TI-Nspire™ CX CAS
Guide de référence

Pour en savoir plus sur la technologie TI, consultez l'aide en ligne à l'adresse education.ti.com/eguide.

Informations importantes

Sauf disposition contraire expressément formulée dans la licence qui accompagne un programme, Texas Instruments n'émet aucune garantie expresse ou implicite, y compris sans s'y limiter, toute garantie implicite de valeur marchande et d'adéquation à un usage particulier, concernant les programmes ou la documentation, ceux-ci étant fournis « tels quels » sans autre recours. En aucun cas, Texas Instruments ne saurait être tenue responsable de dommages spéciaux, collatéraux, fortuits ou indirects en relation avec, ou imputables à l'achat ou à l'utilisation de ce matériel. La seule responsabilité exclusive de Texas Instruments, indépendamment de la forme d'action, ne saurait dépasser le prix fixé dans la licence pour ce programme. Par ailleurs, la responsabilité de Texas Instruments ne saurait être engagée pour quelque réclamation que ce soit en rapport avec l'utilisation desdits matériels par toute autre tierce partie.

Les produits peuvent varier légèrement des images fournies.

Table des matières

Modèles d'expression

Les modèles d'expression facilitent la saisie d'expressions mathématiques en notation standard. Lorsque vous utilisez un modèle, celui-ci s'affiche sur la ligne de saisie, les petits carrés correspondants aux éléments que vous pouvez saisir. Un curseur identifie l'élément que vous pouvez saisir.

Utilisez les touches fléchées ou appuyez sur [tab] pour déplacer le curseur sur chaque élément, puis tapez la valeur ou l'expression correspondant à chaque élément. Appuyez sur [enter] ou [ctrl] [enter] pour calculer l'expression.

Modèle Fraction

Touches [ctrl] [÷]

Remarque : Voir aussi **/ (division)**, page 187.

Exemple :

Modèle Exposant

Touche [^]

Remarque : Tapez la première valeur, appuyez sur [^], puis entrez l'exposant. Pour ramener le curseur sur la ligne de base, appuyez sur la flèche droite (▶).

Remarque : Voir aussi **^ (puissance)**, page 187.

Exemple :

Modèle Racine carrée

Touches [ctrl] [x²]

Remarque : Voir aussi **√() (racine carrée)**, page 195.

Exemple :

Modèle Racine n-ième

Touches [ctrl] [^]

Remarque : Voir aussi **root()**, page 139.

Exemple :

Modèle e Exposant

Touches e^x

$e^{\square}$

La base du logarithme népérien *e* élevée à une puissance

Remarque : Voir aussi **e^()**, page 45.

Exemple :

Modèle Logarithme

Touches ctrl 10^x

$\log_{\square}(\square)$

Calcule le logarithme selon la base spécifiée. Par défaut la base est 10, dans ce cas ne spécifiez pas de base.

Remarque : Voir aussi **log()**, page 92.

Exemple :

$\log_4(2.)$ 0.5

Modèle Fonction définie par morceaux (2 morceaux)

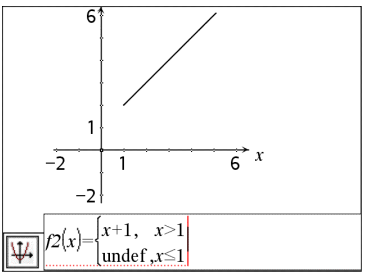
Catalogue >

$\begin{cases} \square, & \square \\ \square, & \square \end{cases}$

Permet de créer des expressions et des conditions pour une fonction définie par deux morceaux.- Pour ajouter un morceau supplémentaire, cliquez dans le modèle et appliquez-le de nouveau.

Remarque : Voir aussi **piecewise()**, page 118.

Exemple :



Modèle Fonction définie par morceaux (n morceaux)

Catalogue >

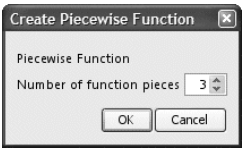
Permet de créer des expressions et des conditions pour une fonction définie par *n*-morceaux. Le système vous invite à définir *n*.

Exemple :

Voir l'exemple donné pour le modèle Fonction définie par morceaux (2 morceaux).

Modèle Fonction définie par morceaux (n morceaux)

Catalogue >



Remarque : Voir aussi **piecewise()**, page 118.

Modèle Système de 2 équations

Catalogue >

$\begin{cases} \square \\ \square \end{cases}$

Crée une système de deux équations . Pour ajouter une nouvelle ligne à un système existant, cliquez dans le modèle et appliquez-le de nouveau.

Remarque : Voir aussi **system()**, page 161.

Exemple :

$\text{solve}\left(\begin{cases} x+y=0 \\ x-y=5 \end{cases}, x, y\right) \quad x=\frac{5}{2} \text{ and } y=\frac{-5}{2}$

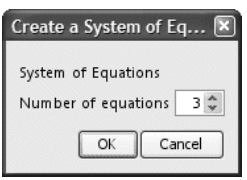
$\text{solve}\left(\begin{cases} y=x^2-2 \\ x+2\cdot y=-1 \end{cases}, x, y\right)$

$x=\frac{-3}{2} \text{ and } y=\frac{1}{4} \text{ or } x=1 \text{ and } y=-1$

Modèle Système de n équations

Catalogue >

Permet de créer un système de Nlinéaires. Le système vous invite à définir N.



Remarque : Voir aussi **system()**, page 161.

Exemple :

Voir l'exemple donné pour le modèle Système de 2 équations.

Modèle Valeur absolue

Catalogue >

$\left|\square\right|$

Remarque : Voir aussi **abs()**, page 7.

Exemple :

$\left|\left\{2,-3,4,-4^3\right\}\right| \qquad \left\{2,3,4,64\right\}$

Modèle dd°mm'ss.ss''

Catalogue >

$\square°\square'\square''$

Exemple :

Permet d'entrer des angles en utilisant le format **dd°mm'ss.ss''**, où **dd** correspond au nombre de degrés décimaux, **mm** au nombre de minutes et **ss.ss** au nombre de secondes.

Modèle Matrice (2 x 2)

Catalogue >

$\begin{bmatrix}\square & \square \\ \square & \square\end{bmatrix}$

Exemple :

Crée une matrice de type 2 x 2.

Modèle Matrice (1 x 2)

Catalogue >

$\begin{bmatrix}\square & \square\end{bmatrix}$.

Exemple :

$\text{crossP}\left(\begin{bmatrix}1 & 2\end{bmatrix},\begin{bmatrix}3 & 4\end{bmatrix}\right) \qquad \begin{bmatrix}0 & 0 & -2\end{bmatrix}$

Modèle Matrice (2 x 1)

Catalogue >

$\begin{bmatrix}\square \\ \square\end{bmatrix}$

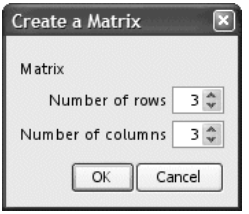
Exemple :

$\begin{bmatrix}5\\8\end{bmatrix}\cdot 0.01 \qquad \begin{bmatrix}0.05\\0.08\end{bmatrix}$

Modèle Matrice (m x n)

Catalogue >

Le modèle s'affiche après que vous ayez saisi le nombre de lignes et de colonnes.



Remarque : si vous créez une matrice dotée de nombreuses lignes et colonnes,

Exemple :

$\text{diag}\left(\begin{bmatrix}4 & 2 & 6\\1 & 2 & 3\\5 & 7 & 9\end{bmatrix}\right) \qquad \begin{bmatrix}4 & 2 & 9\end{bmatrix}$

Modèle Matrice (m x n)

Catalogue >

son affichage peut prendre quelques minutes.

Modèle Somme (Σ)

Catalogue >

$$\sum_{\square=\square}^{\square}(\square)$$

Exemple :

$$\sum_{n=3}^{7}(n) \qquad 25$$

Remarque : voir aussi **Σ() (sumSeq)**, page 196.

Modèle Produit (Π)

Catalogue >

$$\prod_{\square=\square}^{\square}(\square)$$

Exemple :

$$\prod_{n=1}^{5}\left(\frac{1}{n}\right) \qquad \frac{1}{120}$$

Remarque : Voir aussi **Π() (prodSeq)**, page 196.

Modèle Dérivée première

Catalogue >

$$\frac{d}{d\square}(\square)$$

Par exemple :

Remarque : voir aussi **d() (dérivée)**, page 195.

Modèle Dérivée seconde

Catalogue >

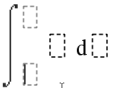
$$\frac{d^2}{d\square^2}(\square)$$

Par exemple :

Remarque : voir aussi **d() (dérivée)**, page 195.

Modèle Intégrale définie

Catalogue > 



Exemple :

Liste alphabétique

Les éléments dont le nom n'est pas alphabétique (comme +, !, et >) apparaissent à la fin de cette section, à partir de la page 186. Sauf indication contraire, tous les exemples fournis dans cette section ont été réalisés en mode de réinitialisation par défaut et toutes les variables sont considérées comme indéfinies.

A

abs() Catalogue >

abs(*Liste1***)⇒***liste*

abs(*Matrice1***)⇒***matrice*

Donne la valeur absolue de l'argument.

Remarque : Voir aussi **Modèle Valeur absolue**, page 3.

Si l'argument est un nombre complexe, donne le module de ce nombre.

Remarque : toutes les variables non affectées sont considérées comme réelles.

amortTbl() Catalogue >

amortTbl(*NPmt***,***N***,***I***,***PV***,** [*Pmt*]**,** [*FV*]**,** [*PpY*]**,** [*CpY*]**,** [*PmtAt*]**,** [*valArrondi*]**)⇒***matrice*

Fonction d'amortissement affichant une matrice représentant un tableau d'amortissement pour un ensemble d'arguments TVM.

NPmt est le nombre de versements à inclure au tableau. Le tableau commence avec le premier versement.

N, *I*, *PV*, *Pmt*, *FV*, *PpY*, *CpY* et *PmtAt* sont décrits dans le tableau des arguments TVM, page 171.

- Si vous omettez *Pmt*, il prend par défaut la valeur *Pmt*=**tvmPmt (***N*,*I*,*PV*,*FV*,*PpY*,*CpY*,*PmtAt***)**.
- Si vous omettez *FV*, il prend par défaut la valeur *FV*=0.
- Les valeurs par défaut pour *PpY*, *CpY*

amortTbl(12,60,10,5000,,,12,12)

0	0.	0.	5000.
1	-41.67	-64.57	4935.43
2	-41.13	-65.11	4870.32
3	-40.59	-65.65	4804.67
4	-40.04	-66.2	4738.47
5	-39.49	-66.75	4671.72
6	-38.93	-67.31	4604.41
7	-38.37	-67.87	4536.54
8	-37.8	-68.44	4468.1
9	-37.23	-69.01	4399.09
10	-36.66	-69.58	4329.51
11	-36.08	-70.16	4259.35
12	-35.49	-70.75	4188.6

et *PmtAt* sont les mêmes que pour les fonctions TVM.

valArrondi spécifie le nombre de décimales pour arrondissement. Valeur par défaut=2.

Les colonnes dans la matrice résultante apparaissent dans l'ordre suivant : Numéro de versement, montant versé pour les intérêts, montant versé pour le capital et solde.

Le solde affiché à la ligne *n* correspond au solde après le versement *n*.

Vous pouvez utiliser la matrice de sortie pour insérer les valeurs des autres fonctions d'amortissement **ΣInt()** et **ΣPrn()**, page 197 et **bal()**, page 16.

Matrice booléenne

Donne true (vrai) ou false (faux) ou une forme simplifiée de l'entrée initiale.

Entier1 **and** *Entier2*⇒*entier*

Compare les représentations binaires de deux entiers réels en appliquant un **and** bit à bit. En interne, les deux entiers sont convertis en nombres binaires 64 bits signés. Lorsque les bits comparés correspondent, le résultat est 1 si dans les deux cas il s'agit d'un bit 1 ; dans les autres cas, le résultat est 0. La valeur donnée représente le résultat des bits et elle est affichée selon le mode Base utilisé.

Les entiers de tout type de base sont admis. Pour une entrée binaire ou hexadécimale, vous devez utiliser respectivement le préfixe 0b ou 0h. Tout entier sans préfixe est considéré comme un nombre en écriture décimale (base 10).

$x \geq 3 \text{ and } x \geq 4$ → $x \geq 4$

$\{x \geq 3, x \leq 0\} \text{ and } \{x \geq 4, x \leq -2\}$ → $\{x \geq 4, x \leq -2\}$

En mode base Hex :

0h7AC36 and 0h3D5F → 0h2C16

Important : utilisez le chiffre zéro et pas la lettre O.

En mode base Bin :

0b100101 and 0b100 → 0b100

En mode base Dec :

37 and 0b100 → 4

Remarque : une entrée binaire peut comporter jusqu'à 64 chiffres (sans compter le préfixe 0b) ; une entrée hexadécimale jusqu'à 16 chiffres.

and

Catalogue >

Si vous entrez un nombre dont le codage binaire signé dépasse 64 bits, il est ramené à l'aide d'une congruence dans la plage appropriée.

angle()

Catalogue >

Donne l'argument de l'expression passée en paramètre, celle-ci étant interprétée comme un nombre complexe.

En mode Angle en degrés :

$\text{angle}(0+2\cdot i)$ 90

En mode Angle en grades :

$\text{angle}(0+3\cdot i)$ 100

En mode Angle en radians :

angle(*Liste1***)**⇒*liste*

angle(*Matrice1***)**⇒*matrice*

Donne la liste ou la matrice des arguments des éléments de *Liste1* ou *Matrice1*, où chaque élément est interprété comme un nombre complexe représentant un point de coordonnée rectangulaire à deux dimensions.

ANOVA

Catalogue >

ANOVA *Liste1*,*Liste2*[,*Liste3*,...,*Liste20*] [,*Indicateur*]

Effectue une analyse unidirectionnelle de variance pour comparer les moyennes de deux à vingt populations. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Indicateur=0 pour Données, *Indicateur*=1 pour Stats

Variable de sortie	Description
stat.F	Valeur de F statistique

Variable de sortie	Description
stat.PVal	Plus petit seuil de signification permettant de rejeter l'hypothèse nulle
stat.df	Degré de liberté des groupes
stat.SS	Somme des carrés des groupes
stat.MS	Moyenne des carrés des groupes
stat.dfError	Degré de liberté des erreurs
stat.SSError	Somme des carrés des erreurs
stat.MSError	Moyenne des carrés des erreurs
stat.sp	Écart-type du groupe
stat.xbarlist	Moyenne des entrées des listes
stat.CLowerList	Limites inférieures des intervalles de confiance de 95 % pour la moyenne de chaque liste d'entrée
stat.CUpperList	Limites supérieures des intervalles de confiance de 95 % pour la moyenne de chaque liste d'entrée

ANOVA2way

Catalogue >

ANOVA2way *Liste1*,*Liste2*[,*...*[,*Liste10*]] [,*NivLign*]

Effectue une analyse de variance à deux facteurs pour comparer les moyennes de deux à dix populations. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

NivLign=0 pour Bloc

NivLign=2,3,...,*Len*-1, pour 2 facteurs, où *Len*=length(*Liste1*)=length(*Liste2*) = ... = length(*Liste10*) et *Len* / *NivLign* ∈ {2,3,...}

Sorties : Bloc

Variable de sortie	Description
stat.F	F statistique du facteur de colonne
stat.PVal	Plus petit seuil de signification permettant de rejeter l'hypothèse nulle
stat.df	Degré de liberté du facteur de colonne
stat.SS	Somme des carrés du facteur de colonne

Variable de sortie	Description
stat.MS	Moyenne des carrés du facteur de colonne
stat.FBlock	F statistique du facteur
stat.PValBlock	Plus petite probabilité permettant de rejeter l'hypothèse nulle
stat.dfBlock	Degré de liberté du facteur
stat.SSBlock	Somme des carrés du facteur
stat.MSBlock	Moyenne des carrés du facteur
stat.dfError	Degré de liberté des erreurs
stat.SSError	Somme des carrés des erreurs
stat.MSError	Moyenne des carrés des erreurs
stat.s	Écart-type de l'erreur

Sorties FACTEUR DE COLONNE

Variable de sortie	Description
stat.Fcol	F statistique du facteur de colonne
stat.PValCol	Valeur de probabilité du facteur de colonne
stat.dfCol	Degré de liberté du facteur de colonne
stat.SSCol	Somme des carrés du facteur de colonne
stat.MSCol	Moyenne des carrés du facteur de colonne

Sorties FACTEUR DE LIGNE

Variable de sortie	Description
stat.Frow	F statistique du facteur de ligne
stat.PValRow	Valeur de probabilité du facteur de ligne
stat.dfRow	Degré de liberté du facteur de ligne
stat.SSRow	Somme des carrés du facteur de ligne
stat.MSRow	Moyenne des carrés du facteur de ligne

Sorties INTERACTION

Variable de sortie	Description
stat.FInteract	F statistique de l'interaction
stat.PValInteract	Valeur de probabilité de l'interaction
stat.dfInteract	Degré de liberté de l'interaction
stat.SSInteract	Somme des carrés de l'interaction
stat.MSInteract	Moyenne des carrés de l'interaction

Sorties ERREUR

Variable de sortie	Description
stat.dfError	Degré de liberté des erreurs
stat.SSError	Somme des carrés des erreurs
stat.MSError	Moyenne des carrés des erreurs
s	Écart-type de l'erreur

Ans

Touches ctrl (–)

Ans⇒*valeur*

Donne le résultat de la dernière expression calculée.

56	56
56+4	60
60+4	64

approx()

Catalogue >

Donne une approximation décimale de l'argument sous forme d'expression, dans la mesure du possible, indépendamment du mode **Auto ou Approché** utilisé.

Ceci est équivalent à la saisie de l'argument suivie d'une pression sur ctrl enter.

$\text{approx}\left(\frac{1}{3}\right)$	0.333333
$\text{approx}\left(\left\{\frac{1}{3},\frac{1}{9}\right\}\right)$	{0.333333,0.111111}
$\text{approx}(\{\sin(\pi),\cos(\pi)\})$	{0.,-1.}
$\text{approx}([\sqrt{2} \quad \sqrt{3}])$	[1.41421 1.73205]
$\text{approx}\left(\begin{bmatrix}\frac{1}{3} & \frac{1}{9}\end{bmatrix}\right)$	[0.333333 0.111111]

approx(*Liste1*)⇒*liste*

approx(*Matrice1*)⇒*matrice*

$\text{approx}(\{\sin(\pi),\cos(\pi)\})$	{0.,-1.}
$\text{approx}([\sqrt{2} \quad \sqrt{3}])$	[1.41421 1.73205]

approx() Catalogue >

Donne une liste ou une *matrice* d'éléments pour lesquels une approximation décimale a été calculée, dans la mesure du possible.

▸approxFraction() Catalogue >

Liste ▸**approxFraction(**[*tol*]**)**⇒*liste*

Matrice ▸**approxFraction (**[*tol*]**)**⇒*matrice*

Donne l'entrée sous forme de fraction en utilisant une tolérance *tol*. Si *tol* est omis, la tolérance 5.E-14 est utilisée.

Remarque : vous pouvez insérer cette fonction à partir du clavier de l'ordinateur en entrant **`@>approxFraction(`**...**`)`**.

$\frac{1}{2}+\frac{1}{3}+\tan(\pi)$ → 0.833333

$0.83333333333333 \blacktriangleright \text{approxFraction}(5.\text{E}^{-}14)$ → $\frac{5}{6}$

$\{\pi,1.5\} \blacktriangleright \text{approxFraction}(5.\text{E}^{-}14)$ → $\left\{\frac{5419351}{1725033},\frac{3}{2}\right\}$

approxRational() Catalogue >

approxRational(*Liste*[**,** *tol*]**)**⇒*liste*

approxRational(*Matrice*[**,** *tol*]**)**⇒*matrice*

Donne l'argument sous forme de fraction en utilisant une tolérance *tol*. Si *tol* est omis, la tolérance 5.E-14 est utilisée.

$\text{approxRational}(0.333,5\cdot 10^{-5})$ → $\frac{333}{1000}$

$\text{approxRational}(\{0.2,0.33,4.125\},5.\text{E}^{-}14)$ → $\left\{\frac{1}{5},\frac{33}{100},\frac{33}{8}\right\}$

arccos() **Voir $\cos^{-1}$(), page 28.**

arccosh() **Voir $\cosh^{-1}$(), page 29.**

arccot() **Voir $\cot^{-1}$(), page 30.**

arccoth() **Voir $\coth^{-1}()$, page 30.**

arccsc() **Voir $\csc^{-1}()$, page 33.**

arccsch() **Voir $\text{csch}^{-1}()$, page 33.**

arcsec() **Voir $\sec^{-1}()$, page 143.**

arcsech() **Voir $\text{sech}^{-1}()$, page 143.**

arcsin() **Voir $\sin^{-1}()$, page 151.**

arcsinh() **Voir $\sinh^{-1}()$, page 152.**

arctan() **Voir $\tan^{-1}()$, page 162.**

arctanh() **Voir $\tanh^{-1}()$, page 163.**

augment() **Catalogue >**

augment(*Liste1, Liste2***)⇒***liste*

Donne une nouvelle liste obtenue en plaçant les éléments de *Liste2* à la suite de ceux de *Liste1*.

$\text{augment}(\{1,-3,2\},\{5,4\})$ $\{1,-3,2,5,4\}$

augment() Catalogue >

augment(*Matrice1***,** *Matrice2***)⇒***matrice*

Donne une nouvelle matrice obtenue en ajoutant les lignes/colonnes de la *Matrice2* à celles de la *Matrice1*. Les matrices doivent avoir le même nombre de lignes et *Matrice2* est ajoutée à *Matrice1* via la création de nouvelles colonnes. *Matrice1* et *Matrice2* ne sont pas modifiées.

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \rightarrow m1 \qquad \begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$$

$$\begin{bmatrix}5\\ 6\end{bmatrix} \rightarrow m2 \qquad \begin{bmatrix}5\\ 6\end{bmatrix}$$

$$\text{augment}(m1,m2) \qquad \begin{bmatrix}1 & 2 & 5\\ 3 & 4 & 6\end{bmatrix}$$

avgRC() Catalogue >

avgRC(*Expr1***,** *Var* [=*Valeur*] [**,** *Incrément*]**)⇒***expression*

avgRC(*Expr1***,** *Var* [=*Valeur*] [**,** *Liste1*]**)⇒***liste*

avgRC(*Liste1***,** *Var* [=*Valeur*] [**,** *Incrément*]**)⇒***liste*

avgRC(*Matrice1***,** *Var* [=*Valeur*] [**,** *Incrément*]**)⇒***matrice*

Donne le taux d'accroissement moyen (quotient à différence antérieure) de l'expression.

Expr1 peut être un nom de fonction défini par l'utilisateur (voir **Func**).

Quand la *valeur* est spécifiée, celle-ci prévaut sur toute affectation de variable ou substitution précédente de type « | » pour la variable.

Incrément correspond à la valeur de l'incrément. Si *Incrément* n'est pas spécifié, il est fixé par défaut à 0,001.

Notez que la fonction comparable **nDeriv()** utilise le quotient à différence symétrique.

Notez que la fonction comparable **centralDiff ()** utilise le quotient à différence centrée.

bal() Catalogue >

bal(*NPmt*,*N*,*I*,*PV*,[*Pmt*], [*FV*], [*PpY*], [*CpY*], [*PmtAt*], [*valArrondi*]**)**⇒*valeur*

bal(*NPmt*,*tblAmortissement***)**⇒*valeur*

Fonction d'amortissement destinée à calculer le solde après versement d'un montant spécifique.

N, *I*, *PV*, *Pmt*, *FV*, *PpY*, *CpY* et *PmtAt* sont décrits dans le tableau des arguments TVM, page 171.

NPmt indique le numéro de versement après lequel vous souhaitez que les données soient calculées.

N, *I*, *PV*, *Pmt*, *FV*, *PpY*, *CpY* et *PmtAt* sont décrits dans le tableau des arguments TVM, page 171.

- Si vous omettez *Pmt*, il prend par défaut la valeur *Pmt*=**tvmPmt (***N*,*I*,*PV*,*FV*,*PpY*,*CpY*,*PmtAt***)**.
- Si vous omettez *FV*, il prend par défaut la valeur *FV*=0.
- Les valeurs par défaut pour *PpY*, *CpY* et *PmtAt* sont les mêmes que pour les fonctions TVM.

valArrondi spécifie le nombre de décimales pour arrondissement. Valeur par défaut=2.

bal(*NPmt*,*tblAmortissement***)** calcule le solde après le numéro de paiement *NPmt*, sur la base du tableau d'amortissement *tblAmortissement*. L'argument *tblAmortissement* doit être une matrice au format décrit à **tblAmortissement()**, page 7.

Remarque : voir également **Σint()** et **Σprn()**, page 197.

bal(5,6,5.75,5000,,12,12)	833.11

tbl:=amortTbl(6,6,5.75,5000,,12,12)

0	0.	0.	5000.
1	−23.35	−825.63	4174.37
2	−19.49	−829.49	3344.88
3	−15.62	−833.36	2511.52
4	−11.73	−837.25	1674.27
5	−7.82	−841.16	833.11
6	−3.89	−845.09	−11.98

bal(4,*tbl*)	1674.27

Entier1 ▸**Base2**⇒*entier*

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **@>Base2**.

Convertit *Entier1* en nombre binaire. Les nombres binaires et les nombres hexadécimaux présentent toujours respectivement un préfixe, 0b ou 0h. Zéro et pas la lettre O, suivi de b ou h.

0b *nombreBinaire*

0h *nombreHexadécimal*

Une entrée binaire peut comporter jusqu'à 64 chiffres (sans compter le préfixe 0b) ; une entrée hexadécimale jusqu'à 16 chiffres.

Si *Entier1* est entré sans préfixe, il est considéré comme un nombre en écriture décimale (base 10). Le résultat est affiché sous forme binaire, indépendamment du mode Base utilisé.

Les nombres négatifs sont affichés sous forme de complément à deux. Par exemple,

−1 s'affiche sous la forme

0hFFFFFFFFFFFFFFFF en mode Base Hex

0b111...111 (64 1's) en mode Base Binaire

-2^{63} s'affiche sous la forme

0h8000000000000000 en mode Base Hex

0b100...000 (63 zéros) en mode Base Binaire

256▸Base2	0b100000000
0h1F▸Base2	0b11111

▸Base2 Catalogue >

Si vous entrez un nombre dont le codage binaire signé est hors de la plage des 64 bits, il est ramené à l'aide d'une congruence dans la plage appropriée. Consultez les exemples suivants de valeurs hors plage.

2^{63} devient -2^{63} et s'affiche sous la forme

0h8000000000000000 en mode Base Hex

0b100...000 (63 zéros) en mode Base Binaire

2^{64} devient 0 et s'affiche sous la forme

0h0 en mode Base Hex

0b0 en mode Base Binaire

$-2^{63} - 1$ devient $2^{63} - 1$ et s'affiche sous la forme

0h7FFFFFFFFFFFFFFF en mode Base Hex

0b111...111 (64 1) en mode Base Binaire

▸Base10 Catalogue >

Entier1 **▸Base10**⇒*entier*

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **@>Base10**.

Convertit *Entier1* en un nombre décimal (base 10). Toute entrée binaire ou hexadécimale doit avoir respectivement un préfixe 0b ou 0h.

0b *nombreBinaire*

0h *nombreHexadécimal*

0b10011▸Base10	19
0h1F▸Base10	31

▸Base10 Catalogue >

Zéro et pas la lettre O, suivi de b ou h.

Une entrée binaire peut comporter jusqu'à 64 chiffres (sans compter le préfixe 0b) ; une entrée hexadécimale jusqu'à 8 chiffres.

Sans préfixe, *Entier1* est considéré comme décimal. Le résultat est affiché en base décimale, quel que soit le mode Base en cours d'utilisation.

▸Base16 Catalogue >

Entier1 ▸**Base16**⇒*entier*

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **@>Base16**.

Convertit *Entier1* en nombre hexadécimal. Les nombres binaires et les nombres hexadécimaux présentent toujours respectivement un préfixe, 0b ou 0h.

0b *nombreBinaire*

0h *nombreHexadécimal*

Zéro et pas la lettre O, suivi de b ou h.

Une entrée binaire peut comporter jusqu'à 64 chiffres (sans compter le préfixe 0b) ; une entrée hexadécimale jusqu'à 16 chiffres.

Si *Entier1* est entré sans préfixe, il est considéré comme un nombre en écriture décimale (base 10). Le résultat est affiché sous forme hexadécimal, indépendamment du mode Base utilisé.

Si vous entrez un nombre dont le codage binaire signé dépasse 64 bits, il est ramené à l'aide d'une congruence dans la plage appropriée. Pour de plus amples informations, voir ▸**Base2**, page 17.

256▸Base16	0h100
0b111100001111▸Base16	0hF0F

binomCdf() Catalogue >

binomCdf(*n*,*p*)⇒*liste*

binomCdf
(*n*,*p*,*lowBound*,*upBound*)⇒*nombre* si les bornes *lowBound* et *upBound* sont des nombres, *liste* si les bornes *lowBound* et *upBound* sont des listes

binomCdf(*n*,*p*,*upBound*)pour P(0≤X ≤*upBound*)⇒*nombre* si la borne *upBound* est un nombre, *liste* si la borne *upBound* est une liste

Calcule la probabilité cumulée d'une variable suivant une loi binomiale de paramètres *n* = nombre d'essais et *p* = probabilité de réussite à chaque essai.

Pour P(X ≤ *upBound*), définissez la borne *lowBound*=0

binomPdf() Catalogue >

binomPdf(*n*,*p*)⇒*liste*

binomPdf(*n*,*p*,*ValX*)⇒*nombre* si *ValX* est un nombre, *liste* si *ValX* est une liste

Calcule la probabilité de *ValX* pour la loi binomiale discrète avec un nombre *n* d'essais et la probabilité *p* de réussite pour chaque essai.

C

ceiling() Catalogue >

Donne le plus petit entier ≥ à l'argument.

L'argument peut être un nombre réel ou un nombre complexe.

Remarque : Voir aussi **floor()**.

ceiling(*Liste1*)⇒*liste*

ceiling(*Matrice1*)⇒*matrice*

$\text{ceiling}(.456) \quad 1.$

$\text{ceiling}(\{-3.1,1,2.5\}) \quad \{-3.,1,3.\}$

$\text{ceiling}\begin{bmatrix} 0 & -3.2\cdot i \\ 1.3 & 4 \end{bmatrix} \quad \begin{bmatrix} 0 & -3.\cdot i \\ 2. & 4 \end{bmatrix}$

ceiling() Catalogue >

Donne la liste ou la matrice de plus petites valeurs supérieures ou égales à chaque élément.

centralDiff() Catalogue >

centralDiff(*Expr1*,*Var* [=*Valeur*] [,*Pas*]**)**⇒*expression*

centralDiff(*Expr1*,*Var* [,*Pas*]**)|***Var*=*Valeur*⇒*expression*

centralDiff(*Expr1*,*Var* [=*Valeur*] [,*Liste*]**)**⇒*liste*

centralDiff(*Liste1*,*Var* [=*Valeur*] [,*Incrément*]**)**⇒*liste*

centralDiff(*Matrice1*,*Var* [=*Valeur*] [,*Incrément*]**)**⇒*matrice*

Affiche la dérivée numérique en utilisant la formule du quotient à différence centrée.

Quand la *valeur* est spécifiée, celle-ci prévaut sur toute affectation de variable ou substitution précédente de type « | » pour la variable.

Incrément correspond à la valeur de l'incrément. Si *Incrément* n'est pas spécifié, il est fixé par défaut à 0,001.

Si vous utilisez *Liste1* ou *Matrice1*, l'opération s'étend aux valeurs de la liste ou aux éléments de la matrice.

Remarque : voir aussi **avgRC()**.

char() Catalogue >

char(*Entier***)**⇒*caractère*

Donne le caractère dont le code dans le jeu de caractères de l'unité nomade est *Entier*. La plage valide pour *Entier* est comprise entre 0 et 65535.

char(38)	"&"
char(65)	"A"

χ²2way

Catalogue >

χ²2way *MatriceObservée*

chi22way *MatriceObservée*

Effectue un test χ^2 d'association sur le tableau 2*2 de valeurs dans la matrice observée *MatriceObservée*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Pour plus d'informations concernant les éléments vides dans une matrice, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.χ^2	Stats Khi^2 : sum(observée - attendue)^2/attendue
stat.PVal	Plus petit seuil de signification permettant de rejeter l'hypothèse nulle
stat.df	Degré de liberté des statistiques khi^2
stat.ExpMat	Matrice du tableau de valeurs élémentaires attendues, acceptant l'hypothèse nulle
stat.CompMat	Matrice des contributions statistiques khi^2 élémentaires

χ²Cdf()

Catalogue >

χ²Cdf(*lowBound***,***upBound***,***dl***)**⇒*nombre* si les bornes *lowBound* et *upBound* sont des nombres, *liste* si les bornes *lowBound* et *upBound* sont des listes

chi2Cdf(*lowBound***,***upBound***,***dl***)**⇒*nombre* si les bornes *lowBound* et *upBound* sont des nombres, *liste* si les bornes *lowBound* et *upBound* sont des listes

Calcule la probabilité qu'une variable suivant une loi χ^2 à *dl* degrés de liberté prenne une valeur entre les bornes *lowBound* et *upBound*.

Pour $P(X \le upBound)$, définissez la borne lowBound=0.

χ^2Cdf()

Catalogue >

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

χ^2GOF

Catalogue >

χ^2GOF *ListeObservée*,*ListeAttendue*,*df*

chi2GOF *ListeObservée*,*ListeAttendue*,*df*

Effectue un test pour s'assurer que les données des échantillons sont issues d'une population conforme à la loi spécifiée. *ListeObservée* est une liste de comptage qui doit contenir des entiers. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.χ^2	Stats Khi^2 : sum(observée - attendue)^2/attendue
stat.PVal	Plus petit seuil de signification permettant de rejeter l'hypothèse nulle
stat.df	Degré de liberté des statistiques khi^2
stat.CompList	Contributions statistiques khi^2 élémentaires

χ^2Pdf()

Catalogue >

χ^2Pdf(*ValX*,*dl***)**⇒*nombre* si *ValX* est un nombre, *liste* si *XVal* est une liste

chi2Pdf(*ValX*,*dl***)**⇒*nombre* si *ValX* est un nombre, *liste* si *ValX* est une liste

Calcule la probabilité qu'une variable suivant une loi χ^2 à *dl* degrés de liberté prenne une valeur *ValX* spécifiée.

χ^2Pdf()

Catalogue >

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

ClearAZ

Catalogue >

ClearAZ

Supprime toutes les variables à une lettre de l'activité courante.

Si une ou plusieurs variables sont verrouillées, cette commande affiche un message d'erreur et ne supprime que les variables non verrouillées. Voir **unLock, page 174.**

ClrErr

Catalogue >

ClrErr

Efface le statut d'erreur et règle la variable système *errCode* sur zéro.

L'instruction **Else** du bloc **Try...Else...EndTry** doit utiliser **EffErr** ou **PassErr**. Si vous comptez rectifier ou ignorer l'erreur, sélectionnez **EffErr**. Si vous ne savez pas comment traiter l'erreur, sélectionnez **PassErr** pour la transférer au traitement d'erreurs suivant. S'il n'y a plus d'autre traitement d'erreurs **Try...Else...EndTry**, la boîte de dialogue Erreur s'affiche normalement.

Remarque : voir également **PassErr**, page 118 et **Try**, page 167.

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

Pour obtenir un exemple de **ClrErr**, reportez-vous à l'exemple 2 de la commande **Try**, page 167.

colAugment() Catalogue >

colAugment(*Matrice1***, *Matrice2*)⇒*matrice***

Donne une nouvelle matrice obtenue en ajoutant les lignes/colonnes de la *Matrice2* à celles de la *Matrice1*. Les matrices doivent avoir le même nombre de colonnes et *Matrice2* est ajoutée à *Matrice1* via la création de nouvelles lignes. *Matrice1* et *Matrice2* ne sont pas modifiées.

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\rightarrow m1 \qquad \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$$

$$\begin{bmatrix}5 & 6\end{bmatrix}\rightarrow m2 \qquad \begin{bmatrix}5 & 6\end{bmatrix}$$

$$\text{colAugment}(m1,m2) \qquad \begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}$$

colDim() Catalogue >

colDim(*Matrice***)⇒*expression***

Donne le nombre de colonnes de la matrice *Matrice*.

Remarque : voir aussi **rowDim()**.

$$\text{colDim}\left(\begin{bmatrix}0 & 1 & 2\\3 & 4 & 5\end{bmatrix}\right) \qquad 3$$

colNorm() Catalogue >

colNorm(*Matrice***)⇒*expression***

Donne le maximum des sommes des valeurs absolues des éléments situés dans chaque colonne de la matrice *Matrice*.

Remarque : les éléments non définis de matrice ne sont pas autorisés. Voir aussi **rowNorm()**.

$$\begin{bmatrix}1 & -2 & 3\\4 & 5 & -6\end{bmatrix}\rightarrow mat \qquad \begin{bmatrix}1 & -2 & 3\\4 & 5 & -6\end{bmatrix}$$

$$\text{colNorm}(mat) \qquad 9$$

conj() Catalogue >

conj(*Liste1***)⇒*liste***

conj(*Matrice1***)⇒*matrice***

Donne le conjugué de l'argument.

Remarque : toutes les variables non affectées sont considérées comme réelles.

constructMat() Catalogue >

constructMat
(
Expr
,
Var1
,
Var2
,*nbreLignes*,*nbreColonnes***)**⇒*matrice*

Donne une matrice basée sur les arguments.

Expr est une expression composée de variables *Var1* et *Var2*. Les éléments de la matrice résultante sont formés en évaluant *Expr* pour chaque valeur incrémentée de *Var1* et de *Var2*.

Var1 est incrémentée automatiquement de **1** à *nbreLignes*. Dans chaque ligne, *Var2* est incrémentée de **1** à *nbreColonnes*.

$\text{constructMat}\left(\frac{1}{i+j},i,j,3,4\right)$

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

CopyVar Catalogue >

CopyVar *Var1*, *Var2*

CopyVar *Var1*., *Var2*.

CopyVar *Var1*, *Var2* copie la valeur de la variable *Var1* dans la variable *Var2* et crée *Var2*, si nécessaire. La variable *Var1* doit avoir une valeur.

Si *Var1* correspond au nom d'une fonction existante définie par l'utilisateur, copie la définition de cette fonction dans la fonction *Var2*. La fonction *Var1* doit être définie.

Var1 doit être conforme aux règles de dénomination des variables ou correspondre à une expression d'indirection correspondant à un nom de variable conforme à ces règles.

Define $a(x)=\frac{1}{x}$	*Done*
Define $b(x)=x^2$	*Done*
CopyVar a,c: $c(4)$	$\frac{1}{4}$
CopyVar b,c: $c(4)$	16

CopyVar

Catalogue >

CopyVar *Var1.*, *Var2*. copie tous les membres du groupe de variables *Var1*. dans le groupe *Var2* et crée le groupe *Var2*. si nécessaire.

Var1. doit être le nom d'un groupe de variables existant, comme *stat*,*le résultat nn* ou les variables créées à l'aide de la fonction **LibShortcut()**. Si *Var2*. existe déjà, cette commande remplace tous les membres communs aux deux groupes et ajoute ceux qui n'existent pas. Si un ou plusieurs membres de *Var2*. sont verrouillés, tous les membres de *Var2*. restent inchangés.

$aa.a := 45$ → 45

$aa.b := 6.78$ → 6.78

CopyVar *aa.*,*bb.* → *Done*

getVarInfo() →

aa.a	"NUM"	"□"	0
aa.b	"NUM"	"□"	0
bb.a	"NUM"	"□"	0
bb.b	"NUM"	"□"	0

corrMat()

Catalogue >

corrMat(*Liste1***,***Liste2***[,…[,***Liste20***]])**

Calcule la matrice de corrélation de la matrice augmentée [*Liste1 Liste2 … List20*].

cos()

Touche trig

cos(*Liste1***)⇒***liste*

cos(*Liste1***)** donne la liste des cosinus des éléments de *Liste1*.

Remarque : l'argument est interprété comme la mesure d'un angle en degrés, en grades ou en radians, suivant le mode angulaire en cours d'utilisation. Vous pouvez utiliser °, G ou r pour préciser l'unité employée temporairement pour le calcul.

En mode Angle en degrés :

En mode Angle en grades :

En mode Angle en radians :

cos(*matriceCarrée1***)⇒***matriceCarrée*

Calcule le cosinus de la matrice *matriceCarrée1*. Ce calcul est différent du calcul du cosinus de chaque élément.

Si une fonction scalaire f(A) opère sur *matriceCarrée1* (A), le résultat est calculé par l'algorithme suivant :

En mode Angle en radians :

$$\cos\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix} \rightarrow \begin{bmatrix}0.212493 & 0.205064 & 0.121389\\ 0.160871 & 0.259042 & 0.037126\\ 0.248079 & -0.090153 & 0.218972\end{bmatrix}$$

cos()

Touche trig

Calcul des valeurs propres (λi) et des vecteurs propres (Vi) de A.

matriceCarrée1 doit être diagonalisable et ne peut pas présenter de variables symboliques sans valeur affectée.

Formation des matrices :

$$B = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & \lambda_n \end{bmatrix} \text{ and } X = [V_1, V_2, \dots, V_n]$$

Alors $A = X\ B\ X^{-1}$ et $f(A) = X\ f(B)\ X^{-1}$. Par exemple, $\cos(A) = X\ \cos(B)\ X^{-1}$ où :

cos (B) =

$$\begin{bmatrix} \cos(\lambda_1) & 0 & \dots & 0 \\ 0 & \cos(\lambda_2) & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & \cos(\lambda_n) \end{bmatrix}$$

Tous les calculs sont exécutés en virgule flottante.

cos⁻¹()

Touche trig

cos⁻¹(*Liste1***)⇒***liste*

cos⁻¹(*Liste1***)** donne la liste des arcs cosinus de chaque élément de *Liste1*.

Remarque : donne le résultat en degrés, en grades ou en radians, suivant le mode angulaire utilisé.

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **`arccos(...)`** .

cos⁻¹(*matriceCarrée1***)⇒***matriceCarrée*

Donne l'arc cosinus de *matriceCarrée1*. Ce calcul est différent du calcul de l'arc cosinus de chaque élément. Pour plus d'informations sur la méthode de calcul, reportez-vous à **cos()**.

En mode Angle en degrés :

En mode Angle en grades :

En mode Angle en radians :

En mode Angle en radians et en mode Format complexe Rectangulaire :

$\cos^{-1}()$ — Touche trig

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

$$\cos^{-1}\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix}$$

$$\begin{bmatrix}1.73485+0.064606\cdot i & -1.49086+2.10514\\ -0.725533+1.51594\cdot i & 0.623491+0.77836\\ -2.08316+2.63205\cdot i & 1.79018-1.27182\cdot\end{bmatrix}$$

Pour afficher le résultat entier, appuyez sur ▲, puis utilisez les touches ◄ et ► pour déplacer le curseur.

cosh() — Catalogue >

cosh(*Liste1***)**⇒*liste*

cosh(*Liste1*) donne la liste des cosinus hyperboliques de chaque élément de *Liste1*.

cosh(*matriceCarrée1***)**⇒*matriceCarrée*

Donne le cosinus hyperbolique de la matrice *matriceCarrée1*. Ce calcul est différent du calcul du cosinus hyperbolique de chaque élément. Pour plus d'informations sur la méthode de calcul, reportez-vous à **cos()**.

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

En mode Angle en degrés :

En mode Angle en radians :

$$\cosh\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix}$$

$$\begin{bmatrix}421.255 & 253.909 & 216.905\\ 327.635 & 255.301 & 202.958\\ 226.297 & 216.623 & 167.628\end{bmatrix}$$

$\cosh^{-1}()$ — Catalogue >

cosh^{-1}(*List1***)**⇒*liste*

cosh^{-1}(*Liste1*) donne la liste des arguments cosinus hyperboliques de chaque élément de *Liste1*.

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **`arccosh(...)`**.

cosh^{-1} (*matriceCarrée1***)**⇒*matriceCarrée*

En mode Angle en radians et en mode Format complexe Rectangulaire :

cosh⁻¹() Catalogue >

Donne l'argument cosinus hyperbolique de la matrice *matriceCarrée1*. Ce calcul est différent du calcul de l'argument cosinus hyperbolique de chaque élément. Pour plus d'informations sur la méthode de calcul, reportez-vous à **cos()**.

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

$$\cosh^{-1}\left(\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}\right)$$

$$\begin{bmatrix} 2.52503+1.73485\cdot i & -0.009241-1.4908 \\ 0.486969-0.725533\cdot i & 1.66262+0.623491 \\ -0.322354-2.08316\cdot i & 1.26707+1.79018 \end{bmatrix}$$

Pour afficher le résultat entier, appuyez sur ▲, puis utilisez les touches ◄ et ► pour déplacer le curseur.

cot() Touche trig

cot(*Liste1***)** ⇒ *liste*

Remarque : l'argument est interprété comme la mesure d'un angle en degrés, en grades ou en radians, suivant le mode angulaire en cours d'utilisation. Vous pouvez utiliser °, G ou ʳ pour préciser l'unité employée temporairement pour le calcul.

En mode Angle en degrés :

En mode Angle en grades :

En mode Angle en radians :

cot⁻¹() Touche trig

cot⁻¹(*Liste1***)**⇒*liste*

Remarque : donne le résultat en degrés, en grades ou en radians, suivant le mode angulaire utilisé.

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **`arccot(`**...**`)`**.

En mode Angle en degrés :

$\cot^{-1}(1)$ 45.

En mode Angle en grades :

$\cot^{-1}(1)$ 50.

En mode Angle en radians :

coth() Catalogue >

coth(*Liste1***)**⇒*liste*

coth⁻¹() Catalogue >

coth⁻¹(*Liste1***)**⇒*liste*

coth⁻¹() Catalogue >

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **arccoth(...)**.

count() Catalogue >

count(*Valeur1ouListe1* [,*Valeur2ouListe2* [,...]]**)⇒***valeur*

Affiche le nombre total des éléments dans les arguments qui s'évaluent à des valeurs numériques.

Un argument peut être une expression, une valeur, une liste ou une matrice. Vous pouvez mélanger les types de données et utiliser des arguments de dimensions différentes.

Pour une liste, une matrice ou une plage de cellules, chaque élément est évalué afin de déterminer s'il doit être inclus dans le comptage.

Dans l'application Tableur & listes, vous pouvez utiliser une plage de cellules à la place de n'importe quel argument.

Les éléments vides sont ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

countif() Catalogue >

countif(*Liste*,*Critère***)⇒***valeur*

Affiche le nombre total d'éléments dans *Liste* qui répondent au *critère* spécifié.

Le critère peut être :

- Une valeur, une expression ou une chaîne. Par exemple, **3** compte uniquement les éléments dans *Liste* qui ont pour valeur 3.
- Une expression booléenne contenant le symbole **?** comme paramètre substituable à tout élément. Par exemple, **?<5** ne compte que les

countIf({1,3,"abc",undef,3,1},3) 2

Compte le nombre d'éléments égaux à 3.

countIf({"abc","def","abc",3},"def") 1

Compte le nombre d'éléments égaux à “def.”

countIf({1,3,5,7,9},?<5) 2

Compte 1 et 3.

countif() Catalogue >

éléments dans *Liste* qui sont inférieurs à 5.

Dans l'application Tableur & listes, vous pouvez utiliser une plage de cellules à la place de *Liste*.

Les éléments vides de la liste sont ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

Remarque : voir également **sumIf()**, page 160 et **frequency()**, page 59.

countIf({1,3,5,7,9},2<?<8) 3

Compte 3, 5 et 7.

countIf({1,3,5,7,9},?<4 or ?>6) 4

Compte 1, 3, 7 et 9.

cPolyRoots() Catalogue >

cPolyRoots(*Poly*,*Var***)**⇒*liste*

cPolyRoots(*ListeCoeff***)**⇒*liste*

La première syntaxe, **cPolyRoots(***Poly*,*Var***)**, affiche une liste de racines complexes du polynôme *Poly* pour la variable *Var*.

La deuxième syntaxe, **cPolyRoots (***ListeCoeff***)**, affiche une liste des racines complexes pour les coefficients de la liste *ListeCoeff*.

Remarque : voir aussi **polyRoots()**, page 120.

crossP() Catalogue >

crossP(*Liste1*, *Liste2***)**⇒*liste*

Donne le produit vectoriel de *Liste1* et de *Liste2* et l'affiche sous forme de liste.

Liste1 et *Liste2* doivent être de même dimension et cette dimension doit être égale à 2 ou 3.

crossP(*Vecteur1*, *Vecteur2***)**⇒*vecteur*

Donne le vecteur ligne ou le vecteur colonne (en fonction des arguments) obtenu en calculant le produit vectoriel de *Vecteur1* et *Vecteur2*.

crossP([1 2 3],[4 5 6]) [-3 6 -3]

crossP([1 2],[3 4]) [0 0 -2]

crossP() — Catalogue >

Ces deux vecteurs, *Vecteur1* et *Vecteur2*, doivent être de même type (ligne ou colonne) et de même dimension, cette dimension devant être égale à 2 ou 3.

csc() — Touche trig

csc(*Liste1***)**⇒*liste*

En mode Angle en degrés :

En mode Angle en grades :

En mode Angle en radians :

csc^{-1}() — Touche trig

csc^{-1}(*Liste1***)** ⇒ *liste*

Remarque : donne le résultat en degrés, en grades ou en radians, suivant le mode angulaire utilisé.

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **`arccsc(...)`**.

En mode Angle en degrés :

$\csc^{-1}(1)$ 90.

En mode Angle en grades :

$\csc^{-1}(1)$ 100.

En mode Angle en radians :

csch() — Catalogue >

csch(*Liste1***)** ⇒ *liste*

csch^{-1}() — Catalogue >

csch^{-1}(*Liste1***)** ⇒ *liste*

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **`arccsch(...)`**.

$\csch^{-1}(1)$ $\sinh^{-1}(1)$

$\csch^{-1}(\{1,2.1,3\})$ $\left\{\sinh^{-1}(1),0.459815,\sinh^{-1}\left(\frac{1}{3}\right)\right\}$

CubicReg *X*, *Y*[, [*Fréq*] [, *Catégorie*, *Inclure*]]

Effectue l'ajustement polynomial de degré $3y = a \cdot x^3+b \cdot x^2+c \cdot x+d$ sur les listes *X* et *Y* en utilisant la fréquence *Fréq*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes, à l'exception de *Inclure*.

X et *Y* sont des listes de variables indépendantes et dépendantes.

Fréq est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *Fréq* correspond à une fréquence d'occurrence pour chaque couple *X* et *Y*. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Catégorie est une liste de codes de catégories pour les couples *X* et *Y* correspondants.

Inclure est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	**Description**
stat.RegEqn	Équation d'ajustement : $a \cdot x^3+b \cdot x^2+c \cdot x+d$
stat.a, stat.b, stat.c, stat.d	Coefficients d'ajustement
stat.R^2	Coefficient de détermination
stat.Resid	Valeurs résiduelles de l'ajustement
stat.XReg	Liste des points de données de la liste *Liste X* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*

Variable de sortie	Description
stat.YReg	Liste des points de données de la liste *Liste Y* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.FreqReg	Liste des fréquences correspondant à *stat.XReg* et *stat.YReg*

cumulativeSum() Catalogue >

cumulativeSum(*Liste1***)⇒***liste***

Donne la liste des sommes cumulées des éléments de *Liste1*, en commençant par le premier élément (élément 1).

cumulativeSum({1,2,3,4}) {1,3,6,10}

cumulativeSum(*Matrice1***)⇒***matrice***

Donne la matrice des sommes cumulées des éléments de *Matrice1*. Chaque élément correspond à la somme cumulée de tous les éléments situés au-dessus, dans la colonne correspondante.

Un élément vide de *Liste1* ou *Matrice1* génère un élement vide dans la liste ou la matrice résultante. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223

$$\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix} \rightarrow m1 \qquad \begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}$$

$$\text{cumulativeSum}(m1) \qquad \begin{bmatrix}1 & 2\\ 4 & 6\\ 9 & 12\end{bmatrix}$$

Cycle Catalogue >

Cycle

Procède au passage immédiat à l'itération suivante de la boucle courante (**For**, **While** ou **Loop**).

La fonction Cycle ne peut pas s'utiliser indépendamment de l'une des trois structures de boucle (**For**, **While** ou **Loop**).

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

Liste de fonctions qui additionne les entiers compris entre 1 et 100, en sautant 50.

```
Define g()=Func                    Done
    Local temp,i
    0→temp
    For i,1,100,1
    If i=50
    Cycle
    temp+i→temp
    EndFor
    Return temp
    EndFunc
g()                                5000
```

▸Cylind

Catalogue >

Vecteur **▸Cylind**

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **@>Cylind**.

Affiche le vecteur ligne ou colonne en coordonnées cylindriques [r,∠θ, z].

Vecteur doit être un vecteur à trois éléments. Il peut s'agir d'un vecteur ligne ou colonne.

D

dbd()

Catalogue >

dbd(*date1,date2***)⇒***valeur*

Calcule le nombre de jours entre *date1* et *date2* à l'aide de la méthode de calcul des jours.

date1 et *date2* peuvent être des chiffres ou des listes de chiffres compris dans une plage de dates d'un calendrier normal. Si *date1* et *date2* sont toutes deux des listes, elles doivent être de la même longueur.

date1 et *date2* doivent être comprises entre 1950 et 2049.

Vous pouvez saisir les dates à l'un des deux formats. L'emplacement de la décimale permet de distinguer les deux formats.

MM.JJAA (format communément utilisé aux Etats-Unis)

JJMM.AA (format communément utilisé en Europe)

dbd(12.3103,1.0104)	1
dbd(1.0107,6.0107)	151
dbd(3112.03,101.04)	1
dbd(101.07,106.07)	151

▸DD

Catalogue >

Valeur **▸DD⇒***valeur*

Liste1 **▸DD⇒***liste*

En mode Angle en degrés :

▸DD

Catalogue >

Matrice1 **▸DD⇒***matrice*

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **@>DD**.

Donne l'équivalent décimal de l'argument exprimé en degrés. L'argument est un nombre, une liste ou une matrice interprété suivant le mode Angle utilisé (grades, radians ou degrés).

(1.5°)▸DD	1.5°
(45°22'14.3")▸DD	45.3706°
({45°22'14.3",60°0'0"})▸DD	{45.3706°,60°}

En mode Angle en grades :

1▸DD	$\frac{9}{10}$°

En mode Angle en radians :

(1.5)▸DD	85.9437°

▸Decimal

Catalogue >

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **@>Decimal**.

Affiche l'argument sous forme décimale. Cet opérateur ne peut être utilisé qu'à la fin d'une ligne.

$\frac{1}{3}$▸Decimal	0.333333

Define

Catalogue >

Define *Var* **=** *Expression*

Define *Fonction*(*Param1*, *Param2*, ...) **=** *Expression*

Définit la variable *Var* ou la fonction définie par l'utilisateur *Fonction*.

Les paramètres, tels que *Param1*, sont des paramètres substituables utilisés pour transmettre les arguments à la fonction. Lors de l'appel d'une fonction définie par l'utilisateur, des arguments (par exemple, les valeurs ou variables) qui correspondent aux paramètres doivent être fournis. La fonction évalue ensuite *Expression* en utilisant les arguments fournis.

Define $g(x,y)=2\cdot x-3\cdot y$	*Done*
$g(1,2)$	-4
$1\rightarrow a$: $2\rightarrow b$: $g(a,b)$	-4
Define $h(x)=$when$(x<2,2\cdot x-3,-2\cdot x+3)$	*Done*
$h(-3)$	-9
$h(4)$	-5

Define

Catalogue >

Var et *Fonction* ne peuvent pas être le nom d'une variable système ni celui d'une fonction ou d'une commande prédéfinie.

Remarque : cette utilisation de **Define** est équivalente à celle de l'instruction : *expression* ➞ *Fonction* (*Param1*,*Param2*).

Define *Fonction*(*Param1*, *Param2*, ...) = **Func**
Bloc
EndFunc

Define *Programme*(*Param1*, *Param2*, ...) = **Prgm**
Bloc
EndPrgm

Dans ce cas, la fonction définie par l'utilisateur ou le programme permet d'exécuter plusieurs instructions (bloc).

Bloc peut correspondre à une instruction unique ou à une série d'instructions réparties sur plusieurs lignes. *Bloc* peut également contenir des expressions et des instructions (comme **If**, **Then**, **Else** et **For**).

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

Remarque : voir aussi **Define LibPriv**, page 38 et **Define LibPub**, page 39.

```
Define g(x,y)=Func                    Done
                If x>y Then
                Return x
                Else
                Return y
                EndIf
                EndFunc
g(3,-7)                                  3
```

```
Define g(x,y)=Prgm
                If x>y Then
                Disp x," greater than ",y
                Else
                Disp x," not greater than ",y
                EndIf
                EndPrgm
                                      Done
g(3,-7)
                        3 greater than -7
                                      Done
```

Define LibPriv

Catalogue >

Define LibPriv *Var* = *Expression*

Define LibPriv *Fonction*(*Param1*, *Param2*, ...) = *Expression*

Define LibPriv *Fonction*(*Param1*, *Param2*, ...) = **Func**

Define LibPriv — Catalogue >

Bloc
EndFunc

Define LibPriv *Programme*(*Param1*, *Param2*, ...) **= Prgm**
Bloc
EndPrgm

S'utilise comme **Define**, mais permet de définir des objets (variables, fonctions, programmes) dans la bibliothèque privée. Les fonctions et programmes privés ne s'affichent pas dans le Catalogue.

Remarque : voir aussi **Define**, page 37 et **Define LibPub**, page 39.

Define LibPub — Catalogue >

Define LibPub *Var* **=** *Expression*

Define LibPub *Fonction*(*Param1*, *Param2*, ...) **=** *Expression*

Define LibPub *Fonction*(*Param1*, *Param2*, ...) **= Func**
Bloc
EndFunc

Define LibPub *Programme*(*Param1*, *Param2*, ...) **= Prgm**
Bloc
EndPrgm

S'utilise comme **Define**, mais permet de définir des objets (variables, fonctions, programmes) dans la bibliothèque publique. Les fonctions et programmes publics s'affichent dans le Catalogue après l'enregistrement et le rafraîchissement de la bibliothèque.

Remarque : voir aussi **Define**, page 37 et **Define LibPriv**, page 38.

deltaList() — Voir ΔList(), page 88.

DelVar

Catalogue >

DelVar *Var1*[**,** *Var2*] [**,** *Var3*] ...

DelVar *Var***.**

Supprime de la mémoire la variable ou le groupe de variables spécifié.

Si une ou plusieurs variables sont verrouillées, cette commande affiche un message d'erreur et ne supprime que les variables non verrouillées. Voir **unLock, page 174.**

DelVar *Var*. supprime tous les membres du groupe de variables *Var*, comme les variables statistiques du groupe *stat*,*le résultat nn* ou les variables créées à l'aide de la fonction **LibShortcut()**. Le point (.) dans cette utilisation de la commande **DelVar** limite la suppression au groupe de variables ; la variable simple *Var* n'est pas supprimée.

aa.a:=45	45
aa.b:=5.67	5.67
aa.c:=78.9	78.9
getVarInfo()	*aa.a* "NUM" "□" *aa.b* "NUM" "□" *aa.c* "NUM" "□"
DelVar *aa.*	*Done*
getVarInfo()	"NONE"

delVoid()

Catalogue >

delVoid(*Liste1***)**⇒*liste*

Donne une liste contenant les éléments de *Liste1* sans les éléments vides.

Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

delVoid({1,void,3})	{1,3}

det()

Catalogue >

det(*matriceCarrée*[**,** *Tolérance*]**)**⇒*expression*

Donne le déterminant de *matriceCarrée*.

det() Catalogue >

L'argument facultatif Tolérance permet de considérer comme nul tout élément de la matrice dont la valeur absolue est inférieure à *Tolérance*. Cet argument n'est utilisé que si la matrice contient des nombres en virgule flottante et ne contient pas de variables symboliques sans valeur affectée. Dans le cas contraire, *Tolérance* est ignoré.

- Si vous utilisez ctrl enter ou définissez le mode **Auto ou Approché** sur Approché, les calculs sont effectués en virgule flottante.
- Si *Tolérance* est omis ou inutilisé, la tolérance par défaut est calculée comme suit :

  5E-14 ·**max(dim(***matriceCarrée***)) ·rowNorm(***matriceCarrée***)**

diag() Catalogue >

diag(*Liste***)⇒***matrice*

diag(*matriceLigne***)⇒***matrice*

diag(*matriceColonne***)⇒***matrice*

Donne une matrice diagonale, ayant sur sa diagonale principale les éléments de la liste passée en argument.

$\text{diag}([2 \quad 4 \quad 6])$ → $\begin{bmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 6 \end{bmatrix}$

diag(*matriceCarrée***)⇒***matriceLigne*

Donne une matrice ligne contenant les éléments de la diagonale principale de *matriceCarrée*.

matriceCarrée doit être une matrice carrée.

$\begin{bmatrix} 4 & 6 & 8 \\ 1 & 2 & 3 \\ 5 & 7 & 9 \end{bmatrix}$ → $\begin{bmatrix} 4 & 6 & 8 \\ 1 & 2 & 3 \\ 5 & 7 & 9 \end{bmatrix}$

$\text{diag}(Ans)$ → $[4 \quad 2 \quad 9]$

dim() Catalogue >

dim(*Liste***)⇒***entier*

Donne le nombre d'éléments de *Liste*.

$\dim(\{0,1,2\})$ → 3

dim() Catalogue >

dim(*Matrice***)⇒***liste*

Donne les dimensions de la matrice sous la forme d'une liste à deux éléments {lignes, colonnes}.

$\dim\begin{bmatrix}1 & -1\\ 2 & -2\\ 3 & 5\end{bmatrix}$ → $\{3,2\}$

dim(*Chaîne***)⇒***entier*

Donne le nombre de caractères contenus dans *Chaîne*.

dim("Hello") → 5

dim("Hello "&"there") → 11

Disp Catalogue >

Disp *exprOuChaîne1* [**,** *exprOuChaîne2*] ...

Affiche les arguments dans l'historique de *Calculator*. Les arguments apparaissent les uns après les autres, séparés par des espaces fines.

Très utile dans les programmes et fonctions pour l'affichage de calculs intermédiaires.

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

```
Define chars(start,end)=Prgm
  For i,start,end
  Disp i," ",char(i)
  EndFor
  EndPrgm
                                  Done
chars(240,243)
                                240 ð
                                241 ñ
                                242 ò
                                243 ó
                                  Done
```

DispAt *int,expr1* [,*expr2* ...] ...

DispAt vous permet de spécifier la ligne où l'expression ou la chaîne de caractère spécifiée s'affichera à l'écran.

Le numéro de ligne peut être spécifié sous forme d'expression.

Veuillez noter que le numéro de ligne n'est pas destiné à l'ensemble de l'écran, mais uniquement à la zone suivant immédiatement la commande/le programme.

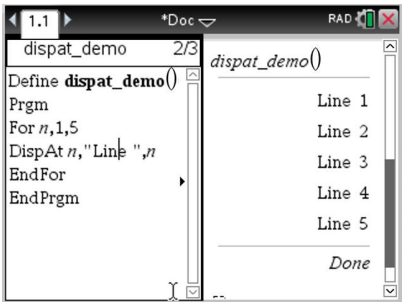
Cette commande permet des sorties de type tableau de bord de programmes où la valeur d'une expression ou d'une lecture de capteur est mise à jour sur la même ligne.

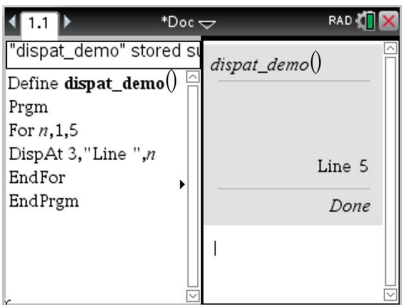
DispAtet Disp peuvent être utilisés au sein du même programme.

Remarque : Le nombre maximum est défini sur 8, du fait que cela correspond à un écran entier de lignes sur l'écran d'une calculatrice - du moment que les lignes ne contiennent pas d'expressions mathématiques 2D. Le nombre exact de lignes dépend du contenu des informations affichées.

DispAt

Par exemple :





Exemples illustratifs :

Define z()=	Output
Prgm	z()
For n,1,3	Itération 1 :
DispAt 1,"N : ",n	Ligne 1 : N :1
Disp "Bonjour"	Ligne 2 : Bonjour
EndFor	
EndPrgm	Itération 2 :
	Ligne 1 : N :2
	Ligne 2 : Bonjour
	Ligne 3 : Bonjour
	Itération 3 :
	Ligne 1 : N :3

	Ligne 2 : Bonjour Ligne 3 : Bonjour Ligne 4 : Bonjour
Define z1()= Prgm For n,1,3 DispAt 1,"N : ",n EndFor For n,1,4 Disp "Bonjour" EndFor EndPrgm	z1() Ligne 1 : N :3 Ligne 2 : Bonjour Ligne 3 : Bonjour Ligne 4 : Bonjour Ligne 5 : Bonjour

Conditions d'erreur :

Message d'erreur	**Description**
Le numéro de ligne DispAt doit être compris entre 1 et 8	L'expression évalue le numéro de la ligne en dehors de la plage 1 - 8 (inclus)
Nombre insuffisant d'arguments	Il manque un ou plusieurs arguments à la fonction ou commande.
Aucun argument	Identique à la boîte de dialogue « erreur de syntaxe » actuelle
Trop d'arguments	Limiter les arguments. Même erreur que Disp.
Type de données incorrect	Le premier argument doit être un nombre.
Vide : DispAt vide	L'erreur de type de données "Hello World" (Datatype error) est renvoyée pour le vide (si le rappel est défini)

▸DMS **Catalogue >**

Liste **▸DMS**

Matrice **▸DMS**

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **@>DMS**.

En mode Angle en degrés :

$(45.371)\blacktriangleright\text{DMS}$ $45°22'15.6"$

$(\{45.371,60\})\blacktriangleright\text{DMS}$ $\{45°22'15.6",60°\}$

▶DMS

Catalogue >

Interprète l'argument comme un angle et affiche le nombre DMS équivalent (DDDDDD°MM'SS.ss''). Voir °, ', ''page 201 pour le détail du format DMS (degrés, minutes, secondes).

Remarque : ▶DMS convertit les radians en degrés lorsque l'instruction est utilisée en mode radians. Si l'entrée est suivie du symbole des degrés °, aucune conversion n'est effectuée. Vous ne pouvez utiliser **▶DMS** qu'à la fin d'une ligne.

dotP()

Catalogue >

dotP(*Liste1***,** *Liste2***)⇒***expression*

Donne le produit scalaire de deux listes.

dotP(*Vecteur1***,** *Vecteur2***)⇒***expression*

Donne le produit scalaire de deux vecteurs.

Les deux vecteurs doivent être de même type (ligne ou colonne).

E

e^()

Touche [eˣ]

Remarque : voir aussi **Modèle e Exposant**, page 2.

Remarque : une pression sur [eˣ] pour afficher e^(est différente d'une pression sur le caractère [E] du clavier.

Vous pouvez entrer un nombre complexe sous la forme polaire $re^{i\theta}$. N'utilisez toutefois cette forme qu'en mode Angle en radians ; elle provoque une erreur de domaine en mode Angle en degrés ou en grades.

e^(*Liste1***)⇒***liste*

Donne une liste constituée des exponentielles des éléments de *Liste1*.

e^() — Touche e^x

e^(*matriceCarrée1***)⇒***matriceCarrée***

Donne l'exponentielle de *matriceCarrée1*. Le résultat est différent de la matrice obtenue en prenant l'exponentielle de chaque élément. Pour plus d'informations sur la méthode de calcul, reportez-vous à **cos()**.

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

$$e^{\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix}} \quad \begin{bmatrix}782.209 & 559.617 & 456.509\\ 680.546 & 488.795 & 396.521\\ 524.929 & 371.222 & 307.879\end{bmatrix}$$

eff() — Catalogue >

eff(*tauxNominal*,*CpY***)⇒***valeur*

Fonction financière permettant de convertir un taux d'intérêt nominal *tauxNominal* en un taux annuel effectif, *CpY* étant le nombre de périodes de calcul par an.

tauxNominal doit être un nombre réel et *CpY* doit être un nombre réel > 0.

Remarque : voir également **nom()**, page 109.

$$\text{eff}(5.75,12) \qquad 5.90398$$

eigVc() — Catalogue >

eigVc(*matriceCarrée***)⇒***matrice*

Donne une matrice contenant les vecteurs propres d'une *matriceCarrée* réelle ou complexe, chaque colonne du résultat correspond à une valeur propre. Notez qu'il n'y a pas unicité des vecteurs propres. Ils peuvent être multipliés par n'importe quel facteur constant. Les vecteurs propres sont normés, ce qui signifie que si V = [x_1, x_2, …, x_n], alors :

$x_1^2 + x_2^2 + \ldots + x_n^2 = 1$

En mode Format complexe Rectangulaire :

$$\begin{bmatrix}-1 & 2 & 5\\ 3 & -6 & 9\\ 2 & -5 & 7\end{bmatrix} \rightarrow m1 \qquad \begin{bmatrix}-1 & 2 & 5\\ 3 & -6 & 9\\ 2 & -5 & 7\end{bmatrix}$$

$$\text{eigVc}(m1)$$

$$\begin{bmatrix}-0.800906 & 0.767947 & (\\ 0.484029 & 0.573804+0.052258\cdot i & 0.5738\\ 0.352512 & 0.262687+0.096286\cdot i & 0.2626\end{bmatrix}$$

Pour afficher le résultat entier, appuyez sur ▲, puis utilisez les touches ◄ et ► pour déplacer le curseur.

eigVc() Catalogue >

matriceCarrée est d'abord transformée en une matrice semblable dont la norme par rapport aux lignes soit le plus proche de celle par rapport aux colonnes. La *matriceCarrée* est ensuite réduite à la forme de Hessenberg supérieure et les vecteurs propres calculés via une factorisation de Schur.

eigVl() Catalogue >

eigVl(*matriceCarrée***)⇒***liste*

Donne la liste des valeurs propres d'une *matriceCarrée* réelle ou complexe.

matriceCarrée est d'abord transformée en une matrice semblable dont la norme par rapport aux lignes soit le plus proche de celle par rapport aux colonnes. La *matriceCarrée* est ensuite réduite à la forme de Hessenberg supérieure et les valeurs propres calculées à partir de la matrice de Hessenberg supérieure.

En mode Format complexe Rectangulaire :

$$\begin{bmatrix} -1 & 2 & 5 \\ 3 & -6 & 9 \\ 2 & -5 & 7 \end{bmatrix} \rightarrow m1 \qquad \begin{bmatrix} -1 & 2 & 5 \\ 3 & -6 & 9 \\ 2 & -5 & 7 \end{bmatrix}$$

$$\text{eigVl}(m1)$$

$$\{-4.40941, 2.20471+0.763006\cdot i, 2.20471-0.$$

Pour afficher le résultat entier, appuyez sur ▲, puis utilisez les touches ◄ et ► pour déplacer le curseur.

Else Voir If, page 73.

ElseIf Catalogue >

If *Expr booléenne1* **Then**
Bloc1
ElseIf *Expr booléenne2* **Then**
Bloc2
⋮
ElseIf *Expr booléenneN* **Then**
BlocN
EndIf
⋮

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

```
Define g(x)=Func
              If x≤-5 Then
              Return 5
              ElseIf x>-5 and x<0 Then
              Return -x
              ElseIf x≥0 and x≠10 Then
              Return x
              ElseIf x=10 Then
              Return 3
              EndIf
              EndFunc
                                    Done
```

EndFor **Voir For, page 56.**

EndFunc **Voir Func, page 61.**

EndIf **Voir If, page 73.**

EndLoop **Voir Loop, page 95.**

EndPrgm **Voir Prgm, page 122.**

EndTry **Voir Try, page 167.**

EndWhile **Voir While, page 178.**

euler ()

Catalogue >

euler(*Expr*, *Var*, *VarDép*, {*Var0, MaxVar*}, *Var0Dép*, *IncVar* [, *IncEuler*]**)** ⇒*matrice*

euler(*SystèmeExpr*, *Var*, *ListeVarDép*, {*Var0*, **Max***Var*}, *ListeVar0Dép*, *IncVar* [, *IncEuler*]**)** ⇒*matrice*

euler(*ListeExpr*, *Var*, *ListeVarDép*, {*Var0*, *MaxVar*}, *ListeVar0Dép*, *IncVar* [, *IncEuler*]**)** ⇒*matrice*

Utilise la méthode d'Euler pour résoudre le système.

Équation différentielle :

y'=0.001*y*(100-y) et y(0)=10

$$\text{euler}\left(0.001\cdot y\cdot(100-y),t,y,\{0,100\},10,1\right)$$
$$\begin{bmatrix} 0. & 1. & 2. & 3. & 4. \\ 10. & 10.9 & 11.8712 & 12.9174 & 14.042 \end{bmatrix}$$

Pour afficher le résultat entier, appuyez sur ▲, puis utilisez les touches ◄ et ► pour déplacer le curseur.

Système d'équations :

$\frac{d\,depVar}{d\,Var} = Expr(Var, depVar)$

avec *VarDép*(*Var0*)=*Var0Dép* pour l'intervalle [*Var0,MaxVar*]. Retourne une matrice dont la première ligne définit les valeurs de sortie de *Var* et la deuxième ligne la valeur du premier composant de la solution pour les valeurs correspondantes de *Var*, etc.

Expr représente la partie droite qui définit l'équation différentielle.

SystèmeExpr correspond aux côtés droits qui définissent le système des équations différentielles (en fonction de l'ordre des variables dépendantes de la *ListeVarDép*).

ListeExpr est la liste des côtés droits qui définissent le système des équations différentielles (en fonction de l'ordre des variables dépendantes de la *ListeVarDép*).

Var est la variable indépendante.

ListeVarDép est la liste des variables dépendantes.

{*Var0*, *MaxVar*} est une liste à deux éléments qui indique la fonction à intégrer de *Var0* à *MaxVar*.

ListeVar0Dép est la liste des valeurs initiales pour les variables dépendantes.

IncVar est un nombre différent de zéro, défini par **sign(***IncVar***)** = **sign (***MaxVar-Var0***)** et les solutions sont retournées pour *Var0*+i·*IncVar* pour tout i=0,1,2,… de sorte que *Var0*+i·*IncVar* soit dans [*var0,MaxVar*] (il est possible qu'il n'existe pas de solution en *MaxVar*).

$$\begin{cases} y1' = -y1 + 0.1 \cdot y1 \cdot y2 \\ y2' = 3 \cdot y2 - y1 \cdot y2 \end{cases}$$

avec $y1(0)=2$ et $y2(0)=5$

$$\text{euler}\left(\begin{cases} -y1+0.1 \cdot y1 \cdot y2 \\ 3 \cdot y2 - y1 \cdot y2 \end{cases}, t, \{y1, y2\}, \{0, 5\}, \{2, 5\}, 1\right)$$

$$\begin{bmatrix} 0. & 1. & 2. & 3. & 4. & 5. \\ 2. & 1. & 1. & 3. & 27. & 243. \\ 5. & 10. & 30. & 90. & 90. & -2070. \end{bmatrix}$$

IncEuler est un entier positif (valeur par défaut : 1) qui définit le nombre d'incréments dans la méthode d'Euler entre deux valeurs de sortie. La taille d'incrément courante utilisée par la méthode d'Euler est *IncVar/IncEuler*.

eval () Menu hub

eval(*Expr***)** ⇒ *chaîne*

eval() n'est valable que dans TI-Innovator™ Hub l'argument de commande des commandes de programmation **Get**, **GetStr** et **Send**. Le logiciel évalue l'expression *Expr* et remplace l'instruction **eval()** par le résultat sous la forme d'une chaîne de caractères.

L'argument *Expr* doit pouvoir être simplifié en un nombre réel.

Définissez l'élément bleu de la DEL RGB en demi-intensité.

```
lum:=127                                  127
Send "SET COLOR.BLUE eval(lum)"           Done
```

Réinitialisez l'élément bleu sur OFF (ARRÊT).

```
Send "SET COLOR.BLUE OFF"                 Done
```

L'argument de eval() doit pouvoir être simplifié en un nombre réel.

```
Send "SET LED eval("4") TO ON"
                        "Error: Invalid data type"
```

Programmez pour faire apparaître en fondu l'élément rouge

```
Define fadein()=
Prgm
For i,0,255,10
  Send "SET COLOR.RED eval(i)"
    Wait 0.1
  EndFor
  Send "SET COLOR.RED OFF"
EndPrgm
```

Exécutez le programme.

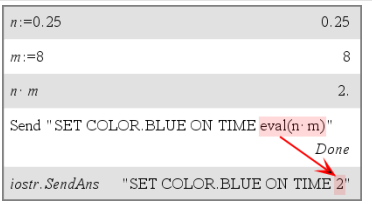
```
fadein()                                  Done
```

eval ()

Menu hub

Même si **eval()** n'affiche pas son résultat, vous pouvez afficher la chaîne de commande Hub qui en découle après avoir exécuté la commande en inspectant l'une des variables spéciales suivantes.

iostr.SendAns
iostr.GetAns
iostr.GetStrAns



Remarque : Voir également **Get** (page 63), **GetStr** (page 70) et **Send** (page 143).

Exit

Catalogue >

Exit

Permet de sortir de la boucle **For**, **While** ou **Loop** courante.

Exit ne peut pas s'utiliser indépendamment de l'une des trois structures de boucle (**For**, **While** ou **Loop**).

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

Liste des fonctions :

```
Define g()=Func                    Done
          Local temp,i
          0→temp
          For i,1,100,1
          temp+i→temp
          If temp>20 Then
          Exit
          EndIf
          EndFor
          EndFunc
g()                                 21
```

exp()

Touche e^x

Remarque : voir aussi Modèle **e** Exposant, page 2.

Vous pouvez entrer un nombre complexe sous la forme polaire $re^{i\theta}$. N'utilisez toutefois cette forme qu'en mode Angle en radians ; elle provoque une erreur de domaine en mode Angle en degrés ou en grades.

exp(*Liste1***)**⇒*liste*

Donne une liste constituée des exponentielles des éléments *Liste1*.

exp()

Touche e^x

exp(*matriceCarrée1***)⇒***matriceCarrée*

Donne l'exponentielle de *matriceCarrée1*. Le résultat est différent de la matrice obtenue en prenant l'exponentielle de chaque élément. Pour plus d'informations sur la méthode de calcul, reportez-vous à **cos()**.

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

$$e^{\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix}} \quad \begin{bmatrix}782.209 & 559.617 & 456.509\\ 680.546 & 488.795 & 396.521\\ 524.929 & 371.222 & 307.879\end{bmatrix}$$

expr()

Catalogue >

expr(*Chaîne***)⇒***expression*

Convertit la chaîne de caractères contenue dans *Chaîne* en une expression. L'expression obtenue est immédiatement évaluée.

ExpReg

Catalogue >

ExpReg X, Y [, [*Fréq*][, *Catégorie, Inclure*]]

Effectue l'ajustement exponentiely = a · (b)ˣsur les listes X et Y en utilisant la fréquence *Fréq*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes, à l'exception de *Inclure*.

X et Y sont des listes de variables indépendantes et dépendantes.

Fréq est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *Fréq* correspond à une fréquence d'occurrence pour chaque couple X et Y. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Catégorie est une liste de codes de catégories pour les couples X et Y correspondants.

Inclure est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.RegEqn	Équation d'ajustement : $a \cdot (b)^x$
stat.a, stat.b	Coefficients d'ajustement
stat.r^2	Coefficient de détermination linéaire pour les données transformées
stat.r	Coefficient de corrélation pour les données transformées (x, ln(y))
stat.Resid	Valeurs résiduelles associées au modèle exponentiel
stat.ResidTrans	Valeurs résiduelles associées à l'ajustement linéaire des données transformées
stat.XReg	Liste des points de données de la liste *Liste X* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.YReg	Liste des points de données de la liste *Liste Y* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.FreqReg	Liste des fréquences correspondant à *stat.XReg* et *stat.YReg*

F

factor() **Catalogue >**

factor(*nombreRationnel***)** factorise le nombre rationnel en facteurs premiers. Pour les nombres composites, le temps de calcul augmente de façon exponentielle avec le nombre de chiffres du deuxième facteur le plus grand. Par exemple, la factorisation d'un entier composé de 30 chiffres peut prendre plus d'une journée et celle d'un nombre à 100 chiffres, plus d'un siècle.

Pour arrêter un calcul manuellement,

factor(152417172689)	123457·1234577
isPrime(152417172689)	false

factor() Catalogue >

- **Calculatrice:** Maintenez la touche [on] enfoncée et appuyez plusieurs fois sur [enter].
- **Windows® :** Maintenez la touche **F12** enfoncée et appuyez plusieurs fois sur **Entrée**.
- **Macintosh® :** Maintenez la touche **F5** enfoncée et appuyez plusieurs fois sur **Entrée**.
- **iPad® :** L'application affiche une invite. Vous pouvez continuer à patienter ou annuler.

Si vous souhaitez uniquement déterminer si un nombre est un nombre premier, utilisez **isPrime()**. Cette méthode est plus rapide, en particulier si *nombreRationnel* n'est pas un nombre premier et si le deuxième facteur le plus grand comporte plus de cinq chiffres.

FCdf() Catalog >

FCdf
(
lowBound
,*upBound*,*dfNumér*,*dfDénom***)**⇒*nombre* si *lowBound* et *upBound* sont des nombres, *liste* si *lowBound* et *upBound* sont des listes

FCdf
(
lowBound
,*upBound*,*dfNumér*,*dfDénom***)**⇒*nombre* si *lowBound* et *upBound* sont des nombres, *liste* si *lowBound* et *upBound* sont des listes

Calcule la fonction de répartition de la loi de Fisher F de degrés de liberté *dfNumer* et *dfDenom* entre *lowBound* et *upBound*.

Pour P($X \le upBound$), utilisez *lowBound* = 0.

VarMatrice doit avoir été définie.

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \rightarrow amatrix \qquad \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$$

$$\text{Fill } 1.01, amatrix \qquad Done$$

$$amatrix \qquad \begin{bmatrix}1.01 & 1.01\\1.01 & 1.01\end{bmatrix}$$

VarListe doit avoir été définie.

$$\{1,2,3,4,5\} \rightarrow alist \qquad \{1,2,3,4,5\}$$

$$\text{Fill } 1.01, alist \qquad Done$$

$$alist \qquad \{1.01,1.01,1.01,1.01,1.01\}$$

FiveNumSummary

Catalogue >

FiveNumSummary *X*[**,**[*Fréq*][**,***Catégorie***,***Inclure*]]

Donne la version abrégée des statistiques à une variable pour la liste *X*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

X est une liste qui contient les données.

Fréq est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *Fréq* correspond à une fréquence d'occurrence pour chaque valeur *X* correspondante. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Catégorie est une liste de codes numériques de catégories pour les valeurs *X* correspondantes.

Inclure est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

Tout élément vide dans les listes *X*, *Fréq* ou *Catégorie* correspond a un élément vide dans l'ensemble des listes résultantes. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

Variable de sortie	Description
stat.MinX	Minimum des valeurs de x

Variable de sortie	Description
stat.Q_1X	1er quartile de x
stat.MedianX	Médiane de x
stat.Q_3X	3ème quartile de x
stat.MaxX	Maximum des valeurs de x

floor() Catalogue >

Donne le plus grand entier ≤ à l'argument (partie entière). Cette fonction est comparable à **int()**.

L'argument peut être un nombre réel ou un nombre complexe.

floor(*Liste1***)**⇒*liste*

floor(*Matrice1***)**⇒*matrice*

Donne la liste ou la matrice de la partie entière de chaque élément.

Remarque : voi aussi **ceiling()** et **int()**.

floor(−2.14) −3.

floor({3/2, 0, −5.3}) {1, 0, −6.}

floor([1.2 3.4; 2.5 4.8]) [1. 3.; 2. 4.]

For Catalogue >

For *Var*, *Début*, *Fin* [, *Incrément*]
Bloc
EndFor

Exécute de façon itérative les instructions de *Bloc* pour chaque valeur de *Var*, à partir de *Début* jusqu'à *Fin*, par incréments équivalents à *Incrément*.

Var ne doit pas être une variable système.

Incrément peut être une valeur positive ou négative. La valeur par défaut est 1.

Bloc peut correspondre à une ou plusieurs instructions, séparées par un « : ».

```
Define g()=Func                    Done
  Local tempsum,step,i
  0→tempsum
  1→step
  For i,1,100,step
  tempsum+i→tempsum
  EndFor
  EndFunc
g()                                5050
```

For

Catalogue >

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

format()

Catalogue >

chaîneFormat doit être une chaîne du type : « F[n] », « S[n] », « E[n] », « G[n][c] », où [ ] identifie les parties facultatives.

F[n] : format Fixe. n correspond au nombre de chiffres à afficher après le séparateur décimal.

S[n] : format Scientifique. n correspond au nombre de chiffres à afficher après le séparateur décimal.

E[n] : format Ingénieur. n correspond au nombre de chiffres après le premier chiffre significatif. L'exposant est ramené à un multiple de trois et le séparateur décimal est décalé vers la droite de zéro, un ou deux chiffres.

G[n][c] : identique au format Fixe, mais sépare également les chiffres à gauche de la base par groupes de trois. c spécifie le caractère séparateur des groupes et a pour valeur par défaut la virgule. Si c est un point, la base s'affiche sous forme de virgule.

[Rc] : tous les formats ci-dessus peuvent se voir ajouter en suffixe l'indicateur de base Rc, où c correspond à un caractère unique spécifiant le caractère à substituer au point de la base.

format(1.234567,"f3")	"1.235"
format(1.234567,"s2")	"1.23ᴇ0"
format(1.234567,"e3")	"1.235ᴇ0"
format(1.234567,"g3")	"1.235"
format(1234.567,"g3")	"1,234.567"
format(1.234567,"g3,r:")	"1:235"

fPart()

Catalogue >

fPart(*Expr1***)**⇒*expression*

fPart(*Liste1***)**⇒*liste*

fPart(-1.234)	-0.234
fPart({1,-2.3,7.003})	{0,-0.3,0.003}

fPart() Catalogue >

fPart(*Matrice1***)⇒***matrice*

Donne la partie fractionnaire de l'argument.

Dans le cas d'une liste ou d'une matrice, donne les parties fractionnaires des éléments.

L'argument peut être un nombre réel ou un nombre complexe.

FPdf() Catalogue >

FPdf(*ValX***,***dfNumér***,***dfDénom***)⇒***nombre* si *ValX* est un nombre, *liste* si *ValX* est une liste

FPdf(*ValX***,***dfNumér***,***dfDénom***)⇒***nombre* si *ValX* est un nombre, *liste* si *ValX* est une liste

Calcule la densité de la loi F (Fisher) de degrés de liberté *dfNumér* et *dfDénom* en *ValX*.

freqTable▸list() Catalogue >

freqTable▸list
(*Liste1***,***listeEntFréq***)⇒***liste*

Donne la liste comprenant les éléments de *Liste1* développés en fonction des fréquences contenues dans *listEntFréq*. Cette fonction peut être utilisée pour créer une table de fréquences destinée à être utilisée avec l'application Données & statistiques.

Liste1 peut être n'importe quel type de liste valide.

```
freqTable▸list({1,2,3,4},{1,4,3,1})
                {1,2,2,2,2,3,3,3,4}
freqTable▸list({1,2,3,4},{1,4,0,1})
                {1,2,2,2,2,4}
```

listEntFréq doit avoir le même nombre de lignes que *Liste1* et contenir uniquement des éléments entiers non négatifs. Chaque élément indique la fréquence à laquelle l'élément correspondant de *Liste1* doit être répété dans la liste des résultats. La valeur zéro (0) exclut l'élément correspond de *Liste1*.

Remarque : vous pouvez insérer cette fonction à partir du clavier de l'ordinateur en entrant **`freqTable@>list(`**...**`)`**.

Les éléments vides sont ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

frequency() Catalogue >

frequency(*Liste1*,*ListeBinaires***)⇒***liste*

Affiche une liste contenant le nombre total d'éléments dans *Liste1*. Les comptages sont effectués à partir de plages (binaires) définies par l'utilisateur dans *listeBinaires*.

Si *listeBinaires* est {b(1), b(2), …, b(n)}, les plages spécifiées sont {**?**≤b(1), b(1)<**?**≤b(2),…,b(n-1)<**?**≤b(n), b(n)>**?**}. Le résultat comporte un élément de plus que *listeBinaires*.

Chaque élément du résultat correspond au nombre d'éléments dans *Liste1* présents dans la plage. Exprimé en termes de fonction **countIf()**, le résultat est { countIf(liste, **?**≤b(1)), countIf(liste, b(1)<**?**≤b(2)), …, countIf(liste, b(n-1)<**?**≤b(n)), countIf(liste, b(n)>**?**)}.

Les éléments de *Liste1* qui ne sont pas “placés dans une plage” ne sont pas pris en compte. Les éléments vides sont également ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

$$datalist := \{1,2,\mathbf{e},3,\pi,4,5,6,\text{"hello"},7\}$$
$$\{1,2,2.71828,3,3.14159,4,5,6,\text{"hello"},7\}$$
$$\text{frequency}(datalist,\{2.5,4.5\}) \quad \{2,4,3\}$$

Explication du résultat :

2 éléments de *Datalist* sont ≤2,5

4 éléments de *Datalist* sont >2,5 et ≤4,5

3 éléments de *Datalist* sont >4,5

L'élément « hello » est une chaîne et ne peut être placé dans aucune des plages définies.

frequency() Catalogue >

Dans l'application Tableur & listes, vous pouvez utiliser une plage de cellules à la place des deux arguments.

Remarque : voir également **countIf()**, page 31.

FTest_2Samp Catalogue >

FTest_2Samp *Liste1*,*Liste2*[,*Fréq1*[,*Fréq2*[,*Hypoth*]]]

FTest_2Samp *Liste1*,*Liste2*[,*Fréq1*[,*Fréq2*[,*Hypoth*]]]

(Entrée de liste de données)

FTest_2Samp *sx1*,*n1*,*sx2*,*n2*[,*Hypoth*]

FTest_2Samp *sx1*,*n1*,*sx2*,*n2*[,*Hypoth*]

(Récapitulatif des statistiques fournies en entrée)

Effectue un test F sur deux échantillons. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Pour H_a : $\sigma 1 > \sigma 2$, définissez *Hypoth*>0

Pour H_a : $\sigma 1 \neq \sigma 2$ (par défaut), définissez *Hypoth* =0

Pour H_a : $\sigma 1 < \sigma 2$, définissez *Hypoth*<0

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.F	Statistique Û estimée pour la séquence de données
stat.PVal	Plus petit seuil de signification permettant de rejeter l'hypothèse nulle
stat.dfNumer	Numérateur degrés de liberté = n1-1
stat.dfDenom	Dénominateur degrés de liberté = n2-1.

Variable de sortie	Description
stat.sx1, stat.sx2	Écarts types de population d'échantillon des séquences de données dans *Liste 1* et *Liste 2*.
stat.x1_bar stat.x2_bar	Moyenne de population d'échantillon des séquences de données dans *Liste 1* et *Liste 2*.
stat.n1, stat.n2	Taille des échantillons

Func

Catalogue >

Func
Bloc
EndFunc

Modèle de création d'une fonction définie par l'utilisateur.

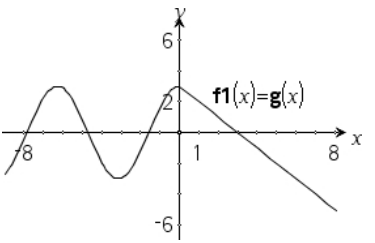
Bloc peut correspondre à une instruction unique ou à une série d'instructions séparées par le caractère ":" ou à une série d'instructions réparties sur plusieurs lignes. La fonction peut utiliser l'instruction **Return** pour donner un résultat spécifique.

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

Définition d'une fonction par morceaux :

```
Define g(x)=Func                    Done
              If x<0 Then
              Return 3·cos(x)
              Else
              Return 3–x
              EndIf
              EndFunc
```

Résultat de la représentation graphique de g (x)



G

gcd()

Catalogue >

gcd(*Nombre1, Nombre2***)⇒***expression*

Donne le plus grand commun diviseur des deux arguments. Le **gcd** de deux fractions correspond au **gcd** de leur numérateur divisé par le **lcm** de leur dénominateur.

```
gcd(18,33)                          3
```

gcd() Catalogue >

En mode Auto ou Approché, le **gcd** de nombre fractionnaires en virgule flottante est égal à 1.

gcd(*Liste1, Liste2***)⇒***liste*

Donne la liste des plus grands communs diviseurs des éléments correspondants de *Liste1* et *Liste2*.

$\gcd(\{12,14,16\},\{9,7,5\})$ $\{3,7,1\}$

gcd(*Matrice1, Matrice2***)⇒***matrice*

Donne la matrice des plus grands communs diviseurs des éléments correspondants de *Matrice1* et *Matrice2*.

$\gcd\left(\begin{bmatrix}2 & 4\\ 6 & 8\end{bmatrix},\begin{bmatrix}4 & 8\\ 12 & 16\end{bmatrix}\right)$ $\begin{bmatrix}2 & 4\\ 6 & 8\end{bmatrix}$

geomCdf() Catalogue >

geomCdf(*p***,***lowBound***,***upBound***)⇒***nombre* si les bornes *lowBound* et *upBound* sont des nombres, *liste* si les bornes *lowBound* et *upBound* sont des listes

geomCdf(*p***,***upBound***)** pour P(1≤X ≤*upBound*)⇒*nombre* si la borne *upBound* est un nombre, *liste* si la borne *upBound* est une liste

Calcule la probabilité qu'une variable suivant la loi géométrique prenne une valeur entre les bornes *lowBound* et *upBound* en fonction de la probabilité de réussite *p* spécifiée.

Pour P(X ≤ *upBound*), définissez *lowBound* = 1.

geomPdf() Catalogue >

geomPdf(*p***,***ValX***)⇒***nombre* si *ValX* est un nombre, *liste* si *ValX* est une liste

Calcule la probabilité que le premier succès intervienne au rang *ValX*, pour la loi géométrique discrète en fonction de la probabilité de réussite p spécifiée.

Get[*promptString*,]*var*[, *statusVar*]

Get[*promptString*,] *fonc*(*arg1*, *...argn*) [, *statusVar*]

Commande de programmation : récupère une valeur d'un hub connecté TI-Innovator™ Hub et affecte cette valeur à la variable *var*.

La valeur doit être demandée :

- À l'avance, par le biais d'une commande **Send "READ ..."** commande.

  — ou —

- En incorporant une demande **"READ ..."** comme l'argument facultatif de *promptString*. Cette méthode vous permet d'utiliser une seule commande pour demander la valeur et la récupérer.

Une simplification implicite a lieu. Par exemple, la réception de la chaîne de caractères "123" est interprétée comme étant une valeur numérique. Pour conserver la chaîne de caractères, utilisez **GetStr** au lieu de **Get**.

Si vous incluez l'argument facultatif *statusVar*, une valeur lui sera affectée en fonction de la réussite de l'opération. Une valeur zéro signifie qu'aucune donnée n'a été reçue.

Dans la deuxième syntaxe, l'argument *fonc*() permet à un programme de stocker la chaîne de caractères reçue comme étant la définition d'une fonction. Cette syntaxe équivaut à l'exécution par le programme de la commande suivante :

Define *fonc*(*arg1*, *...argn*) = *chaîne reçue*

Le programme peut alors utiliser la fonction définie *fonc*().

Exemple : demander la valeur actuelle du capteur intégré du niveau de lumière du hub. Utilisez **Get** pour récupérer la valeur et l'affecter à la variable *lightval*.

Send "READ BRIGHTNESS"	*Done*
Get *lightval*	*Done*
lightval	0.347922

Incorporez la demande READ dans la commande **Get**.

Get "READ BRIGHTNESS",*lightval*	*Done*
lightval	0.378441

Get **Menu hub**

Remarque : vous pouvez utiliser la commande **Get** dans un programme défini par l'utilisateur, mais pas dans une fonction.

Remarque : Voir également **GetStr**, page 70 et **Send**, page 143.

getDenom() **Catalogue >**

Transforme l'argument en une expression dotée d'un dénominateur commun réduit, puis en donne le numérateur.

getKey() Catalogue >

getKey([0|1]) ⇒ returnString

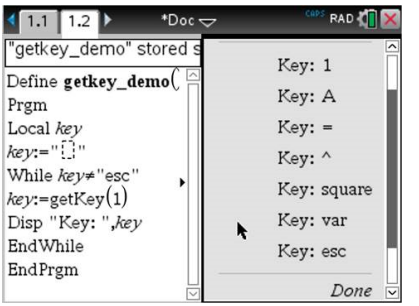
Description :getKey() - permet à un programme TI-Basic de recevoir des entrées de clavier - calculatrice, ordinateur de bureau et émulateur sur ordinateur de bureau.

Par exemple :

- keypressed := **getKey()** retournera une touche ou une chaîne vide si aucune touche n'a été pressée. Cet appel sera immédiatement retourné.
- keypressed := **getKey(1)** attendra l'appui sur une touche. Cet appel mettra en pause l'exécution du programme jusqu'à l'appui sur une touche.

```
getKey()
```

Par exemple :



Traitement des frappes de touche :

Touche de calculatrice/émulateur	Ordinateur	Valeur de retour
Échap	Échap	« échap »
Pavé tactile - Clic en haut	n/a	« haut »
On	n/a	« accueil »
Scratchapps	n/a	"scratchpad"
Pavé tactile - Clic gauche	n/a	« gauche »
Pavé tactile - Clic au centre	n/a	« centre »
Pavé tactile - Clic droit	n/a	« droite »
Classeur	n/a	« classeur »
Tab	Tab	« tab »
Pavé tactile - Clic en bas	Flèche bas	« bas »
Menu	n/a	« menu »

Touche de calculatrice/émulateur	Ordinateur	Valeur de retour
Ctrl	Ctrl	aucun retour
Maj	Maj	aucun retour
Var	n/a	« var »
Suppr	n/a	« suppr »
=	=	"="
trigonométrie	n/a	« trigonométrie »
0 à 9	0-9	« 0 » ... « 9 »
Modèles	n/a	« modèle »
Catalogue	n/a	« cat »
^	^	"^"
X^2	n/a	« carré »
/ (touche division)	/	"/"
* (touche multiplication)	*	"*"
e^x	n/a	« expr »
10^x	n/a	« puissance de 10 »
+	+	"+"
-	-	"-"
(	(	"("
)	)	")"
.	.	"."
(-)	n/a	« - » (signe moins)
Entrée	Entrée	« entrée »
ee	n/a	« E » (notation scientifique E)
a - z	a-z	alpha = lettre pressée (minuscule) ("a" - "z")
maj a-z	maj a-z	alpha = lettre pressée « A » - « Z »
		Note : ctrl-maj fonctionne pour le verrouillage des

Touche de calculatrice/émulateur	Ordinateur	Valeur de retour majuscules
?!	n/a	"?!"
pi	n/a	« pi »
Marque	n/a	aucun retour
,	,	","
Retour	n/a	Retour
Espace	Espace	« » (espace)
Inaccessible	Touches de caractères spéciaux tels que @,!,^, etc.	Le caractère est retourné
n/a	Touches de fonction	Aucun caractère retourné
n/a	Touches de commandes spéciales pour ordinateur	Aucun caractère retourné
Inaccessible	Autres touches pour ordinateur non disponibles sur la calculatrice lorsque getkey() est en attente d'une frappe. ({, },;, :, ...)	Le même caractère que vous obtenez dans l'Éditeur mathématique (pas dans une boîte mathématique)

Remarque : Il est important de noter que la présence de **getKey()** dans un programme modifie la façon dont certains événements sont traités par le système. Certains sont décrits ci-dessous.

Arrête le programme et traite l'événement - Exactement comme si l'utilisateur quittait le programme en appuyant sur la touche **ON**

« **Support** » ci-dessous signifie - le système fonctionne comme prévu - le programme continue à être exécuté.

Événement	Unité nomade	Ordinateur - TI-Nspire™ Student Software
Questions rapides	Arrête le programme, traite l'événement	Comme avec l'unité nomade (TI-Nspire™ Student Software, TI-Nspire™ Navigator™ NC Teacher Software-uniquement)
Gestion des fichiers à distance	Arrête le programme, traite l'événement	Comme avec l'unité nomade. (TI-Nspire™ Student

Événement	Unité nomade	Ordinateur - TI-Nspire™ Student Software
(Incl. l'envoi du fichier « Exit Press 2 Test » d'une unité nomade à une autre ou à un ordinateur)		Software, TI-Nspire™ Navigator™ NC Teacher Software-uniquement)
Fermer la classe	Arrête le programme, traite l'événement	Support (TI-Nspire™ Student Software, TI-Nspire™ Navigator™ NC Teacher Software-uniquement)

Événement	Unité nomade	Ordinateur - TI-Nspire™ Toutes les versions
TI-Innovator™ Hub connexion/déconnexion	Support - Peut émettre avec succès des commandes à TI-Innovator™ Hub. Après que vous ayez quitté le programme, le TI-Innovator™ Hubcontinue de travailler avec l'unité nomade.	Comme avec l'unité nomade

getLangInfo() Catalogue >

getLangInfo()⇒*chaîne*

Retourne une chaîne qui correspond au nom abrégé de la langue active. Vous pouvez, par exemple, l'utiliser dans un programme ou une fonction afin de déterminer la langue courante.

getLangInfo() "en"

Anglais = « en »

Danois = « da »

Allemand = « de »

Finlandais = « fi »

Français = « fr »

Italien = « it »

Néerlandais = « nl »

Néerlandais belge = « nl_BE »

getLangInfo() Catalogue >

Norvégien = « no »

Portugais = « pt »

Espagnol = « es »

Suédois = « sv »

getLockInfo() Catalogue >

getLockInfo(*Var*)⇒*valeur*

Donne l'état de verrouillage/déverrouillage de la variable *Var*.

valeur =**0** : *Var* est déverrouillée ou n'existe pas.

valeur =**1** : *Var* est verrouillée et ne peut être ni modifiée ni supprimée.

Voir **Lock, page 92 et unLock, page 174.**

a:=65	65
Lock *a*	*Done*
getLockInfo(*a*)	1
a:=75	"Error: Variable is locked."
DelVar *a*	"Error: Variable is locked."
Unlock *a*	*Done*
a:=75	75
DelVar *a*	*Done*

getMode() Catalogue >

getMode(*EntierNomMode***)**⇒*valeur*

getMode(0)⇒*liste*

getMode(*EntierNomMode***)** affiche une valeur représentant le réglage actuel du mode *EntierNomMode*.

getMode(0) affiche une liste contenant des paires de chiffres. Chaque paire consiste en un entier correspondant au mode et un entier correspondant au réglage.

Pour obtenir une liste des modes et de leurs réglages, reportez-vous au tableau ci-dessous.

Si vous enregistrez les réglages avec **getMode(0)** → *var*, vous pouvez utiliser **setMode(***var***)** dans une fonction ou un programme pour restaurer temporairement les réglages au sein de l'exécution de la fonction ou du programme uniquement. Voir également **setMode()**, page 146.

Nom du mode	Entier du mode	Entiers de réglage
Afficher chiffres	1	**1**=Flottant, **2**=Flottant 1, **3**=Flottant 2, **4**=Flottant 3, **5**=Flottant 4, **6**=Flottant 5, **7**=Flottant 6, **8**=Flottant 7, **9**=Flottant 8, **10**=Flottant 9, **11**=Flottant 10, **12**=Flottant 11, **13**=Flottant 12, **14**=Fixe 0, **15**=Fixe 1, **16**=Fixe 2, **17**=Fixe 3, **18**=Fixe 4, **19**=Fixe 5, **20**=Fixe 6, **21**=Fixe 7, **22**=Fixe 8, **23**=Fixe 9, **24**=Fixe 10, **25**=Fixe 11, **26**=Fixe 12
Angle	2	**1**=Radian, **2**=Degré, **3**=Grade
Format Exponentiel	3	**1**=Normal, **2**=Scientifique, **3**=Ingénieur
Réel ou Complexe	4	**1**=Réel, **2**=Rectangulaire, **3**=Polaire
Auto ou Approché	5	**1**=Auto, **2**=Approché
Format Vecteur	6	**1**=Rectangulaire, **2**=Cylindrique, **3**=Sphérique
Base	7	**1**=Décimale, **2**=Hexadécimale, **3**=Binaire

getNum() — Catalogue >

Transforme l'argument en une expression dotée d'un dénominateur commun réduit, puis en donne le dénominateur.

GetStr — Menu hub

GetStr[*promptString*,] *var*[**,** *statusVar*]

GetStr[*promptString*,] *fonc*(*arg1*, *...argn*) [**,** *statusVar*]

Commande de programmation : fonctionne de manière identique à la commande **Get**, sauf que la valeur reçue est toujours interprétée comme étant une chaîne de caractères. En revanche, la commande **Get** interprète la réponse comme une expression, à moins que l'utilisateur ne la saisisse entre guillemets ("").

Remarque : Voir également **Get**, page 63 et **Send**, page 143.

Par exemple, voir **Get**.

getType() Catalogue >

getType(*var*)⇒*chaîne* de caractères

Retourne une chaîne de caractère qui indique le type de données de la variable *var*.

Si *var* n'a pas été définie, retourne la chaîne "AUCUNE".

$\{1,2,3\} \rightarrow temp$	$\{1,2,3\}$
getType(*temp*)	"LIST"
$3 \cdot i \rightarrow temp$	$3 \cdot i$
getType(*temp*)	"EXPR"
DelVar *temp*	*Done*
getType(*temp*)	"NONE"

getVarInfo() Catalogue >

getVarInfo()⇒*matrice* ou *chaîne*

getVarInfo (*chaîneNomBibliothèque*)⇒*matrice* ou *chaîne*

getVarInfo() donne une matrice d'informations (nom et type de la variable, accès à la bibliothèque et état de verrouillage/déverrouillage) pour toutes les variables et objets de la bibliothèque définis dans l'activité courante.

Si aucune variable n'est définie, **getVarInfo()** donne la chaîne « NONE » (AUCUNE).

getVarInfo (*chaîneNomBibliothèque*)donne une matrice d'informations pour tous les objets de bibliothèque définis dans la bibliothèque *chaîneNomBibliothèque*. *chaîneNomBibliothèque* doit être une chaîne (texte entre guillemets) ou une variable.

Si la bibliothèque *chaîneNomBibliothèque* n'existe pas, une erreur est générée.

getVarInfo()	"NONE"
Define $x=5$	*Done*
Lock x	*Done*
Define LibPriv $y=\{1,2,3\}$	*Done*
Define LibPub $z(x)=3 \cdot x^2-x$	*Done*
getVarInfo()	$\begin{bmatrix} x & \text{"NUM"} & \text{"}\square\text{"} & 1 \\ y & \text{"LIST"} & \text{"LibPriv "} & 0 \\ z & \text{"FUNC"} & \text{"LibPub "} & 0 \end{bmatrix}$
getVarInfo(*tmp3*)	"Error: Argument must be a string"
getVarInfo("tmp3")	[*volcyl2* "NONE" "LibPub " 0]

getVarInfo() Catalogue >

Observez l'exemple de gauche dans lequel le résultat de **getVarInfo()** est affecté à la variable *vs*. La tentative d'afficher la ligne 2 ou 3 de *vs* génère un message d'erreur “Liste ou matrice invalide” car pour au moins un des éléments de ces lignes (variable *b*, par exemple) l'évaluation redonne une matrice.

Cette erreur peut également survenir lors de l'utilisation de *Ans* pour réévaluer un résultat de **getVarInfo()**.

Le système génère l'erreur ci-dessus car la version courante du logiciel ne prend pas en charge les structures de matrice généralisées dans lesquelles un élément de matrice peut être une matrice ou une liste.

$a:=1$	1
$b:=[1\ \ 2]$	$[1\ \ 2]$
$c:=[1\ \ 3\ \ 7]$	$[1\ \ 3\ \ 7]$
$vs:=\text{getVarInfo}()$	$\begin{bmatrix} a & \text{"NUM"} & \text{"}\square\text{"} & 0 \\ b & \text{"MAT"} & \text{"}\square\text{"} & 0 \\ c & \text{"MAT"} & \text{"}\square\text{"} & 0 \end{bmatrix}$
$vs[1]$	$[1\ \ \text{"NUM"}\ \ \text{"}\square\text{"}\ \ 0]$
$vs[1,1]$	1
$vs[2]$	"Error: Invalid list or matrix"
$vs[2,1]$	$[1\ \ 2]$

Goto Catalogue >

Goto *nomÉtiquette*

Transfère le contrôle du programme à l'étiquette *nomÉtiquette*.

nomÉtiquette doit être défini dans la même fonction à l'aide de l'instruction **Lbl**.

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

```
Define g()=Func                    Done
    Local temp,i
    0→temp
    1→i
    Lbl top
    temp+i→temp
    If i<10 Then
    i+1→i
    Goto top
    EndIf
    Return temp
    EndFunc
g()                                55
```

▸Grad Catalogue >

Expr1 ▸ **Grad**⇒*expression*

Convertit *Expr1* en une mesure d'angle en grades.

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **@>Grad**.

En mode Angle en degrés :

$(1.5)\blacktriangleright\text{Grad}$	$(1.66667)^g$

En mode Angle en radians :

▸Grad

Catalogue >

(1.5)▸Grad	(95.493)^g

I

identity()

Catalogue >

identity(*Entier***)** ⇒ *matrice*

Donne la matrice unité de dimension *Entier*.

Entier doit être un entier positif

identity(4)	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

If

Catalogue >

If *BooleanExpr*
Relevé

If *BooleanExpr* **Then**
Bloc
EndIf

Si *BooleanExpr* est évalué à vrai, exécute l'instruction *Instruction* ou le bloc d'instructions *Bloc* avant de poursuivre l'exécution de la fonction

Si *BooleanExpr* est évalué à faux, poursuit l'exécution en ignorant l'instruction ou le bloc d'instructions

Bloc peut correspondre à une ou plusieurs instructions, séparées par le caractère « : »

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

Define $g(x)$=Func
If $x<0$ Then
Return x^2
EndIf
EndFunc
	Done

$g(-2)$	4

If

Catalogue >

If *BooleanExpr* **Then**
Bloc1
Else
Bloc2
EndIf

Si *BooleanExpr* est évalué à vrai, exécute *Bloc1* et ignore *Bloc2*.

Si *BooleanExpr* est évalué à faux, ignore *Bloc1*, mais exécute *Bloc2*.

Bloc1 et *Bloc2* peuvent correspondre à une seule instruction.

```
Define g(x)=Func                    Done
              If x<0 Then
              Return -x
              Else
              Return x
              EndIf
              EndFunc
g(12)                               12
g(-12)                              12
```

If *BooleanExpr1* **Then**
Bloc1
ElseIf *BooleanExpr2* **Then**
Bloc2
⋮
ElseIf *BooleanExprN* **Then**
BlocN
EndIf

Permet de traiter les conditions multiples. Si *BooleanExpr1* est évalué à vrai, exécute *Bloc1* Si *BooleanExpr1* est évalué à faux, évalue *BooleanExpr2*, et ainsi de suite.

```
Define g(x)=Func
              If x<-5 Then
              Return 5
              ElseIf x>-5 and x<0 Then
              Return -x
              ElseIf x≥0 and x≠10 Then
              Return x
              ElseIf x=10 Then
              Return 3
              EndIf
              EndFunc
                                    Done
g(-4)                               4
g(10)                               3
```

ifFn()

Catalogue >

ifFn(*exprBooléenne***,***Valeur_si_Vrai* [**,***Valeur_si_Faux* [**,***Valeur_si_Inconnu*]]**)** ⇒ *expression, liste ou matrice*

Evalue l'expression booléenne *exprBooléenne* (ou chacun des éléments de *exprBooléenne*) et produit un résultat reposant sur les règles suivantes

- *exprBooléenne* peut tester une valeur unique, une liste ou une matrice
- Si un élément de *exprBooléenne* est évalué à vrai, l'élément correspondant de *Valeur_si_Vrai s'affiche*
- Si un élément de *exprBooléenne* est

```
ifFn({1,2,3}<2.5,{5,6,7},{8,9,10})
                                    {5,6,10}
```

La valeur d'essai **1** est inférieure à 2,5, ainsi l'élément correspondant dans

Valeur_si_Vrai **5** est copié dans la liste de résultats.

La valeur d'essai **2** est inférieure à 2,5, ainsi l'élément correspondant dans

Valeur_si_Vrai **6** est copié dans la liste de résultats.

évalué à faux, l'élément correspondant de *Valeur_si_Faux s'affiche* Si vous omettez *Valeur_si_Faux*, undef s'affiche.

- Si un élément de *exprBooléenne* n'est ni vrai ni faux, l'élément correspondant de *Valeur_si_Inconnu* s'affiche Si vous omettez *Valeur_si_Inconnu*, undef s'affiche
- Si le deuxième, troisième ou quatrième argument de la fonction **ifFn()** est une expression unique, le test booléen est appliqué à toutes les positions dans *exprBooléenne*

Remarque : si l'instruction simplifiée *exprBooléenne* implique une liste ou une matrice, tous les autres arguments de type liste ou matrice doivent avoir la ou les même(s) dimension(s) et le résultat aura la ou les même(s) dimension(s).

La valeur d'essai **3** n'est pas inférieure à 2,5, ainsi l'élément correspondant dans Valeur_si_Faux *10* est copié dans la liste de résultats

$$\text{ifFn}(\{1,2,3\}<2.5,4,\{8,9,10\}) \qquad \{4,4,10\}$$

Valeur_si_Vrai est une valeur unique et correspond à n'importe quelle position sélectionnée

$$\text{ifFn}(\{1,2,3\}<2.5,\{5,6,7\}) \qquad \{5,6,\text{undef}\}$$

Valeur_si_Faux n'est pas spécifié Undef est utilisé.

$$\text{ifFn}(\{2,\text{"a"}\}<2.5,\{6,7\},\{9,10\},\text{"err"}) \qquad \{6,\text{"err"}\}$$

Un élément sélectionné à partir de *Valeur_si_Vrai*. Un élément sélectionné à partir de *Valeur_si_Inconnu*.

imag() Catalogue >

Donne la partie imaginaire de l'argument.

imag(*Liste1***)** ⇒ *liste*

Donne la liste des parties imaginaires des éléments.

imag(*Matrice1***)** ⇒ *matrice*

Donne la matrice des parties imaginaires des éléments.

$$\text{imag}(\{-3,4-i,i\}) \qquad \{0,-1,1\}$$

Indirection Voir #(), page 199.

inString()

Catalogue >

inString(*srcString***,** *subString***[,** *Début***])** ⇒ *entier*

Donne le rang du caractère de la chaîne *chaîneSrce* où commence la première occurrence de *sousChaîne*.

Début, s'il est utilisé, indique le point de départ de la recherche dans *chaîneSrce* Par défaut = 1, la recherche commence à partir du (premier caractère de *chaîneSrce*).

Si *chaîneSrce* ne contient pas *sousChaîne* ou si *Début* est strictement supérieur à la longueur de *ChaîneSrce*, on obtient zéro

inString("Hello there","the")	7
inString("ABCEFG","D")	0

int()

Catalogue >

int(*List1***)** ⇒ *liste*
int(*Matrix1***)** ⇒ *matrice*

Donne le plus grand entier inférieur ou égal à l'argument. Cette fonction est identique à **floor()** (partie entière).

L'argument peut être un nombre réel ou un nombre complexe.

Dans le cas d'une liste ou d'une matrice, donne la partie entière de chaque élément.

int(-2.5)	-3.
int([-1.234 0 0.37])	[-2. 0 0.]

intDiv()

Catalogue >

intDiv(*Number1***,** *Number2***)** ⇒ *entier*
intDiv(*List1***,** *List2***)** ⇒ *liste*
intDiv(*Matrix1***,** *Matrix2***)** ⇒ *matrice*

Donne le quotient dans la division euclidienne de (*Nombre1* ÷ *Nombre2*).

Dans le cas d'une liste ou d'une matrice, donne le quotient de (argument 1 ÷ argument 2) pour chaque paire d'éléments.

intDiv(-7,2)	-3
intDiv(4,5)	0
intDiv({12,-14,-16},{5,4,-3})	{2,-3,5}

interpoler () Catalogue >

interpoler(*Valeurx*, *Listex*, *Listey*, *ListePrincy***)** ⇒ *list*

Cette fonction effectue l'opération suivante :

Étant donné *Listex*, *Listey*=**f(***Listex***)** et *ListePrincy*=**f'(***Listex***)** pour une fonction **f** inconnue, une interpolation par une spline cubique est utilisée pour donner une approximation de la fonction **f** en *Valeurx*. On suppose que *Listex* est une liste croissante ou décroissante de nombres, cette fonction pouvant retourner une valeur même si ce n'est pas le cas. Elle examine la *Listex* et recherche un intervalle [*Listex*[i], *Listex* [i+1]] qui contient *Valeurx*. Si elle trouve cet intervalle, elle retourne une valeur d'interpolation pour **f(Valeurx**), sinon elle donne **undef.**

Listex, *Listey*, et *ListePrincy* doivent être de même dimensions ≥ 2 et contenir des expressions pouvant être évaluées à des nombres.

Équation différentielle :
y'=-3•*y*+6•*t*+5 et *y*(0)=5

```
rk:=rk23(-3·y+6·t+5,t,y,{0,10},5,1)
[0.    1.        2.        3.        4.
 5.  3.19499  5.00394  6.99957  9.00593  10
```

Pour afficher le résultat entier, appuyez sur ▲, puis utilisez les touches ◄ et ► pour déplacer le curseur.

Utilisez la fonction interpolate() pour calculer les valeurs de la fonction pour la listevaleursx :

```
xvaluelist:=seq(i,i,0,10,0.5)
{0,0.5,1.,1.5,2.,2.5,3.,3.5,4.,4.5,5.,5.5,6.,6.5,
xlist:=mat▶list(rk[1])
{0.,1.,2.,3.,4.,5.,6.,7.,8.,9.,10.}
ylist:=mat▶list(rk[2])
{5.,3.19499,5.00394,6.99957,9.00593,10.9978
yprimelist:=-3·y+6·t+5|y=ylist and t=xlist
{-10.,1.41503,1.98819,2.00129,1.98221,2.006
interpolate(xvaluelist,xlist,ylist,yprimelist)
{5.,2.67062,3.19499,4.02782,5.00394,6.00011
```

invχ²() Catalogue >

invχ²(*Aire*,*df***)**

invChi2(*Aire*,*df***)**

Calcule l'inverse de la fonction de répartition de la loi X^2 (Khi2) de degré de liberté *df* en un point donné *Aire*.

invF() Catalogue >

invF(*Aire*,*dfNumer*,*dfDenom***)**

invF(*Zone*,*dfNumer*,*dfDenom***)**

Calcule l'inverse de la fonction de répartition de la loi F (Fisher) de paramètres spécifiée par *dfNumer* et *dfDenom* en un point donné *Aire*

invBinom() Catalogue >

invBinom
(*CumulativeProb***,***NumTrials***,***Prob***,**
*OutputForm***)⇒** *scalaire* ou *matrice*

Étant donné le nombre d'essais (*NumTrials*) et la probabilité de réussite de chaque essai (*Prob*), cette fonction renvoie le nombre minimal de réussites, *k*, tel que la probabilité cumulée de *k* réussites soit supérieure ou égale à une probabilité cumulée donnée (*CumulativeProb*).

OutputForm=**0**, affiche le résultat en tant que scalaire (par défaut).

OutputForm=**1**, affiche le résultat en tant que matrice.

Par exemple : Mary et Kevin jouent à un jeu de dés. Mary doit deviner le nombre maximal de fois où 6 apparaît dans 30 lancers. Si le nombre 6 apparaît autant de fois ou moins, Mary gagne. Par ailleurs, plus le nombre qu'elle devine est petit, plus ses gains sont élevés. Quel est le plus petit nombre que Mary peut deviner si elle veut que la probabilité du gain soit supérieure à 77 % ?

$$\text{invBinom}\left(0.77,30,\frac{1}{6}\right) \qquad 6$$

$$\text{invBinom}\left(0.77,30,\frac{1}{6},1\right) \qquad \begin{bmatrix} 5 & 0.616447 \\ 6 & 0.776537 \end{bmatrix}$$

invBinomN() Catalogue >

invBinomN(*CumulativeProb***,***Prob***,**
*NumSuccess,OutputForm***)⇒** *scalaire* ou *matrice*

Étant donné la probabilité de réussite de chaque essai (*Prob*) et le nombre de réussites (NumSuccess), cette fonction renvoie le nombre minimal d'essais, *N*, tel que la probabilité cumulée de *x* réussites soit inférieure ou égale à une probabilité cumulée donnée (*CumulativeProb*).

OutputForm=**0**, affiche le résultat en tant que scalaire (par défaut).

OutputForm=**1**, affiche le résultat en tant que matrice.

Par exemple : Monique s'entraîne aux tirs au but au volley-ball. Elle sait par expérience que ses chances de marquer un but sont de 70 %. Elle prévoit de s'entraîner jusqu'à ce qu'elle marque 50 buts. Combien de tirs doit-elle tenter pour s'assurer que la probabilité de marquer au moins 50 buts est supérieure à 0,99 ?

$$\text{invBinomN}(0.01,0.7,49) \qquad 86$$

$$\text{invBinomN}(0.01,0.7,49,1) \qquad \begin{bmatrix} 85 & 0.010451 \\ 86 & 0.00709 \end{bmatrix}$$

invNorm() Catalogue >

invNorm(*Aire*[**,**μ[**,**σ]]**)**

Calcule l'inverse de la fonction de répartition de la loi normale de paramètres μ et *σ* en un point donné *Aire*.

invt() Catalogue >

invt(*Aire*,*df*)

Calcule les fractiles d'une loi de Student à *df* degrés de liberté pour une *Aire* donnée.

iPart() Catalogue >

iPart(*Number*) ⇒ *entier*
iPart(*List1*) ⇒ *liste*
iPart(*Matrix1*) ⇒ *matrice*

Donne l'argument moins sa partie fractionnaire.

Dans le cas d'une liste ou d'une matrice, applique la fonction à chaque élément.

L'argument peut être un nombre réel ou un nombre complexe.

$\text{iPart}(-1.234)$ → $-1.$

$\text{iPart}\left\{\frac{3}{2},-2.3,7.003\right\}$ → $\{1.,-2.,7.\}$

irr() Catalogue >

irr(*CF0*,*CFList* [,*CFFreq*]) ⇒ *valeur*

Fonction financière permettant de calculer le taux interne de rentabilité d'un investissement.

MT0 correspond au mouvement de trésorerie initial à l'heure 0 ; il doit s'agir d'un nombre réel.

Liste MT est une liste des montants de mouvements de trésorerie après le mouvement de trésorerie initial MT0.

FréqMT est une liste facultative dans laquelle chaque élément indique la fréquence d'occurrence d'un montant de mouvement de trésorerie groupé (consécutif), correspondant à l'élément de *ListeMT* La valeur par défaut est 1 ; si vous saisissez des valeurs, elles doivent être des entiers positifs < 10 000

Remarque : Voir également **mirr()**, page 101.

list1:={6000,−8000,2000,−3000} → {6000,−8000,2000,−3000}

list2:={2,2,2,1} → {2,2,2,1}

irr(5000,*list1*,*list2*) → −4.64484

isPrime() Catalogue >

isPrime(*Nombre***)** ⇒ *Expression booléenne constante*

Donne true ou false selon que *nombre* est ou n'est pas un entier naturel premier ≥ 2, divisible uniquement par lui-même et 1.

Si *Nombre* dépasse 306 chiffres environ et n'a pas de diviseur ≤1021, **isPrime (***Nombre***)** affiche un message d'erreur.

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

isPrime(5)	true
isPrime(6)	false

Fonction permettant de trouver le nombre premier suivant un nombre spécifié :

Define *nextprim*(*n*)=	Func	*Done*
	Loop	
	$n+1 \rightarrow n$	
	If isPrime(*n*)	
	Return *n*	
	EndLoop	
	EndFunc	
nextprim(7)		11

isVoid() Catalogue >

isVoid(*Var***)** ⇒ *Expression booléenne constante*
isVoid(*Expr***)** ⇒ *Expression booléenne constante*
isVoid(*Var***)** ⇒ *liste d'expressions booléennes constantes*

Retourne true ou false pour indiquer si l'argument est un élément de type données vide.

Pour plus d'informations concernant les éléments vides, reportez-vous à . page 223.

a:=_	_
isVoid(*a*)	true
isVoid({1,_,3})	{false,true,false}

Lbl

Catalogue >

Lbl *nomÉtiquette*

Définit une étiquette en lui attribuant le nom *nomÉtiquette* dans une fonction.

Vous pouvez utiliser l'instruction **Goto** *nomÉtiquette* pour transférer le contrôle du programme à l'instruction suivant immédiatement l'étiquette.

nomÉtiquette doit être conforme aux mêmes règles de dénomination que celles applicables aux noms de variables.

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

```
Define g()=Func                    Done
    Local temp,i
    0→temp
    1→i
    Lbl top
    temp+i→temp
    If i<10 Then
    i+1→i
    Goto top
    EndIf
    Return temp
    EndFunc
g()                                 55
```

lcm()

Catalogue >

lcm(*Nombre1***,** *Nombre2***)⇒***expression*

lcm(*Liste1***,** *Liste2***)⇒***liste*

lcm(*Matrice1***,** *Matrice2***)⇒***matrice*

Donne le plus petit commun multiple des deux arguments. Le **lcm** de deux fractions correspond au **lcm** de leur numérateur divisé par le **gcd** de leur dénominateur. Le **lcm** de nombres fractionnaires en virgule flottante correspond à leur produit.

Pour deux listes ou matrices, donne les plus petits communs multiples des éléments correspondants.

$\text{lcm}(6,9)$ → 18

$\text{lcm}\left(\left\{\frac{1}{3},-14,16\right\},\left\{\frac{2}{15},7,5\right\}\right)$ → $\left\{\frac{2}{3},14,80\right\}$

left()

Catalogue >

left(*chaîneSrce***[,** *Nomb***])⇒***chaîne*

left("Hello",2) → "He"

left() Catalogue >

Donne la chaîne formée par les *Nomb* premiers caractères de la chaîne *chaîneSrce*.

Si *Nomb* est absent, on obtient *chaîneSrce*.

left(*Liste1*[**,** *Nomb*]**)**⇒*liste*

Donne la liste formée par les *Nomb* premiers éléments de *Liste1*.

Si *Nomb* est absent, on obtient *Liste1*.

left(*Comparaison***)**⇒*expression*

Donne le membre de gauche d'une équation ou d'une inéquation.

```
left({1,3,-2,4},3)                {1,3,-2}
left(x<3)                         x
```

libShortcut() Catalogue >

libShortcut(*chaîneNomBibliothèque***,** *chaîneNomRaccourci*[**,** *LibPrivFlag*]**)**⇒*liste de variables*

Crée un groupe de variables dans l'activité courante qui contient des références à tous les objets du classeur de bibliothèque spécifié *chaîneNomBibliothèque*. Ajoute également les membres du groupe au menu Variables. Vous pouvez ensuite faire référence à chaque objet en utilisant la *chaîneNomRaccourci* correspondante.

Définissez *LibPrivFlag*=**0** pour exclure des objets de la bibliothèque privée (par défaut) et *LibPrivFlag*=**1** pour inclure des objets de bibliothèque privée.

Pour copier un groupe de variables, reportez-vous à **CopyVar**, page 26. Pour supprimer un groupe de variables, reportez-vous à **DelVar**, page 40.

Cet exemple utilise un classeur de bibliothèque enregistré et rafraîchi **linalg2** qui contient les objets définis comme *clearmat*, *gauss1* et *gauss2*.

```
getVarInfo("linalg2")
        [clearmat  "FUNC"  "LibPub "]
        [gauss1    "PRGM"  "LibPriv "]
        [gauss2    "FUNC"  "LibPub "]
libShortcut("linalg2","la")
                {la.clearmat,la.gauss2}
libShortcut("linalg2","la",1)
        {la.clearmat,la.gauss1,la.gauss2}
```

LinRegBx Catalogue >

LinRegBx *X*,*Y*[,[*Fréq*][**,***Catégorie*,*Inclure*]]

Effectue l'ajustement linéairey = a+b ·xsur les listes *X* et *Y* en utilisant la fréquence *Fréq*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes, à l'exception de *Inclure*.

X et *Y* sont des listes de variables indépendantes et dépendantes.

Fréq est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *Fréq* correspond à une fréquence d'occurrence pour chaque couple *X* et *Y*. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Catégorie est une liste de codes de catégories pour les couples *X* et *Y* correspondants.

Inclure est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.RegEqn	Équation d'ajustement : a+b ·x
stat.a, stat.b	Coefficients d'ajustement
stat.r^2	Coefficient de détermination
stat.r	Coefficient de corrélation
stat.Resid	Valeurs résiduelles de l'ajustement
stat.XReg	Liste des points de données de la liste *Liste X* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.YReg	Liste des points de données de la liste *Liste Y* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.FreqReg	Liste des fréquences correspondant à *stat.XReg* et *stat.YReg*

LinRegMx *X*,*Y*[,[*Fréq*][,*Catégorie*,*Inclure*]]

Effectue l'ajustement linéaire y = m ·x+b sur les listes *X* et *Y* en utilisant la fréquence *Fréq*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes, à l'exception de *Inclure*.

X et *Y* sont des listes de variables indépendantes et dépendantes.

Fréq est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *Fréq* correspond à une fréquence d'occurrence pour chaque couple *X* et *Y*. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Catégorie est une liste de codes de catégories pour les couples *X* et *Y* correspondants.

Inclure est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	**Description**
stat.RegEqn	Équation d'ajustement : m ·x+b
stat.m, stat.b	Coefficients d'ajustement
stat.r^2	Coefficient de détermination
stat.r	Coefficient de corrélation
stat.Resid	Valeurs résiduelles de l'ajustement
stat.XReg	Liste des points de données de la liste *Liste X* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*

Variable de sortie	Description
stat.YReg	Liste des points de données de la liste *Liste Y* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.FreqReg	Liste des fréquences correspondant à *stat.XReg* et *stat.YReg*

LinRegtIntervals

Catalogue >

LinRegtIntervals *X*,*Y*[,*F*[,**0**[,*NivC*]]]

Pente. Calcule un intervalle de confiance de niveau C pour la pente.

LinRegtIntervals *X*,*Y*[,*F*[,**1**,*Xval*[,*NivC*]]]

Réponse. Calcule une valeur y prévue, un intervalle de prévision de niveau C pour une seule observation et un intervalle de confiance de niveau C pour la réponse moyenne.

Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes.

X et *Y* sont des listes de variables indépendantes et dépendantes.

F est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *F* spécifie la fréquence d'occurrence pour chaque couple *X* et *Y*. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.RegEqn	Équation d'ajustement : a+b ·x
stat.a, stat.b	Coefficients d'ajustement
stat.df	Degrés de liberté

Variable de sortie	Description
stat.r^2	Coefficient de détermination
stat.r	Coefficient de corrélation
stat.Resid	Valeurs résiduelles de l'ajustement

Pour les intervalles de type Slope uniquement

Variable de sortie	Description
[stat.CLower, stat.CUpper]	Intervalle de confiance de pente
stat.ME	Marge d'erreur de l'intervalle de confiance
stat.SESlope	Erreur type de pente
stat.s	Erreur type de ligne

Pour les intervalles de type Response uniquement

Variable de sortie	Description
[stat.CLower, stat.CUpper]	Intervalle de confiance pour une réponse moyenne
stat.ME	Marge d'erreur de l'intervalle de confiance
stat.SE	Erreur type de réponse moyenne
[stat.LowerPred, stat.UpperPred]	Intervalle de prévision pour une observation simple
stat.MEPred	Marge d'erreur de l'intervalle de prévision
stat.SEPred	Erreur type de prévision
stat.$\hat{y}$	a + b ·ValX

LinRegtTest

Catalogue >

LinRegtTest X,Y[,$Fréq$[,$Hypoth$]]

Effectue l'ajustement linéaire sur les listes X et Y et un t-test sur la valeur de la pente β et le coefficient de corrélation ρ pour l'équation y=α+βx. Il teste l'hypothèse nulle $_H$0 :β=0 (équivalent, ρ=0) par rapport à l'une des trois hypothèses.

Toutes les listes doivent comporter le même nombre de lignes.

X et *Y* sont des listes de variables indépendantes et dépendantes.

Fréq est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *Fréq* correspond à une fréquence d'occurrence pour chaque couple *X* et *Y*. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Hypoth est une valeur facultative qui spécifie une des trois hypothèses par rapport à laquelle l'hypothèse nulle (H_0 :$\beta=\rho=0$) est testée.

Pour H_a : $\beta\neq0$ et $\rho\neq0$ (par défaut), définissez *Hypoth*=0

Pour H_a : $\beta<0$ et $\rho<0$, définissez *Hypoth*<0

Pour H_a : $\beta>0$ et $\rho>0$, définissez *Hypoth*>0

Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	**Description**
stat.RegEqn	Équation d'ajustement : a + b ·x
stat.t	*t*-Statistique pour le test de signification
stat.PVal	Plus petit seuil de signification permettant de rejeter l'hypothèse nulle
stat.df	Degrés de liberté
stat.a, stat.b	Coefficients d'ajustement
stat.s	Erreur type de ligne
stat.SESlope	Erreur type de pente
stat.r^2	Coefficient de détermination
stat.r	Coefficient de corrélation
stat.Resid	Valeurs résiduelles de l'ajustement

linSolve() Catalogue >

linSolve(*SystèmÉqLin*, *Var1*, *Var2*, *...***)⇒***liste*

linSolve(*ÉqLin1* **and** *ÉqLin2* **and** *...*, *Var1*, *Var2*, *...***)⇒***liste*

linSolve({*ÉqLin1*, *ÉqLin2*, *...***}**, *Var1*, *Var2*, *...***)** ⇒*liste*

linSolve(*SystèmÉqLin*, **{***Var1*, *Var2*, *...***})** ⇒*liste*

linSolve(*ÉqLin1* **and** *ÉqLin2* **and** *...*, **{***Var1*, *Var2*, *...***})**⇒*liste*

linSolve({*ÉqLin1*, *ÉqLin2*, *...***}**, **{***Var1*, *Var2*, *...***})** ⇒*liste*

Affiche une liste de solutions pour les variables *Var1*, *Var2*, *etc*.

Le premier argument doit être évalué à un système d'équations linéaires ou à une seule équation linéaire. Si tel n'est pas le cas, une erreur d'argument se produit.

Par exemple, le calcul de linSolve(x=1 et x=2,x) génère le résultat “Erreur d'argument”.

$\text{linSolve}\left(\begin{Bmatrix}2\cdot x+4\cdot y=3\\5\cdot x-3\cdot y=7\end{Bmatrix},\{x,y\}\right)\quad\left\{\frac{37}{26},\frac{1}{26}\right\}$

$\text{linSolve}\left(\begin{Bmatrix}2\cdot x=3\\5\cdot x-3\cdot y=7\end{Bmatrix},\{x,y\}\right)\quad\left\{\frac{3}{2},\frac{1}{6}\right\}$

$\text{linSolve}\left(\begin{Bmatrix}apple+4\cdot pear=23\\5\cdot apple-pear=17\end{Bmatrix},\{apple,pear\}\right)\quad\left\{\frac{13}{3},\frac{14}{3}\right\}$

$\text{linSolve}\left(\begin{Bmatrix}apple\cdot 4+\frac{pear}{3}=14\\-apple+pear=6\end{Bmatrix},\{apple,pear\}\right)\quad\left\{\frac{36}{13},\frac{114}{13}\right\}$

Δlist() Catalogue >

Δlist(*Liste1***)**⇒*liste*

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **`deltaList(`**...**`)`**.

Donne la liste des différences entre les éléments consécutifs de *Liste1*. Chaque élément de *Liste1* est soustrait de l'élément suivant de *Liste1*. Le résultat comporte toujours un élément de moins que la liste *Liste1* initiale.

$\Delta\text{List}\left(\{20,30,45,70\}\right)\quad\{10,15,25\}$

list▸mat() Catalogue >

list▸mat(*Liste* **[,** *élémentsParLigne***])⇒***matrice*

Donne une matrice construite ligne par ligne à partir des éléments de *Liste*.

Si *élémentsParLigne* est spécifié, donne le nombre d'éléments par ligne. La valeur par défaut correspond au nombre d'éléments de *Liste* (une ligne).

Si *Liste* ne comporte pas assez d'éléments pour la matrice, on complète par zéros.

Remarque : vous pouvez insérer cette fonction à partir du clavier de l'ordinateur en entrant **`list@>mat`** **(...)** .

$\text{list▸mat}(\{1,2,3\})$ $\begin{bmatrix}1 & 2 & 3\end{bmatrix}$

$\text{list▸mat}(\{1,2,3,4,5\},2)$ $\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 0\end{bmatrix}$

ln() Touches ctrl ex

ln(*Liste1***)⇒***liste*

Donne le logarithme népérien de l'argument.

Dans le cas d'une liste, donne les logarithmes népériens de tous les éléments de celle-ci.

$\ln(2.)$ 0.693147

En mode Format complexe Réel :

$\ln(\{-3,1.2,5\})$

"Error: Non−real calculation"

ln(*matriceCarrée1***)⇒***matriceCarrée*

Donne le logarithme népérien de la matrice *matriceCarrée1*. Ce calcul est différent du calcul du logarithme népérien de chaque élément. Pour plus d'informations sur la méthode de calcul, reportezvous à **cos()**.

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

En mode Format complexe Rectangulaire :

En mode Angle en radians et en mode Format complexe Rectangulaire :

$$\ln\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix}$$

$$\begin{bmatrix}1.83145+1.73485\cdot i & 0.009193-1.49086\\ 0.448761-0.725533\cdot i & 1.06491+0.623491\\ -0.266891-2.08316\cdot i & 1.12436+1.79018\cdot\end{bmatrix}$$

Pour afficher le résultat entier, appuyez sur ▲, puis utilisez les touches ◄ et ► pour déplacer le curseur.

LnReg *X*, *Y*[, [*Fréq*] [, *Catégorie*, *Inclure*]]

Effectue l'ajustement logarithmique y = a+b ·ln(x) sur les listes *X* et *Y* en utilisant la fréquence *Fréq*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes, à l'exception de *Inclure*.

X et *Y* sont des listes de variables indépendantes et dépendantes.

Fréq est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *Fréq* correspond à une fréquence d'occurrence pour chaque couple *X* et *Y*. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Catégorie est une liste de codes de catégories pour les couples *X* et *Y* correspondants.

Inclure est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.RegEqn	Équation d'ajustement : a+b ·ln(x)
stat.a, stat.b	Coefficients d'ajustement
stat.r^2	Coefficient de détermination linéaire pour les données transformées
stat.r	Coefficient de corrélation pour les données transformées (ln(x), y)
stat.Resid	Valeurs résiduelles associées au modèle logarithmique
stat.ResidTrans	Valeurs résiduelles associées à l'ajustement linéaire des données transformées
stat.XReg	Liste des points de données de la liste *Liste X* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*

Variable de sortie	Description
stat.YReg	Liste des points de données de la liste *Liste Y* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.FreqReg	Liste des fréquences correspondant à *stat.XReg* et *stat.YReg*

Local

Catalogue >

Local *Var1*[**,** *Var2*] [**,** *Var3*] ...

Déclare les variables *vars* spécifiées comme variables locales. Ces variables existent seulement lors du calcul d'une fonction et sont supprimées une fois l'exécution de la fonction terminée.

Remarque : les variables locales contribuent à libérer de la mémoire dans la mesure où leur existence est temporaire. De même, elle n'interfère en rien avec les valeurs des variables globales existantes. Les variables locales s'utilisent dans les boucles **For** et pour enregistrer temporairement des valeurs dans les fonctions de plusieurs lignes dans la mesure où les modifications sur les variables globales ne sont pas autorisées dans une fonction.

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

```
Define rollcount()=Func
                    Local i
                    1→i
                    Loop
                    If randInt(1,6)=randInt(1,6)
                    Goto end
                    i+1→i
                    EndLoop
                    Lbl end
                    Return i
                    EndFunc
                                                Done
rollcount()                                       16
rollcount()                                        3
```

Lock

Lock *Var1* [**,** *Var2*] [**,** *Var3*] ...

Lock *Var***.**

Verrouille les variables ou les groupes de variables spécifiés. Les variables verrouillées ne peuvent être ni modifiées ni supprimées.

Vous ne pouvez pas verrouiller ou déverrouiller la variable système *Ans*, de même que vous ne pouvez pas verrouiller les groupes de variables système *stat*. ou *tvm*.

Remarque : La commande **Verrouiller** (**Lock**) efface le contenu de l'historique Annuler/Rétablir lorsqu'elle est appliquée à des variables non verrouillées.

Voir **unLock, page 174** et **getLockInfo(), page 69.**

Catalogue >

$a:=65$	65
Lock a	*Done*
getLockInfo(a)	1
$a:=75$	"Error: Variable is locked."
DelVar a	"Error: Variable is locked."
Unlock a	*Done*
$a:=75$	75
DelVar a	*Done*

log()

Touches ctrl 10^x

Remarque : voir aussi **Modèle Logarithme**, page 2.

Si *Expr2* est omis, la valeur de base 10 par défaut est utilisée.

$\log_{10}(2.)$	0.30103
$\log_{4}(2.)$	0.5
$\log_{3}(10)-\log_{3}(5)$	$\log_{3}(2)$

En mode Format complexe Réel :

En mode Format complexe Rectangulaire :

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

Si l'argument de base est omis, la valeur de base 10 par défaut est utilisée.

En mode Angle en radians et en mode Format complexe Rectangulaire :

$$\log_{10}\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix}$$

$$\begin{bmatrix}0.795387+0.753438\cdot i & 0.003993-0.6474\\ 0.194895-0.315095\cdot i & 0.462485+0.2707\\ -0.115909-0.904706\cdot i & 0.488304+0.7774\end{bmatrix}$$

log()

Touches ctrl 10^x

Pour afficher le résultat entier, appuyez sur ▲, puis utilisez les touches ◀ et ▶ pour déplacer le curseur.

Logistic

Catalogue >

Logistic *X*, *Y*[, [*Fréq*] [, *Catégorie*, *Inclure*]]

Effectue l'ajustement logistiquey = (c/(1+a ·e^-bx))sur les listes *X* et *Y* en utilisant la fréquence *Fréq*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes, à l'exception de *Inclure*.

X et *Y* sont des listes de variables indépendantes et dépendantes.

Fréq est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *Fréq* correspond à une fréquence d'occurrence pour chaque couple *X* et *Y*. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Catégorie est une liste de codes de catégories pour les couples *X* et *Y* correspondants.

Inclure est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.RegEqn	Équation d'ajustement : c/(1+a ·e^-bx)
stat.a, stat.b, stat.c	Coefficients d'ajustement
stat.Resid	Valeurs résiduelles de l'ajustement

Variable de sortie	Description
stat.XReg	Liste des points de données de la liste *Liste X* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.YReg	Liste des points de données de la liste *Liste Y* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.FreqReg	Liste des fréquences correspondant à *stat.XReg* et *stat.YReg*

LogisticD

Catalogue >

LogisticD *X*, *Y* [, [*Itérations*], [*Fréq*] [, *Catégorie*, *Inclure*]]

Effectue l'ajustement logistique y = (c/(1+a ·e^{-bx})+d) sur les listes *X* et *Y* en utilisant la fréquence *Fréq* et un nombre spécifique d'*Itérations*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes, à l'exception de *Inclure*.

X et *Y* sont des listes de variables indépendantes et dépendantes.

L'argument facultatif Itérations spécifie le nombre maximum d'itérations utilisées lors de ce calcul. Si Itérations est omis, la valeur par défaut 64 est utilisée. On obtient généralement une meilleure précision en choisissant une valeur élevée, mais cela augmente également le temps de calcul, et vice versa.

Fréq est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *Fréq* correspond à une fréquence d'occurrence pour chaque couple *X* et *Y*. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Catégorie est une liste de codes de catégories pour les couples *X* et *Y* correspondants.

Inclure est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.RegEqn	Équation d'ajustement : $c/(1+a \cdot e^{-bx})+d$)
stat.a, stat.b, stat.c, stat.d	Coefficients d'ajustement
stat.Resid	Valeurs résiduelles de l'ajustement
stat.XReg	Liste des points de données de la liste *Liste X* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.YReg	Liste des points de données de la liste *Liste Y* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.FreqReg	Liste des fréquences correspondant à *stat.XReg* et *stat.YReg*

Loop

Catalogue >

Loop
Bloc
EndLoop

Exécute de façon itérative les instructions de *Bloc*. Notez que la boucle se répète indéfiniment, jusqu'à l'exécution d'une instruction **Goto** ou **Exit** à l'intérieur du *Bloc*.

Bloc correspond à une série d'instructions, séparées par un « : ».

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

```
Define rollcount()=Func
                   Local i
                   1→i
                   Loop
                   If randInt(1,6)=randInt(1,6)
                   Goto end
                   i+1→i
                   EndLoop
                   Lbl end
                   Return i
                   EndFunc
                                              Done
rollcount()                                    16
rollcount()                                     3
```

LU *Matrice*, *lMatrice*, *uMatrice*, *pMatrice[,Tol]*

Calcule la décomposition LU (lower-upper) de Doolittle d'une matrice réelle ou complexe. La matrice triangulaire inférieure est stockée dans *lMatrice*, la matrice triangulaire supérieure dans *uMatrice* et la matrice de permutation (qui décrit les échange de lignes exécutés pendant le calcul) dans *pMatrice*.

lMatrice · *uMatrice* = *pMatrice* · *matrice*

L'argument facultatif Tol permet de considérer comme nul tout élément de la matrice dont la valeur absolue est inférieure à *Tol*. Cet argument n'est utilisé que si la matrice contient des nombres en virgule flottante et ne contient pas de variables symbolique sans valeur affectée. Dans le cas contraire, *Tol* est ignoré.

- Si vous utilisez ctrl enter ou définissez le mode **Auto ou Approché (Approximate)** sur Approché (Approximate), les calculs sont exécutés en virgule flottante.
- Si *Tol* est omis ou inutilisé, la tolérance par défaut est calculée comme suit : 5E-14 ·max(dim(*Matrice*)) ·rowNorm (*Matrice*)

L'algorithme de factorisation **LU** utilise la méthode du Pivot partiel avec échanges de lignes.

$$\begin{bmatrix}6&12&18\\5&14&31\\3&8&18\end{bmatrix}\rightarrow m1 \qquad \begin{bmatrix}6&12&18\\5&14&31\\3&8&18\end{bmatrix}$$

LU *m1*,*lower*,*upper*,*perm* — *Done*

lower $\begin{bmatrix}1&0&0\\\frac{5}{6}&1&0\\\frac{1}{2}&\frac{1}{2}&1\end{bmatrix}$

upper $\begin{bmatrix}6&12&18\\0&4&16\\0&0&1\end{bmatrix}$

perm $\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$

M

mat▸list(*Matrice***)⇒***liste*

Donne la liste obtenue en copiant les éléments de *Matrice* ligne par ligne.

$\text{mat}\blacktriangleright\text{list}\left(\begin{bmatrix}1&2&3\end{bmatrix}\right)$ $\{1,2,3\}$

$\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}\rightarrow m1$ $\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}$

$\text{mat}\blacktriangleright\text{list}(m1)$ $\{1,2,3,4,5,6\}$

mat▸list() Catalogue >

Remarque : vous pouvez insérer cette fonction à partir du clavier de l'ordinateur en entrant **mat@>list (...)**.

max() Catalogue >

max(*Liste1***, *Liste2*)⇒*liste***

max(*Matrice1***, *Matrice2*)⇒*matrice***

Donne le maximum des deux arguments. Si les arguments sont deux listes ou matrices, donne la liste ou la matrice formée de la valeur maximale de chaque paire d'éléments correspondante.

$\max(2.3,1.4)$ → 2.3

$\max(\{1,2\},\{-4,3\})$ → $\{1,3\}$

max(*Liste***)⇒*expression***

Donne l'élément maximal de *liste*.

$\max(\{0,1,-7,1.3,0.5\})$ → 1.3

max(*Matrice1***)⇒*matrice***

Donne un vecteur ligne contenant l'élément maximal de chaque colonne de la matrice *Matrice1*.

$\max\begin{pmatrix}1 & -3 & 7\\ -4 & 0 & 0.3\end{pmatrix}$ → $\begin{bmatrix}1 & 0 & 7\end{bmatrix}$

Les éléments vides sont ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

Remarque : voir aussi **min()**.

mean() Catalogue >

mean(*Liste*[**,** *listeFréq*]**)⇒*expression***

Donne la moyenne des éléments de *Liste*.

$\text{mean}(\{0.2,0,1,-0.3,0.4\})$ → 0.26

$\text{mean}(\{1,2,3\},\{3,2,1\})$ → $\frac{5}{3}$

Chaque élément de la liste *listeFréq* totalise le nombre d'occurrences de l'élément correspondant de *Liste*.

mean(*Matrice1*[**,** *matriceFréq*]**)** ⇒*matrice*

En mode Format Vecteur Rectangulaire :

Donne un vecteur ligne des moyennes de toutes les colonnes de *Matrice1*.

mean() Catalogue >

Chaque élément de *matriceFréq* totalise le nombre d'occurrences de l'élément correspondant de *Matrice1*.

Les éléments vides sont ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

$\text{mean}\left(\begin{bmatrix}0.2 & 0\\ -1 & 3\\ 0.4 & -0.5\end{bmatrix}\right)$ $\quad [-0.133333 \quad 0.833333]$

$\text{mean}\left(\begin{bmatrix}\frac{1}{5} & 0\\ -1 & 3\\ \frac{2}{5} & \frac{-1}{2}\end{bmatrix}\right)$ $\quad \begin{bmatrix}\frac{-2}{15} & \frac{5}{6}\end{bmatrix}$

$\text{mean}\left(\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix},\begin{bmatrix}5 & 3\\ 4 & 1\\ 6 & 2\end{bmatrix}\right)$ $\quad \begin{bmatrix}\frac{47}{15} & \frac{11}{3}\end{bmatrix}$

median() Catalogue >

median(*Liste***[, ***listeFréq***])⇒***expression***

Donne la médiane des éléments de *Liste*.

Chaque élément de la liste *listeFréq* totalise le nombre d'occurrences de l'élément correspondant de *Liste*.

$\text{median}(\{0.2,0,1,-0.3,0.4\})$ $\quad 0.2$

median(*Matrice1***[, ***matriceFréq***])⇒***matrice***

Donne un vecteur ligne contenant les médianes des colonnes de *Matrice1*.

Chaque élément de *matriceFréq* totalise le nombre d'occurrences consécutives de l'élément correspondant de *Matrice1*.

$\text{median}\left(\begin{bmatrix}0.2 & 0\\ 1 & -0.3\\ 0.4 & -0.5\end{bmatrix}\right)$ $\quad [0.4 \quad -0.3]$

Remarques :

- tous les éléments de la liste ou de la matrice doivent correspondre à des valeurs numériques.
- Les éléments vides de la liste ou de la matrice sont ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

MedMed Catalogue >

MedMed *X*,*Y* [, *Fréq*] [, *Catégorie*, *Inclure*]]

Calcule la ligne Med-Medy = (m ·x+b)sur les listes *X* et *Y* en utilisant la fréquence *Fréq*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes, à l'exception de *Inclure*.

X et *Y* sont des listes de variables indépendantes et dépendantes.

Fréq est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *Fréq* correspond à une fréquence d'occurrence pour chaque couple *X* et *Y*. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Catégorie est une liste de codes de catégories pour les couples *X* et *Y* correspondants..

Inclure est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.RegEqn	Équation de ligne Med-Med : m ·x+b
stat.m, stat.b	Coefficient de modèle
stat.Resid	Valeurs résiduelles de la ligne Med-Med
stat.XReg	Liste des points de données de la liste *Liste X* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.YReg	Liste des points de données de la liste *Liste Y* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.FreqReg	Liste des fréquences correspondant à *stat.XReg* et *stat.YReg*

mid() Catalogue >

mid(*chaîneSrce***,** *Début***[,** *Nbre***])⇒***chaîne*

Donne la portion de chaîne de *Nbre* de caractères extraite de la chaîne *chaîneSrce*, en commençant au numéro de caractère *Début*.

Si *Nbre* est omis ou s'il dépasse le nombre de caractères de la chaîne *chaîneSrce*, on obtient tous les caractères de *chaîneSrce*, compris entre le numéro de caractère *Début* et le dernier caractère.

Nbre doit être ≥ 0. Si *Nbre* = 0, on obtient une chaîne vide.

```
mid("Hello there",2)                "ello there"
mid("Hello there",7,3)              "the"
mid("Hello there",1,5)              "Hello"
mid("Hello there",1,0)              ""
```

mid(*listeSource***,** *Début* **[,** *Nbre***])⇒***liste*

Donne la liste de *Nbre* d'éléments extraits de *listeSource*, en commençant à l'élément numéro *Début*.

Si *Nbre* est omis ou s'il dépasse le nombre d'éléments de la liste *listeSource*, on obtient tous les éléments de *listeSource*, compris entre l'élément numéro *Début* et le dernier élément.

Nbre doit être ≥ 0. Si *Nbre* = 0, on obtient une liste vide.

```
mid({9,8,7,6},3)                    {7,6}
mid({9,8,7,6},2,2)                  {8,7}
mid({9,8,7,6},1,2)                  {9,8}
mid({9,8,7,6},1,0)                  {}
```

mid(*listeChaînesSource***,** *Début***[,** *Nbre***])⇒***liste*

Donne la liste de *Nbre* de chaînes extraites de la liste *listeChaînesSource*, en commençant par l'élément numéro *Début*.

```
mid({"A","B","C","D"},2,2)
                                    {"B","C"}
```

min() Catalogue >

min(*Liste1, Liste2***)⇒***liste*

min(*Matrice1, Matrice2***)⇒***matrice*

```
min(2.3,1.4)                        1.4
min({1,2},{-4,3})                   {-4,2}
```

min() Catalogue >

Donne le minimum des deux arguments. Si les arguments sont deux listes ou matrices, donne la liste ou la matrice formée de la valeur minimale de chaque paire d'éléments correspondante.

min(*Liste***)⇒***expression*

Donne l'élément minimal de *Liste*.

min(*Matrice1***)⇒***matrice*

Donne un vecteur ligne contenant l'élément minimal de chaque colonne de la matrice *Matrice1*.

Remarque : voir aussi **max()**.

$\min(\{0,1,-7,1.3,0.5\})$ -7

$\min\left(\begin{bmatrix} 1 & -3 & 7 \\ -4 & 0 & 0.3 \end{bmatrix}\right)$ $\begin{bmatrix} -4 & -3 & 0.3 \end{bmatrix}$

mirr() Catalogue >

mirr
(
tauxFinancement
,*tauxRéinvestissement***,***MT0***,***ListeMT*
[,*FréqMT***]⇒***expression*

Fonction financière permettant d'obtenir le taux interne de rentabilité modifié d'un investissement.

tauxFinancement correspond au taux d'intérêt que vous payez sur les montants de mouvements de trésorerie.

tauxRéinvestissement est le taux d'intérêt auquel les mouvements de trésorerie sont réinvestis.

MT0 correspond au mouvement de trésorerie initial à l'heure 0 ; il doit s'agir d'un nombre réel.

Liste MT est une liste des montants de mouvements de trésorerie après le mouvement de trésorerie initial MT0.

$list1:=\{6000,-8000,2000,-3000\}$ $\{6000,-8000,2000,-3000\}$

$list2:=\{2,2,2,1\}$ $\{2,2,2,1\}$

$\text{mirr}(4.65,12,5000,list1,list2)$ 13.41608607

mirr() Catalogue >

FréqMT est une liste facultative dans laquelle chaque élément indique la fréquence d'occurrence d'un montant de mouvement de trésorerie groupé (consécutif), correspondant à l'élément de *ListeMT*. La valeur par défaut est 1 ; si vous saisissez des valeurs, elles doivent être des entiers positifs < 10 000.

Remarque : voir également **irr()**, page 79.

mod() Catalogue >

mod(*Liste1***,** *List2***)⇒***liste*

mod(*Matrice1***,** *Matrice2***)⇒***matrice*

Donne le premier argument modulo le deuxième argument, défini par les identités suivantes :

mod(x,0) = x

mod(x,y) = x -Ïy floor(x/y)

Lorsque le deuxième argument correspond à une valeur non nulle, le résultat est de période dans cet argument. Le résultat est soit zéro soit une valeur de même signe que le deuxième argument.

Si les arguments sont deux listes ou deux matrices, on obtient une liste ou une matrice contenant la congruence de chaque paire d'éléments correspondante.

Remarque : voir aussi **remain()**, page 134

$\text{mod}(7,0)$	7
$\text{mod}(7,3)$	1
$\text{mod}(-7,3)$	2
$\text{mod}(7,-3)$	-2
$\text{mod}(-7,-3)$	-1
$\text{mod}(\{12,-14,16\},\{9,7,-5\})$	$\{3,0,-4\}$

mRow() Catalogue >

$\text{mRow}\left(\frac{-1}{3},\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix},2\right)$	$\begin{bmatrix}1 & 2\\ -1 & \frac{-4}{3}\end{bmatrix}$

mRowAdd()

Catalogue >

Donne une copie de *Matrice1* obtenue en remplaçant chaque élément de la ligne *Index2* de *Matrice1* par :

Index2

MultReg

Catalogue >

MultReg *Y*, *X1*[**,***X2*[**,***X3*,**…**[**,***X10*]]]

Calcule la régression linéaire multiple de la liste *Y* sur les listes *X1*, *X2*, …, *X10*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.RegEqn	Équation d'ajustement : b0+b1 ·x1+b2 ·x2+ ...
stat.b0, stat.b1, ...	Coefficients d'ajustement
stat.R^2	Coefficient de détermination multiple
stat.$\hat{y}$Liste	$\hat{y}$Liste = b0+b1 ·x1+ ...
stat.Resid	Valeurs résiduelles de l'ajustement

MultRegIntervals

Catalogue >

MultRegIntervals *Y*, *X1*[**,***X2*[**,***X3*,**…** [**,***X10*]]]**,***listeValX*[**,***CLevel*]

Calcule une valeur y prévue, un intervalle de prévision de niveau C pour une seule observation et un intervalle de confiance de niveau C pour la réponse moyenne.

Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes.

MultRegIntervals

Catalogue >

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.RegEqn	Équation d'ajustement : b0+b1 ·x1+b2 ·x2+ ...
stat.$\hat{y}$	Prévision d'un point : $\hat{y}$ = b0 + b1 · xl + ... pour *listeValX*
stat.dfError	Degrés de liberté des erreurs
stat.CLower, stat.CUpper	Intervalle de confiance pour une réponse moyenne
stat.ME	Marge d'erreur de l'intervalle de confiance
stat.SE	Erreur type de réponse moyenne
stat.LowerPred, stat.UpperrPred	Intervalle de prévision pour une observation simple
stat.MEPred	Marge d'erreur de l'intervalle de prévision
stat.SEPred	Erreur type de prévision
stat.bList	Liste de coefficients de régression, {b0,b1,b2,...}
stat.Resid	Valeurs résiduelles de l'ajustement

MultRegTests

Catalogue >

MultRegTests *Y*, *X1*[,*X2*[,*X3*,...[,*X10*]]]

Le test de régression linéaire multiple calcule une régression linéaire multiple sur les données et donne les statistiques du *F*-test et du *t*-test globaux pour les coefficients.

Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Sorties

Variable de sortie	Description
stat.RegEqn	Équation d'ajustement : b0+b1 ·x1+b2 ·x2+ ...

Variable de sortie	Description
stat.F	Statistique du F-test global
stat.PVal	Valeur P associée à l'analyse statistique F globale
stat.R^2	Coefficient de détermination multiple
stat.AdjR^2	Coefficient ajusté de détermination multiple
stat.s	Écart-type de l'erreur
stat.DW	Statistique de Durbin-Watson ; sert à déterminer si la corrélation automatique de premier ordre est présente dans le modèle
stat.dfReg	Degrés de liberté de la régression
stat.SSReg	Somme des carrés de la régression
stat.MSReg	Moyenne des carrés de la régression
stat.dfError	Degrés de liberté des erreurs
stat.SSError	Somme des carrés des erreurs
stat.MSError	Moyenne des carrés des erreurs
stat.bList	{b0,b1,...} Liste de coefficents
stat.tList	Liste des statistiques t pour chaque coefficient dans la liste bList
stat.PList	Liste des valeurs p pour chaque statistique t
stat.SEList	Liste des erreurs type des coefficients de la liste bList
stat.$\hat{y}$Liste	$\hat{y}$Liste = b0+b1 ·x1+ . . .
stat.Resid	Valeurs résiduelles de l'ajustement
stat.sResid	Valeurs résiduelles normalisées ; valeur obtenue en divisant une valeur résiduelle par son écart-type
stat.CookDist	Distance de Cook ; Mesure de l'influence d'une observation basée sur la valeur résiduelle et le levier
stat.Leverage	Mesure de la distance séparant les valeurs de la variable indépendante de leurs valeurs moyennes

nand

touches ctrl =

BooleanExpr1 **nand** *BooleanExpr2*
renvoie *expression booléenne*

BooleanList1 **nand** *BooleanList2*
renvoie *liste booléenne*

BooleanMatrix1 **nand** *BooleanMatrix2*
renvoie *matrice booléenne*

Renvoie la négation d'une opération logique **and** sur les deux arguments. Renvoie true (vrai) ou false (faux) ou une forme simplifiée de l'équation.

Pour les listes et matrices, renvoie le résultat des comparaisons, élément par élément.

$x \geq 3$ and $x \geq 4$	$x \geq 4$
$x \geq 3$ nand $x \geq 4$	$x < 4$

Integer1 **nand** *Integer2*⇒*entier*

Compare les représentations binaires de deux entiers en appliquant une opération **nand**. En interne, les deux entiers sont convertis en nombres binaires 64 bits signés. Lorsque les bits comparés correspondent, le résultat est 0 si dans les deux cas il s'agit d'un bit 1 ; dans les autres cas, le résultat est 1. La valeur donnée représente le résultat des bits et elle est affichée selon le mode de base utilisé.

Les entiers peuvent être entrés dans tout type de base. Pour une entrée binaire ou hexadécimale, vous devez utiliser respectivement le préfixe 0b ou 0h. Tout entier sans préfixe est considéré comme un nombre en écriture décimale (base 10).

3 and 4	0
3 nand 4	-1
{1,2,3} and {3,2,1}	{1,2,1}
{1,2,3} nand {3,2,1}	{-2,-3,-2}

nCr()

Catalogue >

nCr(*Liste1***,** *Liste2***)**⇒*liste*

nCr({5,4,3},{2,4,2})	{10,1,3}

nCr() Catalogue >

Donne une liste de combinaisons basées sur les paires d'éléments correspondantes dans les deux listes. Les arguments doivent être des listes comportant le même nombre d'éléments.

nCr(*Matrice1***,** *Matrice2***)⇒***matrice*

Donne une matrice de combinaisons basées sur les paires d'éléments correspondantes dans les deux matrices. Les arguments doivent être des matrices comportant le même nombre d'éléments.

$$\mathrm{nCr}\left(\begin{bmatrix}6 & 5\\ 4 & 3\end{bmatrix},\begin{bmatrix}2 & 2\\ 2 & 2\end{bmatrix}\right) \qquad \begin{bmatrix}15 & 10\\ 6 & 3\end{bmatrix}$$

nDerivative() Catalogue >

nDerivative(*Expr1***,***Var=Valeur* [**,***Ordre*])**⇒***valeur***

nDerivative(*Expr1***,***Var*[**,***Ordre*]**) |** *Var=Valeur***⇒***valeur*

Affiche la dérivée numérique calculée avec les méthodes de différenciation automatique.

Quand la *valeur* est spécifiée, celle-ci prévaut sur toute affectation de variable ou substitution précédente de type « | » pour la variable.

L'ordre de la dérivée doit être **1** ou **2**.

$$\mathrm{nDerivative}(|x|,x=1) \qquad 1$$

$$\mathrm{nDerivative}(|x|,x)|x=0 \qquad \text{undef}$$

$$\mathrm{nDerivative}(\sqrt{x-1},x)|x=1 \qquad \text{undef}$$

newList() Catalogue >

newList(*nbreÉléments***)⇒***liste*

Donne une liste de dimension *nbreÉléments*. Tous les éléments sont nuls.

$$\mathrm{newList}(4) \qquad \{0,0,0,0\}$$

newMat() Catalogue >

newMat(*nbreLignes***,** *nbreColonnes***)⇒***matrice*

$$\mathrm{newMat}(2,3) \qquad \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}$$

newMat() Catalogue >

Donne une matrice nulle de dimensions *nbreLignes*, *nbreColonnes*.

nfMax() Catalogue >

nfMax(*Expr***,** *Var***)⇒***valeur*

nfMax(*Expr***,** *Var*, *LimitInf***)⇒***valeur*

nfMax(*Expr***,** *Var*, *LimitInf*, *LimitSup***)⇒***valeur*

nfMax(*Expr*, *Var***) | *LimitInf*≤*Var*≤*LimitSup*⇒***valeur***

Donne la valeur numérique possible de la variable *Var* au point où le maximum local de *Expr* survient.

Si *LimitInf* et *LimitSup* sont spécifiés, la fonction recherche le maximum local dans l'intervalle fermé [*LimitInf*,*LimitSup*].

$\text{nfMax}\left(-x^2-2\cdot x-1,x\right)$ $-1.$

$\text{nfMax}\left(0.5\cdot x^3-x-2,x,-5,5\right)$ $5.$

nfMin() Catalogue >

nfMin(*Expr***,** *Var***)⇒***valeur*

nfMin(*Expr***,** *Var*, *LimitInf***)⇒***valeur*

nfMin(*Expr***,** *Var*, *LimitInf*, *LimitSup***)⇒***valeur*

nfMin(*Expr*, *Var***) | *LimitInf*≤*Var*≤*LimitSup*⇒***valeur***

Donne la valeur numérique possible de la variable *Var* au point où le minimum local de *Expr* survient.

Si *LimitInf* et *LimitSup* sont spécifiés, la fonction recherche le minimum local dans l'intervalle fermé [*LimitInf*,*LimitSup*].

$\text{nfMin}\left(x^2+2\cdot x+5,x\right)$ $-1.$

$\text{nfMin}\left(0.5\cdot x^3-x-2,x,-5,5\right)$ $-5.$

nInt() Catalogue >

nInt(*Expr1, Var, Borne1, Borne2***)⇒***expression*

$\text{nInt}\left(e^{-x^2},x,-1,1\right)$ 1.49365

Si l'intégrande *Expr1* ne contient pas d'autre variable que *Var* et si *Borne1* et *Borne2* sont des constantes, en +∞ ou en -∞, alors **nInt()** donne le calcul approché de ∫**(***Expr1*, *Var*, *Borne1*, *Borne2***)**. Cette approximation correspond à une moyenne pondérée de certaines valeurs d'échantillon de l'intégrande dans l'intervalle *Borne1*<*Var*<*Borne2*.

L'objectif est d'atteindre une précision de six chiffres significatifs. L'algorithme s'adaptant, met un terme au calcul lorsqu'il semble avoir atteint cet objectif ou lorsqu'il paraît improbable que des échantillons supplémentaires produiront une amélioration notable.

Le message « Précision incertaine » s'affiche lorsque cet objectif ne semble pas atteint.

Il est possible de calculer une intégrale multiple en imbriquant plusieurs appels **nInt()**. Les bornes d'intégration peuvent dépendre des variables d'intégration les plus extérieures.

$\text{nInt}\left(\text{nInt}\left(\frac{e^{-x\cdot y}}{\sqrt{x^2-y^2}},y,-x,x\right),x,0,1\right)$ 3.30423

nom() Catalogue >

nom(*tauxEffectif,CpY***)⇒***valeur*

$\text{nom}(5.90398,12)$ 5.75

Fonction financière permettant de convertir le taux d'intérêt effectif *tauxEffectif* à un taux annuel nominal, *CpY* étant le nombre de périodes de calcul par an.

tauxEffectif doit être un nombre réel et *CpY* doit être un nombre réel > 0.

Remarque : voir également **eff()**, page 46.

nor

touches ctrl =

BooleanExpr1 **nor** *BooleanExpr2* renvoie *expression booléenne*

BooleanList1 **nor** *BooleanList2* renvoie *liste booléenne*

BooleanMatrix1 **nor** *BooleanMatrix2* renvoie *matrice booléenne*

$x\geq 3$ or $x\geq 4$	$x\geq 3$
$x\geq 3$ nor $x\geq 4$	$x<3$

Renvoie la négation d'une opération logique **or** sur les deux arguments. Renvoie true (vrai) ou false (faux) ou une forme simplifiée de l'équation.

Pour les listes et matrices, renvoie le résultat des comparaisons, élément par élément.

Integer1 **nor** *Integer2*⇒*entier*

Compare les représentations binaires de deux entiers en appliquant une opération **nor**. En interne, les deux entiers sont convertis en nombres binaires 64 bits signés. Lorsque les bits comparés correspondent, le résultat est 1 si dans les deux cas il s'agit d'un bit 1 ; dans les autres cas, le résultat est 0. La valeur donnée représente le résultat des bits et elle est affichée selon le mode de base utilisé.

Les entiers peuvent être entrés dans tout type de base. Pour une entrée binaire ou hexadécimale, vous devez utiliser respectivement le préfixe 0b ou 0h. Tout entier sans préfixe est considéré comme un nombre en écriture décimale (base 10).

3 or 4	7
3 nor 4	-8
{1,2,3} or {3,2,1}	{3,2,3}
{1,2,3} nor {3,2,1}	{-4,-3,-4}

norm()

Catalogue > 📖

norm(*Matrice***)**⇒*expression*

norm(*Vecteur***)**⇒*expression*

Donne la norme de Frobenius.

normCdf() Catalogue >

normCdf(*lowBound***,***upBound*[**,**μ[**,**σ]]**)**⇒*nombre* si *lowBound* et *upBound* sont des nombres, *liste* si *lowBound* et *upBound* sont des listes

Calcule la probabilité qu'une variable suivant la loi normale de moyenne (*m*, valeur par défaut =0) et d'écart-type (*sigma*, valeur par défaut = 1) prenne des valeurs entre les bornes *lowBound* et *upBound*.

normPdf() Catalogue >

normPdf(*ValX*[**,**μ[**,**σ]]**)**⇒*nombre* si *ValX* est un nombre, *liste* si *ValX* est une liste

Calcule la densité de probabilité de la loi normale à la valeur *ValX* spécifiée pour les paramètres μ et σ.

not Catalogue >

Donne true (vrai) ou false (faux) ou une forme simplifiée de l'argument.

not *Entier1*⇒*entier*

Donne le complément à 1 d'un entier. En interne, *Entier1* est converti en nombre binaire 64 bits signé. La valeur de chaque bit est inversée (0 devient 1, et vice versa) pour le complément à 1. Le résultat est affiché en fonction du mode Base utilisé.

Les entiers de tout type de base sont admis. Pour une entrée binaire ou hexadécimale, vous devez utiliser respectivement le préfixe 0b ou 0h. Tout entier sans préfixe est considéré comme un nombre en écriture décimale (base 10).

Si vous entrez un nombre dont le codage binaire signé dépasse 64 bits, il est ramené à l'aide d'une congruence dans la plage appropriée. Pour de plus amples informations, voir ▶**Base2**, page 17.

En mode base Hex :

Important : utilisez le chiffre zéro et pas la lettre O.

not 0h7AC36	0hFFFFFFFFFF853C9

En mode base Bin :

0b100101▶Base10	37
not 0b100101 0b1111111111111111111111111111111▶	
not 0b100101▶Base10	−38

Pour afficher le résultat entier, appuyez sur ▲, puis utilisez les touches ◀ et ▶ pour déplacer le curseur.

Remarque : une entrée binaire peut comporter jusqu'à 64 chiffres (sans compter le préfixe 0b) ; une entrée hexadécimale jusqu'à 16 chiffres.

nPr() Catalogue >

nPr(*Liste1***, *Liste2*)⇒*liste***

Donne une liste de permutations basées sur les paires d'éléments correspondantes dans les deux listes. Les arguments doivent être des listes comportant le même nombre d'éléments.

$\text{nPr}(\{5,4,3\},\{2,4,2\})$ $\{20,24,6\}$

nPr(*Matrice1***, *Matrice2*)⇒*matrice***

Donne une matrice de permutations basées sur les paires d'éléments correspondantes dans les deux matrices. Les arguments doivent être des matrices comportant le même nombre d'éléments.

$\text{nPr}\left(\begin{bmatrix}6 & 5\\ 4 & 3\end{bmatrix},\begin{bmatrix}2 & 2\\ 2 & 2\end{bmatrix}\right)$ $\begin{bmatrix}30 & 20\\ 12 & 6\end{bmatrix}$

npv() Catalogue >

npv(*tauxIntérêt***,*MTO*,*ListeMT* [,*FréqMT*])**

Fonction financière permettant de calculer la valeur actuelle nette ; la somme des valeurs actuelles des mouvements d'entrée et de sortie de fonds. Un résultat positif pour NPV indique un investissement rentable.

tauxIntérêt est le taux à appliquer pour l'escompte des mouvements de trésorerie (taux de l'argent) sur une période donnée.

MT0 correspond au mouvement de trésorerie initial à l'heure 0 ; il doit s'agir d'un nombre réel.

Liste MT est une liste des montants de mouvements de trésorerie après le mouvement de trésorerie initial *MT0*.

$list1:=\{6000,-8000,2000,-3000\}$ $\{6000,-8000,2000,-3000\}$

$list2:=\{2,2,2,1\}$ $\{2,2,2,1\}$

$\text{npv}(10,5000,list1,list2)$ 4769.91

FréqMT est une liste dans laquelle chaque élément indique la fréquence d'occurrence d'un montant de mouvement de trésorerie groupé (consécutif), correspondant à l'élément de *ListeMT*. La valeur par défaut est 1 ; si vous saisissez des valeurs, elles doivent être des entiers positifs < 10 000.

nSolve(*Équation*,*Var* **[=***Condition***])**⇒ *chaîne_nombre ou erreur*

nSolve(*Équation*,*Var* **[=***Condition***],***LimitInf***)** ⇒*chaîne_ nombre ou erreur*

nSolve(*Équation*,*Var* **[=***Condition***],***LimitInf*,*LimitSup***)** ⇒*chaîne_nombre ou erreur*

nSolve(*Équation*,*Var***[=***Condition***]) |** *LimitInf*≤*Var*≤*LimitSup* ⇒*chaîne_ nombre ou erreur*

Recherche de façon itérative une solution numérique réelle approchée pour *Équation* en fonction de sa variable. Spécifiez la variable comme suit :

variable

– ou –

variable = *nombre réel*

Par exemple, x est autorisé, de même que x=3.

$\text{nSolve}(x^2+5\cdot x-25=9,x)$	3.84429
$\text{nSolve}(x^2=4,x=-1)$	-2.
$\text{nSolve}(x^2=4,x=1)$	2.

Remarque : si plusieurs solutions sont possibles, vous pouvez utiliser une condition pour mieux déterminer une solution particulière.

nSolve() Catalogue >

nSolve() tente de déterminer un point où la valeur résiduelle est zéro ou deux points relativement rapprochés où la valeur résiduelle a un signe négatif et où son ordre de grandeur n'est pas excessif. S'il n'y parvient pas en utilisant un nombre réduit de points d'échantillon, la chaîne « Aucune solution n'a été trouvée » s'affiche.

$\text{nSolve}(x^2+5\cdot x-25=9, x) \mid x<0$ $\quad -8.84429$

$\text{nSolve}\left(\frac{(1+r)^{24}-1}{r}=26, r\right) \mid r>0 \text{ and } r<0.25$ $\quad 0.006886$

$\text{nSolve}(x^2=-1, x)$ $\quad$ "No solution found"

O

OneVar Catalogue >

OneVar [**1,**]*X*[**,**[*Fréq*][**,***Catégorie***,***Inclure*]]

OneVar [*n***,**]*X1***,***X2*[*X3*[**,**...[**,***X20*]]]

Effectue le calcul de statistiques à une variable sur un maximum de 20 listes. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes, à l'exception de *Inclure*.

Les arguments X sont des listes de données.

Fréq est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *Fréq* correspond à une fréquence d'occurrence pour chaque valeur *X* correspondante. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Catégorie est une liste de codes numériques de catégories pour les valeurs *X* correspondantes.

Inclure est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

OneVar

Catalogue >

Tout élément vide dans les listes *X*, *Fréq* ou *Catégorie* a un élément vide correspondant dans l'ensemble des listes résultantes. Tout élément vide dans les listes *X1* à *X20* correspond a un élément vide dans l'ensemble des listes résultantes. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

Variable de sortie	Description
stat.$\bar{x}$	Moyenne des valeurs x
stat.Σx	Somme des valeurs x
stat.Σx^2	Somme des valeurs x^2.
stat.sx	Écart-type de l'échantillon de x
stat. σx	Écart-type de la population de x
stat.n	Nombre de points de données
stat.MinX	Minimum des valeurs de x
stat.Q_1X	1er quartile de x
stat.MedianX	Médiane de x
stat.Q_3X	3ème quartile de x
stat.MaxX	Maximum des valeurs de x
stat.SSX	Somme des carrés des écarts par rapport à la moyenne de x

or

Catalogue >

BooleanExpr1 **or** *BooleanExpr2* renvoie *expression booléenne*

BooleanList1 **or** *BooleanList2* renvoie *liste booléenne*

BooleanMatrix1 **or** *BooleanMatrix2* renvoie *matrice booléenne*

Donne true (vrai) ou false (faux) ou une forme simplifiée de l'entrée initiale.

```
Define g(x)=Func                          Done
              If x≤0 or x≥5
              Goto end
              Return x·3
              Lbl end
              EndFunc
g(3)                                      9
g(0)                    A function did not return a value
```

Donne true si la simplification de l'une des deux ou des deux expressions est vraie. Donne false uniquement si la simplification des deux expressions est fausse.

Remarque : voir **xor**.

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

Entier1 **or** *Entier2*⇒*entier*

Compare les représentations binaires de deux entiers réels en appliquant un or bit par bit. En interne, les deux entiers sont convertis en nombres binaires 64 bits signés. Lorsque les bits comparés correspondent, le résultat est 1 si dans les deux cas il s'agit d'un bit 1 ; le résultat est 0 si, dans les deux cas, il s'agit d'un bit 0. La valeur donnée représente le résultat des bits et elle est affichée selon le mode Base utilisé.

Les entiers de tout type de base sont admis. Pour une entrée binaire ou hexadécimale, vous devez utiliser respectivement le préfixe 0b ou 0h. Tout entier sans préfixe est considéré comme un nombre en écriture décimale (base 10).

Si vous entrez un nombre dont le codage binaire signé dépasse 64 bits, il est ramené à l'aide d'une congruence dans la plage appropriée. Pour de plus amples informations, voir ▶**Base2**, page 17.

Remarque : voir **xor.**

En mode base Hex :

0h7AC36 or 0h3D5F	0h7BD7F

Important : utilisez le chiffre zéro et pas la lettre O.

En mode base Bin :

0b100101 or 0b100	0b100101

Remarque : une entrée binaire peut comporter jusqu'à 64 chiffres (sans compter le préfixe 0b) ; une entrée hexadécimale jusqu'à 16 chiffres.

ord() Catalogue >

ord(*Chaîne***)⇒***entier*

ord(*Liste1***)⇒***liste*

Donne le code numérique du premier caractère de la chaîne de caractères *Chaîne* ou une liste des premiers caractères de tous les éléments de la liste.

ord("hello")	104
char(104)	"h"
ord(char(24))	24
ord({"alpha","beta"})	{97,98}

P

P▶Rx() Catalogue >

P▶Rx(*ExprR***, θ***Expr***)⇒***expression*

P▶Rx(*ListeR***, θ***Liste***)⇒***liste*

P▶Rx(*MatriceR***, θ***Matrice***)⇒***matrice*

Donne la valeur de l'abscisse du point de coordonnées polaires (r, θ).

Remarque : l'argument θ est interprété comme une mesure en degrés, en grades ou en radians, suivant le mode Angle utilisé. Si l'argument est une expression, vous pouvez utiliser °, G ou ʳ pour ignorer temporairement le mode Angle sélectionné.

Remarque : vous pouvez insérer cette fonction à partir du clavier de l'ordinateur en entrant **P@>Rx(...)**.

En mode Angle en radians :

P▶Ry() Catalogue >

P▶Ry(*ListeR***, θ***Liste***)⇒***liste*

P▶Ry(*MatriceR***, θ***Matrice***)⇒***matrice*

Donne la valeur de l'ordonnée du point de coordonnées polaires (r, θ).

Remarque : l'argument θ est interprété comme une mesure en degrés, en grades ou en radians, suivant le mode Angle utilisé.

Remarque : vous pouvez insérer cette fonction à partir du clavier de l'ordinateur en entrant **P@>Ry(...)**.

En mode Angle en radians :

PassErr — Catalogue >

PassErr

Passe une erreur au niveau suivant.

Si la variable système *errCode* est zéro, **PassErr** ne fait rien.

L'instruction **Else** du bloc **Try...Else...EndTry** doit utiliser **EffErr** ou **PassErr**. Si vous comptez rectifier ou ignorer l'erreur, sélectionnez **EffErr**. Si vous ne savez pas comment traiter l'erreur, sélectionnez **PassErr** pour la transférer au niveau suivant. S'il n'y a plus d'autre programme de traitement des erreurs **Try...Else...EndTry**, la boîte de dialogue Erreur s'affiche normalement.

Remarque : Voir aussi **ClrErr**, page 24 et **Try**, page 167.

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

Pour obtenir un exemple de **PassErr**, reportez-vous à l'exemple 2 de la commande **Try**, page 167.

piecewise() — Catalogue >

piecewise(*Expr1* [**,** *Condition1* [**,** *Expr2* [**,** *Condition2* [**,** *...*]]]]**)**

Permet de créer des fonctions définies par morceaux sous forme de liste. Il est également possible de créer des fonctions définies par morceaux en utilisant un modèle.

Remarque : voir aussi **Modèle Fonction définie par morceaux**, page 2.

Define $p(x)=\begin{cases} x, & x>0 \\ \text{undef}, & x\le 0 \end{cases}$	*Done*
$p(1)$	1
$p(-1)$	undef

poissCdf() — Catalogue >

poissCdf(λ**,***lowBound***,***upBound***)**⇒*nombre* si *lowBound* et *upBound* sont des nombres, *liste* si *lowBound* et *upBound* sont des listes

poissCdf(λ**,***upBound***)**(pour P(0≤X

poissCdf() Catalogue >

≤*upBound*)⇒*nombre* si la borne *upBound* est un nombre, liste si la borne *upBound* est une liste

Calcule la probabilité cumulée d'une variable suivant une loi de Poisson de moyenne λ.

Pour P(X ≤ *upBound*), définissez la borne lowBound=0

poissPdf() Catalogue >

poissPdf(λ**,***ValX***)**⇒*nombre* si *ValX* est un nombre, *liste* si *ValX* est une liste

Calcule la probabilité de ValX pour la loi de Poisson de moyenne λ spécifiée.

▸Polar Catalogue >

Vecteur **▸Polar**

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **@>Polar**.

Affiche *vecteur* sous forme polaire [r∠ θ]. Le vecteur doit être un vecteur ligne ou colonne et de dimension 2.

Remarque : ▸Polar est uniquement une instruction d'affichage et non une fonction de conversion. On ne peut l'utiliser qu'à la fin d'une ligne et elle ne modifie pas le contenu du registre *ans*.

Remarque : voir aussi **▸Rect**, page 131.

valeurComplexe **▸Polar**

Affiche *valeurComplexe* sous forme polaire.

- Le mode Angle en degrés affiche (r∠ θ).
- Le mode Angle en radians affiche $re^{i\theta}$.

En mode Angle en radians :

En mode Angle en grades :

$(4\cdot i)\blacktriangleright\text{Polar}$ $(4\angle\ 100.)$

En mode Angle en degrés :

valeurComplexe peut prendre n'importe quelle forme complexe. Toutefois, une entrée $re^{i\theta}$ génère une erreur en mode Angle en degrés.

Remarque : vous devez utiliser les parenthèses pour les entrées polaires ($r\angle\ \theta$).

polyEval() Catalogue > 📖

polyEval(*Liste1***,** *Expr1***)**⇒*expression*

polyEval(*Liste1***,** *Liste2***)**⇒*expression*

Interprète le premier argument comme les coefficients d'un polynôme ordonné suivant les puissances décroissantes et calcule la valeur de ce polynôme au point indiqué par le deuxième argument.

polyRoots() Catalogue > 📖

polyRoots(*Poly***,***Var***)** ⇒*liste*

polyRoots(*ListeCoeff***)** ⇒*liste*

La première syntaxe, **polyRoots(***Poly,Var***)**, affiche une liste des racines réelles du polynôme *Poly* pour la variable *Var*. S'il n'existe pas de racine réelle, une liste vide est affichée : { }.

La deuxième syntaxe, **polyRoots(***ListeCoeff***)**, affiche une liste de racines réelles du polynôme dont les coefficients sont donnés par la liste *ListeCoeff*.

Remarque : voir aussi **cPolyRoots()**, page 32.

PowerReg Catalogue > 📖

PowerReg *X***,***Y* [**,** *Fréq*] [**,** *Catégorie***,** *Inclure*]]

Effectue l'ajustement exponentiely = (a · (x)^b)sur les listes X et Y en utilisant la fréquence $Fréq$. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes, à l'exception de $Inclure$.

X et Y sont des listes de variables indépendantes et dépendantes.

$Fréq$ est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans $Fréq$ correspond à une fréquence d'occurrence pour chaque couple X et Y. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

$Catégorie$ est une liste de codes de catégories pour les couples X et Y correspondants.

$Inclure$ est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.RegEqn	Équation d'ajustement : a ·(x)^b
stat.a, stat.b	Coefficients d'ajustement
stat.r^2	Coefficient de détermination linéaire pour les données transformées
stat.r	Coefficient de corrélation pour les données transformées (ln(x), ln(y))
stat.Resid	Valeurs résiduelles associées au modèle exponentiel
stat.ResidTrans	Valeurs résiduelles associées à l'ajustement linéaire des données transformées
stat.XReg	Liste des points de données de la liste $Liste\ X$ modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de $Fréq$, $Liste\ de\ catégories$ et $Inclure\ les\ catégories$

Variable de sortie	Description
stat.YReg	Liste des points de données de la liste *Liste Y* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.FreqReg	Liste des fréquences correspondant à *stat.XReg* et *stat.YReg*

Prgm — Catalogue >

Prgm
Bloc
EndPrgm

Modèle de création d'un programme défini par l'utilisateur. À utiliser avec la commande **Define**, **Define LibPub**, ou **Define LibPriv**.

Bloc peut correspondre à une instruction unique ou à une série d'instructions séparées par le caractère ":" ou à une série d'instructions réparties sur plusieurs lignes.

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

Calcule le plus grand commun diviseur et affiche les résultats intermédiaires.

```
Define proggcd(a,b)=Prgm
                    Local d
                    While b≠0
                    d:=mod(a,b)
                    a:=b
                    b:=d
                    Disp a," ",b
                    EndWhile
                    Disp "GCD=",a
                    EndPrgm
                                        Done

proggcd(4560,450)
                                    450 60
                                    60 30
                                    30 0
                                    GCD=30
                                        Done
```

prodSeq() — Voir Π(), page 196.

Product (PI) — Voir Π(), page 196.

product() — Catalogue >

product(*Liste*[**,** *Début*[**,** *Fin*]]**)**⇒*expression*

product() Catalogue >

Donne le produit des éléments de *Liste*. *Début* et *Fin* sont facultatifs. Ils permettent de spécifier une plage d'éléments.

product(*Matrice1***[, *Début*[, *Fin*]])⇒***matrice***

Donne un vecteur ligne contenant les produits des éléments ligne par ligne de *Matrice1*. *Début* et *Fin* sont facultatifs. Ils permettent de spécifier une plage de colonnes.

Les éléments vides sont ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

$\text{product}\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} \quad [28 \quad 80 \quad 162]$

$\text{product}\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix},1,2 \quad [4 \quad 10 \quad 18]$

propFrac() Catalogue >

propFrac(*nombre_rationnel***)** décompose *nombre_rationnel* sous la forme de la somme d'un entier et d'une fraction de même signe et dont le dénominateur est supérieur au numérateur (fraction propre).

propFrac(*expression_rationnelle*,*Var***)** donne la somme des fractions propres et d'un polynôme par rapport à *Var*. Le degré de *Var* dans le dénominateur est supérieur au degré de *Var* dans le numérateur pour chaque fraction propre. Les mêmes puissances de *Var* sont regroupées. Les termes et leurs facteurs sont triés, *Var* étant la variable principale.

Si *Var* est omis, le développement des fractions propres s'effectue par rapport à la variable la plus importante. Les coefficients de la partie polynomiale sont ensuite ramenés à leur forme propre par rapport à leur variable la plus importante, et ainsi de suite.

$\text{propFrac}\left(\frac{4}{3}\right) \quad 1+\frac{1}{3}$

$\text{propFrac}\left(\frac{-4}{3}\right) \quad -1-\frac{1}{3}$

$\text{propFrac}\left(\frac{x^2+x+1}{x+1}+\frac{y^2+y+1}{y+1},x\right) \quad \frac{1}{x+1}+x+\frac{y^2+y+1}{y+1}$

$\text{propFrac}(Ans) \quad \frac{1}{x+1}+x+\frac{1}{y+1}+y$

propFrac() Catalogue >

Vous pouvez utiliser la fonction **propFrac ()** pour représenter des fractions mixtes et démontrer l'addition et la soustraction de fractions mixtes.

$\text{propFrac}\left(\frac{11}{7}\right)$	$1+\frac{4}{7}$
$\text{propFrac}\left(3+\frac{1}{11}+5+\frac{3}{4}\right)$	$8+\frac{37}{44}$
$\text{propFrac}\left(3+\frac{1}{11}-\left(5+\frac{3}{4}\right)\right)$	$-2-\frac{29}{44}$

Q

QR Catalogue >

QR *Matrice*, *qMatrice*, *rMatrice* [,*Tol*]

Calcule la factorisation QR Householder d'une matrice réelle ou complexe. Les matrices Q et R obtenues sont stockées dans les NomsMat *spécifiés*. La matrice Q est unitaire. La matrice R est triangulaire supérieure.

L'argument facultatif Tol permet de considérer comme nul tout élément de la matrice dont la valeur absolue est inférieure à *Tol*. Cet argument n'est utilisé que si la matrice contient des nombres en virgule flottante et ne contient pas de variables symbolique sans valeur affectée. Dans le cas contraire, *Tol* est ignoré.

- Si vous utilisez ctrl enter ou définissez le mode **Auto ou Approché (Approximate)** sur Approché (Approximate), les calculs sont exécutés en virgule flottante.
- Si *Tol* est omis ou inutilisé, la tolérance par défaut est calculée comme suit : 5E-14 ·max(dim(*Matrice*)) ·rowNorm (*Matrice*)

Le nombre en virgule flottante (9.) dans m1 fait que les résultats seront tous calculés en virgule flottante.

$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9.\end{bmatrix}\rightarrow m1$	$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9.\end{bmatrix}$
$\text{QR } m1,qm,rm$	*Done*
qm	$\begin{bmatrix}0.123091&0.904534&0.408248\\0.492366&0.301511&-0.816497\\0.86164&-0.301511&0.408248\end{bmatrix}$
rm	$\begin{bmatrix}8.12404&9.60114&11.0782\\0.&0.904534&1.80907\\0.&0.&0.\end{bmatrix}$

QR

Catalogue >

La factorisation QR sous forme numérique est calculée en utilisant la transformation de Householder. La factorisation symbolique est calculée en utilisant la méthode de Gram-Schmidt. Les colonnes de *NomMatq* sont les vecteurs de base orthonormaux de l'espace vectoriel engendré par les vecteurs colonnes de *matrice*.

$$\begin{bmatrix} m & n \\ o & p \end{bmatrix} \rightarrow m1 \qquad \begin{bmatrix} m & n \\ o & p \end{bmatrix}$$

QR *m1*,*qm*,*rm* — *Done*

$$qm = \begin{bmatrix} \frac{m}{\sqrt{m^2+o^2}} & \frac{-\text{sign}(m\cdot p-n\cdot o)\cdot o}{\sqrt{m^2+o^2}} \\ \frac{o}{\sqrt{m^2+o^2}} & \frac{m\cdot\text{sign}(m\cdot p-n\cdot o)}{\sqrt{m^2+o^2}} \end{bmatrix}$$

$$rm = \begin{bmatrix} \sqrt{m^2+o^2} & \frac{m\cdot n+o\cdot p}{\sqrt{m^2+o^2}} \\ 0 & \frac{|m\cdot p-n\cdot o|}{\sqrt{m^2+o^2}} \end{bmatrix}$$

QuadReg

Catalogue >

QuadReg *X*,*Y* [, *Fréq*] [, *Catégorie*, *Inclure*]]

Effectue l'ajustement polynomial de degré 2 y = a ·x2+b ·x+csur les listes *X* et *Y* en utilisant la fréquence *Fréq*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes, à l'exception de *Inclure*.

X et *Y* sont des listes de variables indépendantes et dépendantes.

Fréq est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *Fréq* correspond à une fréquence d'occurrence pour chaque couple *X* et *Y*. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Catégorie est une liste de codes de catégories pour les couples *X* et *Y* correspondants..

Inclure est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.RegEqn	Équation d'ajustement : a ·x^2+b ·x+c
stat.a, stat.b, stat.c	Coefficients d'ajustement
stat.R^2	Coefficient de détermination
stat.Resid	Valeurs résiduelles de l'ajustement
stat.XReg	Liste des points de données de la liste *Liste X* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.YReg	Liste des points de données de la liste *Liste Y* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.FreqReg	Liste des fréquences correspondant à *stat.XReg* et *stat.YReg*

QuartReg

Catalogue >

QuartReg *X*,*Y* [, *Fréq*] [, *Catégorie*, *Inclure*]]

Effectue l'ajustement polynomial de degré 4

y = a ·x^4+b ·x^3+c · x^2+d ·x+esur les listes *X* et *Y* en utilisant la fréquence *Fréq*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes, à l'exception de *Inclure*.

X et *Y* sont des listes de variables indépendantes et dépendantes.

Fréq est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *Fréq* correspond à une fréquence d'occurrence pour chaque couple *X* et *Y*. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Catégorie est une liste de codes de catégories pour les couples *X* et *Y* correspondants..

Inclure est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à "Éléments vides", page 223.

Variable de sortie	Description
stat.RegEqn	Équation d'ajustement : a ·x^4+b ·x^3+c · x^2+d ·x+e
stat.a, stat.b, stat.c, stat.d, stat.e	Coefficients d'ajustement
stat.R^2	Coefficient de détermination
stat.Resid	Valeurs résiduelles de l'ajustement
stat.XReg	Liste des points de données de la liste *Liste X* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.YReg	Liste des points de données de la liste *Liste Y* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.FreqReg	Liste des fréquences correspondant à *stat.XReg* et *stat.YReg*

R

R►Pθ() **Catalogue >**

R►Pθ (*listex***,** *listey***)** ⇒ *liste*
R►Pθ (*matricex***,** *matricey***)** ⇒ *matrice*

Donne la valeur de l'ordonnée θ - du point de coordonnées rectangulaires (*x*,*y*).

Remarque : Donne le résultat en degrés, en grades ou en radians, suivant le mode angulaire utilisé.

Remarque : Vous pouvez insérer cette fonction à partir du clavier de l'ordinateur en entrant **`R@>Ptheta(`**...**`)`** .

En mode Angle en degrés :

En mode Angle en grades :

En mode Angle en radians et en mode Auto :

R▶Pr() Catalogue >

R▶Pr (*listex***,** *listey***)** ⇒ *liste*
R▶Pr (*matricex***,** *matricey***)** ⇒ *matrice*

Donne la coordonnée r d'un point de coordonnées rectangulaires (*x*,*y*)

Remarque : Vous pouvez insérer cette fonction à partir du clavier de l'ordinateur en entrant **R@>Pr(...)** .

En mode Angle en radians et en mode Auto :

▶Rad Catalogue >

Convertit l'argument en mesure d'angle en radians.

Remarque : Vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **@>Rad**.

En mode Angle en degrés :

(1.5)▸Rad	$(0.02618)^r$

En mode Angle en grades :

(1.5)▸Rad	$(0.023562)^r$

rand() Catalogue >

rand() ⇒ *expression*
rand(*#Trials***)** ⇒ *liste*

rand() donne un nombre aléatoire compris entre 0 et 1.

rand(*nbreEssais***)** donne une liste de nombres aléatoires compris entre 0 et 1 pour le nombre d'essais *nbreEssais*

Réinitialise le générateur de nombres aléatoires.

RandSeed 1147	*Done*
rand(2)	{0.158206,0.717917}

randBin() Catalogue >

randBin(*n***,** *p***)** ⇒ *expression*
randBin(*n***,** *p***,** *#Trials***)** ⇒ *liste*

randBin(*n*, p **)** donne un nombre aléatoire tiré d'une distribution binomiale spécifiée

randBin(*n***,** *p***,** *nbreEssais***)** donne une liste de nombres aléatoires tirés d'une distribution binomiale spécifiée pour un nombre d'essais *nbreEssais*.

randInt() Catalogue >

randInt
(
lowBound,*upBound*)
⇒ *expression*
randInt
(
LimiteInf
,
LimiteSup
,*NbrEssais*) ⇒ *liste*

randInt
(
LimiteInf,*LimiteSup*)
donne un entier aléatoire pris entre les limites entières *LimiteInf* et *LimiteSup*

randInt
(
LimiteInf
,
LimiteSup
,*nbreEssais*) donne une liste d'entiers aléatoires pris entre les limites spécifiées pour un nombre d'essais *nbreEssais*.

randMat() Catalogue >

randMat(*nbreLignes*, *nbreColonnes***)**
⇒ *matrice*

Donne une matrice d'entiers compris entre -9 et 9 de la dimension spécifiée.

Les deux arguments doivent pouvoir être simplifiés en entiers.

RandSeed 1147	*Done*
randMat(3,3)	$\begin{bmatrix} 8 & -3 & 6 \\ -2 & 3 & -6 \\ 0 & 4 & -6 \end{bmatrix}$

Remarque : Les valeurs de cette matrice changent chaque fois que l'on appuie sur del.

randNorm() Catalogue >

randNorm(μ, σ**)** ⇒ *expression*
randNorm(μ, σ, *nbreessais***)** ⇒ *liste*

randNorm(μ, σ**)** Donne un nombre décimal issu de la loi normale spécifiée. Il peut s'agir de tout nombre réel, mais le résultat obtenu sera essentiellement compris dans l'intervalle [μ−3•σ, μ+3•σ].

randNorm(μ, σ, *nbreEssais***)** donne une liste de nombres décimaux tirés d'une distribution normale spécifiée pour un nombre d'essais *nbreEssais*.

RandSeed 1147	*Done*
randNorm(0,1)	0.492541
randNorm(3,4.5)	−3.54356

randPoly() Catalogue >

randPoly(*Var*, *Order***)** ⇒ *expression*

Donne un polynôme aléatoire de la variable *Var* de degré *Ordre* spécifié Les coefficients sont des entiers aléatoires situés dans la plage −9 à 9. Le coefficient du terme de plus au degré (Order) sera non nul.

Ordre doit être un entier compris entre 0 et 99

RandSeed 1147	*Done*
randPoly(x,5)	$-2\cdot x^5+3\cdot x^4-6\cdot x^3+4\cdot x-6$

randSamp() Catalogue >

randSamp(*List*,*#Trials*[,*noRepl*]**)** ⇒ *liste*

Donne une liste contenant un échantillon aléatoire de *nbreEssais* éléments choisis dans *Liste* avec option de remise (*sansRem*=0) ou sans option de remise (*sansRem*=1) L'option par défaut est avec remise.

RandSeed Catalogue >

RandSeed *Nombre*

Si *Nombre* = 0, réinitialise le générateur de nombres aléatoires Si *Nombre* ≠ 0, il sert à générer deux germes qui sont stockés dans les variables système seed1 et seed2

RandSeed 1147	*Done*
rand()	0.158206

real() Catalogue >

Donne la partie réelle de l'argument.

real(*List1***)** ⇒ *liste*

Donne les parties réelles de tous les éléments.

real(*Matrix1***)** ⇒ *matrice*

Donne les parties réelles de tous les éléments.

▶Rect Catalogue >

Vecteur ▶**Rect**

Remarque : Vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant `@>Rect`

Affiche *Vecteur* en coordonnées rectangulaires [x, y, z]. Le vecteur doit être un vecteur ligne ou colonne de dimension 2 ou 3.

Remarque : ▶**Rect** est uniquement une instruction d'affichage et non une fonction de conversion. On ne peut l'utiliser qu'à la fin d'une ligne et elle ne modifie pas le contenu du registre *ans*.

Remarque : Voir également ▶**Polar**, page 119.

complexValue ▶**Rect**

Affiche *valeurComplexe* sous forme rectangulaire (a+bi) La *valeurComplex*e peut prendre n'importe quelle forme rectangulaire Toutefois, une entrée $re^{i\theta}$ génère une erreur en mode Angle en degrés

Remarque : Vous devez utiliser des parenthèses pour les entrées en polaire ($r\angle\ \theta$).

En mode Angle en radians et en modes Auto :

En mode Angle en grades :

$$\langle(1\ \angle\ 100)\rangle \blacktriangleright \text{Rect} \qquad i$$

En mode Angle en degrés :

Remarque : Pour taper ∠ à partir du clavier, sélectionnez-le dans la liste des symboles du Catalogue.

ref(*Matrix1*[**,** *Tol*]**)** ⇒ *matrice*

Donne une réduite de Gauss de la matrice *Matrice1*.

L'argument facultatif Tol permet de considérer comme nul tout élément de la matrice dont la valeur absolue est inférieure à *Tol*. Cet argument n'est utilisé que si la matrice contient des nombres en virgule flottante et ne contient pas de variables symbolique sans valeur affectée. Dans le cas contraire, *Tol* est ignoré

- Si vous utilisez ctrl · e^x esc ↻ EE EE 10^x 10^x tab ^ tab esc tab Auto ou Approché **sur** Approché, les calculs sont exécutés en virgule flottante
- Si *Tol* est omis ou inutilisé, la tolérance par défaut est calculée comme suit : 5E−14 •max(dim(*Matrice1*)) •rowNorm (*Matrice1*)

N'utilisez pas d'éléments non définis dans *Matrice1*. L'utilisation d'éléments non définis peut générer des résultats inattendus.

Par exemple, si a est un élément non défini dans l'expression suivante, un message d'avertissement s'affiche et le résultat affiché est le suivant :

$$\text{ref}\begin{bmatrix} a & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & \frac{1}{a} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Un message d'avertissement est affiché car l'élément $1/a$ n'est pas valide pour a=0.

$$\text{ref}\begin{bmatrix} -2 & -2 & 0 & -6 \\ 1 & -1 & 9 & -9 \\ -5 & 2 & 4 & -4 \end{bmatrix} \qquad \begin{bmatrix} 1 & \frac{-2}{5} & \frac{-4}{5} & \frac{4}{5} \\ 0 & 1 & \frac{4}{7} & \frac{11}{7} \\ 0 & 0 & 1 & \frac{-62}{71} \end{bmatrix}$$

Pour éviter ce problème, vous pouvez stocker préalablement une valeur dans a ou utiliser l'opérateur "sachant que" (« | ») pour substituer une valeur, comme illustré dans l'exemple suivant.

$$\text{ref}\left(\begin{bmatrix} a & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right) | a=0 \qquad \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Remarque : Voir également **rref()**, page 142.

RefreshProbeVars

Catalogue >

RefreshProbeVars

Vous permet d'accéder aux données de capteur à partir de toutes les sondes de capteur connectées à l'aide de votre programme TI-Basic.

Valeur StatusVar	État
$statusVar$ =0	Normal (Poursuivez le programme)
$statusVar$ =1	L'application Vernier DataQuest™ est en mode Acquisition de données. **Remarque :** L'application Vernier DataQuest™ doit être en mode compteur pour que cette commande fonctionne.
$statusVar$ =2	L'application Vernier DataQuest™ n'est pas lancée.
$statusVar$ =3	L'application Vernier DataQuest™ est lancée, mais vous n'avez pas encore connecté de sonde.

Par exemple

```
Define temp()=
Prgm
© Vérifier si le système est prêt
RefreshProbeVars status
Si le statut=0 alors
Disp "prêt"
For n,1,50
RefreshProbeVars status
température:=compteur.température
Disp "Température: ",température
Si la température>30 alors
Disp "Trop chaude"
EndIf
© Attendre pendant 1 seconde entre
les échantillons
Wait 1
EndFor
Else
Disp "Pas prêt. Réessayer plus
tard"
```

```
EndIf

EndPrgm
```

Remarque : Ceci peut également être utilisé avec le TI-Innovator™ Hub.

remain()

Catalogue >

remain(*Liste1***,** *Liste2***)** ⇒ *liste*
remain(*Matrice1***,** *Matrice2***)** ⇒ *matrice*

Donne le reste de la division euclidienne du premier argument par le deuxième argument, défini par les identités suivantes :

remain(x,0) x
remain(x,y) x−y•iPart(x/y)

Par conséquent, remarquez que **remain(**−x,y**)**−**remain(**x,y**)**. Le résultat peut soit être égal à zéro , soit être du même signe que le premier argument.

Remarque : Voir aussi **mod()**, page 102.

remain(7,0)	7
remain(7,3)	1
remain(−7,3)	−1
remain(7,−3)	1
remain(−7,−3)	−1
remain({12,−14,16},{9,7,−5})	{3,0,1}

$$\text{remain}\left(\begin{bmatrix}9 & -7\\ 6 & 4\end{bmatrix},\begin{bmatrix}4 & 3\\ 4 & -3\end{bmatrix}\right) \qquad \begin{bmatrix}1 & -1\\ 2 & 1\end{bmatrix}$$

Request

Catalogue >

Request *promptString***,** *var*[**,** *DispFlag* [**,** *statusVar*]]

Request *promptString***,** *func***(***arg1***,** *...argn***)** [**,** *DispFlag* [**,** *statusVar*]]

Commande de programmation : Marque une pause dans l'exécution du programme et affiche une boîte de dialogue contenant le message *chaîneinvite*, ainsi qu'une zone de saisie destinée à la réponse que doit fournir l'utilisateur.

Lorsque l'utilisateur saisit une réponse et clique sur **OK**, le contenu de la zone de saisie est affecté à la variable *var*.

Définissez un programme :

```
Define request_demo()=Prgm
    Request "Rayon : ",r
    Disp "Area = ",pi*r²
EndPrgm
```

Exécutez le programme et saisissez une réponse :

```
request_demo()
```

Si l'utilisateur clique sur **Annuler**, le programme continue sans accepter aucune entrée. Le programme utilise la valeur précédete de la variable *var* si *var* était déjà définie.

L'argument optionnel *IndicAff* peut correspondre à toute expression.

- Si *IndicAff* est omis ou a pour valeur **1**, le message d'invite et la réponse de l'utilisateur sont affichés dans l'historique de l'application Calculs.
- Si *IndicAff* a pour valeur **0**, le message d'invite et la réponse de l'utilisateur ne sont pas affichés dans l'historique.

L'argument optionnel *VarÉtat* indique au programme comment déterminer si l'utilisateur a fermé la boîte de dialogue. Notez que *VarÉtat* nécessite la saisie de l'argument *IndicAff*.

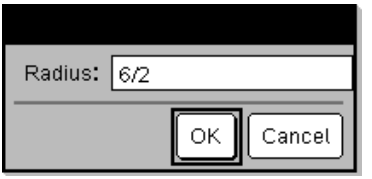
- Si l'utilisateur a cliqué sur **OK**, ou a appuyé sur **Entrée** ou sur **Ctrl+Entrée**, la variable *VarÉtat* prend la valeur **1**.
- Sinon, la variable *StatusVar* prend la valeur **0**.

L'argument de *func*() permet à un programme de stocker la réponse de l'utilisateur sous la forme d'une définition de fonction. Cette syntaxe équivaut à l'exécution par l'utilisateur de la commande suivante :

Definir *func*(*arg1*, *...argn*) = *réponse de l'utilisateur*

Le programme peut alors utiliser la fonction définie *fonc*(). La *chaîneinvite* doit guider l'utilisateur pour la saisie d'une *réponse* appropriée qui complète la définition de la fonction.

Remarque : Vous pouvez utiliser l Request commande dans un programme créé par l'utilisateur, mais pas dans une fonction.



Après avoir sélectionné **OK**, le résultat suivant s'affiche :

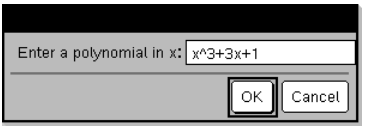
```
Demi-droite : 6/2
Area= 28.2743
```

Définissez un programme :

```
Define polynomial()=Prgm
    Request "Saisissez un polynôme en x
:",p(x)
    Disp "Les racines réelles sont
:",polyRoots(p(x),x)
EndPrgm
```

Exécutez le programme et saisissez une réponse :

```
polynomial()
```



Résultat après avoir saisi x^3+3x+1 et sélectionné **OK** :

```
Les racines réelles sont : {-0.322185}
```

Request

Catalogue >

Pour arrêter un programme qui contient une commande **Request** dans une boucle infinie :

- **Calculatrice:** Maintenez la touche on enfoncée et appuyez plusieurs fois sur enter.
- **Windows® :** Maintenez la touche **F12** enfoncée et appuyez plusieurs fois sur **Entrée**.
- **Macintosh® :** Maintenez la touche **F5** enfoncée et appuyez plusieurs fois sur **Entrée**.
- **iPad® :** L'application affiche une invite. Vous pouvez continuer à patienter ou annuler.

Remarque : Voir également **RequestStr**, page 136.

RequestStr

Catalogue >

RequestStr *chaîneinvite*, *var*[, *IndicAff*]

Commande de programmation : Fonctionne de façon similaire à la première syntaxe de la commande **Request**, excepté que la réponse de l'utilisateur est toujours interprétée comme une chaîne de caractères. Par contre, la commande **Request** interprète la réponse comme une expression, à moins que l'utilisateur ne la saisisse entre guillemets (“”).

Remarque : Vous pouvez utiliser la commande **RequestStr** dans un programme créé par l'utilisateur, mais pas dans une fonction.

Pour arrêter un programme qui contient une commande **RequestStr** dans une boucle infinie :

- **Calculatrice:** Maintenez la touche on enfoncée et appuyez plusieurs fois sur enter.

Définissez un programme :

```
Define requestStr_demo()=Prgm
    RequestStr “Votre nom :”,name,0
    Disp “La réponse comporte “,dim
(name),” caractères.”
EndPrgm
```

Exécutez le programme et saisissez une réponse :

```
requestStr_demo()
```

Après avoir sélectionné **OK**, le résultat affiché est le suivant (notez que si l'argument *IndicAff* a pour valeur **0**, le message d'invite et la réponse de l'utilisateur ne s'affichent pas dans l'historique) :

RequestStr

Catalogue >

- **Windows® :** Maintenez la touche **F12** enfoncée et appuyez plusieurs fois sur **Entrée**.
- **Macintosh® :** Maintenez la touche **F5** enfoncée et appuyez plusieurs fois sur **Entrée**.
- **iPad® :** L'application affiche une invite. Vous pouvez continuer à patienter ou annuler.

Remarque : Voir également **Request**, page 134.

```
requestStr_demo()
        La réponse comporte 5 caractères.
```

Return

Catalogue >

Return [*Expr*]

Donne *Expr* comme résultat de la fonction S'utilise dans les blocs **Func**...**EndFunc**.

Remarque : Vous pouvez utiliser **Return** sans argument dans un bloc **Prgm**...**EndPrgm** pour quitter un programme

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

```
Define factorial(nn)=
Func
Local answer,counter
1→answer
For counter,1,nn
answer·counter→answer
EndFor
Return answer
EndFunc
```

factorial(3) 6

right()

Catalogue >

right(*Liste1*[**,** *Num*]**)** ⇒ *liste*

Donne les *Nomb* éléments les plus à droite de la liste *Liste1*.

Si *Nomb* est absent, on obtient *Liste1*.

right({1,3,-2,4},3) {3,-2,4}

right(*chaîneSrce*[**,***Nomb*]**)** ⇒ *chaîne*

Donne la chaîne formée par les *Nomb* caractères les plus à droite de la chaîne de caractères *chaîneSrce*.

right("Hello",2) "lo"

right() Catalogue >

Si *Nomb* est absent, on obtient *chaîneSrce*.

right(*Comparaison***)** ⇒ *expression*

Donne le membre de droite d'une équation ou d'une inéquation.

right(x<3) 3

rk23 () Catalogue >

rk23(*Expr*, *Var*, *depVar*, {*Var0*, *VarMax*}, *depVar0*, *VarStep* [, *diftol*]**)** ⇒ *matrice*

rk23(*SystemOfExpr*, *Var*, *ListOfDepVars*, {*Var0*, *VarMax*}, *ListOfDepVars0*, *VarStep*[, *diftol*]**)** ⇒ *matrice*

rk23(*ListOfExpr*, *Var*, *ListOfDepVars*, {*Var0*, *VarMax*}, *ListOfDepVars0*, *VarStep*[, *diftol*]**)** ⇒ *matrice*

Utilise la méthode de Runge-Kutta pour résoudre le système d'équations.

$$\frac{d\,depVar}{d\,Var} = Expr(Var, depVar)$$

with *depVar*(*Var0*)=*depVar0* pour l'intervalle [*Var0*,*VarMax*]. Retourne une matrice dont la première ligne définit les valeurs de sortie de *Var*, définies à partir de *IncVar*. La deuxième ligne définit la valeur du premier composant de la solution aux valeurs *Var* correspondantes, etc.

Expr représente la partie droite qui définit l'équation différentielle.

SystèmeExpr correspond aux côtés droits qui définissent le système des équations différentielles (en fonction de l'ordre des variables dépendantes de la *ListeVarDép)*.

ListeExpr est la liste des côtés droits qui définissent le système des équations différentielles (en fonction de l'ordre des variables dépendantes de la *ListeVarDép*).

Équation différentielle :

y'=0.001*y*(100-y) et y(0)=10

rk23(0.001·y·(100−y),t,y,{0,100},10,1)

0.	1.	2.	3.	4.
10.	10.9367	11.9493	13.042	14.2

Pour afficher le résultat entier, appuyez sur ▲, puis utilisez les touches ◀ et ▶ pour déplacer le curseur.

Même équation avec *TolErr* définie à 1.E−6

rk23(0.001·y·(100−y),t,y,{0,100},10,1,1.E−6)

0.	1.	2.	3.	4.
10.	10.9367	11.9495	13.0423	14.2189

Système d'équations :

$$\begin{cases} y1' = -y1 + 0.1 \cdot y1 \cdot y2 \\ y2' = 3 \cdot y2 - y1 \cdot y2 \end{cases}$$

avec *y1*(0)=2 et *y2*(0)=5

rk23({−y1+0.1·y1·y2, 3·y2−y1·y2},t,{y1,y2},{0,5},{2,5},1)

0.	1.	2.	3.	4.
2.	1.94103	4.78694	3.25253	1.82848
5.	16.8311	12.3133	3.51112	6.27245

Var est la variable indépendante.

ListeVarDép est la liste des variables dépendantes.

{*Var0, MaxVar*} est une liste à deux éléments qui indique la fonction à intégrer, comprise entre *Var0* et *MaxVar*.

ListeVar0Dép est la liste des valeurs initiales pour les variables dépendantes.

Si *IncVar* est un nombre différent de zéro, signe(*IncVar*) = signe(*MaxVar*–*Var0*) et les solutions sont retournées pour *Var0*+i**IncVar* pour tout i=0,1,2,... tel que *Var0*+i**IncVar* soit dans [*var0,MaxVar*] (il est possible qu'il n'existe pas de solution en *MaxVar*).

si *IncVar* est un nombre égal à zéro, les solutions sont retournées aux valeurs *Var* "Runge-Kutta".

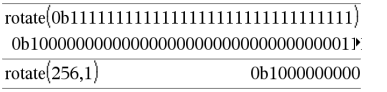
tolErr correspond à la tolérance d'erreur (valeur par défaut 0,001).

Remarque : Voir aussi **Modèle Racine n-ième**, page 1.

rotate(*Entier1*[**,***NbreRotations*]**)** ⇒ *entier*

Permute les bits de la représentation binaire d'un entier. Vous pouvez saisir *Entier1* dans un système de numération quelconque ; il est converti automatiquement en une forme binaire 64 bits signée. Si *Entier1* est trop important pour être codé, il est ramené à l'aide d'une congruence dans la plage appropriée Pour plus d'informations, consultez la section ▶**Base2**, page 17.

En mode base Bin :



Pour afficher le résultat entier, appuyez sur ▲, puis utilisez les touches ◀ et ▶ pour déplacer le curseur.

Si *nbreRotations* est positif, la permutation circulaire s'effectue vers la gauche Si *nbreRotations* est négatif, la permutation circulaire s'effectue vers la droite La valeur par défaut est −1 (permutation circulation de un bit vers la droite)

Par exemple, dans une permutation circulaire vers la droite :

Chaque bit est permuté vers la droite.

0b00000000000000111101011000011010 1

Le bit le plus à droite passe à la position la plus à gauche.

donne :

0b10000000000000011110101100001101 0

Le résultat s'affiche suivant le mode Base utilisé.

En mode base Hex :

rotate(0h78E)	0h3C7
rotate(0h78E,-2)	0h80000000000001E3
rotate(0h78E,2)	0h1E38

Important : Pour une entrée binaire ou hexadécimale, vous devez utiliser respectivement le préfixe 0b ou 0h (zéro, pas la lettre O).

rotate(*Liste1*[**,***NbreRotations*]**)** ⇒ *liste*

Donne une copie de *Liste1* dont les éléments ont été permutés circulairement vers la gauche ou vers la droite de *nbreRotations* éléments Ne modifie en rien *Liste1*

Si *nbreRotations* est positif, la permutation circulaire s'effectue vers la gauche Si *nbreRotations* est négatif, la permutation circulaire s'effectue vers la droite. La valeur par défaut est −1 (permutation circulation de un bit vers la droite)

En mode base Dec :

rotate({1,2,3,4})	{4,1,2,3}
rotate({1,2,3,4},-2)	{3,4,1,2}
rotate({1,2,3,4},1)	{2,3,4,1}

rotate(*Chaîne1*[**,***nbreRotations*]**)** ⇒ *chaîne*

Donne une copie de *Chaîne1* dont les caractères ont été permutés circulairement vers la gauche ou vers la droite de *nbreRotations* caractères. Ne modifie en rien *Chaîne1*

rotate("abcd")	"dabc"
rotate("abcd",-2)	"cdab"
rotate("abcd",1)	"bcda"

rotate() Catalogue >

Si *nbreRotations* est positif, la permutation circulaire s'effectue vers la gauche Si *nbreRotations* est négatif, la permutation circulaire s'effectue vers la droite La valeur par défaut est −1 (permutation vers la droite d'un caractère).

round() Catalogue >

Arrondit l'argument au nombre de chiffres n spécifié après la virgule.

chiffres doit être un entier compris dans la plage 0–12. Si *chiffres* esf absent, affiche l'argument arrondi à 12 chiffres significatifs.

Remarque : Le mode d'affichage des chiffres peut affecter le résultat affiché.

round(*List1***[, *chiffres*]) ⇒ *liste***

Donne la liste des éléments arrondis au nombre de chiffres spécifié.

round(*Matrice1***[, *chiffres*]) ⇒ *matrice***

Donne une matrice des éléments arrondis au nombre de chiffres n spécifié..

$\text{round}(1.234567,3)$ → 1.235

$\text{round}(\{\pi,\sqrt{2},\ln(2)\},4)$ → $\{3.1416,1.4142,0.6931\}$

$\text{round}\left(\begin{bmatrix}\ln(5) & \ln(3)\\ \pi & e^1\end{bmatrix},1\right)$ → $\begin{bmatrix}1.6 & 1.1\\ 3.1 & 2.7\end{bmatrix}$

rowAdd() Catalogue >

rowAdd(*Matrice1***, *rIndex1*, *rIndex2*) ⇒ *matrice***

Donne une copie de *Matrice1* obtenue en remplaçant dans la matrice la ligne *IndexL2* par la somme des lignes *IndexL1* et *IndexL2*

rowDim() Catalogue >

rowDim(*Matrix***) ⇒ *expression***

Donne le nombre de lignes de *Matrice*.

Remarque : Voir aussi **colDim()**, page 25.

$\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix} \to m1$ → $\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}$

$\text{rowDim}(m1)$ → 3

normeLig

Catalogue >

rowNorm(*Matrice***)** ⇒ *expression*

Donne le maximum des sommes des valeurs absolues des éléments de chaque ligne de *Matrice*.

Remarque : La matrice utilisée ne doit contenir que des éléments numériques.
Voir aussi **colNorm()** page 25.

$$\text{rowNorm}\left(\begin{bmatrix} -5 & 6 & -7 \\ 3 & 4 & 9 \\ 9 & -9 & -7 \end{bmatrix}\right) \quad 25$$

rowSwap()

Catalogue >

rowSwap(*Matrice1***,** *IndexL1***,** *IndexL2***)** ⇒ *matrice*

Donne la matrice *Matrice1* obtenue en échangeant les lignes *IndexL1* et *IndexL2*.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \rightarrow mat \qquad \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

$$\text{rowSwap}(mat,1,3) \qquad \begin{bmatrix} 5 & 6 \\ 3 & 4 \\ 1 & 2 \end{bmatrix}$$

rref()

Catalogue >

rref(*Matrice1***[,** *Tol***])** ⇒ *matrice*

Donne la réduite de Gauss-Jordan de *Matrice1*.

$$\text{rref}\left(\begin{bmatrix} -2 & -2 & 0 & -6 \\ 1 & -1 & 9 & -9 \\ -5 & 2 & 4 & -4 \end{bmatrix}\right) \qquad \begin{bmatrix} 1 & 0 & 0 & \frac{66}{71} \\ 0 & 1 & 0 & \frac{147}{71} \\ 0 & 0 & 1 & \frac{-62}{71} \end{bmatrix}$$

L'argument facultatif Tol permet de considérer comme nul tout élément de la matrice dont la valeur absolue est inférieure à *Tol*. Cet argument n'est utilisé que si la matrice contient des nombres en virgule flottante et ne contient pas de variables symbolique sans valeur affectée. Dans le cas contraire, *Tol* est ignoré

$$\text{rref}\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) \qquad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- Si vous utilisez ctrl · e^x esc ↻ EE EE 10^x 10^x tab ^ tab esc tab Auto ou Approché **sur** Approché, les calculs sont exécutés en virgule flottante
- Si *Tol* est omis ou inutilisé, la tolérance par défaut est calculée comme suit : 5E−14 •max(dim(*Matrice1*)) •rowNorm

rref() **Catalogue >**

(*Matrice1*)

Remarque : Voir aussi **ref()** page 132.

S

sec() **Touche**

sec(*Liste1***)** ⇒ *liste*

En mode Angle en degrés :

Remarque : l'argument est interprété comme la mesure d'un angle en degrés, en grades ou en radians, suivant le mode angulaire en cours d'utilisation. Vous pouvez utiliser °, G ou r pour préciser l'unité employée temporairement pour le calcul.

sec^{-1}() **Touche**

sec^{-1}(*Liste1***)** ⇒ *liste*

En mode Angle en degrés :

Remarque : donne le résultat en degrés, en grades ou en radians, suivant le mode angulaire utilisé.

En mode Angle en grades :

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **`arcsec(`**...**`)`**.

En mode Angle en radians :

sech() **Catalogue >**

sech(*Liste1***)** ⇒ *liste*

sech^{-1}() **Catalogue >**

sech^{-1} (*Liste1***)** ⇒ *liste*

En mode Angle en radians et en mode Format complexe Rectangulaire :

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **`arcsech(`**...**`)`**.

Send **Menu hub**

Send*exprOrString1* [**,** *exprOrString2*] ...

Exemple : allumer l'élément bleu de la DEL RGB intégrée pendant 0,5 seconde.

Send

Menu hub

Commande de programmation : envoie une ou plusieurs TI-Innovator™ Hub commandes à un hub connecté.

exprOrString doit être une commande TI-Innovator™ Hub valide. En général, *exprOrString* contient une commande **"SET ..."** pour contrôler un appareil ou une commande **"READ ..."** pour demander des données.

Les arguments sont envoyés au hub les uns après les autres.

Remarque : vous pouvez utiliser la commande **Send** dans un programme défini par l'utilisateur, mais pas dans une fonction.

Remarque : voir également **Get** (page 63), **GetStr** (page 70) et **eval()** (page 50).

Send "SET COLOR.BLUE ON TIME .5"
Done

Exemple : demander la valeur actuelle du capteur intégré du niveau de lumière du hub. Une commande **Get** récupère la valeur et l'affecte à la variable *lightval*.

Send "READ BRIGHTNESS"	*Done*
Get *lightval*	*Done*
lightval	0.347922

Exemple : envoyer une fréquence calculée au haut-parleur intégré du hub. Utilisez la variable spéciale*iostr.SendAns* pour afficher la commande du hub avec l'expression évaluée.

$n:=50$	50
$m:=4$	4
Send "SET SOUND eval(m· n)"	*Done*
iostr.SendAns	"SET SOUND 200"

seq()

Catalogue >

seq(*Expr*, *Var*, *Début*, *Fin*[, *Incrément*]**)⇒***liste*

Incrémente la valeur de Var comprise entre *Début* et *Fin* en fonction de l'incrément (*Inc*) spécifié et affiche le résultat sous forme de liste Le contenu initial de *Var* est conservé après l'application de **seq()**.

La valeur par défaut de *Inc* = 1.

$\text{seq}\left(n^2,n,1,6\right)$	$\{1,4,9,16,25,36\}$
$\text{seq}\left(\frac{1}{n},n,1,10,2\right)$	$\left\{1,\frac{1}{3},\frac{1}{5},\frac{1}{7},\frac{1}{9}\right\}$
$\text{sum}\left(\text{seq}\left(\frac{1}{n^2},n,1,10,1\right)\right)$	$\frac{1968329}{1270080}$

Remarque: Pour afficher un résultat approximatif,

Unité : Appuyez sur ctrl enter.
Windows® : Appuyez sur **Ctrl+Entrée**.
Macintosh® : Appuyez sur **⌘+Entrée**.
iPad® : Maintenez la touche **Entrée** enfoncée et sélectionnez ≈.

$\text{sum}\left(\text{seq}\left(\frac{1}{n^2},n,1,10,1\right)\right)$	1.54977

seqGen() Catalogue >

seqGen(*Expr*, *Var*, *VarDép*, {*Var0*, *MaxVar*}[, *ListeValeursInit* [, *IncVar* [, *ValeurMax*]]]**)** ⇒*liste*

Génère une liste de valeurs pour la suite *VarDép*(*Var*)=*Expr* comme suit : Incrémente la valeur de la variable indépendante *Var* de *Var0* à *MaxVar* par pas de *IncVar*, calcule *VarDép*(*Var*) pour les valeurs correspondantes de *Var* en utilisant *Expr* et *ListeValeursInit*, puis retourne le résultat sous forme de liste.

seqGen(*ListeOuSystèmeExpr*, *Var*, *ListeVarDép*, {*Var0*, *MaxVar*} [, *MatriceValeursInit* [, *IncVar* [, *ValeurMax*]]]**)** ⇒*matrice*

Génère une matrice de valeurs pour un système (ou une liste) de suites *ListeVarDép* (*Var*)=*ListeOuSystèmeExpr* comme suit : Incrémente la valeur de la variable indépendante *Var* de *Var0* à *MaxVar* par pas de *IncVar*, calcule *ListeVarDép* (*Var*) pour les valeurs correspondantes de *Var* en utilisant *ListeOuSystèmeExpr* et *MatriceValeursInit*, puis retourne le résultat sous forme de matrice.

Le contenu initial de *Var* est conservé après l'application de **seqGen()**.

La valeur par défaut de *IncVar* est 1.

Génère les cinq premières valeurs de la suite $u(n) = u(n\text{-}1)^2/2$, avec $u(1)$=**2** et *IncVar*=**1**.

$$\text{seqGen}\left(\frac{(u(n-1))^2}{n}, n, u, \{1,5\}, \{2\}\right) \quad \left\{2, 2, \frac{4}{3}, \frac{4}{9}, \frac{16}{405}\right\}$$

Exemple avec Var0=2 :

$$\text{seqGen}\left(\frac{u(n-1)+1}{n}, n, u, \{2,5\}, \{3\}\right) \quad \left\{3, \frac{4}{3}, \frac{7}{12}, \frac{19}{60}\right\}$$

Système de deux suites :

$$\text{seqGen}\left(\left\{\frac{1}{n}, \frac{u2(n-1)}{2}+u1(n-1)\right\}, n, \{u1, u2\}, \{1,5\}, \left[\begin{matrix}\_\\2\end{matrix}\right]\right) \quad \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ 2 & 2 & \frac{3}{2} & \frac{13}{12} & \frac{19}{24} \end{bmatrix}$$

Remarque : L'élément vide (_) dans la matrice de valeurs initiales ci-dessus est utilisé pour indiquer que la valeur initiale de u1(n) est calculée en utilisant la suite explicite u1 (n)=1/n.

seqn() Catalogue >

seqn(*Expr***(***u***, *n* [, *ListeValeursInit*[, *nMax* [, *ValeurMax*]]])**⇒*liste***

Génère une liste de valeurs pour la suite *u*(*n*)=*Expr*(*u*, *n*) comme suit : Incrémente *n* de 1 à *nMax* par incrément de 1, calcule *u*(*n*) pour les valeurs correspondantes de *n* en utilisant *Expr* (*u*, *n*) et *ListeValeursInit*, puis retourne le résultat sous forme de liste.

Génère les cinq premières valeurs de la suite $u(n) = u(n\text{-}1)/2$, avec $u(1)$=**2**.

$$\text{seqn}\left(\frac{u(n-1)}{n}, \{2\}, 6\right) \quad \left\{2, 1, \frac{1}{3}, \frac{1}{12}, \frac{1}{60}, \frac{1}{360}\right\}$$

$$\text{seqn}\left(\frac{1}{n^2}, 6\right) \quad \left\{1, \frac{1}{4}, \frac{1}{9}, \frac{1}{16}, \frac{1}{25}, \frac{1}{36}\right\}$$

seqn()

Catalogue >

seqn(*Expr***(***n* [**,** *nMax* [**,** *ValeurMax*]]**)**⇒*liste*

Génère une liste de valeurs pour la suite *u*(*n*)=*Expr*(*n*) comme suit : Incrémente *n* de 1 à *nMax* par incrément de 1, calcule *u*(*n*) pour les valeurs correspondantes de *n* en utilisant *Expr*(*n*), puis retourne le résultat sous forme de liste.

Si *nMax* n'a pas été défini, il prend la valeur 2500.

Si *nMax*=0 n'a pas été défini, *nMax* prend la valeur 2500.

Remarque : seqn() appel **seqGen( )** avec *n0*=**1** et *Incn* =**1**

setMode()

Catalogue >

setMode(*EntierNomMode***,** *EntierRéglage***)** ⇒*entier*

setMode(*liste***)** ⇒*liste des entiers*

Accessible uniquement dans une fonction ou un programme.

setMode(*EntierNomMode***,** *EntierRéglage***)** règle provisoirement le mode *EntierNomMode* sur le nouveau réglage *EntierRéglage* et affiche un entier correspondant au réglage d'origine de ce mode. Le changement est limité à la durée d'exécution du programme/de la fonction.

EntierNomMode indique le mode que vous souhaitez régler. Il doit s'agir d'un des entiers du mode du tableau ci-dessous.

EntierRéglage indique le nouveau réglage pour ce mode. Il doit s'agir de l'un des entiers de réglage indiqués ci-dessous pour le mode spécifique que vous configurez.

setMode(*liste***)** permet de modifier plusieurs réglages. *liste* contient les paires d'entiers de mode et d'entiers de réglage. **setMode(***liste***)** affiche une liste dont les paires d'entiers représentent les modes et réglages d'origine.

Affiche la valeur approchée de π à l'aide du réglage par défaut de Afficher chiffres, puis affiche π avec le réglage Fixe 2. Vérifiez que la valeur par défaut est bien restaurée après l'exécution du programme.

setMode() Catalogue >

Si vous avez enregistré tous les réglages du mode avec **getMode(0)** → *var*, **setMode(***var***)** permet de restaurer ces réglages jusqu'à fermeture du programme ou de la fonction. Voir **getMode()**, page 69.

Remarque : Les réglages de mode actuels sont transférés dans les sous-programmes appelés. Si un sous-programme change un quelconque réglage du mode, le changement sera perdu dès le retour au programme appelant.

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

Nom du mode	Entier du mode	Entiers de réglage
Afficher chiffres	1	**1**=Flottant, **2**=Flottant 1, **3**=Flottant 2, **4**=Flottant 3, **5**=Flottant 4, **6**=Flottant 5, **7**=Flottant 6, **8**=Flottant 7, **9**=Flottant 8, **10**=Flottant 9, **11**=Flottant 10, **12**=Flottant 11, **13**=Flottant 12, **14**=Fixe 0, **15**=Fixe 1, **16**=Fixe 2, **17**=Fixe 3, **18**=Fixe 4, **19**=Fixe 5, **20**=Fixe 6, **21**=Fixe 7, **22**=Fixe 8, **23**=Fixe 9, **24**=Fixe 10, **25**=Fixe 11, **26**=Fixe 12
Angle	2	**1**=Radian, **2**=Degré, **3**=Grade
Format Exponentiel	3	**1**=Normal, **2**=Scientifique, **3**=Ingénieur
Réel ou Complexe	4	**1**=Réel, **2**=Rectangulaire, **3**=Polaire
Auto ou Approché	5	**1**=Auto, **2**=Approché
Format Vecteur	6	**1**=Rectangulaire, **2**=Cylindrique, **3**=Sphérique
Base	7	**1**=Décimale, **2**=Hexadécimale, **3**=Binaire

shift() Catalogue >

shift(*Entier1*[**,***nbreDécal*]**)**⇒*entier* En mode base Bin :

Décale les bits de la représentation binaire d'un entier. *Entier1* peut être un entier de n'importe quelle base ; il est automatiquement converti sous forme binaire (64 bits) signée. Si *Entier1* est trop important pour être codé sur 32 bits, il est ramené à l'aide d'une congruence dans la plage appropriée. Pour de plus amples informations, voir ▶**Base2**, page 17.

Si *nbreDécal* est positif, le décalage s'effectue vers la gauche. Si *nbreDécal* est négatif, le décalage s'effectue vers la droite. La valeur par défaut est ⁻1 (décalage d'un bit vers la droite).

Dans un décalage vers la droite, le dernier bit est éliminé et 0 ou 1 est inséré à gauche selon le premier bit. Dans un décalage vers la gauche, le premier bit est éliminé et 0 est inséré comme dernier bit.

Par exemple, dans un décalage vers la droite :

Tous les bits sont décalés vers la droite.

0b00000000000000111101011000011010

Insère 0 si le premier bit est un 0

ou 1 si ce bit est un 1.

donne :

0b00000000000000011110101100001101O

Le résultat est affiché selon le mode Base utilisé. Les zéros de tête ne sont pas affichés.

shift(*Liste1* [**,***nbreDécal*]**)**⇒*liste*

Donne une copie de *Liste1* dont les éléments ont été décalés vers la gauche ou vers la droite de *nbreDécal* éléments. Ne modifie en rien *Liste1*.

shift(0b1111010110000110101)	0b1111010110000011010
shift(256,1)	0b1000000000

En mode base Hex :

shift(0h78E)	0h3C7
shift(0h78E,⁻2)	0h1E3
shift(0h78E,2)	0h1E38

Important : pour une entrée binaire ou hexadécimale, vous devez utiliser respectivement le préfixe 0b ou 0h (zéro, pas la lettre O).

En mode base Dec :

shift({1,2,3,4})	{undef,1,2,3}
shift({1,2,3,4},⁻2)	{undef,undef,1,2}
shift({1,2,3,4},2)	{3,4,undef,undef}

shift() Catalogue >

Si *nbreDécal* est positif, le décalage s'effectue vers la gauche. Si *nbreDécal* est négatif, le décalage s'effectue vers la droite. La valeur par défaut est ⁻1 (décalage d'un élément vers la droite).

Les éléments introduits au début ou à la fin de *liste* par l'opération de décalage sont remplacés par undef (non défini).

shift(*Chaîne1* [**,***nbreDécal*]**)**⇒*chaîne*

Donne une copie de *Chaîne1* dont les caractères ont été décalés vers la gauche ou vers la droite de *nbreDécal* caractères. Ne modifie en rien *Chaîne1*.

Si *nbreDécal* est positif, le décalage s'effectue vers la gauche. Si *nbreDécal* est négatif, le décalage s'effectue vers la droite. La valeur par défaut est ⁻1 (décalage d'un caractère vers la droite).

Les caractères introduits au début ou à la fin de *Chaîne* par l'opération de décalage sont remplacés par un espace.

shift("abcd")	" abc"
shift("abcd",-2)	" ab"
shift("abcd",1)	"bcd "

sign() Catalogue >

sign(*Liste1***)**⇒*liste*

sign(*Matrice1***)**⇒*matrice*

sign(0) donne ⁻1 en mode Format complexe Réel ; sinon, donne lui-même.

sign(0) représente le cercle d'unité dans le domaine complexe.

Dans le cas d'une liste ou d'une matrice, donne les signes de tous les éléments.

En mode Format complexe Réel :

simult() Catalogue >

simult(*matriceCoeff***,** *vecteurConst*[**,** *Tol*]**)**⇒*matrice*

Donne un vecteur colonne contenant les solutions d'un système d'équations.

Résolution de x et y :

x + 2y = 1

3x + 4y = ⁻1

Remarque : voir aussi **linSolve()**, page 88.

matriceCoeff doit être une matrice carrée qui contient les coefficients des équations.

vecteurConst doit avoir le même nombre de lignes (même dimension) que *matriceCoeff* et contenir le second membre.

L'argument facultatif Tol permet de considérer comme nul tout élément de la matrice dont la valeur absolue est inférieure à *Tol*. Cet argument n'est utilisé que si la matrice contient des nombres en virgule flottante et ne contient pas de variables symbolique sans valeur affectée. Dans le cas contraire, *Tol* est ignoré.

- Si vous réglez le mode **Auto ou Approché (Approximate)** sur Approché (Approximate), les calculs sont exécutés en virgule flottante.
- Si *Tol* est omis ou inutilisé, la tolérance par défaut est calculée comme suit : 5E⁻14 ·max(dim(*matriceCoeff*)) ·rowNorm(*matriceCoeff*)

simult(*matriceCoeff***,** *matriceConst***[,** *Tol***])⇒***matrice*

Permet de résoudre plusieurs systèmes d'équations, ayant les mêmes coefficients mais des seconds membres différents.

Chaque colonne de *matriceConst* représente le second membre d'un système d'équations. Chaque colonne de la matrice obtenue contient la solution du système correspondant.

$$\text{simult}\left(\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix},\begin{bmatrix}1\\ -1\end{bmatrix}\right) \qquad \begin{bmatrix}-3\\ 2\end{bmatrix}$$

La solution est x=⁻3 et y=2.

Résolution :

ax + by = 1

cx + dy = 2

Résolution :

x + 2y = 1

3x + 4y = ⁻1

x + 2y = 2

3x + 4y = ⁻3

$$\text{simult}\left(\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix},\begin{bmatrix}1 & 2\\ -1 & -3\end{bmatrix}\right) \qquad \begin{bmatrix}-3 & -7\\ 2 & \frac{9}{2}\end{bmatrix}$$

Pour le premier système, x=⁻3 et y=2. Pour le deuxième système, x=⁻7 et y=9/2.

sin()

Touche trig

sin(*Liste1***)⇒***liste*

sin(*Liste1***)** donne la liste des sinus des éléments de *Liste1*.

Remarque : l'argument est interprété comme mesure d'angle en degrés, en grades ou en radians, suivant le mode angulaire sélectionné. Vous pouvez utiliser °,G ou r pour ignorer temporairement le mode angulaire sélectionné.

En mode Angle en degrés :

En mode Angle en grades :

En mode Angle en radians :

sin(*matriceCarrée1***)⇒***matriceCarrée*

Donne le sinus de la matrice *matriceCarrée1*. Ce calcul est différent du calcul du sinus de chaque élément. Pour plus d'informations sur la méthode de calcul, reportez-vous à **cos()**.

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

En mode Angle en radians :

$$\sin\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix}$$

$$\begin{bmatrix}0.9424 & -0.04542 & -0.031999\\ -0.045492 & 0.949254 & -0.020274\\ -0.048739 & -0.00523 & 0.961051\end{bmatrix}$$

$\sin^{-1}()$

Touche trig

$\sin^{-1}$(*Liste1***)⇒***liste*

$\sin^{-1}$(*List1***)** donne la liste des arcs sinus des éléments de *Liste1*.

Remarque : donne le résultat en degrés, en grades ou en radians, suivant le mode angulaire utilisé.

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **`arcsin(...)`**.

En mode Angle en degrés :

En mode Angle en grades :

En mode Angle en radians :

$\sin^{-1}$(*matriceCarrée1***)⇒***matriceCarrée*

Donne l'argument arc sinus de la matrice *matriceCarrée1*. Ce calcul est différent du calcul de l'argument arc sinus de chaque élément. Pour plus d'informations sur la méthode de calcul, reportez-vous à **cos()**.

En mode Angle en radians et en mode Format complexe Rectangulaire :

$$\sin^{-1}\left(\begin{bmatrix}1 & 5\\ 4 & 2\end{bmatrix}\right)$$

$$\begin{bmatrix}-0.174533-0.12198\cdot i & 1.74533-2.35591\cdot i\\ 1.39626-1.88473\cdot i & 0.174533-0.593162\cdot i\end{bmatrix}$$

$\sin^{-1}()$ — Touche trig

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

sinh() — Catalogue >

sinh(*Liste1***)⇒***liste*

sinh (Liste1) donne la liste des sinus hyperboliques des éléments de *Liste1*.

$\sinh(1.2)$ → 1.50946

$\sinh(\{0,1.2,3.\})$ → $\{0,1.50946,10.0179\}$

sinh(*matriceCarrée1***)⇒***matriceCarrée*

Donne le sinus hyperbolique de la matrice *matriceCarrée1*. Ce calcul est différent du calcul du sinus hyperbolique de chaque élément. Pour plus d'informations sur la méthode de calcul, reportez-vous à **cos()**.

En mode Angle en radians :

$$\sinh\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix} = \begin{bmatrix}360.954 & 305.708 & 239.604\\ 352.912 & 233.495 & 193.564\\ 298.632 & 154.599 & 140.251\end{bmatrix}$$

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

$\sinh^{-1}()$ — Catalogue >

$\sinh^{-1}$(*Liste1***)⇒***liste*

$\sinh^{-1}$(*Liste1***)** donne la liste des arguments sinus hyperboliques des éléments de *Liste1*.

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **arcsinh(…)**.

$\sinh^{-1}$(*matriceCarrée1***)⇒***matriceCarrée*

Donne l'argument sinus hyperbolique de la matrice *matriceCarrée1*. Ce calcul est différent du calcul de l'argument sinus hyperbolique de chaque élément. Pour plus d'informations sur la méthode de calcul, reportez-vous à **cos()**.

En mode Angle en radians :

$$\sinh^{-1}\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix} = \begin{bmatrix}0.041751 & 2.15557 & 1.1582\\ 1.46382 & 0.926568 & 0.112557\\ 2.75079 & -1.5283 & 0.57268\end{bmatrix}$$

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

SinReg *X*, *Y* [, [*Itérations*],[*Période*] [, *Catégorie*, *Inclure*]]

Effectue l'ajustement sinusoïdal sur les listes *X* et *Y*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes, à l'exception de *Inclure*.

X et *Y* sont des listes de variables indépendantes et dépendantes.

Itérations spécifie le nombre maximum d'itérations (1 à 16) utilisées lors de ce calcul. S'il est omis, la valeur par défaut est 8. On obtient généralement une meilleure précision en choisissant une valeur élevée, mais cela augmente également le temps de calcul, et vice versa.

Période spécifie une période estimée. S'il est omis, la différence entre les valeurs de *X* doit être égale et en ordre séquentiel. Si vous spécifiez la *Période*, les différences entre les valeurs de x peuvent être inégales.

Catégorie est une liste de codes de catégories pour les couples *X* et *Y* correspondants..

Inclure est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

Le résultat obtenu avec **SinReg** est toujours exprimé en radians, indépendamment du mode Angle sélectionné.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.RegEqn	Équation d'ajustement : a ·sin(bx+c)+d
stat.a, stat.b, stat.c, stat.d	Coefficients d'ajustement

Variable de sortie	Description
stat.Resid	Valeurs résiduelles de l'ajustement
stat.XReg	Liste des points de données de la liste *Liste X* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.YReg	Liste des points de données de la liste *Liste Y* modifiée, actuellement utilisée dans l'ajustement basé sur les restrictions de *Fréq*, *Liste de catégories* et *Inclure les catégories*
stat.FreqReg	Liste des fréquences correspondant à *stat.XReg* et *stat.YReg*

SortA

Catalogue >

SortA *Liste1*[**,** *Liste2*] [**,** *Liste3*] ...

SortA *Vecteur1*[**,** *Vecteur2*] [**,** *Vecteur3*] ...

Trie les éléments du premier argument en ordre croissant.

Si d'autres arguments sont présents, trie les éléments de chacun d'entre eux de sorte que leur nouvelle position corresponde aux nouvelles positions des éléments dans le premier argument.

Tous les arguments doivent être des noms de listes ou de vecteurs et tous doivent être de même dimension.

Les éléments vides compris dans le premier argument ont été déplacés au bas de la liste. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

$\{2,1,4,3\} \rightarrow list1$	$\{2,1,4,3\}$
SortA *list1*	*Done*
list1	$\{1,2,3,4\}$
$\{4,3,2,1\} \rightarrow list2$	$\{4,3,2,1\}$
SortA *list2*,*list1*	*Done*
list2	$\{1,2,3,4\}$
list1	$\{4,3,2,1\}$

SortD

Catalogue >

SortD *Liste1*[**,** *Liste2*] [**,** *Liste3*] ...

SortD *Vecteur1*[**,***Vecteur2*] [**,***Vecteur3*] ...

Identique à **SortA**, mais **SortD** trie les éléments en ordre décroissant.

$\{2,1,4,3\} \rightarrow list1$	$\{2,1,4,3\}$
$\{1,2,3,4\} \rightarrow list2$	$\{1,2,3,4\}$
SortD *list1*,*list2*	*Done*
list1	$\{4,3,2,1\}$
list2	$\{3,4,1,2\}$

SortD

Catalogue >

Les éléments vides compris dans le premier argument ont été déplacés au bas de la liste. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

▸Sphere

Catalogue >

Vecteur **▸Sphere**

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **@>Sphere**.

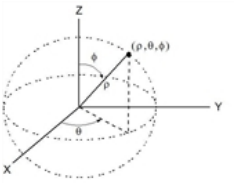
Affiche le vecteur ligne ou colonne en coordonnées sphériques [ρ ∠θ ∠ϕ].

Vecteur doit être un vecteur ligne ou colonne de dimension 3.

Remarque : ▸Sphere est uniquement une instruction d'affichage et non une fonction de conversion. On ne peut l'utiliser qu'à la fin d'une ligne.

$$[1 \quad 2 \quad 3] \blacktriangleright \text{Sphere}$$
$$[3.74166 \quad \angle 1.10715 \quad \angle 0.640522]$$

$$\left(\begin{bmatrix} 2 & \angle \frac{\pi}{4} & 3 \end{bmatrix}\right) \blacktriangleright \text{Sphere}$$
$$[3.60555 \quad \angle 0.785398 \quad \angle 0.588003]$$



sqrt()

Catalogue >

sqrt(*Liste1***)**⇒*liste*

Donne la racine carrée de l'argument.

Dans le cas d'une liste, donne la liste des racines carrées des éléments de *Liste1*.

Remarque : voir aussi **Modèle Racine carrée**, page 1.

stat.results

Affiche le résultat d'un calcul statistique.

Les résultats sont affichés sous forme d'ensemble de paires nom-valeur. Les noms spécifiques affichés varient suivant la fonction ou commande statistique la plus récemment calculée ou exécutée.

Vous pouvez copier un nom ou une valeur et la coller à d'autres emplacements.

Remarque : ne définissez pas de variables dont le nom est identique à celles utilisées dans le cadre de l'analyse statistique. Dans certains cas, cela peut générer une erreur. Les noms de variables utilisés pour l'analyse statistique sont répertoriés dans le tableau ci-dessous.

xlist:={1,2,3,4,5} {1,2,3,4,5}

ylist:={4,8,11,14,17} {4,8,11,14,17}

LinRegMx *xlist*,*ylist*,1: *stat.results*

"Title"	"Linear Regression (mx+b)"
"RegEqn"	"m*x+b"
"m"	3.2
"b"	1.2
"r²"	0.996109
"r"	0.998053
"Resid"	"{...}"

stat.values

"Linear Regression (mx+b)"
"m*x+b"
3.2
1.2
0.996109
0.998053
"{⁻0.4,0.4,0.2,0.,⁻0.2}"

stat.a	stat.dfDenom	stat.MedianY	stat.Q3X	stat.SSBlock
stat.AdjR²	stat.dfBlock	stat.MEPred	stat.Q3Y	stat.SSCol
stat.b	stat.dfCol	stat.MinX	stat.r	stat.SSX
stat.b0	stat.dfError	stat.MinY	stat.r²	stat.SSY
stat.b1	stat.dfInteract	stat.MS	stat.RegEqn	stat.SSError
stat.b2	stat.dfReg	stat.MSBlock	stat.Resid	stat.SSInteract
stat.b3	stat.dfNumer	stat.MSCol	stat.ResidTrans	stat.SSReg
stat.b4	stat.dfRow	stat.MSError	stat.σx	stat.SSRow
stat.b5	stat.DW	stat.MSInteract	stat.σy	stat.tList
stat.b6	stat.e	stat.MSReg	stat.σx1	stat.UpperPred
stat.b7	stat.ExpMatrix	stat.MSRow	stat.σx2	stat.UpperVal
stat.b8	stat.F	stat.n	stat.Σx	stat.$\bar{x}$
stat.b9	stat.FBlock	stat.$\hat{p}$	stat.Σx²	stat.$\bar{x}$1
stat.b10	stat.Fcol	stat.$\hat{p}$1	stat.Σxy	stat.$\bar{x}$2
stat.bList	stat.FInteract	stat.$\hat{p}$2	stat.Σy	stat.$\bar{x}$Diff
stat.χ²	stat.FreqReg	stat.$\hat{p}$Diff	stat.Σy²	stat.$\bar{x}$List

stat.c	stat.Frow	stat.PList	stat.s	stat.XReg
stat.CLower	stat.Leverage	stat.PVal	stat.SE	stat.XVal
stat.CLowerList	stat.LowerPred	stat.PValBlock	stat.SEList	stat.XValList
stat.CompList	stat.LowerVal	stat.PValCol	stat.SEPred	stat.$\bar{y}$
stat.CompMatrix	stat.m	stat.PValInteract	stat.sResid	stat.$\hat{y}$
stat.CookDist	stat.MaxX	stat.PValRow	stat.SEslope	stat.$\hat{y}$List
stat.CUpper	stat.MaxY	stat.Q1X	stat.sp	stat.YReg
stat.CUpperList	stat.ME	stat.Q1Y	stat.SS	
stat.d	stat.MedianX			

Remarque : Chaque fois que l'application Tableur & listes calcule des résultats statistiques, les variables du groupe « stat**.** » sont copiées dans un groupe « stat**#.** », où # est un nombre qui est incrémenté automatiquement. Cela vous permet de conserver les résultats précédents tout en effectuant plusieurs calculs.

stat.values — Catalogue >

stat.values

Affiche une matrice des valeurs calculées pour la fonction ou commande statistique la plus récemment calculée ou exécutée.

Contrairement à **stat.results**, **stat.values** omet les noms associés aux valeurs.

Vous pouvez copier une valeur et la coller à d'autres emplacements.

Voir l'exemple donné pour **stat.results.**

stDevPop() — Catalogue >

stDevPop(*Liste***[,** *listeFréq***])**⇒*expression*

Donne l'écart-type de population des éléments de *Liste*.

Chaque élément de la liste *listeFréq* totalise le nombre d'occurrences de l'élément correspondant de *Liste*.

En mode Angle en radians et en modes Auto :

Remarque : *Liste* doit contenir au moins deux éléments. Les éléments vides sont ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

stDevPop(*Matrice1*[**,** *matriceFréq*]**)**⇒*matrice*

Donne un vecteur ligne des écarts-types de population des colonnes de *Matrice1*.

Chaque élément de *matriceFréq* totalise le nombre d'occurrences de l'élément correspondant de *Matrice1*.

Remarque : *Matrice1* doit contenir au moins deux lignes. Les éléments vides sont ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

stDevSamp() Catalogue >

stDevSamp(*Liste*[**,** *listeFréq*]**)**⇒*expression*

Donne l'écart-type d'échantillon des éléments de *Liste*.

Chaque élément de la liste *listeFréq* totalise le nombre d'occurrences de l'élément correspondant de *Liste*.

Remarque : *Liste* doit contenir au moins deux éléments. Les éléments vides sont ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

stDevSamp(*Matrice1*[**,** *matriceFréq*]**)**⇒*matrice*

Donne un vecteur ligne des écarts-types de population des colonnes de *Matrice1*.

Chaque élément de *matriceFréq* totalise le nombre d'occurrences de l'élément correspondant de *Matrice1*.

stDevSamp() Catalogue >

Remarque : *Matrice1* doit contenir au moins deux lignes. Les éléments vides sont ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

Stop Catalogue >

Stop

Commande de programmation : Ferme le programme.

Stop n'est pas autorisé dans les fonctions.

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

$i:=0$	0
Define $prog1()$=Prgm For i,1,10,1 If i=5 Stop EndFor EndPrgm	*Done*
$prog1()$	*Done*
i	5

Store Voir → (store), page 204.

string() Catalogue >

string(*Expr***)⇒***chaîne*

Simplifie *Expr* et donne le résultat sous forme de chaîne de caractères.

subMat() Catalogue >

subMat(*Matrice1***[, ***colDébut***] [, ***colDébut***] [, ***ligneFin***] [, ***colFin***]) ⇒***matrice***

Donne la matrice spécifiée, extraite de *Matrice1*.

Valeurs par défaut : *ligneDébut*=1, *colDébut*=1, *ligneFin*=dernière ligne, *colFin*=dernière colonne.

$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} \rightarrow m1$	$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}$
$\text{subMat}(m1,2,1,3,2)$	$\begin{bmatrix}4&5\\7&8\end{bmatrix}$
$\text{subMat}(m1,2,2)$	$\begin{bmatrix}5&6\\8&9\end{bmatrix}$

sum() Catalogue >

sum(*Liste*[**,** *Début*[**,** *Fin*]]**)**⇒*expression*

Donne la somme des éléments de *Liste*.

Début et *Fin* sont facultatifs. Ils permettent de spécifier une plage d'éléments.

Tout argument vide génère un résultat vide. Les éléments vides de *Liste* sont ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

sum(*Matrice1*[**,** *Début*[**,** *Fin*]]**)**⇒*matrice*

Donne un vecteur ligne contenant les sommes des éléments de chaque colonne de *Matrice1*.

Début et *Fin* sont facultatifs. Ils permettent de spécifier une plage de colonnes.

Tout argument vide génère un résultat vide. Les éléments vides de *Matrice1* sont ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

$$\text{sum}\left(\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}\right) \qquad \begin{bmatrix}5&7&9\end{bmatrix}$$

$$\text{sum}\left(\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}\right) \qquad \begin{bmatrix}12&15&18\end{bmatrix}$$

$$\text{sum}\left(\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix},2,3\right) \qquad \begin{bmatrix}11&13&15\end{bmatrix}$$

sumIf() Catalogue >

sumIf(*Liste***,***Critère*[, *ListeSommes*]**)**⇒*valeur*

Affiche la somme cumulée de tous les éléments dans *Liste* qui répondent au *critère* spécifié. Vous pouvez aussi spécifier une autre liste, *ListeSommes*, pour fournir les éléments à cumuler.

Liste peut être une expression, une liste ou une matrice. *ListeSommes*, si spécifiée, doit avoir la/les même(s) dimension (s) que *Liste*.

Le critère peut être :

- Une valeur, une expression ou une chaîne. Par exemple, **34** cumule uniquement les éléments dans *Liste* qui donnent la valeur 34.
- Une expression booléenne contenant le symbole **?** comme paramètre substituable à tout élément. Par exemple, **?<10** cumule uniquement les éléments de *Liste* qui sont inférieurs à 10.

Lorsqu'un élément de *Liste* répond au *critère*, il est ajouté à la somme cumulée. Si vous incluez *ListeSommes*, c'est l'élément correspondant dans *ListeSommes* qui est ajouté à la somme.

Dans l'application Tableur & listes, vous pouvez utiliser une plage de cellules à la place de *Liste* et *ListeSommes*.

Les éléments vides sont ignorés. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

Remarque : voir également **countIf()**, page 31.

sumSeq() **Voir Σ(), page 196.**

system() **Catalogue >**

Donne un système d'équations, présenté sous forme de liste. Vous pouvez également créer un système d'équation en utilisant un modèle.

$\text{solve}\left(\begin{cases} x+y=0 \\ x-y=8 \end{cases}, x, y\right)$ $x=4$ and $y=-4$

T

T (transposée) **Catalogue >**

*Matrix1***T**⇒*matrice*

Donne la transposée de la conjuguée de *Matrice1*.

T (transposée)

Catalogue >

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant `@t`.

tan()

Touche

tan(*Liste1***)⇒***liste*

tan(*List1***)** donne la liste des tangentes des éléments de *Liste1*.

Remarque : l'argument est interprété comme mesure d'angle en degrés, en grades ou en radians, suivant le mode angulaire sélectionné. Vous pouvez utiliser °, G ou ʳ pour ignorer temporairement le mode Angle sélectionné.

En mode Angle en degrés :

En mode Angle en grades :

En mode Angle en radians :

tan(*matriceMatrice1***)⇒***matriceCarrée*

Donne la tangente de la matrice *matriceCarrée1*. Ce calcul est différent du calcul de la tangente de chaque élément. Pour plus d'informations sur la méthode de calcul, reportez-vous à **cos()**.

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

En mode Angle en radians :

$$\tan\left(\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix}\right)$$

$$\begin{bmatrix}-28.2912 & 26.0887 & 11.1142\\ 12.1171 & -7.83536 & -5.48138\\ 36.8181 & -32.8063 & -10.4594\end{bmatrix}$$

$\tan^{-1}()$

Touche

$\tan^{-1}$(*Liste1***)⇒***liste*

$\tan^{-1}$(*List1***)** donne la liste des arcs tangentes des éléments de *Liste1*.

Remarque : donne le résultat en degrés, en grades ou en radians, suivant le mode angulaire utilisé.

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant `arctan(...)`.

En mode Angle en degrés :

$$\tan^{-1}(1) \qquad 45$$

En mode Angle en grades :

$$\tan^{-1}(1) \qquad 50$$

En mode Angle en radians :

$$\tan^{-1}(\{0,0.2,0.5\}) \qquad \{0,0.197396,0.463648\}$$

$\tan^{-1}$(*matriceCarrée1***)⇒***matriceCarrée*

En mode Angle en radians :

tan⁻¹() Touche trig

Donne l'arc tangente de la matrice *matriceCarrée1*. Ce calcul est différent du calcul de l'arc tangente de chaque élément. Pour plus d'informations sur la méthode de calcul, reportez-vous à **cos()**.

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

$$\tan^{-1}\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix}$$

$$\begin{bmatrix}-0.083658 & 1.26629 & 0.62263\\ 0.748539 & 0.630015 & -0.070012\\ 1.68608 & -1.18244 & 0.455126\end{bmatrix}$$

tanh() Catalogue >

tanh(*Liste1***)⇒***liste*

tanh(*Liste1***)** donne la liste des tangentes hyperboliques des éléments de *Liste1*.

tanh(*matriceCarrée1***)⇒***matriceCarrée*

Donne la tangente hyperbolique de la matrice *matriceCarrée1*. Ce calcul est différent du calcul de la tangente hyperbolique de chaque élément. Pour plus d'informations sur la méthode de calcul, reportez-vous à **cos()**.

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

En mode Angle en radians :

$$\tanh\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix}$$

$$\begin{bmatrix}-0.097966 & 0.933436 & 0.425972\\ 0.488147 & 0.538881 & -0.129382\\ 1.28295 & -1.03425 & 0.428817\end{bmatrix}$$

tanh⁻¹() Catalogue >

tanh⁻¹(*Liste1***)⇒***liste*

tanh⁻¹(*Liste1***)** donne la liste des arguments tangentes hyperboliques des éléments de *Liste1*.

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **`arctanh(`**...**`)`**.

tanh⁻¹ (*matriceCarrée1***)⇒***matriceCarrée*

En mode Format complexe Rectangulaire :

En mode Angle en radians et en mode Format complexe Rectangulaire :

tanh⁻¹() Catalogue >

Donne l'argument tangente hyperbolique de *matriceCarrée1*. Ce calcul est différent du calcul de l'argument tangente hyperbolique de chaque élément. Pour plus d'informations sur la méthode de calcul, reportez-vous à **cos()**.

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

$$\tanh^{-1}\left(\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix}\right)$$

$$\begin{bmatrix}-0.099353+0.164058\cdot i & 0.267834-1.4908\\ -0.087596-0.725533\cdot i & 0.479679-0.9473\\ 0.511463-2.08316\cdot i & -0.878563+1.7901\end{bmatrix}$$

Pour afficher le résultat entier, appuyez sur ▲, puis utilisez les touches ◄ et ► pour déplacer le curseur.

tCdf() Catalogue >

tCdf(*LimitInf***,***LimitSup***,***df***)⇒***nombre* si *LimitInf* et *LimitSup* sont des nombres, *liste* si *LimitInf* et *LimitSup* sont des listes

Calcule la fonction de répartition de la loi de Student-*t* à *df* degrés de liberté entre *LimitInf* et *LimitSup*.

Text Catalogue >

Text*chaîneinvite*[**,** *IndicAff*]

Commande de programmation : Marque une pause dans l'exécution du programme et affiche la chaîne de caractères *chaîneinvite* dans une boîte de dialogue.

Lorsque l'utilisation sélectionne **OK**, l'exécution du programme se poursuit.

L'argument optionnel *IndicAff* peut correspondre à n'importe quelle expression.

- Si *IndicAff* est omis ou a pour valeur **1**, le message est ajouté à l'historique de l'application Calculs.
- Si *IndicAff* a pour valeur **0**, le message n'est pas ajouté à l'historique.

Si le programme nécessite une réponse saisie par l'utilisateur, voir **Request**, page 134 ou **RequestStr**, page 136.

Définissez un programme qui marque une pause afin d'afficher cinq nombres aléatoires dans une boîte de dialogue.

Dans le modèle Prgm...EndPrgm, validez chaque ligne en appuyant sur ↵ à la place de enter. Sur le clavier de l'ordinateur, maintenez enfoncée la touche **Alt** tout en appuyant sur **Entrée**.

```
Define text_demo()=Prgm
  For i,1,5
    strinfo:="Random number " &
string(rand(i))
    Text strinfo
  EndFor
EndPrgm
```

Exécutez le programme :

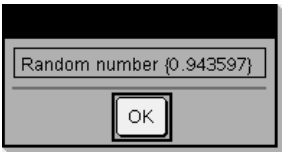
```
text_demo()
```

Text

Catalogue >

Remarque : vous pouvez utiliser cette commande dans un programme créé par l'utilisateur, mais pas dans une fonction.

Exemple de boîte de dialogue :



Then

Voir If, page 73.

tInterval

Catalogue >

tInterval *Liste*[**,***Fréq*[**,***CLevel*]]

(Entrée de liste de données)

tInterval $\bar{x}$**,***sx*,*n*[**,***CLevel*]

(Récapitulatif des statistiques fournies en entrée)

Calcule un intervalle de confiance *t*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.CLower, stat.CUpper	Intervalle de confiance pour une moyenne inconnue de population
stat.$\bar{x}$	Moyenne d'échantillon de la série de données suivant la loi normale aléatoire
stat.ME	Marge d'erreur
stat.df	Degrés de liberté
stat.σx	Écart-type d’échantillon
stat.n	Taille de la série de données avec la moyenne d'échantillon

tInterval_2Samp

Catalogue >

tInterval_2Samp *Liste1*,*Liste2*[,*Fréq1*[,*Freq2*[,*CLevel*[,*Group*]]]]

(Entrée de liste de données)

tInterval_2Samp $\bar{x}1$,*sx1*,*n1*,$\bar{x}2$,*sx2*,*n2*[,*CLevel*[,*Group*]]

(Récapitulatif des statistiques fournies en entrée)

Calcule un intervalle de confiance *t* sur 2 échantillons. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Group=**1** met en commun les variances ; *Groupe*=**0** ne met pas en commun les variances.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.CLower, stat.CUpper	Intervalle de confiance contenant la probabilité du niveau de confiance de la loi
stat.$\bar{x}$1-$\bar{x}$2	Moyennes d'échantillon des séries de données suivant la loi normale aléatoire
stat.ME	Marge d'erreur
stat.df	Degrés de liberté
stat.$\bar{x}$1, stat.$\bar{x}$2	Moyennes d'échantillon des séries de données suivant la loi normale aléatoire
stat.σx1, stat.σx2	Écarts-types d'échantillon pour *Liste 1* et *Liste 2*
stat.n1, stat.n2	Nombre d'échantillons dans les séries de données
stat.sp	Écart-type du groupe. Calculé lorsque Group = YES.

tPdf()

Catalogue >

tPdf(*ValX*,*df***)**⇒*nombre* si *ValX* est un nombre, *liste* si *ValX* est une liste

tPdf() Catalogue >

Calcule la densité de probabilité (pdf) de la loi de Student-*t* à *df* degrés de liberté en *ValX*.

trace() Catalogue >

Donne la trace (somme de tous les éléments de la diagonale principale) de *matriceCarrée*.

Try Catalogue >

Try
bloc1
Else
bloc2
EndTry

Exécute *bloc1*, à moins qu'une erreur ne se produise. L'exécution du programme est transférée au *bloc2* si une erreur se produit au *bloc1*. La variable système *errCode* contient le numéro d'erreur pour permettre au programme de procéder à une reprise sur erreur. Pour obtenir la liste des codes d'erreur, voir la section « Codes et messages d'erreur », page 233.

bloc1 et *bloc2* peuvent correspondre à une instruction unique ou à une série d'instructions séparées par le caractère ":".

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

Pour voir fonctionner les commandes **Try**, **ClrErr** et **PassErr**, saisissez le programme eigenvals() décrit à droite. Exécutez le programme en exécutant chacune des expressions suivantes.

```
Define prog1()=Prgm
  Try
  z:=z+1
  Disp "z incremented."
  Else
  Disp "Sorry, z undefined."
  EndTry
  EndPrgm
                                  Done
z:=1:prog1()
                  z incremented.
                                  Done
DelVar z:prog1()
                  Sorry, z undefined.
                                  Done
```

Définition du programme eigenvals(a,b)=Prgm

© Le programme eigenvals(A,B) présente les valeurs propres A·B

Try

Disp "A= ",a

$$eigenvals\left(\begin{bmatrix}-3\\-41\\5\end{bmatrix},\begin{bmatrix}-1 & 2 & -3.1\end{bmatrix}\right)$$

Remarque : voir aussi **ClrErr**, page 24 et **PassErr**, page 118.

```
  Disp "B= ",b
  Disp " "
  Disp "Eigenvalues of A·B are:",eigVl(a*b)
Else
  If errCode=230 Then
    Disp "Error: Product of A·B must be a
square matrix"
    ClrErr
  Else
    PassErr
  EndIf
EndTry
EndPrgm
```


tTest $\mu 0$**,***Liste*[**,***Fréq*[**,***Hypoth*]]

(Entrée de liste de données)

tTest $\mu 0$**,**$\bar{x}$**,***sx*,*n*,[*Hypoth*]

(Récapitulatif des statistiques fournies en entrée)

Teste une hypothèse pour une moyenne inconnue de population μ quand l'écart-type de population σ est inconnu. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Test de $H_0 : \mu = \mu 0$, en considérant que :

Pour $H_a : \mu < \mu 0$, définissez *Hypoth*<0

Pour $H_a : \mu \neq \mu 0$ (par défaut), définissez *Hypoth*=0

Pour $H_a : \mu > \mu 0$, définissez *Hypoth*>0

tTest

Catalogue >

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.t	$(\bar{x} - \mu 0) / (\text{stdev} / \text{sqrt}(n))$
stat.PVal	Plus petit seuil de signification permettant de rejeter l'hypothèse nulle
stat.df	Degrés de liberté
stat.$\bar{x}$	Moyenne d'échantillon de la série de données dans *Liste*
stat.sx	Écart-type d'échantillon de la série de données
stat.n	Taille de l'échantillon

tTest_2Samp

Catalogue >

tTest_2Samp *Liste1*,*Liste2*[,*Fréq1*[,*Fréq2*[,*Hypoth*[,*Group*]]]]

(Entrée de liste de données)

tTest_2Samp $\bar{x}$*1*,*sx1*,*n1*,$\bar{x}$*2*,*sx2*,*n2*[,*Hypoth*[,*Group*]]

(Récapitulatif des statistiques fournies en entrée)

Effectue un test *t* sur deux échantillons. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Test de H_0 : $\mu 1 = \mu 2$, en considérant que :

Pour H_a : $\mu 1 < \mu 2$, définissez *Hypoth*<0

Pour H_a : $\mu 1 \neq \mu 2$ (par défaut), définissez *Hypoth*=0

Pour H_a : $\mu 1 > \mu 2$, définissez *Hypoth*>0

Group=**1** met en commun les variances

Group=**0** ne met pas en commun les variances

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.t	Valeur normale type calculée pour la différence des moyennes
stat.PVal	Plus petit seuil de signification permettant de rejeter l'hypothèse nulle
stat.df	Degrés de liberté des statistiques t
stat.$\bar{x}$1, stat.$\bar{x}$2	Moyennes d'échantillon des séquences de données dans *Liste 1* et *Liste 2*
stat.sx1, stat.sx2	Écarts-types d'échantillon des séries de données dans *Liste 1* et *Liste 2*
stat.n1, stat.n2	Taille des échantillons
stat.sp	Écart-type du groupe. Calculé lorsque *Group*=1.

tvmFV() Catalogue >

tvmFV(*N*,*I*,*PV*,*Pmt*,[*PpY*],[*CpY*],[*PmtAt*]**)**⇒*valeur*

Fonction financière permettant de calculer la valeur acquise de l'argent.

Remarque : Les arguments utilisés dans les fonctions TVM sont décrits dans le tableau des arguments TVM, page 171. Voir également **amortTbl()**, page 7.

```
tvmFV(120,5,0,-500,12,12)        77641.1
```

tvmI() Catalogue >

tvmI(*N*,*PV*,*Pmt*,*FV*,[*PpY*],[*CpY*],[*PmtAt*]**)**⇒*valeur*

Fonction financière permettant de calculer le taux d'intérêt annuel.

Remarque : Les arguments utilisés dans les fonctions TVM sont décrits dans le tableau des arguments TVM, page 171. Voir également **amortTbl()**, page 7.

```
tvmI(240,100000,-1000,0,12,12)        10.5241
```

tvmN() Catalogue >

tvmN(*I*,*PV*,*Pmt*,*FV*,[*PpY*],[*CpY*],[*PmtAt*]**)**⇒*valeur*

Fonction financière permettant de calculer le nombre de périodes de versement.

Remarque : Les arguments utilisés dans les fonctions TVM sont décrits dans le tableau des arguments TVM, page 171. Voir également **amortTbl()**, page 7.

tvmN(5,0,-500,77641,12,12) 120.

tvmPmt() Catalogue >

tvmPmt(*N*,*I*,*PV*,*FV*,[*PpY*],[*CpY*],[*PmtAt*]**)**⇒*valeur*

Fonction financière permettant de calculer le montant de chaque versement.

Remarque : Les arguments utilisés dans les fonctions TVM sont décrits dans le tableau des arguments TVM, page 171. Voir également **amortTbl()**, page 7.

tvmPmt(60,4,30000,0,12,12) -552.496

tvmPV() Catalogue >

tvmPV(*N*,*I*,*Pmt*,*FV*,[*PpY*],[*CpY*],[*PmtAt*]**)**⇒*valeur*

Fonction financière permettant de calculer la valeur actuelle.

Remarque : Les arguments utilisés dans les fonctions TVM sont décrits dans le tableau des arguments TVM, page 171. Voir également **amortTbl()**, page 7.

tvmPV(48,4,-500,30000,12,12) -3426.7

Argument TVM*	Description	Type de données
N	Nombre de périodes de versement	nombre réel
I	Taux d'intérêt annuel	nombre réel
PV	Valeur actuelle	nombre réel
Pmt	Montant des versements	nombre réel

Argument TVM*	Description	Type de données
FV	Valeur acquise	nombre réel
PpY	Versements par an, par défaut=1	Entier> 0
CpY	Nombre de périodes de calcul par an, par défaut=1	Entier> 0
PmtAt	Versement dû à la fin ou au début de chaque période, par défaut=fin	entier (0=fin, 1=début)

***** Ces arguments de valeur temporelle de l'argent sont similaires aux noms des variables TVM (comme **tvm.pv** et **tvm.pmt**) utilisées par le solveur finance de l'application *Calculator*.Cependant, les fonctions financières n'enregistrent pas leurs valeurs ou résultats dans les variables TVM.

TwoVar

Catalogue >

TwoVar *X*, *Y*[, [*Fréq*] [, *Catégorie*, *Inclure*]]

Calcule des statistiques pour deux variables. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Toutes les listes doivent comporter le même nombre de lignes, à l'exception de *Inclure*.

X et *Y* sont des listes de variables indépendantes et dépendantes.

Fréq est une liste facultative de valeurs qui indiquent la fréquence. Chaque élément dans *Fréq* correspond à une fréquence d'occurrence pour chaque couple *X* et *Y*. Par défaut, cette valeur est égale à 1. Tous les éléments doivent être des entiers ≥ 0.

Catégorie est une liste de codes de catégories pour les couples *X* et *Y* correspondants.

Inclure est une liste d'un ou plusieurs codes de catégories. Seuls les éléments dont le code de catégorie figure dans cette liste sont inclus dans le calcul.

Tout élément vide dans les listes *X*, *Fréq* ou *Catégorie* a un élément vide correspondant dans l'ensemble des listes résultantes. Tout élément vide dans les listes *X1* à *X20* a un élément vide correspondant dans l'ensemble des listes résultantes. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

Variable de sortie	**Description**
stat.$\bar{x}$	Moyenne des valeurs x
stat. x	Somme des valeurs x
stat. x2	Somme des valeurs x2
stat.sx	Écart-type de l'échantillon de x
stat. x	Écart-type de la population de x
stat.n	Nombre de points de données
stat.$\bar{y}$	Moyenne des valeurs y
stat. y	Somme des valeurs y
stat. y^2	Somme des valeurs y2
stat.sy	Écart-type de y dans l'échantillon
stat. y	Écart-type de population des valeurs de y
stat. xy	Somme des valeurs x ·y
stat.r	Coefficient de corrélation
stat.MinX	Minimum des valeurs de x
stat.Q_1X	1er quartile de x
stat.MedianX	Médiane de x
stat.Q_3X	3ème quartile de x
stat.MaxX	Maximum des valeurs de x
stat.MinY	Minimum des valeurs de y
stat.Q_1Y	1er quartile de y
stat.MedY	Médiane de y
stat.Q_3Y	3ème quartile de y

Variable de sortie	Description
stat.MaxY	Maximum des valeurs y
stat. (x-)^2	Somme des carrés des écarts par rapport à la moyenne de x
stat. (y-)^2	Somme des carrés des écarts par rapport à la moyenne de y

U

unitV() Catalogue >

unitV(*Vecteur1***)**⇒*vecteur*

Donne un vecteur unitaire ligne ou colonne, en fonction de la nature de *Vecteur1*.

Vecteur1 doit être une matrice d'une seule ligne ou colonne.

unLock Catalogue >

unLock*Var1* [**,** *Var2*] [**,** *Var3*] ...

unLock*Var***.**

Déverrouille les variables ou les groupes de variables spécifiés. Les variables verrouillées ne peuvent être ni modifiées ni supprimées.

Voir **Lock**, page 92 et **getLockInfo()**, page 69.

a:=65	65
Lock *a*	*Done*
getLockInfo(*a*)	1
a:=75	"Error: Variable is locked."
DelVar *a*	"Error: Variable is locked."
Unlock *a*	*Done*
a:=75	75
DelVar *a*	*Done*

V

varPop() Catalogue >

varPop(*Liste*[**,** *listeFréq*]**)**⇒*expression*

Donne la variance de population de *Liste*.

Chaque élément de la liste *listeFréq* totalise le nombre d'occurrences de l'élément correspondant de *Liste*.

Remarque : *Liste* doit contenir au moins deux éléments.

Si un élément des listes est vide, il est ignoré et l'élément correspondant dans l'autre liste l'est également. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

varSamp(*Liste*[**,** *listeFréq*]**)**⇒*expression*

Donne la variance d'échantillon de *Liste*.

Chaque élément de la liste *listeFréq* totalise le nombre d'occurrences de l'élément correspondant de *Liste*.

Remarque : *Liste* doit contenir au moins deux éléments.

Si un élément des listes est vide, il est ignoré et l'élément correspondant dans l'autre liste l'est également. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

varSamp(*Matrice1*[**,** *matriceFréq*]**)**⇒*matrice*

Donne un vecteur ligne contenant la variance d'échantillon de chaque colonne de *Matrice1*.

Chaque élément de *matriceFréq* totalise le nombre d'occurrences de l'élément correspondant de *Matrice1*.

Remarque : *Matrice1* doit contenir au moins deux lignes.

Si un élément des matrices est vide, il est ignoré et l'élément correspondant dans l'autre matrice l'est également. Pour plus d'informations concernant les éléments vides, reportez-vous à la page 223.

$$\text{varSamp}\begin{bmatrix} 1 & 2 & 5 \\ -3 & 0 & 1 \\ .5 & .7 & 3 \end{bmatrix} \quad \begin{bmatrix} 4.75 & 1.03 & 4 \end{bmatrix}$$

$$\text{varSamp}\left(\begin{bmatrix} -1.1 & 2.2 \\ 3.4 & 5.1 \\ -2.3 & 4.3 \end{bmatrix}, \begin{bmatrix} 6 & 3 \\ 2 & 4 \\ 5 & 1 \end{bmatrix}\right) \quad \begin{bmatrix} 3.91731 & 2.08411 \end{bmatrix}$$

Wait

Catalogue >

Wait *tempsEnSecondes*

Suspend l'exécution pendant une durée de *tempsEnSecondes* secondes.

La commande **Wait** est particulièrement utile dans un programme qui a besoin de quelques secondes pour permettre aux données demandées d'être accessibles.

L'argument *tempsEnSecondes* doit être une expression qui s'évalue en une valeur décimale comprise entre 0 et 100. La commande arrondit cette valeur à 0,1 seconde près.

Pour annuler un **Wait** qui est en cours,

- **Calculatrice:** Maintenez la touche [on] enfoncée et appuyez plusieurs fois sur [enter].
- **Windows® :** Maintenez la touche **F12** enfoncée et appuyez plusieurs fois sur **Entrée**.
- **Macintosh® :** Maintenez la touche **F5** enfoncée et appuyez plusieurs fois sur **Entrée**.
- **iPad® :** L'application affiche une invite. Vous pouvez continuer à patienter ou annuler.

Remarque : Vous pouvez utiliser la commande **Wait** dans un programme créé par l'utilisateur, mais pas dans une fonction.

Pour définir un délai d'attente de 4 secondes :

```
Wait 4
```

Pour définir un délai d'attente d'une 1/2 seconde :

```
Wait 0.5
```

Pour définir un délai d'attente de 1,3 seconde à l'aide de la variable *seccompt* :

```
seccompt:=1.3
Wait seccompt
```

Cet exemple allume une DEL verte pendant 0,5 seconde puis l'éteint.

```
Send "SET GREEN 1 ON"
Wait 0.5
Send "SET GREEN 1 OFF"
```

warnCodes ()

Catalogue >

warnCodes(*Expr1***,** *VarÉtat***)⇒***expression*

Évalue l'expression *Expr1*, donne le résultat et stocke les codes de tous les avertissements générés dans la variable de liste *VarÉtat*. Si aucun avertissement n'est généré, cette fonction affecte une liste vide à *VarÉtat*.

warnCodes ( ) Catalogue >

Expr1 peut être toute expression mathématique TI-Nspire™ ou TI-Nspire™ CAS valide. *Expr1* ne peut pas être une commande ou une affectation.

VarÉtat doit être un nom de variable valide.

Pour la liste des codes d'avertissement et les messages associés, voir page 242.

when() Catalogue >

when(*Condition***,** *résultSiOui* [**,** *résultSiNon*][**,** *résultSiInconnu*]**)⇒***expression*

Donne *résultSiOui*, *résultSiNon* ou *résultSiInconnu*, suivant que la *Condition* est vraie, fausse ou indéterminée. Donne l'entrée si le nombre d'argument est insuffisant pour spécifier le résultat approprié.

Ne spécifiez pas *résultSiNon* ni *résultSiInconnu* pour obtenir une expression définie uniquement dans la région où *Condition* est vraie.

Utilisez **undef** *résultSiNon* pour définir une expression représentée graphiquement sur un seul intervalle.

when() est utile dans le cadre de la définition de fonctions récursives.

$\text{when}(x<0, x+3) \mid x=5$	undef

$\text{when}(n>0, n\cdot factoral(n-1), 1) \rightarrow factoral(n)$	*Done*
$factoral(3)$	6
$3!$	6

While

Catalogue >

While *Condition*
Bloc
EndWhile

Exécute les instructions contenues dans *Bloc* si *Condition* est vraie.

Bloc peut correspondre à une ou plusieurs instructions, séparées par un « : ».

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

```
Define sum_of_recip(n)=Func
                         Local i,tempsum
                         1→i
                         0→tempsum
                         While i≤n
                         tempsum+1/i→tempsum
                         i+1→i
                         EndWhile
                         Return tempsum
                         EndFunc
                                                Done
sum_of_recip(3)                                 11/6
```

X

xor

Catalogue >

BooleanExpr1 **xor** *BooleanExpr2* renvoie *expression booléenne*

BooleanList1 **xor** *BooleanList2* renvoie *liste booléenne*

BooleanMatrix1 **xor** *BooleanMatrix2* renvoie *matrice booléenne*

Donne true si *Expr booléenne1* est vraie et si *Expr booléenne2* est fausse, ou vice versa.

Donne false si les deux arguments sont tous les deux vrais ou faux. Donne une expression booléenne simplifiée si l'un des deux arguments ne peut être résolu vrai ou faux.

Remarque : voir **or**, page 115.

Entier1 **xor** *Entier2* ⇒ *entier*

```
true xor true                    false
5>3 xor 3>5                      true
```

En mode base Hex :

Important : utilisez le chiffre zéro et pas la lettre O.

```
0h7AC36 xor 0h3D5F               0h79169
```

Compare les représentations binaires de deux entiers, en appliquant un **xor** bit par bit. En interne, les deux entiers sont convertis en nombres binaires 64 bits signés. Lorsque les bits comparés correspondent, le résultat est 1 si dans l'un des deux cas (pas dans les deux) il s'agit d'un bit 1 ; le résultat est 0 si, dans les deux cas, il s'agit d'un bit 0 ou 1. La valeur donnée représente le résultat des bits et elle est affichée selon le mode Base utilisé.

Les entiers de tout type de base sont admis. Pour une entrée binaire ou hexadécimale, vous devez utiliser respectivement le préfixe 0b ou 0h. Tout entier sans préfixe est considéré comme un nombre en écriture décimale (base 10).

Si vous entrez un nombre dont le codage binaire signé dépasse 64 bits, il est ramené à l'aide d'une congruence dans la plage appropriée. Pour de plus amples informations, voir ▶**Base2**, page 17.

Remarque : voir **or**, page 115.

En mode base Bin :

0b100101 xor 0b100	0b100001

Remarque : une entrée binaire peut comporter jusqu'à 64 chiffres (sans compter le préfixe 0b) ; une entrée hexadécimale jusqu'à 16 chiffres.

Z

zInterval **Catalogue >**

zInterval σ**,***Liste*[**,***Fréq*[**,***CLevel*]]

(Entrée de liste de données)

zInterval σ**,**$\bar{x}$**,***n* [**,***CLevel*]

(Récapitulatif des statistiques fournies en entrée)

Calcule un intervalle de confiance *z*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.CLower, stat.CUpper	Intervalle de confiance pour une moyenne inconnue de population
stat.$\bar{x}$	Moyenne d'échantillon de la série de données suivant la loi normale aléatoire
stat.ME	Marge d'erreur
stat.sx	Écart-type d'échantillon
stat.n	Taille de la série de données avec la moyenne d'échantillon
stat.σ	Écart-type connu de population pour la série de données *Liste*

zInterval_1Prop

Catalogue >

zInterval_1Prop *x*,*n* [,*CLevel*]

Calcule un intervalle de confiance *z* pour une proportion. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

x est un entier non négatif.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.CLower, stat.CUpper	Intervalle de confiance contenant la probabilité du niveau de confiance de la loi
stat.$\hat{p}$	Proportion calculée de réussite
stat.ME	Marge d'erreur
stat.n	Nombre d'échantillons dans la série de données

zInterval_2Prop

Catalogue >

zInterval_2Prop *x1*,*n1*,*x2*,*n2*[,*CLevel*]

Calcule un intervalle de confiance *z* pour deux proportions. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

zInterval_2Prop

Catalogue >

x1 et *x2* sont des entiers non négatifs.

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.CLower, stat.CUpper	Intervalle de confiance contenant la probabilité du niveau de confiance de la loi
stat.$\hat{p}$Diff	Différence calculée entre les proportions
stat.ME	Marge d'erreur
stat.$\hat{p}$1	Proportion calculée sur le premier échantillon
stat.$\hat{p}$2	Proportion calculée sur le deuxième échantillon
stat.n1	Taille de l'échantillon dans la première série de données
stat.n2	Taille de l'échantillon dans la deuxième série de données

zInterval_2Samp

Catalogue >

zInterval_2Samp σ_1,σ_2,*Liste1*,*Liste2* [,*Fréq1* [,*Fréq2*, [*CLevel*]]]

(Entrée de liste de données)

zInterval_2Samp σ_1,σ_2,$\bar{x}1$,*n1*,$\bar{x}2$,*n2* [,*CLevel*]

(Récapitulatif des statistiques fournies en entrée)

Calcule un intervalle de confiance *z* sur deux échantillons. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.CLower, stat.CUpper	Intervalle de confiance contenant la probabilité du niveau de confiance de la loi

Variable de sortie	Description
stat.$\bar{x}$1-$\bar{x}$2	Moyennes d'échantillon des séries de données suivant la loi normale aléatoire
stat.ME	Marge d'erreur
stat.$\bar{x}$1, stat.$\bar{x}$2	Moyennes d'échantillon des séries de données suivant la loi normale aléatoire
stat.σx1, stat.σx2	Écarts-types d'échantillon pour *Liste 1* et *Liste 2*
stat.n1, stat.n2	Nombre d'échantillons dans les séries de données
stat.r1, stat.r2	Écart-type connu de population pour la série de données *Liste 1* et *Liste 2*

zTest

Catalogue >

zTest μ*0*,σ,*Liste*,[*Fréq*[,*Hypoth*]]

(Entrée de liste de données)

zTest μ*0*,σ,$\bar{x}$,*n*[,*Hypoth*]

(Récapitulatif des statistiques fournies en entrée)

Effectue un test *z* en utilisant la fréquence *listeFréq*. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Test de H_0 : μ = μ0, en considérant que :

Pour H_a : μ < μ0, définissez *Hypoth*<0

Pour H_a : μ ≠ μ0 (par défaut), définissez *Hypoth*=0

Pour H_a : μ > μ0, définissez *Hypoth*>0

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.z	($\bar{x}$ − μ0) / (σ / sqrt(n))
stat.P Value	Plus petite probabilité permettant de rejeter l'hypothèse nulle

Variable de sortie	Description
stat.$\bar{x}$	Moyenne d'échantillon de la série de données dans *Liste*
stat.sx	Écart-type d'échantillon de la série de données Uniquement donné pour l'entrée *Data*.
stat.n	Taille de l'échantillon

zTest_1Prop

Catalogue >

zTest_1Prop *p0*,*x*,*n*[,*Hypoth*]

Effectue un test *z* pour une proportion unique. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

x est un entier non négatif.

Test de H_0 : $p = p0$, en considérant que :

Pour H_a : $p > p0$, définissez *Hypoth*>0

Pour H_a : $p \neq p0$ *(par défaut)*, définissez *Hypoth*=0

Pour H_a : $p < p0$, définissez *Hypoth*<0

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.p0	Proportion de population hypothétique
stat.z	Valeur normale type calculée pour la proportion
stat.PVal	Plus petit seuil de signification permettant de rejeter l'hypothèse nulle
stat.$\hat{p}$	Proportion calculée sur l'échantillon
stat.n	Taille de l'échantillon

zTest_2Prop

Catalogue >

zTest_2Prop *x1*,*n1*,*x2*,*n2*[,*Hypoth*]

zTest_2Prop

Catalogue >

Calcule un test *z* pour deux proportions. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

x1 et *x2* sont des entiers non négatifs.

Test de H_0 : $p1 = p2$, en considérant que :

Pour H_a : $p1 > p2$, définissez *Hypoth*>0

Pour H_a : $p1 \neq p2$ *(par défaut)*, définissez *Hypoth*=0

Pour H_a : $p < p0$, définissez *Hypoth*<0

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	Description
stat.z	Valeur normale type calculée pour la différence des proportions
stat.PVal	Plus petit seuil de signification permettant de rejeter l'hypothèse nulle
stat.$\hat{p}1$	Proportion calculée sur le premier échantillon
stat.$\hat{p}2$	Proportion calculée sur le deuxième échantillon
stat.$\hat{p}$	Proportion calculée de l'échantillon mis en commun
stat.n1, stat.n2	Nombre d'échantillons pris lors des essais 1 et 2

zTest_2Samp

Catalogue >

zTest_2Samp σ_1,σ_2,*Liste1*,*Liste2*[,*Fréq1*[,*Fréq2*[,*Hypoth*]]]

(Entrée de liste de données)

zTest_2Samp σ_1,σ_2,$\bar{x}1$,*n1*,$\bar{x}2$,*n2*[,*Hypoth*]

(Récapitulatif des statistiques fournies en entrée)

Calcule un test *z* sur deux échantillons. Un récapitulatif du résultat est stocké dans la variable *stat.results*. (Voir page 156.)

Test de H_0 : $\mu1 = \mu2$, en considérant que :

Pour H_a : $\mu 1 < \mu 2$, définissez *Hypoth*<0

Pour H_a : $\mu 1 \neq \mu 2$ (par défaut), définissez *Hypoth*=0

Pour H_a : $\mu 1 > \mu 2$, *Hypoth*>0

Pour plus d'informations concernant les éléments vides dans une liste, reportez-vous à “Éléments vides”, page 223.

Variable de sortie	**Description**
stat.z	Valeur normale type calculée pour la différence des moyennes
stat.PVal	Plus petit seuil de signification permettant de rejeter l'hypothèse nulle
stat.$\bar{x}1$, stat.$\bar{x}2$	Moyennes d'échantillon des séquences de données dans *Liste 1* et *Liste 2*
stat.sx1, stat.sx2	Écarts-types d'échantillon des séries de données dans *Liste 1* et *Liste 2*
stat.n1, stat.n2	Taille des échantillons

Symboles

+ (somme) — Touche +

Donne la somme des deux arguments.

56	56
56+4	60
60+4	64
64+4	68
68+4	72

Liste1 **+** *Liste2*⇒*liste*

Matrice1 **+** *Matrice2*⇒*matrice*

Donne la liste (ou la matrice) contenant les sommes des éléments correspondants de *Liste1* et *Liste2* (ou *Matrice1* et *Matrice2*).

Les arguments doivent être de même dimension.

15+{10,15,20}	{25,30,35}
{10,15,20}+15	{25,30,35}

Remarque : utilisez **.+** pour ajouter une expression à chaque élément de la matrice.

$20+\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$	$\begin{bmatrix}21 & 2\\ 3 & 24\end{bmatrix}$

−(soustraction) — Touche −

Liste1 − *Liste2*⇒*liste*

Matrice1 − *Matrice2*⇒*matrice*

Soustrait chaque élément de *Liste2* (ou *Matrice2*) de l'élément correspondant de *Liste1* (ou *Matrice1*) et donne le résultat obtenu.

Les arguments doivent être de même dimension.

15−{10,15,20}	{5,0,−5}
{10,15,20}−15	{−5,0,5}

−(soustraction)

Touche [−]

Remarque : Utilisez **.−** pour soustraire une expression à chaque élément de la matrice.

$$20-\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \qquad \begin{bmatrix}19 & -2\\ -3 & 16\end{bmatrix}$$

·(multiplication)

Touche [×]

Donne le produit des deux arguments.

Liste1 ·*Liste2*⇒*liste*

Donne la liste contenant les produits des éléments correspondants de *Liste1* et *Liste2*.

Les listes doivent être de même dimension.

Matrice1 ·*Matrice2*⇒*matrice*

Donne le produit des matrices *Matrice1* et *Matrice2*.

Le nombre de colonnes de *Matrice1* doit être égal au nombre de lignes de *Matrice2*.

Remarque : Utilisez **.** ·pour multiplier une expression par chaque élément.

/ (division)

Touche [÷]

Remarque : voir aussi **Modèle Fraction**, page 1.

Liste1 / *Liste2*⇒*liste*

Donne la liste contenant les quotients de Liste1 *par Liste2*.

Les listes doivent être de même dimension.

Remarque : Utilisez **. /** pour diviser une expression par chaque élément.

$$\frac{\{1.,2,3\}}{\{4,5,6\}} \qquad \left\{0.25,\frac{2}{5},\frac{1}{2}\right\}$$

^ (puissance)

Touche [^]

Liste1 **^** *Liste2* ⇒ *liste*

Donne le premier argument élevé à la puissance du deuxième argument.

^ (puissance)

Touche ^

Remarque : voir aussi **Modèle Exposant**, page 1.

Dans le cas d'une liste, donne la liste des éléments de *Liste1* élevés à la puissance des éléments correspondants de *Liste2*.

Dans le domaine réel, les puissances fractionnaires possédant des exposants réduits avec des dénominateurs impairs utilise la branche réelle, tandis que le mode complexe utilise la branche principale.

$\{1,2,3,4\}^{-2}$ → $\left\{1,\frac{1}{4},\frac{1}{9},\frac{1}{16}\right\}$

matriceCarrée1 **^** *entier* ⇒ *matrice*

Donne *matriceCarrée1* élevée à la puissance de la valeur de l'*entier*.

matriceCarrée1 doit être une matrice carrée.

Si *entier* = ⁻1, calcule la matrice inverse.

Si *entier* < ⁻1, calcule la matrice inverse à une puissance positive appropriée.

$\begin{bmatrix}1&2\\3&4\end{bmatrix}^{2}$ → $\begin{bmatrix}7&10\\15&22\end{bmatrix}$

$\begin{bmatrix}1&2\\3&4\end{bmatrix}^{-1}$ → $\begin{bmatrix}-2&1\\\frac{3}{2}&\frac{-1}{2}\end{bmatrix}$

$\begin{bmatrix}1&2\\3&4\end{bmatrix}^{-2}$ → $\begin{bmatrix}\frac{11}{2}&\frac{-5}{2}\\\frac{-15}{4}&\frac{7}{4}\end{bmatrix}$

x² (carré)

Touche x²

Donne le carré de l'argument.

*Liste1*² ⇒ *liste*

Donne la liste comportant les carrés des éléments de *Liste1*.

matriceCarrée1 ²⇒ *matrice*

Donne le carré de la matrice *matriceCarrée1*. Ce calcul est différent du calcul du carré de chaque élément. Utilisez .^2 pour calculer le carré de chaque élément.

4^2 → 16

$\{2,4,6\}^2$ → $\{4,16,36\}$

$\begin{bmatrix}2&4&6\\3&5&7\\4&6&8\end{bmatrix}^2$ → $\begin{bmatrix}40&64&88\\49&79&109\\58&94&130\end{bmatrix}$

$\begin{bmatrix}2&4&6\\3&5&7\\4&6&8\end{bmatrix}.^{\wedge}2$ → $\begin{bmatrix}4&16&36\\9&25&49\\16&36&64\end{bmatrix}$

.+ (addition élément par élément)

Touches . +

Matrice1 **.+** *Matrice2* ⇒ *matrice*

Matrice1 **.+** *Matrice2* donne la matrice obtenue en effectuant la somme de chaque paire d'éléments correspondants de *Matrice1* et de *Matrice2*.

.

.− (soustraction élément par élément)

Touches . −

Matrice1 **.−** *Matrice2* ⇒ *matrice*

Matrice1 **.−***Matrice2* donne la matrice obtenue en calculant la différence entre chaque paire d'éléments correspondants de *Matrice1* et de *Matrice2*.

.

. ·(multiplication élément par élément)

Touches . ×

Matrice1 **.** · *Matrice2* ⇒ *matrice*

Matrice1 **.** · *Matrice2* donne la matrice obtenue en calculant le produit de chaque paire d'éléments correspondants de *Matrice1* et de *Matrice2*.

. / (division élément par élément)

Touches . ÷

Matrice1 **. /** *Matrice2* ⇒ *matrice*

Matrice1 **./** *Matrice2* donne la matrice obtenue en calculant le quotient de chaque paire d'éléments correspondants de *Matrice1* et de *Matrice2*.

.

.^ (puissance élément par élément)

Touches . ^

Matrice1 **.^** *Matrice2* ⇒ *matrice*

.^ (puissance élément par élément)

Touches . ^

Matrice1 **.^** *Matrice2* donne la matrice obtenue en élevant chaque élément de *Matrice1* à la puissance de l'élément correspondant de *Matrice2*.

-(opposé)

Touche (−)

-*Liste1* ⇒ *liste*

-*Matrice1* ⇒ *matrice*

Donne l'opposé de l'argument.

Dans le cas d'une liste ou d'une matrice, donne l'opposé de chacun des éléments.

Si l'argument est un entier binaire ou hexadécimal, la négation donne le complément à deux.

`En mode base Bin :`

Important : utilisez le chiffre zéro et pas la lettre O.

```
-0b100101
0b1111111111111111111111111111011
```

Pour afficher le résultat entier, appuyez sur ▲, puis utilisez les touches ◀ et ▶ pour déplacer le curseur.

% (pourcentage)

Touches ctrl 🕮

Liste1 **%** ⇒ *liste*

Matrice1 **%** ⇒ *matrice*

Donne $\frac{argument}{100}$

Dans le cas d'une liste ou d'une matrice, donne la liste ou la matrice obtenue en divisant chaque élément par 100.

Remarque: Pour afficher un résultat approximatif,

Unité : Appuyez sur ctrl enter.
Windows® : Appuyez sur **Ctrl+Entrée**.
Macintosh® : Appuyez sur ⌘**+Entrée**.
iPad® : Maintenez la touche **Entrée** enfoncée et sélectionnez ≈.

13%	0.13
{{1,10,100}}%	{0.01,0.1,1.}

= (égal à)

Touche =

Expr1 **=** *Expr2*⇒*Expression booléenne*

Liste1 **=** *Liste2*⇒ *Liste booléenne*

Matrice1 **=** *Matrice2*⇒ *Matrice booléenne*

Exemple de fonction qui utilise les symboles de test mathématiques : =, ≠, <, ≤, >, ≥

= (égal à)

Touche =

Donne true s'il est possible de vérifier que la valeur de *Expr1* est égale à celle de *Expr2*.

Donne false s'il est possible de déterminer que la valeur de *Expr1* n'est pas égale à celle de *Expr2*.

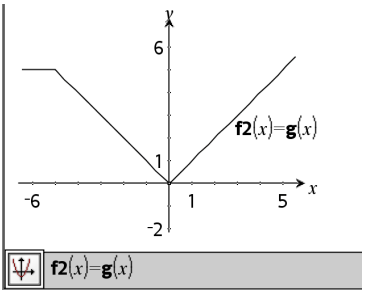
Dans les autres cas, donne une forme simplifiée de l'équation.

Dans le cas d'une liste ou d'une matrice, donne le résultat des comparaisons, élément par élément.

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

```
Define g(x)=Func
    If x≤-5 Then
    Return 5
    ElseIf x>-5 and x<0 Then
    Return -x
    ElseIf x≥0 and x≠10 Then
    Return x
    ElseIf x=10 Then
    Return 3
    EndIf
    EndFunc
                                Done
```

Résultat de la représentation graphique de g (x)



≠ (différent de)

Touches ctrl =

Expr1 ≠ *Expr2* ⇒ *Expression booléenne*

Liste1 ≠ *Liste2* ⇒ *Liste booléenne*

Matrice1 ≠ *Matrice2* ⇒ *Matrice booléenne*

Donne true s'il est possible de déterminer que la valeur de *Expr1* n'est pas égale à celle de *Expr2*.

Donne false s'il est possible de vérifier que la valeur de *Expr1* est égale à celle de *Expr2*.

Dans les autres cas, donne une forme simplifiée de l'équation.

Voir l'exemple fourni pour « = » (égal à).

≠ (différent de)

Touches ctrl =

Dans le cas d'une liste ou d'une matrice, donne le résultat des comparaisons, élément par élément.

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **/=**

< (inférieur à)

Touches ctrl =

Expr1 **<** *Expr2* ⇒ *Expression booléenne*

Liste1 **<** *Liste2* ⇒ *Liste booléenne*

Matrice1 **<** *Matrice2* ⇒ *Matrice booléenne*

Donne true s'il est possible de déterminer que la valeur de *Expr1* est strictement inférieure à celle de *Expr2*.

Donne false s'il est possible de déterminer que la valeur de *Expr1* est strictement supérieure ou égale à celle de *Expr2*.

Dans les autres cas, donne une forme simplifiée de l'équation.

Dans le cas d'une liste ou d'une matrice, donne le résultat des comparaisons, élément par élément.

Voir l'exemple fourni pour « = » (égal à).

≤ (inférieur ou égal à)

Touches ctrl =

Expr1 ≤ *Expr2* ⇒ *Expression booléenne*

Liste1 ≤ *Liste2* ⇒ *Liste booléenne*

Matrice1 ≤ *Matrice2* ⇒ *Matrice booléenne*

Donne true s'il est possible de déterminer que la valeur de *Expr1* est inférieure ou égale à celle de *Expr2*.

Donne false s'il est possible de déterminer que la valeur de *Expr1* est strictement supérieure à celle de *Expr2*.

Dans les autres cas, donne une forme simplifiée de l'équation.

Voir l'exemple fourni pour « = » (égal à).

≤ (inférieur ou égal à)

Touches ctrl =

Dans le cas d'une liste ou d'une matrice, donne le résultat des comparaisons, élément par élément.

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **<=**

> (supérieur à)

Touches ctrl =

Expr1 **>** *Expr2* ⇒ *Expression booléenne*

Voir l'exemple fourni pour « = » (égal à).

Liste1 **>** *Liste2* ⇒ *Liste booléenne*

Matrice1 **>** *Matrice2* ⇒ *Matrice booléenne*

Donne true s'il est possible de déterminer que la valeur de *Expr1* est supérieure à celle de *Expr2*.

Donne false s'il est possible de déterminer que la valeur de *Expr1* est strictement inférieure ou égale à celle de *Expr2*.

Dans les autres cas, donne une forme simplifiée de l'équation.

Dans le cas d'une liste ou d'une matrice, donne le résultat des comparaisons, élément par élément.

≥ (supérieur ou égal à)

Touches ctrl =

Expr1 ≥ *Expr2* ⇒ *Expression booléenne*

Voir l'exemple fourni pour « = » (égal à).

Liste1 ≥ *Liste2* ⇒ *Liste booléenne*

Matrice1 ≥ *Matrice2* ⇒ *Matrice booléenne*

Donne true s'il est possible de déterminer que la valeur de *Expr1* est supérieure ou égale à celle de *Expr2*.

Donne false s'il est possible de déterminer que la valeur de *Expr1* est inférieure ou égale à celle de *Expr2*.

Dans les autres cas, donne une forme simplifiée de l'équation.

≥ (supérieur ou égal à)

Touches ctrl =

Dans le cas d'une liste ou d'une matrice, donne le résultat des comparaisons, élément par élément.

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **>=**

⇒ (implication logique)

touches ctrl =

BooleanExpr1 ⇒ *BooleanExpr2* renvoie *expression booléenne*

BooleanList1 ⇒ *BooleanList2* renvoie *liste booléenne*

BooleanMatrix1 ⇒ *BooleanMatrix2* renvoie *matrice booléenne*

Integer1 ⇒ *Integer2* renvoie *entier*

5>3 or 3>5	true
5>3 ⇒ 3>5	false
3 or 4	7
3 ⇒ 4	-4
{1,2,3} or {3,2,1}	{3,2,3}
{1,2,3} ⇒ {3,2,1}	{-1,-1,-3}

Évalue l'expression **not** <argument1> **or** <argument2> et renvoie true (vrai) ou false (faux) ou une forme simplifiée de l'équation.

Pour les listes et matrices, renvoie le résultat des comparaisons, élément par élément.

Remarque : Vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **=>**

⇔ (équivalence logique, XNOR)

touches ctrl =

BooleanExpr1 ⇔ *BooleanExpr2* renvoie *expression booléenne*

BooleanList1 ⇔ *BooleanList2* renvoie *liste booléenne*

BooleanMatrix1 ⇔ *BooleanMatrix2* renvoie *matrice booléenne*

Integer1 ⇔ *Integer2* renvoie *entier*

5>3 xor 3>5	true
5>3 ⇔ 3>5	false
3 xor 4	7
3 ⇔ 4	-8
{1,2,3} xor {3,2,1}	{2,0,2}
{1,2,3} ⇔ {3,2,1}	{-3,-1,-3}

⇔ (équivalence logique, XNOR)

touches ctrl =

Renvoie la négation d'une opération booléenne **XOR** sur les deux arguments. Renvoie true (vrai) ou false (faux) ou une forme simplifiée de l'équation.

Pour les listes et matrices, renvoie le résultat des comparaisons, élément par élément.

Remarque : Vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **<=>**

! (factorielle)

Touche ?!

*Liste1***!** ⇒ *liste*

*Matrice1***!** ⇒ *matrice*

Donne la factorielle de l'argument.

Dans le cas d'une liste ou d'une matrice, donne la liste ou la matrice des factorielles de tous les éléments.

$5!$	120
$\{\{5,4,3\}\}!$	$\{120,24,6\}$
$\left(\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}\right)!$	$\begin{bmatrix}1 & 2\\ 6 & 24\end{bmatrix}$

& (ajouter)

Touches ctrl 📖

Chaîne1 **&** *Chaîne2* ⇒ *chaîne*

Donne une chaîne de caractères obtenue en ajoutant *Chaîne2* à *Chaîne1*.

"Hello "&"Nick"	"Hello Nick"

d() (dérivée)

Catalogue > 🔤

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **`derivative(`**...**`)`**.

√() (racine carrée)

Touches ctrl x²

√ **(***Liste1***)**⇒*liste*

Donne la racine carrée de l'argument.

√() (racine carrée)

Touches ctrl x²

Dans le cas d'une liste, donne la liste des racines carrées des éléments de *Liste1*.

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **`sqrt(...)`**

Remarque : voir aussi **Modèle Racine carrée**, page 1.

Π() (prodSeq)

Catalogue >

Π(*Expr1*, *Var*, *Début*, *Fin*)⇒*expression*

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **`prodSeq(...)`**.

Calcule *Expr1* pour chaque valeur de *Var* comprise entre *Début* et *Fin* et donne le produit des résultats obtenus.

Remarque : voir aussi **Modèle Produit (Π)**, page 5.

Π(*Expr1*, *Var*, *Début*, *Début*−1)⇒1

Π(*Expr1*, *Var*, *Début*, *Fin*)

⇒**1/**Π(*Expr1*, *Var*, *Fin*+1, *Début*−1) if *Début* < *Fin*−1

Les formules de produit utilisées sont extraites des références ci-dessous :

Ronald L. Graham, Donald E. Knuth et Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Reading, Massachusetts: Addison-Wesley, 1994.

$$\prod_{k=4}^{3}(k) \qquad 1$$

$$\prod_{k=4}^{1}\left(\frac{1}{k}\right) \qquad 6$$

$$\prod_{k=4}^{1}\left(\frac{1}{k}\right)\cdot\prod_{k=2}^{4}\left(\frac{1}{k}\right) \qquad \frac{1}{4}$$

Σ() (sumSeq)

Catalogue >

Σ(*Expr1*, *Var*, *Début*, *Fin*)⇒*expression*

Remarque : vous pouvez insérer cette fonction à partir du clavier en entrant **`sumSeq(...)`**.

Σ() (sumSeq) — Catalogue >

Calcule *Expr1* pour chaque valeur de *Var* comprise entre *Début* et *Fin* et donne la somme des résultats obtenus.

Remarque : voir aussi **Modèle Somme**, page 5.

Σ(*Expr1***,** *Var***,** *Début***,** *Fin*−*1***)**⇒0

Σ(*Expr1***,** *Var***,** *Début***,** *Fin***)**

⇒-**Σ(***Expr1***,** *Var***,** *Fin*+*1***,** *Début*−1**)** if *Fin* < *Début*−*1*

Le formules d'addition utilisées sont extraites des références ci-dessous :

Ronald L. Graham, Donald E. Knuth et Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Reading, Massachusetts: Addison-Wesley, 1994.

$\sum_{k=4}^{3}(k)$ → 0

$\sum_{k=4}^{1}(k)$ → -5

$\sum_{k=4}^{1}(k)+\sum_{k=2}^{4}(k)$ → 4

ΣInt() — Catalogue >

ΣInt(*NPmt1***,** *NPmt2***,** *N***,** *I***,** *PV***,**[*Pmt*]**,** [*FV*]**,** [*PpY*]**,** [*CpY*]**,** [*PmtAt*]**,** [*valArrondi*]**)**⇒*valeur*

ΣInt
(
NPmt1
,*NPmt2***,***tblAmortissement***)**⇒*valeur*

Fonction d'amortissement permettant de calculer la somme des intérêts au cours d'une plage de versements spécifiée.

NPmt1 et *NPmt2* définissent le début et la fin de la plage de versements.

N, *I*, *PV*, *Pmt*, *FV*, *PpY*, *CpY* et *PmtAt* sont décrits dans le tableau des arguments TVM, page 171.

- Si vous omettez *Pmt*, il prend par défaut la valeur *Pmt*=**tvmPmt** **(***N***,***I***,***PV***,***FV***,***PpY***,***CpY***,***PmtAt***)**.
- Si vous omettez *FV*, il prend par

ΣInt(1,3,12,4.75,20000,,,12,12) → -218.11

tbl:=amortTbl(12,12,4.75,20000,,,12,12)

0	0.	0.	20000.
1	-79.17	-1630.69	18369.3
2	-72.71	-1637.15	16732.2
3	-66.23	-1643.63	15088.5
4	-59.73	-1650.13	13438.4
5	-53.19	-1656.67	11781.7
6	-46.64	-1663.22	10118.5
7	-40.05	-1669.81	8448.7
8	-33.44	-1676.42	6772.28
9	-26.81	-1683.05	5089.23
10	-20.14	-1689.72	3399.51
11	-13.46	-1696.4	1703.11
12	-6.74	-1703.12	-0.01

ΣInt(1,3,*tbl*) → -218.11

défaut la valeur *FV*=0.

- Les valeurs par défaut pour *PpY*, *CpY* et *PmtAt* sont les mêmes que pour les fonctions TVM.

valArrondi spécifie le nombre de décimales pour arrondissement. Valeur par défaut=2.

ΣInt(*NPmt1,NPmt2,tblAmortissement***)** calcule la somme de l'intérêt sur la base du tableau d'amortissement *tblAmortissement*. L'argument *tblAmortissement* doit être une matrice au format décrit à **tblAmortissement()**, page 7.

Remarque : voir également ΣPrn() ci dessous et **Bal()**, page 16.

ΣPrn() Catalogue >

ΣPrn(*NPmt1***,** *NPmt2***,** *N***,** *I***,** *PV***,** [*Pmt*]**,** [*FV*]**,** [*PpY*]**,** [*CpY*]**,** [*PmtAt*]**,** [*valArrondi*]**)⇒***valeur*

ΣPrn
(
NPmt1
,*NPmt2***,***tblAmortissement***)⇒***valeur*

Fonction d'amortissement permettant de calculer la somme du capital au cours d'une plage de versements spécifiée.

NPmt1 et *NPmt2* définissent le début et la fin de la plage de versements.

N, *I*, *PV*, *Pmt*, *FV*, *PpY*, *CpY* et *PmtAt* sont décrits dans le tableau des arguments TVM, page 171.

- Si vous omettez *Pmt*, il prend par défaut la valeur *Pmt*=**tvmPmt (***N,I,PV,FV,PpY,CpY,PmtAt***)**.
- Si vous omettez *FV*, il prend par défaut la valeur *FV*=0.
- Les valeurs par défaut pour *PpY*, *CpY* et *PmtAt* sont les mêmes que pour les

ΣPrn(1,3,12,4.75,20000,,,12,12) -4911.47

tbl:=amortTbl(12,12,4.75,20000,,,12,12)

0	0.	0.	20000.
1	-79.17	-1630.69	18369.3
2	-72.71	-1637.15	16732.2
3	-66.23	-1643.63	15088.5
4	-59.73	-1650.13	13438.4
5	-53.19	-1656.67	11781.7
6	-46.64	-1663.22	10118.5
7	-40.05	-1669.81	8448.7
8	-33.44	-1676.42	6772.28
9	-26.81	-1683.05	5089.23
10	-20.14	-1689.72	3399.51
11	-13.46	-1696.4	1703.11
12	-6.74	-1703.12	-0.01

ΣPrn(1,3,*tbl*) -4911.47

fonctions TVM.

valArrondi spécifie le nombre de décimales pour arrondissement. Valeur par défaut=2.

ΣPrn(*NPmt1,NPmt2,tblAmortissement***)** calcule la somme du capital sur la base du tableau d'amortissement *tblAmortissement*. L'argument *tblAmortissement* doit être une matrice au format décrit à **tblAmortissement()**, page 7.

Remarque : voir également ΣInt() ci-dessus et **Bal()**, page 16.

(indirection)

Touches ctrl

*ChaîneNomVar*

Fait référence à la variable *ChaîneNomVar*. Permet d'utiliser des chaînes de caractères pour créer des noms de variables dans une fonction.

Crée ou fait référence à la variable xyz.

$10 \rightarrow r$	10
$"r" \rightarrow s1$	"r"
$\#s1$	10

Donne la valeur de la variable (r) dont le nom est stocké dans la variable s1.

E (notation scientifique)

Touche EE

*mantisse***E***exposant*

Saisit un nombre en notation scientifique. Ce nombre est interprété sous la forme *mantisse* × $10^{exposant}$.

Conseil : pour entrer une puissance de 10 sans passer par un résultat de valeur décimale, utilisez 10^*entier*.

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant **@E**. Par exemple, entrez **2.3@E4** pour avoir 2.3**E**4.

23000.	23000.
2300000000.+4.1E15	4.1E15
$3 \cdot 10^4$	30000

g (grades)

Touche 1

*Liste1*ᵍ ⇒*liste*

*Matrice1*ᵍ ⇒*matrice*

Cette fonction permet d'utiliser un angle en grades en mode Angle en degrés ou en radians.

En mode Angle en radians, multiplie *Expr1* par π/200.

En mode Angle en degrés, multiplie *Expr1* par g/100.

En mode Angle en grades, donne *Expr1* inchangée.

Remarque : vous pouvez insérer ce symbole à partir du clavier de l'ordinateur en entrant `@g`.

En mode Angle en degrés, grades ou radians :

ʳ (radians)

Touche 1

*Liste1*ʳ ⇒*liste*

*Matrice1*ʳ ⇒*matrice*

Cette fonction permet d'utiliser un angle en radians en mode Angle en degrés ou en grades.

En mode Angle en degrés, multiplie l'argument par 180/π.

En mode Angle en radians, donne l'argument inchangé.

En mode Angle en grades, multiplie l'argument par 200/π.

Conseil : utilisez ʳ si vous voulez forcer l'utilisation des radians dans une définition de fonction quel que soit le mode dominant lors de l'utilisation de la fonction.

Remarque : vous pouvez insérer ce symbole à partir du clavier de l'ordinateur en entrant `@r`.

En mode Angle en degrés, grades ou radians :

° (degré)

Touche 1

Liste1°⇒*liste*

Matrice1°⇒*matrice*

Cette fonction permet d'utiliser un angle en degrés en mode Angle en grades ou en radians.

En mode Angle en radians, multiplie l'argument par π/180.

En mode Angle en degrés, donne l'argument inchangé.

En mode Angle en grades, multiplie l'argument par 10/9.

Remarque : vous pouvez insérer ce symbole à partir du clavier de l'ordinateur en entrant **@d**.

En mode Angle en degrés, grades ou radians :

En mode Angle en radians :

$$\cos\left\{\left\{0,\frac{\pi}{4},90°,30.12°\right\}\right\}$$
$$\{1.,0.707107,0.,0.864976\}$$

°, ', '' (degré/minute/seconde)

Touches ctrl 🕮

dd°*mm*'*ss.ss*''⇒*expression*

*dd*Nombre positif ou négatif

*mm*Nombre positif ou nul

*ss.ss*Nombre positif ou nul

Donne *dd*+(*mm*/60)+(*ss.ss*/3600).

Ce format d'entrée en base 60 permet :-

- d'entrer un angle en degrés/minutes/secondes quel que soit le mode angulaire utilisé.
- d'entrer un temps exprimé en heures/minutes/secondes.

Remarque : faites suivre ss.ss de deux apostrophes ('') et non de guillemets (").

En mode Angle en degrés :

25°13'17.5"	25.2215
25°30'	$\frac{51}{2}$

∠ (angle)

Touches ctrl 🕮

[*Rayon*,∠θ_*Angle*]⇒*vecteur*

(entrée polaire)

En mode Angle en radians et avec le Format vecteur réglé sur :

rectangulaire

∠ (angle)

Touches ctrl 📖

[*Rayon*,∠θ_*Angle*,*Valeur_Z*]⇒*vecteur*

(entrée cylindrique)

cylindrique

[*Rayon*,∠θ_*Angle*,∠θ_*Angle*]⇒*vecteur*

(entrée sphérique)

sphérique

Donne les coordonnées sous forme de vecteur, suivant le réglage du mode Format Vecteur : rectangulaire, cylindrique ou sphérique.

Remarque : vous pouvez insérer ce symbole à partir du clavier de l'ordinateur en entrant **@<**.

(*Grandeur* ∠ *Angle*)⇒*valeurComplexe*

(entrée polaire)

Saisit une valeur complexe en coordonnées polaires (r∠θ). L'*Angle* est interprété suivant le mode Angle sélectionné.

En mode Angle en radians et en mode Format complexe Rectangulaire :

$$5+3\cdot i-\left(10\angle\frac{\pi}{4}\right) \qquad -2.07107-4.07107\cdot i$$

_ (trait bas considéré comme élément vide)

Voir "Éléments vides", page 223.

10^()

Catalogue > 📖

10^ (*Liste1***)**⇒*liste*

Donne 10 élevé à la puissance de l'argument.

Dans le cas d'une liste, donne 10 élevé à la puissance des éléments de *Liste1*.

10^(*matriceCarrée1***)**⇒*matriceCarrée*

Donne 10 élevé à la puissance de *matriceCarrée1*. Ce calcul est différent du calcul de 10 élevé à la puissance de chaque élément. Pour plus d'informations sur la méthode de calcul, reportez-vous à **cos()**.

$$10^{\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix}} \qquad \begin{bmatrix}1.14336\text{E}7 & 8.17155\text{E}6 & 6.67589\text{E}6\\ 9.95651\text{E}6 & 7.11587\text{E}6 & 5.81342\text{E}6\\ 7.65298\text{E}6 & 5.46952\text{E}6 & 4.46845\text{E}6\end{bmatrix}$$

10^() — Catalogue >

matriceCarrée1 doit être diagonalisable. Le résultat contient toujours des chiffres en virgule flottante.

^-1 (inverse) — Catalogue >

Liste1 **^-1**⇒*liste*

Donne l'inverse de l'argument.

Dans le cas d'une liste, donne la liste des inverses des éléments de *Liste1*.

matriceCarrée1 **^-1**⇒*matriceCarrée*

Donne l'inverse de *matriceCarrée1*.

matriceCarrée1 doit être une matrice carrée non singulière.

| (opérateur "sachant que") — touches ctrl

Expr **|** *ExprBooléen1* [**and***ExprBooléen2*]...

Expr **|** *ExprBooléen1* [**or***ExprBooléen2*]...

Le symbole (« | ») est utilisé comme opérateur binaire. L'opérande à gauche du symbole | est une expression. L'opérande à droite du symbole | spécifie une ou plusieurs relations destinées à affecter la simplification de l'expression. Plusieurs relations après le symbole | peuvent être reliées au moyen d'opérateurs logiques « **and** » ou « **or** ».

L'opérateur "sachant que" fournit trois types de fonctionnalités de base :

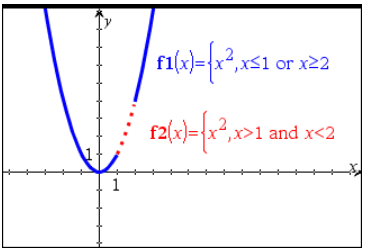
- Substitutions
- Contraintes d'intervalle
- Exclusions

| (opérateur "sachant que")

touches ctrl

Les substitutions se présentent sous la forme d'une égalité, telle que x=3 ou y=sin(x). Pour de meilleurs résultats, la partie gauche doit être une variable simple. *Expr* | *Variable* = *valeur* substituera une *valeur* à chaque occurrence de *Variable* dans *Expr*.

Les contraintes d'intervalle se présentent sous la forme d'une ou plusieurs inéquations reliées par des opérateurs logiques « **and** » ou « **or** ». Les contraintes d'intervalle permettent également la simplification qui autrement pourrait ne pas être valide ou calculable.



Les exclusions utilisent l'opérateur « différent de » (/= ou ≠) pour exclure une valeur spécifique du calcul.

solve(x^2−1=0, x)|x≠1 x=−1

→ (stocker)

Touche ctrl var

Remarque : vous pouvez insérer cet opérateur à partir du clavier de l'ordinateur en entrant `=:` comme un raccourci. Par exemple, tapez **`pi/4 =: Mavar`**.

:= (assigner)

Touches ctrl

Var **:=** *Liste*

Var **:=** *Matrice*

Fonction(*Param1*,...) **:=** *Expr*

Fonction(*Param1*,...) **:=** *Liste*

Fonction(*Param1*,...) **:=** *Matrice*

© (commentaire)

Touches ctrl 📖

© [*texte*]

© traite *texte* comme une ligne de commentaire, vous permettant d'annoter les fonctions et les programmes que vous créez.

© peut être utilisé au début ou n'importe où dans la ligne. Tous les caractères situés à droite de **©**, jusqu'à la fin de la ligne, sont considérés comme partie intégrante du commentaire.

Remarque pour la saisie des données de l'exemple : Pour obtenir des instructions sur la saisie des définitions de fonction ou de programme sur plusieurs lignes, consultez la section relative à la calculatrice dans votre guide de produit.

```
Define g(n)=Func
              © Declare variables
              Local i,result
              result:=0
              For i,1,n,1 ©Loop n times
              result:=result+i^2
              EndFor
              Return result
              EndFunc
                                        Done
g(3)                                    14
```

0b, 0h

Touches 0 B, **touches** 0 H

0b *nombreBinaire*

0h *nombreHexadécimal*

Indique un nombre binaire ou hexadécimal, respectivement. Pour entrer un nombre binaire ou hexadécimal, vous devez utiliser respectivement le préfixe 0b ou 0h, quel que soit le mode Base utilisé. Un nombre sans préfixe est considéré comme décimal (base 10).

Le résultat est affiché en fonction du mode Base utilisé.

En mode base Dec :

```
0b10+0hF+10                             27
```

En mode base Bin :

```
0b10+0hF+10                        0b11011
```

En mode base Hex :

```
0b10+0hF+10                            0h1B
```

TI-Nspire™ CX II - Commandes graphiques

Ceci est un document d'appoint au Guide de référence de la TI-Nspire™ et de la TI-Nspire™ CAS. Toutes les instructions de la TI-Nspire™ CX II seront intégrées et publiées dans la version 5.1 du Guide de référence de la TI-Nspire™ et de la TI-Nspire™ CAS.

Programmation en mode graphique

De nouvelles commandes de programmation graphique ont été ajoutées aux unités TI-Nspire™ CX II et aux applications pour ordinateurs TI-Nspire™.

Les unités TI-Nspire™ CX II basculeront sur ce mode graphique pour exécuter les commandes graphiques avant de revenir au contexte d'exécution du programme initial.

L'écran affiche « En cours d'exécution » sur la barre supérieure pendant que le programme s'exécute. « Terminé » sera affiché à la fin du programme. Il suffit d'appuyer sur une touche quelconque pour faire sortir le système du mode graphique.

- Le passage au mode graphique est automatiquement déclenché lorsqu'une des commandes graphiques est trouvée durant l'exécution d'un programme en TI-Basic.
- Ce passage aura lieu uniquement lors de l'exécution d'un programme dans Calculs, dans un classeur ou dans Calculs dans le scratchpad
- La sortie du mode graphique se produit à la fin de l'exécution du programme.
- Le mode graphique est uniquement disponible sur les unités TI-Nspire™ CX II et dans la vue Unité du logiciel pour ordinateur TI-Nspire™ CX II. Il n'est pas disponible dans la vue Classeur sur PC ou sur Mac.
    - Si une commande graphique est trouvée durant l'exécution d'un programme TI-Basic dans un contexte incorrect, un message d'erreur sera affiché et l'exécution du programme TI-Basic sera interrompue.

Écran de représentation graphique

L'écran de représentation graphique contient une zone d'en-tête dans laquelle les commandes graphiques ne peuvent pas écrire.

La zone de tracé de l'écran graphique sera réinitialisée (couleur = 255,255,255) à son ouverture.

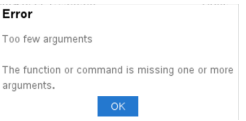
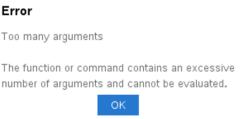
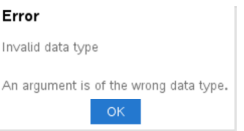
Écran de représentation graphique	**Par défaut**
Hauteur	212
Largeur	318
Couleur	blanc : 255,255,255

Vue et paramètres par défaut

- Les icônes d'état de la barre supérieure (voyant de batterie, verrouillage examen, indicateur de réseau etc.) ne sont pas visibles durant l'exécution d'un programme graphique.
- Couleur de trait par défaut : Noir (0,0,0)
- Style de stylo par défaut - normal, continu
  - Épaisseur : 1 (fin), 2 (normal), 3 (épais)
  - Style 1 (continu), 2 (tirets), 3 (pointillés)
- Toutes les commandes de tracé utiliseront les paramètres courants pour la couleur et le trait (valeurs par défaut ou définies via des commandes TI-Basic).
- La police du texte ne peut pas être modifiée.
- Toute sortie sur l'écran graphique sera dessinée dans une fenêtre dont la taille correspond à la zone de tracé de l'écran graphique. Toute sortie qui dépasse cette zone de tracé délimitée ne sera pas représentée. Aucun message d'erreur ne sera affiché.
- Les coordonnées (x, y) spécifiées par les commandes de tracé sont définies de façon à ce que (0,0) représente le coin supérieur gauche de la zone de tracé de l'écran graphique.
  - **Exceptions :**
    - Pour l'instruction **DrawText**, les coordonnées indiquées comme paramètres désignent l'angle inférieur gauche de la zone de délimitation du texte.
    - **SetWindow** utilise l'angle inférieur gauche de l'écran.
- Tous les paramètres (arguments) des commandes peuvent être fournis sous forme d'expressions qui sont évaluées à des nombres arrondis à l'entier le plus proche.

Messages d'erreur de l'écran graphique

Un message d'erreur sera affiché si la validation échoue.

Message d'erreur	Description	Afficher
Erreur Syntaxe	Si des erreurs de syntaxe sont détectées, un message d'erreur s'affiche et le curseur est placé, dans la mesure du possible, au niveau de la première erreur pour que vous puissiez la rectifier.	
Erreur Nombre insuffisant d'arguments	Un ou plusieurs arguments de la fonction ou de la commande n'ont pas été spécifiés	
Erreur Trop d'arguments	La fonction ou la commande est impossible à évaluer car elle contient trop d'arguments.	
Erreur Type de données incorrect	Le type de donnée de l'un des arguments est incorrect	

Commandes non valides dans le mode graphique

Certaines commandes ne sont pas autorisées une fois que le programme passe en mode graphique. Si ces commandes sont rencontrées en mode graphique, une erreur sera affichée et l'exécution du programme s'arrêtera.

Commande non autorisée	Message d'erreur
Request	La fonction Request ne peut pas être exécutée en mode graphique
RequestStr	La fonction RequestStr ne peut pas être exécutée en mode graphique
Texte	La fonction Text ne peut pas être exécutée en mode graphique

Les commandes qui affichent du texte dans Calculs - **disp** et **dispAt** - sont prises en charge dans le contexte graphique. Le texte de ces commandes sera envoyé à l'écran Calculs (et non pas sur l'écran graphique) et sera visible à la fin du programme, lorsque le système revient à l'application Calculs.

Supprimer

Catalogue > CXII

Clear *x, y, largeur, hauteur*

Efface tout l'écran si aucun paramètre n'est spécifié.

Si *x, y, largeur, hauteur* sont spécifiés, le rectangle spécifié par ces paramètres sera effacé.

```
Supprimer

Efface la totalité de l'écran


Clear 10,10,100,50

Efface une zone rectangulaire dont
le sommet supérieur gauche a pour
coordonnées (10,10), une largeur
égale à 100 et une hauteur à 50.
```

DrawArc

Catalogue > CXII

DrawArc *x, y, largeur, hauteur, startAngle, arcAngle*

Trace un arc dans le rectangle spécifié, avec les angles de départ et d'arc fournis.

x, y : coordonnées du sommet supérieur gauche du rectangle de délimitation

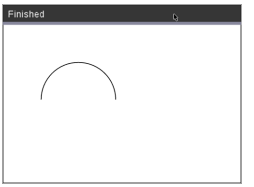
largeur, *hauteur* : dimensions du rectangle de délimitation

L'argument « arc angle » définit l'angle de balayage de l'arc.

Ces paramètres (arguments) peuvent être fournis sous forme d'expressions dont le résultat est arrondi à l'entier le plus proche.

```
DrawArc 20,20,100,100,0,90
```


```
DrawArc 50,50,100,100,0,180
```



Voir également : FillArc

DrawCircle

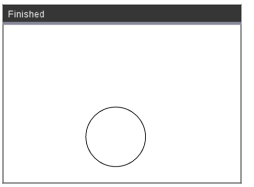
Catalogue > CXII

DrawCircle *x, y, rayon*

x, y : coordonnées du centre

rayon : rayon du cercle

```
DrawCircle 150,150,40
```



Voir également : FillCircle

DrawLine

Catalogue > CXII

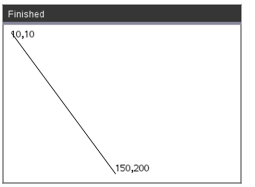
DrawLine *x1, y1, x2, y2*

Trace un segment d'extrémités *(x1, y1) et (x2, y2)*.

Expressions dont le résultat est arrondi à l'entier le plus proche.

Limites de l'écran : Si les coordonnées spécifiées impliquent qu'une partie du segment soit tracée en dehors de l'écran graphique, cette partie sera tronquée sans qu'aucun message d'erreur ne soit affiché.

```
DrawLine 10,10,150,200
```



DrawPoly

Catalogue > CXII

Les instructions ont deux variantes :

DrawPoly *xlist, ylist*

ou

DrawPoly *x1, y1, x2, y2, x3, y3...xn, yn*

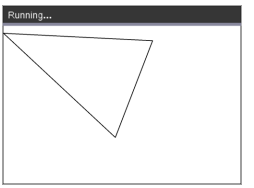
Remarque : DrawPoly *xlist, ylist*
Shape reliera*x1, y1* à *x2, y2*, *x2, y2* à *x3, y3*et ainsi de suite.

Remarque : DrawPoly *x1, y1, x2, y2, x3, y3...xn, yn*
xn, yn ne seront **PAS** reliés automatiquement à *x1, y1*.

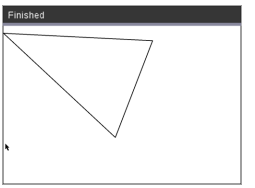
Expressions retournant une liste de nombres réels à virgule flottante
xlist, ylist

Expressions évaluées à un nombre réel à virgule flottante
x1, y1...xn, yn = coordonnées des sommets du polygone

```
xlist:={0,200,150,0}
ylist:={10,20,150,10}
DrawPoly xlist,ylist
```




```
DrawPoly 0,10,200,20,150,150,0,10
```



DrawPoly

Catalogue > CXII

Remarque : DrawPoly : Permet de spécifier les dimensions (largeur/ hauteur) par rapport aux segments tracés.
Les segments sont tracés dans une zone de délimitation autour des coordonnés spécifiées et dimensionnés de façon à ce que la taille réelle du polygone tracé soit supérieure à la largeur et à la hauteur indiquées.

Voir également : FillPoly

DrawRect

Catalogue > CXII

DrawRect *x, y, largeur, hauteur*

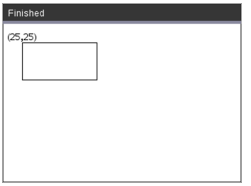
x, y : coordonnées du sommet supérieur gauche du rectangle

hauteur, *largeur* : hauteur et largeur du rectangle (rectangle tracé vers le bas et vers la droite à partir des coordonnées de départ).

Remarque : Les segments sont tracés dans une zone de délimitation autour des coordonnés spécifiées dont les dimensions font que la taille réelle du rectangle tracé sera supérieure à la largeur et à la hauteur indiquées.

Voir également : FillRect

```
DrawRect 25,25,100,50
```



DrawText

Catalogue > CXII

DrawText *x, y, exprOrString1 [,exprOrString2]...*

x, y : coordonnées du texte affiché

Trace le texte inclus dans *exprOrString* à l'emplacement spécifié par les coordonnées *x, y* indiquées.

```
DrawText 50,50,"Hello World"
```

Les règles pour *exprOrString* sont les mêmes que pour **Disp** – **DrawText** peut avoir plusieurs arguments.

FillArc

Catalogue > CXII

FillArc *x, y, largeur, hauteur, startAngle, arcAngle*

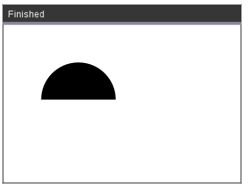
x, y : coordonnées du sommet supérieur gauche du rectangle de délimitation

Trace et remplit un arc dans le rectangle défini, en utilisant l'angle de départ et l'angle de balayage indiqués.

La couleur de remplissage par défaut est le noir. La couleur de remplissage peut être définie via la commande SetColor

L'argument « arc angle » définit l'angle de balayage de l'arc

```
FillArc 50,50,100,100,0,180
```



FillCircle

Catalogue > CXII

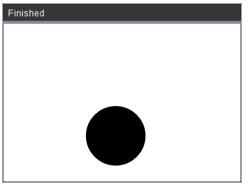
FillCircle *x, y, rayon*

x, y : coordonnées du centre

Trace et remplit un cercle de centre et de rayon spécifiés.

La couleur de remplissage par défaut est le noir. La couleur de remplissage peut être définie via la commande SetColor.

```
FillCircle 150,150,40
```




```
Ici !
```

FillPoly

Catalogue > CXII

FillPoly *xlist, ylist*

ou

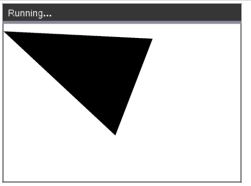
FillPoly *x1, y1, x2, y2, x3, y3...xn, yn*

Remarque : Le trait et la couleur sont définis par SetColor et SetPen

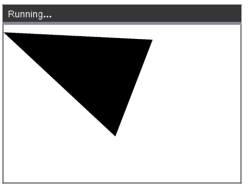
```
xlist:={0,200,150,0}
ylist:={10,20,150,10}
FillPoly xlist,ylist
```

FillPoly

Catalogue > CXII




```
FillPoly 0,10,200,20,150,150,0,10
```



FillRect

Catalogue > CXII

FillRect *x, y, largeur, hauteur*

x, y : coordonnées du sommet supérieur gauche du rectangle

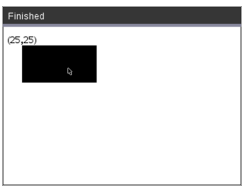
largeur, *hauteur* : largeur et hauteur du rectangle

Trace et remplit un rectangle dont le sommet supérieur gauche a pour coordonnées les valeurs spécifiées (*x*,*y*)

La couleur de remplissage par défaut est le noir. La couleur de remplissage peut être définie via la commande SetColor

Remarque : Le trait et la couleur sont définis par SetColor et SetPen

```
FillRect 25,25,100,50
```



getPlatform() **Catalogue > CXII**

getPlatform()

```
getPlatform()                    "dt"
```

Renvoie :
« dt » sur les applications logicielles pour ordinateur
« hh » sur les unités TI-Nspire™ CX
« ios » sur l'appli TI-Nspire™ CX pour iPad®

PaintBuffer

Catalogue > CXII

PaintBuffer

Dessine le contenu du cache graphique sur l'écran

Cette commande s'utilise en conjonction avec UseBuffer pour augmenter la vitesse d'affichage sur l'écran lorsque le programme génère de multiples objets graphiques.

```
UseBuffer
For n,1,10
x:=randInt(0,300)
y:=randInt(0,200)
radius:=randInt(10,50)
Wait 0,5
DrawCircle x, y, rayon
EndFor
PaintBuffer
Ce programme affichera les 10
cercles simultanément.
Si l'instruction « UseBuffer » est
retirée, chaque cercle sera affiché
lorsqu'il est tracé.
```

Voir également : UseBuffer

PlotXY *x, y, forme*

x, y : coordonnées du tracé de la forme

forme : un nombre compris entre 1 et 13 qui indique la forme

1 - Cercle plein

2 - Cercle vide

3 - Carré plein

4 - Carré vide

5 - Croix

6 - Plus

7 - Fin

8 - point moyen, plein

9 - point moyen, vide

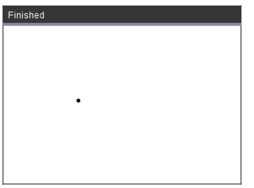
10 - point large, plein

11 - point large, vide

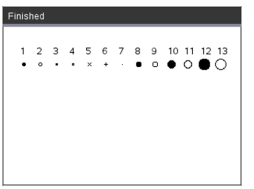
12 - point extra large, plein

13 - point extra large, vide

```
PlotXY 100,100,1
```




```
For n,1,13
DrawText 1+22*n,40,n
PlotXY 5+22*n,50,n
EndFor
```



SetColor

Catalogue > CXII

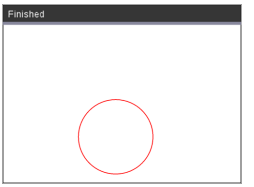
SetColor

valeur rouge, valeur vert, valeur bleu

Les valeurs valides pour le rouge, le vert et le bleu sont comprises entre 0 et 255

Définit la couleur pour les commandes de tracé suivantes.

```
SetColor 255,0,0
DrawCircle 150,150,100
```



SetPen

Catalogue > CXII

SetPen

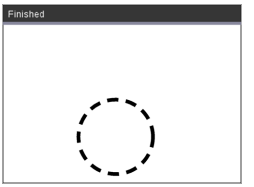
épaisseur, style

épaisseur : 1 <= épaisseur <= 3|1 est le plus fin, 3 est le plus épais

style : 1 = Continu, 2 = Tirets, 3 = Pointillés

Définit le style du stylo pour les commandes de tracé suivantes

```
SetPen 3,3
DrawCircle 150,150,50
```



SetWindow

Catalogue > CXII

SetWindow

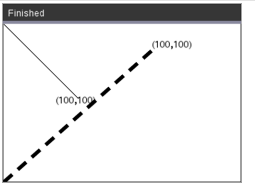
xMin, xMax, yMin, yMax

Établit une fenêtre logique qui correspond à la zone de représentation graphique Tous les paramètres sont obligatoires.

Si une partie de l'objet tracé se situe en dehors de la fenêtre, le résultat sera tronqué (non affiché) sans qu'aucun message d'erreur ne soit affiché.

```
SetWindow 0,160,0,120
Définit les coordonnées de l'angle
inférieur gauche de la fenêtre de
sortie en 0,0 avec une largeur de
160 et une hauteur de 120
DrawLine 0,0,100,100
SetWindow 0,160,0,120
SetPen 3,3
DrawLine 0,0,100,100
```

Si xmin est supérieur ou égal à xmax ou si ymin est supérieur ou égal à ymax, un message d'erreur s'affiche.



Tout objet tracé avant une instruction SetWindow ne sera pas retracé dans la nouvelle configuration.

Pour restaurer les paramètres par défaut de la fenêtre, utilisez :

SetWindow 0,0,0,0

UseBuffer

Catalogue > CXII

UseBuffer

Envoie vers la mémoire tampon de l'écran graphique au lieu d'afficher à l'écran (pour améliorer les performances)

Cette instruction est utilisée avec PaintBuffer pour accélérer l'affichage sur l'écran lorsque le programme génère de multiples objets graphiques.

Avec UseBuffer, tous les graphiques sont affichés uniquement après l'exécution de la commande PaintBuffer suivante.

UseBuffer n'a besoin d'être appelée qu'une seule fois dans le programme : chaque instruction PaintBuffer n'a pas besoin d'avoir une instruction UseBuffer correspondante.

Voir également : PaintBuffer

```
UseBuffer
For n,1,10
x:=randInt(0,300)
y:=randInt(0,200)
radius:=randInt(10,50)
Wait 0,5
DrawCircle x, y, rayon
EndFor
PaintBuffer
```

Ce programme affichera les 10 cercles simultanément.

Si l'instruction « UseBuffer » est retirée, chaque cercle sera affiché lorsqu'il est tracé.

Éléments vides

Lors de l'analyse de données réelles, il est possible que vous ne disposiez pas toujours d'un jeu complet de données. TI-Nspire™ vous permet d'avoir des éléments de données vides pour vous permettre de disposer de données presque complètes plutôt que d'avoir à tout recommencer ou à supprimer les données incomplètes.

Vous trouverez un exemple de données impliquant des éléments vides dans le chapitre Tableur et listes, sous « Représentation graphique des données de tableau ».

La fonction **delVoid()** vous permet de supprimer les éléments vides d'une liste, tandis que la fonction **isVoid()** vous offre la possibilité de tester si un élément est vide. Pour plus de détails, voir **delVoid()**, page 40 et **isVoid()**, page 80.

Remarque : Pour entrer un élément vide manuellement dans une expression, tapez « _ » ou le mot clé **`void`**. Le mot clé **`void`** est automatiquement converti en caractère « _ » lors du calcul de l'expression. Pour saisir le caractère « _ » sur la calculatrice, appuyez sur [ctrl] [␣].

Calculs impliquant des éléments vides

La plupart des calculs impliquant des éléments vides génère des résultats vides. Reportez-vous à la liste des cas spéciaux ci-dessous.

$$|\_| \qquad \_$$

$$\gcd(100,\_) \qquad \_$$

$$3+\_ \qquad \_$$

$$\{5,\_,10\}-\{3,6,9\} \qquad \{2,\_,1\}$$

Arguments de liste contenant des éléments vides

Les fonctions et commandes suivantes ignorent (passent) les éléments vides rencontrés dans les arguments de liste.

count, **countIf**, **cumulativeSum**, **freqTable▸list**, **frequency**, **max**, **mean**, **median**, **product**, **stDevPop**, **stDevSamp**, **sum**, **sumIf**, **varPop** et **varSamp**, ainsi que les calculs de régression, **OneVar**, **TwoVar** et les statistiques **FiveNumSummary**, les intervalles de confiance et les tests statistiques.

$$\text{sum}(\{2,\_,3,5,6.6\}) \qquad 16.6$$

$$\text{median}(\{1,2,\_,\_,\_,3\}) \qquad 2$$

$$\text{cumulativeSum}(\{1,2,\_,4,5\}) \qquad \{1,3,\_,7,12\}$$

$$\text{cumulativeSum}\left(\begin{bmatrix}1 & 2\\ 3 & \_\\ 5 & 6\end{bmatrix}\right) \qquad \begin{bmatrix}1 & 2\\ 4 & \_\\ 9 & 8\end{bmatrix}$$

Arguments de liste contenant des éléments vides

SortA et **SortD** déplacent tous les éléments vides du premier argument au bas de la liste.

$\{5,4,3,\_,1\} \rightarrow list1$	$\{5,4,3,\_,1\}$
$\{5,4,3,2,1\} \rightarrow list2$	$\{5,4,3,2,1\}$
SortA *list1*,*list2*	*Done*
list1	$\{1,3,4,5,\_\}$
list2	$\{1,3,4,5,2\}$

$\{1,2,3,\_,5\} \rightarrow list1$	$\{1,2,3,\_,5\}$
$\{1,2,3,4,5\} \rightarrow list2$	$\{1,2,3,4,5\}$
SortD *list1*,*list2*	*Done*
list1	$\{5,3,2,1,\_\}$
list2	$\{5,3,2,1,4\}$

Dans les regressions, la présence d'un élément vide dans la liste X ou Y génère un élément vide correspondant dans le résidu.

$l1:=\{1,2,3,4,5\}$: $l2:=\{2,\_,3,5,6.6\}$	$\{2,\_,3,5,6.6\}$
LinRegMx *l1*,*l2*	*Done*
stat.Resid	$\{0.434286,\_,-0.862857,-0.011429,0.44\}$
stat.XReg	$\{1.,\_,3.,4.,5.\}$
stat.YReg	$\{2.,\_,3.,5.,6.6\}$
stat.FreqReg	$\{1.,\_,1.,1.,1.\}$

L'omission d'une catégorie dans les calculs de régression génère un élément vide correspondant dans le résidu.

$l1:=\{1,3,4,5\}$: $l2:=\{2,3,5,6.6\}$	$\{2,3,5,6.6\}$
$cat:=\{\text{"M","M","F","F"}\}$: $incl:=\{\text{"F"}\}$	$\{\text{"F"}\}$
LinRegMx *l1*,*l2*,1,*cat*,*incl*	*Done*
stat.Resid	$\{\_,\_,0.,0.\}$
stat.XReg	$\{\_,\_,4.,5.\}$
stat.YReg	$\{\_,\_,5.,6.6\}$
stat.FreqReg	$\{\_,\_,1.,1.\}$

Une fréquence 0 dans les calculs de régression génère un élement vide correspondant dans le résidu.

$l1:=\{1,3,4,5\}$: $l2:=\{2,3,5,6.6\}$	$\{2,3,5,6.6\}$
LinRegMx *l1*,*l2*,$\{1,0,1,1\}$	*Done*
stat.Resid	$\{0.069231,\_,-0.276923,0.207692\}$
stat.XReg	$\{1.,\_,4.,5.\}$
stat.YReg	$\{2.,\_,5.,6.6\}$
stat.FreqReg	$\{1.,\_,1.,1.\}$

Raccourcis de saisie d'expressions mathématiques

Les raccourcis vous permettent de saisir directement des éléments d'expressions mathématiques sans utiliser le Catalogue ni le Jeu de symboles. Par exemple, pour saisir l'expression $\sqrt{6}$, vous pouvez taper **sqrt(6)** dans la ligne de saisie. Lorsque vous appuyez sur enter, l'expression **sqrt(6)** est remplacée par $\sqrt{6}$. Certains raccourcis peuvent s'avérer très utiles aussi bien sur la calculatrice qu'à partir du clavier de l'ordinateur. Certains sont plus spécifiquement destinés à être utilisés à partir du clavier de l'ordinateur.

Sur la calculatrice ou le clavier de l'ordinateur

Pour saisir :	Utilisez le raccourci :
π	**pi**
θ	**theta**
∞	**infinity**
$\leq$	**<=**
$\geq$	**>=**
$\neq$	**/=**
$\Rightarrow$ (implication logique)	**=>**
$\Leftrightarrow$ (équivalence logique, XNOR)	**<=>**
$\rightarrow$ (opérateur de stockage)	**:=**
| | (valeur absolue)	**abs(...)**
$\sqrt{}$()	**sqrt(...)**
Σ() (Modèle Somme)	**sumSeq(...)**
Π() (Modèle Produit)	**prodSeq(...)**
sin^{-1}(), cos^{-1}(), ...	**arcsin(...)**, **arccos(...)**, ...
Δ**List()**	**deltaList(...)**

Sur le clavier de l'ordinateur

Pour saisir :	Utilisez le raccourci :
i (le nombre complexe)	**@i**

Pour saisir :	**Utilisez le raccourci :**
e (base du logarithme népérien e)	**@e**
E (notation scientifique)	**@E**
^{T} (transposée)	**@t**
^{r} (radians)	**@r**
$^{\circ}$ (degré)	**@d**
^{g} (grades)	**@g**
$\angle$ (angle)	**@<**
▸ (conversion)	**@>**
▸Decimal, **▸approxFraction()**, et ainsi de suite.	**@>Decimal**, **@>approxFraction()**, et ainsi de suite.

Hiérarchie de l'EOS™ (Equation Operating System)

Cette section décrit l'EOS™ (Equation Operating System) qu'utilise le labo de maths TI-Nspire™. Avec ce système, la saisie des nombres, des variables et des fonctions est simple et directe. Le logiciel EOS™ évalue les expressions et les équations en utilisant les groupements à l'aide de parenthèses et en respectant l'ordre de priorité décrit ci-dessous.

Ordre d'évaluation

Niveau	Opérateur
1	Parenthèses ( ), crochets [ ], accolades { }
2	Indirection (#)
3	Appels de fonction
4	Opérateurs en suffixe : degrés-minutes-secondes (°,',"), factoriel (!), pourcentage (%), radian (ʳ), indice ([ ]), transposée (^{T})
5	Élévation à une puissance, opérateur de puissance (^)
6	Négation (-)
7	Concaténation de chaîne (&)
8	Multiplication (•), division (/)
9	Addition (+), soustraction (-)
10	Relations d'égalité : égal à (=), différent de (≠ ou /=), inférieur à (<), inférieur ou égal à (≤ ou <=), supérieur à (>), supérieur ou égal à (≥ ou >=)
11	**not** logique
12	**and** logique
13	Logique **or**
14	**xor**, **nor**, **nand**
15	Implication logique (⇒)
16	Équivalence logique, XNOR (⇔)
17	Opérateur "sachant que" (« | »)
18	Stocker (→)

Parenthèses, crochets et accolades

Toutes les opérations entre parenthèses, crochets ou accolades sont calculées en premier lieu. Par exemple, dans l'expression 4(1+2), l'EOS™ évalue en premier la partie de l'expression entre parenthèses, 1+2, puis multiplie le résultat, 3, par 4.

Le nombre de parenthèses, crochets et accolades ouvrants et fermants doit être identique dans une équation ou une expression. Si tel n'est pas le cas, un message d'erreur s'affiche pour indiquer l'élément manquant. Par exemple, (1+2)/(3+4 génère l'affichage du message d'erreur “) manquante”.

Remarque : Parce que le logiciel TI-Nspire™ vous permet de définir des fonctions personnalisées, un nom de variable suivi d'une expression entre parenthèses est considéré comme un « appel de fonction » et non comme une multiplication implicite. Par exemple, a(b+c) est la fonction a évaluée en b+c. Pour multiplier l'expression b+c par la variable a, utilisez la multiplication explicite : a*(b+c).

Indirection

L'opérateur d'indirection (#) convertit une chaîne en une variable ou en un nom de fonction. Par exemple, #(“x”&”y”&”z”) crée le nom de variable « xyz ». Cet opérateur permet également de créer et de modifier des variables à partir d'un programme. Par exemple, si 10→r et “r”→s1, donc #s1=10.

Opérateurs en suffixe

Les opérateurs en suffixe sont des opérateurs qui suivent immédiatement un argument, comme 5!, 25 % ou 60°15' 45". Les arguments suivis d'un opérateur en suffixe ont le niveau de priorité 4 dans l'ordre d'évaluation. Par exemple, dans l'expression 4^3!, 3! est évalué en premier. Le résultat, 6, devient l'exposant de 4 pour donner 4096.

Élévation à une puissance

L'élévation à la puissance (^) et l'élévation à la puissance élément par élément (.^) sont évaluées de droite à gauche. Par exemple, l'expression 2^3^2 est évaluée comme 2^(3^2) pour donner 512. Ce qui est différent de (2^3)^2, qui donne 64.

Négation

Pour saisir un nombre négatif, appuyez sur (−) suivi du nombre. Les opérations et élévations à la puissance postérieures sont évaluées avant la négation. Par exemple, le résultat de −x2 est un nombre négatif et $-9^2 = -81$. Utilisez les parenthèses pour mettre un nombre négatif au carré, comme $(-9)^2$ qui donne 81.

Contrainte (« | »)

L'argument qui suit l'opérateur "sachant que" (« | ») applique une série de contraintes qui affectent l'évaluation de l'argument qui précède l'opérateur.

Fonctions de programmation TI-Basic sur TI-Nspire CX II

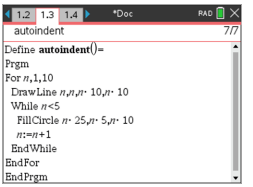
Auto-indentation dans l'Éditeur de programmes

L'Éditeur de programmes de la TI-Nspire™ indente désormais les instructions dans un bloc de commandes.

Les blocs de commandes sont If/EndIf, For/EndFor, While/EndWhile, Loop/EndLoop, Try/EndTry

L'éditeur indente automatiquement les commandes qui se trouvent dans un bloc d'instructions. L'instruction de fin de bloc sera alignée avec l'instruction de début de bloc.

L'exemple ci-dessous illustre l'indentation automatique dans les instructions de bloc imbriquées.



Les fragments de code qui sont copiés -collés conservent leur indentation originale.

Un programme créé avec une version précédente du logiciel conservera son indentation originale à l'ouverture.

Messages d'erreur améliorés pour TI-Basic

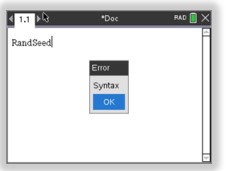
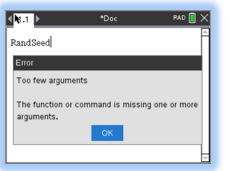
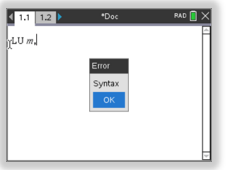
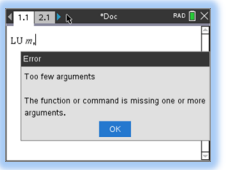
Erreurs

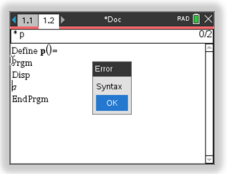
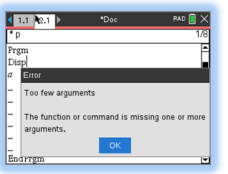
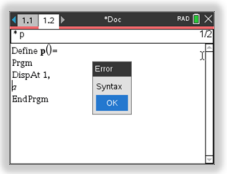
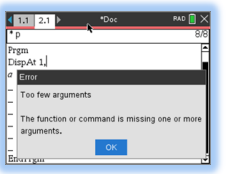
Condition d'erreur	**Nouveau message**
Erreur dans une instruction conditionnelle (**If/While**)	L'une des conditions a renvoyé une valeur qui n'était ni **VRAI** ni **FAUX** **REMARQUE :** Le curseur étant désormais placé sur la ligne où se trouve l'erreur, nous n'avons plus besoin d'indiquer si l'erreur se trouvait dans une instruction « **If** » ou une instruction « **While** ».
Instruction **EndIf** manquante	L'instruction de fin devrait être **EndIf**, mais une instruction de fin différente a été trouvée
Instruction **Endfor** manquante	L'instruction de fin devrait être **EndFor**, mais une instruction de fin différente a été trouvée
Instruction **EndWhile**manquante	L'instruction de fin devrait être **EndWhile**, mais une instruction de fin différente a été trouvée

Condition d'erreur	Nouveau message
Instruction **EndLoop** manquante	L'instruction de fin devrait être **EndLoop**, mais une instruction de fin différente a été trouvée
Instruction **EndTry** manquante	L'instruction de fin devrait être **EndTry**, mais une instruction de fin différente a été trouvée
« **Then** » manquant après**If** <condition>	Instruction **If..Then** manquante
« **Then** » manquant après**ElseIf** <condition>	Instruction **Then** manquante dans le bloc : **ElseIf**
En cas d'instruction « **Then** », « **Else** » ou « **ElseIf** » trouvée en dehors des blocs de contrôle.	Instruction **Else** invalide en dehors des blocs : **If..Then..EndIf** ou **Try..EndTry**
« **ElseIf** » apparaît en dehors d'un bloc « **If..Then..EndIf** »	Instruction **ElseIf** invalide en dehors du bloc : **If..Then..EndIf**
« **Then** » apparaît en dehors d'un bloc « **If....EndIf** »	Instruction **Then** invalide en dehors du bloc : **If..EndIf**

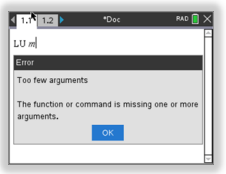
Erreurs de syntaxe

Si des instructions qui attendent un ou plusieurs arguments sont appelées avec un nombre insuffisant d'arguments, une erreur « **Nombre insuffisant d'arguments** » sera générée au lieu d'une « **erreur de syntaxe** »

Comportement actuel	Nouveau comportements de la CX II
	
	

Comportement actuel	Nouveau comportements de la CX II
	
	

Remarque : Lorsqu'une liste d'arguments incomplète n'est pas suivie d'une virgule, le message d'erreur est : « Nombre insuffisant d'arguments » Idem que pour les versions précédentes.



Constantes et valeurs

Le tableau suivant liste les constantes ainsi que leurs valeurs qui sont disponibles lors de la réalisation de conversions d'unités. Elles peuvent être saisies manuellement ou sélectionnées depuis la liste **Constantes** dans **Utilitaires > Conversions d'unité** (Unité nomade : Appuyez sur 📖 **3**).

Constante	Nom	Valeur
_c	Vitesse de la lumière	299792458 _m/_s
_Cc	Constante de Coulomb	8987551792.261 _m/_F
_Fc	Constante de Faraday	96485.33212 _coul/_mol
_g	Accélération de la pesanteur	9.80665 _m/_s^2
_Gc	Constante de gravitation	6.6743E-11 _m^3/_kg/_s^2
_h	Constante de Planck	6.62607015E-34 _J _s
_k	Constante de Boltzmann	1.380649E-23 _J/_°K
_μ0	Perméabilité du vide	1.25663706212E-6 _N/_A^2
_μb	Magnéton de Bohr	9.274009994E-24 _J _m^2/_Wb
_Me	Masse de l'électron	9.1093837015E-31 _kg
_Mμ	Masse du muon	1.883531627E-28 _kg
_Mn	Masse du neutron	1.67492749804E-27 _kg
_Mp	Masse du proton	1.67262192369E-27 _kg
_Na	Nombre d'Avogadro	6.02214076E23 /_mol
_q	Charge de l'électron	1.602176634E-19 _coul
_Rb	Rayon de Bohr	5.29177210903E-11 _m
_Rc	Constante molaire des gaz	8.314462618 _J/_mol/_°K
_Rdb	Constante de Rydberg	10973731.568160/_m
_Re	Rayon de l'électron	2.8179403262E-15 _m
_u	Masse atomique	1.6605390666E-27 _kg
_Vm	Volume molaire	2.241396954E-2 _m^3/_mol
_ε0	Permittivité du vide	8.8541878128E-12 _F/_m
_σ	Constante de Stefan-Boltzmann	5.670367E-8 _W/_m^2/_°K^4
_ϕ0	Quantum de flux magnétique	2.067833831E-15 _Wb

Codes et messages d'erreur

En cas d'erreur, le code correspondant est assigné à la variable *errCode*. Les programmes et fonctions définis par l'utilisateur peuvent être utilisés pour analyser *errCode* et déterminer l'origine de l'erreur. Pour obtenir un exemple d'utilisation de *errCode*, reportez-vous à l'exemple 2 fourni pour la commande **Try**, page 167.

Remarque : certaines erreurs ne s'appliquent qu'aux produits TI-Nspire™ CAS, tandis que d'autres ne s'appliquent qu'aux produits TI-Nspire™.

Code d'erreur	Description
10	La fonction n'a pas retourné de valeur.
20	Le test n'a pas donné de résultat VRAI ou FAUX. En général, les variables indéfinies ne peuvent pas être comparées. Par exemple, le test If a<b génère cette erreur si a ou b n'est pas défini lorsque l'instruction If est exécutée.
30	L'argument ne peut pas être un nom de dossier.
40	Erreur d'argument
50	Argument inadapté Deux arguments ou plus doivent être de même type.
60	L'argument doit être une expression booléenne ou un entier.
70	L'argument doit être un nombre décimal.
90	L'argument doit être une liste.
100	L'argument doit être une matrice.
130	L'argument doit être une chaîne de caractères.
140	L'argument doit être un nom de variable. Assurez-vous que ce nom : • ne commence pas par un chiffre, • ne contienne ni espaces ni caractères spéciaux, • n'utilise pas le tiret de soulignement ou le point de façon incorrecte, • ne dépasse pas les limitations de longueur. Pour plus d'informations à ce sujet, reportez-vous à la section Calculs dans la documentation.
160	L'argument doit être une expression.
165	Piles trop faibles pour envoi/réception Installez des piles neuves avant toute opération d'envoi ou de réception.
170	Bornes

Code d'erreur	Description
	Pour définir l'intervalle de recherche, la limite inférieure doit être inférieure à la limite supérieure.
180	Arrêt de calcul Une pression sur la touche esc ou on a été détectée au cours d'un long calcul ou lors de l'exécution d'un programme.
190	Définition circulaire Ce message s'affiche lors des opérations de simplification afin d'éviter l'épuisement total de la mémoire lors d'un remplacement infini de valeurs dans une variable en vue d'une simplification. Par exemple, a+1->a, où a représente une variable indéfinie, génère cette erreur.
200	Condition invalide Par exemple, solve(3x^2-4=0,x) | x<0 or x>5 génère ce message d'erreur car “or” est utilisé à la place de “and” pour séparer les contraintes.
210	Type de données incorrect Le type de l'un des arguments est incorrect.
220	Limite dépendante
230	Dimension Un index de liste ou de matrice n'est pas valide. Par exemple, si la liste {1,2,3,4} est stockée dans L1, L1[5] constitue une erreur de dimension, car L1 ne comporte que quatre éléments.
235	Erreur de dimension. Le nombre d'éléments dans les listes est insuffisant.
240	Dimension inadaptée Deux arguments ou plus doivent être de même dimension. Par exemple, [1,2]+ [1,2,3] constitue une dimension inadaptée, car les matrices n'ont pas le même nombre d'éléments.
250	Division par zéro
260	Erreur de domaine Un argument doit être situé dans un domaine spécifique. Par exemple, rand(0) est incorrect.
270	Nom de variable déjà utilisé
280	Else et ElseIf sont invalides hors du bloc If..EndIf.
290	La déclaration Else correspondant à EndTry manque.
295	Nombre excessif d'itérations

Code d'erreur	Description
300	Une liste ou une matrice de dimension 2 ou 3 est requise.
310	Le premier argument de nSolve doit être une équation d'une seule variable. Il ne doit pas contenir d'inconnue autre que la variable considérée.
320	Le premier argument de solve ou cSolve doit être une équation ou une inéquation. Par exemple, solve(3x^2-4,x) n'est pas correct car le premier argument n'est pas une équation.
345	Unités incompatibles
350	Indice non valide
360	La chaîne d'indirection n'est pas un nom de variable valide.
380	Ans invalide Le calcul précédent n'a pas créé Ans, ou aucun calcul précédent n'a pas été entré.
390	Affectation invalide
400	Valeur d'affectation invalide
410	Commande invalide
430	Invalide pour les réglages du mode en cours
435	Valeur Init invalide
440	Multiplication implicite invalide Par exemple, x(x+1) est incorrect ; en revanche, x*(x+1) est correct. Cette syntaxe permet d'éviter toute confusion entre les multiplications implicites et les appels de fonction.
450	Invalide dans une fonction ou expression courante Seules certaines commandes sont valides à l'intérieure d'une fonction définie par l'utilisateur.
490	Invalide dans un bloc Try..EndTry
510	Liste ou matrice invalide
550	Invalide hors fonction ou programme Un certain nombre de commandes ne sont pas valides hors d'une fonction ou d'un programme. Par exemple, la commande Local ne peut pas être utilisée, excepté dans une fonction ou un programme.
560	Invalide hors des blocs Loop..EndLoop, For..EndFor ou While..EndWhile Par exemple, la commande Exit n'est valide qu'à l'intérieur de ces blocs de boucle.

Code d'erreur	Description
565	Invalide hors programme
570	Nom de chemin invalide Par exemple, \var est incorrect.
575	Complexe invalide en polaire
580	Référence de programme invalide Les programmes ne peuvent pas être référencés à l'intérieur de fonctions ou d'expressions, comme par exemple 1+p(x), où p est un programme.
600	Table invalide
605	Utilisation invalide d'unités
610	Nom de variable invalide dans une déclaration locale
620	Nom de variable ou de fonction invalide
630	Référence invalide à une variable
640	Syntaxe vectorielle invalide
650	Transmission La transmission entre deux unités n'a pas pu aboutir. Vérifiez que les deux extrémités du câble sont correctement branchées.
665	Matrice non diagonalisable
670	Mémoire insuffisante 1. Supprimez des données de ce classeur. 2. Enregistrez, puis fermez ce classeur. Si les suggestions 1 & 2 échouent, retirez les piles, puis remettez-les en place.
680	(manquante
690	) manquante
700	“ manquant
710	] manquant
720	} manquante
730	Manque d'une instruction de début ou de fin de bloc
740	Then manquant dans le bloc If..EndIf
750	Ce nom n'est pas un nom de fonction ou de programme.
765	Aucune fonction n'est sélectionnée.

Code d'erreur	Description
672	Dépassement des ressources
673	Dépassement des ressources
780	Aucune solution n'a été trouvée.
800	Résultat non réel Par exemple, si le logiciel est réglé sur Réel, $\sqrt{}(-1)$ n'est pas valide. Pour autoriser les résultats complexes, réglez le mode “Réel ou Complexe” sur “RECTANGULAIRE ou POLAIRE”.
830	Capacité
850	Programme introuvable Une référence de programme à l'intérieur d'un autre programme est introuvable au chemin spécifié au cours de l'exécution.
855	Les fonctions aléatoires ne sont pas autorisées en mode graphique.
860	Le nombre d'appels est trop élevé.
870	Nom ou variable système réservé
900	Erreur d'argument Le modèle Med-Med n'a pas pu être appliqué à l'ensemble de données.
910	Erreur de syntaxe
920	Texte introuvable
930	Il n'y a pas assez d'arguments. Un ou plusieurs arguments de la fonction ou de la commande n'ont pas été spécifiés.
940	Il y a trop d'arguments. L'expression ou l'équation comporte un trop grand nombre d'arguments et ne peut pas être évaluée.
950	Il y a trop d'indices.
955	Il y a trop de variables indéfinies.
960	La variable n'est pas définie. Aucune valeur n'a été associée à la variable. Utilisez l'une des commandes suivantes : • sto → • :=

Code d'erreur	Description
	• **Define** pour assigner des valeurs aux variables.
965	O.S sans licence
970	La variable est en cours d'utilisation. Aucune référence ni modification n'est autorisée.
980	Variable protégée
990	Nom de variable invalide Assurez-vous que le nom n'excède pas la limite de longueur.
1000	Domaine de variables de fenêtre
1010	Zoom
1020	Erreur interne
1030	Accès illicite à la mémoire
1040	Fonction non prise en charge. Cette fonction requiert CAS (Computer Algebra System). Essayez d'utiliser TI-Nspire™ CAS.
1045	Opérateur non pris en charge. Cet opérateur requiert CAS (Computer Algebra System). Essayez d'utiliser TI-Nspire™ CAS.
1050	Fonction non prise en charge. Cet opérateur requiert CAS (Computer Algebra System). Essayez d'utiliser TI-Nspire™ CAS.
1060	L'argument entré doit être numérique. Seules les entrées comportant des valeurs numériques sont autorisées.
1070	L'argument de la fonction trig est trop grand pour une réduction fiable.
1080	Utilisation de Ans non prise en charge. Cette application n'assure pas la prise en charge de Ans.
1090	La fonction n'est pas définie. Utilisez l'une des commandes suivantes : • **Define** • **:=** • sto → pour définir une fonction.
1100	Calcul non réel Par exemple, si le logiciel est réglé sur Réel, $\sqrt{}(-1)$ n'est pas valide. Pour autoriser les résultats complexes, réglez le mode “Réel ou Complexe” sur “RECTANGULAIRE ou POLAIRE”.
1110	Limites invalides

Code d'erreur	Description
1120	Pas de changement de signe
1130	L'argument ne peut être ni une liste ni une matrice.
1140	Erreur d'argument Le premier argument doit être une expression polynomiale du second argument. Si le second argument est omis, le logiciel tente de sélectionner une valeur par défaut.
1150	Erreur d'argument Les deux premiers arguments doivent être des expressions polynomiales du troisième argument. Si le troisième argument est omis, le logiciel tente de sélectionner une valeur par défaut.
1160	Nom de chemin de bibliothèque invalide Les noms de chemins doivent utiliser le format *xxx**yyy*, où : • La partie *xxx* du nom peut contenir de 1 à 16 caractères, et • la partie *yyy*, si elle est utilisée, de 1 à 15 caractères. Pour plus d'informations à ce sujet, reportez-vous à la section Bibliothèques dans la documentation.
1170	Utilisation invalide de nom de chemin de bibliothèque • Une valeur ne peut pas être assignée à un nom de chemin en utilisant la commande **Define**, **:=** ou sto →. • Un nom de chemin ne peut pas être déclaré comme variable Local ni être utilisé dans une définition de fonction ou de programme.
1180	Nom de variable de bibliothèque invalide. Assurez-vous que ce nom : • ne contienne pas de point, • ne commence pas par un tiret de soulignement, • ne contienne pas plus de 15 caractères. Pour plus d'informations à ce sujet, reportez-vous à la section Bibliothèques dans la documentation.
1190	Classeur de bibliothèque introuvable : • Vérifiez que la bibliothèque se trouve dans le dossier Ma bibliothèque. • Rafraîchissez les bibliothèques. Pour plus d'informations à ce sujet, reportez-vous à la section Bibliothèques dans la documentation.
1200	Variable de bibliothèque introuvable :

Code d'erreur	Description
	• Vérifiez que la variable de bibliothèque existe dans la première activité de la bibliothèque. • Assurez-vous d'avoir défini la variable de bibliothèque comme objet LibPub ou LibPriv. • Rafraîchissez les bibliothèques. Pour plus d'informations à ce sujet, reportez-vous à la section Bibliothèques dans la documentation.
1210	Nom de raccourci de bibliothèque invalide Assurez-vous que ce nom : • ne contienne pas de point, • ne commence pas par un tiret de soulignement, • ne contienne pas plus de 16 caractères, • ne soit pas un nom réservé. Pour plus d'informations à ce sujet, reportez-vous à la section Bibliothèques dans la documentation.
1220	Erreur d'argument : Les fonctions tangentLine et normalLine prennent uniquement en charge les fonctions à valeurs réelles.
1230	Erreur de domaine. Les opérateurs de conversion trigonométrique ne sont pas autorisés en mode Angle Degré ou Grade.
1250	Erreur d'argument Utilisez un système d'équations linéaires. Exemple de système à deux équations linéaires avec des variables x et y : 3x+7y=5 2y-5x=-1
1260	Erreur d'argument : Le premier argument de nfMin ou nfMax doit être une expression dans une seule variable. Il ne doit pas contenir d'inconnue autre que la variable considérée.
1270	Erreur d'argument La dérivée doit être une dérivée première ou seconde.
1280	Erreur d'argument Utilisez un polynôme dans sa forme développée dans une seule variable.

Code d'erreur	Description
1290	Erreur d'argument Utilisez un polynôme dans une seule variable.
1300	Erreur d'argument Les coefficients du polynôme doivent s'évaluer à des valeurs numériques.
1310	Erreur d'argument : Une fonction n'a pas pu être évaluée en un ou plusieurs de ses arguments.
1380	Erreur d'argument : Les appels imbriqués de la fonction domain() ne sont pas permis.

Codes et messages d’avertissement

Vous pouvez utiliser la fonction **warnCodes()** pour stocker les codes d’avertissement générés lors du calcul d’une expression. Le tableau ci-dessous présente chaque code d'avertissement et le message associé. Pour un exemple de stockage des codes d’avertissement, consultez **warnCodes()**. page 176.

Code d'avertissement	Message
10 000	L’opération peut donner des solutions fausses. Le cas échéant, essayez d’utiliser des méthodes graphiques pour vérifier les résultats.
10 001	L'équation générée par dérivation peut être fausse.
10 002	Solution incertaine Le cas échéant, essayez d’utiliser des méthodes graphiques pour vérifier les résultats.
10 003	Précision incertaine Le cas échéant, essayez d’utiliser des méthodes graphiques pour vérifier les résultats.
10 004	L’opération peut omettre des solutions. Le cas échéant, essayez d’utiliser des méthodes graphiques pour vérifier les résultats.
10 005	CSolve peut donner plus de zéros.
10 006	Solve peut donner plus de zéros. Le cas échéant, essayez d’utiliser des méthodes graphiques pour vérifier les résultats.
10 007	D'autres solutions sont possibles. Essayez de spécifier des bornes inférieure et supérieure ou une condition initiale. Exemples utilisant la fonction solve() : • solve(Equation, Var=Guess)|lowBound<Var<upBound • solve(Equation, Var)|lowBound<Var<upBound • solve(Equation, Var=Guess) Le cas échéant, essayez d’utiliser des méthodes graphiques pour vérifier les résultats.
10 008	Le domaine du résultat peut être plus petit que le domaine de l’entrée.
10 009	Le domaine du résultat peut être plus grand que le domaine de l’entrée.
10 012	Calcul non réel

Code d'avertissement	Message
10 013	∞^0 ou undef^0 remplacé par 1
10 014	undef^0 remplacés par 1
10 015	1^∞ ou 1^undef remplacé par 1
10 016	1^undef remplacés par 1
10 017	Capacité remplacée par ∞ ou −∞
10 018	Requiert et retourne une valeur 64 bits.
10 019	Ressources insuffisantes, la simplification peut être incomplète.
10 020	L'argument de la fonction trigonométrique est trop grand pour une réduction fiable.
10 021	Les données saisies comportent un paramètre non défini. Le résultat peut ne pas être valide pour toutes les valeurs possibles du paramètre.
10 022	La spécification des bornes inférieure et supérieure peut donner une solution.
10 023	Le scalaire a été multiplié par la matrice unité.
10 024	Résultat obtenu par calcul approché.
10 025	L'équivalence ne peut pas être vérifiée en mode EXACT.
10 026	Il est possible que la contrainte soit ignorée. Spécifiez la contrainte sous la forme "\" 'Variable MathTestSymbol Constant' ou une combinaison de ces formats, par exemple 'x<3 and x>-12'

Informations générales

Aide en ligne

education.ti.com/eguide

Sélectionnez votre pays pour obtenir d'autres informations relatives aux produits.

Contacter l'assistance technique TI

education.ti.com/ti-cares

Sélectionnez votre pays pour obtenir une assistance technique ou d'autres types de support.

Informations Garantie et Assistance

education.ti.com/warranty

Sélectionnez votre pays pour en savoir plus sur la durée et les termes de la garantie et sur l'assistance pour le produit.

Garantie limitée. Cette garantie n'affecte pas vos droits statutaires.

Texas Instruments Incorporated

12500 TI Blvd.

Dallas, TX 75243

Index

⇒

→

⇔

©

°

0

1

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Δ

X