Environnement de calcul scientifique et modélisation

Georges KOEPFLER

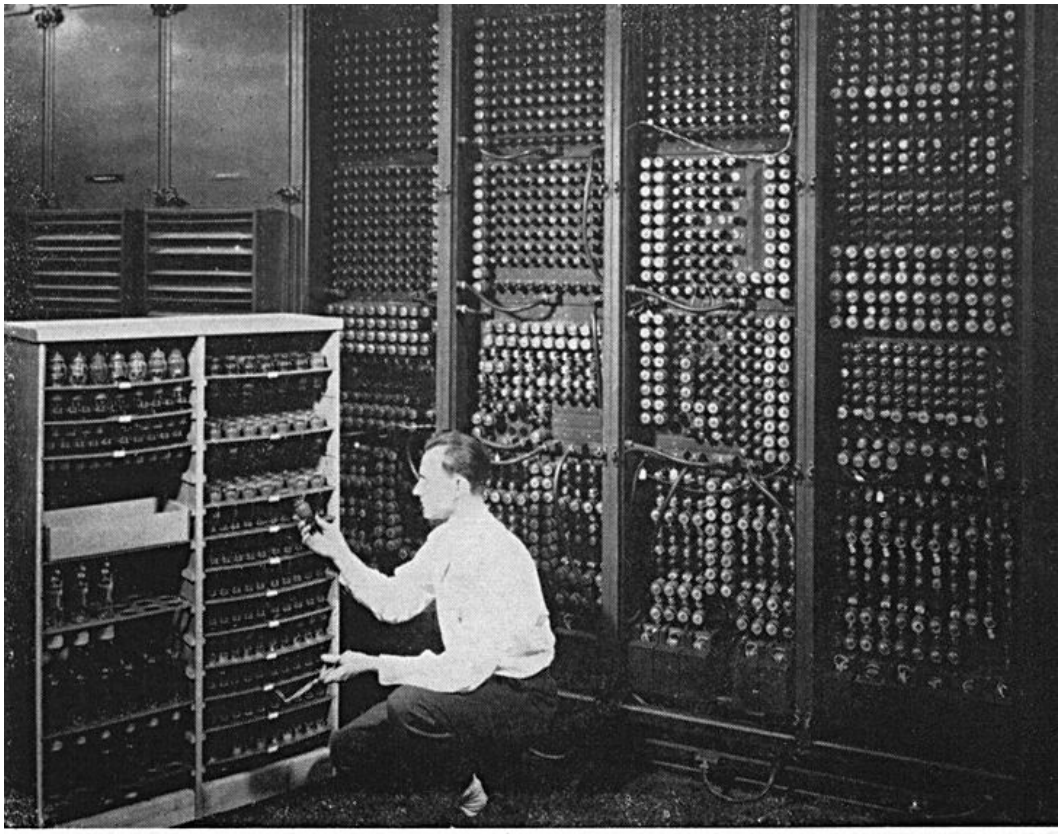
Université Paris Descartes

S4, 2012-13

Instruments de calcul

- Boulier (1100-) ;
- Règle à calcul (1650-1970) ;
- Machine à additionner, à multiplier,
  pour la “Pascaline” de Blaise Pascal (1642), voir le polycopié ;
- Machines électroniques,
  Z3 (1938), *ENIAC* (1946) ;
- Calculatrice (scientifique) de poche électronique (1972) ;
- Ordinateur personnel (1978).

Voir `http://fr.wikipedia.org/wiki/Instrument_de_calcul`



Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

ENIAC, Univ. de Pennsylvanie (1946-55), 30 tonnes sur 167 m^2.
Nombres signés de 10 chiffres : 5.000 additions simples/seconde, 357 multiplications/sec, 38 divisions/sec. (source Wikipedia).

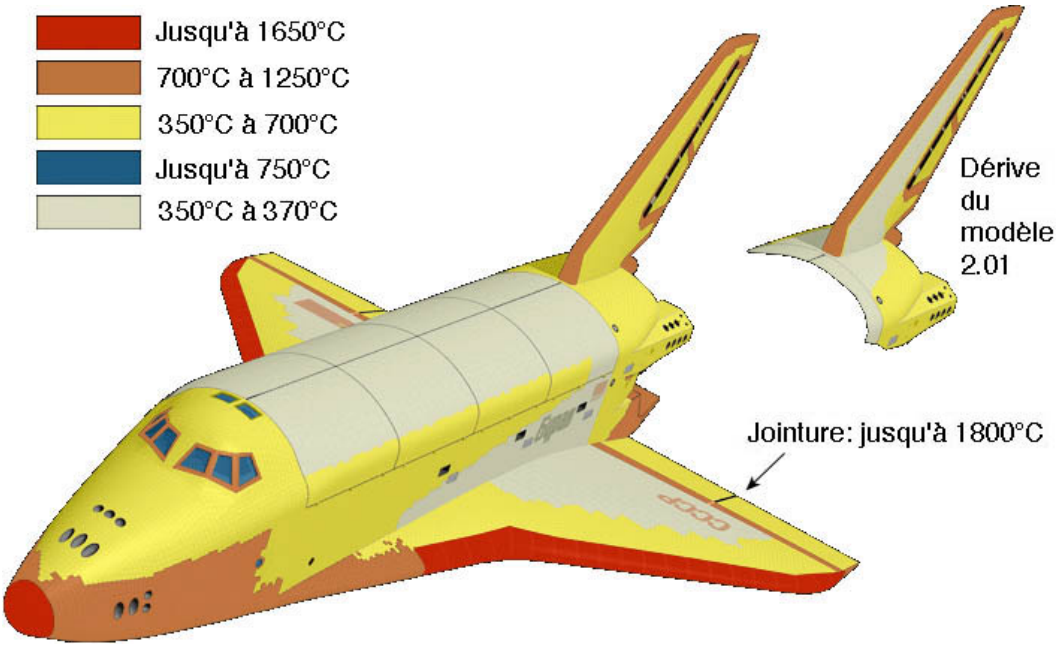
Environnements de calcul scientifique

Pour *modéliser* et *résoudre* des problèmes pratiques, souvent compliqués et faisant intervenir beaucoup de données et paramètres, les chercheurs et ingénieurs font appel

- au **calcul formel** qui permet des opérations abstraites comme la dérivation, l'intégration,...
  Exemples : *Maple*, *Mathematica* (logiciels propriétaires).
- et au **calcul numérique**.
  Des logiciels comme *Scilab* (libre) ou *Matlab* (propriétaire) sont capables d'effectuer de grandes quantités de calculs.
  Des boîtes à outils permettent de résoudre des problèmes d'optimisation, de statistiques,...
  Note : le logiciel *R* est spécialisé dans l'analyse statistique.

Combiné à des interfaces graphiques puissants, on obtient des logiciels de Conception Assistée par Ordinateur, CAO, et de CFA0, si des robots permettent la Fabrication.

Navette spatiale *Bourane* (bouclier thermique)



(source `http://www.buran.fr/`)

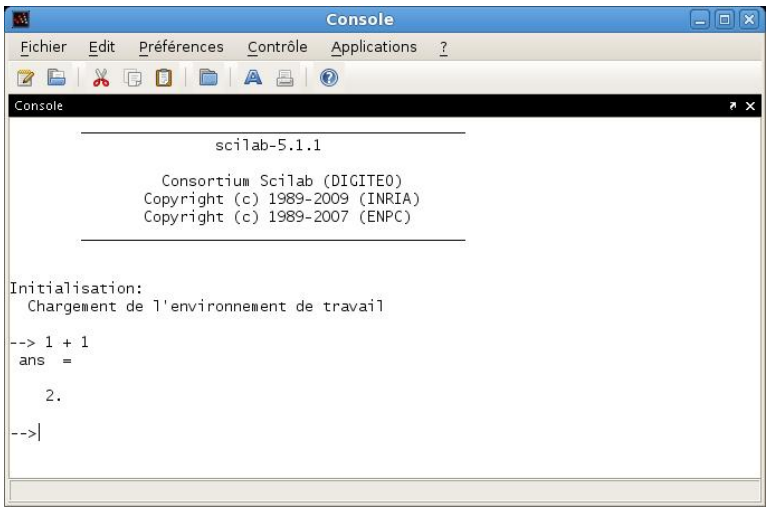
Plan du cours :

- Définition et opérations sur vecteurs, matrices et tableaux ;
- Représentations graphiques ;
- Programmation ;
- Calcul numérique :
  méthode de Horner, résolution de systèmes linéaires,
  intégration numérique,. . .

Objectifs du cours :

- Apprendre à utiliser un logiciel de calcul scientifique standard ;
- Introduction aux problèmes liés au calcul numérique :
  -> erreurs d'arrondi et précision,
  -> complexité des algorithmes et temps de calcul.

Lancer `scilab` par le menu : Applications -> Calcul -> Scilab



On obtient ainsi une ligne commande, indiquée par `-->`

Quelques objets scalaires de base :

- des constantes numériques prédéfinies : `%e`, `%pi`, `%i` ;
- des nombres réels : `2.56`, `1000.5` ;
- des nombres complexes : `%i`, `1.05+2*%i` ;

Les opérations et fonctions standards s'appliquent à ces quantités scalaires :
addition `+`, soustraction `-`, multiplication `*`, division `/`

De plus, toutes les fonctions standard sont disponibles :

`sqrt`, `sin`, `cos`, `log`, ...

On a accès, en un premier temps, à une calculatrice sophistiquée !

Grâce à la programmation, on obtient un logiciel pouvant résoudre des problèmes complexes.

Erreurs d'arrondi

```
                          |
-->%e                     | -->format("v",25)
 %e  =  2.7182818         |
                          | -->1/3
-->sqrt(-1)               |   ans =   0.33333333333333148296
ans  =  i                 |
                          | -->100000/3
-->(2+%i*3)*(1-%i)        |  ans  =   33333.333333333335758653
 ans  =  5. + i           |
                          | -->tan(atan(1))
-->sin(%pi/2)             |  ans  =   0.99999999999999889777
 ans  =  1.               |
                          |
```

Sur l'ordinateur, on a une précision finie et un temps limité !

Erreurs d'arrondi

La mémoire disponible étant finie, on ne peut pas représenter les nombres réels qui ont un développement décimal infini.
On peut représenter le rationnel $1/3$,
mais pas son écriture décimale,
on ne peut représenter π que par un nombre fini de décimales.

Un représentation fréquente est celle en *virgule flottante* des nombres réels :
$f = (-1)^s \cdot 0, d_{-1} d_{-2} \dots d_{-r} \cdot b^j$,
$m \leq j \leq M$ et $0 \leq d_{-i} \leq b-1$;
où s est le *signe*, $d_{-1} d_{-2} \cdots d_{-r}$ la *mantisse*, b la *base* et r est le nombre de *chiffres significatifs*.

La *précision machine* pour un flottant est la distance entre 1 et le nombre flottant immédiatement plus grand $1 + b^{1-r}$.
En **Scilab** `%eps = 2.220D-16`.

Erreurs d'arrondi

Le résultat d'un calcul n'est pas nécessairement représentable dans la machine : on est obligé d'arrondir.
Entre 1 et 1+%eps aucun nombre de la droite réelle est représentable.

```
-->format("v",25)
-->%eps
 %eps  =      0.0000000000000002220446
-->a=1+%eps
 a     =      1.0000000000000002220446
-->m=(a+1)/2
 m     =      1.                          // erreur d'arrondi
-->n=(a-1)/2
 n     =      0.0000000000000001110223
-->2*(m-n)
 ans   =      1.999999999999997779554 // erreur numérique
```

En effectuant un *grand* nombre d'opérations, les *erreurs* d'arrondi vont finir par *perturber* le résultat final.

Gestion d'erreurs

Si le résultat d'un calcul devient trop grand pour la machine, on obtient `Inf`, c'est la «valeur» de l'infini de Scilab.

Lorsque le résultat d'une opération n'est pas défini, le résultat est `NaN` (Not A Number).

Les *divisions par zéro* et les *singularités* d'une fonction donnent aussi lieu à des messages d'erreur.

La gestion des erreurs est importante lorsqu'un programme doit traiter de grandes quantités de données.

Exemple, calcul de $n/(n-1)$:

```
--> for n=1:1000, n/(n-1); end
         !--error 27
Division par zéro ...
```

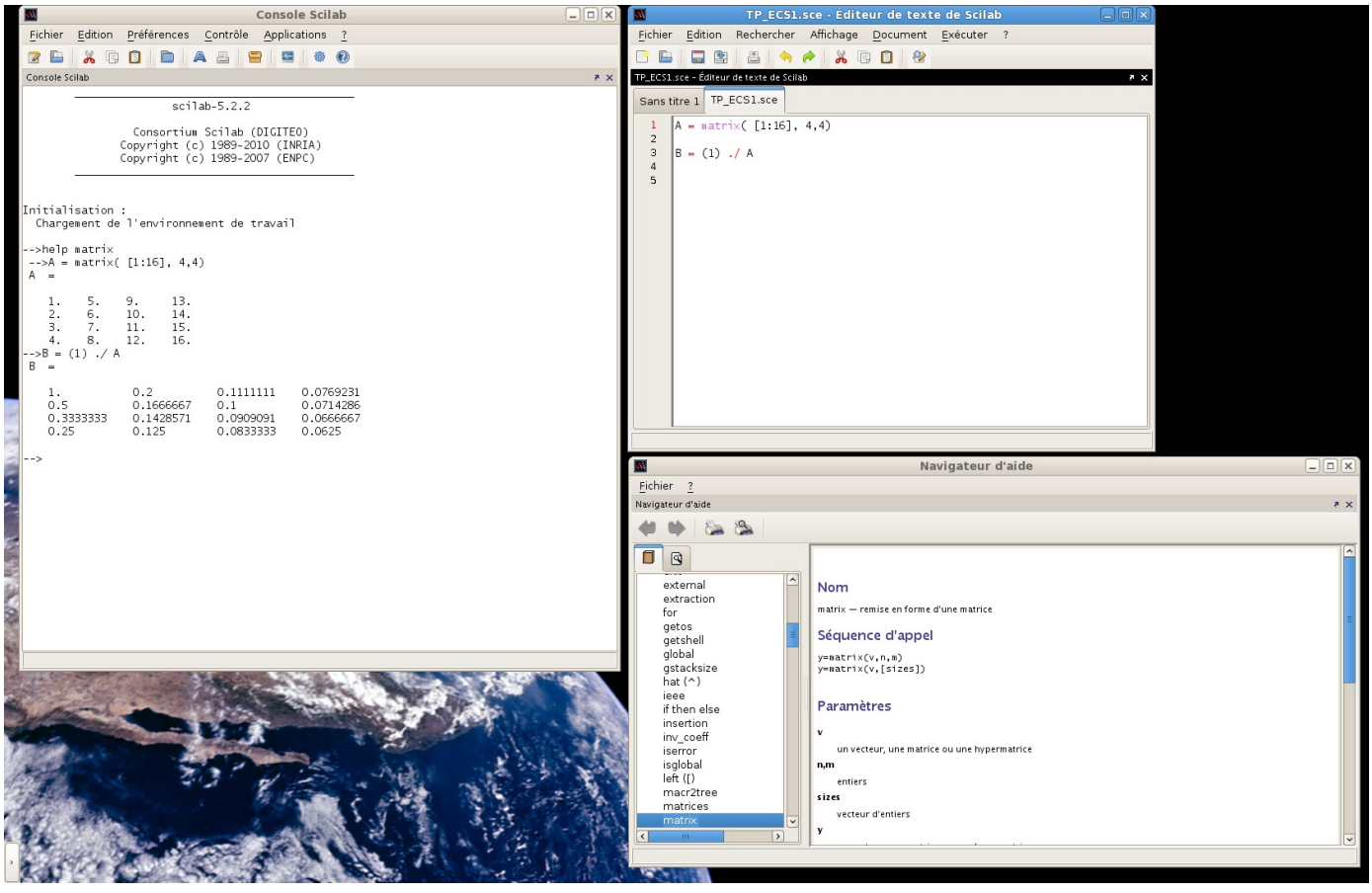
L'erreur sur le premier calcul empêche les autres à être effectués !

Fichiers de commande

Pour pouvoir répéter et sauvegarder des commandes, on peut utiliser des *fichiers de commande* ou fichiers **exécutables**.
Dans Scilab, le nom d'un tel fichier se termine en .sce
On y écrit les commandes comme sur la ligne de commande de Scilab.
Grâce au "copier-coller", on les exécute dans le terminal Scilab.
Le meilleur outil pour créer ou modifier un fichier "**.sce**" est l'éditeur de Scilab dans le menu Applications :

- les mots clefs de Scilab y apparaissent en couleur, ce qui permet d'éviter les fautes de frappe ;
- par le menu Exécuter de l'éditeur, on peut faire exécuter toutes les commandes Chargez ou une partie sélectionnée Évaluez... ;
- on utilisera l'éditeur en programmation pour la sauvegarde des fonctions définies par "`function`".

Trois fenêtres : Terminal Scilab - Editeur Scilab - Aide Scilab



Matrices et tableaux

Scilab permet de manipuler facilement des *matrices* ou *tableaux bidimensionnels* et des *tableaux multi-dimensionnels*.

```
-->A=[ 1, 2; 3, 4]
 A  =
    1.    2.
    3.    4.

-->v=[ 1; 0 ]
  v =
     1
     0

-->B=[ A , v]
  B =
    1.    2.    1.
    3.    4.    0.
```

Les éléments d'une matrice peuvent être :
des nombres *réels*, des nombres *complexes*, des *polynômes*, des *fractions rationnelles*, des *booléens* ou des *chaînes de caractères*.

- Grâce à `M(i,j)` on accède à l'élément de la i-ième ligne et j-ième colonne de la matrice M.
- Avec `M( ld:lf , cd:cf )` on accède au bloc des éléments des lignes `ld` à `lf` et des colonnes `cd` à `cf`.
- Si on **lit** un élément non existant, le message `invalid index` s'affiche.
  Si l'on **affecte** un élément non existant, la matrice est complétée par des zéros.

```
-->A=matrix([1:4],2,2)
 A  =
    1.    3.
    2.    4.
-->A(2,2)
ans = 4
```

```
-->A(2,4)
         !--error 21
invalid index
-->A(2,4)= 5
A  =
    1.    3.    0.    0.
    2.    4.    0.    5.
```

Les opérations $+$, $-$, $*$, $/$, sont interprétées comme

opérations matricielles

Soient `A` et `B` des matrices,

avec `size(A)=[l1,c1]` et `size(B)=[l2,c2]`

- la matrice `C=A+B`, resp. `C=A-B`,

est définie uniquement si `l1==l2` et `c1==c2`,

dans ce cas `size(C)=[l1,c1]` .

- la matrice `C=A*B`, est définie uniquement si `c1==l2`

et dans ce cas `size(C)=[l1,c2]` .

- si A est une matrice carrée,

`A^p` calcule la puissance matricielle avec p réel.

Attention : pour des matrices rectangulaires, on applique la puissance élément par élément, *cf.* opération `.^`

- l'opération `/` est à interpréter comme résolution d'une équation matricielle : `x=u/v` solution de $x * v = u$, c'est un problème *d'algèbre linéaire*.

Si l'on veut manipuler des *tableaux* on fait des opérations

élément par élément

Soient `A` et `B` de **même dimensions** : `size(A)==size(B)` .

Pour $1 \leq$`i`$\leq$`NbLignes`, $1 \leq$`j`$\leq$`NbColonnes`

- les opérations `A+B`, resp. `A-B`, restent inchangées :

`C(i,j)=A(i,j) + B(i,j)` et `C(i,j)=A(i,j) - B(i,j)`

- l'opération `A.*B` calcule le produit élément par élément :

`C(i,j)=A(i,j) * B(i,j)` ;

- l'opération `A./B` fait la division élément par élément :

`C(i,j)=A(i,j) / B(i,j)` ;

- l'opération `A.^B` prend la puissance élément par élément :

`C(i,j)=A(i,j)^B(i,j)` .

Remarques :

1. Pour A et B des *scalaires*, *i.e.* des matrices $(1,1)$, les opérations matricielles et élément par élément sont identiques.

2. Si A est une matrice et B un scalaire, alors l'opération élément par élément considère que B vaut `B*ones(A)`.

De même si B est une matrice et A un scalaire.

Motivations :

Les opérations élément par élément sur les tableaux permettent souvent d'éviter les boucles.

$\Rightarrow$ *Facilite* l'écriture et la vérification de programmes.

$\Rightarrow$ Apporte un *gain* en temps de calcul.

```
-->A=rand(100,100);
-->B=rand(100,100);
-->tic(); C=A.*B; toc()
 ans  =     0.01                  // temps CPU
-->clear C
-->tic(); for i=1:100,for j=1:100, ..
-->              C(i,j)=A(i,j)*B(i,j); end, end; toc()
 ans  =     0.12                  // temps CPU
```


Manipulation par blocs de matrices

En algèbre linéaire, on utilise souvent des **partitions de matrices en blocs** pour simplifier les calculs. Nous allons donner quelques exemples de manipulation par blocs des matrices.

Exemple : résolution de $A * X = B$, où $A = (a_{ij})$, $X = (x_{ij})$ et $B = (b_{ij})$ sont des matrices carrées d'ordre n.

Notons, pour $1 \leq j \leq n$, $X_{1j} = \begin{pmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{n-1,j} \\ x_{nj} \end{pmatrix}$ et $B_{1j} = \begin{pmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{n-1,j} \\ b_{nj} \end{pmatrix}$.

L'équation $A * X = B$ s'écrit, grâce aux colonnes de X et B :

$$A * [X_{11}\, X_{12} \cdots X_{1n}] = [B_{11}\, B_{12} \cdots B_{1n}]$$

où encore

$$A * X_{1j} = B_{1j} \text{ pour } 1 \leq j \leq n\,.$$

Addition par blocs de matrices

Soient A et B des matrices de même dimensions avec

$$A = \begin{pmatrix} A_{11} & \dots & A_{1q} \\ \vdots & & \vdots \\ A_{p1} & \dots & A_{pq} \end{pmatrix} \quad \text{et} \quad B = \begin{pmatrix} B_{11} & \dots & B_{1q} \\ \vdots & & \vdots \\ B_{p1} & \dots & B_{pq} \end{pmatrix}$$

où les blocs qui se correspondent, A_{ij} et B_{ij}, ont même dimensions.

L'addition par blocs de A et B s'écrit :

$$A + B = \begin{pmatrix} A_{11} + B_{11} & \dots & A_{1q} + B_{1q} \\ \vdots & & \vdots \\ A_{p1} + B_{p1} & \dots & A_{pq} + B_{pq} \end{pmatrix} .$$

Multiplication par blocs de matrices

Soient A et B des matrices décomposées en blocs :

$$A = \begin{pmatrix} A_{11} & \dots & A_{1r} \\ \vdots & & \vdots \\ A_{q1} & \dots & A_{qr} \end{pmatrix} \quad \text{et} \quad B = \begin{pmatrix} B_{11} & \dots & B_{1s} \\ \vdots & & \vdots \\ B_{r1} & \dots & B_{rs} \end{pmatrix} ,$$

Si les produits AB, $A_{ik}B_{kj}$ $(1 \leq i \leq q,\ 1 \leq j \leq s,\ 1 \leq k \leq r)$ sont définis, la multiplication par blocs de A et B s'écrit :

$$C = AB = \begin{pmatrix} \sum_{k=1}^{r} A_{1k}B_{k1} & \dots & \sum_{k=1}^{r} A_{1k}B_{ks} \\ \vdots & & \vdots \\ \sum_{k=1}^{r} A_{qk}B_{k1} & \dots & \sum_{k=1}^{r} A_{qk}B_{ks} \end{pmatrix} = \begin{pmatrix} C_{11} & \dots & C_{1s} \\ \vdots & & \vdots \\ C_{q1} & \dots & C_{qs} \end{pmatrix} .$$

Exemple :

On définit
$A_{11} =$ `[1,2; 3,4]`$= \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, $A_{22} = A'_{11}$,
$A_{12} = A_{21} = B_{11} =$ `eye(2,2)`$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$,
$B_{21} =$`zeros(2,2)`, $B_{12} =$`zeros(2,1)` et $B_{22} =$`ones(2,1)`.

On pose $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ et $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$.

Écrire A, B et calculer

$$A * B = \begin{pmatrix} A_{11} * B_{11} + A_{12} * B_{21} & A_{11} * B_{12} + A_{12} * B_{22} \\ A_{21} * B_{11} + A_{22} * B_{21} & A_{21} * B_{12} + A_{22} * B_{22} \end{pmatrix}$$

Déterminant par blocs

Soit A une *matrice carrée* et diagonale par blocs :
$$A = (A_{ij})_{1 \le i,j \le p} \text{ avec } A_{ij} = 0 \text{ pour } i \neq j.$$

Les blocs A_{ii} sont *carrés*, de taille $(n_i, n_i)_{1 \le i \le p}$,
on calcule alors le déterminant par blocs de A :

$$\det(A) = \begin{vmatrix} A_{11} & O & O & \dots & O \\ O & A_{22} & O & \dots & O \\ \vdots & & \ddots & & \vdots \\ O & & O & A_{p-1,p-1} & O \\ O & & \dots & O & A_{pp} \end{vmatrix} = \prod_{i=1}^{p} \det(A_{ii}) .$$

Note :

1. les dimensions p_i des blocs A_{ii} peuvent varier ;
2. par O, on désigne des blocs rectangulaires de zéros, de dimensions adaptées et obtenus par la commande `zeros(n,m)`
.

Exemples :

- On suppose les blocs de A et x compatibles :

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \in \mathcal{M}(d,d) \text{ et } x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^d \ :$$

$$Ax = \begin{pmatrix} A_{11}x_1 + A_{12}x_2 \\ A_{21}x_1 + A_{22}x_2 \end{pmatrix}.$$

- Soit $\alpha \in \mathbb{R}$, $v \in \mathbb{R}^d$ et $B \in \mathcal{M}(d,d)$.
On pose $A = \begin{pmatrix} \alpha & v^t \\ v & B \end{pmatrix} \in \mathcal{M}(d+1,d+1)$, calculer A^2.

- On définit `B=[ 2, 1 ; 1, 2]` et ensuite on exécute :
`A=B, A(3:4,3:4)=B, A(5:6,5:6)=B`
Écrire `A` et calculer le déterminant de `A`.

Temps de calcul et architecture

- Il ne suffit pas d'augmenter la *vitesse de calcul* et la *mémoire centrale des processeurs* pour obtenir des programmes rapides !
- Lors de la multiplication de très grandes matrices, une grande partie du temps de traitement est utilisé pour *transférer des données* et non *pas pour les calculs* !
- Des librairies spécialisées (BLAS, LAPACK) implémentent des opérations par blocs en tenant compte de l'architecture du processeur et du bus de données :
  • les blocs transférés sont adaptés à la taille des mémoires d'accès rapide,
  • on fait moins de transferts lents, on passe plus de temps sur les calculs.
- Des logiciels interprétés tels que **Scilab** ou **Matlab**©, utilisent des opérations par blocs pour accroître leur performances.

Temps de calcul et algorithmes

Soient A, B et C des matrices réelles rectangulaires, de dimensions respectives `[n,m]`, `[m,p]` et `[p,q]` .

Combien d'opérations sur des nombres réels sont nécessaires pour calculer le produit ABC ?
Deux possibilités d'évaluation car $(AB)C = A(BC)$!

Le calcul de AB nécessite $m-1$ additions et m multiplications pour chacun des np termes de AB. D'où en tout $O(2nmp)$ opérations.

Le calcul de $(AB)C$ nécessite $O(2np(m+q))$ opérations, tandis que le calcul de $A(BC)$ se fait en $O(2mq(n+p))$ opérations.

Exemple : pour $n = p = 10$ et $m = q = 100$, on trouve $4 \cdot 10^4$ opérations pour $(AB)C$ et $4 \cdot 10^5$ pour $A(BC)$.

La multiplication des matrices rectangulaires est *associative*, mais le nombre d'opérations nécessaires peut varier avec la position des parenthèses !

Temps de calcul et algorithmes

Pour des matrices carrées A et B, de taille $n = 2^p$, on peut diminuer le nombre de multiplications nécessaires pour calculer $C = A * B$.

```
function C=mult_matrices(A,B)
   n=size(A,1);
   C=zeros(n,n);  // initialisation
   for i=1:n
     for j=1:n
       for k=1:n
       C(i,j)= C(i,j)+A(i,k)*B(k,j);
       end
     end
   end
endfunction
```

Algorithme classique en $O(n^3)$ multiplications.

Temps de calcul et algorithmes

Algorithme de Strassen (1969)

```
// A et B des matrices carrées de même taille n=2^p
function C=strassen(A,B)
   n=size(A,1);
   if (n==1)              // fin de récurrence
     C=A*B;
     return;
   end
   n2=n/2;
// Décomposition en blocs
   A11=A(1:n2,1:n2);      A12=A(1:n2,n2+1:n);
   A21=A(n2+1:n,1:n2);    A22=A(n2+1:n,n2+1:n);

   B11=B(1:n2,1:n2);      B12=B(1:n2,n2+1:n);
   B21=B(n2+1:n,1:n2);    B22=B(n2+1:n,n2+1:n);
```


Temps de calcul et algorithmes

```
// évaluation récurrente de 7 produits matriciels
   M1=strassen((A11+A22),(B11+B22));
   M2=strassen((A21+A22),B11);
   M3=strassen(A11,(B12-B22));
   M4=strassen(A22,(B21-B11));
   M5=strassen((A11+A12),B22);
   M6=strassen((A21-A11),(B11+B12));
   M7=strassen((A12-A22),(B21+B22));

   C=[(M1+M4-M5+M7),(M3+M5);(M2+M4),(M1-M2+M3+M6)];
endfunction
```

Algorithme en $O(7^p) = O(n^{\log_2 7}) = O(n^{2.807})$ multiplications.

Temps de calcul et algorithmes

```
-->A=rand(256,256); B=A';

-->timer(); C = strassen(A,B); timer()
 ans  =       1.12

-->timer(); C = mult_matrices(A,B); timer()
 ans  =     143.21
```

Mais les routines de **Scilab** sont bien sûr les mieux adaptées !

```
-->timer(); C=A*B; timer()
 ans  =       0.06
```


Représentation graphique d'une fonction

Pour représenter une fonction $f : \mathbb{R} \mapsto \mathbb{R}$ sur $[a, b]$:

1. on discrétise $[a, b]$ en N points, grâce à `x=linspace(a,b,N)` ;
2. on calcule les $y_i = f(x_i)_{(1 \leq i \leq N)}$, grâce à `y=f(x)` ;
3. la commande `plot2d` relie les points $(x_i, y_i)_{(1 \leq i \leq N)}$ par des segments (choix par défaut).

Important : Il faut avoir une bonne compréhension de ce que l'on doit obtenir pour ne pas mal interpréter ce qui s'affiche !

À connaître :

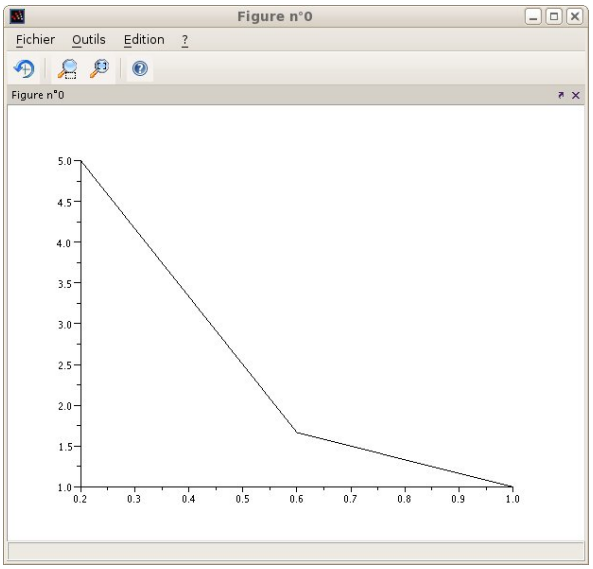
- Domaine de définition de la fonction et éventuelles singularités ;
- Ensemble image ou l'étendue des *valeurs* que peut prendre la fonction ;
- Échantillonage ou le nombre de points discrets nécessaires pour avoir une "bonne" représentation.

Étude et représentation graphique de fonctions

On considère la fonction définie par $f(x) = \frac{1}{x}$ sur $]0, 1]$.

```
--> x = linspace(0,1,3)' ; y = (1)./x  ;
            !--error 27
Division par zéro ...
```

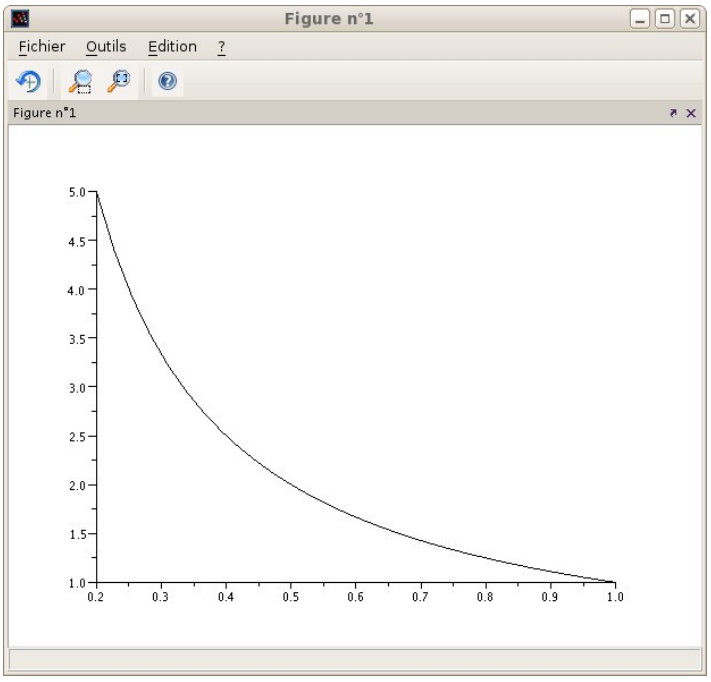
```
--> x = linspace(0.2,1,3)' ; y = (1)./x  ; plot2d(x,y)
```



Pas assez de points.

Étude et représentation graphique de fonctions

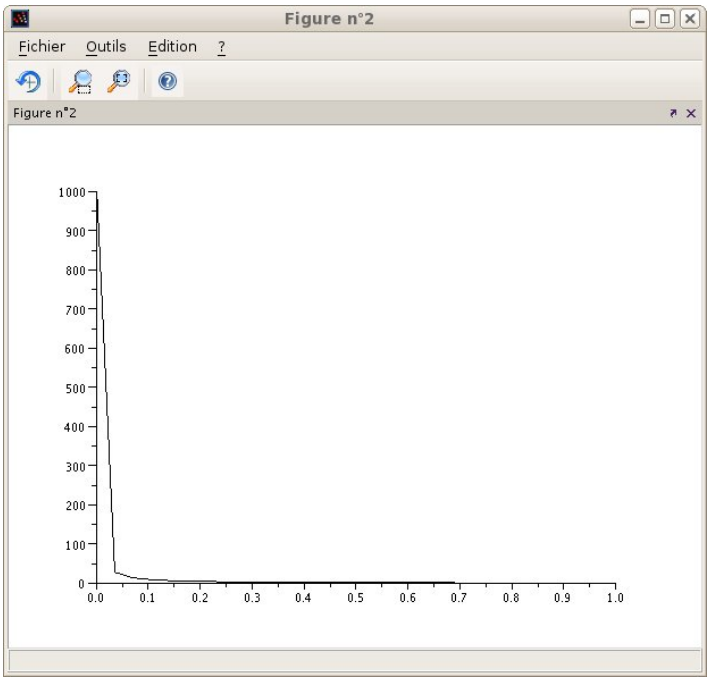
```
--> x = linspace(0.2,1,30)' ; y = (1)./x  ;  plot2d(x,y)
```



Graphe lisse, mais sur [0.2, 1].

Étude et représentation graphique de fonctions

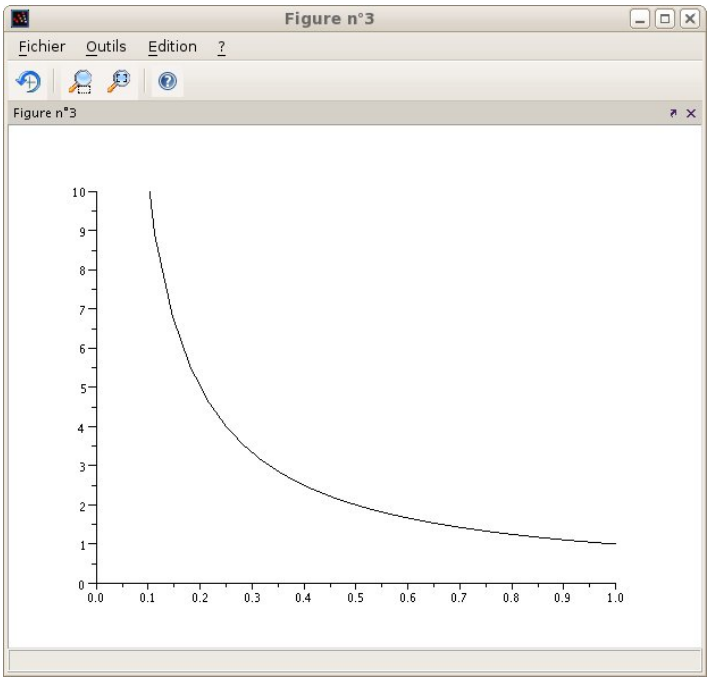
```
--> x = linspace(0.001,1,30)' ; y = (1)./x  ; plot2d(x,y)
```



Près de $x = 0$ la fonction devient trop grande : graphe écrasé.

Étude et représentation graphique de fonctions

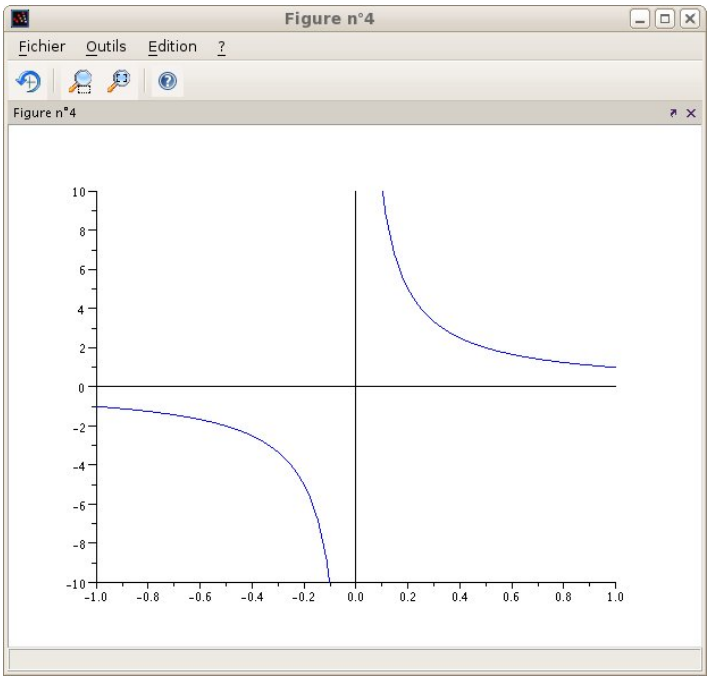
```
--> x = linspace(0.001,1,30)' ; y = (1)./x  ;
--> plot2d(x,y,rect=[0,0,1,10])
```



Graphe correct.

Étude et représentation graphique de fonctions

```
--> x = linspace(0.001,1,30)' ; y = (1)./x  ;
--> plot2d([-x,x], [-y,y], rect=[-1,-10,1,10], style=[2,2])
--> plot2d( [-1, 0; 1, 0], [0, 10; 0, -10], style=[1, 1] )
```



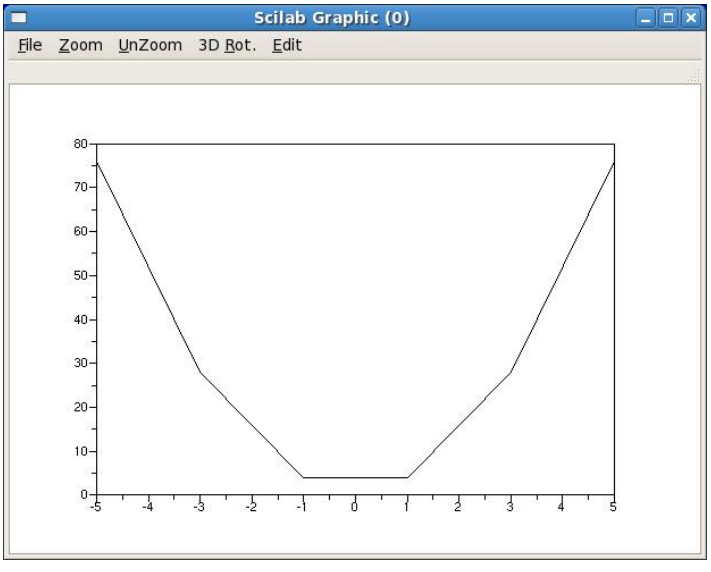
Bonne représentation de la fonction impaire $f(x) = \frac{1}{x}$ avec axes.

Étude et représentation graphique de fonctions

On considère la fonction $f(x) = 3x^2 + 1 + \frac{1}{\pi^4} \log[(\pi - x)^2]$.
On a $\mathrm{dom}(f) = \mathbb{R} \setminus \{\pi\}$ et $\lim_{x \to \pi} = -\infty$.

Essayons un tracé "naïf" :

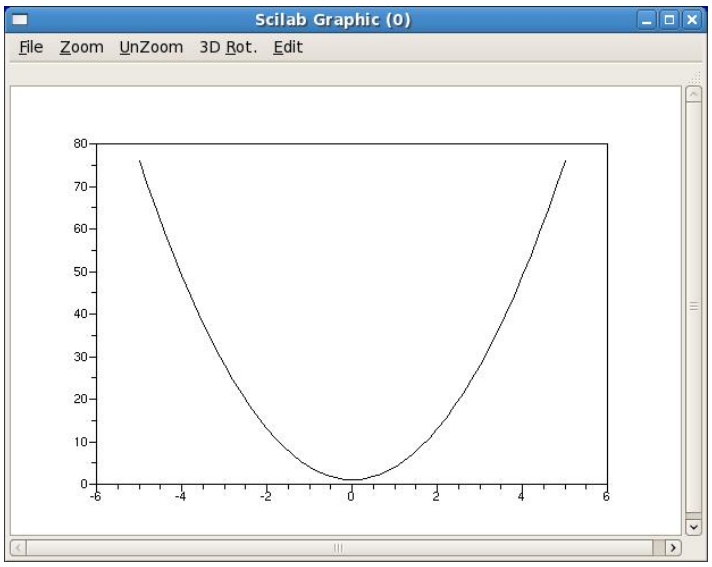
```
--> x = [-5 : 2 : 5] ;
--> y = 3* x.^2 + 1 + log( (%pi-x).^2) /(%pi^4)  ;
--> plot2d(x,y)
```



Représentation graphique de fonctions (suite)

Il faut utiliser suffisamment de points pour avoir un graphe lisse.

```
--> x = [-5 : 0.2 : 5] ;
--> y = 3 *x.^2 + 1 + log( (%pi-x).^2) /(%pi^4) ;
--> plot2d(x,y)
```



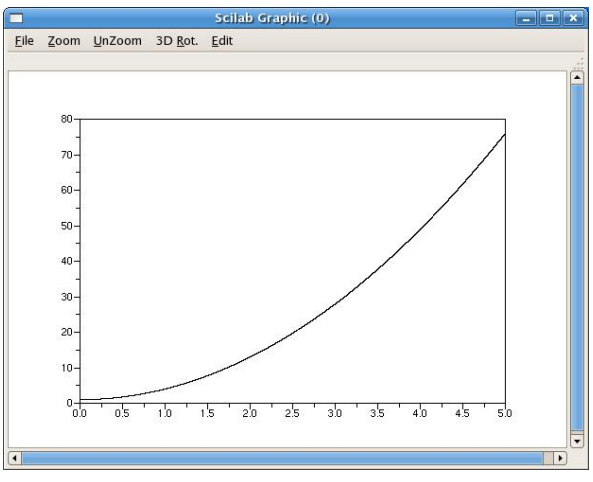
Représentation graphique : singularités invisibles

Même avec 50001 points, la singularité de

$$f(x) = 3x^2 + 1 + \frac{1}{\pi^4} \log[(\pi - x)^2]$$

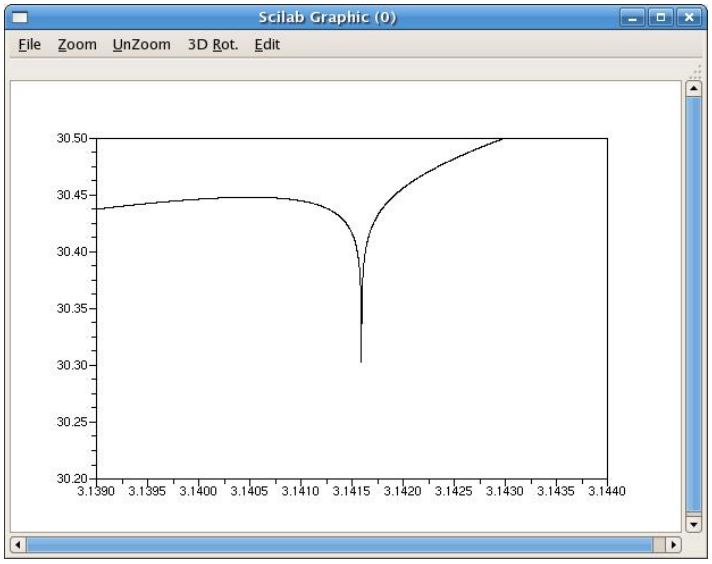
reste “invisible” !

```
--> x=[0:0.0001:5];
--> y = 3 *x.^2 + 1 + log( (%pi-x).^2) /(%pi^4) ;
--> plot2d(x,y)
```



Représentation graphique : singularités

Il faut utiliser la fonction Zoom pour commencer à apercevoir la variation au voisinage de $x = \pi$:



Explication :
On montre que f est négative sur $[\pi - \varepsilon_1, \pi[$ et $]\pi, \pi + \varepsilon_2]$,
où les ε_i sont de l'ordre de $e^{-\pi^6} \sim 10^{-418}$, ce qui est largement en dessous de la précision machine %eps de **Scilab**.

Représentation graphique : échantillonnage

On veut représenter $f(t) = cos(\omega\, t)$ sur $[0, 10]$, avec $\omega = 4\pi$.

```
--> x = linspace(0,10,21); y = cos(4*%pi*x);
--> scf(0); plot2d(x,y)
--> scf(1); plot2d(x,y,style=[0])
```