Université Paris-Sud **Année 2010-11**
Master 1 de Physique Appliquée et Mécanique

Outils Informatiques pour la Physique / Matlab

Eléments de base de Matlab

Introduction

Nous rappelons ici quelques éléments de base sur ce logiciel et nous fournissons une liste des principales commandes. L'information fournie restera tout de même minimale, car le menu "aide" de Matlab contient une description détaillée de toutes les commandes, toujours munis d'exemples clairs. Consultez-le!

Fenêtres

Ouvrant Matlab, nous avons un ensemble de fenêtres.

- **command window**
  Le prompt. Ici nous entrons les commandes pour définir et faire des opérations sur des variables.
- **workspace**
  Les variables qui sont actuellement présentes dans la mémoire vive de l'ordinateur. On peut changer leur nom, voir leur taille et leur type et les sauvegarder.
- **current directory**
  Permet de naviguer dans le systèmes de fichiers. On peut choisir le répertoire de travail, créer des répertoires, ouvrir des fichiers ou changer leur noms.
- **command history**
  La liste des dernières commandes utilisées. Dans le "command window", on peut faire appel aux précédentes commandes avec les opérations (flèches haut & bas).
- **help navigator**
  Une large documentation sur toutes les commandes qui sera souvent utile.
- **editor**
  Editeur de scripts et de fonctions. Permet de créer des listes de commandes Matlab regroupées sous un seul nom (script) ou de créer des fonctions qui agissent sur des variables que l'on procure à l'appel de la fonction.
- **figure**
  Voir et sauvegarder les figures. Modifier leur caractéristiques.

Variables élémentaires

Matlab utilise des variables de divers types "classiques". Contrairement à la plupart langages de programmation, le type des variables est automatiquement reconnu si nous utilisons la bonne synthaxe

- **integer**
  Nombres entiers

```
>> x = 1
```

- **single & double**
  Nombres non-entiers précision single (8 chiffres siginificatifs) ou double (16 chiffres). Par défaut on travaille en précision double. Quelques constantes sont prédéfinies, par exemple π

  ```
  >> pi = 3.141592653589793
  ```

- **complex**
  Il suffit de faire précéder la partie imaginaire par i ou j

  ```
  >> z = 1 + 2i
  ```

- **logical**
  Variables logiques

  ```
  >> vrai = true
  >> faux = false
  ```

- **string**
  Suites de caractères à mettre entre deux primes

  ```
  >> nom = 'fichier1'
  ```

Affichage

Par défaut, les résultats des opérations sont affichés dans le "command window". On peut supprimer cet affichage en mettant un poit-virgule (;) derrière les commandes. Plusieurs commandes peuvent être séparées de virgules et de point virgules. Par exemple,

```
>> a = 1; b = 2, c = 3;
```

n'affichera que b sur l'écran. Le format modifie le nombres de chiffres significatifs à l'écran

```
>> format short    ou   format long
```

Matrices

- **A la main**

  Utilisez crochets carrés [ ] avec des virgules (,) pour séparer des colonnes et des point-virgules (;) pour séparer les lignes. Entrez les données ligne par ligne.

  ```
  >> A = [3,1,4; 4,6,1]
  ```

  $$\text{A} = \begin{bmatrix} 3 & 1 & 4 \\ 4 & 6 & 1 \end{bmatrix}$$

  Si les nombres d'éléments dans chaque ligne ne sont pas égaux, Matlab nous donne un message d'erreur.

- **Prédéfinie**

  Quelques commandes prédéfinies pour créer des matrices de structure simple. Par exemple

  ```
  >> Z1=zeros(2) , Z2=ones(1,3) , Z3=eye(2), Z4 = diag([3,2])
  ```

$$Z1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad , \quad Z2 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \quad , \quad Z3 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad , \quad Z4 = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$$

- **Points équidistants**

  Des listes avec des points équidistants à l'aide du double point (:)

  ```
  >> y=1:1:3.1 , z=9:-3:3
  ```

  ```
  y = [1 2 3] , z = [9 6 3]
  ```

  L'incrément peut donc être négatif et le point final de la liste n'est jamais dépassé.

- **Coller ensemble des matrices**

  Deux matrices peuvent s'assembler si leur tailles sont compatibles

  ```
  >> A=[3,1,4;4,6,1]; x=[1,2,5] ; B=[A;x]
  ```

  $$B = \begin{bmatrix} 3 & 1 & 4 \\ 4 & 6 & 1 \\ 1 & 2 & 5 \end{bmatrix}$$

- **Accéder aux éléments**

  Coordonnées ou listes de coordonnées se trouvent en crochets rondes derrière le variable. Pour la matrice B de ci-dessus

  ```
  >> b = B(2,[1,3])
  ```

  b = [4 1]

  Accédez à une colonne ou ligne entière avec le double-point (:)

  ```
  >> c = B(:,2)
  ```

Quelques commandes de base

Aide

Consultez les documentations (”help navigator”) pour avoir plus de détails sur les options et les nombreuses autres commandes que l'on ne discutera pas ici. Pour la commande au nom `eig`

```
>> help eig
```

donne des documentations directement dans le ”command window”. Pour accéder à plus d'information

```
>> doc eig
```

ouvre la documentation détaillée sur la commande.

Calcul élémentaire

Les principales opérations et fonctions sont prédéfinies avec des commande simples et intuitives. Quelques règles sont à respecter si on opère sur des matrices. Nous notons x et y des matrices, listes ou vecteurs alors que a notera toujours un scalaire s'il est important de faire cette différence.

- `x+y , x−y, x+a`
  Deux matrices x et y ne peuvent s'additionner si elles ont les mêmes dimensions.
  Un scalaire a s'additionnera avec tous les éléments de matrice x.
- `x*y , x*a`
  Multiplication matricielle, si les dimensions de x et y sont compatibles pour la multiplication.
  Un scalaire a se multipliera avec tout élément de la matrice x.
- `x/a`
  L'opérateur / est surtout utilisé pour diviser des matrices x par des scalaires a.
- `x.*y , x./y`
  Multiplication et division élément par élément, si x et y ont la même taille.
- `x^a`
  Matrice x à la puissance a, au sens matriciel.
- `x.^a`
  Matrice x à la puissance a, élément par élément.
- `a.^x`
  Listes de puissances x du scalaire a, élément par élément.
- `x.'`
  Transposé.
- `x'`
  Hermite (transposé & complexe conjugé)
- `real(x) , imag(x) , conj(x)`
  Partie réelle, imaginaire et complex conjugué de tous les éléments de x.
- `sqrt(x)`
  Racine carrée de tous les éléments de tous les éléments de x.
- `exp(x)`
  Exponentielle de tous les éléments de x.
- `log(x) , log10(x)`
  Logarithme en base e, 10 de tous les éléments de x
- `sin(x) , cos(x), tan(x)`
  Fonctions trigonométriques de tous les éléments de x.
- `asin(x) , acos(x), atan(x)`
  Fonctions trigonométriques inverses de tous les éléments de x

Opérations logiques et relationelles

La relation s'exprime toujours élément par élément et nécessite donc des matrices de même taille, sauf dans le cas ou on compare une matrice à un scalaire.

- `x==y , x~=y , x<y , x<=y x >y , x>=y`
  Egalité, différence, plus petit (ou égale), plus grand (ou égale).
- `x&&y , x||y , xor(x,y)`
  Operations logiques AND et OR et XOR

Taille, ordre & statistique de base

- `size(x)`

  Nombre de lignes et de colonnes.

- `length(x)`

  Longuer de la liste.

- `sort(x,'ascend') , sort(x,'descend')`

  Ordonner les listes selon l'ordre montant ou descendant.

- `max(x) , min(x)`

  Maximum et minimum pour des listes ou vecteurs.
  Consultez la documentation pour les matrices.

- `mean(x) , std(x)`

  Valeur moyenne et écart type.

Algèbre linéaire

- `inv(A)`

  Inverse matricielle si A est carré.

- `det(A)`

  Déterminant de A, si A est carré.

- `norm(x)`

  Norme euclidienne de la liste x.

- `eig(A)`

  Valeurs et vecteurs propres de A si A carré.

- `A\y`

  Solution x du système linéaire Ax = y.

Tests & Boucles

Comme dans tout langage de programmation, nous avons besoin de tests et de boucles conditionnelles et inconditionnelles.

test simple	**test multi**	**boucle incond.**	**boucle cond.**
`if` $x == 1$	`switch` x	`for t=1:10`	`while test==1`
...	...	...	...
`elseif x > 1`	`case` $x == 1$	`end`	`end`
...	...		
`else`	`case ...`		
...	...		
`end`	`otherwise`		
	...		
	`end`		

Visualisation

De nombreuses fonctions existent pour visualiser des données. Après création de la figure, nous pouvons la modifier dans la fenêtre de la figure même. Ajoutez toujours les noms des axes et une légende si plusieurs courbes sont tracées.

Général

- `figure, figure(n)`

  Ouvrir une nouvelle fenêtre de figure.
  Ouvrir ou allez vers la figure n.

- `grid on (off)`

  (on) afficher la grille (off) ou pas.

- `hold on (all) (off)`

  (on) Pour afficher plusieurs courbes sur la même figure.
  (off) Arrêter de dessiner sur la même figure.

- `xlim([x1,x2]) , ylim([y1,y2])`

  Montrer seul l'intervalle de x1 à x2 de l'axe x .
  Pareil pour y.

- `xlabel('x') , ylabel('y') , title('titre')`

  Ajouter les noms des axes, un titre

- `[x,y]=ginput(n)`

  Mesurer les coordonnées de n points dans la figure

Courbes y = f (x)

- `plot(x,y) et plot(x,y,options)`

  Courbe y en fonction de x. De nombreuses options existent pour choisir couleurs, marker, type de lignes. Tapez "doc plot" pour trouver des exemples.

- `semilogx(x,y) , semilogy(x,y) , loglog(x,y)`

  Courbe y en fonction de x. x ou y ou tous les deux dans une échelle logarithmique,

- `line(x,y)`

  Dessiner des lignes droites entres les points spécifiés dans les listes x et y

- `legend('data1','data2',...)`

  Ajouter une légende pour spécifier plusieurs courbes sur la même figure.

Surfaces z = F(x, y)

- `[x,y]=meshgrid(x1:x2,y1:y2)`

  Faire une grille de points pour la visualisation d'une surface.

- `surf(x,y,F)`

  Visualisation de la fonction F(x,y) sous forme d'une surface.

- `pcolor(x,y,F) , contour(x,y,F,n)`

  Visualisation de la fonction F(x,y) utilisant un code couleur.
  Afficher n iso-contours de F (x,y).
  A utiliser avec ”shading” et ”colorbar”

- `shading facetted (flat) (interp)`

  Montrer ou cacher la grille, interpoler les couleurs pour une visualisation plus lisse.

- `colorbar`

  Ajoutez la gradation de couleur

- `colormap jet (hsv) (gray) (...)`

  Modifier la palette de couleurs. Plusieurs choix.

Effacer, sauvegarder et charger

- `clear , clear x`

  Effacer tous les variables, ou seul le variable x de la mémoire vive/workspace

- `save test ou save('test')`

  Sauvegarder le workspace sur la disque dur, sous le nom test.mat.

- `load test ou load('test')`

  Charger des variables.

Créer ses propres fonctions

Il est possible et même souvent nécessaire de créer ses propres commandes dans l'éditeur de Matlab. Par exemple, la fonction au nom `fun`

```
function [out1,out2] = fun(in1,in2,in3)

...
```

prend les variables (`in`) à l'entrée et renvoie d'autres variables (`out`) à la sortie qui sont définies quelque part entre le début et la fin de la fonction. Il est nécessaire de la sauvegarder sous le même nom que la fonction porte elle-même, c.a.d. `fun.m` dans notre exemple. La fonction s'utilise comme une commande par la suite. On remarque finalement que toute variable intermédiaire autre que les variables `out` restera invisible pour l'utilisateur et sera effacé après exécution de la fonction.

Commentaires

Des commentaires permettent de lire les programmes plus facilement. Toute ligne précédée par % ... ne sera pas considérée par Matlab.

Université Paris-Sud **Année 2010-11**
Master 1 de Physique Appliquée et Mécanique

Outils Informatiques pour la Physique / Matlab

TP 1 : Eléments de base de Matlab

Le but du premier TP est de se rappeler ou de se familiariser avec la synthaxe de Matlab. Dans le premier exercice, nous faisons quelques opérations de base sur une matrice. Dans le deuxième exercice, nous procédons au calcul de π par des méthodes itératives, ce qui nécessitera l'emploi de boucles.

Matrice de Vandermonde

1. Une matrice de Vandermonde de dimension 4 x 4 a la structure particulière suivante

$$\mathbf{A} = \begin{bmatrix} 1 & a & a^2 & a^3 \\ 1 & b & b^2 & b^3 \\ 1 & c & c^2 & c^3 \\ 1 & d & d^2 & d^3 \end{bmatrix}$$

   avec quatre paramètres que l'on choisira $a = 1$, $b = 2$, $c = 3$, $d = 1.234$. Trouvez une manière simple pour charger cette matrice en Matlab.

2. Calculez les valeurs propres λ et les vecteurs propres $\mathbf{x}$ de la matrice $\mathbf{A}$, utilisant la commande `eig`. Testez pour toutes les valeurs et vecteurs propres séparément que $\mathbf{Ax} = \lambda\mathbf{x}$ soit bien satisfait.

Calcul itératif de π : série arctan

1. Le développement limité de Taylor pour la fonction arctan autour de $x = 0$ s'écrit

$$\arctan x = \lim_{N\to\infty} \sum_{n=0}^{N} (-1)^n \frac{x^{2n+1}}{2n+1}$$

   Le choix $x = 1$ permet de calculer π. Ecrivez la formule de la série pour π basé sur ce choix.

2. A l'aide d'une boucle inconditionnelle `for` , calculez les sommes partielles pour N allant de 0 à 9999. Nommez ce vecteur pi1.

3. Evaluez l'erreur absolue sous le nom de er1, en comparant les sommes partielles avec la valeur de π prédefinie dans Matlab. Sauvegardez pi1 et er1 dans un fichier cas1.mat

4. Tracez la valeur absolue de l'erreur dans un diagramme log-log en fonction de l'index 1 à 10000. Ce diagramme nous apprend que l'erreur décroît selon un loi de puissance

$$|\text{er1}| \sim N^{\alpha}$$

   Trouvez la valeur de l'exposant α.

5. A l'aide d'une boucle conditionnelle `while` , calculez le nombre minimum d'itérations N nécessaires afin d'obtenir une erreur absolue plus faible que 10^{-5}.

Calcul itératif de π : fraction continue

1. Au lieu du développement limité, nous choisissons la fraction continue

$$\arctan x = \cfrac{x}{1+\cfrac{(x)^2}{3+\cfrac{(2x)^2}{5+\cfrac{(3x)^2}{7+\cfrac{(4x)^2}{9+\ldots}}}}}$$

   Comme avant avec les sommes partielles, cette équation permet d'obtenir des approximations successives de la fonction. Les approximations aux ordres $N = 0, 1, 2$ seront alors

$$\begin{array}{lll} N=0 & , & \arctan x = x/1 \\ N=1 & , & \arctan x = x/(1+x^2/3) \\ N=2 & , & \arctan x = x/(1+x^2/(3+(2x)^2/5) \end{array}$$

   et ainsi de suite. Nous choisissons de nouveau $x = 1$ pour l'évaluation de π.

2. Trouvez un algorithme qui vous permet d'obtenir des estimations successives de π pour $N = 0$ à 19. Une double boucle vous facilitera la tâche. Enregistrez les estimations successives dans un vecteur sous le nom pi3 et évaluez l'erreur absolue er3. Les deux vecteurs sont sauvegardés sous le nom cas3.mat.

3. Affichez l'erreur absolue er3 en fonction de N, dans un diagramme semi-logarithmique (er3 en échelle logaritmique). Vous devez obtenir une droite, ce qui démontre une convergence exponentielle de la fraction continue vers π en fonction de N ou encore

$$\text{er3} \sim e^{-\gamma N}$$

4. Trouvez la pente γ en mesurant deux coordonnées sur la droite à l'aide de la commande `ginput`

5. Estimez le nombre $\tilde{N}$ d'itérations tel que l'erreur soit plus petite que 10^{-50} en valeur absolue. Pourquoi est-il en pratique impossible d'atteindre une telle précision à l'aide d'un logiciel comme Matlab ?

Université Paris-Sud **Année 2010-11**

Master 1 de Physique Appliquée et Mécanique

Outils Informatiques pour la Physique / Matlab

TP 2 : Graphiques

Animation d'une onde propagative

On considére la propagation d'une onde à la surface de l'eau. On note $A(x,t)$ la hauteur de la surface à la position x et à l'instant t. On s'intéresse dans un premier temps à une onde monochromatique, de longueur d'onde λ et de période T. On introduit le vecteur d'onde $k = 2\pi/\lambda$, et la pulsation $\omega = 2\pi/T$. La vitesse de phase est donnée par $c = \omega/k$. Pour une propagation vers les x croissant, l'amplitude de l'onde est donnée par

$$A(x,t) = A_0 \cos(kx - \omega t).$$

On se restreint à l'intervalle $x \in [0, 10]$ m, pour une onde de longueur d'onde $\lambda = 2$ m, d'amplitude $A_0 = 0.1$ m, et de période $T = 1.4$ s.

1. Tracez cette onde à l'instant $t = 0$ (on choisira une discrétisation selon x adaptée).
2. Effectuez l'animation de cette onde propagative pendant 10 périodes, c'est-à-dire effectuez une itération du graphique précédent pour t entre 0 et $10T$.
3. On considère maintenant une onde atténuée par une enveloppe donnée par $\exp(-x/L)$ (pour $x > 0$). Comparez visuellement les cas où la longueur d'atténuation L est petite ou grande comparée à λ.
4. Comment modifier cette onde pour qu'elle se propage vers les $x < 0$? Observer l'interférence (c'est-à-dire la somme) de 2 ondes propagatives de même longueur d'onde et de même amplitude, $A_+(x,t)$ et $A_-(x,t)$, se propageant en sens inverse.
5. Observez l'interférence de 2 ondes propagatives de longueur d'onde très proches, se propageant dans le même sens. A quoi correspond visuellement la vitesse de groupe de ce système d'onde ?

Université Paris-Sud **Année 2010-11**

Master 1 de Physique Appliquée et Mécanique

Outils Informatiques pour la Physique / Matlab

TP 3 : Marcheurs aléatoires & diffusion

Considérons un marcheur qui évolue à chaque pas dans une direction arbitraire. Une marche aléatoire de N pas consiste en une réalisation d'un tel trajet. Les marcheurs aléatoires simulent le mouvement Brownien de traceurs microscopiques sous l'effet des fluctuations thermiques. Ils nous offrent une explication microscopique du phénomène macroscopique de la diffusion.

Nous étudions un marcheur aléatoire qui se "promène" sur un plan x-y en deux dimensions. Chaque pas sera de longueur unité, mais la direction dans le plan x-y est aléatoire. Si nous notons $p_x(t)$ et $p_y(t)$ les déplacements selon x et y suite au t-ième pas, nous pouvons définir ces pas aléatoires ainsi :

$$p_x(t) = \cos\phi(t) \quad , \quad p_y(t) = \sin\phi(t)$$

où $t = 1$ à N. Les arguments $\phi(t)$ sont des nombres aléatoires uniformément distribués dans l'intervalle $[0, 2\pi]$. Une fois les pas connus, il n'est pas difficile de calculer une marche partant de l'origine $\mathbf{0}$, selon la recette

$$\begin{array}{l} x(t) = x(t-1) + p_x(t) \\ y(t) = y(t-1) + p_y(t) \end{array} \quad , \quad t = 1, \ldots, N$$

Tous les marcheurs partiront de l'origine $x(0) = 0$, $y(0) = 0$ mais ce point initial ne fera pas partie des listes $x(t)$ et $y(t)$.

Visualisation d'une marche aléatoire

1. Matlab offre la possibilité de créer des matrices de nombres aléatoires uniformément distribués entre 0 et 1 à travers la commande `rand`. Consultez la documentation pour trouver la syntaxe permettant de créer les listes $\phi(t)$ pour $t = 1, \ldots, N$.

2. Créez une fonction `[x,y]=march2d(N)` qui prend comme valeur à l'entrée le nombre de pas N d'une marche aléatoire. A la sortie nous trouvons les coordonnées de la marche x et y sous forme de listes de N points.

3. Utilisez la fonction pour calculer 3 marches aléatoires de 1000 pas. Afficher les marches aléatoires dans le plan x-y en utilisant des couleurs différentes pour chaque réalisation. Ajoutez une légende, un titre et le nom des axes.

Etude statistique

1. Une étude statistique permet de montrer le lien avec la diffusion et repose sur le concept de la moyenne d'ensemble. Créez une nouvelle fonction `[x,y]=march2dstat(N,M)` qui prend à l'entrée le nombre de pas N, mais aussi le nombre M de réalisations indépendantes souhaitées. x et y sont deux matrices de taille N x M qui contiennent sur leurs colonnes les coordonnées des marches aléatoires. Evitez les doubles boucles !

2. Testez la nouvelle fonction pour des petites valeurs de M et N.

3. Calculez un nombre conséquent de $M = 1000$ réalisations de marches de $N = 1000$ pas. Diminuez ces nombres M ou N si jamais votre ordinateur prend trop de temps à calculer autant de réalisations. (Control + c) dans le "Command Window" de MATLAB, permet d'ailleurs d'arrêter les calculs qui prennent trop de temps.

4. Disposant d'un grand nombre M de marches aléatoires $x_r(t)$ et $y_r(t)$ indépendantes $r = 1, \ldots, M$, nous évaluons la position moyenne et l'écart quadratique pour tout temps $t = 1, \ldots, N$ comme :

$$\langle\, x(t) \,\rangle = \frac{1}{M} \sum_{r=1}^{M} x_r(t) \quad , \quad \langle\, y(t) \,\rangle = \frac{1}{M} \sum_{r=1}^{M} y_r(t) \quad , \quad \langle\, r^2(t) \,\rangle = \frac{1}{M} \sum_{r=1}^{M} x_r^2(t) + y_r^2(t)$$

   Utilisez la commande `mean` de la bonne manière pour calculer ces 3 quantités statistiques.

5. Vérifiez que les positions moyennes $\langle\, x(t) \,\rangle$ et $\langle\, y(t) \,\rangle$ restent proches de l'origine pour tout temps t.

6. Visualisez l'écart quadratique moyenne $\langle\, r^2(t) \,\rangle$ en fonction du temps t.

7. La loi de la diffusion du mouvement brownien nous apprend que l'ensemble des marcheurs aléatoires diffuse en moyenne sur une distance telle que

$$\langle\, r(t)^2 \,\rangle = D\, t$$

   Mesurez le coefficient de diffusion D de notre simulation.

8. Si on diminue la longueur du pas à la valeur $1/2$ au lieu de 1, quelle sera la nouvelle valeur du coefficient D ?

Université Paris-Sud **Année 2010-11**

Master 1 de Physique Appliquée et Mécanique

Outils Informatiques pour la Physique / Matlab

TP 4 : Apparition de caustiques dans un système optique

L'objectif de ce TP est de comprendre la formation de bandes lumineuses, appelées "caustiques", qui apparaissent dans le fond d'une piscine éclairée par le soleil (voir photographie). Dans ce problème, la lumière provenant du soleil est réfractée par la surface de l'eau, dont les vagues se comportent comme des lentilles convergentes ou divergentes. La lumière peut alors s'accumuler dans certaines régions, où la densité d'énergie lumineuse devient infinie.

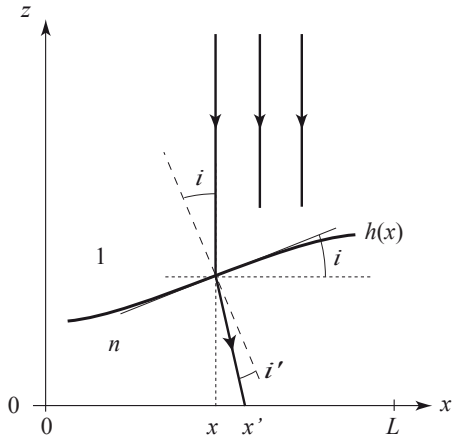

Figure 1: A gauche : Photographie de caustiques au fond d'une piscine. A droite : Refraction d'un rayon lumineux, provenant de $z = \infty$ (milieu d'indice 1), et pénétrant dans l'eau (milieu d'indice n), au passage de l'interface d'équation $z = h(x)$.

Pour simplifier, on ne considère que le cas 1D (voir la figure). On suppose que des rayons lumineux strictement verticaux atteignent la surface de l'eau, décrite par la fonction $z = h(x)$. On va chercher à déterminer l'intensité lumineuse $I(x)$ en $z = 0$ (au fond de la piscine). Pour cela, on va considérer un certain nombre de photons provenant d'en haut, et l'on va calculer leur position d'impact en $z = 0$ grâce à la loi de la réfraction de Descartes. En procédant ainsi sur un grand nombre de photons le long de l'axe x, on va pouvoir reconstruire l'intensité lumineuse $I(x)$ de manière approchée.

Pour un photon provenant d'une abscisse x donnée, l'angle d'incidence par rapport à la surface de l'eau est donné par $i = \tan^{-1}(\partial h/\partial x)$. L'application de la loi de Descartes ($\sin i = n \sin i'$, avec $n = 1.33$ l'indice optique de l'eau et i' l'angle de réfraction) permet de trouver l'expression de l'absisse x' de l'impact de ce photon :

$$x' = x + h(x) \tan \left[i - \sin^{-1} \left(\frac{1}{n} \sin i \right) \right].$$

Dans toute la suite, on étudie le problème sur l'intervalle $x \in [0, L]$, et on considère que la surface de l'eau est décrite par l'onde suivante :

$$h(x) = h_0 + A \cos(2\pi x/\lambda),$$

avec h_0 la profondeur de la piscine, A l'amplitude de l'onde, et λ la longueur d'onde. On fixe les valeurs $h_0 = 1$ m, $L = 3$ m et $\lambda = 1$ m, et l'on ne fera varier que l'amplitude A, dans l'intervalle 0 à 0.2 m.

1. Ecrire une fonction `xp = impact(x,A)` qui, pour un ensemble d'absisses x de photons incidents, renvoie l'ensemble des absisses xp des impacts des photons après refraction. Le second paramètre A est l'amplitude de l'onde. Ne pas utiliser de boucle dans cette fonction.

2. Pour N photons incidents, on pourra choisir `x = linspace(0,L,N)` (photons incidents équirépartis sur l'intervalle $[0, L]$). Vérifier que pour $A = 0$, on a bien $xp = x$ (les photons traversent la surface en ligne droite). Tracer xp en fonction de x pour différentes valeurs de A. Utilisez une valeur de N qui vous semble adaptée pour ces tracés.

3. A partir des valeurs de xp, trouvez une façon "rusée" de déterminer l'intensité lumineuse $I(x)$, en utilisant la fonction de calcul d'histogramme `hist` (consultez l'aide pour cette fonction).

4. Tracer $I(x)$ pour différentes valeurs de A, et mettez en évidence l'apparition de zones très lumineuses sous les crêtes de l'onde lorsque A devient suffisamment grand.

5. On note A_c l'amplitude de l'onde pour laquelle apparaîssent les fameuses "caustiques", définies comme une divergence de l'intensité lumineuse. En remarquant que, lorsque $A > A_c$, ces bandes très lumineuses se dédoublent, en déduire la valeur numérique de A_c (à quelques mm près).

6. Quelle propriété constatez-vous sur le tracé (x, xp) aux endroits où apparaissent les caustiques ?

Université Paris-Sud **Année 2010-11**

Master 1 de Physique Appliquée et Mécanique

Outils Informatiques pour la Physique / Matlab

TP 5 : Equations différentielles

Eléments de base

Equation différentielle ordinaire du 1er ordre

Pour intégrer une équation différentielle du type $y' = f(t, y)$, où $y' = dy/dt$, il faut

- Créer une fonction Matlab dans laquelle est codé $f(t, y)$, par exemple dans un fichier `F.m`. Attention : l'ordre des arguments d'entrée, (t, y), est important. Même si la fonction ne dépend pas de t (système dit "autonome"), la fonction doit quand même avoir ces 2 arguments d'entrée.
- Passer la fonction comme argument à un "solveur" d'équation différentielle, à l'aide de l'opérateur `@`. Le solveur usuel est `ode45` (Runge-Kutta explicite), qui permet d'intégrer la plupart des équations différentielles avec une bonne précision. La syntaxe est :

  ```
  [t, y] = ode45(@F, [t0 tf], y0)
  ```

  où `[t0 tf]` est l'intervalle de t sur lequel y doit être calculé, et `y0 = y(t0)` est la condition initiale. Le solveur renvoie un ensemble de points $y(t_i)$, en choisissant lui-même l'échantillonage (qui n'est pas nécessairement uniforme).
- Pour pouvoir accéder à une valeur $y(t1)$ quelconque, utiliser la syntaxe suivante

  ```
  sol = ode45(@F, [t0 tf], y0)
  y = deval(sol, t1)
  ```

Equation différentielle ordinaire d'ordre plus élevé

Afin de pouvoir utiliser le solveur de matlab pour une équation différentielle d'ordre $n > 1$, il faut se ramener à un système de n équations différentielles d'ordre 1, c'est-à-dire à une équation différentielle d'ordre 1 dont l'inconnu est un vecteur à n composantes. Pour cela, on introduit le vecteur `Y = [y, y', y'' ...]`, et on reformule le problème sous la forme `dY/dt = f(Y)`.

Exercice : Trajectoire d'un pendule

On considère un pendule, constitué d'une masse m suspendue à un fil inextensible de longueur L. On repère la position de la masse par l'angle θ avec la verticale. L'équation du mouvement de ce pendule est

$$\theta'' + \omega_0^2 \sin\theta = 0,$$

où $\omega_0 = \sqrt{g/L}$ est la pulsation propre du pendule et g la gravité. On prendra $L = 1$ et $g = 1$ dans la suite pour simplifier.

Le plus souvent, afin de simplifier le problème, on se contente d'étudier des *petites oscillations* $|\theta| \ll 1$, telles que $\sin\theta \simeq \theta$. Dans ce cas, on obtient l'équation différentielle linéaire classique de l'oscillateur harmonique, $\theta'' + \omega_0^2\theta = 0$, dont la solution analytique est connue, $\theta(t) = \theta_0 \sin(\omega_0 t + \phi)$. Ici, nous allons utiliser une intégration numérique du problème non-linéaire complet (oscillations non petites).

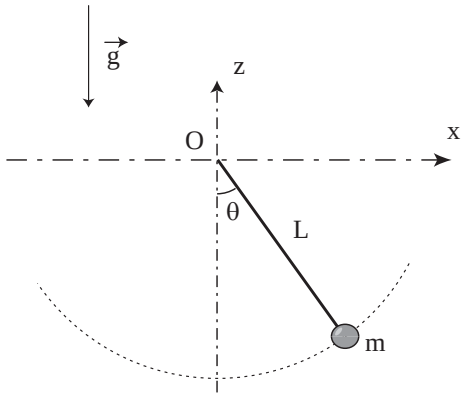

Figure 1:

1. Afin de ramener l'équation différentielle du 2nd ordre à une équation du 1er ordre, on introduit le vecteur à 2 composantes $Y(t) = [\theta, \omega]$ avec $\omega = \theta'$ la vitesse angulaire du pendule. Montrez que l'on peut alors reformuler le problème sous la forme

$$Y' = F(Y),$$

où $F(Y)$ est un vecteur, dont on exprimera les 2 composantes en fonction de celles de Y.

2. Ecrivez une fonction matlab `dYdt = F(t,Y)` qui code cette équation différentielle.

3. Intégrez cette équation différentielle, dans le cas où on lâche le pendule à une position initiale $\theta_0 > 0$ sans vitesse initiale. Tracez $\theta(t)$ et $\omega(t)$ en fonction du temps t dans le cas θ_0 très petit, et retrouver la période attendue pour un oscillateur harmonique.

4. Tracez $\theta(t)$ et $\omega(t)$ dans le cas $\theta_0 \simeq \pi$, et discutez les résultats obtenus.

5. Réalisez une animation de la position du pendule en fonction du temps. Pour cela, vous aurez besoin de connaître la position du pendule à des instants t_i uniformément répartis : utilisez la fonction `deval` (voir plus haut).

6. On introduit une force de frottement proportionnelle à la vitesse (frottement de type "visqueux"), de la forme $f(\theta) = -\alpha\theta'$, avec $\alpha > 0$. Ecrivez une fonction matlab `dYdt = F2(t,Y)` pour inclure ce terme supplémentaire, et observez son effet sur les courbes de $\theta(t)$ et $\omega(t)$.

Bonus : le pendule forcé

On considère pour finir la situation où le pendule est excité par une force extérieure périodique : par exemple, on oscille verticalement le point d'attache du pendule à une pulsation ω_f, différente de la pulsation propre ω_0 a priori. Dans ce cas, on montre que l'équation du mouvement est :

$$\theta'' + \omega_0^2 \sin\theta = A\cos\omega_f t,$$

où A est l'amplitude de l'excitation (on ne considère pas le frottement ici : $\alpha = 0$).

1. Ecrivez une fonction matlab `dYdt = F3(t,Y)` pour inclure ce terme de forçage.

2. On choisit une pulsation de forçage $\omega_f = 2\omega_0/3$. Observez le comportement du pendule, pour une amplitude A allant de 0 à 0.3.

Université Paris-Sud **Année 2010-11**

Master 1 de Physique Appliquée et Mécanique

Outils Informatiques pour la Physique / Matlab

TP 6 : Traitement de signal

Préparation

1. Charger les données contenues dans le fichier 'data' à l' aide de la commande `load` dans le variable au nom `sig1`. Les valeurs dans les deux colonnes représentent des tensions (Volt) mesurées avec deux gauss-mètres dans une expérience d'un métal liquide mis en mouvement sous un champ magnétique.
2. La fréquence d'acquisition de ces mesures est 5 Hz. Construire le vecteur temps `T` associé à ces points de mesure.
3. Afficher les deux signaux en fonction du temps. Le signal est bruité et il y a une dérive linéaire (le signal ne rentre pas parfaitement à la valeur initiale à la fin).

Remise à zéro & dérive linéaire

Nous souhaitons que les signaux commencent et terminent par une valeur proche de zéro. Nous enlevons à la main la dérive linéaire du signal qui est responsable de cette différence. Nous traitons les deux signaux séparément.

1. Afficher le premier signal (première colonne) en fonction du temps.
2. Mesurer deux coordonnées en début et fin du signal à l'aide de la commande `ginput`. Trouvez l'équation de la droite qui relie ces deux points.
3. Calculez les points sur cette droite pour tout instant du vecteur temps `T`. Ceci représente la dérive linéaire du premier signal.
4. Soustraire cette dérive du signal original pour trouver la première colonne du variable `sig2`.
5. Tester le résultat graphiquement.
6. Répétez les opérations pour le deuxième signal (deuxième colonne).

Lissage du signal

Nous voulons maintenant enlever le bruit. L'idée du lissage et de remplacer chaque point du signal `sig2` par une moyenne glissante sur (2n +1) points : le point original (1) et les n derniers et n suivants points avant et après. Les n premiers et n derniers points ne sont pas lissés.

1. Créer une fonction `a=lissage(b,n)`, qui génère b, le signal lissé de a utilisant une telle moyenne glissante. Programmer cette fonction telle que les deux signaux de `sig2` peuvent être traités en même temps.
2. Tester quelques valeurs de $n = 1$ à 100. Afficher les signaux bruts `sig2` et filtrés `sig3` en même temps pour mesurer l'impact du filtre.

Fronts montants & descendants

Partant du premier signal filtré à n=10, nous souhaitons faire une visualisation particulière des données en fonction du type de front. Un point d'un signal fait partie d'un front montant si la valeur du signal au temps précédent est plus petit. Dans le cas inverse on aura un front descendant.

1. Créer une fonction `visufronts(t,s)` qui prend un vecteur temps `t` et un signal `s`.
2. A l'aide d'une boucle sur l'indice du signal et un test simple `if`, réaliser dans cette fonction une courbe (point par point) qui montre un front montant en rouge et un front descendant en bleu, le tout en fonction du temps `t`

Université Paris-Sud **Année 2010-11**

Master 1 de Physique Appliquée et Mécanique

Outils Informatiques pour la Physique / Matlab

Révisions

Exercice 1 (facile) : fonctions

Programmez les fonctions suivantes (et testez-les sur des exemples simples) :

1. `b = renverse(a)` : une fonction qui renvoie le vecteur `b` tel que ses éléments soient ceux de `a` ré-ordonnés à l'envers.
2. `m = geomean(x)` : une fonction qui renvoie la moyenne géométrique des éléments du vecteur `x` (c'est-à-dire $m = (x(1)x(2)..x(N))^{1/N}$, où N est la longueur de x).
3. `Y = polymorceau(X)` : une fonction qui renvoie un vecteur Y dont les éléments sont donnés par x^2 pour chaque élément $x \geq 0$ du vecteur X, et $-x$ pour chaque élément $x < 0$ de X. Tracer le résultat pour $x \in [-1, 1]$.

Exercice 2 (moyen) : boucles et tests

La suite de Collatz (ou de Syracuse) est une suite $(u_n)_n$ définie par son premier élément, u_1, et telle que si u_n est pair, alors $u_{n+1} = u_n/2$, et sinon $u_{n+1} = 3u_n + 1$.

1. Ecrire une fonction `u = collatz(u1, nmax)` qui renvoie les $nmax$ premiers termes de la suite de Collatz de condition intiale $u1$. Remarquez qu'au bout d'un certain n, la suite devient périodique selon la séquence $4 \to 2 \to 1$.
2. Pour une valeur de u_1 donnée, on définit la "durée" de la suite comme l'entier N tel que $u_N = 1$. Ecrire une fonction `N = dureecollatz(u1)`, utilisant la fonction `collatz` précédente, qui renvoie la durée N. On pourra utiliser la boucle conditionnelle `while`.
3. Montrer graphiquement que, pour toute valeur initiale $u_1 \leq 1000$, la suite u_n atteint systématiquement la valeur 1 (c'est-à-dire que N est fini pour tout u_1).

Exercice 3 (difficile) : animation d'une roue

On considère une roue, de rayon $R = 1$, qui roule sans glisser sur un plan horizontal dans la direction des x croissants. Le centre de la roue est donc de coordonnées $(x(t), y(t)) = (Ut, R)$, où $U = 1$ est la vitesse et t le temps.

1. Ecrire une fonction `cercle(x, y, R)` qui trace un cercle de centre (x, y) et de rayon R (sans utiliser de boucle `for`).
2. Réalisez une animation de la roue se déplaçant selon les x croissants.
3. On considère un point P fixé en périphérie de la roue. Ses coordonnées sont $(x_P(t), y_P(t)) = (Ut + \cos(Ut/R), R - \sin(Ut/R))$. Ajouter ce point sur l'animation précédente.