

UNIVERSITY OF CALIFORNIA, DAVIS
DEPARTMENT OF CIVIL AND ENVIRONMENTAL ENGINEERING

COURSE: WATER RESOURCES SIMULATION (ECI 146)

INSTRUCTOR: Fabián A. Bombardelli

(fabianbombardelli2@gmail.com, bmbdrll@yahoo.com, fabombardelli@ucdavis.edu)

OFFICE: 3105, Ghausi Hall (former Engineering III building)

Class: Tuesdays and Thursdays-10:30 to 11:50 PM (119 Wellman)

TEACHING ASSISTANTS AND READER: Corrin Clemons, Doug Kubota, Arturo Palomino and Jhonatan Romero

COMPUTER PROBLEM 1: SOLUTION

Problem 1

- 1-1) Please define explicit and implicit algebraic equations. Please provide examples of these types of equations from the field of water resources.

An explicit algebraic equation may be written as a function of many variables such that one dependent variable is a function of the other set of independent variables. An implicit algebraic equation has no separation of dependent and independent variables. It can not be written as one variable in terms of the other variables.

Examples for explicit: Haaland equation

Examples for implicit: $\frac{1}{\sqrt{f}} = 2 \log_{10} \left(\frac{\varepsilon}{3.7D} + \frac{2.51}{Re\sqrt{f}} \right)$

- 1-2) What type of equations do you know? Please give examples of each type.
- Algebraic equations (can be explicit or implicit)
 - Differential equations
 - Integral equations
 - Combinations of a-c

1-3) What is the physical meaning of smooth and fully-rough regimes? In which area of the Moody chart is the Colebrook-White equation valid?

These are concepts you should recall from ECI141 or any equivalent Fluid Mechanics course. When the regime is turbulent smooth, the viscous sub-layer is much larger than the roughness height. Thus, the viscous sub-layer covers completely the roughness height and beyond. In the fully-rough regime, the viscous sub-layer is much smaller than the roughness height, i.e., the latter protrudes the viscous sub-layer. In this region, the friction factor only depends on the relative roughness of the pipe. The Colebrook-White equation is only valid for turbulent conditions (fully rough flow).

1-4) Please discuss the difference between the “true” and “approximate” errors in the bisection method. Why do we use the latter if these two errors are different?

True error is the absolute value of the difference between the exact root and the guessed root, divided by the former. Approximate error is the absolute value of difference between the last guess and the previous one, divided by the former. Since we do not know the exact root (because it would imply knowing it with infinite number of digits), we do not know the true error, so we have to use the approximate error as the stopping criterion in iterative algorithms. Since the approximate error is larger than the true error, we are safe in using the former value of the error.

1-5) Which method is more robust? Which method is more accurate? What do we mean by robust? What do we mean by accurate?

By the word “robust,” we refer to a method which provides a solution regardless of the function of which we would like to find a root.

Among the three methods used in the problem below, bisection is a robust method. It always converges to the solution regardless of how the function behaves. Obviously, there is a price for this desirable capability: The bisection method is not as fast as Newton-Raphson or Iteration-of-a-point (Fixed-point) methods (there are exceptions, but roughly speaking bisection is slower than the other alternatives). The initial guess, inherent nature of the solution, and search interval are crucial factors and may change the situation.

The accuracy of all methods is associated with the desired error tolerance. Therefore, if the error tolerance is set identically, *all* iterative algorithms will achieve the same accuracy.

Problem 2

- a) Please develop a flow chart for the bisection method used to calculate the normal depth for a trapezoidal channel.

- b) Compute the normal depth for a trapezoidal channel, using the methods of bisection, Newton-Raphson, and fixed point, given the following information for an asphalt channel: $Q=300 \text{ ft}^3/\text{s}$; $S=0.0015$; lateral sides angle=50 degrees; bottom width=6 ft. Please discuss suitable values of the normal depth to start the computations, and select three values.

ECI 146	CP1	Corrin Clemons
<p>Given: $Q = 300 \frac{\text{ft}^3}{\text{s}}$ $S = 0.0015$ $\theta = 50^\circ$ $b = 6 \text{ft}$ asphalt channel $\rightarrow n = 0.016$</p>	<p>$K_n = 1.49$ (Eng units)</p>	
$v = \frac{K_n R_h^{2/3} S_f^{1/2}}{n}$	$R_h = \frac{A}{P}$	
$Q = Av$	$A = \frac{1}{2}(a+b)y$	$P = 2s + b$
$Q = \frac{K_n R_h^{2/3} S_f^{1/2} \cdot A}{n}$	$a = 2k + b$	$\cos \phi = \frac{y}{s}$
$Q = \frac{K_n A^{5/3} S_f^{1/2}}{n^{2/3}}$	$\tan \phi = \frac{k}{y}$	$s = \frac{y}{\cos \phi}$
	$k = y \tan \phi$	$P = \frac{2y}{\cos \phi} + b$
	$a = 2[y \tan \phi] + b$	
	$A = \frac{1}{2}(2y \tan \phi + 2b)y$	
	$A = y^2 \tan \phi + yb$	
$Q = \frac{K_n (y^2 \tan \phi + yb)^{5/3} S_f^{1/2}}{(\frac{2y}{\cos \phi} + b)^{2/3} \cdot n}$		
$F(y) = \frac{K_n (y^2 \tan \phi + yb)^{5/3} S_f^{1/2}}{(\frac{2y}{\cos \phi} + b)^{2/3} \cdot n} - Q$		
Bisection:		

Matlab code for bisection method:

```

%% ECI 146 CP1 Problem 2 Solution
% Created by: Corrin Clemons
% Date: 2023-02-05

% Methods based on Handouts 8 & 9

%% Bisection Method

y1 = 0.1; % guess lower bounds for channel height
yu = 9; % guess upper bounds for channel height
yr = -1; % filler variable for root (impossible solution)
tol = 1E-6; % set a tolerance for F (should be close to 0)
iter = 0; % iteration count

while (abs(F_trap(yr)) > tol)

```

```

iter = iter + 1;
yr = (yl + yu)/2;

if F_trap(yr)*F_trap(yl) < 0
    yu = yr;
else
    yl = yr;
end

end

fprintf('The root is %d.\n',yr)
fprintf('This algorithm took %d iterations. \n',iter)

```

Function for calculating f(y):

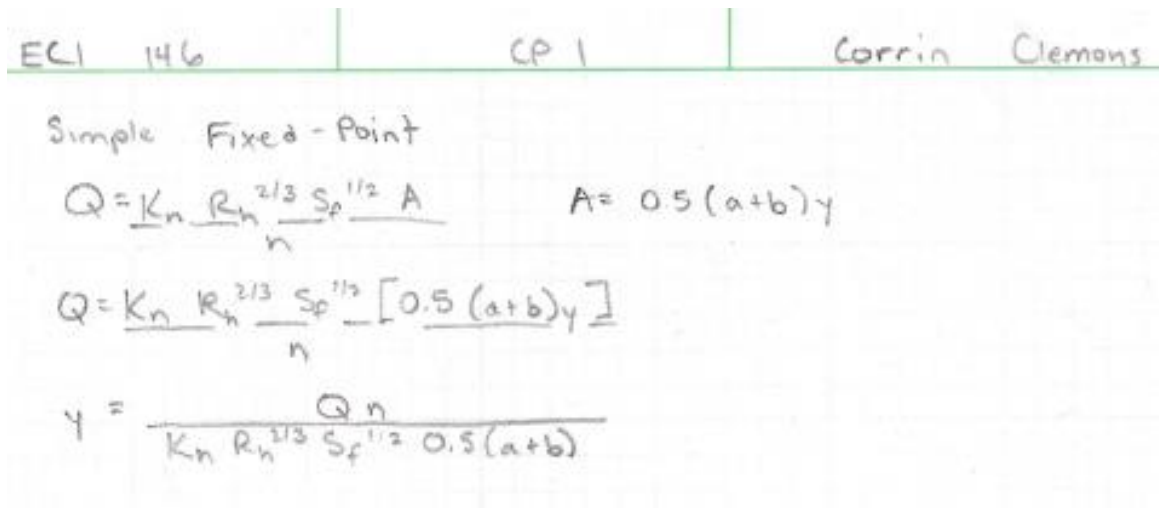
```

function F = F_trap(y)
% Givens
Q = 300; % ft3/s
Sf = 0.0015;
theta = 50; % deg
b = 6; % ft
n = 0.016;
Kn = 1.49; % Eng units

% Preliminary calcs
phi = 90 - theta;
s = y/cosd(phi); % length of side
P = 2*s + b; % wetted perimeter (trapezoid)
k = y*tand(phi);
a = 2*k + b;
Area = 0.5*(a+b)*y; % cross-sectional area (trapezoid)
Rh = Area/P; % hydraulic radius
F = Kn*Rh^(2/3)*Sf^(1/2)*Area/n - Q;

end

```



```

%% Simple Fixed-Point

```

```

tol = 1E-4; % set a tolerance for the approx error (should be close to 0)
Ea = 1; % filler variable for approx error
iter = 0; % iteration count
y1 = 4; % guess (low) for channel height

while (Ea > tol)
    y2 = F_trap_fixed(y1);
    Ea = abs((y2-y1)/y2)*100;
    iter = iter + 1;
    y1 = y2;
end

fprintf('The root is %d.\n',y2)
fprintf('This algorithm took %d iterations. \n',iter)
fprintf('The approximate error is %d. \n',iter)

```

Function for calculating $y = g(y)$:

```

function G = F_trap_fixed(y)
    % Givens
    Q = 300; % ft3/s
    Sf = 0.0015;
    theta = 50; % deg
    b = 6; % ft
    n = 0.016;
    Kn = 1.49; % Eng units

    % Preliminary calcs
    phi = 90 - theta;
    s = y/cosd(phi); % length of side
    P = 2*s + b; % wetted perimeter (trapezoid)
    k = y*tand(phi);
    a = 2*k + b;
    Area = 0.5*(a+b)*y; % cross-sectional area (trapezoid)
    Rh = Area/P; % hydrolic radius
    G = (Q*n)/(Kn*(Rh^(2/3))*(Sf^0.5)*0.5*(a+b));
end

```

Newton - Raphson

$$F(y) = \frac{K_n (y \tan \phi + y b)^{5/3} S^{1/2}}{\left(\frac{2y}{\cos \phi} + b\right)^{2/3} \cdot n} - Q$$

$$F(y) = \frac{1.49 [y^2 \tan(40^\circ) + y(6ft)]^{5/3} (0.0015)^{1/2}}{\left(\frac{2y}{\cos(40^\circ)} + 6ft\right)^{2/3} \cdot (0.016)} - 300 \frac{ft^3}{s}$$

↳ Wolfram Alpha

$$F'(y) = \frac{5.08403(y^2 + 5.55411y + 10.2705)(y(y \tan(40^\circ) + 6))^{2/3}}{(y + 2.29813)(y \sec(\frac{2\pi}{9}) + 3)^{2/3}}$$

$$y_{i+1} = y_i - \frac{F(y_i)}{F'(y_i)}$$

%% Newton-Raphson

```
tol = 1E-4; % set a tolerance for the approx error (should be close to 0)
Ea = 1; % filler variable for approx error
iter = 0; % iteration count
y1 = 4; % guess (low) for channel height
```

```
while (Ea > tol)
    y2 = y1 - F_trap(y1)/F_trap_deriv(y1);
    Ea = abs((y2-y1)/y2)*100;
    iter = iter + 1;
    y1 = y2;
end
```

```
fprintf('The root is %d.\n',y2)
fprintf('This algorithm took %d iterations. \n',iter)
fprintf('The approximate error is %d. \n',iter)
```

Function for calculating $f'(y)$:

```
function F = F_trap_deriv(y)
    % Derivative obtained using Wolfram Alpha

    % Preliminary calcs
    num = 5.08403*(y^2 + 5.55411*y + 10.2705)*(y*(y*tand(40)+6))^(2/3);
    denom = (y + 2.29813)*(y*secd(40) + 3)^(2/3);

    F = num/denom;
end
```

Problem 3

a) Sample Programs in MATLAB

```
%%  
% University of California, Davis  
% Department of Civil and Environmental Engineering  
% ECI 146  
% Instructor: Prof. Fabian Bombardelli  
% Sample program for Newton-Raphson method  
% This program finds the root of "5x^2-4=0" with Newton  
% Raphson method  
% Code by: Kaveh Zamani  
%% initialization  
clear all  
% clears all the current variables from the memory  
clc  
% clears the command window  
tolerance = input('Stopping criteria tolerance ?')  
x_initial = 0.5;  
x_old= x_initial;  
error = 1;  
% one is assigned to error to enter the below while loop  
iteration_number = 0;  
%% Newton-Raphson While loop  
while (error > tolerance)  
    f_x = 5*x_old^2 - 4;  
    f_prim_x = 10*x_old;  
    x_new = x_old - f_x/f_prim_x;  
    error = abs((x_old - x_new)/x_new)*100;  
    iteration_number = iteration_number + 1;  
    % the variables/expressions without semicolon at the end are  
    printed on the screen  
    iteration_number  
    x_old = x_new  
end  
  
disp('Newton_Raphson method has converged!')  
X = x_new
```

```

% University of California, Davis
% Department of Civil and Environmental Engineering
% ECI 146
% Instructor: Prof. Fabian Bombardelli
% by Kaveh Zamani
% This program finds the root of "x=cos(x)" between 0 and 1 by the
% method of False Position
% method is also called "Regula Falsi" method.
%% initialization
clear all
% clears all the current variables from the memory
clc
% clears the command window
tolerance = 0.000001;
max_num_iteration = 10000;
x_left_initial = 0;
x_right_initial = 1;
x_left = x_left_initial;
x_right = x_right_initial;
x_old = 2;

%% False-Position loop
for i=1:max_num_iteration;

    f_left = x_left - cos(x_left);
    f_right = x_right - cos(x_right);
    x_mid = - f_right*(x_right - x_left)/(f_right - f_left) + x_right;
    f_mid = x_mid - cos(x_mid);
    x_new = x_mid;

    error = abs((x_new-x_old)/x_old)*100;

    if(error < tolerance)
        disp('False Position method has converged!');
        X = x_mid
        break
        % break out of if condition
    elseif ( f_right*f_mid > 0)
        x_right = x_mid;
    else
        x_left = x_mid;
    end
    iteration_number = iteration_number + 1;
% the variables/expressions without semicolon at the end are printed on
the screen
    x_old = x_new;

end

```



```

%
% University of California, Davis
% Department of Civil and Environmental Engineering
% ECI 146 Water Resource Simulation
% Instructor: Prof. Fabian Bombardelli
% Sample program for bisection method
% by Kaveh Zamani
% This program finds the root of "x^2-2x=0" between 1 and 4 by
Bisection method
%
%% initialization
clear all
% clears all the current variables from the memory
clc
% clears the command window
tolerance = 0.000001;
max_num_iteration = 10000;
x_left_initial = 1;
x_right_initial = 4;
x_left = x_left_initial;
x_right= x_right_initial;

%% Bisection loop
for i=1:max_num_iteration;
    x_mid_new = (x_left + x_right)/2;
    f_left = x_left^2 - 2*x_left;
    f_right = x_right^2 - 2*x_right;
    f_mid = x_mid_new^2 - 2*x_mid_new;

    error = abs((x_mid_new - x_mid_old)/x_mid_old)*100;

    if(error < tolerance)
        disp('Bisection method has converged!');
        X = x_mid_new
        break
        % break out of if condition
    elseif ( f_right*f_mid > 0)
        x_right = x_mid_new;
    else
        x_left = x_mid_new;
    end
    iteration_number = i
    % the variables/expressions without semicolon at the end are
    printed on the screen
    x_mid_old = x_mid_new;
end

```

b) Sample program written in Fortran for bisection with using the function as part of the stopping criterion:

```
!Calculation of the friction factor using
!Colebrook-White's formula

!Enter the data

    write(*,*) 'Enter roughness Ratio'
    read(*,*) ED
    write(*,*) 'Enter Reynolds number'
    read(*,*) RE

!Interval

    write(*,*) 'Enter minimum value'
    read(*,*) FMIN
    write(*,*) 'Enter maximum value'
    read(*,*) FMAX

!Iteration procedure

    iter=0
10   iter=iter+1
      FR=0.5*(FMIN+FMAX)
      F=1.+sqrt(FR)*0.869*log(ED/3.7+2.51/(RE*sqrt(FR)))
      if (F.gt.0.) FMIN=FR
      if(F.lt.0.) FMAX=FR
      if (abs(F).gt.1.e-6) go to 10

!output: friction factor and number of iteration
    write (*,*) 'the friction factor is', FR
    write(*,*) iter
end
```

The above FORTRAN code follows the flow chart of the notebook.

c) Results:

Bisection Method's Number of Iterations for Different Tolerances (Relative Error)							
Number	e/D	Re	F	# of iterations			Region
				10 ⁻²	10 ⁻³	10 ⁻⁴	
1	0.0008	3.0×10 ⁶	0.01872	16	19	22	Fully rough/Transition
2	0.00005	3.0×10 ⁶	0.01145	17	19	23	Transition
3	0.00001	3.0×10 ⁷	0.00843	17	20	23	Transition
4	0.002	3.0×10 ⁷	0.02341	16	18	22	Fully rough
5	0.015	3.0×10 ⁷	0.04364	15	17	21	Fully rough
6	10 ⁻¹⁰	3.0×10 ⁵	0.01445	17	19	22	Smooth
7	0.002	3.0×10 ⁵	0.02400	16	18	21	Transition
8	0.03	3.0×10 ⁵	0.05722	15	17	20	Fully rough
9	0.002	3.0×10 ⁴	0.02807	16	18	21	Transition
10	0.01	3.0×10 ⁴	0.03979	15	17	21	Transition
11	10 ⁻¹⁰	3.0×10 ²	0.21333	No iteration $f = \frac{64}{Re}$			Laminar

NOTE: The number of iterations depends on initial interval and tolerance level. (Even compiler settings may change it.)

We can see the effect of the tolerance on the number of iterations. Since the tolerance is inversely proportional to the accuracy, when we increase the required accuracy (by decreasing the tolerance), we increase the number of iterations too, as shown in the table above.

Final remarks:

- Please keep in mind that we are controlling the accuracy through the tolerance; the issue here is how fast we obtain the solution.
- 15 to 20 iterations are not too fast to obtain a root. (Imagine that you need to perform this computation thousands of times.) This is why this is a “beginning method.” The best solution is to combine this with a “refining” method.
- How fast a method goes to the root is called “convergence,” while whether a method is capable of providing solutions or not for any given function is called “robustness.”

Problem 4

- Bernoulli equation: $\frac{P_1}{\rho g} + z_1 + \frac{V_1^2}{2g} = \frac{P_2}{\rho g} + z_2 + \frac{V_2^2}{2g} + h_f$
 $P_1 = P_2 = P_{atm}$ and $V_1 = V_2 = 0$, so $h_f = \Delta z$
 $h_f = f \frac{L V^2}{D 2g} \Rightarrow 8 = f \frac{100}{0.3} \frac{V^2}{2 \times 9.81}$ (I) \Rightarrow two unknowns and one equation

$$V = \left(\sqrt{\frac{2gDh_f}{L}} \right) \frac{1}{\sqrt{f}}$$

- Colebrook–White equation:

$$\frac{1}{\sqrt{f}} = -2 \log_{10} \left(\frac{\epsilon/D}{3.7} + \frac{2.51}{Re\sqrt{f}} \right) = -2 \log_{10} \left(\frac{0.0002}{3.7} + \frac{2.51}{\frac{0.3V}{2 \times 10^{-5}} \sqrt{f}} \right) \text{ (II)} \Rightarrow \text{two unknowns and two equations}$$

\Rightarrow (I) and (II) system of equations for V and f
Starting with $f=0.002$ and $V = 5$ m/s we have:

$f= 0.0201$ and $V=4.841$ m/s

The same result can be obtained by directly replacing (I) in (II).

Note

If you want to check your calculations online, you can insert your equation in <http://www.wolframalpha.com/> It is a solver by Wolfram Mathematica.

Or you can use MATLAB for numerical solution of implicit equations, just type:
`x=solve('x^2-x*sin(x)-3=0')` and hit Enter key

$$x = -2.1873720723545800214091973612386$$