Supplementary information for the article Systematic design of active constraint switching using selectors

Dinesh Krishnamoorthy, Sigurd Skogestad

Department of Chemical Engineering, Norwegian University of Science and Technology (NTNU), Trondheim, Norway

1. The simulator model for Example 3: Distillation column

We consider a two-product distillation column with N_T stages as shown in Fig. 1 . The following assumptions are made about the model:

- Binary mixture
- constant pressure, relative volatility and molar flows
- no vapor holdup
- linear liquid dynamics
- equilibrium on all stages

The total mass balance and the mass balance for the light component on stage i, except in the condenser ($i = N_T$), feed stage ($i = N_f$) and reboiler ($i = 1$) is given by:

$$\frac{dM_i}{dt} = L_{i+1} - L_i + V_{i-1} - V_i \tag{1}$$

$$\frac{dM_i x_i}{dt} = L_{i+1}x_{i+1} + V_{i-1}y_{i-1} - L_i x_i - V_i y_i \tag{2}$$

$$\forall i \in \{2, \dots, N_T - 1\} \setminus \{N_f\}$$

where L_i and V_i are the liquid and vapor flows from the i^{th} stage (in kmol/min), respectively, and M_i is the liquid holdup in the i^{th} stage (in kmol). x_i and y_i are the liquid and vapor mole fractions of light component on the i^{th} stage, respectively.

*Corresponding author

Email address: `dinesh.krishnamoorthy@ntnu.no`, `skoge@ntnu.no` (Dinesh Krishnamoorthy, Sigurd Skogestad)

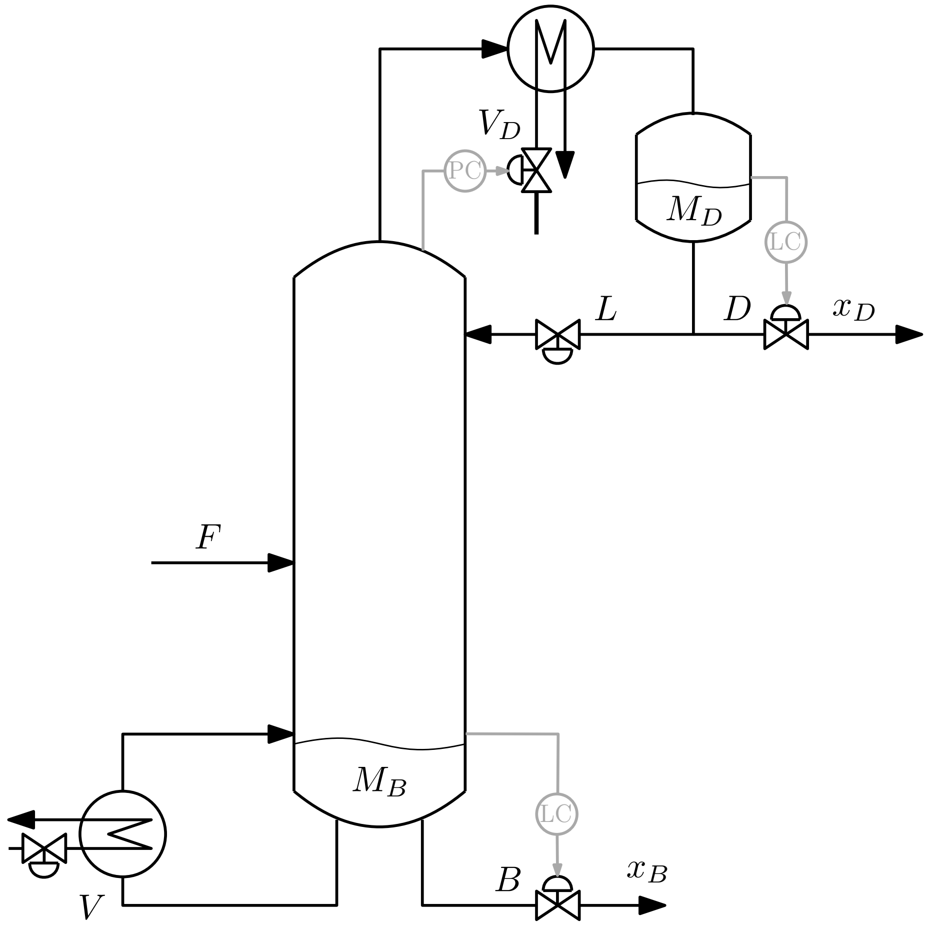

Figure 1: Schematic representation of the distillation column

The mass balance on the feed stage ($i = N_f$) is given by,

$$\frac{dM_{N_f}}{dt} = L_{N_f+1} - L_{N_f} + V_{N_f-1} - V_{N_f} + F \tag{3}$$

$$\frac{dM_{N_f}x_{N_f}}{dt} = L_{N_f+1}x_{N_f+1} + V_{N_f-1}y_{N_f-1} - L_{N_f}x_{N_f} - V_{N_f}y_{N_f} + Fz_F \tag{4}$$

The mass balance on the reboiler ($i = 1$) is given by,

$$\frac{dM_B}{dt} = L_2 - V - B \tag{5}$$

$$\frac{dM_B x_1}{dt} = L_2x_2 - Vy_1 - Bx_1 \tag{6}$$

where B is the bottom flow rate and V is the boilup as shown in Fig. 1.

The mass balance on the condenser ($i = N_T$) is given by,

$$\frac{dM_D}{dt} = V_{N_T-1} - L - D \tag{7}$$

$$\frac{dM_D x_{N_T}}{dt} = V_{N_T-1}y_{N_T-1} - Lx_{N_T} - Dx_{N_T} \tag{8}$$

where D is the distillate flow rate and L is the reflux as shown in Fig. 1.

From this, we get the expression for the rate of change of liquid mole fraction

$$\frac{dx_i}{dt} = \frac{1}{M_i}\left(\frac{dM_i x_i}{dt} - x_i\frac{dM_i}{dt}\right) \quad \forall i \in \{1, \ldots, N_T\} \tag{9}$$

The model therefore has $2N_T$ differential states denoted by $[\{x_i\}_{i=1}^{N_T}, \{M_i\}_{i=1}^{N_T}]^\mathsf{T}$.

The liquid flows depend on the liquid holdup on the stage above and the vapor flow as follows

$$L_i = L0_i + \frac{1}{\tau_i}(M_i - M0_i) + (V - V0)_{i-1}\lambda \tag{10}$$

where $L0_i$ (in kmol/min) and $M0_i$ (in kmol) are the nominal values for the liquid flow and holdup on stage i. The effect of vapor flow on the liquid flow is captured by λ.

The vapor composition can then be computed from the vapor-liquid equilibrium

$$y_i = \frac{\alpha x_i}{1 + (\alpha - 1)x_i} \tag{11}$$

where α is the constant relative volatility.

2. Controller design

We assume that the overhead vapour V_D is used to maintain a constant pressure. Stable operation of the column requires the levels M_B and M_D to be controlled. In this model, the column is stabilized using the LV-configuration where we use D to control M_D, and B to control M_B as shown in Fig. 1. We use a P-controllers for each level control loop, with the controller gain $K_P = 10$ for both the loops.

As mentioned in the manuscript, the purity constraint on x_D will always be active, since this is the most valuable product. The x_D composition is controlled using the reflux L using a PI controller that is tuned using the SIMC tuning rules. For a desired closed loop time constant of $\tau_c = 10$, this results in the proportional gain $K_P = 7.8947$ and $K_I = 0.2193$.

The composition control for the bottom product x_B is also achieved using a PI controller that is tuned using the SIMC rules. For a desired closed loop time constant of $\tau_c = 10$, this results in the proportional gain $K_P = 2.2140$ and $K_I = 0.123$.

The `MATLAB` scripts for the distillation column example is given below or can be found in `https://github.com/dinesh-krishnamoorthy/ConstraintSwitching`.

Listing 1: Main script

```
import casadi.*

clear
clc

Ts = 1; % sampling time in min

global pF pB pD
pF = 10;             % feed price [$/mol]
pB = 8;              % Bottom product price [$/mol]
pD = 86;             % distillate price [$/mol]

[sys,F_integrator,f,par] = cola_lv(Ts);

% ---------- initialization -----------

par.F = 1.42;    % Feedrate
par.pV = 0.015; % Energy price

L = 3.1;
V = 3.9;

dist = vertcat(par.F,par.pV);
u_in = vertcat(L,V);
[xf,exitflag] = solveODE(sys,dist,u_in);

% ---------- Setup PI Controllers ------

% Concentration controller for xD tuned using SIMC rule
CCd.k = 0.456; CCd.tau1 = 36; CCd.tauC = 10;
CCd.Kp = CCd.tau1/(CCd.k*CCd.tauC);
CCd.Ki = CCd.Kp/CCd.tau1;
CCd.err = 0.95-xf(41);     % initialize error
CCd.err0 = CCd.err;

% Concentration controller for xB tuned using SIMC rule
```

```
CCb.k = 0.813; CCb.tau1 = 18; CCb.tauC = 10;
CCb.Kp = CCb.tau1/(CCb.k*CCb.tauC);
CCb.Ki = CCb.Kp/CCb.tau1;
CCb.err = max(0.99,0.99127+0.41077*par.pV-30*par.pV^2)-(1-xf(1));
CCb.err0 = CCb.err;
CCb.V = V;

% -------------- Simulation -------------

for sim_k = 1:10*24*60

% ----- Disturbance in pV ------
    if sim_k>3*24*60 && sim_k≤7*24*60
        par.pV = 0.008;
        else if sim_k>7*24*60
                par.pV = 0.02;
        end
    end

% -------- controllers ---------
    % Concentration controller for xD (always active)
    CCd.err = 0.95-xf(41);
    L = L + CCd.Kp*CCd.err + CCd.Ki*CCd.err -CCd.Kp*CCd.err0;
    CCd.err0 = CCd.err;


% Conceentration controller for xB
    CCb.err = ...
        max(0.99,0.99127+0.41077*par.pV-30*par.pV^2)-(1-xf(1)); ...
        % Max selector
    CCb.V = CCb.V + CCb.Kp*CCb.err + CCb.Ki*CCb.err ...
        -CCb.Kp*CCb.err0;
    V = min(4.008,CCb.V);  % Min selector
    CCb.V = V;  % Output from the max-min structure
    CCb.err0 = CCb.err;

% --------- Simulator ----------
    par.d = vertcat(par.F,par.pV);
    Fk = F_integrator('x0',xf,'p',vertcat(L,V,par.d));
    % set new initial values for the next iteration
    xf =  full(Fk.xf); % states
    qf = full(Fk.qf); % cost function

% ----- Extract and store data -----
    sim.L(sim_k)     = L;
    sim.V(sim_k)     = V;
    sim.B(sim_k)     = par.Bs + (xf(par.NT+1) - par.MBs)*par.KcB;
    sim.D(sim_k)     = par.Ds + (xf(2*par.NT) - par.MDs)*par.KcD;
    sim.xD(sim_k)    = xf(41);
    sim.xB(sim_k)    = 1-xf(1);
    sim.q(sim_k)     = qf;
    sim.pV(sim_k)    = par.pV;

end

sim.t_days = (1:10*24*60)/(24*60);
sim.t_hrs = (1:10*24*60)/(60);
```

```

% save('sim','sim')
```

Listing 2: Column A model with the LV configuration

```
function [sys,F_integrator,f,par] = cola_lv(Ts)
%
% colamod - This is a nonlinear model of a distillation column with
%           NT-1 theoretical stages including a reboiler (stage ...
    1) plus a
%           total condenser ("stage" NT). The liquid flow ...
    dynamics are
%           modelled by a simple linear relationship.
%           Model assumptions: Two components (binary ...
    separation); constant
%           relative volatility; no vapor holdup; one feed and ...
    two products;
%           constant molar flows (same vapor flow on all stages);
%           total condenser
%
%           The model is based on column A in Skogestad and ...
    Postlethwaite
%            (1996). The model has 82 states.
%
%           Re-written using CasADi v3.5.1 by D. Krishnamoorthy ...
     (2019)
%
%

import casadi.*
global pF pB pD

%------------------------------------------------------------
% The following data need to be changed for a new column.
% These data are for "colmn A".
% Number of stages (including reboiler and total condenser:
NT=41;
% Location of feed stage (stages are counted from the bottom):
NF=21;
% Relative volatility
alpha=1.5;
% Nominal liquid holdups
M0(1)=0.5;          % Nominal reboiler holdup (mol)
i=2:NT-1; M0(i)=0.5.*ones(1,NT-2);% Nominal stage (tray) holdups ...
     (mol)
M0(NT)=0.5;         % Nominal condenser holdup (mol)
% Data for linearized liquid flow dynamics (does not apply to ...
    reboiler and condenser):
taul= 0.063;        % time constant for liquid dynamics (s)
F0=1;               % Nominal feed rate (mol/s)
qF0 = 1;            % Nominal fraction of liquid in feed
L0=2.70629;         % Nominal reflux flow (from steady-state data)
L0b=L0 + qF0*F0;    % Nominal liquid flow below feed (mol/s)
lambda=0;           % Effect of vapor flow on liquid flow ...
     ("K2-effect")
V0=3.20629;V0t=V0+(1-qF0)*F0;% Nominal vapor flows - only needed ...
```

```
    if lambda is nonzero
% End data which need to be changed

% Prices

pV  = MX.sym('pV');

%-------------------------------------------------------------

% Differential states
x = MX.sym('x',NT);         % Liquid composition from btm to top
M = MX.sym('M',NT);         % Liquid hold up from btm to top

% Inputs and disturbances

LT = MX.sym('LT');          % Reflux
VB = MX.sym('VB');          % Boilup

F  = MX.sym('F');           % Feedrate

zF = 0.5;% MX.sym('zF');          % Feed composition
% alpha = MX.sym('alpha');   % relative volatility
qF = 1; % MX.sym('qF');          % Feed liquid fraction

% Vapor-liquid equilibria
i=1:NT-1;    y=alpha*x(i)./(1+(alpha-1)*x(i));

% Vapor Flows assuming constant molar flows
i=1:NT-1;    V=VB*ones(1,NT-1);
i=NF:NT-1;   V(i)=V(i) + (1-qF)*F;

% Liquid flows assuming linearized tray hydraulics with time ...
    constant taul
% Also includes coefficient lambda for effect of vapor flow ...
    ("K2-effect").
L = 0;
for i=2:NF
    L = [L; L0b + (M(i)-M0(i)')./taul + lambda.*(V(i-1)-V0)'];
end

for i=NF+1:NT-1
    L = [L; L0  + (M(i)-M0(i)')./taul + lambda.*(V(i-1)-V0t)'];
end
L= [L;LT];

% P-Controllers for control of reboiler and condenser hold up.
par.KcB=10;  par.KcD=10;         % controller gains
par.MDs=0.5; par.MBs=0.5;        % Nominal holdups - these are ...
    rather small
par.Ds=0.5;  par.Bs=0.5;          % Nominal flows
MB=M(1);  MD=M(NT);       % Actual reboiler and condenser holdup
D=par.Ds+(MD-par.MDs)*par.KcD;       % Distillate flow
B=par.Bs+(MB-par.MBs)*par.KcB;       % Bottoms flow

% Time derivatives from  material balances for
% 1) total holdup and 2) component holdup

% Reboiler (assumed to be an equilibrium stage)
dMdt = L(2)  - V(1) - B;
dMxdt= L(2)*x(2) - V(1)*y(1) - B*x(1);

% Column
for i=2:NT-1
    dMdt = [dMdt; L(i+1)  - L(i) + V(i-1)'   - V(i)'];
    dMxdt= [dMxdt; L(i+1).*x(i+1) - L(i).*x(i) + V(i-1)'.*y(i-1) ...
        - V(i)'.*y(i)];
end

% Total condenser (no equilibrium stage)
dMdt = [dMdt; V(NT-1)  - LT - D];
dMxdt = [dMxdt; V(NT-1)*y(NT-1) - LT*x(NT) - D*x(NT)];

% Correction for feed at the feed stage
% The feed is assumed to be mixed into the feed stage
dMdt(NF) =  dMdt(NF)  + F;
dMxdt(NF)= dMxdt(NF) + F*zF;

% Compute the derivative for the mole fractions from d(Mx) = x ...
    dM + M dx
dxdt = [];
for i=1:NT
    dxdt = [dxdt;(dMxdt(i) - x(i).*dMdt(i) )./M(i)];
end

% Build ODE
diff = vertcat(dxdt,dMdt);
x_var = vertcat(x,M);
d_var = vertcat(F,pV);
p_var = vertcat(LT,VB);

J = pF*F + pV*VB - pB*B - pD*D;

nlcon = [];% vertcat(x_var(41),1-x_var(1));
lb = [];%[0.95;0.292];
ub = [];%[1;1];

f = Function('f',{x_var,p_var,d_var},{diff,J},...
        {'x','p','d'},{'xdot','qj'});

ode = ...
    struct('x',x_var,'p',vertcat(p_var,d_var),'ode',diff,'quad',J);
opts = struct('tf',Ts);

% create CVODES integrator
F_integrator = integrator('F','cvodes',ode,opts);

par.NT = NT;
sys.x = x_var;
sys.u = p_var;
sys.d = d_var;
sys.dx = diff;
sys.y = vertcat(T,F,LT,VB,sys.x(41),1-sys.x(1),pV);

sys.L = J;

sys.nlcon = nlcon;
sys.lb = lb;
sys.ub = ub;

par.lbx = 1e-5*ones(2*par.NT,1);
par.ubx = 2*ones(2*par.NT,1); par.lbx(par.NT) = 0.95;
par.dx0 = 0.5*ones(2*par.NT,1);
par.lbu = [0;0;0];
par.ubu = [10;4.008;6];
par.u0  = [2.706;3.206;1];
```

Listing 3: Function to compute the steady-state for a given input and disturbance

```
    function [xf,exitflag] = solveODE(sys,d_val,u_in,opts)

    % Rootfinder function that computes the steady-state at a ...
        given input u_in
    % Written by Dinesh Krishnamoorthy, Jul 2019, NTNU

    import casadi.*

    if nargin<8
        opts = struct('warn_initial_bounds',false, ...
        'print_time',false, ...
        'ipopt',struct('print_level',1)...
        );
    end

    lbx = 0.*ones(size(sys.x));
    ubx = 1e5.*ones(size(sys.x));
    dx0 = 1e-2.*ones(size(sys.x));

    assert(numel(sys.u)==numel(u_in))
    assert(numel(sys.d)==numel(d_val))

    w = {};
    w0 = [];
    lbw = [];
    ubw = [];

    g = {};
    lbg = [];
    ubg = [];

    w = {w{:},sys.x,sys.u};
    lbw = [lbw;lbx;u_in];
    ubw = [ubw;ubx;u_in];
    w0 = [w0;dx0;u_in];

    g = {g{:},vertcat(sys.dx)};
    lbg = [lbg;zeros(numel(sys.dx),1)];
    ubg = [ubg;zeros(numel(sys.dx),1)];

    if ¬isempty(sys.nlcon)
    assert(numel(sys.nlcon)==numel(sys.lb))
    assert(numel(sys.nlcon)==numel(sys.ub))

    g = {g{:},sys.nlcon};
    lbg = [lbg;sys.lb];
    ubg = [ubg;sys.ub];
    end

    nlp = ...
        struct('x',vertcat(w{:}),'p',sys.d,'f',0,'g',vertcat(g{:}));
    solver = nlpsol('solver','ipopt',nlp,opts);

    sol = solver('x0',w0,'p',d_val,'lbx',lbw,'ubx',ubw,...
            'lbg',lbg,'ubg',ubg);
    wf = full(sol.x);
    xf = wf(1:numel(sys.x));

    flag = solver.stats();
    exitflag =  flag.return_status;
```