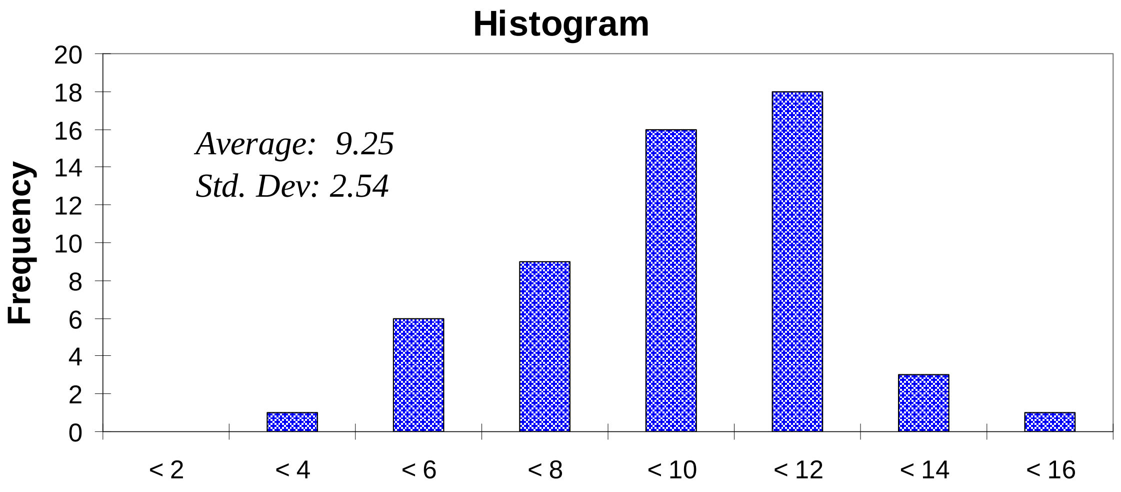
10.34 Quiz 1, October 4, 2006

Solution – *Graded out of a total of 15 points + 1 bonus point*



(a) *Write a couple of Matlab functions that together compute the concentrations [P] and [Nutrients] (units: M = moles/liter), as well as the number of cells per liter, in the output stream when the system is operated at steady-state. Give numerical values for all the inputs. Do you think that scaling will be a problem? Explain and give an appropriate scaling factor if necessary.*

10 points
- 2 pts: general structure of the functions
- 0.5 pts: Unit errors/mismatches
- 4.5 pts: Each balance equation (1.5 pts each)
- 0.5 pts: Writing the fsolve equations correctly as dX/dt = In – Out + Gen – Consum. This is essential to getting the sign of the Jacobian eigenvalues correct (but not the solution).
- 2.5 pts: Scaling assessment (1 pts for scaling could be a problem; 1.5 pts for scaling factor)

For this part, you were essentially asked to write a program that can solve the problem, giving all the numerical inputs necessary to run the function. These could be passed to the function as arguments, or included in the function/script body as parameter definitions. Two Matlab functions (one each for scaled and unscaled N_{cells}) are included at the end as examples of possible solutions (of course, this is not the only way to solve the problem, but it works). Executable `.m` files are also posted on MIT Server. The basic approach should have involved using `fsolve` to solve a set of nonlinear equations. Other functions that could have been used were `fzero` (probably not a good choice), `fminsearch`, and `fmincon`. An ODE solver approach is also valid and would generally give you a stable solution (expect with a very unfortunate initial guess that is close to the unstable conditions). The three equations that needed to be solved were the $dN_{cells}/dt = 0$, the *nutrient balance*, and the *product balance*. These were given to you for the most part in the supplied functions, but you needed to define the *nutrient consumption rate* and the *P production rate* in terms of the known system properties; the resulting equations are: (be sure that you convert the flow rate to *L/sec*)

$$\frac{dN_{cells}}{dt} = 0 = \frac{k_1 N_{cells}\left[Nutrients\right]}{\left(1+c_1\left[Nutrients\right]\right)\left(1+d\left[P\right]\right)} - V_{flow}\frac{N_{cells}}{V_{rxtr}}$$

$$\frac{d\left[Nutrients\right]}{dt} = 0 = V_{flow}\left(\left[Nutrient\right]_{In} - \left[Nutrient\right]\right) - \left[k_2 N_{cells} + c_2\left(Cell\ Multiplication\right)\right]$$

$$\frac{d\left[P\right]}{dt} = 0 = \frac{k_3 N_{cells}\exp\left(-d\left[P\right]\right)}{\left(1+c_1\left[Nutrients\right]\right)}\cdot\left(\left[Nutrients\right]-0.01\right)^2 - V_{flow}\left[P\right]$$

Scaling could be a problem in this scenario due to the large differences in the inherent system variables, but does not make the system intractable. Regardless, scaling is usually beneficial and should be done. The appropriate scaling factor can be determined if one assumes all nutrients are consumed in the reactor and [P] = 0. This forces Cell Multiplication $\rightarrow$ 0, and the nutrient consumption rate equation can be solved for an upper bound on N_{cells}. This value can be used as a scaling factor, similar to the characteristic scales seen in transport problems.

$$N^*_{cells} = \frac{V_{flow}\left[Nutrient\right]_{In}}{k_2} = \frac{0.0383\times 0.2}{1\times 10^{-7}} = 7.67\times 10^4\ cells$$

In a real situation, one would want to scale both the number of cells and the system parameters that are also based on the number of cells (i.e. k_2, k_3, and c_2). However, since the functions were prewritten for you in this exercise, one could not do it without rewriting the functions. As can be seen by running the posted files, proper scaling of the variable does make convergence to the solution significantly faster. For an initial guess of `Ncells_0 = 1e5, C_nut_0 = 0.05, C_P_0 = 0.05`, the convergence is 13 iterations and 56 function calls for unscaled and 4 iterations and 20 function calls for the case where the N_{cells} and parameters are scaled.

(b) *If your program from part (a) works correctly, how would you test whether the solution found is physical and achievable (i.e. stable)? (Explain in words; bonuses for giving correct relevant equations and/or Matlab functions).*

3 points + 0.5 bonus points
- 1 pt: giving the physical bounds on the variables
- 2 pts: Jacobian eigenvalues must be less that zero
- 0.5 pts: giving Jacobian expressions and/or Matlab function

For feasibility, one needs to look at the realistic limits of the problem variables. In this problem scenario, the physical range of the variables is as follows:

$$N_{cells} \geq 0 \qquad 0 \leq [Nutrients] \leq 0.2M \qquad [P] \geq 0$$

In order to test for stability, you can compute the Jacobian matrix for the system of equations $J_{i,k} = (df_i \;/\; dx_k)$ and examine the eigenvalues. Ideally, one would like to see that all of the eigenvalues of the Jacobian are less than zero at the solution, signifying that a perturbation to the solution will decay back to the same solution. This can be accomplished in two ways: calculating the analytical derivatives or asking `fsolve` to return the numerical Jacobian at the solution. The syntax for the latter is:

```
[var,fval,exitflag,output,jacobian] = fsolve(@chemostat,var_0,...
```

For this set of conditions and the unscaled problem, the solution and the eigenvalues of the Jacobian are shown below (you obviously could not calculate this during the test):

```
Number of Cells in Reactor = 7.456518e+004
Concentration of Nutrients (M) = 0.00051114
Concentration of Product (M) = 0.00017513

Eigenvalues of Jacobian: -0.4016, -0.0095, -0.0383
```

If you have the analytical Jacobian, it is often useful to pass this to the solver, as it can greatly enhance the performance and convergence. The analytical Jacobian would be:

$$\underline{\underline{J}} = \begin{bmatrix} \dfrac{\partial F_1}{\partial N_{cells}} & \dfrac{\partial F_1}{\partial [Nut]} & \dfrac{\partial F_1}{\partial [P]} \\ \dfrac{\partial F_2}{\partial N_{cells}} & \dfrac{\partial F_2}{\partial [Nut]} & \dfrac{\partial F_2}{\partial [P]} \\ \dfrac{\partial F_3}{\partial N_{cells}} & \dfrac{\partial F_3}{\partial [Nut]} & \dfrac{\partial F_3}{\partial [P]} \end{bmatrix}$$

Where the *F's* are those three equations given earlier used to solve the problem. The analytical derivative were calculated using Maple and can be seen at the end of this solution.

(c) If your program from part (a) converges to an unphysical or unstable solution, what would you do next to try to find an experimentally-relevant steady-state solution? (Explain in words; bonuses for giving correct relevant equations and/or Matlab functions.)

2 points + 0.5 bonus points
- 2 pts: Systematic and logical varying of initial guesses and/or explaining the concept and use of homotopy to achieve a stable solution
- 0.5 pts: Matlab code showing how to implement the strategy

For nonlinear problems, there is no sure way to ensure a feasible solution (or any solution for that matter). This means that one of the most effective ways of finding a desirable solution is to try different initial guesses. This can be done easily in a systemic way for systems with a small number of variables that solve quickly: make a vector of reasonable initial guess for each variable, and iterate over all combinations of initial guesses to try to find a feasible solution. One could also try random initial guesses using the `rand` function to generate a random number between 0 and 1, and then scaling it by the appropriate amount. Using the concept of homotopy as described in Beers' text on page 121 is also a valid answer. In this approach, you would start with a physical situation in which the solution is trivial (i.e. washout in this case with a very high flow rate), where the inlet and outlet conditions are the same. Then, the flow rate is slowly stepped down to the actual conditions, using each successive solution as the initial guess to the next step.

An example of the scan of the initial guess space is shown below. Even with this simple example, the `fsolve` function must be executed 1000 times.

```
Ncells_0 = logspace(5,7,10);   % number of cells
C_nut_0 = linspace(0,0.2,10);
C_P_0 = logspace(-6,0,10);

L1 = length(Ncells_0);
L2 = length(C_nut_0);
L3 = length(C_P_0);

options = optimset('Display','off','MaxFunEvals',10000,'MaxIter',1000);
for i=1:L1
    for j=1:L2
        for k=1:L3
            var_0 = [Ncells_0(i); C_nut_0(j); C_P_0(k)];

            step = L2*L3*(i-1) + L3*(j-1) + k;

            var(:,step) = fsolve(@chemostat,var_0,options,C_nut_in,params);

        end
    end
    disp(['Step Number ',num2str(L2*L3*(i-1)),' of ',num2str(L1*L2*L3)]);
end
```

These are possible additional functions that are needed to solve the problem. Unscaled and scaled are both presented (note in the scaled case, the parameter definitions also need to be changed (see .m posted on MIT Server**)**

```
% 10.34 - Fall 2006
% Chemostat problem
% Rob Ashcraft - Oct. 4, 2006

% Chemostat problem
function quiz1_main_unscaled

clear; clc;
global V_flow Vrxtr

% define the parameter values
params = param_set;

V_flow = 2.3/60;  % liters/sec
Vrxtr = 150;   % liters
C_nut_in = 0.2;  % in Molar

%initial guesses
Ncells_0 = 1e5;
C_nut_0 = 0.05;
C_P_0 = 0.05;

var_0 = [Ncells_0; C_nut_0; C_P_0];

options = optimset('Display','iter','MaxFunEvals',10000,'MaxIter',1000,'TolX',1e-8,'TolFun',1e-
8');

[var,fval,exitflag,output,jacobian] = fsolve(@chemostat,var_0,options,C_nut_in,params);

jacobian_at_solution = jacobian
Jac_cond_number = cond(jacobian)
Jacobian_eigenvalues = eig(jacobian)

disp('  ');
disp(['Number of Cells in Reactor = ',num2str(var(1),'%8.6e')]);
disp(['Concentration of Nutrients (M) = ',num2str(var(2))]);
disp(['Concentration of Product (M) = ',num2str(var(3))]);
disp('  ');
disp(['Production Rate of Product (mole/hr) = ',num2str(var(3)*V_flow*3600)]);
resid_final = chemostat(var,C_nut_in,params)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function resid = chemostat(var,C_nut_in,params)

global V_flow Vrxtr

Ncells = var(1);
C_nut = var(2);
C_P = var(3);

CMrate = Cell_Multiplication(Ncells,C_nut,C_P,params);
NCrate = Nutrient_Consumption(Ncells,C_nut,C_P,params);
Prate = P_production(Ncells,C_nut,C_P,params);

dNcell_dt = CMrate - V_flow*Ncells/Vrxtr;

nutrient_bal = (C_nut_in - C_nut)*V_flow - NCrate;

P_bal = Prate - V_flow*C_P;

resid = [dNcell_dt; nutrient_bal; P_bal];

return;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% 10.34 - Fall 2006
% Quiz 1
% Rob Ashcraft - Oct. 4, 2006

% Chemostat problem with Ncells and parameters scaled
function quiz1_main_scaled_all

clear; clc;
global V_flow Vrxtr scaling_factor

V_flow = 2.3/60;  % liters/sec
Vrxtr = 150;   % liters
C_nut_in = 0.2;  % in Molar

%scaling factor for Ncells:
scaling_factor = V_flow*C_nut_in/1e-7;

% define the parameter values
params = param_set;

sc_Ncells_0 = 1e5/scaling_factor;   % scaled number of cells
C_nut_0 = 0.05;
C_P_0 = 0.05;

var_0 = [sc_Ncells_0; C_nut_0; C_P_0];

options = optimset('Display','iter','MaxFunEvals',10000,'MaxIter',1000,'TolX',1e-8,'TolFun',1e-
8');

[var,fval,exitflag,output,jacobian] = fsolve(@chemostat,var_0,options,C_nut_in,params);

jacobian_at_solution = jacobian
Jac_cond_number = cond(jacobian)
Jacobian_eigenvalues = eig(jacobian)

disp('  ');
disp(['Scaled Number of Cells in Reactor = ',num2str(var(1),'%8.6e')]);
disp(['Number of Cells in Reactor = ',num2str(var(1)*scaling_factor,'%8.6e')]);
disp(['Concentration of Nutrients (M) = ',num2str(var(2))]);
disp(['Concentration of Product (M) = ',num2str(var(3))]);
disp('  ');
disp(['Production Rate of Product (mole/hr) = ',num2str(var(3)*V_flow*3600)]);

resid_final = chemostat(var,C_nut_in,params)


return;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function resid = chemostat(var,C_nut_in,params)

global V_flow Vrxtr scaling_factor

sc_Ncells = var(1);
C_nut = var(2);
C_P = var(3);

CMrate = Cell_Multiplication(sc_Ncells,C_nut,C_P,params);
NCrate = Nutrient_Consumption(sc_Ncells,C_nut,C_P,params);
Prate = P_production(sc_Ncells,C_nut,C_P,params);

dNcell_dt = CMrate - V_flow*sc_Ncells/Vrxtr;

nutrient_bal = (C_nut_in - C_nut)*V_flow - NCrate;

P_bal =  Prate - V_flow*C_P;

resid = [dNcell_dt; nutrient_bal; P_bal];

return;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
> restart:
>
>
> Cell_growth := k1*Ncells*C_nut/((1 + c1*C_nut)*(1 + d*C_P));
```

$$Cell\_growth := \frac{k1\ Ncells\ C\_nut}{(1 + c1\ C\_nut)\ (1 + d\ C\_P)} \quad (1)$$

```
> dNcell_dt := k1*Ncells*C_nut/((1 + c1*C_nut)*(1 + d*C_P)) -
  V_flow*Ncells/Vrxtr;
```

$$dNcell\_dt := \frac{k1\ Ncells\ C\_nut}{(1 + c1\ C\_nut)\ (1 + d\ C\_P)} - \frac{V\_flow\ Ncells}{Vrxtr} \quad (2)$$

```
> nutrient_bal := (C_nut_in-C_nut)*V_flow - (k2*Ncells + c2*
  Cell_growth);
```

$$nutrient\_bal := (C\_nut\_in - C\_nut)\ V\_flow - k2\ Ncells - \frac{c2\ k1\ Ncells\ C\_nut}{(1 + c1\ C\_nut)\ (1 + d\ C\_P)} \quad (3)$$

```
> P_bal := -V_flow*C_P + k3*Ncells*exp(-d*C_P)*(C_nut - 0.01)^2/(1 +
  c1*C_nut);
```

$$P\_bal := -V\_flow\ C\_P + \frac{k3\ Ncells\ e^{(-d\ C\_P)}\ (C\_nut - 0.01)^2}{1 + c1\ C\_nut} \quad (4)$$

```
>
>
>
> J[1,1]:= diff(dNcell_dt,Ncells);
```

$$J_{1,1} := \frac{k1\ C\_nut}{(1 + c1\ C\_nut)\ (1 + d\ C\_P)} - \frac{V\_flow}{Vrxtr} \quad (5)$$

```
> J[1,2]:= diff(dNcell_dt,C_nut);
```

$$J_{1,2} := \frac{k1\ Ncells}{(1 + c1\ C\_nut)\ (1 + d\ C\_P)} - \frac{k1\ Ncells\ C\_nut\ c1}{(1 + c1\ C\_nut)^2\ (1 + d\ C\_P)} \quad (6)$$

```
> J[1,3]:= diff(dNcell_dt,C_P);
```

$$J_{1,3} := -\frac{k1\ Ncells\ C\_nut\ d}{(1 + c1\ C\_nut)\ (1 + d\ C\_P)^2} \quad (7)$$

```
>
> J[2,1]:= diff(nutrient_bal,Ncells);
```

$$J_{2,1} := -k2 - \frac{c2\ k1\ C\_nut}{(1 + c1\ C\_nut)\ (1 + d\ C\_P)} \quad (8)$$

```
> J[2,2]:= diff(nutrient_bal,C_nut);
```

$$J_{2,2} := -V\_flow - \frac{c2\ k1\ Ncells}{(1 + c1\ C\_nut)\ (1 + d\ C\_P)} + \frac{c2\ k1\ Ncells\ C\_nut\ c1}{(1 + c1\ C\_nut)^2\ (1 + d\ C\_P)} \quad (9)$$

```
> J[2,3]:= diff(nutrient_bal,C_P);
```

(10)

$$J_{2,3} := \frac{c2\, k1\, Ncells\, C\_nut\, d}{(1 + c1\, C\_nut)\,(1 + d\, C\_P)^2} \tag{10}$$

```
>
> J[3,1]:= diff(P_bal,Ncells);
```

$$J_{3,1} := \frac{k3\, e^{(-d\, C\_P)}\,(C\_nut - 0.01)^2}{1 + c1\, C\_nut} \tag{11}$$

```
> J[3,2]:= diff(P_bal,C_nut);
```

$$J_{3,2} := \frac{2\, k3\, Ncells\, e^{(-d\, C\_P)}\,(C\_nut - 0.01)}{1 + c1\, C\_nut} - \frac{k3\, Ncells\, e^{(-d\, C\_P)}\,(C\_nut - 0.01)^2\, c1}{(1 + c1\, C\_nut)^2} \tag{12}$$

```
> J[3,3]:= diff(P_bal,C_P);
```

$$J_{3,3} := -V\_flow - \frac{k3\, Ncells\, d\, e^{(-d\, C\_P)}\,(C\_nut - 0.01)^2}{1 + c1\, C\_nut} \tag{13}$$

```
>
>
>
```