Levantando Modelo de Planta Térmica

Lab. De Controle Automático II

Prof. Fernando Passold

2012/1

Material levantado usando:
MATLAB Version 7.9.0.529 (R2009b)
Operating System: Mac OS X Version: 10.7.3 Build: 11D50b

Engenharia Elétrica/UPF

Lab. De Controle Automático II
Prof. Fernando Passold
Março/2012

Levantando Modelo de Planta Térmica

Lab. De Controle Automático II

A idéia aqui é usar o MATLAB como ferramenta para levantar modelos de 1a-ordem com atraso de transporte. No caso, baseado em dados reais de uma planta térmica (estufa de aquecimento).

Controle temperatura

Primeiramente navegue até o diretório contendo o arquivo com a resposta em malha aberta do sistema:

```
>> pwd
ans =
/Users/fernandopassold/Documents/UPF/Controle_Digital/
Lab_Controle_Digital/Sensor_Temperatura
```

Carregando o arquivo com os dados para dentro do MATLAB:

```
>> load -ascii estufa1.txt
>> u=length(estufa1) % Esta função descobre o tamanho do vetor (ou quantidade de
amostradas contidas dentro do arquivo. Notar que o próprio nome do arquivo sem a
extensão se transforma na variável contendo os dados da amostragem realizada.
u =
       10007 % ou seja, 10.007 amostras
```

A seguir se faz necessário recriar um vetor tempo começando de t=0 até o instante final de amostragem adotado.

>> t=0:10:u*10-1; % amostras a cada 10 seg.

```
>> t=0:u-1; % gera vetor t variando de 0 até 10006
>> whos % Este comando mostra as variáveis e seus tamanhos
  Name             Size                Bytes  Class     Attributes
  ans              1x39                   78  char
  estufa1      10007x1                 80056  double
  t                1x10007             80056  double
  u                1x1                     8  double
>> plot(t,estufa1) % Gera um primeiro gráfico com a resposta real do sistema
```

```
>> title('Resposta Malha Aberta');
>> xlabel('Tempo (s)');
>> ylabel ('Temperatura (^oC)')
```

A figura 1 gerada é mostrada a seguir.

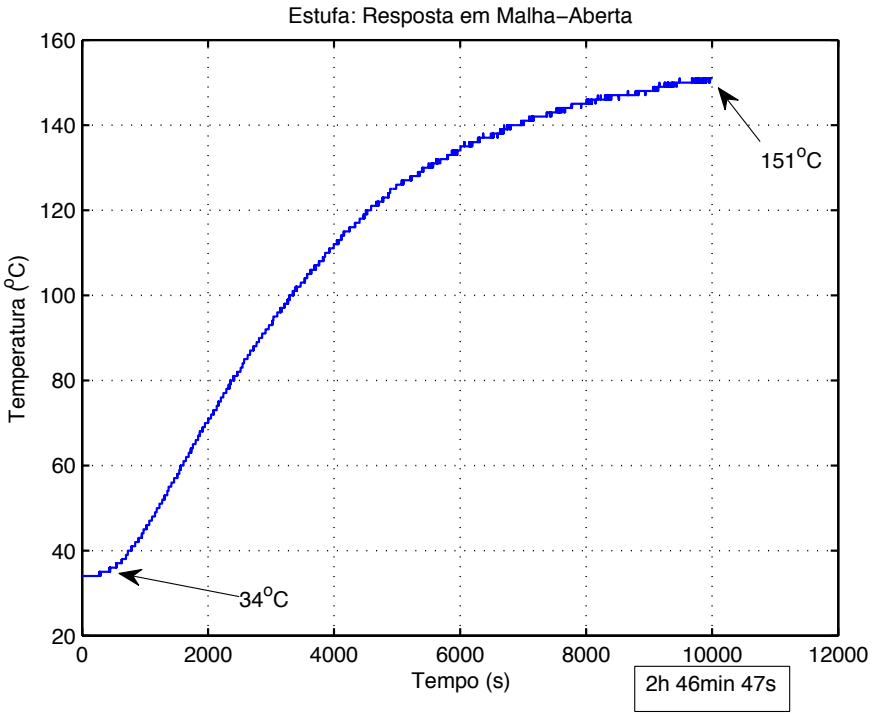
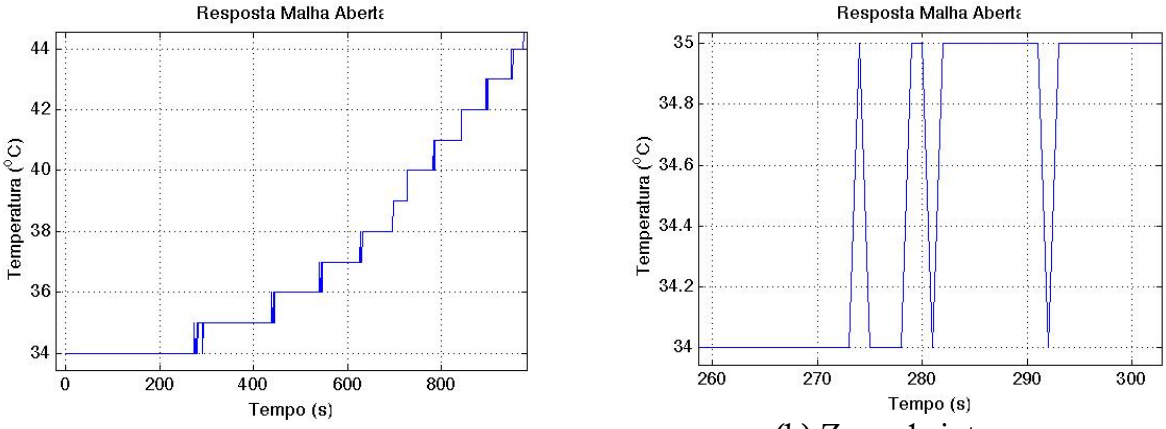


Figura 1. Resposta em malha aberta.

```
>> grid % mostra quadriculado no gráfico
```

A idéia é agora buscar graficamente o atraso de transporte deste sistema, o 'td', realizando zooms na área de interesse – ver próxima figura 2:



(a) Primeiro zoom. (b) Zona de interesse.

Figura 2. Determinando o atraso de transporte deste sistema.

Assim definimos 'td' como sendo:

```
>> td=283;
>> 283/60 % a título de curiosidade, em minutos, este intervalo de tempo
resultaria:
ans =    4.7167 % quase 5 minutos – muito longo...
```

Seguindo a captura dos parâmetros de interesse, as equações que geram a resposta deste sistema (se considerado como de 1ª-ordem são):

$$Y(s) = \frac{K \cdot e^{-t_d s}}{(s-a)}$$ (no domínio frequência) ou:

$$y(t) = K(1 - e^{-a \cdot t}) \qquad (1)$$

A equação (1) é válida se não for considerado o atraso de transporte.

Mas continuando nossa captura de parâmetros, como a idéia é identificar o pólo do sistema ou sua constante de tempo, antes de tentar descobrir o valor de 'a' é mais interessante determinar o ganho estático do sistema, 'K' que possui relação com a variação de temperatura alcançada pelo sistema.

```
>> t_ini=min(estufa1)     % encontra o menor valor valor de temperatura
(temperatura inicial)
t_ini =    34
>> t_max=max(estufa1)     % encontra o maior valor de temperatura (a final).
t_max =   151
>> delta_temp=t_max-t_ini % determina a variação de temperatura.
```

```
delta_temp =   117
```

Este valor permite definir o valor o ganho estático deste sistema, dependendo da forma como se considera a entrada degrau, por exemplo:

Se u = [0 .. 1] então K = 117;

Se u = [0 .. 100%] então K = 1,17;

Como a idéia é futuramente controlar a potencia sendo entregue ao sistema, daqui por diante vamos considerar u (o sinal de controle) como sendo um valor que varia de 0 à 100% e assim K no nosso caso, fica igual à 1,17.

Até este momento já temos os seguintes parâmetros levantados para este sistema:

$$Y(s) = \frac{1.17 \cdot e^{-283 \cdot s}}{(s-a)}$$

E desta forma fica claro observar que agora só está faltando determinar a posição do pólo do sistema, 'a', ou a constante de tempo deste sistema: $\tau = 1/a$.

Notar que a equação (1) não leva em conta o atraso de transporte, então necessitamos definir um novo conjunto de dados que não contenha o período com o atraso no tempo. No MATLAB criamos então um novo vetor de dados, o 'y':

```
>> y=estufa1(283:10007); % separando só a região de interesse (sem o atraso)
>> whos
  Name             Size                Bytes  Class     Attributes
  ans              1x1                     8  double
  estufa1      10007x1                 80056  double
  t                1x10007             80056  double
  td               1x1                     8  double
  u                1x1                     8  double
  y             9725x1                 77800  double
```

Note que o vetor 'y' é menor que 'estufa1' (conjunto de dados original).

Como a idéia aqui é ainda **usar um método numérico** para encontrar o valor do pólo ou constante de tempo a questão agora é aprender a usar uma destas funções já prontas do

MATLAB: a que realiza um ajuste de curva baseado no método dos **mínimos quadrados**, ou a função 'lsqcurvefit(.)'. Um 'help' sobre esta função revela:

```
>> help lsqcurvefit
 LSQCURVEFIT solves non-linear least squares problems.
    LSQCURVEFIT attempts to solve problems of the form:
    min  sum {(FUN(X,XDATA)-YDATA).^2}  where X, XDATA, YDATA and the
     X                                  values returned by FUN can be
                                        vectors or matrices.

    X = LSQCURVEFIT(FUN,X0,XDATA,YDATA) starts at X0 and finds coefficients
    X to best fit the nonlinear functions in FUN to the data YDATA (in the
    least-squares sense). FUN accepts inputs X and XDATA and returns a
    vector (or matrix) of function values F, where F is the same size as
    YDATA, evaluated at X and XDATA. NOTE: FUN should return FUN(X,XDATA)
    and not the sum-of-squares sum((FUN(X,XDATA)-YDATA).^2).
    ((FUN(X,XDATA)-YDATA) is squared and summed implicitly in the
    algorithm.)

    X = LSQCURVEFIT(FUN,X0,XDATA,YDATA,LB,UB) defines a set of lower and
    upper bounds on the design variables, X, so that the solution is in the
    range LB <= X <= UB. Use empty matrices for LB and UB if no bounds
    exist. Set LB(i) = -Inf if X(i) is unbounded below; set UB(i) = Inf if
    X(i) is unbounded above.

    X = LSQCURVEFIT(FUN,X0,XDATA,YDATA,LB,UB,OPTIONS) minimizes with the
    default parameters replaced by values in the structure OPTIONS, an
    argument created with the OPTIMSET function. See OPTIMSET for details.
    Used options are Display, TolX, TolFun, DerivativeCheck, Diagnostics,
    FunValCheck, Jacobian, JacobMult, JacobPattern, LineSearchType,
    LevenbergMarquardt, MaxFunEvals, MaxIter, DiffMinChange and
    DiffMaxChange, LargeScale, MaxPCGIter, PrecondBandWidth, TolPCG,
    PlotFcns, OutputFcn, and TypicalX. Use the Jacobian option to specify
    that FUN also returns a second output argument J that is the Jacobian
    matrix at the point X. If FUN returns a vector F of m components when X
    has length n, then J is an m-by-n matrix where J(i,j) is the partial
    derivative of F(i) with respect to x(j). (Note that the Jacobian J is
    the transpose of the gradient of F.)

    X = LSQCURVEFIT(PROBLEM) solves the non-linear least squares problem
    defined in PROBLEM. PROBLEM is a structure with the function FUN in
    PROBLEM.objective, the start point in PROBLEM.x0, the 'xdata' in
    PROBLEM.xdata, the 'ydata' in PROBLEM.ydata, the lower bounds in
    PROBLEM.lb, the upper bounds in PROBLEM.ub, the options structure in
    PROBLEM.options, and solver name 'lsqcurvefit' in PROBLEM.solver. Use
    this syntax to solve at the command line a problem exported from
    OPTIMTOOL. The structure PROBLEM must have all the fields.

    [X,RESNORM] = LSQCURVEFIT(FUN,X0,XDATA,YDATA,...) returns the value of
    the squared 2-norm of the residual at X: sum {(FUN(X,XDATA)-YDATA).^2}.

    [X,RESNORM,RESIDUAL] = LSQCURVEFIT(FUN,X0,...) returns the value of
    residual, FUN(X,XDATA)-YDATA, at the solution X.

    [X,RESNORM,RESIDUAL,EXITFLAG] = LSQCURVEFIT(FUN,X0,XDATA,YDATA,...)
    returns an EXITFLAG that describes the exit condition of LSQCURVEFIT.
```

```
    Possible values of EXITFLAG and the corresponding exit conditions are
    listed below. See the documentation for a complete description.

      1  LSQCURVEFIT converged to a solution.
      2  Change in X too small.
      3  Change in RESNORM too small.
      4  Computed search direction too small.
      0  Too many function evaluations or iterations.
     -1  Stopped by output/plot function.
     -2  Bounds are inconsistent.
     -3  Regularization parameter too large (Levenberg-Marquardt).
     -4  Line search failed.

    [X,RESNORM,RESIDUAL,EXITFLAG,OUTPUT] = LSQCURVEFIT(FUN,X0,XDATA,
    YDATA,...) returns a structure OUTPUT with the number of iterations
    taken in OUTPUT.iterations, the number of function evaluations in
    OUTPUT.funcCount, the algorithm used in OUTPUT.algorithm, the number of
    CG iterations (if used) in OUTPUT.cgiterations, the first-order
    optimality (if used) in OUTPUT.firstorderopt, and the exit message in
    OUTPUT.message.

    [X,RESNORM,RESIDUAL,EXITFLAG,OUTPUT,LAMBDA] = LSQCURVEFIT(FUN,X0,XDATA,
    YDATA,...) returns the set of Lagrangian multipliers, LAMBDA, at the
    solution: LAMBDA.lower for LB and LAMBDA.upper for UB.

    [X,RESNORM,RESIDUAL,EXITFLAG,OUTPUT,LAMBDA,JACOBIAN] = LSQCURVEFIT(FUN,
    X0,XDATA,YDATA,...) returns the Jacobian of FUN at X.

    Examples
      FUN can be specified using @:
         xdata = [5;4;6];          % example xdata
         ydata = 3*sin([5;4;6])+6; % example ydata
         x = lsqcurvefit(@myfun, [2 7], xdata, ydata)

    where myfun is a MATLAB function such as:

        function F = myfun(x,xdata)
        F = x(1)*sin(xdata)+x(2);

    FUN can also be an anonymous function:
        x = lsqcurvefit(@(x,xdata) x(1)*sin(xdata)+x(2),[2 7],xdata,ydata)

    If FUN is parameterized, you can use anonymous functions to capture the
    problem-dependent parameters. Suppose you want to solve the
    curve-fitting problem given in the function myfun, which is
    parameterized by its second argument c. Here myfun is an M-file
    function such as

        function F = myfun(x,xdata,c)
        F = x(1)*exp(c*xdata)+x(2);

    To solve the curve-fitting problem for a specific value of c, first
    assign the value to c. Then create a two-argument anonymous function
    that captures that value of c and calls myfun with three arguments.
    Finally, pass this anonymous function to LSQCURVEFIT:

        xdata = [3; 1; 4];          % example xdata
```

```
        ydata = 6*exp(-1.5*xdata)+3; % example ydata
        c = -1.5;                    % define parameter
        x = lsqcurvefit(@(x,xdata) myfun(x,xdata,c),[5;1],xdata,ydata)

    See also optimset, lsqnonlin, fsolve, @, inline.

    Reference page in Help browser
       doc lsqcurvefit
```

Note que a função 'lsqcurvefit' exige a programação de uma função que reflete a curva buscada. Esta função deve trabalhar basicamente com 2 argumentos de entrada. O primeiro seria um vetor relacionado com os parâmetros que definem o comportamento da curva. No nosso caso, seria um único parâmetro que falta determinar, ou seja, o pólo do sistema ou sua constante de tempo. O segundo argumento de entrada se refere à variável x ou no nosso caso, a própria variável tempo. Levando em conta a equação (1), o código desta função, que chamaremos de 'resp1ordem' é mostrada a seguir:

```
function y = resp1ordem (p, tempo)
    % y = valor_inicial + K*(1 - exp(-a*tempo));
    % y = K*(1 - exp(-a*tempo));
    y = (1 - exp(-p(1)*tempo));
```

Listagem 1. Código fonte de 'resp1ordem.m'.

Notar que neste caso, o primeiro argumento de entrada, o vetor 'p' possui apenas 1 elemento que é a posição do própio pólo do sistema de 1ª-ordem. Se fosse necessário capturar mais parâmetros, o vetor 'p' assumiria uma dimensão maior que 1.

Notar também que da forma como programamos a função 'resp1ordem', ela não permite que se entre com um parâmetro "extra" como próprio ganho estático da planta, 'K'. Isto foi feito para evitar que a função 'lsqcurvefit' tenha que determinar também este parâmetro. Não é necessário que esta função determine o ganho estático da planta, o 'K' porque este já foi facilmente determinado. O que falta determinar é a localização do pólo. Por este motivo que o vetor 'p' usado para informar os parâmetros da função 'resp1ordem' possui apenas dimensão 1, já que só queremos descobrir um parâmetro que é a localização do pólo do sistema.

Como então a função 'resp1ordem' não contempla o fator ganho estático da planta, isto significa que sua saída, 'y', fica limitado à faixa "normalizada" de [0..1]. Então antes de aplicar o conjunto de dados reais 'y' à função 'lsqcurvefit' se faz necessário escalonar estes dados e desta forma, criamos um segundo vetor chamado 'y2':

```
>> y2=(y-t_ini)/delta_temp; % escalonando dados para faixa de [0..1]
```

Note que (como seria de se esperar), a função 'lsqcurvefit' exige um par de dados de entrada relacionado com o vetor 'x' e 'y' da curva de entrada (ou as variáveis tempo x temperatura no caso do problema sendo tradado). Então temos criar um segundo vetor tempo com a mesma dimensão (quantidade de pontos) que o vetor 'y2' criado anteriormente:

```
>> u2=length(y)
u2 =        9725

>> t2=0:u2-1;
>> whos
  Name                Size              Bytes  Class     Attributes
  estufa1         10007x1               80056  double
  t                   1x10007           80056  double
  t2                  1x9725            77800  double
  t_ini               1x1                   8  double
  t_max               1x1                   8  double
  td                  1x1                   8  double
  u                   1x1                   8  double
  u2                  1x1                   8  double
  y                  9725x1             77800  double
  y2                 9725x1             77800  double
```

Podemos graficar este novo par de dados: t2 x y2 para verificar se está tudo conforme até aqui – ver figura 3.

```
>> plot(t2,y2)
```

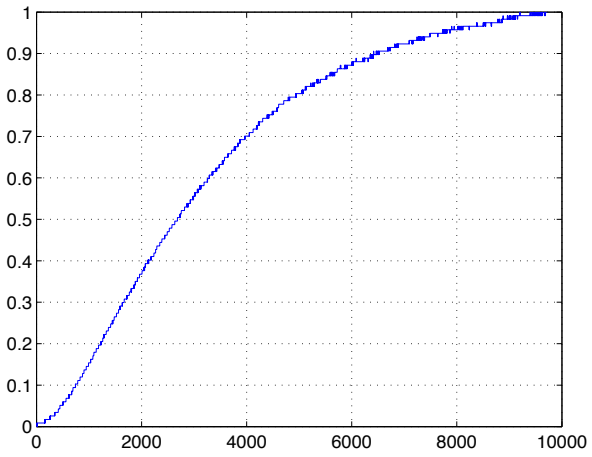


Figura 3. Resposta do sistema sem o atraso de transporte e escalonado para faixa de saída entre [0..1].

Notar ainda que a função 'lsqcurvefit' exige como segundo parâmetro de entrada, um 'chute inicial'. Este valor inicial pode ser facilmente obtido lembrando-se que a constante de tempo de um sistema de primeira ordem ocorre quando a saída do sistema alcança 63% do seu valor de regime permanente. O instante de tempo a partir do qual a saída do sistema ultrapassa os 63% do seu valor final pode se facilmente buscado através da função 'find' que retorna as posições (não os valores) em que um vetor respeita a condição buscada. Então:

```
>> t_aux=find(y2>.63); % busca os instantes em que y2(t) > 0,63
>> t_aux(1)           % Só nos interessa o primeiro valor encontrado
ans =
        3476          % Então o chute inicial para a = 3476 (s^-1)
>> t_aux(1)/60
ans =
   57.9333            % Ou a contante de tempo seria de quase 1 minuto
```

Finalmente então podemos usar a função 'lsqcurvefit':

```
>> a=lsqcurvefit('resp1ordem', 1/3476, t2, y2)
??? Error using ==> lsqcurvefit at 253
Function value and YDATA sizes are incommensurate.
```

O erro acusado aqui se deve a um simples problema de "incompatibilidade" entre os valores 't2' e 'y2' de entrada. Um deles possui dimensão 1x9725 e o outro 9725x1:

```
>> whos
  Name                Size                Bytes  Class     Attributes

  t2                  1x9725              77800  double
  y2                9725x1                77800  double
```

Esta "incompatibilidade" pode ser facilmente corrigida realizando a transposta do vetor 't2':

```
>> t2=t2';
>> a=lsqcurvefit('resp1ordem', 1/3476, t2, y2)

Local minimum possible.

lsqcurvefit stopped because the size of the current step is less than
the default value of the step size tolerance.

<stopping criteria details>

a =
   2.9671e-04
```

Executando a função 'lsqcurvefit' encontramos que o pólo está localizado em a= 2,9671x10^{-4} (s^{-1}) ou:

```
>> tau=1/a
tau =
   3.3703e+03      % ou seja, 3370,3 segundos ou…

>> tau/60
ans =
   56.1718         % quase 1 hora para a constante de tempo.
```

Temos então finalmente o modelo de 1ª-ordem para este sistema:

```
>> num=1.17*a      % compensa o valor final de regime permanente
num =
   3.4715e-04
```

ou seja:

$$Y(s) = \frac{0{,}00034715 \cdot e^{-283s}}{(s + 0{,}00029671)} \qquad (2)$$

Podemos comprovar o resultado da aproximação realizada, gerando gráficos que comparam a curva real com a simulada:

```
>> y2_teste=resp1ordem(a,t2);
>> plot(t2,y2,'b:', t2,y2_teste,'m-')
```

O gráfico resultante pode ser apreciado na figura 4.

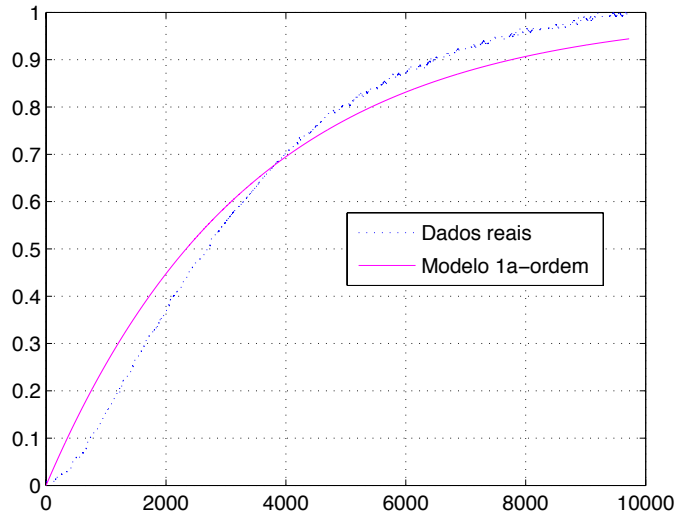


Figura 4. Aproximação final.

Percebe-se pela figura 4 que o modelo encontrado não se aproxima muito da curva real. A justificativa para este caso é que a curva real de resposta do sistema corresponde a um sistema de 2ª-ordem ou superior e portanto, neste caso, a aproximação da resposta deste sistema para um de 1ª-ordem não gera um bom resultado.

A curva da figura 4 também pode ser obtida usando-se a função 'step' do MATLAB. Esta função permite ingressar até um valor para amplitude do degrau de entrada. Mas para tanto necessitamos antes informar ao MATLAB a equação (2) desconsiderando o atraso de transporte:

```
>> num=a;              % para tornar valor em regime permanente = 1
>> den=[1 a];
>> printsys(num,den)
num/den =
     0.00029671
   --------------
   s + 0.00029671
>> step(num,den)
>> hold on
>> plot(t2,y2,'b:', t2,y2_teste,'m-')
>> legend('Step','Real', 'Modelado')
```

Executando a função 'step' sobre a função obtemos o gráfico mostrado na figura 5.

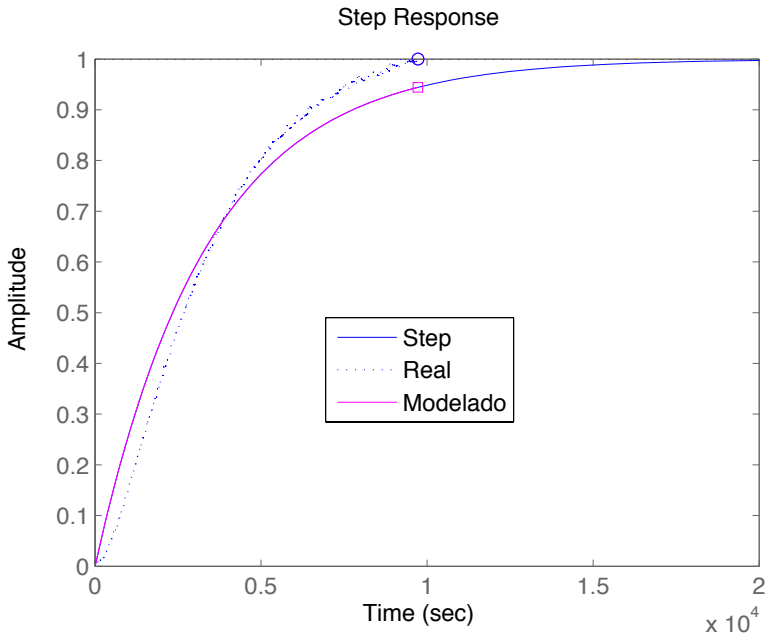


Figura 5. Resposta ao degrau para o sistema sem considerar o atraso de transporte.

Para gerar dados que podem ser sobrepostos à curva original se faz necessário fazer:

```
>> t3=0:20e3;     % gera novo vetor tempo (período de tempo superior)
>> y3=t_ini+100.*step(num,den,t3);       % estima resposta com base no modelo
levantado – usa a função step para calcular a resposta
>> plot(t3,y3,'m-', t2,y,'bx:'); grid    % gera o novo gráfico comparativo
```

O gráfico resultante aparece na figura 6.

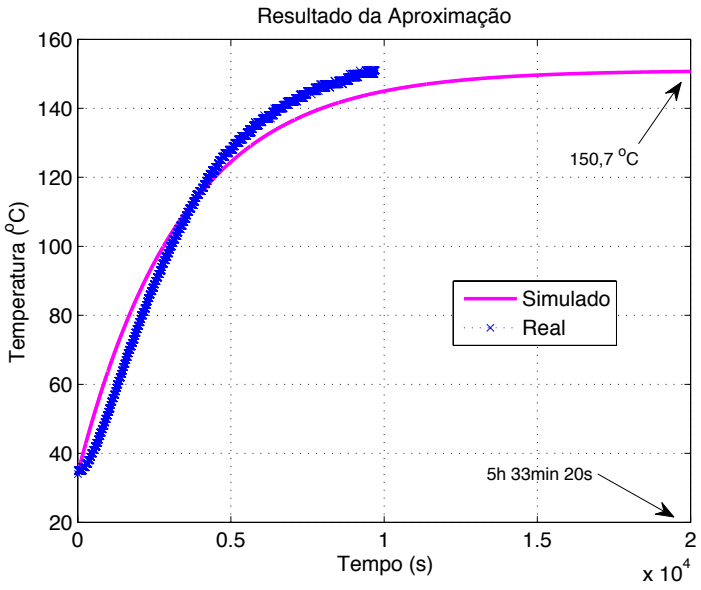


Figura 6. Resposta modelada sem considerar o atraso de transporte.

Outro gráfico (figura 7) mais completo pode ser realizado levando em conta o atraso de transporte:

```
>> new_t3=[0:283-1];
>> new_t3=[new_t3 t3+283];
>> new_y3=34*ones(1,283);
>> new_y3=[new_y3(1,:) y3(1,:)];
>> whos
  Name               Size                Bytes  Class     Attributes

  estufa1        10007x1                 80056  double
  new_t3             1x20284            162272  double
  new_y3             1x20284            162272  double
  t                  1x10007             80056  double
>> plot(t,estufa1,'b:', new_t3,new_y3,'m-')
>> xlabel('Tempo (s)');
>> ylabel('Temperatura (^oC)')
>> title('Resposta Real x Simulada')
>> legend('Real', 'Simulado')
>> grid
```

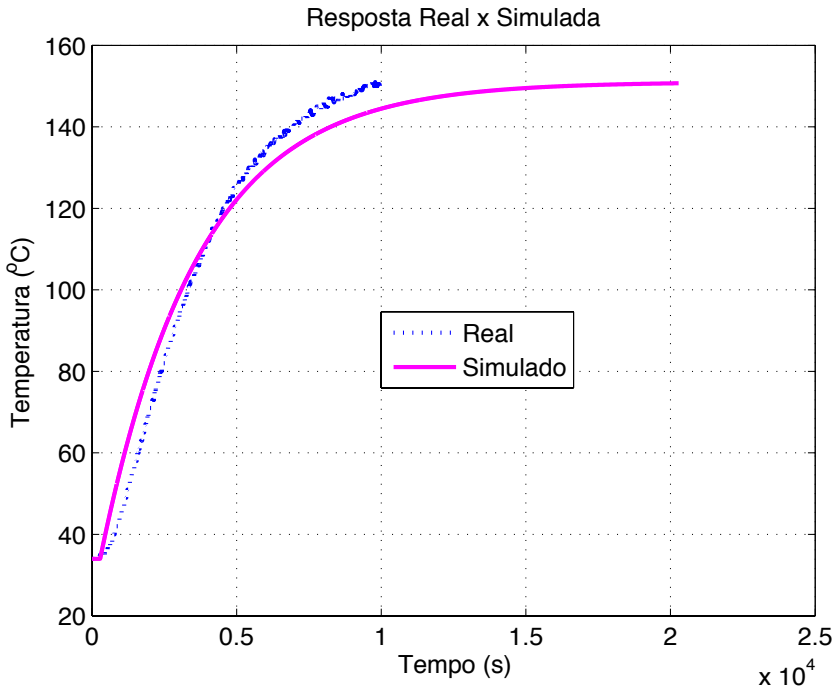


Figura 7. Resultado final da curva real x simulada.

Problema Proposto:

Para este sistema seria mais interessante **levantar um modelo de segunda ordem** com fator de amortecimento, $\zeta > 1$ e depois criando outra função como 'resp2ordem' para trabalhar com a função 'lsqcurvefit'. Neste caso, 2 parâmetros seriam buscados: os 2 pólos reais do sistema:

```
>> [a b]=lsqcurvefit('resp2ordem', [1/3476 1/1000], t2, y2)
```