# MATLAB : OPTIMISATION DE PARAMÈTRES

## 1. Ajustement d'un polynôme

Les équations permettant d'obtenir les meilleurs coefficients d'un polynôme de degré *n* sur un ensemble de points $(x_i, y_i)$ vous ont été présentées en cours. Que faire toutefois si les incertitudes sur la variable indépendante *x* ne sont pas négligeables face à celles de la variable dépendante *y* ? On doit alors modifier l'incertitude de chaque point de façon à tenir compte de l'incertitude sur *x* :

$$\sigma_i^2 = (\Delta y_i)^2 + \left[ \Delta x_i \left. \frac{\partial y}{\partial x} \right|_{x=x_i} \right]^2 .$$

Il faut donc prévoir une procédure itérative où les coefficients du polynôme sont initialement déterminés sans mettre les incertitudes en *x* de façon à pouvoir évaluer $\partial y / \partial x$. En général, deux itérations suffisent. La fonction *polyajus* listée ci-dessous effectue cette procédure.[1]

### Fonction polyajus (disponible sur le site du cours)

```
function [p,dp,chi2n,epsilon]=polyajus(x,sigx,y,sigy,n)
% function [p,dp,chi2n,epsilon]=polyajus(x,sigx,y,sigy,n)
%
% ajustement polynomial d'ordre n:
%
%           y=p(1)*x^n+p(2)*x^(n-1)+...+p(n)*x+p(n+1)
%
% ENTRÉE:
% x,y: données;
% n: ordre du polynôme;
% sigx: incertitudes sur x; si égales, entrer le chiffre seulement;
%     si inconnues ou négligeables, mettre 0
% sigy: incertitudes sur y; si égales, entrer le chiffre seulement;
%     si inconnues ou négligeables, mettre 0;
%
% SORTIE:
% p: coefficients du polynôme;
% dp: incertitudes sur les coefficients;
% chi2n: chi carré réduit ( chi2n=chi2/(#points-n-1) );
% epsilon: matrice d'erreur (ou de variance);
%
```

[1] La fonction *polyfit* de Matlab trouve également les meilleurs coefficients d'un polynôme sans toutefois tenir compte des incertitudes sur les points. La fonction *polyval* de Matlab permet d'évaluer la valeur du polynôme en un point *x* donné.

```
npoints=length(x);
test=1;
iteration=0;
while test
  if sigx==0
    test=0;
  end
  if iteration==0
    if sigy==0
      sig2=1 ;
    else
      sig2=sigy.^2 ;
    end
  else
    sig2=sigy.^2 +sigx.^2.*polyval(p(1:n-1),x).^2;
    if iteration==2
      test=0;
    end
  end
  beta=zeros(1,n+1);
  alpha=zeros(n+1,n+1);
  for i=1:n+1
    beta(i)=sum(y.*x.^(n+1-i)./sig2);
    for j=i:n+1
      alpha(i,j)=sum(x.^(n+1-i).*x.^(n+1-j)./sig2);
      alpha(j,i)=alpha(i,j);
    end
  end
  epsilon=inv(alpha);
  p=(beta*epsilon)';
  iteration=iteration+1;
end
chi2n=sum((y-polyval(p,x)).^2./sig2)/(npoints-n-1);
if (sigx==0 & sigy==0)
  epsilon=epsilon*chi2n;
end
dp=sqrt(diag(epsilon));

% fin de la fonction polyajus
```

## 2. Ajustement par moindre carrés d'une fonction quelconque

Soit une série de **N** points expérimentaux $(x_i, y_i)$ avec des incertitudes $\sigma_i$. À ces données, on associe un modèle $\mathbf{F(p_k, x)}$ qui contient **n** paramètres $\mathbf{p_k}$. Une procédure d'optimisation consiste à trouver le jeu de paramètres $\mathbf{\{p_k\}_{opt}}$ qui minimise le $\chi^2$, défini par la relation

$$\chi^2 = \sum_i \frac{1}{\sigma_i^2}\left[y_i - F(x_i)\right]^2 \quad . \qquad (1)$$

Bien entendu, dans le cas où $\mathbf{F(p_k,x)}$ dépend linéairement des paramètres ajustables $\mathbf{p_k}$, une procédure de régression linéaire telle que décrite dans le cas d'un ajustement polynomial peut être appliquée. Sinon, la procédure d'optimisation consiste à partir d'un jeu de paramètres $\mathbf{\{p_k\}_o}$ initiaux et de les varier jusqu'à ce qu'un minimum du $\chi^2$ ait été obtenu. Il est à noter que, peut importe la méthode utilisée, on n'obtient qu'un minimum local du $\chi^2$. Il revient à l'utilisateur de s'assurer que ce minimum est effectivement global.

Une fois le jeu de paramètres optimal $\mathbf{\{p_k\}_{opt}}$ trouvé, les incertitudes $\mathbf{\Delta p_k}$ sont donnés par les éléments diagonaux de la matrice d'erreur $\boldsymbol{\varepsilon}$ (l'inverse de la matrice de courbure) :

$$\Delta p_k = \sqrt{\varepsilon_{kk}} \ , \tag{2a}$$

avec

$$\varepsilon = \alpha^{-1}$$

et

$$\alpha_{ij} = \frac{1}{2}\frac{\partial^2 \chi^2}{\partial p_i \partial p_j}\bigg|_{\{p_k\}_{opt}} \ , \tag{2b}$$

où $\alpha$ est la matrice de courbure. Lorsqu'il n'y a pas de corrélation entre les paramètres, les termes non diagonaux de $\varepsilon$ sont nettement plus petits que les termes diagonaux. Dans le cas contraire, il faut prendre les incertitudes $\Delta p_k$ avec des « pincettes ».

Les dérivées partielles peuvent être calculées analytiquement si la fonction à ajuster est suffisamment simple, ou numériquement selon :

$$\frac{\partial \chi^2}{\partial p_i} \approx \frac{\chi^2(\ldots, p_i + \Delta p_i, \ldots) - \chi^2(\ldots, p_i - \Delta p_i, \ldots)}{2\Delta p_i} \ , \tag{3a}$$

$$\frac{\partial^2 \chi^2}{\partial p_i^2} \approx \frac{\chi^2(\ldots, p_i + \Delta p_i, \ldots) - 2\chi^2(\ldots, p_i, \ldots) + \chi^2(\ldots, p_i - \Delta p_i, \ldots)}{\Delta p_i^2} \tag{3b}$$

$$\frac{\partial^2 \chi^2}{\partial p_i \partial p_j} \approx \frac{1}{4\Delta p_i \Delta p_j}\Big[\chi^2(\ldots, p_i + \Delta p_i, \ldots, p_j + \Delta p_j, \ldots) - \chi^2(\ldots, p_i + \Delta p_i, \ldots, p_j - \Delta p_j, \ldots) - \chi^2(\ldots, p_i - \Delta p_i, \ldots, p_j + \Delta p_j, \ldots) + \chi^2(\ldots, p_i - \Delta p_i, \ldots, p_j - \Delta p_j, \ldots)\Big] \tag{3c}$$

Avant d'aborder les routines d'optimisation disponibles dans Matlab, il vaut la peine de s'attarder sur deux points fondamentaux.

i) Si le modèle $\mathbf{F(p_k,x)}$ est approprié, le chi carré normalisé $\chi_N^2 = \chi^2/(N-n-1)$ doit être voisin de 1. Cela signifie tout simplement que si le modèle est approprié, l'écart

entre celui-ci et les points expérimentaux devrait être en moyenne $\sigma_i$, pourvu bien sûr que les incertitudes soient réalistes.

ii) Si les incertitudes $\sigma_i$ ne sont pas a priori connues, on peut quand même effectuer l'optimisation en posant

$$\chi^2 = \sum [y_i - F(p_k, x_i)]^2$$

et

$$\sigma^2 = \chi_N^2 = \chi^2 / (N - n - 1).$$

On ne pourra par contre tester la validité du modèle puisque $\chi_N^2 / \sigma^2$ vaut d'office 1.

Dans Matlab, la routine la plus flexible pour optimiser des paramètres est *fminunc* (vient de *function*, *minimization* et *unconstrained*).

Pour fixer les idées, soit le cas où on cherche à trouver les valeurs de C, L et $R_L$ d'un circuit RLC série dont on a mesuré la courbe de résonance. Les données sont sous la forme ($x_i$, $y_i$, $\sigma_i$), où $x_i$ est la fréquence, $y_i$ le rapport V/E et $\sigma_i$ l'incertitude sur $y_i$. Outre les trois paramètres ajustables C, L et $R_L$, on désire ajouter au modèle un quatrième paramètre fixe R, qui correspond à la valeur (connue) de la résistance sur laquelle V a été mesuré.

On commence par définir le modèle :

```
function y=vse(p,x,r)
% p=[c*1e9,l,rl]
%
% c est en nanofarad pour que les valeurs à optimiser ne soient pas trop
% éloignées de l'unité

c=p(1)*1e-9;
l=p(2);
rl=p(3);
omega=2*pi*x;
z=r+rl+i*(omega*l-1./(omega*c));
y=abs(r./z);
```

Les paramètres à optimiser **doivent** être regroupés à l'intérieur d'une seule variable, ici le vecteur p, qui **doit** être le premier argument de la fonction modèle.

On définit ensuite une fonction très simple qui calcule le $\chi^2$ :

```
function chi2=chivse(p,x,y,sig,r)
% mettre sig=1 si sig est inconnu.

chi2= sum(((y-vse(p,x,r))./sig).^2);
```

Finalement, dans Matlab, on définit un vecteur de valeurs initiales $p_o$ et on appelle ***fminunc*** :

```
» [p,chi2]=fminunc('chivse',p0,[ ],x,y,sig,r);
```

***fminunc*** utilise certaines valeurs défaut contenues dans la structure *options.* Généralement, ces valeurs défauts sont convenables et on substitue la matrice vide [ ] à la structure *options* dans la liste des arguments d'entrée de ***fminunc***. La liste des paramètres qui suit cette matrice vide doit être dans le même ordre que lors de la définition de la fonction *chivse*.

Les incertitudes sur les paramètres p optimisés peuvent être obtenues avec la fonction *erreur*, qui calcule les éléments de la matrice de courbure $\alpha$ à partir des équations (3b) et (3c) :

**Fonction erreur (disponible sur le site du cours)**

```
function [dp,epsilon,chicarre]=erreur(fonct,poptim,facteur,x,y,sig,varargin)
% function [dp,epsilon,chicarre]=erreur(fonct,poptim,facteur,x,y,sig,varargin)
%
% ENTRÉE :
% fonct : nom de la fonction
% poptim : paramètres optimisés
% facteur :Delta p_i=|facteur X p_i|; 0.01 suffit  généralement
% x,y : données
% sig : incertitudes; sig=1 si inconnu
% varargin : structure qui prend en charge toutes les autres variables d'entrée requises par fonct
%       fonct doit être sous la forme y=fonct(poptim,x,autres arguments )
%
% SORTIE
% dp :incertitudes sur les paramètres
% epsilon : matrice d'erreur
% chicarre : chi carré

n=length(poptim);
N=length(y);
alpha=zeros(n);
dp=zeros(size(poptim));
di=abs(poptim*facteur);
z=feval(fonct,poptim,x,varargin{ :});
chicarre=sum(((z-y)./sig).^2);
if sig==1
        sig=sqrt(chicarre/(N-n-1));
        chicarre=chicarre/sig^2;
end

for i=1 :n
        p=poptim;
        p(i)=p(i)+di(i);
        z=feval(fonct,p,x,varargin{ :});
        chi1=sum(((z-y)./sig).^2);
        p=poptim;
```

```
        p(i)=p(i)-di(i);
        z=feval(fonct,p,x,varargin{ :});
        chi2=sum(((z-y)./sig).^2);
        alpha(i,i)=(chi1-2*chicarre+chi2)/2/di(i)/di(i);
        for j=i+1 :n
                p=poptim;
                p(i)=p(i)+di(i);
                p(j)=p(j)+di(j);
                z=feval(fonct,p,x,varargin{ :});
                chi1=sum(((z-y)./sig).^2);
                p=poptim;
                p(i)=p(i)+di(i);
                p(j)=p(j)-di(j);
                z=feval(fonct,p,x,varargin{ :});
                chi2=sum(((z-y)./sig).^2);
                p=poptim;
                p(i)=p(i)-di(i);
                p(j)=p(j)+di(j);
                z=feval(fonct,p,x,varargin{ :});
                chi3=sum(((z-y)./sig).^2);
                p=poptim;
                p(i)=p(i)-di(i);
                p(j)=p(j)-di(j);
                z=feval(fonct,p,x,varargin{ :});
                chi4=sum(((z-y)./sig).^2);
                alpha(i,j)=(chi1-chi2-chi3+chi4)/8/di(i)/di(j);
                alpha(j,i)=alpha(i,j);
        end
end
epsilon=inv(alpha);
dp=sqrt(diag(epsilon));
```