Première partie

Exercices d'introduction et d'application

Session A

Premières notions

Objectifs

- **Utilisation de la ligne de commande.**
  - **Expressions et instructions**
  - **Commandes** `help`, `lookfor`, `whos`, `who`, `format`
- **Utilisation de l'éditeur.**
  - **Scripts et fonctions**
  - **Exemples**

MATLAB est un environnement performant de programmation scientifique et de génération de graphisme. Il peut travailler en deux modes distincts : immédiat ou différé (interpréteur ou compilateur).

Avant d'effectuer les exercices de cette session, lors d'un tout premier contact avec* MATLAB*, il est recommandé de lire l'annexe b (p222).

A.1 Ligne de commande : mode immédiat

MATLAB est avant tout un interpréteur, c'est dire qu'il propose à l'utilisateur une invite (prompt) à laquelle on peut répondre par une commande.

Les commandes peuvent être de deux sortes : expression ou instruction.

L'invite est toujours donnée par un double chevron qui ne doit pas être tapé.

Exercice A.1

Tapez à l'invite les expressions suivantes, suivies de la touche entrée :

```
» 2*3                » 2*3;
» pi                 » sin(pi/2)
» [1 4; 6 8]         » 6:2:20
» 6:-2:-3            » b
```

Une expression est en quelque sorte une formule permettant de calculer un résultat. Pour qu'une formule soit correcte, il faut qu'au moment de son évaluation par l'interpréteur, tous ses éléments soient connus. C'est pour cela que la dernière formule produit une erreur.

Si ces exemples ne vous semblent pas clairs, vous pouvez dans l'ordre : lire ce qui suit, vous reporter à la seconde partie (en passant s'il le faut par l'index), ou vous aider de la commande `help` (par : `help punct` ou `help sin`), ou de l'aide html (commande `doc`, ou menu d'aide), enfin de la commande `lookfor` (par `lookfor sin`).

Les choses principales à noter ici, sont que :

1. Sauf pour le deuxième et le dernier exemple, toutes les lignes vous répondent `ans =` suivi du résultat du calcul sur la ligne suivante. La *variable* `ans` est créée automatiquement par MATLAB et contient la valeur de l'expression courante.
2. La touche du clavier représentant une flèche vers le haut vous permet de circuler rapidement à travers toutes vos frappes précédentes, puis de les éditer (corriger) avec les flèches gauches et droites.

   Essayez également la flèche vers le haut après avoir tapé les premières lettres de la commande que vous désirez modifier...
3. La commande `format` vous permet de changer facilement l'apparence des résultats (tapez `format long` et utilisez la remarque précédente pour revoir rapidement vos résultats.)
4. La troisième expression est une variable prédéfinie en MATLAB qui vaut la constante π, il en existe d'autre comme `eps`.
5. La quatrième expression utilise la fonction sinus qui est une des très nombreuses fonctions définies en MATLAB. Vous pouvez accéder à la liste de ces fonctions par l'aide du menu de MATLAB, l'index du présent ouvrage ou son annexe a (p196).
6. Les deux dernières expressions construisent des matrices et des vecteurs qui sont à la base des fonctionnalités de MATLAB.

Les instructions sont soit des instructions structurées qui ne sont pas pratiques à utiliser dans les lignes de commandes (Cf. deuxième partie, section 2.2, p86), soit l'affectation.

L'affectation donne des valeurs à des variables qu'elle crée ou modifie.
Elle est notée par le signe =[1]

Exercice A.2

Tapez à l'invite les expressions suivantes, suivies de la touche entrée :

```
» a = 2*3                    » b = 2*3;
» zz = pi                    » pipo = sin(pi/2)
» zorro = [1 4; 6 8]         » f = 6:2:20
» g = 6:-2:-3                » b
```

La grosse différence est ici que la réponse (sauf pour le deuxième exemple) est donnée par le nom de la variable (une lettre (seule ou) suivie de lettres, de chiffres ou de soulignés) suivi du signe d'affectation = et de la réponse. De plus si l'on retape le nom de la variable à l'invite, elle contient encore la valeur précédemment entrée et qui lui a été *affectée*.

[1] ni := comme en Pascal ni == qui est utilisé pour les tests.

1. Les commandes `who` et `whos` permettent de connaître les variables que l'on a affectées.
2. La commande `clear` permet de supprimer des variables
   (essayez `who ; ↵ clear all ↵ who ; ↵`).
3. Une variable qui n'a pas été affectée ou qui a subit un `clear` n'existe pas. Essayez `zorro ↵` après avoir tapé les instructions précédentes, puis après avoir exécuté un `clear zorro`.
4. Le `;` est simplement un *inhibiteur d'affichage*. Si la seconde instruction n'affiche pas `b`, la dernière expression le fait.

Exercice A.3

1. *A quoi sert la fonction `lookfor` ?*
2. *Est-il possible d'appliquer la fonction `sin` au tableau `zorro`. Quel est le résultat ? (Attention `sin(zorro)` n'est pas `sin zorro`).*
3. *Est-il possible de calculer `zorro-1`, `zorro+zorro`, `2*zorro`, `zorro+f`, `zorro*zorro` ?*
   Analyser les résultats.

A.2 Utilisation de l'éditeur

La ligne de commande est pratique pour utiliser MATLAB comme une calculette. Mais si on désire rejouer plusieurs fois la même séquence de commandes, il vaut mieux utiliser l'éditeur/dévermineur intégré. Il faut noter que la version 7 fournit la notion de « *short cuts* » dont nous ne parlerons pas, Cf. cependant l'annexe b (p222).

A l'aide de l'éditeur on peut écrire des scripts et des fonctions (Cf. chapitre 2, p86) ou entrer des données.

On accède à l'éditeur par le menu fichier, ou les icones « nouveau fichier » ou « rappeler ancien fichier ». (Cf. aussi l'appendice b.2 (p224)).

On peut également l'invoquer sur la ligne de commande de MATLAB par `edit` suivi du (chemin) nom du fichier à éditer.

Les fichiers contenant un script ou une fonction **doivent avoir l'extension** `.m` pour pouvoir être utilisés par la suite sur la ligne de commande, leur noms doivent être formés d'une lettre (non accentuée) suivie de lettres (non accentuées), de chiffres ou de soulignés. Tout autre caractère pourra être cause d'ennuis dans la suite de l'utilisation du fichier.

Une autre source fréquente d'ennuis est la collision de noms : il faut éviter de donner à un script ou à une fonction le nom d'une fonction système, et il faut éviter de donner à une variable le nom d'une fonction.

Donnons deux exemples très simples dans une session.

Invoquez l'éditeur par un des moyens suggérés plus haut et tapez dans un nouveau fichier :

```
disp('bonjour');
```

Sauvez le fichier sous le nom `essai` (l'éditeur rajoutera automatiquement l'extension `.m`. Utilisez la combinaison de touches `Alt-Tab` (ou cliquez MATLAB sur la barre des tâches) pour passer de l'éditeur à la fenêtre de commande MATLAB et tapez `essai` suivi de ↵.

Le message `bonjour` apparaîtra à l'écran.

Attention : le fichier `essai.m` doit être accessible par MATLAB*, c'est à dire qu'il faut le sauver sous le répertoire proposé par* MATLAB*, où émettre, avant toute tentative d'exécution*

la commande `cd <rep>` *(où* `<rep>` *représente le répertoire de sauvegarde) sur la ligne de commande de* MATLAB.

Retournez sous l'éditeur, ouvrez un nouveau fichier vide. Tapez et collez à partir du fichier précédent en sorte d'avoir :

```
function essai1(nom)
disp(['bonjour ' nom ' !']);
```

Sauvez (la fonction de sauvetage proposera `essai1` pour nom) et exécutez de la ligne de commande `essai1('jean-thierry')` ou `essai1 jean-thierry` pour voir apparaître

```
bonjour jean-thierry !
```

à l'écran.

Exercice A.4
Pour chaque exercice les nombres entre crochets désignent les chapitres pertinents de la seconde partie.

1. *Écrivez une fonction calculant le signe d'un nombre (*-1, 0 *ou* 1 *suivant que le nombre est négatif, nul ou positif). La structure de contrôle* `if...elseif..else...end` *peut être utile ici (!). [1 (p80),2 (p86)].*
2. *A l'aide de l'opérateur «:» écrivez une fonction de trois paramètres* `a`, `b`, `n`, *qui calcule un vecteur à* `n` *composantes également réparties entre* `a` *et* `b`. *[2 (p86), 3 (p98)].*
3. *Tapez dans l'éditeur le script de nom* `glob.m` *contenant la ligne unique :*

   ```
   a = 2512
   ```

   Exécutez-le. Tapez ensuite sur la ligne de commande de MATLAB `a` ↵. *Tapez dans l'éditeur la fonction de nom* `loc.m` *contenant les deux lignes :*

   ```
   function loc
   a = 0
   ```

   Exécutez-la. Tapez ensuite sur la ligne de commande de MATLAB `a` ↵. *Que remarquez-vous ? Expliquez. [2 (p86)].*
4. *Résolvez les équations du premier et second degré avec deux fonctions d'entêtes respectives :* `function x = equ1(a, b)` *et* `function x = equ2(a, b, c)` *en réfléchissant aux problèmes suivants [2 (p86)] :*
   - *Quel est le domaine de validité de vos fonctions.*
   - *Que doit être* `x` *? Un paramètre de sortie unique est il suffisant ?*
   - *Peut-on faire en sorte que l'appel* `equ2(a,b)` *soit valide et équivalent à* `equ1(a,b)`.

Lors de l'écriture d'une fonction, il faut toujours se poser les deux questions suivantes :
- Les paramètres d'entrée sont-ils suffisants (et nécessaires) pour effectuer le calcul ?
- Les paramètres de sortie permettent-ils à l'utilisateur de la fonction de prendre toute décision relative à la poursuite du programme ?

A.3 Solutions des exercices de la Session A

Solution de l'exercice A.3

Δ

1. A quoi sert la fonction `lookfor` ?
   Elle permet de retrouver toutes les fonctions qui sur les chemins d'accès MATLAB *contiennent une chaîne de caractères donnée sur leur première ligne de commentaires*
2. Est-il possible d'appliquer la fonction `sin` au tableau `zorro`. Quel est le résultat ?
   Oui. Le résultat est un tableau de même taille contenant les sinus de chacun des éléments.
3. Est-il possible de calculer `zorro-1`, `zorro+zorro`, `2*zorro`, `zorro+f`, `zorro*zorro` ? Analyser les résultats.
   Oui, oui, oui, non (dimensions incompatibles), oui.

∇

Solution de l'exercice A.4

Δ

1. Écrivez une fonction calculant le signe d'un nombre (-1, 0 ou 1 suivant que le nombre est négatif, nul ou positif). La structure de contrôle `if...else...end` peut être utile ici (!).

```
function s = sgn(a)

if a > 0
  s = 1;
elseif a < 0
  s = -1;
else
  s = 0;
end;
```

2. A l'aide de l'opérateur «:» écrivez une fonction de trois paramètres `a`, `b`, `n`, qui calcule un vecteur à `n` composantes également réparties entre `a` et `b`.

```
function v = linearspace(a, b, n)

if n < 2
  error('n doit être entier > 1');
end;
h = (b-a)/(n-1);
v = a:h:b
```

3. Tapez dans l'éditeur le script de nom `glob.m` contenant la ligne unique :

   ```
   a = 2512
   ```

   Exécutez-le. Tapez ensuite sur la ligne de commande de MATLAB `a` ↵.
   Tapez dans l'éditeur la fonction de nom `loc.m` contenant les deux lignes :

   ```
   function loc
   a = 0
   ```

   Exécutez-la. Tapez ensuite sur la ligne de commande de MATLAB `a` ↵.
   Que remarquez-vous? Expliquez.
   les deux valeurs de a sont identiques et égales à 2512. *Le* `a = 0` *de la fonction ne modifie pas la valeur de la variable du script.*
4. Résolvez les équations du premier et second degré avec deux fonctions d'en-têtes respectives : `function x = equ1(a, b)` et `function x = equ2(a, b, c)` en réfléchissant aux problèmes suivants :

- Quel est le domaine de validité de vos fonctions.
- Que doit être `x` ? Un paramètre de sortie unique est il suffisant ?
- Peut-on faire en sorte que l'appel `equ2(a,b)` soit valide et équivalent à `equ1(a,b)`.

```
function x  = equ2(a, b, c)

if nargin == 2
  x = equ1(a, b);
elseif a == 0;
   x = equ1(b, c);
else
   delta = b*b-4*a*c;
   if delta >= 0
      if b > 0
         x = (-b-sqrt(delta))/(2*a);
      else
         x = (-b+sqrt(delta))/(2*a);
      end;
      if x == 0
         x = [x 0];
      else
         x = [x (c/a)/x];
      end;
   end;
end;
```

```
function x  = equ1(a, b)

if a == 0
   if b ~= 0
     x = [];
   else
     x = NaN;
   end;
else
   x = -b/a;
end;
```

Ces fonctions sont valides pour les réels « machine » sous réserve de non dépassement de capacité au cours du calcul.
Un paramètre est suffisant, il contient le vecteur des solutions. En cas d'indétermination le résultat est `NaN` (Cf. 4.3.3 (p114)).
La dernière condition est remplie à l'aide de l'usage de la fonction `nargin` (Cf. 2.3.1 (p92)).

∇

Session B

Boucles itératives et vectorisation

Objectifs

- **Boucle itérative (`for`) versus vectorisation**
  - **Exemples : somme des puissances p$^{\text{ième}}$ des entiers**
  - **Commandes** `ones`, `zeros`, `rand`, `sum`, `prod`
  - **Outils de vectorisation**
  - **Commandes** `plot`, `linspace`, `meshgrid`, `surf`

B.1 Boucles itératives

Une boucle itérative est une boucle pour laquelle on sait au départ combien de fois l'exécution doit être faite. Il existe deux manières en MATLAB d'effectuer de telles boucles.

- **La boucle `for`** (Cf. deuxième partie, section 3.3, p103) qui est un moyen explicite de répéter un nombre de fois fixé à l'avance une instruction structurée.
- **La vectorisation**, qui consiste à réunir dans un vecteur les éléments sur lesquels on veut effectuer le même traitement et effectuer ce traitement directement.

Par exemple : Calculer le sinus des entiers entre 1 et 100

Boucle `for`

```
for i=1:100;
  y(i)=sin(i);
end;
```

Vectorisation

```
x = 1:100;
y = sin(x);
```

Comment choisir ?
On peut déjà énoncer :

> **Si la vectorisation est possible, alors elle est préférable.**

De plus :

> **La vectorisation est possible (au moins partiellement) si les opérations de la boucle ne dépendent pas de l'ordre dans lequel elles sont effectuées.**

Nous allons écrire une fonction calculant la somme des puissances p des n premiers entiers en passant de la méthode traditionnelle à celle vectorisée.

Méthode traditionnelle L'algorithme est simple, basé sur l'assertion : *si n est nul la somme vaut zéro, si l'on connaît la somme pour $n-1$, celle pour n s'obtient en ajoutant n^p.* La boucle `for` servira à rajouter itérativement ces valeurs à une variable contenant 0 au départ.

```
function s = sump(n, p)

s = 0;
for i = 1:n
  s = s + i^p;
end;
```

Il faut remarquer que même ici une certaine vectorisation est utilisée du fait de la nature même de la boucle `for` de MATLAB : `for i = 1:n` signifie que l'on va donner à `i` les valeurs successives des colonnes du vecteur ligne `1:n`, colonnes qui contiennent exactement les nombres de 1 à `n`.

Exercice B.1

1. *Combien de fois ces boucles sont-elles évaluées ? Quelles sont les valeurs de `i` écrites ?*
   Expliquer [3.3 (p103)].
   (a) `for i=12:-2:1 ; disp(i), end ;`
   (b) `for i=12:-2:1 ; disp(i), i = 0 ; end ;`
   (c) `for i=12:-2:1 ; disp(i) ; if i == 8, break ; end ; end ;`
2. *Une suite de Fibonacci est définie par ses deux premières valeurs $f_1 = a$ et $f_2 = b$ et par la règle $f_{n+2} = f_n + f_{n+1}$. Écrivez une fonction calculant le $n^{\text{ième}}$ terme de la suite.*
3. *Écrivez une fonction calculant les n premières lignes du triangle de Pascal. On pourra utiliser les relations : $C_n^p = C_{n-1}^{p-1} + C_{n-1}^p$ et $C_n^1 = C_n^n = 1$.*

Méthode vectorisée Calculons le vecteur des entiers entre 1 et n à la puissance p et calculons sa somme. Ce qui donne :

```
function s = sumpv(n,p)

s = sum((1:n).^p);
```

L'écriture est plus concise et le temps d'exécution 10 fois meilleur.

Comme précédemment `1:n` est le vecteur des `n` premiers nombres entiers, `.^` est l'opérateur qui calcule la puissance terme à terme (Cf. deuxième partie, section 4.1.1, p108). Enfin `sum` (Cf. deuxième partie, section 8.4, p147) permet d'obtenir la somme des coefficients d'un vecteur.

Si `sum` n'avait pas existé on aurait pu utiliser la multiplication matricielle car la somme des coefficients d'un vecteur n'est autre que le produit de ce vecteur (ligne) par un vecteur colonne de 1 : `v*ones(length(v),1)` ;

Exercice B.2

Peut-on vectoriser facilement les solutions des exercices précédents ?

B.2 Outils de vectorisation

Toutes les fonctions MATLAB de base portant sur des scalaires (à fort peu d'exceptions près) ont des capacités vectorielles.

Cependant, la possibilité de vectorisation dépend dans de nombreux cas de celle de créer, modifier et extraire des matrices de taille donnée sans avoir à faire de boucles :

Ceci est rendu possible par les moyens principaux suivants :

- La création (Cf. deuxième partie, section 3.1, p99) :
  1 - l'opérateur «:» qui permet de créer des progressions arithmétiques ;
  2 - la fonction `linspace` (resp. `logspace`) qui crée des séquences linéairement (resp. logarithmiquement) espacées ;
  3 - Les fonctions `ones`, `eye`, `zeros` et `rand` qui créent des matrices avec des remplissages divers ;
  4 - Les crochets carrés `[]` qui permettent d'empiler des matrices.
- La modification :
  1 - les opérateurs arithmétiques termes à termes : `+`, `-`, `.*`, `./`, `.\`, `.^`, `|`, `&` ; (Cf. chapitre 4, p107) ;
  2 - toutes les fonctions scalaires prédéfinies ;
  3 - la fonction `reshape` ;
  4 - l'affectation indexée (Cf. deuxième partie, section 3.2, p101) ;
  5 - les fonctions de cumul : `sum`, `cumsum`, `prod`, `cumprod`, etc. (Cf. deuxième partie, section 8.4, p147).
- L'extraction : (Cf. deuxième partie, section 3.2, p101)
  1 - l'indexation ;
  2 - l'indexation logique ;
  3 - la fonction `find` (Cf. deuxième partie, section 4.3.1, p113).

Chacun des items des exercices qui suivent se traitent en 4 lignes au plus et sans boucle. On suppose dans ce qui suit que n est un entier positif et `alea` est la matrice 10×10 définie par `alea = rand(10)`. Le cheval de bataille est ici l'indexation [3.2 (p101)].

Exercice B.3

1. *Quels sont les résultats des ordres suivant, expliquez :*

```
a = 1:16 ;
a = reshape(a, 4,4) ; % on peut consulter l'aide en ligne : help...
a1 = a(:, 4:-1:1) ;
a2 = a(4:-1:1, 4:-1:1) ;
a3 = a' ;
```

2. Créez le vecteur qui contient aux indices p pairs le carré du $p^{\text{ième}}$ entier et aux indices p impairs le cube du $p^{\text{ième}}$ entier (1 4 27 16 125 ...)
3. On donne un vecteur ligne `x`, créez le tableau `a` à n lignes dont toutes les lignes sont égales à `x` et le tableau `b` à n colonnes dont toutes les colonnes sont égales à `x.'`.

Exercice B.4

1. *Extrayez la matrice* `b` *formée des éléments situés dans les lignes* 3 à 8 *aux colonnes* 7, 1 *et* 3 *de* `alea`.
2. *Mettez à zéro les lignes impaires de* `alea`. *Mettez à* 0.8 *les éléments supérieurs à* 0.7 *restants.*
3. *Écrivez une fonction qui calcule à partir du vecteur des indices des éléments de* `y` *dans* `x` *la permutation inverse : celle qui donne les indices des éléments de* `x` *dans le vecteur* `y`. *C'est à dire que si l'on suppose que* `y(i)==x`, *le problème est de trouver* `j` *avec* `x(j)=y`.

Exercices supplémentaires.

Exercice B.5

1. *Tracez la fonction*
$$y = 3x^2 + \frac{\ln(x-\pi)^2}{\pi^4} + 1$$
dans l'intervalle $[\pi - 1, \pi + 1]$.
On utilisera les fonctions `linspace`, `plot` *et l' opérateur terme à terme «.^».*
Pourquoi ne voit-on pas la discontinuité en π ?
2. *Tracez la surface*
$$z = sin(x^2 + y^2)$$
pour $0 < x < 4\sqrt{\pi}, \quad 0 < y < 4\sqrt{\pi}$. *On utilisera les fonctions* `linspace`, `meshgrid`, `surf` *et l'opérateur terme à terme «.^»*
3. *Écrivez la fonction* `v = vnd(x)` *qui à partir du vecteur* `x` *de composante* $(x_0, \ldots, x_n)$ *construit la matrice suivante dite de Vandermonde*
$$v = \begin{pmatrix} x_0^n & \cdot & \cdot & \cdot & x_0^2 & x_0 & 1 \\ x_1^n & \cdot & \cdot & \cdot & x_1^2 & x_1 & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ x_n^n & \cdot & \cdot & \cdot & x_n^2 & x_n & 1 \end{pmatrix}$$

B.3 Solutions des exercices de la Session B

Solution de l'exercice B.1

Δ

1. Combien de fois ces boucles sont-elles évaluées ? Quelles sont les valeurs de `i` écrites ? Expliquer.
   (a) `for i=12:-2:1 ; disp(i), end ;` *12, 10, 8, 6, 4, 2, 6 fois*
   (b) `for i=12:-2:1 ; disp(i), i = 0 ; end ;` *12, 10, 8, 6, 4, 2, 6 fois*
   (c) `for i=12:-2:1 ; disp(i) ; if i == 8, break ; end ; end ;` *12, 10, 8, 3 fois*

   Le vecteur `12:-2:1` *est calculé une fois pour toute. Le fait d'affecter* 0 *à* `i` *dans la boucle* (b) *n'y change rien. Par contre la rencontre de* `break` *fait sortir prématurément de la boucle* (c).

2. Une suite de Fibonacci est définie par ses deux premières valeurs $f_1 = a$ et $f_2 = b$ et par la règle $f_{n+2} = f_n + f_{n+1}$. Écrivez une fonction calculant le $n^{\text{ième}}$ terme de la suite.
   Voici une solution simple et une améliorée, très différentes l'une de l'autre.

```
function f = fibs(a, b, n)
% calcule  le terme n de la suite
% de Fibonacci dont les deux premiers
% termes sont a et b

if n == 1
  f = a;
elseif n == 2
  f = b;
else
  for i = 3:n
    c = a+b;
    a = b;
    b = c;
 end;
 f = c;
end;
```

```
function f = fib(a, b, n)
% calcule en les conservant les termes
% de la suite de Fibonacci dont les
% deux premiers termes sont a et b
% et dont les indices sont dans n

persistent res

if length(res) < 2
  res = [a,b];
elseif res(1)-a | res(2)-b
  res = [a,b];
end;
if length(res) < max(n)
  for i = length(res)+1:max(n)
    res(i)=res(i-1)+res(i-2);
  end;
end;
f = res(n);
```

   La première fonction n'utilise pas de tableaux. Si n est supérieur à deux, à chaque itération elle fait glisser les deux dernières valeurs calculées dans `a` et `b` en sorte que l'itération suivante fasse le bon calcul.

   La seconde fonction stocke ses résultats dans la variable persistente `res`. De cette manière plusieurs appels consécutifs à `fib` avec les mêmes valeurs de `a` et `b` n'auront pas à refaire les calculs déjà effectués.

3. Écrivez une fonction calculant les n premières lignes du triangle de Pascal.

```
function c =  triang(n)

c = zeros(n, n);
c(:, 1) = 1;
for i = 2:n
  for j = 2:i
    c(i, j) = c(i-1, j-1)+c(i-1, j);
  end;
end;
```

   Évidemment la fonction `nchoosek` donne directement le résultat.

∇