Assignment 2: Conjugate gradients Report

1. Explanation

I've implemented conjugate gradient and steepest descent using golden section from assignment 1 as **line search method**, and a GS helper function to calculate mean optimum value among 20 attempts, I thought that might be helpful, but it remains to be verified.

Initial point is selected by a normal distribution with $\mu = 0, \sigma = 5$ over randomly chosen points: (-1,1) for Rosenbrock and (1,3,3,2,2,1,1,1,2,1) for 10 dimension function.

Stopping criterion is $|F_{new} - F_{old}| \leq F_{tolerance}$ where $F_{tolerance} = 10^{-5}$ by default. For conjugate gradients optimization in Rosenbrock, it easily goes wrong and expand quickly, thus I set an early stopping criterion $F_{new} \geq 5F_{old}$ to constrain the result within a rational range.

2. Evaluation

Two **test functions** are listed as follows:

RosenBrock:

$f_{rb} = (1 - x_0)^2 + 100(x_1 - x_0^2)^2$

f'_{rb}: $[x_0 + 400x_0^3 - 400x_0x_1 - 2,200(x_1 - x_0^2)]$

$x_{opt} = (1, 1), f_{opt} = 0$

10D function:

$f_{10d} = \sum_{i=0}^{9} x_i^2 + x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_8x_9$

$f'_{10d} = [2x_0 + x_1, 2x_1 + x_0, 2x_2 + x_3, 2x_3 + x_2, 2x_4 + x_5, 2x_5 + x_4, 2x_6 + x_7, 2x_7 + x_6, 2x_8 + x_9, 2x_9 + x8]$

$x_{opt} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0), f_{opt} = 0$

Problem 1. conjugate_gradients

Average line search & CG restart iteration

Function	CG_Iter	Avg_LS	X_Err	F_Err	Iter	Time
Rosenbrock	1	0	0.51706479+-1.10829232	0.97501074+-3.94450272	3.28000000+-3.33221750	3.00675392+-2.46785691
	1	1	0.92423445+-0.20308629	92.29594376+-11.48132493	0.00000000+-0.00000000	10.84670544+-0.83343125
	2	0	1.08345956+-0.82845602	15.19100744+-83.96569932	3.18000000+-2.36202957	5.31500340+-2.96408966
	4	0	3.09624032+-4.93645101	5.11256259+-7.79990837	3.40000000+-1.90595201	9.31428432+-3.98360468
	8	0	4.89791346+-9.14767835	385.68351613+-2690.53212679	2.46000000+-2.78633693	15.50422668+-13.01084472
10D function	1	0	0.02900540+-0.05638423	0.00278680+-0.01354818	6.08000000+-2.82727243	6.77239418+-3.05299231
	1	1	0.03256240+-0.01499617	0.00112570+-0.00107806	8.62000000+-5.11057327	205.49767971+-104.87833221
	2	0	0.05965729+-0.14488835	0.02768039+-0.10760186	3.80000000+-1.90595201	8.80976677+-3.54065476
	4	0	0.07617165+-0.24253735	0.07357131+-0.36078158	2.74000000+-1.08439770	16.76701069+-11.03484072
	8	0	0.05739675+-0.19678478	0.04440229+-0.25192309	2.48000000+-1.31304289	36.39761448+-27.58575542

Problem 2. Comparing with steepest descent

	Method	X_Err	F_Err	Iter	Time
Rosenbrock	**CG**	0.86345914+-0.22054580	88.50003083+-13.24840213	0.00000000+-0.00000000	10.64334393+-1.67197003
	SD	1.02541592+-0.09880844	99.12549701+-5.61156770	199.58000000+-187.59923524	2115.54028034+-2081.98762707
10D function	**CG**	0.03179915+-0.02174872	0.00104206+-0.00125786	8.88000000+-5.73083318	199.09227848+-113.02737867
	SD	1.29280715+-0.00099467	0.88815848+-0.00071673	22.02000000+-18.80891018	451.46525383+-358.83582273

As we can see from above, if we evaluate these two optimizer 50 times, the CG performs slightly better at X and F error distances. CG is much more efficient since the it iterates far less than SD, we can also tell by running time(ms). Although SD is less accurate and efficient, but it's more stable than CG as its standard deviation in errors are smaller.

3. Implementation

hw2.ipynb: Detailed implementation and experiments

testfile.py: Required basic functions for testing

conjugate_gradients:

Different format of fprime input: my code take in fprime as a array, storing every partial derivatives for each dimension (fprime = [fp1, fp2, fp3] , it's easier to implement this way by lambda function).

The example function in assignment is hard to optimize with this CG, after adding try-except, there may be one out ten success rate.