Simulação com sincronismo de tempo real usando funções Range Kutta no Matlab

Anísio Rogério Braga[1]*

Resumo
Apresenta-se a modelagem matemática de um sistema dinâmico contínuo de 1^a ordem e uma genereralização de um modelo de função de transferência de ordem n. Um diagrama de simulação baseado em integradores em cascata é apresentado para o sistema de ordem n. Uma implementação da função com as equações diferenciais adequada para simulação com funções Range Kutta no ambiente Matlab é descrito e ilustrado com alguns exemplos. O sistema é executado em tempo real usando como sincronismo o relógio do computador para diferentes escalas de tempo para ilustrar as vantagens de se modificar a escala da dinâmica de modelos de ordem n sem necessidade de normalização dos coeficientes do modelo dinâmico.

Palavras chaves
ODE— Range Kutta

[1] *Núcleo de Tecnologia da Informação, COLTEC-UFMG, Belo Horizonte-MG, Brasil*
*__Sugestões e colaborações para erratas__: leic.coltec@gmail.com

Sumário

1. Introdução

Plantas analógicas com entradas e saídas digitais e analógicas são essencias para se testar estratégias de automação e controle de processos integrados. Da perspectiva de um CLP (Controlador Lógico Programável) uma planta é percebida e modificada através de sinais analógicos (contínuos ou amostrados) e digitais (liga/desliga) por meio de interfaces de cartões de hardware dedicados. Portanto, uma planta ou processo reais (e.g. tanques, motores, fornos, linhas de manufatura, etc) pode ser feita indistinguível de uma planta simulada digitalmente e acoplada a um hardware dedicado para a transdução digital analógica e vice versa valendo-se de sincronismos de escala de tempo ajustável com um relógio de tempo real.

O projeto de um simulador híbrido de baixo custo é viável utilizando e.g. uma plataforma *Raspberry PI 3+* para experimentos com *hardware in the loop* com um CLP. A ideia é simular sistemas dinâmicos contínuos e discretos com o tempo escalonável de simulação sincronizado com um relógio de tempo real com entradas e saídas dotadas de interface analógico-digital e digital-analógica. O sincronismo da solução via simulação permite que constantes de tempo ou características dinâmicas sejam aceleradas (plantas muito lentas como forno demoram horas para convergir) ou desaceleradas como no caso de dinâmicas de motores sem necessidade de reparametrizar os modelos dinâmicos. Os componentes básicos para uma simulação híbrida compreendem circuitos de expansão com condicionadores de sinais para entradas analógicas (4@20mA -> ADC 12 bits) e saídas analógicas (DAC 12 bits – volts). Isoladores óticos para interface de entradas e saídas digitais (liga/desliga). Entradas digitais são importantes para simular sinalizadores de estado como partida, parada, emergência, alarmes de sensores com nível alto e baixo, condições de falhas, etc. Um circuito gerador de sinais de 4@ 20mA com ajuste de escala via potenciômetro permite realizar testes interativos facilmente com a planta simulada. Sinais de corrente podem ser usados para testar entradas analógicas do CLP, acionar manualmente a planta híbrida simulada no microcontrolador ou acionar instrumentos reais como válvulas pneumáticas.

Na figura 1 apresenta-se um esboço de uma interface para um simulador de processo com ajustes para a função de transferência, associações de portas analógicas de entrada e saída e o passo do Range Kutta que é o intervalo de tempo real para o sincronismo entre simulação e o hardware de interface analógica.

Apresenta-se a modelagem de um circuito de filtro passa-baixas de 1^a ordem com um exemplo ilustrando a simulação com sincronismo de tempo real e mudanças de escala de tempo. A seguir ilustra-se a obtenção sistematizada de um

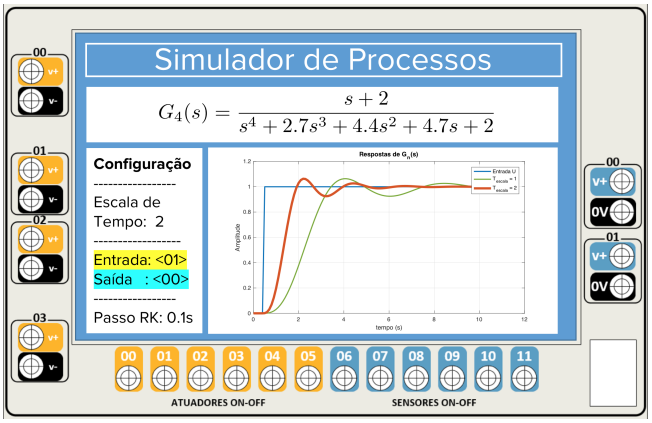

Figura 1. Ilustração de uma tela de simulação híbrida projetada para o módulo MICAsh.

diagrama de simulação de um modelo de função de transferência de ordem n-ésima, a partir do qual obtém um conjunto de equações diferencias de 1 ordem adequados para simulação numérica de sistemas dinâmicos utilizando o algoritmo Range Kutta. Exemplos e códigos Matlab que implementam a simulação híbrida sincronizada com o relógio de tempo real são apresentados.

2. Modelagem de um sistema contínuo de 1^a ordem

A figura 2 ilustra o circuito de um filtro passa-baixa de 1^a ordem. A corrente elétrica que circula através do resistor R é a mesma que atravessa o capacitor C. Aplicando a Lei de Ohm e de Kirchhoff obtemos a corrente através de R

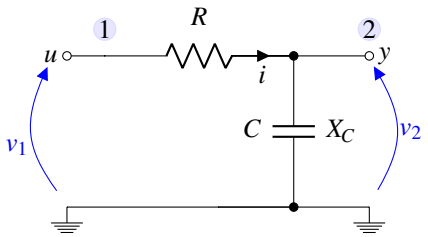

Figura 2. Circuito de um filtro passa-baixas de 1^a ordem.

$$i = \frac{u-y}{R}, \tag{1}$$

e a corrente através do capacitor C

$$i = C\frac{\mathrm{d}y}{\mathrm{d}t}. \tag{2}$$

Igualando as equações (1) e (2) e isolando saída y e entrada u obtemos

$$RC\frac{\mathrm{d}y}{\mathrm{d}t} + y = u. \tag{3}$$

A função de transferência de (3) é

$$G(s) = \frac{1}{RCs+1}. \tag{4}$$

Exemplo 1 *Simulação de filtro passa-baixas de 1^a ordem. A simulação do sistema de 1^a ordem (3) usando a função* ***ODE45*** *do Matlab é obtida escrevendo-se um arquivo da função (3) como segue:*

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \frac{-y}{RC} + \frac{u}{RC}. \tag{5}$$

Listing 1. Função ODE para um sistema de 1^a ordem.

```
function dydt = odeFPB1(t,y,u,Parametros)
% Filtro RC passa-baixas
Tau = Parametros(1);
dydt = zeros(1,1);
dydt(1) = -y/Tau+ u/Tau;
```

A simulação em tempo real para um sistema dado por $G(s) = \frac{1}{10s+1}$ é realizada por meio do programa Listings-2 como segue:

Listing 2. Função de simulação escalonável de um sistema de 1^a ordem.

```
%% Simulacao digital-analogica de processos
% Anisio R. Braga, COLTEC-UFMG
% 2017-08-07
clear all; %close all;
% Definicao de funcao de entrada
uat = @(t) (1 + sign(t-0.1))/2; % Degrau
    unitario
%---------------------------
% Definicao do sistema
% Parametros de um filtro PB1
R = 10;
C = 1.0;
Tau = R*C;
Parametros =[ Tau];
% ------------------------------------
% Leitura inicial do relogio de tempo real
T_escala = 2; % acelerador de simulacao
dt  = 0.1; % intervalo de tempo real de
    amostragem do simulador
t_0 = cputime;  % lê o relogio ao energizar
    o simulador
t_atual = cputime - t_0; %zera o relogio
    t_atual = 0;
t_final = t_atual + 10; % tempo real de
    simulacao do modelo
t_alvo  = t_atual + dt; % tempo alvo inicial
% Partida do Sistema
t_tend = t_atual;
y_tend = 0; % condicao inicial
u_tend = 0;
y0 = 0;
i  = 0;
th=[];
```

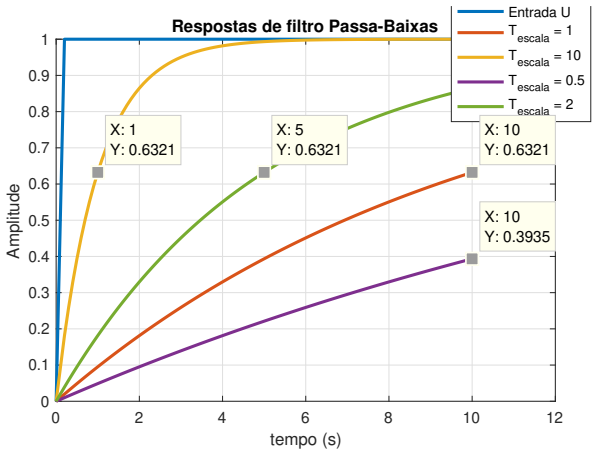

Figura 3. Resultados de simulação de múltiplas escalas de tempo para um intervalo de tempo real de 10s.

```
% -------------------------------------
t_transcorrido =  cputime;
while t_atual <= t_final,
t_atual = cputime - t_0; % le o tempo
    atualif t_atual >= t_alvo,
   t_alvo = t_alvo + dt;  % calcula o
       proximo instante
   % para atualizar a saida
   % ler ADC com a acao de controle u_atual
   % u_atual = analog_in(t_alvo);
   u_atual = uat(t_atual);
   % Simula processo
   t = T_escala *
       linspace(t_tend(end),t_alvo,4);
   [t,y] = ode45(@(t,y)
       odeFPB1(t,y,u_atual,Parametros),t,y0);
   y0 = y(end); % salva o ultimo valor da
       saida para a proxima iteracao
   % Salva o valor final do Range Kutta
       para desenhar tendencias
   t_tend = [t_tend;  t(end)/T_escala];
   u_tend = [u_tend; u_atual];
   y_tend = [y_tend;  y(end)];
   % Escreve o valor final do RK no DAC...
   % analog_out(y(end);
else % do nothing
end % if
end % while
t_transcorrido =  cputime - t_transcorrido
plot(t_tend,u_tend, t_tend, y_tend)
```

Na figura 3 observa-se que a mudança de escala permite acelerar e reduzir a dinâmica para uma mesma duração de execução em tempo real. Nota-se também que a constante de tempo efetiva (indicada nas bandeirinhas amarelas) varia de acordo com a escala escolhida de tempo real.

No caso em que se deseja simular processos dinâmicos de ordem maior que 1, é desejável que se estabeleça uma forma sistematizada para modelar e simular processos dinâmicos a partir de funções de transferência de qualquer ordem. A seguir, na seção 3, apresenta-se o diagrama de simulação a partir do qual se utiliza uma variável auxiliar para representar o comportamento dinâmico através de um conjunto de derivadas de 1^a ordem.

3. Diagrama de simulação de um sistema contínuo de n-ésima ordem

Um sistema de n-ésima ordem é descrito pela função de transferência

$$G(s) = \frac{Y}{U} = \frac{b_m s^m + b_{m-1}s^{m-1} + \ldots + b_0}{s^n + a_{n-1}s^{n-1} + \ldots + a_0}, \tag{6}$$

em que $m < n$ e o polinômio do denominador é mônico, i.e. o coeficiente do termo de maior ordem é unitário, $a_n = 1$. Para simular (6) cria-se um nó auxiliar denominado E que inserido em (6) resulta em

$$G(s) = \frac{Y}{U} = \frac{(b_m s^m + b_{m-1}s^{m-1} + \ldots + b_0)E}{(s^n + a_{n-1}s^{n-1} + \ldots + a_0)E}. \tag{7}$$

Reescreve-se uma equação para cada variável de entrada e saída

$$Y = (b_m s^m + b_{m-1}s^{m-1} + \ldots + b_0)E \tag{8}$$

$$U = (s^n + a_{n-1}s^{n-1} + \ldots + a_0)E, \tag{9}$$

e fatorando-se o termo de maior ordem s^n produz-se apenas integradores em cascata, i.e.

$$Y = (b_m s^{m-n} + b_{m-1}s^{m-n-1} + \ldots + b_0 s^{-n})E \tag{10}$$

$$U = (1 + a_{n-1}s^{-1} + \ldots + a_0 s^{-n})E. \tag{11}$$

Explicitando-se o nó auxiliar E tem-se:

$$Y = (b_m s^{m-n} + b_{m-1}s^{m-n-1} + \ldots + b_0 s^{-n})E \tag{12}$$

$$E = U - (a_{n-1}s^{-1} + \ldots + a_0 s^{-n})E. \tag{13}$$

As equações algébricas (12) e (13) simulam o sistema (7) usando somente blocos integradores e blocos de ganhos dados pelos coeficientes da função de transferência (6) como ilustrado na figura 4 do exemplo 2.

Exemplo 2 *Obter o diagrama de simulação para a função de transferência*

$$\frac{Y}{U} = \frac{b_2 s^2 + b_1 s + b_0}{s^3 + a_2 s^2 + a_1 s + a_0}. \tag{14}$$

Cria-se o nó auxilar E e fatora-se o termo de maior potência como segue:

$$Y = (b_2 s^{-1} + b_1 s^{-2} + b_0 s^{-3})E \tag{15}$$

$$E = U - (a_2 s^{-1} + a_1 s^{-2} + a_0 s^{-3})E. \tag{16}$$

Na figura 4 está representado o diagrama de blocos das equações algébricas (15) e (16).

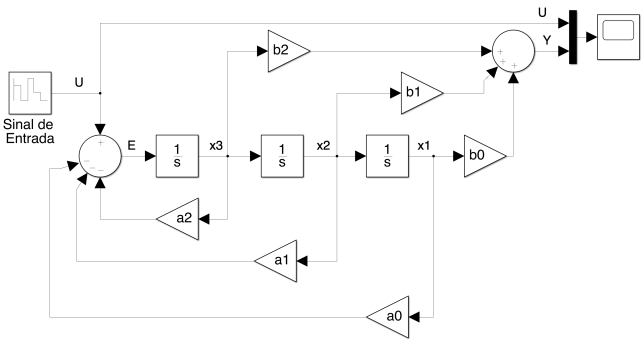

Figura 4. Diagrama de simulação de uma função de transferência de ordem $n = 3$ usando blocos integradores e de ganho.

3.1 Equação diferencial de um sistema de n-ésima ordem

A equação diferencial que representa um sistema de n-ésima ordem é facilmente obtida da equação (9), assumindo o nó auxiliar E como sendo a função com derivadas sucessivas, também conhecida como representação por *Variável de Fase*. Numa representação de espaço de estados por *Variável de Fase* atribui-se estados a variável de saída, $x_1 = e$, e respectivas derivadas sucessivamente, $x_2 = \frac{de}{dt}, \cdots, x_n = \frac{d^{n-1}e}{dt^{n-1}}$.

$$s^n E = U - (a_{n-1}s^{n-1} + \ldots + a_0)E. \tag{17}$$

Na forma matricial tem-se

$$\begin{bmatrix} \frac{dx_1}{dt} \\ \frac{dx_2}{dt} \\ \vdots \\ \frac{dx_{n-1}}{dt} \\ \frac{dx_n}{dt} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ -a_0 & -a_1 & -a_2 & \ldots & -a_{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ U \end{bmatrix} \tag{18}$$

$$y = \begin{bmatrix} b_0 & b_1 & \ldots & b_{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{19}$$

A simulação do sistema de n-ésima ordem (18) e (19) usando a função **ODE45** do Matlab é obtida escrevendo-se um arquivo da função (18) como segue:

Listing 3. Função ODE de um sistema genérico de ordem n.

```
function dxdt = odeGn(t,x,u,Den)
% Denominador F.T. SISO de n-esima ordem
n        = length(Den)-1;
dxdt     = zeros(n,1);
A        = [zeros(n,1) eye(n,n-1)];
A(end,:) = -fliplr(Den(2:end));
B        = [zeros(n-1,1); 1];

dxdt     = A*x + B*u;
```

O programa de simulação com o algoritmo *Range Kutta* com sincronismo de tempo real de (18) e (19) [1] é apresentado a seguir:

Listing 4. Função de simulação escalonável de um sistema genérico de ordem n.

```
%% Simulação digital-analógica de processos
% Anísio R. Braga, COLTEC-UFMG
% 2017-08-07
clear all; %close all;
%==============================
% SETUP
% Definição de função de entrada
uat = @(t) (1 + sign(t-0.4))/2; % Degrau
    unitario
%---------------------------
% Definição do sistema G(s) = Num/Den
% Ordem(Num) = Ordem(Den)-1  << completar
    com zeros
Num = [1 ]; Den = [10 1]; % FPB1
Num = [1 2]; Den = [1 0.7 2]; % G2
Num = [1 2]; Den = conv(Den,[1 2 1]); % G4

% Den tem que ser ôpolinmio ômnico, i.e.
    Den(1)=1
Num = Num/Den(1);
Den = Den/Den(1);
% Ajusta ordem do vetor Num nb=na-1
na = length(Den); % Ordem de Den
nb = length(Num); % Ordem de Num
if nb < na-1, Num =[zeros(1,na-nb-1) Num];
    end;
% ------------------------------------
% Leitura inicial do relógio de tempo real
T_escala = 1; % acelerador de simulação
dt  = 0.1; % intervalo de tempo real de
    amostragem do simulador
t_0 = cputime;  % lê o relógio ao energizar
    o simulador
t_atual = cputime - t_0; %zera o relógio
    t_atual = 0;
t_final = t_atual + 10; % tempo real de
    simulação do modelo
t_alvo  = t_atual + dt; % tempo alvo inicial
% Partida do Sistema
t_tend = t_atual;
y_tend = 0; % condição inicial
u_tend = 0;
x0 = zeros(length(Den)-1,1);

%==============================
% LOOP
t_transcorrido =  cputime;
```

[1] Note que a saída y dada por (19) é calculada depois do retorno da função (18) implementada em **odeGn.m**.

```
while t_atual <= t_final,
t_atual = cputime - t_0; % lê o tempo
    atualif t_atual >= t_alvo,
   t_alvo = t_alvo + dt;  % calcula o
       próximo instante
   % para atualizar a saída
   % ler ADC com a ação de controle u_atual
   % u_atual = analog_in(t_alvo);
   u_atual = uat(t_atual);
   % Simula processo
   t = T_escala *
       linspace(t_tend(end),t_alvo,4);
   [t,x] = ode45(@(t,x)
       odeGn(t,x,u_atual,Den),t,x0);
   x0 = x(end,:);
   y  = fliplr(Num)*[x(end,:)]';
   % Salva o valor final do Range Kutta
       para desenhar tendências
   t_tend = [t_tend;  t(end)/T_escala];
   u_tend = [u_tend; u_atual];
   y_tend = [y_tend;  y];
   % Escreve o valor final do RK no DAC...
   % analog_out(y(end);
else % do nothing
end % if
end % while
t_transcorrido =  cputime - t_transcorrido
plot(t_tend,u_tend, t_tend, y_tend,
    'Linewidth',2);
hold on
[y,t]=lsim(tf(Num,Den),u_tend,...
 [0:length(u_tend)-1]/length(u_tend)*t_final);
plot(t,y,'--','Linewidth',2)
```

Exemplo 3 (Simulação de uma função de 4^a ordem.) *A simulação do sistema*

$$G_4(s) = \frac{s+2}{(s^2+0.7s+2)(s^2+2s+1)} \tag{20}$$

é feita usando a função Listing-3 e o programa Listing-4 com os seguintes parâmetros:

Num=[1 2] e Den=[1 2.7 4.4 4.7 2].

Os resultados de simulação para o sistema com escala de tempo real e acelerada de 2 vezes estão ilustrados na figura 5.

4. Simulação de processos híbridos e não-lineares

<Propor uma interface padronizada para edição de funções ODE para simulação de processos híbridos>

Variáveis de interface com o hardware do MICAsh para uma função genérica de processo híbrido (vide figura 1).

- Entradas Digitais: A00, A01, A02, A03, A04, A05.
- Saídas Digitais: S00, S01, S02, S03, S04, S05.
- Entradas Analógicas: EA00, EA01, EA02, EA03.

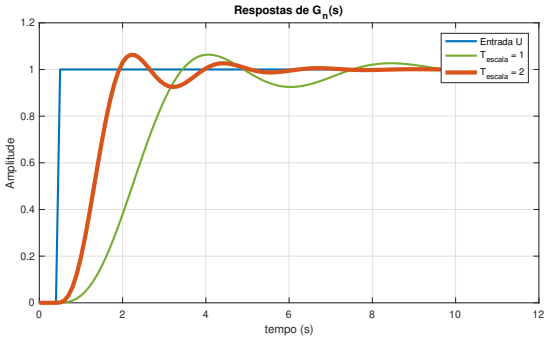

Figura 5. Resultados de simulação de $G_4(s)$ em escala de tempo normal e acelerada 2 vezes.

- Saídas Analógicas: SA00, SA01.

Além das variáveis de hardware se faz necessário padronizar variáveis para a interface genérica do Simulador de Processos do MICAsh. Na figura 1 vislumbra-se as seguintes variáveis para uma interface básica de simulação baseado nos dados de SETUP do programa Listing-4, como segue:

- Processo

  $\text{Num} = [b_m\ b_{m-1}\ \cdots\ b_0]$

  $\text{Den} = [1\ a_{n-1}\ \cdots\ a_0]$

- Escala de tempo: $T_{escala} = 1$ (valor default).
- Atribuições de entrada e saída:

  Entrada: <variável> = <porta>. Exemplo: u=EA00.

  Saída: <variável> = <porta>. Exemplo: y=SA00.

- Passo do Range Kutta: dt = 0.1s.

5. Comentários finais

Apresentou-se algoritmo e estrutura apropriados para simulação usando funções Range Kutta de sistemas com sincronismo de tempo real adequados para implementação de processos e plantas híbridas simuladas digitalmente com flexibilidade de escalonamento das dinâmicas dos modelos utilizados.