Exercise 1

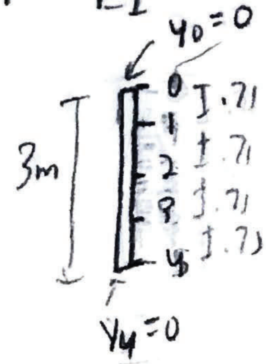
$$\frac{d^2y}{dx^2} + p^2 y = 0 \rightarrow \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + p^2 y_i = 0$$

$$-\left(\frac{y_{i+1}}{h^2} + \left(-2/h^2 + p^2\right) y_i + \frac{y_{i-1}}{h^2}\right) = 0 \qquad p^2 = \frac{P}{EI}$$

$$-\cancel{y_0/h^2}^{\,0} + (2/h^2 - p^2) y_1 - y_2/h^2$$

$$-y_1/h^2 + (2/h^2 - p^2) y_2 - y_3/h^2 \qquad h = 0.75\text{m}$$

$$-y_2/h^2 + (2/h^2 - p^2) y_3 - \cancel{y_4/h^2}^{\,0}$$



$$\begin{bmatrix} (2/h^2 - p^2) & -1/h^2 & 0 \\ -1/h^2 & (2/h^2 - p^2) & -1/h^2 \\ 0 & -1/h^2 & (2/h^2 - p^2) \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \vec{0}$$

need lowest eigenvalue

$$\begin{bmatrix} 2/h^2 & -1/h^2 & 0 \\ -1/h^2 & 2/h^2 & -1/h^2 \\ 0 & -1/h^2 & 2/h^2 \end{bmatrix} - \underbrace{\begin{bmatrix} p^2 & 0 & 0 \\ 0 & p^2 & 0 \\ 0 & 0 & p^2 \end{bmatrix}}_{\lambda I} = \vec{0} \qquad [A - \lambda \cdot I]\vec{x} = \vec{0}$$

when $\lambda = p^2$

$A \xrightarrow{\text{inverted}} A^{-1}$

5 iterations $\boxed{\text{Power method}}$ $\rightarrow 1/\lambda \rightarrow \lambda = p^2$

$$p^2 = 1.041398 \rightarrow P = EIp^2 = \boxed{130174.77\text{ N}}$$

Exercise 1

$$k = \frac{k_{max} c^2}{c_s + c^2} \qquad \frac{1}{k} = \frac{c_s + c^2}{k_{max} c^2}$$

$$\boxed{\frac{1}{k} = \frac{c_s}{k_{max}} \frac{1}{c^2} + \frac{1}{k_{max}}} \rightarrow \boxed{y^* = b^* x^* + a^*} \rightarrow \text{From: LSRL}$$

$$y^* = \frac{1}{k} = k_T \quad x^* = \frac{1}{c^2} = c_T$$

$$b^* = \frac{c_s}{k_{max}} \quad a^* = \frac{1}{k_{max}}$$

$$\underset{\text{transformed}}{k_T} = b^* c_T + a^*$$

$b^* = 0.202489$

$a^* = 0.099396$

$$\frac{c_s}{k_{max}} = b^* \overset{\text{from LSRL}}{=} 0.202489 = \frac{c_s}{k_{max}} \qquad \frac{1}{k_{max}} = 0.099396$$

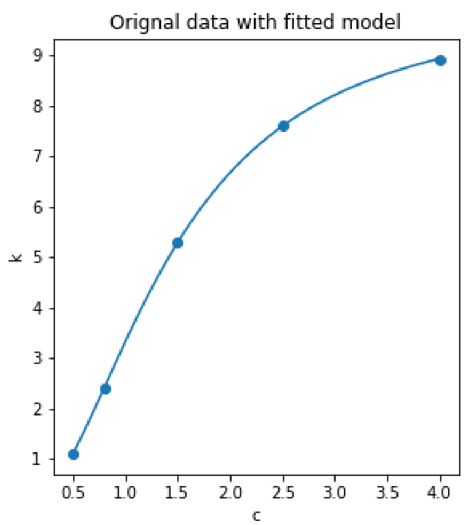
$$\boxed{c_s = 2.0372} \qquad \boxed{k_{max} = 10.061}$$

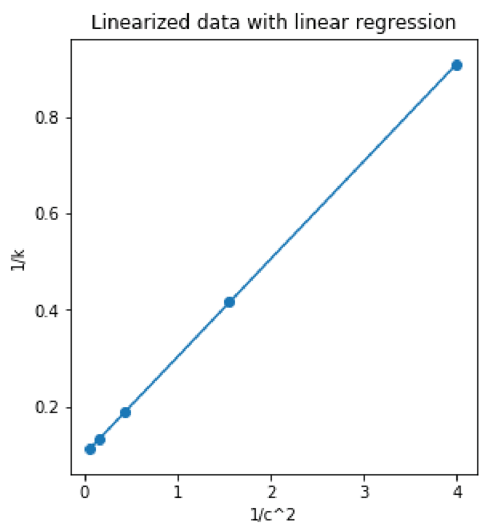
Back to original equation

$$k = \frac{k_{max} c^2}{c_s + c^2} = \frac{10.061 c^2}{2.0372 + c^2}$$

$$\boxed{k(c) = \frac{10.061 c^2}{2.0372 + c^2}} \qquad k(c = 2\,mg/L) = 6.68$$

```
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.4.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/hoops/OneDrive/Documents/School/ME EN 2450 Numerical Methods/
HW6/HW6a.py', wdir='C:/Users/hoops/OneDrive/Documents/School/ME EN 2450 Numerical Methods/
HW6')
Exercise 0:
Analytical Solution: P = 137077.83890401886 N
Using the power method with 5 nodes and 1 iterations: p^2 = 1.0427706256679103 and thus P
= 130346.32820848879 N
Using the power method with 5 nodes and 2 iterations: p^2 = 1.0418474763101524 and thus P
= 130230.93453876906 N
Using the power method with 5 nodes and 3 iterations: p^2 = 1.0414011866938964 and thus P
= 130175.14833673704 N
Using the power method with 5 nodes and 4 iterations: p^2 = 1.041398149896716 and thus P =
130174.7687370895 N
Using the power method with 5 nodes and 5 iterations: p^2 = 1.0413981712558384 and thus P
= 130174.77140697981 N

Determine level of discretization:
Using the power method with 11 nodes and 10 iterations: p^2 = 1.08763297121887 and thus P
= 135954.12140235875 N and error percentage from analytical solution is 0.82%
Through testing it was found that discretization with 11 nodes gives a 0.819765988904254%
error with 10 iterations



Exercise 1:
Regression slope: 0.20248898968029247  Regression intercept: 0.09939628142914936
kmax = 10.060738546972804  cs = 2.0371887838140967
Predicted growth rate at c = 2 mg/L = 6.665843263968141
```






```
In [2]:
```

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Apr  3 18:03:09 2019
HW06a
@author: Ryan Dalby
"""
import numpy as np
import matplotlib.pyplot as plt

#Exercise 0
E = 10.0e9 #Pa
I = 1.25e-5 #m^4
L = 3.0 #m
n = 1 #mode number

print("Exercise 0:")
PAnalytical = (n**2 * np.pi**2 * E * I) / (L**2) #analytical solution
print("Analytical Solution: P = {} N".format(PAnalytical))

def powerMethod(A, numIters):
    """
    Given an input coefficent matrix and number of iterations 
    Returns the resulting dominant eigenvector(bk) and eigenvalue pair
    """
    length = np.shape(A)[0]
    bk = np.random.rand(length)
    
    for i in range(numIters):
        numerator = np.dot(A, bk)
        bk =  numerator/np.linalg.norm(numerator)
        
    eigenvalue = np.linalg.norm(np.dot(A, bk))  ##uses final bk value to find the eigenvalue
    return bk, eigenvalue

h=0.75
A = np.array([[2/h**2,-1/h**2,0],[-1/h**2,2/h**2,-1/h**2],[0,-1/h**2,2/h**2]]) #matrix in which its
for i in range(1,6):
    _,invEigenval = powerMethod(np.linalg.inv(A), i) #inverse to find 1 / (smallest eigenvalue) wh
    littlepsquared = 1/invEigenval #gets little p squared
    bigP = E * I * littlepsquared #find P from p^2
    print("Using the power method with 5 nodes and {} iterations: p^2 = {} and thus P = {} N".forma

#Determine level of discretization
print("\nDetermine level of discretization:")
numNodes = 11
iterations = 10
h = 3.0 / (numNodes-1)
diagNum = 2.0/(h**2)
n = numNodes - 2
sideDiagNum = -1.0/(h**2)
A = np.diag(np.full((n),diagNum)) + np.diag(np.full((n-1),sideDiagNum), 1) + np.diag(np.full((n-1),
_,invEigenval = powerMethod(np.linalg.inv(A), iterations) #inverse to find smallest eigenvalue whic
littlepsquared = 1/invEigenval #gets little p squared
bigP = E * I * littlepsquared #find P from p^2
errorPercent = ((PAnalytical - bigP) / (PAnalytical)) * 100
print("Using the power method with {} nodes and {} iterations: p^2 = {} and thus P = {} N and error
```

```
print("Through testing it was found that discretization with {} nodes gives a {}% error with {} ite


print("\n\n")
#Exercise 1
print("Exercise 1:")
c = np.array([0.5,0.8,1.5,2.5,4])
k = np.array([1.1, 2.4, 5.3, 7.6, 8.9])
n = np.size(c)

cT = 1/(c**2) #Linearized c (Transformed)
kT = 1 /k #Linearized k (Transformed)

slope = (n * np.sum(cT*kT) - (np.sum(cT) * np.sum(kT))) / (n * np.sum(cT**2) - np.sum(cT)**2) #Slop
intercept = np.average(kT) - slope * np.average(cT) #Intercept of LSRL
print("Regression slope: {}  Regression intercept: {}".format(slope, intercept))

kmax = 1/intercept # From original linearization 1/kmax = intercept
cs = kmax * slope # From original linearization cs/kmax = slope
transformedEq = lambda x: slope * x + intercept
print("kmax = {}  cs = {}".format(kmax, cs))

originalEq = lambda x: (kmax * x**2) / (cs + x**2)
print("Predicted growth rate at c = 2 mg/L = {}".format(originalEq(2.0)))

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize = (10,5))
cVals = np.linspace(c[0],c[-1], 100)
ax1.scatter(c,k)
ax1.plot(cVals, originalEq(cVals))
ax1.set_xlabel("c")
ax1.set_ylabel("k")
ax1.set_title("Orignal data with fitted model")

cTVals = np.linspace(cT[0],cT[-1],100)
ax2.scatter(cT,kT)
ax2.plot(cTVals, transformedEq(cTVals))
ax2.set_xlabel("1/c^2")
ax2.set_ylabel("1/k")
ax2.set_title("Linearized data with linear regression")
```