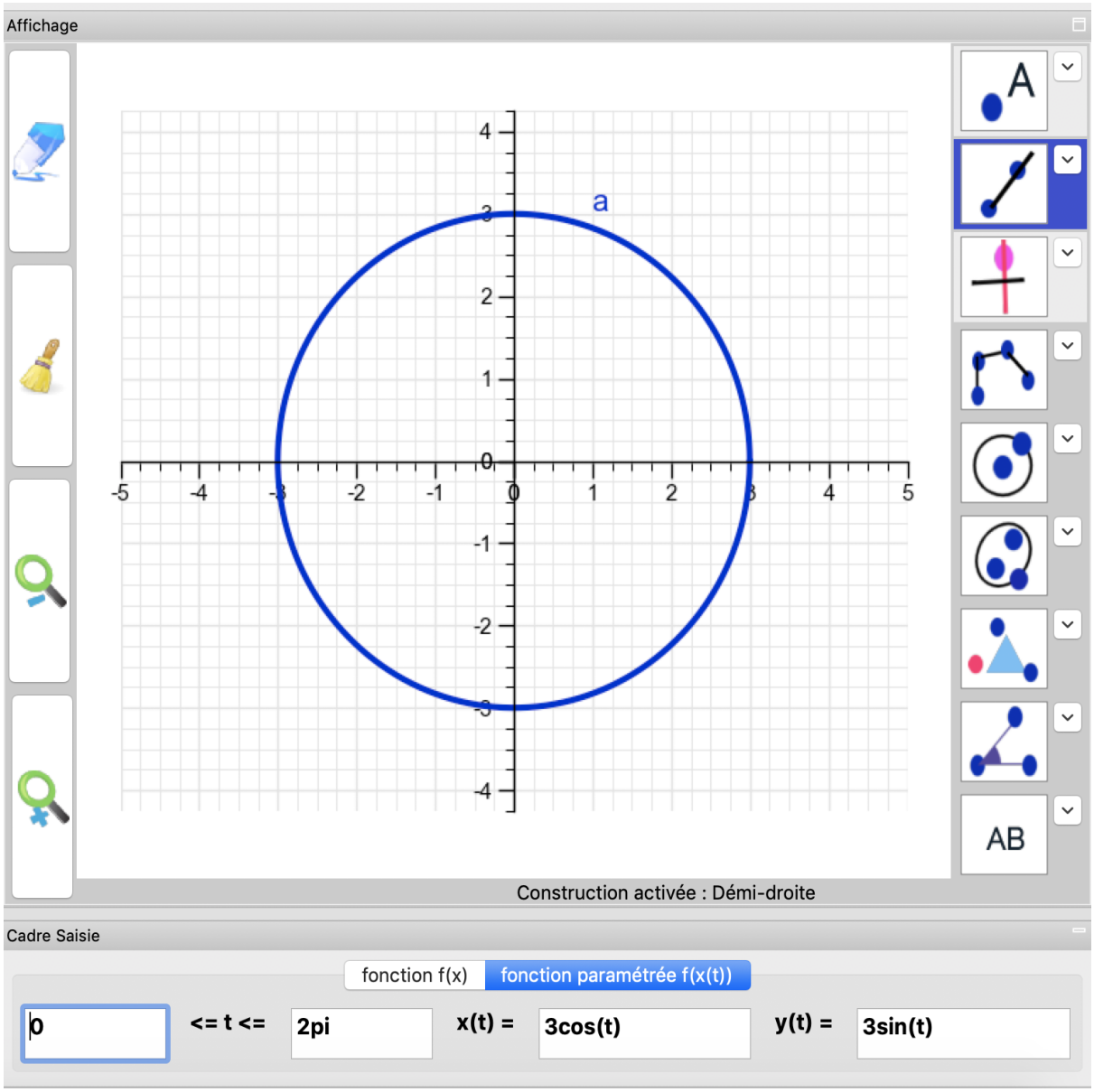
Graphe de

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

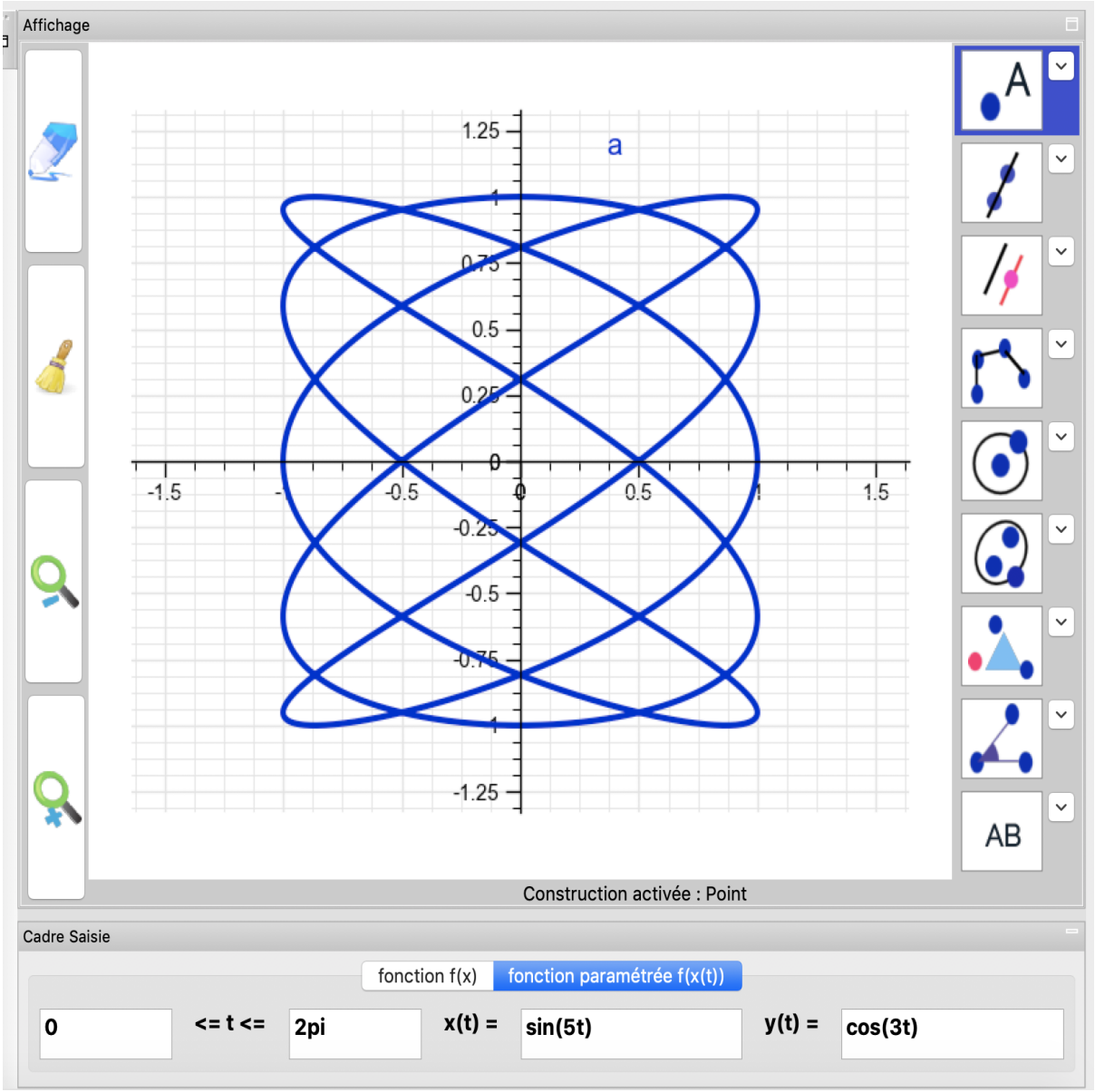


4.5.3 Fonctions paramétrées

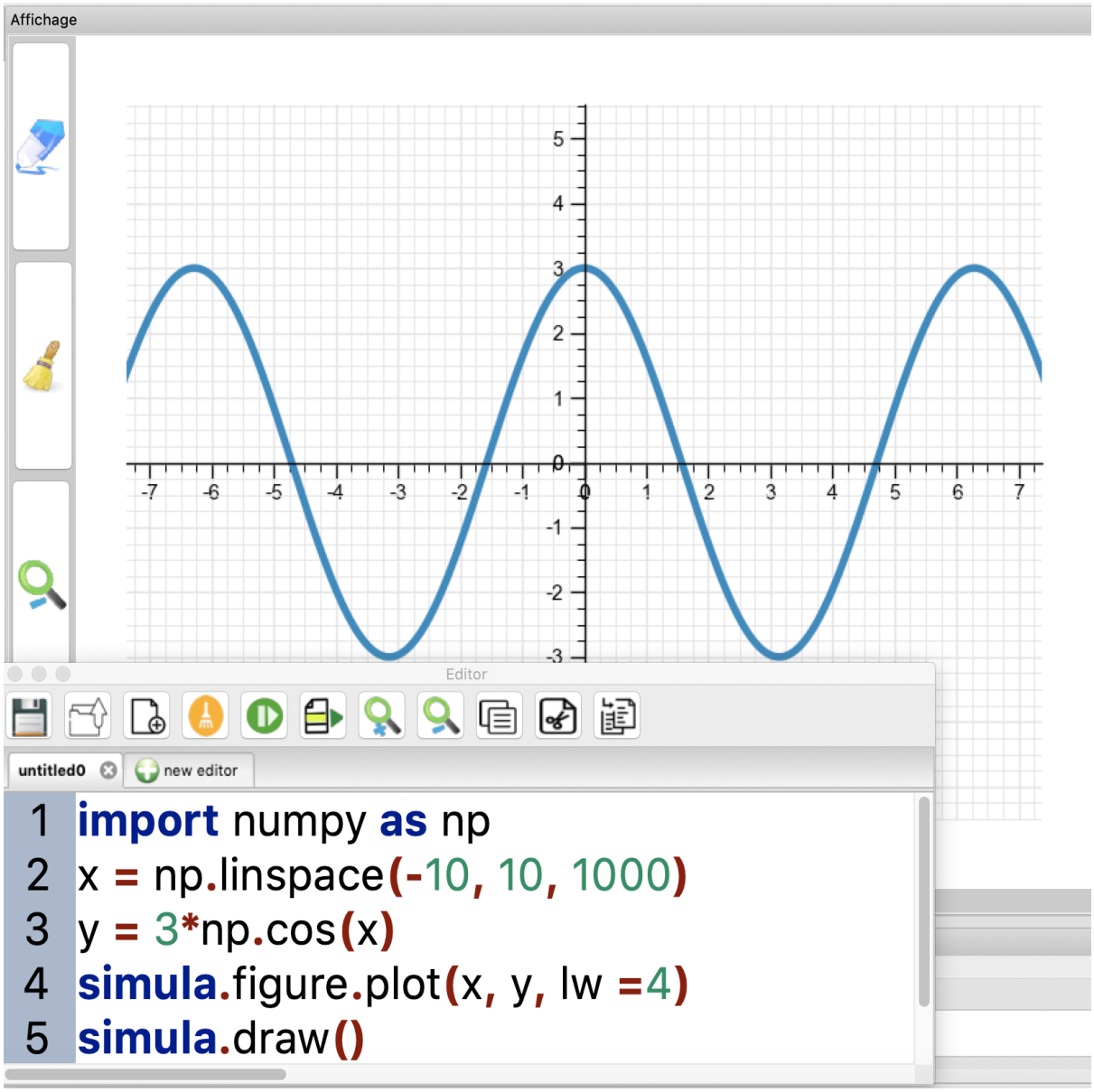
— Traçons la courbe de la fonction paramétrique $x(t) = 3\cos(t); y(t) = 3\sin(t)$ où $t \in [0, 2\pi]$.



— Figures de Lissajous : $x(t) = \sin(5t); y(t) = \cos(3t)$ où $t \in [0, 2\pi]$.

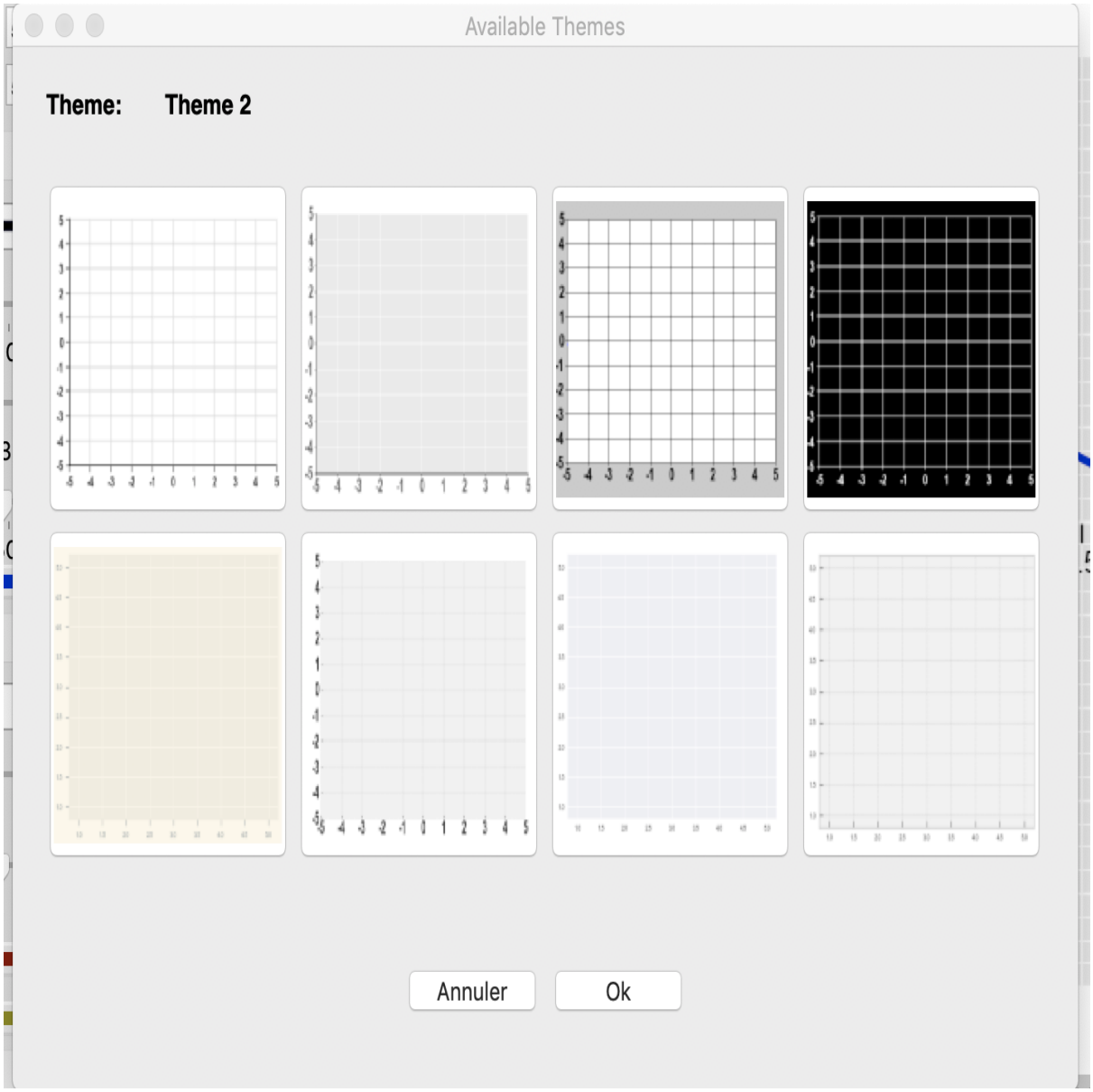


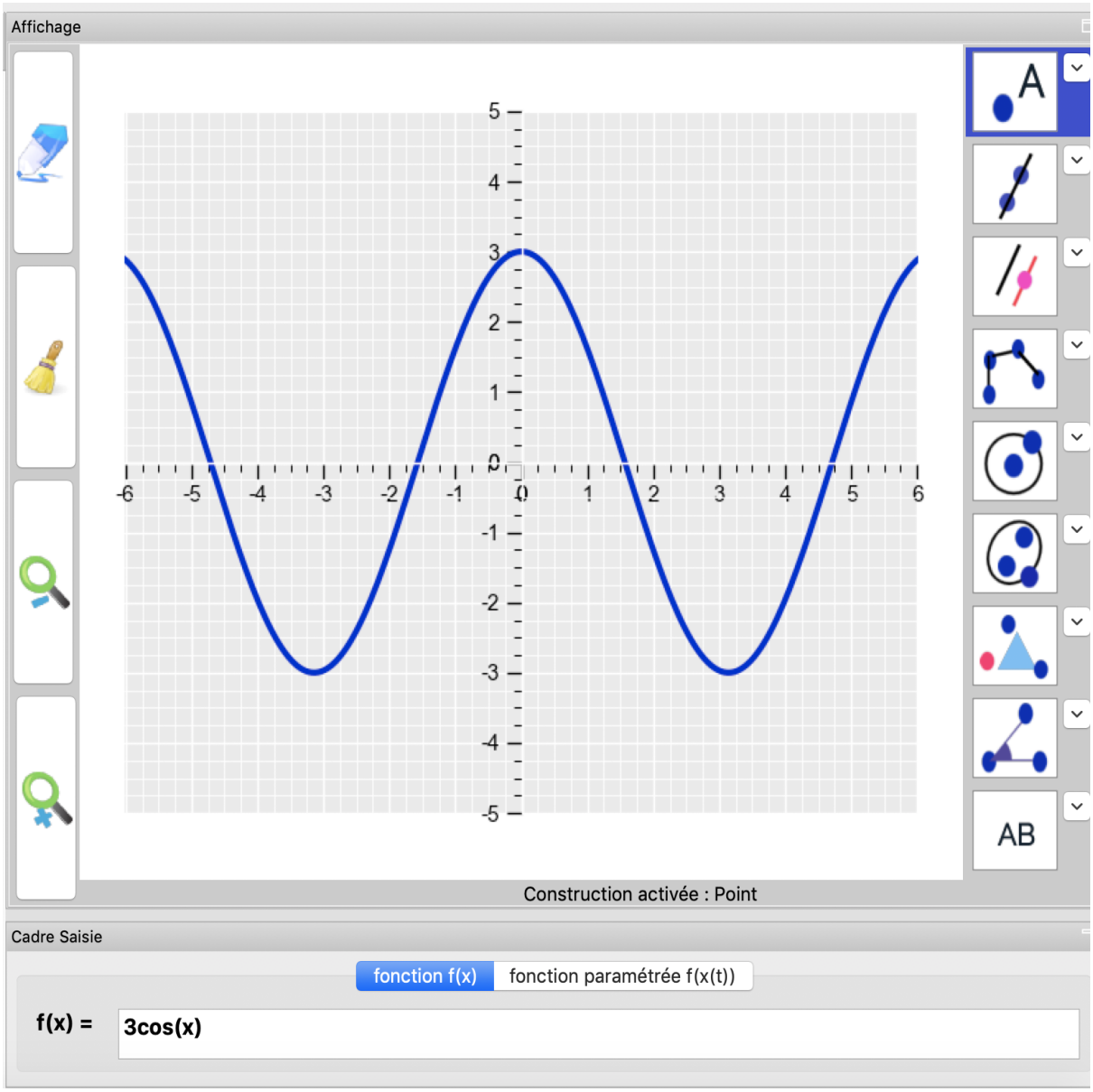
4.5.4 Graphiques 2D et programmation



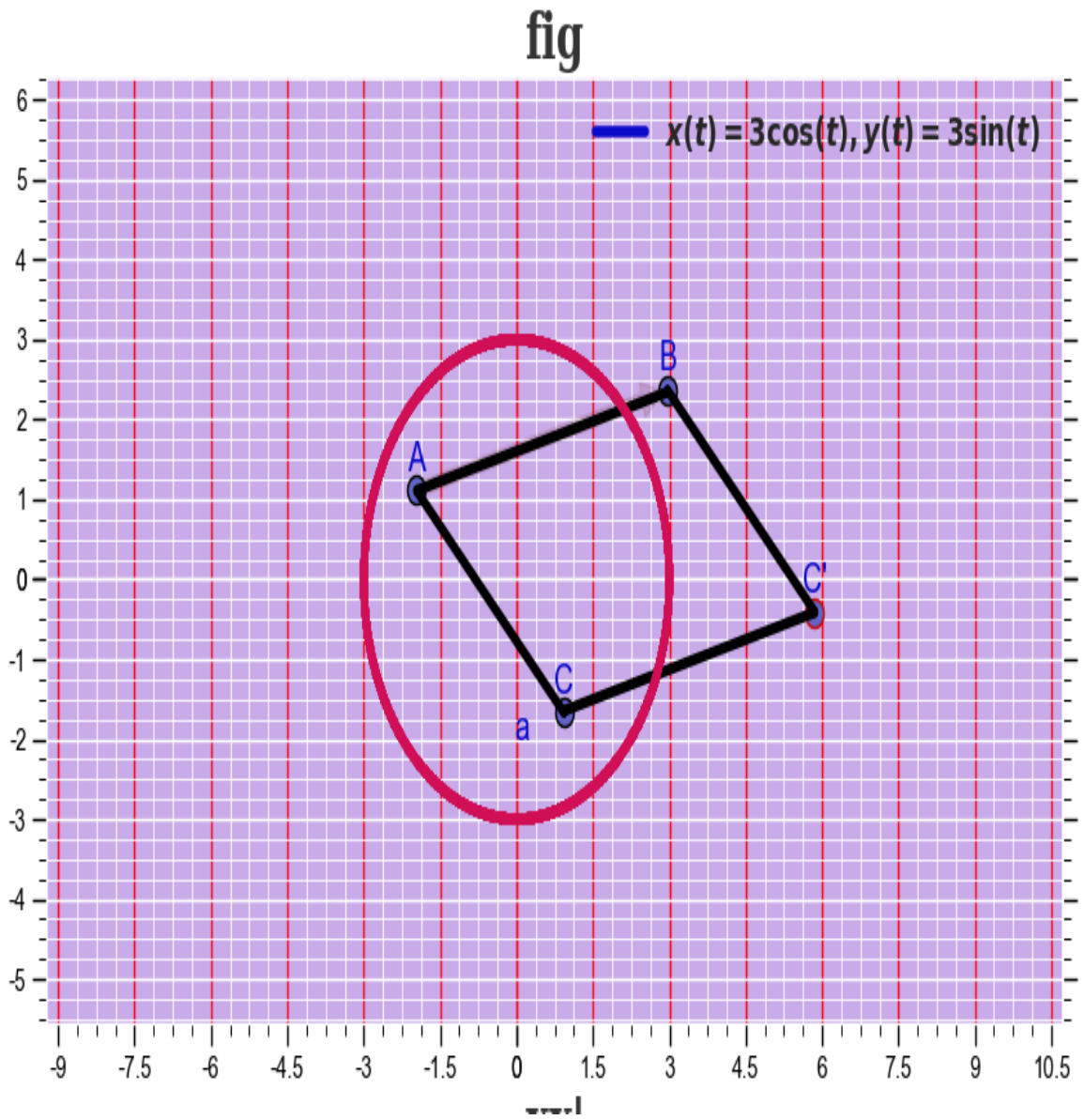
4.5.5 Thèmes de vos graphiques

Vous pouvez choisir entre 8 thèmes personnalisés.


Available Themes
Theme:
Theme 2
Annuler
Ok



Vous pouvez également personnaliser votre propre thème.



4.5.6 Objets géométriques en 2D

Vous pouvez construire environ une centaine d'objets géométriques :

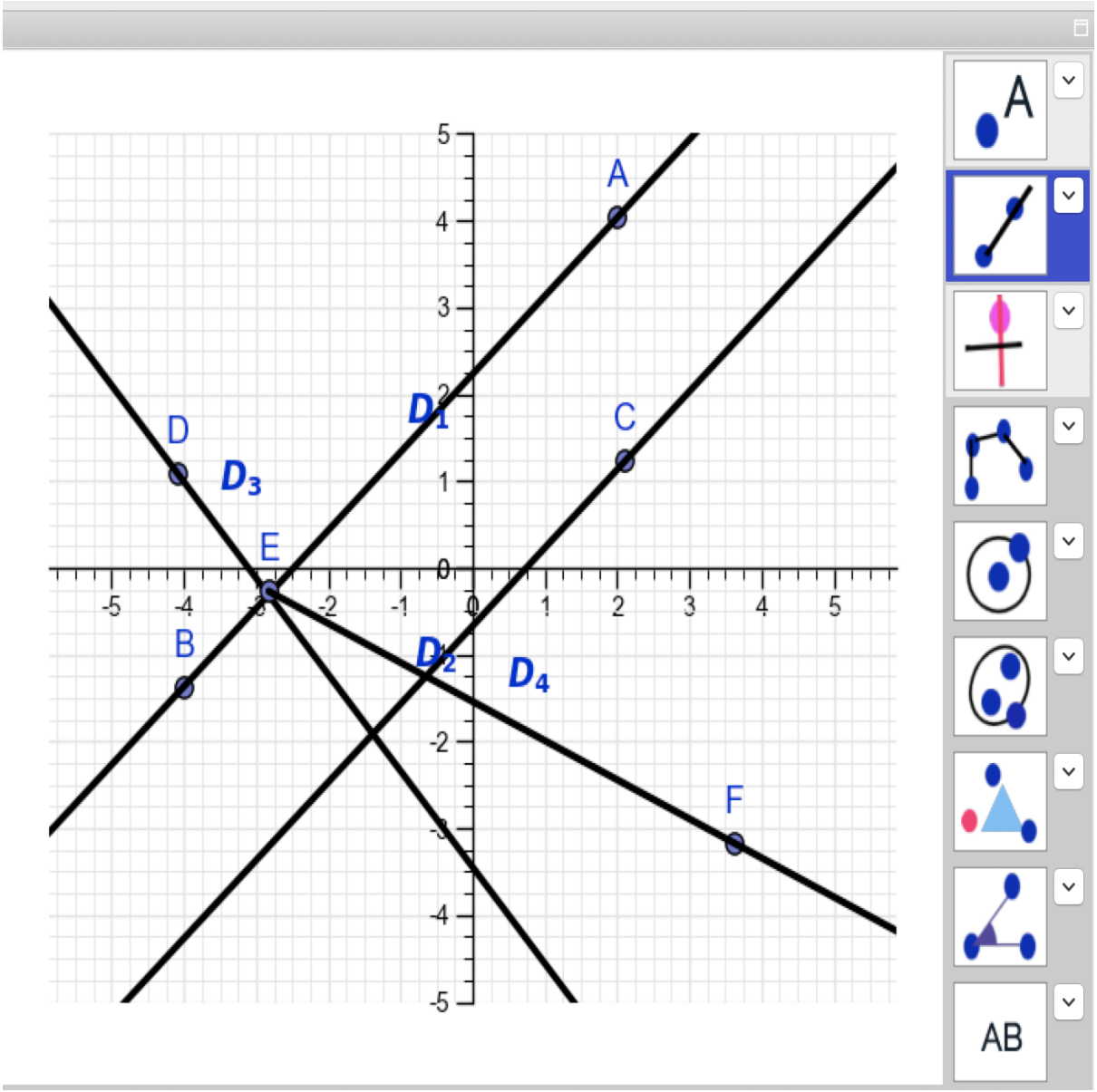
- Points
- Droites,
- Démi-droites,
- Segments,
- Cercles,
- Demi-Cercles,
- Arcs
- Secteurs
- Polygones,

— Droites parallèles,
— Droites perpendiculaires,
— Vecteurs,
— Angles,
— Bissectrices,
— Ellipses,
— Paraboles,
— Hyperboles,
— Rotation,
— Homothétie,
— Translation,
— Symétrie centrale et axiale,
— Surfaces,
— Barycentre
— Textes,
— Images,
— etc.

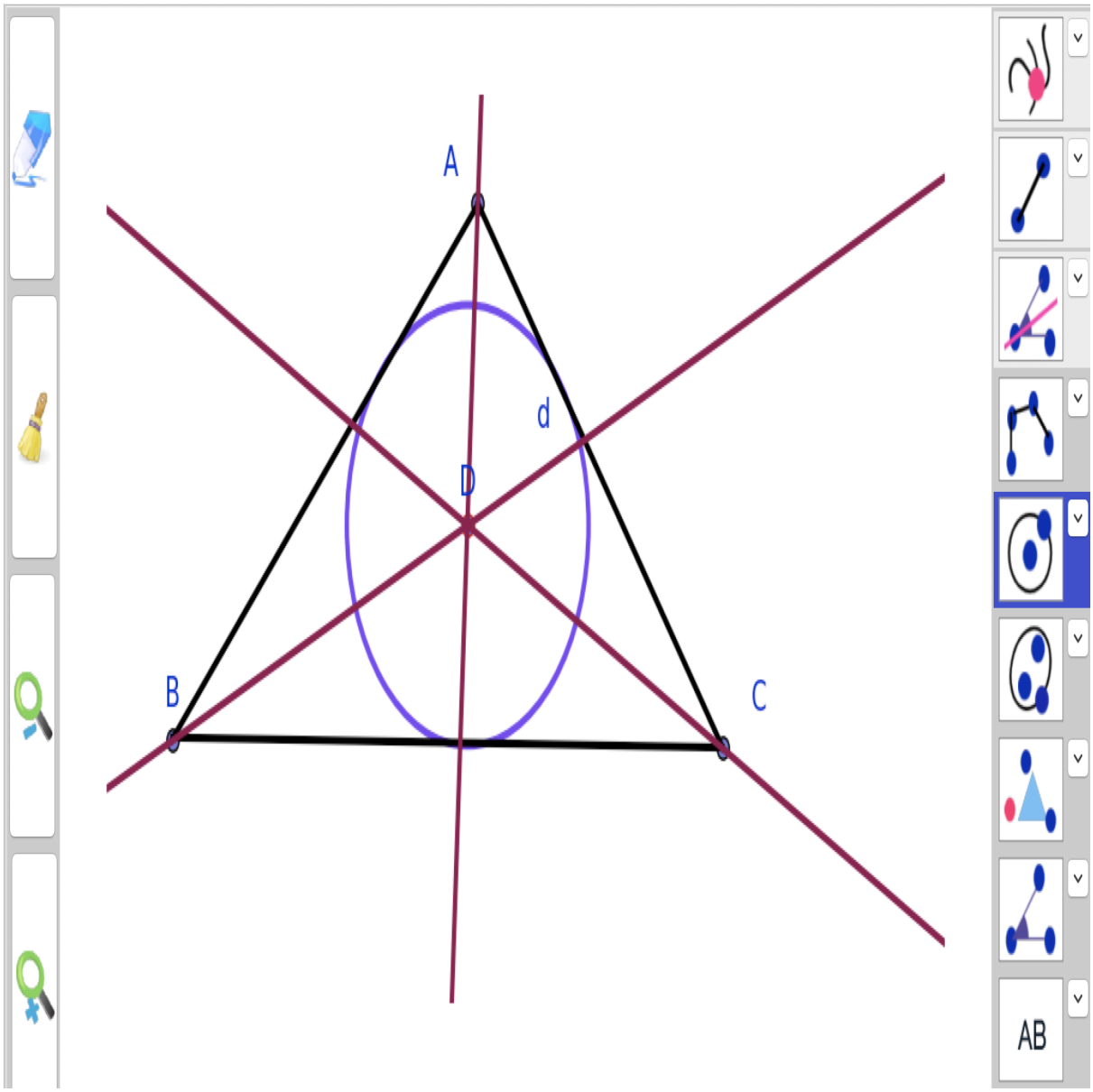
Note :

— Toutes les constructions d'objets géométriques commencent et se terminent par un double-clic.
— Pour un polygone ou un barycentre, double-cliquez sur le premier point, puis un clic sur chaque point intermédiaire, et enfin double-cliquez sur le dernier point.

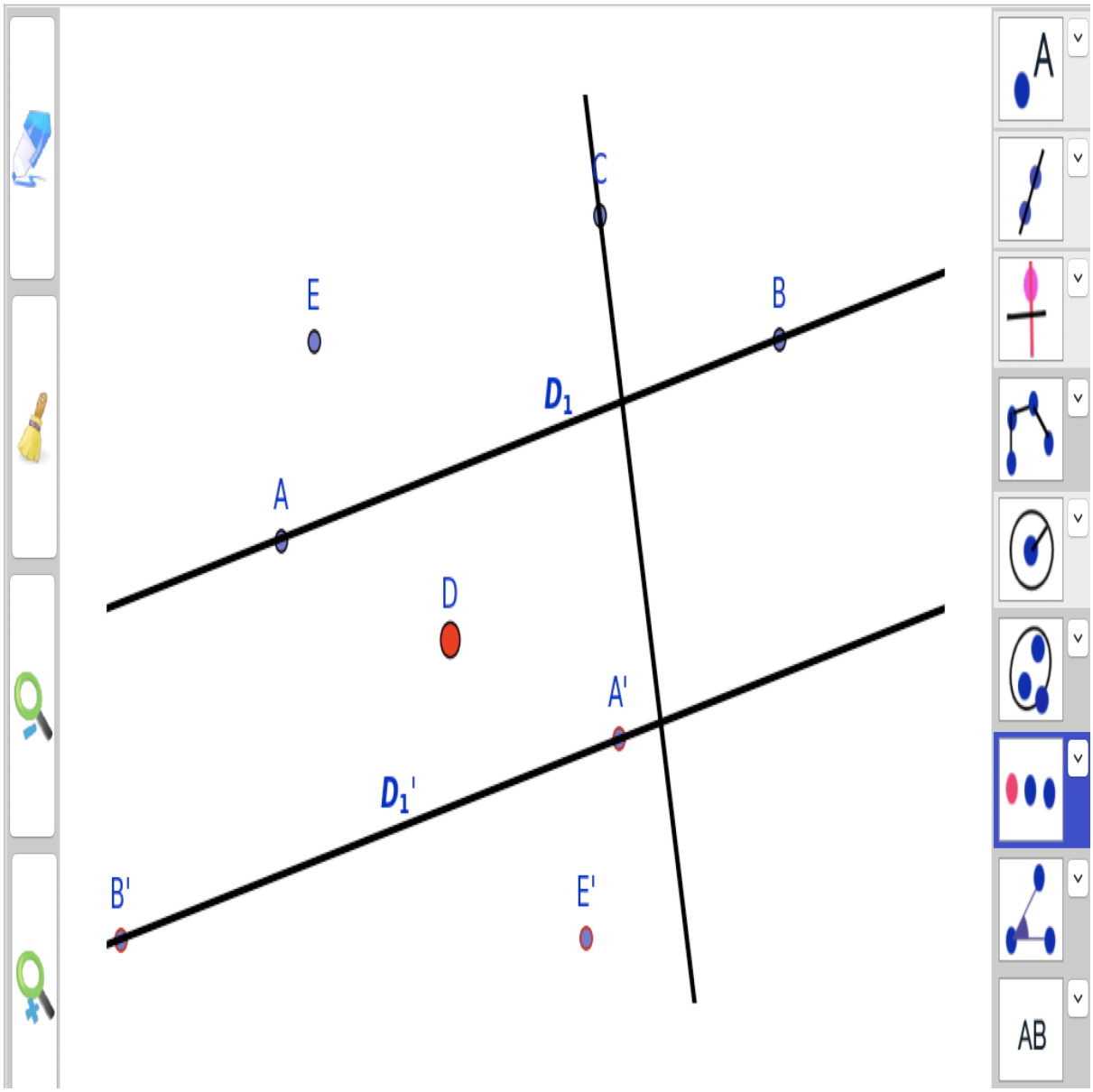
Droites, segments et démi-droites



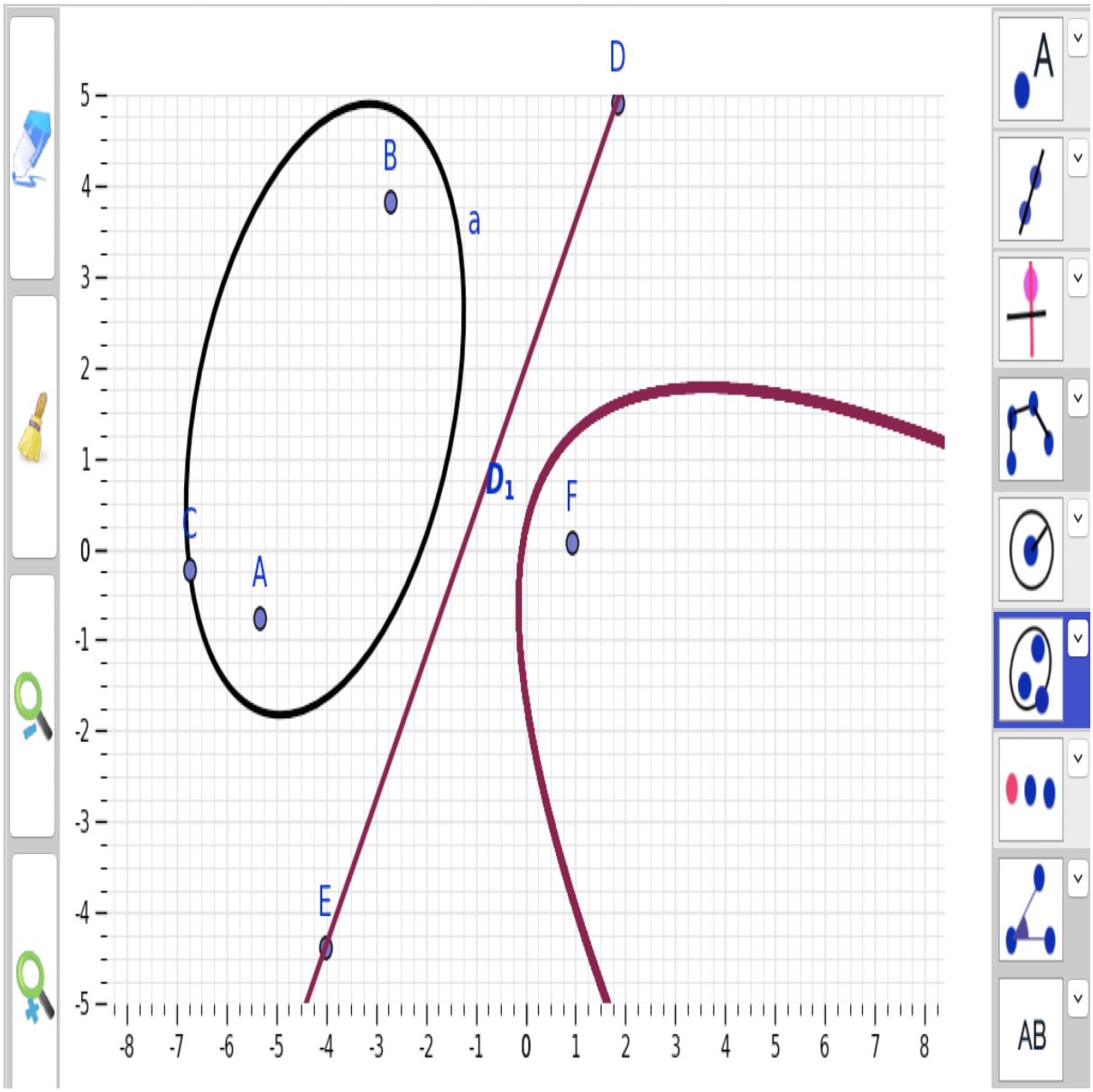
Triangles et Cercles



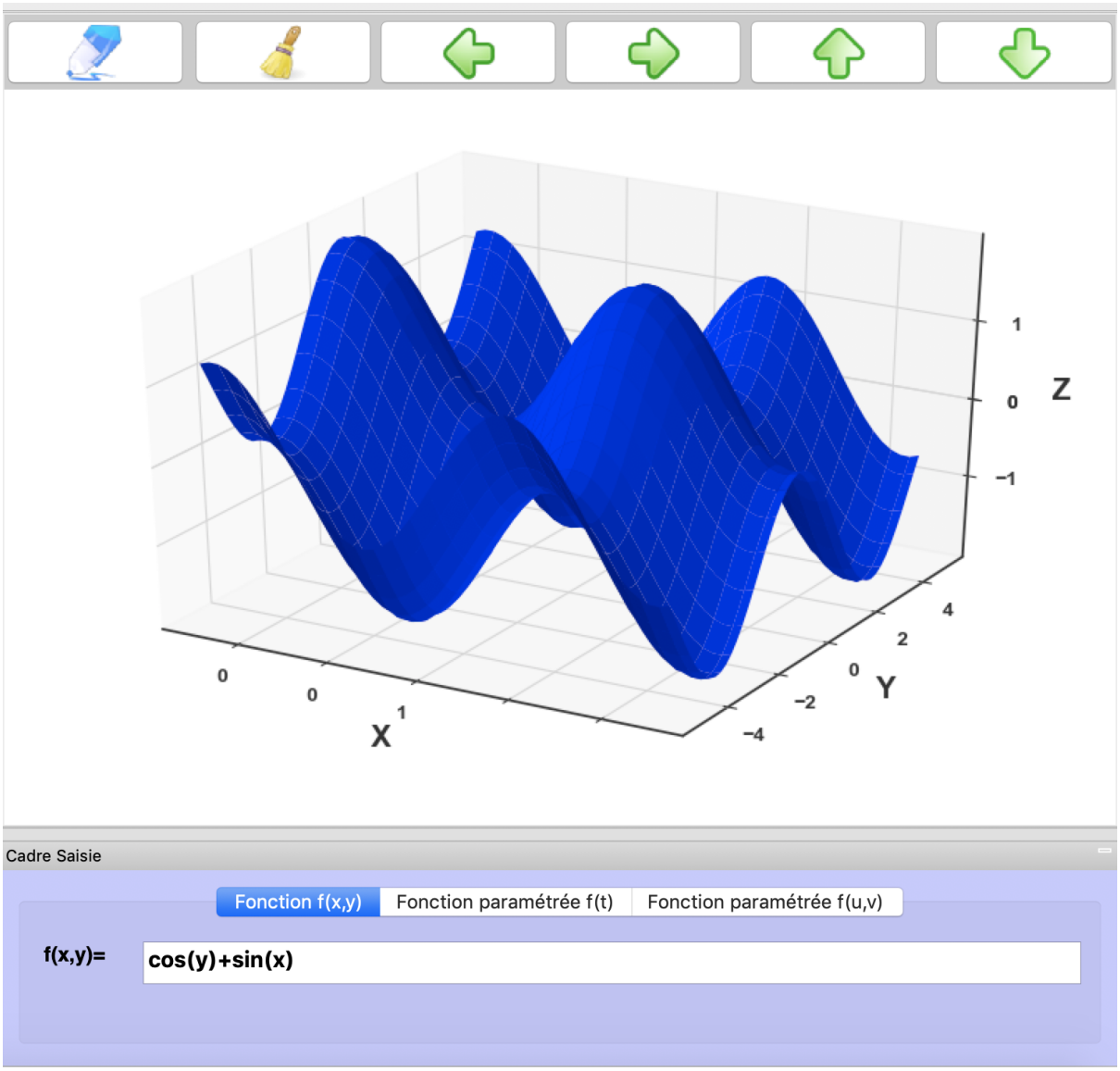
Transformations : Réflexions, rotations, translations et homothéties



Ellipses, paraboles et hyperboles

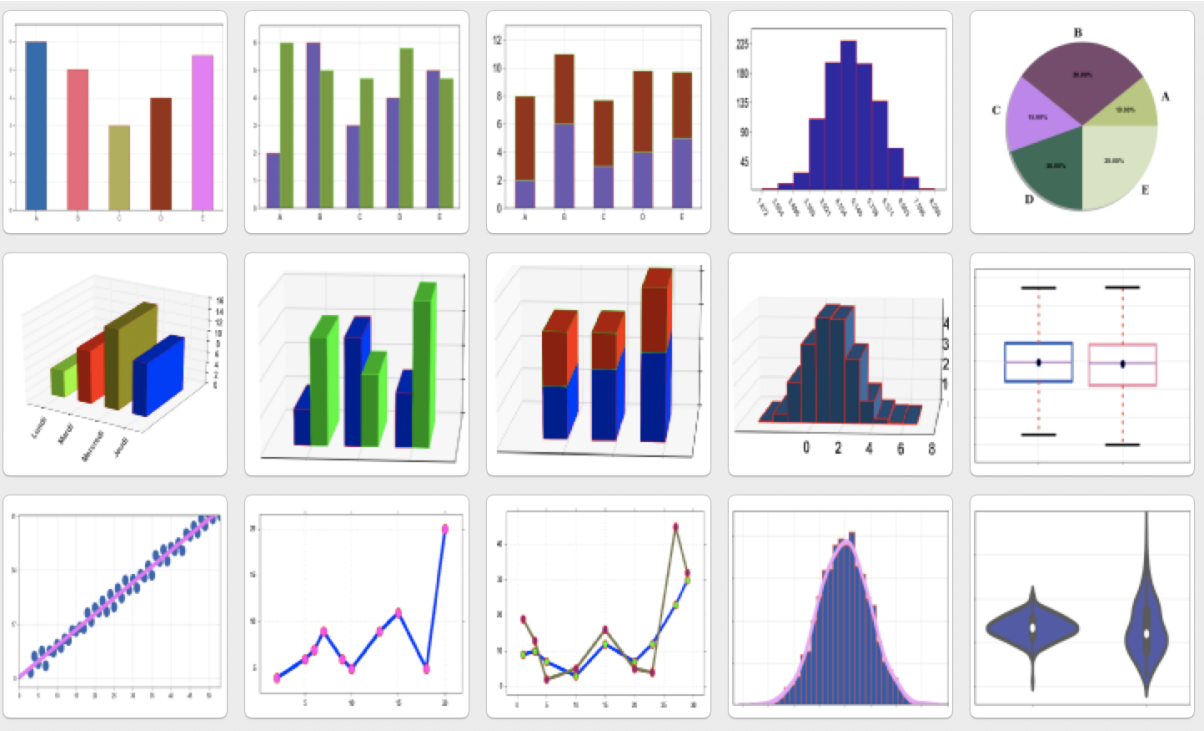

D
B
a
C
A
D1
F
E
AB

4.6 Graphiques en 3D



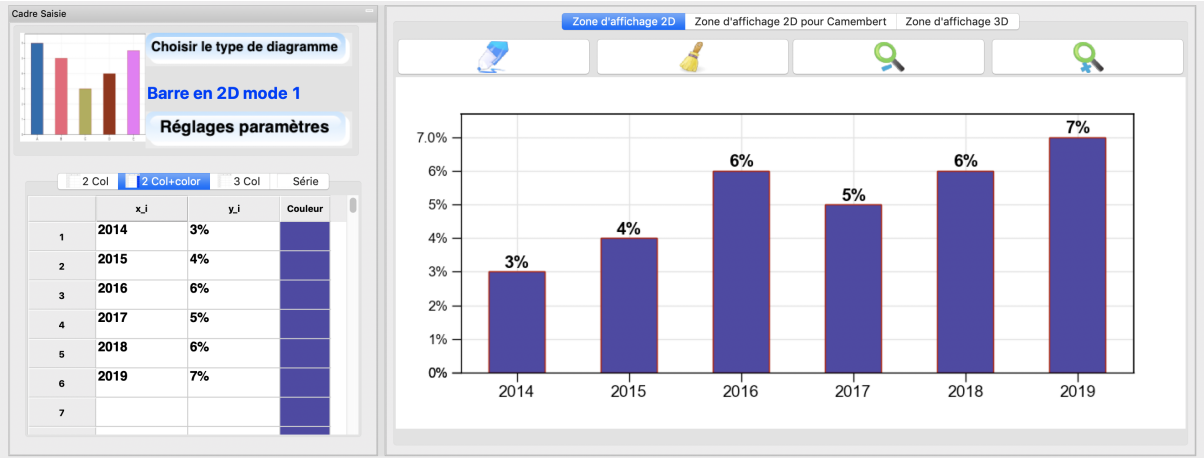
4.7 Diagrammes

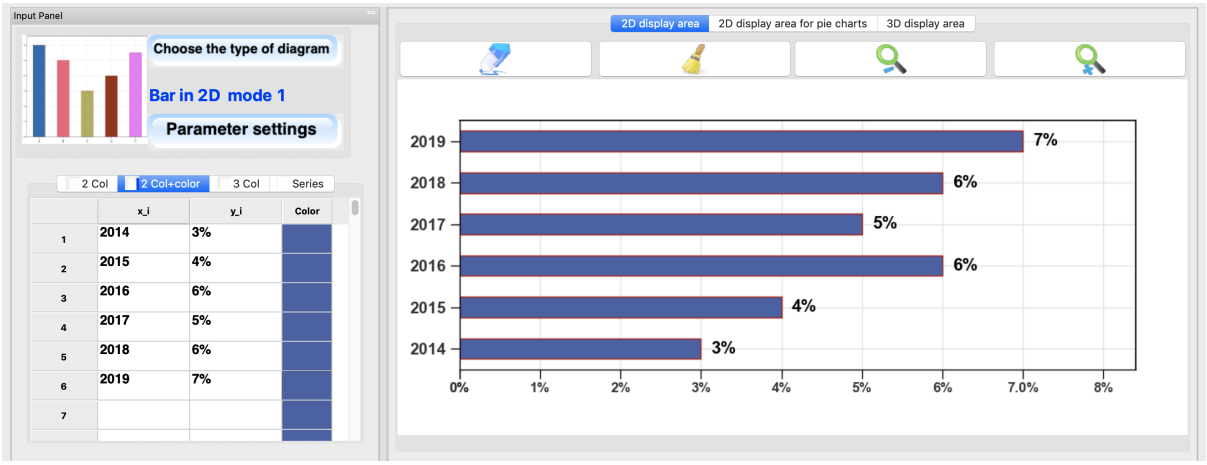
Pour tracer un diagramme, il faut d'abord choisir le type de diagramme : il en existe 15.



4.7.1 Diagrammes en barres

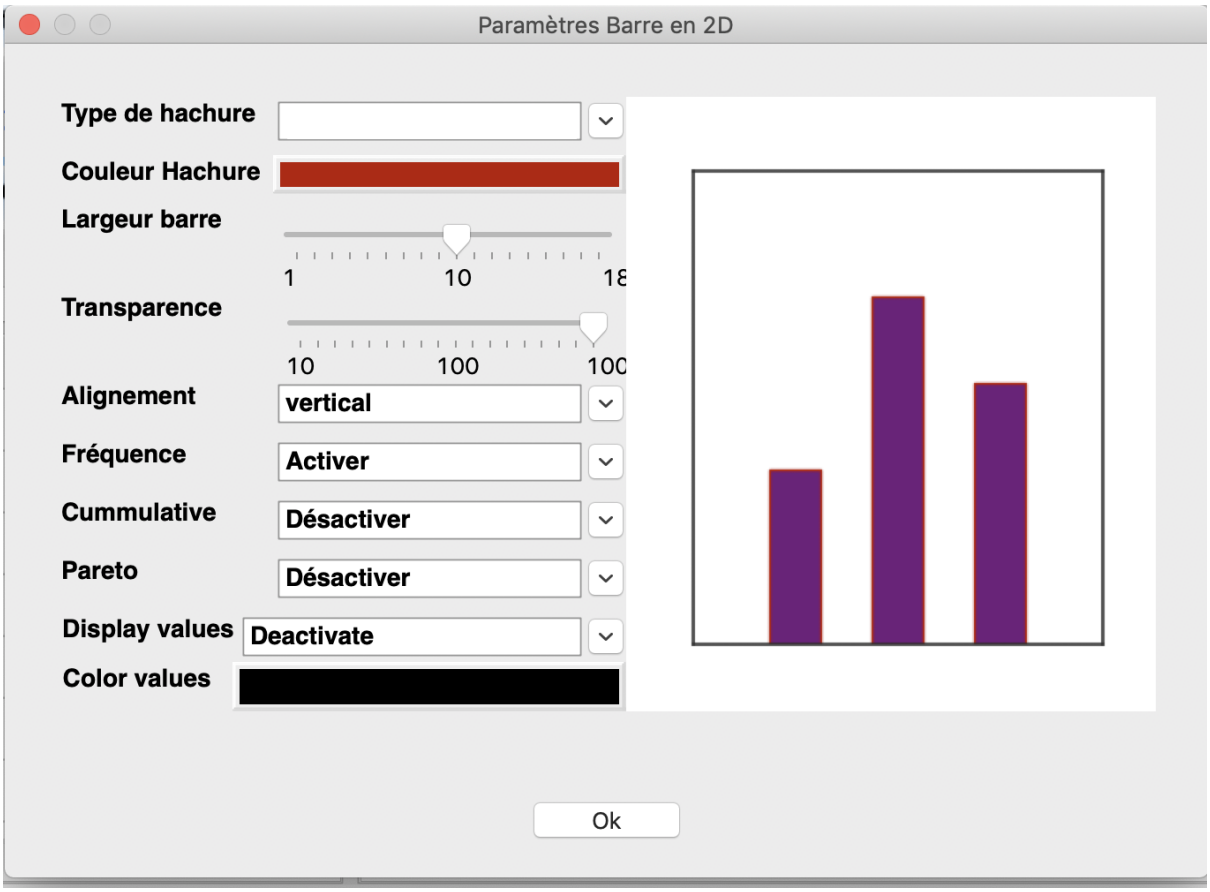
Diagrammes en barres à deux dimensions

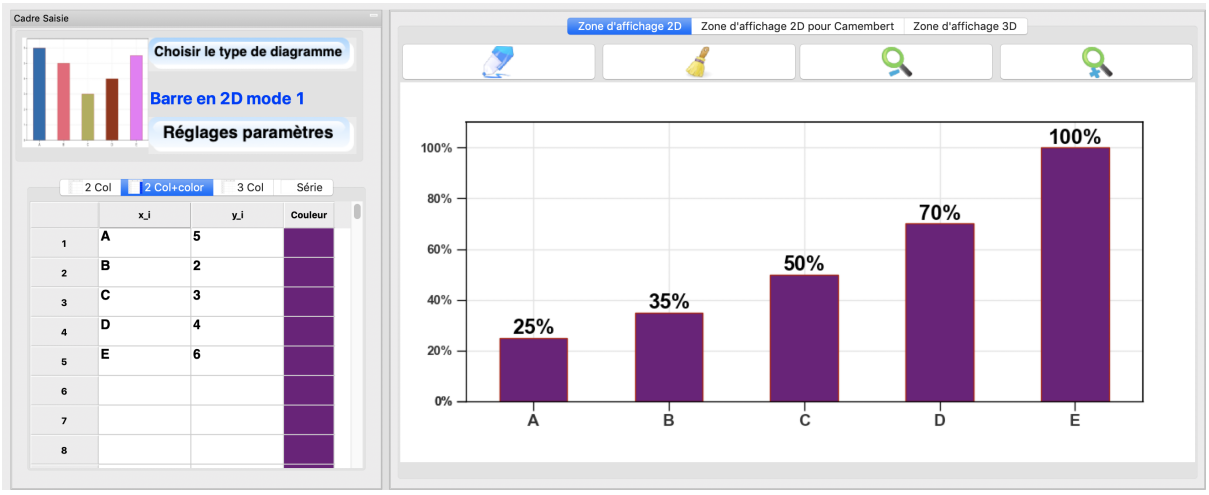




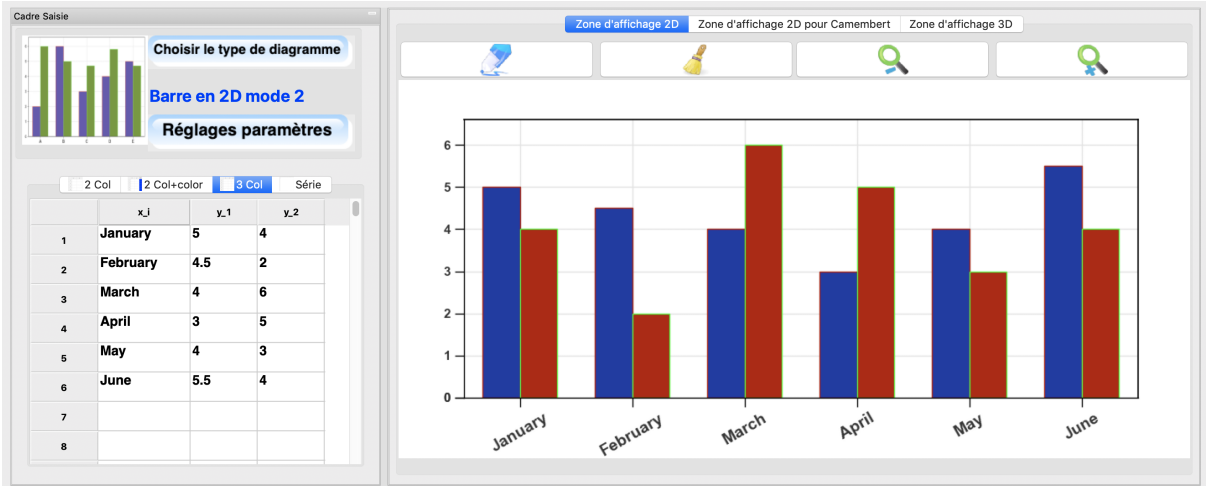
Note : En plus du diagramme à barres des effectifs, vous pouvez également tracer le diagramme à barres des fréquences et des effectifs cumulés croissants ou décroissants. Vous pouvez également tracer le diagramme de Pareto.

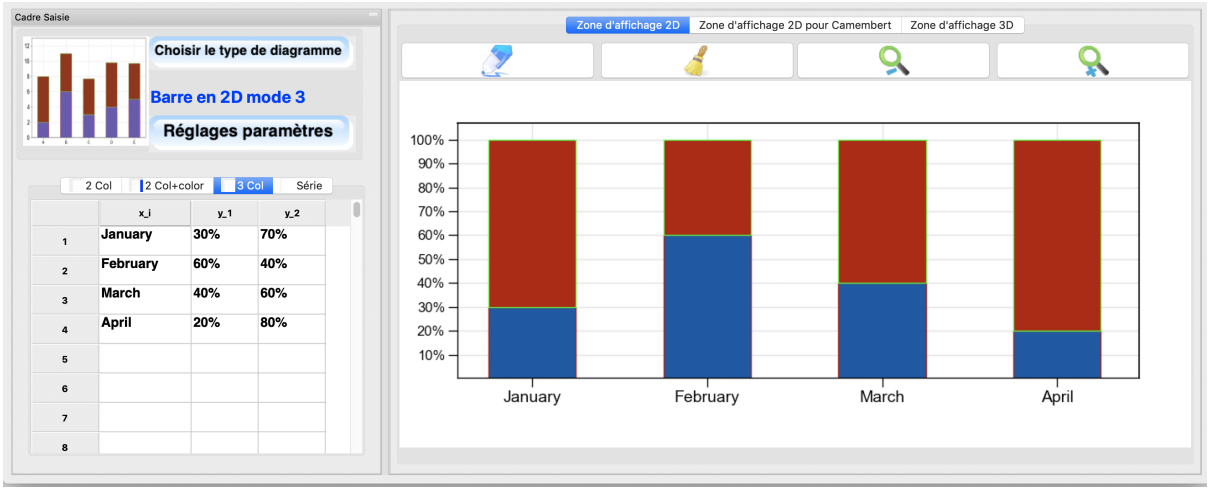
Pour tracer le diagramme des fréquences cumulées (ordre croissant), cliquer d'abord sur le bouton « Paramètres », le panneau de configuration s'ouvre, pour les fréquences sélectionner « Activer » et pour cumulative sélectionner « Ordre croissant ».

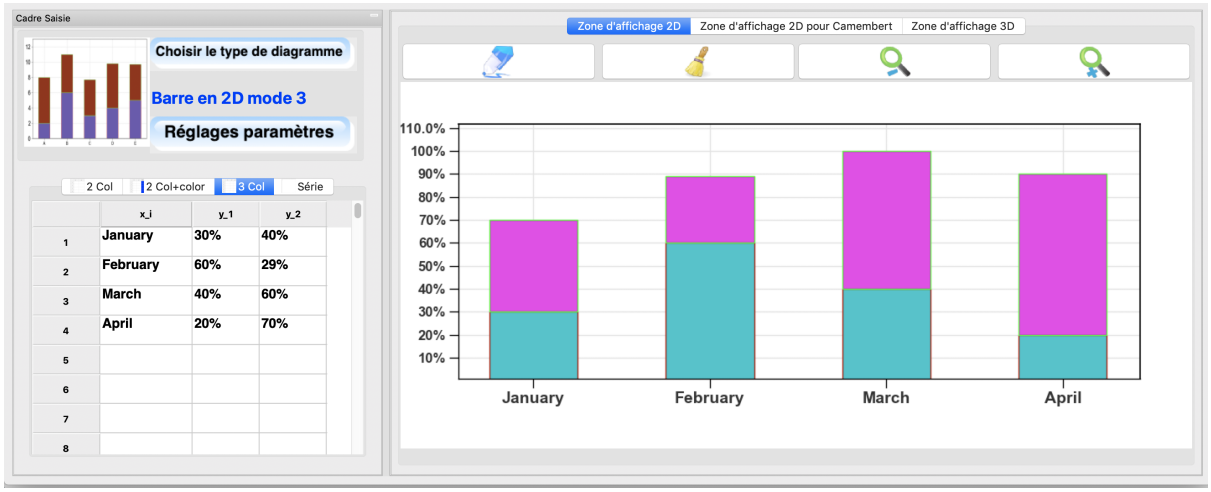




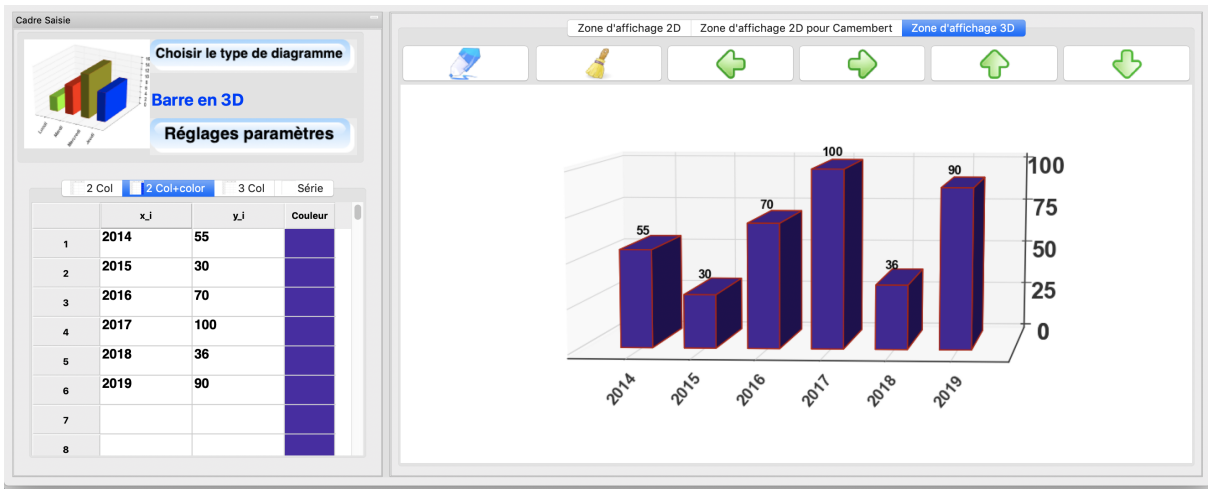
Note : Vous pouvez également tracer deux diagrammes en parallèle, soit en les juxtaposant, soit en les alignant.

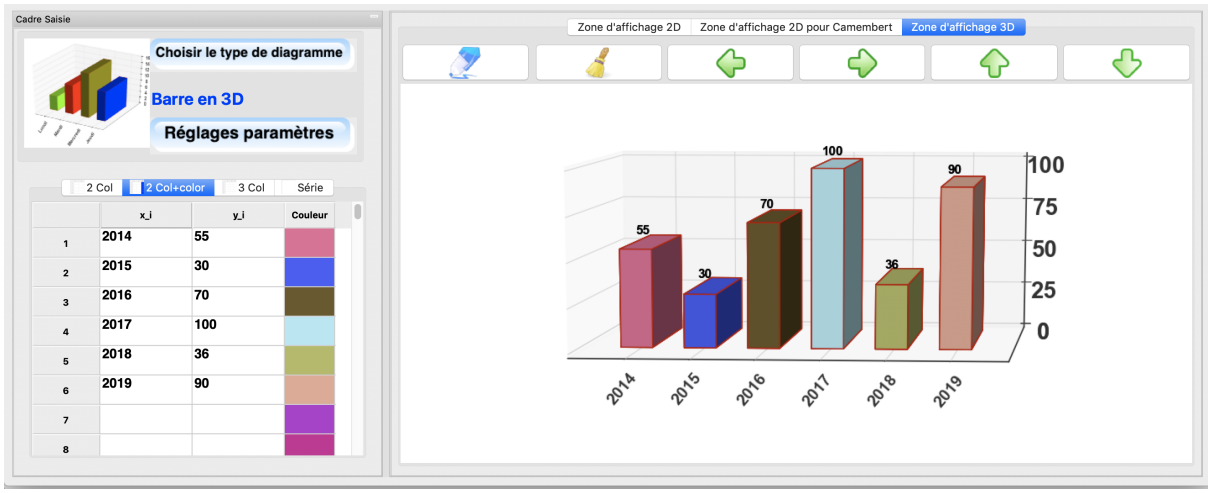




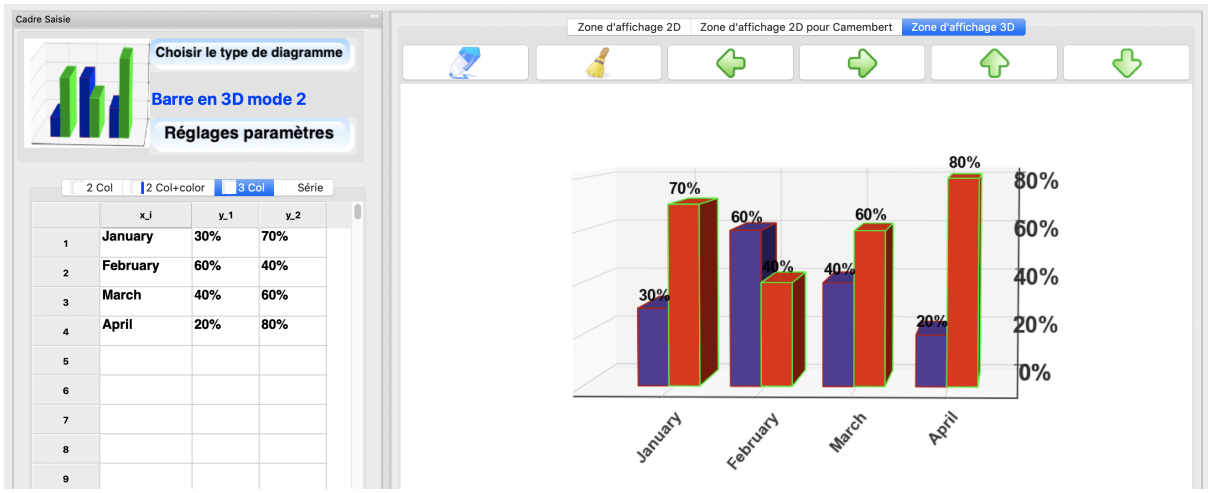


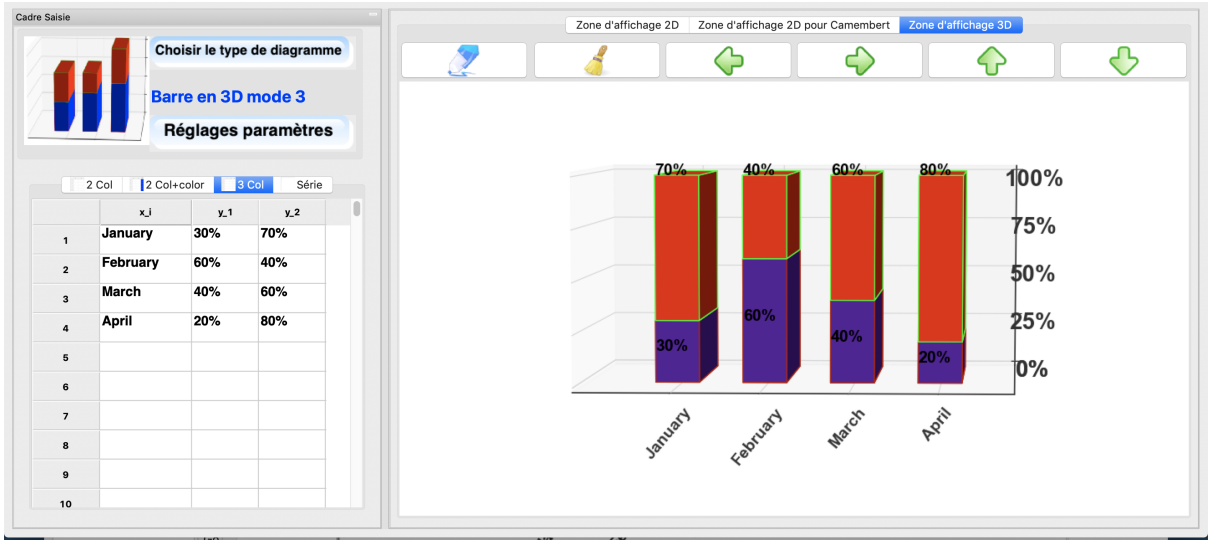
Diagrammes en barres à trois dimensions





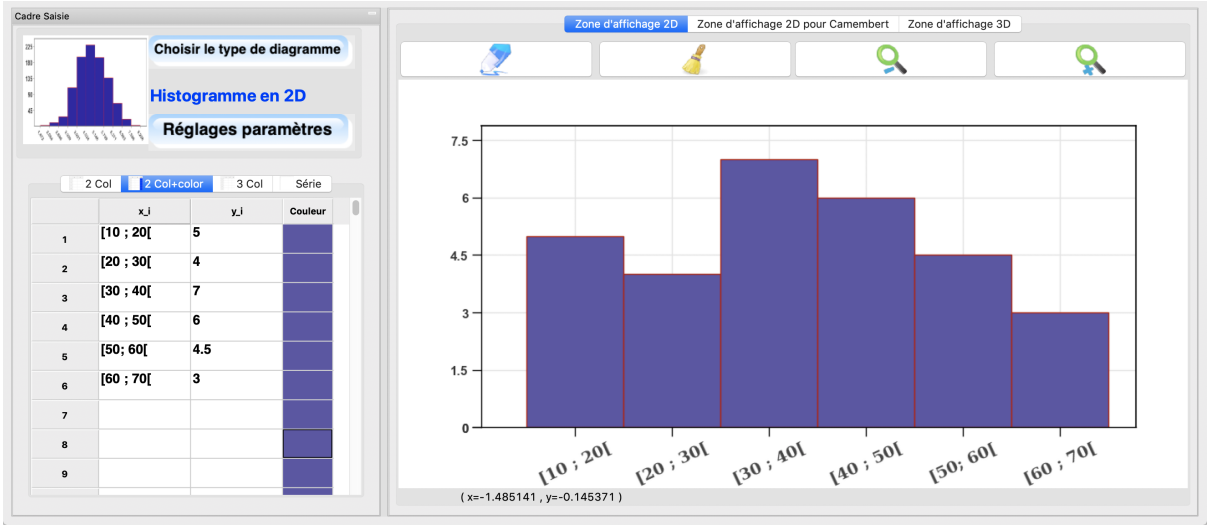

Cadre Saisie
Choisir le type de diagramme
Barre en 3D
Réglages paramètres
2 Col
2 Col+color
3 Col
Série
x_i
y_i
Couleur
A 4
B 2
C 3
D 1
E 5
Zone d'affichage 2D
Zone d'affichage 2D pour Camembert
Zone d'affichage 3D
E
A
C
B
D

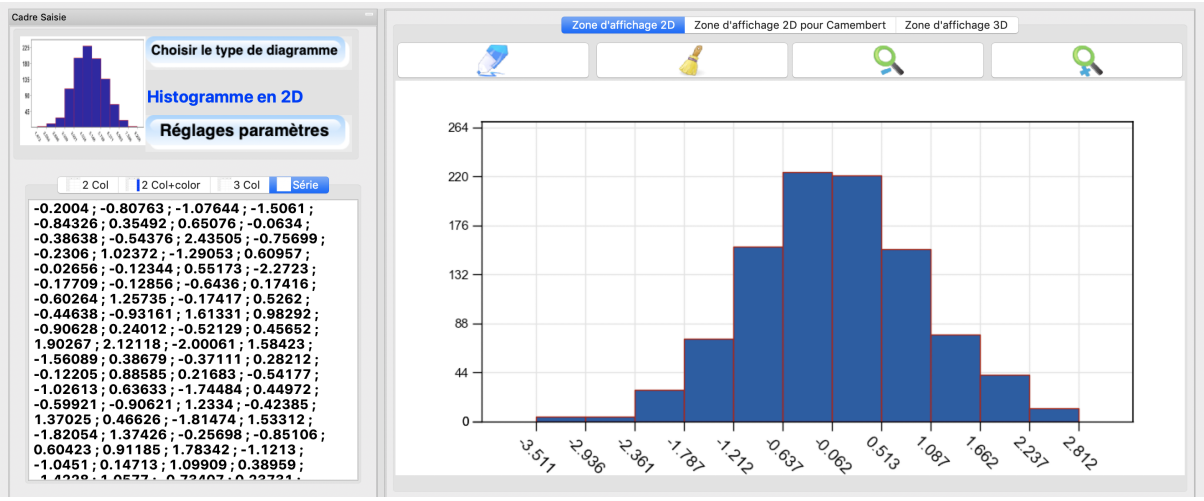

Cadre Saisie
Choisir le type de diagramme
Barre en 3D mode 2
Réglages paramètres
2 Col
2 Col+color
3 Col
Série
x_i
y_1
y_2
January 30% 70%
February 60% 40%
March 40% 60%
April 20% 80%
Zone d'affichage 2D
Zone d'affichage 2D pour Camembert
Zone d'affichage 3D
January
February
March
April


Cadre Saisie
Choisir le type de diagramme
Barre en 3D mode 3
Réglages paramètres
2 Col
2 Col+color
3 Col
Série
x_i
y_1
y_2
January 30% 70%
February 60% 40%
March 40% 60%
April 20% 80%
Zone d'affichage 2D
Zone d'affichage 2D pour Camembert
Zone d'affichage 3D
January
February
March
April

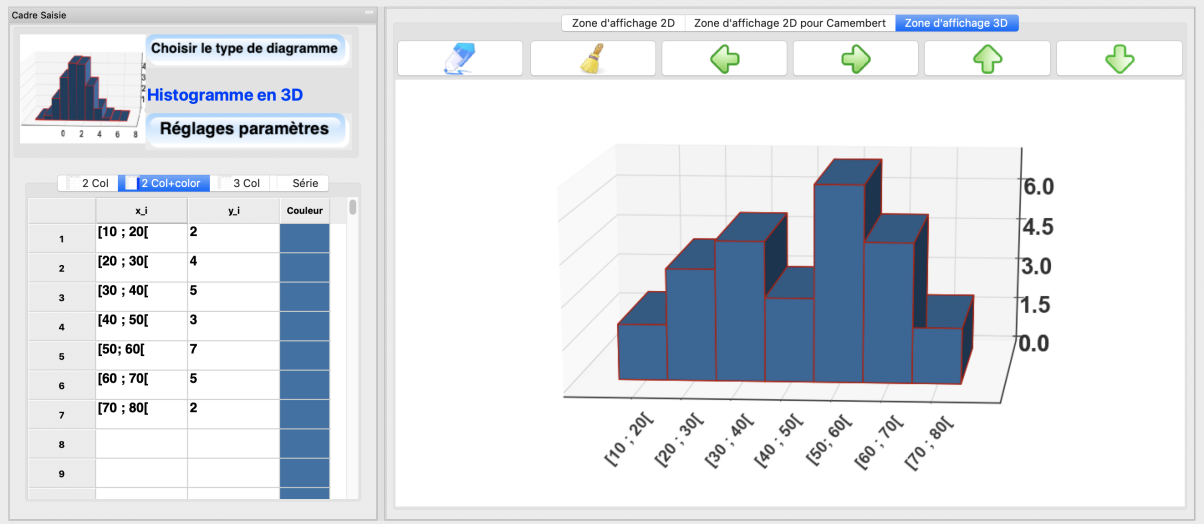
4.7.2 Histogrammes

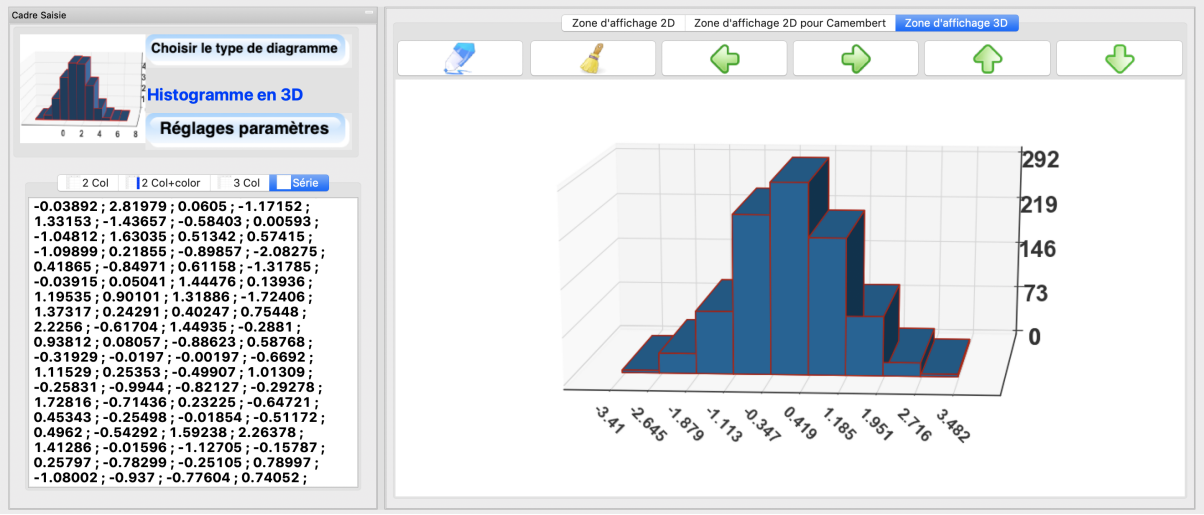
Histogramme en deux dimensions



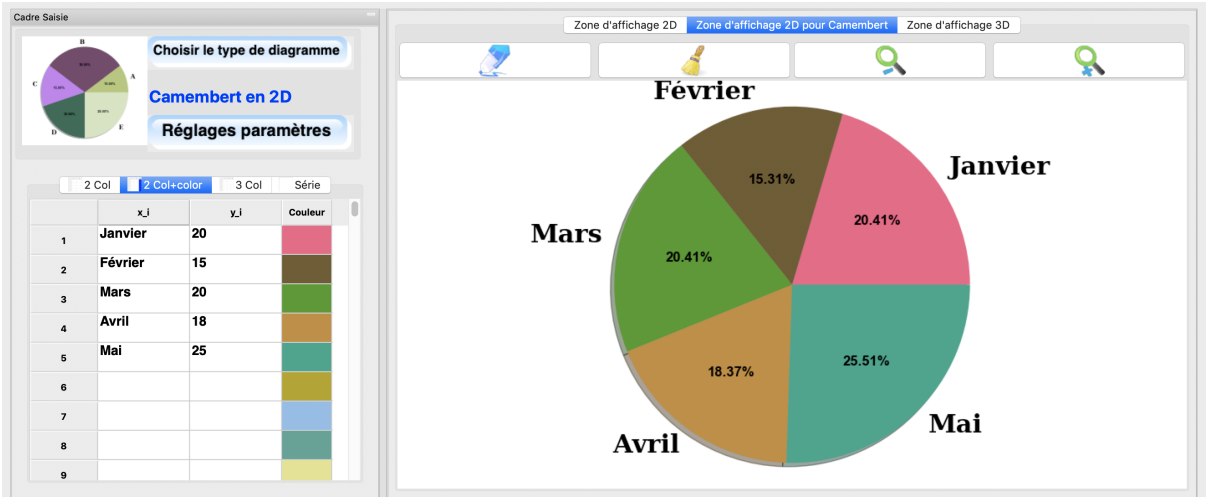


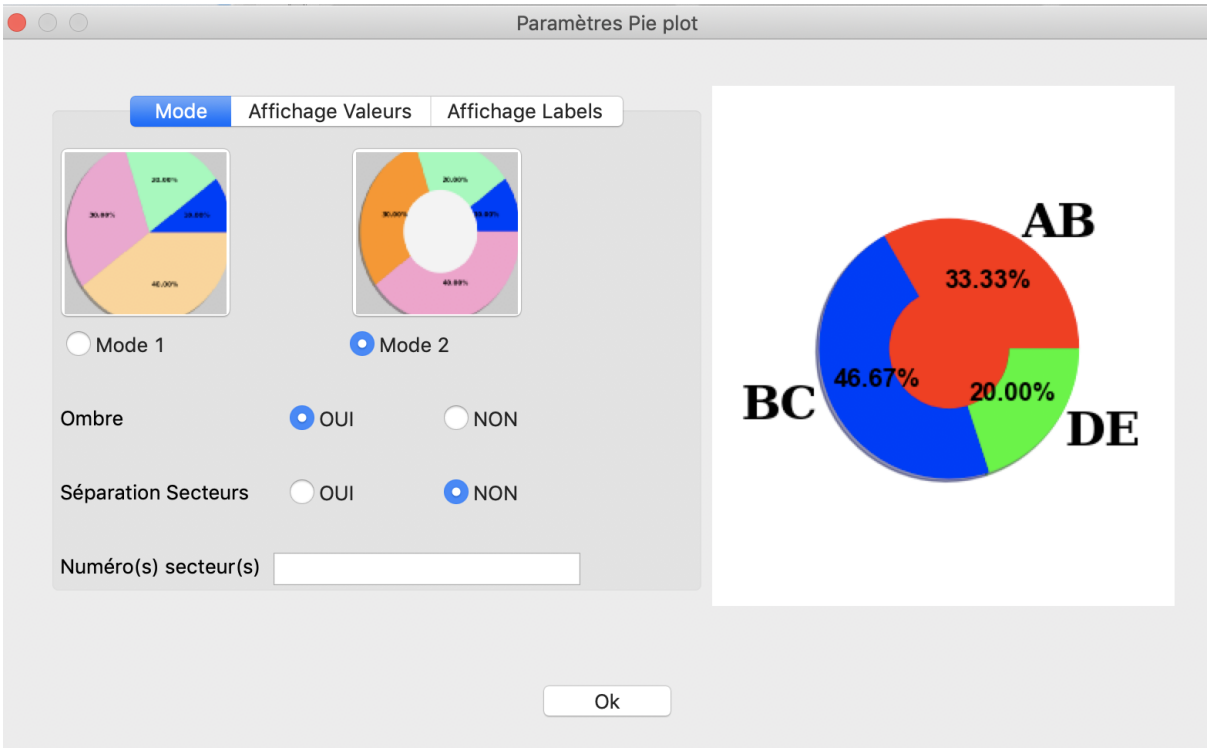
Histogramme en trois dimensions

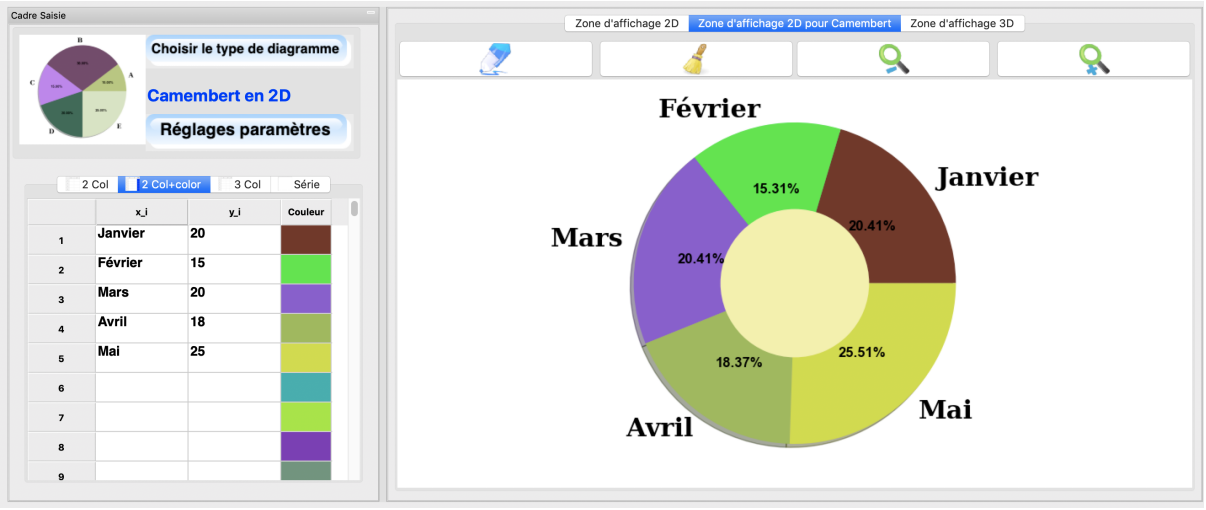


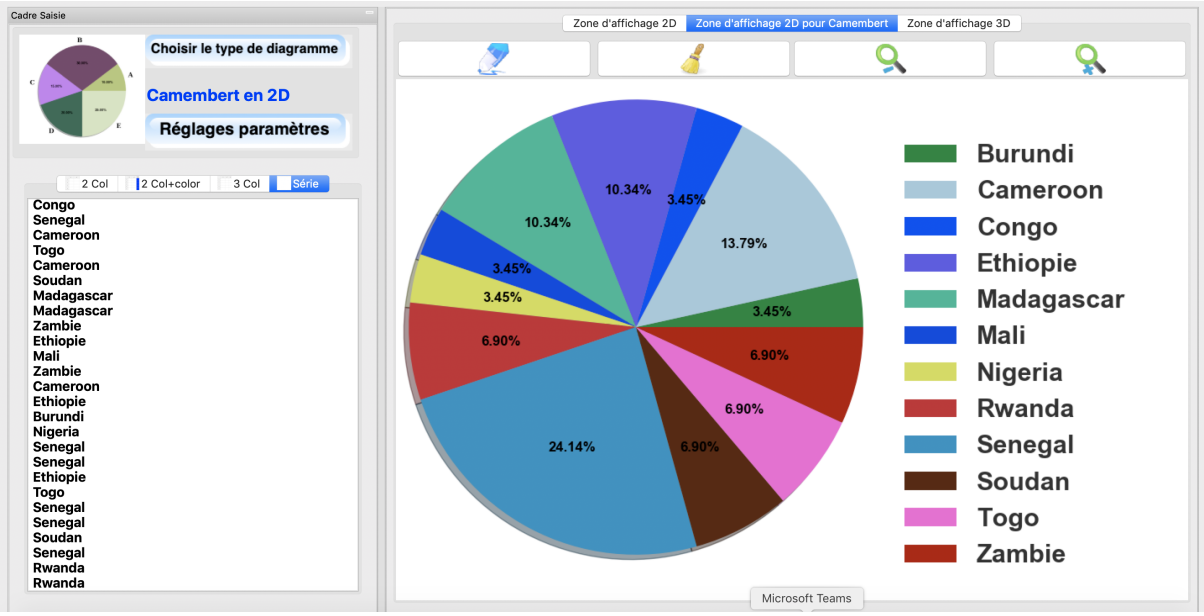


4.7.3 Diagrammes circulaires







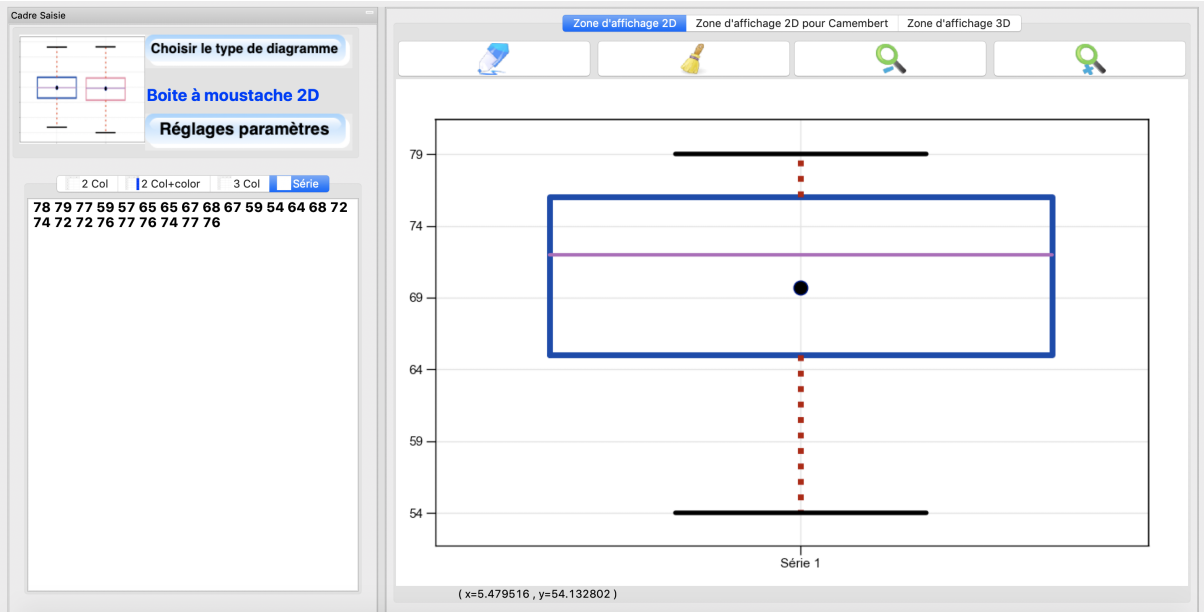


4.7.4 Diagramme en boîte

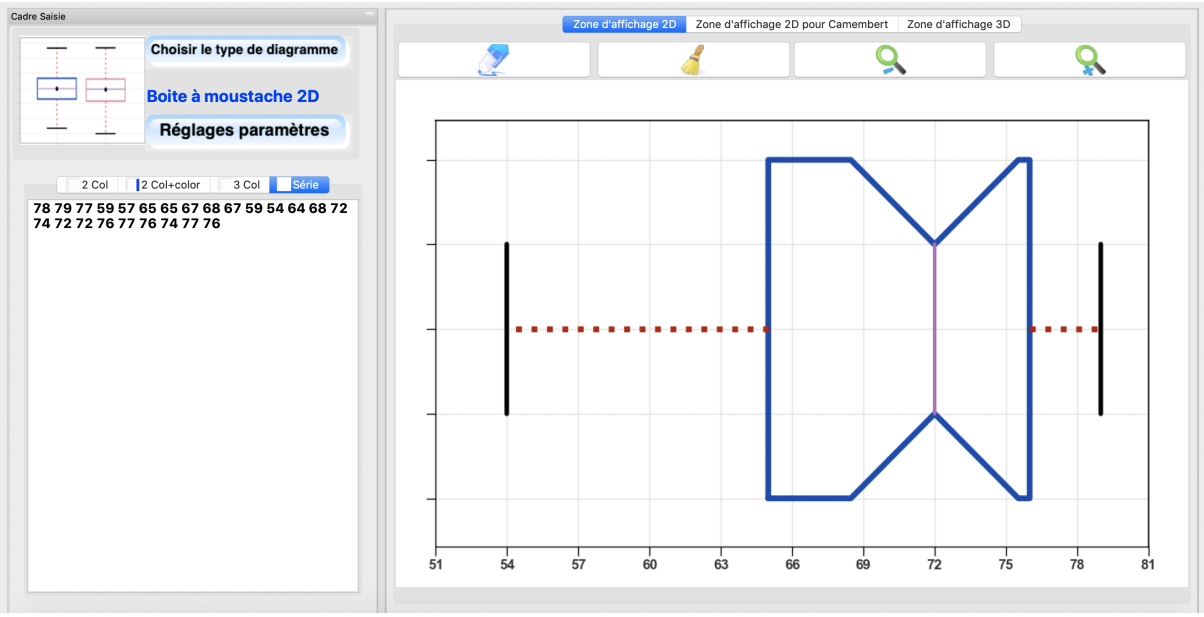
Les notes de 24 étudiants d'une classe ont été enregistrées sur un examen de 100 points.

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

Traçons la boîte à moustaches pour cette série statistique.



On peut également modifier l'alignement et les propriétés de la boîte.



4.7.5 Nuage de points

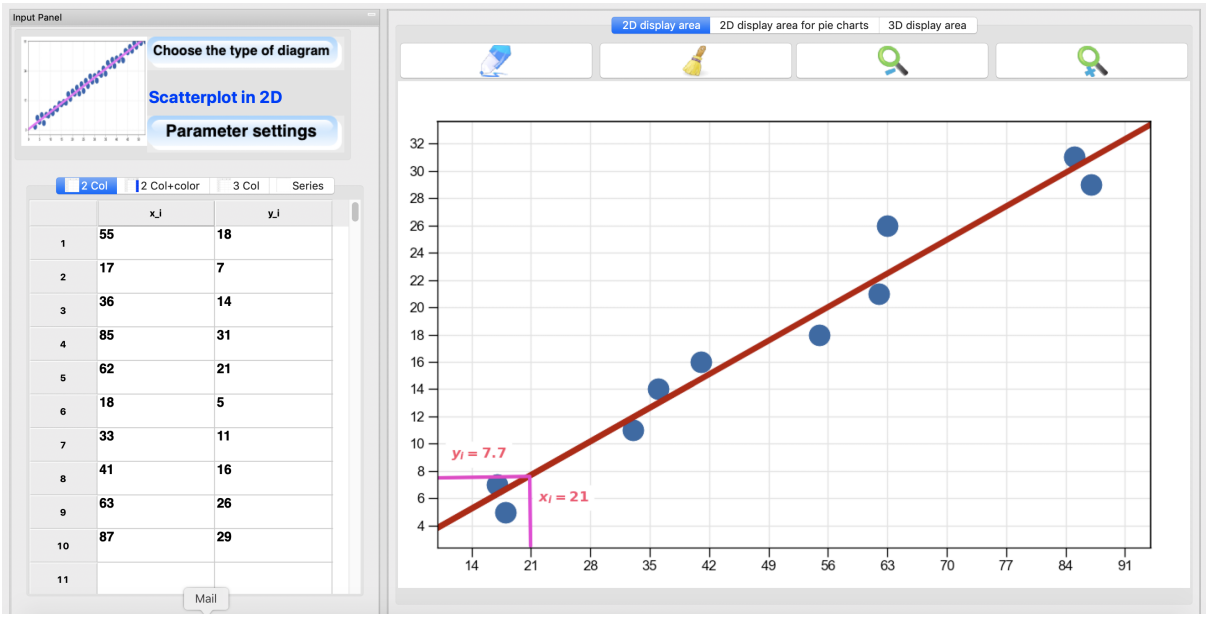
Nous donnons les paires d'observations suivantes

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

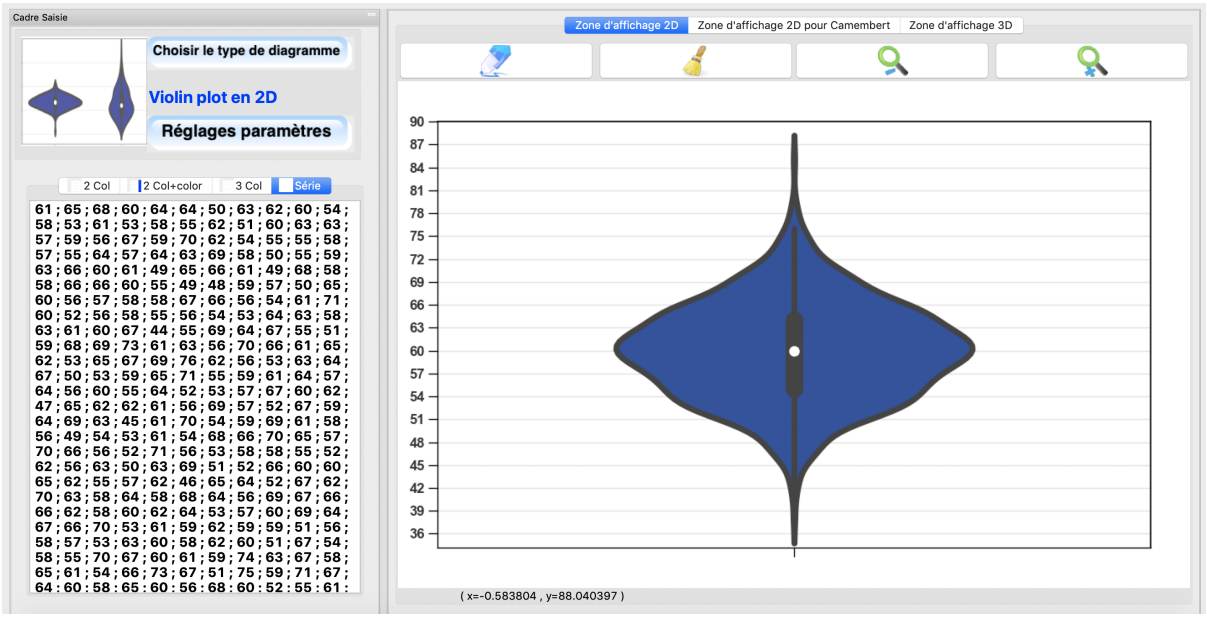
a. Tracer le nuage de points des paires (x_i, y_i).

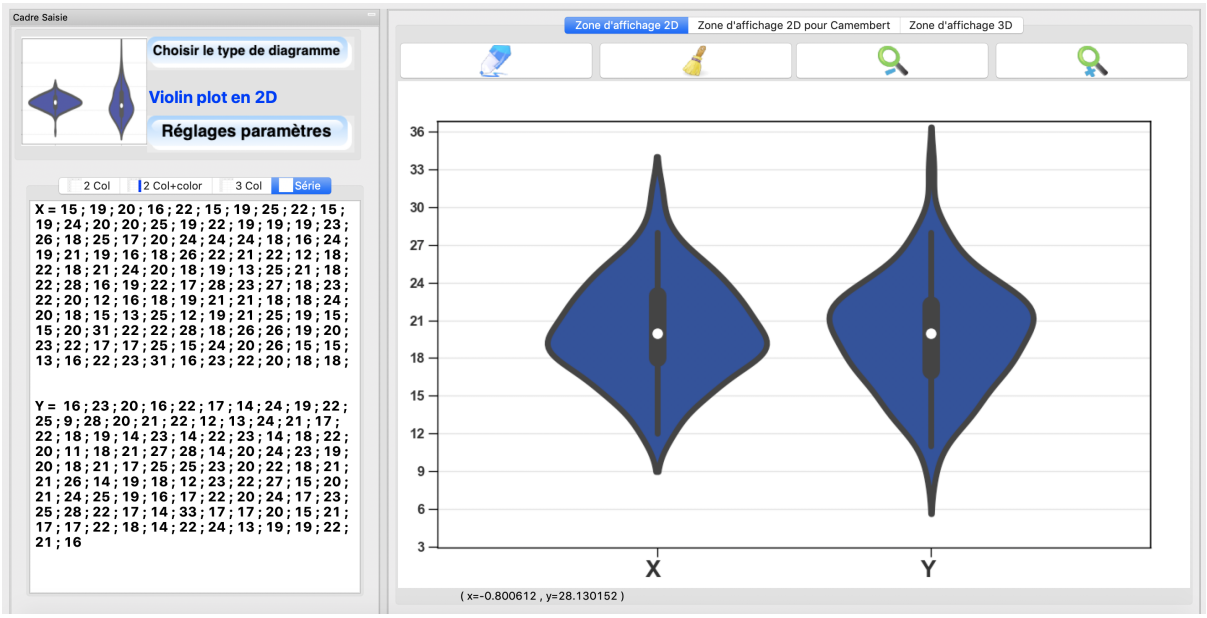
b. Déterminer la droite de régression pour ces observations.

c. Donner une estimation plausible de y pour $x_i = 21$.

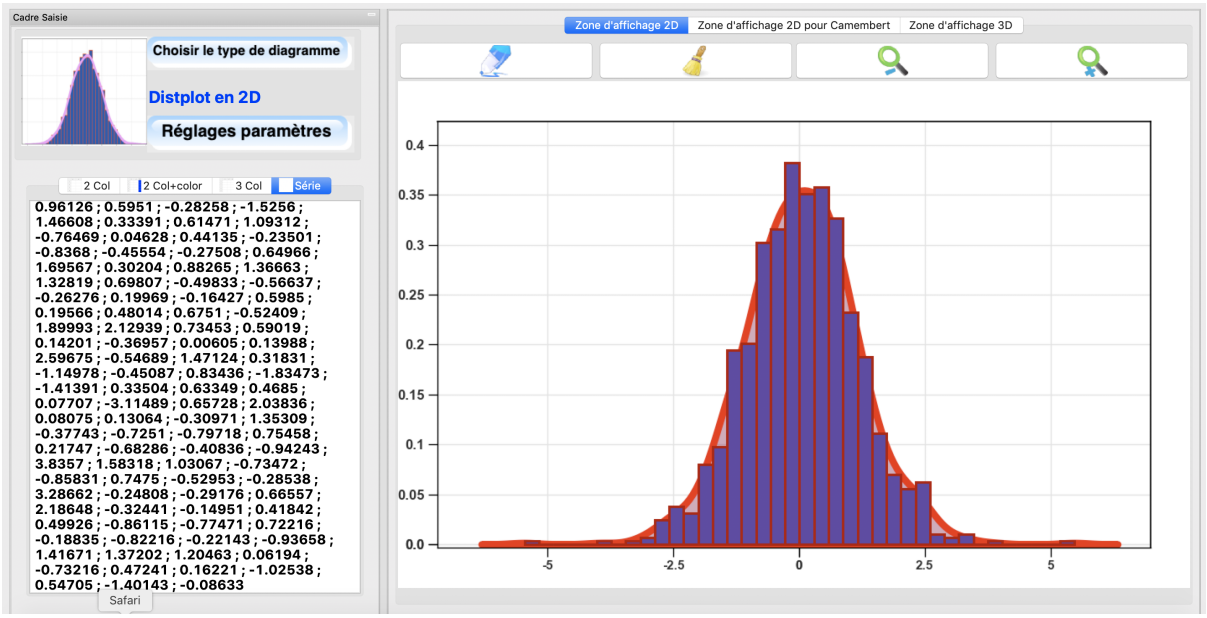


4.7.6 Violin plot



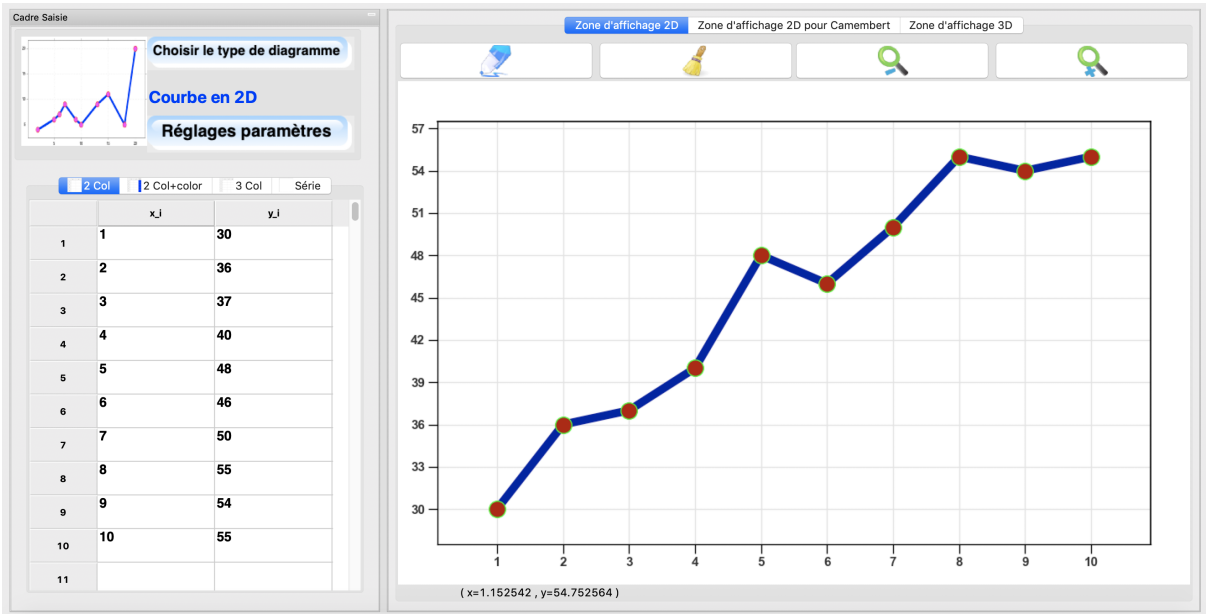


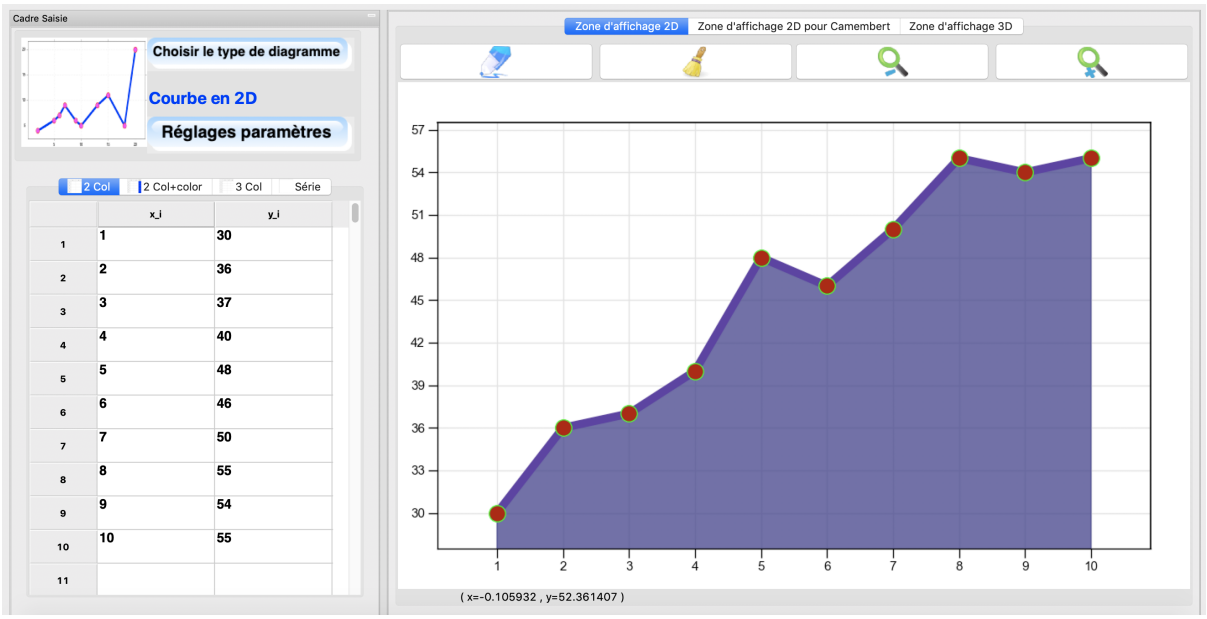
4.7.7 Distplot



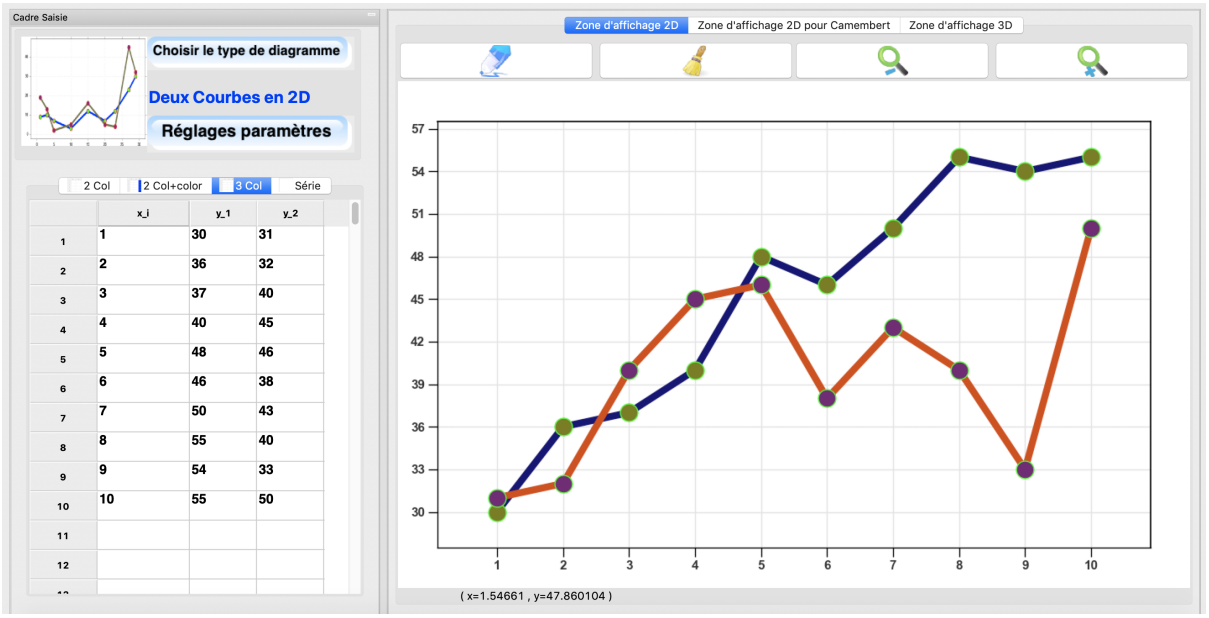
4.7.8 Courbes

Une courbe



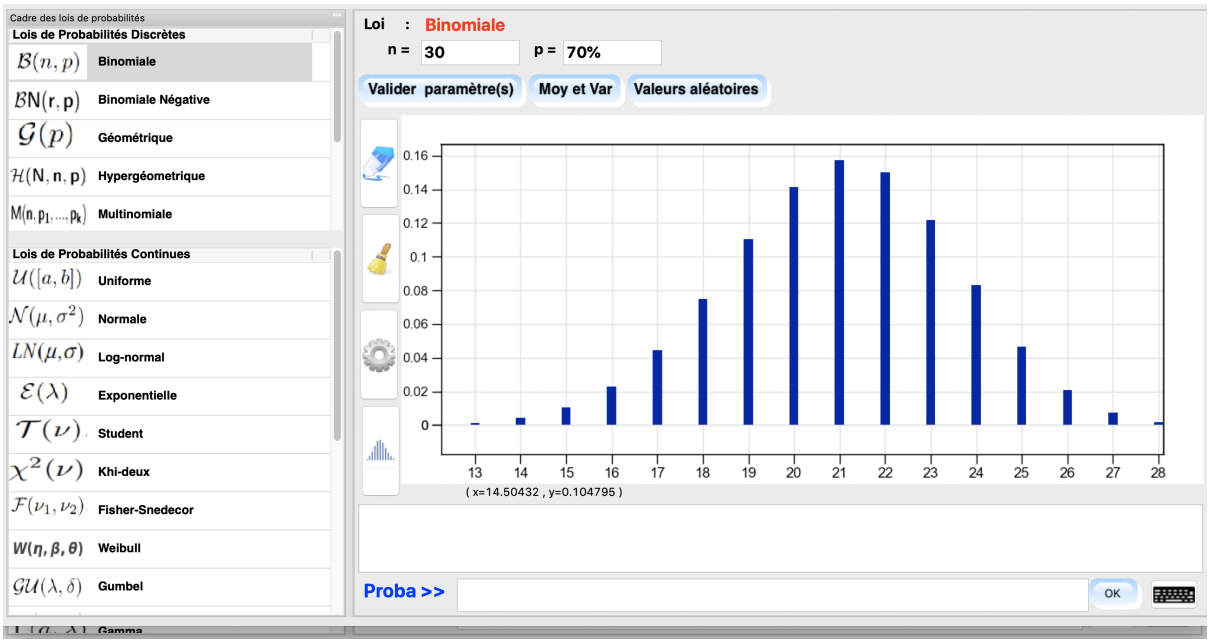


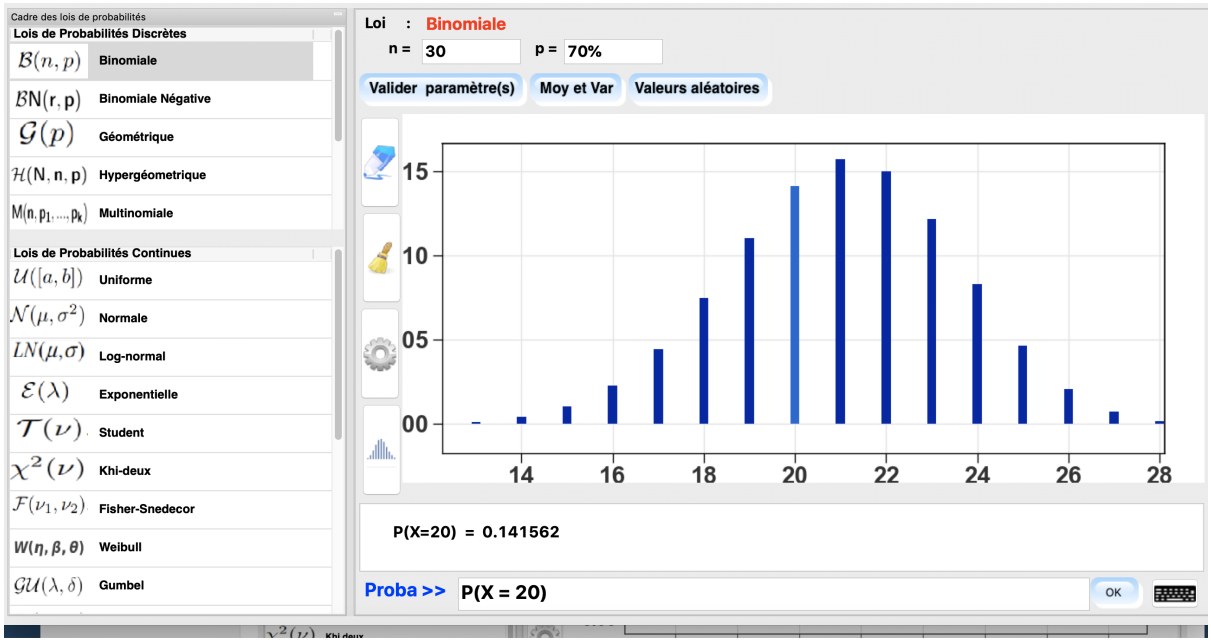
Deux courbes simultanément

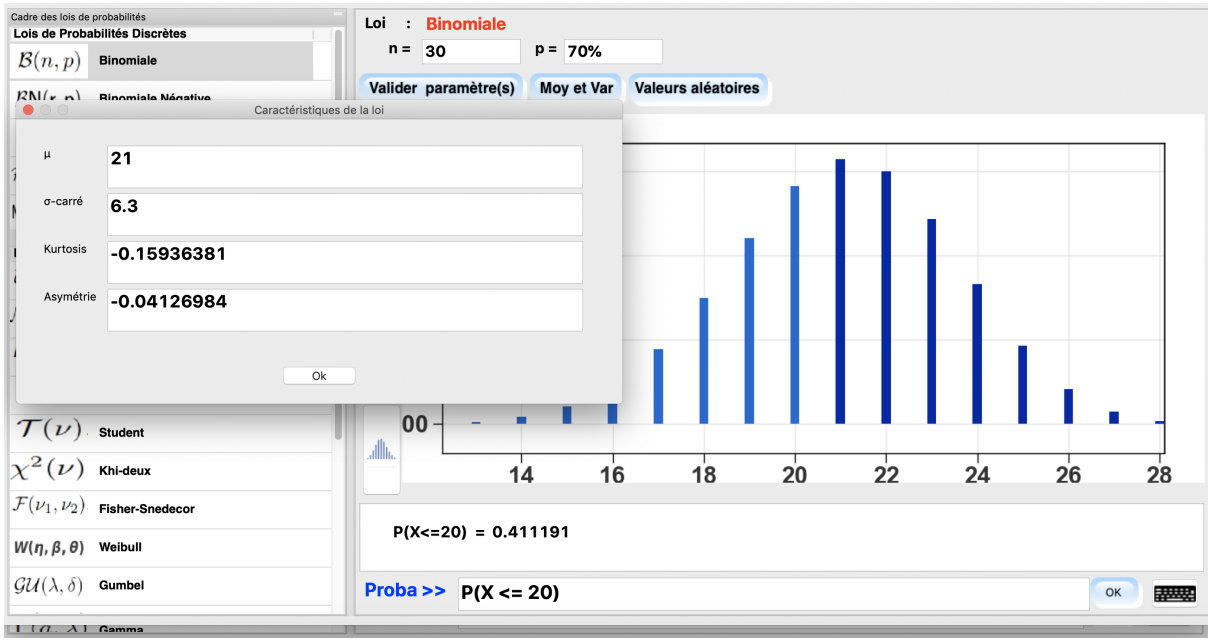


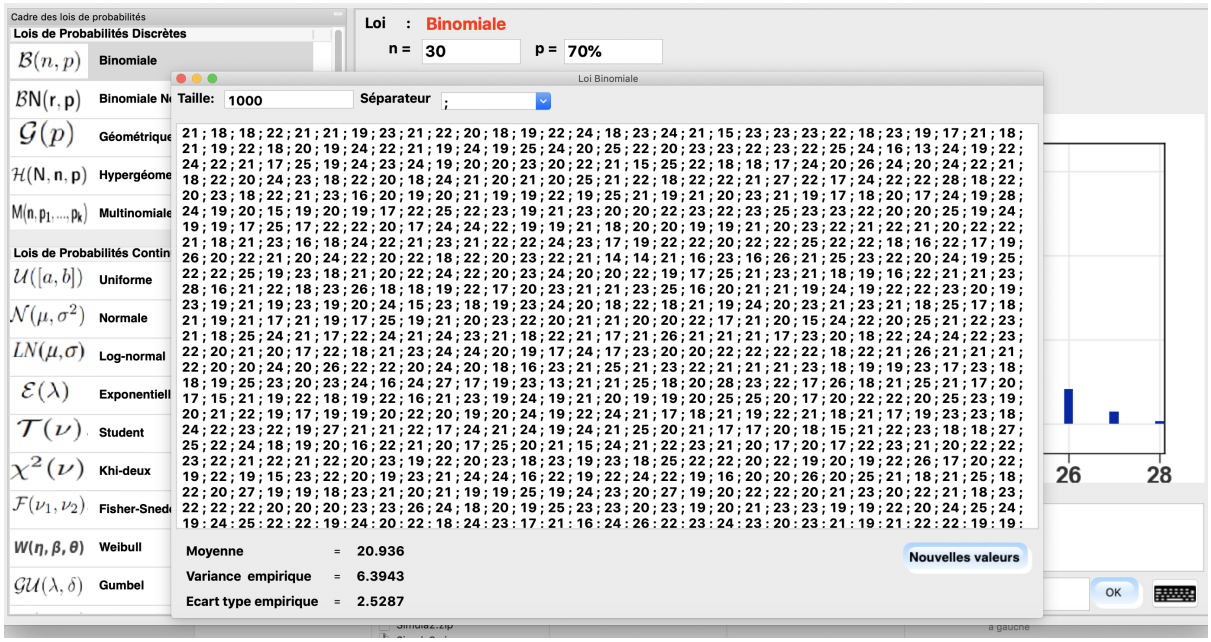
4.8 Probabilités

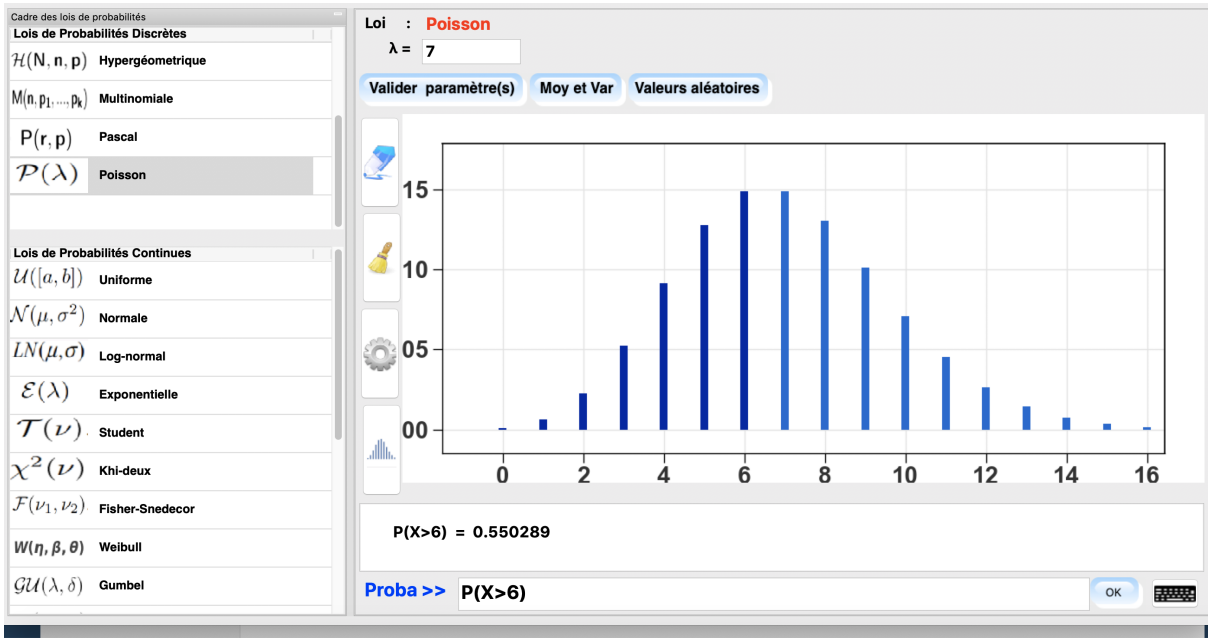
4.8.1 Lois discrètes




Cadre des lois de probabilités
Lois de Probabilités Discrètes
Binomiale
Binomiale Négative
Géométrique
Hypergéométrique
Multinomiale
Lois de Probabilités Continues
Uniforme
Normale
Log-normal
Exponentielle
Student
Khi-deux
Fisher-Snedecor
Weibull
Gumbel
Loi : Binomiale
n = 30
p = 70%
Valider paramètre(s)
Moy et Var
Valeurs aléatoires
15
10
05
00
14
16
18
20
22
24
26
28
P(X=20) = 0.141562
Proba >>
P(X = 20)
OK

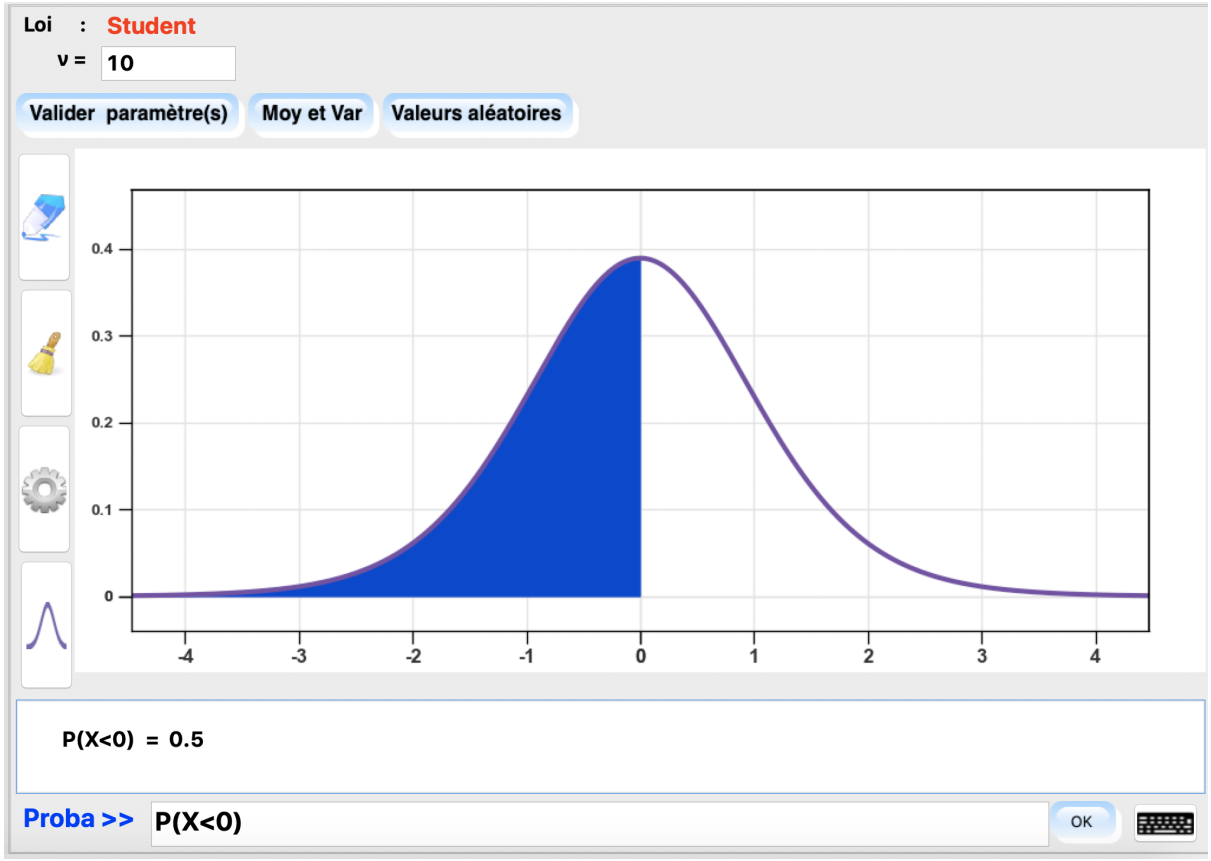

Caractéristiques de la loi
μ
21
σ-carré
6.3
Kurtosis
-0.15936381
Asymétrie
-0.04126984
Ok
P(X<=20) = 0.411191
Proba >>
P(X <= 20)
OK

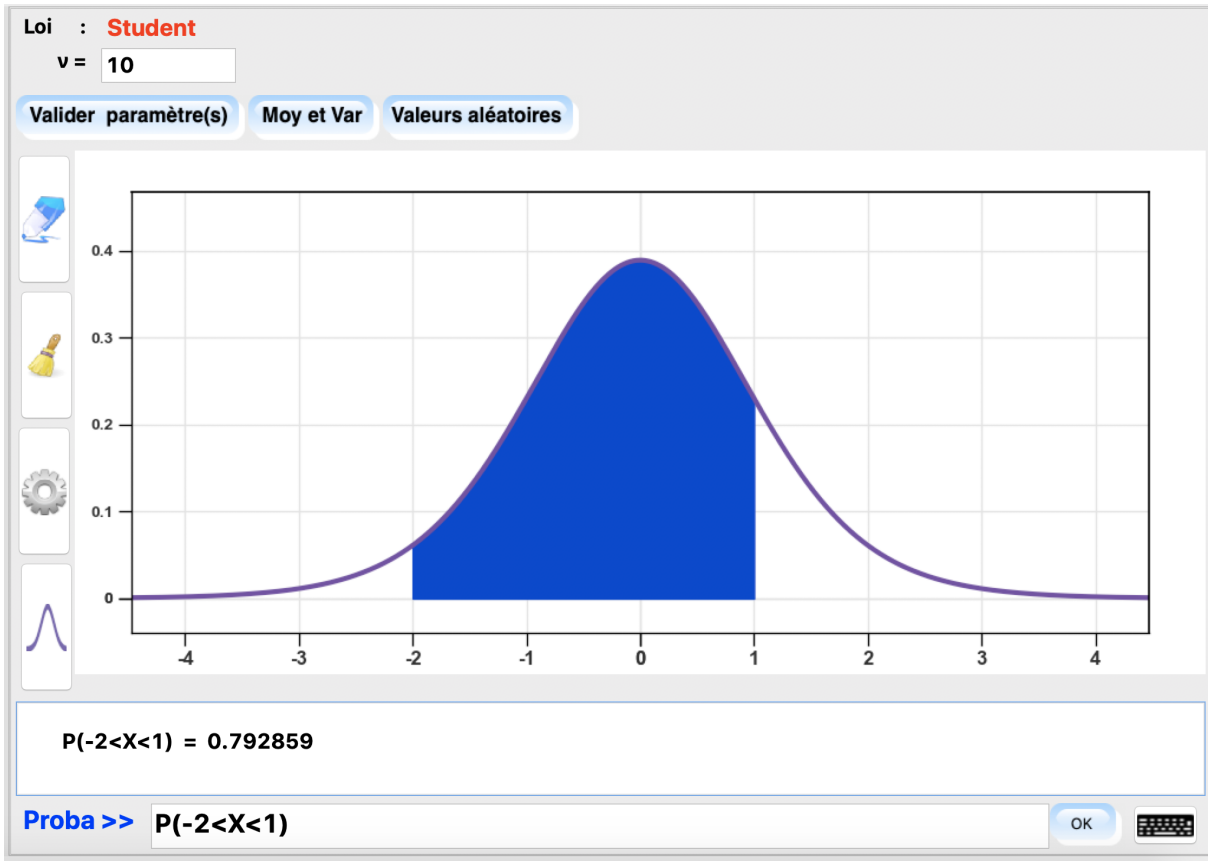


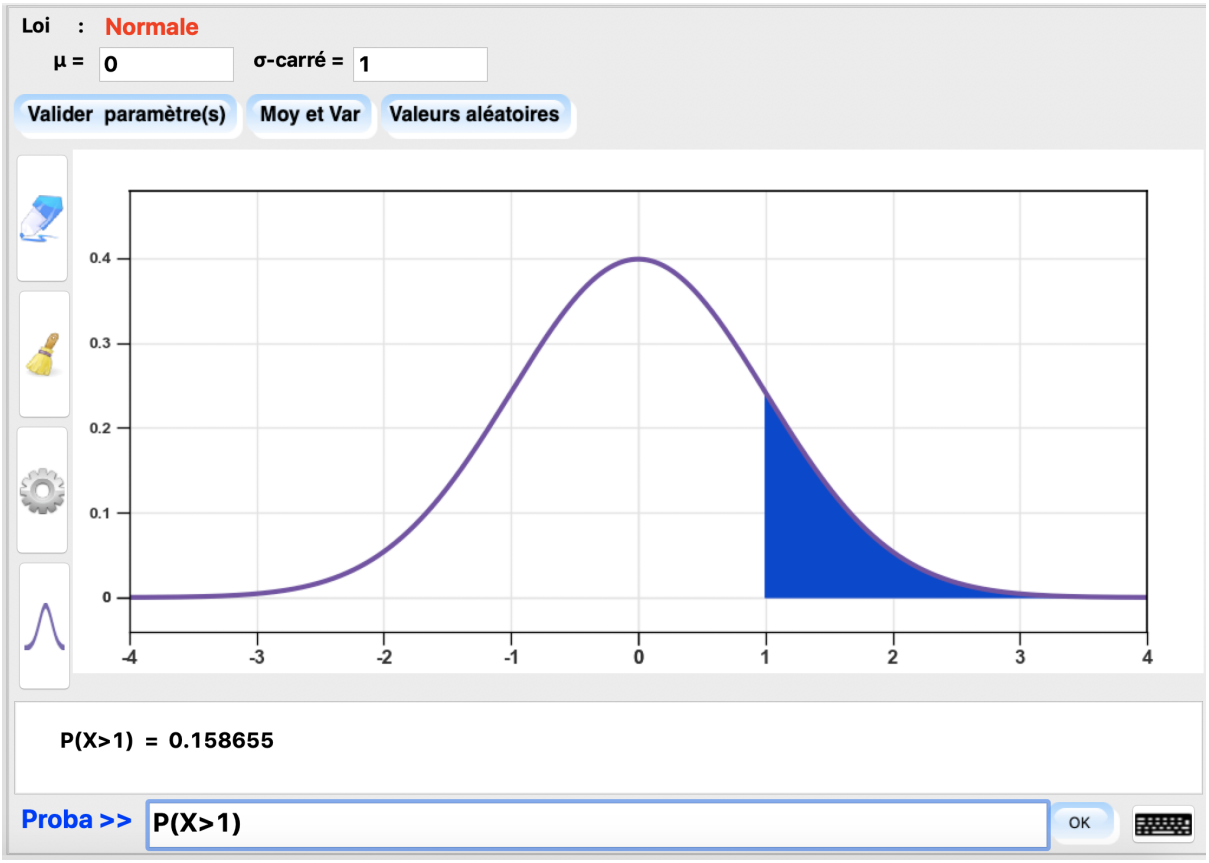


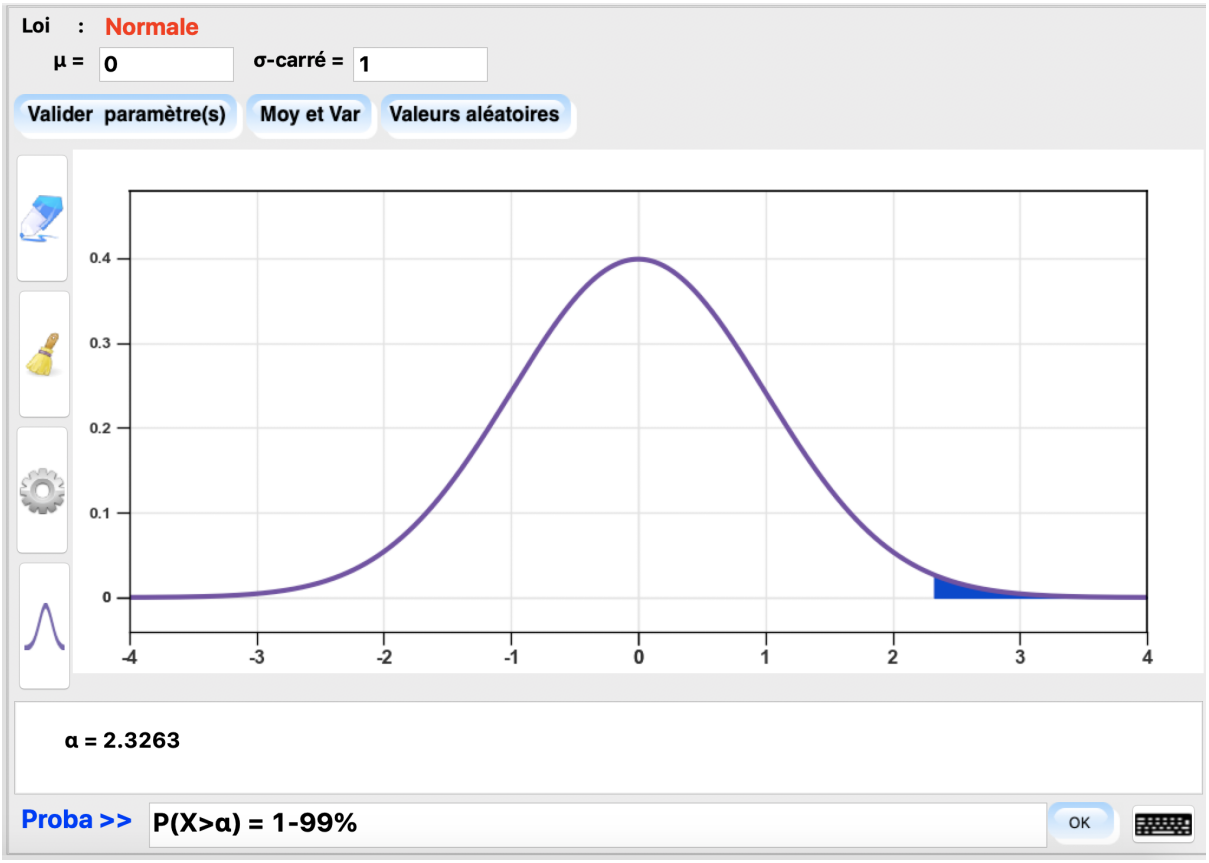
4.8.2 Lois continues

Calcul de Probabilités

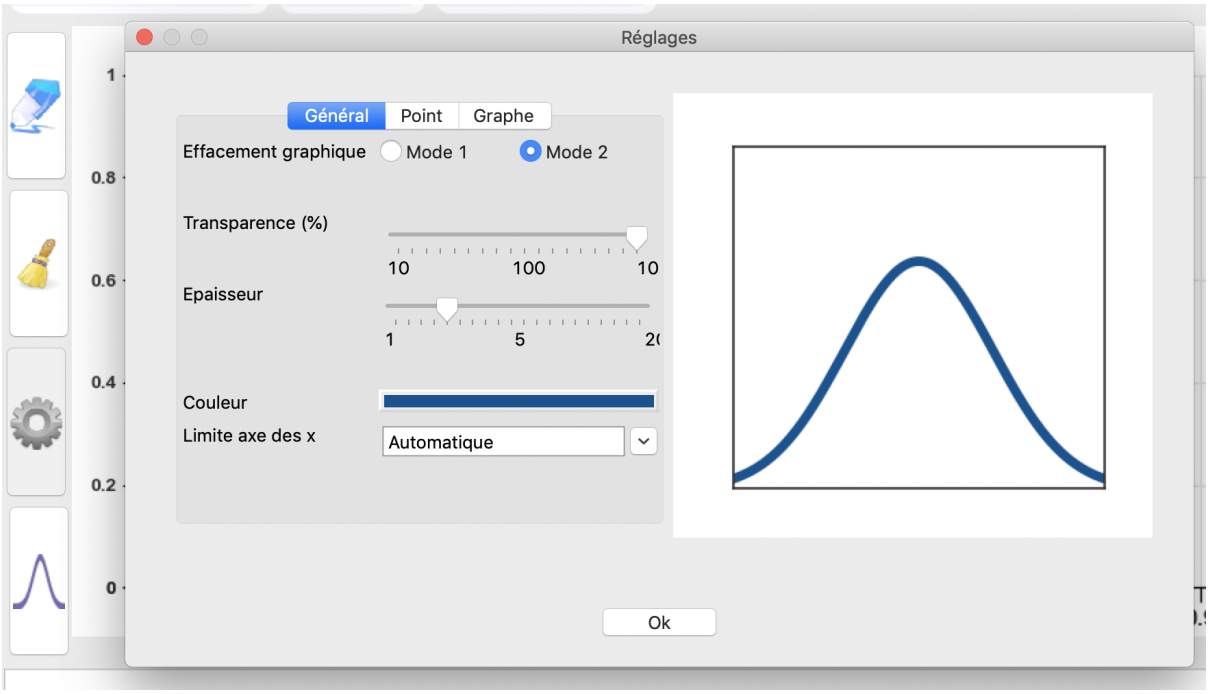


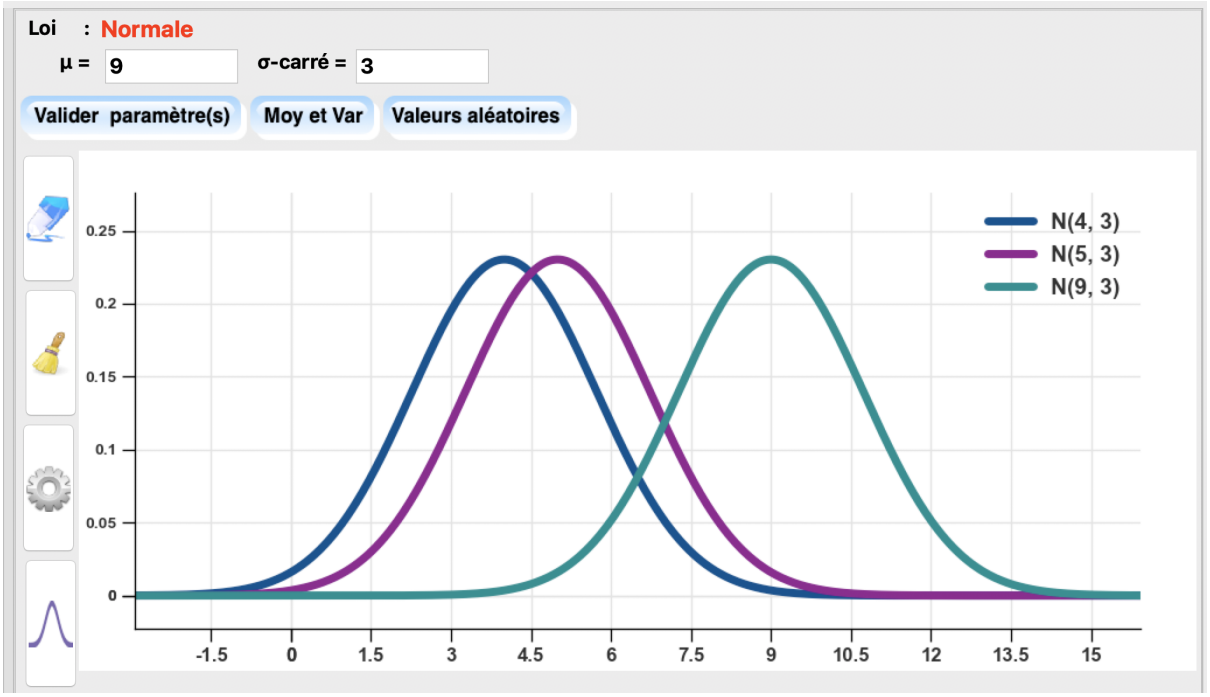


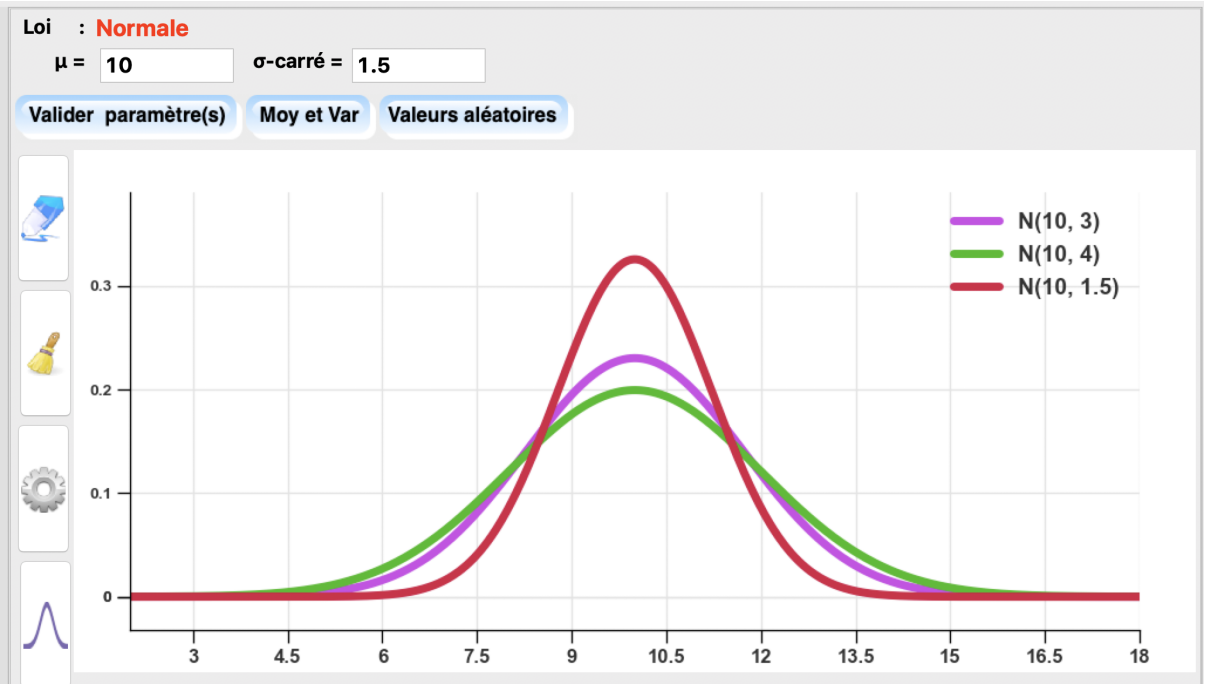

Loi : Normale
μ = 0
σ-carré = 1
Valider paramètre(s)
Moy et Var
Valeurs aléatoires
0.4
0.3
0.2
0.1
0
-4
-3
-2
-1
0
1
2
3
4
P(X>1) = 0.158655
Proba >>
P(X>1)
OK

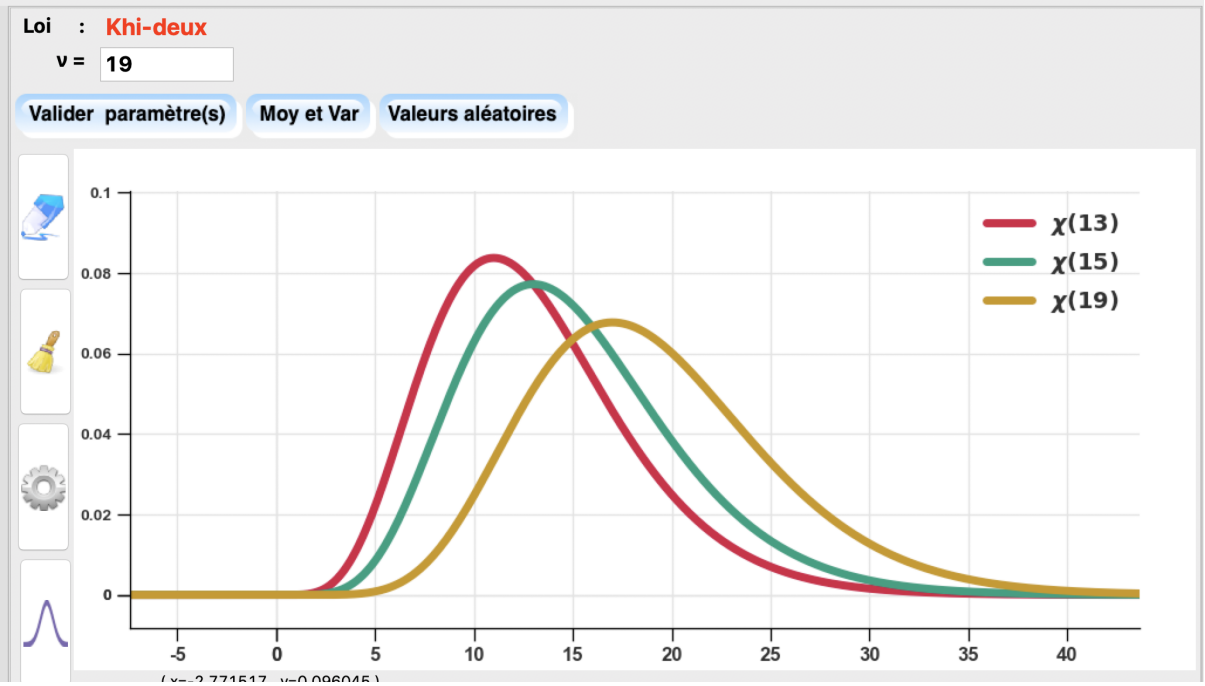

Loi : Normale
μ = 0
σ-carré = 1
Valider paramètre(s)
Moy et Var
Valeurs aléatoires
0.4
0.3
0.2
0.1
0
-4
-3
-2
-1
0
1
2
3
4
α = 2.3263
Proba >>
P(X>α) = 1-99%
OK

Simulation de distributions de probabilité avec différents paramètres



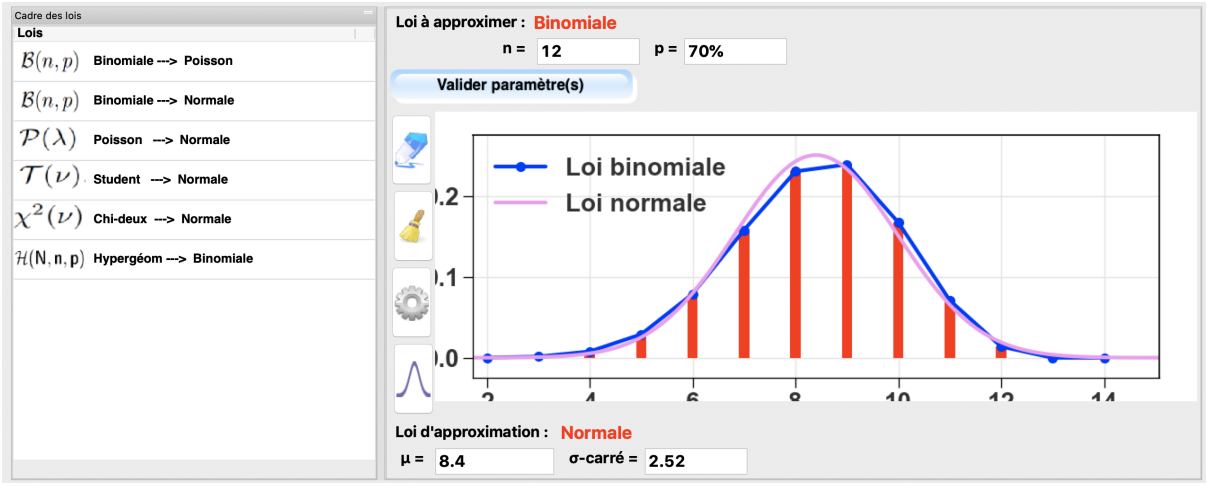


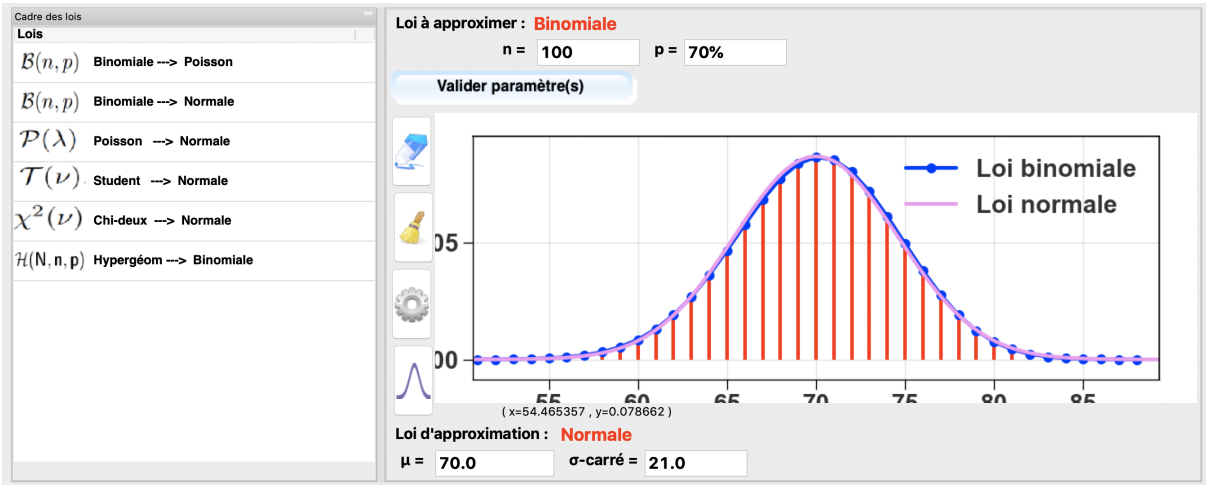

Loi : Normale
μ = 10
σ-carré = 1.5
Valider paramètre(s)
Moy et Var
Valeurs aléatoires
N(10, 3)
N(10, 4)
N(10, 1.5)
0.3
0.2
0.1
0
3
4.5
6
7.5
9
10.5
12
13.5
15
16.5
18


Loi : Khi-deux
ν = 19
Valider paramètre(s)
Moy et Var
Valeurs aléatoires
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

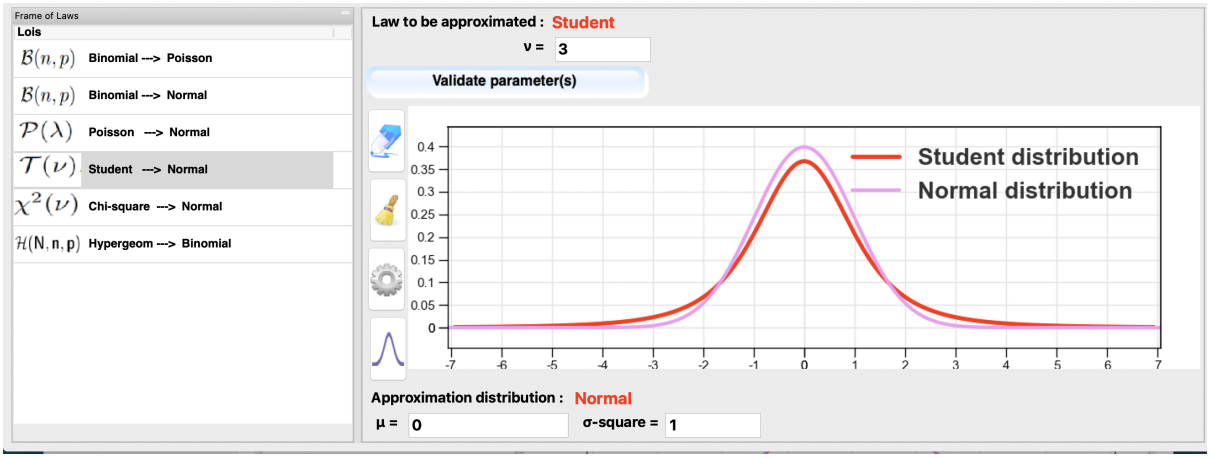
4.9 Approximation de lois de probabilité

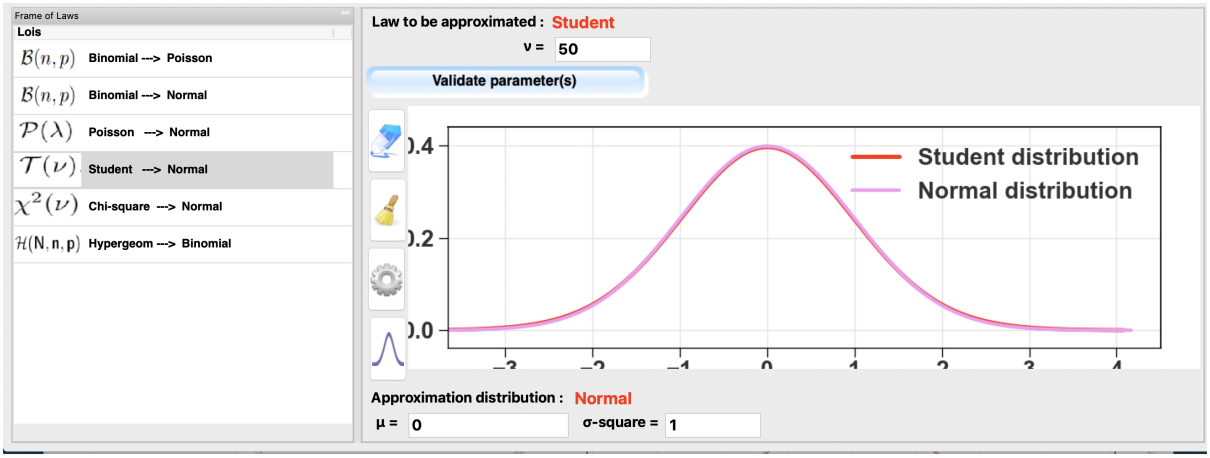
4.9.1 Approximation de la loi binomiale





4.9.2 Approximation de la loi de Student





4.9.3 Approximation d'autres distributions de probabilité

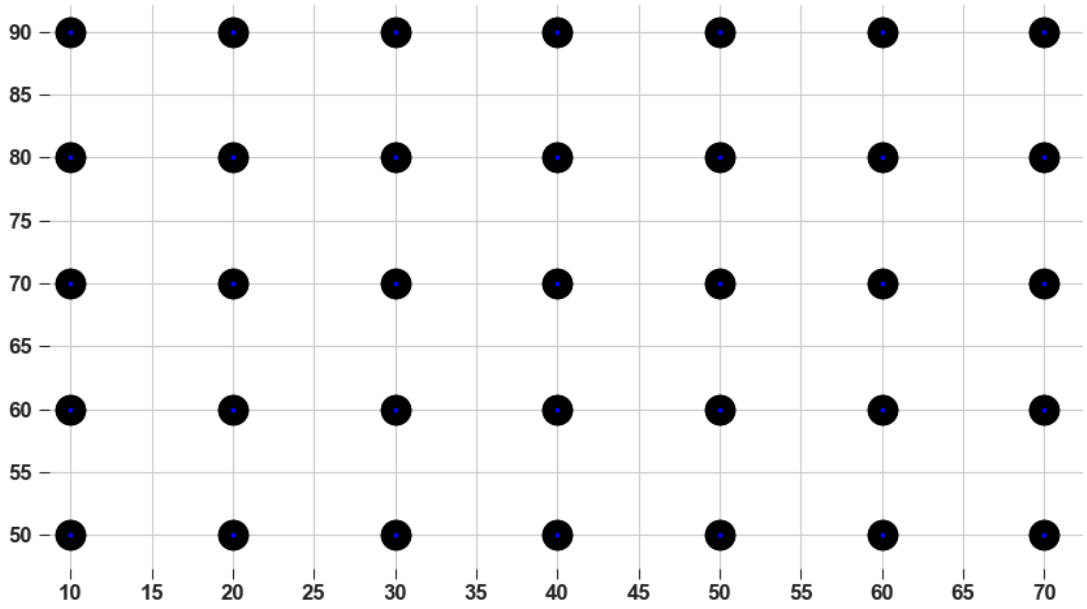
Avec **SimulaMath**, il est également possible de simuler sous quelle(s) condition(s) on peut approximer :

— la loi *binomiale* par une distribution *Poisson*,
— la loi du *Khi-deux* par une distribution *normale*,
— la loi de *Poisson* par une loi *normale*,
— la loi *hypergéométrique* par une loi *binomiale*.

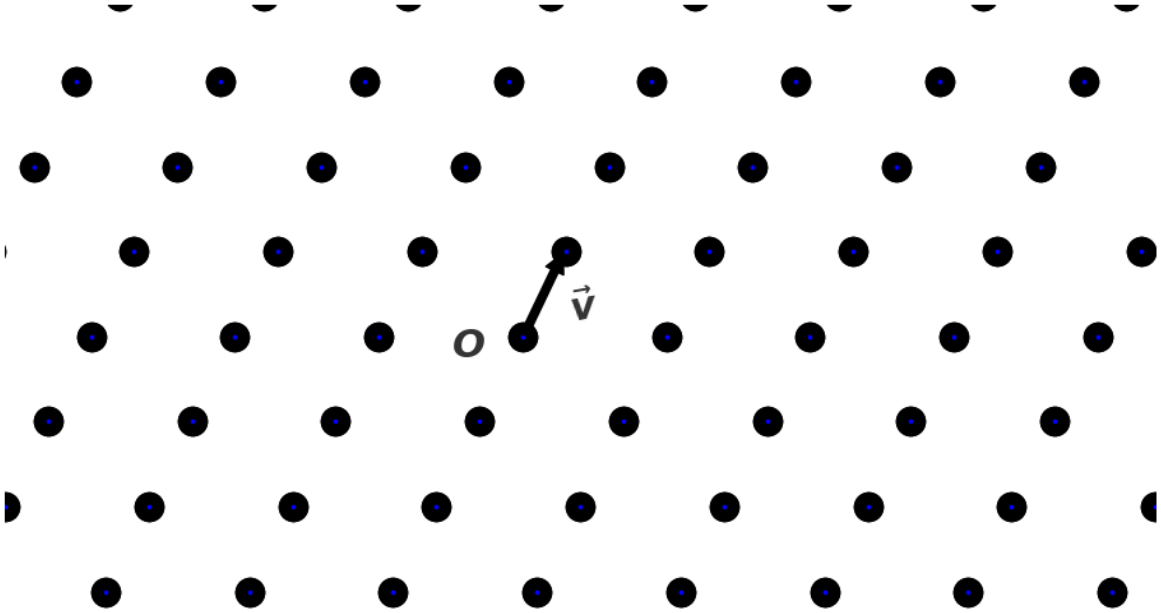
4.10 Réseaux euclidiens

La simulation de problèmes difficiles sur les réseaux

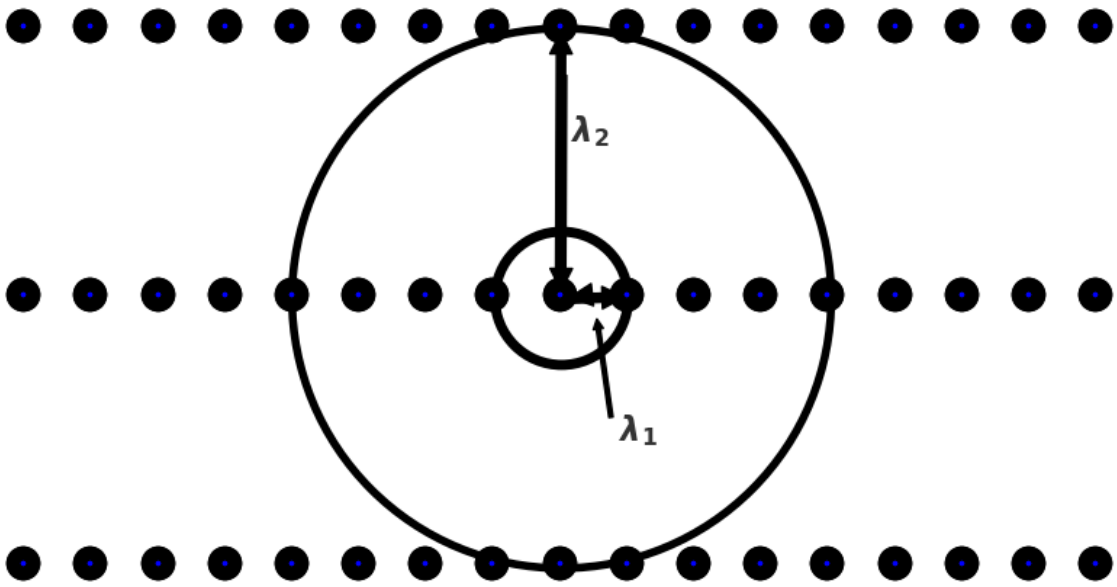
4.10.1 Réseau entier



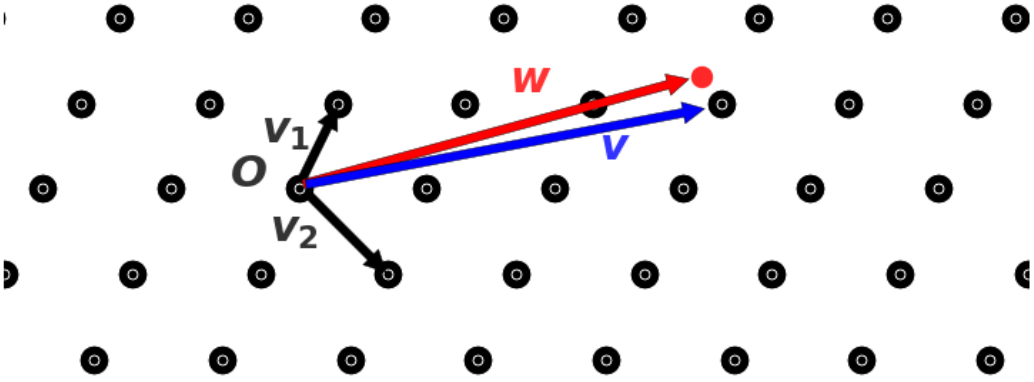
4.10.2 SVP (Shortest Vector Problem)



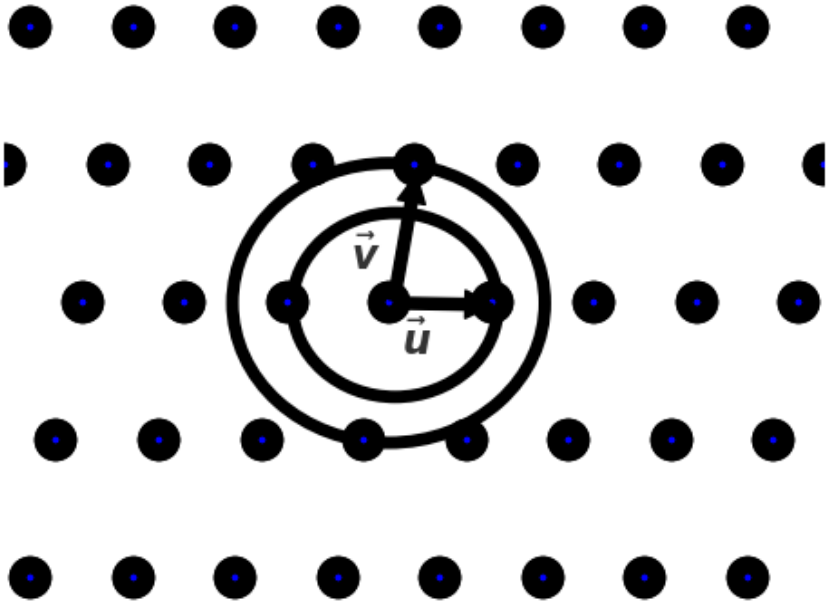
4.10.3 uSVP (Unique Shortest Vector Problem)



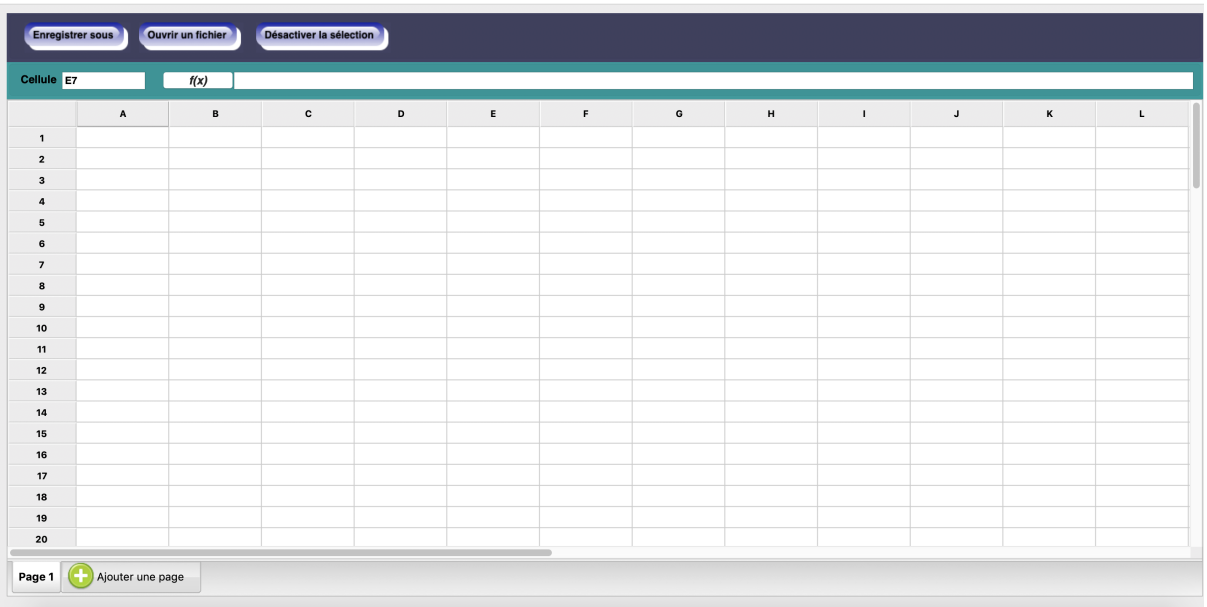
4.10.4 CVP (Closest Vector Problem)



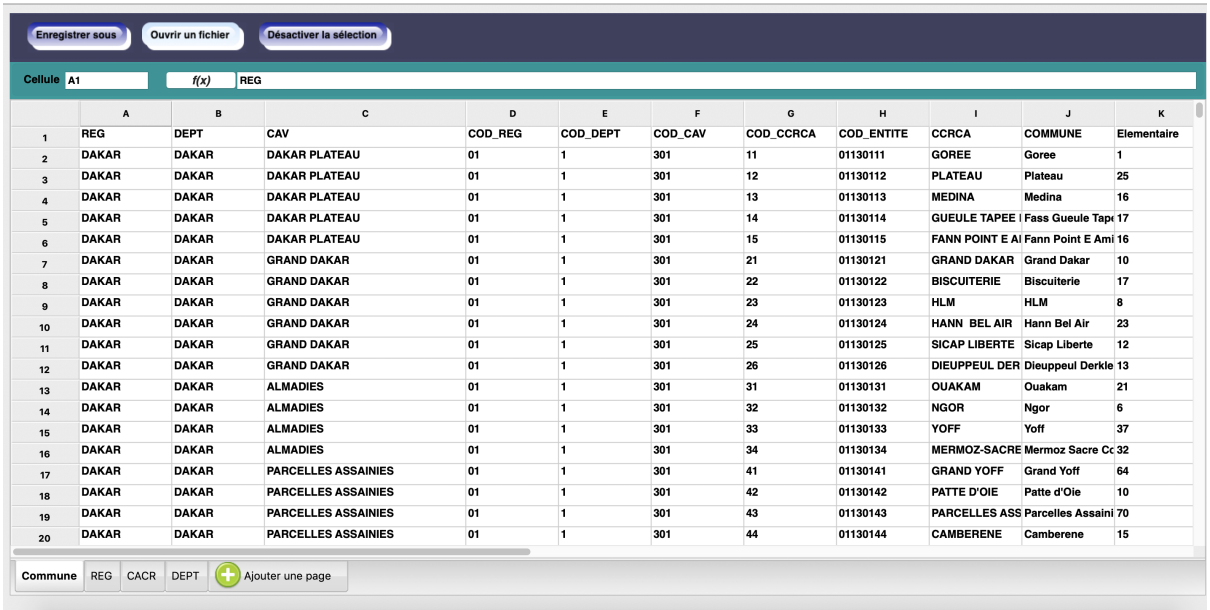
4.10.5 Minima successifs



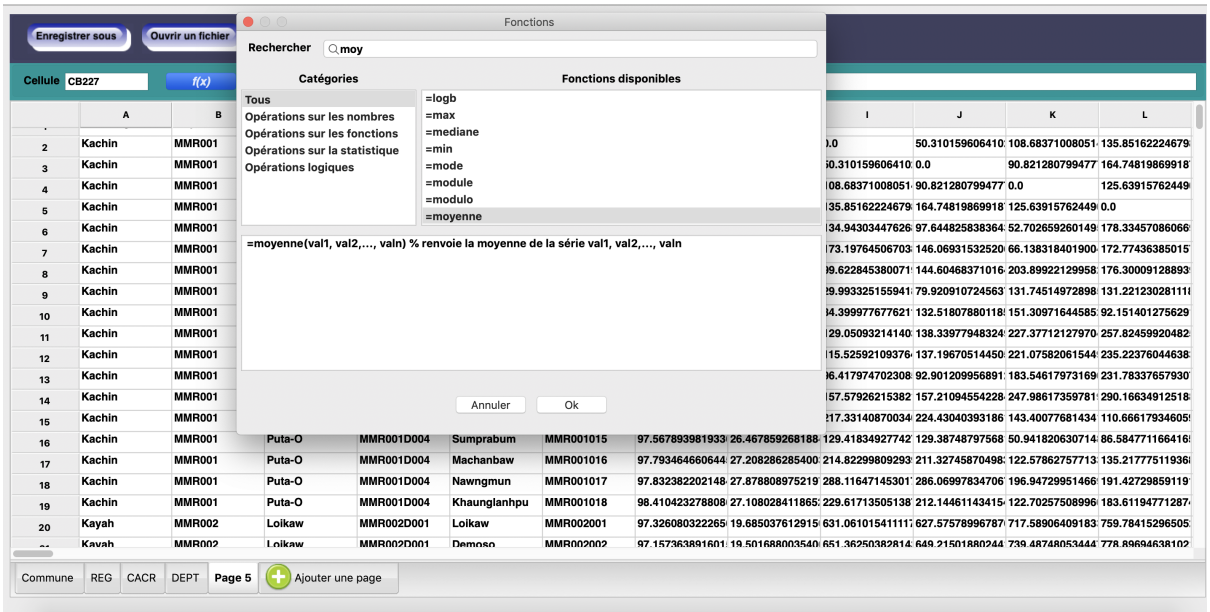
4.11 Tableur

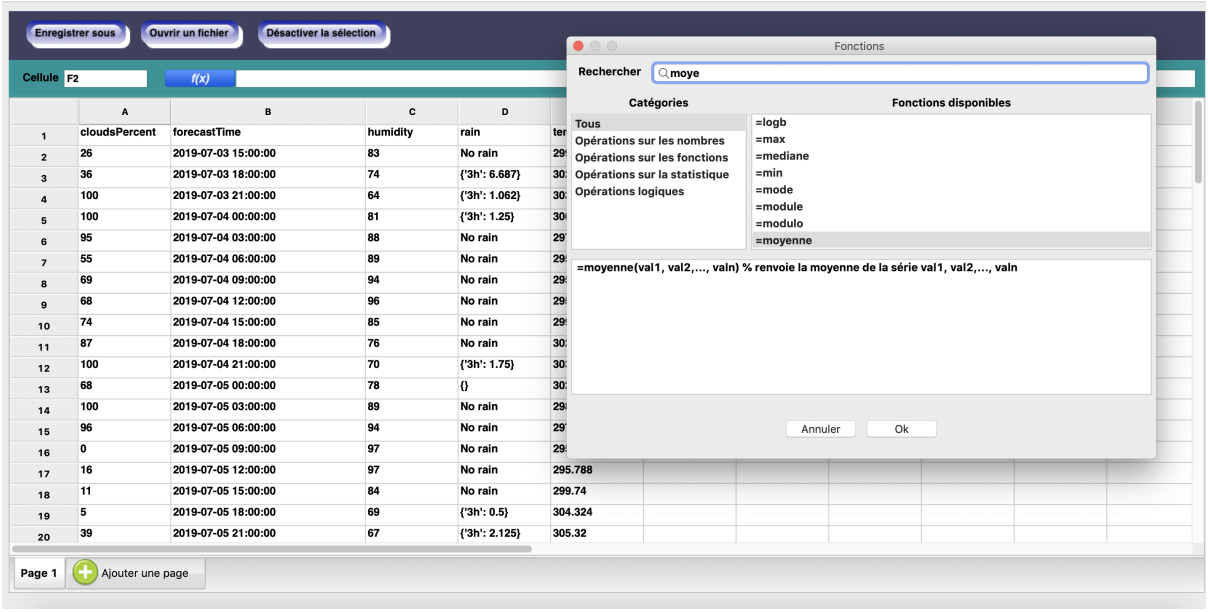


4.11.1 Fichiers Excel ou CSV

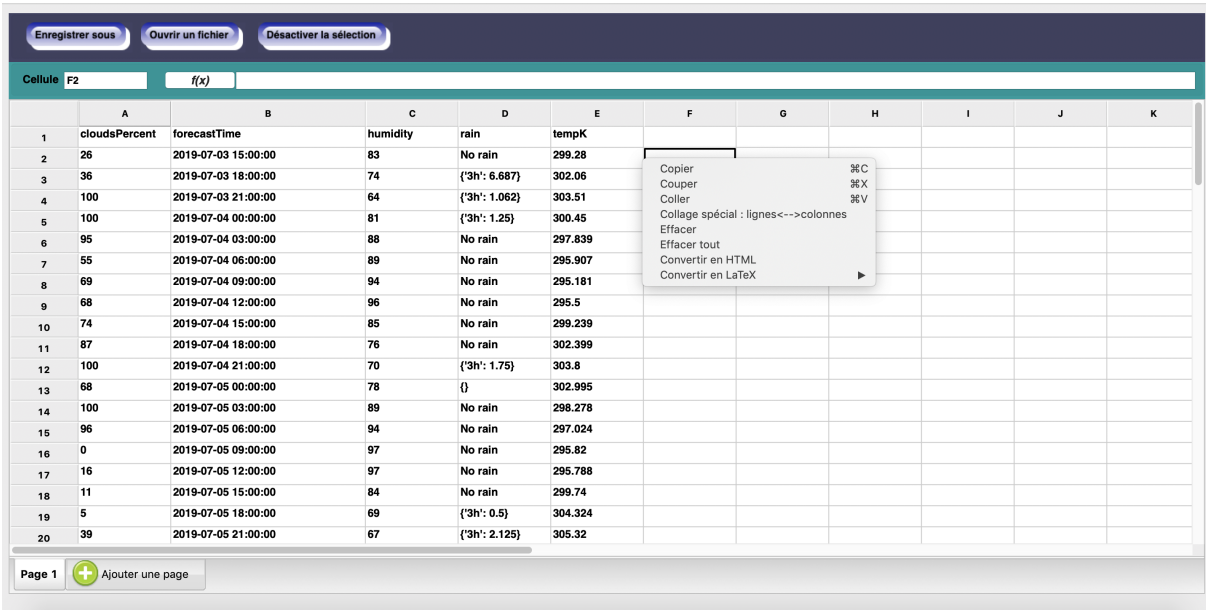


	A	B	C	D	E	F	G	H	I	J	K
1	REG	DEPT	CAV	COD_REG	COD_DEPT	COD_CAV	COD_CCRCA	COD_ENTITE	CCRCA	COMMUNE	Elementaire
2	DAKAR	DAKAR	DAKAR PLATEAU	01	1	301	11	01130111	GOREE	Goree	1
3	DAKAR	DAKAR	DAKAR PLATEAU	01	1	301	12	01130112	PLATEAU	Plateau	25
4	DAKAR	DAKAR	DAKAR PLATEAU	01	1	301	13	01130113	MEDINA	Medina	16
5	DAKAR	DAKAR	DAKAR PLATEAU	01	1	301	14	01130114	GUEULE TAPEE I	Fass Gueule Tape	17
6	DAKAR	DAKAR	DAKAR PLATEAU	01	1	301	15	01130115	FANN POINT E AI	Fann Point E Ami	16
7	DAKAR	DAKAR	GRAND DAKAR	01	1	301	21	01130121	GRAND DAKAR	Grand Dakar	10
8	DAKAR	DAKAR	GRAND DAKAR	01	1	301	22	01130122	BISCUITERIE	Biscuiterie	17
9	DAKAR	DAKAR	GRAND DAKAR	01	1	301	23	01130123	HLM	HLM	8
10	DAKAR	DAKAR	GRAND DAKAR	01	1	301	24	01130124	HANN BEL AIR	Hann Bel Air	23
11	DAKAR	DAKAR	GRAND DAKAR	01	1	301	25	01130125	SICAP LIBERTE	Sicap Liberte	12
12	DAKAR	DAKAR	GRAND DAKAR	01	1	301	26	01130126	DIEUPPEUL DER	Dieuppeul Derkle	13
13	DAKAR	DAKAR	ALMADIES	01	1	301	31	01130131	OUAKAM	Ouakam	21
14	DAKAR	DAKAR	ALMADIES	01	1	301	32	01130132	NGOR	Ngor	6
15	DAKAR	DAKAR	ALMADIES	01	1	301	33	01130133	YOFF	Yoff	37
16	DAKAR	DAKAR	ALMADIES	01	1	301	34	01130134	MERMOZ-SACRE	Mermoz Sacre Cc	32
17	DAKAR	DAKAR	PARCELLES ASSAINIES	01	1	301	41	01130141	GRAND YOFF	Grand Yoff	64
18	DAKAR	DAKAR	PARCELLES ASSAINIES	01	1	301	42	01130142	PATTE D'OIE	Patte d'Oie	10
19	DAKAR	DAKAR	PARCELLES ASSAINIES	01	1	301	43	01130143	PARCELLES ASS	Parcelles Assaini	70
20	DAKAR	DAKAR	PARCELLES ASSAINIES	01	1	301	44	01130144	CAMBERENE	Camberene	15



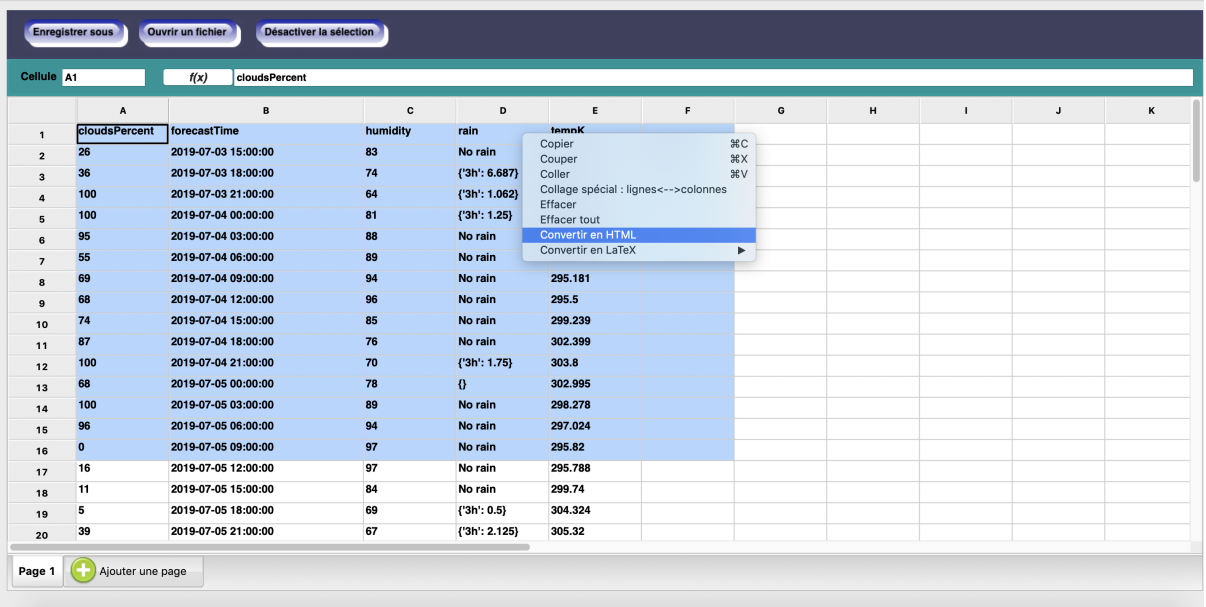


	A	B	C	D	E
1	cloudsPercent	forecastTime	humidity	rain	ter
2	26	2019-07-03 15:00:00	83	No rain	29
3	36	2019-07-03 18:00:00	74	{'3h': 6.687}	30
4	100	2019-07-03 21:00:00	64	{'3h': 1.062}	30
5	100	2019-07-04 00:00:00	81	{'3h': 1.25}	30
6	95	2019-07-04 03:00:00	88	No rain	29
7	55	2019-07-04 06:00:00	89	No rain	29
8	69	2019-07-04 09:00:00	94	No rain	29
9	68	2019-07-04 12:00:00	96	No rain	29
10	74	2019-07-04 15:00:00	85	No rain	29
11	87	2019-07-04 18:00:00	76	No rain	30
12	100	2019-07-04 21:00:00	70	{'3h': 1.75}	30
13	68	2019-07-05 00:00:00	78	{}	30
14	100	2019-07-05 03:00:00	89	No rain	29
15	96	2019-07-05 06:00:00	94	No rain	29
16	0	2019-07-05 09:00:00	97	No rain	29
17	16	2019-07-05 12:00:00	97	No rain	295.788
18	11	2019-07-05 15:00:00	84	No rain	299.74
19	5	2019-07-05 18:00:00	69	{'3h': 0.5}	304.324
20	39	2019-07-05 21:00:00	67	{'3h': 2.125}	305.32

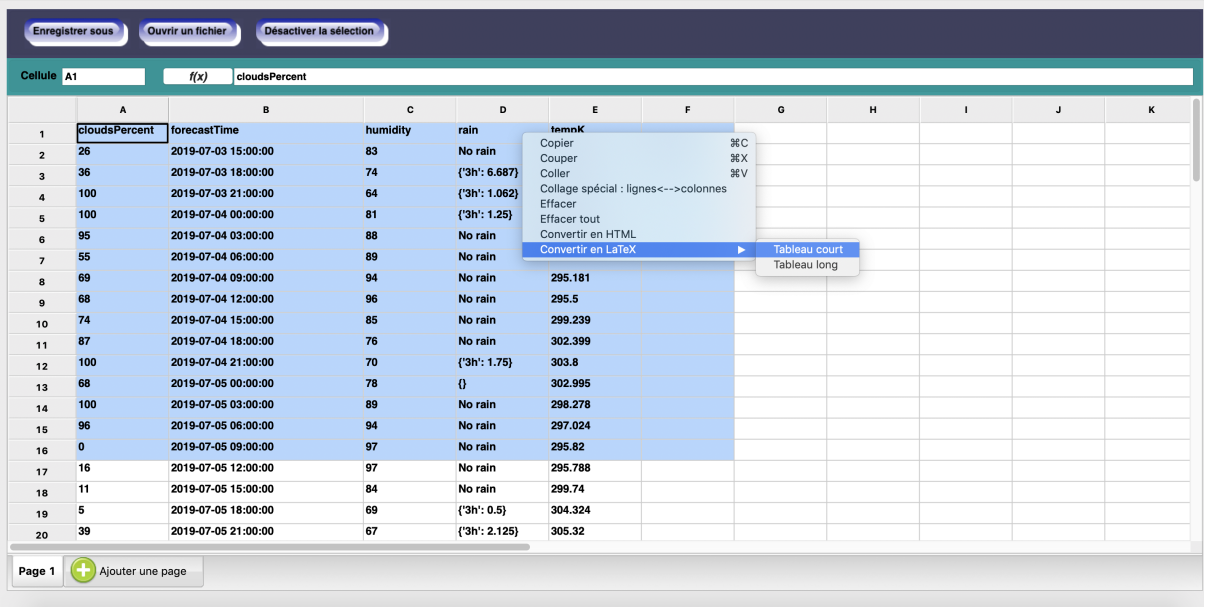


	A	B	C	D	E	F	G	H	I	J	K
1	cloudsPercent	forecastTime	humidity	rain	tempK						
2	26	2019-07-03 15:00:00	83	No rain	299.28						
3	36	2019-07-03 18:00:00	74	{'3h': 6.687}	302.06						
4	100	2019-07-03 21:00:00	64	{'3h': 1.062}	303.51						
5	100	2019-07-04 00:00:00	81	{'3h': 1.25}	300.45						
6	95	2019-07-04 03:00:00	88	No rain	297.839						
7	55	2019-07-04 06:00:00	89	No rain	295.907						
8	69	2019-07-04 09:00:00	94	No rain	295.181						
9	68	2019-07-04 12:00:00	96	No rain	295.5						
10	74	2019-07-04 15:00:00	85	No rain	299.239						
11	87	2019-07-04 18:00:00	76	No rain	302.399						
12	100	2019-07-04 21:00:00	70	{'3h': 1.75}	303.8						
13	68	2019-07-05 00:00:00	78	{}	302.995						
14	100	2019-07-05 03:00:00	89	No rain	298.278						
15	96	2019-07-05 06:00:00	94	No rain	297.024						
16	0	2019-07-05 09:00:00	97	No rain	295.82						
17	16	2019-07-05 12:00:00	97	No rain	295.788						
18	11	2019-07-05 15:00:00	84	No rain	299.74						
19	5	2019-07-05 18:00:00	69	{'3h': 0.5}	304.324						
20	39	2019-07-05 21:00:00	67	{'3h': 2.125}	305.32						

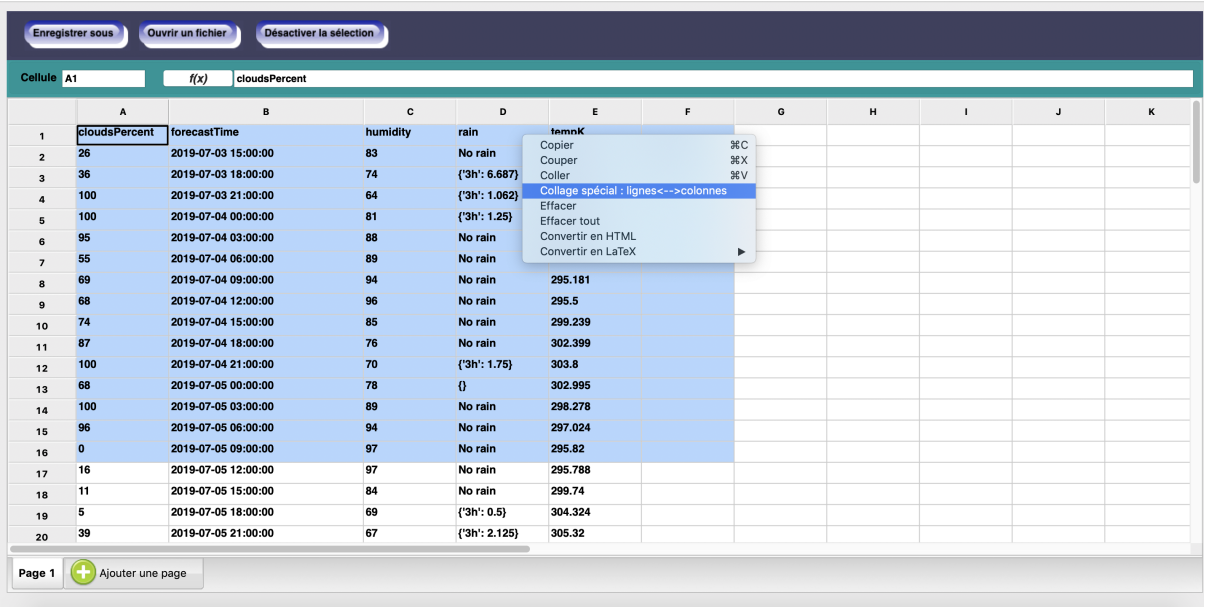
4.11.2 Opérations sur la feuille de calcul



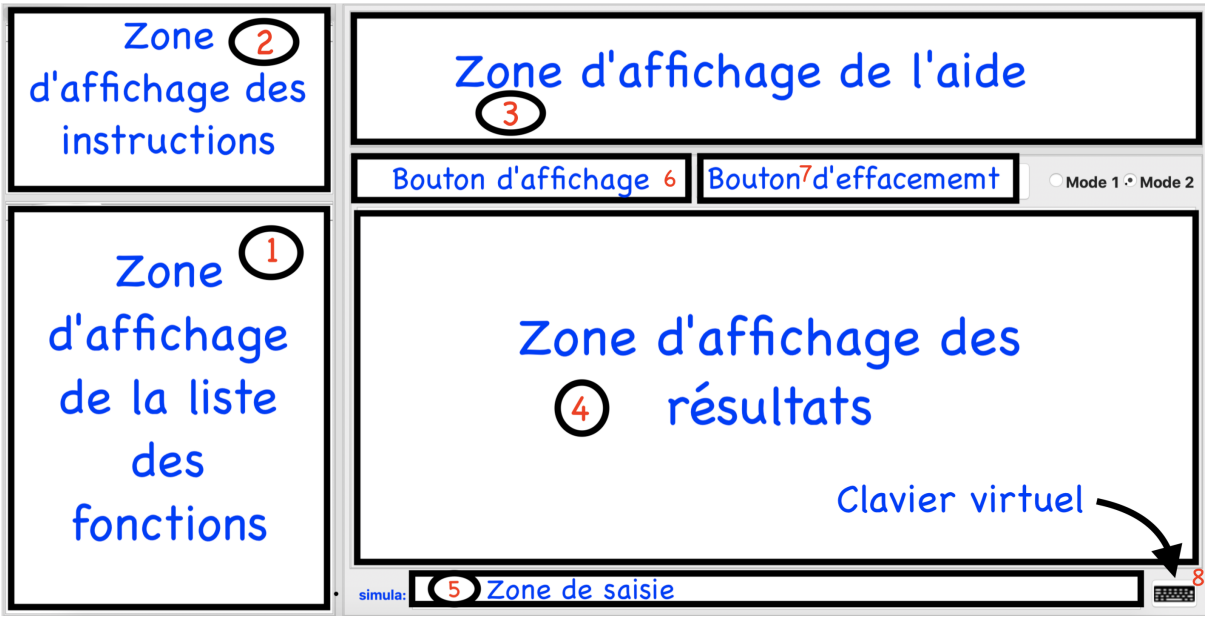
	A	B	C	D	E
1	cloudsPercent	forecastTime	humidity	rain	tempK
2	26	2019-07-03 15:00:00	83	No rain	
3	36	2019-07-03 18:00:00	74	{'3h': 6.687}	
4	100	2019-07-03 21:00:00	64	{'3h': 1.062}	
5	100	2019-07-04 00:00:00	81	{'3h': 1.25}	
6	95	2019-07-04 03:00:00	88	No rain	
7	55	2019-07-04 06:00:00	89	No rain	
8	69	2019-07-04 09:00:00	94	No rain	295.181
9	68	2019-07-04 12:00:00	96	No rain	295.5
10	74	2019-07-04 15:00:00	85	No rain	299.239
11	87	2019-07-04 18:00:00	76	No rain	302.399
12	100	2019-07-04 21:00:00	70	{'3h': 1.75}	303.8
13	68	2019-07-05 00:00:00	78	{}	302.995
14	100	2019-07-05 03:00:00	89	No rain	298.278
15	96	2019-07-05 06:00:00	94	No rain	297.024
16	0	2019-07-05 09:00:00	97	No rain	295.82
17	16	2019-07-05 12:00:00	97	No rain	295.788
18	11	2019-07-05 15:00:00	84	No rain	299.74
19	5	2019-07-05 18:00:00	69	{'3h': 0.5}	304.324
20	39	2019-07-05 21:00:00	67	{'3h': 2.125}	305.32



	A	B	C	D	E
1	cloudsPercent	forecastTime	humidity	rain	tempK
2	26	2019-07-03 15:00:00	83	No rain	
3	36	2019-07-03 18:00:00	74	{'3h': 6.687}	
4	100	2019-07-03 21:00:00	64	{'3h': 1.062}	
5	100	2019-07-04 00:00:00	81	{'3h': 1.25}	
6	95	2019-07-04 03:00:00	88	No rain	
7	55	2019-07-04 06:00:00	89	No rain	
8	69	2019-07-04 09:00:00	94	No rain	295.181
9	68	2019-07-04 12:00:00	96	No rain	295.5
10	74	2019-07-04 15:00:00	85	No rain	299.239
11	87	2019-07-04 18:00:00	76	No rain	302.399
12	100	2019-07-04 21:00:00	70	{'3h': 1.75}	303.8
13	68	2019-07-05 00:00:00	78	{}	302.995
14	100	2019-07-05 03:00:00	89	No rain	298.278
15	96	2019-07-05 06:00:00	94	No rain	297.024
16	0	2019-07-05 09:00:00	97	No rain	295.82
17	16	2019-07-05 12:00:00	97	No rain	295.788
18	11	2019-07-05 15:00:00	84	No rain	299.74
19	5	2019-07-05 18:00:00	69	{'3h': 0.5}	304.324
20	39	2019-07-05 21:00:00	67	{'3h': 2.125}	305.32



4.12 Calculs



Vous pouvez effectuer certaines opérations dans différents domaines des mathématiques sans avoir de connaissances en programmation.

- Opérations sur l'algèbre linéaire
- Opérations sur les fonctions et les suites
- Opérations sur les corps finis et les polynômes mod p
- Opérations sur les bases de Groebner et les polynômes multivariés
- Opérations sur les codes linéaires
- Opérations sur les cryptosystèmes classiques

Chapitre 5

Interface de programmation

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
simula : 3x
```

(suite sur la page suivante)

(suite de la page précédente)

```
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
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')
(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'})
```

(suite sur la page suivante)

(suite de la page précédente)

```
{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)
```

Paramètres **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
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()
```

(suite sur la page suivante)

(suite de la page précédente)

```
(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]
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)
```

(suite sur la page suivante)

(suite de la page précédente)

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

(suite sur la page suivante)

(suite de la page précédente)

```
20
25
30
```

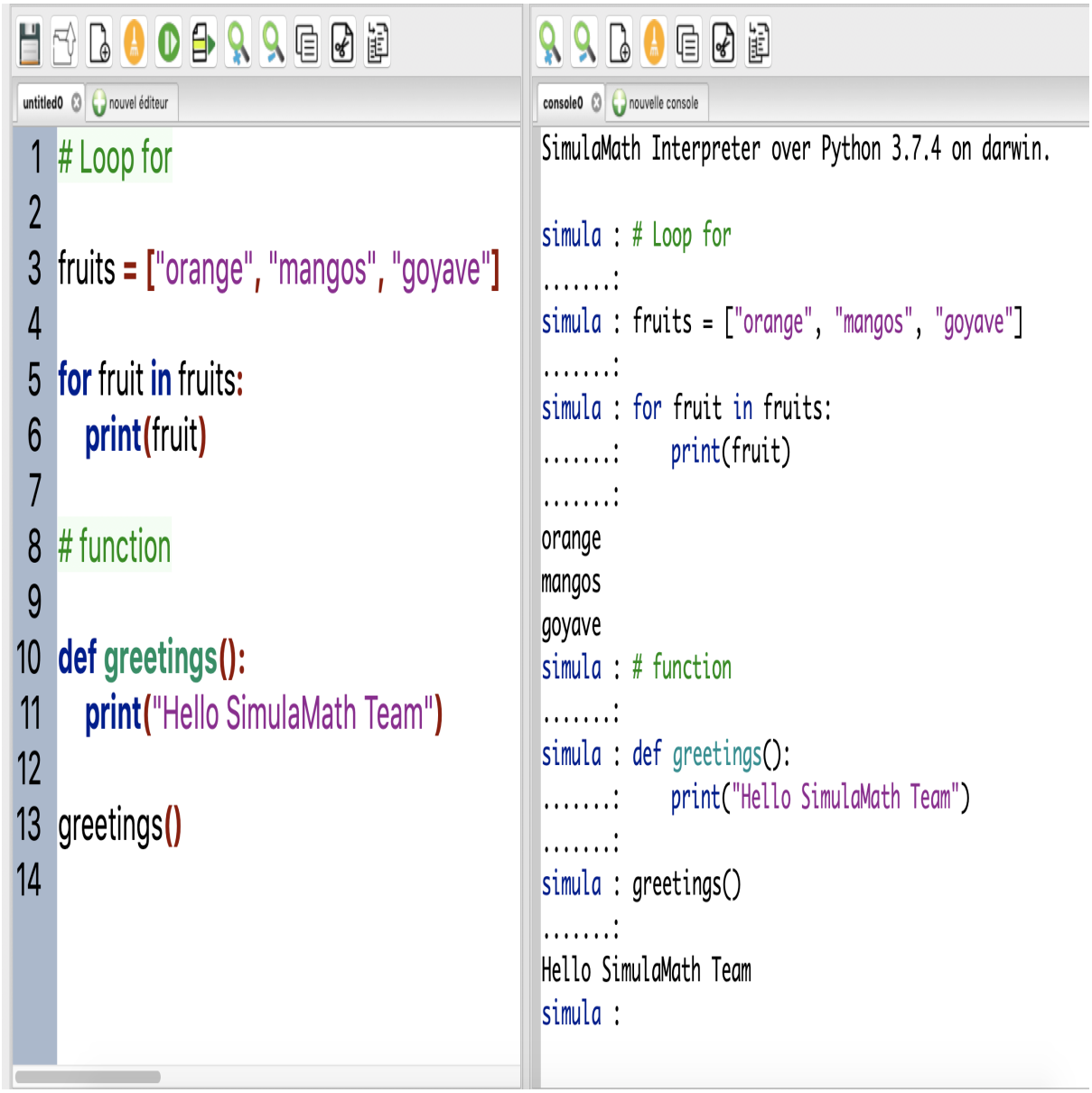
— **Functions**

```
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 Algèbre linéaire

5.2.1 Matrices

Operations over matrices

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

Returns an identity matrix of order n.

Paramètres **n** – an integer

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

(suite sur la page suivante)

(suite de la page précédente)

```
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
Matrix([
[5, 6,  7],
```

(suite sur la page suivante)

(suite de la page précédente)

```
[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]]))
simula : B*N == N*B
True
simula : B.is_diagonalizable()
True
```

(suite sur la page suivante)

(suite de la page précédente)

```
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()
[(-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`.

Paramètres **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([
[-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).

Paramètres **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.

Paramètres
— **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.

Paramètres
— **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],
```

(suite sur la page suivante)

(suite de la page précédente)

```
[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],
[ 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)
```

(suite sur la page suivante)

(suite de la page précédente)

```
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

Paramètres **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.

Paramètres **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

Paramètres **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.

Paramètres

— **expr** – an expression

— **symbols** – list of symbols

Type renvoyé (<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],
[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.

Paramètres

— **rows** – (an integer) the number or rows

— **cols** – (optional) the number or columns

EXAMPLES :

```
simula : ones_matrix(2, 4)
Matrix([
```

(suite sur la page suivante)

(suite de la page précédente)

```
[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.

Paramètres

— **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`.

Paramètres

— **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`.

Paramètres

— **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

Paramètres

— **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],
```

(suite sur la page suivante)

(suite de la page précédente)

```
[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`.

Paramètres **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`.

Paramètres **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 :

Paramètres
— **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``.

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

Renvoie 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.

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

Renvoie a boolean

EXAMPLES :

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

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

Renvoie 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
```

Paramètres **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`.

Paramètres

— **v** – a vector

— **family** – a list vectors

Renvoie 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
```

Paramètres **family** (*Iterable*) –

is_basis (*family*)

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

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

Renvoie 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.

Paramètres **family** – a set of vectors

Renvoie 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`.

Paramètres **domain** – a vector space

Renvoie a boolean True or False

linear_combination (*family*, *coeffs*)

Returns a linear combination of `family` by the coefficients `coeffs`.

Paramètres

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

Renvoie 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`.

Paramètres

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

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

subspace (*family*)

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

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

Renvoie 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]
[ 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.

Paramètres

- **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)
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)
3.14159265358979323846
```

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 $\frac{n!}{(n-k)! \times k!}$.

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`.

Paramètres **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
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.

Paramètres **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.

Paramètres

— **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`.

Paramètres

— **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).

Paramètres

— **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).

Paramètres

— **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.

Paramètres

— **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}.

Paramètres **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`.

Paramètres

— **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)
3.14159265358979323846
```

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`.

Paramètres **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`.

Paramètres

- **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`.

Paramètres

- **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`.

Paramètres

- **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`.

Paramètres **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
```

(suite sur la page suivante)

(suite de la page précédente)

```
simula : power_mod(99, 99876655, 13)
5
```

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

Returns the i-th prime number less than n.

Paramètres **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.

Paramètres **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.

Paramètres **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.

Paramètres **n** – a positive integer

EXAMPLES :

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

Paramètres **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`.

Paramètres **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`.

Paramètres **b** (*Optional[int]*) –

simula.api.ntheory.functions.**random_prime**(*a*, *b=None*)
Returns randomly a prime number between a and b.

Paramètres **b** (*Optional[int]*) –

simula.api.ntheory.functions.**random_prime_size**(*size*)
Returns randomly a prime of size size bits.

Paramètres **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
```

Paramètres **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.

Paramètres
- **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)
```

(suite sur la page suivante)

(suite de la page précédente)

```
[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`.

Paramètres **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`.

Paramètres **z** – a complex number

EXAMPLES :

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

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

return the exponential form of `z`.

Paramètres **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.

Paramètres **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.

Paramètres
- **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.

Paramètres **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.

Paramètres **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.

Paramètres **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|.

Paramètres **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.

Paramètres **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
```

(suite sur la page suivante)

(suite de la page précédente)

```
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)

Paramètres **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)
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.

Paramètres

— **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.

Paramètres
- **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
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
```

(suite sur la page suivante)

(suite de la page précédente)

```
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
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.

Paramètres

- **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.

Paramètres **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.

Paramètres **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.

Paramètres

— **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`.

Paramètres **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.

Paramètres **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.

Paramètres

— **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'
```

Paramètres

- **cipher** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[str, int]*) –

Type renvoyé 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'
```

Paramètres

— **message** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –

— **key** (*Union[str, int]*) –

Type renvoyé 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'
```

Paramètres

— **cipher** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –

— **key** (*Union[str, int]*) –

Type renvoyé 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'
```

Paramètres

— **message** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –

— **key** (*Union[str, int]*) –

Type renvoyé 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'
```

Paramètres

- **cipher** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*dict*) –

Type renvoyé 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'
```

Paramètres

- **message** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*dict*) –

Type renvoyé 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'
```

Paramètres

- **cipher** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[str, int]*) –

Type renvoyé str

classmethod encipher(*message*, *key*)

Encryption algorithm for Shift Cryptosystem.

EXAMPLES :

```
simula : M, k = "TOPSECRET", 7
simula : C = ShiftCryptosystem.encipher(M, k); C
'AVWZLJYLA'
```

Paramètres

- **message** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[str, int]*) –

Type renvoyé 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,
```

(suite sur la page suivante)

(suite de la page précédente)

```
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'
```

Paramètres

- **cipher** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*dict*) –

Type renvoyé 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'
```

Paramètres

- **message** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*dict*) –

Type renvoyé 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
```

(suite sur la page suivante)

(suite de la page précédente)

```
'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'
```

Paramètres

- **cipher** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[Tuple[str], Tuple[int]]*) –

Type renvoyé 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'
```

Paramètres

- **message** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[Tuple[str], Tuple[int]]*) –

Type renvoyé 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'
```

Paramètres

- **cipher** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –
- **key** (*Union[Tuple[str], Tuple[int]]*) –

Type renvoyé 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'
```

Paramètres

- **message** (*Union[str, List[int], List[str], Tuple[int], Tuple[str]]*) –

— **key** (*Union[Tuple[str], Tuple[int]]*) –

Type renvoyé 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.

Paramètres **message** (*str*) –

Type renvoyé 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`.

Paramètres
— **N** (*Optional[int]*) –
— **L** (*Optional[int]*) –

classmethod **signing**(*message*, *sk*, *hash_func=None*)
DSA signing algorithm.

Paramètres
— **message** (*int*) – an integer
— **sk** (*Tuple[int, int, int, int]*) – the private key `sk = (p, q, g, x)`
— **hash_func** – (optional) message digest

Type renvoyé Tuple[int, int]

classmethod **verifier**(*sign*, *pk*, *message*, *hash_func=None*)
DSA Verification algorithm.

Paramètres
— **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

Type renvoyé 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`.

Paramètres
— **size** (*Optional[int]*) –
— **p** (*Optional[int]*) –

classmethod signing(*message*, *sk*, *hash_func=None*)
ECDSA signing algorithm.

Paramètres
— **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

Type renvoyé Tuple[int, int]

classmethod verifier(*sign*, *pk*, *message*, *hash_func=None*)
ECDSA Verification algorithm.

Paramètres
— **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

Type renvoyé 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.

Paramètres
— **cipher** (*int*) – an integer

— **sk** (*int*) – the private key `sk = (p, a)`

Type renvoyé int

classmethod encipher(*message*, *pk*)

El Gamal encipher algorithm.

Paramètres

— **message** (*int*) – an integer

— **pk** (*int*) – the public key *pk = (p, g, ga)*`

Type renvoyé 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`.

Paramètres **size** (*Optional[int]*) –

classmethod signing(*message*, *sk*, *hash_func=None*)

El Gamal signing algorithm.

Paramètres

— **message** (*int*) – an integer

— **sk** (*Tuple[int, int, int]*) – the private key `sk = (p, g, a)`

— **hash_func** – (optional) message digest

Type renvoyé Tuple[int, int]

classmethod verifier(*sign*, *pk*, *message*, *hash_func=None*)

El Gamal Verification algorithm.

Paramètres

— **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

Type renvoyé 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.

Paramètres
- **cipher** (*int*) – an integer
- **sk** (*Union[Tuple[int, int], Tuple[int, int, int]]*) – the private key `sk = (p, q, d)`

Type renvoyé int

classmethod encipher(*message*, *pk*)
RSA encipher algorithm.

Paramètres
- **message** (*int*) – an integer
- **pk** (*Tuple[int, int]*) – the public key *pk = (N, e)*`

Type renvoyé int

classmethod keygen(*size=None*)
Returns a keypair `sk = (p, q, d)` and `pk = (N, e)` when `p` and `q` have size `size`.

Paramètres **size** (*Optional[int]*) –

classmethod signing(*message*, *sk*, *hash_func=None*)
RSA signing algorithm.

Paramètres
- **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

Type renvoyé int

classmethod verifier(*sign*, *pk*, *message*, *hash_func=None*)
RSA Verification algorithm.

Paramètres
- **sign** (*int*) – an integer
- **pk** (*Tuple[int, int]*) – the public key `pk = (N, e)`
- **message** (*int*) – an integer
- **hash_func** – (optional) message digest

Type renvoyé 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`.

Paramètres
- **size** (*Optional[int]*) –
- **p** (*Optional[int]*) –

classmethod signing(*message*, *sk*, *hash_func=None*)
ECDSA signing algorithm.

Paramètres
- **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

Type renvoyé Tuple[int, int]

classmethod verifier(*sign*, *pk*, *message*, *hash_func=None*)

ECDSA Verification algorithm.

Paramètres
- **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

Type renvoyé 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
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],
```

(suite sur la page suivante)

(suite de la page précédente)

```
[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`.

Type renvoyé *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`.

Type renvoyé *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`.

Type renvoyé *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],
[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).

Paramètres

- **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
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.

Paramètres **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]])
```

Type renvoyé *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.

Paramètres

- **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 :

Paramètres

- **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`.

Paramètres
- **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
```

(suite sur la page suivante)

(suite de la page précédente)

```
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()
[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`.

Paramètres **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.

Paramètres
— **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()
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.

Paramètres

— **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`.

Chapitre 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

Sympy

Pandas

WEVER 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.

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.

Chapitre 7

Index et tableaux

— genindex
— search

Index des modules Python

S

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z