Syntaxe des fonctions du Kit_ETS_MB pour Nspire CAS (et quelques références à Kit_ETS_FH)

Michel Beaudin
michel.beaudin@etsmtl.ca
Version du 22 avril 2014

Les fonctions qui suivent ont été définies dans le classeur Nspire CAS portant le nom de « Kit_ETS_MB » et sont utiles ─ mais pas indispensables ─ dans certains cours de mathématiques à l'ÉTS. Elles ont l'avantage d'être assez générales et ont été principalement développées en mémoire du logiciel *Derive*. Si vous enregistrez ce classeur, faites-le dans MyLib et rafraîchissez les bibliothèques : cela vous permettra de l'utiliser dans un autre classeur. On trouvera des informations pertinentes à ce sujet à l'adresse http://seg-apps.etsmtl.ca/nspire/documents/librairie%20publique.pdf.

Dans certains cas, des fonctions de la librairie de Kit_ETS_MB en appellent certaines provenant de deux librairies : ETS_specfunc et Kit_ETS_FH. Il est **important** de les placer aussi dans MyLib. Dans le document http://seg-apps.etsmtl.ca/nspire/documents/transf%20Laplace%20prog.pdf, Chantal Trottier présente le fichier ETS_specfunc.tns (on dira souvent le « package Laplace ») qui permet notamment de calculer des transformées directe et inverse de Laplace. Le collègue Frédérick Henri est l'auteur de la plupart des fonctions qu'on retrouve dans le Kit_ETS_FH.tns : cette librairie vient d'être mise à jour après notre collaboration d'hiver 2014. Quelques exemples commentés, relatifs à un cours spécifique, sont inclus dans le PDF *Exemples d'utilisation*.

La version du fichier Kit_EST_MB.tns, de même que celle du présent document et du document « Exemple d'utilisation » fait toujours référence à la date qui y apparaît. Une première version est apparue en décembre 2011, une seconde en avril 2013. Ceci est la troisième. De temps en temps, d'autres fonctions pourront s'y rajouter.

Résultats de la collaboration avec Frédérick Henri

Commencée à l'hiver 2013, poursuivie à l'automne 2013 et à l'hiver 2014, cette collaboration a permis de programmer des fonctions fort utiles permettant à Nspire CAS d'améliorer sa performance concernant l'intégration des fonctions par morceaux. Une description détaillée de tout cela se trouve dans les présentations faites aux conférences ACA 2013 et 2014, disponibles à partir du site suivant: https://cours.etsmtl.ca/seg/mbeaudin/Liste_WEB.pdf.

Nous donnons ici un tableau des plus importantes fonctions de la librairie Kit_ETS_FH. Pour de plus amples renseignements, on consultera tout simplement la librairie (Fred commente toujours très bien ses fonctions!). Aussi, le document d'exemples d'utilisation contient des exemples qui illustrent certaines des fonctions de Fred, décrites dans le tableau qui suit.

Principales fonctions du Kit_ETS_FH

Nom de la fonction	Description
grouper_fct(*f*, *x*)	Si *f* est une somme, différence, produit, quotient ou exponentiation de plusieurs expressions (par morceaux par exemple) de la variable *x*, cette fonction groupe le résultat en une *unique* expression (par morceaux)

integral_mcx(*f*, *x*)	Si l'expression *f* est du type précédent, cette fonction calcule une intégrale indéfinie par rapport à la variable *x*.
integral_mcx_d(*f*, *x*, *a*, *b*)	Comme *integral_mcx* mais pour l'intégrale définie entre *a* and *b*.
unpiece(*f*, *x*)	Si *f* est une expression de la variable *x* définie par morceaux, la fonction la réécrit sous forme de combinaison linéaire de fonctions indicatrices des sous-intervalles (ouverts) de chacun des morceaux.
signtopiece(*f*, *x*)	Si *f* est une expression de la variable *x* contenant des termes avec « sign », la fonction remplace tous les « sign » par des fonctions par morceaux. En utilisant ensuite « grouper_fct », on aura une unique fonction par morceaux.
integral2(*f*, *x*)	Cette fonction vient généraliser une autre fonction (« integral_sign ») afin d'intégrer une expression *f* de la variable *x*, expression où un ou plusieurs facteurs « sign » apparaît. Il est bien connu que l'intégrateur de Nspire CAS n'intègre pas un produit du type $\text{sign}(a \cdot x + b) \cdot f(x)$: la règle requise est utilisée (et donnée!) dans *Derive*.

Principales fonctions du Kit_ETS_MB

Des fonctions pour Mat 145

Nom de la fonction	Description
newton(*f*, *x*, *a*, *n*)	Applique la méthode de Newton *n* fois en partant du point *a* pour une équation de la forme $f(x) = 0$.
fixed_point(*f*, *x*, *a*, *n*)	Applique la méthode du point fixe *n* fois en partant du point *a* pour une équation de la forme $x = f(x)$.
rightsum(*f*, *x*, *a*, *b*, *n*), *leftsum*(*f*, *x*, *a*, *b*, *n*) et *midsum*(*f*, *x*, *a*, *b*, *n*)	Sommes de droite, gauche et du point milieu avec *n* subdivisions de l'intervalle [*a*, *b*] pour estimer $\int_a^b f(x)dx$.
trap(*f*, *x*, *a*, *b*, *n*) *simpson*(*f*, *x*, *a*, *b*, *n*)	Méthode des trapèzes pour le calcul de $\int_a^b f(x)dx$. Méthode de Simpson (*n* doit être pair).
ratio_test(*t*, *n*)	Se simplifie en la limite de la valeur absolue du rapport du $(n + 1)$ ième terme de la série sur le *n* ième terme, quand *n* tend vers l'infini. « *t* » est le terme général de la série. La série converge si le résultat est plus petit que 1.

Des fonctions pour Mat 165

Nom de la fonction	Description
ang_2_vect(**u**, **v**)	Donne l'angle (entre 0 et π) entre les 2 vecteurs de **u** et **v**, vecteurs de $\mathbb{R}^2$ ou $\mathbb{R}^3$.
atan2(*y*, *x*)	Donne l'angle θ $(-\pi < \theta \leq \pi)$ du vecteur [*x*, *y*]. Notez bien l'ordre d'entrée des variables.
ptcri(*f*, **v**)	Retourne la matrice des points critiques de l'expression *f* de 2 variables **v** = [*x*, *y*]. Efficace si *f* est polynomiale.
nature(*f*, **v**, **vo**)	Retourne le couple $\left[D(a,b),\ f''_{xx}(a,b)\right]$ où *D* est le hessien. Le point $\mathbf{v_0} = [a,b]$ est l'un des points critiques de *f*, de variables **v** = [*x*, *y*].
lagrange1(*f, g,* **v**, λ)	Se simplifie en la liste utilisée pour appliquer la méthode de Lagrange à une contrainte. Ici *f* est le champ scalaire, *g* la contrainte valant 0, **v** le vecteur des variables (2 ou 3) et λ le multiplicateur.
lagrange2(*f,* **g**, **v**, λ, μ)	Se simplifie en la liste utilisée pour appliquer la méthode de Lagrange à 2 contraintes, avec 3 variables. Ici *f* est le champ scalaire, **g** le vecteur des 2 contraintes valant 0, **v** le vecteur des variables et λ et μ sont les multiplicateurs de Lagrange.
grad(*f*, **v**)	Se simplifie en le gradient du champ scalaire *f*, **v** étant le vecteur des variables.
jacobian(**F**, **v**)	Se simplifie en la matrice jacobienne du champ vectoriel **F**, **v** étant le vecteur des variables.
div(**F**, **v**)	Se simplifie en la divergence du champ vectoriel **F**, **v** étant le vecteur des variables.
laplacian(*f*, **v**)	Se simplifie en le laplacien du champ scalaire *f*, **v** étant le vecteur des variables
curl(**F**, **v**)	Se simplifie en le rotationnel du champ vectoriel à 3 composantes **F**, **v** étant le vecteur des variables.
potential(**F**, **v**)	Se simplifie en le potentiel du champ vectoriel **F, v** étant le vecteur des variables. *Toujours vérifier la réponse en calculant le gradient du résultat!*
int_curv(**F**, **v**, **r**, *t*, *a*, *b*)	Calcule l'intégrale curviligne du champ vectoriel **F** dont les variables sont dans le vecteur **v**, le long d'une courbe paramétrée par **r**(*t*), pour $a < t < b$.
newtons(**u,** **v**, **vo**, *n*)	En approximant, la méthode de Newton à plusieurs variables est appliquée au système d'équations **u** = **0**. **v** est le vecteur des variables, $\mathbf{v_0}$ le vecteur de départ et *n* le nombre d'itérations.
limvect(*f*, **v**, **vo**)	La limite (composante par composante) du champ scalaire *f* lorsque le vecteur **v** tend vers le vecteur $\mathbf{v_0}$.

Des fonctions pour Mat 265

Nom de la fonction	Description
euler_ode(*f*, *x*, *y*, *xo*, *yo*, *h*, *n*) (Voir Note 1)	Applique la méthode d'Euler pour l'É.D. $\frac{dy}{dx} = f(x, y)$, $y(x_0) = y_0$, avec un pas de *h*, en *n* étapes. La matrice est présentée sur 2 colonnes.
picard(*f*, *p*, *x*, *y*, *xo*, *yo*)	Étant donné l'É.D. $\frac{dy}{dx} = f(x, y)$, $y(x_0) = y_0$, la méthode itérative de Picard est appliquée, partant avec *p* (*p* peut dépendre de *x* mais habituellement on prend *yo*). « *s* » est la variable d'intégration choisie.
lin_frac_ode(*r*, *a*, *b*, *d*, *p*, *q*, *k*, *x*, *y*)	Trouve une solution générale de l'équation différentielle du premier ordre $y' = r \equiv \frac{a\,x + b\,y + d}{p\,x + q\,y + k}$. Cette É.D. n'est pas résolue par la commande « deSolve » et n'est pas « homogène ».
eu_ca_ode(*a*, *b*, *x*, *y*, *r*)	Trouve une solution générale de l'équation différentielle de Euler-Cauchy du second ordre $x^2 y'' + a\,x\,y' + b\,y = r(x)$. Le « deSolve » en est souvent incapable.
solpart(*y1*, *y2*, *r*, *x*) (Voir Note 2)	Trouve une solution particulière à $y''(x) + p(x)\,y'(x) + q(x)\,y(x) = r(x)$ en appliquant la méthode de variation des paramètres. La fonction *r* peut même être par morceaux. En premier, on trouve la solution complémentaire $y_c = c_1 y_1 + c_2 y_2$ en résolvant $y''(x) + p(x)\,y'(x) + q(x)\,y(x) = 0.$
desolve2_gen(*p*, *q*, *r*, *x*) (Voir Note 2)	Trouve la solution générale à $y''(x) + p(x)\,y'(x) + q(x)\,y(x) = r(x)$ en termes de constantes ***c1*** et ***c2*** par la méthode de variation des paramètres. La fonction *r* peut même être par morceaux.
wronskian(**y**, *x*)	Si **y** = $[y_1, y_2, \ldots, y_n]$ est un vecteur de *n* solutions d'une équation différentielle linéaire d'ordre *n* où *x* est la variable indépendante, cela donne le wronskien et permet de vérifier l'indépendance linéaire des solutions. Utile puisque le « deSolve » est limité à l'ordre 2.
step(*t*) (aussi dénotée *uu*(*t*) dans le kit)	Fonction échelon-unité de Heaviside, valant 1 si $t > 0$ et valant 0 si $t < 0$.
chi(*a*, *t*, *b*)	Fonction indicatrice de l'intervalle [*a*, *b*], valant 1 si *t* est entre *a* et *b* et 0 ailleurs.

convolap(*x*, *h*) (Voir Note 3)	Se simplifie en la convolution, au sens de Laplace, des signaux *x*(*t*) et *h*(*t*).
ressort(*m*, *b*, *k*, *f*, *yo*, *vo*) (Voir Note 3)	Résout l'É.D. de masse-ressort, avec position (x_0) et vitesse (v_0) initiales données : $m\,y''(t)+b\,y'(t)+k\,y(t)=f(t).$ Donc, donne la position en fonction du temps.
circuit(*R*, *L*, *C*, *E*, *vo*, *io*) *cir_rc*(*R*, *C*, *E*, *vo*) *cir_rl*(*R*, *L*, *E*, *io*) (Voir Note 3)	Trouve le voltage aux bornes du condensateur $v_C(t)$ dans un circuit *RLC*, source *E*(*t*), voltage (v_0) et courant (i_0) initiaux donnés. Donc résout l'É.D. $L\cdot C\cdot v_C'' + R\cdot C\cdot v_C' + v_C = E(t).$ Trouve le voltage aux bornes du condensateur $v_C(t)$ dans un circuit *RC*, source *E*(*t*) avec voltage initial v_0. Donc résout $R\cdot C\cdot v_C' + v_C = E(t).$ Trouve le courant *i*(*t*) dans un circuit *RL*, source *E*(*t*) avec courant initial i_0. Donc résout l'É.D. $L\cdot\frac{di}{dt}+R\cdot i = E(t).$
taylor_ode1(*r, x, y, xo, yo, n*)	Se simplifie en le polynôme de Taylor d'ordre *n* pour l'équation différentielle $\frac{dy}{dx} = r(x,y),\ y(x_0) = y_0.$
taylor_ode2(*r, x, y, v, xo, yo, vo, n*)	Se simplifie en le polynôme de Taylor d'ordre *n* pour l'équation différentielle $\frac{d^2y}{dx^2} = r(x,y,y'),\ y(x_0) = y_0,\ y'(x_0) = v_0.$ Ici, *v* joue le rôle de *y*'.
rk23_ode(**r**, **v**, **vo**, *h*, *n*, *tol*) (Voir Note 1)	Tout comme euler_ode, rk23_ode s'approxime en une matrice à 2 colonnes, où **r** = [*f*(*x*, *y*)], **v** = [*x*, *y*], $\mathbf{v_o} = [x_0, y_0]$ et *tol* est l'erreur de tolérance.
fourier(*f, t, t1, t2, n*) (voir Note 4)	Se simplifie en la somme partielle de Fourier d'ordre *n* de l'expression périodique *f*. La période de *f* est *t2* − *t1* et sa variable est *t*.

Note 1 : remarquez que les fonctions internes de Nspire CAS « euler » et « rk23 » sont même programmées pour les *systèmes* d'É.D.

Note 2 : cette fonction *donnait* exactement ce que donnait le « deSolve » de la V200 (dans sa partie pour la solution particulière). Avec Nspire CAS, la réponse du « deSolve » est souvent plus compacte.

L'originalité des fonctions « solpart » et « desolve2_gen » réside en le fait qu'elles acceptent des membres de droite ($r(x)$) continus par morceaux, ce que le « deSolve » de la TI ne peut faire.

Note 3 : ces fonctions appellent les fonctions « laplace » et « ilaplace » de la librairie « ETS_specfunc.tns ». Puisque la variable indépendante dans le domaine du temps de cette dernière librairie est nécessairement « t », il était inutile de l'inclure dans les arguments des fonctions comme « convolap », « ressort » « circuit », « cir_rc » et « cir_rl ».

Note 4 : plutôt que d'utiliser l'intégrateur de Nspire CAS pour le calcul des coefficients de Fourier, notre fonction « fourier » utilise la fonction « integral_mcx_d » de la librairie « Kit_ETS_FH » qui intègre symboliquement les produits de fonctions par morceaux lorsque multipliées par d'autres expressions.

Des fonctions pour Mat 472 (voir aussi Mat 165)

Nom de la fonction	Description
eigen(**A**) *eigen2*(**A**)	Donne, en mode exact lorsque possible, les valeurs propres de la matrice carrée **A**, y compris les valeurs propres répétées. Si **A** est une matrice symbolique, on utilisera la fonction *eigen2*.
*poly_int(***A**, x)	Se simplifie en le polynôme d'interpolation de Lagrange. Ici **A** est la matrice à 2 colonnes des n points par lesquels passe le polynôme $p(x)$.
reflect_x *reflect_y* *reflect_yx* *reflect_y_x* *reflect_o*	Matrice de réflexion par rapport à l'axe des x. Matrice de réflexion par rapport à l'axe des y. Matrice de réflexion par rapport à la droite $y = x$. Matrice de réflexion par rapport à la droite $y = -x$. Matrice de réflexion par rapport à l'origine.
cont_hor(k) *cont_ver*(k)	Matrice de dilatation horizontale d'un facteur k. Matrice de dilatation verticale d'un facteur k.
shear_hor(k) *shear_ver*(k)	Matrice de cisaillement horizontal de facteur k. Matrice de cisaillement vertical de facteur k.
parti2D(**A**)	Si **A** est une matrice de format 2 × 2, la fonction retourne alors une matrice de format 3 × 3 qui la représente en coordonnées homogènes par une matrice partitionnée de la forme $\begin{bmatrix} \mathbf{A} & 0 \\ 0 & 1 \end{bmatrix}$. En pratique, la matrice **A** est l'une des nombreuses transformations linéaires géométriques de $\mathbb{R}^2$ (réflexion, dilatation, cisaillement).
*trans*2*D*(h, k) *trans3D*(h, k, l)	Se simplifie en une matrice 3 × 3 qui permet d'effectuer une translation par un vecteur [h, k]. Se simplifie en une matrice 4 × 4 qui permet d'effectuer une translation par un vecteur [h, k, l].

rotate_x(θ) *rotate_y*(θ) *rotate_z*(θ)	Se simplifie en une matrice **A** telle que **A.v** effectue une rotation du vecteur 3D **v** d'un angle de θ radians autour de l'axe des *x* (resp. *y*, resp. *z*), dans le sens anti-horaire lorsqu'on regarde depuis la partie positive de l'axe des *x* (resp. *y*, resp. *z*) vers l'origine.
parti3D(**A**)	Si **A** est une matrice de format 3 × 3, la fonction retourne alors une matrice de format 4 × 4 qui la représente en coordonnées homogènes par une matrice partitionnée de la forme $\begin{bmatrix} \mathbf{A} & 0 \\ 0 & 1 \end{bmatrix}$. En pratique, la matrice **A** est l'une des matrices de rotation précédentes ou de translation 3D.

Des fonctions plus avancées

Nom de la fonction	Description
expmat(**A**) (Voir Note 5)	Donne, en mode exact lorsque possible, $e^{\mathbf{A}t}$ où **A** est une matrice carrée constante.
iterate(*u*, *x*, *xo*, *n*) (Voir Note 6)	En simplifiant (ou approximant dans plusieurs cas), la formule $x \leftarrow u(x)$ est répétée *n* fois, en partant de *x*.
res_pole(*f, z, zo ,m*)	**Si** z_0 est un pôle d'ordre *m* de l'expression *f* d'une variable complexe *z*, alors cette fonction donne le résidu en ce pôle.
taylor_solve(*u, x, y, xo, yo, n*) (Voir Note 7)	Étant donnée la courbe implicite 2D définie par l'équation *u*(*x*, *y*) = 0 et passant par le point (x_0, y_0), cela se simplifie en le polynôme de Taylor d'ordre *n* de la solution.
taylor_inv(*u, x, y, xo, n*) (Voir Note 8)	Étant donné l'équation *y* = *u*(*x*), cela se simplifie en le polynôme de Taylor d'ordre *n* de la fonction réciproque (« inverse »), autour du point $y_0 = u(x_0)$. Utile lorsqu'on ne peut résoudre l'équation *y* = *u*(*x*) pour *x*.
syst_ed(**A**, **g**, *to*, **yo**)	Résout le système d'équations différentielles linéaires $\frac{d\mathbf{y}}{dt} = \mathbf{A}\mathbf{y} + \mathbf{g}(t), \mathbf{y}(t_0) = \mathbf{y}_0$, **A** étant une matrice carrée constante. **Utilisez** « *t* » comme variable indépendante.
convol_gen(*x*, *h*, *t*) (Voir Note 9)	Se simplifie en la convolution des signaux continus *x*(*t*) et *h*(*t*). Il s'agit de la convolution au sens général telle que rencontrée dans les cours de traitement de signaux. .

conv_abs_to_p(*f*, *x*) (Voir Note 10)	Convertit une somme de valeurs absolues (ou une seule) de l'expression *f* de variable *x* en une fonction par morceaux.

Note 5 : la fonction utilise le « package » Laplace puisqu'il est bien connu que $e^{\mathbf{A}t} = \pounds^{-1}\left[\left(s\mathbf{I}-\mathbf{A}\right)^{-1}\right]$, où « £ » désigne la transformée de Laplace et **I** la matrice identité.

Note 6 : notez que *u*, *x* et x_0 peuvent aussi être des vecteurs. Des fonctions comme « newton », « newtons » et « fixed_point » utilisent « iterate ». Des applications intéressantes sont données dans le document d'exemples.

Note 7 : cette fonction peut s'avérer utile en attendant le « plotteur implicite 2D » …

Note 8 : cette fonction peut s'avérer utile lorsqu'on ne peut résoudre l'équation $y = u(x)$ pour x. Remarquez toutefois que Nspire CAS (depuis OS 3.2) trace des courbes définies par $x = g(y)$. Cela permet donc de visualiser la réciproque.

Note 9 : Il s'agit de la convolution au sens général telle que rencontrée dans les cours de traitement de signaux : $x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)\, h(t-\tau)\, d\tau.$

Cette fonction fait appel à de nombreuses fonctions programmées par Frédéric Henri. Elle est passablement performante lorsque chacun des signaux est à support compact. Particulièrement utile si les signaux sont définis par morceaux avec le modèle de fonctions par morceaux de Nspire CAS. Des fonctions programmées par l'auteur de ces lignes préparent alors l'intégrande $x(\tau)\, h(t-\tau)$ qui, rendu sous la forme de combinaisons linéaires de fonctions indicatrices, devient intégrable via la fonction « integral2 » de Fred. Ne reste plus qu'à évaluer la primitive entre $-\infty$ et ∞.

Note 10 : lorsqu'on veut remplacer par le modèle (« template ») de fonction par morceaux une réponse contenant *uniquement* des valeurs absolues, on peut utiliser la fonction « conv_abs_to_p » afin de revenir à une fonction par morceaux.