The Chinese University of Hong Kong, Shenzhen
SDS · School of Data Science
Andre Milzarek · Zizhuo Wang · Haoxiang Yang · Spring Semester 2022

MAT 3007 – Optimization

Exercise Sheet Nr.: 8

Name: 石雯岚 Shi Wenlan Student ID: 119010265

In the creation of this solution sheet, I worked together with:

Name: _____ Student ID: _____

Name: _____ Student ID: _____

Name: _____ Student ID: _____

For correction:

Exercise								$\sum$
Grading								

The Chinese University of Hong Kong, Shenzhen
SDS · School of Data Science
Andre Milzarek · Zizhuo Wang · Haoxiang Yang · Spring Semester 2022

MAT 3007 – Optimization

Exercise Sheet 8

Problem 1 (A One-Dimensional Problem): (approx. *20* pts)
We consider the optimization problem

$$\min_x \; f(x) := -\frac{1}{(x-1)^2}\left[\log(x) - \frac{2(x-1)}{x+1}\right] \qquad \text{s.t.} \qquad x \in [1.5, 4.5].$$

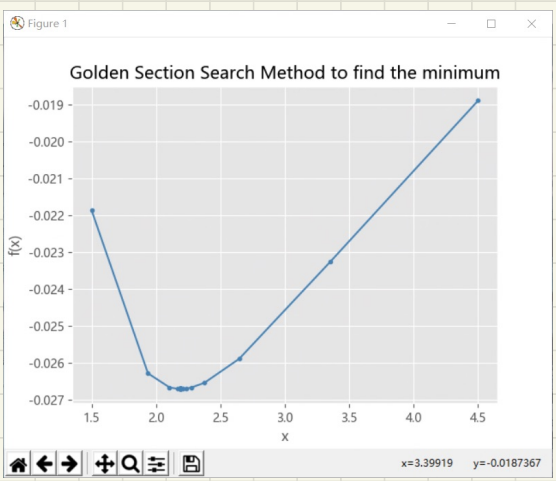
Implement the bisection **or** the golden section method to solve this problem and output a solution with accuracy at least 10^{-5}.

Solution:

I implement the golden section method using python, the code is in the attachment

And the output is as following:

```
Golden Section Method converges in 27 steps, with accuracy 6.820403712470835e-06
final derivative value 2.188706701562353 , final objective function value -0.026707190223050868
```



Problem 2 (The Gradient Method): (approx. *30* pts)

In this exercise, we want to solve the optimization problem

$$\min_{x\in\mathbb{R}^2} f(x) := x_1^4 + \frac{2}{3}x_1^3 + \frac{1}{2}x_1^2 - 2x_1^2x_2 + \frac{4}{3}x_2^2. \tag{1}$$

via the gradient descent method. (This is problem 1 discussed in sheet 6).

Implement the gradient method that was presented in the lecture as a function `gradient_method` in `MATLAB` or `Python`.

The following input functions and parameters should be considered:

- `obj`, `grad` – function handles that calculate and return the objective function $f(x)$ and the gradient $\nabla f(x)$ at an input vector $x \in \mathbb{R}^n$. You can treat these handles as functions or fields of a class or structure `f` or you can use f and ∇f directly in your code. (For example, your function can have the form `gradient_method(obj,grad,...)`).
- x^0 – the initial point.
- `tol` – a tolerance parameter. The method should stop whenever the current iterate x^k satisfies the criterion $\|\nabla f(x^k)\| \leq$ `tol`.

We want to investigate the performance of the gradient method for different step size strategies. In particular, we want to test and compare backtracking and exact line search. The following parameters will be relevant for these strategies:

- $\sigma, \gamma \in (0,1)$ – parameters for backtracking and the Armijo condition. (At iteration k, we choose α_k as the largest element in $\{1, \sigma, \sigma^2, \dots\}$ satisfying the condition $f(x^k - \alpha_k \nabla f(x^k)) - f(x^k) \leq -\gamma\alpha_k \cdot \|\nabla f(x^k)\|^2$).
- You can use the golden section method to determine the exact step size α_k. The parameters for the golden section method are: `maxit` (maximum number of iterations), `tol` (stopping tolerance), $[0, \mathtt{a}]$ (the interval of the step size).

You can organize the latter parameters in an appropriate `options` class or structure. It is also possible to implement separate algorithms for backtracking and exact line search. The method(s) should return the final iterate x^k that satisfies the stopping criterion.

a) Apply the gradient method with backtracking and parameters $(\sigma, \gamma) = (0.5, 0.1)$ and exact line search (`maxit` $= 100$, `tol` $= 10^{-6}$, `a` $= 2$) to solve the problem $\min_x f(x)$.

The algorithms should use the stopping tolerance `tol` $= 10^{-5}$. Test the methods using the initial point $x^0 = (3,3)^\top$ and report the behavior and performance of the methods. In particular, compare the number of iterations and the point to which the different gradient descent methods converged.

b) Let us define the set of initial points

$$\mathcal{X}^0 := \left\{ \begin{pmatrix} -3 \\ -3 \end{pmatrix}, \begin{pmatrix} 3 \\ -3 \end{pmatrix}, \begin{pmatrix} -3 \\ 3 \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \end{pmatrix} \right\}$$

Run the methods:

- Gradient descent method with backtracking and $(\sigma, \gamma) = (0.5, 0.1)$,
- Gradient method with exact line search and `maxit` $= 100$, `tol` $= 10^{-6}$, `a` $= 2$,

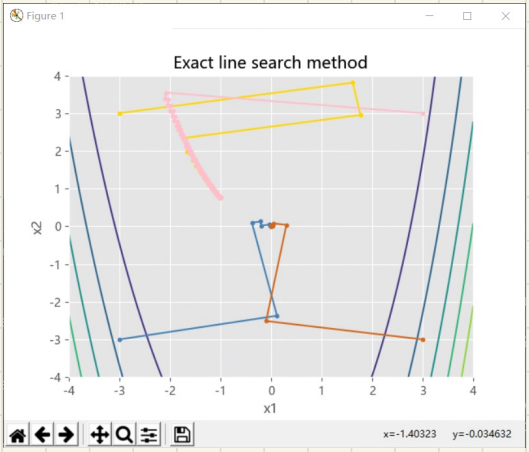
again for every initial point in the set $\mathcal{X}^0$ using the tolerance `tol` $= 10^{-5}$. For each algorithm/step size strategy create a single figure that contains all of the solution paths generated for the different initial points. The initial points and limit points should be clearly visible. Add a contour plot of the function f in the background of each figure.

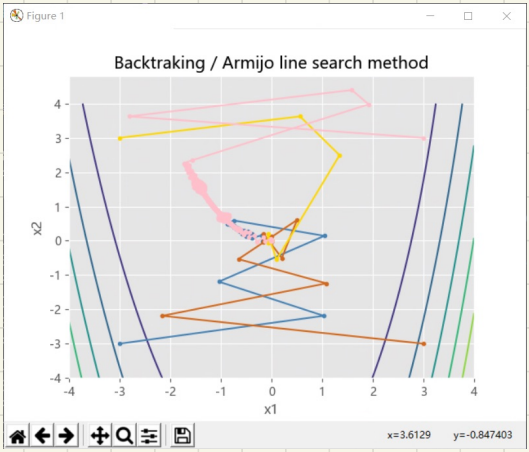
Solution:

(a) I implement both methods using python, the code is in the attachment
And the output is as following:

```
Exact line search method converges in 3971 steps, with accuracy 9.994239291476097e-06
final derivative value [-1.00361286  0.75542991] , final objective function value 0.08333334909610934
Backtraking / Armijo line search method converges in 1962 steps, with accuracy 3.5503692863230763e-06
final derivative value [ 2.52654264e-08 -1.33135477e-06] , final objective function value 2.3636598685
151007e-12
```

(b) I implement both methods using python, the code is in the attachment And the output is as following:






```
Initial point: [-3 -3]
Exact line search method converges in 13 steps, with accuracy 1.5681812037565073e-06
final derivative value [-1.53728086e-06 -1.16170893e-07] , final objective function value 1.1996085825706978e-12
Initial point: [ 3 -3]
Exact line search method converges in 10 steps, with accuracy 5.033331712070309e-06
final derivative value [7.83616330e-07 1.86448526e-06] , final objective function value 4.942098998301707e-12
Initial point: [-3  3]
Exact line search method converges in 2776 steps, with accuracy 9.96583898179408e-06
final derivative value [-1.00364522  0.75548149] , final objective function value 0.08333334954107285
Initial point: [3 3]
Exact line search method converges in 3971 steps, with accuracy 9.994239291476097e-06
final derivative value [-1.00361286  0.75542991] , final objective function value 0.08333334909610934
```

```
Initial point: [-3 -3]
Backtraking / Armijo line search method converges in 50 steps, with accuracy 7.234067469914164e-06
final derivative value [-3.52262216e-07 -2.70955698e-06] , final objective function value 9.85097698811822e-12
Initial point: [ 3 -3]
Backtraking / Armijo line search method converges in 20 steps, with accuracy 9.63535949409848e-06
final derivative value [-1.67474643e-09 -3.61325976e-06] , final objective function value 1.7407529485323415e-11
Initial point: [-3  3]
Backtraking / Armijo line search method converges in 17 steps, with accuracy 5.143827194440494e-06
final derivative value [-3.16710316e-07  1.92527558e-06] , final objective function value 4.99240040107068e-12
Initial point: [3 3]
Backtraking / Armijo line search method converges in 1962 steps, with accuracy 3.5503692863230763e-06
final derivative value [ 2.52654264e-08 -1.33135477e-06] , final objective function value 2.3636598685151007e-12
```

When converge to the same saddle point, exact line search converges faster than backtraking line search method.

Problem 3 (A Limited-Memory Version of Adagrad): (approx. 25 pts)

In this exercise, we investigate a limited-memory version of the gradient descent method with adaptive diagonal scaling. The update of the method is given by

$$x^{k+1} = x^k - \alpha_k D_k^{-1} \nabla f(x^k), \tag{2}$$

where $\alpha_k \geq 0$ is a suitable step size and $D_k \in \mathbb{R}^{n \times n}$ is a diagonal matrix that is chosen as follows: $D_k = \mathrm{diag}(v_1^k, v_2^k, ..., v_n^k)$ and

$$v_i^k = \sqrt{\epsilon + \sum_{j=t_m(k)}^{k} (\nabla f(x^j))_i^2}, \quad t_m(k) = \max\{0, k-m\}, \quad \forall\, i = 1, ..., n.$$

the sum of at most m items

Here, $\epsilon > 0$, the memory constant $m \in \mathbb{N}$, and the initial point $x^0 \in \mathbb{R}^n$ are given parameters.

a) Show that the direction $d^k = -D_k^{-1} \nabla f(x^k)$ is a descent direction for all $k \in \mathbb{N}$ (assuming $\nabla f(x^k) \neq 0$).

b) Write a `MATLAB` or `Python` code and implement the gradient method with adaptive diagonal scaling (Adagrad) and backtracking (i.e., implement the abstract descent method with d^k as descent direction). Run Adagrad on problem (1) introduced in the last exercise and compare its performance with the tested approaches in **Problem 2** using the same initial points and stopping tolerance as in **Problem 2** b). You can use the following parameters:

   - $(\sigma, \gamma) = (0.5, 0.1)$ – parameter for backtracking.
   - $\epsilon = 10^{-6}$ – scaling parameter in the update (2).
   - $m = 25$ – memory parameter.

   Generate a plot of the different solution paths of Adagrad using the different initial points from $\mathcal{X}^0$ and add a contour plot of f in the background.

c) How does Adagrad behave when different memories m are chosen? (Suitable other choices might include $m \in \{5, 10, 15, 25, ..., \texttt{maxit}\}$)? Does the performance or convergence behavior change?

Solution:

(a) $\nabla f(x^k)^T d^k = \nabla f(x^k)^T \cdot (-D_k^{-1} \nabla f(x^k))$

$= \sum_{i=1}^{n} -\underbrace{\frac{1}{v_i^k}}_{>0} \cdot \underbrace{\nabla f(x^k)_i^2}_{>0}$

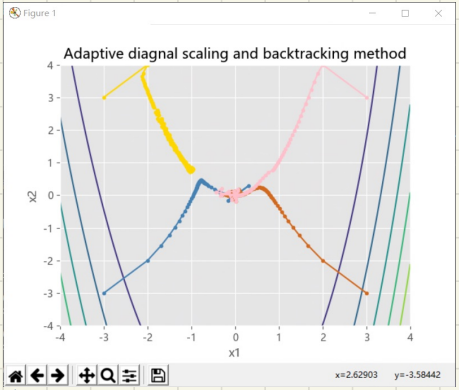
$\therefore \nabla f(x^k)^T d^k < 0$

$\therefore d^k$ is a descent direction.

(b) I implement using python, the code is in the attachment

And the output is as following:

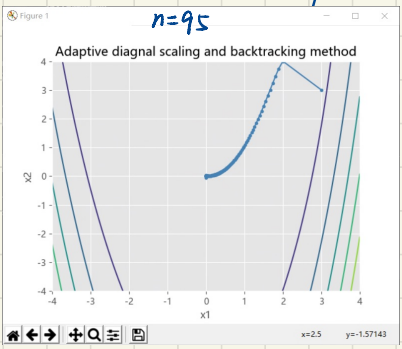
```
Initial point: [-3 -3]
Adaptive diagnal scaling and backtracking method converges in 112 steps, with accuracy 8.999614079538812e-06
final derivative value [-1.48363165e-10 -2.82031441e-06] , final objective function value 1.0605564520739175e-11
Initial point: [ 3 -3]
Adaptive diagnal scaling and backtracking method converges in 179 steps, with accuracy 7.283633471615621e-06
final derivative value [1.40164613e-11 1.26887941e-06] , final objective function value 2.146739946562035e-12
Initial point: [-3  3]
Adaptive diagnal scaling and backtracking method converges in 1052 steps, with accuracy 8.030546066792864e-06
final derivative value [-1.00140742  0.75211344] , final objective function value 0.08333333426450795
Initial point: [3 3]
Adaptive diagnal scaling and backtracking method converges in 140 steps, with accuracy 9.06414610132815e-06
final derivative value [-8.27705149e-15 -2.81275826e-06] , final objective function value 1.0548812073031753e-11
```

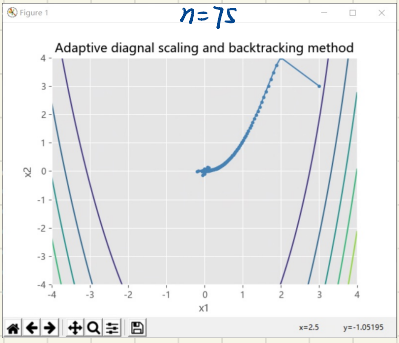


It uses more steps to converge to $(0,0)$, and uses less steps to converge to $(-1,\frac{3}{4})$

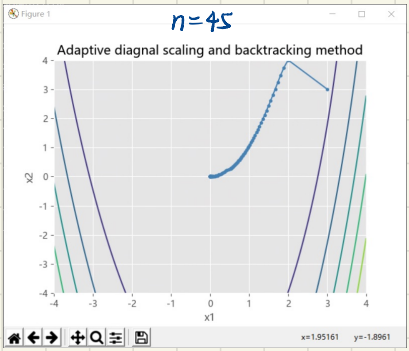
(c) I tried m in $[95, 90, \cdots, 10, 5]$, and with initial point $(3,3)$

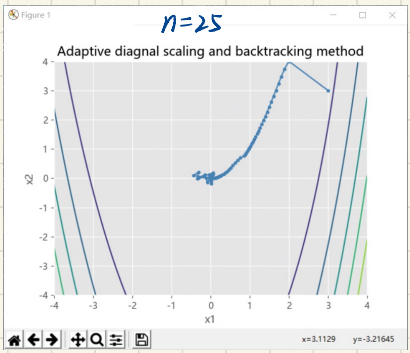
And the result is as following:

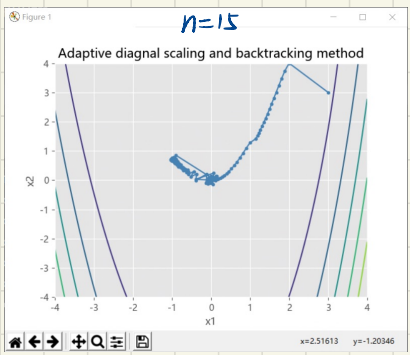


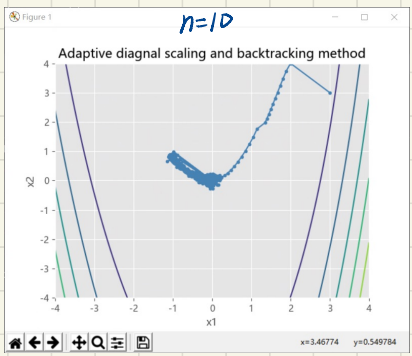


smooth









rugged

m = 95
Adaptive diagnal scaling and backtracking method converges in 203 steps, with accuracy 9.093465925912629e-06
final derivative value [6.68494927e-08 -7.00003440e-07] , final objective function value 6.555741890998225e-13
m = 90
Adaptive diagnal scaling and backtracking method converges in 199 steps, with accuracy 9.639863844187289e-06
final derivative value [4.53110212e-10 7.38971840e-07] , final objective function value 7.281059432349698e-13
m = 85
Adaptive diagnal scaling and backtracking method converges in 195 steps, with accuracy 7.1574555425490615e-06
final derivative value [5.74344924e-14 -5.61087611e-07] , final objective function value 4.197590767456513e-13
m = 80
Adaptive diagnal scaling and backtracking method converges in 183 steps, with accuracy 7.51302600066753e-06
final derivative value [1.64666036e-13 6.42664305e-07] , final objective function value 5.506898794161563e-13
m = 75
Adaptive diagnal scaling and backtracking method converges in 215 steps, with accuracy 7.811016984297556e-06
final derivative value [-2.26532784e-07 -2.45417442e-06] , final objective function value 8.056288248442264e-12
m = 70
Adaptive diagnal scaling and backtracking method converges in 202 steps, with accuracy 7.918621707763688e-06
final derivative value [-5.68670810e-15 -2.55824234e-06] , final objective function value 8.726138521912437e-12
m = 65
Adaptive diagnal scaling and backtracking method converges in 191 steps, with accuracy 6.6920776815175825e-06
final derivative value [5.06106675e-31 -2.16345556e-06] , final objective function value 6.240719926618743e-12
m = 60
Adaptive diagnal scaling and backtracking method converges in 182 steps, with accuracy 5.828274587814212e-06
final derivative value [-6.99946904e-15 -1.85609485e-06] , final objective function value 4.593450790535577e-12

m = 55
Adaptive diagnal scaling and backtracking method converges in 163 steps, with accuracy 3.049303908482302e-06
final derivative value [3.54470931e-08 -6.00442802e-07] , final objective function value 4.813369950497639e-13
m = 50
Adaptive diagnal scaling and backtracking method converges in 153 steps, with accuracy 6.455873374256729e-06
final derivative value [-1.69476601e-07 -1.18293705e-06] , final objective function value 1.880147983735883e-12
m = 45
Adaptive diagnal scaling and backtracking method converges in 96 steps, with accuracy 8.316757986886668e-06
final derivative value [5.24692584e-06 8.46621024e-07] , final objective function value 1.472085460554849e-11
m = 40
Adaptive diagnal scaling and backtracking method converges in 88 steps, with accuracy 7.2986613422622455e-06
final derivative value [2.78158744e-06 6.58905330e-07] , final objective function value 4.447493470982745e-12
m = 35
Adaptive diagnal scaling and backtracking method converges in 120 steps, with accuracy 3.893211822839622e-06
final derivative value [1.21563426e-08 8.25857265e-07] , final objective function value 9.094608512222755e-13
m = 30
Adaptive diagnal scaling and backtracking method converges in 105 steps, with accuracy 5.3376099884429616e-06
final derivative value [7.33089619e-07 1.03851340e-06] , final objective function value 1.7067227855525125e-12
m = 25
Adaptive diagnal scaling and backtracking method converges in 140 steps, with accuracy 9.06414610132815e-06
final derivative value [-8.27705149e-15 -2.81275826e-06] , final objective function value 1.0548812073031753e-11
m = 20
Adaptive diagnal scaling and backtracking method converges in 170 steps, with accuracy 7.705297331326579e-06
final derivative value [9.74021389e-07 -1.48637102e-06] , final objective function value 3.420094014038074e-12

m = 15
Adaptive diagnal scaling and backtracking method converges in 808 steps, with accuracy 9.477032421540415e-06
final derivative value [-4.47207288e-11 1.69105364e-06] , final objective function value 3.812883229576068e-12
m = 10
Adaptive diagnal scaling and backtracking method converges in 3979 steps, with accuracy 9.991975645116461e-06
final derivative value [-1.00212693 0.7531916] , final objective function value 0.08333333655204911

the result of $m=5$ is unattainable.

It can be seen that when m is small enough, the number of step to take to converge increase rapidly.

Problem 4 (Globalized Newton's Method): (approx. *25* pts)

Implement the globalized Newton method with backtracking that was presented in the lecture as a function `newton_glob` in `MATLAB` or `Python`.

The pseudo-code for the full Newton method is given as follows:

Algorithm 1: The Globalized Newton Method

Initialization: Select an initial point $x^0 \in \mathbb{R}^n$ and parameter $\gamma, \gamma_1, \gamma_2, \sigma \in (0,1)$ and `tol`.

for $k = 0, 1, \ldots$ **do**

  If $\|\nabla f(x^k)\| \leq$ `tol`, then STOP and x^k is the output.

  Compute the Newton direction s^k as solution of the linear system of equations:

$$\nabla^2 f(x^k) s^k = -\nabla f(x^k).$$

  If $-\nabla f(x^k)^\top s^k \geq \gamma_1 \min\{1, \|s^k\|^{\gamma_2}\}\|s^k\|^2$, then accept the Newton direction and set $d^k = s^k$. Otherwise set $d^k = -\nabla f(x^k)$.

  Choose a step size α_k by backtracking and calculate $x^{k+1} = x^k + \alpha_k d^k$.

The following input functions and parameters should be considered:

- `obj`, `grad`, `hess` – function handles that calculate and return the objective function $f(x)$, the gradient $\nabla f(x)$, and the Hessian $\nabla^2 f(x)$ at an input vector $x \in \mathbb{R}^n$. You can treat these handles as functions or fields of a class or structure `f` or you can use f, ∇f, and $\nabla^2 f$ from part a) and b) directly in the algorithm. (For example, your function can have the form `newton_glob(obj,grad,hess,...)`).
- x^0 – the initial point.
- `tol` – a tolerance parameter. The method should stop whenever the current iterate x^k satisfies the criterion $\|\nabla f(x^k)\| \leq$ `tol`.
- $\gamma_1, \gamma_2 > 0$ – parameters for the Newton condition.
- $\sigma, \gamma \in (0,1)$ – parameters for backtracking and the Armijo condition.

You can again organize the latter parameters in an appropriate `options` class or structure. You can use the backslash operator `A\b` in `MATLAB` or `numpy.linalg.solve(A,b)` to solve the linear system of equations $Ax = b$. If the computed Newton step $s^k = -\nabla^2 f(x^k)^{-1}\nabla f(x^k)$ is a descent direction and satisfies

$$-\nabla f(x^k)^\top s^k \geq \gamma_1 \min\{1, \|s^k\|^{\gamma_2}\}\|s^k\|^2,$$

we accept it as next direction d^k. Otherwise, the gradient direction $d^k = -\nabla f(x^k)$ is chosen. The method should return the final iterate x^k that satisfies the stopping criterion.

a) Test your approach on the Rosenbrock function $f : \mathbb{R}^2 \to \mathbb{R}$

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

with initial point $x^0 = (-1, -0.5)^\top$ and parameter $(\sigma, \gamma) = (0.5, 10^{-4})$ and $(\gamma_1, \gamma_2) = (10^{-6}, 0.1)$. (Notice that γ is smaller here). Besides the globalized Newton method also run the gradient method with backtracking ($(\sigma, \gamma) = (0.5, 10^{-4})$) on this problem and compare the performance of the two approaches using the tolerance `tol` $= 10^{-7}$.

Does the Newton method always utilize the Newton direction? Does the method always use full step sizes $\alpha_k = 1$?

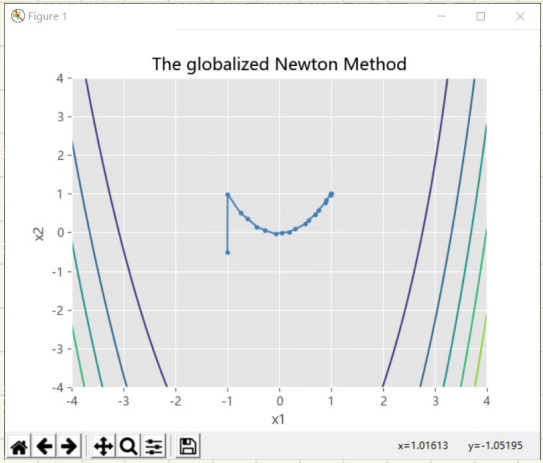
b) Repeat the performance test described in **Problem 2** b) and **Problem 3** b) for problem (1) using the globalized Newton method. You can use the parameter $(\sigma, \gamma) = (0.5, 0.1)$, $(\gamma_1, \gamma_2) = (10^{-6}, 0.1)$, and `tol` $= 10^{-5}$.

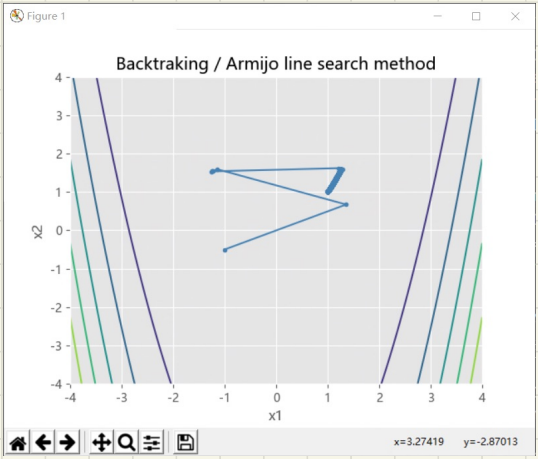
Plot all of the solution paths obtained by Newton's method for the different initial points in $\mathcal{X}^0$ in one figure (with a contour plot of f in the background).

Solution:

(a) I implement using python, the code is in the attachment

And the output is as following:






```
The globalized Newton Method converges in 21 steps, with accuracy 1.047779575182955e-12
final derivative value [1. 1.] , final objective function value 1.0462607323135512e-23
utilize the Newton direction 21 times, use full step size ak 16 times
```

```
Backtraking / Armijo line search method converges in 17205 steps, with accuracy 9.927503520556047e-08
final derivative value [1.00000008 1.00000016] , final objective function value 6.1697955620125855e-15
```

Note that:

number of steps = times to calculate d^k

$\therefore$ times to utilize Newton direction $\leq$ number of steps

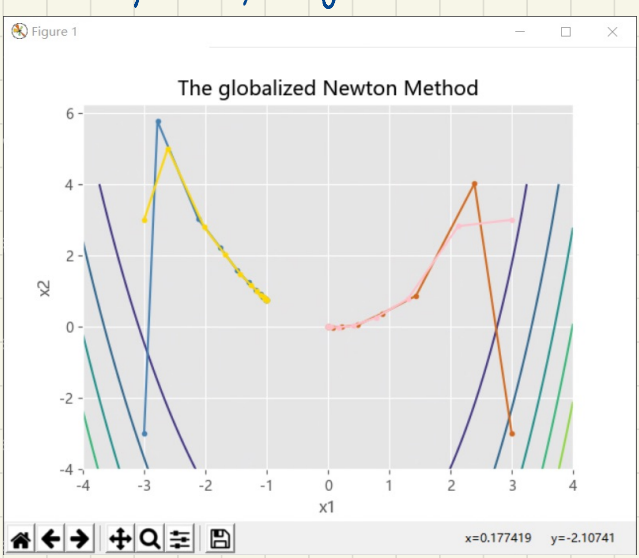
times to use full step size $a^k \leq$ numbe of steps -1

Newtons method always utilize Newton direction, and often use full step size $a^k = 1$

And it converge much faster than Backtracking method.

(b) I implement using python, the code is in the attachment

And the output is as following:



It uses much less steps to converge!!

```
Initial point: [-3 -3]
The globalized Newton Method converges in 15 steps, with accuracy 0.0009410188973456732
final derivative value [-1.00104611  0.75156918] , final objective function value 0.08333333371610396
utilize the Newton direction 15 times, use full step size ak 14 times
Initial point: [ 3 -3]
The globalized Newton Method converges in 10 steps, with accuracy 6.308679102360189e-07
final derivative value [ 6.12036882e-07 -1.52992276e-07] , final objective function value 2.1850368875427277e-13
utilize the Newton direction 10 times, use full step size ak 9 times
Initial point: [-3  3]
The globalized Newton Method converges in 15 steps, with accuracy 0.0008052537224110056
final derivative value [-1.00089492  0.75134239] , final objective function value 0.08333333357286998
utilize the Newton direction 15 times, use full step size ak 14 times
Initial point: [3 3]
The globalized Newton Method converges in 10 steps, with accuracy 7.043763554443152e-08
final derivative value [ 6.83368270e-08 -1.70745604e-08] , final objective function value 2.7236821516999796e-15
utilize the Newton direction 10 times, use full step size ak 9 times
```