Physique avec Mathematica

Christophe Coste

Cours *Mathematica* (Version 2006)

C. Coste

Physique avec *Mathematica*. Introduction générale.

0) Note sur la présentation

Toute **`Fonction`** *Mathematica* citée dans le texte adopte la typographie des cellules de commande [voir I-1) ci-dessous], comme sur cet exemple. Toute touche du Clavier, ou intitulé de Menu, est en noir sur fond gris. Les remarques purement syntaxiques faites au cours du TD seront en **rouge et en gras** à l'écran.

I) Introduction

I-1) Effectuer une commande

Avec *Mathematica*, vous pouvez réunir dans le même document, appelé *Notebook,* du texte (y compris des typographies mathématiques), des demandes de calculs et leurs résultats, ainsi que des graphiques. Notez les accolades carrées à droite qui délimitent une cellule (*Cell*). Dans le cas présent, il s'agit d'une cellule de texte.
La cellule ci-dessous est une cellule de commande. Vous effectuez la commande en sélectionnant cette cellule avec la souris (ou en activant le pointeur à l'intérieur de cette cellule) puis en appuyant sur la touche Enter du clavier. Effectuez la commande ci-dessous.
Notez la différence de typographie entre les cellules de commande, de résultat, et cellules de texte.

```
1 + 1
```

NB 0 : Tant que vous n'avez pas explicitement entrée une commande, celle-ci n'est pas prise en compte par le logiciel. Une fois qu'une commande est effectuée, il lui est attribué un numéro d'entrée (In[#]) et un numéro de sortie pour le résultat, lorsqu'il s'affiche (Out[#]). Ici le résultat s'affiche, et le numéro doit être 1 pour l'entrée et la sortie.
NB 1 : La valeur par défaut de toute frappe au clavier est une cellule de commande. Pour changer, par exemple en une cellule de texte, sélectionner la barre de cellule à droite, et naviguez dans les menus Format -> Style -> Text
NB 2 : Les cellules de textes sont inactives.

I-2) Sauvegarder le Notebook

En effectuant cette commande, vous avez modifié ce document. C'est le bon moment pour vérifier que vous savez sauvegarder un document!
Allez dans le menu File->Save As
Même si votre compte personnel est actif, je vous conseille de sauvegarder dans le dossier docu-

ment du disque dur du PC sur lequel vous travaillez, et de transférer le document dans votre dossier personnel en fin de TD (qui se trouve physiquement sur un autre disque dur, accessible en réseau). Ensuite quittez le logiciel, et rouvrez votre document. Vérifiez qu'il s'agit bien de la version que vous avez modifié!
Par la suite, n'hésitez pas à sauvegarder aussi souvent que possible! Faire File->Save sauvegardera automatiquement dans le même dossier du disque dur.

I-3) Opérations numériques

La multiplication s'écrit avec le signe * , ou plus simplement en laissant un espace entre deux nombres ou deux symboles. Attention à ne pas oublier cet espace!

```
2 * 3
```

```
2 × 3
```

```
23
```

Remarque : La numérotation des entrées/sorties permet le rappel des résultats, qui s'écrivent `%` Pourcentage % (résultat précédent), `%%` (résultat avant le précédent; généralisable à `n` signes `%`) ou `%p` (p-ième résultat). Cette écriture condensée est pratique, mais oblige à un ordre chronologique strict dans les calculs. On peut s'en affranchir en donnant un nom aux résultats (quels qu'ils soient; voir II-5).
Quelques exemple d'utilisations de cette syntaxe :

```
% + 2
```

```
%3 ^ 2
```

```
√%%%%
```

I-4) Syntaxe des fonctions *Mathematica*. Délimiteurs

Toute fonction *Mathematica* débute par une Majuscule. Ses arguments sont délimités par des [crochets droits].

```
Sin[π]
```

Les {accolades} délimitent les *listes*, dont les éléments peuvent être quelconques.
Voici un exemple de liste numérique (vous remarquerez que l'argument de la fonction `Sin` peut être une liste de nombre, le résultat étant alors la liste des sinus de ces nombres. On dit que `Sin` est une fonction qui possède l'attribut `Listable`, ce qui est le cas de la plupart des fonctions *Mathematica*. Certaines ne le sont pas, on y remédie au besoin en utilisant la fonctionnelle `Map`, décrite au TD N°2)

```
Sin[{π, π/2, π/3, π/4, π/5, π/6}]
```

Et un autre exemple comportant deux listes : une liste des fonctions dont on fait le tracé, et une liste donnant le domaine de variation de la variable.

```
Plot[{Sin[x], Cos[x]}, {x, 0, 2 π}]
```

Enfin, les (parenthèses) ont exactement la même fonction qu'en mathématiques

```
3 * (2 + 5)
```

Vous avez peut-être noté l'emploi d'un point-virgule lors de l'utilisation de `Plot` ci-dessus. **Le point-virgule a pour effet, sauf pour les fonctions à sorties graphiques telles que `Plot`, d'empêcher l'affichage du résultat**. Cela permet d'obtenir des Notebook bien plus lisibles, et doit être encouragé lorsque le résultat est évident (définition d'une variable) , inutile ou trop long (construction d'une

longue liste, par exemple). Ici, en son absence, une cellule indique la nature graphique de la sortie.

I-5) Utiliser l'aide en ligne

Pour obtenir des informations sur n'importe quelle fonction, il suffit d'entrer une requête où elle est précédée d'un point d'interrogation

```
? Sin
```

```
? Plot
```

Avec la version 5, ces explications sommaires sont suivies par un lien http vers l'aide en ligne. Cliquer sur le lien permet d'accéder à plus d'informations. Les fonctions ont un ou plusieurs *arguments* (un pour **Sin**, deux pour **Plot** (la fonction à tracer -ou la liste de fonctions à tracer- et le domaine de variation de la variable, qui est une liste), et des *options*. On accède aux options d'une fonction par la requête **Options**

```
Options[Sin]
```

```
Options[Plot]
```

Sin n'a aucune option, **Plot** en a beaucoup dont les valeurs par défaut sont indiquées.

I-6) Structure du logiciel

Mathematica, d'une certaine façon, est constitué de deux logiciels. L'un, le **FrontEnd,** assure *l'interprétation* des commandes de l'utilisateur, et en tire une *représentation interne* qu'un second logiciel, le **Kernel** utilise pour les calculs. L'interpréteur retranscrit alors les résultats sous une forme lisible par l'utilisateur. Un schéma général pourrait-être celui-ci :

Une requête peut ainsi être codée de différentes façons, toutes acceptables par l'interpréteur. Voici une demande de calcul d'intégrale formelle, en explicitant la fonction Integrate, puis en utilisant la palette standard
(menu File -> Palettes -> BasicInput)

```
Integrate[Sin[x], x]
```

$$\int \text{Sin}[x]\, dx$$

Cet agrément de lecture a une contrepartie, parfois facheuse : on oublie très vite que ce qu'on voit à l'écran ne correspond pas nécessairement à la structure interne de *Mathematica*! En particulier, une écriture aussi simple que a+b a exactement la même structure interne que toute fonction *Mathematica*, comme on le voit en utilisant **FullForm** :

```
FullForm[{a + b, a * b, a c, ∫ f[x] dx}]
```

ou de façon équivalente, en notation *postfix*

```
{a + b, a * b, a c, ∫ f[x] dx} // FullForm
```

La fonction **FullForm** a précisément comme objet de fournir à l'utilisateur la représentation exacte avec laquelle travaille le **Kernel**.

II) Quelques opérations basiques

II-1) Effectuer un calcul numérique

Si les valeurs numériques sont entrées sous forme *entières*, *Mathematica* s'efforce de fournir un résultat exact, et laisse la requête inévaluée s'il ne le peut pas. Si les valeurs numériques sont entrées sous forme décimale (lorsque les nombres sont terminés par, ou contiennent un point qui est le séparateur reconnu par le logiciel), il fournit une approximation numérique calculée avec 16 chiffres significatifs (Sur Mac, en version 4.2; ce nombre est une variable par défaut, notée **$MachinePrecision** - le **$** précéde tous les paramètres par défaut-).
Vous noterez que *Mathematica* connaît les angles en degrés, la notation ° correspondant à la constante $\frac{\pi}{180}$

```
$MachinePrecision
```

```
Sin[{π/5, π/5., 3, 3., 1/3, 0.3333, 45 °, 45. °}]
```

Pour obtenir une approximation numérique, on peut aussi employer la fonction **N**.

```
N[Sin[{π/5, π/5., 3, 3., 1/3, 0.3333, 45 °, 45. °}]]
```

Voici une syntaxe différente mais équivalente, la notation *postfix*.

```
Sin[{π/5, π/5., 3, 3., 1/3, 0.3333, 45 °, 45. °}] // N
```

Ces calculs peuvent être faits avec une précision arbitraire; **N** admet comme *option* le nombre de chiffres significatifs requis, mais ne les calcule que si l'erreur sur les données initiales permet d'obtenir la précision demandée. Par exemple, pour obtenir un résultat à 25 chiffres significatifs, les données initiales doivent comporter au moins autant de décimales, où être "exactes", c'est à dire sous forme rationnelle. Les constantes usuelles, telles que π et e doivent être entrées sous forme symbolique.

```
N[π/3, 25]
```

```
N[3.1416/3, 25]
```

Toutes les fonctions numériques sont définies dans le plan complexe (lorsque cela est possible, bien sûr!)

```
ArcSin[2.]
```

```
Sin[π/3 + I √π] // N
```

II-2) Effectuer un calcul symbolique

La grande richesse de *Mathematica* vient de ses capacités de calcul symboliques. Citons les dérivations (**Derivative**), intégrations (**Integrate**), recherche de solutions d'équations (**Solve**), intégrations d'équations différentielles (**DSolve**). Les deux premières n'ont pas besoin d'être explicitées, mais peuvent être codées avec la palette.

```
Derivative[1][Sin][t]
```

```
Sin'[t]
```

Syntaxe : bien noter que le prime s'applique à la fonction!
Nous verrons au TD2 que l'opération de dérivation est en fait une fonctionnelle, c'est à dire une fonction s'appliquant à d'autres fonctions.

```
∂t Sin[t]
```

$$\left\{\partial_{x,x} \frac{x\,y}{1+y^2}, \partial_{x,y} \frac{x\,y}{1+y^2}, \partial_{y,x} \frac{x\,y}{1+y^2}, \partial_{y,y} \frac{x\,y}{1+y^2}\right\}$$

```
Solve[∂x,y x y/(1 + y^2) == 0, y]
```

Syntaxe 1: première apparition du double signe égal, qui sert à définir une *équation*.
Il est indispensable de ne pas le confondre avec le signe égal simple, qui sert lui à nommer une variable et sera décrit au paragraphe II-5). En cas d'erreur, *Mathematica* interprète ce qui est sensé être une équation comme une définition, sans aucun contrôle de l'utilisateur, ce qui peut donner lieu à des conflits de définitions. Soyez donc attentifs dans l'écriture des équations.
Syntaxe 2: Notez aussi la structure particulière du résultat; il s'agit d'une liste de *règles de substitution*, sur lesquelles nous revenons dans le TD suivant.

```
DSolve[x''[t] + ω^2 x[t] == 0, x[t], t]
```

Syntaxe : Faites attention; x'' est codé par x prime prime et pas par x doubleprime !

Il est possible d'introduire des conditions initiales; le premier argument de la fonction est alors une *liste* d'équations (toujours écrites à l'aide du double signe égal)

```
DSolve[{x''[t] + ω^2 x[t] == 0, x[0] == a, x'[0] == 0}, x[t], t]
```

II-3) Effectuer des calculs numériques plus compliqués

Des fonctions comme **Solve**, **Integrate** ou **DSolve** n'ont souvent pas de solution exacte. Il est néanmoins toujours possible (lorsque seules des valeurs numériques interviennent) d'obtenir une approximation numérique. Pour **Integrate** ou **Solve**, on peut appliquer **N** au résultat, ou utiliser directement les fonctions **NIntegrate** et **NSolve**
Remarque : Pour **DSolve**, il faut appliquer la fonction **NDSolve**, qui n'a pas tout-à-fait la même syntaxe que **DSolve** et que nous étudions au paragraphe suivant.

```
∫_0^1 Exp[Cos[t]] dt
```

```
N[%]
```

```
NIntegrate[Exp[Cos[t]], {t, 0, 1}]
```

```
Solve[x^5 + 3 x^4 + 5 x^3 + 7 x^2 + 9 x + 11 == 0]
```

```
N[%]
```

```
NSolve[x^5 + 3 x^4 + 5 x^3 + 7 x^2 + 9 x + 11 == 0]
```

Comme autres fonctions utiles, nous allons évoquer les fonctions **FindRoot** (recherche numérique de racines d'équations) et **FindMinimum** (recherche numérique de minima de fonctions). Ces deux fonctions admettent deux syntaxes différentes.
NB : Dans ce type de calculs, il est très fortement recommandé de tracer d'abord les fonctions. Le point de départ de la recherche est en effet crucial, et on n'obtient pas nécessairement le même résultat selon le choix initial.

```
Plot[{x - 1, 1/(x + 1) + x^2 - 3}, {x, -0.5`, 3}]
```

Première écriture : lorsqu'une dérivée formelle de la fonction à étudier est calculable.

```
FindRoot[x - 1 == 1/(x + 1) + x^2 - 3, {x, 2}]
```

```
FindRoot[x - 1 == 1/(x + 1) + x^2 - 3, {x, 0}]
```

```
FindMinimum[1/(x + 1) + x^2 - 3, {x, 0.5}]
```

Deuxième écriture : indispensable lorsqu'une dérivée formelle de la fonction à étudier n'est pas calculable; facultative lorsque la dérivée existe. Il faut cette fois fournir un intervalle, et *Mathematica* recherche la solution dans cet intervalle.

```
FindRoot[x - 1 == 1/(x + 1) + x^2 - 3, {x, {0, 2}}]
```

```
FindRoot[x - 1 == 1/(x + 1) + x^2 - 3, {x, {-0.5, 0.5}}]
```

```
FindMinimum[1/(x + 1) + x^2 - 3, {x, {0, 1}}]
```

II-4) Résoudre numériquement des équations différentielles ordinaires

La fonction **NDSolve** n'a pas tout-à-fait la même syntaxe que **DSolve** car il faut impérativement fournir le bon nombre (égal à l'ordre de l'équation différentielle) de *conditions initiales* (plus rarement, on peut trouver des solutions en fixant des conditions aux limites; mais à la différence du cas précédent, l'existence et l'unicité de la solution n'est pas garantie).
Par ailleurs, remarquez que, pour l'utiliser ultérieurement, nous avons donné un nom (**sol**) au résultat du premier calcul.

```
sol = NDSolve[{x''[t] + x[t] == 0, x[0] == 1, x'[0] == 0}, x, {t, 0, 10 π}]
```

La solution, dans l'approche la plus simple de résolution numérique d'équations différentielles, est une liste de valeurs numériques correspondant à des intervalles de temps équidistants. *Mathematica* donne la solution sous forme de fonction interpolée (**InterpolatingFunction**) qui permet de calculer les valeurs de la fonction en un point quelconque, grâce à une interpolation (meilleure que linéaire) entre valeurs calculées successives.
Syntaxe : Comme la solution est donnée sous forme de règle de substitution, nous utilisons la syntaxe `x[t]/.sol` qui signifie en clair "Lorsque la fonction `x` apparaît dans une expression, elle doit être remplacée selon la règle définie par le résultat nommé `sol`".

```
Plot[{x[t] /. sol, x'[t] /. sol}, {t, 0, 2 π}]
```

L'instruction suivante est un test classique permettant de vérifier la précision de la solution numérique. La fonction **Random[]**, qui donne un nombre aléatoire entre 0 et 1, permet de faire ce test sur des valeurs "typiques".

```
Table[{t, x[t]^2 + x'[t]^2} /. t → 10 π RandomReal[] /. sol, {5}]
```

II-5) Définir une variable globale

On définit une *variable globale* à l'aide du signe *égal* simple. La variable peut être de n'importe quel type (numérique, symbolique, graphique, etc...). Nous venons de le faire en définissant la variable **sol**.

```
a = 3
```

Notez que lors de la définition d'une variable, *Mathematica* imprime sa valeur. Cette sortie étant parfois inutile, on peut finir la requête par un point-virgule

```
a = 4;
```

```
b = √(1 + z^3) ;
```

```
g1 = Plot[Sin[x], {x, 0, 2 π}]
```

Ces variables sont "globales", c'est-à-dire que leur définition est désormais valable dans tout le Notebook. On peut désormais obtenir des informations sur les variables auxquelles nous avons *assigné* une valeur

```
? a
```

```
? b
```

Vous noterez que la deuxième définition de `a` a écrasée la première! On peut maintenant utiliser ces variables

```
b^2
```

```
Show[g1, Frame → True]
```

II-6) Annuler une définition de variable globale

En informatique, et ce quel que soit le langage utilisé, une fréquente source d'erreur est le conflit entre variables. Voyons un exemple.

```
Solve[a^2 - 3 a - 5 == 0]
```

Cette instruction échoue car a est désormais une variable globale, et ne peut plus être utilisée comme variable muette. En effet

```
a^2 - 3 a - 5
```

est bien différent de 0! Il est donc très important, quand on change de problème, de supprimer toutes les définitions de variables globales. Pour cela, il existe la fonction `Clear`

```
Clear[a, b, g1]
```

```
? a
```

Si maintenant nous essayons à nouveau de résoudre notre équation, ça marche!

```
Solve[a^2 - 3 a - 5 == 0]
```

Ceci étant, lorsqu'on résout un problème, on peut être amené à définir plusieurs variables lors de la recherche, et il est fréquent qu'on s'y perde un peu au bout d'un moment. Si on se trompe dans l'écriture d'une équation, en mettant un seul signe égal au lieu de deux, on crée ainsi des définitions incontrôlées, et une situation inextricable! Dans tous les cas, il peut être alors souhaitable d'éliminer toutes les variables, pour repartir sur des bases saines.
Une première solution, assez brutale, consiste à quitter le Kernel. Toutes les définitions sont alors automatiquement supprimées, et il suffit de ré-exécuter les instructions dont on est satisfait. Il suffit de naviguer dans les menus et faire
Kernel -> Quit Kernel -> Local . Une fois ceci fait, il faut entrer à nouveau toutes les requêtes nécessaires.
Une autre solution, plus astucieuse, consiste à utiliser l'instruction suivante :

```
Remove["Global`*"]
```

L'explication derrière cette ruse de Sioux est assez longue. Vous la trouverez dans l'appendice II-9, que vous pourrez d'ailleurs lire quand vous serez plus familier du logiciel.

II-7) Listes, vecteurs, matrices

Les listes sont une structure fondamentale du logiciel *Mathematica*. Délimitées par des accolades { et }, leurs éléments peuvent être eux-même des listes, et sont séparés par des virgules.
Elles sont fréquemment utilisées comme arguments de fonctions (domaine de variation de la variable dans **Plot**, ou **NDSolve**, liste d'équations dans **Solve**, **DSolve**, **NDSolve**, etc...).
Elles apparaissent aussi très souvent comme résultat d'une requête, et il est indispensable d'apprendre à les manipuler correctement.

Une liste peut bien sûr s'écrire directement au clavier. Les listes unidimensionnelles sont des vecteurs colonne, les matrices sont écrites lignes par ligne, chaque ligne étant elle-même une liste. On les visualise avec la fonction **MatrixForm.** Voici quelques exemples:

```
{a, b, c, d} // MatrixForm
```

```
m = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

```
MatrixForm[m]
```

Remarque importante : Ceux et celles d'entre vous qui sont habitué(e)s à des langages moins évolués seront peut-être frappés par la simplicité d'utilisation de *Mathematica*. Il n'y a pas à se poser la question du type de variable que l'on définit. Mieux, on peut changer ce type en réaffectant à un symbole donné une autre expression!

```
m[[2, 2]]
```

```
m = 3
```

```
m[[2, 2]]
```

Il est possible d'écrire les matrices à l'aide de la palette, paramétrée par défaut pour des matrices 2 × 2. Il est possible d'écrire des matrices quelconques en utilisant le menu Edit -> Expression Input -> Add Row (pour les lignes) ou
Add Column (pour les colonnes). On peut aussi utiliser directement le menu Input -> Create Table/Matrix/Palette et configurer la matrice ou la table désirée dans le menu qui apparaît alors.

La fonction permettant de créer des listes est **Table**. Ces listes peuvent être symboliques ou numériques

```
Table[a^i, {i, 1, 5}]
```

```
Table[0.1 i, {i, 1, 5}]
```

```
mm = Table[1/Sqrt[9 + i^2], {i, 0, 4}]
```

Elles peuvent être de dimensions quelconques. Construisons par exemple la matrice **m** définie plus haut :

```
Table[i + j - 1, {i, 1, 7, 3}, {j, 1, 3}] // MatrixForm
```

Syntaxe : Les éléments de toute liste sont appelés entre [[doubles crochets droits]]. Cette notation est une abréviation pour la fonction Part

```
mm[[3]]
```

```
Part[mm, 3]
```

Dans **m[[i,j]]**, qui est une matrice, **i** est le numéro de ligne et **j** le numéro de colonne. Une ligne

étant elle-même une liste, il est équivalent d'écrire **m[[i]][[j]]**

```
m[[2, 3]]
```

```
m[[3, 2]]
```

```
m[[2]][[3]]
```

Appel de la deuxième ligne

```
m[[2]]
```

Appel de la deuxième colonne

```
m[[All, 2]]
```

Pour extraire plusieurs éléments d'une liste, il faut procéder ainsi :

```
long = Table[Prime[i], {i, 1, 100}];
```

```
long[[{1, 2, 3, 4, 5}]]
```

```
long[[Table[i, {i, 1, 100, 5}]]]
```

Il est impératif de savoir manipuler des listes, car bien souvent les réponses sont données sous forme de *liste de règles de substitutions*, comme nous l'avons vu en II-2) et II-3).

Une des difficultés avec les listes est de bien traiter leur structure, qui peut-être extrèmement imbriquée. Commençons par cet exemple qui paraît simple, mais se révèle un peu subtil.

```
sol = DSolve[{x''[t] + x[t] == 0, x[0] == 1, x'[0] == 0}, x, t]
```

La requête suivante ne donne pas le résultat escompté...

```
(x /. sol)[u]
```

Tout vient de la structure de sol, qui est une liste de liste (deux niveaux d'accolades imbriquées!). Si on se réfère au bon élément, qui constitue une liste de règle de substitution (à un seul élément), on trouve un résultat correct :

```
sol[[1]]
```

```
(x /. sol[[1]])[u]
```

Il est donc très important de bien analyser la structure, parfois compliquée, des listes fournies automatiquement par *Mathematica*. Il existe une fonction très puissante, **Flatten**, qui permet d'obtenir toujours une liste d'éléments, à partir d'une liste aussi imbriquée soit-elle. Comparez **solbis** à **sol** calculé précédemment!

```
solbis = DSolve[{x''[t] + x[t] == 0, x[0] == 1, x'[0] == 0}, x, t] // Flatten
```

```
(x /. solbis)[u]
```

```
listex = {1, 2, {3, 4, {5, 6, {7, 8, {9, 10}}}}};
```

```
Flatten[listex]
```

On peut aussi supprimer progressivement les imbrications

```
Flatten[listex, 1]
```

```
Do[Print[Flatten[listex, i]], {i, 1, 4}]
```

Finissons par une remarque délicate, qui peut être laissée de côté en première lecture. En utilisant **FullForm**, on voit qu'une liste n'est jamais qu'une fonction comme une autre

```
liste = {1, 2, a, π/4}
```

```
FullForm[liste]
```

De ce fait, la fonction **Part** ne s'applique pas seulement aux listes, mais en fait à toute fonction *Mathematica*

```
liste[[4]]
```

```
liste[[4]][[1]]
```

```
liste[[4]][[2]]
```

II-8) Exercices

Exercice II-1 : Pouvez-vous interpréter le résultat suivant?

```
N[Sin[{π/5, π/5., 3, 3., 1/3, 0.3333}], 25]
```

Exercice II-2 : Calculer les coordonnées des points d'intersection entre la droite y = $\frac{x}{2}$ et la tangente hyperbolique (attention à la notation anglosaxonne **Tanh**)

Exercice II-3 : Calculer exactement l'aire entre la parabole y = 2 - x^2 + 3 x et la droite y = x - 1, lorsque la parabole est au dessus de la droite.

Exercice II-4: Entrer avec la palette la matrice $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$. Extraire sa deuxième ligne, puis sa troisième colonne. Construire directement cette matrice avec la fonction **Table**.

Exercice II-5 : Reprenez la liste **listex** définie plus haut. A partir de cette liste, obtenez par quatre instructions successives les listes des nombres de 3 à 10, 5 à 10, 7 à 10 puis enfin 9 à 10.
Nettement plus difficile : faites le avec une seule ligne de commande, en programmant une boucle **Do**, en utilisant la fonction **Part** et la fonction **Fold** (ou **FoldList**, qu'il est d'ailleurs fortement conseillé de regarder pour comprendre comment fonctionne **Fold**).

II-9) Appendice : Nettoyage du Kernel.

Il arrive que des erreurs successives (très fréquemment, l'écriture fautive d'équations avec un seul signe égal) introduisent des conflits de définition inextricables. Nous avons déjà vu une solution, qui consiste à quitter puis relancer le **Kernel**.
Il existe une façon plus astucieuse d'obtenir le même résultat, en utilisant la fonction **Remove[]**.
Nous allons d'ailleurs partir de bases bien contrôlées en nettoyant le Kernel!

```
Remove["Global`*"]
```

II-9-1) Fonctions Clear[] et Remove[]

La fonction **Clear[]** supprime la valeur attribuée à un symbole, mais celui-ci reste présent dans la mémoire du **Kernel**, comme on le voit pour les exemples ci-dessous

```
a = 3;
```

```
a
```

```
? a
```

a est bien définit dans le *contexte* **Global** (voir point suivant), et prends la valeur 3.

```
Clear[a]
```

```
? a
```

Aucune valeur n'est plus attribuée à **a**, mais il figure toujours dans le contexte **Global**.

```
Remove[a]
```

```
? a
```

```
? b
```

Désormais, **a** ne figure plus dans le contexte, de même que **b** qui n'a pas été défini.

Nous pouvons retenir que la meilleure façon de supprimer une définition malencontreuse d'une variable est donc d'appliquer la fonction **Remove[]**.

II-9-2) Notion de contexte (Context)

Au cours du développement d'applications dans *Mathematica*, par exemple en réalisant un **Package**, on est amené à définir des variables dont ils est indispensable que les noms n'entrent pas en conflit avec ceux d'autres variables. Par ailleurs, il est souhaitable que ces noms soient courts, pour que les programmes restent lisibles. La notion de *contexte* permet d'avoir des noms abrégés. Dès que vous commencez à travailler dans un **NoteBook**, les symboles que vous définissez sont dans le contexte **Global**.

```
x = 1;
```

```
Context[x]
```

La variable **x** a désormais deux noms : un nom abrégé, **x**, et un nom complet qui est **Global`x**.

```
{x, Global`x}
```

Le caractère ` est l'accent grave, correspondant au code ASCII 96; si on ne le trouve pas sur le clavier, on peut taper

```
FromCharacterCode[96]
```

et faire du copier/coller!

Toutes les fonctions prédéfinies de *Mathematica* appartiennent au contexte **System**. La fonction **Context** n'est pas **Listable**, d'où la commande ci-dessous :

```
Attributes[Context]
```

```
Context /@ {Plot, BesselJ, Sin}
```

Il n'est pas possible de redéfinir par inadvertance une variable du système.

```
Context[Pi]
```

```
Pi = 4
```

Dans un **NoteBook** existent deux contextes, **System** pour les fonctions prédéfinies et **Global** pour les fonctions définies par l'utilisateur. Règle d'or pour éviter les conflits : Toujours commencer ses propres fonctions par une minuscule!

L'action de la fonction **Remove** est de supprimer toute définition propre à l'utilisateur du contexte **Global**, sans toucher au contexte **System**. **x** est correctement éliminé, par contre **Pi** reste défini.

```
Remove[x, Pi]
```

```
? x
```

```
Pi // N
```

II-9-3) Conflits entre contextes.

L'exemple typique où cette notion de contexte intervient pour un utilisateur peu expérimenté est lorsqu'il tente d'utiliser une fonction d'un **Package** avant d'avoir chargé celui-ci.

```
PolarPlot[Exp[θ/10], {θ, 0, 8 π}]
```

L'instruction échoue car **PolarPlot** n'est pas définie par défaut, il faut charger un **Package**.Mais le simple fait d'entrer une commande fautive fait que **PolarPlot** appartient désormais au contexte **Global**, ceci de manière totalement involontaire, et incontrôlée.

```
Context[PolarPlot]
```

De ce fait, lorsqu'on charge le Package *Mathematica* identifie le conflit, mais pas nécessairement de manière très transparente pour l'utilisateur, surtout débutant!

```
<< "BarCharts`"; << "Histograms`"; << "PieCharts`"
```

```
PolarPlot[Exp[θ/10], {θ, 0, 8 π}]
```

L'instruction échoue du fait de ce conflit de contexte! La bonne méthode est alors d'éliminer **PolarPlot** du contexte **Global** avec **Remove**; *Mathematica* utilisera alors avec succès la définition donnée par le **Package**.

```
Remove[PolarPlot]
```

```
PolarPlot[Exp[θ/10], {θ, 0, 8 π}]
```

```
Context[PolarPlot]
```

On voit, par la dernière instruction, qu'un des effets d'un **Package** est précisément d'introduire un nouveau contexte pour les définitions de fonctions.

II-9-4) Nettoyage complet du contexte Global

Le cas d'erreur précédent était assez simple, car on connaissait explicitement la commande fautive. Ce n'est pas toujours le cas, en particulier quand on a commis plusieurs erreurs dans un long document. Les causes d'erreurs les plus fréquentes, outre l'appel d'une fonction avant d'avoir chargé le **Package** qui la définit, sont l'utilisation du signe égal simple pour écrire une équation, qui introduit des définitions fausses, et l'utilisation du même symbole pour une expression et une fonction, sauf si celle-ci est définie comme une fonction pure.

```
f = 2 * x + 1
```

```
f[x_] := 1/(1 + x)
```

```
g = √(3 + x)
```

```
g = #^2 + 1 &
```

```
g[2]
```

On peut donc être amené à vouloir écraser toutes les définitions du contexte **Global**. Pour ceci, on peut utiliser une syntaxe plus puissante pour **Remove**.

```
? Remove
```

Les noms de tous les symboles définis par l'utilisateurs ont la structure (*pattern*) **Global`nom**. Ils rentrent tous dans la famille **Global`***, car * est le caractère qui, en informatique, représente toute chaîne de caractère. C'est le cas dans Unix et Dos, et dans pas mal de moteurs de recherche où la requète **Dur*** vous donnera une réponse pour **Durand**, **Durant** et **Duroc**.
Cette syntaxe classique est reconnue par *Mathematica,* comme on le voit dans cette requète (**StringMatchQ** - qui n'est pas une fonction **Listable** -réponds à la question -**Q**- de savoir si une chaîne de caractères - **String** - corresponds - **Match** - à une structure)

```
StringMatchQ[#, "Dur*"] & /@ {"Durand", "Dubois", "Duroc", "Duval", "Durant"}
```

Les instructions suivantes vous permettent de connaître tous les noms que vous avez introduits (y compris sans le vouloir, bien sûr!) dans le contexte **Global**.

```
Names["Global`*"]
```

```
? Global`*
```

```
g[3]
```

Et cette dernière instruction vous permet de toutes les effacer, sans quitter le **Kernel**

```
Remove["Global`*"]
```

```
Names["Global`*"]
```

```
g[3]
```

Physique avec *Mathematica*. Introduction avancée.

I) Régles de substitution

I-1) Syntaxe des règles de substitution

Opérer une substitution dans une expression s'effectue avec la syntaxe ci-dessous. La flèche est obtenue en tapant au clavier la série de touches - puis >

```
1 + x + 1/Sqrt[1 - a x^2] /. x -> y
```

I-2) Pourquoi sont-elles importantes?

Ces structures sont si importantes qu'elles méritent une section entière.

La première raison est qu'elles sont utilisées par de nombreuses fonctions (résolutions d'équations, d'équations différentielles, recherches de minima, ...). Leur utilisation permet à *Mathematica* de ne pas assigner de valeurs à des variables, ce qui pourrait perturber les calculs de l'utilisateur, qui ne maitriserait plus l'introduction des variables. Voyons un exemple :

```
Clear[x]
```

```
Solve[a x^2 + b x + c == 0, x]
```

```
? x
```

La dernière instruction permet de vérifier qu'aucunevaleur n'a été assignée à `x`.

La deuxième raison de l'importance des règles de substitution vient des simplifications formelles qu'elles autorisent. Nous le verrons à la question I-4).

I-3) Comment utiliser des règles de substitution fournies par le logiciel?

Elles s'utilisent comme n'importe quelle règle de substitution, en prenant bien garde que le logiciel en fournit une *liste* (sauf solution unique) dont il faut éventuellement prendre le bon élément.

```
solu = NSolve[x^4 + 1/3 x^3 - 1/4 x - 1 == 0]
```

```
solu[[3]]
```

Cette requête permet d'obtenir une liste de nombres

```
x /. solu
```

```
Sin[x /. solu]
```

```
Log[x /. solu[[4]]]
```

Remarque : si vous êtes vraiment mal à l'aise avec le logiciel, notez qu'il est toujours possible d'utiliser un résultat comme entrée d'un calcul suivant par simple copier / coller, comme sur les exemples ci-dessous :

```
Sin[0.9819830819514718`]
```

```
DSolve[{x'[t] == x[t], x[0] == 0.5}, x[t], t]
```

```
Plot[0.5` e^t, {t, -1, 1}]
```

I-4) Comment utiliser des règles de substitution pour simplifier des expressions?

Voyons deux exemples qui semblent particulièrement simple

$$\frac{\sqrt{1+\sqrt{1+\sqrt{x}}}}{\sqrt{1-\sqrt{1-\sqrt{x}}}} /. \{\sqrt{x} \to a\}$$

$$\sqrt{x} + \frac{1}{\sqrt{x}} /. \{\sqrt{x} \to a\}$$

Pourquoi la deuxième substitution a-t-elle échouée? Un cerveau humain repère instantanément la structure $\sqrt{x}$, et opère le remplacement de tête. Le logiciel ne travaille absolument pas comme cela. Il travaille sur une *représentation interne* transmise par le **FrontEnd** au **Kernel**, en *interprétant* les expressions tapées au clavier (voir **TD N°1**). On accède à cette représentation interne par la fonction **FullForm**, qui permet de comprendre d'où vient l'erreur.

```
FullForm[Sqrt[x]]
```

```
FullForm[Sqrt[x] + 1/Sqrt[x]]
```

Il y a une règle simple pour effectuer des substitutions formelles sans risque d'échec :

TOUJOURS SUBSTITUER LE COMPLIQUE AU SIMPLE

Dans ce cas, plutôt que $\sqrt{x} \to a$, qui substitue à une expression "compliquée" $\sqrt{x}$ l'expression simple "a", il faut faire $x \to a^2$, et ensuite utiliser les fonctions de simplification du logiciel. C'est une habitude à prendre, d'autant plus difficilement que, pour simplifier des expressions, notre cerveau fonctionne exactement à l'inverse : il repère rapidement des structures compliquées, et les substitue en bloc!

$$\sqrt{x} + \frac{1}{\sqrt{x}} /. \{x \to a^2\}$$

```
Simplify[%, a > 0]
```

Vous noterez que *Mathematica* est très rigoureux : il sait que $\sqrt{a^2} = a$ si et seulement si a > 0!

I-5) Exercices

Exercice I-1 :. Dans l'expression suivante, remplacer $\sqrt{u^2-1}$ par A.

$$\frac{1}{1+\sqrt{u^2-1}} - \sqrt{4+\sqrt{u^2-1}} + \text{Tan}\left[\frac{1}{\sqrt{u^2-1}}\right]$$

Exercice I-2 :. Trouver la solution générale de l'équation différentielle $y' = 2 - y^2$. Déterminer la constante d'intégration lorsque y[0] = 1. Vérifier votre résultat par un calcul direct.

Exercice I-3 :. Trouver les racines du polynôme $x^6 + 5x^3 - x + 2$. Contruire une table des puissances 1 à 10 de la racine comprise dans l'intervalle [-1,0].

II) Fonctions pures. Fonctionnelles.

II-1) Comment définir une fonction pure?

Une fonction `f[t]` doit pouvoir se définir à l'aide d'une variable muette `t`. Par "muette", on signifie que le nom effectif de la variable n'a aucune importance : `Sin[t]` et `Sin[u]` définissent la même fonction.

La syntaxe de ce type de définitions est très précise. Elle utilise le signe Deux points Egal ":=", la variable muette doit apparaître dans le membre de gauche avec un caractère Underscore "_", et sans underscore dans le membre de droite

(Au point de vue informatique, l'underscore est un *filtre* qui indique à *Mathematica* quels symboles du membre de droite doivent être considérés comme les arguments de la fonction)

```
f[t_] := 1/(1 + t^2)
```

Vous noterez que cette entrée ne provoque aucune sortie. Une fois définie, cette fonction permet aussi bien le calcul de valeurs numériques que symboliques

```
f[3.]
```

```
f[3/4]
```

```
Plot[f[u], {u, 0, 5}]
```

```
f[y]
```

```
Solve[f[z] == z/2]
```

Il existe une autre notation pour les fonctions, ne nécessitant pas de nommer les arguments. C'est ce type d'expressions qui sont appelées *fonctions pures*.

Syntaxe : L'expression de la fonction doit être terminée par une Eperluette "&", et les arguments désignés par la touche Dièse, suivie du numéro de l'argument : "#1, #2, ... #n"

```
g = 1/(1 + #1^2) &
```

```
g[t]
```

```
g[3.]
```

```
g[Sqrt[2]]
```

Syntaxe : Les fonctions pures ont d'ailleurs une autre notation, moins compacte mais plus explicite

```
h = Function[t, 1/(1 + t^2)];
```

```
h[u]
```

```
h[3.]
```

II-2) Qu'est-ce qu'une fonctionnelle? La fonctionnelle `Derivative`.

Une *fonctionnelle* est une fonction dont les arguments sont eux-même des fonctions. Un des exemples les plus simple est la dérivée, notée en abrégée par un Prime ou de façon équivalente

```
Derivative[1][f][t]
```

```
f'[t]
```

```
Derivative[1][f][t]
```

On voit mieux que **Derivative** est une fonctionnelle si on l'applique directement à la fonction, sans spécifier son argument. Notez bien que **f**, tout comme **g**, représente une fonction pure, si l'on ne spécifie pas son argument.

```
f'
```

```
g'
```

II-3) La fonctionnelle Map.

Nous avons déjà vu que certaines fonctions s'appliquent directement à des arguments sous forme de liste. Elles ont comme *attribut* **Listable,**

```
Attributes[Sin]
```

(Les attributs de **Sin** sont évidents à comprendre. Toutes les fonctions *Mathematica* ont l'attribut "**Protected**", qui évite que l'utilisateur redéfinisse une fonction existante. On peut néanmoins le faire, avec la commande **Unprotect**. Son utilisation est bien entendu radicalement déconseillée! - et ce, même si une redéfinition ne prend effet que pour une session de travail donnée-)

Certaines fonctions n'ont pas cette propriété, comme cette fonction graphique (qui, pour être utilisée, necessite d'employer les fonctions **Show** (affichage d'une sortie graphique) et **Graphics** (création d'un objet graphique 2D))

```
Attributes[Circle]
```

```
Show[Graphics[Circle[{0, 0}, 1]], AspectRatio → 1]
```

Les fonctions définies sous forme de fonction pure ne sont pas "listables". La fonctionnelle **Map** (aussi écrite **/@**), permet justement d'appliquer n'importe quelle fonction à une liste d'arguments. On voit ci-dessous que la première requête échoue, à la différence des suivantes :

```
√(1 + #1^2) & {1, a, 0.5}
```

```
Map[√(1 + #1^2) &, {1, a, 0.5}]
```

```
√(1 + #1^2) & /@ {1, a, 0.5}
```

Comme application, un très joli exemple (que j'emprunte au livre de J. Zizi, ***Mathematica* pour classes préparatoires**) permettant de comprendre comment fonctionne la fonction **Plot**.
Celle-ci est assez raffinée, car elle comporte des algorithme de recherche automatique de singularités. Plus de points sont calculés lorsque la pente de la courbe évolue rapidement. Regardons un exemple

```
f[x_] := Abs[x - 1] + Abs[(x - 1)/x]
```

```
g = Plot[f[x], {x, -10, 10}]
```

Pouvez-vous comprendre la construction de l'instruction permettant de visualiser les points effectivement calculés?

```
Short[FullForm[g], 10]
```

```
Show[g /. Line[pts_] :→ Point /@ pts]
```

Par ordre de difficulté croissante, il a fallu résoudre trois problèmes.
(1) Comprendre, à l'aide de **FullForm**, que la fonction **Plot** représente le graphe de la fonction à l'aide de la primitive graphique **Line**. Celle-ci, dont l'argument doit être une liste ordonnée de points (à 2 ou 3 coordonnées, selon que l'on utilise **Graphics** ou **Graphics3D**), trace une ligne joignant successivement ces points.
(2) Savoir que pour les visualiser, il faut utiliser la fonction **Point** qui trace un point. L'argument de cette fonction est une liste de deux ou trois nombres seulement, suivant que le point est dans un espace à deux ou trois dimentions. C'est une fonction qui n'est pas **Listable**, il faut donc utiliser **Map** ou **/@** pour l'appliquer à chacun des éléments de l'argument de **Line**.
(3) Mais ce n'est pas tout! Il faut que dans la sustitution, *Mathematica* identifie la fonction **Line**, mais aussi son argument **pts**, à l'aide du filtre **pts_**. D'où l'utilisation d'une règle de subsitution plus subtile que d'habitude, codé avec **:>** . Cette règle identifie toute occurence de la fonction **Line**, puis l'argument correspondant, et enfin remplace **Line[pts]** par **Map[Point,pts]** (ici codé **Points/@pts**)

II-4) Fonctions pures comme solutions d'équations différentielles

Il est possible d'obtenir les solutions de **DSolve** sous la forme de fonctions pures. Si vous comparez soigneusement la syntaxe de ces deux exemples, vous verrez qu'il faut prendre comme inconnue la fonction **y**, sans spécifier la variable.

```
sol1 = DSolve[y''[t] + ω^2 y[t] == 0, y[t], t]
```

```
sol2 = DSolve[y''[t] + ω^2 y[t] == 0, y, t]
```

L'intérêt d'avoir la solution sous forme de fonctions pures est que les règles de substitutions deviennent utilisables dans des fonctionnelles, qui s'appliquent a la fonction pure. En effet

```
y'[t] /. sol1
```

```
y'[t] /. sol2
```

```
y''[t] + ω^2 y[t] == 0 /. sol1 // Simplify
```

```
y''[t] + ω^2 y[t] == 0 /. sol2 // Simplify
```

II-5) Fonctions logiques

Mathematica peut travailler sur des variables logiques **True** (vrai) et **False** (faux)

```
And[True, False]
```

```
False
```

```
Or[True, False]
```

```
True
```

Il existe des fonctions à valeur logique, dont la principale utilisation est comme fonction pures, dans des fonctionnelles. Nous allons étudier en détail la fonctionnelle **Select[]**, qui permet de sélectionner au sein d'une liste les éléments vérifiant un critère donné. Il existe toute une gamme de fonctions logiques, de syntaxe générale **FonctionQ**, réalisant des tests. **IntegerQ**, par exemple, teste si un nombre donné est un entier. Elle n'est pas **Listable**, nous utilisons donc la fonction **Map**

```
IntegerQ /@ {3, 3., π, √2, √9, 20/4, 20/3}
```

```
{True, False, False, False, True, True, False}
```

Elle peut être utilisée dans **Select** pour ne sélectionner que les arguments entiers de la liste ci-dessus

```
Select[{3, 3., π, √2, √9, 20/4, 20/3}, IntegerQ]
```

```
{3, 3, 5}
```

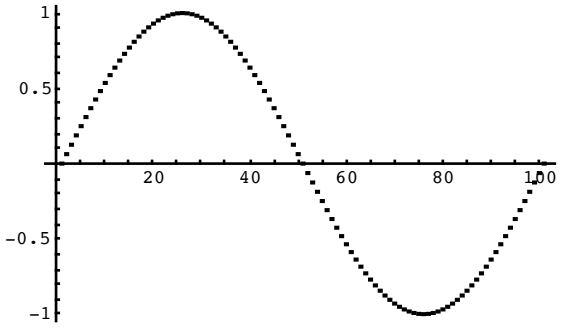
La requète suivante permet de connaitre toutes les fonctions de ce type

```
? *Q
```

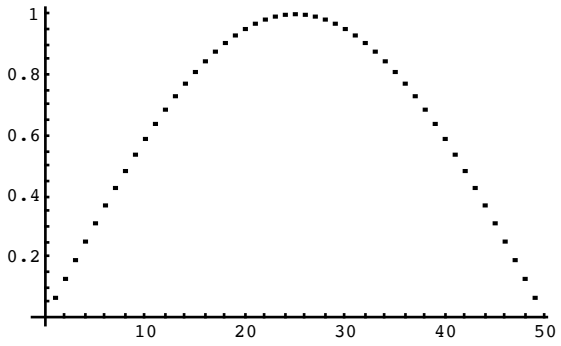
Dans **Select**, on peut aussi construire sa propre fonction logique, comme dans l'exemple ci-dessous

```
tabsin = Table[Sin[2 π i/100], {i, 0, 100}];
```

```
ListPlot[tabsin]
```




```
ListPlot[Select[tabsin, #1 > 0 &]]
```



II-6) Exercices

Exercice II-1: Ecrire sous forme de fonction pure la fonction $\frac{1}{\sqrt{1-t^2}}$. En proposer un graphe, calculer sa dérivée puis sa primitive.

Exercice II-2: Utiliser **Map** pour tracer 4 cercles concentriques, puis 4 cercles de même rayon centrés aux 4 coins d'un carré. (il faut construire une fonction pure à partir de la primitive **Circle**)

Exercice II-3:. Utiliser **FullForm** pour expliquer la raison des échecs des substitutions à la question II**-4)**

Exercice II-4:. Résoudre l'équation différentielle $x^2\,y'' + x\,y' + \left(2 - x^2\right) y = 0$, puis vérifier la solution obtenue par substitution.

Exercice II-5:. Construire une liste des entiers de 1 à 10, puis sélectionner parmi eux ceux qui sont premiers. Sélectionner ensuite ceux qui ne sont pas premiers. (Il existe une fonction qui vérifie

qu'un entier est premier. Je vous laisse la trouver!)

II-A) Appendice : Fonctions pures en notation postfix

Comme vous l'avez déjà fait, il est possible d'utiliser des fonctions *Mathematica* avec une syntaxe un peu inhabituelle, dite *postfix* (en Anglais, difficilement traduisible). Ainsi, les deux écritures suivantes sont équivalentes

```
N[Sqrt[3]/2]
```

```
Sqrt[3]/2 // N
```

Cette écriture est très commode pour deux raisons. D'abord, quand on souhaite appliquer une fonction directement après le résultat d'une autre (c'est typiquement le cas avec **`Simplify`**), il est beaucoup plus facile d'utiliser la syntaxe postfix qui permet de ne pas se tromper dans les crochets. Ensuite, le résultat est souvent bien plus lisible que des fonctions imbriquées.

```
Det[{{α, α^2, α^3}, {β, β^2, β^3}, {γ, γ^2, γ^3}}]
```

```
Simplify[%]
```

On peut préférer simplement

```
Det[{{α, α^2, α^3}, {β, β^2, β^3}, {γ, γ^2, γ^3}}] // Simplify
```

Ceci étant, comment faire s'il faut indiquer, toujours dans **`Simplify`**, une option? Par exemple dans ce calcul, où *Mathematica* ne sait opérer une simplification que si on lui indique le signe de **`x`**?

```
(Sqrt[x^4] - Sqrt[x^2])/(Sqrt[x^4] + Sqrt[x^2]) // Simplify
```

Il suffit d'introduire une *fonction pure*, correctement construite. Ainsi, les trois syntaxes ci-dessous sont équivalentes, les deux dernières utilisant les deux syntaxes possibles pour les fonctions pures

```
Simplify[(Sqrt[x^4] - Sqrt[x^2])/(Sqrt[x^4] + Sqrt[x^2]), x > 0]
```

```
(Sqrt[x^4] - Sqrt[x^2])/(Sqrt[x^4] + Sqrt[x^2]) // Simplify[#, x > 0] &
```

```
(Sqrt[x^4] - Sqrt[x^2])/(Sqrt[x^4] + Sqrt[x^2]) // Function[f, Simplify[f, x > 0]]
```

Si vous appréciez cette syntaxe, c'est une excellente utilisation des fonctions pures.

III) La fonction `Evaluate`

Une des utilisations standard de *Mathematica* est le travail sur des fonctions; il est souvent très pratique d'introduire des fonctions de plusieurs variables, ce qui permet de les représenter pour différentes valeurs d'un paramètre. Comme exemple, nous allons étudier la fonction gaussienne

$$\text{gauss}(x) = \frac{1}{2\sqrt{\pi a}} e^{-\frac{x^2}{4a}}$$

```
gauss[x_, a_] := 1/(2 Sqrt[π a]) e^(-x^2/(4 a))
```

```
Plot[gauss[x, 1], {x, -8, 8}, Frame → True]
```

III-1) Comment tracer cette fonction pour différentes valeurs du paramètre a?

Essayons ce qui vient naturellement à l'esprit; vérifiez que cette commande n'est pas bonne.

```
Plot[Table[gauss[x, 4 i/5], {i, 1, 5}], {x, -8, 8}, Frame → True]
```

Pourquoi? Cela vient du fait que **Plot** commence par évaluer les valeurs de la variable (ici **x**), puis seulement ensuite les valeurs prises par le premier argument pour ces valeurs de la variable. Si ce premier argument est une fonction, ou une liste de fonctions données explicitement, tout marche bien. Ici, la table n'est pas explicitée, et le tracé impossible. Il faut forcer l'évaluation du premier argument avec la commande **Evaluate**

```
Plot[Evaluate[Table[gauss[x, 4 i/5], {i, 1, 5}]], {x, -8, 8}, Frame → True]
```

(dessin plus joli, avec les courbes en couleur)

```
Plot[Evaluate[Table[gauss[x, 4 i/5], {i, 1, 5}]],
  {x, -8, 8}, Frame → True, PlotStyle → Table[Hue[i/5], {i, 1, 5}]]
```

III-2) Comment définir comme fonction la solution exacte d'une équation différentielle?

```
solu = DSolve[{x''[t] + ω^2 x[t] == 0, x[0] == 1, x'[0] == 0}, x[t], t]
```

Premier essai intuitif, mais qui ne donne pas le résultat.

```
f[t_, ω_] := x[t] /. solu[[1]]
```

```
f[2., 1.]
```

```
Trace[f[2., 1.]]
```

On comprend le problème en utilisant **Trace**. En effet, la variable **t** est instantanément remplacée par sa valeur numérique, et la substitution ne peut plus se faire.

Le problème vient de ce que, lorsqu'on utilise une définition avec "**:=**", l'évaluation du membre de droite n'est pas faite *lors de la définition de la fonction*, mais seulement lorsqu'une requête demande explicitement un calcul. Il est donc possible d'éviter cet inconvénient en utilisant la fonction **Evaluate** dès la définition initiale.

```
f[t_, ω_] := Evaluate[x[t] /. solu[[1]]]
```

```
f[2., 1.]
```

```
Trace[f[2., 1.]]
```

Une autre solution, d'ailleurs plus simple, consiste à définir la fonction sans utiliser l'évalutation différée, c'est-à-dire en utilisant le simple signe "=" plutôt que ":="

```
g[t_, ω_] = x[t] /. solu[[1]]
```

```
g[2., 1.]
```

```
Trace[g[2., 1.]]
```

III-3) Comment définir comme fonction la dérivée d'une fonction?

De façon peut-être surprenante, le problème se pose exactement dans les mêmes termes lorsqu'on veut définir comme fonction une dérivée. *Mathematica* sait parfaitement calculer des dérivées. Reprenons notre gaussienne, et proposons nous d'en tracer la dérivée.

```
dpgauss[x_, a_] := ∂x gauss[x, a]
```

```
dpgauss[2., 1]
```

Cette définition échoue, car les variables sont remplacées par leurs valeurs numériques, et ce n'est qu'ensuite que *Mathematica* cherche à effectuer la dérivation. Le problème se régle comme précédemment, en forçant l'évaluation. Celle-ci est alors faite dès la définition de la fonction, et c'est dans le résultat que les valeurs viennent remplacer les variables.

```
dpgauss[x_, a_] := Evaluate[∂x gauss[x, a]]
```

```
dpgauss[2., 1]
```

```
Plot[dpgauss[x, 1], {x, -8, 8}, Frame → True]
```

Cette méthode n'est pas la seule. En effet, l'opération de dérivation (ou de dérivation partielle) est une *fonctionnelle*, et si on écrit explicitement la forme fonctionnelle on ne rencontre pas de problème. On aurait ainsi pu écrire

```
dpgaussbis[x_, a_] := Derivative[1, 0][gauss][x, a]
```

```
dpgaussbis[2., 1]
```

```
Plot[dpgaussbis[x, 1], {x, -8, 8}, Frame → True]
```

Il est possible d'utiliser une syntaxe très condensée, équivalente aux notations mathématiques standard, *pour une fonction d'une seule variable*. Dans ce cas en effet, la notation `f'` (resp. `f''`, `f'''`, etc...) est synonyme de `Derivative[1][f]` (resp. `Derivative[2][f]`, `Derivative[3][f]`). Les dérivations ainsi notées sont effectuées sur les fonctions pures, et leur argument n'est évalué qu'après. Les définitions ci-dessous fonctionnent donc parfaitement.
Syntaxe : Attention, `f''` est écrit avec les touches f -> Prime -> Prime (et NON f -> Double prime)

```
g[x_] := gauss[x, 1]
```

```
dg[x_] := g'[x]
```

```
ddg[x_] := g''[x]
```

```
Plot[{g[u], dg[u], ddg[u]}, {u, -8, 8}, PlotStyle →
  {RGBColor[1, 0, 0], RGBColor[0, 1, 0], RGBColor[0, 0, 1]}, PlotRange → All]
```

III-4) Comment représenter graphiquement la solution numérique d'une équation différentielle?

Là encore, dans `Plot`, il faut forcer l'évaluation de la solution numérique. C'est du moins ce qui est indiqué dans le manuel, mais ne semble pas se produire...

```
solu = NDSolve[{y'[x] == Sin[y[x]], y[0] == 1}, y, {x, 0, 4}]
```

```
Plot[y[t] /.solu, {t, 0, 4}]
```

III-5) Exercices

Exercice III-1 : Tracer les solutions de l'équation différentielle $x' - x + a\,x^2 = 0$, x(0) = 1 pour plusieurs valeurs du paramètre a (cette équation a une solution analytique) entre 0 et 1.

Exercice III-2 : Définir une fonction qui soit la primitive de $\frac{1}{1+t^2}$. En donner une représentation graphique.

Utilisation des fonctions graphiques

I. Lignes de champ et équipotentielles d'un dipôle électrique.

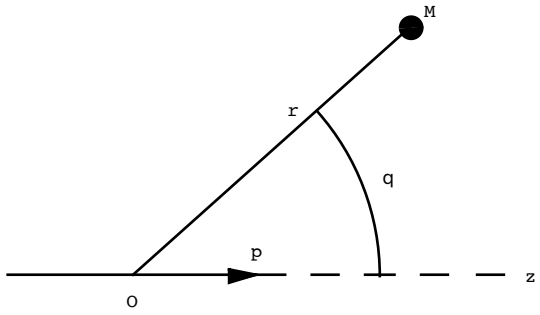
Le potentiel au point M exercé par un dipôle *p*, orienté selon l'axe Oz, est

$$V(r, \theta) = \frac{1}{4 \pi \epsilon_0} \frac{p \cos \theta}{r^2}.$$

La figure suivante définit r et θ :

```
<< Graphics`Arrow`

ϕ = π/4 Random[];
Show[Graphics[{Arrow[{-0.5, 0}, {0.5, 0}],
    Disk[{1.5 Cos[ϕ], 1.5 Sin[ϕ]}, 0.05], Line[{{0, 0}, {1.5 Cos[ϕ], 1.5 Sin[ϕ]}}],
    {Dashing[{0.05, 0.05}], Line[{{0.5, 0}, {1.5, 0}}]}, Circle[{0, 0}, 1, {0, ϕ}],
    Text["p", {0.5, 0.1}], Text["r", {Cos[ϕ] - 0.1, Sin[ϕ]}],
    Text[Text["θ", 1.1 {Cos[ϕ/2], Sin[ϕ/2]}], Text["M", 1.6 {Cos[ϕ], Sin[ϕ]}],
    Text["O", {0, -0.1}], Text["z", {1.6, 0}]}], AspectRatio → Automatic];
```



I-1) En déduire les composantes du champ électrique. Deux remarques :
(1) éviter absolument la notation E qui une notation interne à *Mathematica*, la base de l'exponentielle.
(2) réfléchissez, faites attention à n'introduire aucune constante superflue!

I-2) Trouver la forme des lignes de champ, dont l'équation est (pourquoi ?) $\frac{dr}{\Psi_r} = \frac{r d\theta}{\Psi_\theta}$

I-3) Tracer sur le même graphe le dipôle, quelques lignes de champ et quelques équipotentielles (ces dernières en pointillé). L'instruction ci-dessous permet de représenter le dipôle lui-même comme une flèche.

```
<< Graphics`Arrow`
```

Plusieurs possibilités. D'abord avec la fonction **`ParametricPlot`**, ensuite en chargeant un package graphique, et en utilisant une fonction **`PolarPlot`**

```
<< Graphics`Graphics`
```

I-4) Vérifier qu'équipotentielles et lignes de champs sont orthogonales. (la solution est naturellement donnée sous la forme d'une courbe en coordonnées polaires; il est néanmoins assez simple de la transformer en une courbe paramétrée par l'angle θ)

II. Diffraction d'une onde plane normale par un trou circulaire de rayon *a*

Dans l'approximation de Fraunhoffer, dite aussi "diffraction à l'infini", l'amplitude de la vibration lumineuse [scalaire, à cette approximation; l'amplitude incidente dans le plan diffractant Π est *S(x, y)*] transmise par un écran diffractant Π [supposé plan, lié à un repère *(O,x,y)*] de facteur de transmission *t(x,y),* dans la direction (*α*,*β*), est donnée à une constante multiplicative près par

$$S(\alpha,\beta) \propto \iint_{\Pi} t(x,y)\, S(x,y) \exp\left[-I \frac{2\pi}{\lambda}(\alpha x + \beta y)\right] dx\, dy$$

Si (X,Y) sont les coordonnées d'un point de l'écran d'observation, L la distance de cet écran à l'objet diffractant, on a bien sûr $\alpha = \frac{X}{L}$, $\beta = \frac{Y}{L}$.

II-1) En quel système de coordonnées faut-il se placer ? Pourquoi ne perd-on pas en généralité en posant $\beta = 0$? Exprimer l'argument de l'exponentielle dans des variables plus appropriées.
II-2) Exprimer l'intégrale ci-dessus et la calculer lorsque l'onde incidente est une onde plane sous incidence normale (soit *S(x,y) = 1*). Attention : il faut forcer *Mathematica* à commencer par intégrer sur la variable angulaire.

Simplifier le résultat en introduisant la variable $\frac{\pi a \alpha}{\lambda} \equiv u$

II-3) Calculer le rayon angulaire α de la tache centrale de diffraction en fonction de $\frac{\lambda}{2a}$. Ceci nécessite la recherche numérique de racines d'une équation. Il est vivement conseillé de commencer par une représentation graphique.
II-4) Représenter comme **`DensityPlot[]`** la figure d'interférence *en intensité*. (on pourra jouer sur les options pour améliorer la figure)
II-5) Calculer le rapport entre l'énergie lumineuse contenue dans le premier et le deuxième anneau, et celle contenue dans la tache centrale (commencer par la recherche numérique des racines nécessaires)

III. Diffraction d'une onde plane normale par un demi-plan à l'approximation de Fresnel

Nous allons nous intéresser à la géométrie la plus simple que l'on puisse envisager : la diffraction par le demi-plan *y > 0* d'une onde plane sous incidence normale. Nous prendrons *Ox* dans le plan, *Oz* perpendiculaire au plan. Le problème ne dépend clairement pas de *x*, et l'amplitude du champ à une distance *z*, au point *Y*, est donnée par

$$U(Y) = \int_0^{\infty} \mathrm{Exp}\left[I \frac{\pi}{\lambda z}(y - Y)^2\right] dy$$

En posant $v - V \equiv \sqrt{\frac{2}{\lambda z}}\,(y - Y)$, on obtient *U(V) = CF(∞) + CF(V) + I (SF(∞) + SF(V))* où on définit

$$CF(V) = \int_0^V \mathrm{Cos}\left[\frac{\pi}{2}u^2\right] du\,,\quad SF(V) = \int_0^V \mathrm{Sin}\left[\frac{\pi}{2}u^2\right] du$$

Ces fonctions spéciales sont appelées *intégrales de Fresnel. Mathematica* les connait :

$$\left\{\int_0^V \mathbf{Cos}\left[\frac{\pi}{2}\mathbf{u}^2\right] \mathbf{du},\ \int_0^V \mathbf{Sin}\left[\frac{\pi}{2}\mathbf{u}^2\right] \mathbf{du}\right\}$$

```
{FresnelC[V], FresnelS[V]}
```

III-1) Calculer les valeurs en +∞, ainsi qu'un développement limité à l'origine (de deux façons

différentes)

III-2) Représenter la courbe paramétrique **(FresnelC[V],FresnelS[V])**. En déduire une interprétation géométrique de la formule de diffraction de Fresnel par un demi-plan.

III-3) Représenter l'intensité lumineuse $|U(v)|^2$. Quelles sont les différences avec l'optique géométrique ?

IV. Vibrations d'une membrane circulaire

Le système physique est une membrane, parfaitement élastique, uniformément tendue avec une tension T par unité de longueur. On note σ la masse surfacique de la membrane. Enfin, on suppose qu'elle est attachée à un contour circulaire rigide.
On s'interesse aux déformations transverses, en régime linéaire, de la membrane. Définissons un repère *(O,x,y,z)* tel que *(O,x,y)* soit confondu avec le plan de la membrane au repos, et *(Oz)* perpendiculaire. Le point *(x,y)* se trouve à la position $z = \Psi(x,y,t)$, et la dynamique des déformations transverses est décrite par l'équation

$$\Delta\Psi - \frac{1}{c^2}\partial_{t,t}\Psi = 0,$$

où $c \equiv \sqrt{\frac{T}{\sigma}}$ est la vitesse des ondes transverses, et Δ représente le Laplacien en deux dimensions.
Il est naturel, vue la géométrie des conditions aux limites, de se placer en coordonnées polaires. L'équation est alors

$$\partial_{r,r}\Psi(r,\theta,t) + \frac{1}{r}\partial_r\Psi(r,\theta,t) + \frac{1}{r^2}\partial_{\theta,\theta}\Psi(r,\theta,t) - \frac{1}{c^2}\partial_{t,t}\Psi(r,\theta,t) = 0$$

La méthode de résolution consiste à chercher une solution sous la forme $\Psi(r,\theta,t) = R(r)\,\Theta(\theta)\,e^{-i\omega t}$.
Si on pose $\frac{\omega^2}{c^2} \equiv k^2$, on aboutit à

$$\frac{r^2}{R(r)}\left(R''(r) + \frac{1}{r}R'(r)\right) + k^2 r^2 = -\frac{1}{\Theta(\theta)}\Theta''(\theta) = \text{Cste}$$

r et θ étant deux variables indépendantes, les deux fonctions prennent des valeurs indépendantes et ne peuvent être toujours identiques que si elles sont égales à une même constante.

IV-1) En étudiant l'équation pour la fonction $\Theta(\theta)$ de la variable angulaire θ, exprimer pourquoi la constante doit être positive, égale à m^2 où m est un entier naturel.

IV-2) En déduire l'équation vérifiée par R(r). Cette équation est l'équation de Bessel. La résoudre pour m = 0. Les solutions doivent être valables dans tout le domaine $0 \le r \le a$, si a est le rayon de l'anneau sur lequel la membrane est tendue. Nous allons donc nous intéresser au comportement des deux fonctions de Bessel près de l'origine $r = 0$. Tracer $J_0(r)$ et $Y_0(r)$ (que *Mathematica* note **BesselJ[0,r]** et **BesselY[0,r]**) Que suggère ce dessin? Vérifier votre hypothèse en faisant un développement en série au voisinage de r = 0. En déduire la valeur d'une des constantes d'intégration.

IV-3) Etudier ensuite les cas m = 1 et m = 2.

IV-4) Quelle est la condition aux limites en $r = a$? En déduire que seules certaines valeurs de $k\,a$ sont possibles. Ces valeurs discrètes définissent les *modes propres* de la membrane, qui sont donc indexés par deux nombres entiers, m définit précédemment et n le numéro de la racine $k\,a$. Calculer numériquement les valeurs de ka pour les modes (0,1), (0,2), (1,1) et (2,1).

IV-5) Représenter graphiquement, en donnant à chaque figure le titre (m,n), les modes (0,1), (0,2), (1,1) et (2,1). Le dessin le plus clair utilise la fonction **DensityPlot[]**; il est plus joli aussi d'utiliser la fonction **GraphicsArray[].**
Il faut réfléchir un peu à la façon dont fonctionne **DensityPlot[]**. Cette fonction trace au point de coordonnées cartésiennes (x,y) un pixel d'un niveau de gris fixé par la valeur f(x,y) de la fonction,

dans un domaine rectangulaire. Dans le cas présent, la fonction est donnée pour des variables (r,θ); pensez à convertir les coordonnées! Pour r c'est facile, pensez à définir proprement une fonction θ[x,y]. Par ailleurs, la fonction est à tracer dans un domaine circulaire; on peut utiliser une fonction **`If[]`**, ou la fonction logique **`Boole[]`**, disponible à partir de la version 5 de *Mathematica*..

IV-6) Construire une table fournissant les 4 premiers modes pour *m* = *0, 1* et *2*. La meilleure description consiste à fournir les *fréquences propres*, par rapport à celle du mode fondamental (Rq : il est utile de commencer par une représentation graphique des fonctions, et d'en déduire une façon automatique de recherche des modes)

Oscillateur harmonique libre et forcé. Filtres analogiques.

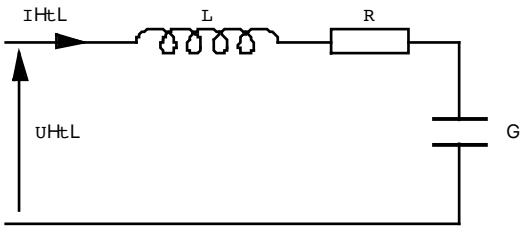
I-1) Oscillations libres d'un circuit RLC

I-1-a) Obtenir le schéma ci-dessous (facultatif).

NB 1 : La notation Γ pour la capacité n'est pas choisie au hasard, **C** est une notation interne à *Mathematica,* qu'il faut donc éviter. (c'est le nom des constantes d'intégration)

```
<< Graphics`Arrow`

g1 = ParametricPlot[{ϕ - 0.5 Sin[2 π ϕ], 1/(2 π) - 0.5 Cos[2 π ϕ]},
    {ϕ, 1/4, 19/4}, Axes → None, DisplayFunction → Identity];
X0 = ϕ - 0.5 Sin[2 π ϕ] /. {ϕ → 0.25};
Y0 = 1/(2 π) - 0.5 Cos[2 π ϕ] /. {ϕ → 0.25};
X1 = ϕ - 0.5 Sin[2 π ϕ] /. {ϕ → 4.75};
Y1 = 1/(2 π) - 0.5 Cos[2 π ϕ] /. {ϕ → 4.75};
Show[{g1, Graphics[{Arrow[{X0 - 5, Y0}, {X0 - 2, Y0}],
      Line[{{X0 - 2, Y0}, {X0, Y0}}], Line[{{X1, Y1}, {X1 + 2, Y1}}],
      Line[{{X1 + 5, Y1}, {X1 + 7, Y1}}], Line[{{X1 + 2, Y1 - 0.5},
        {X1 + 2, Y1 + 0.5}, {X1 + 5, Y1 + 0.5}, {X1 + 5, Y1 - 0.5}, {X1 + 2, Y1 - 0.5}}],
      Line[{{X1 + 7, Y1}, {X1 + 7, Y1 - 3}}], {AbsoluteThickness[2],
       Line[{{X1 + 6, Y1 - 3}, {X1 + 8, Y1 - 3}}], Line[{{X1 + 6, Y1 - 4}, {X1 + 8, Y1 - 4}}]},
      Line[{{X1 + 7, Y1 - 4}, {X1 + 7, Y1 - 7}}], Line[{{X0 - 5, Y1 - 7}, {X1 + 7, Y1 - 7}}],
      Arrow[{X0 - 4.5, Y1 - 6.5}, {X0 - 4.5, Y0 - 0.5}], Text["I(t)", {X0 - 3.5, Y0 + 1}],
      Text["L", {(X0 + X1) / 2, Y0 + 1}], Text["R", {X1 + 3.5, Y0 + 1}],
      Text["Γ", {X1 + 9, Y0 - 3.5}], Text["U(t)", {X0 - 3, Y0 - 3.5}]}]},
  DisplayFunction → $DisplayFunction, AspectRatio → Automatic,
  PlotRange → All];
```



I-1-b) Adimensionalisation de l'équation

Soit q(t) la charge électrique du condensateur. L'équation différentielle du circuit, en régime libre (U(t) = 0) est

$$\partial_{t,t} q + \frac{R}{L} \partial_t q + \frac{1}{L\Gamma} q = 0$$

Il est interessant de faire apparaître une pulsation $\omega^2 \equiv \frac{1}{L\Gamma}$ et de poser $\eta \equiv \frac{R}{2}\sqrt{\frac{\Gamma}{L}}$. Quelle est sa dimension physique ? Quelle est sa signification physique ? Attention à faire les substitutions dans le bon ordre!

Fonctions à utiliser : substitutions de variables, et **`Simplify`**

I-1-c) Résoudre cette équation avec comme condition initiale q(0) = 1, q'(0) = 0. Simplifier le résultat en introduisant un temps adimensionné $\tau \equiv \omega$ t.

Physiquement, cette situation correspond à un condensateur initialement chargé.

Fonctions à utiliser : **`DSolve`**

I-1-d) Définir une fonction charge[t,η] qui représente la charge aux bornes du condensateur, et la tracer pour $\eta < 1$, $\eta = 1$ et $\eta > 1$ (trois couleurs différentes pour chacune des courbes). Comment traiter le cas $\eta = 1$?

Fonctions à utiliser : **`DSolve`**, **`Plot`** en spécifiant **`PlotStyle`** avec **`RGBColor`**

I-1-e) Montrer qu'en régime pseudopériodique le signal est encadré par deux fonctions exponentielles décroissantes.

La forme dont on dispose pour la solution est fort peu pratique. Le but du calcul est de mettre la solution sous la forme

$$A \, \text{Cos}\left[\sqrt{1-\eta^2}\ \tau + \phi\right] e^{-\eta \tau}$$

où A est une constante.

La simplification est difficile à obtenir, car la forme finale souhaitée comporte à la fois une fonction trigonométrique et une fonction exponentielle. Une bonne méthode est de multiplier la fonction à simplifier par $e^{\eta t}$. Ensuite, un bon moyen d'utiliser le fait que $\eta < 1$ est de poser $\eta = \text{Sin}[\phi]$, avec $0 < \phi < \frac{\pi}{2}$ puisque $\eta > 0$. Repasser à une expression dépendant de η à la fin du calcul.

Fonctions à utiliser : substitutions de variables, **`Simplify, ExpToTrig, TrigToExp`**

I-1-f) Représenter le signal et ses enveloppes sur un graphe. Calculer la position de quelques maxima de la fonction. Pourquoi appelle-t-on ce régime pseudopériodique ? Quelle est la valeur de la pseudo période ? Est-ce conforme à l'expression trouvée pour la solution ? Les maxima sont-ils confondus avec les points de contact entre la courbe et son enveloppe positive ? (ceci pour une valeur quelconque de η, $0 < \eta < 1$, c'est-à-dire une valeur quelconque de ϕ)

Le but de l'exercice est de faire la recherche de maxima automatiquement, et d'en obtenir une liste (que l'on notera **`tabmax`**).

Fonctions à utiliser : **`Table, FindMinimum`**

I-2) Oscillations forcées

I-2-a) On s'interesse à la résonance de charge. La tension aux bornes du condensateur est reliée à la tension U(t), supposée sinusoïdale de pulsation Ω par $\hat{U_C} = H(\Omega)\hat{U}$. On a

$$H(\Omega) = \frac{1}{I\,\Omega\,\Gamma\left(I\,L\,\Omega + R + \frac{1}{I\,\Gamma\,\Omega}\right)}$$

Calculer module et argument de $H(\Omega)$, en fonction des paramètres sans dimension précédents η et ω, ainsi que de $v \equiv \frac{\Omega}{\omega}$. En donner une représentation graphique pour η = 0.1, 0.25 et 1.

NB : Pour les calculs formels avec des complexes, il est vivement recommandé de charger le package **Algebra`ReIm`**, qui permet en particulier de déclarer qu'un symbole est réel. Les instructions sont tapées ci-dessous, bien retenir leur syntaxe. En outre, *Mathematica* sait, pour cette formule déjà compliquée, calculer partie réelle et imaginaire, mais pas directement module et argument.
Fonctions à utiliser : **ComplexExpand** (utiliser l'option **TargetFunction**), **Plot** (tracer les courbes en couleur), **GraphicsArray, Which** ou **If** (ces dernières pour représenter de façon convenable la phase)
Facultatif : vous pouvez jouer avec l'option **DisplayFunction**, pour éviter des sorties graphiques inutiles. Il est aussi plus agréable, pour la phase, d'avoir les ordonnées en sous-multiples de π.

```
<< Algebra`ReIm`
```

```
Γ /: Im[Γ] = 0; Ω /: Im[Ω] = 0; R /: Im[R] = 0; L /: Im[L] = 0;
```

I-2-b) Calculer la pulsation à la résonance, ainsi que la bande passante du filtre RLC. Jusqu'à quelle valeur de η peut-on parler de résonance? Lorsque la dissipation est faible (en un sens qu'on précisera), calculer la pulsation à la résonance, l'amplitude de la résonance et la bande passante, au premier ordre.

Fonctions à utiliser : **Solve, Series**

II - Chaînes de quadripôles

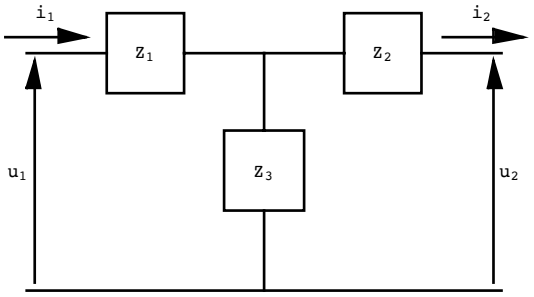
II-1) Obtenir le schéma ci-dessous (facultatif) .

Nous allons étudier dans cette partie des quadripôles passifs, selon le schéma suivant dit "forme en T".

Le but de cette question est d'utiliser des fonctions graphiques pour réaliser un dessin. Les flèches nécessitent le chargement d'un package graphique.
Fonctions à utiliser : **Graphics, Line, Text, Arrow**

```
<< Graphics`Arrow`
```

```
Show[Graphics[{Line[{{0, 0}, {2, 0}, {2, 1}, {4, 1}, {4, -1}, {2, -1}, {2, 0}}],
    Text["Z₂", {3, 0}], Line[{{4, 0}, {6, 0}}],
    Line[{{0, 0}, {-2, 0}, {-2, 1}, {-4, 1}, {-4, -1}, {-2, -1}, {-2, 0}}],
    Text["Z₁", {-3, 0}], Line[{{-4, 0}, {-6, 0}}],
    Line[{{0, 0}, {0, -2}, {1, -2}, {1, -4}, {-1, -4}, {-1, -2}, {0, -2}}],
    Text["Z₃", {0, -3}], Line[{{0, -4}, {0, -6}}], Line[{{-6, -6}, {6, -6}}],
    Arrow[{-5.8, -5.8}, {-5.8, -0.2}], Text["u₁", {-6.2, -3}],
    Arrow[{5.8, -5.8}, {5.8, -0.2}], Text["u₂", {6.2, -3}],
    Arrow[{-6.5, 0.4}, {-4.5, 0.4}], Text["i₁", {-5.5, 1}],
    Arrow[{4.5, 0.4}, {6.5, 0.4}], Text["i₂", {5.5, 1}]}],
  AspectRatio → Automatic, PlotRange → All];
```



II - 2) Obtenir , sous forme matricielle, la relation entre les variables d' entrée et de sortie

$$\begin{pmatrix} u_1 \\ i_1 \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} u_2 \\ i_2 \end{pmatrix}$$

Que vaut le déterminant de la matrice ? (C' est un résultat remarquable, qu' on peut démontrer par des considérations énergétiques pour des impédances non dissipatives, ou comme ici à partir des lois de mailles et noeuds)

Il faut écrire la matrice automatiquement, à partir des lois de Kirchoff.
Fonctions à utiliser : **`Solve, Coefficient, MatrixForm, Det`**
Remarque : l'utilisation de Solve est inhabituelle. En effet nous avons trois équations, pour cinq inconnues! Il faut donc préciser quelle variable doit être éliminée, et quelles variables doivent être exprimées en fonction des inconnues restantes.
Remarque 2 : Il y a une façon très astucieuse d'utiliser la fonction **`Coefficient`** pour avoir directement la matrice cherchée, en utilisant la fonction **`Outer`**, dont je vous donne ci-dessous un exemple d'utilisation.

```
Outer[f, {a, b}, {c, d}]
```

```
{{f[a, c], f[a, d]}, {f[b, c], f[b, d]}}
```

II-3) Impédance itérative.

On ferme le quadripôle par une impédance Z_0, de telle sorte que $Z_0 = u_2/i_2$. Montrer qu'il est possible de choisir Z_0de sorte que cette impédance soit *itérative,* c'est-à-dire qu'on ait $Z_0 = u_1/i_1$. Examiner le cas particulier $Z_1 = Z_2 = Z_3 = Z$, et en conclure qu'une seule de cas valeurs a un sens physique.

II-4) Quelques exemples de filtres. (tous les composants sont supposés parfaits)

III-4-a) Z_1et Z_2correspondent à la même inductance L, et Z_3est réalisée par une capacité 2 Γ.
Montrer que, si l'on ferme le quadripôle sur son impédance itérative, on construit un *filtre passe-bas*.

(*Mathematica* simplifie les radicaux avec difficulté; il est plus simple de travailler sur $Z_0(\omega)^2$)
Tracer $Z_0(\omega)$ où ω est la pulsation (Afin de comparer entre eux ces trois filtres, on les tracera sur le même graphe, en fonction de la même fréquence sans dimension).

L'adimensionalisation se fera, ainsi que dans les questions suivantes, en posant $\omega = \sqrt{\frac{1}{L\,\Gamma}}\ v$ pour ce qui est des fréquences. Pour l'impédance $Z_0{}^2$, l'adimensionalisation se fait ici en multipliant par $\frac{2\,\Gamma}{L}$.

III-4-b) Z_1et Z_2correspondent à la même capacité $\frac{\Gamma}{2}$, et Z_3est réalisée par une inductance L. Montrer que l' on construit un *filtre passe-haut*. Pour l'impédance $Z_0{}^2$, l'adimensionalisation se fait en multipliant par $\frac{\Gamma}{2\,L}$.

III-4-c) Montrer que l'on construit un *filtre passe-bande* en prenant $Z_1 = Z_2$constituées d'une inductance L en série avec une capacité Γ, et Z_3constituée de la même inductance L en parallèle avec la même capacité Γ. Tracer $Z_0(\omega)$. Calculer la bande passante du filtre. Pour l'impédance $Z_0{}^2$, l'adimensionalisation se fait en multipliant par $\frac{\Gamma}{L}$.

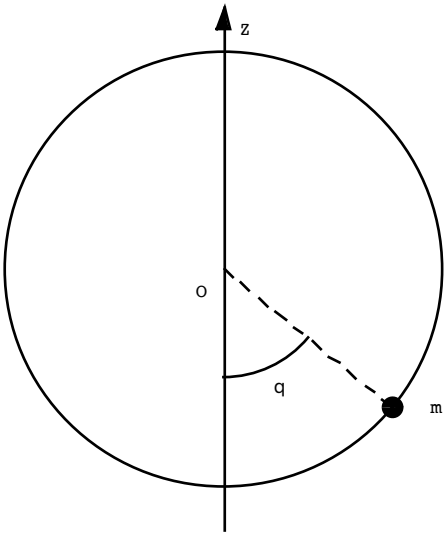
Pendule pesant tournant

A. Pendule pesant.

Le système que nous étudierons est une bille, assimilée à une masse ponctuelle *m,* coulissant sans frottements le long d'un anneau circulaire rigide de rayon *R*.
Dans toute la partie A, cet anneau est immobile dans le référentiel du Laboratoire, que nous supposerons Galiléen. On note *g* l'accélération de la pesanteur, l'axe *OZ* est supposé vertical, orienté vers le haut.
Le système peut être schématisé comme ci-dessous.

```
<< Graphics`Arrow`

θ = π/4 * Random[];
Show[Graphics[{Circle[{0, 0}, 1],
    Circle[{0, 0}, 0.5, {3 π/2, 2 π - θ}], Arrow[{0, -1.2}, {0, 1.2}],
    {Dashing[{0.025, 0.025}], Line[{{0, 0}, {Cos[θ], -Sin[θ]}}]},
    Disk[{Cos[θ], -Sin[θ]}, 0.05], Text["m", {Cos[θ] + 0.2, -Sin[θ]}],
    Text["Z", {0.1, 1.1}], Text["O", {-0.1, -0.1}],
    Text["θ", 0.6 {Cos[7 π/4 - θ/2], Sin[7 π/4 - θ/2]}]}], AspectRatio → Automatic];
```



A.I- Mise en équation

La méthode la plus rapide est de passer par les équations de Lagrange. Soient donc E_C l'énergie cinétique, E_P l'énergie potentielle, *L* le Lagrangien et enfin *H* l'Hamiltonien.

```
Ec = 1/2 m R^2 (∂t θ[t])^2; Ep = m g R (1 - Cos[θ[t]]);
```

Il est préférable d'utiliser une variable temporelle sans dimension, $\tau \equiv \sqrt{\frac{g}{R}}\, t$. Cela élimine des paramètres inutiles, et par ailleurs permet de mieux distinguer le comportement de l'oscillateur en

régime linéaire et non linéaire.

```
L = -g m R (1 - Cos[ϕ[Sqrt[g/R] t]]) + 1/2 m R^2 (∂t ϕ[Sqrt[g/R] t])^2 /. Sqrt[g/R] t → τ;
L̂ = L / (m g R) // Simplify
∂τ ∂∂τ ϕ[τ] L̂ - ∂ϕ[τ] L̂
```

L'équation du mouvement est donc

$$\phi'' = -\text{Sin}[\phi]$$

et l'unité d'énergie est m g R. Désormais, toutes les énergies seront exprimées dans cette unité. Le Hamiltonien vaut

```
Ĥ = ∂τ ϕ[τ] (∂∂τ ϕ[τ] L̂) - L̂
```

C'est une constante du mouvement puisque le Lagrangien ne dépend pas explicitement du temps. Dans ce cas simple, son interprétation physique est très claire : il s'identifie à l'énergie totale de la particule, dans le référentiel (galiléen) du Laboratoire.

A.II- Oscillations d'amplitude finie

A.II-1) Soit E l'énergie mécanique de la bille . Tracer l'énergie potentielle $\hat{U}(\theta)$, et montrer que pour $E < E_{max}$ (qu'on précisera) la bille effectue des oscillation d'amplitude A. Mettre en évidence sur le dessin l'énergie cinétique du pendule pour un angle quelconque $\phi < A$, par une droite en pointillés. Donner A en fonction de E. Quel est le mouvement de la bille lorsque $E > E_{max}$?

A.II-4) Portrait de phase de l'oscillateur. L'espace des phase est le plan (ϕ,ϕ'), et faire un "*portrait de phase*" consiste à tracer quelques trajectoires dans cet espace de phase. On tracera 4 trajectoires, dont une en régime linéaire ($\phi \ll 1$ et Sin[ϕ]~ϕ), une dans le cas $E < E_{max}$, une pour $E > E_{max}$ et enfin $E = E_{max}$. On utilisera avec profit les propriétés de symétries des trajectoires.

A.III) Période des oscillations en fonction de leur amplitude.

Lorsque l'amplitude des oscillations est infinitésimale, leur période est indépendante de l'amplitude : c'est *l'isochronisme* des petites oscillations. Pour une amplitude finie, il n'en est plus ainsi. A partir de l'expression du Hamiltonien, la période peut se mettre sous la forme

$$T = 4 \int_0^A \frac{1}{\sqrt{2}\ \sqrt{E - 1 + \text{Cos}[\theta]}}\, d\theta$$

et l'énergie E est reliée à l'amplitude maximale A par E = 1 - Cos[A], soit

$$T = 2\sqrt{2} \int_0^A \frac{1}{\sqrt{\text{Cos}[\theta] - \text{Cos}[A]}}\, d\theta$$

Sous cette forme, *Mathematica* (en version 4.2!) ne sait pas calculer l'intégrale, il faut donc faire des changements de variable.

```
Dt[θ] / Sqrt[Cos[θ] - Cos[A]] /. {A → 2 ArcSin[k]} // Simplify
```

Il faut ensuite effectuer le changement de variable ci-dessous. Il est indispensable de préciser que **k** doit être considéré comme une constante. Il faut aussi utiliser **FullSimplify**, avec les bonnes hypothèses (pour information, on donne le résultat sans ces hypothèses)

```
SetAttributes[k, Constant]
```

$$\frac{\text{Dt}[\theta]}{\sqrt{-1+2\,k^2+\text{Cos}[\theta]}} /. \{\theta \to 2\,\text{ArcSin}[k\,\text{Sin}[\phi]]\} // \text{FullSimplify}$$

$$\frac{\text{Dt}[\theta]}{\sqrt{-1+2\,k^2+\text{Cos}[\theta]}} /. \{\theta \to 2\,\text{ArcSin}[k\,\text{Sin}[\phi]]\} //$$

$$\text{FullSimplify}\left[\#,\ 0<k<1 \&\&\ 0<\phi<\frac{\pi}{2}\right]\ \&$$

Reste à transformer la borne d'intégration!

$$\text{Solve}\left[\left(\text{A} /. \{\text{A} \to 2\,\text{ArcSin}[k]\}\right) == 2\,\text{ArcSin}[k\,\text{Sin}[\phi]],\ \phi\right]$$

On peut alors calculer l'intégrale, et revenir à la variable A

$$2\sqrt{2}\int_0^{\frac{\pi}{2}} \frac{2}{\sqrt{2-2\,k^2\,\text{Sin}[\phi]^2}}\,d\phi /. \left\{k \to \text{Sin}\left[\frac{\text{A}}{2}\right]\right\}$$

A.III-1) Tracer le graphe de la période en fonction de A. Que vaut la période pour A = 0? Mettre en évidence la divergence logarithmique de la période pour $A \to \pi$.

A.III-2) Obtenir les trois premiers termes du développement en puissances de A de la période. Commentez le résultat.

B. Anneau en rotation.

Désormais, l'anneau tourne à la vitesse angulaire constante ω dans le référentiel du Laboratoire.

Mise en équation

Cette fois, les équations de Lagrange sont de loin les plus pratiques pour trouver l'équation du mouvement. Nous prendrons la même unité de temps que précédemment, et on introduira la fréquence sans dimension $\Omega \equiv \omega\sqrt{\frac{R}{g}}$

$$\text{E}_\text{C} = \frac{1}{2}\,\text{m}\,\text{R}^2\left((\partial_t\,\theta[t])^2+\omega^2\,\text{Sin}[\theta[t]]^2\right);\ \text{E}_\text{P} = \text{m}\,\text{g}\,\text{R}\,(1-\text{Cos}[\theta[t]]);$$

$$\text{L} = -\text{g}\,\text{m}\,\text{R}\left(1-\text{Cos}\left[\phi\left[\sqrt{\frac{\text{g}}{\text{R}}}\,\text{t}\right]\right]\right)+$$

$$\frac{1}{2}\,\text{m}\,\text{R}^2\left(\omega^2\,\text{Sin}\left[\phi\left[\sqrt{\frac{\text{g}}{\text{R}}}\,\text{t}\right]\right]^2+\left(\partial_t\phi\left[\sqrt{\frac{\text{g}}{\text{R}}}\,\text{t}\right]\right)^2\right) /. \sqrt{\frac{\text{g}}{\text{R}}}\,\text{t} \to \tau;$$

$$\text{L} = \text{L}\,/\,(\text{m}\,\text{g}\,\text{R})\ //\ \text{Simplify}$$

$$\text{L} = \text{L} /. \left\{\omega \to \Omega\sqrt{\frac{\text{g}}{\text{R}}}\right\}$$

$$\partial_\tau\,\partial_{\partial_\tau\phi[\tau]}\,\text{L} - \partial_{\phi[\tau]}\,\text{L}\ //\ \text{Simplify}$$

L'équation du mouvement est donc

$$\phi''[\tau] = -(1-\Omega^2\,\text{Cos}[\phi[\tau]])\,\text{Sin}[\phi[\tau]]$$

B.I- Positions d'équilibre. Bifurcation fourche

B.I-1) Quelles sont les positions d'équilibre de la bille lorsque l'anneau est en rotation ? Déterminer

leur stabilité en linéarisant l'équation du mouvement au voisinage de la position d'équilibre.

B.I-2) Représenter sur un graphique les positions d'équilibre en fonction de Ω, en traits pleins (resp. pointillé) pour les positions stables (resp. instables). Montrer qu'au voisinage de la bifurcation la nouvelle position d'équilibre varie comme la racine carrée de l'écart au seuil : $A \propto \sqrt{\Omega - 1}$.

B.I-3) Représenter l'évolution de la période d'oscillation de la bille autour d'une position d'équilibre stable, en fonction de Ω. Etudier le comportement asymptotique près de la bifurcation.

B.II Etude énergétique

Le Hamiltonien du système se calcule sans difficulté.

```
H = ∂τ ϕ[τ] (∂∂τ ϕ[τ] L) - L // Simplify
```

C'est une constante du mouvement, puisque le Lagrangien ne dépend pas explicitement du temps. Cependant, son interprétation physique est moins simple que dans le cas du pendule simple : il ne s'agit pas de l'énergie totale dans le référentiel du Laboratoire, mais de l'énergie dans le référentiel (non galiléen) mobile lié au cerceau.
On définit une nouvelle énergie potentielle :

```
U[ϕ_, Ω_] := 1/2 (2 - 2 Cos[ϕ] - Ω^2 Sin[ϕ]^2)
```

B.II-1) Représenter celle-ci en fonction de ϕ, pour différentes valeurs de Ω (avant, à et après la bifurcation). Retrouver les conclusions précédentes sur la stabilité des positions d'équilibre. Calculer le développement limité de l'énergie potentielle à la bifurcation U[ϕ,1] au voisinage de $\phi = 0$. Commentaire?

B.II-2) Tracer un portrait de phase de l'oscillateur, après la bifurcation (une "bonne" valeur est Ω = 3). On tracera en gras les trajectoires passant par les points d'équilibre instables ($\phi = \pi$, $\phi' = 0$) et ($\phi = 0$, $\phi' = 0$), qui sont appelées *séparatrices* (voyez-vous pourquoi?). On tracera ensuite des trajectoires correspondants aux autres cas (à l'intérieur de la première séparatrice, entre les deux, à l'extérieur).

Oscillations non-linéaires

A) Développement asymptotique : Introduction.

A-1) Impossibilité d'un développement "naïf"

L'équation différentielle θ"[t] + Sin[θ[t]]==0 n'a pas de solutions exactes (sauf en terme de fonctions spéciales) pour une amplitude θ quelconque (non infinitésimale). Une stratégie naturelle est d'essayer de trouver une solution à amplitude faible, sous forme d'un développement en série. Nous introduirons explicitement un petit paramètre ϵ,

```
θ[t] = ϵ θ1[t] + ϵ^2 θ2[t] + ϵ^3 θ3[t]
```

On parle de *développement asymptotique*, car à la différence d'un développement en série de Taylor, ou en série de Fourier, il n'existe en général pas de théorème assurant la convergence de ce type de développement. On espère juste (et on constate en pratique!) que les termes sont de plus en plus petits, $\theta 1[t] > \theta 2[t] > \theta 3[t]$.

```
∂t,t θ[t]
```

```
Series[-Sin[θ[t]], {ϵ, 0, 3}]
```

Si l'on veut transformer ce développement en série en un polynome en ϵ, pour en extraire par exemple les coefficients, il faut appliquer la fonction Normal

```
poly = Normal[%];
```

```
Coefficient[poly, ϵ, 2]
```

Essayons alors de résoudre l'équation à l'ordre ϵ. Comme condition initiale, nous prendrons θ[0]=ϵ, θ'[0]=0, soit θ1[0]=1, θn[0] = 0 pour n > 1, θn'[0]=0.

```
solun = DSolve[{θ1''[t] + θ1[t] == 0, θ1[0] == 1, θ1'[0] == 0}, θ1[t], t]
```

Jusque là, tout va bien puisqu'on trouve la solution pour une amplitude infinitésimale. On espère que les termes d'ordre suivant vont apporter une petite correction, valable pour ϵ petit mais non strictement infinitésimal.
Le calcul de l'ordre ϵ^2n'apporte rien, car le premier terme non linéaire provenant du Sinus est cubique. Vérifions le quand même!

```
DSolve[{θ2''[t] + θ2[t] == 0, θ2[0] == 0, θ2'[0] == 0}, θ2[t], t]
```

A l'ordre ϵ^3, le terme en $\frac{1}{6}\theta 1[t]^3$ intervient comme un forçage dans l'équation pour θ3[t]

```
1/6 θ1[t]^3 /. solun
```

```
DSolve[{θ3''[t] + θ3[t] == 1/6 Cos[t]^3, θ3[0] == 0, θ3'[0] == 0}, θ3[t], t]
```

Tâchons maintenant d'analyser ce résultat.

Les deux premiers termes sont conformes à ce que l'on pouvait attendre, avec une petite correction d'ordre ϵ^3au terme en Cos[t], et l'apparition d'un harmonique 3 qui vient de la nonlinéarité cubique.

Par contre, le terme en t Sin[t] diverge aux temps grands, alors que l'énergie est conservée, ce qui est physiquement inacceptable. Le terme en $\epsilon^3\,\theta 3[t]$, qui dans notre raisonnement était sensé être une petite correction, diverge, donc quelque chose ne va pas!

Si on analyse le second membre de l'équation différentielle, on voit d'où provient le problème :

```
1/6 θ1[t]^3 /. solun // TrigReduce
```

Cette forme fait apparaître un terme en Cos[t], qui correspond à un forçage résonant, à la fréquence propre de l'oscillateur.

A-2) Nécessité d'un double développement. Méthode de Poincaré-Lindstedt.

En fait, notre méthode était inadaptée dès le début. En effet, avec le développement choisi les termes non linéaires ne peuvent faire apparaître que la fréquence du terme linéaire, et ses harmoniques. Or nous avons calculé au TD précédent la fréquence de l'oscillateur dans le cas général, et montré qu'elle dépend de l'amplitude. La bonne méthode consiste donc à faire un *double développement*, en supposant dès le départ que la solution cherchée dépend d'une variable $\tau = \omega t$ où ω est une fonction de ϵ. On aura ainsi

$$\frac{\partial \theta}{\partial t} = \frac{\partial \theta}{\partial \tau}\frac{\partial \tau}{\partial t} = \omega \frac{\partial \theta}{\partial \tau} \text{ et } \frac{\partial^2 \theta}{\partial t^2} = \frac{\partial}{\partial t}\left(\frac{\partial \theta}{\partial t}\right) = \omega^2 \frac{\partial^2 \theta}{\partial \tau^2}$$

Explicitons maintenant le double développement. (on ne garde dès le départ que les termes qui vont vraiment servir, en éliminant les termes en $\epsilon\,\omega 1$ et en $\epsilon^2\,\theta 2$!)

```
Remove["Global`*"]
```

```
ω = 1 + ω2 ϵ^2;
```

```
θ[τ] = ϵ θ1[τ] + ϵ^3 θ3[τ];
```

```
Series[ω^2 (∂_{τ,τ} θ[τ]), {ϵ, 0, 3}]
```

```
Series[-Sin[θ[τ]], {ϵ, 0, 3}]
```

L'équation différentielle à l'ordre ϵ est donc inchangée (notez toutefois la syntaxe très légèrement différente; pourquoi avoir choisit cette syntaxe?)

```
solun = DSolve[{θ1''[τ] + θ1[τ] == 0, θ1[0] == 1, θ1'[0] == 0}, θ1, τ]
```

A l'ordre ϵ^3, le terme de forçage devient

```
forcage = -2 ω2 θ1''[τ] + 1/6 θ1[τ]^3 /. solun // TrigReduce
```

On retrouve donc le terme précédent, avec en plus un terme de forçage en $\omega 2$ Cos[τ]. On peut alors choisir $\omega 2$ de façon à supprimer tout terme de forçage résonant!

```
ω2 = -1/16;
```

```
forcage
```

```
soltrois = DSolve[{θ3''[τ] + θ3[τ] == forcage[[1]], θ3[0] == 0, θ3'[0] == 0}, θ3, τ]
```

```
((θ[τ] /. solun) /. soltrois) /. {τ → ω t}
```

Cette solution est maintenant physiquement correcte. Il apparaît bien des harmoniques, mais la période de la solution dépend de l'amplitude ϵ, et les termes en τ Sin[τ] ont été éliminés. On vérifie que l'on retrouve le résultat du TD précédent; en effet la période que nous venons de calculer, à l'ordre ϵ^2vaut

```
Series[2 π/ω, {ϵ, 0, 2}]
```

tandis que le développement limité de la formule générale, au même ordre, donne

```
Series[4 EllipticK[Sin[ϵ/2]^2], {ϵ, 0, 2}]
```

B) Développement asymptotique : Application.

Si l'on veut une solution précise, il faut calculer beaucoup de termes, ce qui devient vite très lourd à la main. L'idée est d'utiliser les possibilités du calcul formel pour mener les calculs automatiquement.

B-1) Pousser le développement à l'ordre ϵ^7. (avec les mêmes conditions initiales $\theta(0) = \epsilon$ et $\theta'(0) = 0$).

Si vous ne vous sentez pas trop à l'aise, vous pouvez commencer par faire les calculs "manuellement", c'est-à-dire en procédant par copier / coller pour écrire vos commandes.
Il est néanmoins possible de contruire une procédure qui fasse l'intégralité du calcul automatiquement. Voici quelques astuces qui vous permettront de construire ce programme

(1) Ecriture automatique des développements

```
θ[t] = Sum[ϵ^i ToExpression["θ" <> ToString[i]][t], {i, 1, 7, 2}]
```

Quelques astuces pour la suite:
(2) La fonction **`Coefficient[poly, X, n]`** vous permet d'extraire les coefficients de la variable **`X`** d'un polynôme **`poly`**, prise à l'ordre **`n`**
(3) Pour obtenir un polynôme à partir d'un développement en série, appliquer la fonction **`Normal[]`**
(4) Il est fortement conseillé d'utiliser comme syntaxe dans **`DSolve`** celle qui permet d'obtenir le résultat sous forme de fonction pure
(5) Pensez à utiliser **`Flatten`** pour contrôler la "profondeur" des listes fournies automatiquement par le logiciel.

B-2) Comparer le résultat de votre calcul à une intégration numérique de l'équation différentielle, pour diverses valeurs de ϵ. (A l'ordre 7, le résultat reste très bon même si ϵ est plus grand que 1)

B-3) Le développement asymptotique donne un développement (à quel ordre?) de la période en fonction de ϵ. Comparer avec la formule analytique de la période obtenue au TD précédent.

Equation de Korteweg-de Vries

Nous allons étudier dans ce TD quelques solutions de l'équation de Korteweg-de Vries (KdV)

$$\partial_t \psi + \psi \partial_x \psi + \partial_{x,x,x} \psi = 0$$

Cette équation décrit la propagation d'ondes (par exemple, les ondes de gravité à la surface d'un fluide de profondeur uniforme, petite devant la taille typique des ondes) à une dimension, en régime nonlinéaire. Cette équation, à la fois dispersive (terme en $\partial_{x,x,x}\psi$) et non linéaire (terme en $\psi \partial_x \psi$) présente des propriétés remarquables.

Elle possède comme solution des *ondes solitaires*, excitations nonlinéaires qui réalisent une balance exacte entre effets dispersifs et nonlinéaires, et se propagent sans déformation. Nous les étudierons dans la partie I.

Nous verrons dans la partie II qu'il est possible de trouver une solution exacte décrivant la propagation, et l'intéraction, de deux ondes solitaires. Cette intéraction est *élastique*, les deux ondes retrouvant leur profil après l'intéraction. Comme elles gardent leur individualité, on les appelle *solitons,* par analogie avec la physique des particules.

Nous ne le montrerons pas, mais ce résultat se généralise à un nombre quelconque de solitons. L'équation KdV est un exemple, désormais classique, d'équation *intégrable*.

I - Solution de type "Onde solitaire"

La méthode que nous allons utiliser est tout-à-fait générale, et permet de trouver des solutions de type "onde solitaire" dès qu'elles existent. Une *onde solitaire* est une excitation localisée (donc nulle en ±∞) se propageant comme une onde, à une vitesse V. Plus précisément, nous cherchons (et rien ne dit a priori qu'elles existent!) des solutions de la forme

$$\psi(x,t) = \Psi(\xi \equiv x - V t), \text{ avec } \Psi(\pm\infty) = 0$$

On considèrera des ondes se propageant vers la droite, donc avec V > 0.

I - 1) Chercher une solution de KdV sous cette forme. Aboutir à une équation différentielle pour $\Psi[\xi]$

I - 2) Intégrer une première fois cette équation. *Mathematica* n'écrit pas la constante d'intégration : quelle est sa valeur pour une onde solitaire?

I - 3) L'équation s'écrit alors formellement comme un problème de mécanique décrivant le mouvement d'une particule à la "position" Ψ dans un "potentiel" U[Ψ] : $\ddot{\Psi} = -\frac{dU}{d\Psi}$. Identifier le potentiel U[Ψ] et le représenter graphiquement pour V = 1.

I - 4) Identifier, pour une valeur quelconque de V, les positions d'équilibre pour ce potentiel. Déterminer leur stabilité.

I - 5) Montrer qu'il existe des solutions oscillantes. Quelle inégalité vérifie leur "énergie mécanique"? Faire un portrait de phase pour cet oscillateur (pour V = 1). Quelle "énergie" correspond à la séparatrice?

NB : On peut aller voir en fin de TD le complément sur les portraits de phase

I - 6) Expliquer que la solution "onde solitaire" du problème d'ondes nonlinéaires correspond à la séparatrice pour le problème mécanique associé. Calculer cette onde solitaire, et la représenter graphiquement.

II - Solution à deux solitons

II-1) Transformation de Cole-Hopf

On fait la transformation $\psi \equiv \partial_x p$, puis on intègre sur x. L'équation de KdV se met alors sous la forme

$$\partial_t p + \frac{1}{2}(\partial_x p)^2 + \partial_{x,x,x} p = 0$$

Faire dans cette expression la transformation de Cole-Hopf, qui consiste à poser

$$p = 12 \frac{\partial_x F}{F}$$

(Rq importante : Il faut définir la fonction p à partir de la fonction F, comme ci-dessous)

```
p[x_, t_] := 12 ∂x F[x, t] / F[x, t]
```

II-2) Réécriture de la solution à un soliton

Vérifier qu'on retrouve la solution à un soliton en posant

F = 1 + exp(θ) où θ = -α(x-a) + α^3t

II-3) Solution à deux solitons : Méthode de Hirota

Nous allons chercher une solution à deux solitons par la méthode de Hirota. L'équation aux dérivées partielles vérifiée par F s'écrit

$$F \partial_{x,x,x,x} F - 4 \partial_x F \partial_{x,x,x} F + 3 (\partial_{x,x} F)^2 + F \partial_{x,t} F - \partial_x F \partial_t F = 0$$

On se propose de la résoudre par un *développement formel*

$$F = 1 + \epsilon F1 + \epsilon^2 F2 + \epsilon^3 F3 + \ldots$$

où ϵ ne sera considéré comme un petit paramètre que pour résoudre le problème ordre par ordre. En fin de calcul, ϵ doit être pris égal à 1. Ce type de méthode transforme une équation aux dérivées partielles non linéaire en une *hiérarchie* (infinie, a priori) d'équations aux dérivées partielles linéaires. On ne gagne donc strictement rien! Sauf si, comme nous allons le voir, cette hiérarchie d'équations se révèle n'en comporter qu'un nombre fini!

II − 3 − a) Obtenir les équations jusqu' à l' ordre ϵ^4 inclus.

(Rq : penser que pour utiliser les fonctions d'analyse de polynômes (telles que **Coefficient**, par ex.) sur un développement en série il faut mettre celui-ci sous forme "normal" avec **Normal**)

Clear[F]

II-3-b) Vérifier que la fonction

F1[x,t] = Exp[θ1[x,t]], où θ1[x,t] = α (x − a) − α^3 t

est solution de l'équation obtenue à l'ordre ϵ.

II-3-c) L'équation obtenue à l'ordre 1 étant *linéaire*, on peut prendre une forme plus générale pour F1[x,t]. On posera donc

$$F1[x,t] = \text{Exp}[\theta 1[x,t]] + \text{Exp}[\theta 2[x,t]], \text{ où } \begin{cases} \theta 1[x,t] = \alpha(x-a) - \alpha^3 t \\ \theta 2[x,t] = \beta(x-b) - \beta^3 t \end{cases}$$

En déduire l'équation vérifiée par F2[x,t]. Montrer que la forme de F2 est γ Exp[θ1 + θ2], et calculer la constante γ.

(Rq : Bien penser à effacer la définition précédente de F1)

II-3-d) Reporter dans l'équation pour l'ordre ϵ^3 les solutions trouvées pour F1 et F2. En déduire qu'on peut prendre F3 = 0. Faire le même calcul pour F4. En déduire Fn = 0 pour n ⩾ 3.

(Rq : Comme précédemment, nettoyer la définition de F2.)

II-3-e) Calculer la solution trouvée F[x,t]. On prendra comme valeur des constantes $\alpha = 1$, $\beta = 2$, a = 0 et b = -10. Tracer la solution à t = 0. Pourquoi cette solution est-elle dite à deux solitons? Estimer les vitesses de chaque soliton. Leur amplitude était-elle prévisible? Proposer un dessin lorsque le plus rapide a rejoint le plus lent, puis lorsqu'il l'a dépassé. Conclure.

(Rq : Les fonctions d'animation de graphiques de *Mathematica* permettent une représentation vivante du phénomène.)

A - Compléments sur les portraits de phase

Plaçons nous dans le cadre d'un problème de Mécanique du point. Dans un cas très général, il s'écrira (pour une particule de masse unité) sous la forme

$$\ddot{x} = f(x, \dot{x}, t) \text{ avec les conditions initiales } x_0 = x(t = 0),\ \dot{x}_0 = \dot{x}(t = 0).$$

Sous des hypothèses assez large de régularité de la fonction f, on montre qu'il *existe une solution unique* à ce problème.

Il est en pratique assez rare que l'on soit capable d'exprimer la solution à l'aide de fonctions connues. Ce théorème d'existence et d'unicité des solutions d'une équation différentielle ordinaire, a néanmoins une conséquence pratique importante. Il établit en effet que la recherche numérique de la solution doit donner un résultat, unique pour des conditions initiales données.

Pour se faire une idée du comportement des solutions, il est pratique de faire un portrait de phase de l'équation, c'est à dire de représenter quelques trajectoires $(x(t),\ \dot{x}(t))$ pour différentes conditions initiales. De façon la plus générale possible, ces trajectoires sont obtenues comme la représentation paramétrique des solutions de l'intégration numérique de l'équation. Commençons par l'exemple d'une équation différentielle qui n'est pas intégrable analytiquement.

A-1-Oscillateur de Van der Pol

Etudions l'oscillateur de Van der Pol, dont l'équation différentielle sur la fonction *u(t)*, en variables sans dimensions, s'écrit

$$u'' - \epsilon(1 - u^2)\,u' + u = 0$$

où ϵ (qui n'est pas nécessairement petit) est le seul paramètre du problème.

A-1-a) On prend $\epsilon = 0.1$. Résoudre numériquement l'équation différentielle avec les conditions initiales *u(0) = A* et *u'(0) = 0.* L'amplitude initiale *A* sera choisi aléatoirement entre 0 et 1. Représenter *u(t)*, ainsi que la trajectoire dans l'espace de phase *(u , u')*.

A-1-b) Reprendre ces questions en partant cette fois d'une amplitude initiale *A* choisie aléatoirement entre 2 et 4. On dit que *l'attracteur de l'oscillateur est un cycle limite*. Voyez-vous pourquoi?

A-2-Cas particulier de la conservation de l'énergie mécanique.

Lorsque le problème présente une intégrale première, le dessin du portrait de phase est grandement simplifié. En effet, si l'équation s'écrit

$$u'' = -\frac{dE_p}{du},$$

où E_p est une énergie potentielle,on en déduit une intégrale première

$$\frac{1}{2}u'^2 + E_p(u) = E_m = \text{Cste}$$

et les trajectoires dans l'espace de phase sont donc données par

$$\pm\sqrt{2\,(E_m - E_p(u))} = u'.$$

Intégrer l'équation différentielle est donc inutile pour tracer le portrait de phase.

Reprenons le cas particulier du potentiel qui apparaît lorsqu'on résout le problème mécanique associé à l'équation de Korteweg-de Vries. Je vous propose de vérifier, même si cela va

largement de soi, que les deux méthodes donnent les mêmes trajectoires dans l'espace de phase.

Limitons nous au cas où des oscillations existent, et prenons comme valeur initiale $\Psi(t = 0) = \Psi_0$, et $\dot{\Psi}(t = 0) = 0$. Considérons le cas V = 1.