INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

OPTIMIZATION METHODS

ASSIGNMENT 2

REPORT

Name: Deepti Rawat

Roll No.[redacted]

Date: 05 April 2024

1. Derive the Jacobians and Hessians for the new functions.
2. Using the Jacobians and Hessians, calculate the minima for the new functions.

Matyas Function

$$f(\bar{x}) = 0.26({x_1}^2 + {x_2}^2) - 0.48x_1 \times x_2$$

<u>Step 1</u>: Computing Jacobian

- $\frac{\partial f(\bar{x})}{\partial x_1} = 0.52x_1 - 0.48x_2$
- $\frac{\partial f(\bar{x})}{\partial x_2} = 0.52x_2 - 0.48x_1$

$\therefore \quad \nabla f(\bar{x}) = \begin{bmatrix} \partial f(\bar{x})/\partial x_1 \\ \partial f(\bar{x})/\partial x_2 \end{bmatrix} = \begin{bmatrix} 0.52x_1 - 0.48x_2 \\ 0.52x_2 - 0.48x_1 \end{bmatrix}$

<u>Step 2</u>: Computing Hessian

$$\therefore \quad H = \nabla^2 f(\bar{x}) = \begin{bmatrix} \frac{\partial^2 f(\bar{x})}{\partial {x_1}^2} & \frac{\partial^2 f(\bar{x})}{\partial x_1 x_2} \\ \frac{\partial^2 f(\bar{x})}{\partial x_2 x_1} & \frac{\partial^2 f(\bar{x})}{\partial {x_2}^2} \end{bmatrix} = \begin{bmatrix} 0.52 & -0.48 \\ -0.48 & 0.52 \end{bmatrix}$$

<u>Step 3</u>: Calculating minima

- Stationary point (equate Jacobian to 0):

$$\nabla f(\bar{x}) = \begin{bmatrix} \partial f(\bar{x})/\partial x_1 \\ \partial f(\bar{x})/\partial x_2 \end{bmatrix} = \begin{bmatrix} 0.52x_1 - 0.48x_2 \\ 0.52x_2 - 0.48x_1 \end{bmatrix} = \mathbf{0}$$

$$\Rightarrow \quad 0.52x_1 - 0.48x_2 = 0$$

$$\Rightarrow \quad 0.52x_2 - 0.48x_1 = 0$$

  Stationary point: $(x_1, x_2) = (0,0)$

- Validating whether stationary point is a minima by checking the positive definiteness of the Hessian:

$$\Rightarrow det(H) > 0 \; (positive\ definite)$$

- Hessian is positive definite. Hence the stationary point (0,0) is a strict local minimum.

Rotated Hyper-Ellipsoid Function

$$f(\bar{x}) = \sum_{i=1}^{d} \sum_{j=1}^{i} {x_j}^2$$

<u>Step 1</u>: Generalizing for 'd'

- For **d = 1**: $f(\bar{x}) = {x_1}^2$
- For **d = 2**: $f(\bar{x}) = \sum_{i=1}^{2} \sum_{j=1}^{i} {x_j}^2 = \sum_{j=1}^{1} {x_j}^2 + \sum_{j=1}^{2} {x_j}^2$

  $\Rightarrow f(\bar{x}) = 2{x_1}^2 + {x_2}^2$

- For **d = 3**: $f(\bar{x}) = \sum_{i=1}^{3} \sum_{j=1}^{i} {x_j}^2 = \sum_{j=1}^{1} {x_j}^2 + \sum_{j=1}^{2} {x_j}^2 + \sum_{j=1}^{3} {x_j}^2$

  $\Rightarrow f(\bar{x}) = 3{x_1}^2 + 2{x_2}^2 + {x_3}^2$

- For **d = 4**: $f(\bar{x}) = \sum_{i=1}^{4} \sum_{j=1}^{i} {x_j}^2 = \sum_{j=1}^{1} {x_j}^2 + \sum_{j=1}^{2} {x_j}^2 + \sum_{j=1}^{3} {x_j}^2 + \sum_{j=1}^{4} {x_j}^2$

  $\Rightarrow f(\bar{x}) = 4{x_1}^2 + 3{x_2}^2 + 3{x_3}^2 + 4{x_4}^2$

- For **d**: $f(\bar{x}) = \underbrace{d{x_1}^2}_{1^{st}} + \underbrace{(d-1){x_2}^2}_{2^{nd}} + (d-2){x_3}^2 + \cdots + 2{x_{(d-1)}}^2 + \underbrace{{x_d}^2}_{d^{th}}$

<u>Step 2</u>: Computing Jacobian

- For **d = 1**: $\nabla f(x_1) = 2x_1$
- For **d = 2**: $\nabla f(x_1, x_2) = \begin{bmatrix} \partial f(\bar{x})/\partial x_1 \\ \partial f(\bar{x})/\partial x_2 \end{bmatrix} = \begin{bmatrix} 4x_1 \\ 2x_2 \end{bmatrix}$
- For **d = 3**: $\nabla f(x_1, x_2, x_3) = \begin{bmatrix} \partial f(\bar{x})/\partial x_1 \\ \partial f(\bar{x})/\partial x_2 \\ \partial f(\bar{x})/\partial x_3 \end{bmatrix} = \begin{bmatrix} 6x_1 \\ 4x_2 \\ 2x_3 \end{bmatrix}$
- For **d**:

$$\therefore \ \nabla f(\bar{x}) = \begin{bmatrix} \partial f(\bar{x})/\partial x_1 \\ \partial f(\bar{x})/\partial x_2 \\ \partial f(\bar{x})/\partial x_3 \\ \vdots \\ \partial f(\bar{x})/\partial x_{d-1} \\ \partial f(\bar{x})/\partial x_d \end{bmatrix} = \begin{bmatrix} (2d)x_1 \\ (2d-2)x_2 \\ (2d-4)x_2 \\ \vdots \\ 4x_{d-1} \\ 2x_d \end{bmatrix}$$

Step 3: Computing Hessian

- For **d = 1**: $\nabla^2 f(x_1) = 2$
- For **d = 2**: $\nabla^2 f(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f(\bar{x})}{\partial {x_1}^2} & \frac{\partial^2 f(\bar{x})}{\partial x_1 x_2} \\ \frac{\partial^2 f(\bar{x})}{\partial x_2 x_1} & \frac{\partial^2 f(\bar{x})}{\partial {x_2}^2} \end{bmatrix} = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix}_{2x2}$
- For **d = 3**: $\nabla f(x_1, x_2, x_3) = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 2 \end{bmatrix}_{3x3}$

- For **d**: matrix size will be $d \times d$

$$\therefore \quad H = \nabla^2 f(\bar{x}) = \begin{bmatrix} \frac{\partial^2 f(\bar{x})}{\partial {x_1}^2} & \frac{\partial^2 f(\bar{x})}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f(\bar{x})}{\partial x_1 x_d} \\ \frac{\partial^2 f(\bar{x})}{\partial x_2 x_1} & \frac{\partial^2 f(\bar{x})}{\partial {x_2}^2} & \cdots & \vdots \\ \vdots & \vdots & & \\ \frac{\partial^2 f(\bar{x})}{\partial x_d x_1} & \frac{\partial^2 f(\bar{x})}{\partial x_d x_2} & \cdots & \frac{\partial^2 f(\bar{x})}{\partial {x_d}^2} \end{bmatrix} = \begin{bmatrix} 2n & 0 & 0 & \cdots & 0 & 0 \\ 0 & 2n-2 & 0 & & : & : \\ 0 & 0 & 2n-4 & & : & \\ 0 & 0 & 0 & & 0 & : \\ 0 & & & & 4 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 2 \end{bmatrix}$$

Step 4: Calculating minima

- Stationary point (equate Jacobian to 0):

$$\nabla f(\bar{x}) = \begin{bmatrix} \partial f(\bar{x})/\partial x_1 \\ \partial f(\bar{x})/\partial x_2 \\ \vdots \\ \partial f(\bar{x})/\partial x_d \end{bmatrix} = \frac{1}{2} \begin{bmatrix} (2d)x_1 \\ (2d-2)x_2 \\ \vdots \\ 2x_d \end{bmatrix} = \mathbf{0}$$

  Stationary point: $(x_1, x_2, \ldots, x_d) = (0,0, \ldots ,0)$

- Validating whether stationary point is a minima by checking the positive definiteness of the Hessian:

$$\Rightarrow det(H) > 0 \; (positive\ definite)$$

- Hessian is positive definite. Hence the stationary point $(0_1, 0_2, \ldots, 0_d)$ is a strict local minimum.

3. State which algorithms failed to converge and under which circumstances.
4. Plot f(x) vs iterations and |f′(x)| vs iterations.
5. Make a contour plot with arrows indicating the direction of updates for all 2-d functions.

Test case	Conjugate: HS	Conjugate: PR	Conjugate: FR	SR1	DFP	BFGS
0	[2. 2.]	[2. 2.]	[2. 2.]	[2. 2.]	[2. 2.]	[2. 2.]
1	[2. 2.]	[2. 2.]	[2. 2.]	[2. 2.]	[2. 2.]	[2. 2.]
2	[-1.748 0.874]	[0. 0.]	[-1.748 0.874]	[-1.748 0.874]	[-1.748 0.874]	[-1.748 0.874]
3	[1.748 -0.874]	[-0. -0.]	[1.748 -0.874]	[1.748 -0.874]	[1.748 -0.874]	[1.748 -0.874]
4	[-1.748 0.874]	[0. 0.]	[-0. -0.]	[0. 0.]	[1.748 -0.874]	[1.748 -0.874]
5	[1.748 -0.874]	[-0. -0.]	[0. 0.]	[-0. -0.]	[-1.748 0.874]	[-1.748 0.874]
6	[1. 1. 1. 1.]	[1. 1. 1. 1.]	[-0.776 0.613 0.382 0.146]	[nan nan nan nan]	[1. 1. 1. 1.]	[1. 1. 1. 1.]
7	[1. 1. 1. 1.]	[1. 1. 1. 1.]	[1. 1. 1. 1.]	[nan nan nan nan]	[1. 1. 1. 1.]	[1. 1. 1. 1.]
8	[1. 1. 1. 1.]	[-0.776 0.613 0.382 0.146]	[1. 1. 1. 1.]	[nan nan nan nan]	[1. 1. 1. 1.]	[1. 1. 1. 1.]
9	[1. 1. 1. 1.]	[1. 1. 1. 1.]	[1. 1. 1. 1.]	[nan nan nan nan]	[1. 1. 1. 1.]	[1. 1. 1. 1.]
10	[nan nan nan nan]	[-2.904 -2.904 -2.904 -2.904]	[-2.904 -2.904 -2.904 -2.904]	[-2.904 -2.904 -2.904 -2.904]	[-2.904 -2.904 -2.904 -2.904]	[-2.904 -2.904 -2.904 -2.904]
11	[nan nan nan nan]	[-2.904 -2.904 -2.904 -2.904]	[-2.904 -2.904 -2.904 -2.904]	[2.747 2.747 2.747 2.747]	[2.747 2.747 2.747 2.747]	[2.747 2.747 2.747 2.747]
12	[nan nan nan nan]	[-2.904 -2.904 -2.904 -2.904]	[-2.904 -2.904 -2.904 -2.904]	[-2.904 -2.904 -2.904 -2.904]	[-2.904 -2.904 -2.904 -2.904]	[-2.904 -2.904 -2.904 -2.904]
13	[-2.904 -2.904 -2.904 -2.904]	[2.747 -2.904 2.747 -2.904]	[2.747 -2.904 2.747 -2.904]	[2.747 -2.904 2.747 -2.904]	[2.747 -2.904 2.747 -2.904]	[2.747 -2.904 2.747 -2.904]
14	[-0. -0.]	[0. 0.]	[-0. -0.]	[0. 0.]	[0. 0.]	[0. 0.]
15	[0. -0.]	[-0. 0.]	[0. -0.]	[-0. 0.]	[-0. 0.]	[-0. 0.]
16	[0. -0.]	[0. 0.]	[-0. 0.]	[0. -0.]	[0. -0.]	[-0. -0.]
17	[0. 0.]	[0. 0.]	[0. 0.]	[0. 0.]	[0. 0.]	[0. 0.]
18	[-0. -0.]	[0. 0.]	[-0. -0.]	[0. 0.]	[0. 0.]	[0. 0.]
19	[-0. 0. 0.]	[0. 0. 0.]	[-0. 0. 0.]	[0. 0. -0.]	[0. -0. -0.]	[0. 0. 0.]
20	[-0. -0. -0. 0. 0. 0. -0.]	[0. 0. -0. 0. 0. 0. -0.]	[0. 0. -0. 0. -0. 0. -0.]	[0. -0. 0. -0. -0. -0. 0.]	[-0. 0. 0. -0. -0. 0. -0.]	[0. 0. 0. -0. 0. 0. -0.]

- The aforementioned snapshot is the terminal output after running all seven functions for all 21 test cases.

- We have computed the minima for different functions manually:
  - Trid Function, for **d** = 2, minima = (2,2)
  - Three Hump Camel function, for **d** = 2, minima = $(0,0),\ \ (-1.7468, 0.8734),\ \ (1.7468, -0.8734)$
  - For Rosenbrock function, for **d**=4, minima = (1, 1, 1, 1)
  - Styblinski-Tang function, for **d** = 4, minima $-2.667 > x_i > 2.667$
  - Root of Square function, for **d** = 2, minima = (0,0)
  - Matyas function, for **d** = 2, minima = (0,0)
  - Rotated Hyper-Ellipsoid function, for **d** = n, minima = $(0_1, 0_2, \dots, 0_n)$

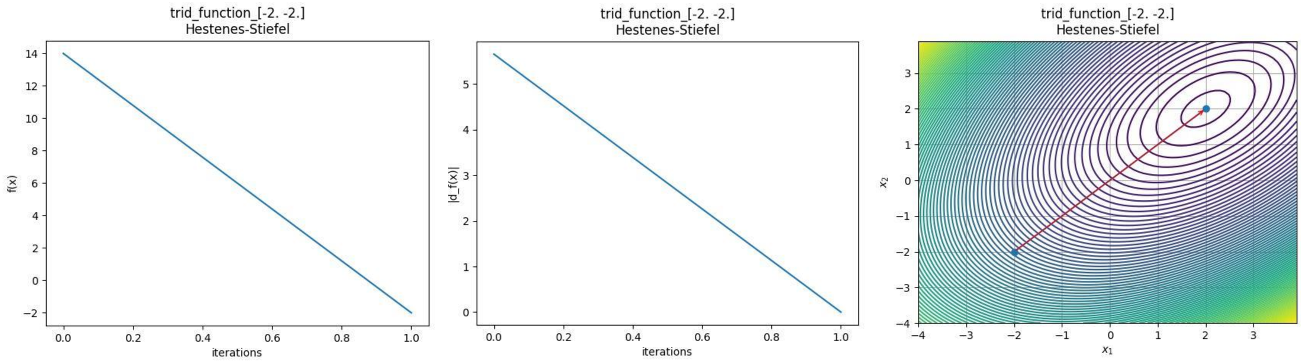
- Comparing the computed minima values with the values in the table, we can conclude that the following functions fail to converge:

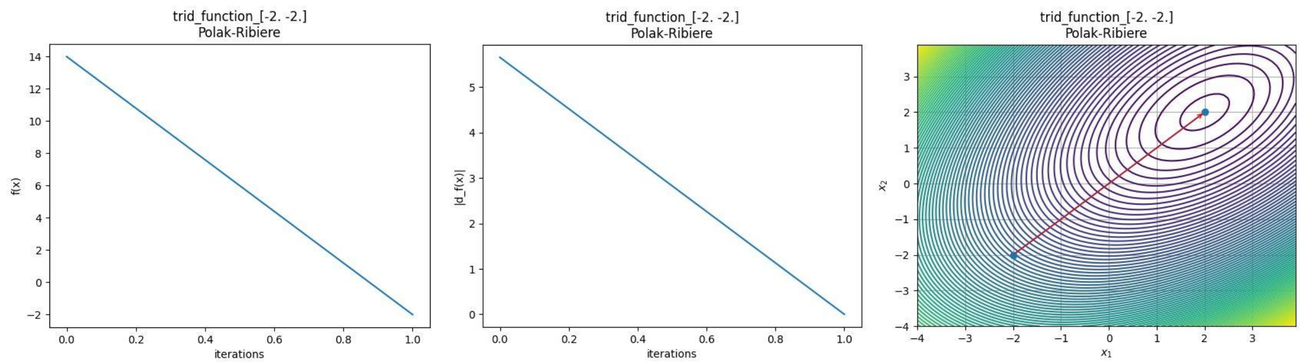
Test Case (0-21)	Function name	Condition	Initial Point	Value
6	Rosenbrock Function	Conjugate: FR	[2.0, 2, 2, -2]	[-0.776, 0.613, 0.382, 0.146]
6	Rosenbrock Function	SR1	[2.0, 2, 2, -2]	[nan, nan, nan, nan]
7	Rosenbrock Function	SR1	[2.0, -2, -2, 2]	[nan, nan, nan, nan]
8	Rosenbrock Function	SR1	[-2.0, 2, 2, 2]	[nan, nan, nan, nan]
8	Rosenbrock Function	Conjugate: PR	[-2.0, 2, 2, 2]	[-0.776, 0.613, 0.382, 0.146]
9	Rosenbrock Function	SR1	[3.0, 3, 3, 3]	[nan, nan, nan, nan]
10	Styblinski-Tang function	Conjugate: HS	[0.0, 0, 0, 0]	[nan, nan, nan, nan]
11	Styblinski-Tang function	Conjugate: HS	[3.0, 3, 3, 3]	[nan, nan, nan, nan]
12	Styblinski-Tang function	Conjugate: HS	[-3.0, -3, -3, -3]	[nan, nan, nan, nan]

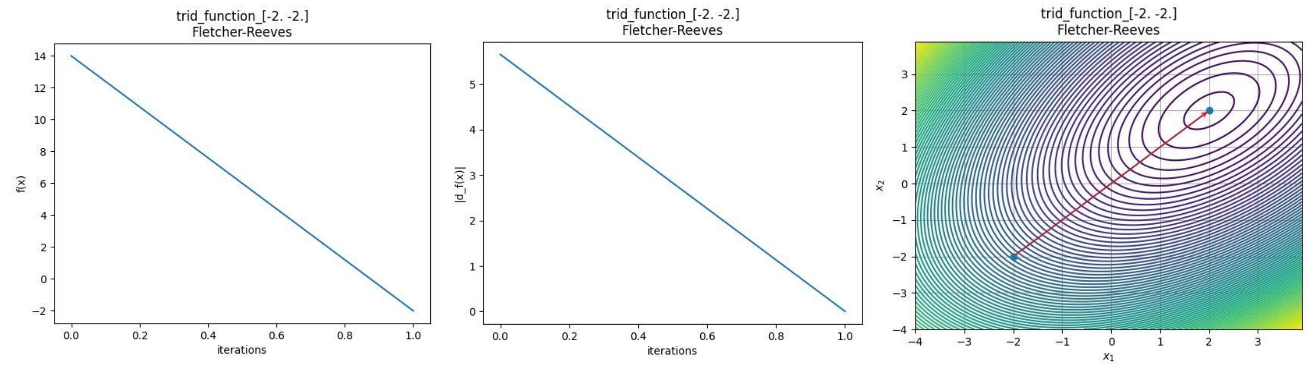
Note: Plots of only some test cases are included in the report. Plots for rest of the test cases are stored in the plots directory.

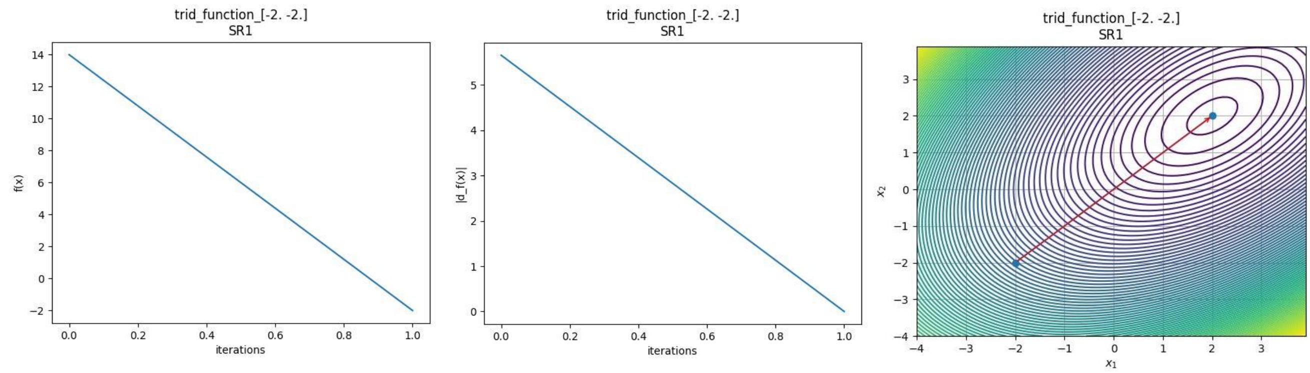
For test case 0 – 1:

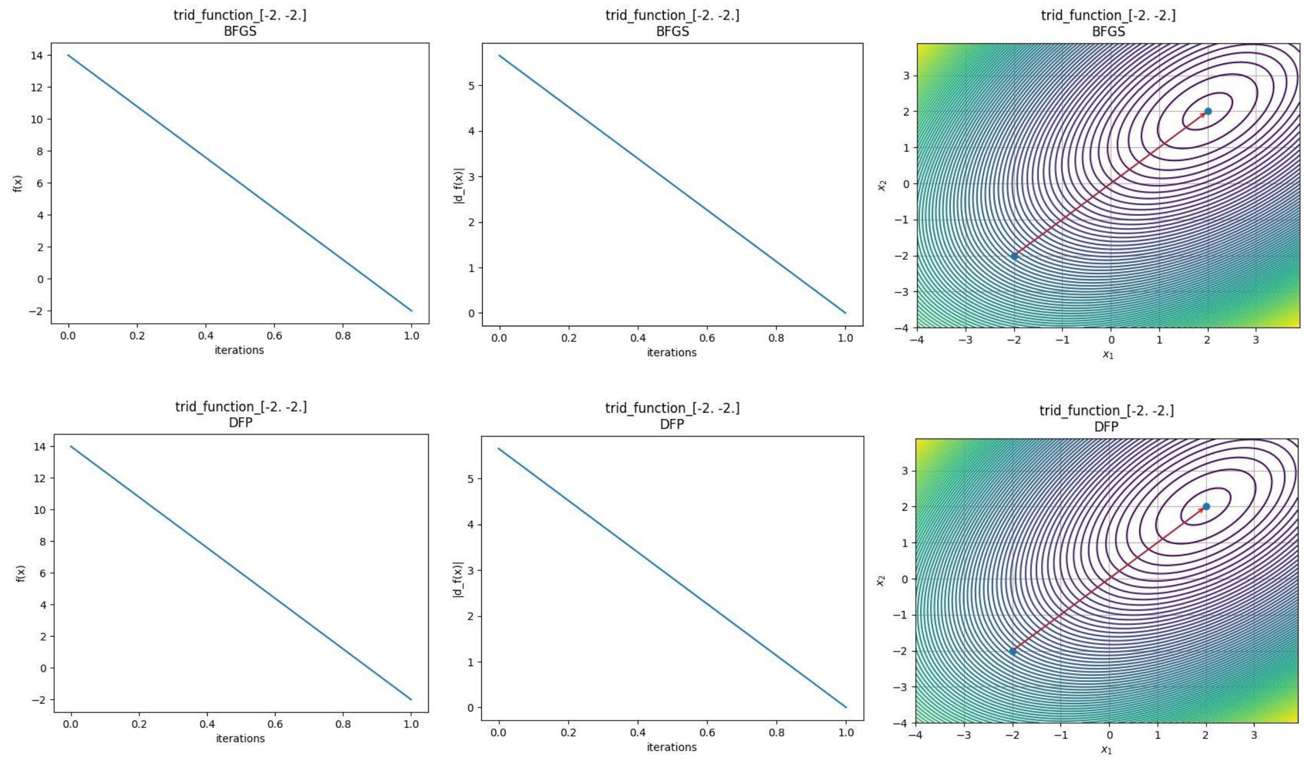
- From the terminal output, we observe that for two different initial points, ***Trid*** function converges to minima for all six algorithms (Conjugate Descent - Hestenes-Stiefel, Conjugate Descent - Polak-Ribiere, Conjugate Descent - Fletcher-Reeves, Rank-One, BFGS and DFP).
- **Note**: Including the graphs of all 6 algorithms for one of the test cases of ***Three hump camel*** function.





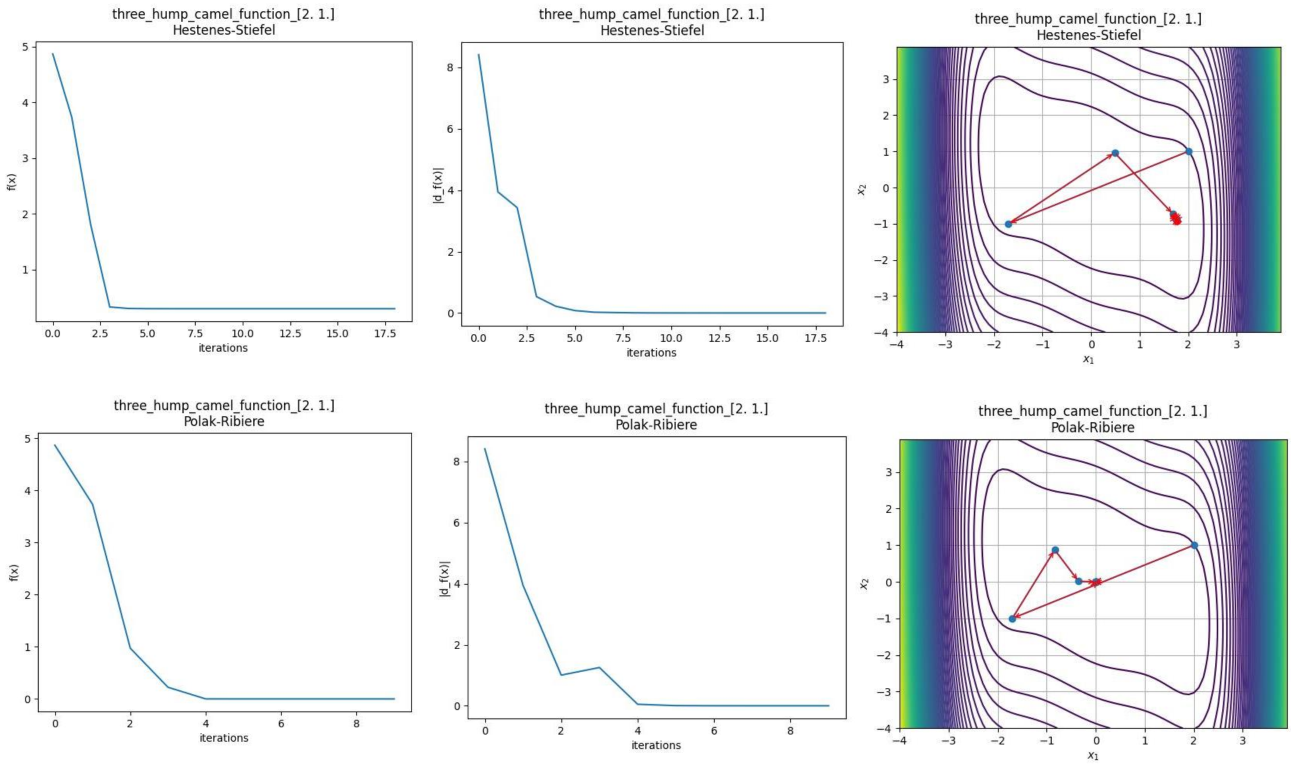


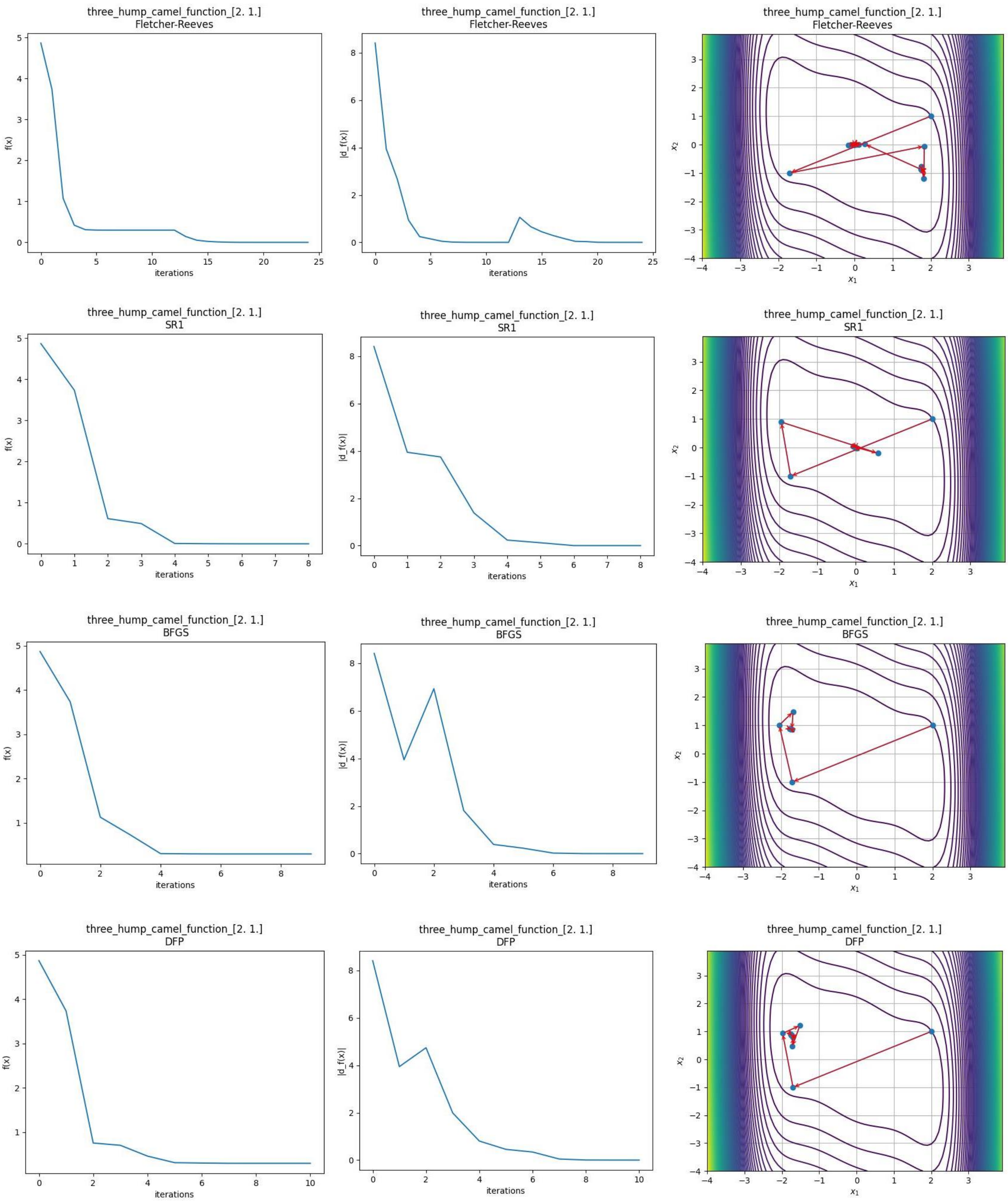




For test case 2 – 5:

- From the terminal output, we observe that for four different initial points, ***Three Hump Camel*** function converges to minima for all six algorithms (Conjugate - HS, Conjugate - PR, Conjugate - FR, SR1, BFGS and DFP).
- **Note**: Including the graphs of all 6 algorithms for one of the test cases of ***Three hump camel*** function.





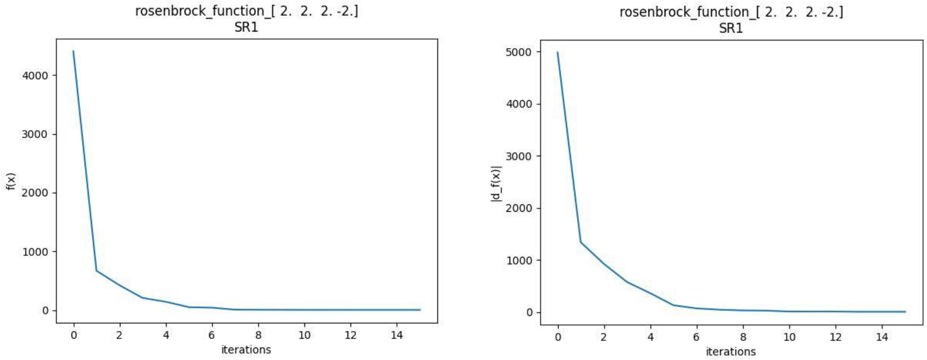
For test case 6 – 9:

- From the terminal output, we can observe that the ***Rosenbrock*** function fails to converge for ***SR1*** algorithm for all 4 test cases with value [nan, nan, nan, nan].
- **Reason**: The update formula of B_k (approximation of Hessian inverse) is given as:

$$B_{k+1} = B_k + \frac{(\boldsymbol{\delta}_k - B_k \boldsymbol{\gamma}_k)(\boldsymbol{\delta}_k - B_k \boldsymbol{\gamma}_k)^T)}{(\boldsymbol{\delta}_k - B_k \boldsymbol{\gamma}_k)^T \boldsymbol{\gamma}_k}$$

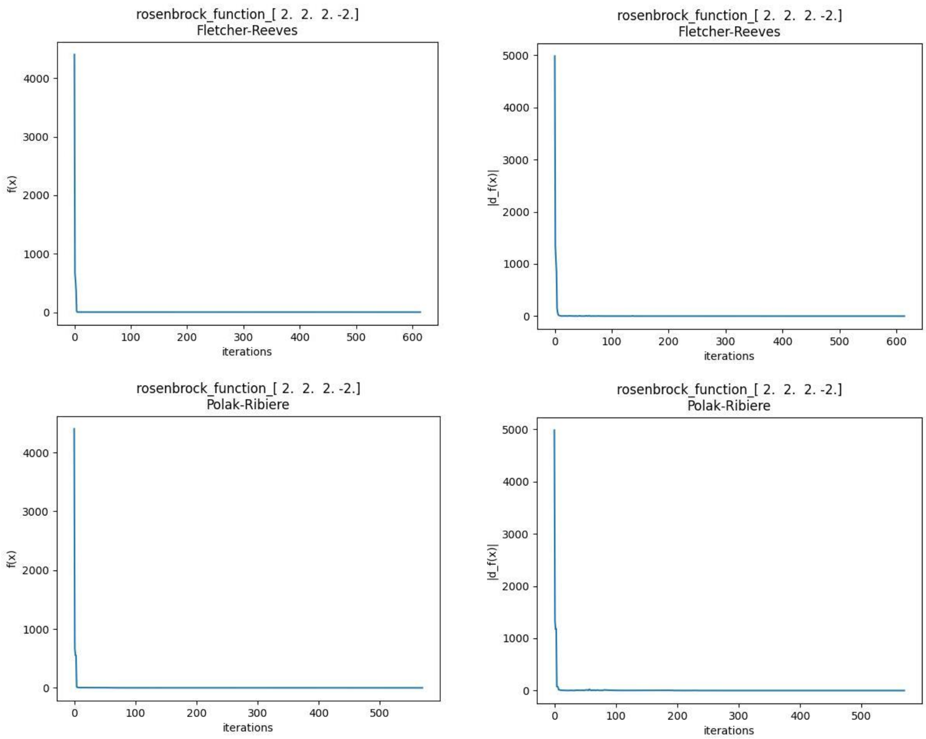
Updated B_{k+1} should be positive definite.

I observe for Test case 6 that initially the update was positive definite and the denominator was large negative number -27478260.341950398 but as the iteration count increased the denominator value approached zero and at nearly 13^{th} iteration it is no more positive definite as the denominator value got very close to zero, and then at 14^{th} iteration the entire update has the value undefined, leading to an undefined solution. Similar pattern is observed for the test case 7,8 and 9.



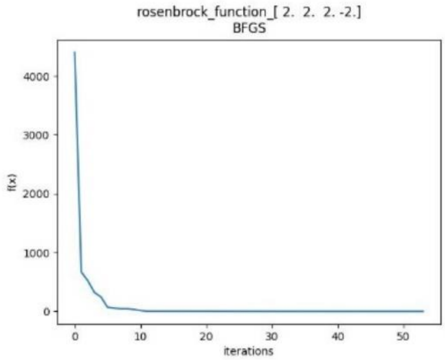
- Also, *Rosenbrock* function fails to converge for *Conjugate - FR* and *Conjugate - PR* algorithm for the following test cases:

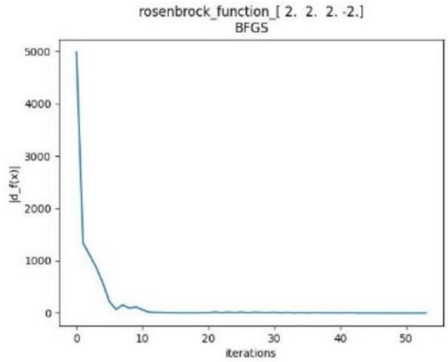
Algorithm	Initial point	Value
Conjugate - FR	[2.0, 2, 2, -2]	[-0.776, 0.613, 0.382, 0.146]
Conjugate - PR	[-2.0, 2, 2, 2]	[-0.776, 0.613, 0.382, 0.146]



- **Reason**: As the value x_k approaches near the minima, the gradient becomes extremely small and eventually dies down to zero before actually reaching the minima. These phenomena can be observed via plots for both Fletcher-Reeves and Polak-Ribiere algorithms.

- However, ***Rosenbrock*** function converges to minima for all 4 test cases, for ***Conjugate-HS***, ***DFP*** and ***BFGS*** algorithms.

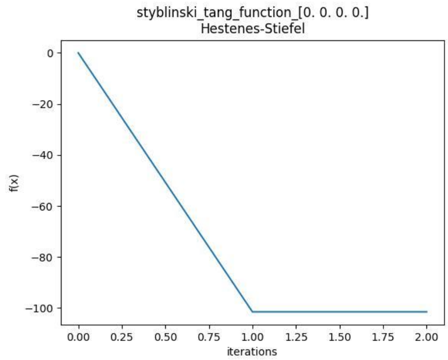


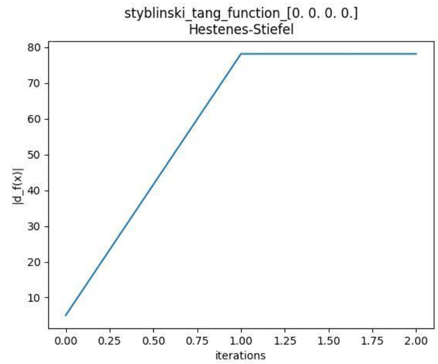


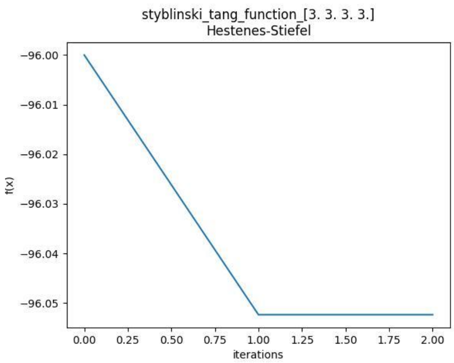
For test case 10 – 13:

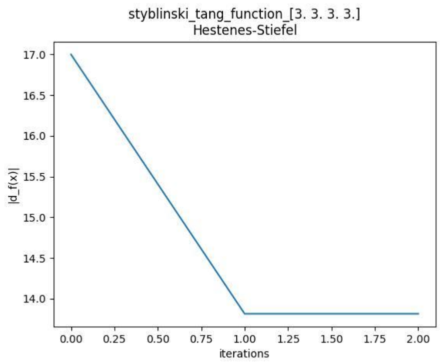
- From the terminal output, we can observe that the ***Styblinski Tang*** function fails to converge for ***Conjugate-HS*** algorithm for 3 test cases with value.

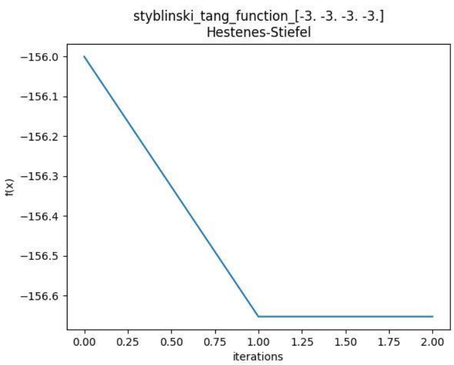
Initial point	Value
[0.0, 0, 0, 0]	[nan, nan, nan, nan]
[3.0, 3, 3, 3]	[nan, nan, nan, nan]
[-3.0, -3, -3, -3]	[nan, nan, nan, nan]

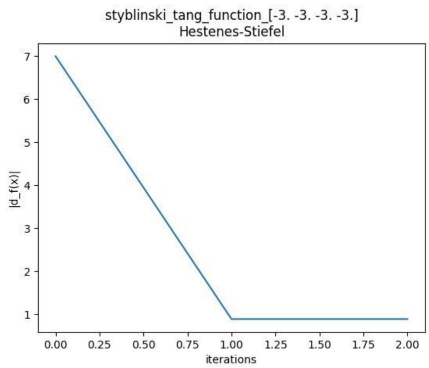




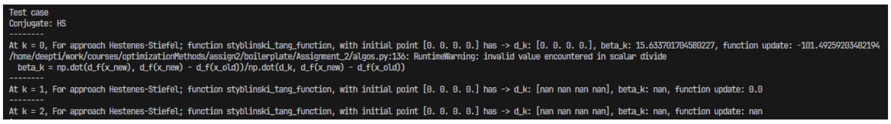








- **Reason**:

  ```
  Test case
  Conjugate: HS
  --------
  At k = 0, For approach Hestenes-Stiefel; function styblinski_tang_function, with initial point [0. 0. 0. 0.] has -> d_k: [0. 0. 0. 0.], beta_k: 15.633701704580227, function update: -101.49259203482194
  /home/deepti/work/courses/optimizationMethods/assign2/boilerplate/Assignment_2/algos.py:136: RuntimeWarning: invalid value encountered in scalar divide
    beta_k = np.dot(d_f(x_new), d_f(x_new) - d_f(x_old))/np.dot(d_k, d_f(x_new) - d_f(x_old))
  --------
  At k = 1, For approach Hestenes-Stiefel; function styblinski_tang_function, with initial point [0. 0. 0. 0.] has -> d_k: [nan nan nan nan], beta_k: nan, function update: 0.0
  --------
  At k = 2, For approach Hestenes-Stiefel; function styblinski_tang_function, with initial point [0. 0. 0. 0.] has -> d_k: [nan nan nan nan], beta_k: nan, function update: nan
  ```

  As we can see from the terminal snapshot above, after first iteration, the descent direction for Styblinski function at point [0,0,0,0] using Hestenes-Stiefel approach becomes zero. This results in difference in gradient at denominator of beta_k formula to become zero, leading to undefined beta_k after k=1. And since beta_k becomes undefined, the descent direction is not updated any further leading to failure in convergence. A similar case has occurred for the rest two test cases. The aforementioned plot also shows that for this case, the gradient at first diverges and then because of no change in descent direction, the gradient also shows no change from the next iteration. A similar pattern can be observed for the plots of other two test cases.
- For rest of the algorithms, *Styblinski Tang* function converges to minima.

For test case 14 – 16:

- From the terminal output, we can observe that the *Root of Square* function converges for all test cases for all the algorithms.

For test case 17 – 18:

- From the terminal output, we can observe that the *Matyas* function converges for *all* algorithms for all test cases.

For test case 19 – 20:

- From the terminal output, we can observe that the *Rotated Hyper-Ellipsoid* function converges for *all* algorithms for given 2 test cases.