Contents

Chapter 11

Analysis and Design of Fractional-order Systems

Fractional-order system theory is an active field of research over the last two decades. In recent years, many progresses have been made in both fractional-order calculus and fractional-order control. The research on fractional-order control has both theoretical value and practical perspectives.

The so-called "fractional-order systems" really mean that the order of the systems are no longer integers, and this is different from all discussed in the book so far. We all know that $\mathrm{d}^n y/\mathrm{d}t^n$ represents the nth order derivative of y with respect to t. What happens if $n = 1/2$? This was the question a famous French mathematician Guillaume François Antoine L'Hôpital asked one of the inventors of calculus, Gottfried Wilhelm Leibniz[1–3], 300 years ago. From that time on, researchers began to study fractional-order calculus problems. Thus, fractional-order calculus is a 300 year-old topic, however, earlier work focused on pure mathematics. In the 19th century, various definitions on fractional-order calculus appeared, and it was not until 1960, the first publication on non-integer order integrator appeared in the field of control[4], however, few attention to the topics were received. In late 1990's, fractional-order PID controller appeared[5–7].

The operator $\mathscr{D}^\alpha$ is used in this book to describe fractional-order differentiation and integration. When $\alpha > 0$, αth order derivative is used, while $\alpha < 0$ means $-\alpha$th order integration, and $\alpha = 0$ means the original function. This unified notation will be used throughout the book.

Strictly speaking, "fractional-order" is a misused term. The precise one should be "non-integer-order", since the order can either be fractional, or an irrational, or even complex numbers. For instance, $\mathrm{d}^{\sqrt{2}} y/\mathrm{d}t^{\sqrt{2}}$ means the $\sqrt{2}$th order derivative. However, the word "fractional-order" was used for a very time among the researchers. In this book, "fractional-order" is used, however, it actually means "non-integer-order".

In Section 11.1, various definitions on fractional-order calculus and their MATLAB implementations are presented. Properties of fractional-order calculus are also presented. Mittag–Leffler functions and commonly used Laplace transforms in fractional-order functions are also presented. In Section 11.2, numerical computations of fractional-order derivatives are presented. Also Mittag–Leffler functions and commonly used Laplace transforms in fractional-order functions are also presented. Section 11.3 presents the numerical and analytical solutions of linear fractional-order differential equations. The fractional-order transfer function is used as an example to demonstrate object-oriented programming in Section 11.4, where the methods of class creation, overload function design are all illustrated. With the programming technique, linear fractional-order systems analysis — stability, time and frequency domain analysis are presented. In Section 11.5,

integer-order approximation to fractional-order operators and fractional-order systems are presented, and sub-optimal reduction technique are proposed. In Section 11.6, block diagram-based nonlinear fractional-order system simulation techniques are illustrated. In Section 11.7, optimal design methods and interface of fractional-order PID controllers for fractional-order plants are presented.

11.1 Definitions and Numerical Computations in Fractional-order Calculus

In the development of fractional-order calculus, various definitions were proposed and applied. The most widely used Cauchy integral formula, Grünwald–Letnikov definition, Riemann–Liouville definition and Caputo definition are all extended directly from integral-order calculus. In this section, the definitions and their relationships are given, then the numerical computation and properties are presented.

11.1.1 *Definitions of Fractional-order Calculus*

1. Fractional Cauchy integral formula

The formula is extended directly from its integer-order counterpart

$$\mathscr{D}_t^{\gamma} f(t) = \frac{\Gamma(\gamma+1)}{2\pi \mathrm{j}} \int_{\mathrm{C}} \frac{f(\tau)}{(\tau - t)^{\gamma+1}} \mathrm{d}\tau, \tag{11.1}$$

where C is the closed-path that encircles all the poles of the function $f(t)$, and $\mathscr{D}_t$ is the fractional-order differentiation operator.

2. Grünwald–Letnikov definition

The definition of fractional-order differentiation is

$${}_{t_0}\mathscr{D}_t^{\alpha} f(t) = \lim_{h \to 0} \frac{1}{h^{\alpha}} \sum_{j=0}^{[(t-t_0)/h]} (-1)^j \binom{\alpha}{j} f(t - jh), \tag{11.2}$$

where $w_j^{(\alpha)} = (-1)^j \binom{\alpha}{j}$ is the binomial coefficients of $(1-z)^{\alpha}$, and it can be obtained iteratively from

$$w_0^{(\alpha)} = 1, \ w_j^{(\alpha)} = \left(1 - \frac{\alpha+1}{j}\right) w_{j-1}^{(\alpha)}, \ j = 1, 2, \cdots. \tag{11.3}$$

The following formula can be used in calculating fractional-order derivative

$${}_{t_0}\mathscr{D}_t^{\alpha} f(t) \approx \frac{1}{h^{\alpha}} \sum_{j=0}^{[(t-t_0)/h]} w_j^{(\alpha)} f(t - jh). \tag{11.4}$$

If step size h is small enough, Eqn. (11.4) can be used to directly evaluate the approximate values of fractional-order differentiation. It can be shown that[2] the accuracy of the definition is $\mathrm{o}(h)$.

3. Riemann–Liouville definition

Fractional-order integral is defined as

$$ {}_{t_0}\mathscr{D}_t^{-\alpha} f(t) = \frac{1}{\Gamma(\alpha)} \int_{t_0}^{t} \frac{f(\tau)}{(t-\tau)^{1-\alpha}} \mathrm{d}\tau, \tag{11.5} $$

where $0 < \alpha < 1$, and t_0 is the initial instance. Normally $t_0 = 0$, and the notation is simplified to $\mathscr{D}_t^{-\alpha} f(t)$. Riemann–Liouville definition is the most widely used definition in fractional-order calculus. Especially the subscripts on both sides of $\mathscr{D}$ are the lower and upper bounds in the integral[8].

Fractional-order differentiation can also be defined. Assume that the order α of differentiation satisfies $n - 1 < \beta \leqslant n$, the differentiation is defined as

$$ {}_{t_0}\mathscr{D}_t^{\beta} f(t) = \frac{\mathrm{d}^n}{\mathrm{d}t^n} \left[{}_{t_0}\mathscr{D}_t^{-(n-\beta)} f(t) \right] = \frac{1}{\Gamma(n-\beta)} \frac{\mathrm{d}^n}{\mathrm{d}t^n} \left[\int_{t_0}^{t} \frac{f(\tau)}{(t-\tau)^{\beta-n+1}} \mathrm{d}\tau \right]. \tag{11.6} $$

4. Caputo definition

The definition of Caputo fractional-order differentiation is

$$ {}_{t_0}\mathscr{D}_t^{\alpha} f(t) = \frac{1}{\Gamma(1-\gamma)} \int_{t_0}^{t} \frac{f^{(m+1)}(\tau)}{(t-\tau)^{\gamma}} \mathrm{d}\tau, \tag{11.7} $$

where $\alpha = m + \gamma$, or $m = \lfloor \alpha \rfloor$ is an integer such that $0 < \gamma < 1$.

Similarly, the Caputo fractional-order integral is defined as

$$ {}_{t_0}\mathscr{D}_t^{-\alpha} f(t) = \frac{1}{\Gamma(\alpha)} \int_{t_0}^{t} \frac{f(\tau)}{(t-\tau)^{1-\alpha}} \mathrm{d}\tau, \quad \alpha > 0, \tag{11.8} $$

and it is exactly the same as Riemann–Liouville definition in Eqn. (11.5).

11.1.2 *The Relationship of Different Definitions*

It can be shown that for a wide class of practical functions, the Grünwald–Letnikov and Riemann–Liouville definitions are equivalent[2]. In this book, we shall not distinguish them.

The major differences of Caputo definition and Riemann–Liouville definition are that the former considers the nonzero initial condition problems. Thus, Caputo definition is more suitable to deal with fractional-order differential equations with nonzero initial conditions.

If the initial value of function $f(t)$ is nonzero, and $\alpha \in (0, 1)$, compared with the definitions of Caputo's and Riemann–Liouville's, it can be seen that

$$ {}_{t_0}^{\mathrm{C}}\mathscr{D}_t^{\alpha} f(t) = {}_{t_0}^{\mathrm{RL}}\mathscr{D}_t^{\alpha} (f(t) - f(t_0)), \tag{11.9} $$

where the fractional-order derivative of constant $f(t_0)$ is

$$ {}_{t_0}^{\mathrm{RL}}\mathscr{D}_t^{\alpha} f(t_0) = \frac{f(t_0)(t-t_0)^{-\alpha}}{\Gamma(1-\alpha)}. \tag{11.10} $$

The relationship of these two definitions are

$$ {}_{t_0}^{\mathrm{C}}\mathscr{D}_t^{\alpha} f(t) = {}_{t_0}^{\mathrm{RL}}\mathscr{D}_t^{\alpha} f(t) - \frac{f(t_0)(t-t_0)^{-\alpha}}{\Gamma(1-\alpha)}. \tag{11.11} $$

More generally, if $\alpha > 1$, denote $m = \lceil \alpha \rceil$, then

$$ {}_{t_0}^{\mathrm{C}}\mathscr{D}_t^{\alpha} f(t) = {}_{t_0}^{\mathrm{RL}}\mathscr{D}_t^{\alpha} f(t) - \sum_{k=0}^{m-1} \frac{f^{(k)}(t_0)}{\Gamma(k-\alpha+1)}(t-t_0)^{k-\alpha}, \tag{11.12} $$

and the relationship shown for $0 < \alpha < 1$ is just a special case of the formula.

11.1.3 *Properties of Fractional-order Calculus*

Some of the important properties of fractional-order calculus are summarized below without proofs[9]

(1) The fractional-order differentiation ${}_{t_0}\mathscr{D}_t^{\alpha} f(t)$ of an analytic function $f(t)$ with respect to t is also analytic.

(2) If $\alpha = n$, the fractional-order derivative is identical to integer-order derivative, and also ${}_{t_0}\mathscr{D}_t^{0} f(t) = f(t)$.

(3) The fractional-order differentiation is linear, i.e., for any constants c, d

$$ {}_{t_0}\mathscr{D}_t^{\alpha}\Big[cf(t) + dg(t)\Big] = c\ {}_{t_0}\mathscr{D}_t^{\alpha} f(t) + d\ {}_{t_0}\mathscr{D}_t^{\alpha} g(t). \tag{11.13} $$

(4) Fractional-order differentiation operators satisfy commutative-law, i.e.,

$$ {}_{t_0}\mathscr{D}_t^{\alpha}\Big[{}_{t_0}\mathscr{D}_t^{\beta} f(t)\Big] = {}_{t_0}\mathscr{D}_t^{\beta}\Big[{}_{t_0}\mathscr{D}_t^{\alpha} f(t)\Big] = {}_{t_0}\mathscr{D}_t^{\alpha+\beta} f(t). \tag{11.14} $$

(5) The Laplace transform of the fractional-order derivative is

$$ \mathscr{L}\Big[{}_{t_0}\mathscr{D}_t^{\alpha} f(t)\Big] = s^{\alpha}\mathscr{L}\big[f(t)\big] - \sum_{k=1}^{n-1} s^{k}\Big[{}_{t_0}\mathscr{D}_t^{\alpha-k-1} f(t)\Big]_{t=t_0}. \tag{11.15} $$

Especially, if the initial values of the function $f(t)$ and its derivatives at $t = t_0$ are all zero, $\mathscr{L}\Big[{}_{t_0}\mathscr{D}_t^{\alpha} f(t)\Big] = s^{\alpha}\mathscr{L}\big[f(t)\big]$. It is the same as the one in integer-order calculus, and it is the basis of fractional-order transfer functions, for the analysis and design of fractional-order systems.

11.2 Numerical Computations in Fractional-order Calculus

In this section, numerical computation of Grünwald–Letnikov and Caputo derivatives of given function $f(t)$ are presented. Also, the computations of the important functions in fractional-order calculus, the Mittag–Leffler functions are presented.

11.2.1 *Numerical Solutions with Grünwald–Letnikov Definition*

With Grünwald–Letnikov definition, the following MATLAB function can be immediately written to evaluate the fractional-order differentiations[10], which implements directly Eqn. (11.4).

```
function dy=glfdiff(y,t,gam)
h=t(2)-t(1); w=1; y=y(:); t=t(:); a0=y(1); dy(1)=0;
if a0~=0 & gam>0, dy(1)=sign(a0)*inf; end
for j=2:length(t), w(j)=w(j-1)*(1-(gam+1)/(j-1)); end
for i=2:length(t), dy(i)=w(1:i)*[y(i:-1:1)]/h^gam; end
```

The syntax of the function is $\boldsymbol{y}_1 = \texttt{glfdiff}(\boldsymbol{y},\boldsymbol{t},\gamma)$, where $\boldsymbol{y}$ and $\boldsymbol{t}$ are the vectors composed of sample values of the given function and time, respectively. The vector $\boldsymbol{t}$ should be evenly spaced. The argument γ is the order. The γth order derivative of $\boldsymbol{y}$ is returned in $\boldsymbol{y}_1$ vector. If $y(t_0) \neq 0$, the first value in the derivative is set to infinity.

Example 11.1. In integer-order calculus, it is known that the derivative function of a step function is an impulse function, and first-order integral is straight line. The following statements can be used to calculate the 0.5th order derivative and integral, and the results are shown in Fig. 11-1.

```
>> t=0:0.01:5; u=ones(size(t));
   y1=glfdiff(u,t,0.5); y2=glfdiff(u,t,-0.5);
   plot(t,y1,'-',t,y2,'--')
```

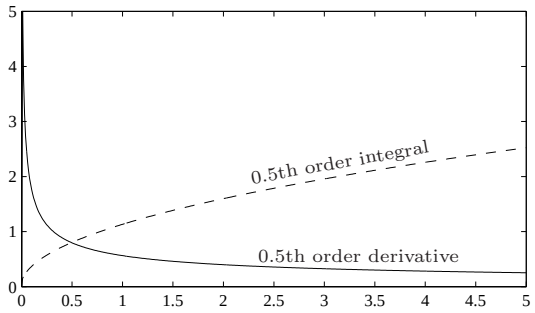


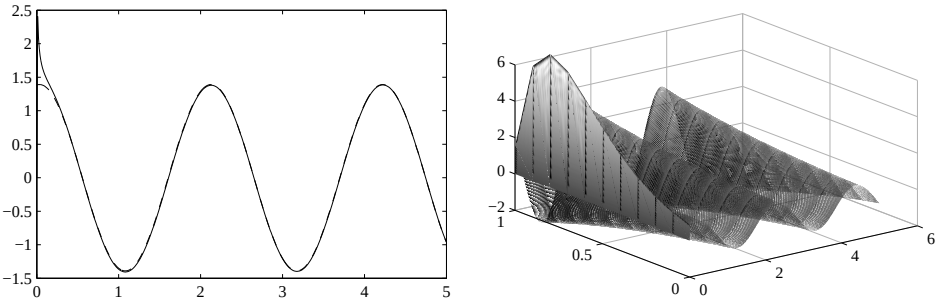
Fig. 11-1 Fractional derivative and integral of a step function.

It can be seen that the fractional integrals are no longer straight lines, and the derivatives are no longer impulse functions. Gradual changes are observed in fractional-order integrals and derivatives. The fractional-order calculus is regarded as having memories.

Example 11.2. Consider the sinusoidal function $f(t) = \sin(3t + 1)$. Its 0.3th order derivative is shown in Fig. 11-2(a), and the surface plots for the derivatives of different orders are shown in Fig. 11-2(b).

```
>> t=0:0.01:5; u=sin(3*t+1); ww=0:0.1:1; Y=[];
   y1=glfdiff(u,t,0.3); y2=3^0.3*sin(3*t+1+0.3*pi/2);
```

```
plot(t,y1,'-',t,y2,'--'), figure
for w=ww, Y=[Y; glfdiff(u,t,w)]; end, surf(t,ww,Y)
```



(a) 0.3th order derivative (b) surface plots for different orders

Fig. 11-2 Fractional-order differentiation of sinusoidal function $f(t) = \sin(3t + 1)$.

The αth order derivative under Cauchy integral formula is $3^\alpha \sin(3t + 1 + \alpha\pi/2)$. This derivative curve is also given in Fig. 11-2(a). It can be seen that the major difference between the two definitions is, in Grünwald–Letnikov definition, the initial values of $f(t)$ is assumed to be zero for $t \leqslant 0$. Thus, the function jumped from 0 to $\sin 1$ at time $t = 0_+$. While in the Cauchy formula, the function at $t \leqslant 0$ time is still assumed to be $\sin(3t + 1)$, thus there is no jump at $t = 0_+$.

It can be seen from the fractional-order derivatives of sinusoidal wave in Fig. 11-2(b) that gradual changes between sinusoidal and cosine waves are obtained, while in integer-order calculus, only sinusoidal and cosine waves are obtained. Thus, fractional-order calculus are more informative than the traditional integer-order calculus.

11.2.2 *Numerical Solutions with Caputo Definition*

From the relationships of the definitions, the following function can be used to calculate Caputo derivatives, through Eqn. (11.12).

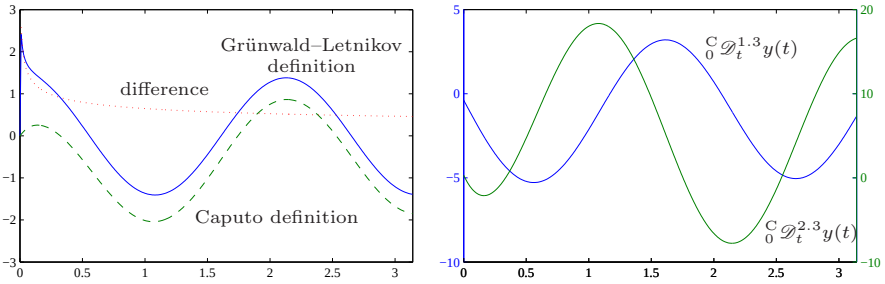
```
function dy=caputo(y,t,gam,L,vec)
t0=t(1); dy=glfdiff(y,t,gam); if nargin<=3, L=10; end
if gam>0, m=ceil(gam); if gam<=1, vec=y(1); end
   for k=0:m-1, dy=dy-vec(k+1)*(t-t0).^(k-gam)./gamma(k+1-gam); end
   yy1=interp1(t(L+1:end),dy(L+1:end),t(2:L),'spline'); dy(2:L)=yy1;
end
```

The syntax of the function is $\boldsymbol{y}_1 =$ `caputo(`$\boldsymbol{y}$`,`$\boldsymbol{t}$`,`α`,`$\boldsymbol{y}_0$`,`L`)`, where if $\alpha \leqslant 0$, the results of Grünwald–Letnikov are returned directly; If $\alpha < 1$, the initial value of $y(t_0)$ is directly extracted from vector $\boldsymbol{y}$. If $\alpha > 1$, $\boldsymbol{y}_0 = [y(t_0), y'(t_0), \cdots, y^{(m-1)}(t_0)]$ should be provided, where $m = \lceil \alpha \rceil$. In numerical implementation, it is always found that the first few terms may have large errors, thus it is better to reconstruct the first L terms with interpolation approaches. The default value of L is 10, however, if the order of differentiation is high, L should be increased.

Example 11.3. Consider the function $f(t) = \sin(3t + 1)$ studied in Example 11.2.

It can be seen that at $t = 0$, the initial value of the function is $f(0) = \sin 1$. The difference between the two definitions is $d(t) = t^{-0.3} \sin 1/\Gamma(0.7)$. The following statements can be used to find the 0.3th order derivatives using Grünwald–Letnikov definition and Caputo definition, as shown in Fig. 11-3(a). The difference is also shown.

```
>> t=0:0.01:pi; y=sin(3*t+1); d=t.^(-0.3)*sin(1)/gamma(0.7);
   y1=glfdiff(y,t,0.3); y2=caputo(y,t,0.3); plot(t,y1,t,y2,'--',t,d,':')
```



(a) 0.3th order derivative (b) 1.3th and 2.3th order derivatives

Fig. 11-3 Differentiation with Caputo definition.

Since ${}_0^{\mathrm{C}}\mathscr{D}_t^{2.3}y(t)$ is to be calculated, the initial values $y'(0), y''(0)$ are needed. These values can be obtained with symbolic computation and convert the result back to double precision variables. The 1.3th and 2.3th order derivatives can also be obtained as shown in Fig. 11-3(b).

```
>> syms t; y=sin(3*t+1); y00=sin(1); y10=double(subs(diff(y,t),t,0));
   y20=double(subs(diff(y,t,2),t,0)); t=0:0.01:pi; y=sin(3*t+1);
   y1=caputo(y,t,1.3,[y00 y10],15); y2=caputo(y,t,2.3,[y00,y10,y20],40);
   plotyy(t,y1,t,y2)
```

11.2.3 *Mittag–Leffler Functions and Their Computations*

We all know that the exponential function is very important in the solutions of integer-order systems. Like exponential function in integer-order systems, the so-called Mittag–Leffler can be regarded as an extension of exponential functions to fractional-order calculus.

1. Mittag–Leffler function with one parameter

The definition is

$$\mathscr{E}_\alpha(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k + 1)}, \tag{11.16}$$

where $\alpha \in \mathbb{C}$. The convergent condition for the infinite series is $\mathscr{R}(\alpha) > 0$.

It is obvious that exponential function e^z is a particular case of Mittag–Leffler function, with $\alpha = 1$.

$$\mathscr{E}_1(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(k+1)} = \sum_{k=0}^{\infty} \frac{z^k}{k!} = \mathrm{e}^z. \tag{11.17}$$

Other particular cases of Mittag–Leffler functions can be derived

$$\mathscr{E}_2(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(2k+1)} = \sum_{k=0}^{\infty} \frac{(\sqrt{z})^{2k}}{(2k)!} = \cosh\sqrt{z}, \tag{11.18}$$

$$\mathscr{E}_{1/2}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(k/2+1)} = \mathrm{e}^{z^2}(1+\mathrm{erf}(z)) = \mathrm{e}^{z^2}\mathrm{erfc}(-z). \tag{11.19}$$

2. Mittag–Leffler function with two parameters

Two-parameter Mittag–Leffler function can be defined when 1 in the Γ function of the one-parameter Mittag–Leffler function is substituted by a free variable β

$$\mathscr{E}_{\alpha,\beta}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k+\beta)}, \tag{11.20}$$

where $\alpha, \beta \in \mathbb{C}$, and the convergent conditions for $z \in \mathbb{C}$ are $\mathscr{R}(\alpha) > 0$ and $\mathscr{R}(\beta) > 0$. If $\beta = 1$, the two-parameter Mittag–Leffler function is changed to one-parameter Mittag–Leffler function, i.e.,

$$\mathscr{E}_{\alpha,1}(z) = \mathscr{E}_{\alpha}(z). \tag{11.21}$$

Besides, three- and four-parameter generalized Mittag–Leffler functions can also be defined[11]. A MATLAB function `ml_func()` for the computation of generalized Mittag–Leffler functions and their integer-order derivatives[5].

```
function f=ml_func(aa,z,n,eps0)
aa=[aa,1,1,1]; a=aa(1); b=aa(2); c=aa(3); q=aa(4);
f=0; k=0; fa=1; if nargin<4, eps0=eps; end
if nargin<3, n=0; end
if n==0
   while norm(fa,1)>=eps0
      fa=gamma(k*q+c)/gamma(c)/gamma(k+1)/gamma(a*k+b)*z.^k;
      f=f+fa; k=k+1;
   end
   if ~isfinite(f(1))
      if c==1 & q==1
         f=mlf(a,b,z,round(-log10(eps0))); f=reshape(f,size(z));
      else, error('Error: truncation method failed'); end, end
else
   aa(2)=aa(2)+n*aa(1); aa(3)=aa(3)+aa(4)*n;
   f=gamma(q*n+c)/gamma(c)*ml_func(aa,z,0,eps0);
end
```

The syntax of the function is `y = ml_func(v,z,n,ϵ)`, where $\boldsymbol{z}$ is a vector, and the vector $\boldsymbol{v}$ can be set to $\boldsymbol{v} = \alpha$ or $\boldsymbol{v} = [\alpha, \beta]$, to indicate one- or two-parameter Mittag–Leffler functions. Vector $\boldsymbol{v}$ can also be assigned to a three or four element vector, $\boldsymbol{v} = [\alpha, \beta, \gamma]$, or $\boldsymbol{v} = [\alpha, \beta, \gamma, c]$, to indicate three- or four-parameter Mittag–Leffler functions. The argument n is the order of derivative for the Mittag–Leffler function. The argument ϵ is the error tolerance. Since truncation algorithm is used in the function, the speed of the function is relatively fast, however, the values may sometimes become infinite. In this

case, the `mlf()` function[12] is embedded to tackle the problem. However, the speed may become very slow. The output vector $\boldsymbol{y}$ is the Mittag–Leffler function.

Example 11.4. The following statements can be used to draw the Mittag–Leffler function $\mathscr{E}_1(-t)$, $\mathscr{E}_{3/2,3/2}(-t)$, and $\mathscr{E}_{1,2}(-t)$, as shown in Fig. 11-4. The exponential function is also drawn. It can be seen that $\mathscr{E}_1(-t)$ is the same as e^{-t}, and the decay rates of the other two curves are slower than exponential function.

```
>> t=0:0.1:5; y1=ml_func(1,-t); y2=ml_func([1,2],-t);
   y3=ml_func([3/2,3/2],-t); plot(t,y1,t,y2,t,y3)
```

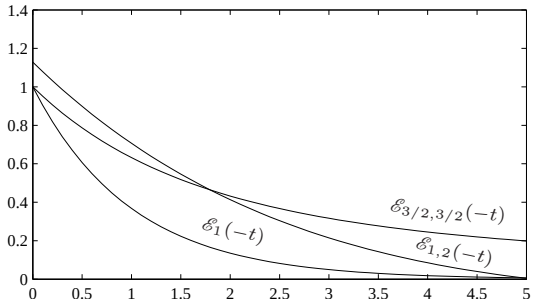


Fig. 11-4 Mittag–Leffler function curves.

11.3 Solutions of Linear Fractional-order Systems

Fractional-order systems are the direct extension of integer-order systems. In practical applications, there are some systems which can only be expressed accurately with fractional-order differential equations. In this section, analytical solutions for some linear fractional-order differential equations are first presented, and Mittag–Leffler functions are normally used to express the analytical solutions. For more linear fractional-order differential equation, closed-form numerical solution algorithms are presented.

11.3.1 *Numerical Solutions of Linear Fractional-order Differential Equations*

The typical form of linear fractional-order differential equation is given by

$$\begin{aligned} a_1\,{}_{t_0}\mathscr{D}^{\eta_1}y(t) + a_2\,{}_{t_0}\mathscr{D}^{\eta_2}y(t) + \cdots + a_{n-1}\,{}_{t_0}\mathscr{D}^{\eta_{n-1}}y(t) + a_n\,{}_{t_0}\mathscr{D}^{\eta_n}y(t) \\ = b_1\,{}_{t_0}\mathscr{D}^{\gamma_1}u(t) + b_2\,{}_{t_0}\mathscr{D}^{\gamma_2}u(t) + \cdots + b_m\,{}_{t_0}\mathscr{D}^{\gamma_m}u(t), \end{aligned} \tag{11.22}$$

where b_i and a_i are real coefficients, and γ_i and η_i are orders.

Let us consider a simpler differential equation

$$a_1\,{}_{t_0}\mathscr{D}_t^{\eta_1}y(t) + a_2\,{}_{t_0}\mathscr{D}_t^{\eta_2}y(t) + \cdots + a_{n-1}\,{}_{t_0}\mathscr{D}_t^{\eta_{n-1}}y(t) + a_n\mathscr{D}_t^{\eta_n}y(t) = u(t), \tag{11.23}$$

where $u(t)$ is a given function. The Grünwald–Letnikov definition given in Eqn. (11.4) can be used directly, and the closed-form solution to the original differential equation is written as

$$ {}_{t_0}\mathscr{D}_t^{\eta_i} y(t) \approx \frac{1}{h^{\eta_i}} \sum_{j=0}^{[(t-t_0)/h]} w_j^{(\eta_i)} y_{t-jh} = \frac{1}{h^{\eta_i}} \left[y_t + \sum_{j=1}^{[(t-t_0)/h]} w_j^{(\eta_i)} y_{t-jh} \right], \tag{11.24} $$

where $w_0^{(\beta_i)}$ can be evaluated recursively from Eqn. (11.3). Substitute it back to the original equation, the closed-form solution of the differential equation can be written as[10]

$$ y_t = \frac{1}{\sum_{i=1}^{n} \frac{a_i}{h^{\eta_i}}} \left[u_t - \sum_{i=1}^{n} \frac{a_i}{h^{\eta_i}} \sum_{j=1}^{[(t-t_0)/h]} w_j^{(\eta_i)} y_{t-jh} \right]. \tag{11.25} $$

For the differential equation in Eqn. (11.22), the equivalent input $u(t)$ can be calculated first, then Eqn. (11.25) can be used to solve the numerical solution of the original equation. The following MATLAB function can be written

```
function y=fode_sol(a,na,b,nb,u,t)
h=t(2)-t(1); D=sum(a./[h.^na]); nT=length(t);
vec=[na nb]; W=[]; D1=b(:)./h.^nb(:); nA=length(a);
y1=zeros(nT,1); W=ones(nT,length(vec));
for j=2:nT, W(j,:)=W(j-1,:).*(1-(vec+1)/(j-1)); end
for i=2:nT,
   A=[y1(i-1:-1:1)]'*W(2:i,1:nA);
   y1(i)=(u(i)-sum(A.*a./[h.^na]))/D;
end
for i=2:nT, y(i)=(W(1:i,nA+1:end)*D1)'*[y1(i:-1:1)]; end
```

The syntax of the function is `y=fode_sol(a,η,b,γ,u,t)`, where time vector $\boldsymbol{t}$ and input vector $\boldsymbol{u}$ should be given, and $\boldsymbol{y}$ returns the numerical solution, vectors $\boldsymbol{a}$ and $\boldsymbol{\eta}$ are the coefficients and orders of output $y(t)$ in the equation, while $\boldsymbol{b}$ and $\boldsymbol{\gamma}$ are the coefficients and orders of input signal $u(t)$.

Example 11.5. Consider the linear fractional-order differential equation

$$ \mathscr{D}^{1.6}y(t)+10\mathscr{D}^{1.2}y(t)+35\mathscr{D}^{0.8}y(t)+50\mathscr{D}^{0.4}y(t)+24y(t) = \mathscr{D}^{1.2}u(t)+3\mathscr{D}^{0.4}u(t)+5u(t), $$

with zero initial conditions. Assume the input $u(t)$ is unit step signal. The following statements can be used to solve the differential equation, and the solution is shown in Fig. 11-5.

```
>> a=[1,10,35,50,24]; na=[1.6 1.2 0.8 0.4 0];
   b=[1 3 5]; nb=[1.2 0.4 0]; t=0:0.01:10; u=ones(size(t));
   y=fode_sol(a,na,b,nb,u,t); plot(t,y)
```

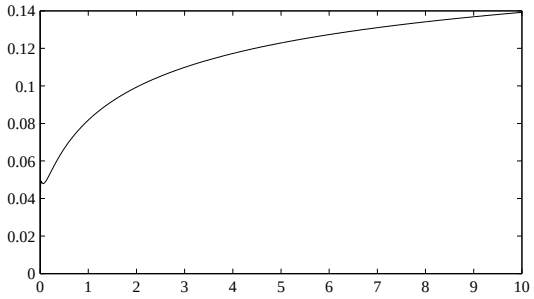


Fig. 11-5 Numerical solution of the different equation under unit step input.

11.3.2 *Numerical Solutions of Caputo Differential Equations*

If the initial values of the input and output signals are not zero, Caputo definition should be used, and we refer the fractional-order differential equations as Caputo differential equations.

Consider the Caputo linear differential equation

$$a_n \, {}_{t_0}^{\mathrm{C}}\mathscr{D}_t^{\beta_n} y(t) + a_{n-1} \, {}_{t_0}^{\mathrm{C}}\mathscr{D}_t^{\beta_{n-1}} y(t) + \cdots + a_1 \, {}_{t_0}^{\mathrm{C}}\mathscr{D}_t^{\beta_1} y(t) + a_0 \, {}_{t_0}^{\mathrm{C}}\mathscr{D}_t^{\beta_0} y(t) = \hat{u}(t). \tag{11.26}$$

For convenience, assume that $\beta_n > \beta_{n-1} > \cdots > \beta_1 > \beta_0 \geqslant 0$. The right-hand-side contains only $\hat{u}(t)$ function. For equations with linear combinations of $u(t)$ and its derivatives, the right-hand-side $\hat{u}(t)$ should be calculated first.

If $m = \lceil \beta_n \rceil$, m initial values, $y(t_0)$, $y'(t_0)$, $\cdots$, $y^{(m-1)}(t_0)$, are expected to uniquely solve the Caputo differential equations. Thus, the auxiliary variable $z(t)$ can be introduced

$$z(t) = y(t) - y(t_0) - y'(t_0)\, t - \cdots - y^{(m-1)}(t_0)\, t^{m-1}. \tag{11.27}$$

The initial values of the first $(m-1)$th order derivatives of $z(t)$ are zero. Slightly change the form of the expression yield

$$y(t) = z(t) + y(t_0) + y'(t_0)\, t + \cdots + y^{(m-1)}(t_0)\, t^{m-1}. \tag{11.28}$$

Since the initial values of $z(t)$ are zeros, ${}_{t_0}^{\mathrm{C}}\mathscr{D}_t^{\beta_i} z(t) = {}_{t_0}^{\mathrm{RL}}\mathscr{D}_t^{\beta_i} z(t)$. The β_ith order Caputo derivative of the polynomial $y(t_0) + y'(t_0)\, t + \cdots + y^{(m-1)}(t_0)\, t^{m-1}$ can be obtained with the following function

```
function s=poly2caputo(a,r), syms u tau;
s=int(diff(poly2sym(a,'tau'),ceil(r))/((u-tau)^(r-ceil(r)+1))...
        /gamma(ceil(r)-r),tau,0,u);
```

The syntax of the function is `s=poly2caputo(a,β)`, where $\boldsymbol{a}$ is the initial condition vector $\boldsymbol{a} = [y^{(m-1)}(t_0), \cdots, y'(t_0), y(t_0)]$, β is the order of differentiation. The returned variable s is symbolic expression of the polynomial, with independent variable u.

With the compensation function described earlier, the original differential equation

can be converted to

$$\begin{aligned} a_n \,{}_{t_0}^{\mathrm{RL}}\mathscr{D}_t^{\beta_n} y(t) + a_{n-1}\,{}_{t_0}^{\mathrm{RL}}\mathscr{D}_t^{\beta_{n-1}} y(t) + \cdots + a_1\,{}_{t_0}^{\mathrm{RL}}\mathscr{D}_t^{\beta_1} y(t) + a_0\,{}_{t_0}^{\mathrm{RL}}\mathscr{D}_t^{\beta_0} y(t) \\ = \hat{u}(t) - \sum_{i=0}^{n} a_i\,{}_{t_0}^{\mathrm{C}}\mathscr{D}^{\beta_i}\big[y(t_0) + y'(t_0)t + \cdots + y^{(m-1)}(t_0)t^{m-1}\big]. \end{aligned} \tag{11.29}$$

Similar to `fode_sol()` function, the following MATLAB function can be written to solve Caputo differential equation. Currently, this function can only be used in solving differential equations with right-hand-side of the equation contains only $u(t)$. The syntax of the function is $\boldsymbol{y}=$`fode_caputo(`$\boldsymbol{a},\boldsymbol{n}_{\mathrm{a}},\boldsymbol{y}_0,\boldsymbol{u},\boldsymbol{t}$`)`, where $\boldsymbol{u}$ is the sample of the input signal, and $\boldsymbol{t}$ is the evenly spaced time vector.

```
function [y,z]=fode_caputo(a,na,y0,u,t)
h=t(2)-t(1); D=sum(a./[h.^na]); nT=length(t); nb=0; b=1;
vec=[na nb]; W=[]; D1=b(:)./h.^nb(:); nA=length(a);
y1=zeros(nT,1);  W=ones(nT,length(vec));
for i=1:length(a), u=u-a(i)*subs(poly2caputo(y0,na(i)),'u',t); end
for j=2:nT, W(j,:)=W(j-1,:).*(1-(vec+1)/(j-1)); end
for i=2:nT,
   A=[y1(i-1:-1:1)]'*W(2:i,1:nA); y1(i)=(u(i)-sum(A.*a./[h.^na]))/D;
end
z=y1'; y=z+polyval(y0,t);
```

Example 11.6. Consider the Caputo linear differential equation

$$\mathscr{D}_t^{3.5}y(t) + 8\mathscr{D}_t^{3.1}y(t) + 26\mathscr{D}_t^{2.3}y(t) + 73\mathscr{D}_t^{1.2}y(t) + 90\mathscr{D}_t^{0.5}y(t) = 90\sin t^2,$$

with initial conditions, $y(0)=1$, $y'(0)=-1$, $y''(0)=2$, and $y'''(0)=3$.

The initial condition vector can be entered, and with the following statements, the Caputo differential equation can be solved, as shown in Fig. 11-6.

```
>> a=[1,8,26,73,90]; n=[3.5,3.1,2.3,1.2,0.5]; t=0:0.001:10;
   u=90*sin(t.^2); y0=[3 2 -1 1]; y=fode_caputo(a,n,y0,u,t); plot(t,y)
```

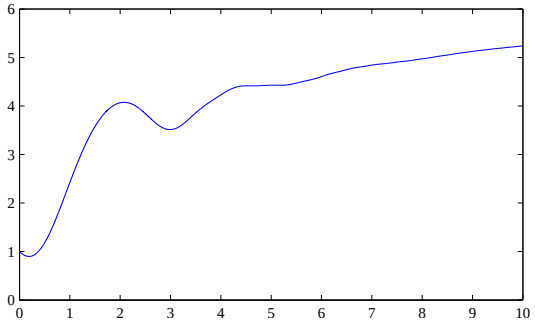


Fig. 11-6 Numerical solutions of Caputo differential equations.

11.3.3 *Some Important Laplace Transforms*

In the analytical solution approach presented later, some of the important Laplace transforms are needed. Here, we list some of the useful formula. All the Laplace transforms presented later are the variations of the following formula[11, 13]

$$\mathscr{L}^{-1}\left[\frac{s^{\alpha\gamma-\beta}}{(s^\alpha+a)^\gamma}\right]=t^{\beta-1}\mathscr{E}^{\gamma}_{\alpha,\beta}\left(-at^\alpha\right). \tag{11.30}$$

For different values of the parameters, the following formula can be derived

(1) If $\gamma=1$, and $\alpha\gamma=\beta$, or $\beta=\alpha$, the above formula can be written as

$$\mathscr{L}^{-1}\left[\frac{1}{s^\alpha+a}\right]=t^{\alpha-1}\mathscr{E}_{\alpha,\alpha}\left(-at^\alpha\right). \tag{11.31}$$

This formula can be regarded as the analytical solution of the impulse response of the fractional-order transfer function $1/(s^\alpha+a)$. It can be seen that the essential representation in fractional-order system is Mittag–Leffler function, just as the exponential function in integer-order systems.

(2) If $\gamma=1$, and $\alpha\gamma-\beta=-1$, or $\beta=\alpha+1$, Eqn. (11.30) can be written as

$$\mathscr{L}^{-1}\left[\frac{1}{s(s^\alpha+a)}\right]=t^{\alpha}\mathscr{E}_{\alpha,\alpha+1}\left(-at^\alpha\right). \tag{11.32}$$

The formula can be regarded as the analytical solution of the step response of the transfer function $1/(s^\alpha+a)$. The above equation can also be written as

$$\mathscr{L}^{-1}\left[\frac{1}{s(s^\alpha+a)}\right]=\frac{1}{a}\left[1-\mathscr{E}_{\alpha}\left(-at^\alpha\right)\right]. \tag{11.33}$$

(3) If $\gamma=k$ is an integer, and $\alpha\gamma=\beta$, i.e., $\beta=\alpha k$, Eqn. (11.30) is written as

$$\mathscr{L}^{-1}\left[\frac{1}{(s^\alpha+a)^k}\right]=t^{\alpha k-1}\mathscr{E}^{k}_{\alpha,\alpha k}\left(-at^\alpha\right). \tag{11.34}$$

and the formula can be regarded as the impulse response of $1/(s^\alpha+a)^k$.

(4) If $\gamma=k$ is an integer, and $\alpha\gamma-\beta=-1$, i.e., $\beta=\alpha k+1$, Eqn. (11.30) can be written as

$$\mathscr{L}^{-1}\left[\frac{1}{s(s^\alpha+a)^k}\right]=t^{\alpha k}\mathscr{E}^{k}_{\alpha,\alpha k+1}\left(-at^\alpha\right). \tag{11.35}$$

which can be regarded as the unit step response of $1/(s^\alpha+a)^k$.

11.3.4 *Analytical Solutions of Commensurate-order Linear Differential Equations*

Consider the orders in Eqn. (11.22). If a greatest common divisor α can be found among the orders, such that the original equation can be written as

$$\begin{aligned}&a_1\mathscr{D}_t^{n\alpha}y(t)+a_2\mathscr{D}_t^{(n-1)\alpha}y(t)+\cdots+a_n\mathscr{D}_t^{\alpha}y(t)+a_{n+1}y(t)\\&\quad=b_1\mathscr{D}_t^{m\alpha}v(t)+b_2\mathscr{D}_t^{(m-1)\alpha}v(t)+\cdots+b_m\mathscr{D}_t^{\alpha}v(t)+b_{m+1}v(t),\end{aligned} \tag{11.36}$$

the original differential equation is referred to as commensurate-order differential equations of the base order α. Denote $\lambda = s^\alpha$, the original differential equation can be expressed by the integer-order transfer function of λ. If there is no repeated poles in the system, the original transfer function can be written as the following form with the partial fraction expansion technique

$$G(\lambda) = \sum_{i=1}^{n} \frac{r_i}{\lambda + p_i} = \sum_{i=1}^{n} \frac{r_i}{s^\alpha + p_i}. \tag{11.37}$$

From the formula of Laplace transform given in Eqns. (11.31) and (11.32), the analytical solution of the impulse and step responses of the system can be obtained as

$$\mathscr{L}^{-1}\left[\sum_{i=1}^{n} \frac{r_i}{s^\alpha + p_i}\right] = \sum_{i=1}^{n} r_i t^{\alpha-1} \mathscr{E}_{\alpha,\alpha}\left(-p_i t^\alpha\right), \tag{11.38}$$

$$\mathscr{L}^{-1}\left[\sum_{i=1}^{n} \frac{r_i}{s(s^\alpha + p_i)}\right] = \sum_{i=1}^{n} r_i t^{\alpha} \mathscr{E}_{\alpha,\alpha+1}\left(-p_i t^\alpha\right). \tag{11.39}$$

The latter can alternatively written as

$$\mathscr{L}^{-1}\left[\sum_{i=1}^{n} \frac{r_i}{s(s^\alpha + p_i)}\right] = \sum_{i=1}^{n} \frac{r_i}{p_i}\left[1 - \mathscr{E}_{\alpha}\left(-p_i t^\alpha\right)\right]. \tag{11.40}$$

If the system has repeated poles, Eqns. (11.34) and (11.35) should be used to write the analytical solutions to impulse and step responses.

Example 11.7. Consider the fractional-order differential equation

$$\mathscr{D}^{1.2}y(t) + 5\mathscr{D}^{0.9}y(t) + 9\mathscr{D}^{0.6}y(t) + 7\mathscr{D}^{0.3}y(t) + 2y(t) = u(t),$$

with zero initial conditions, where $u(t)$ is the unit step signal. If base order is selected as $\lambda = s^{0.3}$, the transfer function can be written as

$$G(\lambda) = \frac{1}{\lambda^4 + 5\lambda^3 + 9\lambda^2 + 7\lambda + 2}.$$

The following MATLAB statements can be used to find its partial fraction expansion regarding to λ,

```
>> num=1; den=[1 5 9 7 2]; [r,p]=residue(num,den)
```

and the results can be obtained as

$$G(\lambda) = -\frac{1}{\lambda+2} + \frac{1}{\lambda+1} - \frac{1}{(\lambda+1)^2} + \frac{1}{(\lambda+1)^3},$$

and the Laplace transform of the output signal can be written as

$$Y(s) = \frac{1}{s}G(\lambda) = -\frac{1}{s(s^{0.3}+2)} + \frac{1}{s(s^{0.3}+1)} - \frac{1}{s(s^{0.3}+1)^2} + \frac{1}{s(s^{0.3}+1)^3}.$$

Thus, the analytical solution to the step input can be obtained as

$$\begin{aligned} y(t) &= -t^{0.3}\mathscr{E}_{0.3,1.3}\left(-2t^{0.3}\right) + t^{0.3}\mathscr{E}_{0.3,1.3}\left(-t^{0.3}\right) - t^{0.6}\mathscr{E}^2_{0.3,1.6}\left(-t^{0.3}\right) + t^{0.9}\mathscr{E}^3_{0.3,1.9}\left(-t^{0.3}\right) \\ &= \frac{1}{2} + \frac{1}{2}\mathscr{E}_{0.3}(-2t^{0.3}) + \mathscr{E}_{0.3}\left(-t^{0.3}\right) - \left[1 - \mathscr{E}_{0.3}\left(-t^{0.3}\right)\right]^2 + \left[1 - \mathscr{E}_{0.3}\left(-t^{0.3}\right)\right]^3. \end{aligned}$$

11.3.5 *Analytical Solutions of Linear Fractional-order Differential Equations*

Consider the following $(n+1)$-term fractional-order differential equation

$$a_n \mathscr{D}_t^{\beta_n} y(t) + a_{n-1} \mathscr{D}_t^{\beta_{n-1}} y(t) + \cdots + a_0 \mathscr{D}_t^{\beta_0} y(t) = u(t), \tag{11.41}$$

with step input. The analytical solution can be written as

$$\begin{aligned} y(t) \;=\; & \frac{1}{a_n} \sum_{m=0}^{\infty} \frac{(-1)^m}{m!} \sum_{\substack{k_0+k_1+\cdots+k_{n-2}=m \\ k_0 \geqslant 0,\ \cdots,\ k_{n-2} \geqslant 0}} (m; k_0, k_1, \cdots, k_{n-2}) \\ & \prod_{i=0}^{n-2} \left(\frac{a_i}{a_n} \right)^{k_i} t^{(\beta_n - \beta_{n-1})m + \beta_n + \sum_{j=0}^{n-2} (\beta_{n-1} - \beta_j) k_j - 1} \\ & \mathscr{E}^{(m)}_{\beta_n - \beta_{n-1},\ \beta_n + \sum_{j=0}^{n-2} (\beta_{n-1} - \beta_j) k_j} \left(-\frac{a_{n-1}}{a_n} t^{\beta_n - \beta_{n-1}} \right), \end{aligned} \tag{11.42}$$

where $(m; k_0, k_1, \cdots, k_{n-2})$ is defined as

$$(m; k_0, k_1, \cdots, k_{n-2}) = \frac{m!}{k_0! k_1! \cdots k_{n-2}!}. \tag{11.43}$$

It is difficult to write out analytical solutions to a certain system, since the solution form is too complicated. This method is quite useful in practical problems.

11.4 Modeling and Analysis of Fractional-order Transfer Functions

Consider the linear fractional-order system shown in Eqn. (11.22). If the initial values of the input signal $\boldsymbol{u}(t)$ and output $y(t)$ and their derivatives are all zero, with Laplace transform, the fractional-order transfer function when appended a time delay T can be written as

$$G(s) = \frac{b_1 s^{\gamma_1} + b_2 s^{\gamma_2} + \cdots + b_m s^{\gamma_m}}{a_1 s^{\eta_1} + a_2 s^{\eta_2} + \cdots + a_{n-1} s^{\eta_{n-1}} + a_n s^{\eta_n}} \mathrm{e}^{-Ts}. \tag{11.44}$$

Compared with the integer-order transfer functions, apart from the numerator and denominator coefficients, the orders can also be declared. Thus, normally four vectors and a delay constant can be used to describe uniquely the fractional-order transfer function model in Eqn. (11.44).

Since this model is useful in the analysis and design of linear fractional-order systems, we can construct a MATLAB class FOTF to describe the model as it is done in TF class, in Control System Toolbox. When the class is created, overload functions can be written to implement the modeling, analysis and design tasks, in a simple and straightforward way.

In this section, creation of class, and object-oriented programming technique are demonstrated first, then based on the class, the modeling and analysis of fractional-order transfer functions are presented.

11.4.1 *FOTF — Creation of a MATLAB Object*

If one wants to create a MATLAB class, a name of the class should be selected. For instance, for fractional-order transfer function, we selected FOTF as its name. A folder `@fotf` should be created for it, and all the files related io the class should be placed in the folder. Normally for a new class, at least two functions should be written, `fotf.m` is used to define the class, and `display.m` is used to display the class. The programming of the two functions and other supporting functions are illustrated below:

(1) **Defining FOTF class**. A function `fotf.m` should be written and placed in the `@fotf` folder. The listing of the function is

```
function G=fotf(a,na,b,nb,T)
if nargin==0,
   G.a=[]; G.na=[]; G.b=[]; G.nb=[]; G.ioDelay=0; G=class(G,'fotf');
elseif isa(a,'fotf'), G=a;
elseif nargin==1 & isa(a,'double'), G=fotf(1,0,a,0,0);
elseif nargin==1 & a=='s', G=fotf(1,0,1,1,0);
else, ii=find(abs(a)<eps); a(ii)=[]; na(ii)=[];
   ii=find(abs(b)<eps); b(ii)=[]; nb(ii)=[];
   if nargin==5, G.ioDelay=T; else, G.ioDelay=0; end
   G.a=a; G.na=na; G.b=b; G.nb=nb; G=class(G,'fotf');
end
```

The command G=`fotf(`$\boldsymbol{a},\boldsymbol{n}_{\rm a},\boldsymbol{b},\boldsymbol{n}_{\rm b},T$`)` can be used to enter a FOTF object, where $\boldsymbol{a}=[a_1,a_2,\cdots,a_n]$,$\boldsymbol{b}=[b_1,b_2,\cdots,b_m]$, $\boldsymbol{n}_{\rm a}=[\eta_1,\eta_2,\cdots,\eta_n]$ and $\boldsymbol{n}_{\rm b}=[\gamma_1,\gamma_2,\cdots,\gamma_m]$ can be used to represent the coefficients and orders of the numerator and denominator of the system, and T is the delay constant. If there is no delay in the model, the variable can be omitted.

Similar to `tf()` function, function $s=$`fotf('s')` command can be used to declare an s operator for the fractional-order model. The command $G=$`fotf(`k`)` can be used to convert a constant to FOTF object. If G is an LTI object in Control System Toolbox, the command $G=$`fotf(`G`)` can be used to convert the TF object into an FOTF object.

(2) **Writing display function**. Another function, `display.m`, should be written in that folder, which is used to display the FOTF object, once it is created. The listing of the function is

```
function display(G)
strN=fpoly2str(G.b,G.nb); strD=fpoly2str(G.a,G.na);
nn=length(strN); nd=length(strD); nm=max([nn,nd]);
disp([char(' '*ones(1,floor((nm-nn)/2))) strN]), ss=[];
T=G.ioDelay; if T>0, ss=[' exp(-' num2str(T) 's)']; end
disp([char('-'*ones(1,nm)), ss]);
disp([char(' '*ones(1,floor((nm-nd)/2))) strD])
function strP=fpoly2str(p,np)
if isempty(np), p=0; np=0; end
P=''; [np,ii]=sort(np,'descend'); p=p(ii);
for i=1:length(p), P=[P,num2str(p(i)),'s^',num2str(np(i)),'+']; end
P=strrep(strrep(strrep(P,'s^0+','+'),'s^1+','s+'),'s^1-','s-');
P=strrep(strrep(strrep(P,'+-','-'),'+1s','+s'),'-1s','-s');
```

```
strP=P(1:end-1); nP=length(strP);
if nP>=2 &  strP(1:2)=='1s', strP=strP(2:end); end
```

Example 11.8. For the fractional-order transfer function model

$$G(s) = \frac{0.8s^{1.2} + 2}{1.1s^{1.8} + 1.9s^{0.5} + 0.4} \mathrm{e}^{-0.5s},$$

the following MATLAB commands can be used to directly enter the model. The display result is displayed as

```
>> G=fotf([1.1,1.9,0.4],[1.8,0.5,0],[0.8,2],[1.2,0],0.5)
```

It should be noted that these files must be placed in `@fotf` folder, and should not be placed elsewhere. Otherwise, the files cannot be called, and they may affect the existing MATLAB files with the same names.

(3) **Other facilities**. Further, apart from the two essential functions fotf.m and display.m, if we want to access the members in the FOTF object directly with MATLAB, the following two files are written. With these functions, commands like G`.nb` and G`.na = [0.1,0.2]` are supported

```
function A=subsasgn(G,index,InputVal)
switch index.subs
   case {'a','na','b','nb','ioDelay'},
      eval(['G.' index.subs,'=InputVal;']);
      if length(G.a)~=length(G.na) | length(G.b)~=length(G.nb)
         error('Error: field pairs (na,a) or (nb,b) mismatched.')
      else, A=fotf(G.a,G.na,G.b,G.nb,G.ioDelay); end
   otherwise, error('Error: Available fields are a, na, b, na, ioDelay.');
end
```

```
function A=subsref(G,index)
switch index.subs
   case {'a','na','b','nb','ioDelay'}, A=eval(['G.' index.subs]);
   otherwise,
      error('Error: Available fields are a, na, b, na, ioDelay.');
end
```

(4) **Conversion function from TF to FOTF object**. The following function should be placed in the `@tf` folder.

```
function G1=fotf(G)
[n,d]=tfdata(tf(G),'v'); nn=length(n)-1:-1:0;
nd=length(d)-1:-1:0; G1=fotf(d,nd,n,nn,G.ioDelay);
```

11.4.2 *Interconnections of FOTF Blocks*

It has been shown in Chapter 4 that integer-order models can be calculated with `+`, `*` and `feedback()` functions to process the parallel, series and feedback connection. Similar to the idea, the following overload functions can be written. These files should be placed in the `@fotf` folder. Most of the functions are from the book [5], however, some of them are modified and extended.

(1) **Multiplications of FOTF blocks**. To define the expression $G=G_1*G_2$, the overload function `mtimes()` should be written. This function is used to evaluate the series connection of two FOTF blocks, $G_1(s)$ and $G_2(s)$, the algorithm is

$$G(s) = G_1(s)G_2(s) = \frac{N_1(s)N_2(s)}{D_1(s)D_2(s)}. \tag{11.45}$$

The overload function can be written as

```
function G=mtimes(G1,G2)
G1=fotf(G1); G2=fotf(G2); na=[]; nb=[];
a=kron(G1.a,G2.a); b=kron(G1.b,G2.b);
for i=1:length(G1.na), na=[na,G1.na(i)+G2.na]; end
for i=1:length(G1.nb), nb=[nb,G1.nb(i)+G2.nb]; end
G=simple(fotf(a,na,b,nb,G1.ioDelay+G2.ioDelay));
```

(2) **Adding FOTF blocks**. The expression $G=G_1+G_2$ should be described by the overload function `plus()` to evaluate the parallel connection of two FOTF blocks, with the algorithm

$$G(s) = G_1(s) + G_2(s) = \frac{N_1(s)D_2(s) + N_2(s)D_1(s)}{D_1(s)D_2(s)}. \tag{11.46}$$

The following overload function is implemented

```
function G=plus(G1,G2)
G1=fotf(G1); G2=fotf(G2); na=[]; nb=[];
if G1.ioDelay==G2.ioDelay
   a=kron(G1.a,G2.a); b=[kron(G1.a,G2.b),kron(G1.b,G2.a)];
   for i=1:length(G1.a),
      na=[na G1.na(i)+G2.na]; nb=[nb, G1.na(i)+G2.nb];
   end
   for i=1:length(G1.b), nb=[nb G1.nb(i)+G2.na]; end
   G=simple(fotf(a,na,b,nb,G1.ioDelay));
else, error('cannot handle different delays'); end
```

(3) **Feedback function**. The function $G=$`feedback(`G_1`,`G_2`)` evaluates the negative connection of the two FOTF blocks. If the positive structure is used, the forward path can be converted to $-G_2$, such that negative feedback structure can still be used.

$$G(s) = \frac{G_1(s)}{1 + G_1(s)G_2(s)} = \frac{N_1(s)D_2(s)}{D_1(s)D_2(s) + N_1(s)N_2(s)}. \tag{11.47}$$

The listing of the overload function is

```
function G=feedback(F,H)
F=fotf(F); H=fotf(H); na=[]; nb=[];
if F.ioDelay==H.ioDelay
   b=kron(F.b,H.a); a=[kron(F.b,H.b), kron(F.a,H.a)];
   for i=1:length(F.b),
      nb=[nb F.nb(i)+H.nb]; na=[na,F.nb(i)+H.nb];
   end
   for i=1:length(F.a), na=[na F.na(i)+H.na]; end
```

```
   G=simple(fotf(a,na,b,nb,F.ioDelay));
else, error('cannot handle blocks with different delays'); end
```

(4) **Simple supporting functions**. Function `uminus()` is used to evaluate $G_1(s) = -G(s)$, with the syntax $G_1 = -G$; Function $G = \texttt{inv}(G_1)$ is used to evaluate $G(s) = 1/G_1(s)$; Function `minus()` is used to evaluate $G(s) = G_1(s) - G_2(s)$, with $G = G_1 - G_2$. Function `eq()` judges whether two FOTF blocks G_1 and G_2 equal or not, with $\texttt{key} = G_1 == G_2$, if equal, the returned `key` is 1.

```
function G=uminus(G1), G=G1; G.b=-G.b;
function G=inv(G1), G=fotf(G1.b,G1.nb,G1.a,G1.na,-G1.ioDelay);
function G=minus(G1,G2), G=G1+(-G2);
function key=eq(G1,G2), key=0; G=G1-G2;
if length(G.nb)==0 | norm(G.b)<1e-10, key=1; end
```

(5) **Right division**. With $G = G_1/G_2$ to evaluates $G(s) = G_1(s)/G_2(s)$

```
function G=mrdivide(G1,G2)
G1=fotf(G1); G2=fotf(G2); G=G1*inv(G2);
G.ioDelay=G1.ioDelay-G2.ioDelay;
if G.ioDelay<0, warning('block with positive delay'); end
```

(6) **Power function**. With $G = G_1\hat{}n$, the power of G_1 can be evaluated, if n is integer. Otherwise, only the case G_1 is a Laplace operator can be handled.

```
function G1=mpower(G,n)
if n==fix(n),
   if n>=0, G1=1; for i=1:n, G1=G1*G; end
   else, G1=inv(G^(-n)); end, G1.ioDelay=n*G.ioDelay;
elseif G==fotf(1,0,1,1), G1=fotf(1,0,1,n);
else, error('mpower: power must be an integer.'); end
```

(7) **Simplification function**. With $G = \texttt{simple}(G)$, the coefficients of numerator and denominator are collected to simplify the description. Sub-function `polyuniq()` can be used to collect coefficients of polynomials. The sub function cannot be called directly.

```
function G=simple(G1)
[a,n]=polyuniq(G1.a,G1.na); G1.a=a; G1.na=n; na=G1.na;
[a,n]=polyuniq(G1.b,G1.nb); G1.b=a; G1.nb=n; nb=G1.nb;
if length(nb)==0, nb=0; G1.nb=0; G1.b=0; end
nn=min(na(end),nb(end)); nb=nb-nn; na=na-nn;
G=fotf(G1.a,na,G1.b,nb,G1.ioDelay);
function [a,an]=polyuniq(a,an)
[an,ii]=sort(an,'descend'); a=a(ii); ax=diff(an); key=1;
for i=1:length(ax)
   if ax(i)==0, a(key)=a(key)+a(key+1); a(key+1)=[]; an(key+1)=[];
   else, key=key+1; end
end
```

Example 11.9. The following commands can be used to express the fractional-order PID controller $G_c(s) = 5 + 2s^{-0.2} + 3s^{0.6}$ into MATLAB workspace

```
>> s=fotf('s'); Gc=5+2*s^(-0.2)+3*s^0.6
```

Example 11.10. The fractional-order transfer function model

$$G(s) = \frac{(s^{0.3}+3)^2}{(s^{0.2}+2)(s^{0.4}+4)(s^{0.4}+3)},$$

can be entered into MATLAB workspace, with the following MATLAB statements, the expanded model can be obtained

$$G(s) = \frac{s^{0.6}+6s^{0.3}+9}{s+2s^{0.8}+7s^{0.6}+14s^{0.4}+12s^{0.2}+24}.$$

```
>> s=fotf('s'); G=(s^0.3+3)^2/(s^0.2+2)/(s^0.4+4)/(s^0.4+3)
```

Example 11.11. Assume that typical unity negative feedback control system is

$$G(s) = \frac{0.8s^{1.2}+2}{1.1s^{1.8}+0.8s^{1.3}+1.9s^{0.5}+0.4},\ G_{\rm c}(s) = \frac{1.2s^{0.72}+1.5s^{0.33}}{3s^{0.8}},$$

the following statements can be used to enter the model into MATLAB workspace

```
>> G=fotf([1.1,0.8 1.9 0.4],[1.8 1.3 0.5 0],[0.8 2],[1.2 0]);
   Gc=fotf([3],[0.8],[1.2 1.5],[0.72 0.33]); GG=feedback(G*Gc,1)
```

and the closed-loop model obtained is

$$G(s) = \frac{0.96s^{1.59}+1.2s^{1.2}+2.4s^{0.39}+3}{3.3s^{2.27}+2.4s^{1.77}+0.96s^{1.59}+1.2s^{1.2}+5.7s^{0.97}+1.2s^{0.47}+2.4s^{0.39}+3}.$$

11.4.3 *Analysis of FOTF Objects*

1. Stability analysis

The stability assessment of commensurate-order systems can be carried out directly. If the base order of the commensurate-order system is $\lambda = s^\alpha$, the stable regions for the commensurate-order system are shown in Fig. 11-7. If the poles of the system of λ are located in the stable regions, then the system is stable, otherwise the system is unstable[14]. For the base order α, the boundaries of stable regions are the straight lines with slopes of $\pm\alpha\pi/2$. When the base order is $\alpha = 1$, the system is of integer-order, and the stable boundary is changed to the imaginary axis, which agrees well with the cases in integer-order systems.

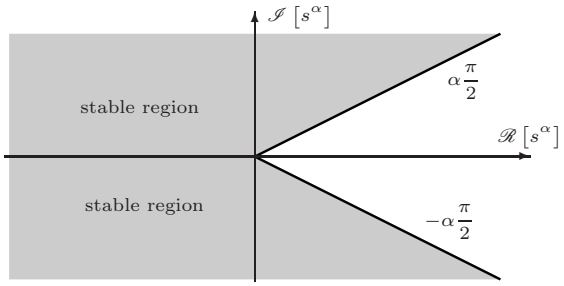


Fig. 11-7 Stable regions for commensurate-order systems.

Based on the idea, the following MATLAB function can be written. The function can be used to convert FOTF object to commensurate-order system first, and then the poles

of the system can be evaluated. Although sometimes the order of the commensurate-order system is extremely high, it can also be processed by MATLAB easily. The syntax `[key,`α`,`ϵ`,`a_1`]=isstable(`G`,`a_0`)` can be used to assess the stability of the FOTF object, where `key` is one for stable. The argument α returns the base order, ϵ is the error tolerance in root finding, a_1 is the slopes of $\pm\alpha\pi/2$, and a_0 is the user selected base order, with default of 0.01.

```
function [K,alpha,err,apol]=isstable(G,a0)
if nargin==1, a0=0.01; end
a=G.na; a1=fix(a/a0); n=gcd(a1(1),a1(2));
for i=3:length(a1), n=gcd(n,a1(i)); end
alpha=n*a0; a=fix(a1/n); b=G.a; c(a+1)=b; c=c(end:-1:1);
p=roots(c); p=p(abs(p)>eps); err=norm(polyval(c,p));
plot(real(p),imag(p),'x',0,0,'o')
apol=min(abs(angle(p))); K=apol>alpha*pi/2;
xm=xlim; xm(1)=0; line(xm,tan(alpha*pi/2)*xm)
title('Pole-Zero Map'), xlabel('Real Axis'), ylabel('Imaginary Axis')
```

Example 11.12. Consider the fractional-order transfer function

$$G(s)=\frac{-2s^{0.63}-4}{2s^{3.501}+3.8s^{2.42}+2.6s^{1.798}+2.5s^{1.31}+1.5}.$$

The following statements can be used to enter the model into MATLAB workspace first, then assess the stability of the system.

```
>> b=[-2,-4]; nb=[0.63,0];
   a=[2,3.8,2.6,2.5,1.5]; na=[3.501,2.42,1.798,1.31,0];
   G=fotf(a,na,b,nb); [key,alpha,err,apol]=isstable(G,0.001)
```

It is obvious that the base order is $\alpha=0.001$, the commensurate-order model of the system can be rewritten as an integer-order transfer function of λ

$$G(\lambda)=\frac{-2\lambda^{630}-4}{2\lambda^{3501}+3.8\lambda^{2420}+2.6\lambda^{1798}+2.5\lambda^{1310}+1.5}.$$

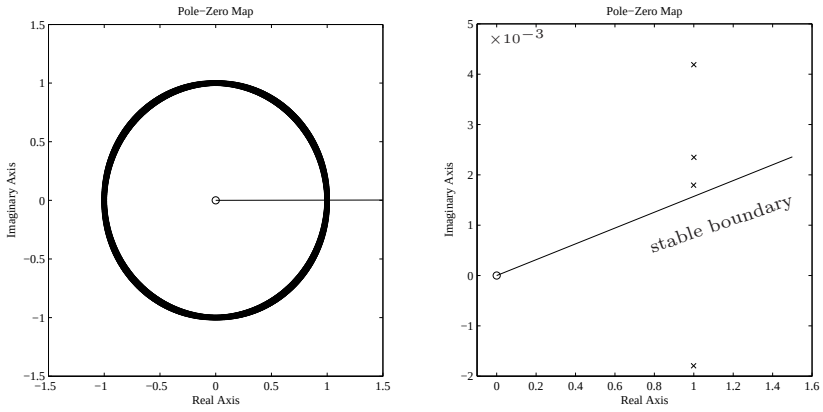
The roots of the polynomials of λ can be obtained automatically in the function, as shown in Fig. 11-8(a). The zoomed plots around the x-axis can be obtained in Fig. 11-8(b). It can be seen that all the poles of the system are located in the stable regions. Thus, the system is stable. Since $\alpha=0.001$, the polynomial is of 3501th order. It may take some time to find all the poles.

2. Norms of fractional-order systems

The norms of the systems are important quantities in robust control design. The evaluation algorithms of the norms of fractional-order systems are illustrated here. The $\mathcal{H}_2$ and $\mathcal{H}_\infty$ norms of $G(s)$ are defined respectively as

$$||G(s)||_2=\sqrt{\frac{1}{2\pi\mathrm{j}}\int_{-\mathrm{j}\infty}^{\mathrm{j}\infty}G(s)G(-s)\mathrm{d}s}, \tag{11.48}$$

$$||G(s)||_\infty=\sup_\omega\big|G(\mathrm{j}\omega)\big|. \tag{11.49}$$



(a) pole locations (b) zoomed plot

Fig. 11-8 Pole positions and stability assessment.

It can be seen that the $\|G(s)\|_2$ norm can be evaluated through numerical integration methods, while $||G(s)||_\infty$ norm can be obtained with numerical optimization approaches. The overload function `norm()` can be written and placed in the `@fotf` folder, with the syntaxes `norm(`G`)` and `norm(`G`,inf)`. In old versions of MATLAB, the `integral()` function can be replaced with `quadgk()`.

```
function n=norm(G,eps0)
j=sqrt(-1); dx=1; f0=0; if nargin==1, eps0=1e-6; end
if nargin==2 & ~isfinite(eps0) % H∞ norm, find the maximum value
   f=@(w)[-abs(freqresp(j*w,G))];
   w=fminsearch(f,0); n=abs(freqresp(j*w,G));
else % H2 norm, numerical integration
   f=@(s)freqresp(s,G).*freqresp(-s,G)/(2*pi*j);
   while (1)
      n=integral(f,-dx*j,dx*j);
      if abs(n-f0)<eps0, n=sqrt(n); break; else, f0=n; dx=2*dx;
end, end, end
```

where the low-level frequency response function `freqresp()` is given by

```
function H1=freqresp(w,G)
a=G.a; na=G.na; b=G.b; nb=G.nb; j=sqrt(-1); T=G.ioDelay;
for i=1:length(w)
   P=b*(w(i).^nb.'); Q=a*(w(i).^na.'); H1(i)=P*exp(-T*w(i))/Q;
end
```

Example 11.13. Consider the fractional-order model given in Example 11.12. The norms of the system can be evaluated, and the results are $n_1 = 2.7168$, and $n_2 = 8.6115$.

```
>> a=[2,3.8,2.6,2.5,1.5]; na=[3.501,2.42,1.798,1.31,0];
   b=[-2,-4]; nb=[0.63,0]; G=fotf(a,na,b,nb);
   n1=norm(G), n2=norm(G,inf)
```

11.4.4 *Frequency Domain Analysis of FOTF Objects*

Consider a fractional-order transfer function $G(s)$. If $j\omega$ is used to substitute s, through simple complex number computation, the exact frequency response data can be obtained directly. The data can be written in the form of the `frd()` function in the Control System Toolbox, so that the frequency domain analysis functions such as `bode()` can be used to draw frequency domain plots. Overload functions for these can also be written, and placed in `@fotf` folder. The listing of the overload function `bode()` is as follows, with the supporting function `freqresp()` as its kernel.

```
function H=bode(G,w)
if nargin==1, w=logspace(-4,4); end
j=sqrt(-1); H1=freqresp(j*w,G); H1=frd(H1,w);
if nargout==0, bode(H1); else, H=H1; end
```

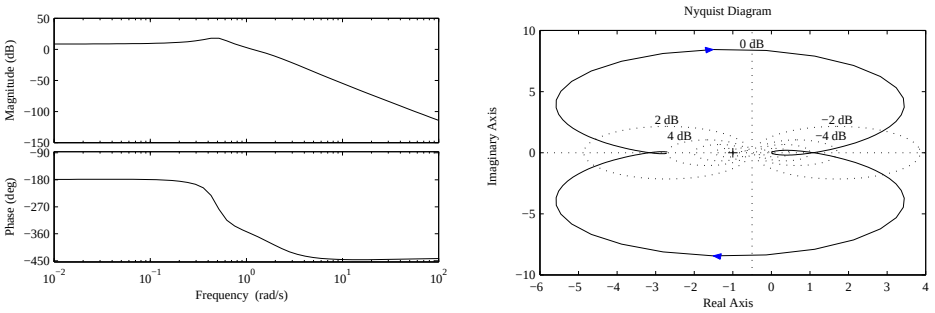
Similarly, overload functions for Nyquist plots and Nichols chart are

```
function nyquist(G,w)
if nargin==1, w=logspace(-4,4); end, H=bode(G,w); nyquist(H);
```

```
function nichols(G,w)
if nargin==1, w=logspace(-4,4); end, H=bode(G,w); nichols(H);
```

Example 11.14. Consider again the fractional-order model in Example 11.12. The following statements can be used to draw the Body diagram and Nyquist plot as shown in Figs. 11-9(a) and (b).

```
>> b=[-2,-4]; nb=[0.63,0]; w=logspace(-2,2);
   a=[2,3.8,2.6,2.5,1.5]; na=[3.501,2.42,1.798,1.31,0];
   G=fotf(a,na,b,nb); bode(G,w);
   figure, w=logspace(-2,4,400); nyquist(G,w); grid
```



(a) Bode diagrams (b) Nyquist plots

Fig. 11-9 Frequency domain plots of fractional-order transfer function.

11.4.5 *Time Domain Analysis of FOTF Objects*

Based on the closed-form solutions of linear fractional-order differential equations, and its MATLAB implementation in function `fode_sol()`, the overload step response function

`step()` and arbitrary input time response function `lsim()` of the fractional-order transfer functions can easily be written

```
function y=step(G,t)
y1=fode_sol(G.a,G.na,G.b,G.nb,ones(size(t)),t);
ii=find(t>G.ioDelay); lz=zeros(1,ii(1)-1);
y1=[lz, y1(1:end-length(lz))];
if nargout==0,
   plot(t,y1,t,c_term(G.b,G.nb)/c_term(G.a,G.na),'--'),
   title('Step Response'), xlabel('Time (Sec)'), ylabel('Magnitude')
else, y=y1; end
function c=c_term(a,na) % this function is to find constant term in polynomials
i=find(na==0); c=0; if length(i)>0, c=a(i(1)); end
```

```
function y=lsim(G,u,t)
y1=fode_sol(G.a,G.na,G.b,G.nb,u,t);
ii=find(t>G.ioDelay); lz=zeros(1,ii(1)-1);
y1=[lz, y1(1:end-length(lz))];
if nargout==0, plot(t,y1,t,u,'--'),
   title('Step Response'), xlabel('Time (Sec)'), ylabel('Magnitude')
else, y=y1; end
```

The syntaxes of the two functions are

$\boldsymbol{y}=$`step(`G`,`$\boldsymbol{t}$`)`, and $\boldsymbol{y}=$`lsim(`G`,`$\boldsymbol{u}$`,`$\boldsymbol{t}$`)`

where G is the FOTF model, $\boldsymbol{t}$ is an evenly spaced time vector, $\boldsymbol{u}$ is a vector of the input samples. We tried to make the syntaxes of these overload functions similar to those in the Control System Toolbox. It should be noted that vector $\boldsymbol{t}$ cannot be omitted here.

Example 11.15. Consider the following fractional-order differential equation

$$\mathscr{D}_t^{3.5}y(t)+8\mathscr{D}_t^{3.1}y(t)+26\mathscr{D}_t^{2.3}y(t)+73\mathscr{D}_t^{1.2}y(t)+90\mathscr{D}_t^{0.5}y(t)=90\sin t^2,$$

and it can be seen that the fractional-order transfer function is

$$G(s)=\frac{90}{s^{3.5}+8s^{3.1}+26s^{2.3}+73s^{1.2}+90s^{0.5}},$$

and the input is $u(t)=\sin t^2$. The following statements can be used to draw the time response of the output $y(t)$ as shown in Fig. 11-10, where the solid curve is the output while the dash plot is the input.

```
>> a=[1,8,26,73,90]; n=[3.5,3.1,2.3,1.2,0.5];
   G=fotf(a,n,90,0); t=0:0.002:10; u=sin(t.^2); lsim(G,u,t);
```

Similar to other computation problems in MATLAB, the results obtained should be validated. Smaller step sizes can be tried to seen whether the same results can be obtained. If the results with smaller step sizes are the same, the results can be accepted, otherwise the step size should be reduced again. For this example, the results are validated.

11.4.6 *Root Locus for Commensurate-order Systems*

For commensurate-order systems, if the base order is α, we can let $\lambda=s^\alpha$, and the original system can be written as the integer-order model of λ, denoted by $G_1(\lambda)$. Function

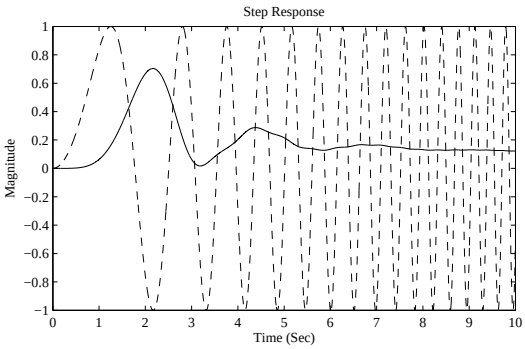


Fig. 11-10 Input and output signals of the system.

`rlocus()` in Control System Toolbox can be used to draw the root locus of integer-order model $G_1(\lambda)$, and superimpose the stability boundaries $\pm\alpha\pi/2$ on the root locus. Based on the idea, the overload function `rlocus()` can be designed. The syntax of the function is `rlocus(G)`, where G is a FOTF object.

```
function rlocus(G)
nx=unique(round(1000*[G.na,G.nb])); nx=nx(nx~=0); nd=max(nx);
for i=1:length(nx), nd=gcd(nd,nx(i)); end
alpha=nd*0.001; na=round(G.na/alpha); nb=round(G.nb/alpha);
b=G.a; den(a+1)=b; den=den(end:-1:1);
b=G.b; num(x+1)=b; num=num(end:-1:1); G1=tf(num,den);
rlocus(G1), xm=xlim; if xm(2)<=0, xm(2)=-xm(1); end
xm(1)=0; line(xm,tan(alpha*pi/2)*xm)
```

Interactive method in the original `rlocus()` function is inherited to get the critical gain of the system with mouse clicks.

Example 11.16. Assume that the fractional-order transfer function is given by

$$G(s) = \frac{1}{s^{3.5} + 10s^{2.8} + 35s^{2.1} + 50s^{1.4} + 24s^{0.7}}.$$

It can be seen that the base order is $\alpha = 0.7$. Let $\lambda = s^{0.7}$, the integer-order transfer function can be written as

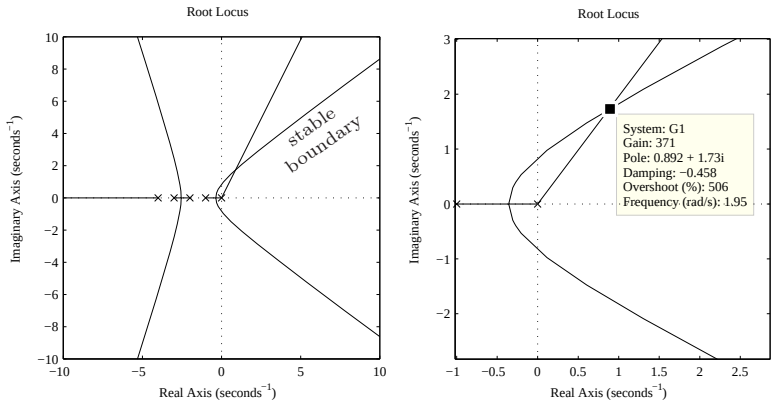
$$G(\lambda) = \frac{1}{\lambda^5 + 10\lambda^4 + 35\lambda^3 + 50\lambda^2 + 24\lambda}.$$

The root locus of the fractional-order system can be obtained as shown in Fig. 11-11(a). Zooming the root locus, the critical gain can be read $K = 371$, as shown in Fig. 11-11(b).

```
>> G=fotf([1 10 35 50 24 0],0.7*[5:-1:0],1,0); rlocus(G)
```

11.4.7 *State Space Models of Commensurate-order Systems*

If the fractional-order system can be expressed as the commensurate-order transfer function $G(\lambda)$ with base order α, the matrices $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$ for integer-order model $G(\lambda)$ can be obtained, and the state space representation of the fractional-order system can be



(a) root locus (b) zoomed plot with critical gain

Fig. 11-11 Root locus analysis of fractional-order system.

written as

$$\begin{cases} \mathscr{D}^{\alpha}\boldsymbol{x}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \\ \boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t). \end{cases} \tag{11.50}$$

The state space models of fractional-order system are not covered in the book. The interested readers are advised to create their own FOSS class and overload functions. These tasks are left as a problem.

11.5 Approximation and Reduction of Fractional-order Systems

11.5.1 *Oustaloup Filter for Fractional-order Differentiators*

The Grünwald–Letnikov definition presented earlier can be used to evaluate accurately the fractional-order derivatives for given functions. However, there are certain limitations in control systems. Control systems are always described by block diagrams, for instance, the input signal to the plant model is often generated by its previous block, i.e., the controller. Thus, the control signal is not known precisely before numerical derivatives can be calculated. Thus, block diagram approximations, usually with filters, to the fractional-order derivative actions are expected.

Normally, filters are classified as continuous and discrete ones, here, continuous filters to fit the Laplace operator s^{γ} are mainly discussed. It can be seen that fractional-order derivatives can be approximated by filters.

1. Oustaloup filter approximation

Several filter approximation approaches are summarized in [9], including continued fraction approximation, Charef approximation[15] and Oustaloup approximation[16]. Here, only Oustaloup algorithm is presented.

Since pure fractional-order derivative can be represented by straight lines in Bode diagrams, it is not possible to fit them at the whole frequency range with integer-order filters. A specific interested frequency interval $(\omega_{\mathrm{b}}, \omega_{\mathrm{h}})$ should be assigned, with the

continuous transfer function

$$G_{\rm f}(s) = K\prod_{k=1}^{N}\frac{s+\omega_k'}{s+\omega_k}, \tag{11.51}$$

where the poles, zeros and gain can be obtained from

$$\omega_k' = \omega_{\rm b}\omega_{\rm u}^{(2k-1-\gamma)/N},\quad \omega_k = \omega_{\rm b}\omega_{\rm u}^{(2k-1+\gamma)/N},\quad K=\omega_{\rm h}^{\gamma}, \tag{11.52}$$

with $\omega_{\rm u} = \sqrt{\omega_{\rm h}/\omega_{\rm b}}$. Based on the algorithm, the following MATLAB function can be written to implement Oustaloup filter. If $y(t)$ is the input signal to the filter, the output of the filter can be approximately regarded as $\mathscr{D}_t^{\gamma}y(t)$.

```
function G=ousta_fod(gam,N,wb,wh)
k=1:N; wu=sqrt(wh/wb); wkp=wb*wu.^((2*k-1-gam)/N);
wk=wb*wu.^((2*k-1+gam)/N); G=zpk(-wkp,-wk,wh^gam); G=tf(G);
```

with the syntax `G=ousta_fod(γ,N,ωb,ωh)`, where γ is the order of derivative, N is the order of the filter. The variables $\omega_{\rm b}$ and $\omega_{\rm h}$ are the lower- and upper-bounds of the frequency of users' choice. Normally within the frequency range, the Bode diagram fractional-order operator are satisfactory, while the fitting outside the range is not. The algorithm presented here avoided the restriction on $\omega_{\rm b}\omega_{\rm h}=1$, the two frequencies can be selected independently.

In the function, γ can either be positive or negative, for differentiation and integrals. Also, the absolute values of γ can be larger than 1, for instance, $\gamma = 3.7$. However, in this case, it is suggested to keep $-1<\gamma<1$, and leave the remaining integers as TF object. For instance, better to use $s^{3.7}=s^3s^{0.7}$, or $s^{3.7}=s^4s^{-0.3}$.

2. Improved Oustaloup filter

In practical applications, the orders of the numerator and denominator are the same, and if at the boundaries of the frequency the fitting is not good, improved Oustaloup filters[17] and optimal filter design method[18] can be used. The latter can also be extended to fit complex orders, however, its implementation is rather complicated.

The following improved Oustaloup filter is presented[17]. The limitations of the filter is that the order should be between 0 and 1. The improved filter is

$$s^{\gamma}\approx\left(\frac{d\omega_{\rm h}}{b}\right)^{\gamma}\left(\frac{ds^2+b\omega_{\rm h}s}{d(1-\gamma)s^2+b\omega_{\rm h}s+d\gamma}\right)\prod_{k=1}^{N}\frac{s+\omega_k'}{s+\omega_k}, \tag{11.53}$$

and the definitions of ω_k, ω_k' are the same as in Oustaloup filter. Two adjustable parameters b, d are introduced, and normally they can be set to $b=10$, $d=9$. The MATLAB implementation is given by

```
function G=new_fod(r,N,wb,wh,b,d)
if nargin==4, b=10; d=9; end, k=1:N; wu=sqrt(wh/wb);
wkp=wb*wu.^((2*k-1-r)/N); wk=wb*wu.^((2*k-1+r)/N);
G=zpk(-wkp,-wk,(d*wh/b)^r)*tf([d,b*wh,0],[d*(1-r),b*wh,d*r]);
```

3. Filter approximation of high-order fractional-order systems

Based on Oustaloup filter and its improved form, high-order integer-order approximation to fractional-order transfer function, i.e., each fractional-order term can be

approximated with the filters. Based on the idea, the following MATLAB function is designed, and placed in the `@fotf` folder.

```
function Ga=high_order(G,filter,wb,wh,N)
if nargin==1, filter='ousta_fod'; wb=1e-3; wh=1e3; N=5; end
Ga=pseudo_poly(G.b,G.nb,filter,wb,wh,N)...
          /pseudo_poly(G.a,G.na,filter,wb,wh,N);
Ga=minreal(Ga);
function G1=pseudo_poly(a,na,filter,wb,wh,N), G1=0; s=tf('s');
for i=1:length(a), na0=na(i); n1=floor(na0);
   if na0>n1, g1=eval([filter '(na0-n1,N,wb,wh)']);
   else, g1=1; end
   G1=G1+a(i)*s^n1*g1;
end
```

The syntax of the function is $G_1 =$`high_order(`G`,filter,`ω_b`,`ω_h`,`N`)`, where G is the FOTF object, `filter` can be selected as `'ousta_fod'` or `'new_fod'`. The arguments ω_b, ω_h and N are the parameters of the Oustaloup filters. The default values are $\omega_b = 10^{-3}$, $\omega_h = 10^3$, $N = 5$, and the default filter is Oustaloup filter.

It is interesting to note that if the FOTF object G is, in fact, an integer-order model, $G_1 =$`high_order(`G`)` will convert it into a TF object G_1.

Example 11.17. Consider high-order fractional-order transfer function

$$G(s) = \frac{-2s^{0.63} - 4}{2s^{3.501} + 3.8s^{2.42} + 2.6s^{1.798} + 2.5s^{1.31} + 1.5}.$$

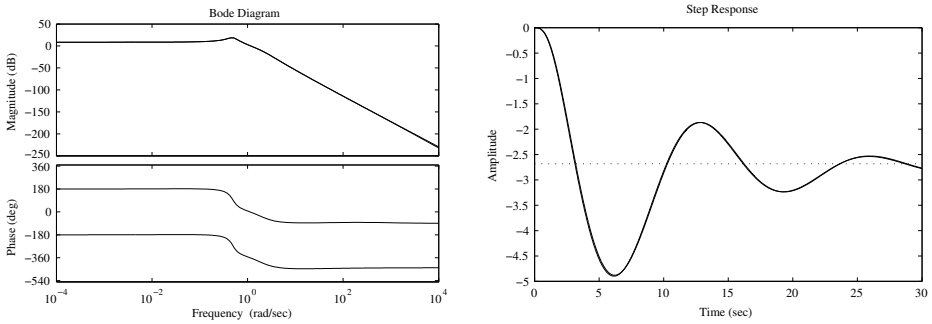
Selecting frequency interval (ω_1, ω_2), and suitable order N, the high-order integer-order approximation can be obtained, with examples $s^{3.501} = s^3 s^{0.501}$. The low-level command is rather complicated and tedious

```
>> N=9; w1=1e-3; w2=1e3; g1=ousta_fod(0.501,N,w1,w2); s=tf('s');
   g2=ousta_fod(0.42,N,w1,w2); g3=ousta_fod(0.798,N,w1,w2);
   g4=ousta_fod(0.31,N,w1,w2); g5=ousta_fod(0.63,N,w1,w2);
   G1=(-2*g5-4)/(2*s^3*g1+3.8*s^2*g2+2.6*s*g3+2.5*s*g4+1.5)
```

Alternatively, `high_order()` function can be used directly

```
>> b=[-2 -4]; nb=[0.63 0]; a=[2 3.8 2.6 2.5 1.5];
   na=[3.501 2.42 1.798 1.31 0]; G=fotf(a,na,b,nb);
   G2=high_order(G,'ousta_fod',w1,w2,N); order(G2)
   bode(G1,G2); hold on; bode(G); t=0:0.004:30;
   figure; y=step(G,t); step(G1,G2,30); line(t,y)
```

It can be seen that a 45th order integer-order model can be obtained. The exact Bode diagram and its integer-order approximation can be obtained as shown in Fig. 11-12(a). It can be seen that the magnitude curves are almost the same in the specified frequency interval, and the phase difference is 360°, thus they are effectively the same as well. For larger frequency intervals, the frequency response fitting are also satisfactory. Step responses of the fractional-order model and the integer-order approximation can also be obtained as shown in Fig. 11-12(b). It can be seen that the approximation of step response is also satisfactory.



(a) Bode diagrams (b) step responses

Fig. 11-12 The time and frequency domain comparisons.

11.5.2 *Approximations of Fractional-order Controllers*

In control system design, the controller obtained may be rather complicated, and may not be easy to implement. For instance, if $[(as+b)/(cs+d)]^{\alpha}$ term is contained in the controller, the following procedures can be used in controller approximation:

(1) Generate exact frequency response samples to the fractional-order controller;

(2) Select suitable orders in numerator and denominator for the controller;

(3) Use function `invfreqs()` to get the frequency response fitting model;

(4) Validate the fitting results. If it is not satisfactory, the orders should be increased in step (2), or go to step (1) to select again the interested frequency range until a satisfactory model can be obtained.

Example 11.18. Consider the following fractional-order QFT controller model [5]

$$G_{\rm c}(s)=1.8393\left(\frac{s+0.011}{s}\right)^{0.96}\left(\frac{8.8\times10^{-5}s+1}{8.096\times10^{-5}s+1}\right)^{1.76}\frac{1}{(1+s/0.29)^2}.$$

The controller is rather complicated, so that approximate integer-order controllers are expected, with frequency response fitting approach. Since the `frd()` function in MATLAB can only be used to deal with integer-order systems, its member variable `ResponseData` can be used to complete the computation with non-integer powers. Thus, the frequency response data of $G(s)$ can be obtained. Function `invfreqs()` can then be used to get the fitting model, within the frequency range of $\omega\in(10^{-4},10^{0})$ rad/s, with the following statements

```
>> w=logspace(-4,0); G1=tf([1 0.011],[1 0]); F1=frd(G1,w);
   G2=tf([8.8e-5 1],[8.096e-5 1]); F2=frd(G2,w);
   s=tf('s'); G3=1/(1+s/0.29)^2; F3=frd(G3,w); F=F1;
   h1=F1.ResponseData; h2=F2.ResponseData; h3=F3.ResponseData;
   h=1.8393*h1.^0.96.*h2.^1.76.*h3; F.ResponseData=h;
   [n,d]=invfreqs(h(:),w,4,4); G=tf(n,d)
```

The approximate controller obtained is

$$G(s)=\frac{2.213\times10^{-7}s^4+1.732\times10^{-6}s^3+0.1547s^2+0.001903s+2.548\times10^{-6}}{s^4+0.5817s^3+0.08511s^2+0.000147s+1.075\times10^{-9}}.$$

We can use a larger frequency interval $(10^{-6},10^{2})$ rad/s to validate the fitting results.

It can be seen the Bode diagram of the two controllers is shown in Fig. 11-13. It can be seen that apart from the frequency range at very low frequency, the fitting is satisfactory in other frequencies. If we want to enhance the fitting results, the number of frequency samples in step (1) should be increased.

```
>> w=logspace(-6,2,200); F1=frd(G1,w); F2=frd(G2,w); F=F1;
   F3=frd(G3,w); h1=F1.ResponseData; h2=F2.ResponseData;
   h3=F3.ResponseData; h=1.8393*h1.^0.96.*h2.^1.76.*h3;
   F.ResponseData=h; bode(F,'-',G,'--',w)
```

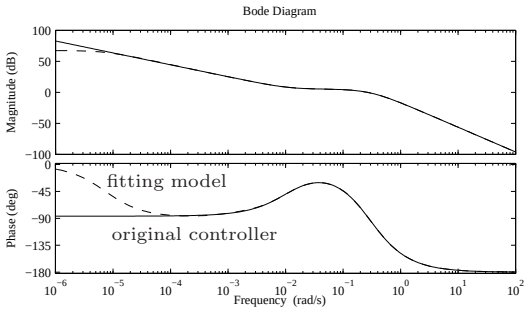


Fig. 11-13 Comparisons of fractional-order QFT controller and integer-order approximation.

11.5.3 *Optimal Reduction Algorithm for Fractional-order Models*

To get better reduction model, the error between the original model and reduced model to the same input signal should be defined, and the target is to minimize the error criterion. The optimal reduced model can be converted to the minimization of the following objective function

$$J = \min_{\boldsymbol{\theta}} \ \left\| \widehat{G}(s) - G_{r/m,\tau}(s) \right\|_2, \tag{11.54}$$

where $\boldsymbol{\theta}$ is the vector of undetermined model parameters, i.e.,

$$\boldsymbol{\theta} = \left[\beta_1, \beta_2, \cdots, \beta_r, \alpha_1, \alpha_2, \cdots, \alpha_m, \tau \right]^{\mathrm{T}}. \tag{11.55}$$

Since there is delay terms in Eqn. (11.54), Padé approximation is used for them, and the objective function can be changed to the following norm evaluation form

$$J = \min_{\boldsymbol{\theta}} \ \left\| \widehat{G}(s) - \widehat{G}_{r/m}(s) \right\|_2. \tag{11.56}$$

There is no analytical solution to the problem, thus numerical optimization technique can be used. Based on the optimal model reduction algorithm in Chapter 4, we can solve directly the optimal model reduction problem.

Example 11.19. Consider again the fractional-order transfer function in Example 11.17, the following commands can be used to get 45th order approximate model

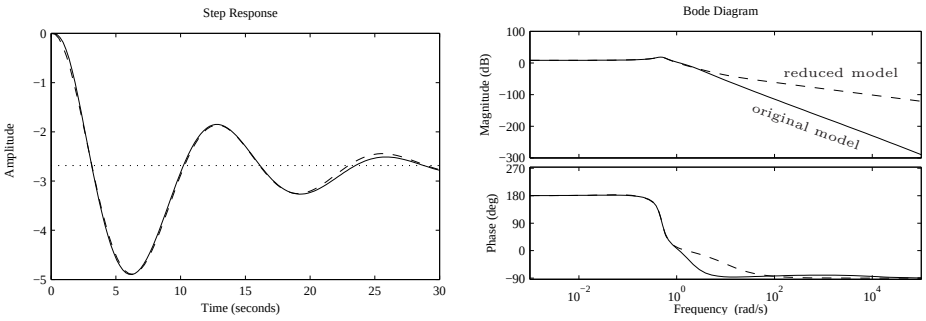
```
>> b=[-2 -4]; nb=[0.63 0]; a=[2 3.8 2.6 2.5 1.5];
   na=[3.501 2.42 1.798 1.31 0]; G=fotf(a,na,b,nb);
   G1=high_order(G,'ousta_fod',1e-3,1e3,9); order(G1)
```

Since the order of the original approximation is too high, model reduction techniques can be used with MATLAB function `opt_app()` presented in Chapter 4. The third-order reduced model is obtained as

$$G_{\rm r}(s) = \frac{0.6122s^2 + 0.6244s + 0.02588}{s^3 + 0.2014s^2 + 0.1972s + 0.01494}.$$

Comparisons on step responses and Bode diagrams are shown in Figs. 11-14(a) and (b). It can be seen that the approximate third-order model is very close to that of the original system. For high frequency magnitude fitting, it appears that the fitting is not good. However, since logarithmic scale is used, there should not be too much difference.

```
>> Gr=opt_app(G1,2,3,0); step(G1,Gr,'--',30);
   hold on, step(G,0:0.01:30);
   figure; bode(G1,Gr,'--'); hold on; bode(G);
```



(a) step response of the reduced model (b) Bode diagrams

Fig. 11-14 Time and frequency domain comparisons.

11.6 Simulation Methods for Complicated Fractional-order Systems

11.6.1 *Simulation with Numerical Laplace Transform*

It has been shown that inverse Laplace transforms can be used to solve simulation problems. However, Laplace and inverse Laplace transforms are not always solvable for complicated systems. For instance, the system with the controller in Example 11.18 cannot be easily solved. Thus, numerical techniques have to be used instead. The `INVLAP()` function is a powerful tool for performing numerical inverse Laplace transforms[19, 20], with the syntax `[t,y]=INVLAP(fun,`t_0`,`$t_{\rm f}$`,`n`)`.

The essential input arguments are: `fun`, which is a string describing the Laplace transform form $F(s)$, and $(t_0,\ t_{\rm f})$, which is the time interval, with n points.

There was a bug in the original code when $t_0 = 0$, and this is fixed in the package of the book. The remaining input arguments are internal parameters and the use of default values is suggested. More input arguments can be used in the function call, details can be found in `doc INVLAP`.

Example 11.20. Considering a fractional-order transfer function

$$G(s) = \frac{(s^{0.4} + 0.4s^{0.2} + 0.5)}{\sqrt{s}(s^{0.2} + 0.02s^{0.1} + 0.6)^{0.4}(s^{0.3} + 0.5)^{0.6}},$$

where the input is $u(t) = \mathrm{e}^{-0.2t} \sin t$, the Laplace transform of it can be obtained and converted to a string. The output signal can be obtained as shown in Fig. 11-15. The execution speed of `INVLAP()` is extremely fast, the whole process needing only 0.3 s.

```
>> G='(s^0.4+ 0.4*s^0.2+0.5)/(s^0.2+0.02*s^0.1+0.6)^0.4/(s^0.3+0.5)^0.6';
   syms t; u=exp(-0.2*t)*sin(t); Y=['(' G ')*' char(laplace(u))];
   [t,y]=INVLAP(Y,0.01,25,1000); plot(t,y)
```

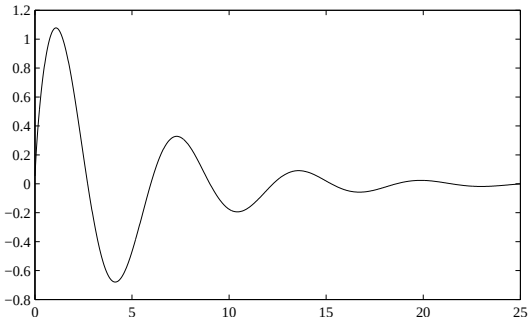


Fig. 11-15 Response to complicated fractional-order system.

To further illustrate the topic, suppose that the input signal is given by a set of sample points. Of course, using the symbolic Laplace transform is not possible for functions given by sample points. Interpolation, and then numerical integration, should be performed to find the numerical Laplace transform. Based on the source code of `INVLAP()`, numerical Laplace transform functions can be embedded to solve complicated problems. A new MATLAB function is written, with the syntax

`[`$\boldsymbol{t}$`,`$\boldsymbol{y}$`] = num_laplace(fun,`t_0`,`t_f`,`n`,`$\boldsymbol{x}_0$`,`$\boldsymbol{u}_0$`)`

where `fun` is a string describing the transfer function of the system. The extra arguments $\boldsymbol{x}_0$ and $\boldsymbol{u}_0$, specify the sample points of the input signal. Note that the function is extremely slow. The listing of the M-function is

```
function [t,y]=num_laplace(G,t0,tf,nnt,x0,u0)
FF=strrep(strrep(strrep(G,'*','.*'),'/','./'),'^','.^');
a=6; ns=20; nd=19; t=linspace(t0,tf,nnt);
if t0==0, t=t(2:end); nnt=nnt-1; end  % the original bug is fixed here
n=1:ns+1+nd; alfa=a+(n-1)*pi*1j; beta=-exp(a)*(-1).^n; n=1:nd;
bdif=fliplr(cumsum(gamma(nd+1)./gamma(nd+2-n)./gamma(n))./2^nd;
beta(ns+2:ns+1+nd)=beta(ns+2:ns+1+nd).*bdif; beta(1)=beta(1)/2;
```

```
for kt=1:nnt
   tt=t(kt); s=alfa/tt; bt=beta/tt;
   U=integral(@(x)interp1(x0,u0,x,'spline').*exp(-s.*x),...
              t0,tf,'ArrayValued',true);
   btF=bt.*eval(FF).*U; y(kt)=sum(real(btF));
end
```

Example 11.21. If the analytical form of the input function is not known, but instead only a set of sample points is given, the following statements can be given to calculate the output signal. After around 30 s, the numerical solution can be found and it is exactly the same as the one shown in Fig. 11-15. It can be seen from the example that extremely complicated problems can be solved in this way.

```
>> x0=0:0.5:25; y0=exp(-0.2*x0).*sin(x0);
   [t,y]=num_laplace(G,0,25,200,x0,y0); plot(t,y)
```

11.6.2 *Block Diagram Modeling and Simulation of Linear Fractional-order Systems*

It can be seen from the presentation earlier that the best way to evaluate the fractional-order derivatives to signals inside the system is to use filters with Oustaloup algorithms and other similar algorithms. Besides, since the orders of numerator and denominator are the same in Oustaloup filter and may lead to algebraic loops in simulation, a low-pass filter can be appended with bandwidth $\omega_{\rm h}$. The Simulink model in Fig. 11-16(a) can be established to approximate fractional-order differentiation. We can mask the model into a reusable fractional-order differentiation block. With suitably selected frequency ranges and order, the block can be used in the block diagram-based modeling of complicated nonlinear fractional-order systems.

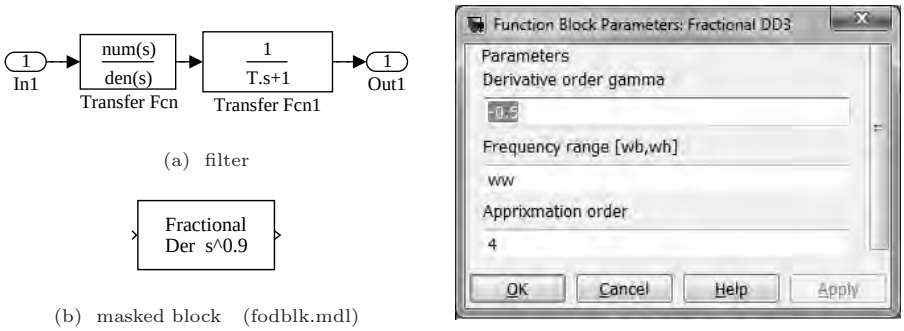
The masked block is given in Fig. 11-16(b). Double click the block, the dialog box is shown in Fig. 11-16(c). The necessary parameters can be assigned in the dialog box. In the initialization column in the masking process, the following statements can be specified, and the labels in the icon can correctly be displayed.

```
wb=ww(1); wh=ww(2);
if key==1, G=ousta_fod(gam,n,wb,wh); else, G=new_fod(gam,n,wb,wh); end
num=G.num{1}; den=G.den{1}; T=1/wh; str='Fractional\n';
if isnumeric(gam)
   if gam>0, str=[str, 'Der  s^' num2str(gam) ];
   else, str=[str, 'Int  s^{' num2str(gam) '}']; end
else, str=[str, 'Der  s^gam']; end
```

In real simulation processes, the algorithms `ode15s` or `ode23tb` are recommended, since the model is likely to be stiff equations. The following examples are used to demonstrate of fractional-order differential equations.

Example 11.22. Consider the linear fractional-order differential equation studied in Example 11.15, expressed as

$$\mathscr{D}_t^{3.5}y(t)+8\mathscr{D}_t^{3.1}y(t)+26\mathscr{D}_t^{2.3}y(t)+73\mathscr{D}_t^{1.2}y(t)+90\mathscr{D}_t^{0.5}y(t)=90\sin t^2.$$



(a) filter

(b) masked block (fodblk.mdl)

(c) parameter dialog box

Fig. 11-16 Design of fractional-order differentiator block.

Let $z(t) = \mathscr{D}_t^{0.5} y(t)$, the original model can be rewritten as

$$z(t) = \sin t^2 - \frac{1}{90}\left[\mathscr{D}_t^3 z(t) + 8\mathscr{D}_t^{2.6} z(t) + 26\mathscr{D}_t^{1.8} z(t) + 73\mathscr{D}_t^{0.7} z(t)\right].$$

Simulink model for the differential equation can be established, as shown in Fig. 11-17. After simulation, the output signal can be obtained and it is exactly the same as the one obtained in Example 11.15.

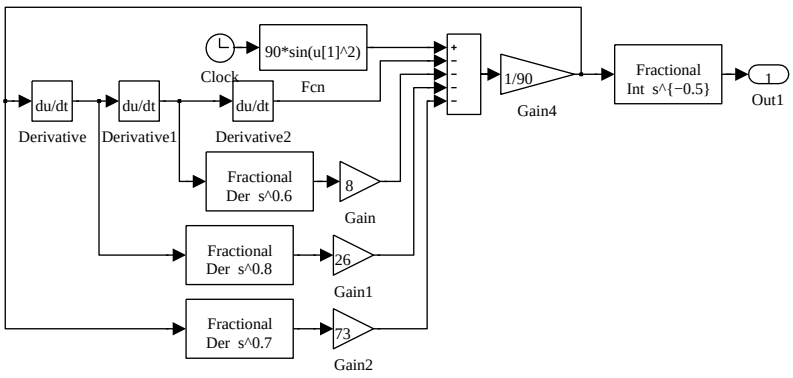
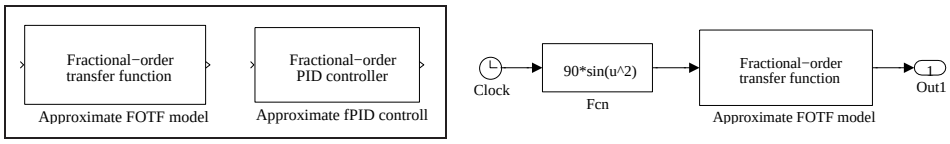


Fig. 11-17 Simulink representation for the linear fractional-order model (c11fode1.mdl).

Two more masked blocks are established for the package of the book, one for fractional-order transfer function, and the other for fractional-order PID controller. These blocks can be used directly in Simulink as shown in Fig. 11-18(a).



(a) fractional-order blocks (c11mblks.mdl)

(b) linear model (c11mlin1.mdl)

Fig. 11-18 Simulink library for fractional-order linear blocks.

The code for the Initialization pane of the Approximate FOTF model block can be expressed as

```
if strcmp(class(na),'fotf'), G=na;
else
   if length(na)~=length(a),
      errordlg('Error','Mismatch on the denominator')
   end
   if length(b)~=length(nb),
      errordlg('Error','Mismatch on the denominator')
   end
   G=fotf(a,na,b,nb);
end
if kFilter==1, str='ousta_fod'; else, str='new_fod'; end
wb=ww(1); wh=ww(2);
G0=high_order(G,str,wb,wh,N); [numG,denG]=tfdata(G0,'v');
```

Example 11.23. With the Approximate fractional-order transfer function block, the Simulink model for the differential equation in Example 11.15 can be established as shown in Fig. 11-18(b). Double click the Approximate fractional-order transfer function block, the parameters of the system can be entered as shown in Fig. 11-19. Alternatively, if the FOTF object G is established in MATLAB workspace, we can simply enter G in the Orders of denominator edit box. It can be seen that the model is much simpler than the one in the previous example. Simulation result with the new model is exactly the same as the one in the previous example.

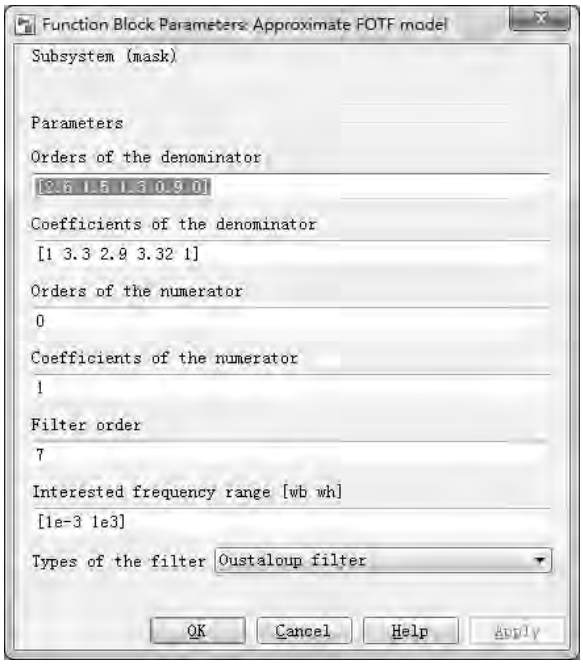


Fig. 11-19 Parameter dialog box for the FOTF block.

11.6.3 *Block Diagram Modeling and Simulation of Nonlinear Fractional-order Systems*

For complicated nonlinear fractional-order systems, the overload functions `step()`, `lsim()` and the numerical inverse Laplace transforms cannot be used. Block diagram-based modeling and simulation strategies are more important. Here, the modeling and simulation methods are illustrated through an example.

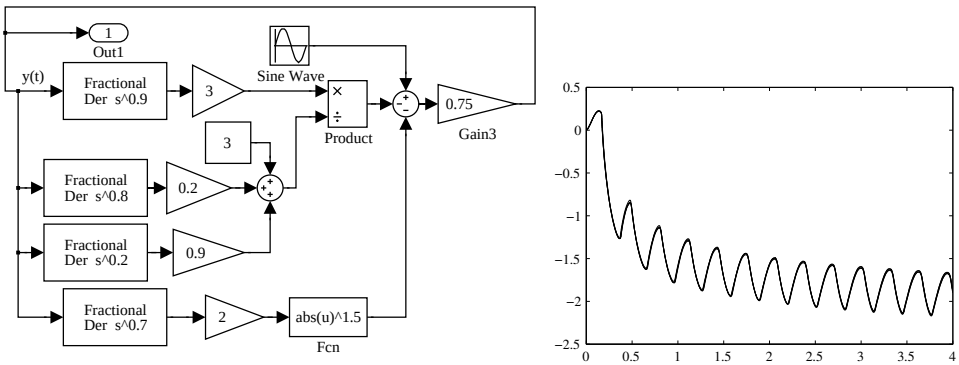
Example 11.24. Consider the following nonlinear fractional-order differential equation

$$\frac{3\mathscr{D}^{0.9}y(t)}{3+0.2\mathscr{D}^{0.8}y(t)+0.9\mathscr{D}^{0.2}y(t)}+\left|2\mathscr{D}^{0.7}y(t)\right|^{1.5}+\frac{4}{3}y(t)=5\sin 10t.$$

Based on the equation, the explicit form of the $y(t)$ signal is written as

$$y(t)=\frac{3}{4}\left[5\sin 10t-\frac{3\mathscr{D}^{0.9}y(t)}{3+0.2\mathscr{D}^{0.8}y(t)+0.9\mathscr{D}^{0.2}y(t)}-\left|2\mathscr{D}^{0.7}y(t)\right|^{1.5}\right].$$

A Simulink model can be constructed as shown in Fig. 11-20(a). It can be seen from the simulation model that the accuracy of the simulation results are to some extent, related to the specifications of the filters. Different frequency intervals and filter orders may affect the simulation results. Simulation results for different frequency intervals and filter orders are obtained in Fig. 11-20(b). It can be seen that all the curves agree well.



(a) Simulink model (c11mfod2.mdl) (b) simulation results

Fig. 11-20 Simulink model and simulation results.

11.7 Design of Optimal Fractional-order PID Controllers

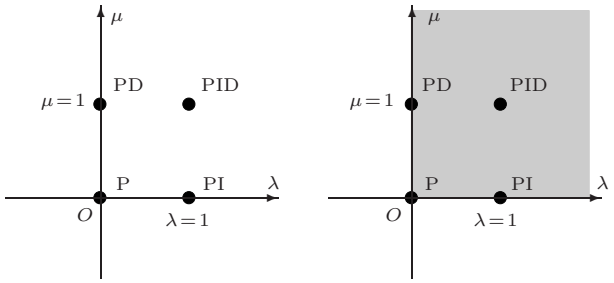
11.7.1 *Optimal Design of PI$^\lambda$D$^\mu$ Controllers*

The structures of the fractional-order PID controllers are different from the PID controllers studied in Chapter 8. The fractional PI$^\lambda$D$^\mu$ model can be expressed as

$$G_{\rm c}(s)=K_{\rm p}+\frac{K_{\rm i}}{s^\lambda}+K_{\rm d}s^\mu. \tag{11.57}$$

In the illustration shown in Fig. 11-21, the orders of integral and differentiation are

used as the two axes. It can be seen that the conventional PID-type controllers are just a few specific points on the order planes. However, the orders of the controllers in fractional-order PID controllers can be relatively arbitrarily chosen. Normally with stability considerations, $0 < \lambda, \mu < 2$. Because there are two more parameters to tune than the conventional PID controller, the fractional-order PID controllers are usually more flexible, and may expect better performances[21].



(a) conventional PID (b) fractional-order $\mathrm{PI}^{\lambda}\mathrm{D}^{\mu}$

Fig. 11-21 Illustration of fractional-order PID controllers.

From loop shaping point of view, the slopes in Bode magnitude plots are multiples of 20 dB/dec in integer-order systems, while in fractional-order systems there is no such restrictions. Thus, the shape can be arbitrarily shaped. For instance, at the places around the crossover frequency, the slope of the magnitude plot can be assigned to very small values, such that the robustness of the closed-loop system can be increased.

Example 11.25. For the given fractional-order plant model

$$G(s) = \frac{1}{s^{2.6} + 2.2s^{1.5} + 2.9s^{1.3} + 3.32s^{0.9} + 1},$$

it might be difficult to design a PID controller. Thus, there are some attempts to approximate the model with FOPDT model $G_{\mathrm{p}}(s) = k\mathrm{e}^{-Ls}/(Ts+1)$, so that the methods in Chapter 8 can be used to design PID controllers. For instance, Wang–Juang–Chan algorithm [22] can be used to design optimal ITAE criterion PID controller

$$K_{\mathrm{p}} = \frac{(0.7303 + 0.5307T/L)(T + 0.5L)}{K(T + L)}, \quad T_{\mathrm{i}} = T + 0.5L, \quad T_{\mathrm{d}} = \frac{0.5LT}{T + 0.5L}. \tag{11.58}$$

It can be seen that the FOPDT model can be obtained with

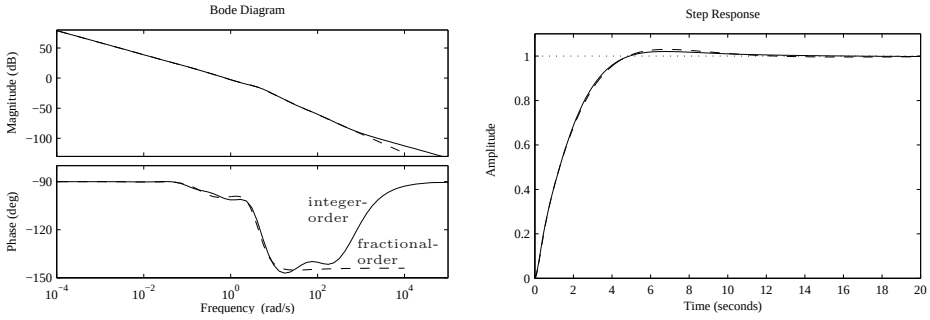
```
>> N=5; w1=1e-3; w2=1e3; s=fotf('s');
   G=1/(s^2.6+3.3*s^1.5+2.9*s^1.3+3.32*s^0.9+1);
   G0=high_order(G,'ousta_fod',w1,w2,N); Gr=opt_app(G0,0,1,1)
```

The reduced model can be obtained as $G_{\mathrm{r}}(s) = 0.1836\mathrm{e}^{-0.827s}/(s + 0.1836)$. Then a PID controller can be designed with the following statements

```
>> L=Gr.ioDelay; [n,d]=tfdata(Gr,'v'); K=n(2)/d(2); T=d(1)/d(2);
   Ti=T+0.5*L; Kp=(0.7303+0.5307*T/L)*Ti/(K*(T+L));
   Td=(0.5*L*T)/(T+0.5*L); s=tf('s'); Gc=Kp*(1+1/Ti/s+Td*s),
   w=logspace(-4,4,200); C=fotf(Gc);
```

```
H=bode(G*C,w); bode(G0*Gc,'-',H,'--');  figure;
t=0:0.01:20; step(feedback(G0*Gc,1),20),
y=step(feedback(G*C,1),t); hold on; plot(t,y,'--')
```

The PID controller can be designed as $G_c(s) = 3.9474\,(1 + 1/(5.8232s) + 0.3843s)$. Under such a controller, the open-loop Bode diagram and closed-loop step response can be obtained as shown in Figs. 11-22(a) and (b). It can be seen that the two systems are quite close.



(a) Bode diagrams (b) closed-loop step responses

Fig. 11-22 Fractional-order PID control results.

Example 11.26. Consider the plant model in the previous example. Now the searching algorithm can be used to design optimal PI$^\lambda$D$^\mu$ controller. The following MATLAB function can be written to describe the objective function

```
function fy=fpidfun(x,G,t,key)
s=fotf('s'); C=x(1)+x(2)*s^(-x(4))+x(3)*s^(x(5));
dt=t(2)-t(1); y=step(feedback(G*C,1),t); e=1-y;
if key==1, fy=dt*sum(t.*abs(e)); else, fy=dt*sum(e.^2); end
disp([x(:); fy].')
```

where in the last statement, the intermediate results can be obtained. The function has three additional arguments, G is the FOTF plant model, $\boldsymbol{t}$ is the evenly spaced time vector, and `key` is the criterion, with 1 for ITAE criterion, otherwise for ISE criterion.

Assume the terminate time is 10 s, and assume the parameters of the PI$^\lambda$D$^\mu$ controller are all smaller than 10, and the orders are in the interval $(0, 2)$. The function `fminsearchbnd()` is recommended to find optimal PI$^\lambda$D$^\mu$ controller

```
>> xm=zeros(5,1); xM=[10; 10; 10; 2; 2];
   x0=[Kp,Kp/Ti,Kp*Td,1,1].'; t=0:0.01:20;
   x=fminsearchbnd(@fpidfun,x0,xm,xM,[],G,t,1)
   s=fotf('s'); Gc1=x(1)+x(2)*s^(-x(4))+x(3)*s^(x(5));
   step(feedback(G*Gc1,1),t);
   y=step(feedback(G*C,1),t); hold on; plot(t,y,'--')
```

The optimal controller is $G_c(s) = 10 + 2.3088s^{-0.9877} + 8.9811s^{0.4286}$. The step responses of the systems under this controller and the one obtained in the previous example,

are obtained as shown in Fig. 11-23. It can be seen that the fractional-order PID controller is better than the integer-order controller.

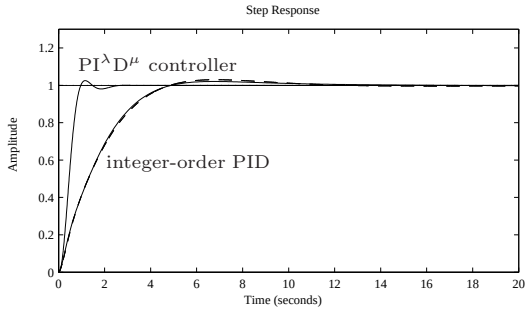


Fig. 11-23 Comparisons of different PID controllers.

Based on the above idea, an optimal fractional-order PID controller design function can be designed for linear fractional-order plants

```
function [Gc,x,y]=fpidtune(G,type,t,key,x0,xm,xM,ff)
if nargin==7, ff=optimset; ff.MaxIter=50; end
x=fminsearchbnd(@fpidfuns,x0,xm,xM,ff,G,t,key,type);
[y,Gc]=fpidfuns(x,G,t,key,type);
```

The syntax of the function is

$[G_{\rm c},\boldsymbol{x},y]=$ `fpidtune(`G`,type,`$\boldsymbol{t}$`,key,`$\boldsymbol{x}_0$`,`$\boldsymbol{x}_{\rm m}$`,`$\boldsymbol{x}_{\rm M}$`,ff)`

where G is the FOTF plant model, `type` is the expected controller type, with options `'fpid'`, `'fpi'`, `'fpd'`, `'fpidx'`, and `'pid'`, with `'fpidx'` for PID$^\mu$ controller with integer integral. The argument $\boldsymbol{t}$ is the evenly spaced time vector, `key` is the type of criterion, with options `'itae'`, `'ise'`, `'iae'` and `'itse'`, with `'itae'` recommended. The definitions of variables $\boldsymbol{x}_0$, $\boldsymbol{x}_{\rm m}$, $\boldsymbol{x}_{\rm M}$ are the same as defined earlier. Variable `ff` is the optimization control template, and it can be omitted.

For different types of fractional-order controllers and criteria, the supporting MATLAB function describing the objective function can be written as

```
function [fy,C]=fpidfuns(x,G,t,key,type), s=fotf('s');
switch type
   case 'fpid', C=x(1)+x(2)*s^(-x(4))+x(3)*s^(x(5));
   case 'fpi', C=x(1)+x(2)*s^(-x(3));
   case 'fpd', C=x(1)+x(2)*s^x(3);
   case 'fpidx', C=x(1)+x(2)/s+x(3)*s^x(4);
   case {'pid','PID'}, C=x(1)+x(2)/s+x(3)*s;
end
dt=t(2)-t(1); y=step(feedback(G*C,1),t); e=1-y;
switch key
   case {'itae','ITAE'}, fy=dt*sum(t.*abs(e));
   case {'ise','ISE'}, fy=dt*sum(e.^2);
   case {'iae','IAE'}, fy=dt*sum(abs(e));
   case {'itse','ITSE'}, fy=dt*sum(t.*e.^2);
```

```
   otherwise, error('Error: available criteria are itae, ise, iae, itse.')
end
disp([x fy])
```

Since this is an open structure with `switch` commands, other controller structures and criteria selections can be added to the source code directly by the readers, if necessary.

Example 11.27. Consider again the problem in Example 11.26. The optimal fractional-order PID controller can be obtained directly with the following MATLAB function, and the results are exactly the same as the ones in the previous example.

```
>> s=fotf('s'); G=1/(s^2.6+3.3*s^1.5+2.9*s^1.3+3.32*s^0.9+1);
   xm=zeros(5,1); xM=[10; 10; 10; 2; 2]; x0=[1;1;1;1;1].';
   t=0:0.01:20; [Gc,x]=fpidtune(G,'fpid',t,'itae',x0,xm,xM)
```

The following statements can also be used to design the optimal PID controller

```
>> xm=zeros(3,1); xM=[10; 10; 10]; x0=[1;1;1].';
   t=0:0.01:20; [Gc1,x]=fpidtune(G,'pid',t,'itae',x0,xm,xM)
   step(feedback(G*Gc,1),t); hold on; step(feedback(G*Gc1,1),t);
```

with an optimal conventional PID controller $G_{c1} = 9.9945 + 1.5107/s + 9.1101s$. The closed-loop step responses under the two controllers are shown in Fig. 11-24. It can be seen that the closed-loop response under $\mathrm{PI}^{\lambda}\mathrm{D}^{\mu}$ controller is much better than the conventional PID controller for the fractional-order plant model.

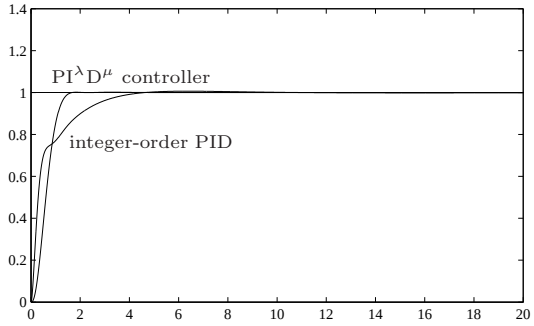


Fig. 11-24 Comparisons of different PID controllers.

If the plant model and controller model are both approximated with Oustaloup filters, the following statements should be specified, and integer-order closed-loop model, usually of extremely high order (for this example, a 43th order closed-loop model is obtained, with 7th order Oustaloup filter), can be obtained. The closed-loop step response of the high order integer-order system is almost the same as the one obtained in Fig. 11-24.

```
>> Gc0=high_order(Gc,'ousta_fod',1e-3,1e3,7);
   G0=high_order(G,'ousta_fod',1e-3,1e3,7); G1=feedback(G0*Gc0,1);
   order(G1), step(G1,t); hold on; step(feedback(G*Gc,1),t)
   Gc10=high_order(Gc1); step(feedback(G0*Gc10,1),t)
```

Example 11.28. The fractional-order PID control system is modeled with Simulink as shown in Fig. 11-25. When the parameters in the plant and fractional-order PID controller

are specified, simulation results can be obtained. It can be seen that the control results are exactly the same as the one obtained in the previous example.

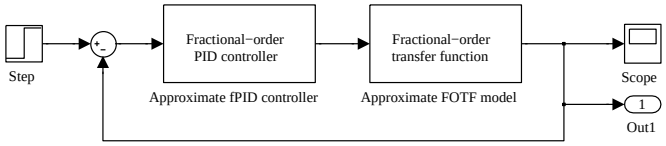


Fig. 11-25 Fractional-order PID controller system (file name: fPID_simu.mdl).

11.7.2 *OptimFOPID — An Optimal Fractional-order PID Controller Design Interface*

Based on the algorithms presented earlier, a graphical user interface, OptimFOPID, is designed. This function can be used to design optimal fractional-order PID controllers with user interface[23].

The plant model G in FOTF format should be entered into MATLAB workspace first. Then type `optimfopid` command at MATLAB prompt. The user interface shown in Fig. 11-26 will be displayed. Click Plant model, the model G can be loaded into the interface. Then the controller type, object function type and terminate simulation time should be selected in the interface. Clicking Optimize button will invoke the optimal controller design process, and finally the optimal controller can be obtained in G_c, in MATLAB workspace. Clicking Closed-loop response button will show the step response of the closed-loop system.

Example 11.29. Consider the plant model

$$G(s) = \frac{1}{0.8s^{2.2} + 0.5s^{0.9} + 1},$$

The following procedures can be used to design optimal fractional-order PID controller.

(1) Type `optimfopid` command to invoke the interface.

(2) Enter the FOTF model G into MATLAB workspace, and click Plant model to load the model into the interface.

```
>> G=fotf([0.8 0.5 1],[2.2 0.9 0],1,0)
```

(3) Set the upper bounds of the controller parameters to 15, and terminate time at 8. It should be noted that the upper bounds of controller parameters may affect the final search results.

(4) Click the Optimize button to initiate the optimization process, and the optimal fractional-order controller can be obtained, and for this example, the optimal vector is

$$\boldsymbol{x} = [\,6.5954 \;\; 15.7495 \;\; 11.4703 \;\; 0.9860 \;\; 1.1932\,].$$

The controller model can be written as

$$G_c(s) = 6.5954 + \frac{15.7495}{s^{0.986}} + 11.4703s^{1.1932}.$$

(5) Click Closed-loop response to draw the closed-loop step response of the system, as shown in Fig. 11-27(a). Since the order of integrator is very close to 1, PIDˆmu item from

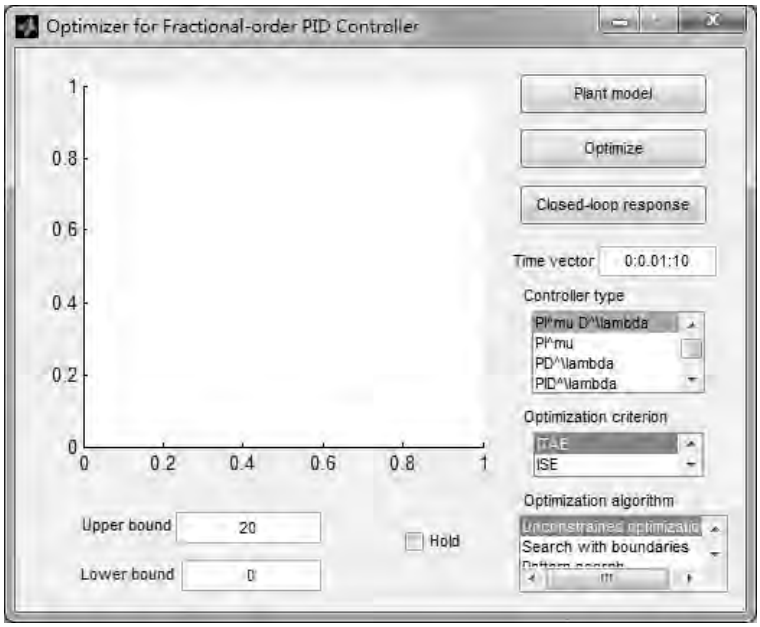
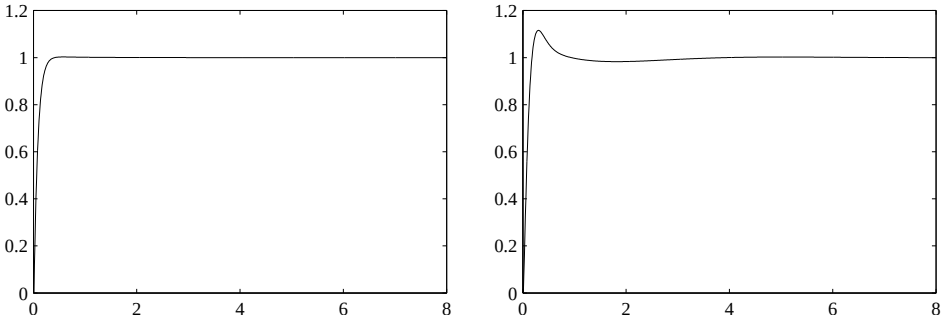


Fig. 11-26 Optimal fractional-order PID controller design interface.

the controller type listbox can be selected, and optimal PID$^{\mu}$ controller can be designed. The result is very close to the one obtained by PI$^{\lambda}$D$^{\mu}$ controller.

```
>> t=0:0.01:8; y=step(feedback(G*Gc,1),t); plot(t,y)
```



(a) fractional-order PID controller (b) optimal integer-order PID controller

Fig. 11-27 Comparisons of optimal fractional-order and integer-order PID controllers.

If the PID item from the Controller Type listbox is selected, and then if Optimize button is clicked, the optimal integer-order PID controller can be designed, and the closed-loop step response can be obtained as shown in Fig. 11-27(b). It can be seen that for this example, the results of fractional-order PID controller is better than the integer-order one.

For linear fractional-order plant models, the OptimFOPID interface can be used to

directly design fractional-order PID controllers in a user friendly manner. There are also limitations in the interface. For instance, in the current version, the plant should not contain time delays. Also, controllers with actuator saturation cannot be processed.

11.8 Problems

(1) Assume that the fractional-order differential equation is [2]

$$0.8\mathscr{D}_t^{2.2}y(t) + 0.5\mathscr{D}_t^{0.9}y(t) + y(t) = 1,\ y(0) = y'(0) = y''(0) = 0,$$

find the numerical solutions. If orders 2.2 and 0.9 are approximated by integers 2 and 1, the original differential equation can be approximated by integer-order differential equation. Please compare the approximation results.

(2) With the code for Mittag–Leffler functions, verify the following
(i) $\mathscr{E}_{\alpha,\beta}(x) + \mathscr{E}_{\alpha,\beta}(-x) = 2\mathscr{E}_{\alpha,\beta}(x^2)$, (ii) $\mathscr{E}_{\alpha,\beta}(x) - \mathscr{E}_{\alpha,\beta}(-x) = 2x\mathscr{E}_{\alpha,\alpha+\beta}(x^2)$,
(iii) $\mathscr{E}_{\alpha,\beta}(x) = \dfrac{1}{\Gamma(\beta)} + \mathscr{E}_{\alpha,\alpha+\beta}(x)$, (iv) $\mathscr{E}_{\alpha,\beta}(x) = \beta\mathscr{E}_{\alpha,\beta+1}(x) + \alpha x\dfrac{\mathrm{d}}{\mathrm{d}x}\mathscr{E}_{\alpha,\beta+1}(x)$.

(3) Two filter approximation approaches are proposed in the chapter on fractional-order derivatives. Please compare the two filters for the following fractional-order system, in frequency and step response fitting.

$$G(s) = \frac{s+1}{10s^{3.2} + 185s^{2.5} + 288s^{0.7} + 1}.$$

(4) Analyze the stability of the closed-loop system, and draw Bode diagram and closed-loop step response for the following system.

$$G(s) = \frac{s^{1.2} + 4s^{0.8} + 7}{8s^{3.2} + 9s^{2.8} + 9s^2 + 6s^{1.6} + 5s^{0.4} + 9},\quad G_{\mathrm{c}}(s) = 10 + \frac{9}{c^{0.97}} + 10s^{0.98}.$$

(5) Draw root locus for the following fractional-order plant models and find the critical gains of them.
(i) $G_1(s) = \dfrac{s^{1.5} + 9s + 24s^{0.5} + 20}{3s^2 + 16s^{1.5} + 9s + 20s^{0.5}}$, (ii) $G_2(s) = \dfrac{s+1}{10s^{3.2} + 185s^{2.5} + 288s^{0.7} + 1}$.

(6) With reference to the FOTF class definition and overload function programming methods, please define a FOSS class for state space representation of commensurate-order systems, and write suitable overload functions. Write out FOTF and FOSS conversion functions. The transfer functions in Problem 5 can be used to validate the class.

(7) Consider the complicated plant models

$$G(s) = \frac{(s^{0.2} + 3s^{0.1} + 3)^{0.6}}{s^{0.7}(s^{0.1} + 2)^{0.5}(s^{0.4} + 2)^{0.3}}.$$

It is obvious that FOTF class cannot be used to handle the plant model. Thus, the overload `bode()` function cannot be used directly. Please draw the Bode diagram of the system through low-level commands.

(8) Please consider the following fractional-order control system with

$$G(s) = \frac{1}{s^{0.5}(s^{0.2}+2)^{0.7}} \mathrm{e}^{-1.3s}, \text{ with } G_{\mathrm{c}}(s) = 0.8 + \frac{2}{s^{0.45}} + 0.6s^{0.3}.$$

Again the FOTF class cannot be used to handle the closed-loop system representation. Please try to draw the unit step response of the closed-loop system through numerical inverse Laplace transform approach.

(9) Solve the following nonlinear fractional-order differential equation with zero initial conditions, where $f(t) = 2t + 2t^{1.545}/\Gamma(2.545)$.

$$\mathscr{D}^2 x(t) + \mathscr{D}^{1.455} x(t) + \left[\mathscr{D}^{0.555} x(t)\right]^2 + x^3(t) = f(t).$$

(10) For the fractional-order model

$$G_1(s) = \frac{5}{s^{2.3}+1.3s^{0.9}+1.25}, \; G_2(s) = \frac{5s^{0.6}+2}{s^{3.3}+3.1s^{2.6}+2.89s^{1.9}+2.5s^{1.4}+1.2},$$

please find integer-order approximations, and find out the suitable order of the filters. Find also a suitable reduced order model, and compare frequency domain and step response of the reduced order systems.

(11) Find suitable low-order approximations to the following fractional-order models, and compare frequency domain fitting results.

(i) $G(s) = \dfrac{25}{(s^2+8.5s+25)^{0.2}}$, (ii) $G(s) = \dfrac{562920(s+1.0118)^{0.6774}}{(s^2+54.7160s+590570)^{0.8387}}$.

(12) Design optimal integer-order PID controller and PI$^{\lambda}$D$^{\mu}$ controller, and observe the control results.

$$G(s) = \frac{5s^{0.6}+2}{s^{3.3}+3.1s^{2.6}+2.89s^{1.9}+2.5s^{1.4}+1.2}.$$

(13) Consider the `fpidfuns()` function. If the following controller and criterion are used, please extend the function

$$G_{\mathrm{c}}(s) = K_{\mathrm{p}}\left(1 + \frac{K_{\mathrm{i}}}{s}\right)\left(1 + \frac{K_{\mathrm{d}}s}{Ts+1}\right), \; I = \int_0^{\infty} t^2 e^2(t)\mathrm{d}t.$$

(14) Extend the OptimFOPID interface, such that more optimization algorithms can be used, to design optimal PI$^{\lambda}$D$^{\mu}$ controllers with global optimization algorithms.

(15) Consider the following uncertain fractional-order plant model $G = b/(as^{0.7}+1)$, with nominal values $a = b = 1$, approximate the plant model with integer-order transfer function, and design robust optimal $\mathcal{H}_{\infty}$ controller, and observe control results with simulation methods for $a \in (0.2, 5)$, and $b \in (0.2, 1.5)$.

Bibliography and References

[1] Miller K S, Ross B. An introuction to fractinal calculus and fractional differential equations. New York: John Wiley & Sons, 1993

[2] Podlubny I. Fractional differential equations. San Diago: Academic Press, 1999

[3] Vinagre B M, Chen Y Q. Fractional calculus applications in automatic control and robotics. Las Vegas: 41st IEEE CDC, Tutorial workshop 2, 2002

[4] Manabe S. The non-integer integral and its application to control systems. Japanese Institute of Electrical Engineers Journal, 1960, 80(860):589–597
[5] Monje C A, Chen Y Q, Vinagre B M, Xue D Y, Feliu V. Fractional-order systems and controls — fundamentals and applications. London: Springer, 2010
[6] Lakshmikantham V, Leela S. Theory of fractional dynamic systems. Cornwall, UK: Cambridge Scientific Publishers, 2010
[7] Caponetto R, Dongola G, Fortuna L, Petráš I. Fractional order systems — modeling and control applications. Singapore: World Scientific Publishing, 2009
[8] Hilfer R. Applications of fractional calculus in physics. Singapore: World Scientific Publishing, 2000
[9] Petráš I, Podlubny I, O'Leary P, Dorčák L, Vinagre B M. Analogue realization of fractional order controllers. Fakulta BERG, Technical University of Košice, 2002
[10] Xue D Y, Chen Y Q. Solving applied mathematical problems with MATLAB. Boca Raton: CRC Press, 2008
[11] Shukla A K, Prajapati J C. On a generalization of Mittag–Leffler function and its properties. Journal of Mathematical Analysis and Applications, 2007, 336(1):797–811
[12] Podlubny I. Mittag–Leffler function, 2005. `http://www.mathworks.cn/MATLABcentral/fileexchange/8738-mittag-leffler-function`
[13] Kilbas A A, Saigob M, Saxena R K. Generalized Mittag–Leffler function and generalized fractional calculus operators. Integral Transforms and Special Functions, 2004, 15(1):31–49
[14] Matignon D. Stability properties for generalized fractional differential systems. Matignon D, Montseny D, eds., Proceedings of the Colloquium Fractional Differential Systems: Models, Methods and Applications, 5. Paris, 1998 145–158
[15] Charef A, Sun H H, Tsao Y Y, Onaral B. Fractal system as represented by singularity function. IEEE Transactions on Automatic Control, 1992, 37(9):1465–1470
[16] Oustaloup A, Levron F, Mathieu B, Nanot F M. Frequency-band complex noninteger differentiator: characterization and synthesis. IEEE Transaction on Circuit and Systems-I: Fundamental Theory and Applications, 2000, TCS-47(1):25–39
[17] Xue D Y, Zhao C N, Chen Y Q. A modified approximation method of fractional order system. Proceedings of IEEE Conference on Mechatronics and Automation. Luoyang, China, 2006: 1043–1048
[18] Meng L, Xue D Y. An approximation algorithm of fractional order pole models based on an optimization process. Proceedings of Mechatronics and Embedded Systems and Applications. Qingdao, China, 2010: 486–491
[19] Valsa J, Brančik L. Approximate formulae for numerical inversion of Laplace transforms. International Journal of Numerical Modelling: Electronic Networks, Devices and Fields, 1998, 11(3):153–166
[20] Valsa J. Numerical inversion of Laplace transforms in MATLAB. MATLAB Central File ID: #32824, 2011
[21] I. Podlubny. Fractional-order systems and PI$^{\lambda}$D$^{\mu}$controllers. IEEE Transactions on Automatic Control, 1999, 44(1):208–214
[22] Wang F S, Juang W S, Chan C T. Optimal tuning of PID controllers for single and cascade control loops. Chemical Engineering Communications, 1995, 132:15–34
[23] Xue D Y, Chen Y Q. OptimFOPID: a MATLAB interface for optimum fractional-order PID controller design for linear fractional-order plants. Proceedings of Fractional Derivatives and Its Applications. Nanjing, China, 2012 #307

Index of Functions

Index