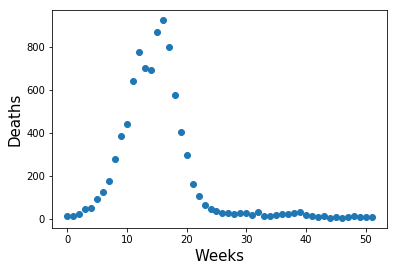
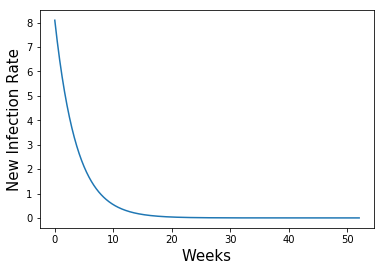
**Problem 3.**

(a)



(b)



(c)

Error1 is 7.30512820513

Error2 is 53.3648980934

**Problem 3 code:**

# -\*- coding: utf-8 -\*-

"""

Created on Sat Feb 17 18:37:29 2018

@author: Jiaqi Li

"""

from numpy import \*

from matplotlib.pyplot import \*

from scipy import array, linspace, integrate

from scipy.optimize import fmin

week = p3datatxt[:,0]

death = p3datatxt[:,1]

figure(1)

scatter(week,death)

xlabel("Weeks", fontsize=15)

ylabel("Deaths", fontsize=15)

#b&c-------------------------------------------------------

def ok(x, t, gama, beta):

I = x[0]

S = x[1]

return array([-beta\*S\*I, beta\*S\*I-gama\*I])

def main():

# set up our initial conditions

S0 = 9000

I0 = 30

x0 = array([I0, S0])

# Parameters

gama = 0.01

beta = 3\*10\*\*(-5)

# choose the time's we'd like to know the approximate solution

t = linspace(0, 52, 500)

# and solve

x = integrate.odeint(ok, x0, t, args=(gama, beta))

I = x[:, 0]

S = x[:, 1]

#Plot I(t) and V (t) on separate subplots using ’semilogy()’

figure(2)

plot(t, beta\*S\*I)

xlabel("Weeks", fontsize=15)

ylabel("New Infection Rate", fontsize=15)

def Error1(data, beta):

n = 0

E = 0

beta = beta

y = data[:,0]

x = data[:,1]

while n < y.shape[0]:

E = E + abs(x[n]/y[n] - beta\*S[n]\*I[n])

n += 1

return E

print('Error1 is',Error1(p3datatxt,beta))

def Error2(data,beta):

n = 0

E = 0

beta = beta

y = data[:,0]

x = data[:,1]

while n < y.shape[0]:

E = E + (x[n]/y[n] - beta\*S[n]\*I[n])\*\*2

n += 1

return E

print('Error2 is',Error2(p3datatxt,beta))

main()

#d----------------------------------------------------------------------------

t = linspace(0, 52, 53)

def ok(x, t, gama, beta):

I = x[0]

S = x[1]

return array([-beta\*S\*I, beta\*S\*I-gama\*I])

def Error2(data):

n = 0

E = 0

I0 = data[0]

S0 = data[1]

x0 = array([I0, S0])

beta = data[2]

gama = data[3]

x = integrate.odeint(ok, x0, t, args=(gama, beta))

I = x[:, 0]

S = x[:, 1]

while n < death.shape[0]:

E = E + (death[n]/week[n] - beta\*S[n]\*I[n])\*\*2

n += 1

return E

Z0 = array([1000,10000,0.001,0.25])

Zbest = fmin(Error2,Z0)

print('fmin finds a minimum at approximately',Zbest)

def main():

# set up our initial conditions

S0 = 8.19569017e+03

I0 = 1.06202593e+03

x0 = array([I0, S0])

# Parameters

gama = 3.99778578e-01

beta = 9.13222275e-08

# choose the time's we'd like to know the approximate solution

t = linspace(0, 52, 53)

# and solve

x = integrate.odeint(ok, x0, t, args=(gama, beta))

I = x[:, 0]

S = x[:, 1]

#Plot I(t) and V (t) on separate subplots using ’semilogy()’

figure(3)

scatter(t, gama\*I)

scatter(week, death)

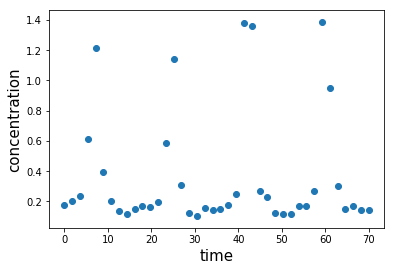
xlabel("Weeks", fontsize=15)

ylabel("New Infection Rate", fontsize=15)

main()

**Problem 4.**

(a)



**Problem 5 code:**

# -\*- coding: utf-8 -\*-

"""

Created on Sat Feb 17 19:35:06 2018

@author: Jiaqi Li

"""

from numpy import \*

from matplotlib.pyplot import \*

from scipy import array, linspace, integrate

time = p4datatxt[:,0]

concentration = p4datatxt[:,1]

figure(1)

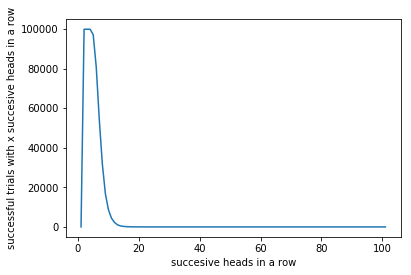
scatter(time,concentration)

xlabel("time", fontsize=15)

ylabel("concentration", fontsize=15)

**Problem 5.**

distribution for the maximum number of sequential heads in 100 flips of a fair coin:



**Problem 5 code:**

# -\*- coding: utf-8 -\*-

"""

Created on Sat Feb 17 17:18:19 2018

@author: Jiaqi Li

"""

import random

from matplotlib.pyplot import \*

def montecarlo(experiment, trials):

positive = 0

for \_ in range(trials):

positive += 1 if experiment() else 0

return positive

def successive\_heads(trials,heads\_in\_a\_row):

count = 0

for \_ in range(trials):

if random.random() < 0.5:

count += 1

if count == heads\_in\_a\_row:

return True

else:

count = 0

return False

def distribution(n):

x = 0

Y = [0]\*101

while x < n+1:

Y[x] = montecarlo(lambda: successive\_heads(100,x), 100000)

x += 1

return Y

X = [0]\*101

for i in range(102):

X[i-1] = i

plot(X,distribution(100))

**Problem 6**

(a)

The Montecarlo simulation achieves at most 3 digits of accuracy and at least 2 digits of accuracy.

(b)

By using the Taylor series of arctan, I need 6 terms to achieve 3 digits of accuracy and 5 terms to achieve 2 digits. Thus, in order to obtain the same accuracy as the montecarlo simulation, I need to use 6 terms.

(c)

For the montecarlo simulation, every time I run the simulation will get a different result than previous. Also, the accuracy of the motecarlo simulation is not stable. Some results are better than others when I use montecarlo simulation for the same question. If I want to be more accurate, I need to spend more time on doing the montecarlo simulation.

For the Taylor series of acrtan, the results are always the same if the model has the same terms, and I can achieve more digits of accuracy by adding more terms to the model, which will also take more time to calculate. This means that Taylor series is more stable than montecarlo simulation.

When I compare the time used for both methods, I find out that to achieve 3 digits of accuracy, it will take my computer about 7 seconds to use montecarlo simulation to complete the task while it will take my computer less than 1 second to use Taylor series to complete the task. This is because in my method, montecarlo simulation runs 10,000,000 trials while Taylor series only need to do the 6 terms calculations. Thus, Taylor series is less time-consuming.

Therefore, Taylor series of arctan is a better way to calculate π in my case.

**Problem 6 code:**

# -\*- coding: utf-8 -\*-

"""

Created on Sat Feb 17 19:45:45 2018

@author: Jiaqi Li

"""

from random import random

from math import pow, sqrt

trial=10000000

points\_the\_circle = 0

throws\_in\_square = 0

for i in range (1, trial):

throws\_in\_square += 1

x = random()

y = random()

distance = sqrt(pow(x, 2) + pow(y, 2))

if distance <= 1.0:

points\_the\_circle = points\_the\_circle + 1.0

# hits / throws = 1/4 Pi

pi = 4 \* (points\_the\_circle / throws\_in\_square)

print("Estimate value for pi by using montecarlo simulation is",pi)

#b-------------------------------------------------------

def function1(x):

y = x - x\*\*3/3 + x\*\*5/5 - x\*\*7/7 + x\*\*9/9 - x\*\*11/11

return y

def function2(x):

y = x - x\*\*3/3 + x\*\*5/5 - x\*\*7/7 + x\*\*9/9

return y

x = sqrt(3)/3

pi1 = 6\*function1(x)

pi2 = 6\*function2(x)

print("The estimated pi (3 digits of accuracy) by using Taylor series of arctan is",pi1)

print("The estimated pi (2 digits of accuracy) by using Taylor series of arctan is",pi2)

def function1(x):

y = x - x\*\*3/3 + x\*\*5/5 - x\*\*7/7 + x\*\*9/9 - x\*\*11/11

return y

x = sqrt(3)/3

print("The estimated pi (3 digits of accuracy) by using Taylor series of arctan is",pi1)