Octave vs. Python for Example 1.11

Below, we present an example file to show that, for our purposes, Python isn't that much different from Octave/MATLAB On the left side, we show the the script written using Octave (MATLAB compatibility requires breaking out the subfunctions), while on the right, we show the script rewritten to use Python+Casadi (with a bit of `mpc-tools-casadi` as well).

```
% cstr.m applies offset free linear MPC
% to the linearized and
% nonlinear CSTR.
% See Pannocchia and Rawlings, AIChE J, 2002.
%
% Edits by Michael Risbeck (April 2015).

global F F0 r E k0 DeltaH rhoCp T0 c0 U Tc hs

% parameters and sizes for the nonlinear system
delta = 1;
n = 3;
m = 2;
nmeas = n;
np = 1;
small = 1e-5; % Small number.

F0 = 0.1; %          m^3/min
T0 = 350; %          K
c0 =  1;  %          kmol/m^3
r  = 0.219; %        m
k0 = 7.2e10; %       minâĹŠ1
E  =  8750; %        K
U  =  54.94; %       kJ/(min m^2 K)
rho = 1e3;   %       kg/m^3
Cp  = 0.239;  %      kJ/(kg K)
DeltaH = -5e4; %     kJ/kmol
Tcs = 300; %         K
hs = 0.659; %        m
ps = 0.1*F0; %        m^3/min

rhoCp = rho*Cp;

function df = partial(x)
    global F F0 r E k0 DeltaH rhoCp T0 c0 U Tc
    c = x(1);
    T = x(2);
    h = x(3);
    k = k0*exp(-E/T);
    kprime = k*E/(T^2);
    Fprime = F0/(pi*r^2*h);
    df = [-Fprime - k, -kprime*c, ...
    -Fprime*(c0-c)/h, ...
    0, 0, (c0-c)/(pi*r^2*h);
    -k*DeltaH/rhoCp, -Fprime ...
        - kprime*c*DeltaH/(rhoCp) ...
        - 2*U/(r*rhoCp),
    -Fprime*(T0-T)/h, ...
    2*U/(r*rhoCp), 0, (T0-T)/(pi*r^2*h);
    0, 0, 0, 0, -1/(pi*r^2), 1/(pi*r^2)];
endfunction

function rhs = massenbalstst(x)
```

```
# Example 1.11 from Rawlings and Mayne.
import mpctools as mpc
import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt
from matplotlib import gridspec

# Define some parameters and then the CSTR model.
Nx = 3
Nu = 2
Nd = 1
Ny = Nx
Delta = 1
eps = 1e-6 # Use this as a small number.

T0 = 350
c0 = 1
r = .219
k0 = 7.2e10
E = 8750
U = 54.94
rho = 1000
Cp = .239
dH = -5e4

def ode(x,u,d):
    # Grab the states, controls, and disturbance.
    [c, T, h] = x[0:Nx]
    [Tc, F] = u[0:Nu]
    [F0] = d[0:Nd]

    # Now create the ODE.
    rate = k0*c*np.exp(-E/T)

    dxdt = [
        F0*(c0 - c)/(np.pi*r**2*h) - rate,
        F0*(T0 - T)/(np.pi*r**2*h)
            - dH/(rho*Cp)*rate
            + 2*U/(r*rho*Cp)*(Tc - T),
        (F0 - F)/(np.pi*r**2)
    ]
    return np.array(dxdt)

# Turn into casadi function and simulator.
ode_casadi = mpc.getCasadiFunc(ode,[Nx,Nu,Nd],["x","u","d"]
    global F F0 r E k0 DeltaH rhoCp T0 c0 U Tc hs
    c = x(1);
    T = x(2);
    h = x(3);
    k = k0*exp(-E/T);
    rate = k*c;
    dcdt = F0*(c0-c)/(pi*r^2*h) - rate;
    dTdt = F0*(T0-T)/(pi*r^2*h) - ...
        DeltaH/rhoCp*rate + ...
        2*U/(r*rhoCp)*(Tc-T);
    % fix the reactor height
    dhdt = h - hs;
    rhs = [dcdt; dTdt; dhdt];
endfunction

function rhs = massenbal(t, x)
    global F F0 r E k0 DeltaH rhoCp T0 c0 U Tc
    c = x(1);
    T = x(2);
    h = x(3);
    k = k0*exp(-E/T);
    rate = k*c;
    dcdt = F0*(c0-c)/(pi*r^2*h) - rate;
    dTdt = F0*(T0-T)/(pi*r^2*h) - ...
      DeltaH/rhoCp*rate + ...
      2*U/(r*rhoCp)*(Tc-T);
    dhdt = (F0 - F)/(pi*r^2);
    rhs = [dcdt; dTdt; dhdt];
endfunction
```

```
cstr = mpc.DiscreteSimulator(ode, Delta, [Nx,Nu,Nd], ["x",

# We don't need to take any derivatives by hand because Ca
```

```
%% find the steady-state
Fs = F0; F = F0;
Tc = Tcs;
z0 = [c0; Tc; hs];
[z, fval, info] = fsolve(@massenbalstst, z0);
if ( info ~= 1 )
    warning('failure␣to␣find␣steady␣state!')
endif
cs = z(1);
Ts = z(2);
hs = z(3);
zs = [cs; Ts; hs];
```

```
# Steady-state values.
cs = .878
Ts = 324.5
hs = .659
Fs = .1
Tcs = 300
F0s = .1

# Update the steady-state values a few times to make sure
for i in range(10):
    [cs,Ts,hs] = cstr.sim([cs,Ts,hs],[Tcs,Fs],[F0s]).tolis
xs = np.array([cs,Ts,hs])
us = np.array([Tcs,Fs])
ds = np.array([F0s])
```

```
%check the linear model
G = partial(zs);
Ac =  G(:, 1:n);
Bc = G(:, n+1:n+2);
Bpc = G(:,n+3:end);
C = eye(n);
sys = ss(Ac, [Bc, Bpc], C, ...
    zeros(size(C,1),size([Bc, Bpc],2)));
dsys = c2d(sys, delta);
A = dsys.a;
B = dsys.b(:,1:m);
Bp = dsys.b(:,m+1:end);
```

```
# Now get a linearization at this steady state.
ss = mpc.util.linearizeModel(ode_casadi, [xs,us,ds], ["A",
A = ss["A"]
B = ss["B"]
Bp = ss["Bp"]
C = np.eye(Nx)
```

```
% set up linear controller
```

```
# Weighting matrices for controller.
```

```
Q = diag(1./zs.^2);
R = diag(1./[Tcs; Fs].^2);
[K, P] = dlqr(A, B, Q, R);
K = -K;

% Pick whether to use good disturbance model.
useGoodDisturbanceModel = true();

if useGoodDisturbanceModel
    % disturbance model 6; no offset
    nd = 3;
    Bd = zeros(n, nd);
    Bd(:,3) = B(:,2);
    Cd = [1 0 0; 0 0 0; 0 1 0];
else
    % disturbance model with offset
    nd = 2;
    Bd = zeros(n, nd);
    Cd = [1 0; 0 0; 0 1];
end

Aaug = [A, Bd; zeros(nd, n), eye(nd)];
Baug = [B; zeros(nd, m)];
Caug = [C, Cd];
naug = size(Aaug,1);

% detectability test of disturbance model
detec = rank([eye(n+nd) - Aaug; Caug]);
if (detec < (n+nd))
  warning ('augmented␣system␣is␣not␣detectable\n')
endif

% set up state estimator; use KF
Qw = zeros(naug);
Qw(1:n,1:n) = small*eye(n);
Qw(n+1:end,n+1:end) = small*eye(nd); Qw(end,end)=1.0;
Rv = diag((5e-4*zs).^2);
[L, M, P] = dlqe(Aaug, eye(naug), Caug, Qw, Rv);
Lx = L(1:n,:);
Ld = L(n+1:end,:);

ntimes = 50;
x0 = zeros(n, 1);
x = zeros(n, ntimes);
x(:, 1) = x0;
y = zeros(nmeas, ntimes);
u = zeros(m, ntimes);
randn('seed', 0);
v = zeros(nmeas, ntimes);
xhat_ = zeros(n, ntimes);
dhat_ = zeros(nd, ntimes);
xhat = xhat_;
dhat = dhat_;
time = (0:ntimes-1)*delta;
```

```
Q = np.diag(xs**-2)
R = np.diag(us**-2)

[K, Pi] = mpc.util.dlqr(A,B,Q,R)

# Define disturbance model.
useGoodDisturbanceModel = True # Can be false to pick the

# Bad disturbance model with offset.
if useGoodDisturbanceModel:
    Nid = Ny # Number of integrating disturbances.
else:
    Nid = Nu

Bd = np.zeros((Nx,Nid))
Cd = np.zeros((Ny,Nid))

if useGoodDisturbanceModel:
    Cd[0,0] = 1
    Cd[2,1] = 1
    Bd[:,2] = B[:,1] # or Bp[:,0]
else:
    Cd[0,0] = 1
    Cd[2,1] = 1

# Augmented system. Trailing .A casts to array type.
Aaug = np.bmat([[A,Bd],[np.zeros((Nid,Nx)),np.eye(Nid)]]).
Baug = np.vstack((B,np.zeros((Nid,Nu))))
Caug = np.hstack((C,Cd))

# Check rank condition for augmented system.
svds = linalg.svdvals(np.bmat([[np.eye(Nx) - A, -Bd],[C,Cd
rank = sum(svds > 1e-10)
if rank < Nx + Nid:
    print "***Warning: augmented system is not detectable!
# Build augmented penalty matrices for KF.
Qw = eps*np.eye(Nx + Nid)
Qw[-1,-1] = 1
Rv = eps*np.diag(xs**2)

# Get Kalman filter.
[L, P] = mpc.util.dlqe(Aaug, Caug, Qw, Rv)
Lx = L[:Nx,:]
Ld = L[Nx:,:]

# Now simulate things.
Nsim = 50
t = np.arange(Nsim+1)*Delta
x = np.zeros((Nsim+1,Nx))
u = np.zeros((Nsim,Nu))
y = np.zeros((Nsim+1,Ny))
err = y.copy()
v = y.copy()
xhat = x.copy() # State estimate after measurement.
xhatm = xhat.copy() # State estimate prior to measurement.
dhat = np.zeros((Nsim+1,Nid))
dhatm = dhat.copy()
```

```
xs = zeros(n, ntimes);
us = zeros(m, ntimes);
ys = zeros(nmeas, ntimes);
etas = zeros(nmeas, ntimes);

% Disturbance and setpoint.
p = [zeros(np, 10), ps*ones(np, ntimes-10)];
yset = zeros(nmeas,1);


% Steady-state target matrices.
H  = [1 0 0; 0 0 1];
Ginv = inv([eye(n)-A, -B; ...
    H*C, zeros(size(H,1), m)]);



for i = 1: ntimes
    %% measurement
    y(:,i) = C*x(:,i) + v(:,i);

    %% state estimate
    ey = y(:,i) - C*xhat_(:,i) -Cd*dhat_(:,i);
    xhat(:,i) = xhat_(:,i) + Lx*ey;
    dhat(:,i) = dhat_(:,i) + Ld*ey;

    % Stop if at last time.
    if (i == ntimes) break; endif

    %% target selector
    tmodel.p = dhat(:,i);

    qs = Ginv*[Bd*dhat(:,i); H*(yset-Cd*dhat(:,i))];
    xss = qs(1:n);
    uss = qs(n+1:end);
    xs(:,i) = xss;
    us(:,i) = uss;
    ys(:,i) = C*xss + Cd*dhat(:,i);

    %% control law
    x0 = xhat(:,i) - xs(:,i);
    u(:,i) = K*x0 + us(:,i);

    %% plant evolution
    t = [time(i); mean(time(i:i+1)); time(i+1)];
    z0 = x(:,i) + zs;
    Tc = u(1,i) + Tcs;
    F  = u(2,i) + Fs;
    F0 = p(:,i) + Fs;
    [tout, z] = ode15s(@massenbal, t, z0, options);
    if sum(tout ~= t)
        warning('integrator␣failed!')
    end
    x(:,i+1) = z(end,:)' - zs;

    %% advance state estimates
    xhat_(:,i+1) = A*xhat(:,i) + Bd*dhat(:,i) + B*u(:,i);
    dhat_(:,i+1) = dhat(:,i);
end
u(:,end) = u(:,end-1); % Repeat this for stairstep plot.
```

```
# Pick disturbance and setpoint.
d = np.zeros((Nsim,Nd))
d[:,0] = (t[:-1] >= 10)*.1*F0s
ysp = np.zeros(y.shape)
contVars = [0,2] # Concentration and height.

# Steady-state target selector matrices.
H = np.zeros((Nu,Ny))
H[range(len(contVars)),contVars] = 1
Ginv = np.array(np.bmat([
    [np.eye(Nx) - A, -B],
    [H.dot(C), np.zeros((H.shape[0], Nu))]
]).I) # Take inverse.

for n in range(Nsim + 1):
    # Take plant measurement.
    y[n,:] = C.dot(x[n,:]) + v[n,:]

    # Update state estimate with measurement.
    err[n,:] = y[n,:] - C.dot(xhatm[n,:]) - Cd.dot(dhatm[n
    xhat[n,:] = xhatm[n,:] + Lx.dot(err[n,:])
    dhat[n,:] = dhatm[n,:] + Ld.dot(err[n,:])

    # Make sure we aren't at the last timestep.
    if n == Nsim: break

    # Steady-state target.
    rhs = np.concatenate((Bd.dot(dhat[n,:]), H.dot(ysp[n,:
        - Cd.dot(dhat[n,:]))))
    qsp = Ginv.dot(rhs)
    xsp = qsp[:Nx]
    usp = qsp[Nx:]

    # Regulator.
    u[n,:] = K.dot(xhat[n,:] - xsp) + usp

    # Simulate with nonlinear model.
    x[n+1,:] = cstr.sim(x[n,:] + xs, u[n,:] + us, d[n,:] +

    # Advance state estimate.
    xhatm[n+1,:] = A.dot(xhat[n,:]) + Bd.dot(dhat[n,:]) + 
    dhatm[n+1,:] = dhat[n,:]
```

```
% dimensional units
yd = y + kron(ones(1, ntimes), zs);
ud = u + kron(ones(1, ntimes), [Tcs; Fs]);

% *** Plots ***
figure()
axmul = eye(4) + .05*[1,-1,0,0;-1,1,0,0;0,0,1,-1;0,0,-1,1];
subplot(3,2,1)
plot(time, yd(1,:), '-or')
ylabel('c␣(kmol/m^3)')
axis(axis()*axmul)
subplot(3,2,3)
plot(time, yd(2,:), '-or')
ylabel('T␣(K)')
axis(axis()*axmul)
subplot(3,2,5)
plot(time, yd(3,:), '-or')
ylabel('h␣(m)')
xlabel('Time')
axis(axis()*axmul)
subplot(3,2,2)
stairs(time, ud(1,:), '-r')
ylabel('Tc␣(K)')
axis(axis()*axmul)
subplot(3,2,4)
stairs(time, ud(2,:), '-r')
ylabel('F␣(m^3/min)')
xlabel('Time')
axis(axis()*axmul)
print('cstr_octave.pdf','-dpdf','-S720,600')
```

```
# Define plotting function.
def cstrplot(x,u,ysp=None,contVars=[],title=None):
    u = np.concatenate((u,u[-1:,:]))
    t = np.arange(0,x.shape[0])*Delta
    ylabelsx = ["$c$ (mol/L)", "$T$ (K)", "$h$ (m)"]
    ylabelsu = ["$T_c$ (K)", "$F$ (kL/min)"]

    gs = gridspec.GridSpec(Nx*Nu,2)

    fig = plt.figure(figsize=(10,6))
    for i in range(Nx):
        ax = fig.add_subplot(gs[i*Nu:(i+1)*Nu,0])
        ax.plot(t,x[:,i] + xs[i],'-ok')
        if i in contVars:
            ax.step(t,ysp[:,i] + xs[i],'-r',where="post")
        ax.set_ylabel(ylabelsx[i])
        mpc.plots.zoomaxis(ax,yscale=1.1)
    ax.set_xlabel("Time (min)")
    for i in range(Nu):
        ax = fig.add_subplot(gs[i*Nx:(i+1)*Nx,1])
        ax.step(t,u[:,i] + us[i],'-k',where="post")
        ax.set_ylabel(ylabelsu[i])
        mpc.plots.zoomaxis(ax,yscale=1.25)
    ax.set_xlabel("Time (min)")
    fig.tight_layout(pad=.5)
    if title is not None:
        fig.canvas.set_window_title(title)
    return fig
fig = cstrplot(x,u,ysp=None,contVars=[],title=None)
fig.savefig("cstr_python.pdf")
```