Université de Picardie Jules Verne
Licence 2
Langage de programmation - Scilab
2019-2020

# Autour des polynômes

## 1 Construction des polynômes en Scilab

### 1.1 Définition par les coefficients et par les racines

Soit $\mathbf{P}_n$ l'espace vectoriel des polynômes à coefficients réels, de degré inférieur ou égal à $n$. On les représente la base canonique $\{1, x, x^2, \cdots, x^n\}$ comme suit:

$$p(x) = \sum_{i=0}^{n} a_i x^i,$$

les $a_i$ sont les *coefficients*. On rappelle le théorème fondamental de l'algèbre ou aussi appelé théorème de d'Alembert :

**Théorème 1 (Théorème fondamental de l'algèbre)**
Tout polynôme à coefficient complexe de degré $n$ admet $n$ racines complexes, ou, de manière équivalente, on peut écrire tout $p \in \mathbf{P}_n$ sous la forme

$$p(x) = a_n \prod_{i=1}^{n} (x - \xi_i),$$

les nombres $\xi_i$ étant les racines de $p(x)$.

On considérera aussi sa formulation équivalente

**Théorème 2 (Forme équivalente)**
Tout polynôme non constant coefficients réels s'écrit comme un produit de polynômes à coefficients réels de degrés 1 ou 2.

On dispose donc de deux façons de définir un polynôme : par ses racines ou par ses coefficients. Il en est de même en Scilab.

```
// Définition par les racines
-->p1 = poly([1 2 3],"x")
p1=

   -6 +11x -6x^2 +x^3
// Définition par les coefficients
--> p2 = poly([1 2 3],"x","coeff")
p2=

   1 +2x +3x^2
```

### 1.2 Les commandes `roots` `coeff` et `pfact`

La commande `roots` (racines en anglais) permet de calculer numériquement les racines d'un polynôme, la commande `coeff` retourne les coefficient d'un polynôme dans la base canonique. On reprend ci-après les exemples précédents.

```
//retrouvons les racines de p1, défini par ses racines !
--> r=roots(p1)
r =

3.000000000000035527137
1.999999999999928945726
1.000000000000004440892

// On retrouve alors les coefficients de p1 par

--> cp1=coeff(p1)
cp1 =

-6.   11.   -6.   1.
// Les coefficients sont rangés par ordre croissant des puissances, de a_0 à a_n
```

Exemple 1

```
// Voici un polynomial avec des racines complexes
p=poly([0,10,1+%i,1-%i],"x");
p=

                2    3    4
-20x +22x  -12x  +x

r=roots(p)
r=

9.999999999999996447286
1.  + 1.000000000000001110223i
1.  - 1.000000000000001110223i
0.
```

La commande `pfact` permet de calculer numériquement les facteurs de degré 1 ou 2 du polynôme, ce qui illustre la version équivalente du théorème de d'Alembert :

```
polfact(p1)

column 1 to 3

1 -3.000000000000035527137 +x -1.999999999999928945726 +x

column 4

-1.000000000000004440892 +x
```

Le résultat est présenté sous la forme d'un vecteur ligne à 4 entrées : ici $p1$ est de degré 3, `pfact` le met sous la forme $a_n \prod_{i=1}^{n}(x-\xi_i)$, c'est à dire ici $1 \times (x-1) \times (x-2) \times (x-3)$ (à l'ordre près des facteurs) et lui associe donc le vecteur $(1, x-1, x-2, x-3)$ .

### 1.3 Evaluation d'un polynôme

Soit $x$ un réel fixé. Le calcul de la valeur de $p(x) = \sum_{i=0}^{n} a_i x^i$ en évaluant séparément les termes $a_i x^i$ l'aide de $i$ multiplications, puis en prenant la somme de ces terme nécessite $N_{tot} = n + \frac{n(n+1)}{2} \simeq \frac{n^2}{2}$ opérations (voir aussi la feuille sur l'étude ds fonctions). L'agorithme de Horner permet de calculer $p(x)$ de proche en proche par multiplication de facteurs successifs. On rééecrit

$$p(x) = a_0 + x\left(a_1 + x\left(a_2 + (a_3 + \cdots (a_{n-1} + a_n x))\cdots\right)\right).$$

Ainsi, on évalue $p(x)$ comme suit :

`Algorithme de Horner`

| | |
|---|---|
| `Initialisation` | $b_0 = a_n$ |
| `Pour` $k = 1, \cdots n$ | |
| `poser` | $b_k = x b_{k-1} + a_{k-1}$ |
| | |
| `poser` | $p(x) = b_n$ |

On effectue 1 addition et une multiplication à chaque étape, si bien qu'ici le nombre totale d'opérations est $N_{tot}^{Horner} = 2n$. Cette méthode est implantée en Scilab avec la commande `horner` :

```
//evaluation d'un polynome
--> p=poly([1 2 3],"x");
-->horner(p,[1 2 5])
ans =

0.   0.   24.

horner(p,[1 2 5]+%i)
ans =

3.  + i  -2.i  15.  + 25.i
```

### 1.4 Dérivation d'un polynôme

On peut calculer formellement la dérivée d'un polynôme à l'aide de la commande `derivat`

```
  -->p3 = poly(ones(1, 10), 'z', 'coeff')
p3 =

                2  3  4  5  6  7  8  9
1 +z +z  +z  +z  +z  +z  +z  +z  +z

  --> derivat(p3)
ans =

                2  3  4  5  6  7  8
2z +3z  +4z  +5z  +6z  +7z  +8z  +9z
```

### 1.5 Polynômes en algèbre linéaire

```
// Calcul des valeurs propres comme racines du polynôme caractéristique
-->A=rand(3,3)//matrice 3 x 3 de coefficients tirés suivant une loi uniforme sur [0,1]
A=

column 1 to 2

0.211324865464121103287   0.3303270917385816574097
0.756043854169547557830    0.6653811042197048664093
0.000221134629100561142   0.628391778834110498428 34

column 3

0.8497452358715236186981
0.6857310198247432708740
0.8782164813019335269928

-->p = poly(A,'x')//retourne le polynôme caractéristique de A

-0.216730554095534294623 0 +0.2297117866228667315553x

                                2  3
-1.7549224509857594966888x  +x

// calculons ses racines (donc les valeurs propres de A)
rA=roots(p)
rA =

1.6948371110182862686 6
0.03004266998373655850 33 + 0.3563346188599859631907i
0.03004266998373655850 33 - 0.3563346188599859631907i
```

La commande `spec` permet de calculer les valeurs propres `spA=spec(A)`

```
spA =

0.030042669983736509931 + 0.3563346188599859631907i
0.030042669983736509931 - 0.3563346188599859631907i
1.6948371110182864907046
//on retombe sur nos pieds !
```

### 1.5.1 Théorème de Cayley Hamilton et applications

Le théorème de Cayley Hamilton est un des premiers résultats fondamentaux rencontrés en algèbre linéaire. Il est du aux mathématiciens anglais Arthur Cayley et William Hamilton et l'on doit sa première dmonstration au mathématicien allemand Ferdinand Georg Frobenius en 1878.

**Théorème 3 (Cayley Hamilton)**
Soit $A$ une matrice carrée d'ordre $n$ sur un corps commutatif $K$ ($K = \mathbb{R}$ ou $K = \mathbb{C}$). Soit $p_A(x)$ sont polynôme caractéristique

$$p_A(x) = det(x\ Id_n - A) = x^n + \alpha_{n-1}x^{n-1} + \cdots \alpha_1 x + \alpha_0.$$

Alors

$$P_A(A) = A^n + \alpha_{n-1}A^{n-1} + \cdots \alpha_1 A + \alpha_0 = 0.$$

Scilab ne nous permet de démontrer ce théorème pour de petites valeurs de $n$, ce qui est possible avec un logiciel de calcul formel tel que Maple. Nous allons le vérifier sur des matrices de notre choix, pas n'importe lesquelles : nous les choisirons "au hasard". A titre d'illustration, on reprend l'exemple précédent où $A$ avait été construite "au hasard". On suppose connues les valeurs propres de `A` et rangées dans le vecteur `r`. On peut évaluer $p_A(A)$ comme suit :

```
Id=eye(n,n);//la matrice identité
PA=Id; for i=1:n
PA=PA*(r(i)*Id-A);
end
PA =

column 1 to 2

0.00000000000000001974862 0.00000000000000000127808
0.00000000000000003853102 0.00000000000000005584136
0.  0.00000000000000002220446

column 3
0.00000000000000003330669
0.00000000000000002220446
0.00000000000000006661338
```

Le résultat est acceptable, il y a évidemment des erreurs d'arrondis.

Une application amusante est le calcul de l'inverse d'une matrice (inversible !).

**Exercice 1 (Cayley-Hamilton pour le calcul de l'inverse d'une matrice)**

1. Soit $A$ une matrice $n \times n$ inversible. On note $P(x) = \sum_{i=0} a_i x^i$ son polynôme caractéristique.

   (a) Montrer que nécessairement on a $a_0 \neq 0$.

   (b) En déduire que
   $$Id = A\left(\frac{1}{a_0}\sum_{i=1}^{n} a_i A^{i-1}\right) = \left(\frac{1}{a_0}\sum_{i=1}^{n} a_i A^{i-1}\right) A.$$
   *Ind. On pourra utiliser le théorème de Caley-Hamilton*

   (c) En déduire que
   $$A^{-1} = \frac{1}{a_0}\sum_{i=1}^{n} a_i A^{i-1}.$$

   (d) Méthode de Fadeev-Leverrier. Soit $A$ une matrice inversible. On considère la suite
   $$B^{(0)} = Id; \alpha_k = Trace(AB^{(k)})/k,\ B^{(k+1)} = \alpha_k Id - AB^{(k)}$$
   Alors si $\alpha_n \neq 0$,
   $$A^{-1} = \frac{1}{\alpha_n} B_{n-1}$$
   Programmer cette méthode.

### 1.5.2 Matrice compagnon

Les valeurs propres d'une matrice carrée sont les racines de son polynôme caractéristique $p(x) = det(xId - A)$. On peut ainsi associer un polynôme à toute matrice carrée. On peut donc, en principe, utiliser toute méthode numérique de calcul de racines d'un polynôme pour déterminer les valeurs propres d'une matrice ; le calcul des racines d'un polynme par les méthodes itératives classiques (Newton, etc) nécessite de connaître au préalable leur localisation et aussi leur multiplicité. Par ailleurs, le calcul des coefficients du polynôme caractéristique est instable numériquement[1].

Inversement, étant donné un polynôme $p(x) = x^n + \sum_{i=0}^{n-1} a_i x^i$, peut-on construire une matrice $M$ telle que $det(xId - M) = p(x)$ ? Si oui, on pourrait alors utiliser toute méthode de calcul numérique de valeurs propres d'une matrice (cela existe) pour déterminer les racines d'un polynôme. Soit $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots a_{n-1} x^{n-1} + x^n$. A ce polynme $p$ on associe la matrice

$$M = \begin{pmatrix} 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & -a_{n-1} \end{pmatrix}$$

On dit que $M$ est la *matrice compagnon de p.* Nous avons le

**Théorème 4**
$p(x)$ est le polynôme caractéristique de M.

[1] La méhode Faddev-Leverrier est simple mais instable

**Remarque 1**
On peut considérer la transposée de cette matrice et toute permutation, ce qui ne change pas ses valeurs propres.

En scilab, la commande `companion` permet de construire la matrice companion d'un polynôme donné.

```
--> p=poly([1 2 3],"x")
p=

-6 +11x -6x^2 +x^3
--> M=companion(p)
M =

6. -11.   6.
1.   0.   0.
0.    1.   0.
//
//On retrouve bien p
//
--> poly(M,"x") ans =

-5.999999999999982236432 +10.99999999999999822364x -6x^2 +x^3
```

# 2 Quelques polynômes remarquables

## 2.1 Préambule

On considère ici les polynômes orthogonaux. Soit $[a, b]$ un intervalle borné. On commence par se donner une fonction $w$ intégrable telle que

- $w(x) \geq 0 \forall x \in ]a, b[$
- tous les moments $\mu_k = \int_a^b x^k w dx$ existent et sont finis.
- pour les polynômes positifs $P$ sur $[a, b]$ $\int_a^b P w dx = 0 \Rightarrow P = 0$.

On vérifie aisément que la quantité $I(f) = \int_a^b f w dx$ est bien définie pour toute fonction continue sur $[a, b]$. On peut montrer qu'elle l'est en fait pour toute fonction $f$ telle que $I(f^2) = \int_a^b f^2 w dx < +\infty$. On notera par $L^2_w([a, b])$ l'ensemble des fonctions telles que $I(f^2) = \|f\|^2_w < +\infty$ On montre également que l'application $(.,.)_w : (f, g) \mapsto \int_a^b f g w dx$ est un produit scalaire sur $L^2_w([a, b])$. Nous pouvons dès lors définir une famille de polynômes orthogonaux par rapport au produit $(.,.)_w$ :

**Définition 1**
La famille $(p_n)_{n\geq 0}$ est une suite de polynômes orthogonaux par rapport au produit scalaire $(.,.)_w$ si

- $p_n \in P_n,\ \forall n \geq 0$,

- $(p_n, p_m)_w = 0$ si $n \neq m$.

On construit une telle suite de polynômes orthogonaux en appliquant le procédé d'orthogonalisation de Gram-Schmidt à la base canonique $\{1, x, x^2, \cdots\}$ ; faut imposer ds conditions supplémentaires pour s'assurer de l'unicité.

## 2.2 Polynômes de Tchebytcheff

**Définition 2**
On définit le polynôme de Tchebytcheff de degré $n$, $n = 0, 1, \cdots$ par $T_n(x) = \cos(ncos(x)), \in [-1, 1]$.

Ces polynômes possèdent (notamment) les propriétés suivantes

**Théorème 5 (Polynômes de Tchebytcheff)**

- orthogonalité

$$\int_{-1}^{1} T_n T_m \frac{1}{\sqrt{1-x^2}dx} = 0, \text{ si } n \neq m,$$

- relation de récurrence à 3 termes

$$T_0 = 1, \ T_1 = x, \ \ T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \forall n \geq 1.$$

On retrouve ces polynômes dans la minimisation de l'erreur d'interpolation polynomiale. En scilab la commande `chepol` permet de les construire, if faut préciser le degré et la variable de sortie.

```
--> chepol(4,'x')
ans=

1 -8x +8x  2
```

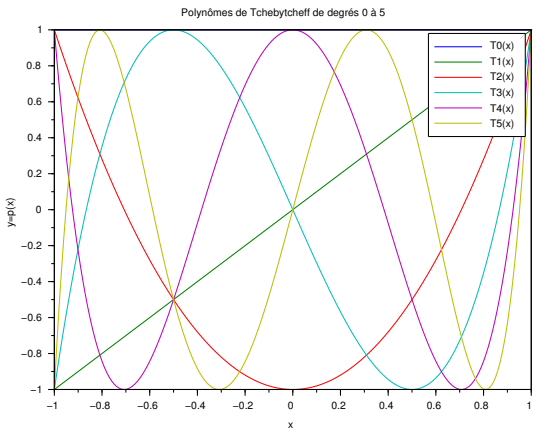


Figure 1: Polynômes de Tchebytcheff $T_N$, $n = 0, \cdots, 5$

## 2.3 Polynômes de Legendre

Ces polynômes possèdent (notamment) les propriétés suivantes

**Théorème 6 (Polynômes de Legendre)**

- orthogonalité

$$\int_{-1}^{1} L_n L_m dx = 0 \text{ si } n \neq m,$$

- relation de récurrence à 3 termes

$$L_0 = 1,\ L_1 = x,\ (n+1)L_{n+1} = (2n+1)xL_n - nL_{n-1}, \forall n \geq 1.$$

Si `x` est un vecteur d'abscisses, `legendre(n,0,x)` retourne les valeurs du polynôme de Legendre de degré $n$ en `x`.

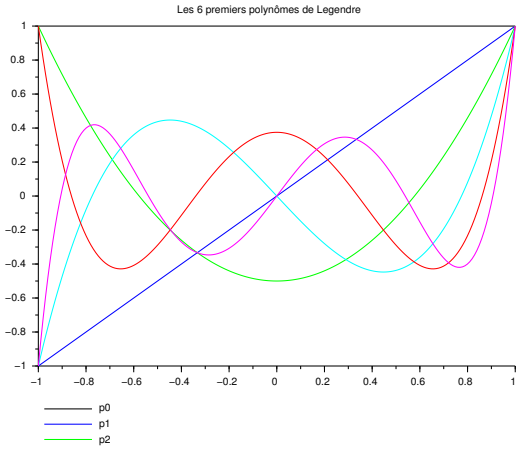


Figure 2: Polynômes de Legendre $L_N$, $n = 0, \cdots, 5$

### 2.4 Exercices

Scilab possède une commande d'intégration numérique `intg`. On peut l'utiliser ainsi :

```
//
// définition de la fonction
//
function y=f(x)
y=sqrt(1-x^2)
endfunction
//Calcul approché de l'intégrale
I=4*intg(0,1,f)
I =

3.1415927
```

**Exercice 2 (Vérification numérique de la relation d'orthogonalité)**

1. Représenter graphiquement les quantités
$$\int_{-1}^{1} L_n L_m dx \text{ et } \int_{-1}^{1} T_n T_m \frac{1}{\sqrt{1-x^2}dx}.$$
Pour quelques valeurs de $n$ et de $m$. Qu'observez-vous ?
2. Calculer les valeurs numériques de ces intégrales.

**Exercice 3 (Racines)**

1. Calculer les racines des polynômes de Tchebytcheff $T_n$ pour $n = 1, \cdots, 6$. Les représenter graphiquement avec des couleurs différentes.Que remarquez vous ?
2. Mêmes question avec les polynômes de Legrendre.

## 3 Difficulté du calcul des racines

Le calcul des racines d'un polynôme est une tâche très difficile pour plusieurs raisons :

la première, très profonde est une conséquence du théorème d'Abel :

**Théorème 7**
Pour tout entier $n \geq 5$, il n'existe pas de formule générale exprimant *par radicaux* les racines d'un polynôme quelconque de degré $n$, c'est-à-dire de formule n'utilisant que les coefficients, la valeur 1, les quatre opérations et l'extraction des racines n-ièmes.

La seule façon de calculer les racines d'un polynôme est donc d'utiliser une méthode itérative adéquate.

La seconde, est liée au très mauvais conditionnement des polynômes et la sensibilité des racines aux variations des coefficients. Pour s'en convaincre, voici un exemple simple et fameux du à Wilkinson. On considère la famille de polynômes
$$p_n(x) = \prod_{i=1}^{n} (x - i).$$
Le calcul numérique des racines de $p_n(x)$ retourne des résultats très éloignés des valeurs $\{1, 2, \cdots, n\}$ dès que $n \geq 25$, voir la figure ci-après.

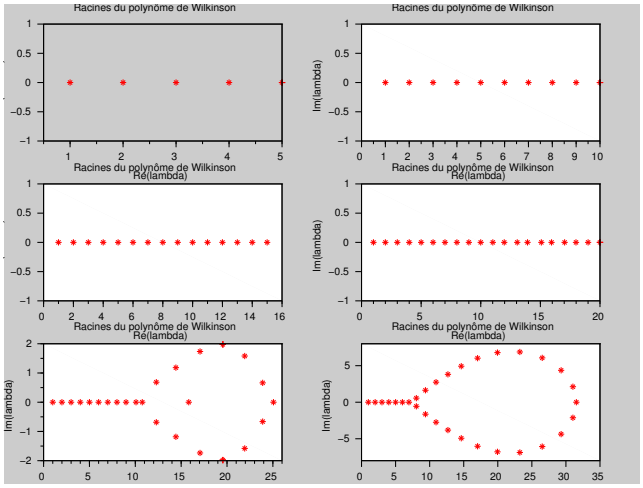


Figure 3: Racines du polynôme de Wilkinson pour, $n = 5, 10, 15, 20, 25, 30$

# 4 Interpolation polynomiale

Soit $f$ continue sur $[a, b]$ et $x_i$, $i = 1, \cdots, n+1$, $n+1$ points 2 à 2 disctincts. On cherche à construire un polynôme $p(x)$ tel que

$$(\mathcal{P}) \quad f(x_i) = p(x_i), i = 1, \cdots, n+1.$$

Ce problème équivaut à déterminer les réels $(a_i)_{i=0}^n$ tels que

$$p(x_i) = \sum_{i=0}^{n} a_j x_i^j.$$

Le polynôme $p$ peut se construire de diverses manières, par exemple en résolvant le Vandermonde de matrice (voir aussi la feuille Etude de fonctions)

$$V = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_1^n \\ \vdots & & & & \\ 1 & x_k & x_k^2 & & x_k^n \\ \vdots & & & & \\ 1 & x_{n+1} & x_{n+1}^2 & \cdots & x_{n+1}^n \end{pmatrix},$$

L'hypothèse $x_i$ 2 à 2 disctincts est nécessaire et suffisante pour avoir l'existence et l'unicité du polynôme d'interpolation.

# 5 Approximation polynômiale

Il s'agit ici d'illustrer quelques techniques d'approximation polynomiale et, notamment d'apprécier visuellement sur des exemples, la vitesse de convergence de l'approximant vers la fonction considérée. Ces expériences numériques ont pour but de susciter des questions.

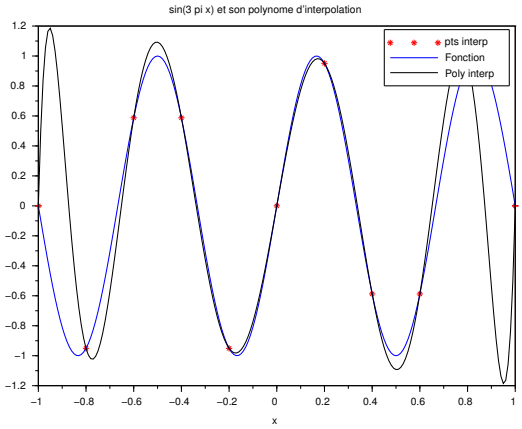


Figure 4: La fonction $f(x) = \sin(3\pi x)$ et son polynôme d'interpolation de degré 10 sur $[-1, 1]$

**Exercice 4 (Comparaison des vitesses de convergence)**

1. Soit $f \in \mathcal{C}([0,1])$ avec $f(0) = f(1) = 0$. On rappelle que la suite des polynômes de Bernstein $B_n(f)(x) = \sum_{k=0}^{n} \begin{pmatrix} n \\ k \end{pmatrix} x^k (1-x)^{n-k} f(\frac{k}{n})$ associée converge uniformément vers $f$. Cette convergence peut-être très lente.
   (a) Soit $f(x) = sin(\pi x), x \in [0,1]$. Construire la suite $B_n(f)$ pour $n = 10, 20, 30 \cdots 100$ pour et les représenter graphiquement avec $f$.Que remarquez-vous ?
   (b) Même question avec $f(x) = \sin(144x(1-x))$.
2. Mêmes questions avec le polynôme d'interpolation de $f$ aux points $x_i = \frac{(i-1)}{n}, i = 1, \cdots, n+1$.

**Exercice 5 (Phénomène de Runge)**

Considérons la fonction $f(x) = \dfrac{1}{1+16x^2}$, $x \in [-1, 1]$.

1. Construire la suite $p_n$ de polynômes d'interpolation de $f$ aux points $x_i = -1 + \frac{2(i-1)}{n}, i = 1, \cdots n+1$.
2. Représenter $f$ et les $p_n$. Qu'observez-vous ?

# 6 Annexe

## 6.1 Quelques commandes en algèbre linéaire avec Scilab

### 6.1.1 Matrices particulières

- Matrice simple avec coefficients rentrés au clavier : `A=[1,2;3,4]`
- Matrice $n \times m$ à coefficients suivant une loi uniforme sur $[0, 1]$ : `rand(n,m)`
- Matrice identité $n \times n$ : `eye(n,n)` ou plus généralement `eye(n,m)`
- Matrice composée uniquement de 0 : `zeros(n,n)` ou plus généralement `zeros(n,m)`
- Matrice composée uniquement de 1 : `ones(n,n)` ou plus généralement `ones(n,m)`

## 6.2 Structures

- Partie triangulaire inférieure de $A$ : `liu(A)`
- Partie triangulaire supérieure de $A$ : `triu(A)`
- Partie diagonale de $A$ : `diag(diag(A))`

### 6.2.1 Produits

Soient $u$ et $v$ deux vecteur colonnes de même taille ; `u'` est le vecteur (ligne) transposé de `u`.

- `u'*v` : produit scalaire de `u` par `v`
- `u*v'` : produit extérieur de `u` par `v`
- `cross(u,v)` : produit vectoriel de `u` par `v` (en dimension 3)

Soient $A$ et $B$ deux matrices carrées de même taille ; `A'` est la matrice transposé de `A`.

- `A*B` : produit de A par B
- `A.*B` : produit d'Hadamard de A par B, i.e., composante par composante

### 6.2.2 Normes, spectre et rayon spectral

- Norme 2 de $A$ : `norm(A)` ou `norm(A,2)`
- Norme 1 de $A$ : `norm(A,1)`
- Norme infine de $A$ : `norm(A,'inf')`
- Norme de Frobenius de $A$ : `norm(A,'fro')`

## 6.3 Quelques fonctions

Soit $A$ une matrice carrée

- Déterminant de $A$ : `det(A)`
- Polynôme caractéristique de $A$ : `p=poly(A, 's')`
- Racines de $p$ : `roots(p)`
- Rang de $A$ : `rank(A)`
- Noyau de $A$ : `kernel(A)`
- Conditionnement de $A$ : `cond(A)`

- Extraction d'une base orthonormée des colonnes de $A$ : `orth(A)`
- Trace de $A$ : `trace(A)`
- Inverse de $A$ : `inv(A)`
- Valeur minimale des coeffs de $A$ : `min(min(A))`
- Valeur maximale des coeffs de $A$ : `max(max(A))`

### 6.3.1 Réductions

- Décomposition de Schur de $A$ : `[T,U]=schur(A)` Test la matrice triangulaire supérieure et `U` la matrice unitaire.
- Valeurs propres de $A$ : `spec(A)`
- Vecteurs et valeurs propres de $A$ : `[R,diagevals]=spec(A)`, `R` contient les vecteurs propres et `diagevals` la matrice diagonale associée
- Décomposition en valeurs singulières : `svd(A)`

### 6.4 Interpolation polynomiale

```
//************************************
//Construction du polynôme d'interpolation
// via la résolution du système de Vandermonde
//
//************************************
clear all
clf
//Vandermonde
x=-1:0.2:1;
n=max(size(x))-1;
V=zeros(n+1,n+1);
for i=0:n
V(:,i+1)=x.^i;
end
y=sin(3*%pi*x)';
//
coco=V\ y;
pp=poly(coco,'x','coeff')

xg=-1:0.01:1;
yg=sin(3*%pi*xg)';

px=horner(pp,x);
pxg=horner(pp,xg);
plot(x,px,'*r')
plot(xg,yg)
plot(xg,pxg,'-k')
legend('pts interp','Fonction','Poly interp')
title("sin(3 pi x) et son polynome d"'interpolation")
xlabel('x')
drawnow
```

### 6.5 L'exemple de Wilkinson

```
k=1;
for n=5:5:30
q=poly(1:n,"x");
r=roots(q);
subplot(3,2,k),plot(real(r),imag(r),'*r')
title('Racines du polynme de Wilkinson')
xlabel('Re(lambda)')
ylabel('Im(lambda)')
k=k+1;
drawnow
end
```

### 6.6 Polynômes de Tchebytcheff et Legendre

```
//*********************************
//Représentation graphique des
//Polynômes de Tchebytcheff
//*********************************
clear all
clf
x=-1:0.01:1;
p0=chepol(0,'x');
p1=chepol(1,'x');
p2=chepol(2,'x');
p3=chepol(3,'x');
p4=chepol(4,'x');
p5=chepol(5,'x');
//
y0=horner(p0,x);
y1=horner(p1,x);
y2=horner(p2,x);
y3=horner(p3,x);
y4=horner(p4,x);
y5=horner(p5,x);
plot(x,y0,x,y1,x,y2,x,y3,x,y4,x,y5)
title('Polynômes de Tchebytcheff de degrs 0 à 5')
xlabel('x')
ylabel('y=p(x)')
legend('T0(x)','T1(x)','T2(x)','T3(x)','T4(x)','T5(x)')

// représentation graphique des 6 premiers polynômes de Legendre sur (-1,1)
l = nearfloat("pred",1);
x = linspace(-l,l,200)';
y = legendre(0:5, 0, x);
clf()
plot2d(x,y', leg="p0@p1@p2@p3@p4@p5@p6")
xtitle("6 premiers polynômes de Legendre")
```