The Simplified Geophysical Inversion Project

1, Overview

The Simplified Geophysical Inversion Project is a set of code that explores the minimum point of an objective function using optimization algorithms. It also stores the iteration steps per user's desire and store the searching history, including the result, residual and iteration steps into a .txt file.

Inversion of seismic wave is crucial in understanding subsurface information and properties. This involves the minimization of the misfit between the observed seismic data collected from field and predicted data generated from models. The concept behind minimizing the misfit is optimization of objective function, which is the difference between the observed data and predicted data, through manipulation of parameters of the model. This project proposes optimization algorithms that minimizes objective functions in 3D space. The objective function to minimize are simplified as equations in analytic form (i.e., sin(x)), and has two independent variables, x and y, and one dependent variable, value. In another word, objective function can be expressed as: value = f (x, y).

There are two optimization algorithms proposed, gradient descend method and quasi-newton method. User is asked to provide an initial guess of the minimum point before the operation, and the algorithms shifts the initial guess point to the true local minimum point. Quasi-newton method, which is the main optimization method, computes the direction and the amount of shifting by implicitly calculating the local gradient:

$$\begin{bmatrix} \frac{\partial value}{\partial x} & \frac{\partial value}{\partial y} \end{bmatrix} \tag{1}$$

and the inverse of local hessian:

$$\begin{bmatrix} \frac{\partial^2 value}{\partial x^2} & \frac{\partial^2 value}{\partial x \partial y} \\ \frac{\partial^2 value}{\partial y \partial x} & \frac{\partial^2 value}{\partial y^2} \end{bmatrix}^{-1} \tag{2}$$

Alternatively, gradient descend method, which computes shifting by calculating the local gradient and the maximum eigenvalue of the hessian, is an extension of quasi-newton method that can be used either 1) the user selects this method for higher computing speed, or 2) automatic switch from quasi-newton method if the local hessian is ill-conditioned.

Apart from optimization algorithms, the code stores the iteration steps, whose sequences can be set by the user, into a doubly linked list. The iteration steps, alongside with the final result of optimization are stored in a .txt file. This programme continues to ask for new optimization mission until user terminates the programme.

2, Optimization Algorithms

The quasi-newton method and the gradient descend method are designed and implemented based on the course materials of Geophysical Inversion module and Kiusalaas (2013).

Both quasi-newton and gradient descend methods are based on Newton's method of root finding and the Taylor series expansion. For the quasi-newton method, the residual step that shifts the minimizing point can be computed in matrix form:

$$\partial \boldsymbol{m} = -\alpha \boldsymbol{H}^{-1} \boldsymbol{\nabla}_{\boldsymbol{m}} f \tag{3}$$

where $\partial \boldsymbol{m}$ is the residual step, α is the learning rate, $\boldsymbol{H}$ is the hessian in the form of (2), and $\boldsymbol{\nabla}_{\boldsymbol{m}} f$ is the gradient of the function in the form of (1) above. Notably, the learning rate α is simplified as the same as the step size used for derivative computation. Therefore, α is renamed as the step size and cancels out with the step size in the gradient. We can rewrite (3) as:

$$\partial \boldsymbol{m} = -\begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}^{-1} \begin{bmatrix} \frac{f(x+\alpha,y)-f(x-\alpha,y)}{2} \\ \frac{f(x,y+\alpha)-f(x,y-\alpha)}{2} \end{bmatrix} \tag{4}$$

However, the hessian computation is often hampered by very small numbers, and sometimes uninvertible. The gradient descend method is an alternative approach:

$$\partial \boldsymbol{m} = -\lambda_{max} \begin{bmatrix} \frac{f(x+\alpha,y)-f(x-\alpha,y)}{2} \\ \frac{f(x,y+\alpha)-f(x,y-\alpha)}{2} \end{bmatrix} \tag{5}$$

in which λ_{max} is the maximum eigenvalue of the hessian.

3, Example Running of Code

The set up of this code is simple. As both the “data_structure.h” and “main_function.cpp” files are installed in the .sln project file, it will work when the input file “input.txt” is stored in the project folder. The “input.txt” file should contain the step size, the tolerance for hessian norm, the convergence tolerance, and the maximum number of iterations. The template of “input.txt” can be found in the Appendix 1.

Clicking the running button, the four parameters should be shown in the screen. Enter “y” to continue. After it, a few user inputs are required:

1) The objective function selection. Currently there are three functions to choose from:
$$1, f(x,y) = 2x^{-2} + y^{-2} + 3y^2 - 2x^{-1} + 4x^2 \tag{6}$$
$$2, f(x,y) = 0.3 * (x^2 + y^2) - 0.48xy \tag{7}$$
$$3, f(x,y) = \sin(xy) \tag{8}$$
Enter the number specified before the equation to continue. You can manually change the objective functions in the functions under the pointer class of .cpp file.
2) The initial guesses of x and y. For equation 1, the recommended first guess for x is 2, and for y is 2; for equation 2, the recommended first guess for x is 1, and for y is 1; for equation 3, there is a wide range of initial guess can make, if the values are not too large (i.e., x=2, y=3).
If the recommended initial guesses are implemented, the code should be able to return a result similar to the result computed from the “scipy.optimize” function in python (with method = BFGS). Admittedly, the final result is volatile due to the variation of initial guess and might causes failure of convergence.
3) The optimization method. Choose quasi-Newton method for better accuracy; choose gradient descend method for faster convergence and equations with complicated terms.
4) The number of iterations before recording the information at iteration step. The code can record the x-y value and the magnitude of residual step in the output file once every n iteration, specified by the user. Particularly, for equation 2, a recommended number is 200.

All inputs need to be valid to make the code works. If there are invalid input, such as enter 4 in the equation selection and any none-digit input, the code will terminate immediately to avoid crashing. A warning message will also show in the “result.txt” file.

If the inputs are valid, the optimization code will run. If the convergence is successful, there will be a message that shows the final x and y values of the minimal point, and its residual step size. The results and recorded steps will be print in the “result.txt” file. The containers that store these values will be deleted instantly. Following up will be the message “if you want to do this again”. Enter “y” if you want to have another go and enter “n” will terminate the programme.

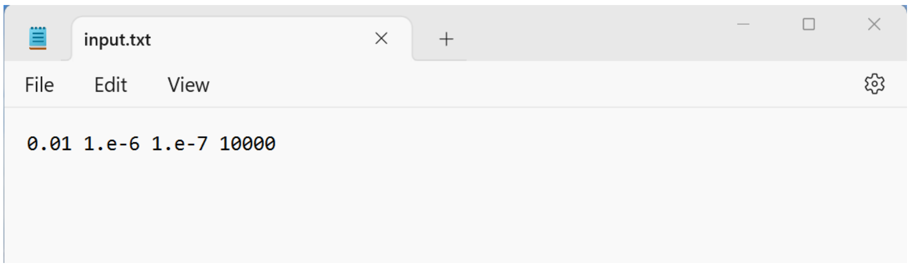
The code has been tested extensively to make it terminates elegantly when invalid input is detected and should be stable if the inputs are valid. However, rarely the code has stability issues not related to the code itself, including unexpected shutdown and main function running non-stop. If any of these happens, simply restart the code and everything should be normal.

Reference

Kiusalaas, J., 2013. Numerical methods in engineering with Python 3. Cambridge university press.

Appendix

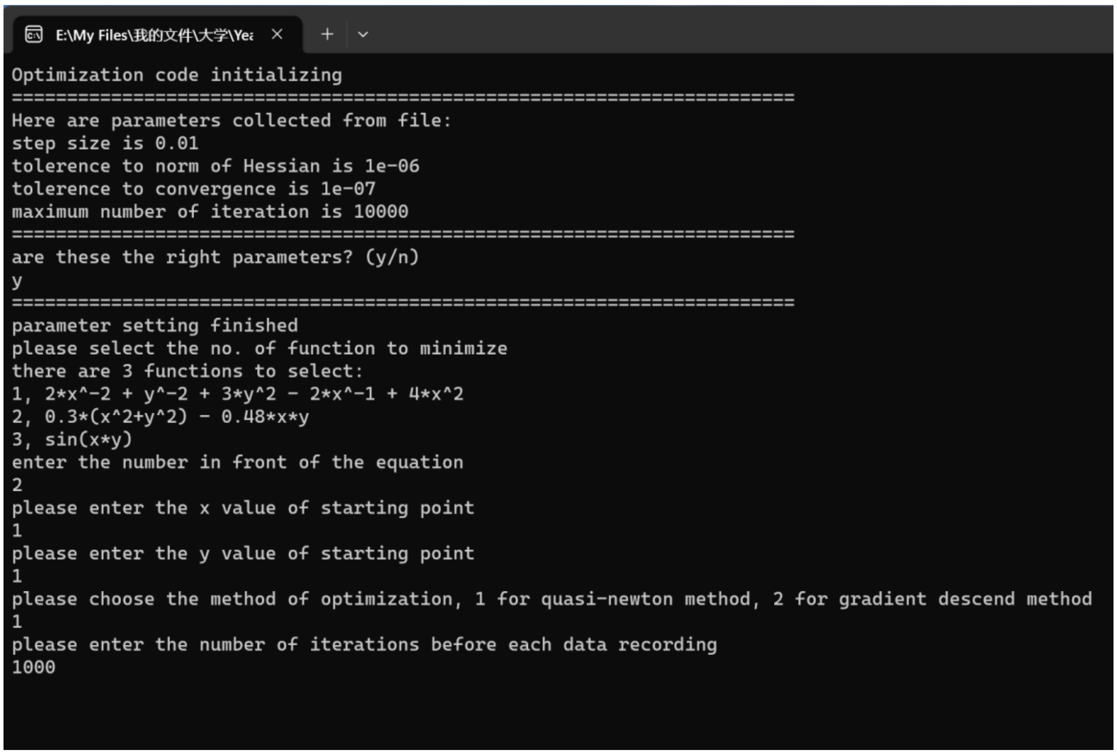
1, A template of input.txt file



```
0.01 1.e-6 1.e-7 10000
```

2, An Example User Input at the Start of Programme

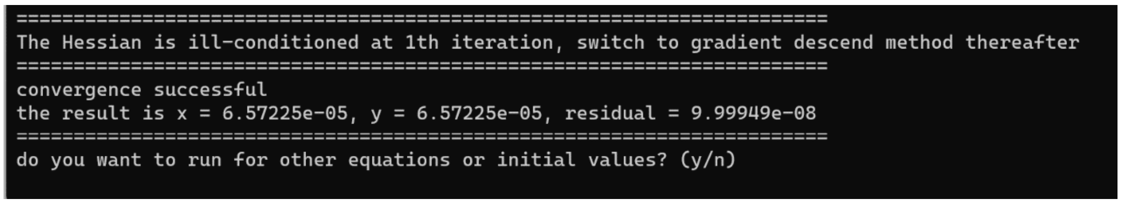
We are trying to optimize the second equation, with initial guess of x=1 and y=1. The optimization method is quasi-Newton, and the number of recording sequences is 1000 times of iteration.



```
Optimization code initializing
======================================================================
Here are parameters collected from file:
step size is 0.01
tolerence to norm of Hessian is 1e-06
tolerence to convergence is 1e-07
maximum number of iteration is 10000
======================================================================
are these the right parameters? (y/n)
y
======================================================================
parameter setting finished
please select the no. of function to minimize
there are 3 functions to select:
1, 2*x^-2 + y^-2 + 3*y^2 - 2*x^-1 + 4*x^2
2, 0.3*(x^2+y^2) - 0.48*x*y
3, sin(x*y)
enter the number in front of the equation
2
please enter the x value of starting point
1
please enter the y value of starting point
1
please choose the method of optimization, 1 for quasi-newton method, 2 for gradient descend method
1
please enter the number of iterations before each data recording
1000
```

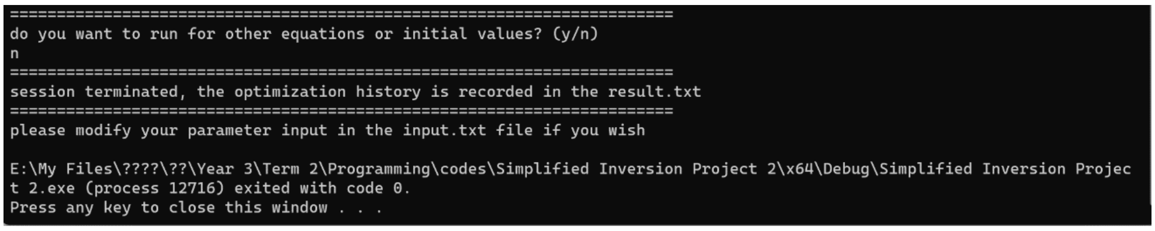
3, The Running Result output

The code automatically detects that from the first iteration on, the hessian is ill conditioned. Therefore, the code runs gradient descend method and shows a message notifying the change.



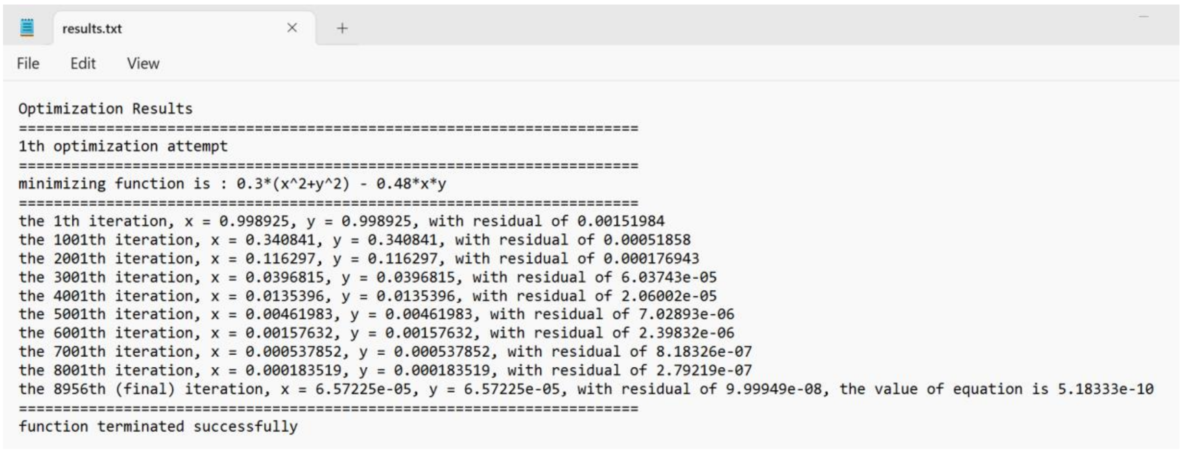
```
======================================================================
The Hessian is ill-conditioned at 1th iteration, switch to gradient descend method thereafter
======================================================================
convergence successful
the result is x = 6.57225e-05, y = 6.57225e-05, residual = 9.99949e-08
======================================================================
do you want to run for other equations or initial values? (y/n)
```

4, Termination of the programme



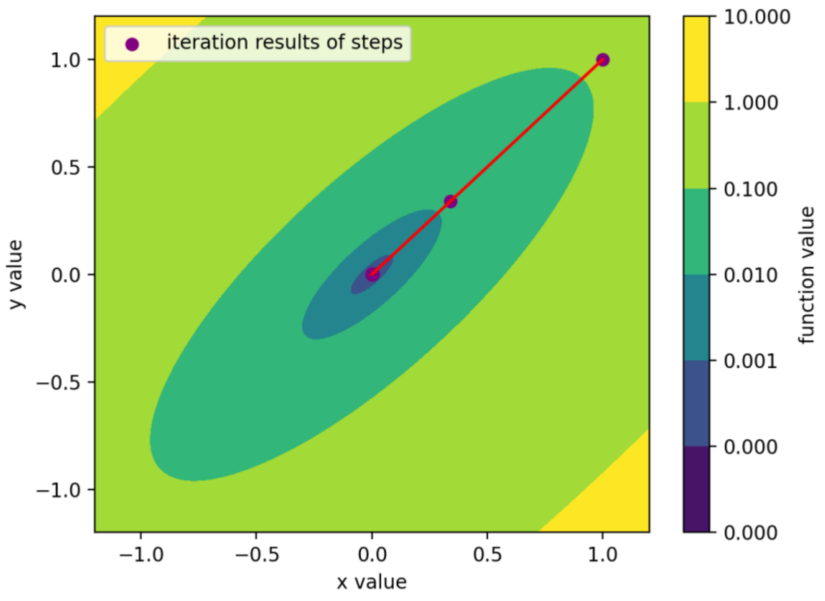
```
======================================================================
do you want to run for other equations or initial values? (y/n)
n
======================================================================
session terminated, the optimization history is recorded in the result.txt
======================================================================
please modify your parameter input in the input.txt file if you wish

E:\My Files\????\??\Year 3\Term 2\Programming\codes\Simplified Inversion Project 2\x64\Debug\Simplified Inversion Projec
t 2.exe (process 12716) exited with code 0.
Press any key to close this window . . .
```

5, The result.txt file, which stores the results from this operation



```
Optimization Results
======================================================================
1th optimization attempt
======================================================================
minimizing function is : 0.3*(x^2+y^2) - 0.48*x*y
======================================================================
the 1th iteration, x = 0.998925, y = 0.998925, with residual of 0.00151984
the 1001th iteration, x = 0.340841, y = 0.340841, with residual of 0.00051858
the 2001th iteration, x = 0.116297, y = 0.116297, with residual of 0.000176943
the 3001th iteration, x = 0.0396815, y = 0.0396815, with residual of 6.03743e-05
the 4001th iteration, x = 0.0135396, y = 0.0135396, with residual of 2.06002e-05
the 5001th iteration, x = 0.00461983, y = 0.00461983, with residual of 7.02893e-06
the 6001th iteration, x = 0.00157632, y = 0.00157632, with residual of 2.39832e-06
the 7001th iteration, x = 0.000537852, y = 0.000537852, with residual of 8.18326e-07
the 8001th iteration, x = 0.000183519, y = 0.000183519, with residual of 2.79219e-07
the 8956th (final) iteration, x = 6.57225e-05, y = 6.57225e-05, with residual of 9.99949e-08, the value of equation is 5.18333e-10
======================================================================
function terminated successfully
```

6, A plot of the optimization process



iteration results of steps
y value
x value
function value