Homework 2

Kathryn Atherton

ABE 30100

February 15, 2019

Intracellular biochemical reactions can be complicated. Assume adenosine triphosphate (ATP) is created and consumed within a cell at different rates by different reactions (see reaction rates below).

If the cell is at steady state, what is the ATP concentration in the cell?

Rates of reaction

r1(CATP) = + k1*CATP   r2(CATP) = - Vm*CATP/(Km+ CATP)   r3(CATP)= -k3* CATP^2

Value of reaction rate constants

k1=7/min   Vm=0.1 mol/L-min   Km=1 mol/L   k3= 0.02 L/mol-min

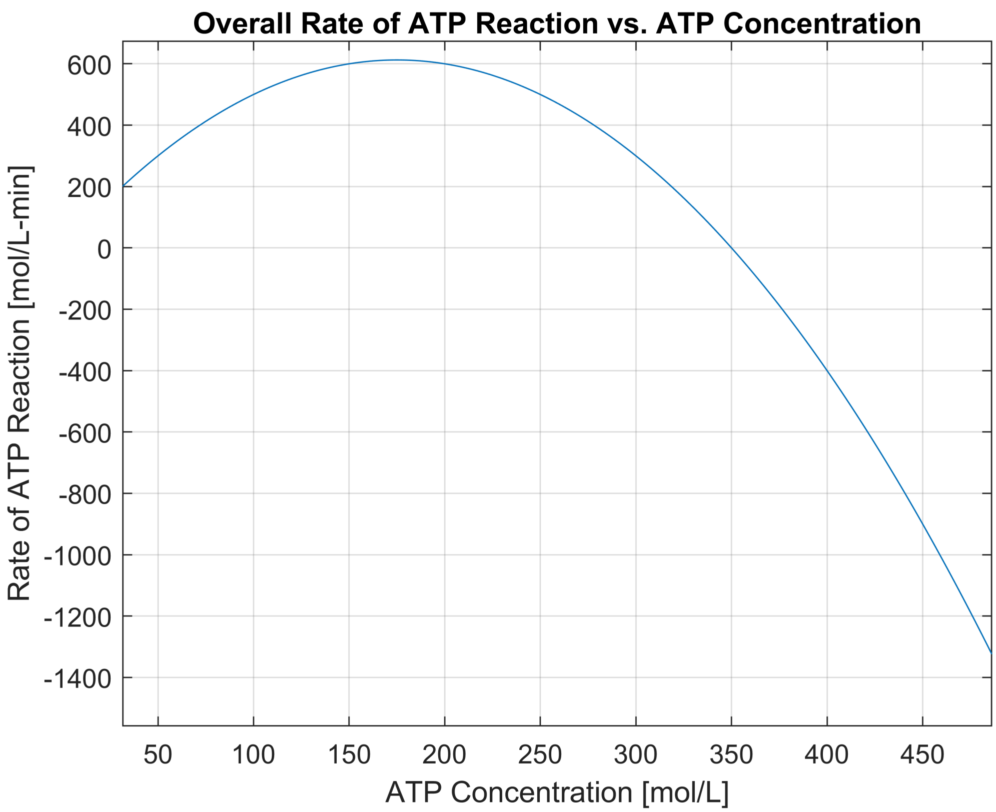
Part A:

(2 points) Provide an appropriate plot of the overall rate of ATP reaction vs. ATP concentration

```
k1 = 7; % [min^-1]
vm = 0.1; % [mol/L-min]
km = 1; % [mol/L]
k3 = 0.02; % [L/mol-min]

syms Catp % [mol/L]
r1 = k1 * Catp; % [mol/L-min]
r2 = - (vm * Catp) / (km + Catp); % [mol/L-min]
r3 = -k3 * Catp ^ 2; % [mol/L-min]

roverall = r1 + r2 + r3;
ezplot(roverall, [0,500])
set(gca,'XGrid','on','YGrid','on')
title('Overall Rate of ATP Reaction vs. ATP Concentration')
ylabel('Rate of ATP Reaction [mol/L-min]')
xlabel('ATP Concentration [mol/L]')
```



Part B:

(9 points) Please write computer programs using 3 different root finding methods and use each to calculate the steady state ATP concentration (CATP, mol/L) within an error limit of 0.0001 mol/L. (Note: you do not need to provide program code)

- Bisection method (use starting points 200 and 400)
- False position method (use starting points 200 and 400)
- Newton-Raphson method(starting point 300)

```
error_tol = 0.0001;

x1 = 200; % for bisection and regula falsi
y1 = double(subs(roverall,Catp,x1)); % for bisection and regula falsi
x2 = 400; % for bisection and regula falsi
y2 = double(subs(roverall,Catp,x2)); % for bisection and regula falsi
x3 = 300; % for Newton-Raphson

colNames = {'Iteration', 'Catp', 'rCatp'};
bisection_method = bisection(roverall, x1, y1, x2, y2, error_tol);
Table_bisection = array2table(bisection_method, 'VariableNames', colNames)
```

Table_bisection = 22×3 table

	Iteration	Catp	rCatp
1	1	300.0000	299.9003

	Iteration	Catp	rCatp
2	2	350.0000	-0.0997
3	3	325.0000	162.4003
4	4	337.5000	84.2753
5	5	343.7500	42.8690
6	6	346.8750	21.5800
7	7	348.4375	10.7890
8	8	349.2188	5.3568
9	9	349.6094	2.6316
10	10	349.8047	1.2667
11	11	349.9023	0.5837
12	12	349.9512	0.2420
13	13	349.9756	0.0712
14	14	349.9878	-0.0143
15	15	349.9817	0.0285
16	16	349.9847	0.0071
17	17	349.9863	-0.0036
18	18	349.9855	0.0018
19	19	349.9859	-0.0009
20	20	349.9857	0.0004
21	21	349.9858	-0.0003
22	22	349.9857	0.0001

```
false_position_method = regula_falsi(roverall, x1, y1, x2, y2, error_tol);
Table_false_position = array2table(false_position_method, 'VariableNames', colNames)
```

Table_false_position = 22×3 table

	Iteration	Catp	rCatp
1	1	319.9801	192.0159
2	2	367.9841	-132.4566
3	3	348.7777	8.4267
4	4	360.2996	-74.3185
5	5	355.6897	-40.5749
6	6	352.9242	-20.7400
7	7	351.2652	-8.9879
8	8	350.2699	-1.9906

	Iteration	Catp	rCatp
9	9	349.6729	2.1880
10	10	350.0310	-0.3171
11	11	349.8877	0.6858
12	12	349.9737	0.0843
13	13	350.0081	-0.1565
14	14	349.9943	-0.0601
15	15	349.9861	-0.0023
16	16	349.9811	0.0323
17	17	349.9841	0.0115
18	18	349.9853	0.0032
19	19	349.9858	-0.0001
20	20	349.9856	0.0012
21	21	349.9857	0.0004
22	22	349.9857	0.0001

```
newton_raphson_method = newton_raphson(roverall, x3, error_tol);
Table_newton_raphson = array2table(newton_raphson_method, 'VariableNames', colNames)
```

Table_newton_raphson = 4×3 table

	Iteration	Catp	rCatp
1	1	359.9801	-71.9521
2	2	350.2557	-1.8912
3	3	349.9860	-0.0015
4	4	349.9858	-0.0000

For each method provide the following:

- A table showing the calculated values of the estimated root, CATP, the value of the rate, r(CATP), and the number of iterations needed to reach the final value.

Part C:

(4 points) Based on your results, explain the benefits and limitations of each method.

Bisection Method benefits

- guaranteed convergence on a root (as long as there is a root between the starting points)
- generally a uniform convergence upon the root

Bisection Method limitations

- slowest method of the three

False Postition Method benefits

- guaranteed convergence on a root (as long as there is a root between the starting points)
- theoretically faster than bisection method (although here it did not perform the algorithm in less iterations)

False Position Method limitations

- convergence upon the root is not uniform (i.e. estimated zero goes from -132 to 8 to -74)

Newton Raphson Method benefits

- fastest method of the three
- only requires one point

Newton Raphson Method limitations

- requires the derivative to be known (can be difficult if the function is very complex)
- convergence depends on slopes between the starting point and the zero (i.e. if there is a maximum or minimum that the method converges on, the method fails; if the method converges near a maximum or minimum, the method takes more iterations to converge on the zer

Functions

```
function [matrix] = bisection(f, x1, y1, x2, y2, error_tol)
    % sets up the output matrix which will be formatted as a table later
    matrix = zeros(1,3);

    % sets zero to something greater than the error tolerance to start the
    % while loop
    zero = 1 + error_tol;

    % starts iteration counter
    i = 1;

    % checks that x1 < x2
    if x1 > x2
        fprintf('Error: x1 > x2.\n');
        zero = 'N/A';
        Catp = 'N/A';

    % checks that f(x1) and f(x2) are of opposite signs
    elseif ((y1 > 0) && (y2 > 0)) || ((y1 < 0) && (y2 < 0))
        fprintf('Error: y1 and y2 have the same sign.\n');
        zero = 'N/A';
        Catp = 'N/A';
```

```
    else
        % checks to see if another iteration should be performed
        while abs(zero) > error_tol
            % finds midpoint between x values
            Catp = (x2 + x1)/2;

            % finds the y value of the midpoint
            zero = double(subs(f));

            % replaces the x value of the same sign as the midpoint of the
            % x values with the midpoint
            if (y1 > 0) && (zero >= 0)
                x1 = Catp;
            elseif (y1 < 0) && (zero <= 0)
                x1 = Catp;
            else
                x2 = Catp;
            end

            % stores values in table to be output
            matrix(i, 1) = i;
            matrix(i, 2) = Catp;
            matrix(i, 3) = zero;

            % adds iteration to counter
            i = i + 1;
        end
    end
end

function [matrix] = newton_raphson(f, x1, error_tol)
    % sets up the output matrix which will be formatted as a table later
    matrix = zeros(1,3);

    % sets zero to the value of the function at the given x point
    Catp = x1;
    zero = double(subs(f));

    % starts iteration counter
    i = 1;

    % checks to see if another iteration should be performed
    while abs(zero) > error_tol
        % finds the slope of the function at the given point
        slope = double(subs(diff(f)));

        % checks for a minimum or maximum
        if slope == 0
            fprintf('Error: stuck at minimum or maximum of function.\n')

            % breaks the while loop so that the function doesn't go on
            % forever
            zero = 0;
```

```
            Catp = 'N/A';
        else
            % finds the b of the function y = mx + b
            b = zero - slope * Catp;

            % finds the new x where y = 0 for the linear function
            Catp = double(-b / slope);

            % finds the value of the function at the x found above
            zero = double(subs(f));

            % stores values in table to be output
            matrix(i, 1) = i;
            matrix(i, 2) = Catp;
            matrix(i, 3) = zero;

            % adds iteration to counter
            i = i + 1;
        end
    end

    % changes the zero value to N/A in the case that a maximum was found
    % after loop break
    if Catp == 'N/A'
        zero  = 'N/A';
    end
end


function [matrix] = regula_falsi(f, x1, y1, x2, y2, error_tol)
    % sets up the output matrix which will be formatted as a table later
    matrix = zeros(1,3);

    % sets zero to something greater than the error tolerance
    zero = 1 + error_tol;

    % starts iteration counter
    i = 1;
    % checks that x1 < x2
    if x1 > x2
        fprintf('Error: x1 > x2.\n');
        zero = 'N/A';
        Catp = 'N/A';

    % checks that y1 and y2 are of different signs
    elseif ((y1 > 0) && (y2 > 0)) || ((y1 < 0) && (y2 < 0))
        fprintf('Error: y1 and y2 have the same sign.\n');
        zero = 'N/A';
        Catp = 'N/A';
    else
        % checks to see if another iteration should be performed
        while abs(zero) > error_tol
            % calculates slope from y and x values
```

```
            slope  = (y2 - y1)/(x2 - x1);

            % calculates b from y = mx + b formula
            b = y2 - slope * x2;

            % calculates new x from y = mx + b
            Catp = -b / slope;

            % calculates new root from function at x calculated
            zero = double(subs(f));

            % replaces the x value of the same sign as the midpoint of the
            % x values with the midpoint
            if (y1 > 0) && (zero >= 0)
                x1 = Catp;
            elseif (y1 < 0) && (zero <= 0)
                x1 = Catp;
            else
                x2 = Catp;
            end

            % stores values in table to be output
            matrix(i, 1) = i;
            matrix(i, 2) = Catp;
            matrix(i, 3) = zero;

            % adds to the iteration counter
            i = i + 1;
        end
    end
end
```