THE UNIVERSITY of EDINBURGH
School of Engineering

Roots of Equations: Engineering Example

1. Previously, we covered a wide range of root-finding numerical techniques that you might expect to use in your engineering career.
2. Today we look at one specific example of how such techniques might be used to consolidate our understanding of these techniques further.
3. This example comes directly from **Chapra Chapter 8, Section 8.4 p. 209.**
4. In this exercise we study the calculation of pipe friction and fluid flow in a pipe.
5. The **resistance to flow in a pipe, f** is calculated using the following implicit *Colebrook* equation:

$$0 = \frac{1}{\sqrt{f}} + 2.0 \log\left(\frac{\varepsilon}{3.7D} + \frac{2.51}{\mathrm{Re}\sqrt{f}}\right)$$

6. Here, f is the friction factor, D is the pipe diameter [m], ε is the roughness of the pipe surface [m], and Re is the Reynold’s Number.

THE UNIVERSITY *of* EDINBURGH
School of Engineering

Roots of Equations: Engineering Example

7. In a typical situation, we will know the diameter of a pipe, and we can calculate the Re number from the velocity of flow and the density and viscosity of the fluid as follows:

$$\mathrm{Re} = \frac{\rho VD}{\mu}$$

8. In this problem, the properties of the fluid are fixed, and the engineer has the option of varying velocity in the pipe (or its diameter by redesign) to control fluid flow and pressure drop (which itself is a strong function of f.

9. The input values for this problem are:

Velocity, V	Viscosity, μ	Density, ρ	Diameter, d
$m.s^{-1}$	$Ns.m^{-2}$	$kg.m^{-3}$	m
40	1.79×10^{-5}	1.23	0.005

10. Pipe roughness, ε, is 0.0015 m

THE UNIVERSITY *of* EDINBURGH
School of Engineering

Roots of Equations: Engineering Example

1. We have an independent means of checking the value of f that we will calculate numerically using an explicit expression for f called the Swamee-Jain equation:

$$f = \frac{1.325}{\left[\ln\left(\frac{\varepsilon}{3.7D} + \frac{5.74}{\text{Re}^{0.9}}\right)\right]^2}$$

2. **SOLUTION:** To solve the problem we calculate as many variables in the Colebrook equation as we can from known information first: the obvious parameter we can calculate is Re

$$\text{Re} = \frac{\rho V D}{\mu} = \frac{1.23(40)0.005}{1.79 \times 10^{-5}} = 13{,}743$$

3. We can now substitute this value into Colebrook to get the following expressic

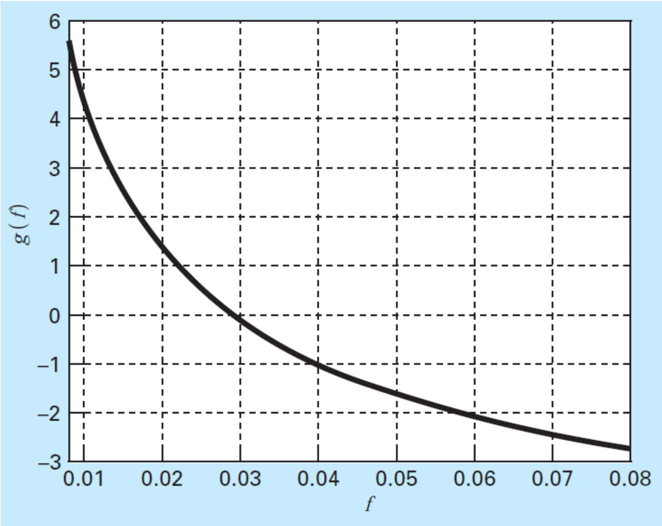
$$g(f) = \frac{1}{\sqrt{f}} + 2.0 \log\left(\frac{0.0000015}{3.7(0.005)} + \frac{2.51}{13{,}743\sqrt{f}}\right)$$

THE UNIVERSITY of EDINBURGH
School of Engineering

Roots of Equations: Engineering Example

4. There is a difference between this version of the Colebrook expression and the one in introduced initially.
5. Note, that the left hand side is not 0, but an unknown function g(f), i.e., the test function for the numerical method.
6. To solve this problem, we need to find the value of f that makes g(f) = 0.
7. **STEP 1: GRAPH THE FUNCTION:**

```
>> rho=1.23;mu=1.79e-5;D=0.005;V=40;e=0.0015/1000;
>> Re=rho*V*D/mu;
>> g=@(f) 1/sqrt(f)+2*log10(e/(3.7*D)+2.51/(Re*sqrt(f)));
>> fplot(g,[0.008 0.08]),grid,xlabel('f'),ylabel('g(f)')
```



Roots of Equations: Engineering Example

1. The graph gives us some immediate information about this function and its likely solution
   a. **A solution actually exists** for g(f) = 0.
   b. The solution is in the region of f = 0.03.
   c. **The function is continuous and differentiable** in the region of the route.
   d. I can use a bracketing method or an open method to solve this equation.
   e. **I know which initial values to use** to start a wide range of numerical methods.
2. Now, I am free to choose my own initial values and numerical technique to solve this problem, having confidence that it is **well-posed.**
3. Now, I need to choose a technique or a selection of techniques that I think might solve the problem.
4. Let's start with the bisection technique that can be implemented in a MATLAB code as follows:

Roots of Equations: Engineering Example

4. The pseudocode for bisection is as follows:

```
FUNCTION Bisect(xl, xu, es, imax, xr, iter, ea)
  iter = 0
  fl = f(xl)
  DO
    xrold = xr
    xr = (xl + xu) / 2
    fr = f(xr)
    iter = iter + 1
    IF xr ≠ 0 THEN
      ea = ABS((xr − xrold) / xr) * 100
    END IF
    test = fl * fr
    IF test < 0 THEN
      xu = xr
    ELSE IF test > 0 THEN
      xl = xr
      fl = fr
    ELSE
      ea = 0
    END IF
    IF ea < es OR iter ≥ imax EXIT
  END DO
  Bisect = xr
END Bisect
```

To use this algorithm, I need to convert it into Python syntax, and then define the function f(x) to be the one of interest here.

Computational Methods and Modelling

Lecture 2 Examples: Dr. Edward McCarthy

Finding Roots of Equations Numerically

THE UNIVERSITY of EDINBURGH
School of Engineering

Exercise 1: Bisection Code to Solve Colebrook

```
def bisection(f,a,b,N):
    if f(a)*f(b) >= 0:
        print("Bisection method fails.")
        return None
    a_n = a
    b_n = b
    for n in range(1,N+1):
        m_n = (a_n + b_n)/2
        f_m_n = f(m_n)
        if f(a_n)*f_m_n < 0:
            a_n = a_n
            b_n = m_n
        elif f(b_n)*f_m_n < 0:
            a_n = m_n
            b_n = b_n
        elif f_m_n == 0:
            print("Found exact solution.")
            return m_n
        else:
            print("Bisection method fails.")
            return None
    return (a_n + b_n)/2
f = lambda x: x**3 - x - 1
approx_phi = bisection(f,1,2,25)
print(approx_phi)
```

Exercise 1 Input and test this code for any quadratic equation and compare with classic exact quadratic solution. **Then solve the Colebrook equation**

Exercise 2: False Position Code

```
import math

def f(x):
    return x**10-2*(x**2)+5

def root(a, b):
    return b - (f(b)*((a-b)/(f(a)-f(b))))

def regF(a, b):
    itr = 0
    maxItr = 100
    with open('rfValues.csv', 'w') as f:
        f.write('#iteration, f(a), f(b), currentRoot\n')

        while (itr < maxItr):
            r = root(a,b)
            if (r < 0):
                b = r
            else:
                a = r

f.write('str(itr)'+','+'str(f(a))'+','+'str((f(b)))' + ',' +
'str(r)' + '\n')
            itr = itr + 1
    return r
rootVal = regF(0,1)
print ("Value of root is: " + str(rootVal))
```

THE UNIVERSITY of EDINBURGH
School of Engineering

Exercise 3: Newton Raphson

```
Def newton(f,Df,x0,epsilon,max_iter):
xn = x0
    for n in range(0,max_iter):
        fxn = f(xn)
        if abs(fxn) < epsilon:
            print('Found solution after',n,'iterations.')
            return xn
        Dfxn = Df(xn)
        if Dfxn == 0:
            print('Zero derivative. No solution found.')
            return None
        xn = xn - fxn/Dfxn
    print('Exceeded maximum iterations. No solution found.')
    return None

f = lambda x: x**4 - x - 1
df= lambda x: 4*x**3 - 1
x0=1
epsilon=0.001
max_iter=100
solution = newton(f,df,x0,epsilon,max_iter)
print(solution)
```

Exercise 4: Secant Technique

```
def secant(f,a,b,N):
    if f(a)*f(b) >= 0:
        print("Secant method fails.")
        return None
    a_n = a
    b_n = b
    for n in range(1,N+1):
        m_n = a_n - f(a_n)*(b_n - a_n)/(f(b_n) - f(a_n))
        f_m_n = f(m_n)
        if f(a_n)*f_m_n < 0:
            a_n = a_n
            b_n = m_n
        elif f(b_n)*f_m_n < 0:
            a_n = m_n
            b_n = b_n
        elif f_m_n == 0:
            print("Found exact solution.")
            return m_n
        else:
            print("Secant method fails.")
            return None
    return a_n - f(a_n)*(b_n - a_n)/(f(b_n) - f(a_n))

f = lambda x: x**4 - x - 1
solution = secant(f,1,2,25)
print(solution)
```