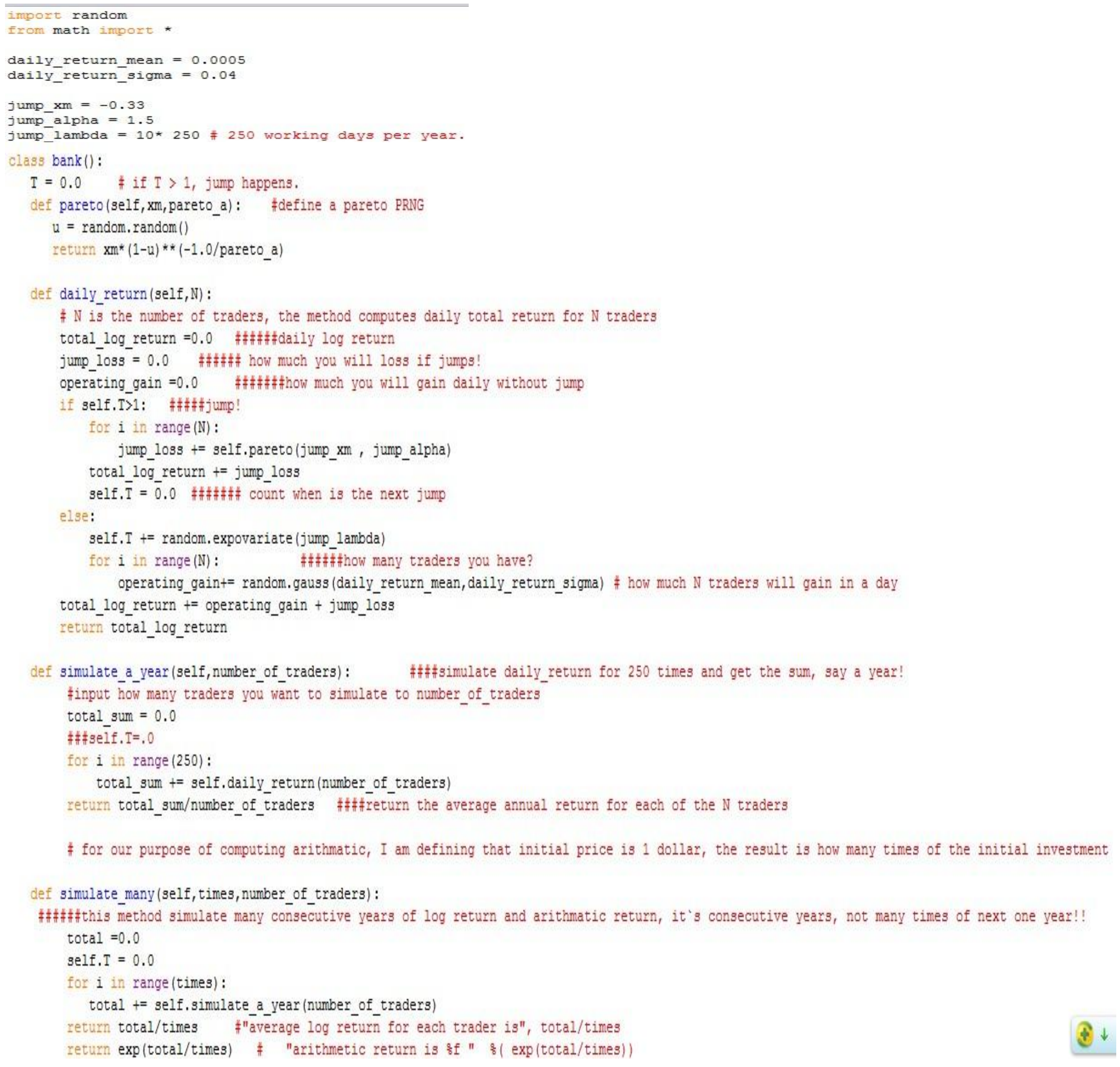
# Assignment 4 by Zhenyang Lu

0. What is the total average log return for each trader? What is the average total arithmetic return for all traders?

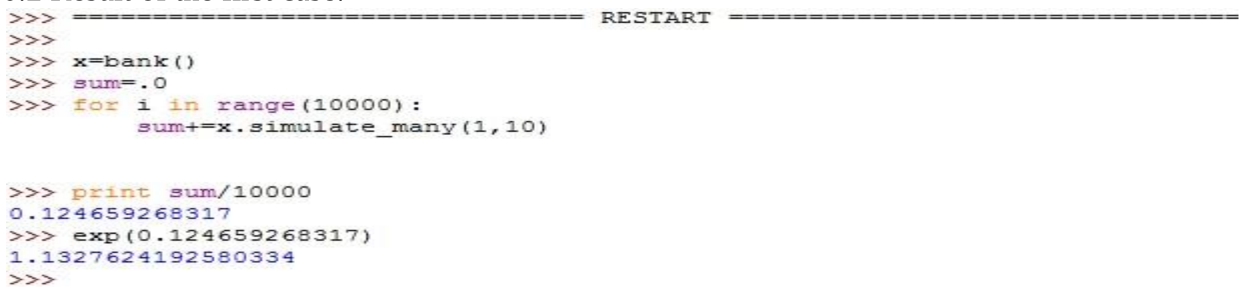
0.0 Analysis
To solve this problem, I created a method called simulate_once( ) to simulate one year total average log return for N traders, Which is a parameter for the method, then convert it into arithmetic return. Then, created simulate_many( ) to simulate M consecutive years` log return. Use this method to compute consecutive years` total return.

0.1 Code



```
import random
from math import *

daily_return_mean = 0.0005
daily_return_sigma = 0.04

jump_xm = -0.33
jump_alpha = 1.5
jump_lambda = 10* 250 # 250 working days per year.
class bank():
   T = 0.0      # if T > 1, jump happens.
   def pareto(self,xm,pareto_a):    #define a pareto PRNG
      u = random.random()
      return xm*(1-u)**(-1.0/pareto_a)

   def daily_return(self,N):
       # N is the number of traders, the method computes daily total return for N traders
       total_log_return =0.0   ######daily log return
       jump_loss = 0.0    ###### how much you will loss if jumps!
       operating_gain =0.0    #######how much you will gain daily without jump
       if self.T>1:   #####jump!
           for i in range(N):
               jump_loss += self.pareto(jump_xm , jump_alpha)
           total_log_return += jump_loss
           self.T = 0.0  ####### count when is the next jump
       else:
           self.T += random.expovariate(jump_lambda)
           for i in range(N):           ######how many traders you have?
               operating_gain+= random.gauss(daily_return_mean,daily_return_sigma) # how much N traders will gain in a day
       total_log_return += operating_gain + jump_loss
       return total_log_return

   def simulate_a_year(self,number_of_traders):       ####simulate daily_return for 250 times and get the sum, say a year!
        #input how many traders you want to simulate to number_of_traders
        total_sum = 0.0
        ###self.T=.0
        for i in range(250):
            total_sum += self.daily_return(number_of_traders)
        return total_sum/number_of_traders   ####return the average annual return for each of the N traders

        # for our purpose of computing arithmatic, I am defining that initial price is 1 dollar, the result is how many times of the initial investment

   def simulate_many(self,times,number_of_traders):
    ######this method simulate many consecutive years of log return and arithmatic return, it`s consecutive years, not many times of next one year!!
        total =0.0
        self.T = 0.0
        for i in range(times):
           total += self.simulate_a_year(number_of_traders)
        return total/times     #"average log return for each trader is", total/times
        return exp(total/times)   #   "arithmetic return is %f "  %( exp(total/times))
```

0.2 Result of the first case:



```
>>> ================================ RESTART ================================
>>>
>>> x=bank()
>>> sum=.0
>>> for i in range(10000):
        sum+=x.simulate_many(1,10)


>>> print sum/10000
0.124659268317
>>> exp(0.124659268317)
1.1327624192580334
>>>
```

The first number, 0.124659268317, is the average yearly log return for a trader, the second figure, 1.1327624192580334, is the arithmetic return. To be convenient, I assume the initial investment at beginning of the first year is $1 dollar. The second number, arithmetic return, means for every dollar you invest, you will get 1.1328 dollar, earning 13.28% profit for each trader. **So 0.1247 is the log average total return for each trader and 13.28% is the arithmetic return.**

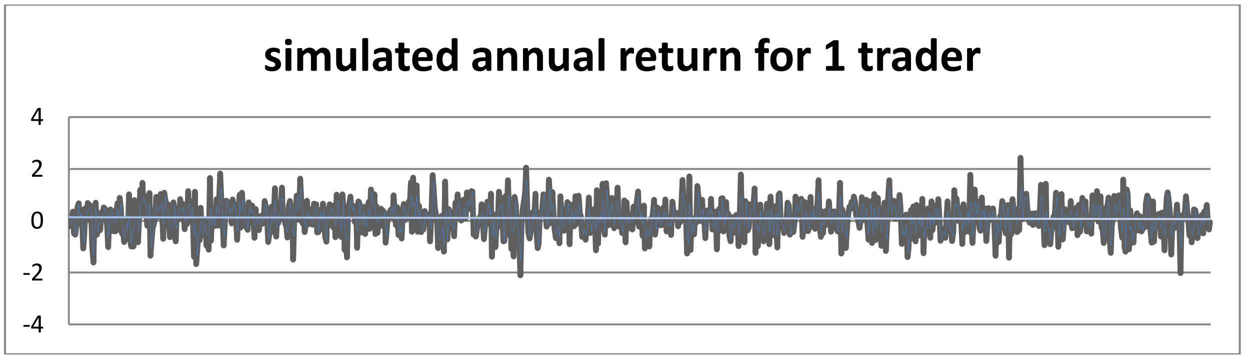
1. Discuss the similarity between this problem and pricing an option using jump diffusion.

1.0 When pricing an call option using jump diffusion, if the price of the underlying assets goes far beyond the strike price, your potential gains or losses (depending on you are long or short) of holding the option will go to unlimited (take it to extreme) and if the price of the underlying assets goes far below the strike price your potential gain or loss will be strike price minus asset price. These two cases are the same as the loss when jump happens in this assignment.

2. Discuss the effects of increasing N, decreasing N, increasing A, decreasing A. In our own self interest, what is the best strategy? In your company interest what is the best strategy?

2.0 Effects of A and N

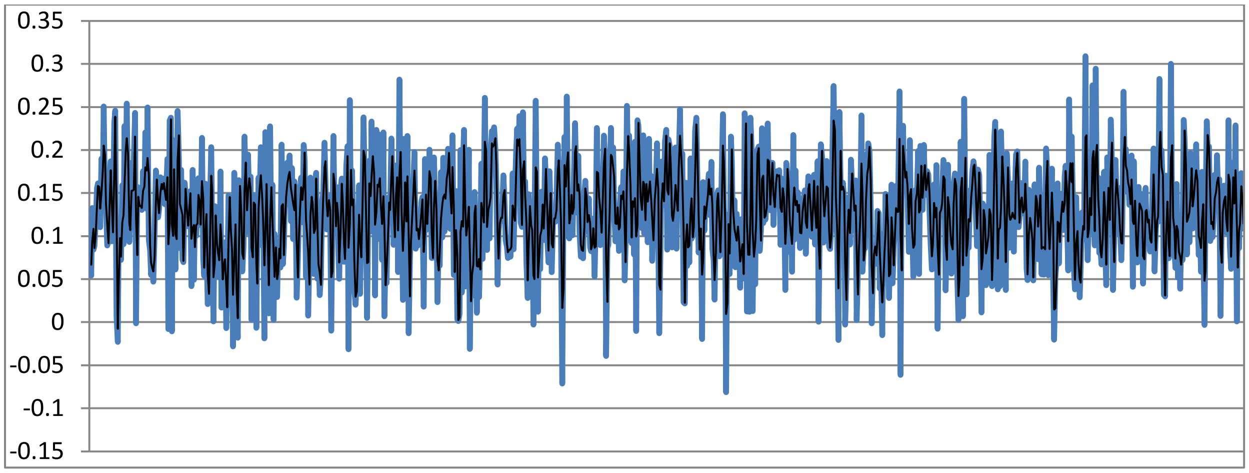
If you have **only** one trader, the expected average return is still the same as the case that have many traders. However, the fluctuation of the return is very fierce. Namely, if manager has only one trader, he/she are likely to loss job for the unstable return of the trader, the plot below shows the average return for one trader simulating 500 times, the mean = 0.103939 and std = 0.623893



From the preceding plot, you could see that the average log return fluctuates around zero.

2.1 Effects of increasing N

To show the effect of increasing N, assume you have 1000 traders, the following plot show the average return for each trader in 1000 traders` case. While the mean is 0.1260 closer to the convergence of 0.13, which is shown in question 0, the standard deviation is only 0.0617, which is significantly lower than the case of only one trader. It`s like running a portfolio, the more trader you get the less volatile your return. However, if you hire more traders, the human resource cost is going to increase. Also, too many traders will cause the company to loss more money once the jump occurs. It`s unlikely to give you too many traders.

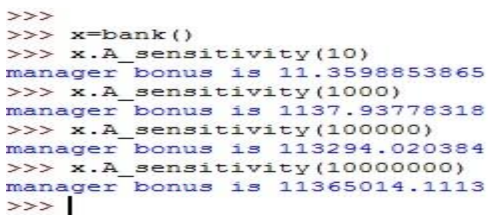


2.2 Effects of increasing and decreasing A

Given the previous analysis, we could see that a higher A results in a higher total bonus for the manager. This can be show by the following code:

```
def simulate_many(self,times,number_of_traders, A):     ##### A is total investment
 ######this method simulate many consecutive years of log return and arithmatic return, it`s consecutive years, not many times of next one year!!
    total =0.0
    for i in range(times):
      total += self.simulate_a_year(number_of_traders)
    #print "average log return for each trader is",total/times     #"average log return for each trader is", total/times
    #print "average arithmatic for each trader return is", exp(total/times)-1   #   "arithmetic return is %f "  %( exp(total/times))
    print "manager bonus is", max((A*exp(total/times)),0)

def A_sensitivity(self,A):
      self.simulate_many(5000, 10, A)
```

, which hold times of simulation and number of traders to see the change of bonus. We get



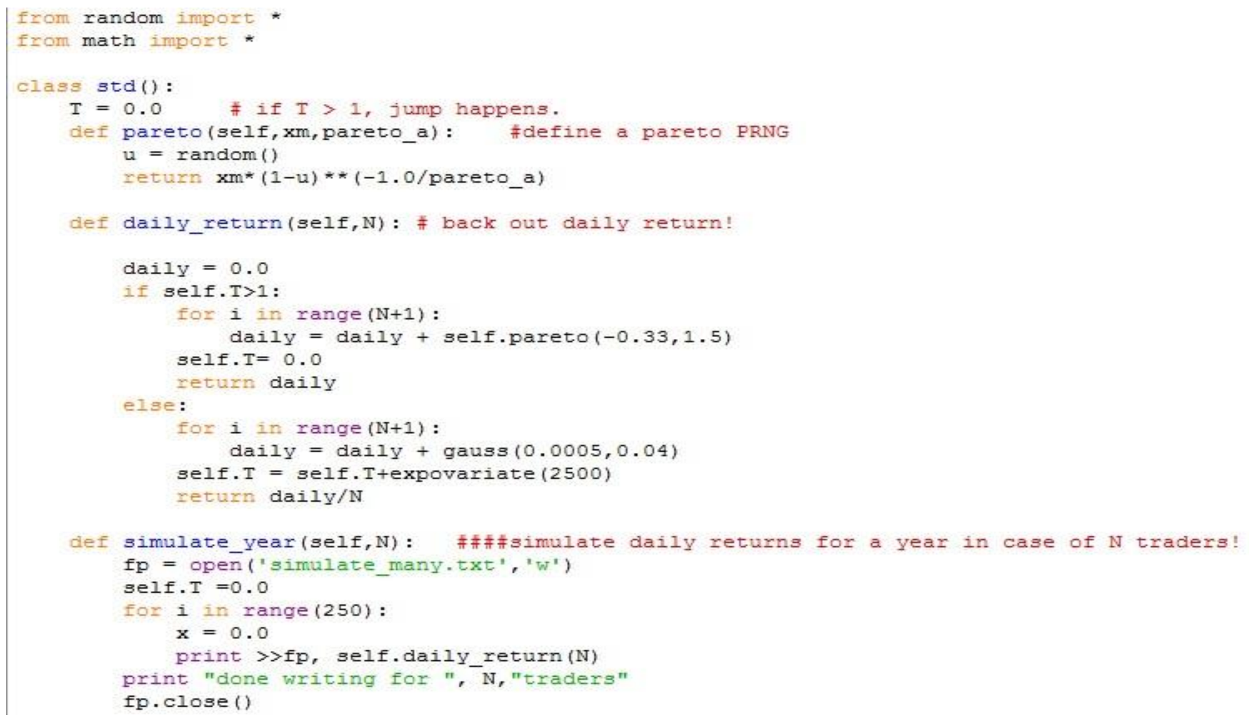
```
>>>
>>> x=bank()
>>> x.A_sensitivity(10)
manager bonus is 11.3598853865
>>> x.A_sensitivity(1000)
manager bonus is 1137.93778318
>>> x.A_sensitivity(100000)
manager bonus is 113294.020384
>>> x.A_sensitivity(10000000)
manager bonus is 11365014.1113
>>> 
```

By running the A_sensitivity () method, whose parameter is 10, 1000, 100000 and 1000000. We could see, the more the parameter, the more the manager bonus.

2.3 Best strategy

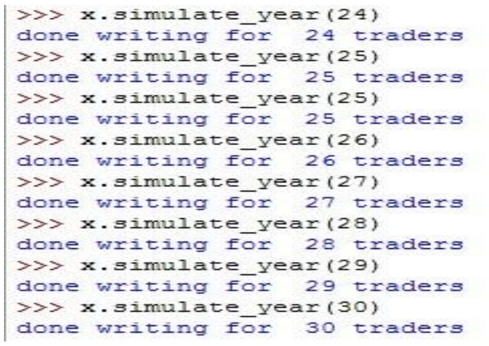
Regarding A, for the manager and the company, the best strategy is to have as much initial A as possible, if we only consider this in mathematical simulation case. Since this will generate the best and most stable bonus for the manager and best absolute profit for the company. However, for the company, the more money you put, the more absolute profit you earn while the more you loss once jump happens! For the manager, how much money is invested does not affect whether he/she will loss job, since this is only affected by the percent loss, not absolute profit!

Regarding N, I applied portfolio management theory which says the unsystematic risk cannot avoid by holding many stocks(traders) in your portfolio and reorganized my code and use it to simulate how much exactly it is for the best number of trader by testing the average standard deviation of the daily log return for each traders, the standard deviation should go down with the increasing number of traders, when the standard deviation goes stable, which means the standard deviation cannot be pull down by more traders, then at this time, the number of traders is the best strategy. My code is like:



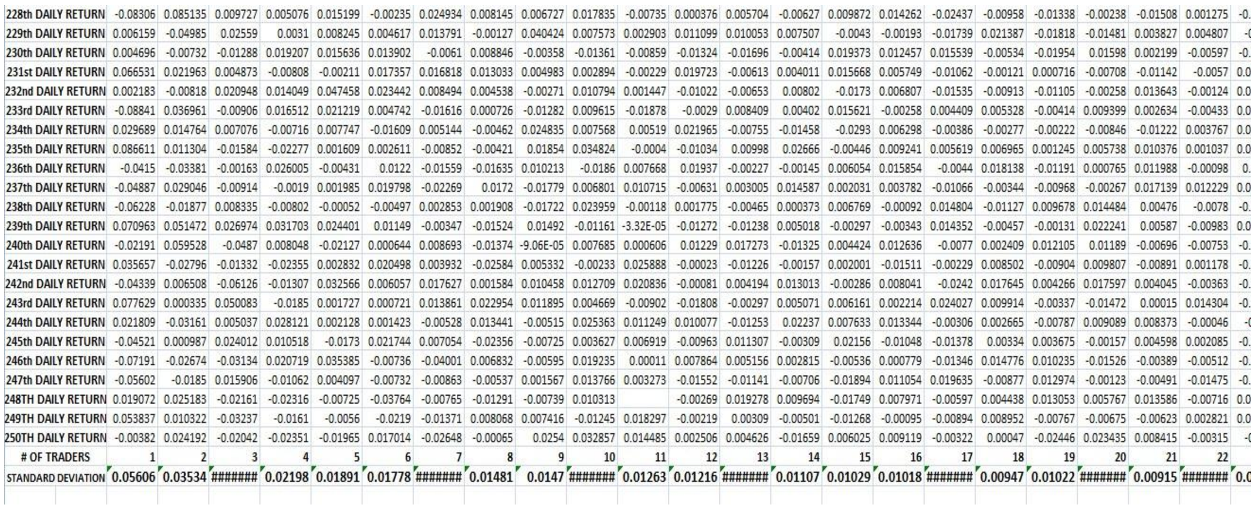
```
from random import *
from math import *

class std():
    T = 0.0     # if T > 1, jump happens.
    def pareto(self,xm,pareto_a):    #define a pareto PRNG
        u = random()
        return xm*(1-u)**(-1.0/pareto_a)

    def daily_return(self,N): # back out daily return!

        daily = 0.0
        if self.T>1:
            for i in range(N+1):
                daily = daily + self.pareto(-0.33,1.5)
            self.T= 0.0
            return daily
        else:
            for i in range(N+1):
                daily = daily + gauss(0.0005,0.04)
            self.T = self.T+expovariate(2500)
            return daily/N

    def simulate_year(self,N):   ####simulate daily returns for a year in case of N traders!
        fp = open('simulate_many.txt','w')
        self.T =0.0
        for i in range(250):
            x = 0.0
            print >>fp, self.daily_return(N)
        print "done writing for ", N,"traders"
        fp.close()
```

The running result is (non-exhausted) like:



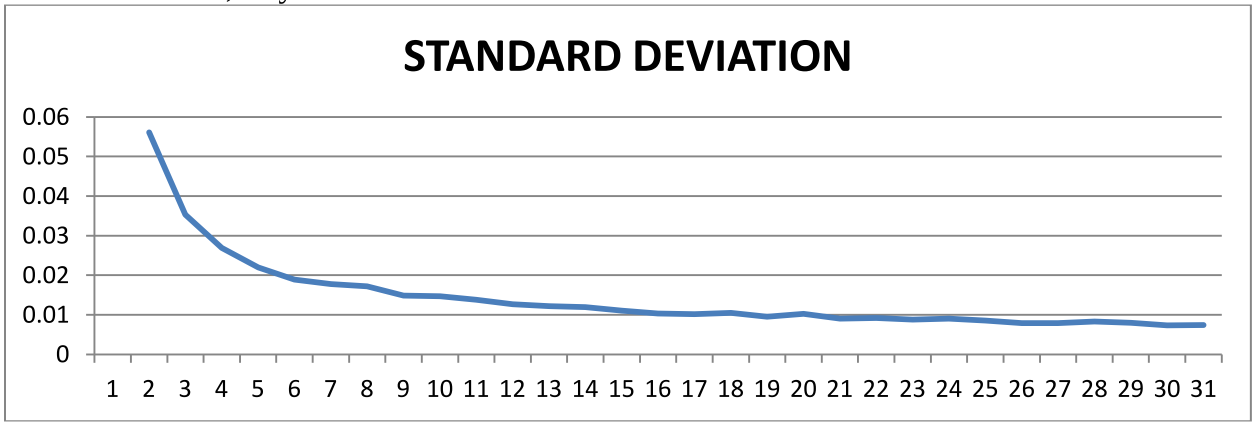
```
>>> x.simulate_year(24)
done writing for  24 traders
>>> x.simulate_year(25)
done writing for  25 traders
>>> x.simulate_year(25)
done writing for  25 traders
>>> x.simulate_year(26)
done writing for  26 traders
>>> x.simulate_year(27)
done writing for  27 traders
>>> x.simulate_year(28)
done writing for  28 traders
>>> x.simulate_year(29)
done writing for  29 traders
>>> x.simulate_year(30)
done writing for  30 traders
```

The simulating data has been put into a local .txt file. I put the data into excel and computed the standard deviation, and the result shows the best number of traders is around 25-30, since at those numbers the standard variance changes not too many and stays stable. The excel data is like (non-exhausted due to large amount of data and there are 30 rows and 252 columns in total):



228th DAILY RETURN	-0.08306	0.085135	0.009727	0.005076	0.015199	-0.00235	0.024934	0.008145	0.006727	0.017835	-0.00735	0.000376	0.005704	-0.00627	0.009872	0.014262	-0.02437	-0.00958	-0.01338	-0.00238	-0.01508	0.001275	-0.
229th DAILY RETURN	0.006159	-0.04985	0.02559	0.0031	0.008245	0.004617	0.013791	-0.00127	0.040424	0.007573	0.002903	0.011099	0.010053	0.007507	-0.0043	-0.00193	-0.01739	0.021387	-0.01818	-0.01481	0.003827	0.004807	-
230th DAILY RETURN	0.004696	-0.00732	-0.01288	0.019207	0.015636	0.013902	-0.0061	0.008846	-0.00358	-0.01361	-0.00859	-0.01324	-0.01696	-0.00414	0.019373	0.012457	0.015539	-0.00534	-0.01954	0.01598	0.002199	-0.00597	-0.
231st DAILY RETURN	0.066531	0.021963	0.004873	-0.00808	-0.00211	0.017357	0.016818	0.013033	0.004983	0.002894	-0.00229	0.019723	-0.00613	0.004011	0.015668	0.005749	-0.01062	-0.00121	0.000716	-0.00708	-0.01142	-0.0057	0.0
232nd DAILY RETURN	0.002183	-0.00818	0.020948	0.014049	0.047458	0.023442	0.008494	0.004538	-0.00271	0.010794	0.001447	-0.01022	-0.00653	0.00802	-0.0173	0.006807	-0.01535	-0.00913	-0.01105	-0.00258	0.013643	-0.00124	0.0
233rd DAILY RETURN	-0.08841	0.036961	-0.00906	0.016512	0.021219	0.004742	-0.01616	0.000726	-0.01282	0.009615	-0.01878	-0.0029	0.008409	0.00402	0.015621	-0.00258	0.004409	0.005328	-0.00414	0.009399	0.002634	-0.00433	0.0
234th DAILY RETURN	0.029689	0.014764	0.007076	-0.00716	0.007747	-0.01609	0.005144	-0.00462	0.024835	0.007568	0.00519	0.021965	-0.00755	-0.01458	-0.0293	0.006298	-0.00386	-0.00277	-0.00222	-0.00846	-0.01222	0.003767	0.0
235th DAILY RETURN	0.086611	0.011304	-0.01584	-0.02277	0.001609	0.002611	-0.00852	-0.00421	0.01854	0.034824	-0.0004	-0.01034	0.00998	0.02666	-0.00446	0.009241	0.005619	0.006965	0.001245	0.005738	0.010376	0.001037	0.0
236th DAILY RETURN	-0.0415	-0.03381	-0.00163	0.026005	-0.00431	0.0122	-0.01559	-0.01635	0.010213	-0.0186	0.007668	0.01937	-0.00227	-0.00145	0.006054	0.015854	-0.0044	0.018138	-0.01191	0.000765	0.011988	-0.00098	0.
237th DAILY RETURN	-0.04887	0.029046	-0.00914	-0.0019	0.001985	0.019798	-0.02269	0.0172	-0.01779	0.006801	0.010715	-0.00631	0.003005	0.014587	0.002031	0.003782	-0.01066	-0.00344	-0.00968	-0.00267	0.017139	0.012229	0.0
238th DAILY RETURN	-0.06228	-0.01877	0.008335	-0.00802	-0.00052	-0.00497	0.002853	0.001908	-0.01722	0.023959	-0.00118	0.001775	-0.00465	0.000373	0.006769	-0.00092	0.014804	-0.01127	0.009678	0.014484	0.00476	-0.0078	-0.
239th DAILY RETURN	0.070963	0.051472	0.026974	0.031703	0.024401	0.01149	-0.00347	-0.01524	0.01492	-0.01161	-3.32E-05	-0.01272	-0.01238	0.005018	-0.00297	-0.00343	0.014352	-0.00457	-0.00131	0.022241	0.00587	-0.00983	0.0
240th DAILY RETURN	-0.02191	0.059528	-0.0487	0.008048	-0.02127	0.000644	0.008693	-0.01374	-9.06E-05	0.007685	0.000606	0.01229	0.017273	-0.01325	0.004424	0.012636	-0.0077	0.002409	0.012105	0.01189	-0.00696	-0.00753	-0.
241st DAILY RETURN	0.035657	-0.02796	-0.01332	-0.02355	0.002832	0.020498	0.003932	-0.02584	0.005332	-0.00233	0.025888	-0.00023	-0.01226	-0.00157	0.002001	-0.01511	-0.00229	0.008502	-0.00904	0.009807	-0.00891	0.001178	-0.
242nd DAILY RETURN	-0.04339	0.006508	-0.06126	-0.01307	0.032566	0.006057	0.017627	0.001584	0.010458	0.012709	0.020836	-0.00081	0.004194	0.013013	-0.00286	0.008041	-0.0242	0.017645	0.004266	0.017597	0.004045	-0.00363	-0.
243rd DAILY RETURN	0.077629	0.000335	0.050083	-0.0185	0.001727	0.000721	0.013861	0.022954	0.011895	0.004669	-0.00902	-0.01808	-0.00297	0.005071	0.006161	0.002214	0.024027	0.009914	-0.00337	-0.01472	0.00015	0.014304	-0.
244th DAILY RETURN	0.021809	-0.03161	0.005037	0.028121	0.002128	0.001423	-0.00528	0.013441	-0.00515	0.025363	0.011249	0.010077	-0.01253	0.02237	0.007633	0.013344	-0.00306	0.002665	-0.00787	0.009089	0.008373	-0.00046	-
245th DAILY RETURN	-0.04521	0.000987	0.024012	0.010518	-0.0173	0.021744	0.007054	-0.02356	-0.00725	0.003627	0.006919	-0.00963	0.011307	-0.00309	0.02156	-0.01048	-0.01378	0.00334	0.003675	-0.00157	0.004598	0.002085	-0.
246th DAILY RETURN	-0.07191	-0.02674	-0.03134	0.020719	0.035385	-0.00736	-0.04001	0.006832	-0.00595	0.019235	0.00011	0.007864	0.005156	0.002815	-0.00536	0.000779	-0.01346	0.014776	0.010235	-0.01526	-0.00389	-0.00512	-0.
247th DAILY RETURN	-0.05602	-0.0185	0.015906	-0.01062	0.004097	-0.00732	-0.00863	-0.00537	0.001567	0.013766	0.003273	-0.01552	-0.01141	-0.00706	-0.01894	0.011054	0.019635	-0.00877	0.012974	-0.00123	-0.00491	-0.01475	-0.
248TH DAILY RETURN	0.019072	0.025183	-0.02161	-0.02316	-0.00725	-0.03764	-0.00765	-0.01291	-0.00739	0.010313		-0.00269	0.019278	0.009694	-0.01749	0.007971	-0.00597	0.004438	0.013053	0.005767	0.013586	-0.00716	0.0
249TH DAILY RETURN	0.053837	0.010322	-0.03237	-0.0161	-0.0056	-0.0219	-0.01371	0.008068	0.007416	-0.01245	0.018297	-0.00219	0.00309	-0.00501	-0.01268	-0.00095	-0.00894	0.008952	-0.00767	-0.00675	-0.00623	0.002821	0.0
250TH DAILY RETURN	-0.00382	0.024192	-0.02042	-0.02351	-0.01965	0.017014	-0.02648	-0.00065	0.0254	0.032857	0.014485	0.002506	0.004626	-0.01659	0.006025	0.009119	-0.00322	0.00047	-0.02446	0.023435	0.008415	-0.00315	-
# OF TRADERS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
STANDARD DEVIATION	0.05606	0.03534	#######	0.02198	0.01891	0.01778	#######	0.01481	0.0147	#######	0.01263	0.01216	#######	0.01107	0.01029	0.01018	#######	0.00947	0.01022	#######	0.00915	#######	0.0

From the following plot, we could see that when the number of traders is about 25-30, the standard variance is stable, which means even if you put more traders, the average risk for each trader is the same, so you do not have to have more traders



This accords to the modern portfolio theory that the more number of stocks (number of traders in this case) in your portfolio, the less portfolio standard deviation is, when it goes within a certain range, the standard deviation is stable, leaving only systematic risk.

So the manager should pick 25-30 traders.