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 6 mai 2020
(corrections mineures le 13.07.2021)

Les fonctions qui suivent ont été définies dans le classeur Nspire CX CAS portant le nom de « Kit_ETS_MB » et peuvent être utiles dans plusieurs des cours de mathématiques et génie à l'ÉTS. Elles ont l'avantage d'être assez générales et ont été principalement développées en mémoire du logiciel *Derive*, ce qui explique plusieurs noms en anglais. Qui plus est, plusieurs de ces fonctions ont été extraites de librairies de *Derive*, pratiquement intégralement ou adaptées, tenant compte de la syntaxe de Nspire. Ce classeur, de même que les deux autres dont il est question plus loin, se trouvent sur ma page principale et doivent être enregistrés dans MyLib (ce sont des « librairies de fonctions »). On rafraîchit ensuite les bibliothèques afin de pouvoir l'utiliser dans un autre classeur. Pour des informations, consultez ceci. Dans certains cas, des fonctions de la librairie Kit_ETS_MB utilisent certaines fonctions provenant des librairies ETS_specfunc et Kit_ETS_FH. Consultez le lien suivant où Chantal Trottier décrit le fichier ETS-specfunc.tns. Cela concerne les transformées de Laplace pour le calcul des transformées directe et inverse. Le collègue Frédérick Henri est l'auteur de la plupart des fonctions qu'on retrouve dans le fichier Kit_ETS_FH (voir à la fin de ce document pour des détails).

Plusieurs exemples commentés, relatifs à un cours spécifique, sont inclus dans le PDF *Exemples d'utilisation*. La version du fichier Kit_EST_MB, de même que celle du présent document et du document « Exemple d'utilisation » fait toujours référence à la date qui y apparaît. Les versions précédentes furent en avril aux années 2018, 2016, 2015, 2014 et 2013 et en décembre 2011. Cette version du 6 mai 2020 inclut désormais la syntaxe à même le fichier Nspire. En effet, nous avons défini toutes les fonctions « publiques » en passant par l'éditeur de programmes, de sorte qu'un commentaire inséré a permis d'y voir la syntaxe dans le catalogue. Il n'en reste pas moins que cela ne remplace pas les tableaux qui suivent puisque il n'y aura jamais suffisamment d'espace dans le catalogue pour *expliquer* les paramètres d'une fonction et ce qu'elle retourne. De plus, pour un habitué du calculs symbolique, il est facile de comprendre que « *rightsum*(*f*, *x*, *a*, *b*, *n*) signifie qu'il s'agit d'une somme de Riemann faite à partir des extrémités droites des rectangles de subdivision; et que cette fonction requiert une expression *f* en la variable *x*, les bornes d'intégration étant respectivement *a* et *b* et que l'intervalle d'intégration [*a*, *b*] est subdivisé en *n* parties de longueur égale. Et la lecture dans le catalogue de *comp_co_rev*(**fi**, **po**, **ve**, *θ*, *i*) ne nous indique pas grand-chose ─ d'autant plus qu'il faudrait, dans le catalogue, dire que les arguments ne sont pas tous de même nature. Décrire cette fonction, comme c'est fait au début du tableau de MAT165 (voir plus loin), nous apparaît important. Il reste aussi important donner, dans le fichier d'exemples, un exemple de son utilisation commenté et où l'on insiste pour dire que, très souvent, on peut se passer de cette fonction! Voilà pourquoi nous croyons que les deux fichiers PDF restent pertinents.

Dans les tableaux qui suivent, tous les vecteurs et toutes les matrices sont dénotés par des symboles **gras**. Pour distinguer nos fonctions de celles déjà implémentées dans Nspire, nous avons utilisé des italiques pour les noms de nos fonctions ─ d'autant plus que Nspire fera de même dans une page de calcul. Notez aussi que dans certains cas il est plus rapide et plus efficace d'approximer plutôt que de simplifier certaines fonctions. Notez finalement que plusieurs de ces fonctions permettent une vérification rapide d'une réponse qui aura été trouvée en faisant les calculs au long. Comme quoi, l'un n'empêche pas l'autre.

Quelques fonctions pour MAT144

Nom de la fonction	Description
pente(**p1**, **p2**)	Donne la pente du segment de droite 2D passant par les points **p1** et **p2**.
droite_2D(**p1**, **p2**)	Retourne l'équation de la droite 2D passant par les 2 points **p1** et **p2**. Pour des raisons de commodité, les variables *x* et *y* sont celles utilisées pour la réponse.
info_parabole(*f*, *x*)	Si *f* est une expression du second degré en la variable *x*, la fonction retourne les informations suivantes : l'ouverture (vers le haut ou le bas), le delta et la nature des racines.

Nous sommes conscients que dans un cours de « révision », on a d'autres chats à fouetter et de commencer à inonder les étudiants de nouvelles fonctions programmées serait non seulement une perte de temps mais en enlèverait pour s'approprier l'utilisation de la calculatrice. Par contre, les étudiants ont déjà intérêt à voir comment on peut définir certaines petites fonctions simples en utilisant un système symbolique : ainsi, notre fonction « *pente* » ci-haut nous permet déjà de montrer aux étudiants comment extraire une entrée d'une matrice (« tableau ») et d'utiliser deux plutôt que quatre arguments.

Des fonctions pour MAT145

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 $x = f(x)$. Si plusieurs variables, notation vectorielle à utiliser.
imp_tangent(*u*, *x*, *y*, *xo*, *yo*)	Se simplifie en une expression linéaire en la variable *x*. Le résultat est la coordonnée *y* de la droite tangente au point (*xo*, *yo*) à la courbe définie implicitement par l'équation *u*(*x*, *y*).
rightsum(*f*, *x*, *a*, *b*, *n*), *leftsum*(*f*, *x*, *a*, *b*, *n*) et *midsum*(*f*, *x*, *a*, *b*, *n*) *trap*(*f*, *x*, *a*, *b*, *n*)	Sommes de droite, de gauche, du point milieu et du trapèze avec *n* subdivisions de l'intervalle [*a*, *b*] pour estimer $\int_a^b f(x)dx$
simpson(*f*, *x*, *a*, *b*, *n*)	Méthode de Simpson (*n* doit être pair).
comp_int_num(*f*, *x*, *a*, *b*, *n*)	Donne le tableau complet des valeurs des sommes de droite, de gauche, du point milieu, du trapèze et de Simpson.
ratio_test(*t*, *n*)	Applique le test du rapport à une série dont le terme général est *t* et la variable *n*. La série converge si le résultat est plus petit que 1.
root_test(*t*, *n*)	Applique le test de la racine à une série dont le terme général est *t* et la variable *n*. La série converge si le résultat est plus petit que 1.

Tout comme MAT144, il est assez rare que nous fassions usage en classe des fonctions du tableau précédent. En effet, appliquer la méthode de Newton pour résoudre l'équation $f(x) = 0$ avec la calculatrice est très facile puisqu'on peut rappeler la dernière réponse : il suffit alors de définir la fonction $x - \frac{f(x)}{f'(x)}$. Par contre, son « output » en tableau est très joli. Une fonction comme « *imp_tangent* » a été programmée puisque la différentiation implicite constitue toujours un problème pour les étudiants et la fonction interne « tangentLine » ne s'applique qu'à une expression d'une seule variable. Les différentes sommes de Riemann constituent une supplément à ce qui est décrit dans les notes de cours de MAT145.

Deux petites fonctions pour MAT210

Nom de la fonction	Description
lin1_difference(*p*, *q*, *x*, *xo*, *yo*)	Se simplifie en une solution spécifique d'une équation aux différences de la forme $y(x + 1) = p(x)\, y(x) + q(x)$ étant donnée la condition initiale $y(xo) = yo$. Cela s'appliquera donc pour résoudre une équation aux différences, linéaire d'ordre 1.
lin2_ccf(*p*, *q*, *r*, *x*, *c*1, *c*2)	Se simplifie en une solution générale de l'équation aux différences linéaire du second ordre à coefficients *p* et *q* constants de la forme $y(x + 2) + p\, y(x + 1) + q\, y(x) = r(x)$ en terme des constantes symboliques *c*1 et *c*2. Si jamais des conditions initiales étaient données, on fera résoudre (« complex solve ») pour *c*1 et *c*2 après substitution.

L'auteur de ces lignes n'a jamais enseigné le cours de MAT210 et ne compte pas le faire dans un avenir rapproché. Mais l'éditeur de suites de Nspire permet de définir des relations de récurrence et les deux fonctions ci-haut pourraient, dans certains cas, donner la solution sous forme close.

Des fonctions pour MAT165

Nom de la fonction	Description
com_co_rev(**fi**, **po**, **ve**, θ, *i*) (utile pour tracer/animer des solides de révolution/permet un retour à MAT145)	Soit **fi** une matrice à 3 rangées dont chaque colonne contient les expressions paramétriques d'une courbe 3D. On veut faire tourner, d'un angle θ, par rapport à une droite passant par le point **po** et parallèle au vecteur **ve**, la figure décrite par **fi**. La fonction se simplifie en une liste de la *i* ième composante de la courbe de révolution. La variable *i* doit être comprise entre 1 et le nombre de colonne de **fi**.
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*.
droite_tan_3D(**r**, *t*, *to*) (aussi en 2D)	Se simplifie en le vecteur paramétrique des composantes de la droite tangente à la courbe **r**(*t*) au point où *t* = *to*.
*long_arc(***r**, *t*, *a*, *b*)	Se simplifie en la longueur de l'arc de la courbe paramétrée **r**(*t*), $a \leq t \leq b$.
droite_3D(**p1**, **p2**, *t*) (aussi en 2D)	Se simplifie en le vecteur des expressions paramétriques de la droite passant par les points **p1** et **p2**, *t* étant le paramètre.
*plan(***p1**, **p2**, **p3**)	Se simplifie en l'équation du plan passant par les 3 points **p1**, **p2** et **p3**. Si les points sont colinéaires, un message nous en avertit. Pour des raisons de commodité, les variables *x*, *y* et *z* sont utilisées pour la réponse.
plan_tan(*u*, **v**, **vo**) *plan_tan_para*(**r**, **p**, **po**, **v**)	Se simplifie en l'équation du plan tangent à la surface définie par *u* = 0. Les variables sont contenues dans le vecteur **v** et le point de tangence est **vo**. Se simplifie en le plan tangent à la surface paramétrique définie par **r**, **p** est le vecteur (à 2 composantes) des paramètres et **po** le vecteur (à 2 composantes) nous donnant le point de tangence qui est **r**(**po**). La réponse est donnée en fonction des variables contenues dans le vecteur **v**.
droite_nor_3D(*u*, **v**, **vo**, *t*)	Se simplifie en le vecteur paramétrique (linéaire en *t*) des composantes de la droite normale à la surface *u* = 0 au point **vo**. Les variables de *u* sont contenues dans le vecteur **v**.
nature(*f*, **v**, **vo**)	Retourne l'information concernant 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. *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.
grad(*f*, **v**) *grad_p*(*f*, **v**, **vo**) *laplacian*(*f*, **v**)	Se simplifie en le gradient du champ scalaire *f*, **v** étant le vecteur des variables. Se simplifie en le gradient du champ scalaire *f* de variable **v** au point **vo**. Se simplifie en le laplacien du champ scalaire *f*, **v** étant le vecteur des variables.

jacobian(**F**, **v**) *div*(**F**, **v**) *curl*(**F**, **v**)	Se simplifie en la matrice jacobienne de **F**. Se simplifie en la divergence de **F**. Se simplifie en le rotationnel de **F**. **F** est un champ vectoriel dont le vecteur des variables est **v**. Pour la fonction *curl*, la dimension doit être 3.
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*) *int_curv_arc*(*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 $a < t < b$. Calcule l'intégrale curviligne du champ scalaire f relativement à la longueur d'arc. Les variables de f sont dans le vecteur **v**, l'arc est représenté par la courbe paramétrée **r**(*t*), $a < t < b$.

Avec le tableau précédent, on peut dire qu'on commence vraiment être conscient que chaque étudiant de l'École a dû acheter une machine symbolique. Beaucoup de calculs sont longs, répétitifs et, s'ils ne sont pas automatisés, on accorde moins de temps à la compréhension des concepts. Le cours de MAT165 avec son contenu chargé profite beaucoup de l'utilisation de plusieurs des fonctions décrites ci-haut : que ce soit pour *vérifier* sa réponse à l'équation du plan qu'on devait trouver, pour avoir du temps pour *tracer le graphique* de la surface dont on vient de localiser un maximum local, que ce soit pour comprendre comment une fonction *potential* peut être programmée … en utilisant la preuve d'un théorème fait en classe ! Les résumés sur la première partie du cours et celui sur la seconde partie du cours de MAT165 contiennent plusieurs exemples où des fonctions du tableau précédent sont utilisées.

Des fonctions pour MAT265

Nom de la fonction	Description
euler_ode(*f*, *x*, *y*, *xo*, *yo*, *h*, *n*)	Applique la méthode d'Euler pour $\frac{dy}{dx} = f(x, y)$ avec la c.i. $y(xo) = yo$ et un pas de *h*, en *n* étapes. La matrice est présentée sur 2 colonnes.
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*], **vo** = [**xo**, **yo**] et *tol* est l'erreur de tolérance.
picard(*f*, *p*, *x*, *y*, *xo*, *yo*)	Étant donné l'É.D. $\frac{dy}{dx} = f(x, y)$, $y(xo) = yo$, 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 si l'on veut continuer en appelant la dernière réponse.
picard_its(*f*, *p*, *x*, *y*, *xo*, *yo*, *n*)	Donnera automatiquement la *n*-ième itération de Picard.
lin_frac_ode(*r*, *x*, *y*)	Solution générale de É.D. $y' = r \equiv \frac{a\,x + b\,y + d}{p\,x + q\,y + k}$. 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'É.D. de Euler-Cauchy $x^2 y'' + a\,x\,y' + b\,y = r(x)$.
solpart(*y1*, *y2*, *r*, *x*) (Voir Note 2)	Trouve une solution particulière par la méthode de variation des paramètres à $y''(x) + p(x)\,y'(x) + q(x)\,y(x) = r(x)$. La fonction r peut même être par morceaux. y_1 et y_2 sont les solutions lin. ind. de l'équation complémentaire associée.

desolve2_gen(*p*, *q*, *r*, *x*) (Voir Note 2)	Solution générale de $y''(x) + p(x)\,y'(x) + q(x)\,y(x) = r(x)$ 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*)	Wronskien de *n* solutions d'une équation différentielle linéaire d'ordre *n* de variable *x*, les solutions étant $\mathbf{y} = [y_1, y_2, \ldots, y_n]$.
step(*t*) (aussi dénotée *uu*(*t*) dans le kit)	Fonction échelon-unité de Heaviside.
chi(*a*, *t*, *b*)	Fonction indicatrice de l'intervalle [*a*, *b*].
convolap(*x*, *h*) (Voir Note 3)	Se simplifie en la convolution, au sens de Laplace, des signaux *x*(*t*) et *h*(*t*).
lap_per(*f*, *P*) (Voir Note 4)	Se simplifie en la transformée de Laplace d'une expression *périodique f* de période *P* (**Utilisez** « *t* » comme variable).
ressort(*m*, *b*, *k*, *f*, *yo*, *vo*) (Voir Note 3)	Résout, par utilisation de la transformée de Laplace, l'É.D. $m\,y''(t) + b\,y'(t) + k\,y(t) = f(t)$ qui modélise un problème de masse-ressort. *y*(*t*) est la position et *f*(*t*) la force extérieure, *yo* et *vo* sont respectivement la position et la vitesse initiales.
circuit_rlc(*R*, *L*, *C*, *E*, *vo*, *io*) *cir_rc*(*R*, *C*, *E*, *vo*) *cir_rl*(*R*, *L*, *E*, *io*) (Voir Note 3)	Se simplifie en la tension aux bornes du condensateur $v_C(t)$ dans un circuit *RLC* . La source est *E*(*t*), La tension initiale est *vo* et le courant initial est *io*. Donc cette fonction résout, par la méthode de Laplace, l'É.D. $L\,C\,v_C'' + R\,C\,v_C' + v_C = E(t)$. Donne $v_C(t)$ dans un circuit *RC*, source *E*(*t*) avec tension initiale *vo*. Donc résout l'É.D. $R\,C\,v_C' + v_C = E(t)$. Donne le courant *i*(*t*) dans un circuit *RL*, source *E*(*t*), courant initial *io*. Donc résout l'É.D. $L\frac{di}{dt} + R\,i = E(t)$.
u_to_piece(*f*, *x*)	Transforme l'expression *f* de variable *x* en fonction par morceaux après avoir posé $u(t) := \frac{1 + \mathrm{sign}(t)}{2}$.
taylor_ode1(*r*, *x*, *y*, *xo*, *yo*, *n*)	Polynôme de Taylor d'ordre *n* autour de *xo* pour l'É.D. $\frac{dy}{dx} = r(x, y)$ avec la c.i. $y(xo) = yo$.
taylor_ode2(*r*, *x*, *y*, *v*, *xo*, *yo*, *vo*, *n*)	Polynôme de Taylor d'ordre *n* autour de *xo* pour l'É.D. $\frac{d^2y}{dx^2} = r\left(x, y, y'\right)$ où *v* remplace y'. Les c.i. sont $y(xo) = yo$ et $y'(xo) = vo$.
fourier(*f*, *t*, *t*1, *t*2, *n*) (voir Note 4)	Soit *f* une expression de la variable *t* initialement définie sur l'intervalle entre *t*1 et *t*2. La fonction se simplifie en la somme partielle de Fourier d'ordre *n* de *f*.

S'il y a un cours qui a profité en premier de la présence d'un calculateur symbolique, c'est bien celui d'équations différentielles ! Nos fonctions de MAT265 puisent fréquemment dans les librairies ets_specfunc et kit_ets_fh et certaines sont le fruit de belles collaborations avec des collègue s du SEG. Certaines fonctions ont automatisé des solutions à des problèmes d'application (masse-ressort, circuits électriques) en utilisant une syntaxe beaucoup plus simple et rapide que celle de la commande « *solved* » de la librairie ets_specfunc. D'autres, comme les fonctions « *fourier* » , « *taylor_ode1* » et « *taylor_ode2* » s'avèrent d'excellents moyens de vérification de réponses de façon super rapide. Le fichier PDF suivant réfère à MAT265 avec plein d'exemples.

Des fonctions pour MAT472 (voir aussi MAT165)

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*.
adj(**A**)	Matrice adjointe de la matrice carrée **A**.
replace_r(**A**,*i*,**r**)	Si **A** est une matrice, remplace la ligne *i* par le vecteur **r**
replace_c(**A**,*i*,**r**)	Si **A** est une matrice, remplace la colonne *i* par le vecteur **r**. On entre encore **r** comme vecteur ligne.
poly_int(**A**, *x*)	Se simplifie en le polynôme d'interpolation de Lagrange de variable *x*. La matrice **A** est la matrice à 2 colonnes des *n* points par lesquels passe le polynôme cherché.
reflect_x *reflect_y* *reflect_yx* *reflect_y_x* *reflect_o* *sym_orth2D*(**v**)	Matrice 2×2 de réflexion par rapport à l'axe des *x*. Matrice 2×2 de réflexion par rapport à l'axe des *y*. Matrice 2×2 de réflexion par rapport à la droite $y = x$. Matrice 2×2 de réflexion par rapport à la droite $y = -x$. Matrice 2×2 de réflexion par rapport à l'origine. Matrice 2×2 de symétrie orthogonale par rapport à une droite 2D passant par l'origine et normale au vecteur **v**. Généralise les précédentes (les quatre premières)
proj_orth2D(**v**) *project_x* *project_y*	Matrice 2×2 de projection orthogonale sur le vecteur **v**. Permet donc de projeter orthogonalement sur la droite d'équation $a\,x + b\,y = 0$ en prenant $\mathbf{v} = [a, b]$. Cas particulier de la projection sur l'axe des *x*. Cas particulier de la projection sur l'axe des *y*.
cont_hor(*k*) *cont_ver*(*k*)	Matrice 2×2 de dilatation/contraction horizontale de facteur *k*. Matrice 2×2 de dilatation/contraction verticale de facteur *k*.
shear_hor(*k*) *shear_ver*(*k*)	Matrice 2×2 de cisaillement horizontal de facteur *k*. Matrice 2×2 de cisaillement vertical de facteur *k*.
parti2D(**A**) *parti3D*(**A**)	Si **A** est une matrice de format 2×2 (resp. 3×3), la fonction retourne une matrice de format 3×3 (resp. 4×4) qui la représente en coordonnées homogènes par une matrice partitionnée de la forme $\begin{bmatrix} \mathbf{A} & 0 \\ 0 & 1 \end{bmatrix}$.
trans2D(*h*, *k*) *trans3D*(*h*, *k*, *l*)	Se simplifie en une matrice 3×3 (resp. 4×4) qui permet d'effectuer une translation par un vecteur de dimension deux [*h*, *k*] (resp. un vecteur de dimension trois [*h*, *k*, *l*]).
proj_orth3D(**v**)	Matrice 3×3 de projection orthogonale sur le vecteur 3D **v**. Permet donc de projeter orthogonalement sur le plan d'équation $a\,x + b\,y + c\,z = 0$ en prenant $\mathbf{v} = [a, b, c]$.
sym_orth3D(**v**)	Matrice 3×3 de réflexion ou de symétrie orthogonale sur le vecteur 3D **v**. Permet donc de trouver le symétrique d'un vecteur **w** par rapport au plan dont **v** est la normale en effectuant le produit *sym_orth3D*(**v**)· **w**.
rotate_x(θ) *rotate_y*(θ) *rotate_z*(θ)	Se simplifie en une matrice 3×3 **A** telle que **A.v** effectue une rotation du vecteur **v** (de dimension 3) 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.
rotate3D(**v**, θ)	Matrice 3×3 de rotation d'angle θ autour de la droite orientée par le vecteur 3D **v**. En prenant pour **v** respectivement **i, j** et **k**, on obtient les cas particuliers décrits dans la ligne plus haut. Le vecteur **v** n'a pas à être unitaire.

Des manipulations de matrices de toutes sortes interviennent en MAT472. Les plus utilisées sont définies dans le tableau ci-haut et il devient particulièrement intéressant de visualiser le tout de deux façons : en insérant des curseurs pour voir « bouger » les objets et parallèlement en utilisant le logiciel de géométrie intégré de Nspire. Un résumé montre des exemples concernant la première partie du cours et un second s'occupe de la deuxième partie du cours.

Autres fonctions diverses/fonctions pour MAT805, MEC532, ELE265/fonctions spéciales

La très grande majorité des fonctions du tableau qui suit sont définies essentiellement pour le cours de maîtrise MAT805 où nous avons constaté que Nspire peut faire un excellent travail … si on lui rajoute des fonctions spéciales !

Nom de la fonction	Description
lim_mat(**mat**, **v**, **vo**)	La limite (composante par composante) de **mat** lorsque le vecteur **v** tend vers le vecteur **vo**. « **mat** » peut être un champ scalaire, un vecteur, une matrice.
subst(**F**, **v**, **vo**) ou *subst*(*f*, *x*, *a*)	Si **F** est un champ vectoriel (ou même une matrice), de variables contenues dans le vecteur **v**, alors **v** est remplacé par le vecteur **vo**. Fonctionne aussi pour les scalaires.
newtons2(**u,** **v**, **vo**, *n*)	S'approxime en le tableau des itérations de la méthode de Newton à *deux* variables appliquée au système de deux équations **u** = **0**. **v** est le vecteur des variables, **vo** le vecteur de départ et *n* le nombre d'itérations.
fixed_points(**g**, **v**, **vo**, *n*)	S'approxime en le tableau des itérations de la méthode du point fixe à plusieurs variables appliquée au système des *n* équations **g** = **v**, partant de **vo**.
expmat(**A**) (Voir Note 5)	Donne, en mode exact lorsque possible, $e^{\mathbf{A}t}$ où **A** est une matrice carrée constante (et réelle).
iterates(*u*, *x*, *xo*, *n*) ou *iterates*(**u**, **x**, **xo**, *n*)	En simplifiant (ou approximant dans plusieurs cas), la formule $x \leftarrow u(x)$ est répétée *n* fois, en partant de *xo*. Le résultat est présenté par un vecteur pour une expression scalaire et par une matrice lorque **u** est un vecteur.
res_pole(*f, z, zo ,m*)	**Si** *zo* est un pôle d'ordre *m* de *f* de variable complexe *z*, alors cette fonction donne le résidu en ce pôle.
taylor_solve(*u, x, y, xo, yo, n*) (Voir Note 6)	Soit *u*(*x*, *y*) = 0 une courbe implicite 2D définie passant par le point (*xo*, *yo*). La fonction se simplifie en le polynôme de Taylor d'ordre *n* de l'équation.
taylor_inv(*u, x, y, xo, n*) (Voir Note 7)	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 *yo* = *u*(*xo*). Utile lorsqu'on ne peut résoudre l'équation *y* = *u*(*x*) pour *x*.
cexpand(*f*, *x*, *x_*)	Extension de la commande « expand » de Nspire utile pour faire un développement en fractions partielles dans le corps des complexes. Au besoin, approximer la réponse pour plus de commodités et/ou réappliquer la commande interne « expand » de Nspire. Rappel : dans Nspire, une variable « var » suivi du symbole « _ » est considérée complexe.
de_syst(**A**, **g**, *to*, **yo**)	Résout le système d'É.D. linéaires $\frac{d\mathbf{y}}{dt} = \mathbf{A}\mathbf{y} + \mathbf{g}(t)$ pour **A** une matrice carrée constante et la c.i. **y**(*to*) = **yo**. **Utilisez** « *t* » comme variable.

convol_gen(*x*, *h*, *t*) (Voir Note 8)	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 9)	Convertit une somme de valeurs absolues (ou une seule) de l'expression *f* de variable *x* en une fonction par morceaux.
compact_cubic(p, x)	Lorsque que la fonction « czeros » donne (en exact) des réponses inutilement lourdes pour les zéros d'un polynôme *p* de degré 3, alors cette fonction retourne des réponses compactes comme le bon vieux *Derive* pour les trois zéros!
cardano(*p*, *q*)	Retourne l'expression $u - p/u$ où $u = \left(q + \sqrt{q^2 + p^3}\right)^{1/3}$. Lorsque *p* est positif, c'est la seule solution réelle de l'équation $y^3 + 3p\,y - 2q = 0$.
gamma(*n*)	Valeur numérique exacte de la fonction Gamma pour un entier positif *n* ou un multiple impair de ½. Valeur approchée autrement.
beta(*x*, *y*)	Fonction Beta (x, $y > 0$).
kappa(*z*)	Se simplifie (s'approxime au besoin) en le « unwinding number » défini par le duo Jeffrey/Corless. Donc, si $z \in \mathbb{C}$, *kappa*$(z) = -n$ si $(2n-1)\pi < \text{Imag}(z) \leq (2n+1)\pi$.
bessel_J(*n*, *x*) (Voir note 10)	Fonction de Bessel de première espèce d'ordre *n* dénotée habituellement par $J_n(x)$. La fonction est définie pour *n* entier ainsi que pour $2n$ entier impair. Sinon, utilisez l'approximation par la série (voir fonction suivante)
bessel_J_series(*n*, *x*, *m*)	Se simplifie en $m + 1$ termes de la série de la fonction de Bessel $J_n(x)$ pour *n* quelconque.
bessel_Y_series(*n*, *x*, *m*)	Se simplifie en $m + 1$ termes de la série de la fonction de Bessel de seconde espèce $Y_n(x)$. Cette dernière n'est pas programmée dans la librairie. On aura une bonne approximation de $Y_0(x)$ en prenant $n = 0$ et *m* suffisamment grand.
erf(*x*) *si*(*x*) *ci*(*x*) *fresnel_sin*(*x*) *fresnel_cos*(*x*)	Fonctions d'erreur. Fonction sinus intégral. Fonction cosinus intégral. Fonction sinus intégral de Fresnel. Fonction cosinus intégral de Fresnel.
chebychev_T(*n*, *x*) (Voir note 11)	*n*-ième polynôme de Tcheybychev de première espèce dénoté habituellement par $T_n(x)$. Ces polynômes sont orthogonaux sur l'intervalle [−1, 1] en prenant comme fonction poids $1/\sqrt{1-x^2}$.
chebychev_U(*n*, *x*) (Voir note 11)	*n*-ième polynôme de Tcheybychev de seconde espèce dénoté habituellement par $U_n(x)$. Ces polynômes sont orthogonaux sur l'intervalle [−1, 1] en prenant comme fonction poids $1/\sqrt{1-x^2}$.
legendre_P(*n*, *x*) (Voir note 11)	*n*-ième polynôme de Legendre habituellement dénoté $P_n(x)$. Ils sont orthogonaux sur l'intervalle [−1, 1].
hermite_H(*n*, *x*) (Voir note 11)	*n*-ième polynôme d'Hermite habituellement dénoté $H_n(x)$. Ils sont orthogonaux sur l'intervalle $]-\infty, \infty[$ en prenant comme fonction de poids l'expression e^{-x^2}.

laguerre_L(*n*, *x*) (Voir note 11)	*n*-ième polynôme de Laguerre habituellement dénoté $L_n(x)$. Ils sont orthogonaux sur l'intervalle $]0, \infty[$ en prenant comme fonction de poids e^{-x}.
*ei_*(*x*, *m*)	S'approxime en *m* + 1 termes de la série de la fonction *exponentielle intégrale*. Cette fonction est définie, pour $x > 0$, par VPC $\int_{-x}^{\infty} \frac{e^{-t}}{t} dt$. (« VPC » : valeur principale de Cauchy).
*li_*(*x*, *m*)	S'approxime en *m* + 1 termes de la série de la fonction *logarithme intégral* qui est une fonction définie, pour $x \neq 1$, par VPC $\int_0^x \frac{1}{\ln(t)} dt$
elliptic_f(ϕ, *m*)	Intégrale elliptique de première espèce définie par $\int_0^{\phi} \frac{1}{\sqrt{1 - m\sin(t)^2}} dt$
elliptic_e(ϕ, *m*)	Intégrale elliptique de seconde espèce définie par $\int_0^{\phi} \sqrt{1 - m\sin(t)^2}\ dt$.
elliptic_pi(ϕ, *m*, *n*)	Intégrale elliptique de troisième espèce définie par $\int_0^{\phi} \frac{1}{\left(1 - n\sin(t)^2\right)\sqrt{1 - m\sin(t)^2}} dt$.
lambertw(*k*, *z*)	Donne la valeur de la fonction LambertW au point *z*, un nombre complexe. La branche est *k*, un entier. Avec les branches $k = 0$ et $k = -1$ qui sont les seules à donner des valeurs réelles, la fonction permet de se vérifier dans la résolution papier/crayon d'une équation ramenable à la forme $we^w = z$.

Informations sur les notes rencontrées dans les tableaux

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 *donne* un peu 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_rlc*, *cir_rc* et *cir_rl*.

Note 4 : plutôt que d'utiliser l'intégrateur de Nspire CAS pour le calcul des coefficients de Fourier, nos fonctions « lap_per » et « fourier » utilisent 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.

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

Note 6 : cette fonction peut s'avérer utile en attendant le « plotteur implicite 2D » …

Note 7 : cette fonction peut s'avérer utile lorsqu'on ne peut résoudre l'équation $y = u(x)$ pour x.

Note 8 : Il s'agit de la convolution telle que rencontrée dans les cours de traitement de signaux : $x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)\, h(t-\tau)\, d\tau$.

Note 9 : 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.

Note 10 : les fonctions de Bessel $J_n(x)$ et $Y_n(x)$ sont des solutions de l'équation différentielle de Bessel

$$x^2\, y'' + x\, y' + \left(x^2 - n^2\right) y = 0$$

et où le paramètre n est positif ou 0. Ce paramètre est souvent dénoté par la lettre grecque ν (« nu »).

Note 11 : on peut montrer que (en comprenant que la dérivée d'ordre 0 est la fonction elle-même)

$T_n(x) = \cos(n \arccos(x))$, $U_n(x) = \dfrac{\sin\left((n+1)\arccos(x)\right)}{\sqrt{1-x^2}}$, $P_n(x) = \dfrac{1}{2^n\, n!}\dfrac{d^n}{dx^n}\left(\left(x^2-1\right)^n\right)$ (formule de Rodrigues),

$H_n(x) = (-1)^n\, e^{x^2}\, \dfrac{d^n}{dx^n}\left(e^{-x^2}\right)$, $L_n(x) = \dfrac{e^x}{n!}\dfrac{d^n\left(x^n\, e^{-x}\right)}{dx^n}$.

Principales fonctions du Kit_ETS_FH

Nous donnons ici un tableau des plus importantes fonctions de la librairie Kit_ETS_FH. On consultera le document d'exemples d'utilisation pour des exemples intéressants.

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*) *integral_mcx_d*(*f*, *x*, *a*, *b*)	Si l'expression *f* est du type précédent, cette fonction calcule une intégrale indéfinie par rapport à la variable *x*. 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 sign($a\,x + b$) $f(x)$: la règle requise est utilisée (et donnée!) dans *Derive*.