Aide-mémoire SCILAB

1. Divers

`help(func)` Aide en ligne

`clc` Efface la console

`clear` Réinitialise l'environnement (suppression de toute les variables)

`clear X` Supprime la variable X

`stacksize(n)` Redéfini l'espace mémoire pour les variables

`//` Commentaire sur une ligne

`exec('filename.sce')` Exécution d'un script

`getf('func')` Chargement d'une fonction à partir d'un fichier `sci`

`errcatch(-1,'pause')` Permet d'interrompre l'exécution à la prochaine erreur rencontrée (**très utile pour la mise au point**)

1.1 Raccourcis claviers Console

`F1` Ouvrir l'aide

`F2` Effacer la console

1.2 Raccourcis claviers Scinote

`F5` Éxécution du fichier, sans écho

`CTRL+l` Éxécution du fichier, avec écho

`CTRL+e` Éxécution sélection courante (ou jusqu'au curseur), avec écho

`CTRL+d` Commenter la sélection

`CTRL+D` Dé-commenter la sélection

2. Matrices

2.1 Création des matrices

`[a b c;d e f] = [a,b,c;d,e,f]` $= \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix}$

`A=matrix(v,n,m)` Reformate le vecteur / matrice v sous la forme d'une matrice m par n

`zeros(m,n)` Matrice nulle : $\mathbb{R}^{\mathbb{N}_m \times \mathbb{N}_n} = (i,j) \mapsto 0$

`ones(m,n)` $\mathbb{R}^{\mathbb{N}_m \times \mathbb{N}_n} = (i,j) \mapsto 1$

`rand(m,n)` Matrice aléatoire : $\mathbb{R}^{\mathbb{N}_m \times \mathbb{N}_n} = (i,j) \mapsto U(0,1)$

`eye(m,n)` Matrice identité : $\mathbb{R}^{\mathbb{N}_m \times \mathbb{N}_n} = (i,j) \mapsto \delta_{i=j}$

`diag([a,b,c])` Matrice diagonale : $\begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix}$

`toeplitz([a,b,c])` $\begin{pmatrix} a & b & c \\ b & a & b \\ c & b & a \end{pmatrix}$

2.2 Création de vecteurs

`[a1 a2 ... an]` ou `[a1,a2,...,an]` Vecteur ligne

`[a1;a2;...;an]` (ou retours à la ligne) Vecteur colonne

`x'` Transforme un vecteur ligne en vecteur colonne ou réciproquement

`n1:n2` Intervalle entier : $[n_1, n_1+1, n_1+2, \cdots, n_2-1, n_2]$

`min:s:max` n échantillons entre min et max séparé du pas s

`linspace(min,max,n)` n valeurs entre min et max

`logspace(e1,e2,n)` n valeurs entre 10^{e_1} et 10^{e_2}

2.3 Lire les dimensions d'une matrice / d'un vecteur

`size(x)` `[size(x,1) size(x,2) ...]` (en fonction du nombre de dimensions)

`size(x,s)` Nombre d'éléments suivant la dimension s

`length(x)` Nombre total d'éléments (= nombre d'éléments pour un vecteur ligne ou colonne, ou nombre de ligne fois nombre de colonnes pour une matrice)

2.4 Accès aux éléments d'un vecteur

Les éléments d'une matrice peuvent être accédés aussi bien en lecture qu'en écriture :

`x(i)` Accès à l'élément i (attention : l'index du premier élément est 1, et non 0 comme en C),

`x(i:j)` Extraction du sous-vecteur $x_i \dots x_j$

`x(i:$)` Extraction du sous-vecteur $x_i \dots x_n$

2.5 Accès aux éléments d'une matrice

Les éléments d'une matrice peuvent être accédés aussi bien en lecture qu'en écriture :

`A(i,j)` Accès à l'élément situé en ligne i et colonne j

`A(:,j)` Extraction d'une ligne entière

`A(i,:)` Extraction d'une colonne entière

`A(i1:i2,j1:j2)` Extraction d'une sous-matrice

`diag(A)` Extraction de la diagonale d'une matrice sous forme de vecteur colonne

2.6 Opérations matricielles élémentaires

`A'` Transposée (et conjugaison si nombres complexes) : A^H

`A.'` Transposée seule : A^T

`M=A*B` Produit matriciel : $M = AB$

`A+B,A-B` Addition / soustraction,

`A.*B, A.^B, A./B` Produit / exposant / division terme à terme,

`x=A\b` Division à gauche : $x = A^{-1}b$ si A est carrée et non singulière, sinon solution au sens des moindres carrés.

`X=B/A` Division à droite : résolution de $XA = B$

`[m[,k]] = min(M), resp. max(M)` Minimum ou maximum d'une matrice / vecteur (et k = position de l'extrema)

3. Constantes

`%pi, %e, %i, %eps, %inf, %nan, %t, %f`

4. Fonctions scalaires

`floor(x), ceil(x), fix(x), round(x)` Fonctions d'arrondis, respectivement, à l'entier inférieur, supérieur, vers zéro, au plus proche.

`exp(x)` Exponentielle complexe ou réelle

`log(x),log10(x),log2(x)` Logarithmes, resp. naturel, base 10 et base 2

`sqrt(x)` $\sqrt{x}$

4.1 Fonctions trigonométriques

`sin, cos, tan, asin, acos, atan, sinh, cosh, ...` fonctions usuelles (unité d'angle = radians)

`sind, cosd, ...` variantes avec angles en degrés

`sinc` Sinus cardinal

5. Nombres complexes

`%i` $\mathbf{i}$

`exp(a+b*%i)` $e^{a+b\mathbf{i}}$

`abs(z)` $|z|$ (ou valeur absolue pour un réel)

`conj(z)` Conjugé $z^\star$

`real(z), imag(z)` Parties réelles et imaginaires,

`[rho,theta]=polar(z)` Forme polaire (norme et angle)

6. Polynômes

`poly(vec,'x'[,'r'])` Définition d'après les racines

`poly(vec,'x','c')` Définition d'après les coefficients

`poly(M,'x')` Pôl. caractéristique de la matrice : $\det(M - xI)$

`coeff(p)` Conversion polynôme vers vecteur ligne

`degree(p)`

`roots(p)` Racines d'un polynôme

`denom(f), numer(f)` Dénominateur et numérateur d'une fraction rationnelle

`factors(p)` Factorisation (polynôme ou fraction rationnelle)

`horner(p,x)` Évaluation de $p(x)$ (p polynôme ou fraction rationnelle)

`[pgcd]=gcd(p)` PGCD d'un ensemble de polynômes / entiers, sous forme de vecteur ligne.

`[ppcm]=lcm(p)` Idem, PPCM.

6.1 Polynômes prédéfinis

`%s = poly(0,'s'), %z = poly(0,'z')`

7. Tracé de courbes

7.1 Gestion des figures

`xinit()` Ouvre une fenêtre graphique vierge

`scf(n=1,2,...)` Défini la figure courante,

`h = gcf()` Récupère le handle de la figure courante,

`clf()` Efface la figure courante

`xs2png(n,file)` Export de la figure spécifiée (n) sous forme d'image (nombreux format possibles : `xs2jpg, xs2pdf, xs2svg, xs2eps, xs2bmp, ...`)

7.2 Dessin 2D

`plot([x,],y,opt)` Options possibles :

type de ligne : `'-'|'--'|':'|'-.'`

Couleurs : `'r'|'g'|'b'|'c'|'m'|'y'|'k'|'w'`

type de marqueur : `'+'|'o'|'*'|'.'|'x'|'s'|'d'|'^'|'v'|'>'|'<'|'p'`

`plot2d(x,f,style)`

`errbar(x,y,em,ep)` Ajout de barres d'erreur (`[-em,ep]` = intervalle de confiance)

`Matplot(M)` Représentation d'une matrice sous la forme d'une échelle de couleurs

`Sgrayplot(x,y,z)` Idem, avec lissage (interpolation)

`champ(x,y,fx,fy)` Tracé d'un champs de vecteurs 2D

`histplot(n,x)` Histogramme (n = nombre de classes)

`polarplot(theta,rho)` Tracé en fonction des coordonnées polaires

`pie(x,txt)` Diagramme de type camembert

`plot2d2(x,y)` Dessin par paliers horizontaux

`plot2d3(x,y)` Dessine des barres verticales

8. Dessin 3D

8.1 Spécification des mailles

Mailles orthogonales, forme compacte : $x(m\times 1), y(n\times 1)$ → `surf,mesh,plot3d,plot3d1,fplot3d,fplot3d1`

Mailles rectangulaire (facettes contigües à 4 points) : $(X_{i,j}, Y_{i,j}, Z_{i,j})$ → `surf,mesh,plot3d2,plot3d3`. Cas particulier : mailles orthogonales sous forme redondante : x : lignes identiques, et y : colonnes identiques.

Facettes à n_f points : $X(1:n_f,i), Y(1:n_f,i), Z(1:n_f,i)$ → `plot3d,plot3d1`

8.2 Pour générer les grilles

`[xx,yy,zz]=genfac3d(x,y,z)` $xx(1:4,i), yy(1:4,i), zz(1:4,i)$ sont les coordonnées des quatre points de la facette i. En entrée, grille orthogonale spécifiée de manière compacte : $x(1\times m), y(1\times n), z(m\times n)$

`[X,Y]=ndgrid(x,y)` Construit une grille orthogonale $(X_{i,j}, Y_{i,j})$ à partir de (x_i) et (y_i) : $\forall i,j, X(i,j) = x(i)$, et $Y(i,j) = y(j)$

`[X,Y]=meshgrid(x,y)` Construit une grille orthogonale $(X_{i,j}, Y_{i,j})$ à partir de (x_i) et (y_i).

8.3 Pour dessiner

`surf([X,Y,]Z)` Tracé d'une surface 3D. En entrée : grille orthogonale, forme redondante ou compacte. Couleurs en fonction de Z ou spécifiée pour chaque point.

`mesh([X,Y],Z)` Tracé d'une surface 3D. Version spécialisée de `surf` sans affichage des couleurs (affichage seulement du maillage).

`plot3d(x,y,z)` Tracé d'une surface 3D : accepte une grille orthogonale (forme compacte) ou des facettes à `nf` points

`plot3d1(x,y,z)` Idem plot3d, la couleur dépend de z.

`plot3d2(X,Y,Z)` Tracé de surfaces arbitraires à partir de facettes rectangulaires

`plot3d3(X,Y,Z)` Idem mesh, avec même paramètres que plot3d2.

`fplot3d(x,y,f)` Tracé d'une surface 3D à partir d'une fonction (x,y : grille orthogonale forme compacte). Note : `fplot3d1` : idem avec des couleurs.

`contour(x,y,z,nz)` Courbes de niveaux. Grille orthogonale sous forme compacte. `nz:` : nombre de niveaux.

`param3d(x,y,z)` Courbe paramétrique $x(t), y(t), z(t)$. x,y,z = 3 vecteurs de même dimension.

`f = scf()` Récupère le handle de la figure courante

`f.color_map = ...` Définition de l'échelle des couleur. Exemples :

`jetcorlormap(n)` Échelle du bleu foncé vers rouge (n = nombre de teintes)

`graycolormap(n)` Niveaux de gris

8.4 Annotations

`legend(['courbe 1';'courbe 2';...])` Ajoute un cartouche avec la description des différentes courbes tracées.

`title('titre')` Titre de la fenêtre graphique

`xtitle('titre'[,'axe x'[,'axe y'[,'axe z']]])` Défini le titre de la figure et les labels associés aux différents axes

9. Entrées / sorties

9.1 Sortie standard

`disp(var)`

`printf("x = %.3f.", x)`

`format('v'|'e',ndigits)` Sélection du format d'affichage variable et du nombre de décimales à afficher. 'e' : format exponentiel, 'v' : format variable.

9.2 Entrées / sorties fichier

`save('filename'[,x1,x2,...]` Enregistre des variables dans un fichier binaire

`load('filename')` Relis des variables à partir d'un fichier binaire

`M = fscanfMat(file)` Lecture matrice à partir d'un fichier texte

`M = csvRead(file,separator), csvWrite(M,file)` Lecture / écriture fichier CSV (tableur)

`[fd,SST,Sheetnames,Sheetpos]=xls_open('file')` Ouverture d'un fichier Exel

`[M,text]=xls_read(fd,Sheetpos(i)))` Lecture d'une feuille de fichier Exel

`x = loadwave(file) / savewave(file,x,rate)` Lecture / enregistrement d'un fichier audio `.wav`

10. Algèbre linéaire

10.1 Analyse matricielle

`y=norm(x[,flag=2])` Norme induite (`flag=1|2|'inf'`) ou Froebenius (`flag='fro'`)

`rank(A)` Rang

`cond(A[,p=2])` Conditionnement, p = norme souhaitée

`det(A)` Déterminant

`trace(A)` $\sum A_{ii}$

`kernel(A)` Noyau

`[X,dim]=range(A)`

10.2 Équations linéaires

`inv(A)` Inverse d'une matrice carrée

`x = lsq(A,b)` Résolution au sens de moindre carrés de $Ax = b$

`X = pinv(A)` Pseudo-inverse

`[x0,ker] = linsolve(A,b)` Calcul du noyau de A, et d'une solution particulière x_0 de $Ax + b = 0$.

10.3 Décomposition, valeurs propres, et valeurs singulières

`[Q,R]=qr(X)` Décomposition QR : $X = QR$, Q orthogonale, et R triangulaire supérieure

`[L,U]=lu(X)` Décomposition LU : $X = LU$, L triangulaire inférieure, et U triangulaire supérieure

`[L]=chol(X)` (X SDP) Factorisation de Cholesky : $X = L'L$, L triangulaire sup. à coeffs. diagonaux positifs

`[eigenvectors, eigenvalues] = spec(A)` Calcul des valeurs propres / vecteurs propres

`[U,S,V]=svd(X)` Décomposition en valeurs singulières

`[U,T]=schur(A)` Décomposition de Schur : $A = UTU'$, $U'U = I$, T triangulaire sup., forme réelle ou complexe selon A

`[Ab,X,bs]=bdiag(A)` Décomposition en valeurs propres généralisées

10.4 Matrices creuses

`A=sparse(B)` Conversion matrice pleine → matrice creuse

`B=full(A)` Conversion matrice creuse → matrice pleine

`I=speye(n,m)` Matrice identité (creuse)

`sprand(n,m,density,'uniform'|'normal')` Matrice aléatoire (creuse)

10.5 Méthodes itératives

`x=conjgrad(A,b,method[,...])` Gradient conjugé, méthodes disponibles :

- `'pcg'` Conjugate gradient (dans ce cas, A doit être SDP)
- `'cgs'` Conjugate gradient squared
- `'bicg'` Biconjugate gradient
- `'bicgstab'` Biconjugate gradient stabilized (default)

11. Statistiques

`mean(x)` $\mathbb{E}[x]$

`median(x)` Médiane

`s2=variance(x)` $\sigma^2 = \mathbb{E}[x^2] - \mathbb{E}[x]^2$

`s=st_deviation(x)` Écart-type σ

`s=mad(x)` *Mean absolute deviation*

`s=covar(x,y,fre)` Covariance entre 2 variables scalaires. $x, y =$ vecteurs des réalisations, `fre(i,j) =` fréquence de (x_i, y_i).

`cdfxxx('X',...,P,Q)` Fonction cumulative inverse

`p=datafit(G[,DG],Z[W][,contr],p0[,algo])` Soient n_p le nombre de paramètres à estimer, n le nombre d'observations, et n_z le nombre de dimension de chaque observation (entrées et sorties : $n_z = n_x + n_y$) :

`G` Fonction d'erreur `e=G(p,z)` avec `p: np x 1, z: nz x 1, e: ny x 1`

`DG` Dérivées partielles de G (jacobienne) par rapport à p (paramètre). Dimension : `ny x np`

`Z` Matrice $[z_1, z_2, \ldots, z_n]$ où chaque z_i est une mesure (dimension `nz x 1`). Typiquement, $z_i = \begin{pmatrix} y_i \\ x_i \end{pmatrix}$

`W` Matrice de pondération (dimension : `ne x ne`)

`contr` Bornes optionnelles. Format : `'b',binf,bsup`

`p0` Valeur initiale du paramètre (`np x 1`)

`algo` Méthode : `'qn' | 'gc' | 'nd'`

Trouve p minimisant $\sum_{i=1}^{n} \|G(p, z_i)\|_2^2$

Note : basé sur la fonction `optim`

`[a,b,sig]=reglin(x,y)` Régression linéaire

12. Calcul différentiel, intégration

`y=diff(x)` Différenciation terme à terme : $y_n = x_n - x_{n-1}$

`[J[,H]]=numderivative(F,x[,h,order])` Estimation numérique de la jacobienne et éventuellement de la Hessienne.

`y=integrate('f(x)','x',x0,x1)` Intégration numérique à partir d'une expression

`y=inttrap(t,x)` Intégration à partir des valeurs d'une fonction (interpolation linéaire)

`y=intspline(t,x)` Idem (interpolation par splines)

13. Traitement du signal

13.1 Fonctions diverses

`[y[,zf]] = filter(num,den,x[,zi])` Application d'une filtre FIR ou IIR.

`window('re'|'tr'|'hm'|'hn'|'kr',n)` Génération d'une fenêtre de taille n, resp. rectangulaire, triangulaire, Hann, Hamming, ou Kaiser.

`y=convol(h,x)` Produit de convolution : $y_k = \sum h_i x_{k-i}$

`X=fft(x[,signe])` FFT (par défaut), ou FFT inverse (signe = 1), sans normalisation.

`[cov,mean]=corr(x,[y],nlags)` Corrélation / covariance ou auto-corrélation (biaisée) : $\frac{1}{n}\sum_{k=1}^{n-m}(x_k - \bar{x})(y_{k+m} - \bar{y})^\star$

`[c,lag]=xcorr(x[,y][,nlags][,scaling])` Corrélation ou auto-corrélation : $\alpha \sum_{k=1}^{n-m} x_k y^\star_{k+m}$, avec facteur de normalisation biaisé ($\frac{1}{n}$) ou non biaisé ($\alpha = \frac{1}{n-m}$).

`[c,lag]=xcov(x[,y][,nlags][,scaling])` Covariance ou auto-covariance : $\alpha \sum_{k=1}^{n-m}(x_k - \bar{x})(y_{k+m} - \bar{y})^\star$, avec facteur de normalisation biaisé ($\frac{1}{n}$) ou non biaisé ($\alpha = \frac{1}{n-m}$).

`[y] = intdec(x,lom)` Change la période d'échantillonnage par le facteur lom.

13.2 Génération filtre FIR

`[h[,fm,fr]]=wfir(ftype,order,fcuts,wtype,par)` Génération filtre FIR par la méthode de fenêtrage. `type='lp'|'hp'|'bp'|'sb'`. `wtype=` 're' (rectangulaire), 'tr' (triangulaire), 'hm' (Hamming), 'hn' (Hann), 'kr' (Kaiser), 'ch' (Chebychev)

`hn=eqfir(nf,edge,des,wate)` Design FIR suivant critère minimax (utilise `remez`)

`xh=hilb(n)` Approximation FIR de la transformée de Hilbert

13.3 Génération filtre IIR

`iir(n,type, design,frq,delta)` Filtre IIR à partir d'un prototype analogique. `type='lp'|'hp'|'bp'|'sb'` et `design='butt'|'cheb1'|'cheb2'|'ellip'`

`[cells,fact,zeros,poles] = eqiir(type,design,frq,deltap,deltas)` Idem, trouve automatiquement l'ordre nécessaire en fonction des spécs., et factorisation sous forme de sections du second ordre.

`ar=lev(r,n)` Algorithme de Levinson-Durbin pour trouver les coefficient d'un IIR à pôle uniquement (n coefficients) et possédant une séquence d'auto-corrélation donnée (r).

13.4 Analyse d'un filtre

`[xm,fr]=frmag(num,den,npts)` Réponse fréquentielle d'un filtre discret

`plzr(sl)` Dessin des zéros et pôles d'un système linéaire

`[tg,fr]=group(npts,h)` Temps de groupe d'un filtre discret

13.5 Estimation spectrale

`sm=pspect(step,wnd,wtype,x[,y])` Périodogramme (méthode de Welch)

`sm=cspect(nlags,n,wtype,x[,y])` Spectre d'après la méthode de corrélation

`sm=mese(x[,n])` Spectre par maximisation de l'entropie

14. Interpolation

`y=intdec(x,lom)` Ré-échantillonnage, à base de FFT.

`y=interpln(xyd,x)` Interpolation linéaire

`d=spline(x,y[stype])` Calcul des dérivées pour des splines cubiques (`type='natural'|'clamped'|'periodic'|'not a knot'`)

`yp=interp(xp, x, y, d)` Interpolation par des splines cubiques

`yp=interp1(x, y, xp, method)` Interpolation 1D : `method = 'linear'|'spline'|'nearest'`

`zp=interp2d(xp,yp,x,y,C)` Interpolation 2D (`C=` coefficients de la spline obtenus par `splin2d(...)`)

15. Systèmes linéaires invariants dans le temps (LTI)

Un système LTI peut être définit par :

Sa représentation d'état (*state / space form*) :

$$\begin{aligned} kx &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

Sa fonction de transfert : $y(k) = H(k)u(k)$

avec $k = s$ pour un système linéaire continu (SLC), et $k = z$ pour un système linéaire discret (SLD).

`sl=syslin('c'|'d',A,B,C,D,x0)` Définition d'un SL, représentation *state/space*

`sl=syslin('c'|'d',H)` Définition d'un SL, à partir de la fonction de transfert

`sl=syslin('c'|'d',num,den)` Idem (numérateur et dénominateur de la FT)

`sl=tf2ss(h)` Conversion fonction de transfert vers représentation *state/space*

`h=ss2tf(sl)` Conversion réciproque

`[A,B,C,D]=abcd(sl)` Extrait les matrices A, B, C, D à partir d'un SL

Interconnexion des systèmes :

`sl1*sl2` Mise en série de 2 systèmes

`sl1+sl2` 2 systèmes, même entrée, sortie sommée

`sl1,sl2` 2 entrées, sortie sommée

`sl1;sl2` Entrée commune, 2 sorties

`sld=dscr(slc,dt)` Conversion d'un SLC vers un SLD, dt = pas d'échantillonnage

`[y,x]=flts(u,sl[,x0])` Application du SLD sur le signal d'entrée u.

`[y,x]=csim(u,t,sl[,x0[,tol]])` Simulation d'un SLC (à partir de `ode`)

16. Optimisation

16.1 Fonctions non différentiables

`[x,fval]=fminsearch(costf,x0)` version sans contrainte de l'algorithme de Nelder-Mead. Fonction de coût de la forme $y = f(x)$.

`neldermead_xxx(...)` bornes possibles (mais API complexe)

16.2 Fonctions différentiables

`[fopt,xopt]=optim(cosft,[,contr]x0[,algo])` avec :

`costf` : (x,ind) → `[f(x),gradient(x),ind]` ou bien de la forme `list(NDcost,f)` si on ne peut pas fournir le gradient.

`contr` Contraintes optionnelles sur x : `['b',xl,xu]`

`algo` `'qn'` (par défaut) : Quasi-Newton avec BFGS, `'gc'` : BFGS à mémoire limitée (L-BFGS), `'nd'` : non différentiable (algo ≪ min-max ≫)

`fopt,xopt` Valeur optimale et position du minima.

16.3 Moindres carrés non linéaires

`[f,xopt]=leastsq(fun[,Dfun][,contr],x0[,algo])` Fonction basée sur `optim` (quasi-Newton BFGS par défaut)

`[f,xopt]=lsqrsolve(x0,fct,m[,fjac[,stop[,diag]]])` Méthode de Levenberg - Marquardt, avec diag : facteur d'échelles pour les variables.

16.4 Programmation linéaire / quadratique

`xopt=karmarkar(Aeq,beq,c,x0[...])` Réso. pb de prog. linéaire sous forme normale ou générale.

Forme générale

$$\begin{aligned} &\min_x p^T x \\ &c_{inf} \leq x \leq c_{sup} \\ &c_i^T x = d_i, \forall i = 1 \dots n_e \\ &e_i^T x \leq f_j, \forall i = 1 \dots n_i \end{aligned}$$

Forme normale

$$\begin{aligned} &\min_x p^T x \\ &c_i^T x = d_i, \forall i = 1 \dots n_e \\ &x \geq 0 \end{aligned}$$

`xopt=qpsolve(A,p,C,b,ci,cs,ne)` Réso. pb de prog. quadratique :

$$\begin{aligned} &\min_x \frac{1}{2} x^T Q x + p^T x \\ &c_{inf} \leq x \leq c_{sup} \\ &c_i^T x = d_i, \forall i = 1 \dots n_e \\ &e_i^T x \leq f_j, \forall i = 1 \dots n_i \end{aligned}$$

17. Équations différentielles

17.1 Équations différentielles ordinaires

$$\begin{aligned} &\forall t, y'(t) = f(t, y(t)) \\ &y(t_0) = y0 \end{aligned}$$

`y=ode([type,]y0,t0,t[,rtol,atol],f[,jac])`

17.2 Équations différentielles implicites

$$A(t, y)y'(t) = g(t, y), y(t_0) = y_0$$

`y=impl([type,]y0,ydot0,t0,t[,atol,rtol],res,adda[,jac])`

17.3 Équations différentielles algébriques (DAE)

$F(t, y, y') = 0$

$$r = \begin{pmatrix} t_0 \cdots t_N \\ y_0(t_0) \cdots y_0(t_N) \\ \vdots \cdots \vdots \\ y_n(t_0) \cdots y_n(t_N) \\ y_0'(t_0) \cdots y_0'(t_N) \\ \vdots \cdots \vdots \\ y_n'(t_0) \cdots y_n'(t_N) \end{pmatrix}$$

`r=dassl(x0,t0,t[,atol,rtol],res[,jac][,info][hd])`

`[r,nn]=dasrt(x0,t0,t[,atol,rtol],res[,jac],...ng,surf,[,info][hd])` Idem `dassl` avec détection de traversée de surface.

17.4 Problèmes aux valeurs limites (BVP)

$$\begin{aligned} \frac{d^{m_i} u_i}{dx} &= f_i\left(x, u(x), \frac{du}{dx}, \dots, \frac{d^{m_i-1}u}{dx^{m_i-1}}\right), 1 \leq i \leq n_c \\ 0 &= g_j\left(\zeta_j, u(\zeta_j), \dots, \frac{d^{m_\star} u}{dx^{m_\star}}(\zeta_j)\right), 1 \leq j \leq m_\star \end{aligned}$$

Où (ζ_j) définissent les $m_\star$ valeurs limites

`z=bvode(points,ncomp,m,aleft,aright,zeta,ipar,ltol,...tol,fixpnt,fsub1,dfsub1,gsub1,dgsub1,guess1)`

17.5 Problèmes hybrides

Parties discrètes et continues :

$$\begin{aligned} y_c'(t) &= f_0(t, y_c(t), y_d(t)), t_k \leq t < t_{k+1} \\ y_d(t_{k+1}) &= f_1(t_{k+1}, y_c(t_{k+1}), y_d(t_k)) \end{aligned}$$

`yt=odedc(y0,nd,stdel,t0,t,f)`

18. Recherche de racine

`[x]=fsolve(x0,fct[,fjac[,tol]])` Résolution de $f(x) = 0$, f fonction arbitraire.