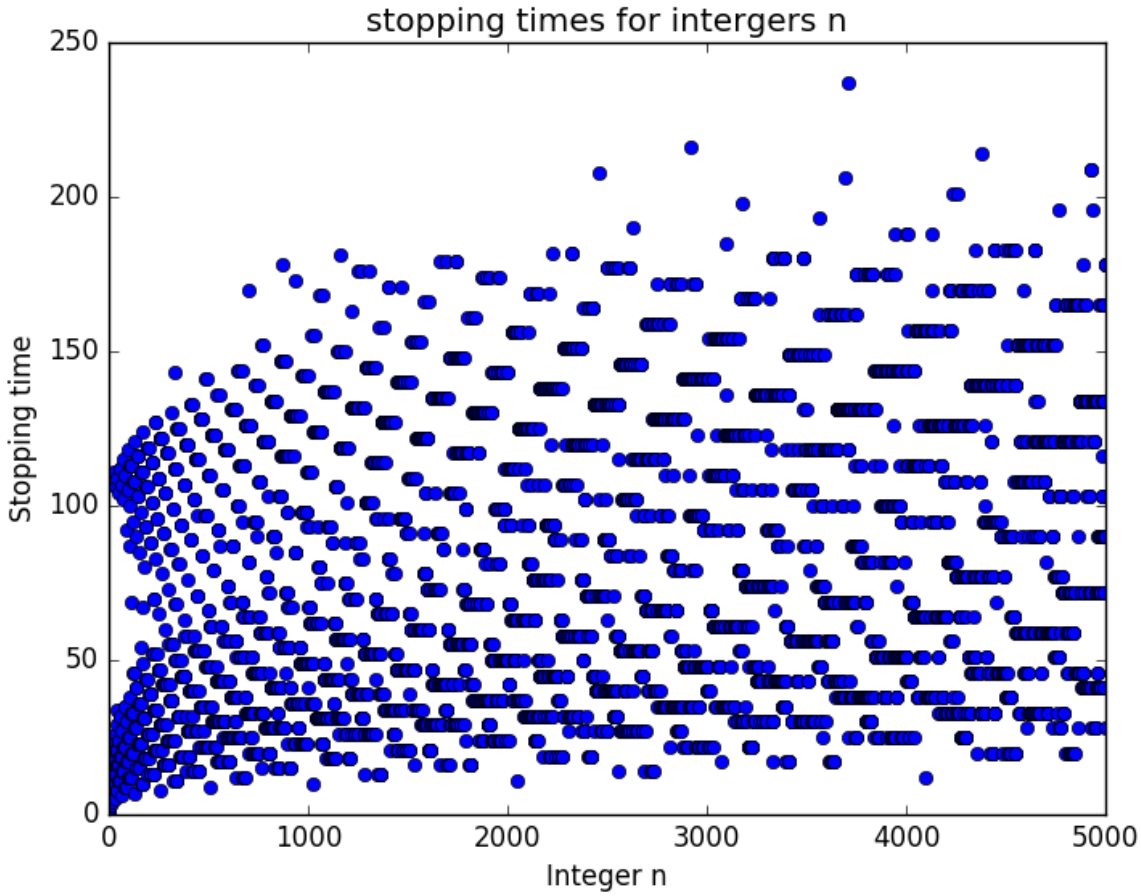
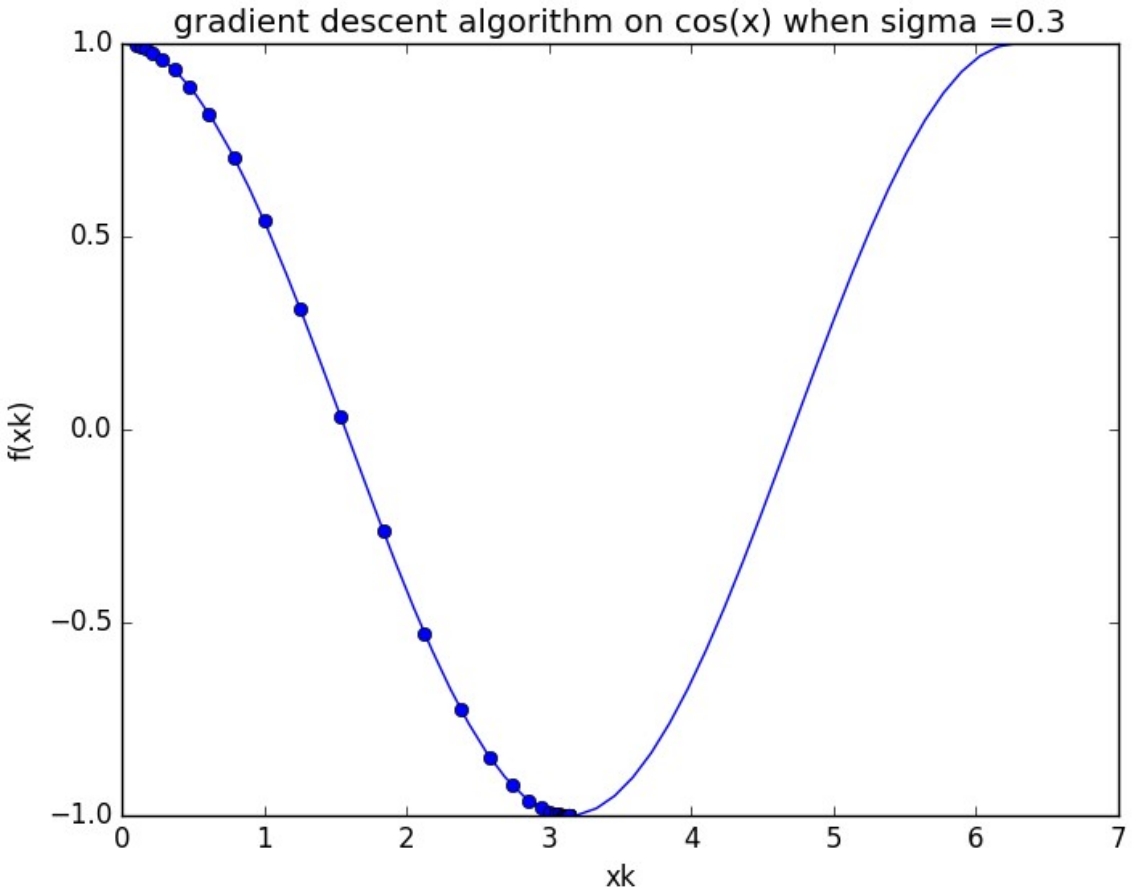
Report HW1

Exercise 1

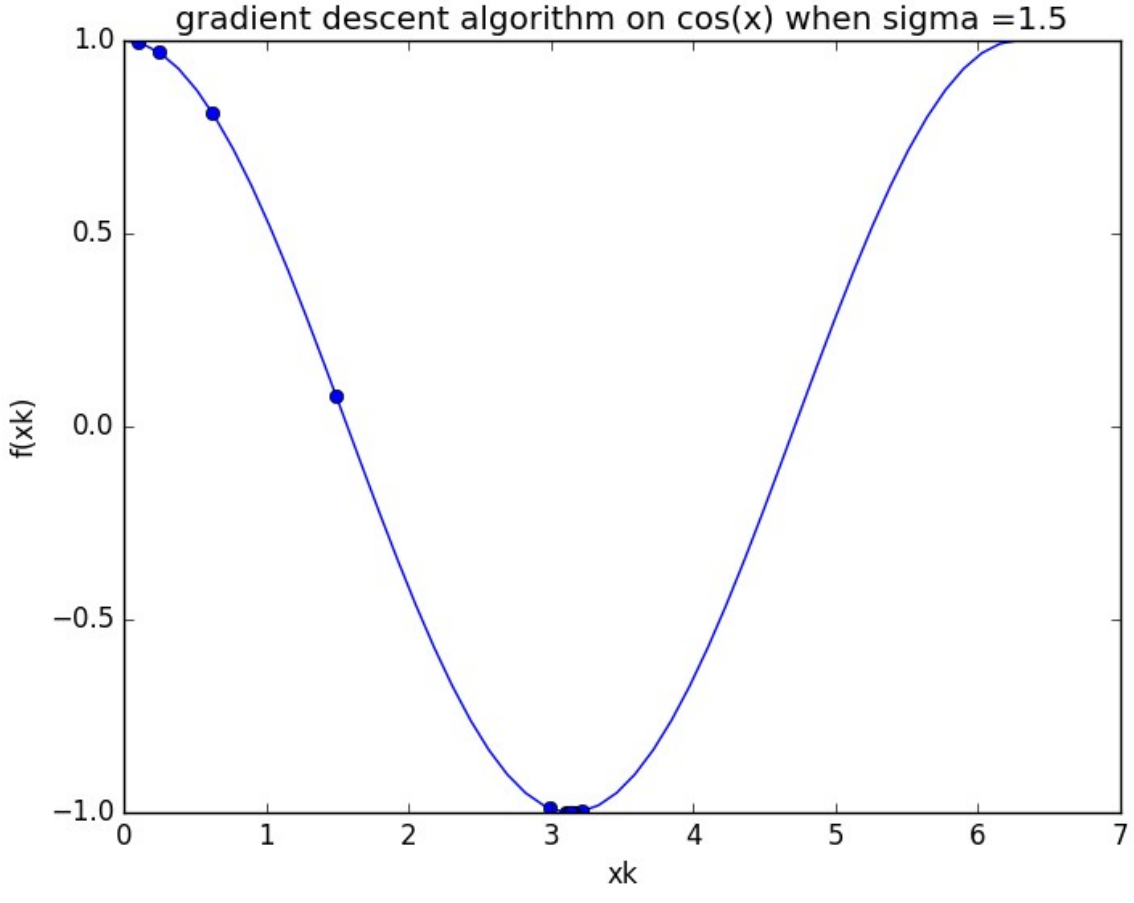


The variable on horizontal axis is input integer n and stopping time on vertical axis. From the definition of collatz, it asserts that every n has a well-defined total stopping time and sequence of n is finite. Otherwise, if such stopping time doesn't exist, let's say that n has infinite total stopping time and the conjecture is false. From the image above, we can say the stopping time is finite so it is true. Based on this, we can say that even n is larger, as long as it has a finite stopping time, it is true. But this picture might be insufficient because we don't know the sequence of n whether end with the number one.

Exercise 2



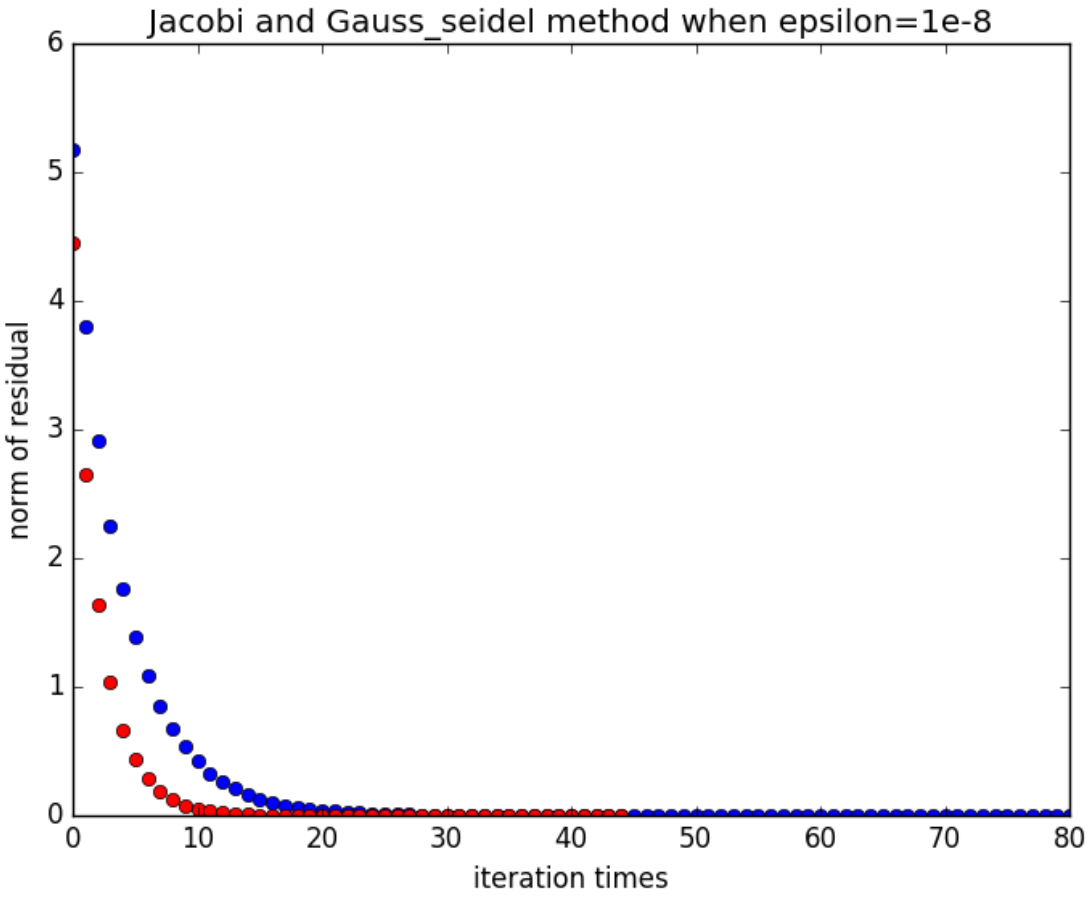
Using f(x) = cos(x) plot the function over the interval (0,2π) with initial guess 0.1. let's first look at the image when sigma is 0.3. Sigma is the factor decides how far you are going down or let's say we are moving sigma amount of steps. So when sigma is 0.3, it takes relatively more steps to find the



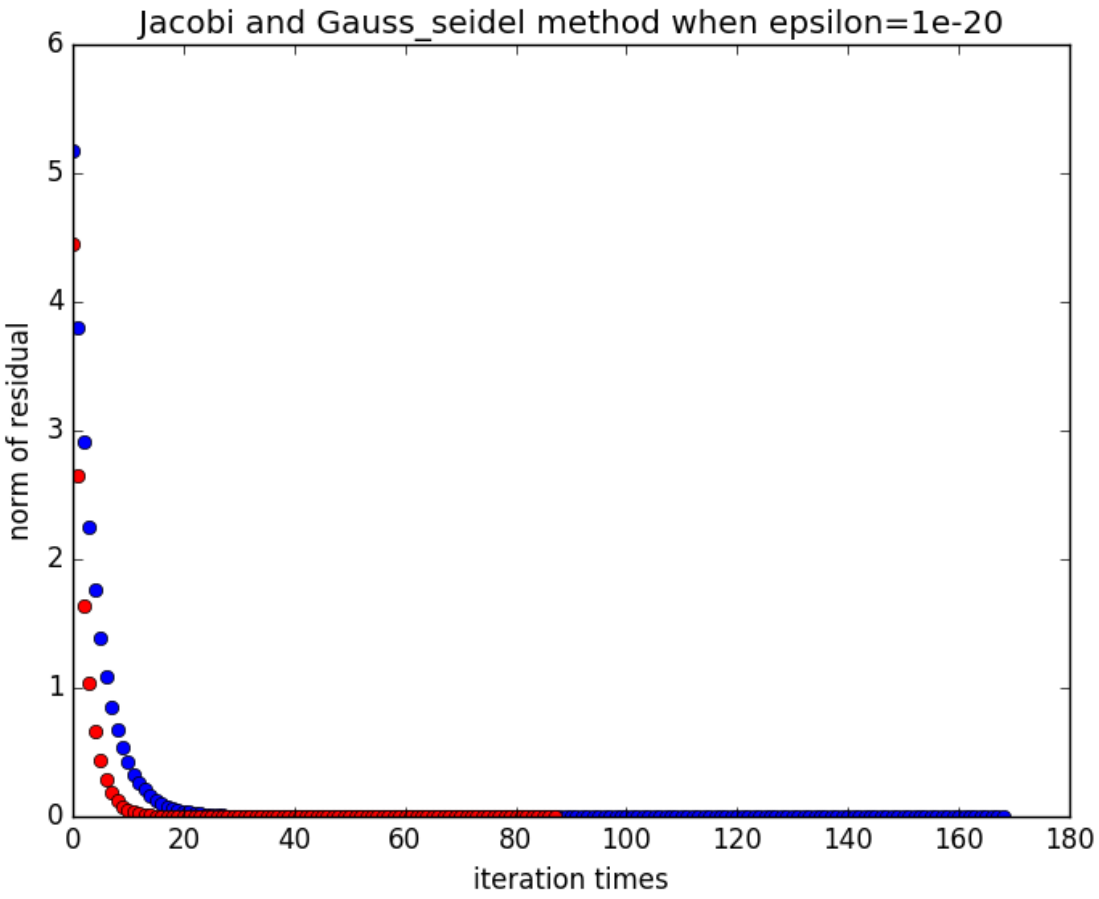
minimal.

Then, lets look at the situation when sigma is 1.5. You can clearly see that it takes less steps than the first situation. However, if sigma is too big, let's in this case sigma = 3, the algorithm is not going to work. As we already know, gradient descent doesn't work when the slope is positive or increasing. The gradient descent might over shoot on the positive slope of a function when learning rate (sigma) is too big. Thus we use smaller learning rate in general.

Exercise 3



The red color represent Gauss seidel iteration and the blue dot represents Jabobi method. We can tell the norm of residual of Gauss-seidel method is smaller than Jacobi method when iteration times get bigger. And finally the norm of residual of Gauss-seidel method converges to zero first.



Let's look at the case when epsilon is 10^(-20), it takes more times for both to converge. But still Gauss method is more effective than Jacobi method.

Code for plotting images:

ex1:

```
n= numpy.linspace(1,10000,10000)
data=[]
for x in n:
    m= collatz(x)
    data= data + [len(m)-1]

plt.title('stopping times for intergers n')
plt.ylabel('Stopping time')
plt.xlabel('Integer n')
plt.plot(n,data,'bo',color='blue')
```

ex2:

```
f= lambda x: cos(x)
x_val = numpy.linspace(0, 2*pi)
plt.plot(x_val, f(x_val))

df= lambda x: -sin(x)
epsilon=1e-8
```

```
x0=0.1
xk1 = x0
xk = x0+1
x_gval = [xk1]
while abs(xk1-xk) > epsilon:
        xk = xk1
        xk1= gradient_step(xk,df,sigma=2)
        x_gval = x_gval + [xk1]

plt.title('gradient descent algorithm on cos(x) when sigma =1.5')
plt.ylabel('f(xk)')
plt.xlabel('xk')
plt.plot(x_gval,f(x_gval), 'bo')
plt.show()


ex3:
a = zeros((32,32))
i,j = numpy.indices(a.shape)
a[i==j] = -5
a[i==j-1] = 1
a[i==j-2] = 1
a[i==j+1] = 1
a[i==j+2] = 1

x0 = ones(32)
b = numpy.sin(10*numpy.linspace(-1,1,32))
A=a
epsilon=1e-8
# jacobi
D,L,U= decompose(A)
xk =x0
xk1 = jacobi_step(D, L, U, b, xk)
rk0 = norm(b- dot(A,xk1))
rk_norm = [rk0]
itr = 0
while norm(xk1 - xk) > epsilon:

    xk = xk1
    xk1 = jacobi_step(D, L, U, b, xk)
    itr = itr +1
    rk =norm(b- dot(A,xk1))
    rk_norm = rk_norm +[rk]

itr_x = range(itr+1)


#guass
xk =x0
xk1 = gauss_seidel_step(D, L, U, b, xk)
rk0 = norm(b- dot(A,xk1))
rk_norm = [rk0]
```

```
itr = 0
while norm(xk1 - xk) > epsilon:

    xk = xk1
    xk1 = gauss_seidel_step(D, L, U, b, xk)
    itr = itr +1
    rk =norm(b- dot(A,xk1))
    rk_norm = rk_norm +[rk]

itr_x = range(itr+1)
plt.plot(itr_x,rk_norm, 'ro')
plt.title('Jacobi and Gauss_seidel method when epsilon=1e-8')
plt.ylabel('norm of residual')
plt.xlabel('iteration times')
plt.plot(itr_x,rk_norm,'bo')
```