Ch. 5. Bracketing Methods (Bungee Jump Company)

Problem of significant vertebrae injury if the free-fall velocity exceeds 36 m/s after 4 s of free fall.

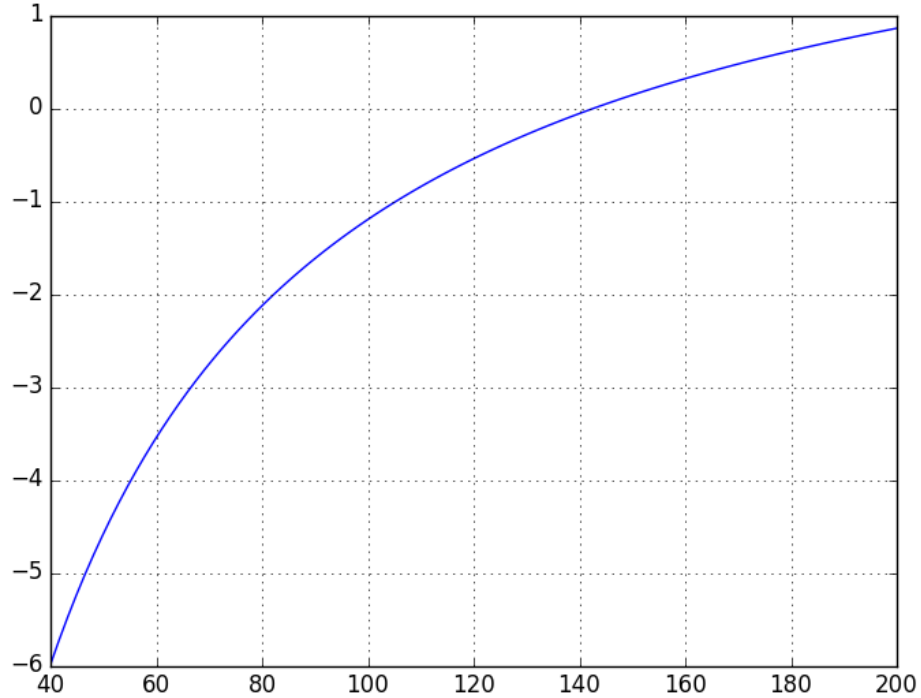
Prof. Sang-Chul Kim

Find out the mass that exceeds the 36m/s after 4sec.

- m=68
- np.sqrt(g*m/cd)*np.tanh(np.sqrt(g*cd/m)*t)
- 33.104494998108365
- m=69
- v=np.sqrt(g*m/cd)*np.tanh(np.sqrt(g*cd/m)*t)
- 33.177015062742854
- m=70
- v=np.sqrt(g*m/cd)*np.tanh(np.sqrt(g*cd/m)*t)
- 33.24783965767994

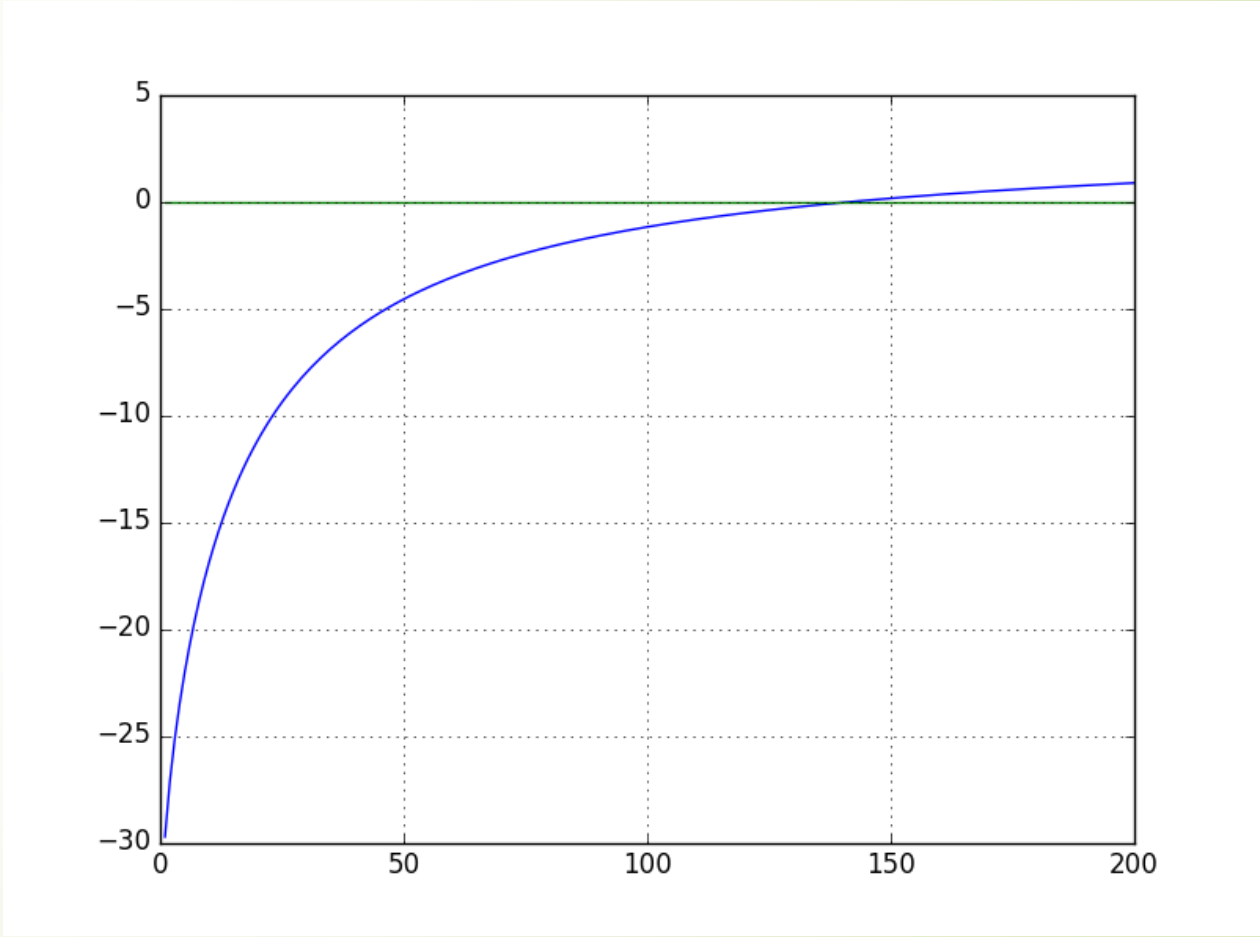
Graphical Method

- g=9.81
- cd=0.25
- t=4
- V=36
- m=np.linspace(40, 200, 100)
- fm=np.sqrt(g*m/cd)*np.tanh(np.sqrt(g*cd/m)*t)-v
- plt.plot(m,fm)



Graphical Method

- k=np.linspace(0,0,200)
- plt.plot(m, k)



Incremental search (증분법)

```
import numpy as np
def incsearch(func, xmin, xmax):
    x=np.arange(xmin, xmax+1)
    #np.linspace(xmin, xmax, ns)
    f=func(x)
    nb=0
    xb=[]

    for k in np.arange(np.size(x)-1):
        if np.sign(f[k]) != np.sign(f[k+1]):
            nb=nb+1
            xb.append(x[k])
            xb.append(x[k+1])

    return nb, xb
```

```
g=9.81; m=68.1; cd=0.25; v=36; t=4;
fp=lambda mp: np.sqrt(g*np.asarray(mp)/cd)*np.tanh(np.sqrt(g*cd/np.asarray(mp))*t)-v
nb, xb=incsearch(fp, 1, 200)
print('number of brackets= ', nb)
print('root interval=', xb)
```

Lambda function (in line function)

- fx=lambda x: 3*x**2
- fx(2)
- 12
- fx=lambda x,y: x**2+y**2
- fx(2,3)
- 13

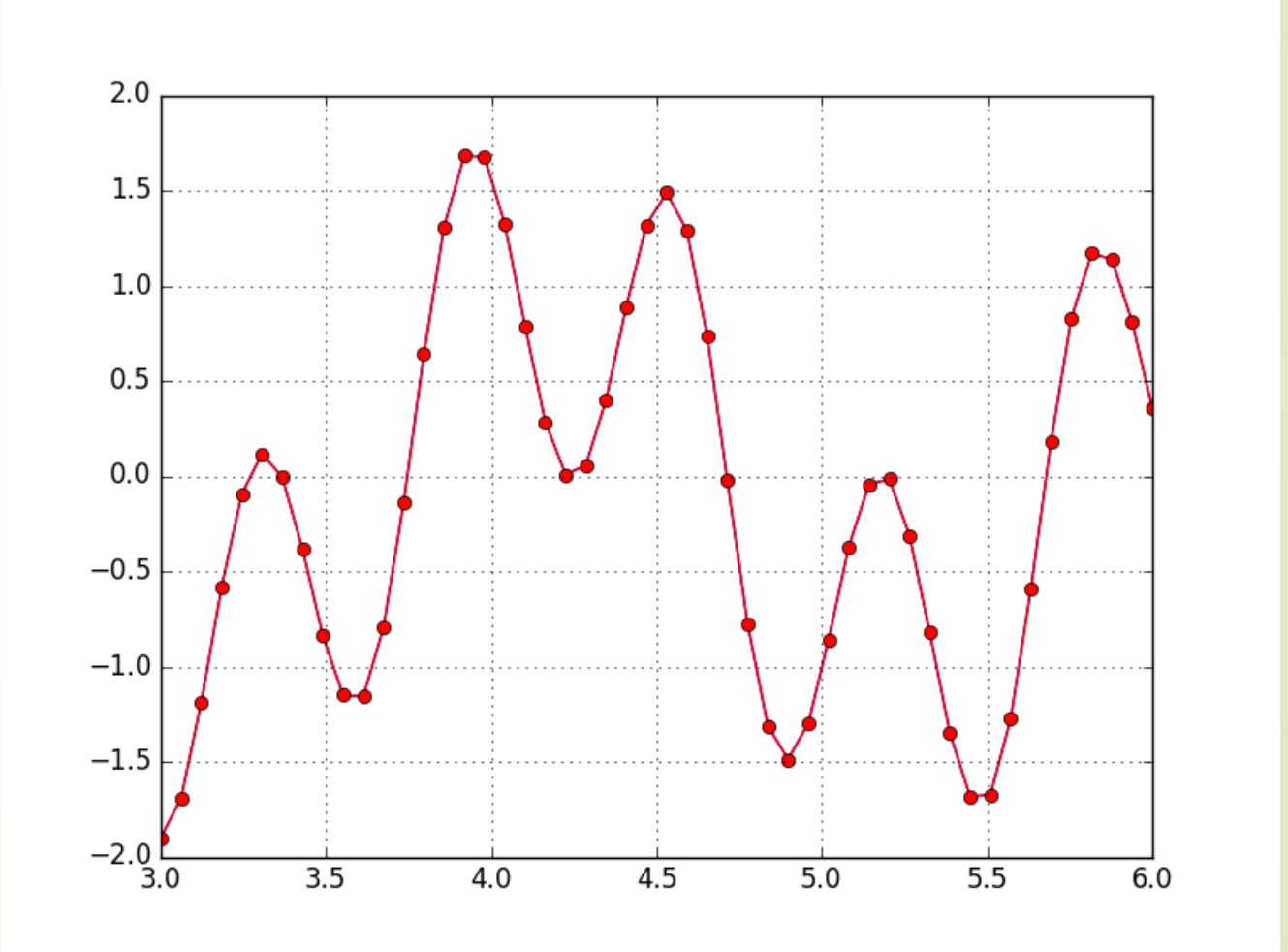
Incremental with ns=50

```
import numpy as np

def incsearch(func, xmin, xmax, ns):

    x=np.linspace(xmin, xmax, ns)
    f=func(x)
    nb=0;     xb=[]

    for k in np.arange(np.size(x)-1):
        if np.sign(f[k]) != np.sign(f[k+1]):
            nb=nb+1
            xb.append(x[k])
            xb.append(x[k+1])

    xbt=np.hstack(xb)
    xb=xbt.reshape(nb, 2)

    return nb, xb

xmin=3; xmax=6

func=lambda x: np.sin(np.dot(10.0, x))+np.cos(np.dot(3.0, x))

nb, xb=incsearch(func, 3, 6, 50)
print('number of brackets= ', nb)
print('root interval=', xb)
```

```
number of brackets=  5
root interval= [[ 3.24489796
3.30612245]
 [ 3.30612245  3.36734694]
 [ 3.73469388  3.79591837]
 [ 4.65306122  4.71428571]
 [ 5.63265306  5.69387755]]
```

Plotting Incremental with ns=50

- x=np.linspace(3, 6, 50)
- func=lambda x: np.sin(np.dot(10.0, x))+np.cos(np.dot(3.0, x))
- f1=func(x)
- plt.figure(1)
- plt.plot(x,f1, 'ro-')
- plt.grid()



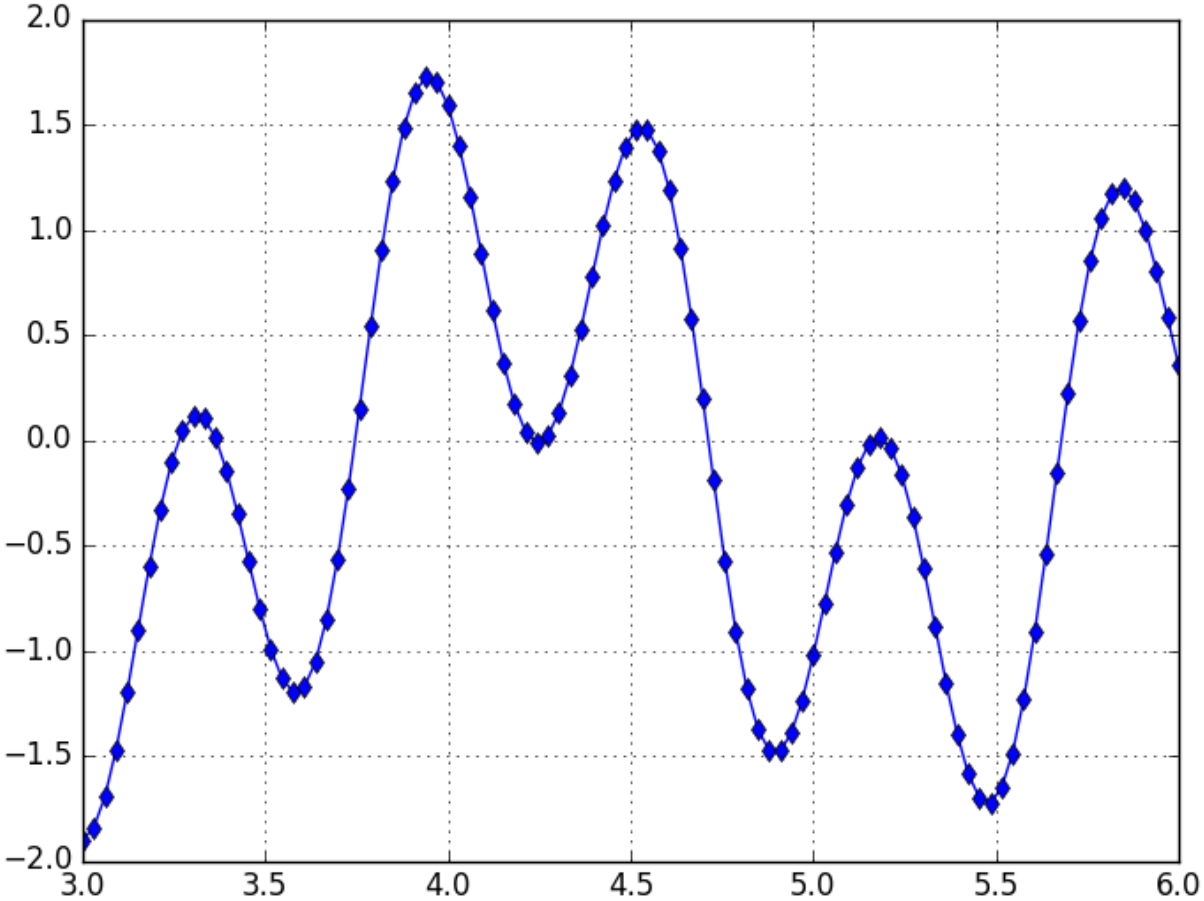
Incremental with ns=100

```
import numpy as np

def incsearch(func, xmin, xmax, ns):

    x=np.linspace(xmin, xmax, ns)
    f=func(x)
    nb=0;     xb=[]

    for k in np.arange(np.size(x)-1):
        if np.sign(f[k]) != np.sign(f[k+1]):
            nb=nb+1
            xb.append(x[k])
            xb.append(x[k+1])

    xbt=np.hstack(xb)
    xb=xbt.reshape(nb, 2)

    return nb, xb

xmin=3; xmax=6

func=lambda x: np.sin(np.dot(10.0, x))+np.cos(np.dot(3.0, x))

nb, xb=incsearch(func, 3, 6, 100)
print('number of brackets= ', nb)
print('root interval=', xb)
```

```
number of brackets=  9
root interval= [[ 3.24242424
3.27272727]
 [ 3.36363636  3.39393939]
 [ 3.72727273  3.75757576]
 [ 4.21212121  4.24242424]
 [ 4.24242424  4.27272727]
 [ 4.6969697   4.72727273]
 [ 5.15151515  5.18181818]
 [ 5.18181818  5.21212121]
 [ 5.66666667  5.6969697 ]]
```

Plotting Incremental with ns=100

- x=np.linspace(3, 6, 100)
- func=lambda x: np.sin(np.dot(10.0, x))+np.cos(np.dot(3.0, x))
- f2=func(x)
- plt.figure(2)
- plt.plot(x,f2, ‘bd-')
- plt.grid()



Bisection (이분법)

```
import numpy as np

def bisect(func, xl, xu):
    maxit=100
    es=1.0e-4

    test=func(xl)*func(xu)


    if test>0:
        print("No Sign Change")
        return [],[],[],[]

    iter=0
    xr=xl

    ea=100
```

```
    while(1):
        xrold=xr
        xr=np.float((xl+xu)/2)
        iter=iter+1

        if xr != 0:
            ea=np.float(np.abs((np.float(xr)-np.float(xrold))/np.float(xr))*100)

        test=func(xl)*func(xr)
```

```
        if test > 0:
            xl=xr
        elif test < 0:
            xu=xr
        else:
            ea=0

        if np.int(ea<es) | np.int(iter >= maxit):
            break

    root=xr
    fx=func(xr)

    return root, fx, ea, iter
```

Bisection (이분법)

```
fm=lambda m: np.sqrt(9.81*m/0.25)*np.tanh(np.sqrt(9.81*0.25/m)*4)-36
root, fx, ea, iter=bisect(fm, 40, 200)

print('root = ', root)
print('f(root) = ', fx, '(must be zero)')
print('estimated error= ', ea, '(must be zero error)')
print('iterated number to find root =', iter)
```

root = 142.73765563964844
f(root) = 4.60891335763e-07 (must be zero)
estimated error= 5.345046825282713 6e-05 (must be zero error)
iterated number to find root = 21

Get the Real Root

```
import numpy as np

import math
from scipy.optimize import fsolve

fm=lambda m: np.sqrt(9.81*m/0.25)*np.tanh(np.sqrt(9.81*0.25/m)*4)-36
m=fsolve(fm, 1)


print("Real Root= ", m)
```

Real Root= [142.73763311]