Laboratoire d'Automatique, Productique et Signal

Volume 1/5

Fractional Calculus Module

User's Guide

Equipe CRONE

Commande – Robotique
Ordres Non Entiers

Version 4.0
April 21, 2006

LAPS - UMR 5131 CNRS
Université Bordeaux 1- ENSEIRB
351 cours de la Libération - F33405 TALENCE Cedex - France
Tél : +33 (0)5 40 00 66 07 - Fax +33 (0)5 40 00 66 44
Email : accueil@laps.u-bordeaux1.fr - URL : www.laps.u-bordeaux1.fr

Equipe CRONE

Commande – Robotique
Ordres Non Entiers

Scientific Responsible:
Alain OUSTALOUP, Professor, ENSEIRB

Coordinator:
Pierre MELCHIOR, Associate Professor, ENSEIRB

Project Manager:
Nicolas PETIT, Engineer

Scientific Contributor and Developers:
Olivier ALTET, Doctor
Mohamed AOUN, PhD Student
Olivier COIS, Doctor
Nicolas PETIT, Engineer
Patrick LANUSSE, Associate Professor, ENSEIRB

Scientific Contributor:
François LEVRON, Associate Professor, University Bordeaux 1
Xavier MOREAU, Associate Professor, University Bordeaux 1
Jocelyn SABATIER, Associate Professor, IUT, Dpt GEII, University Bordeaux 1

Contact: pierre.melchior@laps.u-bordeaux1.fr; crone@laps.u-bordeaux1.fr

Contents

Preface

This software treats of fractional derivative, how to calculate it, how to synthesize it, its applications in mathematics and in engineer science, such as automatic, identification and control. The objective of this toolbox fits with the will to transfer, distribute and enhance the value on international level, as well as in teaching, as in research, or in the industry, of upstream concept developed in laboratory.

The toolbox "***CRONE Toolbox: Fractional Systems Toolbox***" has been developed since the begin of the nineties. It is the subject of several publications, thesis and articles, and has been registered at the "Agence pour la Protection des Programmes" (APP, Software Protection Agency) in 1993 and 1994 [APP94].

At the moment, the toolbox CRONE is made up of four modules, each modulus treats one of the application theme of fractional differentiation :
- Fractional calculus
- Time Domain System Identification by Fractional Model
- Frequency Domain System Identification by Fractional Model
- CRONE Control System Design.

These modules focus on a will to limit, for the beginning, on the scalar case, in order to ensure a progressive and incentive learning to the user.

Matlab was chosen for its numerous advantages : numeric calculation algorithm on complex matrix, high level programming language, graphical display functions, easy IHM creation (menus, capture area, etc …). The portability on other environment is also a significant advantage to facilitate the distribution of this toolbox.

Moreover, most of the university laboratories and the Research and Development industrial services use this software. It becomes in fact a worldwide standard of pluridisciplinary calculus software, particularly in the automatic domain.

The development of this toolbox was done in collaboration with the group PSA Peugeot-Citroën and the financial support of the Aquitaine region.

1 Introduction

Other software tools currently available cannot be used for systems with fractional derivatives (i.e. non integer or complex order derivatives). The "Fractional Calculus" module includes all algorithms which allow the use of fractional or complex derivation. The following modules of the CRONE Toolbox have been developed on the base of the theoretical work of the CRONE team of the LAP, gradually since the end of the eighties :

- "Fractional Derivative" unit,
- "Explicit Form System (Differential Equations)" unit,
- "Implicit Form System" unit,
- "Fractional Differentiator" unit,
- "Fractional Polynomial Roots" unit,
- "Laplace Transform" unit.

2 Principle

"Fractional Derivative" unit

This unit enables the user to compute the fractional or complex order derivative of time functions from literal expressions or from external data files.

The definition of a complex order derivative was given by Riemann-Liouville [MIL93] [SAM93] [COIS00] in the ninetieth century.
The *n* complex order integral of a complex function *f(t)* is defined by :

$$I^n f(t) = \frac{1}{\Gamma(n)} \int_0^t \frac{f(\tau)}{(t-\tau)^{1-n}} d\tau \text{, with } \begin{cases} t > 0 \\ n \in \mathbb{C} \\ \Re e(n) > 0 \end{cases}, \quad (1)$$

where $\Gamma(n)$ is the Euler Gamma function extended to complex numbers.

$$\Gamma(n) = \int_0^\infty e^{-x} x^{n-1} dx \ . \quad (2)$$

Also, the *n* complex order derivative, such as $\mathcal{R}e(n)>0$, of a complex function *f(t)* is defined by :

$$D^n f(t) = \frac{1}{\Gamma(m-n)} \left(\frac{d}{dt}\right)^m \left(\int_0^t \frac{f(\tau)}{(t-\tau)^{1-(m-n)}} d\tau \right), \quad (3)$$

with $\begin{cases} t > 0 \\ \mathcal{R}e(n) > 0 \\ m = \lfloor \mathcal{R}e(n) \rfloor + 1 \end{cases}$, $\lfloor \mathcal{R}e(n) \rfloor$ is the integer part of $\mathcal{R}e(n)$.

Remark

If $\mathcal{R}_e(n)<0$, the n complex order integral of a complex function $f(t)$ is then defined by :

$$I^n f(t) = D^{-n} f(t), \tag{4}$$

also the n complex order derivative is defined by :

$$D^n f(t) = I^{-n} f(t). \tag{5}$$

♦

The use of this first definition has many drawbacks. A second definition which computes integer, real or complex order derivatives is the work basis on which the computation algorithm of the fractional derivative of the CRONE toolbox is developed. The idea underlying this definition consists to generalize the integration notion illustrated by the surface under the plot by introducing the memory notion (and a forgetting factor) with a weighting higher for the oldest samples, the weighting being function of the order of the derivation.

The definition [OUS91], [MIL93], [SAM93], can developed into an algorithm which computes a fractional derivative with a data vector [GRU67] :

$$D_h^{(n)} f(t) = \frac{1}{h^n} \sum_{k=0}^{+\infty} \left[(-1)^k \binom{n}{k} f(t-kh) \right] \tag{6}$$

The calculation error is in $O(h)$, where h is the sample period. A second algorithm can improve the precision, and enables to have an error in $O(h^2)$, $O(h^3)$,… depending on the option chosen by the user. In this case the following relation is used :

$$\begin{aligned} D^{(n)}f(t) &= D_h^{(n)}f(t) + \frac{nh}{2}D_h^{(n+1)}f(t) \\ &+ \frac{(3n^2+5n)h^2}{24}D_h^{(n+2)}f(t) + \frac{(n^3+5n^2+6n)h^3}{48}D_h^{(n+3)}f(t) \\ &+ \frac{(15n^4+150n^3+485n^2+502n)h^4}{5760}D_h^{(n+4)}f(t) + O\left(h^5\right) \end{aligned} \tag{7}$$

Remark

The computation algorithm of the fractional derivative gives a result with an asymptotic error which can be estimate with the following relation:

$$D_h^n\left(f(t)\right) = \frac{1}{h^n}\sum_{k=0}^{\infty}(-1)^k\binom{n}{k}f(t-kh), \tag{8}$$

The Laplace transform of this expression is:

$$L\left[D_h^n\left(f(t)\right)\right] = \frac{1}{h^n}\sum_{k=0}^{\infty}(-1)^k\binom{n}{k}F(p)\exp(-khp) = F(p)\left(\frac{1-\exp(-hp)}{h}\right)^n. \tag{9}$$

Using Taylor development of the exponential term, it becomes:

$$L\left[D_h^n\left(f(t)\right)\right] = F(p)\left[\frac{1}{h}\left(1-\left(1-hp+\frac{h^2p^2}{2!}-\frac{h^3p^3}{3!}+O\left(h^4\right)\right)\right)\right]^n, \tag{10}$$

whence

$$L\left[D_h^n\left(f(t)\right)\right] = p^n\left(1-\frac{hp}{2!}+\frac{h^2p^2}{3!}-\frac{h^3p^3}{4!}+\frac{h^4p^4}{5!}+O\left(h^5\right)\right)^n F(p), \tag{11}$$

Using Taylor development of the term $(1-u)^n$, one finally finds :

$$L\left[D_h^n\left(f(t)\right)\right] = p^n\left(1-\frac{n}{2}hp+\left(\frac{n+3n^2}{24}\right)h^2p^2-\left(\frac{n^3+n^2}{48}\right)h^3p^3+O\left(h^4\right)\right)F(p). \tag{12}$$

The inverse Laplace transform of this expression is:

$$D_h^n(f(t)) = D^n(f(t)) - \frac{n}{2}hD^{n+1}(f(t)) + \frac{n+3n^2}{24}h^2D^{n+2}(f(t)) - \frac{n^3+n^2}{48}h^3D^{n+3}(f(t)) + O(h^4), \quad (13)$$

Using the following expression:

$$D^n(f(t)) = D_h^n(f(t)) + a_1(n)hD_h^{n+1}(f(t)) + a_2(n)h^2D_h^{n+2}(f(t)) + a_3(n)h^3D_h^{n+3}(f(t)) + O(h^4), \quad (14)$$

Substituting the expression of $D^n(f(t))$ in the equation (13) by the equation (14) gives:

$$D_h^n(f(t)) = D_h^n(f(t)) + \left(a_1(n) - \frac{n}{2}\right)hD_h^{n+1}(f(t)) + \left(a_2(n) - a_1(n+1)\frac{n}{2} + \frac{n+3n^2}{24}\right)h^2D_h^{n+2}(f(t)) + O(h^3), \quad (15)$$

Whence

$$\begin{cases} a_1(n) - \frac{n}{2} = 0 \\ a_2(n) - a_1(n+1)\frac{n}{2} + \frac{n+3n^2}{24} = 0 \\ \vdots \end{cases} \quad (16)$$

The result is:

$$\begin{aligned} D^{(n)}f(t) = {} & D_h^{(n)}f(t) + \frac{nh}{2}D_h^{(n+1)}f(t) \\ & + \frac{(3n^2+5n)h^2}{24}D_h^{(n+2)}f(t) + \frac{(n^3+5n^2+6n)h^3}{48}D_h^{(n+3)}f(t) \\ & + \frac{(15n^4+150n^3+485n^2+502n)h^4}{5760}D_h^{(n+4)}f(t) + O(h^5) \end{aligned} \quad (17)$$

The error is directly linked to the sample period *h*.

♦

"Explicit Form System (Differential Equations)" unit

In a time description of the dynamic behavior of a scalar linear system, a differential equation is "fractional" when the first member is a linear combination of fractional derivatives of output signals, and the second member a linear combination of fractional derivatives of input signals [OUS91]:

$$\sum_{r=1}^{R} a_r D^{(n_r)} s(t) = \sum_{q=1}^{Q} a_q D^{(n_q)} e(t) . \qquad (18)$$

The second member is immediately calculable with algorithms from "Fractional derivative" unit. In the first member, the output s(t) is determined with the past of the signal by the following relation:

$$\sum_{r=1}^{R} \frac{a_r}{h^{n_r}} s(t) = \sum_{q=1}^{Q} a_q D^{(n_p)} e(t) - \sum_{r=1}^{R} a_r \left[\frac{1}{h^{n_r}} \sum_{k=1}^{+\infty} \left((-1)^k \binom{n_r}{k} s(t-kh) \right) \right] . \qquad (19)$$

This unit enables to simulate the time response of such a differential equation with complex order derivations (so real and integer are included).

This unit was extended to the linear multivariable processes (MIMO). These systems are described by fractional differential equations with linked inputs and outputs [NAN96]:

$$\left\{ \sum_{v=1}^{V} \sum_{r=1}^{R} a_{r,v} D^{(n_{r,v})} s_v(t) = \sum_{w=1}^{W} \sum_{q=1}^{Q} a_{q,w} D^{(n_{q,w})} e_w(t) . \qquad (20)$$

In the same way, the outputs are given at the time t according to the past of the outputs and inputs signals.

In the monovariable and multivariable cases, the algorithm uses the second definition of fractional order derivative [OUS91], [MIL93], [SAM93] :

$$D^{(n)}e(t)=\frac{1}{h^n}\sum_{k=0}^{+\infty}\left[(-1)^k\binom{n}{k}e(t-kh)\right]. \tag{21}$$

"Implicit Form System" unit

This unit allows to simulate the time response of a process described by a transfer function with implicit fractional derivative [OUS91], such as :

$$\frac{S(s)}{E(s)} = \frac{\sum_p \left[\prod_q \left(1+\tau_{p,q} s\right)^{n_{p,q}} \right]}{\prod_k \left(1+\tau_k s\right)^{n_k}}. \qquad (22)$$

Example of implicit form system :

$$\frac{S(s)}{E(s)} = \frac{(1+2.89s)^{0.7}(1+5s)^{1.2+0.8i}(1+4.77s)^{0.4} + (1+3.65s)^{1.6}}{(1+8s)^{0.6+1.3i}(1+1.32s)^{3.8}} \qquad (23)$$

♦

There is no simple form of differential equation of this transfer function with implicit fractional derivative.
Indeed, the inverse Laplace transform of the transfer function :

$$S(s) = (1+\tau s)^n E(s) \qquad (24)$$

is: $$s(t) = \tau^n \exp\left(\frac{-t}{\tau}\right) D^n_{imp,\tau}\left(e(t)\right). \qquad (25)$$

Remark
Setting $s' = s + 1/\tau$, we obtain :

$$S(s'-1/\tau) = \tau^n s'^n E(s'-1/\tau) \qquad (26)$$

The inverse Laplace transform of the equation is :

$$s(t)\exp\left(\frac{t}{\tau}\right) = \tau^n D^n \left(e(t)\exp\left(\frac{t}{\tau}\right)\right), \qquad (27)$$

thus: $$s(t) = \tau^n \exp\left(\frac{-t}{\tau}\right) D^n_{imp,\tau}(e(t)). \quad (28)$$

♦

To compute the time response, the output S(s) is expressed according to the input E(s):

$$S(s) = \sum_{p=0}^{P}\left[\prod_{q=0}^{Q_p}\left(1+\tau_{p,q}s\right)^{n_{p,q}}\left[\prod_{k=0}^{K}\left(1+\tau_k s\right)^{-n_k}\left(E(s)\right)\right]\right] \quad (29)$$

From this form, the output signal s(t) is then determined by:

$$\boxed{\begin{aligned} s(t) &= \sum_{p=0}^{P}\left\{\tau_{p,Q_p}^{n_{p,Q_p}} e^{\left(\frac{-t}{\tau_{p,Q_p}}\right)} D^{n_{p,Q_p}}_{imp,\tau_{p,Q_p}}\left[\ldots\left[\tau_{p,0}^{n_{p,0}} e^{\left(\frac{-t}{\tau_{p,0}}\right)} D^{n_{p,0}}_{imp,\tau_{p,0}}\left(f(t)\right)\right]\ldots\right]\right\} \\ avec\ \ f(t) &= \left\{\tau_K^{-n_K} e^{\left(\frac{-t}{\tau_K}\right)} D^{-n_K}_{imp,\tau_K}\left[\ldots\left[\tau_0^{-n_0} e^{\left(\frac{-t}{\tau_0}\right)} D^{-n_0}_{imp,\tau_0}\left(e(t)\right)\right]\ldots\right]\right\} \end{aligned}}. \quad (30)$$

Although this expression is rather heavy to write, it is relatively simple to program with the fractional derivative algorithm, defined by the relation:

$$D^{(n)}e(t) = \frac{1}{h^n}\sum_{k=0}^{+\infty}\left[(-1)^k\binom{n}{k}e(t-kh)\right]. \quad (31)$$

"Fractional Differentiator" unit

Because the fractional derivative takes into account all the past of the function, its real time calculation cannot be carried out; only the fractional derivative of the functions with finished past can be calculated.

In the general case, it is advisable to calculate the fractional derivative with a recursive equation obtain through discretisation of a fractional derivative which truncates low and high frequencies.

This unit synthesizes a frequency-bounded fractional differentiator:

$$D_{fbl}(s) = C_0 \left(\frac{1+\frac{s}{\omega_b}}{1+\frac{s}{\omega_h}} \right)^n \tag{32}$$

The synthesis of the rational differentiator uses a recursive distribution of real zeros and poles and a rational order derivative unit:

$$D_{rational}(s) = C_0 \prod_k \left(\frac{1+\frac{s}{\omega_{bk}}}{1+\frac{s}{\omega_{hk}}} \right) \tag{33}$$

N is the number of zeros and poles and the recursion factors α and η are determined by the relations: $n = \frac{\log \alpha}{\log \alpha\eta}$, and $\alpha = \left(\frac{\omega_h}{\omega_b} \right)^{n/N}$ & $\eta = \left(\frac{\omega_h}{\omega_b} \right)^{(1-n)/N}$ and $\frac{\omega_{hk}}{\omega_{bk}} = \alpha$ & $\frac{\omega_{b(k+1)}}{\omega_{hk}} = \eta$.

To compare the frequency responses of the three differentiators are displayed in a Bode diagram and Nichols charts:

- fractional differentiator :

$$D_{fractional}(s) = C_0 \left(\frac{s}{\omega_u} \right)^n \tag{34}$$

- frequency-bounded differentiator:

$$D_{fbl}(s) = C_0 \left(\frac{1 + \frac{s}{\omega_b}}{1 + \frac{s}{\omega_h}} \right)^n \tag{35}$$

- rational differentiator:

$$D_{rational}(s) = C_0 \prod_k \left(\frac{1 + \frac{s}{\omega_{bk}}}{1 + \frac{s}{\omega_{hk}}} \right) \tag{36}$$

This comparison makes it possible to evaluate the degradation related to the approximation of the fractional differentiator by a recursive distribution of zeros and poles.

"Fractional Polynomial Roots" unit

Defined as the denominator of the transmittance of a fractional differential equation equaled to zero, the characteristic equation can have integer, fractional, real or complex powers:

$$\sum_{l=1}^{L} a_l \, s^{n_l} = 0 \, . \qquad (37)$$

This unit enables to calculate the roots of such an equation. Its resolution in the particular case of real fractional powers rests on the approximation of these powers by rational powers. A variable change makes it possible to be brought back to an integer degrees equation of fractional order variable. The roots of this equation are then given in accordance with a cut of the complex plane following $\mathbb{R}$- (the cut can be parameterized).

The algorithm which results from this is the one which uses the module "Fractional Polynomial Roots" of CRONE toolbox.

The poles of a system can be given by specifying either the denominator of transmittance, or the matrix of evolution A, or the vector of the eigenvalues.

"Laplace Transform" unit

The current algorithms of numerical calculation of Laplace transform or inverse Laplace transform, are not very satisfying. The Maple software, for example, allows calculation symbolic system of simple functions, but is inoperative for the functions met in the case of fractional derivation.

New methods must be developed to allow the user to solve fractional differential equations. Within this framework, a new method is proposed; this one calculates a numerical approximation of Laplace transform of a real or complex function, or inverse Laplace transform. The precision obtained with this method is however excellent only in the small times (or at the high frequencies). This is why this subject of search remains open. De nouvelles méthodes doivent être développées pour permettre la résolution d'équations différentielles généralisées.

It was demonstrated that the optimal expression with a precision point of view of the Laplace transform F(s) (reciprocally inverse Laplace transform f(t)) may be approximated by a finished sum of terms depending on the original function f(t) (resp. function F(s) symbolic system), in which the time variable t (resp. the variable symbolic system s) is replaced by a new variable proportional to its Laplace transform 1/s (resp. with its original 1/t).

Thus, the method consists in determining this series by a direct optimization aiming at minimizing the difference between its Taylor development and that of the Laplace transform (reciprocally, the inverse Laplace transform) of the original function f(t) (resp. the function F(s) symbolic system) written in the form of a polynomial sum of the variable t (variable s). This optimization is carried out by solving the system of Aitken.

We assume the function f(t) can be rewrite in this form :

$$f(t) = \sum_{k=0}^{2N-1} \left(\frac{a_k t^{(a-1+kb)}}{\Gamma(a+kb)} \right). \qquad (38)$$

with 2N the term numbers of the series.

The Laplace transform is then:

$$\mathcal{L}(f(t)) = \sum_{k=0}^{2N-1}\left(\frac{a_k}{s^{(a+kb)}}\right). \qquad (39)$$

The coefficients a and b depend on properties of the function f(t):
- a depends on the asymptotic behavior of f(t) near zero $f(t) \approx At^{a-1}$.
- b is the common step of all powers of the function f(t) written in the form of series (for example, b=1 if the function is odd, and b=2 if the function is even).

We obtain an approximate expression:

$$\mathcal{L}(f(t)) = \sum_{k=1}^{N}\left(A_k f(\frac{x_k}{s})\right), \qquad (40)$$

which leads to determine A_l and x_l so to solve the equation :

$$\sum_{l=1}^{N} A_l x_l^{(a-1+kb)} = \frac{\Gamma(a+kb)}{s}, \text{ with } 0 \le k \le 2N-1. \qquad (41)$$

Thus, the method consists in solving the system of N equations to determine N first terms of the series (eq. .41), and to calculate the Laplace transform of the function f(t). The precision obtained with this method is excellent, but only at small times.

For the inverse Laplace transform, the result is completely similar.
This new method and the associated developments will be presented in a specific article. We will present the method in it but also a comparison with the principal already existing methods of numerical calculation.

Bibliography

[APP94] Agence pour la Protection des Programmes - N° 93.30.006.00 (IDDN.FR.001.300006.00. R.P.1993.000.00000 - 28/07/1993) ; N° 94.11.015.00 (IDDN.FR.001.110015.00. R.P.1994.000.00000) - le16/03/1994.

[BAN93] M. Bansard Du formalisme de la dérivation généralisée à l'unité Outils Mathématiques de la toolbox CRONE, Thèse de Doctorat, Université Bordeaux I, 1993.

[COIS00] O. Cois et F. Levron - Systèmes à dérivées d'ordre complexe - IEEE Conférence Internationale Francophone d'Automatique, CIFA'2000, sous l'égide du GdR Automatique du CNRS et du GRAISYyHM - Lille, France, 5-8 Juillet 2000.

[GRU67] A.K. Grünwald, Über "begrenzte" Derivation und deren Anwendung, Z. Angew. Math. Phys., 12, 441-480, 1867.

[LEV99] F. Levron, J. Sabatier, A. Oustaloup and L. Habsieger - From partial differential equations of propagative recursive systems to non integer differentiation - Fractional Calculus and Applied Analysis (FCAA) : an international journal for theory and applications, Vol. 2, N° 3, pp 246-264, July 1999.

[MEL99] P. Melchior, P. Lanusse, F. Dancla et O. Cois, Valorisation de l'approche non entière par le logiciel CRONE, Colloque sur l'Enseignement des Technologies et des Sciences de l'Information et des Systèmes en EEA, CETSIS-EEA'99, Montpellier, 1999.

[MIL93] K.S. Miller and B. Ross, An introduction to the fractional calculus and fractional differential equations, A Wiley-Interscience Publication, 1993.

[NAN96] F. Nanot, "Dérivateur Généralisé et Représentation Généralisée des Systèmes Linéaires", Thèse de Doctorat, Université Bordeaux I, 1996.

[OLD74] K. B. Oldham and J. Spanier, The fractional calculus, Academic Press, New York and London, 1974.

[OUS83] A. Oustaloup, Systèmes asservis linéaires d'ordre fractionnaire, Masson, 1983.

[OUS97] A Oustaloup, P. Lanusse et P. Melchior, Le logiciel CRONE , Journées d'Etude sur les Logiciels pour le traitement de l'Image, du Signal et l'Automatique, Club EEA - Elisa'97, Nancy, Mars 1997.

[OUS91] A. Oustaloup, "La commande CRONE", Hermès 1991.

[OUS95] A. Oustaloup, La dérivation non entière: théorie, synthèse et applications, Editions Hermès, Paris, 1995.

[OUS00a] A. Oustaloup, F. Levron, B. Mathieu and F. Nanot - Frequency-Band Complex Non Integer Differentiator : Characterization and Synthesis - IEEE Transactions on Circuits and Systems, Vol. 47, N°1, pp 25-40, January 2000.

[OUS00b] A. Oustaloup, P. Melchior, P. Lanusse, O. Cois and F. Dancla, The CRONE toolbox for Matlab, 11th IEEE Inrternational Symposium on Computer-Aided Control System Design, CACSD, Anchorage, Alaska - USA, September 2000.

[SAM93] S.G. Samko, A.A. Kilbas and O.I. Marichev, Fractional integrals and derivatives: theory and applications, Gordon and Breach Science Publishers, 1993.

3 Object oriented programming

Definitions

The object-oriented programming consists in modelling, by means of computer, a whole of elements of a part of the real world (which called ***field***) in a whole of data-processing entities. These data-processing entities are called ***objects***. They are data-processing which gather the principal characteristics of the elements of the real world. An object is characterized by several concepts:

Attributes:
They are the data characterizing the object. They are variables which store information representing state of the object
Methods:
The methods of an object characterize its behavior, i.e. the whole of the actions that the object is capable to realize. These operations make possible the reaction between object and the external requests (or to act on the other objects). Moreover, the operations are closely related to the attributes, because their actions can depend on the values of the attributes, or modify them.
A class is the structure of an object, i.e. the declaration of all entities which will compose an object. An object results thus from a class. Actually it is said that an object is an instantiation of a class, this is why we will be able to speak indifferently about “object” or “instance”.
A class is made up of two parts:
The attributes:
They are the data which represent the state of the object.
The methods:
They are the operations applicable to the objects.

Interests

The POO have three interests:

Encapsulation

It is a mechanism which consists in gathering the data and the methods within a structure by hiding the implementation of the object, i.e. by preventing the access to the data by another means that the services suggested. The encapsulation thus makes it possible to guarantee the integrity of the data contained in the object.

Heritage

The heritage is a principle suitable for the object-oriented programming, making it possible to create a new class starting from an existing class. The name of "heritage" (called also derivation of class) comes owing to the fact that the derived class (the class lately created) contains the attributes and the methods of its super class (the class from which it derives). The major interest of the heritage is to be able to define new attributes and new methods for the derived class, which come to be added to those inherited. By this means we can create a hierarchy of increasingly specialized classes. That has a great interest; we don’t begin from zero when we want to create a new class.

Polymorphism

Polymorphism is a mechanism which makes it possible to have the same functions name with different parameters (in a number and/or type). The automatic choice of the good method to be adopted is done according to the type of data passed in parameter.

4 Frac_poly objects

Introduction

Toolbox CRONE makes it possible to handle fractional polynomials (frac_poly), i.e. polynomials whose orders are not integers.
These polynomials can be either explicit (for example the polynomial $a_n.s^n + a_m.s^m + a_p.s^p + a_q.s^q$), or implicit (i.e. in the form $(\text{exp}licit\ polynomial)^{order}$).

Description of the fractional polynomials

a) Explicit fractional polynomials frac_poly_exp:

i) Creation of the explicit fractional polynomials:

The method frac_poly_exp makes it possible to create an explicit fractional polynomial, whose coefficients and orders are given in parameters to the function.
An explicit fractional polynomial

$$P(s) = a_n.s^n + a_m.s^m + a_p.s^p + a_q.s^q$$

is characterized by it's coefficients [a_n, a_m, a_p, a_q] and its orders [n, m ,p, q].

$$P = frac\_poly\_\exp([a_n, a_m, a_p, a_q],[n, m, p, q])$$

Creates the polynomial P(s), considered as a frac_poly_exp object

For example:

$$P = frac\_poly\_\exp([1,1,1],[2,0.2,0])$$

Create the fractional object

$$P(s) = s^2 + s^{0.2} + 1$$

From this object, we can extract the vectors of the coefficients, the vector of the orders, and even the variable of the polynomial "s", a variable which we can change by giving the new variable in parameter of the function frac_poly_exp, this variable can be "s", "p", "z" or "q". For example:

$$Q = frac\_poly\_\exp([1,1],[0.2,0],'z')$$

Creates the explicit fractional object:

$$Q(z) = z^{0.2} + 2$$

The toolbox also makes it possible to add two explicit polynomials, to multiply them and also to withdraw them.

For example, if we consider the two following polynomials:

$$P(s) = s^2 + s^{0.2} + 1$$

$$Q(s) = s^{0.2} + 2$$

Then:

$$R(s) = P(s) + Q(s) = s^2 + 2.s^{0.2} + 3$$

$$T(s) = P(s) * Q(s) = s^{2,2} + 2.s^2 + s^{0.4} + 3.s^{0,2} + 2$$

$$V(s) = P(s) - Q(s) = s^2 - 1$$

The method *frac_poly_exp* can also turn over a matrix of polynomials, for that we have to provide a cell matrix of coefficients and a cell matrix of orders in parameters of entry.

For example, if we consider the two matrices:

$$C = \begin{Bmatrix} [1\ 1] & 1 & 1 \\ 1 & [1\ 2\ 1] & 1 \\ 1 & 1 & [1\ 3\ 3\ 1] \end{Bmatrix}$$

$$O = \begin{Bmatrix} [0.5\ 0] & 0.2 & 0.6 \\ 4 & [1.1\ 0.7\ 0] & 0.4 \\ 2 & 0.5 & [1.2\ 0.8\ 0.4\ 0] \end{Bmatrix}$$

Then:

$\mathrm{A} = \mathrm{frac\_poly\_exp}(\mathrm{C}, \mathrm{O})$ returns

$$A = \begin{pmatrix} s^{0.5} + 1 & s^{0.2} & s^{0.6} \\ s^{4} & s^{1.1} + 2 * s^{0.7} + 1 & s^{0.4} \\ s^{2} & s^{0.5} & s^{1.2} + 3 * s^{0.8} + 3 * s^{0.4} + 1 \end{pmatrix}$$

ii) Attributes of the frac_poly_exp objects:

The preceding section shows how to create frac_poly_exp objects, which encapsulate the data relative to an explicit fractional polynomial. This section will give the various attributes of these objects, i.e. the various parts of information which are attached to such an object. Those are described in the table below:

Attribute name	**Description**	**Value**
Coef	Coefficients of the polynomial	Cell matrix M*N
Order	Orders of the polynomial	Cell matrix M*N
Variable	Variable of the polynomial	String ‘s’, ‘p’, ‘z’ or ‘q’

Notice
The attributes “Coef” and “Order” must have the same dimensions.

b) Implicit fractional polynomials frac_poly_imp:

i) Creation of the implicit fractional polynomials:

The method frac_poly_imp makes it possible to create an implicit fractional polynomial. This method takes in entry the data relating to the polynomial and produces a frac_poly_imp object.

An implicit fractional polynomial

$$P(s) = \left(\exp licit\ polynomial\right)^{order}$$

is characterized by an "Explicit polynomial" (frac_poly_exp object of order 1) and an "Implicit order".

$$P = frac\_poly\_imp(fpe, imp\_order)$$

creates the polynomial P(s), considered as a *frac_poly_imp* object.

For example:

$$\mathrm{P(s) = frac\_poly\_imp(1,0.3,1)}$$

creates the implicit polynomial $P(s) = (1+s)^{0,3}$.

We can also create a matrix of implicit polynomials, for that we have to provide in parameters of the function *frac_poly_imp* a "Fpe" as cell matrix of frac_poly_exp matrix (1*M) and an "Order" in the form of a cell matrix of vector(1*M).

For example:

$$A = \begin{Bmatrix} (s+1,\ 2.s+1) & 2.s+1 & s+1 \\ s+2 & 3.s+1 & s+1 \\ s+3 & 4.s+2 & 3.s+1 \end{Bmatrix}$$

$$O = \begin{Bmatrix} [0.5\ 0.2] & 0.2 & 0.6 \\ 0.8 & 0.7 & 0.4 \\ 1.2 & 0.5 & 0.1 \end{Bmatrix}$$

$$\mathrm{P(s) = frac\_poly\_imp(A, O)}$$

creates the matrix of implicit polynomials following:

$$A = \begin{pmatrix} (s+1)^{0.5} * (2.s+1)^{0.2} & (2.s+1)^{0.2} & (s+1)^{0.6} \\ (s+2)^{0.8} & (3.s+1)^{0.7} & (s+1)^{0.4} \\ (s+3)^{1.2} & (4.s+1)^{0.5} & (3.s+1)^{0.1} \end{pmatrix}$$

ii) Attributes of the frac_poly_imp object

The attributes of the *frac_poly_imp* objects are describe in the table below:

Attribute name	Description	Value
fpe	Fractional polynomial of the implicit polynomial	Cell matrix M*N
Imp_order	Order of the polynomial	Cell matrix M*N

Notice
The attributes “fpe” and “imp_order” must have the same dimensions.

5 Frac_tf objects

Introduction:

Toolbox CRONE offers extensive tools to manipulate fractional transfer functions (frac_tf), i.e. transfer functions which numerator and denominator are fractional polynomials (frac_poly objects).

Creation of the fractional transfer function frac_tf

The method frac_tf makes it possible to create a fractional transfer function, i.e. transfer functions which numerator and denominator are polynomials whose orders are not integers.

A fractional transfer function

$$T = frac\_tf(num, den)$$

Is characterized by its numerator « Num », and its denominator "Den", both of them are fractional polynomials.

For example, if we consider the following fractional polynomials:

$$P1 = \left(1 + s^{0.6}\right)$$

$$P2 = s^{2.1} + 2.s + 1$$

Then

$$T = frac\_tf(P1, P2)$$

Creates the fractional transfer function

$$T(s) = \frac{1 + s^{0.6}}{s^{2.1} + 2.s + 1}$$

The toolbox also makes it possible to add two transfer functions, to multiply them or to withdraw them, but this provided that the numerator and denominator are two explicit polynomials.

Example
If we consider the two fractional transfer functions following:

$$T(s) = \frac{1}{s^{0.2} + 1}$$

$$F(s) = \frac{s}{s^{1.1} + 1}$$

Then:

$$S(s) = T(s) + F(s) = \frac{s^{1.2} + s^{1.1} + s + 1}{s^{1.3} + s^{1.1} + s^{0.2} + 1}$$

$$P(s) = T(s) * F(s) = \frac{s}{s^{1.3} + s^{1.1} + s^{0.2} + 1}$$

$$D(s) = T(s) - F(s) = \frac{-s^{1.2} + s^{1.1} - s + 1}{s^{1.3} + s^{1.1} + s^{0.2} + 1}$$

Attributes of frac_tf objects

The attributes of the frac_tf objects are describe in table below:

Attribute name	Description	Value
Num	Numerator	Frac_poly_exp or Frac_poly_imp object
Den	Denominator	Frac_poly_exp or Frac_poly_imp object
Variable	Variable of frac_tf object	String ‘s’, ‘p’, ‘z’ or ‘q’

6 Graphic User Interface

The graphic user interface of the module ***Fractional Calculus*** is made up of pull-down menus and dialog boxes making it possible to enter the data and the parameters useful for calculations. The main window has a menu bar as follows organized :

File : manages all data backup
Fractional Derivative : computes a fractional order derivative
Explicit Form System : computes the time and frequency response of a system described by a system of differential equations or by its transfer functions
Implicit Form System : computes the time and frequency response of Implicit Form System described by its implicit fractional transfer functions
Fractional Differentiator : synthesizes a rational differentiator
Fractional Polynomial Roots : computes the fractional polynomial roots
Laplace Transform : computes Laplace transform and inverse Laplace transform
Help : help menu

6 Graphic User Interface

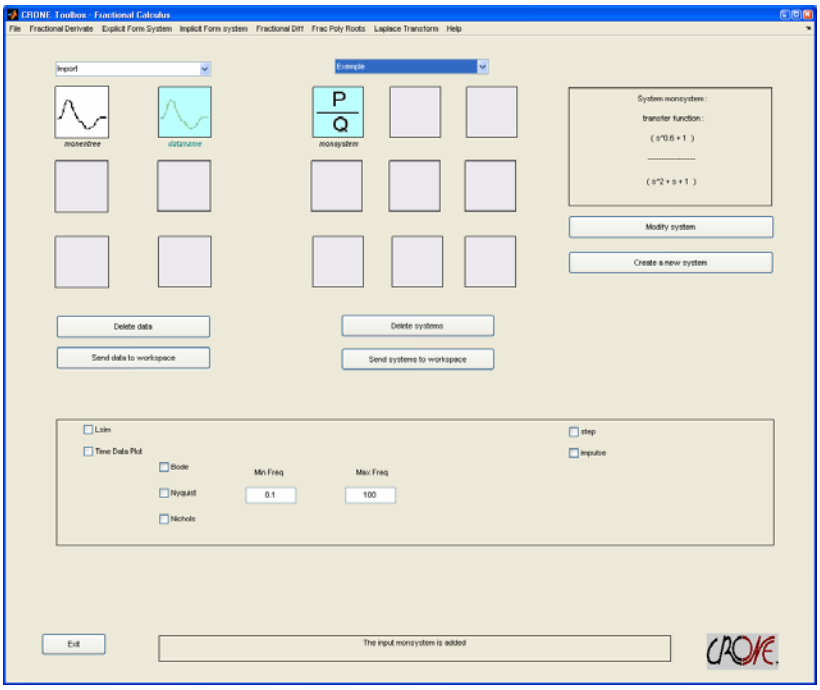


Figure 1 : main window of the module ***Fractional Calculus***

"File" Menu

File menu :

New session :	begins a new session
Open session :	opens a session from file of saved session
Save session as … :	saves all data of current session
Exit :	quits the module

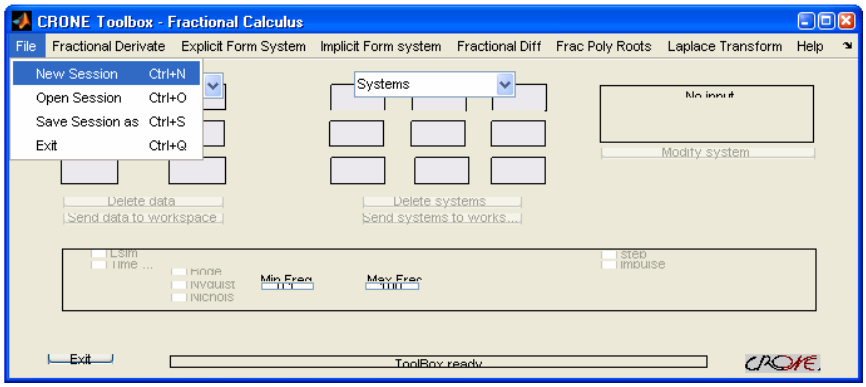


Figure 2 : ***File*** menu

The ***New*** command makes erase the session in progress and to start again a new session. The user can give a title to the session.

The ***Open*** command erase the session in progress and open a session from a file corresponding to a saved session, and in which are the data of the user.

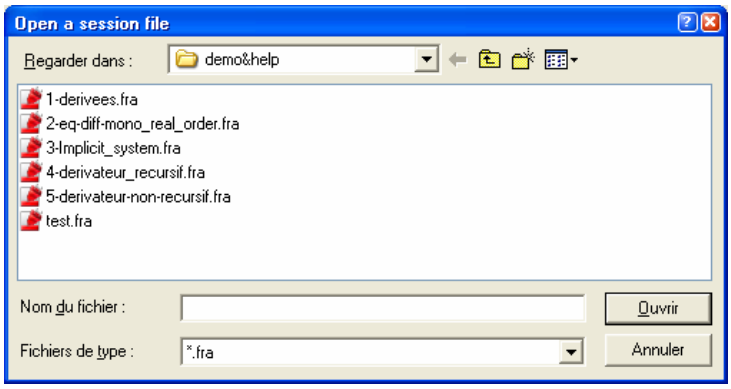


Figure 3 : Dialog box used to retrieve a file

If the selected file is not a session file, an error appears in the status bar:

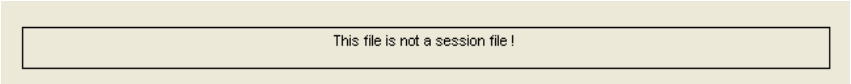


Figure 4 : Status bar display

The ***Save as …*** command save data in a Matlab file (*.fra).

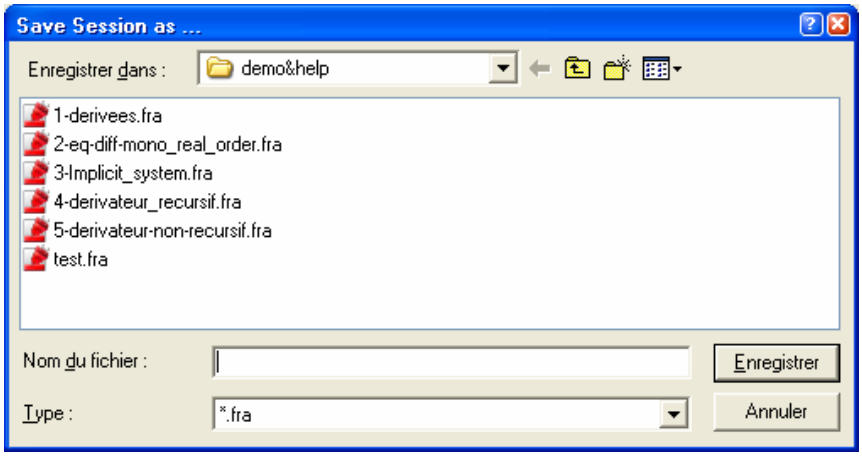


Figure 5 : Dialog box used to save a file

The ***Exit*** command quit the module ***Fractional Calculus***.

When data are present in memory, a message asks if the user wants to save them before carrying out a command.

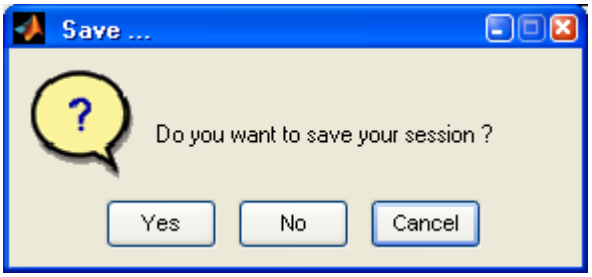


Figure 6 : Question dialog box

"Fractional Derivative" Menu

Fractional Derivative menu :

Data :	sets new data
Data processing :	data processing
Compute :	computes the fractional derivative
Option :	option about the tolerance

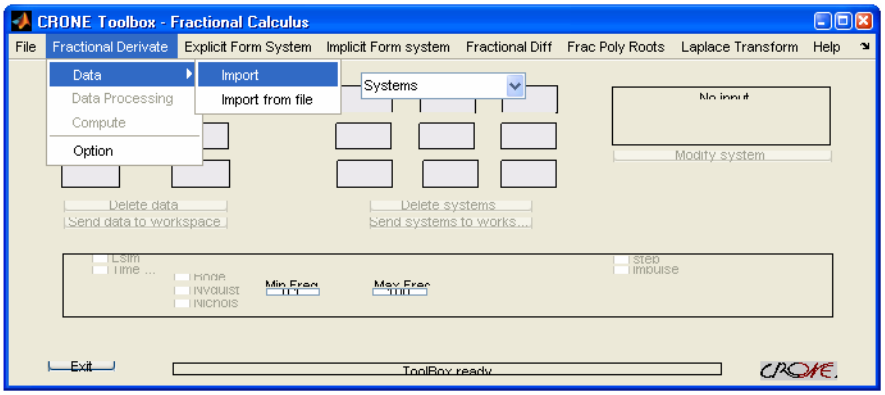


Figure 7 : ***Fractional Derivative*** menu

This menu computes the fractional order derivative of data vector.
This vector is entered by the user with the ***Data*** command.

Data command

The user can import data from Matlab workspace or from saved Matlab files (*.mat).

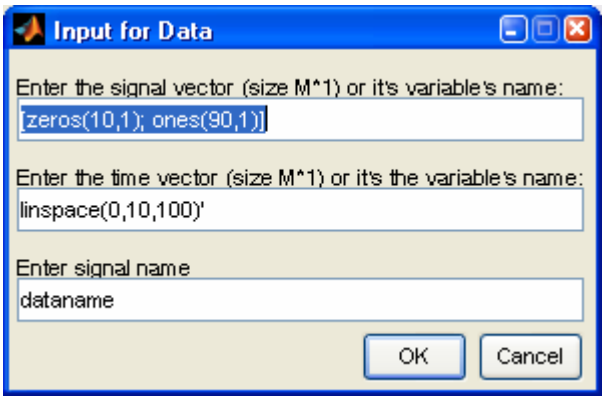


Figure 8 : Import dialog box

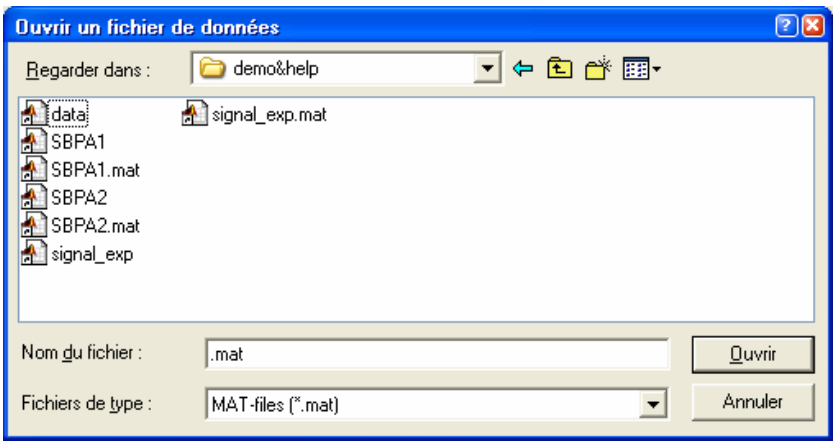


Figure 9 : Dialog box used to retrieve a file

The list of variables included in the selected file appears in the following window :

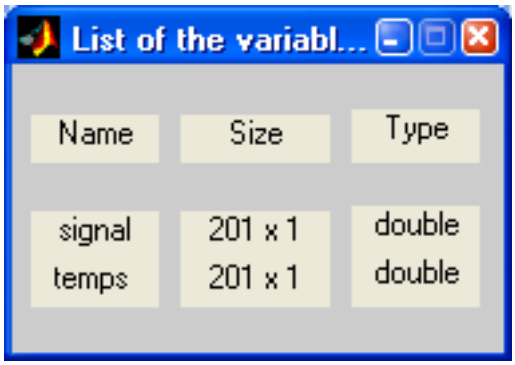


Figure 10 : List of variables

If a file was selected at the previous step, it is then possible to use the variables included in this file and displayed in a window (Figure 10).

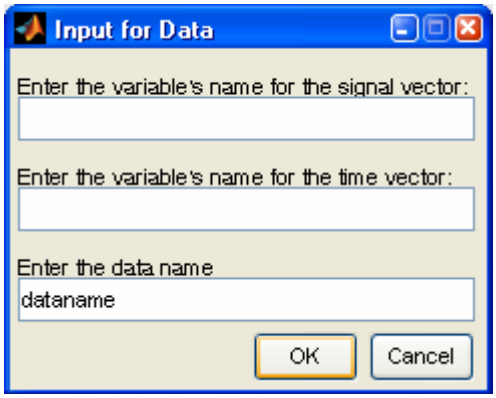


Figure 11 : Import from file dialog box

Data Processing command

The ***Input Signal Processing*** command allows to process the signal in memory. It opens the datashaping window.

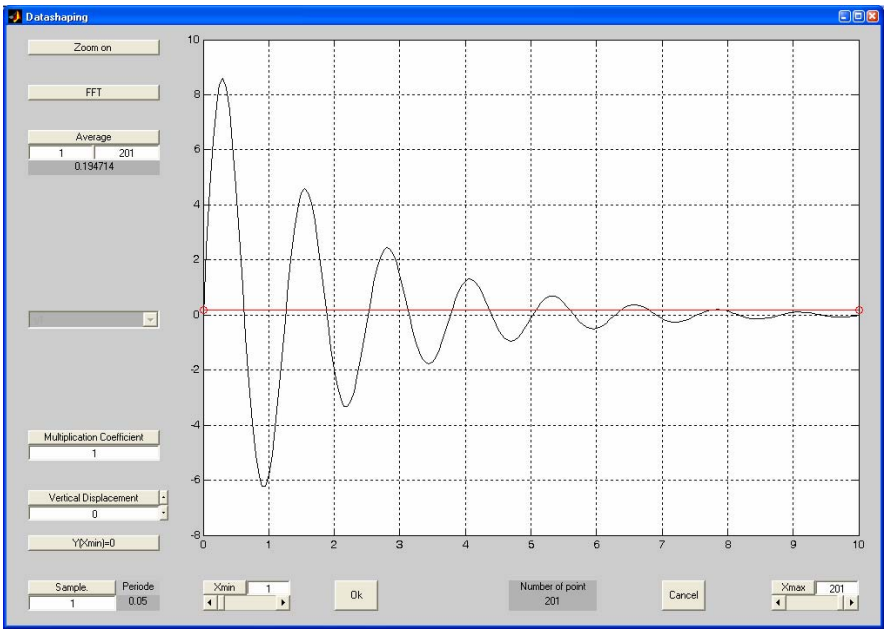


Figure 12 : Datashaping window

The ***FFT*** button gives the FFT of the signal.
The ***average*** button gives and plots the average of the signal from the point in the first box under the button to the point in the second box under the button.
The ***multiplication*** button multiplies the signal by the number given in the box under the button.
The ***vertical displacement*** moves the signal by the number given in the box under the button.
The ***sample*** gives which points will be used to plot the signal, for example if you choose 10 for the sample and you have 200 points for the signal only 20 of them will be used to plot the signal.
The ***Xmin*** button gives the begin point of the plot of the signal.
The ***Xmax*** button gives the ending point of the plot of the signal.

Compute command

The ***Compute*** command computes the fractional derivative.

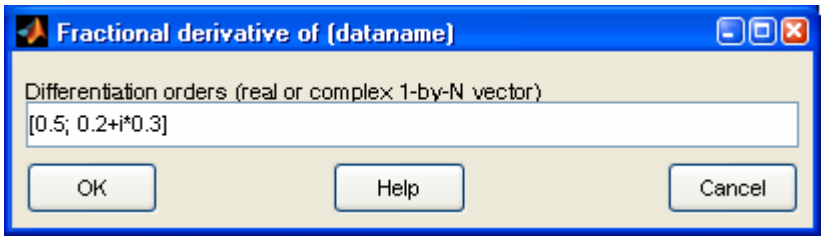


Figure 13 : Editable boxes to input orders

Once the derivation orders are edited, the result is drawn in a new figure.

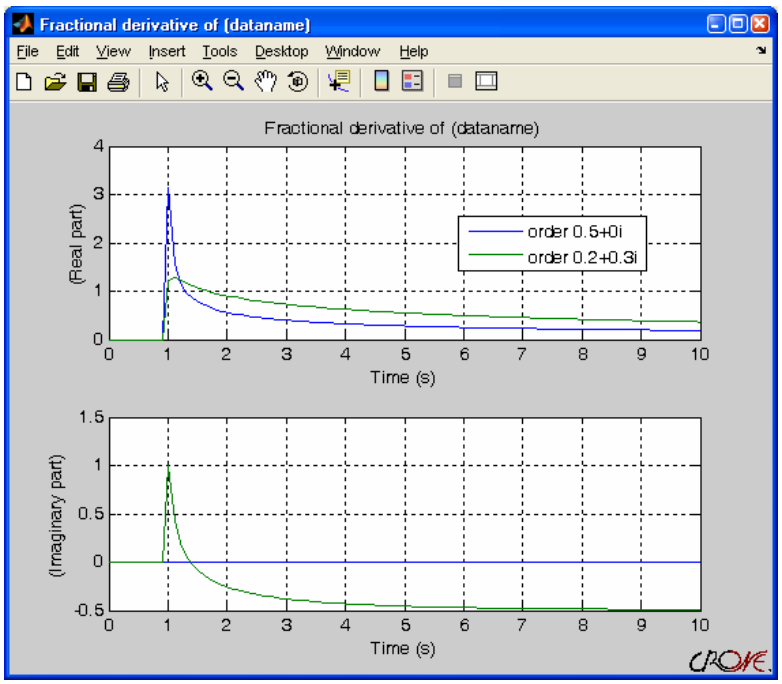


Figure 14 : Results of fractional derivatives

Option Command

The ***Option*** command permits to select the tolerance to obtain a better accuracy in terms of h^2, h^3, etc… (eq. (7).

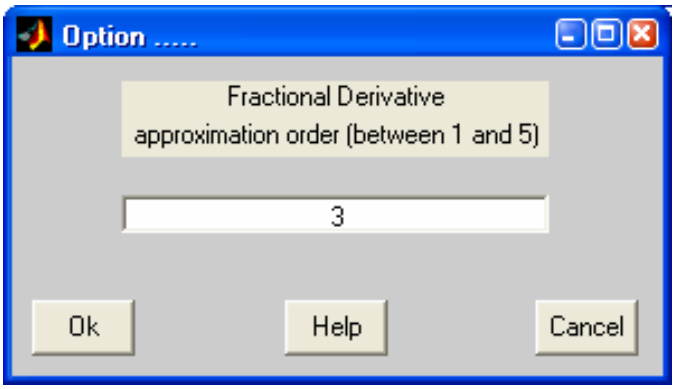


Figure 15 : ***Option*** window

"Explicit Form System (Differential Equations)" Menu

Explicit Form System (Differential Equations) menu :

System definition :	sets the fractional differential equations
Data :	sets input signals
Data processing :	data processing
Output time responses :	computes output time responses
Frequency responses :	computes frequency responses
Eigenvalue and Poles :	computes eigenvalues, zeros and poles

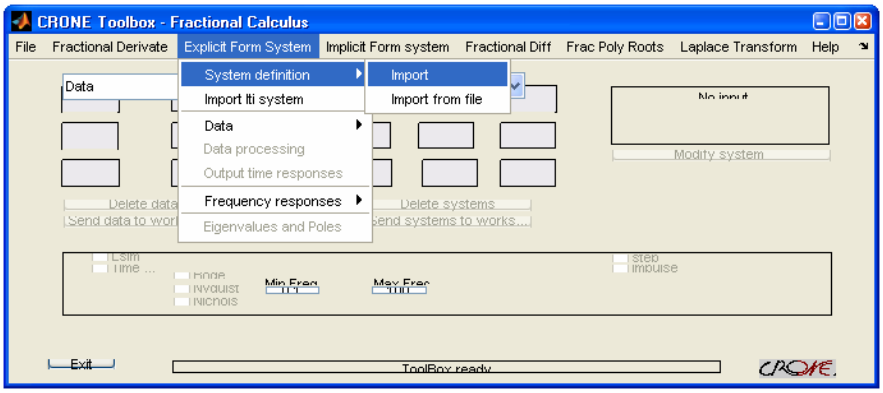


Figure 16 : ***Explicit Form System (Differential Equations)*** menu

System definition command

The user can import data from Matlab workspace or from saved Matlab files (*.mat) to enter the coefficients and the orders of the differential equations describing the system.

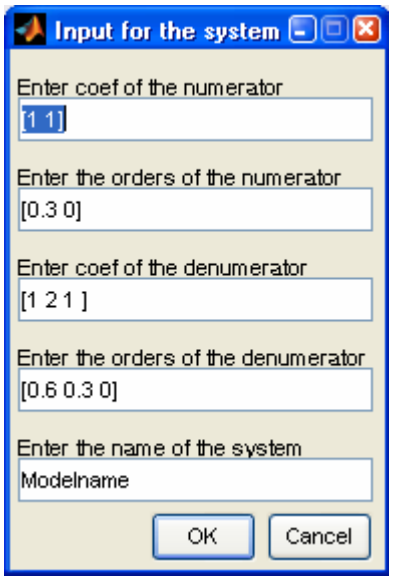


Figure 17 : Import dialog box

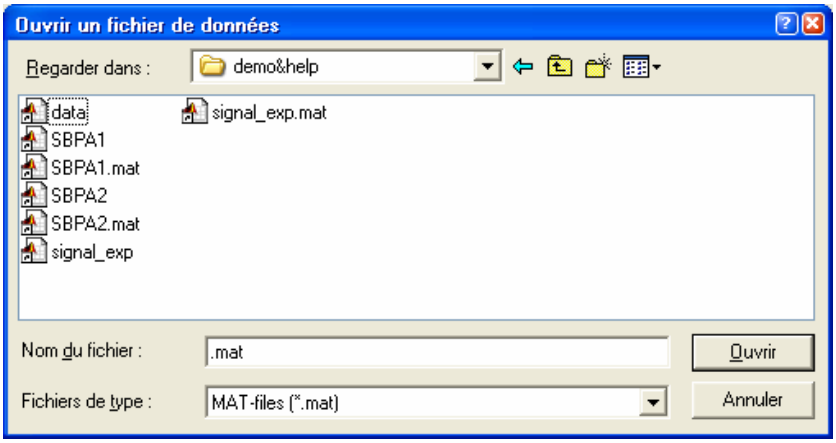


Figure 18 : Dialog box used to retrieve a file

The list of variables included in the selected file appears in the following window:

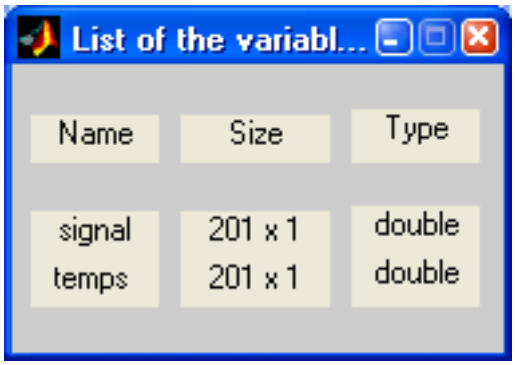


Figure 19 : List of variables

If a file was selected at the previous step, it is then possible to use the variables included in this file and displayed in a window (Figure 10).

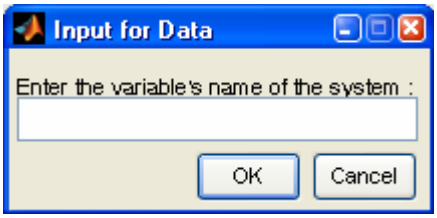


Figure 20 : Import from file dialog box

Data command

The user can import data from Matlab workspace or from saved Matlab files (*.mat).

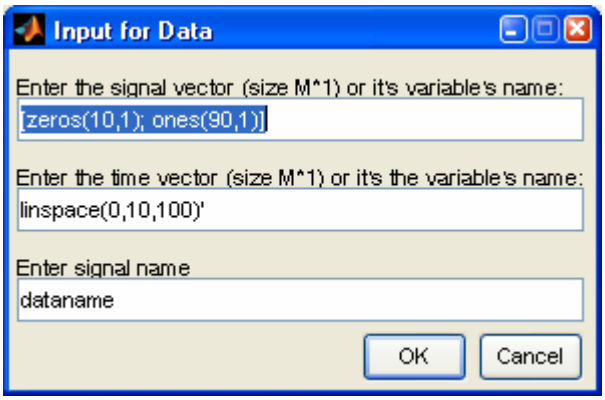


Figure 21 : Import dialog box

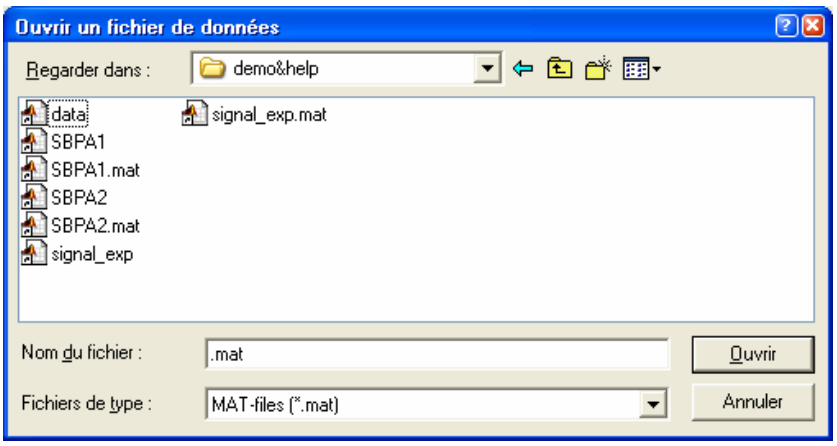


Figure 22 : Dialog box used to retrieve a file

The list of variables included in the selected file appears in the following window:

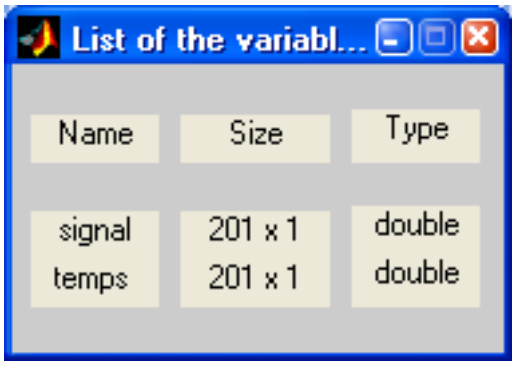


Figure 23 : List of variables

If a file was selected at the previous step, it is then possible to use the variables included in this file and displayed in a window (Figure 10).

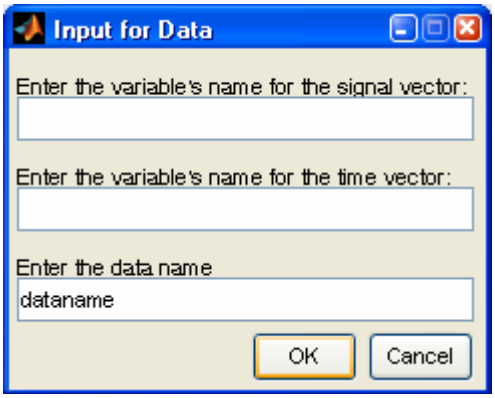


Figure 24 : Import from file dialog box

Data Processing command

The ***Input Signal Processing*** command allows to process the signal in memory. It opens the datashaping window.

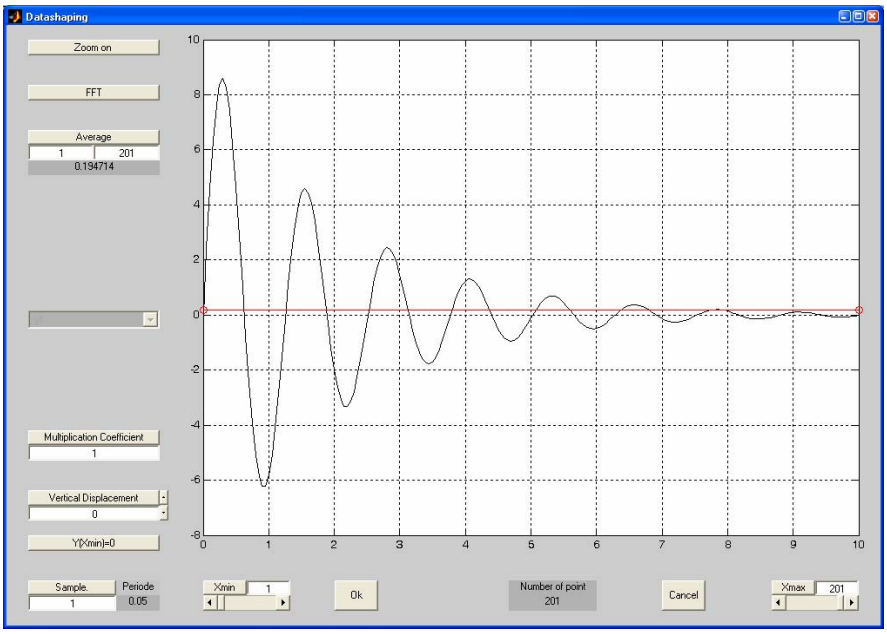


Figure 25 : Datashaping window

The ***FFT*** button gives the FFT of the signal.
The ***average*** button gives and plots the average of the signal from the point in the first box under the button to the point in the second box under the button.
The ***multiplication*** button multiplies the signal by the number given in the box under the button.
The ***vertical displacement*** moves the signal by the number given in the box under the button.
The ***sample*** gives which points will be used to plot the signal, for example if you choose 10 for the sample and you have 200 points for the signal only 20 of them will be used to plot the signal.
The ***Xmin*** button gives the begin point of the plot of the signal.
The ***Xmax*** button gives the ending point of the plot of the signal.

Output time responses command

The ***Compute*** command computes the output time response of the system.

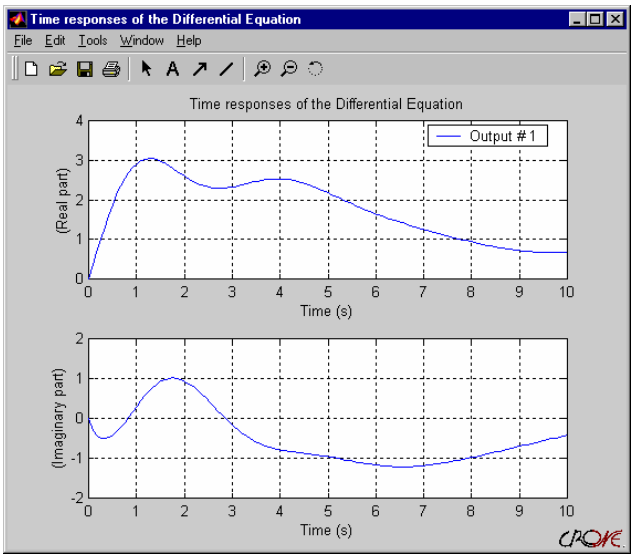


Figure 26 : plots of time responses

Frequency Responses & Eigenvalues and Poles command

The ***Frequency Responses*** submenu includes all commands to plot Bode diagram, Nichols charts and Nyquist plot.

The ***Eigenvalues and Poles*** commands display poles and eigenvalues of the system into a Matlab window.

"Implicit Form System" menu

Although these developments are still being developed, the menus and commands concerning the time and frequency simulation of the implicit form systems are envisaged.

This menu is organized as the ***Explicit Form System (Differential Equations)*** menu:

Implicit Form System menu:

System definition:	sets the implicit form system
Data:	sets input signals
Data processing:	data processing
Output time responses:	computes output time responses
Frequency responses:	computes frequency responses

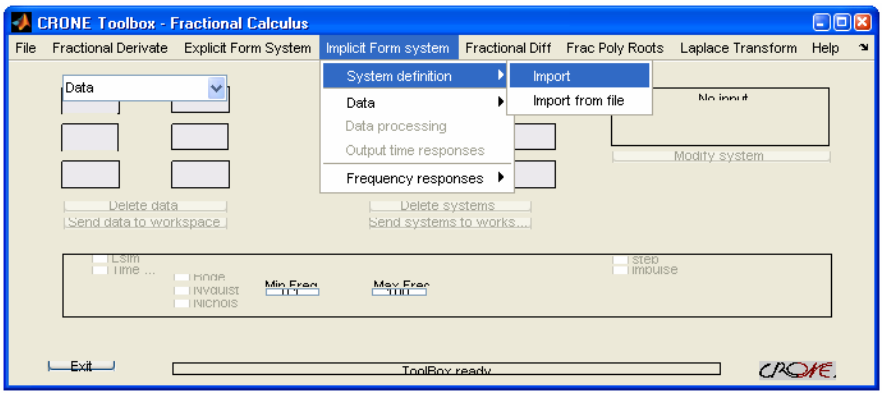


Figure 27 : ***Implicit Form System*** menu

System definition menu

The user can import data from Matlab workspace or from saved Matlab files (*.mat) to enter the coefficients and the orders of the differential equations describing the system.

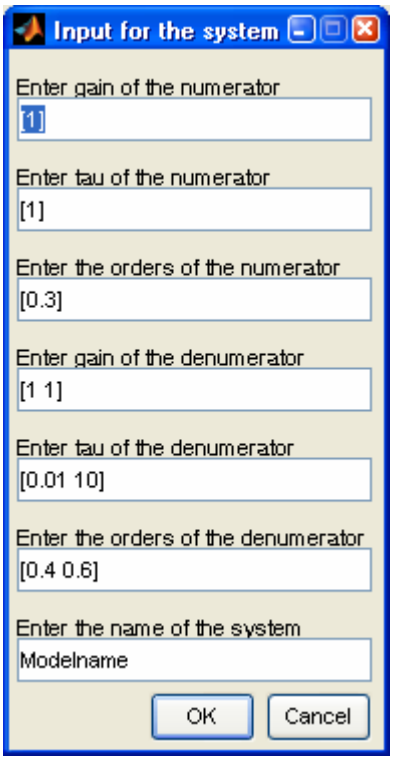


Figure 28 : Import dialog box

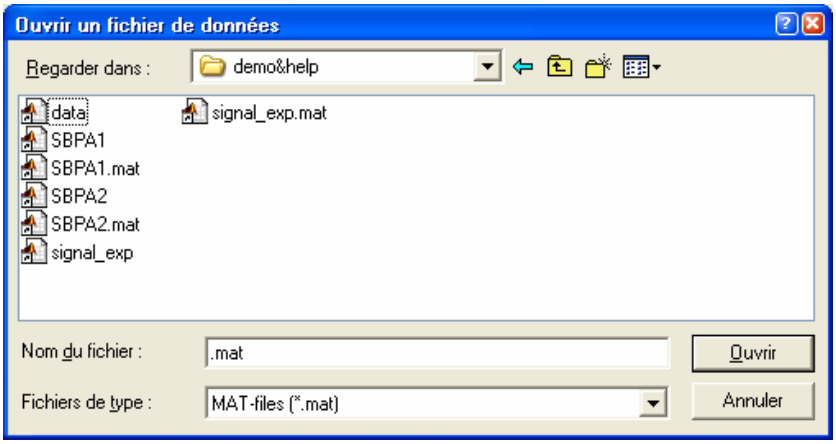


Figure 29 : Dialog box used to retrieve a file

The list of variables included in the selected file appears in the following window :

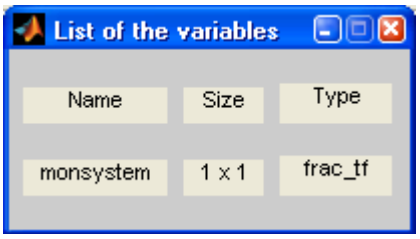


Figure 30 : List of variables

If a file was selected at the previous step, it is then possible to use the variables included in this file and displayed in a window (Figure 10).

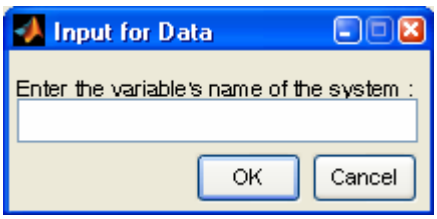


Figure 31 : Import from file dialog box

Data menu

The user can import data from Matlab workspace or from saved Matlab files (*.mat).

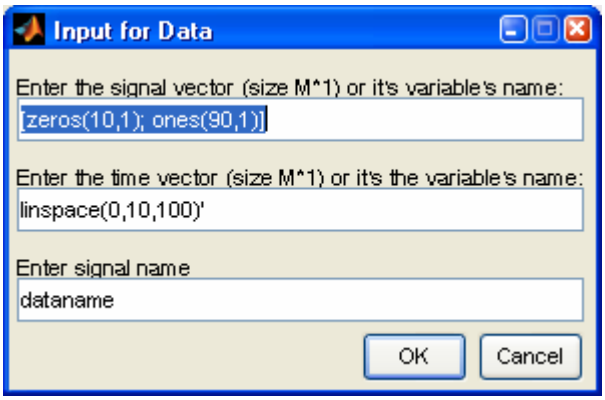


Figure 32 : Import dialog box

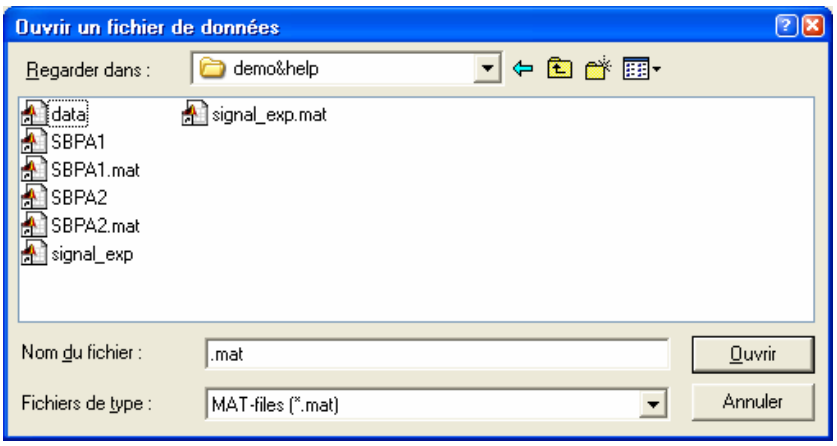


Figure 33 : Dialog box used to retrieve a file

The list of variables included in the selected file appears in the following window:

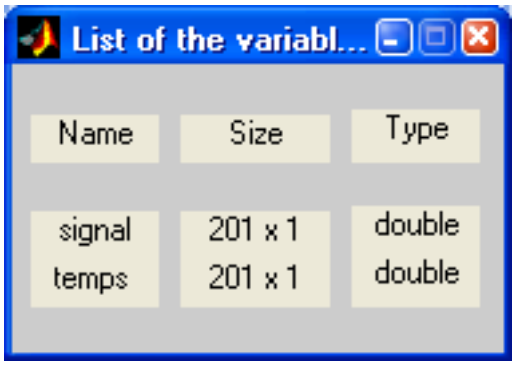


Figure 34 : List of variables

If a file was selected at the previous step, it is then possible to use the variables included in this file and displayed in a window (Figure 10).

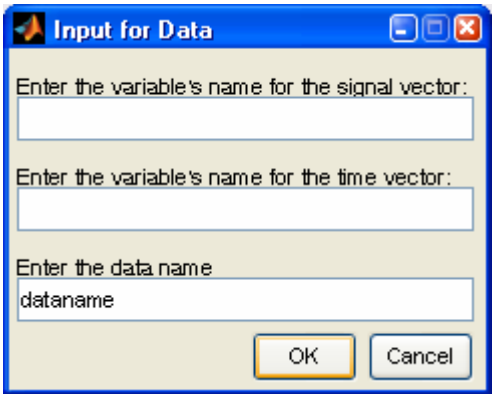


Figure 35 : Import from file dialog box

Data Processing command

The ***Input Signal Processing*** command allows to process the signal in memory. It opens the datashaping window.

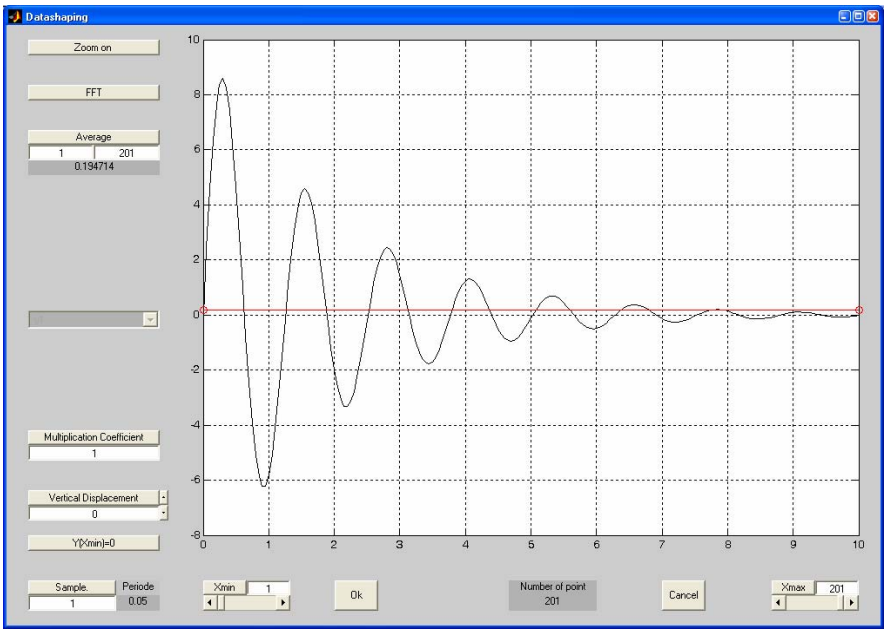


Figure 36 : Datashaping window

The ***FFT*** button gives the FFT of the signal.
The ***average*** button gives and plots the average of the signal from the point in the first box under the button to the point in the second box under the button.
The ***multiplication*** button multiplies the signal by the number given in the box under the button.
The ***vertical displacement*** moves the signal by the number given in the box under the button.
The ***sample*** gives which points will be used to plot the signal, for example if you choose 10 for the sample and you have 200 points for the signal only 20 of them will be used to plot the signal.
The ***Xmin*** button gives the begin point of the plot of the signal.
The ***Xmax*** button gives the ending point of the plot of the signal.

Output time responses command

The ***Compute*** command computes the output time response of the system.

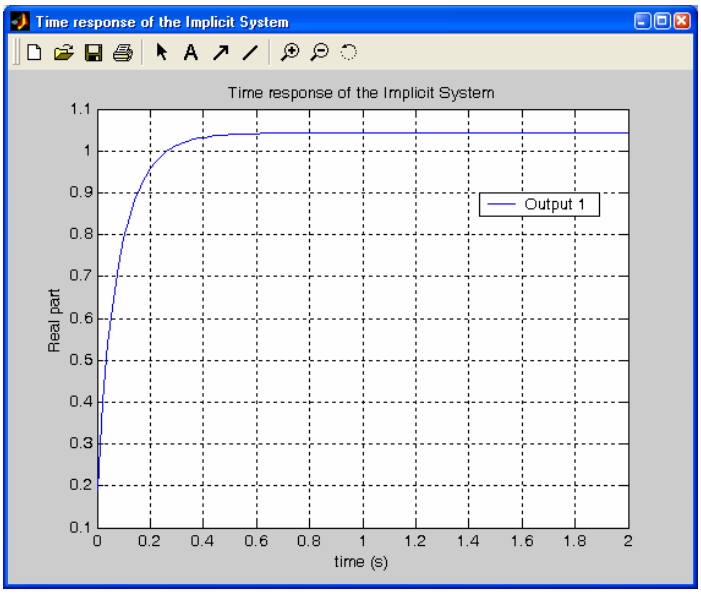


Figure 37 : plots of time responses

Frequency responses

The ***Frequency responses*** menu regroups the commands that draw the Bode, Nichols and Nyquist diagrams.

"Fractional Differentiator" menu

Fractional Differentiator menu:

Differentiator parameters: sets differentiator parameters
View fractional differentiator: displays differentiators
Modify Bode diagrams: modifies parameters and Bode diagrams
Bode diagrams: plots Bode diagrams
Unit choice: sets units and axes properties

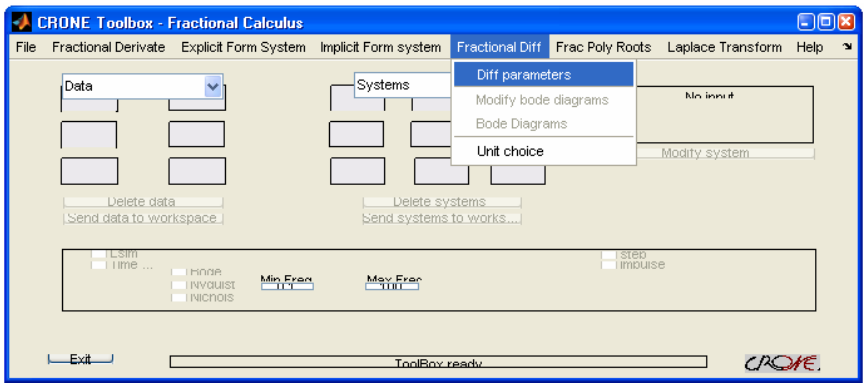


Figure 38 : ***Fractional Differentiator*** menu

Differentiator parameters command

The dialog window allows to set all the parameters of the differentiator. This window includes editable boxes for each parameter.

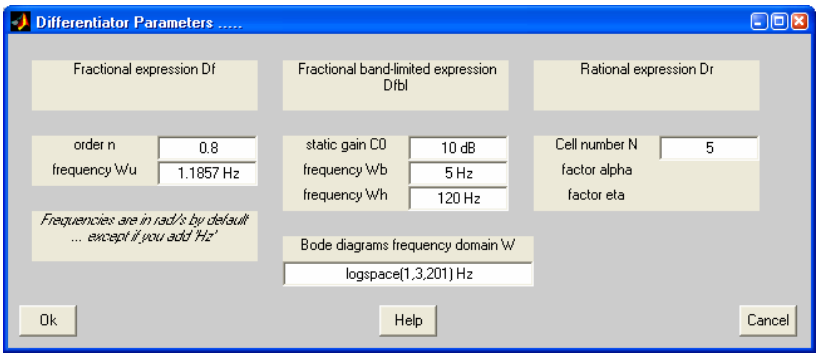


Figure 39 : Parameters editable boxes

The static gain can be specified in linear unit, or in dB; to do that, type the value followed by the text "dB". The frequencies can be also specified either in radian per second (rad/s - by default), or in Hertz; to do that, type the value followed by the text "Hz".

The recursive factors *alpha* and *eta* are deducted from the other parameters.

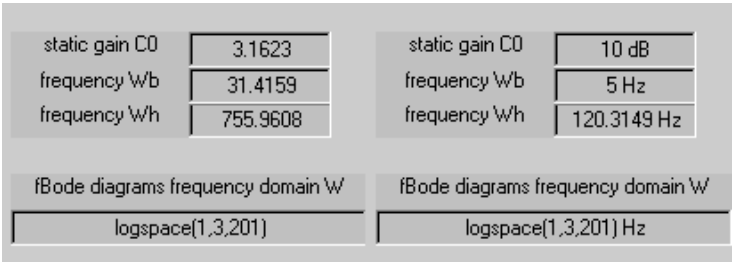


Figure 40 : Editable boxes for gain and frequencies

View fractional differentiator command

This command, will soon allow to display the differentiators:

- fractional differentiator

$$D_{fractional}(p) = C_0 \left(\frac{p}{\omega_u} \right)^n \tag{42}$$

- fractional frequency band-limited differentiator :

$$D_{fbl}(p) = C_0 \left(\frac{1 + \frac{p}{\omega_b}}{1 + \frac{p}{\omega_h}} \right)^n \tag{43}$$

- rational differentiator :

$$D_{rational}(p) = C_0 \prod_k \left(\frac{1 + \frac{p}{\omega_{bk}}}{1 + \frac{p}{\omega_{hk}}} \right) . \tag{44}$$

Modify Bode diagrams command

The ***Modify Bode diagrams*** command synthesizes the differentiator in two ways: either by modifying the numerical values of the parameters, or by moving the poles or the zeros ω_b and ω_h of the frequency-band or rational differentiator.

All editable boxes of differentiators (Figure 39) are displayed again on bode diagrams windows (Figure 41); after each modification, all plots are updated. Conversely, at each modification of the position of a point of the curve, the numerical values of the parameters are updated.

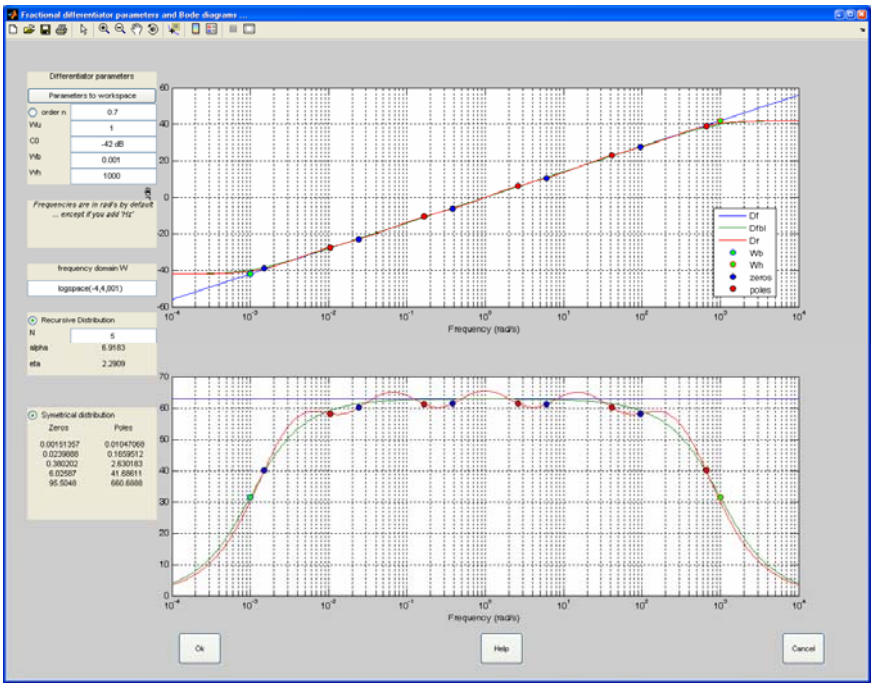


Figure 41 : Plots of Bode diagrams

All the points of the curve can be dragged by the mouse directly on the plot; to do it, place the cursor over the chosen point, select it while pressing on the key and move the point. The new value of the frequency of one of the zeros or one of the poles is used to update the Bode diagrams and editable boxes.

When the zero ω_b or the pole ω_h is moved, only the new frequency of the moved point is modified. The magnitude and the phase in this case are refreshed according to the order and the gain at the unity frequency.
If the editable box of the order n is operational, the new position of the zero ω_b or the pole ω_h makes it possible to compute the new order n of the differentiator, thanks to the X-coordinate (the frequency) and to the Y-coordinate (magnitude or the phase) of the moved point (contrary to the previous case).

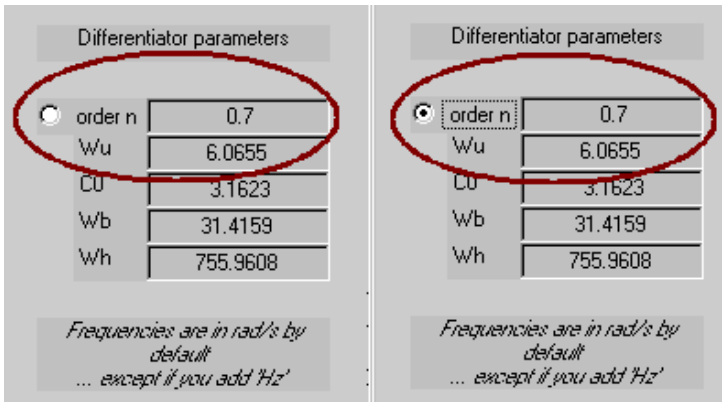


Figure 42 : Modification of the order
with new position of the points ω_b et ω_h
(selected radio button : right case)

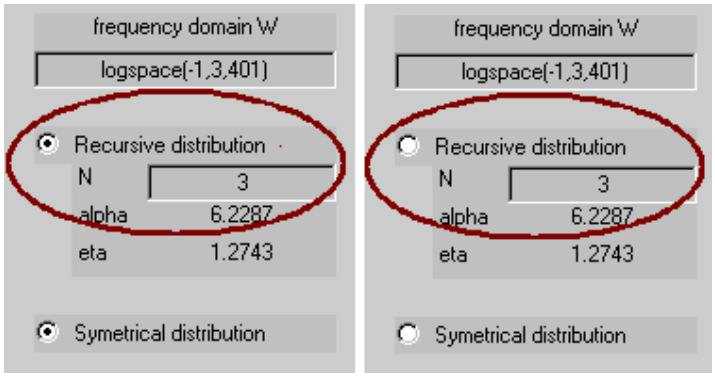


Figure 43 : ***Recursive distribution*** radiobutton

When one zero (or one pole) of the rational differentiator is moved, its symmetrical point compared to ω_b and ω_h is also moved in the same proportions; for example, when the first zero is moved towards the low frequencies, the last pole is moved in the same proportions towards the high frequencies, in an identical way for each zero and pole of the rational differentiator. To modify a single point without influencing "its symmetrical", select the ***Symmetrical distribution*** radiobutton to disable

this functionality; selecting this radiobutton again validates the functionality.

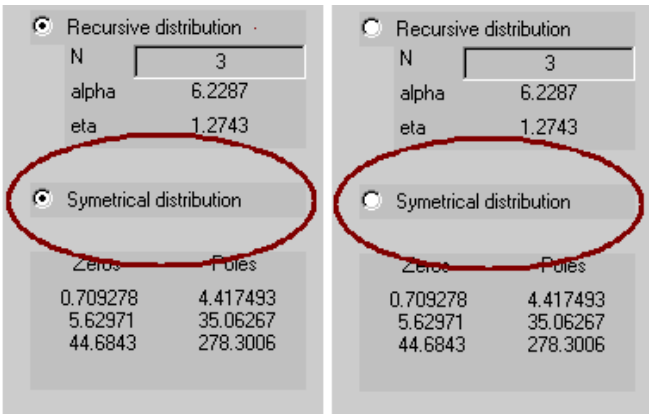


Figure 44 : ***Symmetrical distribution*** radiobutton

Bode diagrams command

This command plots Bode diagrams of the differentiators into a new window (to print the plots for example).

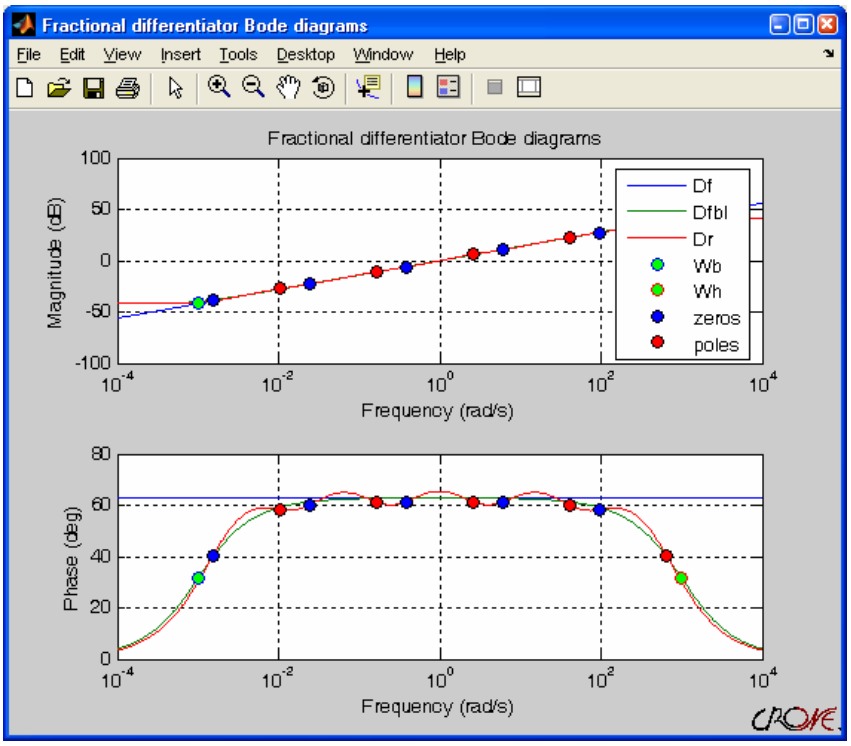


Figure 45 : Plots of Bode diagrams

Unit choice command

This command makes it possible to set the units and axes scale. Moreover, the frequency responses can be displayed according to the frequencies either in Hertz (Hz), or in radian per second (rad/s). The magnitude can be displayed either in linear scale, or in decibel (dB).

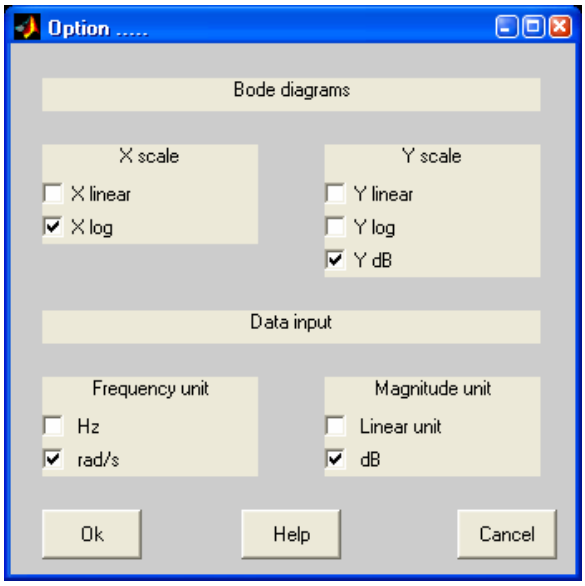


Figure 46 : ***Unit choice*** window

In this example, the frequency response is plotted on Bode diagrams (with logarithmic X-coordinate and linear Y-coordinate in dB) with the angular frequency (rad/s); the magnitude is displayed in dB.

"Fractional Polynomial Roots" menu

The ***Fractional Polynomial roots*** command computes the fractional polynomial roots.

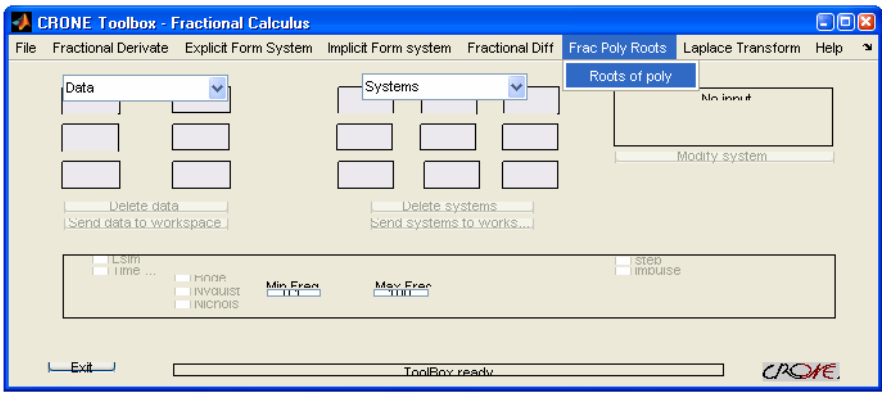


Figure 47 : ***Fractional Polynomial Roots*** menu

The coefficients and orders are set in the following input dialog box :

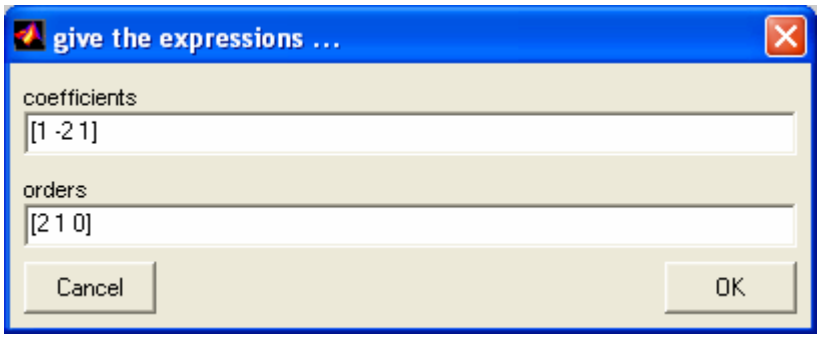


Figure 48 : Input dialog box

The results are displayed into a window as this example:

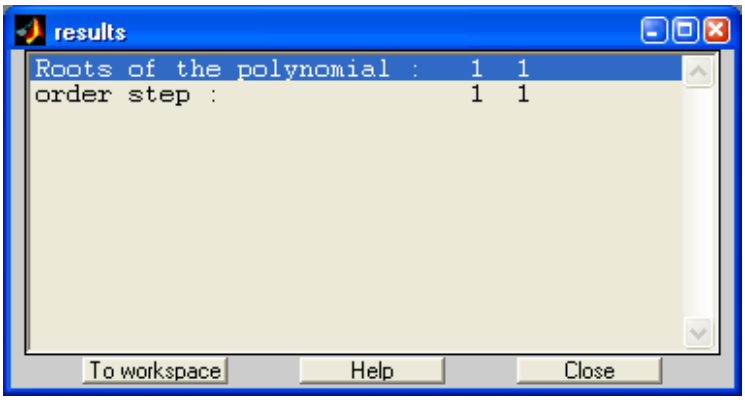


Figure 49 : Example of polynomial roots

"Laplace transform" menu

Laplace Transform menu:

Laplace transform:	computes Laplace transform
Inv Laplace transform:	computes inverse Laplace transform
Option:	method option

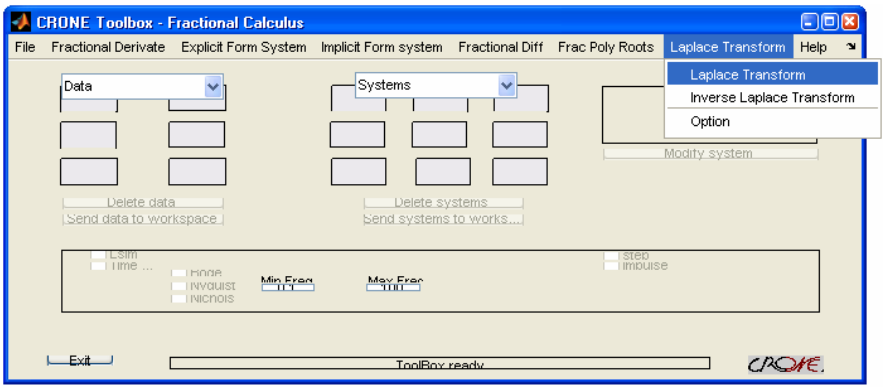


Figure 50 : ***Laplace transform*** menu

This menu makes it possible to compute either the Laplace transform of a function, or the inverse Laplace transform of a function. To carry out calculation, the frequency or time vector is required. The function must be written in matlab language knowing that the variables are vectors.

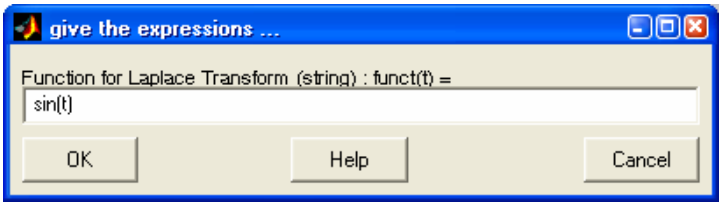


Figure 51 : Function to transform

Then the user is asked to give the decade frequency limit of the band and the coefficient a and b of the function form. The frequency limits is in decade units so if you set the frequency limits at [1 3] the frequency range will be between 10 and 10^3. The coefficients a and b are the coefficients of the function form which is:

$$f(t) = \sum_{k=0}^{\infty} \left(\frac{a_k t^{(a-1+kb)}}{\Gamma(a+kb)} \right).$$

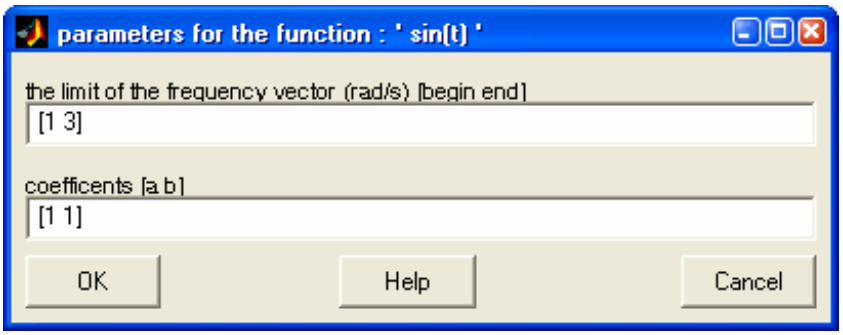


Figure 52 : Frequency vector and parameters a & b of the function

The software then asks you to check your options.
The ***Option*** command sets tolerance and maximum iteration for the Aitken method.
The number of terms is the number N explained in the principles.
The tolerance is the maximum difference of the computed laplace transform between two iteration.

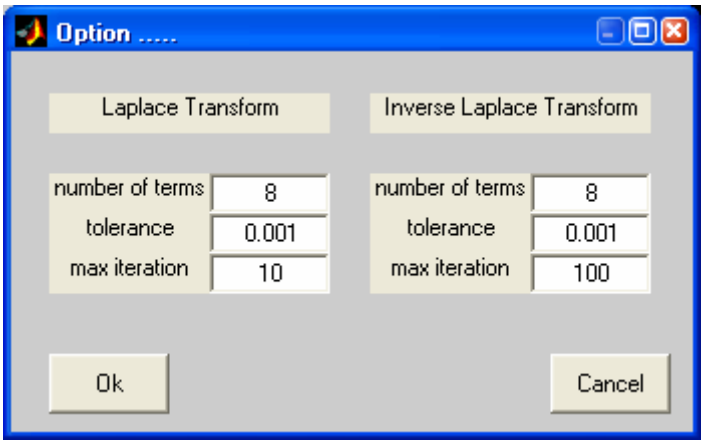


Figure 53 : ***Option*** dialog boxes about Laplace transform and inverse Laplace transform

The result is then plotted.

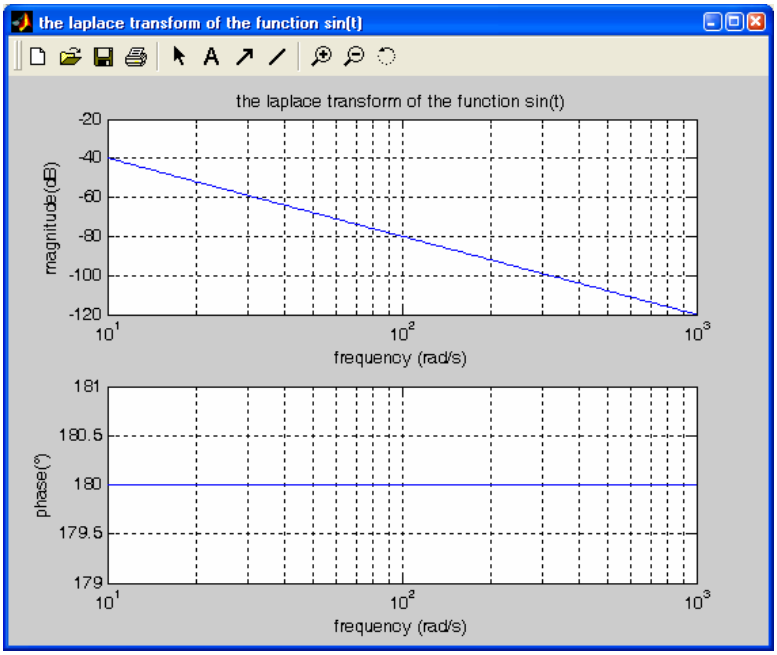


Figure 54 : Result of Laplace transform

"Help" menu

Help menu:

Differentiator parameters: sets differentiator parameters
Contents : Describes briefly the unit
Index : Index of the help
About …

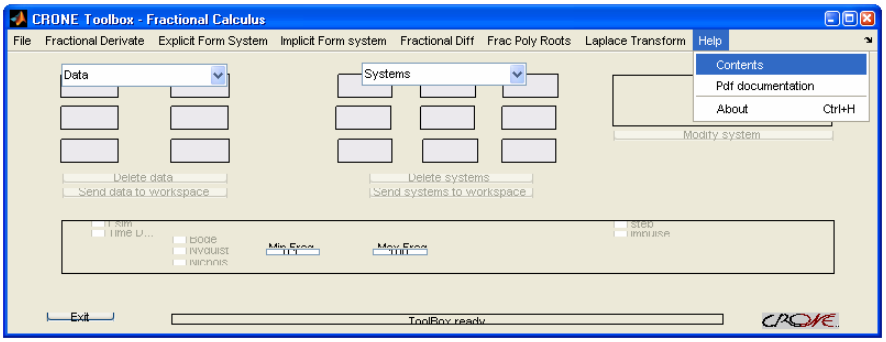


Figure 55 : ***Help*** menu

7 Reference

In this paragraph, the significant functions of the ***Fractional Calculus*** module are presented by topic.

The organization of the variables was set up to standardize their use, and to simplify the lines of command by using structured variables.

In addition, the objective is to develop similar commands of the Matlab functions, and to make functions of ***CRONE toolbox***, an extension of the Matlab functions.

bode

Bode frequency response of fractional transfer functions

Syntax

```
    bode(Tf)
    bode(Tf ,W)
[Mag ,Phase ,W]=bode(Tf)
[Mag ,Phase]=bode(Tf,W)
```

Description

Bode computes the magnitude and phase of the frequency response of fractional transfer functions. When invoked without left-hand arguments, bode produces a Bode plot on the screen.
Bode(T*f*) produces a Bode plot of the fractional transfer function T*f*. The frequency range is determined automatically based on the system poles and zeros.
Bode(Tf, W) explicitly specifies the frequency range or frequency points to be used for the plot. To focus on a particular frequency interval [Wmin,Wmax], set w = {Wmin,Wmax}. To use particular frequency points, set w to the vector of desired frequencies.
When invoked with left-hand arguments,

```
[Mag ,Phase ,W]=bode(Tf)
[Mag ,Phase]=bode(Tf,W)
```

return the magnitude and phase of the frequency response at the frequency W.

Arguments

Argument in :
Tf : fractional object
W : frequency range (vector or cell)

Argument out :
Mag : magnitude (vector)
Phase : phase (vector)

W : frequency range (vector)

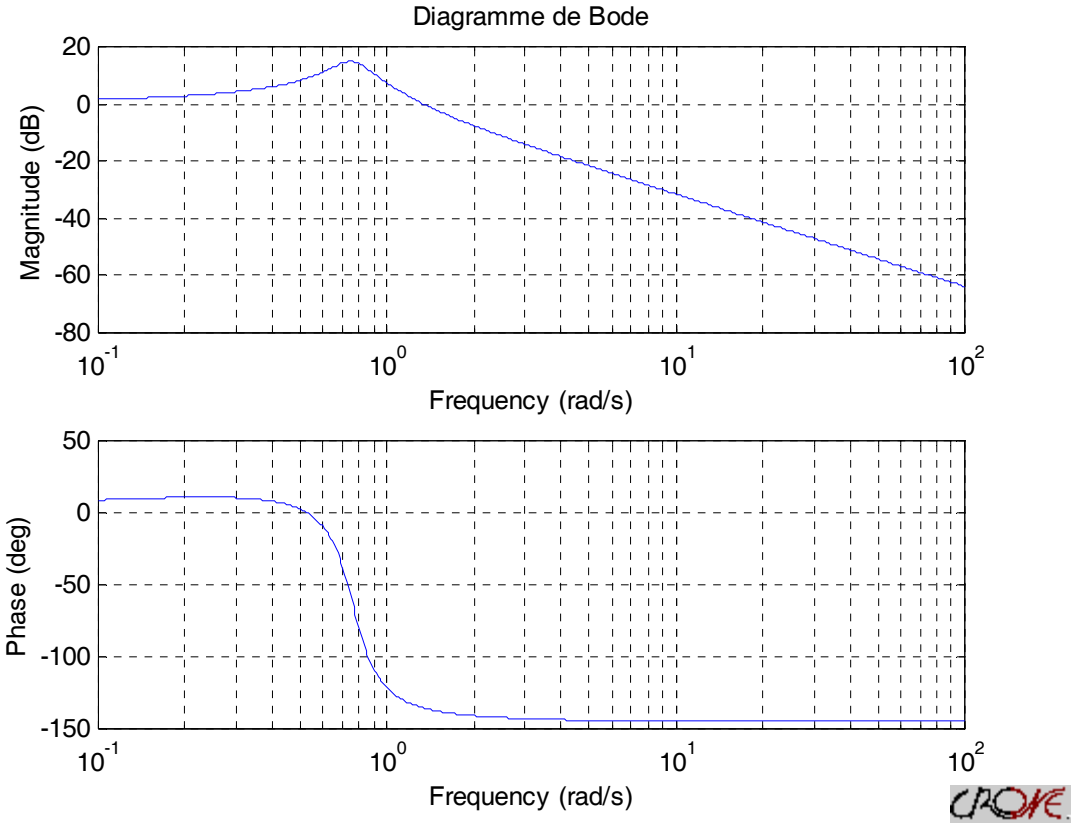
Example

```
» t
```

```
Transfer function:
```

$$\frac{s^{0.6}+1}{s^{2.2}+\hat{}\, s^{1,5}+1}$$

```
» bode(t)
```



char

Char Converts explicit fractionnal object to string.

Syntax

```
st = char(P)
```

Arguments

Argument in:
P: fractional object (frac_tf or frac_poly_exp or frac_poly_imp)

Argument out:
st: string

Example

```
» p=frac_poly_exp([1 2 3 2 -1],[0.5 0.2 6 3 0.2])
» st = char(p)
```
st='3 s^6 + 2 s^3 + s^0.5 + s^0.2'

clean(frac_poly_exp)

Sorts the orders of an explicit fractional polynomial in the descending order and removes the null coefficients.

Syntax

```
Q = clean(P)
```

Arguments

Argument in:
P: frac_poly_exp or frac_tf object

Argument out:
Q: frac_poly_exp object or frac_tf object

Example

```
» p=frac_poly_exp([1 2 3 2 -1],[0.5 0.2 6 3 0.2])
» c = clean(p)
```

$$c = 3.s^{6} + 2.s^{3} + s^{0,5} + s^{0,2}$$

coef(frac_poly_exp)

Quick access to explicit fractional polynomial coefficients.

Syntax

```
C = coef(P)
```

Arguments

Argument in :
P: frac_poly_exp object.

Argument out :
C: coefficients of P (vector)

Example

```
» p=frac_poly_exp([1 2 3 2 -1],[0.5 0.2 6 3 0.2])
```

$$p = 3.s^{6} + 2.s^{3} + s^{0,5} + s^{0,2}$$

```
» c=coef(p)

c =

     3     2     1     1
```

commensurate(frac_poly_exp)

Computes de step order of a fractional transfer function.

Syntax

```
[New_order,Step_order]= commensurate(T)
[New_order1, New_order2,Step_order]=
commensurate(T1,T2)
```

Arguments

Argument in:
T : frac_poly_exp object

Argument out:
New_order: the integer order of T
New_order1: the integer orders of T1
New_order2: the integer orders of T2
Step_order: the step order (scalar)

Example

```
» t=frac_poly_exp([1 1 1],[2.2 1.5 0])

Transfer function:
```

$s^{2.2} +\hat{} \; s^{1,5} + 1$

```
» [a,s]=commensurate(t)

a =

    22 15 0

S =
    0.1
```

den

Quick access to the denominator of a fractional transfer function.

Syntax

```
D = den(T)
```

Arguments

Argument in :
T : fractional transfer function (frac_tf object)

Argument out :
D : T denominator (frac_poly_imp object)

Example

```
» t=frac_tf([1 1],[0.6 0],[1 1 1],[2.2 1.5 0])

Transfer function:
```

$$\frac{s^{0.6}+1}{s^{2.2}+\hat{}\, s^{1,5}+1}$$

```
» d=den(t)
```

$$d = s^{2,2} + s^{1,5} + 1$$

dn (frac_poly_imp)

Computes the implicit derivative of order n of a data x, with n the order of the implicit fractional polynomial given in parameters to the function Dn.

Syntax

```
[Dy, Error]=dn(sys, x, time)
```

Arguments

Argument in:
sys: implicit fractional polynomial(frac_poly_imp object)
x: data (complex vector N*1)
time: time vector(vector of scalars N*1) or simple time (scalar)

Argument out :
Dy : data (complex matrix)
Error : error (string)

Example

```
» i=frac_poly_imp(1,0.5,1)
```

$i = (1+s)^{0,5}$

```
» x=(1:10)'
```

```
» t=(1:10)'
```

```
» dn(i,x,t)

ans =

  1.0e+005 *

    0.0000
    0.0001
    0.0005
    0.0019
    0.0062
    0.0202
    0.0636
    0.1965
    0.5986
    1.8022
```

dnh (frac_poly_imp)

Compute the implicit derivative of order n of a data x, with n the order of the implicit fractional polynomial given in parameters to the function Dn.

Syntax

```
[Dy, Error]=dnh(sys, x, time, level)
```

Arguments

Argument in:
sys: implicit fractional polynomial (frac_poly_imp object)
x: data (complex vector N*1)
time: time vector(vector of scalars N*1) or simple time (scalar)
level: approximation level (integer between -1 and 5) (1 by default)

Argument out :
Dy : data (complex matrix)
Error : error (string)

Example

```
» i=frac_poly_imp(1,0.5,1)
```

$i = (1+s)^{0,5}$

```
» x=(1:10)'

» t=(1:10)'
```

```
» dnh(i,x,t,4)

ans =

  1.0e+009 *

    0.0000
    0.0000
    0.0000
    0.0000
    0.0001
    0.0013
    0.0110
    0.0926
    0.7686
    6.2999
```

eig (frac_poly_exp)

Gives the eigenvalues of an explicit fractional polynomial and their orders.

Syntax

```
[eigen_value, eigen_order ]=eig(P)
```

Arguments

Argument in :
P : frac_poly_exp object
Argument out :
eigen_value : eigenvalues of P (cell array)
eigen_order : order of the eigenvalues of P(cell array)

Example

```
» p=frac_poly_exp([1 1 -1],[0.5 0.2 0])
```

$p = s^{0,5} + s^{0,2} - 1$

```
» eig(p)

ans =

   0.4649 + 1.0715i
   0.4649 - 1.0715i
  -0.8693 + 0.3883i
  -0.8693 - 0.3883i
   0.8087
```

eig (frac_tf)

Gives the eigenvalues of a fractional transfer function and their orders.

Syntax

```
[eigen_value,eigen_order]=eig(t)
```

Arguments

Argument in :
t : frac_tf object
Argument out :
eigen_value : eigenvalues of *Tf* (vector)
eigen_order : orders of the eigenvalues of *Tf* (vector)

Example

```
» t=frac_tf([1 1],[0.6 0],[1 1],[2.2 1.5])

Transfer function:
```

$$\frac{s^{0.6}+1}{s^{2.2}+s^{1,5}}$$

```
» eig(t)
ans =

     0
    -1
```

frac_poly_exp

Create an explicit fractional polynomial.

Syntax

```
Sys=frac_poly_exp (Coef, order)
```

Description

frac_poly_exp(coef) creates an explicit polynomial with fractional orders. Coefficient is specified by “coef” a scalar in this case. The resulting system is a frac_poly_exp object.
frac_poly_exp(coef, order) creates an explicit polynomial with fractional orders. Coefficients and orders of this polynomial are specified by the vectors “coef” and “order” given in parameters to the function. The resulting system is a frac_poly_exp object.

Arguments

Argument in :
coef : coefficients of the polynomial(row vector or cell array of row vector)
order: orders of the polynomial(row vector or cell array of row vector)

Argument out :
sys : explicit fractional polynomial (frac_poly_exp object)

Examples

$P = frac\_poly\_\exp([1,1,1],[2,0.2,0])$

Creates the following fractional polynomial:

$$P(s) = s^2 + s^{0.2} + 1$$

frac_poly_imp

Create an implicit fractional polynomial.

Syntax

```
sys = frac_poly_imp(fpe, implicit_order)
```

Description

Frac_poly_imp(*fpe, implicit_order*) creates an implicit fractional polynomial, i.e. a polynomial which is under the form $(fpe)^{implicit\_order}$. The resulting system is a frac_poly_imp object.

Arguments

Argument in :

fpe: first order explicit polynomial or explicit polynomial if the implicit order is set to 0 (frac_poly_exp object(size(1*M) or cell array of frac_poly_exp object(size(1*M))

implicit_order: fractional polynomial order (row vector or cell array of row vectors)

Argument out :

sys: implicit fractional polynomial (frac_poly_imp object)

Example

$P = s + 1$
$Q = frac\_poly\_imp(P, 0.3)$

Creates the following fractional polynomial:

$$P(s) = (1+s)^{0,3}$$

frac_tf

Create a fractional transfer function.

Syntax

```
Sys=frac_tf(P1,P2)
```

Description

We can also create a fractional transfer function by specifying in parameters of the function frac_tf the nominator and denominator which are explicit or implicit fractional polynomial (frac_poly_imp object).

Arguments

Argument in:
P1: numerator (frac_poly_imp or frac_poly_exp)
P2: denominator (frac_poly_imp or frac_poly_exp)

Argument out:
S*ys*: fractional transfer function. (Frac_tf object)

Example

$T1 = (s^{2.1} + 2.s + 1)^1$
$T2 = (s^{0.1} + 3)^1$
$T = frac\_tf(T1, T2)$
Creates the following fractional transfer function:

$$T(s) = \frac{s^{0.1} + 3}{s^{2.1} + 2.s + 1}$$

freqresp

Computes the frequency response of a fractional transfer function.

Syntax

```
G=freqresp(Tf,W)
```

Arguments

Argument in :
T*f* : fractional object (frac_tf or frac_poly_exp or frac_poly_imp)
W: frequency range (cell or vector)

Argument out :
G : frequency response

Example

```
» t

Transfer function:
```

$$\frac{s^{0.6}+1}{s^{2.2}+\hat{}\, s^{1,5}}$$

```
» freqresp(t,[1 100])

ans =

  -0.7946 - 0.6787i
  -0.0005 - 0.0004i
```

impulse

Impulse response of fractional transfer function.

Syntax

```
Rep=impulse(f,Time)
```

Arguments

Argument in :
f : fractional object (frac_tf or frac_poly_exp or frac_poly_imp)
Time : time vector(under the form Ti :Ts :Tf)

Argument out :
Rep : impulse response (Vector)

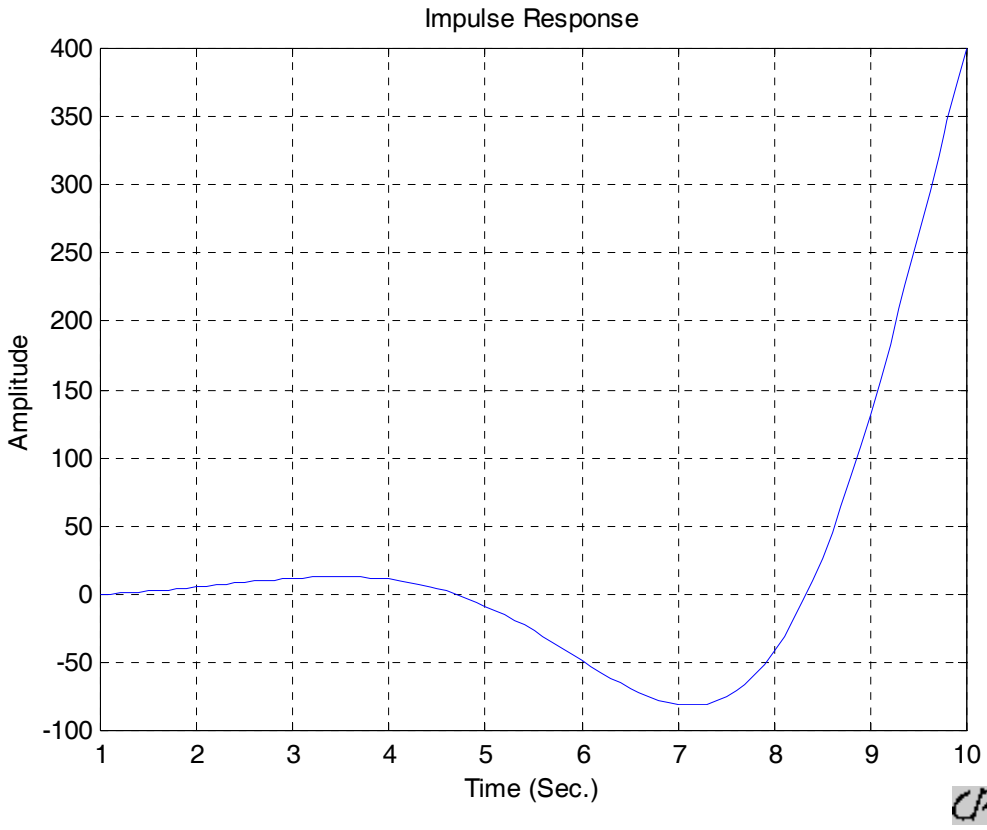
Example

```
» t
```

```
Transfer function:
```

$$\frac{s^{0.1}+1}{s^{2.1}-s\ +1}$$

```
» impulse(t,1:0.1:10)
```



lsim

Simulate implicit fractional polynomial response to arbitrary inputs.

Syntax

```
y=lsim(sys,u,time)
```

Arguments

Argument in :
sys : fractional object (frac_tf or frac_poly_exp or frac_poly_imp)
u : input signal (vector)
time : time vector (vector) or simple time(scalar)

Argument out :
y : output response (complex matrix)

Example

```
» i
```
$i = (1+s)^{0,5}$

```
» x=(1:10)'

» t=(1:10)'
```

```
» lsim(i,x,t)

ans =

    1.0000
    1.8161
    2.6152
    3.4112
    4.2066
    5.0017
    5.7968
    6.5918
    7.3869
    8.1820
```

nichols

Black frequency response of fractional transfer functions

Syntax

```
   nichols(Tf)
   nichols(Tf ,W)
[Mag ,Phase ,W]=nichols(Tf)
[Mag ,Phase]=nichols(Tf,W)
```

Description

Nichols computes the frequency response of fractional transfer functions and plots it in the Nichols coordinates.
Nichols(T*f*) produces a Nichols plot of the fractional transfer function T*f*. The frequency range is determined automatically based on the system poles and zeros.
Nichols(Tf, W) explicitly specifies the frequency range or frequency points to be used for the plot. To focus on a particular frequency interval [Wmin,Wmax], set w = {Wmin,Wmax}. To use particular frequency points, set w to the vector of desired frequencies.
When invoked with left-hand arguments,

```
[Mag ,Phase ,W]=nichols(Tf)
[Mag ,Phase]=nichols(Tf,W)
```

return the magnitude and phase of the frequency response at the frequency W.

Arguments

Argument in :
Tf : fractional object (frac_tf or frac_poly_exp or frac_poly_imp)
W : frequency range (vector or cell)

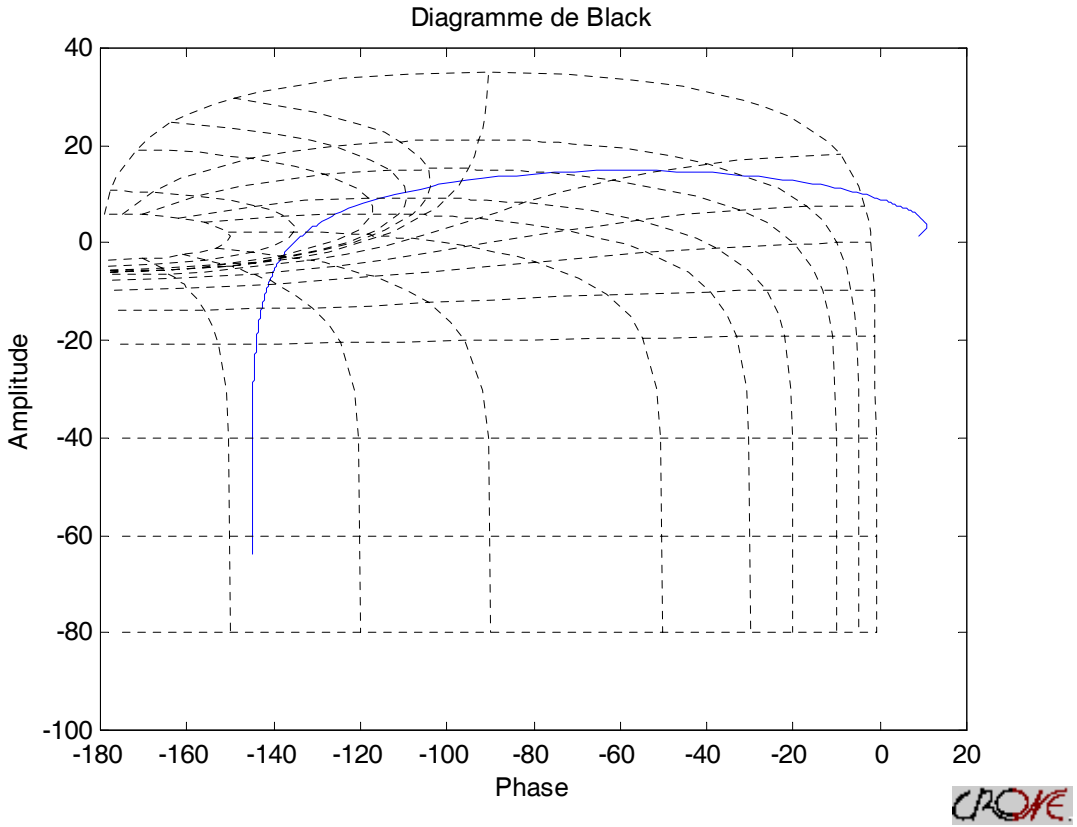
Argument out :
G : magnitude (vector)
Phase : phase (vector)
W : frequency range (vector)

Example

```
» t=frac_tf([1 1],[0.6 0],[1 1 1],[2.2 1.5 0])
Transfer function:
```

$$\frac{s^{0.6}+1}{s^{2.2}+\hat{}\, s^{1,5}+1}$$

```
» nichols(t)
```



norm

Calculate the norm of fractional transfer function.

Syntax

```
[N, error] = Norm(Sys)
```

Arguments

Argument in:
Sys : fractional transfer function (frac_tf object)

Argument out:
N: norm of *Sys*.
Error: Error (string)

Notice

If the fractional transfer function given in parameters to the function "norm" isn't stable, the result is "NaN".

Example

```
» t=frac_tf([1 1],[0.6 0],[1 1 1],[2.2 1.5 0])

Transfer function:
```

$$\frac{s^{0.6}+1}{s^{2.2}+\hat{}\,s^{1,5}+1}$$

```
» norm(t)

ans =

    1.8698
```

num

Quick access to fractional transfer function numerator.

Syntax

```
N = num(T)
```

Arguments

Argument in :
T : fractional transfer function (frac_tf object)

Argument out :
N : numerator of *T* (frac_poly_imp object)

Example

```
» t
Transfer function:
```

$$\frac{s^{0.6}+1}{s^{2.2}+\hat{}\, s^{1,5}+1}$$

```
» num(t)
```

$$\left(s^{0,6}+1\right)^{1}$$

nyquist

Nyquist frequency response of fractional transfer functions

Syntax

```
   nyquist(Tf)
   nyquist(Tf, W)
[Re, Im, W]=nyquist(Tf)
[Re, Im]=nyquist(Tf, W)
```

Description

Nyquist calculates the Nyquist frequency response of fractional transfer functions. When invoked without left-hand arguments, nyquist produces a Nyquist plot on the screen.
Nyquist(T*f*) plots the Nyquist response of the fractional transfer function T*f*. The frequency points are chosen automatically based on the system poles and zeros.
Nyquist(Tf, W) explicitly specifies the frequency range or frequency points to be used for the plot. To focus on a particular frequency interval [Wmin,Wmax], set w = {Wmin,Wmax}. To use particular frequency points, set w to the vector of desired frequencies.
When invoked with left-hand arguments,

```
[Re, Im,W]=nyquist(Tf)
[Re, Im]=nyquist(Tf,W)
```

return the magnitude and phase of the frequency response at the frequency W.

Arguments

Argument in :
Tf : fractional object (frac_tf or frac_poly_exp or frac_poly_imp)
W : frequency range (vector or cell)

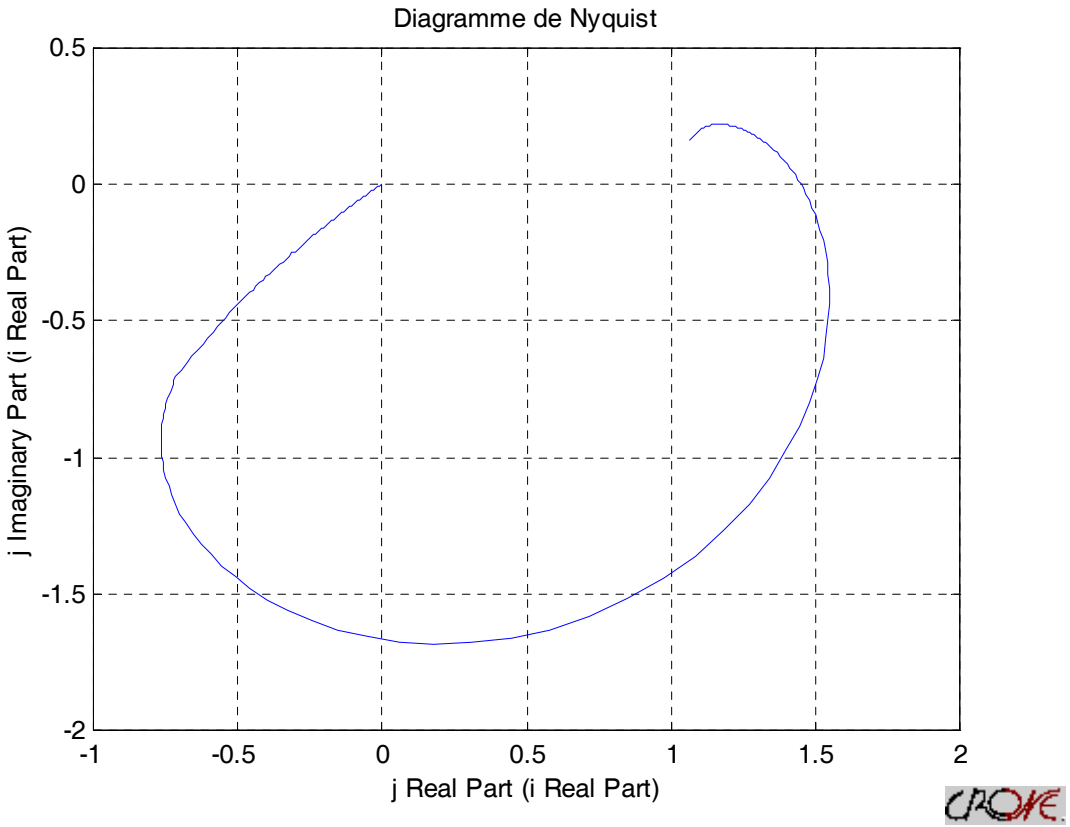
Argument out :
G : magnitude (vector)
Phase : phase (vector)
W : frequency range (vector)

Example

```
» t
Transfer function:
```

$$\frac{s^{0.6}+1}{s^{2.2}+\hat{}\, s^{1,5}+1}$$

```
» nyquist(t)
```



poles

Compute the poles of a fractional transfer function.

Syntax

```
[Poles,eigen_valu,eigen_order]=poles(T,Phi)
```

Arguments

Argument in :

T: fractional transfer function (frac_tf object)
Phi: position of the cut (scalar between 90 and 270)

Argument out :
Poles : poles of T(cells)
eigen_value: eigenvalues of T (vector)
eigen_order: orders of the eigenvalues of *T* (vector)
Error: error (string)

Example

```
» t

Transfer function:
```

$$\frac{s^{0.6}+1}{s^{2.2}+\hat{}\, s^{1,5}+1}$$

```
» poles(t)

ans =

    [-0.1236+ 0.7518i]
    [-0.1236- 0.7518i]
```

roots

Compute the roots of an explicit fractional polynomial.

Syntax

```
[roots,eigen_value,order_step]=roots(p)
```

Arguments

Argument in :
P : frac_poly_exp object.

Argument out :

roots	: roots of P (cells)
order_step	: step order (complex vector)
eigen_value	: eigenvalues of *P*(complex vector)

Example

```
»p=frac_poly_exp([1 1 -1],[2.5 0.2 0])
```
$s^{2,5} + s^{0.2} + 1$

```
»roots(p)
     [-0.3106+ 0.6169i]
    [-0.3106- 0.6169i]
    [0.4603]
```

scalar

Computes the scalar product of two fractional transfer functions.

Syntax

```
[C, error] = scalar(SYS1,SYS2)
```

Arguments

Argument in :
SYS1, SYS2: fractional transfer functions(frac_tf objects)

Argument out :
C: scalar product of sys1 and sys2.
Error: error (string)

Example

```
» t=frac_tf([1 1],[0.6 0],[1 1 1],[2.2 1.5 0])

Transfer function:
```

$$\frac{s^{0.6}+1}{s^{2.2}+\hat{}\, s^{1,5}+1}$$

```
» s=frac_tf([1],[0],[1 -1 1],[0.3 0.1 0])

Transfer function:
```

$$\frac{1}{s^{0,3}-s^{0,1}+1}$$

```
» scalar(s,t)

ans =
    0.5038
```

sort

Sort the orders of an explicit fractional polynomial in the descending order

Syntax

```
Q = sort(P)
```

Arguments

Argument in:
P: frac_poly_exp object

Argument out:
Q: frac_poly_exp object

Example

```
»p=frac_poly_exp([1 2 3 2 -1],[0.5 0.2 6 3 0.2])
»c=sort(p)
```

$$c = 3.s^{6} + 2.s^{3} + s^{0,5} + s^{0,2}$$

step

Step response of fractional transfer function.

Syntax

```
[Rep,Error]=step(sys,Time)
```

Arguments

Argument in :
Sys : frac_tf object.
Time : time vector (under the form Ti :Ts :Tf)

Argument out :
Rep : step response (Vector)
Error : error (string).

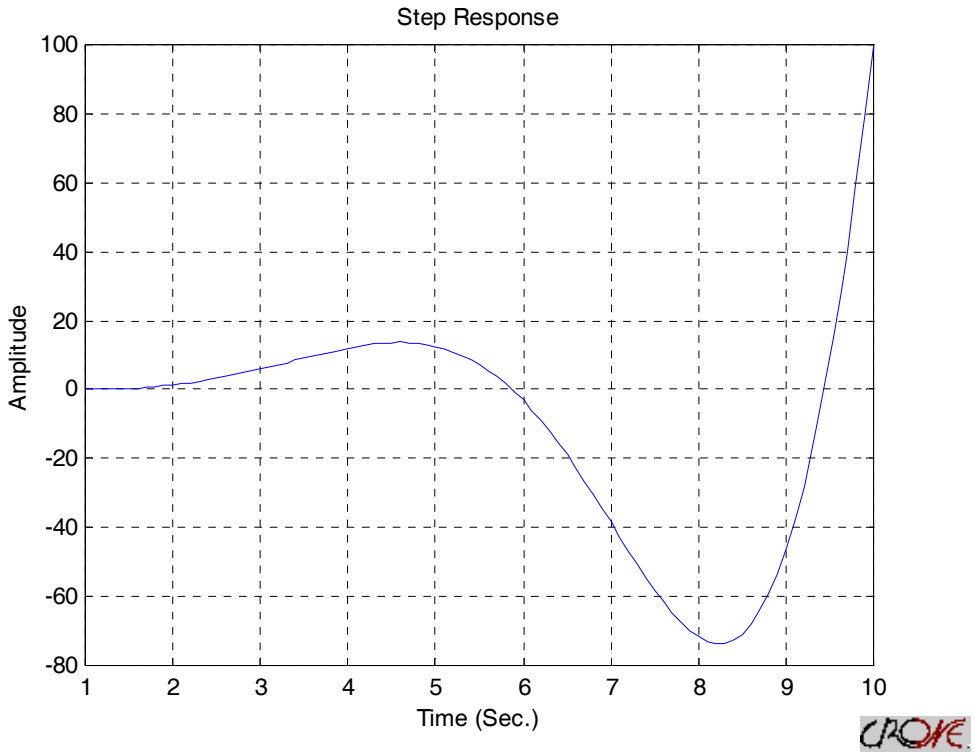
Example

```
» t=frac_tf([1 1],[0.1 0],[1 -1 1],[2.1 1 0])

Transfer function:
```

$$\frac{s^{0.1}+1}{s^{2.1}-s\ \ +1}$$

```
» step(t,1:0.1:10)
```


Step Response
Amplitude
Time (Sec.)
100
80
60
40
20
0
-20
-40
-60
-80
1
2
3
4
5
6
7
8
9
10

uncommensurate

Computes de fractional transfer from a LTI system.

Syntax

```
[New_tf]= uncommensurate(frac_tf, step_order)
```

Arguments

Argument in:
Frac_tf : fractional transfer function(frac_tf object)
Step_order : the step order (scalar)

Argument out:
New_tf: fractional transfer function (frac_tf object)

Example

```
» sys
```

```
Transfer function:
```

$$\frac{s^6+1}{s^{10}+s^5+1}$$

```
» t=uncommensurate(sys,0.1)
```

```
Transfer function:
```

$$\frac{s^{0.6}+1}{s\ +s^{0,5}+1}$$