Quasi-Newton methods

Marcus Greiff

6/16/2016

1 Introduction

This project is concerned with creating a solver for unconstrained non-linear optimization algorithm in Matlab, using rank-2 updated Newton iterations (DFP and BFGS) in combination with a line-search algorithm. The primary purpose is to create a decent optimization tool, comparable to Matlab's `fminunc` fucntion but without the need for an expensive license, once the code is proven functional, it can easily be rewritten in C or Python.

The work is divided into three parts, where the first aims to find a good line-search method by investigating performance on a 2D-test function with two global minimums on the scale of $|\mathbf{x}| \approx 10^{58}$ and $|\mathbf{x}| \approx 10^{-58}$. The second part evaluates the non-linear solver in terms of consistency and convergence for the notoriously difficult Rosenbrock function. The final part solves three constrained problems using the non-linear solver and penalty/barrier-methods, and the result compared to the the `fminunc` function of Matlab's Optimization toolbox.

The code in **Section 2** and **Section 3** can be run independently of eachother, but the directory of `nonlinearmin.m` and `linesearch.m` of **Section 3** needs to be accessible when running the code to **Section 4**. A simple way to do this would be to use the `addpath` command, or putting the code from all three parts in the same directory.

2 Linesearches

Three line search methods were implemented in Matlab, based on the golden section method, the newton method, and Armijjo's rule. These methods were tuned to yield as good results as possible on the extremely difficult test function

$$f(\mathbf{x}) = (10^{58}x_1 - 1)^{100} + (10^{-58}x_2 - 1)^2 - 2; \tag{1}$$

in the directions $\mathbf{d} = (0,1)$ and $\mathbf{d} = (1,0)$, effectively solving the problem

$$\min_{\lambda}(f(\mathbf{x} + \lambda\mathbf{d})). \tag{2}$$

This testfunction is particularly treacherous, as it has one minimum on scale of $\lambda \approx 10^{-58}$ in some directions, and $\lambda \approx 10^{58}$ in others (see Figure 1).

This implies that the linesearches must be very robust and have and extremely low tolerance, to actually find a minimum in the search domain. For example, if we use the Golden section

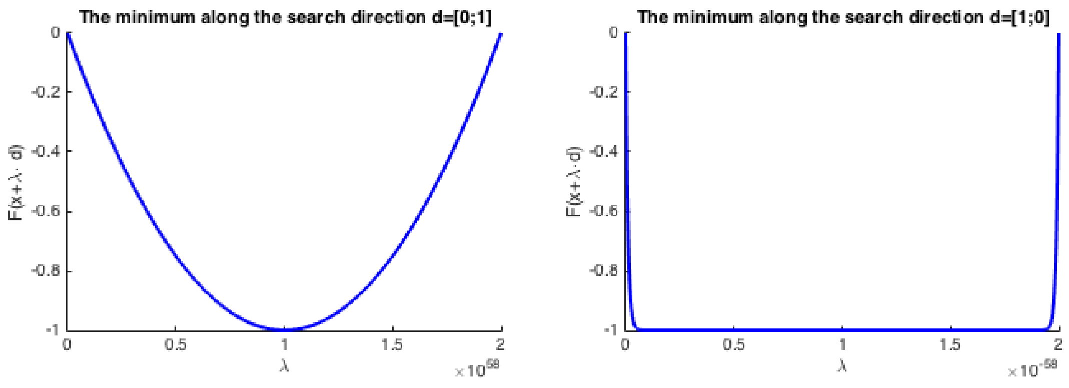


Figure 1: The minima of the test function from the point $\mathbf{x} = (0, 0)$ along the directions $\mathbf{d} = (0, 1)$ (left) and $\mathbf{d} = (1, 0)$ (right).

method, the initial interval must encompass all possible minimising lambda solving (2) in the direction $\mathbf{d} = (0, 1)$, which is done in for example $I_0 = [a_1, b_1] = [0, 10^{59}]$. However, to find the minimum in the second search direction, we must demand an incredibly small final interval, on the order of $L = 10^{-58}$. Since we cannot alter this tolerance based on the search direction, we have a fundamental restriction on how many times, N, we need to shorten the interval before reaching a satisfactory minimum,

$$\left(\frac{1}{\alpha}\right)^{N-1} \leq \frac{b_1 - a_1}{L} = 10^{117} \Leftrightarrow N \leq \frac{\log(10^{117})}{\log(\alpha^{-1})} + 1 = 559.8417 \tag{3}$$

This result can clearly be seen in (see Table 1), from which we may conclude that every linesearch performed with the golden secttion method, with this tuning will take ≥ 0.040557, which is completely unacceptable is if the method is to be used a vast number of times. When instead looking at the Newton method, we see that the magnitude derivative of the (2) in the direction $\mathbf{d} = (1, 0)$ is extremely large, and that the approximation of the derivative must be made very close to the point $\lambda = 0$ in order for the method to find the minimum. When, approximating the derivative such that $\Delta\lambda \approx 10^{-58}$ the method find a local minimum very quickly in this direction. However, in the other direction, the derivative in 0 is of a very small magnitude $F'_\lambda(\mathbf{x}) \approx 10^{-58}$ and the method has trouble converging to start with, and the derivative approximations risk becoming undefined when using them small $\Delta\lambda$ required in the other search direction (see Table 2).

As for the inexact line search with Armijjo's rule (as described in), implemented such that the step decreases by a factor $\beta = 0.83$ when the stopping criterion is not met, and with a slope relaxation parameter $\sigma = 0.3$, we converge to the minimum in the search direction $\mathbf{d} = (0, 1)$ very quickly (see Table 2), by far outperforming the GS method. Furthermore, the computations necessary to evaluate one iteration in the algorithm is simpler in the inexact linesearch, when compared to the Golden Section. Consequently, despite the inexact algorithm taking more iterations than the Golden section method when searching along $\mathbf{d} = (1, 0)$, a suitable minimum is found in roughly half the time of that when using GS (see Table 1). Based on this result, the inexact line search was used in the `nonlinearmin.m` function, referred to as `linesearch.m` for future reference. This method was tweaked slightly, so as to start at the point $\lambda = 1$, and then search for a minima in both directions, drastically decreasing the number of iterations required to find a minima in the functions considered (as the default λ in Newton's original method is $\approx$). The results were computed in the script `linesearch_demo.m`.

Method	Newton	Armijjo	Golden Section
λ_{opt}	5.0000e-59	1.8139e-58	4.6333e-59
$F(\mathbf{x} + \lambda_{opt}\mathbf{d})$	-1	-1	-1
Number of iterations	1	1456	560
Computational time [s]	0.0019998	0.026227	0.040557

Table 1: Finding a minimum to the test function, $F(\mathbf{x} + \lambda_{opt}\mathbf{d})$, with $\mathbf{x} = (0, 0)$ along $\mathbf{d} = (1, 0)$ with Newton-, Armijjo- and Golden Section linesearches.

Method	Newton	Armijjo	Golden Section
λ_{opt}	NaN	1.9982e+58	1.0000e+58
$F(\mathbf{x} + \lambda_{opt}\mathbf{d})$	NaN	-0.0036	-1
Number of iterations	1	22	560
Computational time [s]	0.0023019	0.0016266	0.041348

Table 2: Finding a minimum to the test function, $F(\mathbf{x} + \lambda_{opt}\mathbf{d})$, with $\mathbf{x} = (0, 0)$ along $\mathbf{d} = (0, 1)$ with Newton-, Armijjo- and Golden Section linesearches.

3 Algorithm testing

3.1 Convergence

When running the algorithms with no line search to minimize the unconstrained problem

$$f(\mathbf{x}) = (x_1 + 2)^p + (x_2 - 2)^p \tag{4}$$

which is convex for all even $p \geq 2$, we should expect the rate of convergence to decrease as the order of the polynomial function increases. The reason for this is that the different Newron-Rhapson schemes used are derived from a second order approximation of the objective function, thus, an objective function with an order far from $p = 2$ should converge slower, and convergence should be found in a single step when the objective function is quadratic. This is clearly the case when running the `nonlinearmin.m` on the problem above with different polynomial orders of (4) with a starting point of $\mathbf{x}_0 = (10, 10)$ (see Table 3). We also note that a difference in convergence rate between the Newton method and the updated newton methods, this is due to the updated methods approximating the inverted hessian instead of computing a new hessian on each iteration. We get an accumulative small error which decreases the rate of convergence, but this vastly improves the computational time for larger problems, as computing and inverting the hessian is very costly. Finally, when we try to minimise a non-convex function, such as (4) with

p	Regular Newton	DFP	BFGS
2	1	1	1
4	15	22	22
6	21	30	30
8	27	39	39

Table 3: The number of iterations required to converge using the three different methods on problems of order p. *Note that this result was made using no line search and a single newton iteration loop, just to chow the general convergence properties of the methods as implemented in Matlab.*

an odd $p \geq 3$, we expect the method to diverge as no global minima exist, and this was behaviour was indeed confirmed.

The script `nonlinearmin_demo.m` was used to generate the data, but in order to replicate the result, the line-search method must be turned off manually inside the non-linear solver, and a single newton loop must be used. When using two loops (one outer load loop and one inner iteration loop) and approximating the inverted hessian as the identity matrix on each iteration loop, the DFP and BFGS methods were used to find the minima of the Rosenbrock function, defined as

$$f_2(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2. \tag{5}$$

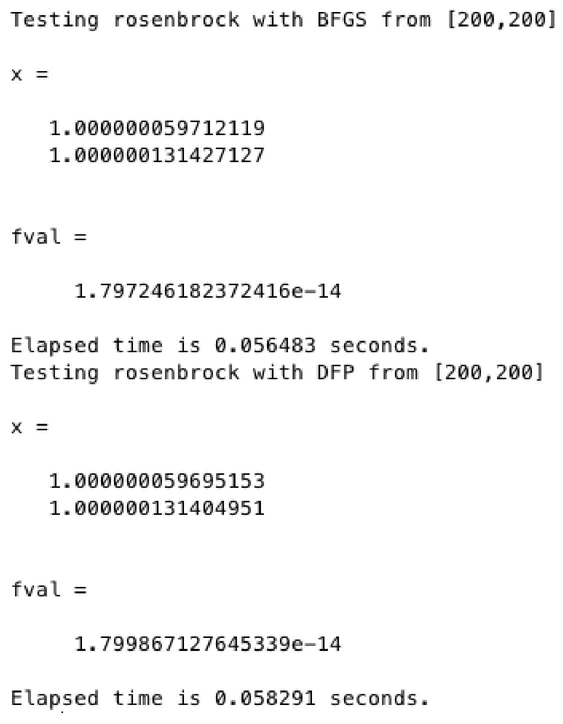
The starting point was set to $\mathbf{x}_0 = 200$, the numerical tolerance to 10^{-6} and the solution (see Figure 2) comes very close to the analytical minimum, at $\mathbf{x}_{opt} = (1, 1)$ with $f(\mathbf{x}_{opt}) = 0$.

Both methods reach the minima of the Rosenbrock function, and the function value is correct down to ≈ 15 decimals. But we see a clear difference in convergence rate, as the BFGS requires 31 load iterations to converge, and the DFP method needs twice that, converging after a total of 61 load steps. This results in a computational time which is slightly shorter in the BFGS method at ≈ 0.056 s (59 load iterations) when compared to that of the DFP method, at ≈ 0.060 s (59 load iterations). This shows that for more difficult problems, the BFGS is likely the better method to use, especially when approximating the inverted hessian by the identity matrix on each load step. The BFGS method is well known to have better direction-correcting properties than the DFP method, hence the slight difference in computational time can be expected to increase for larger problems. When the number of inner iterations increase (same as the number of variables, n), a poor approximated initial hessian of $\mathbf{I}_n$ will be more difficult to correct using the DFP method.

3.2 Consistency

To test the consistency of the algorithm, the scrip `nonlinearmin_demo.m` was written to minimise two different convex 2D-functions, with randomly selected starting points in the set $\{(x_1, x_2) \in \mathbb{Z} : -200 \leq x_1 \leq 200, -200 \leq x_1 \leq 200\}$, and checking the solution against the known optimal solution $\mathbf{x}_{opt}$ and $f(\mathbf{x}_{opt})$. This was repeated 50 times, and successful solution was defined as being close to the optimal solution both in terms of position in the x_1, x_2-plane, and being close to the optimal function value. This criteria for a computed solution, $\mathbf{x}_{comp}$, was formulated as

$$\max(|\mathbf{x}_{opt} - \mathbf{x}_{comp}|) < \epsilon \quad \text{and} \quad |f(\mathbf{x}_{opt}) - f(\mathbf{x}_{comp})| < \epsilon. \tag{6}$$



```
Testing rosenbrock with BFGS from [200,200]

x =

   1.000000059712119
   1.000000131427127


fval =

     1.797246182372416e-14

Elapsed time is 0.056483 seconds.
Testing rosenbrock with DFP from [200,200]

x =

   1.000000059695153
   1.000000131404951


fval =

     1.799867127645339e-14

Elapsed time is 0.058291 seconds.
```

Figure 2: The solution to the problem of minimising the Rosenbrock function starting from $\mathbf{x}_0 = (200, 200)$ with a tolerance of 10^{-6} using BFGS and DFP respectively.

The two test functions were defined as

$$f_1(\mathbf{x}) = (x_1+2)^2 + (x_2-2)^2 + (x_2-2)^4 \tag{7}$$

and

$$f_2(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1-x_1)^2, \tag{8}$$

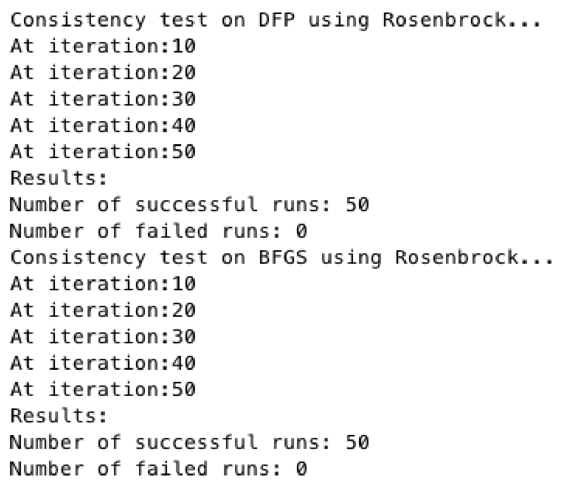
where the latter is a variation of the Rosenbrock function, a notoriously difficult optimisation problem. The optimal solutions are easily seen to be

$$\begin{cases} \mathbf{x}_{opt}^{(1)} = (-2,2) \\ \mathbf{x}_{opt}^{(2)} = (1,1) \end{cases} \tag{9}$$

and the optimal function value is $f_{opt} = 0$ in both cases.

Using $\epsilon = 10^{-5}$ and running the consistency test with f_1, both the DFP and the BFGS methods pass with flying colours, generating the correct solution 100% of the time using both methods.

When running this consistency check with 50 tests using $\epsilon = 10^{-3}$, the computed solution was accurate in 100% of the cases, both with DFP and BFGS (see Figure 3). Having proved consistency of the methods, the solver was applied to three problems of constrained nonlinear optimisation, using penalty and barrier methods.



```
Consistency test on DFP using Rosenbrock...
At iteration:10
At iteration:20
At iteration:30
At iteration:40
At iteration:50
Results:
Number of successful runs: 50
Number of failed runs: 0
Consistency test on BFGS using Rosenbrock...
At iteration:10
At iteration:20
At iteration:30
At iteration:40
At iteration:50
Results:
Number of successful runs: 50
Number of failed runs: 0
```

Figure 3: Results on the consistency check for the Rosenbrock function.

4 Constrained optimisation examples

4.1 Problem 1

The problem is to

$$\min(e^{x_1x_2x_3x_4x_5}) \quad \text{subject to} \quad \begin{cases} h_1(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0 \\ h_2(\mathbf{x}) = x_2x_3 - 5x_4x_5 = 0 \\ h_3(\mathbf{x}) = x_1^3 + x_3^3 + 1 = 0 \end{cases} \tag{10}$$

using a penalty formulation and the sequence $\mu = [0.01, 0.1, 1, 10, 100]$. To do this, we define the penalty function as

$$\beta(\mathbf{x}) = h_1(\mathbf{x})^2 + h_2(\mathbf{x})^2 + h_3(\mathbf{x})^2 \tag{11}$$

as a quadratic function of the constraints in order to get a good rate of convergence. The auxiliary function is then

$$aux(\mathbf{x}) = f(\mathbf{x}) + \mu\beta(\mathbf{x}) \tag{12}$$

where f is the objective function. The solver was implemented in the script `Problem1.m`, in which tolerances, methods and μ-strategies can be easily defined by the user. In addition, the solution history is stored in the variable `solhist`, and the Lagrange multipliers for the final point are estimated and stored in the variable `multiplies`.

When solving the first problem with an initial point $\mathbf{x}_0 = (-2, 2, 2, -1, -1)$ the methods "fminunc", "BFGS" and "DFP" yield the very same solution history, down to a decimal specified by the tolerance in the sovlver (see Figure 5). This result indicates that the methods have been implemented correctly.

When looking at the constraints at the initial point x_0, we find that

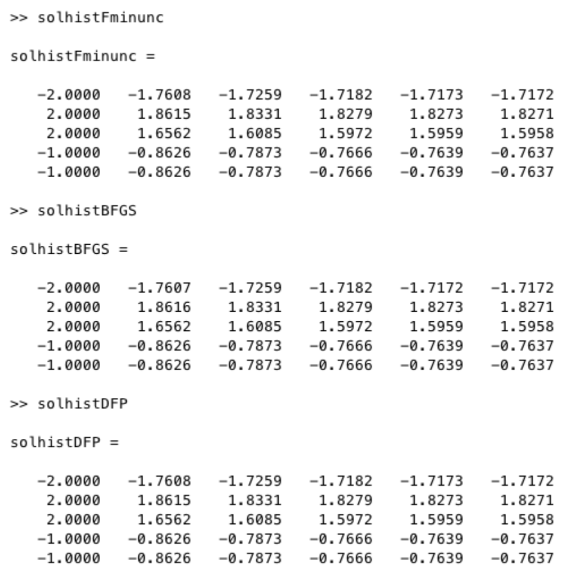
$$\begin{cases} h_1(\mathbf{x}_0) = 4 \\ h_2(\mathbf{x}_0) = -1 \\ h_3(\mathbf{x}_0) = 1 \end{cases} \tag{13}$$

indicating that the initial point resides in the complement to the feasible set, and that it therefore is a good starting point for the minimisation using a penalty method. In addition, when estimating the Lagrange multipliers at the terminal solution $\mathbf{x}_{opt} \approx \mathbf{x}_5 = (-1.7172, 1.8271, 1.5958, -0.7637, -0.7637)$, we find that

$$\begin{cases} v_1(\mathbf{x}_{opt}) = 0.0401 \\ v_2(\mathbf{x}_{opt}) = -0.0380 \\ v_3(\mathbf{x}_{opt}) = 0.0052 \end{cases} \tag{14}$$

which are all reasonably close to zero, indicating that we are very close to the feasible set. Furthermore, we see that the objective function value increases on every iteration, which is precisely what we would expect when approaching the optimal point from a point outside of the feasible set, which has a lower value that the optimal point on the feasible set (see Table 4).
6).

This solution can be concluded to be a good approximation of the minimum to (10), for which the function value is $f(\mathbf{x}_{opt}) = 0.0539$.



```
>> solhistFminunc

solhistFminunc =

   -2.0000   -1.7608   -1.7259   -1.7182   -1.7173   -1.7172
    2.0000    1.8615    1.8331    1.8279    1.8273    1.8271
    2.0000    1.6562    1.6085    1.5972    1.5959    1.5958
   -1.0000   -0.8626   -0.7873   -0.7666   -0.7639   -0.7637
   -1.0000   -0.8626   -0.7873   -0.7666   -0.7639   -0.7637

>> solhistBFGS

solhistBFGS =

   -2.0000   -1.7607   -1.7259   -1.7182   -1.7172   -1.7172
    2.0000    1.8616    1.8331    1.8279    1.8273    1.8271
    2.0000    1.6562    1.6085    1.5972    1.5959    1.5958
   -1.0000   -0.8626   -0.7873   -0.7666   -0.7639   -0.7637
   -1.0000   -0.8626   -0.7873   -0.7666   -0.7639   -0.7637

>> solhistDFP

solhistDFP =

   -2.0000   -1.7608   -1.7259   -1.7182   -1.7173   -1.7172
    2.0000    1.8615    1.8331    1.8279    1.8273    1.8271
    2.0000    1.6562    1.6085    1.5972    1.5959    1.5958
   -1.0000   -0.8626   -0.7873   -0.7666   -0.7639   -0.7637
   -1.0000   -0.8626   -0.7873   -0.7666   -0.7639   -0.7637
```

Figure 4

Iteration (i)	0	1	2	3	4	5
$f(\mathbf{x_i})$	0.0003	0.0176	0.0427	0.0525	0.0538	0.0539

Table 4: The objective function evaluated at each solution $\mathbf{x}_i$.

When using a point with a large initial value of f, where the gradient of the objective function is of a much higher magnitude than the previously tested initial point $\mathbf{x}_0 = (-2, 2, 2, -1, -1)$, the strategy of choosing μ meeds to be adjusted. By choosing the initial point as $\hat{\mathbf{x}_0} = -\mathbf{x}_0 = (2, -2, -2, 1, 1)$, the constraints are violated by a magnitude similar to that of the previously tested point,

$$\begin{cases} h_1(\hat{\mathbf{x}_0}) = 4 \\ h_2(\hat{\mathbf{x}_0}) = -1 \\ h_3(\hat{\mathbf{x}_0}) = 1 \end{cases} \tag{15}$$

but the initial function value is radically different, at $f(\hat{\mathbf{x}_0}) = 2.9810 \cdot 10^3$. Starting here with the μ strategy of the previous problem, the optimal solution doesn't converge to the previous minimum, but to a $f(\hat{\mathbf{x}}_{opt}) \approx 1$ which is much higher.

4.2 Problem 2

The problem is to

$$\min(e^{x_1} + x_1^2 + x_1 x_2) \quad \text{subject to} \quad h(\mathbf{x}) = \frac{1}{2}x_1 + x_2 - 1 = 0 \tag{16}$$

using a penalty formulation with a strategy $\mu = 4$ for three iterations. The penalty function is simply defined as a quadratic function of the residual, and the auxiliary problem can be written

$$aux(x) = f(\mathbf{x}) + \mu\alpha(\mathbf{x}) = f(\mathbf{x}) + \mu h(\mathbf{x})^2. \tag{17}$$

where f is the objective function. This problem was minimised using the `Problem3.m` and set up so that the user may choose to use the `nonlinearmin.m` method, with "BFGS", "DPF" or the Matlab `fminunc` as a reference solution. The results are plotted automatically in the script, and the problem was solved for the starting points $\mathbf{x}_0 = \{(-10, 0), (-8, 8), (-4, 4)\}$ (see Figure 5).

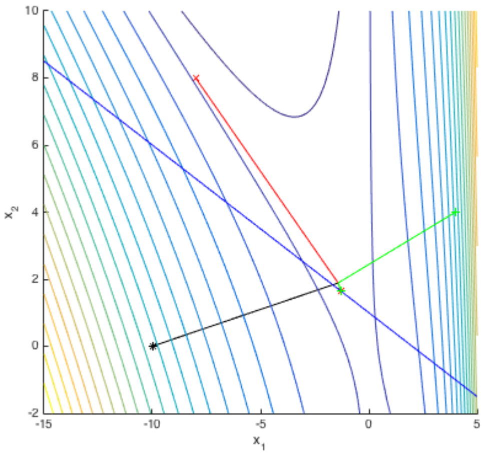


Figure 5: The level curves of the objective function, constraint (blue) and solution to problem 2 with three different initial starting points, $\mathbf{x}_0 = (-10, 0)$ (black), $\mathbf{x}_0 = (-8, 8)$ (red), $\mathbf{x}_0 = (-4, 4)$ (green).

Graphically, solution converges to the same point in the constraint line (blue) for all tested initial points in the exterior of the feasible set, as expected. This implies that the problem has been set up and solved correctly. In addition, when looking at the solution history in more detail with the starting points $\mathbf{x}_0 = (-4, 4)$, the solutions are identical down to the numerical tolerance (see Table 6) on *each* iteration. For illustrative purposes, the problem was here minimised with $\mu = [4, 40, 400]$. This indicates that both the BFGS and DFP methods have both been implemented correctly. When solving the problem for a fixed $\mu = 4$ in all auxiliary problems, the algorithm takes one step, but then remains in the same position (as we effectively solve a minimisation problem with a starting point already in the global minimum when the parameter μ remains the same through the iterations).

When looking at the number of newton iterations for each solver, the BFGS and DFP methods converge to the solution is roughly the same time (≈ 0.28s). There is a slight difference in the number of internal steps required, as expected from the discussion in **Section 2**, but in practice, both methods work sufficiently well (see Table 6).

In conclusion, the solution to the problem after three iterations with $\mu = 4$ is $\mathbf{x}_{opt} = (-1.419, 1.887)$ with a lagrange multiplier estimated to $v_3 = 1.4193$ in the terminal state. This

Iteration	0	1	2	3
$x_1^{(bfgs)}$	4.0000	-1.4193	-1.2911	-1.2797
$x_2^{(bfgs)}$	4.0000	1.8871	1.6617	1.6415
$x_1^{(dfp)}$	4.0000	-1.4193	-1.2911	-1.2797
$x_2^{(dfp)}$	4.0000	1.8871	1.6617	1.6415
$x_2^{(fminunc)}$	4.0000	-1.4193	-1.2911	-1.2797
$x_2^{(fminunc)}$	4.0000	1.8871	1.6617	1.6415

Table 5: Solution history when starting at $\mathbf{x}_0 = (4, 4)$ for the BFGS scheme (top), DFP scheme (center) and Matlab's "fminunc" (bottom).

Iteration (i)	1	2	3
μ_i	4	40	400
BFGS	25	12	11
DFP	19	16	16

Table 6: Number of load iterations required to converge at each auxillary problem, with the μ used in the auxiliary problem formulation.

is exactly the result which can be found in the course book. However, a better strategy is to use $\mu = [4, 40, 400]$ which reduces the Lagrange multiplier to $v_3 = 0.0128$ i.e. the point is much closer to the constraint line, at an optimal solution of $\mathbf{x}_{opt} = (-1.2797, 1.6415)$.

4.3 Problem 3

The problem is to

$$\min((x_1 - 5)^2 + (x_2 - 3)^2) \quad \text{subject to} \quad x_1 + x_2 \leq 3, \quad -x_1 + 2x_2 \leq 4 \tag{18}$$

using a barrier forlmulation and two iterations with the strategy $\epsilon = 1$. The barrier function was defined as

$$\beta(\mathbf{x}) = -\frac{1}{g_1(\mathbf{x})} - \frac{1}{g_2(\mathbf{x})} \tag{19}$$

where

$$\begin{cases} g_1(\mathbf{x}) = x_1 + x_2 - 3 \leq 0 \\ g_2(\mathbf{x}) = -x_1 + 2x_2 - 4 \leq 0 \end{cases} \tag{20}$$

and implemented in the Matlab function `barrierP3.m` such that $\beta(\mathbf{x}) = 10^{99}\ \forall \mathbf{x}$ which violate the constraints (20). This was done so that the line-search methods would detect the barrier and not cross it. The auxillary problem thus becomes

$$aux(x) = f(\mathbf{x}) + \epsilon\beta(\mathbf{x}), \tag{21}$$

where f is the objective function. The solver to this problem was implemented in `Problem3.m` and written similarly to that of Problem 2. The user can choose tolerances and one of the four defined methods in the script, and the objective function level curves as well as the constraint are plotted.

The ϵ-sequence was chosen as $[1, 0.1, 0.01]$ and determined by testing different sequences on the problem numerically. When using this ϵ-sequence and `nonlinearmin.m` solver, with three different starting points on the interior of the feasible set, the DFP and BFGS, all solutions converge to the same point on the first iteration and then successively approach the theoretical minimum at $\mathbf{x}_{opt} = (2.5, 0.5)$ (see Figure 6 and Figure 7).

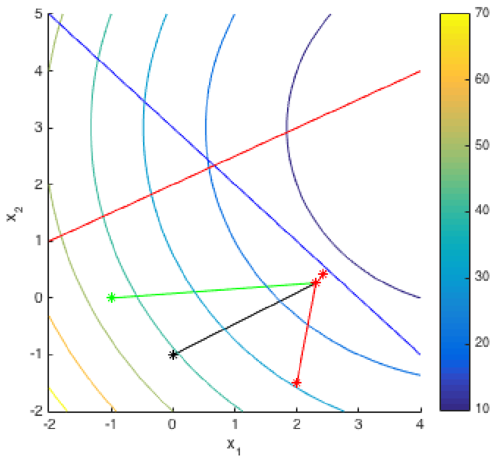


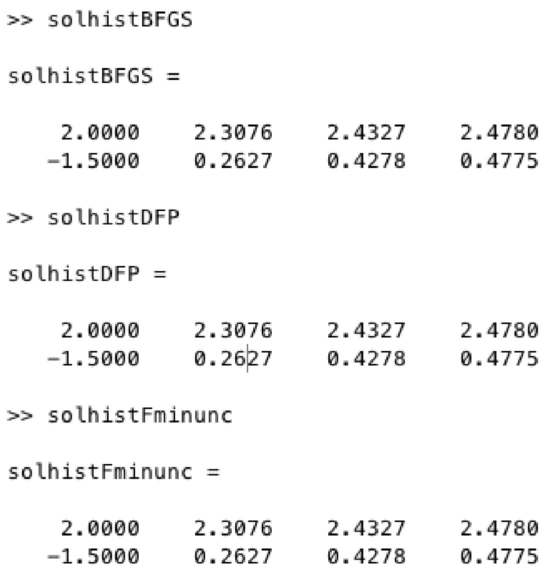
Figure 6: The level curves of the objective function, constraint $g_1(\mathbf{x})$ (blue) and $g_2(\mathbf{x})$ (red), as well as the solution to problem 3 with three different initial starting points, $\mathbf{x}_0 = (0, -1)$ (black), $\mathbf{x}_0 = (2, -1.5)$ (red), $\mathbf{x}_0 = (-1, 0)$ (green).

Matlab's `fminunc` function was used to confirm the accuracy of the penalty formulation, and gave the same solution as the BFGS and DFP methods (down to the a numerical error specified by tolerance). Thus confirming that the penalty method and the `nonlinearmin` solver had been implemented correctly. When starting at the point $\mathbf{x}_0 = (2, -1.5)$, the function values corresponding to the each decreasing monotonically, as expected when using the penalty method on such the relatively simple problem (18) (see Table 7).

Iteration (i)	0	1	2	3
$f(\mathbf{x_i})$	29.2500	14.7421	13.2073	12.7236

Table 7: The objective function evaluated at each solution $\mathbf{x}_i$.

In addition, the function values tend to 12.5, which can easily be seen to be the theoretical, optimal value of the problem. In conclusion, the terminal point of $\mathbf{x}_{comp} = (2.4780, 0.4775)$,



```
>> solhistBFGS

solhistBFGS =

    2.0000    2.3076    2.4327    2.4780
   -1.5000    0.2627    0.4278    0.4775

>> solhistDFP

solhistDFP =

    2.0000    2.3076    2.4327    2.4780
   -1.5000    0.2627    0.4278    0.4775

>> solhistFminunc

solhistFminunc =

    2.0000    2.3076    2.4327    2.4780
   -1.5000    0.2627    0.4278    0.4775
```

Figure 7

with $f(\mathbf{x}_{comp}) = 12.7236$ was found when using the ϵ-sequence was chosen as $[1, 0.1, 0.01]$. The same result (down to a small numerical error) was given by all the three solvers BFGS, DFP and Matlab's `fminunc` and the computational time when using the BFGS compared to the DFP scheme was very small on account of the simplicity of the problem. Finally, the computed solution is comparable to the result in the book, further validating the solver.