Modelling and simulation of a tubular heater via spatial discretisation (finite elements)

This code was successfully executed in Matlab **R2022a**

Presentations in video: http://personales.upv.es/asala/YT/V/tubulFE1EN.html ,

http://personales.upv.es/asala/YT/V/tubulFEsim1EN.html , http://personales.upv.es/asala/YT/V/tubulFEsim2EN.html .

Objectives: Modelling with discrete (finite, i.e., not *infinitesimal*) elements a tubular heater/exchanger to approximate the Partial Differential Equation with a finite-order set of ODEs (state-space). Simulation of the resulting ODE.

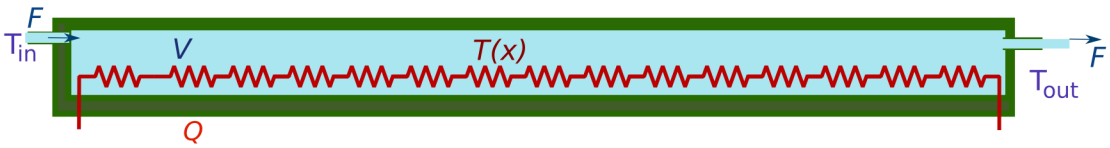


Table of Contents

Heater/exchanger data (constant parameters)

```
Ltot=1; %total length
S=0.01; %cross-section
barkappa=350; %heat conduction per unit length
```

```
Vtot=Ltot*S; %m^3 total volume
Ce=4.18e3; %J/Kg/K Water heat capacity
rho=1000; %Kg/m^3 density (water)

%stored to recover it after symbolic manipulations with the same name.
problem.Ce=Ce;problem.rho=rho;

F_pf=0.0002; %nominal flow operating point.
```

1.- Modelling, quic recap.

Constant, zero ambient outside temperature assumed (equiv., we work in increments over such temperature, so it does not appear in the equations).

1.1- Partial differential equation

$$\frac{\partial T}{\partial t} = -\frac{1}{S} F \frac{\partial T}{\partial x} - \frac{\bar{\kappa}}{S\rho C_e} T + \frac{\bar{Q}}{S\rho C_e}$$

$\bar{Q}$ is heating power per unit length. $T(x,t)$ is our sought solution. Probably, we'll just need $T_{out} := T(L,t)$ in our applications.

1.1.1- PDE solution for constant flow F

Developed in other materials via Laplace's method. If we denote $\phi = L/v$, we have a transfer function matrix in the form

$$\mathbb{T}_{out}(s) = \begin{pmatrix} (1 - e^{-\phi \cdot s} e^{-a\phi}) \dfrac{b}{s+a} & e^{-\phi \cdot s} e^{-a\phi} \end{pmatrix} \cdot \begin{pmatrix} \bar{\mathbb{Q}}(s) \\ \mathbb{T}_{in}(s) \end{pmatrix}$$

Parameters are $a = \dfrac{\bar{\kappa}}{S\rho C_e}$, $b = \dfrac{1}{S\rho C_e}$, $v = \dfrac{F}{S}$, $\phi = \dfrac{L}{v} = \dfrac{V}{F}$ (time delay); thus, $a\phi = \dfrac{\bar{\kappa} L}{F\rho C_e} = \dfrac{\bar{\kappa} L}{vS\rho C_e}$ (spatial exponential).

1.1.2- Steady-state model (static, no dynamics)

- Setting, $\dfrac{\partial T}{\partial t} = 0$, denoting the solution as $T_{eq}(x)$, we have $\dfrac{\partial T_{eq}}{\partial x} = -\dfrac{\bar{\kappa}}{F\rho C_e} T_{eq} + \dfrac{1}{F\rho C_e} \bar{Q}_{eq}(x)$, so,

assuming uniform heating along all the pipe, defining $\lambda = \dfrac{\bar{\kappa}}{F\rho C_e}$, it results in:

$$T_{eq}(x) = T_{in} \cdot e^{-\lambda \cdot x} + (1 - e^{-\lambda \cdot x})\bar{\kappa}^{-1} \cdot \bar{Q}_{eq}$$

$$T_{out} = T_{eq}(L) = T_{in} \cdot e^{-\lambda \cdot L} + (1 - e^{-\lambda \cdot L})\bar{\kappa}^{-1} \cdot \bar{Q}_{eq}$$

Note that $\lambda L = a\phi$. Of course, the same is obtained if we let $s \to 0$ in $\mathbb{T}_{out}(s)$ in $1.1.1.

As coefficients depend on flow, we will define some functions with flow and position as arguments:

```
lambda=@(F) barkappa/rho/Ce./F;
gainTin=@(F,L) exp(-lambda(F)*L); %lo que multiplica a Tin
gainQ=@(F,L) 1e3*(1-gainTin(F,L))/barkappa/L; %lo que multiplica a Q en kW
```

In our heater with the above-given numerical coefficients, we have:

```
ganTin_pf=gainTin(F_pf,Ltot)
```

```
ganTin_pf = 0.6579
```

```
gainQ(F_pf,Ltot) %°C/kW
```

```
ans = 0.9773
```

1.1.3- Steady state mean temperature, as a function of inlet and outlet temperatures

As above discused, we can prove that $T_{eq}(x) = Ae^{-\lambda x} + B$, with $A = \frac{T_{in} - T_{out}}{1 - e^{-\lambda L}}, \quad B = \frac{T_{out} - e^{-\lambda L}T_{in}}{1 - e^{-\lambda L}},$

and $\lambda = \frac{\bar{\kappa}}{F\rho C_e}$.

Mean temperature $\bar{T} := \frac{1}{L}\int_0^L T_{eq}(x)\,dx$ to be used in energy balances so that at least in steady state it replicates the PDE solution, is:

$$\bar{T} = (1 - \beta)T_{in} + \beta T_{out}$$

with:

$$\beta(\lambda L) = \frac{\lambda L - (1 - e^{-\lambda L})}{\lambda L(1 - e^{-\lambda L})}$$

```
beta1ele=@(F,L) (lambda(F)*L-1+gainTin(F,L))/(lambda(F)*L*(1-gainTin(F,L)));
beta_1element_exponential_profile=beta1ele(F_pf,Ltot) %for our actual numerical values
```

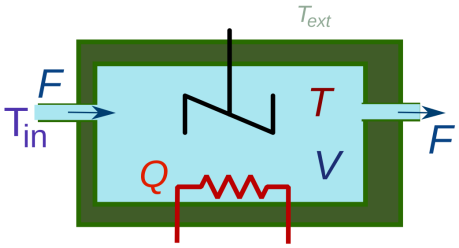
```
beta_1element_exponential_profile = 0.5348
```

Actually, it is not too different from 0.5 (linear temperature profile) for the given data.

Using this expression for $\bar{T}$ introduces non-minimum-phase elements in the model... not too "natural" (inverse response) in step responses, but they approximate (in a "better" way than other options) the delay at low frequency.

1.2- First-principle 'one element' lumped-parameter model

A 1st order tank with a heating resistor can be modelled as:



Inputs:

```
syms F real %input flow
syms Tin real %inlet temp.
syms Q real %heating power (resistor)
```

Constant parameters:

```
syms V real %volume
syms rho real %density
syms kappa real %conduction heat loss
syms Ce real %heat capacity
```

State variable and its derivative:

```
syms T dTdt real %mean temperature
```

Temperature profile adjustment $\bar{T} = \beta T_{out} + (1-\beta)T_{in}$

```
syms beta real  %0.5 lineal, 1 Tout=T...
```

Energy balance on the "mean" temperature $\bar{T}$ is the state equation (we will use just T instead of $\bar{T}$ in the symbolic toolbox code):

```
Model=  V*rho*Ce*dTdt == 1/beta*F*rho*Ce*(Tin - T) - kappa*T + Q;
```

Normalised internal representation:

```
dTdt_sym=simplify(solve(Model,dTdt),10)
```

```
dTdt_sym =
```

$$\frac{Q - T\,\kappa}{\mathrm{Ce}\,V\,\rho} - \frac{F\,(T - \mathrm{Tin})}{V\,\beta}$$

In the form $\frac{dT}{dt} = \frac{F}{V\beta}(T - T_{in}) - aT + bQ$, with $a = \frac{\kappa}{C_e V \rho}$, $b = \frac{1}{V\rho C_e}$.

If input were heat "per unit length" $\bar{Q} = Q/L$ and thermal leaks were also given per unit length, $\bar{\kappa} = \kappa/L$,

we would express the equations as:

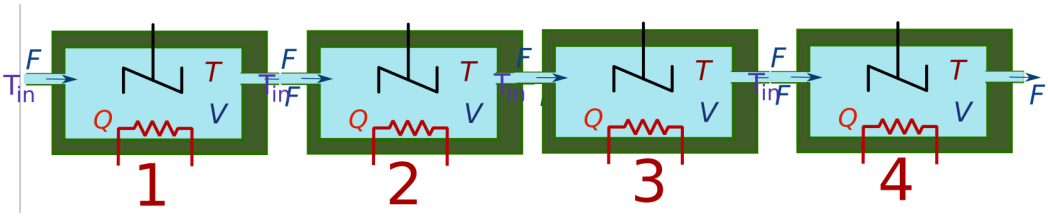
$$\frac{dT}{dt} = \frac{F}{V\beta}(T - T_{in}) - \bar{a} \cdot T + \bar{b} \cdot \bar{Q}, \text{ with } \bar{a} = \frac{\bar{\kappa} L}{C_e V \rho} = \frac{\bar{\kappa}}{C_e S \rho},\ \bar{b} = \frac{1}{S\rho C_e}.$$

Output equation will be typed later on, see below.

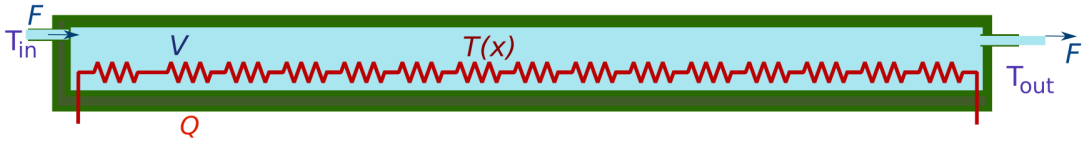
The model is nonlinear, due to the product FT and FT_{in} when F is not constant.

2.- Many-element model

It's a sort of "serial" interconnect of N elements:



in order to model the whole tubular heater:



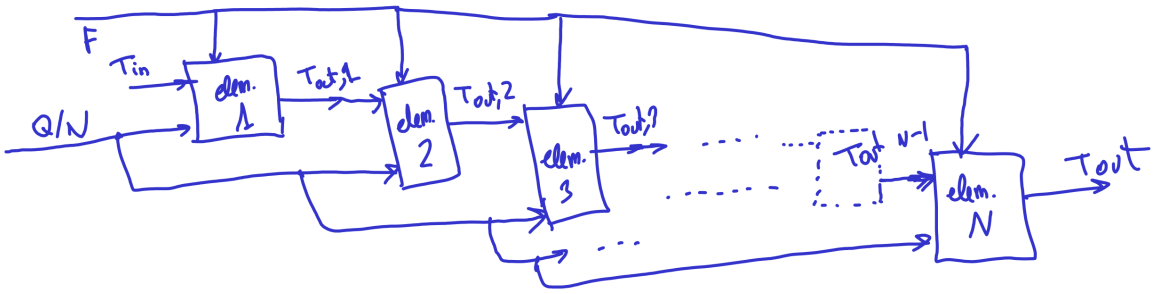
Each element has three inputs (flow F, heating power per element Q_i, element inlet temperature $T_{in,i}$), one state T_i, one output $T_{out,i}$.

Heating power (uniformly distributed), length and volume will be split between the elements:

We have: $F_i = F_{i-1} = F$ (no leaks, continuity), $V_i = V/N$, $Q_i = Q/N$, $\kappa_i = \kappa/N$.

Inlet temperature $T_{in,i}$ to one element will be the output temperature from the one at its left side $T_{in,i} = T_{out,i-1}$ except for $i = 1$, where, of course, $T_{in,1} \equiv T_{in}$ (boundary condition in PDE jargon, but just an input to the system in our ODE simulation below).

Our goal is writing the dynamics of each T_i; also, the heater's outlet temperature will be $T_{out,N}$.



For each of the elements, we will build a first-order lumped-parameter state and output equations as above discussed.

Building the numerical N- element model

```
%We recover numerical values for symbolic stuff
Ce=problem.Ce;rho=problem.rho;
```

Constant parameters for each element:

```
N=200; %Number of elements
V=Vtot/N; %volume of each element
Qporelem=Q/N; %heating power per element (uniform distribution assumed)
kappa=barkappa*Ltot/N; %heat loss coefficient per element
L=Ltot/N; %element length
beta_exact_exponential_profile=beta1ele(F_pf,L) %exact value of beta from exponential f
```

```
beta_exact_exponential_profile = 0.5002
```

```
%beta=beta_exact_exponential_profile;
beta=0.55; %we may change it on purpose, it may avoid unnatural oscillations.
```

With many elements, β should not be that important, as $T_{out} \approx T_{in} \approx \overline{T}$ if it is very thin. Nevertheless, it somehow affects the delay approximation and the simulation behaviour.

State and output equations for each element:

```
dTdt_elem=subs(dTdt_sym,Q,Qporelem);
StateEq1elem=eval(dTdt_elem); %we replace V, Q, kappa, etc. by numerical values in work
vpa(StateEq1elem,4)%for display purposes
```

ans = $2.392\text{e-}5\,Q - 0.008373\,T - 36360.0\,F\,(T - 1.0\,\mathrm{Tin})$

```
OutputEq1elem=T/beta-(1/beta-1)*Tin;
vpa(OutputEq1elem,4) %for display purposes
```

ans = $1.818\,T - 0.8182\,\mathrm{Tin}$

```
Tvector=sym('T',[N 1],'real');
%code below just to show the vector of state variables
if N>=20 %we show just a few at the beginning and at the end
    Tvector([1:10 (N-10):N])'
else %we show everything
    Tvector'
end
```

ans = $\begin{pmatrix} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 & T_{10} & T_{190} & T_{191} & T_{192} & T_{193} & T_{194} & T_{195} & T_{196} & T_{197} & T_{198} & T_{199} & T \end{pmatrix}$

Building the whole model:

```
tic
StateEqsEachElement=sym(zeros(N,1));
OutputEqsEachElement=sym(zeros(N,1));
%First eleemnt's equations are different:
StateEqsEachElement(1)=subs(StateEq1elem,{T},{Tvector(1)});
OutputEqsEachElement(1)=subs(OutputEq1elem,{T},{Tvector(1)});
%second element onwards:
for i=2:N
    StateEqsEachElement(i)=subs(StateEq1elem,{T,Tin},{Tvector(i),OutputEqsEachElement(i
    OutputEqsEachElement(i)=subs(OutputEq1elem,{T,Tin},{Tvector(i),OutputEqsEachElement
end
toc
```

```
Elapsed time is 1.389534 seconds.
```

```
size(StateEqsEachElement)
```

```
ans = 1×2
   200     1
```

These N equations are the ODE model, of order N, that approximates the tubular heater PDE (infinite order, in theory).

Let us see the first five state equations:

```
vpa(simplify(StateEqsEachElement(1:5),20),3)
```

ans =

$$\begin{pmatrix} \text{2.39e-5}\,Q - 0.00837\,T_1 - \text{3.64e+4}\,F\,\,(T_1 - 1.0\,\text{Tin}) \\ \text{2.39e-5}\,Q - 0.00837\,T_2 - \text{3.64e+4}\,F\,\,(T_2 - 1.82\,T_1 + 0.818\,\text{Tin}) \\ \text{2.39e-5}\,Q - 0.00837\,T_3 - \text{3.64e+4}\,F\,\,(1.49\,T_1 - 1.82\,T_2 + T_3 - 0.669\,\text{Tin}) \\ \text{2.39e-5}\,Q - 0.00837\,T_4 - \text{3.64e+4}\,F\,\,(1.49\,T_2 - 1.22\,T_1 - 1.82\,T_3 + T_4 + 0.548\,\text{Tin}) \\ \text{2.39e-5}\,Q - 0.00837\,T_5 - \text{3.64e+4}\,F\,\,(0.996\,T_1 - 1.22\,T_2 + 1.49\,T_3 - 1.82\,T_4 + T_5 - 0.448\,\text{Tin}) \end{pmatrix}$$

And the output equations for the first five finite elements (of course, unneeded in the final setup if only the outlet temperature is of interest)

```
vpa(simplify(OutputEqsEachElement(1:5),20),3)
```

```
ans =
```

$$\begin{pmatrix} 1.82\,T_1 - 0.818\,\text{Tin} \\ 1.82\,T_2 - 1.49\,T_1 + 0.669\,\text{Tin} \\ 1.22\,T_1 - 1.49\,T_2 + 1.82\,T_3 - 0.548\,\text{Tin} \\ 1.22\,T_2 - 0.996\,T_1 - 1.49\,T_3 + 1.82\,T_4 + 0.448\,\text{Tin} \\ 0.815\,T_1 - 0.996\,T_2 + 1.22\,T_3 - 1.49\,T_4 + 1.82\,T_5 - 0.367\,\text{Tin} \end{pmatrix}$$

Compilation (translation) from symbolic toolbox object to numeric function:

Model equations are symbolic. Let us compile them to numerical functions for the ODE solvers:

```
StateEqsFE_num=matlabFunction(StateEqsEachElement,'Vars',{Tvector,F,Q,Tin})
```

```
StateEqsFE_num = function_handle with value:

    @(in1,F,Q,Tin)[Q./4.18e+4-in1(1,:).*(7.0./8.36e+2)-F.*(in1(1,:)-Tin).*3.636363636363636e+4;Q./4.18e+4-
```

Arguments are the state vector "in1" (vector of N temperatures), flow F, heating power Q and inlet temperature T_{in}. This models f for a model $\dot{x} = f(x,u)$, returning the N derivatives of each of the elements' mean temperatures.

```
tmp=StateEqsFE_num(randn(N,1),2,1,12);
size(tmp) %N floating-point numbers
```

```
ans = 1×2
   200     1
```

Non-linear simulation

We will set up arbitrary steps in the input variables to carry out our simulations:

```
Q_t=@(t) 8000*(t>2500); %heating power input temporal profile
F_t=@(t) F_pf+0.0001*sign(sin(t/360+0.5)); %Flow input temporal profile
Tin_t=@(t) 15+5*sign(sin(pi*t/1700)); %Inlet temperature temporal profile
```

*Actually, step changes have high spatial/time frequency components that exacerbate differences between models. In applications with a smoother input/heat temperature profiles the differences between the various modelling options would be be less noticeable.

Model with no dynamics

As a first step, we will simulate the steady-state model:

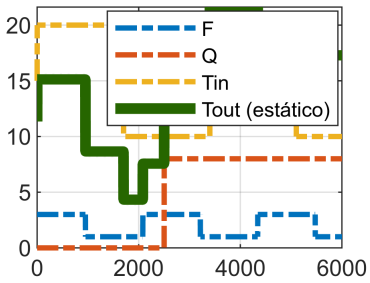
$$T_{out} = T_{eq}(L) = T_{in} \cdot e^{-\lambda(F)\cdot L} + (1 - e^{-\lambda(F)\cdot L})\bar{\kappa}^{-1} \cdot \bar{Q}_{eq} := T_{out}(F, \bar{Q}_{eq}, T_{in})$$

Indeed, it will be the most commonly used in applications where washout time is short in comparison to other process time constant this heater is a subsystem of. The resulting simulation (well, this is not an actual "simulation" understood as "numerical integration", as there are no ODEs to integrate here) is:

```
T_end=6000;
TT=(0:2:T_end)'; %times for the static model (no ODE integration, just evaluation of a
tic
Tsol_estatico=gainTin(F_t(TT),Ltot).*Tin_t(TT)+gainQ(F_t(TT),Ltot).*Q_t(TT)/1000;
toc
```

```
Elapsed time is 0.002305 seconds.
```

```
plot(TT,[F_t(TT)*1e4 Q_t(TT)*1e-3 Tin_t(TT)], LineWidth=2, LineStyle='-.'), grid on, ho
plot(TT, Tsol_estatico, LineWidth=4,Color=[0.15 0.4 0]), hold off
legend("F","Q","Tin","Tout (estático)",Location="best")
```



Dynamic model simulation

We carry out numerical integration with the code below:

```
options = odeset('RelTol',1e-5,'AbsTol',1e-4) ;
ModeloFENum=@(t,T) StateEqsFE_num(T,F_t(t),Q_t(t),Tin_t(t));
tic
[timeV,Tsol]=ode15s(ModeloFENum,[0 T_end],zeros(N,1),options);
toc
```

```
Elapsed time is 4.082201 seconds.
```

```
length(timeV)
```

```
ans = 5366
```

```
%tic
%[time45,Tsol45]=ode45(ModeloFENum,[0 T_end],zeros(N,1),options);
%toc
length(time45)
```

```
ans = 387933
```

It seems that **`ode15s`** is more efficient than **`ode45`** for this particular problem with the same tolerances. There are other problems where ode45 is better, these issues are out of the scope of this paper (I am not an expert in the topic).

As a result, Tsol contains all N state variables in all simulation times (the mean temperatures in each element). The output temperature of each element will be governed by the output equation, which we need to "compile" to numerical from symbolic representation:

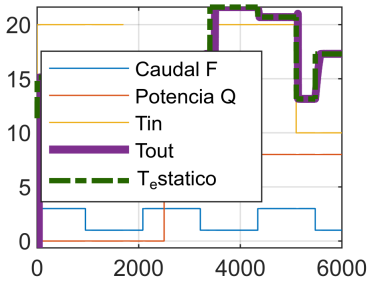
```
ES_num=matlabFunction(OutputEq1elem)
```

```
ES_num = function_handle with value:
    @(T,Tin)T.*(2.0e+1./1.1e+1)-Tin.*(9.0./1.1e+1)
```

```
if(N>1)
    Tsol_salida=ES_num(Tsol(:,end),Tsol(:,end-1));
else
 Tsol_salida=ES_num(Tsol(:,end),Tin_t(timeV));
end
```

Graphical representation of the obtained solution:

```
plot(timeV,[F_t(timeV)*1e4 Q_t(timeV)*1e-3 Tin_t(timeV)]), grid on, hold on
plot(timeV,Tsol_salida,LineWidth=3), grid on,
plot(TT,Tsol_estatico,'-.',LineWidth=2,Color=[0.15 0.4 0])
hold off
legend("Caudal F","Potencia Q","Tin","Tout","T_estatico",Location="best")
```



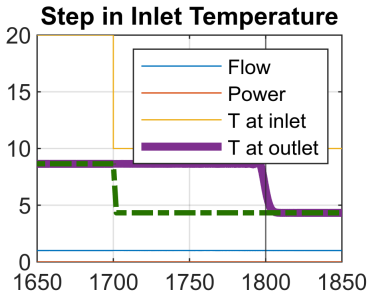
Zoom fragment 1: no heat increment, inlet temperature increment

```
plot(timeV,[F_t(timeV)*1e4 Q_t(timeV)*1e-3 Tin_t(timeV)]), grid on, hold on
%plot(tiempo,Tsol(:,end),LineWidth=3), grid on,
```

```
plot(timeV,Tsol_salida,LineWidth=3), grid on
plot(TT,Tsol_estatico,"-.",LineWidth=2,Color=[0.15 0.45 0])
%la solución exacta es un retraso (no estamos dando potencia), de
%longitud/velocidad=volumen/caudal
truedelay=S*Ltot/(F_pf-0.0001)
```

```
truedelay = 100
```

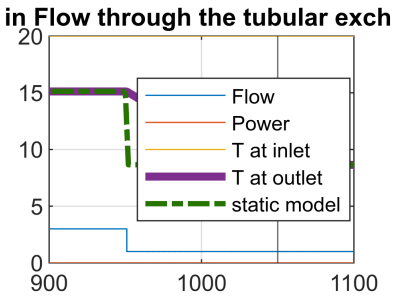
```
xline(1700+truedelay)
hold off
xlim([1650 1850])
legend("Flow","Power","T at inlet","T at outlet")
title("Step in Inlet Temperature")
```



The exact PDE solution (transport delay) would be a step in output temperature (with some gain, not the same amplitude as input one due to losses)

Zoom fragment 2: flow step

```
plot(timeV,[F_t(timeV)*1e4 Q_t(timeV)*1e-3 Tin_t(timeV)]), grid on, hold on
plot(timeV,Tsol_salida,LineWidth=3), grid on,
plot(TT,Tsol_estatico,"-.",LineWidth=2,Color=[0.15 0.45 0])
xline(950+truedelay)
hold off
xlim([900 1100])
legend("Flow","Power","T at inlet","T at outlet","static model",Location="best")
title("Step in Flow through the tubular exchanger")
```

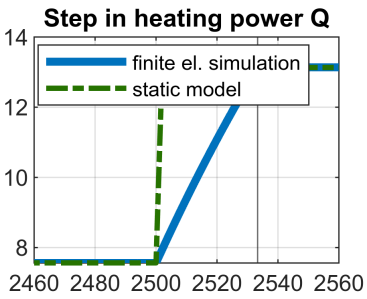


Zoom fragment 3: heating power step

```
plot(timeV,Tsol_salida,LineWidth=3), grid on, hold on
plot(TT,Tsol_estatico,'-.',LineWidth=2,Color=[0.15 0.45 0]), hold off
truedelay=S*Ltot/(F_pf+0.0001)
```

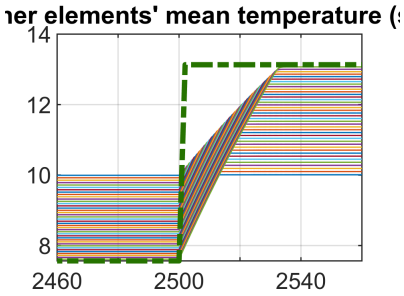
```
truedelay = 33.3333
```

```
xline(2500+truedelay)
xlim([2460 2560])
title("Step in heating power Q"), legend("finite el. simulation","static model",Locatio
```

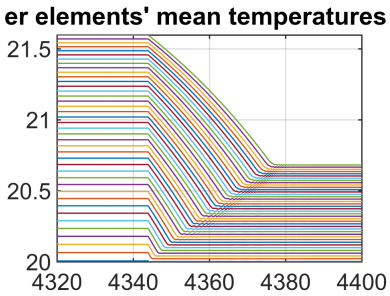


Simulation of internal states

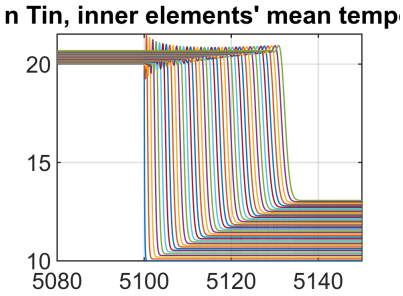
```
plot(timeV,Tsol(:,1:5:end)), grid on, hold on
plot(TT,Tsol_estatico,'-.',LineWidth=2,Color=[0.15 0.45 0]), hold off
xlim([2460 2560])
title("Step in Q: inner elements' mean temperature (some of them)")
```




```
plot(timeV,Tsol(:,1:5:end))
xlim([4320 4400]), grid on
title("Step in F: inner elements' mean temperatures (some of them)")
```




```
plot(timeV,Tsol(:,1:5:end))
xlim([5080 5150]), grid on
title("Step in Tin, inner elements' mean temperature")
```



Conclusions

Simulating the (nonlinear if flow changes) transient response of a tubular heater may be "infinitely complex" (infinite number of elements) or just "first order"... or even a nonlinear gain with no dynamics at all (steady-state model). If the heater is a subsystem of a larger system, we cannot devote a lot of computing resources to its simulation, we must determine a "sensible" number of elements, depending on the "input bandwith" (speed of input changes) and the bandwith (time constants) of downstream elements and the overall precision requirements.

As I said, in most applications, the model of choice is the static steady-state one, if residence time is small compared to other time constants... that's a wise choice in many cases, except for, say, higher-bandwith outlet temperature control.