Introduction à *Mathematica*

UPMC, Paris.
Julien Toulouse (julien.toulouse@upmc.fr)

Introduction

Qu'est-ce-que *Mathematica* ?

Mathematica est un logiciel de calcul formel et numérique développé par Wolfram Research. Il permet essentiellement de faire du **calcul formel** (manipulation d'expressions mathématiques sous forme symbolique, par exemple : calcul de dérivées, de primitives, simplication d'expressions, etc...) et du **calcul numérique** (évaluation d'expressions mathématiques sous forme numérique; par exemple : calcul des premières décimales du nombre π, évaluation approchée d'intégrales, etc...). *Mathematica* incorpore un langage de programmation sophistiqué et permet aussi de faire des graphiques. C'est un logiciel très utilisé en enseignement, dans la recherche scientifique et dans l'industrie.

Ou trouver *Mathematica* ?

L'UPMC a une licence globale permettant d'installer *Mathematica* sur tous les ordinateurs d'enseignement. En particulier, il est disponible sur les ordinateurs du L'UTES (Bâtiment Atrium), y compris ceux en libre service. La licence globale permet également aux enseignants et aux étudiants d'installer *Mathematica* sur n'importe quel ordinateur personnel (à la maison). Pour cela, connectez vous sur le site **https://mon.upmc.fr**, allez dans la rubrique "Mes Outils", puis suivez les instructions pour télécharger et installer *Mathematica*. L'activation se fait par un mot de passe et nécessite l'appartenance à l'UPMC.

Quelques autres outils et logiciels

Wolfram Alpha (www.wolframalpha.com) est un outil en ligne gratuit répondant à toutes sortes de questions (qui doivent être posées en anglais) et qui utilise une grande base de donnée et des calculs *Mathematica*. Il permet d'accéder avec un simple navigateur internet à beaucoup de fontionnalités de *Mathematica*. Par exemple, la demande "derivative of sin(x)" (dérivée de sin(x)) calcule la réponse "cos(x)" et donne au passage d'autres informations utiles (propriétés de la fonction, courbes, développements limités, etc...). A essayer !

Un autre logiciel de calcul formel et numérique très utilisé est *Maple*. Il a des capacités similaires à celles de *Mathematica*. Il est également installé sur les ordinateurs du L'UTES.

Maxima est un logiciel libre et gratuit constituant une alternative aux logiciels commerciaux que sont *Mathematica* ou *Maple*, avec néanmoins une interface graphique moins conviviale.

Matlab est un logiciel commercial de calcul numérique très utilisé, mais ne permet pas de faire du calcul formel. *Scilab* est un logiciel libre alternatif à *Matlab*.

Débuter en *Mathematica*

Noyau et interface graphique

Mathematica est composé de deux parties : le noyau ("kernel") et l'interface graphique ("front end"). Le noyau constitue le coeur du logiciel; il interprète les instructions d'entrée (écrites en langage *Mathematica*), puis calcule et retourne le résultat. L'interface graphique s'occupe de l'interaction avec l'utilisateur. Elle gère le fichier de travail (souvent appellé "feuille *Mathematica*" ou "notebook"), permet de taper les instructions et de visualiser les résultats. Le logiciel dispose aussi d'un traitement de texte, permettant ainsi d'inclure du texte parmi les calculs effectués; ce document a d'ailleurs été rédigé avec *Mathematica*. Plusieurs palettes d'outils sont disponibles pour aider à l'édition aussi bien de textes que d'expressions mathématiques (voir menu "Palettes").

Un fichier *Mathematica*

Un fichier Mathematica, aussi appelé "notebook", a une extension ".nb". Il est structuré en cellules ("cells"). Une cellule est constitué d'une ou de plusieurs lignes et est repérée par un crochet à droite du fichier. On peut avoir des cellules contenant des instructions Mathematica (cellule de type "In"), des résultats de calculs (cellule de type "Out"), du texte (cellule du type "Text"), un titre de paragraphe (cellule de type "Section"), etc ...

On peut sélectionner la totalité d'une cellule en cliquant sur son crochet à droite. On peut ensuite effacer (⇐), copier (CTRL+c), couper (CTRL+x), coller (CTRL+v) la cellule, comme dans un logiciel de traitement de texte normal. Par défaut, les nouvelles cellules créées sont des cellules d'instructions *Mathematica*. On peut changer le type d'une cellule, après l'avoir selectionée, par le menu "Format > Style".

Une cellule peut contenir plusieurs sous-cellules, formant ainsi des groupes structurés de cellules. Pour faciliter la lecture du fichier, on peut ouvrir ou fermer des groupes de cellules en double-cliquant sur le crochet correspondant (ou en cliquant sur le triangle à gauche du titre).

Pensez à **sauvergarder souvent** votre fichier *Mathematica* (menu "File", puis "Save" ou "Save As"), les mauvaises manipulations étant fréquentes...

Menu Aide

Mathematica inclut une documentation exhaustive (voir le menu "Help", puis "Documentation Center" ou "Virtual Book"), incluant de nombreux exemples directement exécutables dans les pages d'aide.

La commande "Find Selected Function" (ou touche "F1") est très pratique : dans un fichier *Mathematica*, après avoir placé le curseur sur une instruction *Mathematica*, appuyez sur F1 pour afficher la page d'aide correspondante à cette instruction.

Mon premier calcul en *Mathematica*

On tape une instruction (par exemple "1+2"), puis on l'exécute en maintenant enfoncé la touche "Maj" ou "Shift" (⇑) et en appuyant sur la touche "Entrée" (↵),le résultat ("3") s'affichant sur une nouvelle ligne :

```
In[1]:= 1 + 2

Out[1]= 3
```

Après exécution, la ligne d'instruction est désignée par "In" suivi du numéro de l'instruction depuis le démarrage du noyau. La ligne de résultat est désignée par "Out" suivi du numéro de l'instruction correspondante.

Le noyau ("kernel") de *Mathematica* est démarré lors du premier calcul. Si un calcul prend trop de temps et que l'on décide d'y renoncer en cours d'évaluation, on peut utiliser le menu "Evaluation", puis "Abort Evaluation". Parfois, il peut être également utile de quitter puis rédemarrer le noyau, afin d'effacer de la mémoire tous les résultats des calculs précédents. Pour cela, on utilise le menu "Evaluation" puis "Quit Kernel". Le noyau sera alors redémarré lors du prochain calcul.

Calculs arithmétiques simples

Comme avec une calculatrice, on peut bien sûr faire de simples calculs arithmétiques avec *Mathematica.* Nous avons déjà vu l'exemple d'une addition. Voici un exemple d'un calcul avec une soustraction (-), une multiplication (*) et une division (/) sur des nombres décimaux :

```
In[2]:= (10.2 - 3.1) * 6.2 / 2.9
Out[2]= 15.1793
```

Voici l'exemple d'un calcul avec une puissance :

```
In[3]:= 2^4
Out[3]= 16
```

Par défaut, *Mathematica* simplifie les fractions mais ne donne pas de valeur décimale :

```
In[4]:= 4 / 6
Out[4]= 2/3
```

Pour avoir une valeur décimale approché d'une expression, on peut utiliser la fonction "N" en donnant l'argument entre crochets :

```
In[5]:= N[4 / 6]
Out[5]= 0.666667
```

On peut aussi faire des calculs avec des nombres complexes, le nombre imaginaire i étant écrit "I" :

```
In[6]:= (2 + 3 I) * (4 - 5 I)
Out[6]= 23 + 2 i
```

Définir des variables

Il est souvent pratique de définir des variables mathématiques contenant une valeur numérique. Par exemple, pour définir une variable nommée "x" et lui donner la valeur "5", on tape

In[7]:=
```
x = 5
```

Out[7]=
```
5
```

Le signe "=" réalise ce que l'on appelle en informatique une "affectation" (et on dit qu'on affecte à "x" la valeur 5). Dans tous les calculs suivants, *Mathematica* replacera la variable "x" par sa valeur :

In[8]:=
```
x^2
```

Out[8]=
```
25
```

La ligne suivante change la valeur de "x" :

In[9]:=
```
x = 6 + 8
```

Out[9]=
```
14
```

Le nom d'une variable peut être composé de plusieurs lettres et chiffres, par exemple

In[10]:=
```
abc5 = 78
```

Out[10]=
```
78
```

mais le nom d'une variable ne peut pas commencer par un chiffre.

On peut taper des équations avec les variables que l'on a définies de la même manière qu'en mathématiques, par exemple :

In[11]:=
```
(2 x + abc5) / 4
```

Out[11]=
$$\frac{53}{2}$$

le signe de la multiplication (*) entre "2" et "x" n'est pas nécessaire.

Pour effacer la valeur numérique d'une variable, on utilise la fonction "Clear" :

In[12]:=
```
Clear[x]
```

On peut vérifier que la variable n'a alors plus de valeur numérique :

In[13]:=
```
x
```

Out[13]=
```
x
```

Quelques constantes mathématiques

Mathematica dispose de quelques variables déjà définies (constantes) et prêtes à être utilisées, par exemple :

- le nombre π

In[14]:=
```
Pi
```

Out[14]=
```
π
```

- le nombre e

In[15]:=
```
E
```

Out[15]=
```
ⅇ
```

- le nombre imaginaire i

In[16]:=
```
I
```

Out[16]=
```
ⅈ
```

- l'infini ∞ :

In[17]:=
```
Infinity
```

Out[17]=
```
∞
```

Rappellons que pour forcer l'affichage d'une valeur décimale approchée, on peut utiliser la fonction "N". Voici les 100 premiers chiffres de π :

In[18]:=
```
N[Pi, 100]
```

Out[18]=
```
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862
 8034825342117068
```

Définir des fonctions

On peut définir ses propres fonctions en *Mathematica*. Par exemple, pour définir la fonction f(x) = x^2 , on tape :

In[19]:=
```
f[x_] := x^2
```

Dans la définition d'une fonction, on utilise habituellement le signe ": =" qui signifie une "affectation retardée", c'est-à -dire que le membre de droite n'est pas évalué et affecté à f(x) lors de la définition de la fonction ci-dessus mais il sera évalué plus tard à chaque utilisation de la fonction après substitution de "x" par une valeur. Parfois, on peut vouloir forcer l'évaluation du membre de droite lors de la définition de la fonction et on utilise alors le signe de l'affection "="; mais il faut alors s'assurer que la variable "x" ne contient pas de valeur.

Dans le membre de gauche, l'argument "x" de la fonction est donné entre crochets et doit obligatoirement être suivi d'un tiret bas "_" (ou "underscore"), ce qui informe *Mathematica* que "x" est une variable muette qui devra être remplacée par l'argument avec lequel la fonction sera appelée.

On peut alors appeler la fonction f avec comme argument "3" par exemple

In[20]:=
```
f[3]
```

Out[20]=
```
9
```

ou n'importe quelle expression, par exemple "2+4 z"

In[21]:=
```
f[2 + 4 z]
```

Out[21]=
```
(2 + 4 z)^2
```

La fonction f peut être utilisée dans une expression quelconque

In[22]:=
```
3 + 2 y + 5 f[y]
```

Out[22]=
```
3 + 2 y + 5 y^2
```

On peut définir des fonctions à plusieurs variables

In[23]:=
```
g[x_, y_] := x + y / x^2
```

Comme pour les variables, on peut effacer les fonctions qui viennent d'être définies par la fonction "Clear" :

In[24]:=
```
Clear[f]
```

In[25]:=
```
Clear[g]
```

Quelques fonctions mathématiques

Mathematica dispose de nombreuses fonctions déjà définies.Voici quelques fonctions mathématiques courantes :

- racine carrée : $\sqrt{x}$

In[26]:=
```
Sqrt[x]
```

Out[26]= $\sqrt{x}$

- exponentielle : e^x

In[27]:=
```
Exp[x]
```

Out[27]= e^x

- logarithme népérien : ln(x)

In[28]:=
```
Log[x]
```

Out[28]=
```
Log[x]
```

- logarithme en base b : $\log_b(x)$ (le plus souvent, $b = 10$)

In[29]:=
```
Log[b, x]
```

Out[29]= $\frac{\text{Log}[x]}{\text{Log}[b]}$

- fonctions trigonométriques :

In[30]:=
```
Sin[x]
```

Out[30]=
```
Sin[x]
```

In[31]:=
```
Cos[x]
```

Out[31]=
```
Cos[x]
```

In[32]:=
```
Tan[x]
```

Out[32]=
```
Tan[x]
```

In[33]:=
```
ArcSin[x]
```

Out[33]=
```
ArcSin[x]
```

In[34]:=
```
ArcCos[x]
```

Out[34]=
```
ArcCos[x]
```

In[35]:=
```
ArcTan[x]
```

Out[35]=
```
ArcTan[x]
```

- valeur absolue : |x|

In[36]:=
```
Abs[x]
```

Out[36]=
```
Abs[x]
```

- factorielle : n !

In[37]:=
```
n!
```

Out[37]=
```
n!
```

ou

In[38]:=
```
Factorial[n]
```

Out[38]=
```
n!
```

A retenir : les fonctions *Mathematica* déjà définies commencent toujours par une majuscule. Placez le curseur sur une fonction et appuyez sur F1 pour afficher la page d'aide de cette fonction. Si on se demande si *Mathematica* dispose d'une fonction particulière, ou si on a oublié le nom d'une fonction, le plus simple est de faire une recherche dans l'aide (menu "Help > Documentation Center").

Exemples de calculs formels

Mathematica dispose de fonctions très puissantes permettant d'effectuer toutes sortes de calculs formels. Quelques exemples :

- simplification de $\sin(x)^2 + \cos(x)^2$

In[39]:=
```
Simplify[Sin[x]^2 + Cos[x]^2]
```

Out[39]=
```
1
```

- développement de l'expression $(2\,x + 3)\left(x^2 - 6\right)$

In[40]:=
```
Expand[(2 x + 3) (x^2 - 6)]
```

Out[40]=
$$-18 - 12\,x + 3\,x^2 + 2\,x^3$$

- factorisation de l'expression $x^2 - 2\,x - 3$

In[41]:=
```
Factor[x^2 - 2 x - 3]
```

Out[41]=
$$(-3 + x)\,(1 + x)$$

- calcul de la dérivée de x^n par rapport à x

In[42]:=
```
D[x^n, x]
```

Out[42]=
$$n\,x^{-1+n}$$

- calcul d'un primitive de cos(x)

In[43]:=
```
Integrate[Cos[x], x]
```

Out[43]=
$$\text{Sin}[x]$$

- calcul de l'intégrale $\int_0^\infty e^{-x^2}\,dx$

In[44]:=
```
Integrate[Exp[-x^2], {x, 0, Infinity}]
```

Out[44]=
$$\frac{\sqrt{\pi}}{2}$$

- calcul de la limite de $(x - 1)\ln(x - 1)$ quand x→ 1

In[45]:=
```
Limit[(x - 1) Log[x - 1], x -> 1]
```

Out[45]=
$$0$$

- calcul du développement limité de cos(x) autour de $x = 0$ à l'ordre 10

In[46]:=
```
Series[Cos[x], {x, 0, 10}]
```

Out[46]=
$$1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + \frac{x^8}{40\,320} - \frac{x^{10}}{3\,628\,800} + O[x]^{11}$$

- Résolution de l'équation $x^2 - 3x + 2 = 0$ en l'inconnue x

In[47]:=
```
Solve[x^2 - 3 x + 2 == 0, x]
```

Out[47]=
```
{{x → 1}, {x → 2}}
```

On obtient deux solutions réelles. Notez que pour définir une équation, on utilise le signe "= =" pour ne pas confondre avec le signe d'affectation "=".

Ces fonctions disposent en général d'un grand nombre d'options permettant de faire face à des cas particuliers ou difficiles. Consultez les pages d'aide de ces fonctions.

Exemples de calculs numériques

Parfois, un calcul formel n'est pas faisable, et il faut alors faire un calcul numérique approché. Quelques exemples :

- calcul d'une valeur approchée de l'intégrale $\int_1^3 \ln(\arctan(x))\,dx$

In[48]:=
```
NIntegrate[Log[ArcTan[x]], {x, 1, 3}]
```

Out[48]=
```
0.135924
```

- calcul approché des solutions de l'équation $x^5 + 3x^4 + 2x^3 - x^2 + x + 1 = 0$

In[49]:=
```
NSolve[x^5 + 3 x^4 + 2 x^3 - x^2 + x + 1 == 0, x]
```

Out[49]=
```
{{x → -1.69305 - 0.668971 i}, {x → -1.69305 + 0.668971 i},
 {x → -0.566333}, {x → 0.476216 - 0.55321 i}, {x → 0.476216 + 0.55321 i}}
```

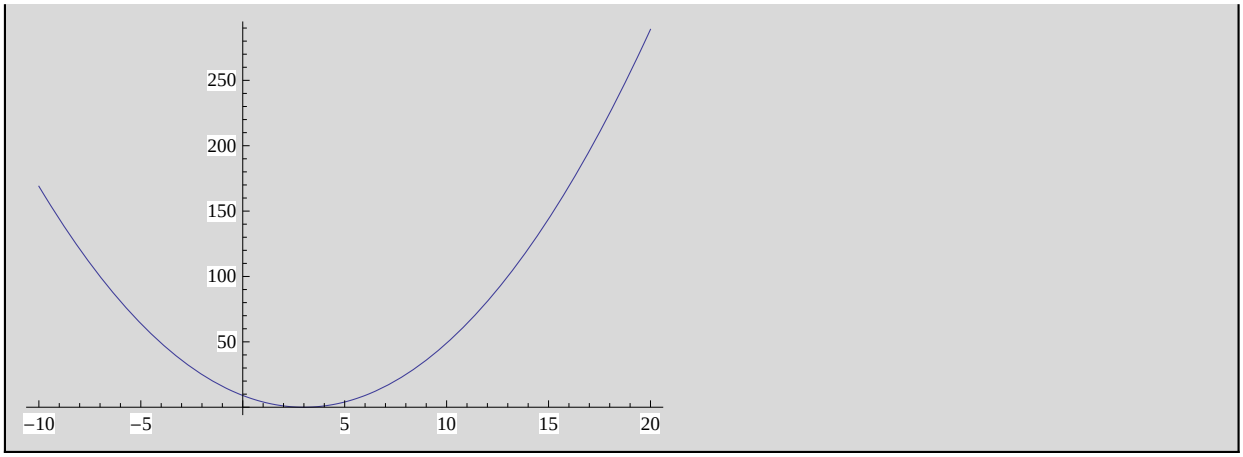
Faire des graphiques

Mathematica permet de faire toutes sortes de graphiques. Exemples :

- Tracé de la courbe de $(x - 3)^2$ pour x variant de -10 à 20 :

In[50]:=
```
Plot[(x - 3)^2, {x, -10, 20}]
```

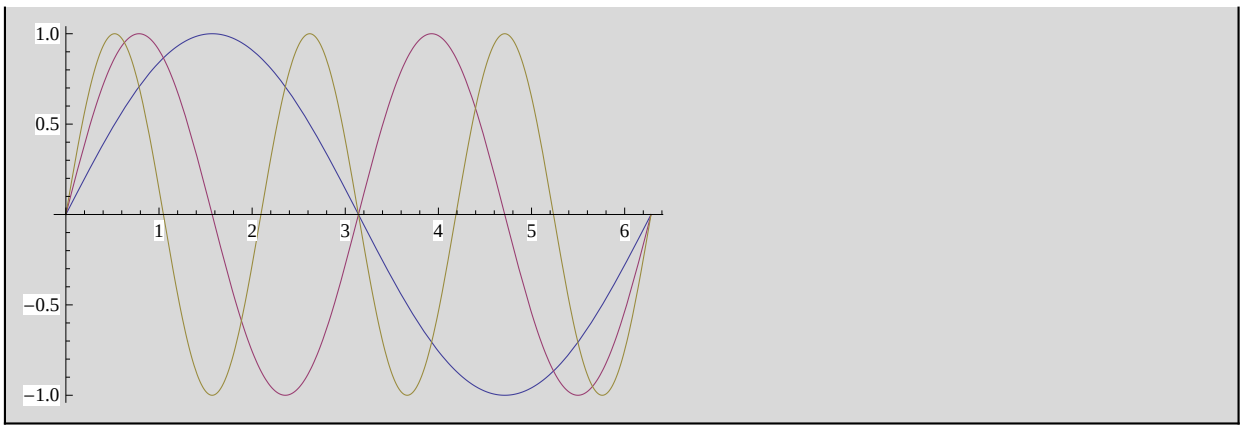
Out[50]=



- Tracé superposé des courbes de sin(x), sin(2 x), et sin(3 x) pour x variant de 0 à 2π :

In[51]:=
```
Plot[{Sin[x], Sin[2 x], Sin[3 x]}, {x, 0, 2 Pi}]
```

Out[51]=

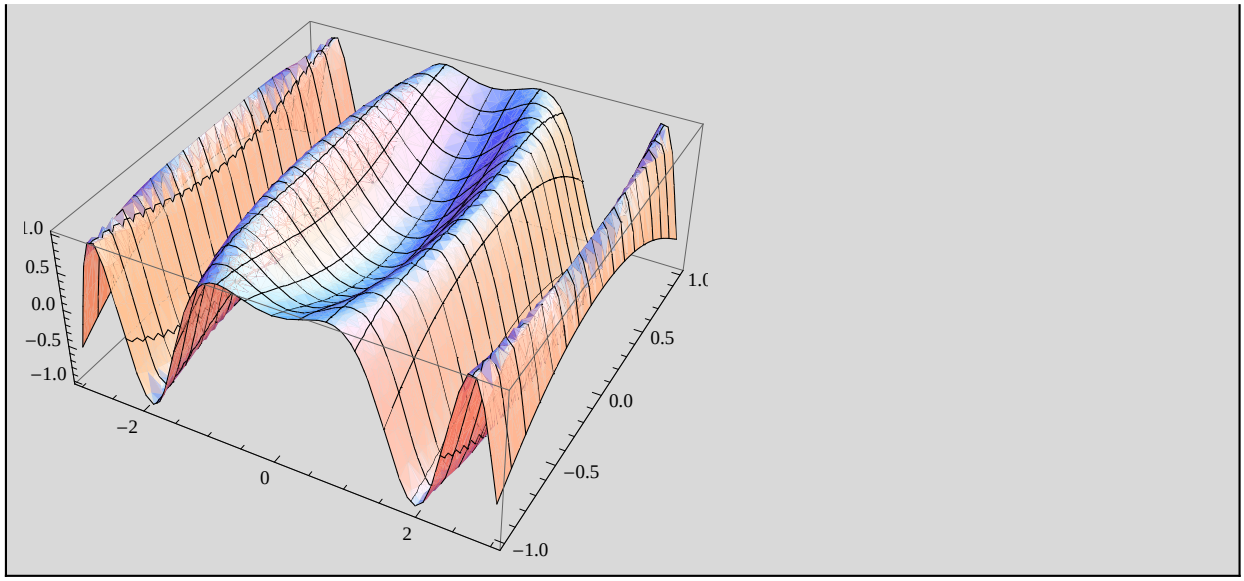


Consultez la page d'aide de la fonction "Plot" pour voir toutes les options disponibles (on peut changer l'échelle, les couleurs, afficher le nom des axes, etc...).

- Tracé en 3D d'une fonction à deux variables :

In[52]:=
```
Plot3D[Sin[x^2 + y^2], {x, -3, 3}, {y, -1, 1}]
```

Out[52]=



Exercices

Exercice 1 : étude d'une fonction

1) Définir la fonction $f(x) = x + 2 + (3x + 5)/(x + 2)^2$.

2) Calculer la dérivée de $f(x)$ par rapport à x. Calculer les zéros de la dérivée. Exprimer la dérivée sous forme factorisée afin de vérifier ses zéros.

3) Calculer les valeurs numériques de *f(x)* aux zéros de sa dérivée.

4) Calculer la dérivée seconde de $f(x)$ par rapport x. Calculer les valeurs numériques de la dérivée seconde aux zéros de la dérivée première calculés précédement. Conclure sur la convexité/concavité de $f(x)$.

5) Calculer les limites de $f(x)$ quand $x \to -\infty$, $x \to 2$ et $x \to +\infty$.

6) Calculer le développement limité de $f(x)$ en $x = 0$ à l'ordre 4. Calculer le développement asymptotique de $f(x)$ à l'ordre 2 quand $x \to +\infty$ et $x \to -\infty$. En déduire que la courbe de $f(x)$ admet une asymptote oblique quand $x \to +\infty$ et $x \to -\infty$.

7) Tracer la courbe de $f(x)$ ainsi que son asymptote oblique. Ajuster éventuellement l'échelle en ordonnée avec l'option "PlotRange" de la fonction "Plot" (consultez l'aide).

Exercice 2 : intégration

1) Calculer une primitive de la fonction $f(x) = \sin(1/x)$. On observe que *Mathematica* donne une primitive exprimée avec une fonction spéciale pour laquelle nous ne disposons pas d'expression explicite. Consultez l'aide pour connaitre la définition de cette fonction spéciale.

2) Calculer la valeur exacte de l'intégrale de $f(x)$ pour x variant de -1 à 2, puis trouver une valeur numérique avec dix chiffres significatifs en utilisant la fonction "N".

3) Recalculer une valeur approchée de l'intégrale précédente par intégration numérique (fonction "NIntegrate"), puis comparer à la valeur trouvée précédemment par intégration exacte.

4) Tracer la courbe de la fonction $f(x)$ pour comprendre la source du problème.

5) En consultant l'aide, trouver l'option adéquate de la fonction "NIntegrate" permettant d'améliorer la précision de l'intégration numérique précédente.

6) Faire (à la main) le changement de variable $t = 1/x$ (on pensera à couper l'intervalle d'intégration en deux en $x = 0$), puis réessayer une intégration numérique. Commenter. Pour comprendre, tracer la courbe de la nouvelle fonction à intégrer.

Exercice supplémentaire : particule quantique dans une boite de potentiel

On considère une particule quantique de masse m dans une boite de potentiel à une dimension et de longueur L. Le potentiel $V(x)$ est nul dans la boite $0 \le x \le L$, et infini en dehors. La fonction d'onde de cette particule dans un état n est :

$$\phi_n(x) = A \sin\left(\frac{n \pi x}{L}\right) \tag{1}$$

où A est la constante de normalisation et n est en entier supérieur ou égal à 1.

1) Définir la fonction d'onde comme fonction de x et de n, et calculer l'intégrale $\int_0^L \phi_n(x)^2 \, dx$. On simplifiera l'expression obtenue en indiquant que n est en entier avec l'option "Assumptions → n ∈ Integers" dans la fonction "Simplify" En déduire la valeur de la constante A.

2) Vérifier que $\phi_n(x)$ est solution de l'équation de Schrödinger dans la boite, et calculer l'énergie E_n de l'état n.

3) On utilise les unités atomiques (u.a.) pour lesquelles $\hbar = m = 1$. Pour $L = 2$ u.a., tracer sur un même graphique les fonctions d'onde $\phi_n(x)$ en fonction de x pour les états $n = 1, 2, 3, 4$ et 5. On pourra aussi utiliser la fonction "Manipulate" pour faire varier le paramètre n de manière itérative (consulter l'aide). Tracer également la densité de probabilité de présence $\phi_n(x)^2$ pour plus

4) Tracer également la densité de probabilité de présence $\phi_n(x)^2$ pour plusieurs valeurs de n. Comment se répartit la densité de probabilité de présence dans la limite $n \to \infty$? A quoi correspond physiquement cette limite.

5) Tracer les 10 premiers niveaux d'énergie E_n sur un même graphique. Pour cela, on construira une "liste" contenant les énergies E_n à l'aide de la fonction "Table".

6) Calculer la valeur moyenne de la position de la particule x dans un état n, pour une valeur quelconque de L.

Utilisation plus avancée de *Mathematica*

Palettes et raccourcis clavier

Les diverses palettes (voir menu "Palettes", puis par exemple "Classroom Assistant") permettent de trouver facilement les commandes *Mathematica* les plus courantes et de taper des formules mathématiques avec le format d'écriture naturel. Par exemple, elles permettent de taper l'expression suivante

In[53]:=
```
(x^2 + 3) Exp[-y] / (z + Pi)
```

Out[53]=
$$\frac{e^{-y}\left(3+x^2\right)}{\pi+z}$$

sous une forme plus lisible

In[54]:=
```
(x^2 + 3) e^-y / (z + π)
```

Out[54]=
$$\frac{e^{-y}\left(3+x^2\right)}{\pi+z}$$

Pour taper plus vite, il est également utile de connaitre quelques raccourcis clavier :

- CTRL+6 : pour faire les puissances (exemple : x^2)
- CTRL+/ : pour faire les fractions (exemple : $\frac{1}{2}$)
- ESC ee ESC : pour écrire l'exponentielle e (équivalent à Exp)
- ESC p ESC : pour écrire le nombre π (équivalent à Pi)
- ESC ii ESC : pour écrire le nombre imaginaire i (équivalent à I)
- ESC inf ESC : pour écrire ∞ (équivalent à Infinity)
- ESC -> ESC : pour écrire $\rightarrow$ (équivalent à ->)

etc...

Listes

Dans *Mathematica*, on utilise souvent les *listes* qui sont des ensembles d'éléments regroupés entre accolades et séparés par des virgules. Par exemple,

In[55]:=
```
{a, b, c}
```

Out[55]=
```
{a, b, c}
```

est une liste. Les éléments a, b,... d'une liste peuvent être n'importe quoi (nombre, variables, fonctions, ...) y compris une liste eux-mêmes ! On peut donc imaginer des listes imbriquées complètement générales du genre :

In[56]:=
```
{{a, 3}, b, {coucou, h[x], -56}}
```

Out[56]=
```
{{a, 3}, b, {coucou, h[x], -56}}
```

Nous verrons plus loin qu'une liste simple du type {a,b,c} représente un *vecteur* de dimension 3 et qu'une liste de listes du type {{a,b},{c,d},{e,f}} représente une *matrice* de dimension 3× 2.

On peut bien sûr affecter une liste à une variable :

In[57]:=
```
maliste = {a, b, -5}
```

Out[57]=
```
{a, b, -5}
```

On peut accéder individuellement au ième élément d'une liste avec la syntaxe suivante [[i]], par exemple :

In[58]:=
```
maliste[[2]]
```

Out[58]=
```
b
```

In[59]:=
```
maliste[[3]]
```

Out[59]=
```
-5
```

Un détail : on peut utiliser les raccourcis clavier ESC[[ESC et ESC]]ESC pour rendre les doubles crochets plus jolis et plus lisibles :

In[60]:=
```
maliste⟦2⟧
```

Out[60]=
```
b
```

Il existe dans *Mathematica* beaucoup de fonctions de manipulation des listes (pour ajouter des éléments, en extraire, etc...). Consultez l'aide si besoin.

Substitution

L'opération "/." (correspondant à une fonction appelée "ReplaceAll" en *Mathematica*) permet de substituer dans une expression une variable par une valeur ou par une autre variable. Par exemple, pour remplacer z par 2 dans l'expression $(z+1)^2$, on peut écrire

In[61]:=
```
(z + 1)^2 /. z -> 2
```

Out[61]=
```
9
```

"z -> 2" s'appelle une *règle* ("rule" en anglais) de substitution. L'avantage est que "2" n'est pas affecté à la variable z globalement dans tout le notebook, mais la substitution ne se fait que localement dans l'expression ci-dessus, et on peut continuer à utiliser z par la suite comme une variable non définie :

In[62]:=
```
z
```

Out[62]=
```
z
```

On peut faire plusieurs substitutions avec la syntaxe suivante

In[63]:=
```
(z + 1 + abc)^2 /. {z -> 2, abc -> m}
```

Out[63]=
```
(3 + m)^2
```

Souvent les fonctions *Mathematica* retournent les résultats sous forme d'une liste de règles, par exemple

In[64]:=
```
resultat = Solve[x^2 - x - 2 == 0, x]
```

Out[64]=
```
{{x → -1}, {x → 2}}
```

où nous avons stocké le résultat dans la variable "resultat". La première solution $x = -1$ correspond au premier élément de la liste "resultat" :

In[65]:=
```
resultat[[1]]
```

Out[65]=
```
{x → -1}
```

Puisque "resultat[[1]]" est sous la forme d'une règle, il suffit d'appliquer une substitution sur la variable "x" pour extraire la valeur "-1" :

In[66]:=
```
x /. resultat[[1]]
```

Out[66]=
```
-1
```

Pipeline

L'opération "//" (appelée "pipeline", ou tube en français) permet d'appliquer une fonction au dernier résultat avec un syntaxe inversée. Ainsi, l'instruction suivante

In[67]:=
```
x // f
```

Out[67]=
```
f[x]
```

est équivalente à

In[68]:=
```
f[x]
```

Out[68]=
```
f[x]
```

L'exemple ci-dessus ne présente pas vraiment d'intérêt et rend l'écriture beaucoup moins claire. En revanche, le pipeline est très pratique pour appliquer une fonction *Mathematica* à une expression déjà tapée sans avoir à revenir au début de la ligne, par exemple :

In[69]:=
```
(Cos[x] + Sin[x])^2 - 2 Cos[x] Sin[x] // Expand
```

Out[69]=
```
Cos[x]^2 + Sin[x]^2
```

On peut empiler plusieurs fonctions avec les pipelines, par exemple appliquons la fonction "Simplify" au résultat précédent

In[70]:=
```
(Cos[x] + Sin[x])^2 - 2 Cos[x] Sin[x] // Expand // Simplify
```

Out[70]=
```
1
```

Autre exemple, appliquons la fonction "N" au nombre π avec un pipeline

In[71]:=
```
π // N
```

Out[71]=
```
3.14159
```

Il est également possible d'utiliser le pipeline pour appliquer une fonction avec plus d'un seul argument, on utilise alors la syntaxe suivante

In[72]:=
```
π // N[#, 100] &
```

Out[72]=
```
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803
 4825342117068
```

qui est équivalent à

In[73]:=
```
N[π, 100]
```

Out[73]=
```
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803
 4825342117068
```

Le symbole "#" repère l'endroit où l'expression qui précéde (ici π) sera insérée et "&" marque la fin de la fonction. La simplicité de la syntaxe est néanmoins un peu perdue.

Minimisation/maximisation

On cherche souvent le minimum ou le maximum *global* d'une fonction $f(x)$ par rapport à x. Les fonctions "Minimize" et "Maximize" permet d'effectuer cette tâche :

In[74]:=
```
f[x_] := 5 + (x - 3)^2
```

In[75]:=
```
Minimize[f[x], x]
```

Out[75]=
```
{5, {x → 3}}
```

In[76]:=
```
g[x_] := Exp[-Abs[x + 2]]
```

In[77]:=
```
Maximize[g[x], x]
```

Out[77]=
```
{1, {x → -2}}
```

Ces fonctions retourne la valeur de l'extré mum et sa localisation en x.

Il existe également des variantes numériques de ces fonctions dans le cas où l'extré mum n'est pas calculable exactement :

In[78]:=
```
NMinimize[x^4 + x^3 + x^2 + x + 1, x]
```

Out[78]=
```
{0.673553, {x → -0.60583}}
```

In[79]:=
```
NMaximize[-x^4 + x^3 + x^2 + x + 1, x]
```

Out[79]=
```
{3.33157, {x → 1.28858}}
```

Les fonctions "FindMinimum" et "FindMaximum" permettent de rechercher des minimums et maximums *locaux* en donnant un point de départ. Par exemple, la recherche d'un minimum local de la fonction $\sin(x)\exp\left(-x^2\right)$ en partant de $x = -1.5$ donne

In[80]:=
```
FindMinimum[Sin[x] Exp[-x^2], {x, -1.5}]
```

Out[80]=
```
{-0.396653, {x → -0.653271}}
```

En partant de $x = 1.2$, on trouve un autre minimum local

In[81]:=
```
FindMinimum[Sin[x] Exp[-x^2], {x, 1.2}]
```

Out[81]=
```
{-2.94488 × 10^-6, {x → 3.29231}}
```

De la même manière, voici un exemple de recherche d'un maximum local en partant de x = 2

In[82]:=
```
FindMaximum[Sin[x] Exp[-x^2], {x, 2}]
```

Out[82]=
```
{0.396653, {x → 0.653271}}
```

Systèmes d'équations

La fonction "Solve" que l'on a déjà vue ne permet pas uniquement de résoudre une seule équation à une seule inconnue. Elle peut traiter n'importe quel nombre d'équations et d'inconnues. Par exemple, pour résoudre le système d'équations linéaires à 3 inconnues suivant

$$\begin{cases} x + 2y - z = 2 \\ 4x + 2y - 3z = 1 \\ 2x - y + z = -5 \end{cases}$$

on utilise "Solve" de la façon suivante (après éventuellement avoir effacé avec la fonction "Clear" toutes valeurs pouvant être déjà contenues dans *x*, *y* et *z*)

In[83]:=
```
Clear[x, y, z]
```

In[84]:=
```
Solve[{x + 2 y - z == 2, 4 x + 2 y - 3 z == 1, 2 x - y + z == -5}, {x, y, z}]
```

Out[84]=
$$\left\{\left\{x \to -\frac{17}{13},\ y \to \frac{12}{13},\ z \to -\frac{19}{13}\right\}\right\}$$

On peut aussi essayer de résoudre des équations non linéaires (qui peuvent avoir de multiples solutions)

In[85]:=
```
Solve[{x^2 + 2 / y == 2, 4 x + 2 y == 1}, {x, y}]
```

Out[85]=

$$\left\{\left\{x \to \frac{1}{12}\left(1 - \frac{97}{\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3}} - \left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3}\right),\right.\right.$$
$$\left.y \to \frac{1}{3} + \frac{97}{6\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3}} + \frac{1}{6}\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3}\right\},$$
$$\left\{x \to \frac{1}{4}\left(\frac{1}{3} + \frac{97}{6\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3}} + \frac{97\,i}{2\sqrt{3}\,\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3}} + \right.\right.$$
$$\left.\left.\frac{1}{6}\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3} - \frac{i\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3}}{2\sqrt{3}}\right),\right.$$
$$\left.y \to \frac{1}{3} - \frac{97\left(1 + i\sqrt{3}\right)}{12\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3}} - \frac{1}{12}\left(1 - i\sqrt{3}\right)\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3}\right\},$$
$$\left\{x \to \frac{1}{4}\left(\frac{1}{3} + \frac{97}{6\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3}} - \frac{97\,i}{2\sqrt{3}\,\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3}} + \right.\right.$$
$$\left.\left.\frac{1}{6}\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3} + \frac{i\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3}}{2\sqrt{3}}\right),\right.$$
$$\left.\left.y \to \frac{1}{3} - \frac{97\left(1 - i\sqrt{3}\right)}{12\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3}} - \frac{1}{12}\left(1 + i\sqrt{3}\right)\left(-577 + 12\,i\,\sqrt{4026}\right)^{1/3}\right\}\right\}$$

mais très vite les calculs exacts ne sont plus faisables même par *Mathematica* et il faut alors utiliser une résolution numérique approchée avec "NSolve"

In[86]:=
```
NSolve[{x^2 + y^5 == 2, 4 x^3 + 2 / y == 1}, {x, y}]
```

Out[86]=
```
{{x → -0.072384 + 0.739891 i, y → 0.38212 - 1.14319 i},
 {x → -0.072384 - 0.739891 i, y → 0.38212 + 1.14319 i}, {x → 1.41431, y → -0.193873},
 {x → -1.41417, y → 0.162433}, {x → 0.874548 - 0.128913 i, y → -0.82637 - 0.646612 i},
 {x → 0.874548 + 0.128913 i, y → -0.82637 + 0.646612 i},
 {x → -0.320864 + 0.7946 i, y → -0.948779 - 0.748774 i},
 {x → -0.320864 - 0.7946 i, y → -0.948779 + 0.748774 i},
 {x → -0.586783, y → 1.1061}, {x → -0.618552 - 0.450056 i, y → 0.286654 - 1.10064 i},
 {x → -0.618552 + 0.450056 i, y → 0.286654 + 1.10064 i},
 {x → 0.271527 + 0.492429 i, y → 1.16887 - 0.0286861 i},
 {x → 0.271527 - 0.492429 i, y → 1.16887 + 0.0286861 i},
 {x → 0.677515 - 0.330657 i, y → 0.399648 - 1.03907 i},
 {x → 0.677515 + 0.330657 i, y → 0.399648 + 1.03907 i},
 {x → -0.518467 + 0.692542 i, y → -0.999477 + 0.634312 i},
 {x → -0.518467 - 0.692542 i, y → -0.999477 - 0.634312 i}}
```

Boucle Do

Une boucle "Do" permet d'exécuter plusieurs fois une instruction. La commande "Do[instruction, {i, debut, fin}]" fait varier l'entier "i" (appelé "itérateur") de l'entier "debut" à l'entier "fin" et exécute à chaque fois "instruction". Par exemple, pour afficher "Coucou" 5 fois, on utilise une boucle Do exécutant l'instruction Print["Coucou"] lorsque i varie de 1 jusqu'à 5 :

In[87]:=
```
Do[Print["Coucou"], {i, 1, 5}]
```

```
Coucou

Coucou

Coucou

Coucou

Coucou
```

ce qui ne semble pas servir à grand chose... Les boucles Do sont plus intéressantes si on a besoin d'utiliser l'entier i dans l'instruction. Par exemple, affichons les 10 premiers entiers

In[88]:=
```
Do[Print[i], {i, 1, 10}]
```

```
1

2

3

4

5

6

7

8

9

10
```

Plus intéressant encore, utilisons une boucle Do pour calculer la somme des 100 premiers entiers $s = \sum_{i=1}^{100} i$. On commence par initialiser la variable s à 0

In[89]:=
```
s = 0
```

Out[89]=
```
0
```

puis on fait varier i de 1 jusqu'à 100 et à chaque itération on calcule s+i et on affecte le résultat à la variable s

In[90]:=
```
Do[s = s + i, {i, 1, 100}]
```

A la fin des itérations, la variable s contient la somme recherchée

In[91]:=
```
s
```

Out[91]=
```
5050
```

Il est souvent plus clair (et plus sûr) de regrouper ces trois instructions en une seule instruction dans une seule cellule en les séparant par des ";" (qui permettent de supprimer les affichages intermédiaires)

In[92]:=
```
s = 0;
Do[s = s + i, {i, 1, 100}];
s
```

Out[94]=
```
5050
```

Il se trouve que *Mathematica* dispose d'une fonction "Sum" permettant de faire cette somme plus directement

In[95]:=
```
Sum[i, {i, 1, 100}]
```

Out[95]=
```
5050
```

Néanmoins, les boucles Do permettent de faire beaucoup plus de choses.

Par défaut, l'itérateur varie par incrément de 1, mais on peut aussi utiliser n'importe quel autre incrément. Par exemple, dans la boucle Do suivante l'entier j varie de 3 jusqu'à 10 par incrément de 2

In[96]:=
```
Do[Print[j], {j, 3, 10, 2}]
```

```
3

5

7

9
```

Et dans la boucle suivante l'entier j varie de 5 jusqu'à 2 par incrément de -1

In[97]:=
```
Do[Print[j], {j, 5, 2, -1}]
```

```
5

4

3

2
```

Enfin, on peut exécuter plusieurs intructions dans une boucle Do en les séparant par ";". Par exemple, calculons la somme s et le produit p des 10 premiers entiers, $s = \sum_{i=1}^{10} i$ et $p = \prod_{i=1}^{10} i$, avec une seule boucle Do :

In[98]:=
```
s = 0;
p = 1;
Do[
  s = s + i;
  p = p * i, {i, 1, 10}];
Print[s];
Print[p];
```

```
55

3 628 800
```

Les retours à la ligne ne sont pas obligatoires mais facilitent la lecture.

Structure conditionnelle If

La structure conditionnelle "If" permet de choisir d'exécuter une instruction ou une autre suivant qu'une condition (appelée *expression logique* en informatique) est vraie ou fausse. La syntaxe est la suivante : "If [expression logique, instruction1, instruction2]". Si "expression logique" est vraie ("True" en *Mathematica*) alors uniquement "instruction1" sera exécutée; si au contraire "expression logique" est fause ("False" en *Mathematica*) alors uniquement "instruction2" sera exécutée.

Par exemple, utilisons une structure "If" pour écrire la fonction $f(x)$ définie par morceaux : $f(x) = e^x$ si $x > 0$ et $f(x) = x^2$ si $x \leq 0$

In[103]:=
```
f[x_] := If[x > 0, Exp[x], x^2]
```

Etant donné deux variables x et y (les ";" permettent de supprimer l'affichage du résultat qui est sans intérêt ici)

In[104]:=
```
x = -2;
```

In[105]:=
```
y = 3;
```

voici quelques exemples d'expressions logiques pouvant être utilisées dans un "If" :

- test d'égalité (attention, il faut utiliser "==" et non pas "=")

In[106]:=
```
x == y
```

Out[106]=
```
False
```

- test de non égalité

In[107]:=
```
x != y
```

Out[107]=
```
True
```

- test de supériorité

In[108]:=
```
x > y
```

Out[108]=
```
False
```

- test de supériorité ou d'égalité

In[109]:=
```
x >= y
```

Out[109]=
```
False
```

- test d'infériorité

In[110]:=
```
x < y
```

Out[110]=
```
True
```

- test d'infériorité ou d'égalité

In[111]:=
```
x <= y
```

Out[111]=
```
True
```

On peut aussi combiner des expressions logiques avec un ET logique ("&&" en *Mathematica*) ou "OU" logique ("| |" en *Mathematica*). Par exemple, pour tester si $x < 0$ ET $y = 0$, on écrit

In[112]:=
```
x < 0 && y == 0
```

Out[112]=
```
False
```

pour tester si $x < 0$ OU $y = 0$, on écrit

In[113]:=
```
x < 0 || y == 0
```

Out[113]=
```
True
```

Calcul vectoriel

En *Mathematica*, les composantes d'un vecteur s'écrivent avec une liste simple, c'est-à-dire qu'elles sont regroupées entre accolades et séparées par des virgules. Par exemple, dans un espace de dimension 3, définissons le vecteur **u** suivant :

In[114]:=
```
u = {2, -3, 1}
```

Out[114]=
```
{2, -3, 1}
```

Chaque composante d'un vecteur est individuellement accessible (et modifiable) avec la syntaxe u[[i]] où i est le numéro de la composante. Ainsi, la 2ème composante s'obtient par

In[115]:=
```
u[[2]]
```

Out[115]=
```
-3
```

ou en utilisant les raccourcis clavier ESC[[ESC et ESC]]ESC pour rendre les doubles crochets plus jolis et plus lisibles :

In[116]:=
```
u⟦2⟧
```

Out[116]=
```
-3
```

On peut faire un certain nombre d'opérations sur ce vecteur. Par exemple, une multiplication par un scalaire :

In[117]:=
```
3 u
```

Out[117]=
```
{6, -9, 3}
```

ou le calcul de sa norme :

In[118]:=
```
Norm[u]
```

Out[118]=
$$\sqrt{14}$$

Définissons maintenant un deuxième vecteur **v**

In[119]:=
```
v = {4, -1, 5}
```

Out[119]=
```
{4, -1, 5}
```

En *Mathematica*, la multiplication directe **u** * **v** (ou simplement **u** **v**) signifie multiplier la ième composante de **u** avec la même ième composante de **v**

In[120]:=
```
u * v
```

Out[120]=
```
{8, 3, 5}
```

Le produit scalaire est obtenu par l'opération "."

In[121]:=
```
u.v
```

Out[121]=
```
16
```

Si nous chargons le paquetage "VectorAnalysis" (qui n'est pas chargé par défaut)

In[122]:=
```
Needs["VectorAnalysis`"]
```

nous avons alors accès à d'autres fonctions d'analyse vectorielle, comme la fonction "CrossProduct" qui calcule le produit vectoriel de deux vecteurs

In[123]:=
```
CrossProduct[u, v]
```

Out[123]=
```
{-14, -6, 10}
```

Exercice : vérifier que le produit scalaire peut être calculé plus explicitement en faisant les produits composante par composante avec la boucle Do suivante :

In[124]:=
```
produit = 0;
Do[produit = produit + u[[i]] v[[i]], {i, 1, 3}];
produit
```

Out[126]=
```
16
```

Calcul matriciel

En *Mathematica*, une matrice est définie comme une liste de vecteurs lignes. Par exemple, définissons la matrice **A** suivante de dimension 3× 3

In[127]:=
```
A = {{1, 1, 0}, {2, -2, 0}, {0, 0, 3}}
```

Out[127]=
```
{{1, 1, 0}, {2, -2, 0}, {0, 0, 3}}
```

On peut utiliser la fonction "MatrixForm" (par exemple en pipeline) pour afficher la matrice sous forme plus claire

In[128]:=
```
A // MatrixForm
```

Out[128]//MatrixForm=

$$\begin{pmatrix} 1 & 1 & 0 \\ 2 & -2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

La syntaxe A[[i,j]](ou A[[i]][[j]]) permet d'accéder à l'élement (i,j) de **A**, par exemple

In[129]:=
```
A[[2, 1]]
```

Out[129]=
```
2
```

Essayons quelques opérations sur cette matrice :

- multiplication par un scalaire

In[130]:=
```
3 A
```

Out[130]=
```
{{3, 3, 0}, {6, -6, 0}, {0, 0, 9}}
```

- calcul de la transposée

In[131]:=
```
Transpose[A] // MatrixForm
```

Out[131]//MatrixForm=

$$\begin{pmatrix} 1 & 2 & 0 \\ 1 & -2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

- calcul de la trace

In[132]:=
```
Tr[A]
```

Out[132]=
```
2
```

- calcul du déterminant

In[133]:=
```
Det[A]
```

Out[133]=
```
-12
```

- calcul de la matrice inverse

In[134]:=
```
Inverse[A]
```

Out[134]=
$$\left\{\left\{\frac{1}{2}, \frac{1}{4}, 0\right\}, \left\{\frac{1}{2}, -\frac{1}{4}, 0\right\}, \left\{0, 0, \frac{1}{3}\right\}\right\}$$

- calcul du cube de la matrice, $\mathbf{A}^3$

In[135]:=
```
MatrixPower[A, 3]
```

Out[135]=
```
{{1, 5, 0}, {10, -14, 0}, {0, 0, 27}}
```

- calcul des valeurs propres et des vecteurs propres

In[136]:=
```
Eigenvalues[A]
```

Out[136]=
$$\left\{3, \frac{1}{2}\left(-1-\sqrt{17}\right), \frac{1}{2}\left(-1+\sqrt{17}\right)\right\}$$

In[137]:=
```
Eigenvectors[A]
```

Out[137]=
$$\left\{\{0, 0, 1\}, \left\{1+\frac{1}{4}\left(-1-\sqrt{17}\right), 1, 0\right\}, \left\{1+\frac{1}{4}\left(-1+\sqrt{17}\right), 1, 0\right\}\right\}$$

Définissons maintenant une deuxième matrice rectangulaire **B**

In[138]:=
```
B = {{-3, 2}, {1, 0}, {7, 8}}
```

Out[138]=
```
{{-3, 2}, {1, 0}, {7, 8}}
```

In[139]:=
```
B // MatrixForm
```

Out[139]//MatrixForm=
$$\begin{pmatrix} -3 & 2 \\ 1 & 0 \\ 7 & 8 \end{pmatrix}$$

Le produit matriciel de **A** par **B** se calcule avec l'opération "."

In[140]:=
```
A.B
```

Out[140]=
```
{{-2, 2}, {-8, 4}, {21, 24}}
```

Par contre, les dimensions de ces matrices interdissent le produit matriciel de **B** par **A** et *Mathematica* ne s'y trompe pas et refuse de le faire

In[141]:=
```
B.A
```

```
Dot::dotsh :
 Tensors {{-3, 2}, {1, 0}, {7, 8}} and {{1, 1, 0}, {2, -2, 0}, {0, 0, 3}} have incompatible
     shapes. >>
```

Out[141]=
```
{{-3, 2}, {1, 0}, {7, 8}}.{{1, 1, 0}, {2, -2, 0}, {0, 0, 3}}
```

Exercice : vérifier que le produit matriciel peut être calculé plus explicitement en faisant les produits élément par élément avec les boucles Do imbriquées suivantes :

In[142]:=
```
produit = {{0, 0}, {0, 0}, {0, 0}};
Do[
  Do[
   Do[produit[[i, j]] = produit[[i, j]] + A[[i, k]] B[[k, j]], {k, 1, 3}],
   {i, 1, 3}],
  {j, 1, 2}];
produit
```

Out[144]=
```
{{-2, 2}, {-8, 4}, {21, 24}}
```

Equations différentielles

La fonction "DSolve" permet de résoudre des équations différentielles. Prenons l'exemple d'une équation différentielle linéaire du 1er ordre : y'(x)+2 y(x)=sin(x)

In[145]:=
```
Clear[x, y]
```

In[146]:=
```
DSolve[y'[x] + 2 y[x] == Sin[x], y[x], x]
```

Out[146]=
$$\left\{\left\{y[x] \to e^{-2x} C[1] + \frac{1}{5}(-\text{Cos}[x] + 2\,\text{Sin}[x])\right\}\right\}$$

C[1] est une constante d'intégration. On peut ajouter une condition initiale (ou aux limites) y(0)=0 pour déterminer la constante C[1] :

In[147]:=
```
DSolve[{y'[x] + 2 y[x] == Sin[x], y[0] == 0}, y[x], x]
```

Out[147]=
$$\left\{\left\{y[x] \to -\frac{1}{5} e^{-2x}\left(-1 + e^{2x}\,\text{Cos}[x] - 2\,e^{2x}\,\text{Sin}[x]\right)\right\}\right\}$$

Voici l'exemple d'une équation différentielle linéaire du 2ème ordre

In[148]:=
```
DSolve[y''[x] + 3 y'[x] + 2 y[x] == Exp[x], y[x], x]
```

Out[148]=
$$\left\{\left\{y[x] \to \frac{e^x}{6} + e^{-2x} C[1] + e^{-x} C[2]\right\}\right\}$$

et d'une équation différentielle non linéaire

In[149]:=
```
DSolve[(x^2 - y[x]^2) y'[x] - 2 x y[x] == 0, y[x], x]
```

Out[149]=
$$\left\{\left\{y[x] \to \frac{1}{2}\left(e^{C[1]} - \sqrt{e^{2C[1]} - 4x^2}\right)\right\}, \left\{y[x] \to \frac{1}{2}\left(e^{C[1]} + \sqrt{e^{2C[1]} - 4x^2}\right)\right\}\right\}$$

Si l'équation différentielle n'est pas soluble exactement, on peut trouver une solution numérique avec la fonction "NDSolve" (voir l'aide).

Traitement de données numériques

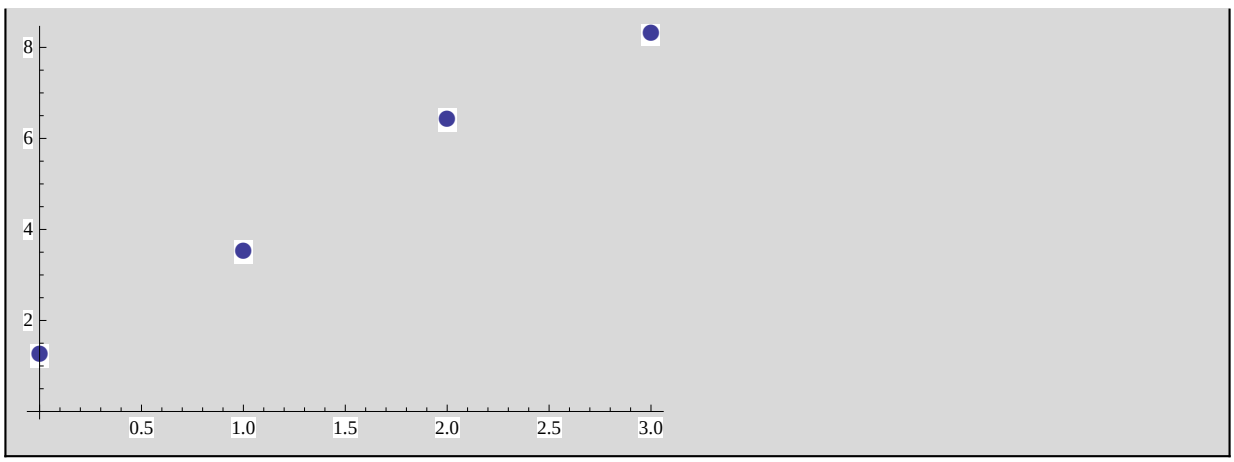
Imaginons que nous disposions de données numériques sous la forme d'une série de points (x, y). Le format adéquat pour représenter ces données dans *Mathematica* est une liste de listes :

In[150]:=
```
donnees = {{0., 1.23}, {1., 3.5}, {2., 6.4}, {3, 8.3}};
```

On peut alors facilement tracer un graphique avec ces données

In[151]:=
```
ListPlot[donnees, PlotMarkers → {Automatic, Medium}]
```

Out[151]=



On a souvent besoin de faire un *fit* (aussi appelé ajustement ou régression) de ces données. Par exemple, les paramètres a et b d'un fit linéaire $y = a\,x + b$ peuvent être déterminés avec la fonction" FindFit" :

In[152]:=
```
Clear[a, b, x]
```

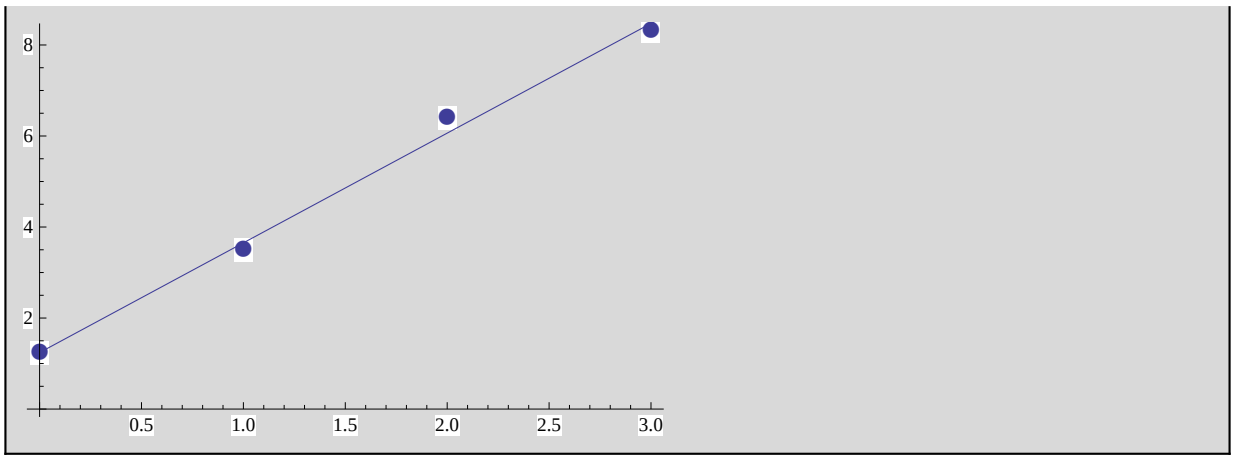
In[153]:=
```
parametres = FindFit[donnees, a x + b, {a, b}, x]
```

Out[153]=
```
{a → 2.411, b → 1.241}
```

Vérifions la qualité du fit en superposant (avec la fonction "Show") le graphique des données et celui du fit :

In[154]:=
```
Show[ListPlot[donnees, PlotMarkers → {Automatic, Medium}],
 Plot[a x + b /. parametres, {x, 0, 5}]]
```

Out[154]=



Parfois, au lieu d'un fit, on préfère utiliser une interpolation des données. La fonction "Interpolation" réalise cette tâche :

In[155]:=
```
f = Interpolation[donnees]
```

Out[155]=
```
InterpolatingFunction[{{0., 3.}}, <>]
```

Elle retourne une fonction f donnant une interpolation des données pour n'importe quelle valeur de x dans le domaine de définition indiqué [0,3]. Par défaut, *Mathematica* réalise une interpolation de degré 3. Par exemple, la valeur de la fonction d'interpolation $f(x)$ pour $x = 1.5$ est

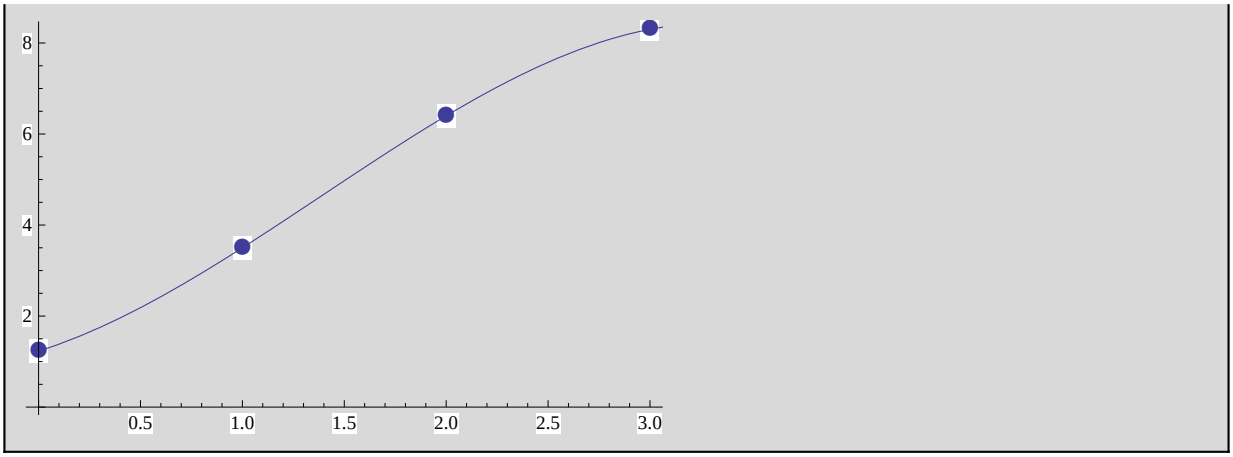
In[156]:=
```
f[1.5]
```

Out[156]=
```
4.97313
```

On peut aussi tracer l'interpolation, superposée aux données :

In[157]:=
```
Show[ListPlot[donnees, PlotMarkers → {Automatic, Medium}], Plot[f[x], {x, 0, 5}]]
```

Out[157]=



Programmation avec des modules

En *Mathematica*, un "Module" est l'équivalent de ce que l'on appelle souvent une "routine" dans d'autres langages de programmation. Il s'agit d'un ensemble d'instructions regroupés dans une même entité indépendante. Reprenons l'exemple du calcul du produit scalaire de deux vecteurs par une boucle Do. On peut utiliser un module pour définir une fonction "produitscalaire" prenant comme arguments deux vecteurs **u** et **v**, et retournant la valeur de leur produit scalaire :

In[158]:=
```
produitscalaire[u_, v_] := Module[{produit, i},
  produit = 0;
  Do[produit = produit + u[[i]] v[[i]], {i, 1, 3}];
  produit]
```

Dans cet exemple, "{produit, i}" sur la première ligne indique que "produit" et "i" sont des variables locales, c'est-à-dire qu'elles seront modifiées uniquement à l'intérieur du Module, et pas globalement pour tout le notebook. Utiliser des variables locales limite les risques d'erreur en évitant les possibles conflits avec d'autres parties du notebook où ces mêmes variables pourraient être aussi utilisées. L'autre avantage de créer des modules est de pouvoir réutiliser facilement autant que l'on veut une même partie de code. Par exemple, on peut calculer le produit scalaire des vecteurs {2, -3, 1} et {4, -1, 5}

In[159]:=
```
produitscalaire[{2, -3, 1}, {4, -1, 5}]
```

Out[159]=
```
16
```

puis le produit scalaire des vecteurs {2, 0, 1} et {2, -1, 3}

In[160]:=
```
produitscalaire[{2, 0, 1}, {2, -1, 3}]
```

Out[160]=
```
7
```

Découper un long code en petits modules indépendants permet de diminuer la complexité du code. C'est une bonne pratique de programmation.