Introduction à Octave

1 Qu'est-ce que Octave ?

Octave est un logiciel de **calcul numérique**, c'est-à-dire qu'il permet d'effectuer des calculs sur des valeurs approchées réelles ou complexes (comme les calculatrices usuelles), à l'inverse de logiciels comme Maple qui permettent de faire du **calcul symbolique** (résolution exacte d'équations, calcul exact d'intégrales, manipulation d'expressions algébriques, etc.).
Octave peut se voir comme une « calculatrice améliorée » spécialisée dans les **calculs sur des matrices**, avec une vaste bibliothèque de fonctions prédéfinies, d'importantes capacités de **représentation graphique** de données, le tout pouvant faire l'objet d'une **programmation**.
Octave est un **logiciel libre gratuit**, téléchargeable à `http://www.octave.org`, disponible sur toutes les plateformes usuelles (Windows, Mac, Linux). Il existe de nombreux programmes similaires à Octave, parmi lesquels on peut citer Matlab (le plus connu, qui est un logiciel propriétaire payant), Scilab (libre gratuit, développé en France par des chercheurs de l'INRIA et l'ENPC) ou R (libre gratuit, spécialisé en statistiques). Du côté des logiciels de calcul symbolique, citons Maple et Mathematica (propriétaires, payants), ainsi que Maxima et Sage (libres, gratuits).
NB. Bien que proche de Matlab, Octave n'en est pas un clone, et les programmes écrits pour Matlab fonctionnent souvent avec Octave et vice-versa, mais il faut parfois faire des modifications.

2 Manipulation du logiciel

NB. L'interface du logiciel (menus, icônes, raccourcis au clavier) dépend de la version installée, mais pas les commandes.

2.1 Fenêtres

Au lancement, Octave présente diverses sous-fenêtres : l'**éditeur**, la **fenêtre de commandes** (ou **console**), la liste des variables et une arborescence de fichiers. Selon la configuration, certains de ces outils sont accessibles en cliquant sur des onglets. Par ailleurs, lorsque l'on demandera des graphiques, ils apparaîtront dans de nouvelles fenêtres de **figures**.

1) Ouvrir l'éditeur et l'aide. Dans l'éditeur, taper "`6*7`", taper F5, choisir où enregistrer le fichier, puis la console affiche le résultat.

2) Dans la console, taper `x=0:0.01:5;plot(x,sin(x))` puis Entrée, pour un exemple de figure (qui sera expliqué après). Copier et coller cette commande dans l'éditeur, et taper à nouveau F5.

Il est souvent pratique de voir simultanément l'éditeur et la fenêtre de commandes. Si les deux font partie du même groupe d'onglets, afficher l'éditeur et cliquer-déplacer à partir du titre "Éditeur" (qui apparaît au-dessus de l'éditeur) vers le haut ou le bas pour l'envoyer vers une sous-fenêtre distincte (relâcher le bouton quand une zone apparaît en couleur).

2.2 Modes d'utilisation

Comme on vient de le voir, il y a deux façons complémentaires d'utiliser Octave :
- par la console, en tapant une commande puis Entrée pour l'exécuter
- par l'éditeur, en tapant une suite de commandes (une par ligne) puis en les faisant exécuter par Octave une par une.

Un inconvénient de la première méthode est de ne pas sauvegarder dans un fichier les commandes lancées (ceci dit, la touche flèche ↑ permet de retrouver les commandes précédentes). On utilisera surtout la console pour tester le fonctionnement d'une commande avant de l'intégrer à un fichier dans l'éditeur, ou pour un calcul très bref qu'on n'a pas besoin de reproduire. Ce sera le cas de tout le début du TP.
Pour la seconde méthode (éditeur), on peut exécuter le fichier en cliquant sur l'icône avec des rouages, ou via la touche **F5**. Octave commence par sauvegarder le fichier, puis l'exécute.
→ Dans le début du TP, on pourra tester les commandes sur la console, puis les copier dans l'éditeur pour sauvegarde.

2.3 La console comme calculatrice : nombres réels et complexes, booléens

— **nombres réels** : `14.54` (utiliser un point et non une virgule), `2.2E-16` ($2{,}2 \cdot 10^{-16}$)
— **opérations** : `+`, `-`, `*`, `/`, `^` (exponentiation, avec les touches AltGr et 9), parenthèses `(` et `)` autour d'expressions
— **fonctions** : `floor` (partie entière), `abs` (valeur absolue), `sqrt` (racine carrée), `exp` (exponentielle), `log` (logarithme népérien), `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `real` (partie réelle), `imag` (partie imaginaire), `conj` (conjugué), `sinh`, etc., en plaçant l'argument entre parenthèses : `atan(1)`
— **constantes** : `i` ou `1i` (`sqrt(-1)`), `pi`, `e`

NB. Tous les calculs sont approchés, il faut donc être vigilant vis-à-vis de possibles erreurs d'arrondis.
La commande `format("long")` affiche 16 chiffres significatifs, **mais** ne change pas la précision des calculs.
La constante `eps` ($2^{-52} \simeq 2 \cdot 10^{-16}$) donne le plus petit nombre ε tel que $1+\varepsilon$ est stocké de la même façon que 1.
La constante `realmin` ($2^{-1022} \simeq 2 \cdot 10^{-308}$) donne le plus petit nombre ε tel que ε est stocké différemment de 0.

3) Calculer $(\sqrt{2})^2 - 2$, $e^{i\pi}$, $(1+\sqrt{2})^{10}$, $3\sqrt[12]{2}\sqrt[7]{5}$, $e^{\pi} - \pi$, $\left(-\frac{1}{2} + i\frac{\sqrt{3}}{2}\right)^3$, `1+eps-1`, `1+eps/2-1`, `0.1+0.2-0.3`

— **constantes booléennes** : `true` (vrai), `false` (faux)
— **opérations** : `&` (et), `|` (ou), `!` (non, obtenu aves les touches AltGr et 2)
— **comparaisons (à valeurs booléennes)** : `<`, `<=` (inférieur ou égal), `>`, `>=`, `==`, `!=` (différent)

4) Tester `2<3`, `1+1==2`, `tan(atan(1))==1`

De plus `true` est assimilé à `1` et `false` à `0` dans les opérations.

5) Que renvoie `(2>1)*9` ?

2.4 Variables

Après chaque calcul, la console affiche "`ans =`" puis le résultat. En fait, `ans` (pour "answer") est une **variable** à laquelle Octave affecte le résultat du dernier calcul, et que l'on peut réutiliser ensuite dans une commande.
NB. La variable `ans` n'est utile que pour les calculs dans la console, pas dans l'éditeur.

6) En utilisant `ans`, obtenir les 10 premières décimales de pi par des opérations : commencer par `10*(pi-3)` (la 1ère décimale est la partie entière), puis `10*(ans-1)` (2è décimale), `10*(ans-4)`, etc.

7) En déduire de même les 10 premiers bits du développement en base 2 de $\pi - 3$.

8) Trouver d'ici 30 secondes une valeur approchée à 10^{-4} d'un réel a tel que $a > 0$ et $a = 2\sin(a)$. Indication : la suite $u_{n+1} = 2\sin(u_n)$ peut converger vers a, pour u_0 bien choisi. Utiliser `ans` pour calculer les termes successifs.

On peut aussi définir ses propres variables pour réutiliser le résultat d'un calcul, par la commande d'**affectation** "`=`".
Par exemple, `a=sqrt(2)` affecte à la variable `a` le résultat de l'opération `sqrt(2)`, et l'on peut ensuite demander `a^2-2`, `b=(a+1)^2` ou même `a=1/a` (qui assigne à `a` le résultat de l'opération `1/a` : attention, dans Octave `=` n'est pas un signe d'égalité mais d'affectation, à la différence de `==`).
En guise de nom de variable, toute suite de caractères est possible (hormis les mots réservés par Octave), par exemple `racine_de_deux=sqrt(2)` ou `r2=sqrt(2)`. On pourrait même poser `sqrt=sqrt(2)` mais la fonction `sqrt` ne serait plus définie ensuite... Cela peut créer des difficultés qui ne se résolvent que par la commande `clear all` qui efface toutes les variables définies par l'utilisateur.

9) Calculer $\frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}-1} + \frac{1}{\sqrt{2}+1}\right)$ en utilisant une variable intermédiaire

3 Erreurs et aide

3.1 Erreurs

En cas d'erreur dans une commande, Octave indique la ligne (dans l'éditeur) et en général la nature de l'erreur.

10) Tester `1/0`, `srqt(3)`, `2*(1+1, 1+2)`, `*1`, `asin(2)`, `tan(pi/2)`

Programme bloqué. Si par erreur on a demandé à Octave un calcul trop long (ou infini), on peut l'interrompre manuellement en tapant Ctrl-C.

3.2 Aide

Octave possède une aide assez complète, en général assortie d'exemples. On y a accès par
- la touche **F1**, le menu, ou l'onglet "Documentation"
- taper `help tan` dans la console si l'on souhaite de l'aide sur la fonction `tan`

De plus, dans la console ou l'éditeur, on peut utiliser la touche tabulation (avec deux flèches) pour **compléter automatiquement** la commande (taper `sq` puis tabulation pour tester).

4 Vecteurs et matrices

Matlab signifie "Matrix Laboratory", et Octave est aussi tout particulièrement conçu pour manipuler des matrices. Comme d'habitude, une **matrice** est un tableau de valeurs réelles ou complexes, et on parle de **vecteur** s'il n'a qu'une ligne, ou une colonne. Pour Octave, toute variable est une matrice, les scalaires étant des matrices à une ligne et une colonne.

4.1 Définir une matrice

De façon explicite, on peut définir

```
A=[ a11, a12, a13;
a21, a22, a23]
```

(le passage à la ligne est facultatif). On retient qu'une **virgule** (ou un **espace**) sépare les colonnes, et un **point-virgule** sépare les lignes. La **matrice vide** est `[]`.
On peut ensuite faire appel à `A(1,1)`, `A(1,2)`, etc. pour accéder et modifier les coefficients. Pour les vecteurs, il suffit d'indiquer la composante significative : `A(5)` au lieu de `A(5,1)` (si c'est un vecteur-colonne) ou `A(1,5)` (vecteur-ligne).
Attention. À la différence du langage C, les indices des matrices (et donc des vecteurs) commencent toujours à 1, et on y accède avec des parenthèses et non des crochets.

11) Définir la matrice $\begin{pmatrix} 1 & i \\ -i & 1 \end{pmatrix}$ et les vecteurs $\begin{pmatrix} 3 & 1 & 5 \end{pmatrix}$ et $\begin{pmatrix} 4 \\ 0 \\ -7 \end{pmatrix}$

Dans le cas de grandes matrices (par exemple des données, en statistiques), on se donne plutôt un fichier texte qui liste les coefficients ligne par ligne, en les séparant par des espaces (ou des tabulations), et on l'importe sous forme de matrice `A` par `A=dlmread('fichier')`. Quelques lignes de texte au début du fichier peuvent décrire la nature des données.

12) Télécharger le fichier `http://www.math.univ-paris13.fr/~tournier/matrice.txt` (avec un navigateur internet) puis le charger dans Octave

On peut aussi définir une matrice en partant d'une fonction qui renvoie une matrice particulière :
- `zeros(m,n)` (matrice nulle), `eye(n,n)` (matrice identité), `ones(m,n)` (matrice constituée de 1), `diag([a1, ..., a_n])` (matrice diagonale). Sous la forme `zeros(A)`, `eye(A)`, `ones(A)` ces matrices sont de même taille que `A`
- pour obtenir des vecteurs lignes équirépartis, `linspace(x1,x2,n)` (de `x1` à `x2` en `n` colonnes) ou `x1:delta:x2` (de `x1`, par accroissements de taille `delta`, sans dépasser `x2`), ou `x1:x2` (idem, avec `delta=1`) ; vu les problèmes d'arrondis, `linspace` est préférable si on veut être sûr du nombre de colonnes

13) Définir les matrices $\begin{pmatrix} 1 & 2 & 3 & \cdots & 15 \end{pmatrix}$, $\begin{pmatrix} 0 & \frac{\pi}{8} & \frac{2\pi}{8} & \cdots & \pi \end{pmatrix}$ et $\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$ (sans taper chaque coefficient)

`size(A)` fournit la taille de `A` (c'est un vecteur : `[lignes, colonnes]`)
Pour un vecteur `v`, `length(v)` fournit sa longueur (c'est un entier).

4.2 Sous-matrices

Si `u` et `v` sont des vecteurs d'entiers, `A(u,v)` est la sous-matrice correspondante. L'exemple essentiel est `A(k1:k2,l1:l2)` pour la sous-matrice donnée par les lignes `k1` à `k2` et les colonnes `l1` à `l2`.
Cas particuliers : "`end`" désigne le plus grand indice (par exemple, `A(2:end,1:2)`), et "`:`" désigne tous les indices (par exemple, `A(:,2)` donne la deuxième colonne).

14) Demander la taille de la matrice précédente, et extraire sa première ligne et sa dernière colonne

On peut changer toute une sous-matrice : `A(2:3,1:3)=zeros(2,3)`
voire si possible la supprimer (modifiant la dimension) : `A(:,2)=[]` supprime la deuxième colonne.

4.3 Opérations

On peut concaténer (« accoler ») des matrices : `[A, B]` place `A` et `B` côte-à-côte et `[A ; B]` place `A` et `B` l'une par-dessus l'autre (si les dimensions le permettent).
Par exemple, pour ajouter un élément à la fin du vecteur-ligne `v` : `v=[v, 42]`
De plus,

— `A'` est la transposée de `A`

15) Tester avec une matrice complexe : est-ce juste la transposée ?

16) Calculer le produit scalaire des vecteurs lignes `x=1:10` et `y=10:-1:1`

— `*`, `^`, `+` sont les opérations matricielles usuelles sur les matrices et s'appliquent aussi entre matrices et scalaires

17) Que donne l'opération `A+5` où `A` est une matrice ? Que donne `2^A` où `A` est une matrice ? Et `A^-1` si `A` est carrée ?

— `.*`, `.^`, `./` sont des opérations coefficient-par-coefficient

18) Pour une matrice `A` inversible (`A=ones(3,3)+diag([0,1,2])`), comparer `A^-1` et `A.^-1`

19) Définir les vecteurs $(n^2)_{1\leq n\leq 10}$ et $(\frac{n}{1+n^2+n^3})_{1\leq n\leq 10}$ (commencer par poser `vn=1:10` et faire des calculs sur `vn`)

— `<`, `>`, `<=`, etc., s'appliquent coefficient-par-coefficient entre matrices ou avec un scalaire

20) Poser `v=rand(1,10)` et regarder le résultat de `v<0.5` Comprendre ce que fait `v(v<0.5)`

La plupart des fonctions sur les réels (ou complexes) s'appliquent aussi coefficient-par-coefficient : `sin(A)`, `exp(A)` (ce n'est pas l'exponentielle matricielle `expm`), `sqrt(A)` (ce n'est pas la racine matricielle `sqrtm`), etc.

21) Définir les vecteurs lignes $(\sin\frac{k\pi}{10})_{0\leq k\leq 10}$, $(\exp(k^2)-k)_{1\leq k\leq 10}$ et $((k+5)^k)_{1\leq k\leq 10}$ (commencer par poser `vk=1:10` et faire des opérations à partir de `vk`)

— `sum(A)` est la somme des coefficients de `A`
— `sum(A,1)` est la somme selon la première coordonnée, c'est un vecteur-ligne donnant la somme de chaque colonne
— `sum(A,2)` est de même un vecteur-colonne donnant la somme de chaque ligne
— `mean(A)`, `variance(A)` donnent la moyenne et la variance des valeurs de `A`
— `cumsum(A)` calcule les sommes cumulées (suite des sommes partielles) et `cumsum(A,1)` ou `cumsum(A,2)` effectue cette opération colonne-par-colonne ou ligne-par-ligne.
— `prod(A)` est le produit des coefficients de `A` (avec les mêmes variations que `sum`)

22) Calculer $\sum_{k=1}^{N} \frac{1}{k^2}$ et $\prod_{k=2}^{N} \left(1 - \frac{1}{k^2}\right)$ où $N = 1000$

— si `v` est un vecteur de booléens (obtenu avec un test, cf. 19), `find(v)` renvoie les indices où `v` est égal à `%T`.

23) Quels éléments de la suite $(\sin(n))_{1 \leq n \leq N}$ sont-ils positifs ? (choisir une valeur de N, par exemple `N=100`) Combien y en a-t-il ? Définir la suite restreinte à ces indices. Et combien de termes appartiennent à $[0, \frac{1}{2}]$?

— `det(A)`, `kernel(A)`, `[P,D]=spec(A)` donnent le déterminant, une base du noyau (en colonnes) et le spectre (dans `D`) ainsi que des vecteurs propres (dans `P`, dans le même ordre) de la matrice carrée `A`
— `linsolve(A,b)` renvoie une solution de $AX + b = 0$, s'il y en a une (même si `A` est singulière, ou non carrée)

24) Résoudre $\begin{cases} x_1 + x_2 + x_3 &= 3 \\ x_1 - x_2 + x_3 &= 1 \\ x_1 - 3x_2 + x_3 &= -1 \end{cases}$

5 Représentations graphiques

Pour afficher la valeur d'une variable dans la console on utilise `disp(x)` ("display"). Par ailleurs, les résultats des lignes non terminées par `;` s'affichent dans la console.

25) Tester `x=5;disp('x =');disp(x);`

Pour afficher de nombreuses valeurs, il est souvent préférable de les représenter graphiquement.

5.1 Graphes avec "plot"

Partant de deux vecteurs `vx` et `vy` de même longueur, `plot(vx,vy)` représente les points `(vx(i),vy(i))` et les relie dans l'ordre d'apparition.

```
vx=linspace(0,1,100); plot(vx,sqrt(vx))
```

L'échelle s'adapte automatiquement aux valeurs, et la couleur est choisie dans une liste (elle change cycliquement).
La commande `figure(5)` ouvre la fenêtre "Figure 5" et le prochain graphe y sera représenté.
On utilise `hold on` pour activer la superposition des graphes suivants ; et `clf` pour vider la fenêtre.

26) Représenter les graphes de sin et cos sur $[0, \pi]$ (avec 100 points, par exemple)

27) Représenter le graphe de $\frac{\sin x}{x}$ sur $[0, 100]$, dans une autre fenêtre

On modifie le rendu via un troisième paramètre : par exemple `plot(vx,vy,'+r')` ou `plot(vx,vy,'g')`. Description :
— les caractères `.+xo-` changent le type de ligne (points, croix, croix, rond, trait continu)
— les caractères `kbrgcym` changent la couleur (black, blue, red, green, cyan, yellow, magenta)

28) Représenter en points bleus les 30 premiers termes de la suite $u_n = \sum_{k=1}^{n} \frac{(-1)^{k+1}}{k}$ (penser à `cumsum`) et en rouge une droite correspondant à sa limite $\ln 2$

On peut ajouter titre, noms des axes et légende (s'il y a plusieurs courbes) :

— `title('titre')`, `xlabel('axe X')`, `ylabel('axe Y')` donnent un titre au graphe et aux axes
— `legend('courbe 1', 'courbe 2', 'courbe 3'...)` donne des légendes pour chaque courbe (dans l'ordre de dessin), et 2 indique la position (2 pour en haut à gauche, 1 pour en haut à droite,...)

29) Représenter les fonctions $x \mapsto x^k$ sur $[0,2]$ pour $k \in \{0.5, 1, 2, 3, 4\}$ et ajouter titre et légendes (après la section suivante, on pourra faire une boucle pour automatiser la répétition des commandes)

Pour modifier la plage de valeurs représentée, on peut utiliser :
— `xlim([xmin,xmax])`, `ylim([ymin,ymax])` (attention aux crochets : le paramètre est un vecteur)
— `axis equal` pour rendre les axes orthonormés

30) Dessiner un cercle de centre $(0,0)$ et de rayon 2 : par équation cartésienne, et aussi par équation paramétrique

31) Représenter la courbe d'équation polaire $r = \sin^2 \frac{\theta}{2}$ (toujours avec la fonction `plot`)

Les fonctions `loglog`, `semilogx` et `semilogy` fonctionnent comme `plot` mais utilisent une échelle logarithmique sur les 2 axes, l'axe des abscisses, et l'axe des ordonnées respectivement.

32) Représenter $u_n = \left| e - \left(1 + \frac{1}{n}\right)^n \right|$ pour $n = 1, .., 1000$ sur une échelle linéaire, puis sur une échelle logarithmique en ordonnée.

33) Représenter $f : x \mapsto \left(1 + \frac{1}{x}\right)^x$ pour 100 valeurs de x entre 1 et 10^{16}. Choisir les valeurs équiréparties sur une échelle linéaire (et utiliser `plot`), puis choisir les valeurs équiréparties sur une échelle logarithmique, c'est-à-dire que les valeurs de $\log x$ sont équiréparties (et utiliser `semilogx`). Commenter les graphes.

5.2 Gestion de la fenêtre graphique

Pour afficher plusieurs graphes simultanément, on utilise en général plusieurs fenêtres.
On commence par choisir une fenêtre par `figure(1)` ("select current figure") pour ouvrir la fenêtre 1 par exemple, puis `figure(2)`, etc.
Toutes les opérations (y compris `clf`) se rapportent alors la figure courante.

```
vx=linspace(0,1,1000);
figure(1);clf;hold on;
plot(vx,1./(1+vx.^2));
plot(vx,1./(1+vx.^3));
figure(2);clf;
plot(vx,vx.^2);
```

34) Tester cet exemple

On peut aussi afficher plusieurs graphes dans la même fenêtre : `subplot(m,n,k)` découpe la fenêtre courante en `m` lignes et `n` colonnes, et définit comme fenêtre courante la `k`-ième (dans le sens de lecture). Par exemple,

```
vx=linspace(0,2*pi,100);
subplot(2,1,1);
plot(vx,sin(vx));
subplot(2,1,2);
plot(vx,cos(vx));
```

35) Tester cet exemple, puis ajouter en-dessous le graphe de $\sin + \cos$ sur $[0, 2\pi]$

6 Bases de programmation

Dorénavant, on utilisera l'éditeur et non la console : on aura souvent besoin d'écrire plusieurs lignes. On pourra également sauvegarder dans plusieurs fichiers afin d'éviter de relancer toutes les instructions précédentes à chaque fois.

6.1 Séquence de commandes

En écrivant une commande par ligne dans l'éditeur, celles-ci seront exécutées l'une après l'autre par Octave en pressant **F5**. C'est la forme la plus simple de programme : une suite de commandes.

```
commande1
commande2
...
```

Il est possible de mettre plusieurs commandes sur une ligne, en les séparant par des points-virgules `;`.

6.2 Commentaires

On peut (et même on *doit*) compléter son programme par des commentaires, c'est-à-dire du texte explicatif. Pour qu'il ne soit pas considéré comme une commande, on le précède de `#`

```
# Commentaire
commande1 # commentaire sur la commande
commande2
```

6.3 Branchement conditionnel

Octave (comme tout language de programmation) permet aussi d'exécuter certaines commandes uniquement si une condition est réalisée, en utilisant la structure suivante :

```
if(condition)
    # Si la condition est vraie, les commandes suivantes sont exécutées
    commande_oui_1
    commande_oui_2
    ...
else
    # Si la condition est fausse, les commandes suivantes sont exécutées
    commande_non_1
    commande_non_2
    ...
end
```

où `condition` est un booléen (par exemple, `a<0`) et les points de suspension remplacent une ou plusieurs commandes.
Si la condition est vraie, alors le premier bloc de commandes est exécuté, sinon le deuxième est exécuté.
La partie `else ...` est optionnelle.
L'indentation des blocs de commandes n'est pas indispensable, mais facilite la relecture du programme.

6.4 Boucle "for"

Pour faire répéter à Octave une suite de commandes plusieurs fois, on utilise la structure suivante :

```
for compteur=debut:fin
    # suite de commandes à répéter pour compteur = debut, debut+1,...,fin :
    commande_1
    ...
end
```

où `compteur` est un nom de variable, et `debut` et `fin` sont deux entiers (par exemple, `for i=1:10`). Ceci équivaut à :

```
compteur=debut
commande_1
...
compteur=debut+1
commande_1
...
(...)
```

```
compteur=fin
commande_1
...
```

Autrement dit le bloc de commandes est exécuté successivement avec `variable` égal à `debut` puis à `debut+1`, etc., jusqu'à `fin`.

36) Écrire un programme qui calcule avec une boucle "for" la somme et le produit des carrés des entiers de 1 à `N` (où `N` est une variable définie en début de programme). On pourra commencer par :

```
N=15
S=0 # somme initiale
P=1 # produit initial
for ...
```

37) À l'aide de deux boucles "for" imbriquées, et d'une structure "if...then...", écrire un programme (bête) qui compte le nombre de paires (i, j) d'entiers positifs telles que $i^2 + j^2 < 1000$.

38) On définit la suite $(S_n)_{n\geq 0}$ par $S_n = \int_0^\pi \left(\frac{x}{\pi}\right)^{2n} \sin x \mathrm{d}x$. Justifier que la suite est positive et converge vers 0. Montrer que $S_0 = 2$ et que, pour tout n, $S_n = 1 - \frac{2n(2n-1)}{\pi^2} S_{n-1}$. Calculer $S_0, \ldots, S_{16}$ et observer les valeurs : commenter.

En fait, on peut définir plus généralement la structure "`for compteur=vecteur`" qui exécute la suite de commandes avec `compteur` successivement égal à chaque coefficient du vecteur **ligne** `vecteur`.

39) Représenter graphiquement les fonctions $x \mapsto |x|^\alpha$ sur $[-1, 1]$ pour 10 valeurs de α régulièrement espacées dans $[0, 2]$. Indication : commencer par "`for alpha=linspace(0,2,10)`"

6.5 Boucle "while"

Enfin, on peut faire répéter à Octave une séquence de commandes tant qu'une expression reste vraie :

```
while(condition)
    # Bloc de commandes répétées tant que condition est vraie
    commande_1
    ...
end
```

Attention, ce type de boucle peut tourner indéfiniment si la condition n'est jamais réalisée.

40) Tester `while(1<2);end` (utiliser `ctrl-C` pour interrompre)

41) Écrire un programme qui donne le nombre de termes de la suite de terme général $u_n = \sum_{k=1}^n \frac{1}{k}$ qui sont inférieurs à `M` (où `M` est une variable définie en début de programme, par exemple par `M=4`)

42) Écrire un programme qui calcule les termes de la suite définie par $u_0 = 1$ et $u_{n+1} = \cos(u_n)$ jusqu'à ce que deux termes successifs diffèrent de moins de `delta`, où `delta` est fixé en début de programme.

6.6 Fonctions

En plus des fonctions prédéfinies, on peut en écrire de nouvelles pour y faire appel ensuite. Pour définir une fonction `fonct` qui dépend de deux paramètres, il faut **ouvrir un nouveau fichier**, écrire

```
function a=fonct(x1,x2)
    # suite de commandes qui calculent la fonction et placent sa valeur dans a
    commande1 # commande qui peut dépendre de x1 et x2
    ...
    a=...
endfunction
```

et sauvegarder sous le nom `fonct.m` (en adaptant nom de la fonction) : chaque fonction est dans un fichier différent ! On pourra ensuite appeler par exemple `fonct(2,0.23)` (si les paramètres sont supposés scalaires). Cette expression sera égale à la valeur de `a` à la fin de l'exécution du bloc de commandes ci-dessus avec `x1` et `x2` égales à `2` et `0.23`.
Attention.

- Dans le programme ci-dessus, `a`, `x1` et `x2` sont des « variables muettes » : elles n'ont d'existence que pour l'écriture de la fonction, et n'apparaissent que dans le bloc de commandes de la fonction
- Plus généralement, toute variable utilisée dans une fonction est **locale**, c'est-à-dire qu'elle n'existe qu'au sein de la fonction. Ceci est une sécurité : une fonction doit pouvoir être réutilisée dans un programme différent, donc ne doit pas modifier les variables qui y sont définies. Si on veut utiliser la valeur d'une variable dans la fonction, il faut donc en faire un paramètre supplémentaire. *On pourrait aussi utiliser des variables globales.*
- pour faire appel à une fonction, il faut que le fichier qui la contient se trouve dans le répertoire courant (celui que l'on voit dans la fenêtre Octave avec l'arborescence de fichiers, que l'on obtient aussi par `pwd`).

43) Définir la fonction $f : (x, y) \mapsto \max(x, y)$ (en fait la fonction `max(v)` existe déjà, pour un vecteur `v`)

44) Définir la fonction `binaire(x,n)` qui renvoie le vecteur-ligne des `n` premiers chiffres binaires du développement du réel `x` compris entre 0 et 1.

NB. Il est possible à une fonction de s'appeler *elle-même*, on parle de définition **récursive**. Attention aux boucles infinies dans ce cas : dans certains cas (selon les paramètres), la fonction doit être définie sans faire appel à elle-même, et tous les autres cas finissent par se ramener à ceux-ci.

45) Définir la fonction factorielle de façon récursive : selon le schéma $0! = 1$ et $n! = n \cdot (n-1)!$
Calculer 15! et évaluer l'erreur (en pourcentage) par rapport à la formule de Stirling.

46) Définir la fonction binôme de façon récursive : selon le schéma $\binom{n}{0} = 1$ et $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$ (ou 0 si $k > n$).

Une fonction peut aussi renvoyer plusieurs valeurs. On définit alors `function [a,b,...]=fn(x1,x2,...)`, et pour enregistrer le résultat de la fonction dans des variables on pose `[a,b]=fn(x,y,z)` par exemple.

47) Définir une fonction qui prend `a,b,c` pour arguments et renvoie les deux racines (éventuellement confondues) du polynôme $\mathtt{a}X^2 + \mathtt{b}X + \mathtt{c}$

6.7 Quelques conseils

Afin d'écrire des programmes plus simples à relire (et donc aussi moins sujets à erreur),

- bien choisir ses noms de variables et de fonctions (sans les rendre trop longs)
- utiliser des variables plutôt que des valeurs numériques : il sera plus facile de les modifier, et leur nom peut indiquer leur signification
- indenter boucles et branchements
- commenter les programme (pour soi et pour les autres) : rôles des variables et des fonctions, explications éventuelles
- définir des fonctions dès qu'un morceau de programme prend plusieurs lignes et ressert plusieurs fois (en faisant varier certains paramètres)