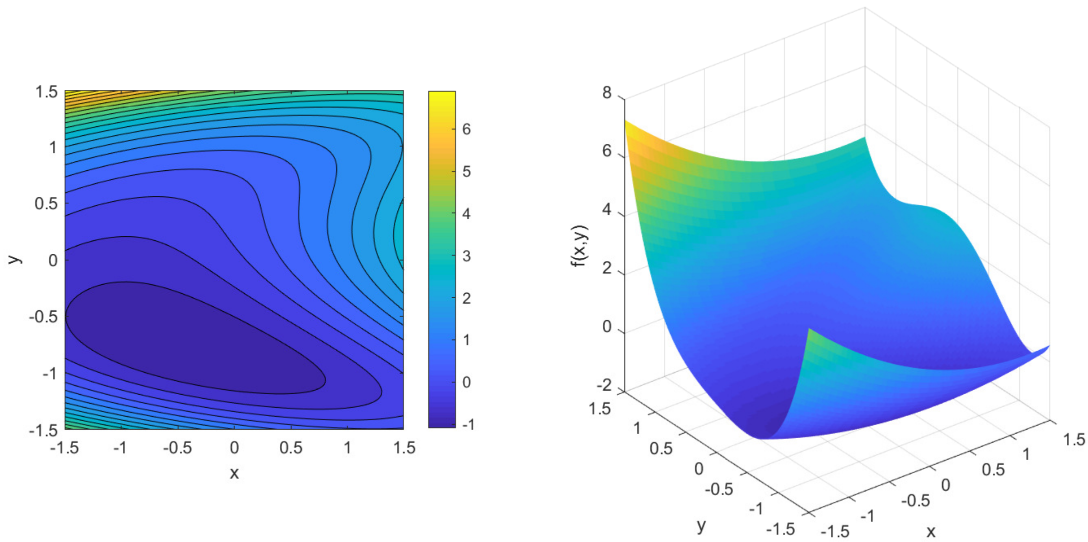
7.3

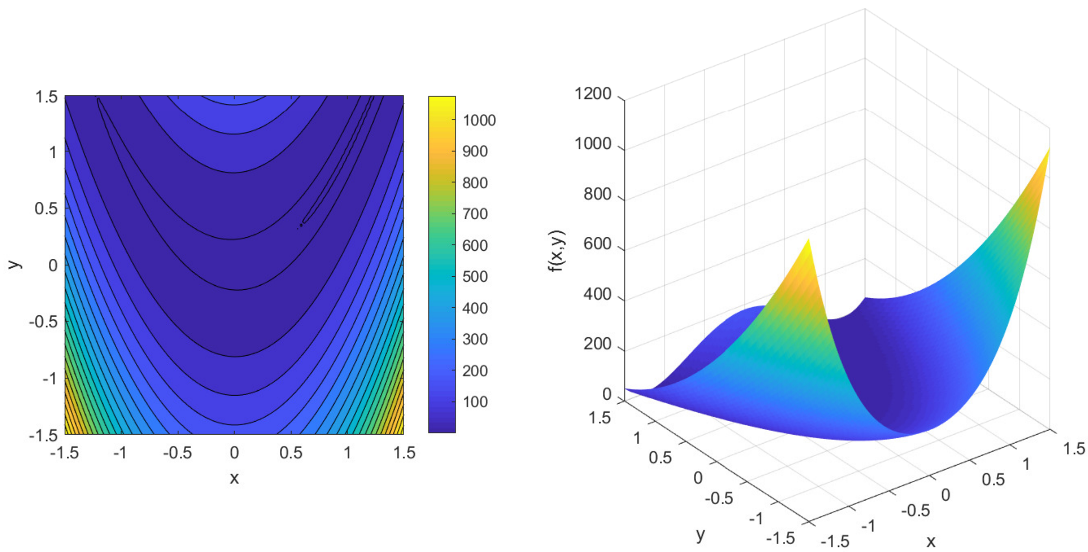
7.3 Minimisation Methods

Steepest Descents

Question 1



(a) Contour and surface plot for $f_4(x, y)$



(b) Contour and surface plot for $f_5(x, y)$

Figure 1: Contour and surface plots for $f_4,\ f_5$ where $-1.5 \leq x, y, \leq 1.5$

Let

$$f_4(x,y) = x + y + \frac{x^2}{4} - y^2 + \left(y^2 - \frac{x}{2}\right)^2$$
$$f_5(x,y) = (1-x)^2 + 80\left(y - x^2\right)^2$$
$$f_6(x,y,z) = 0.4x^2 + 0.2y^2 + z^2 + xz$$

At the minima (x^*, y^*) of f_4 we have

$$\frac{\partial f_4}{\partial x}(x^*, y^*) = 1 + x^* - {y^*}^2 = 0$$
$$\frac{\partial f_4}{\partial y}(x^*, y^*) = 1 - 2y^* - 2x^*y^* + 4{y^*}^3 = 0$$

Solving these simultaneous equations we get $(x^*, y^*) = \left(2^{-\frac{2}{3}} - 1, -2^{-\frac{1}{3}}\right) \approx (-0.3700, -0.7944,)$, and subbing into f_4 we find

$$f_4\left(2^{-\frac{2}{3}} - 1, -2^{-\frac{1}{3}}\right) = -\frac{3}{8}2^{\frac{2}{3}} - \frac{1}{2} \approx -1.0953$$

At the minima (x^*, y^*) of f_5 we have

$$\frac{\partial f_5}{\partial x}(x^*, y^*) = 2x^* - 2 + 320{x^*}^3 - 320x^*y^* = 0$$
$$\frac{\partial f_5}{\partial y}(x^*, y^*) = -160{x^*}^2 + 160y^* = 0$$

Solving these simultaneous equations we get $(x^*, y^*) = (1, 1)$, and subbing into f_5 we find

$$f_5(1,1) = 0$$

Question 2

Using the contour map for the function f_4, and the fact that near the minimum the f_4 is convex we can draw an approximately elliptical contour at the final value of f_4 we get via Steepest Descents, and the minima must lie inside this contour. The iterations of the Steepest Descents method, as well as the contour described above, can be seen in Figure 2.

From Figure 3 and the program output (shown in the Appendix A.1), we can see that the algorithm tends to the minimum value of f_4 from above and is approaching a value that is approximately equal to -1.1. We can be confident this is close to the actual minimum of the f_4 as the function is generally flat near the minimum so as long as the point (x, y) is close to the true minima then $f_4(x, y)$ is close to the minimum value of f_4. If we again look at the output of our function we find that after 10 iterations that function has not converged to more than 3 significant figures so we can only estimate to a max of 2 significant figures reliably. This is due to the fact that the function is quite flat near the minima, so the rate of convergence slows down as we reach the minima.

Question 3

We can see from Figure 4, that the path taken by the Steepest Descents method is slowly converging to the point $(x, y) = (1, 1)$. The rate of convergence is very slow due to the fact that the gradient vector will always point up the valley so for the algorithm to reach the minima it must slowly zigzag to the point of convergence. The iteration path is very sensitive to variations in λ^* as the gradient is always relatively large and is pointing out of the valley. Gradient Descent is inefficient for functions that have a small gradient everywhere (as each step will be small, thus leading to small changes in the function that is trying to be minimised), and those where the minimum lies in a steep valley, where the floor of the valley has a shallow slope (as small steps will be required to ensure the gradient stay in the valley and reaches the minima).

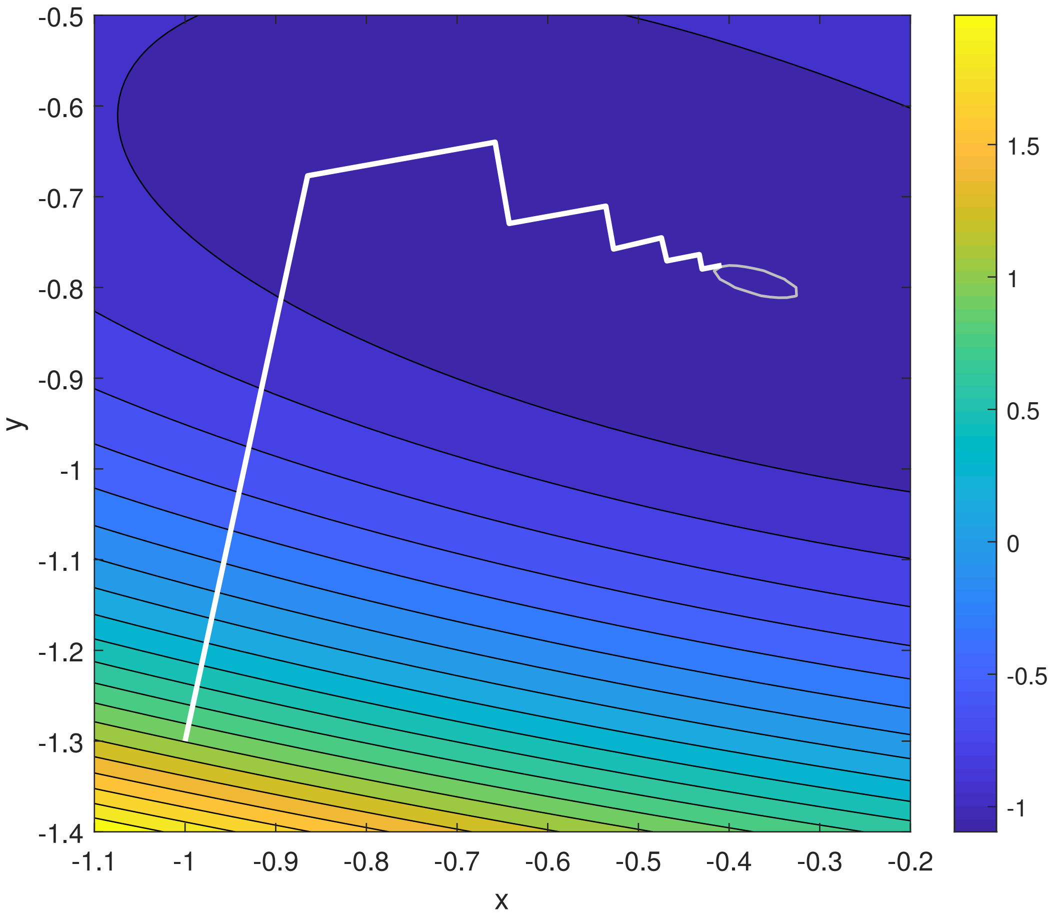


Figure 2: Contour plot showing 10 iterations of the Steepest Descents method for f_4 starting at $(-1.0, -1.3)$

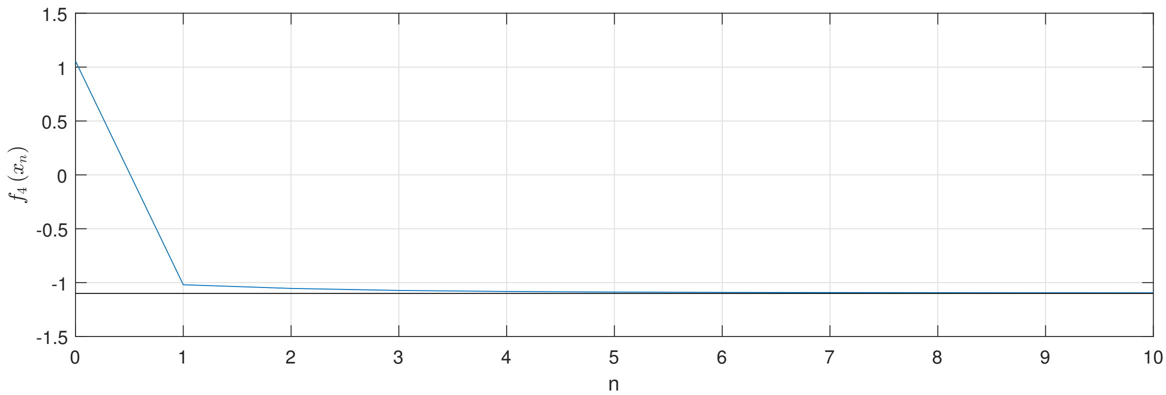


Figure 3: Plot showing the $f_4(x_n)$ against n for the method of Steepest Descents

Conjugate Gradients

Question 4

Using the contour map for the function f_4, and the fact that near the minimum the f_4 is convex we can draw an approximately elliptical contour at the final value of f_4 we get via Conjugate Gradients, and the minima must lie inside this contour. The iterations of the Conjugate Gradients algorithm, as well as the contour described above, can be seen in Figure 5. In this figure the gray ellipse cannot be seen indicating that the minima found is extremely close to the true minima.

From Figure 6 and the program output (shown in the Appendix B.1), we can see that the algorithm tends to the minimum value of f_4 from above and is approaching a value that is approximately equal

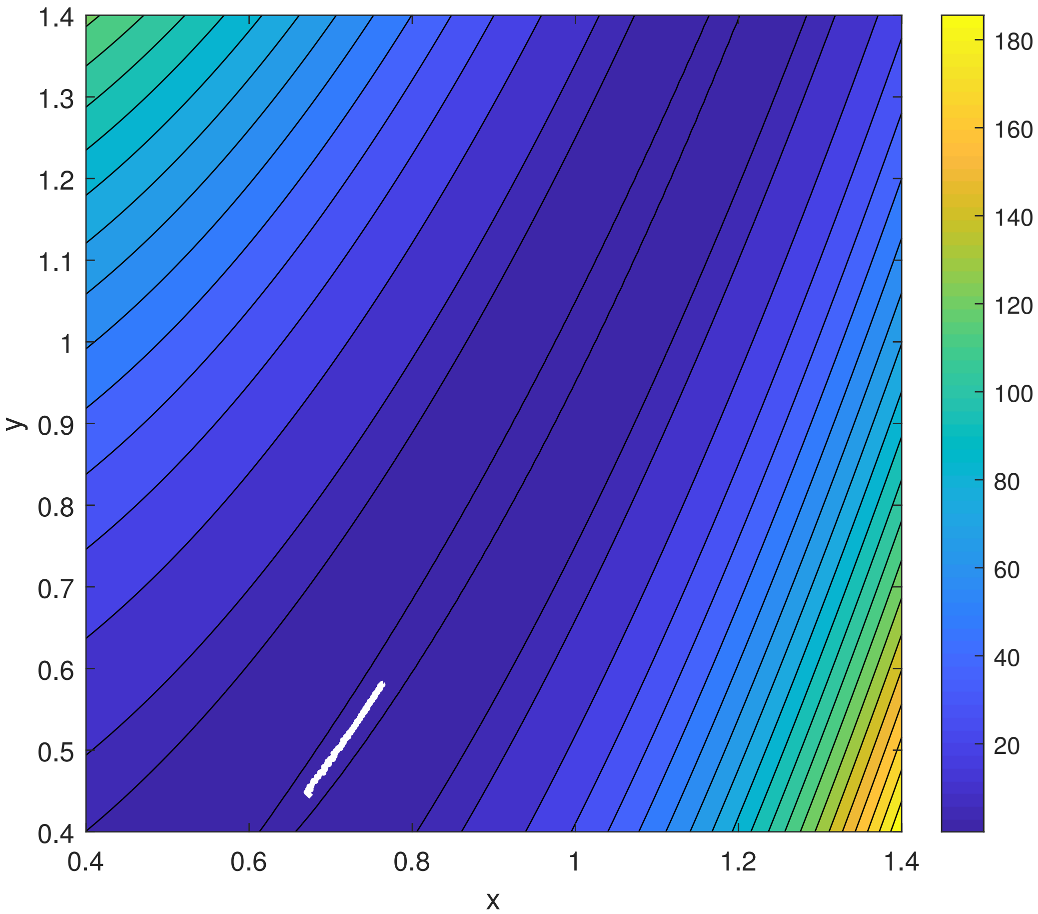


Figure 4: Contour plot showing 100 iterations of the Steepest Descents method for f_5 starting at $(0.676, 0.443)$

to -1.09527539. We can be confident this is close to the actual minimum of the f_4 as the function is generally flat near the minimum so as long as the point (x, y) is close to the true minima then $f_4(x, y)$ is close to the minimum value of f_4. If we again look at the output of our function we find that after 10 iterations that estimated minimum of f_4 has converged to 10, thus giving us a more accurate approximation of where the minima lies as compared to Steepest Descents.

Question 5

We can see in Figure 7 that the method of Conjugate Gradients is able to reach the minima of f_5 in a reasonable number of steps compared to the method of Steepest Descents which was not able to get close to the minima, even after 100 iteration. From analysing the convergence of the method of Steepest Descents and the method of Conjugate Gradients, we can see that Conjugate Gradients in general performs much better than Steepest Descents, but at the cost of extra calculations required per iteration (on average, as Conjugate Gradients uses Steepest Descents periodically), and thus extra time taken.

DFP Algorithm

Question 6

Looking at the output of the DFP algorithm shown in the Appendices C.1, C.2, C.3, we can see the method converges rapidly for the values of λ^* give and that if we round all of the inputs to 3 decimal places then the algorithm still converges towards zero but at a slower rate. We then find that if we round the inputs to 2 d.p., i.e

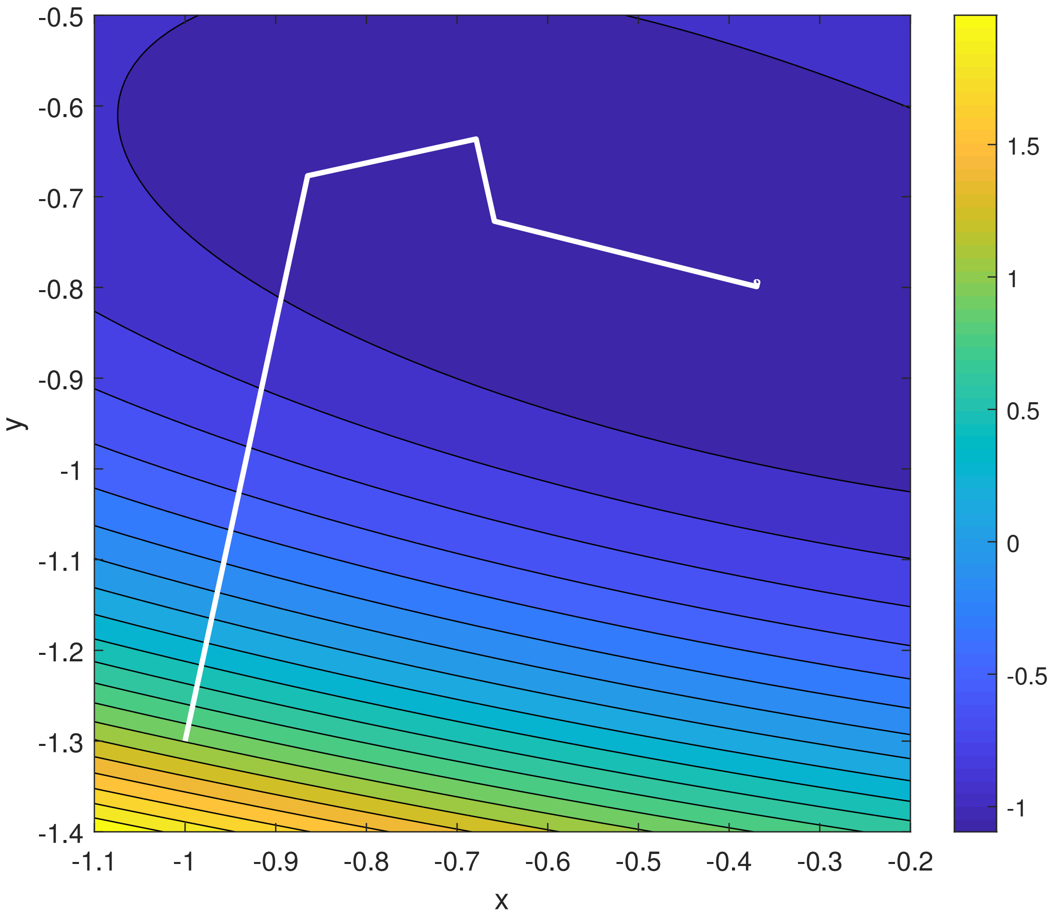


Figure 5: Contour plot showing 10 iterations of the Conjugate Gradients algorithm for f_4 starting at $(-1.0, -1.3)$

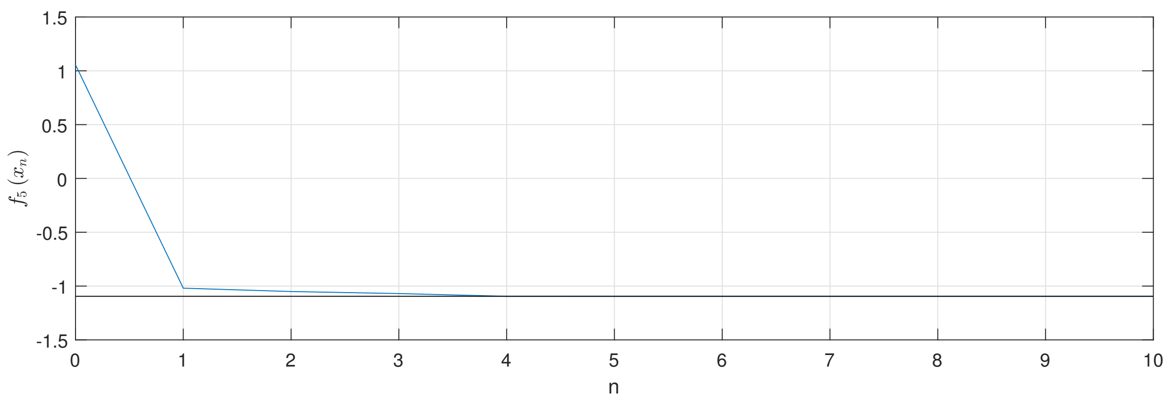


Figure 6: Plot showing the $f_4(x_n)$ against n for the Conjugate Gradients algorithm

$$\lambda^* = 0.39, 2.55, 4.22$$

Then we in fact start to move away from the minima at the third iteration so we can deduce that the optimality of this method is highly dependent on the accuracy to which one can find the minima of $\phi(\lambda)$

Note that the Hessian of f_6 at its minima is

$$\nabla^2 f_6 = \begin{pmatrix} 0.8 & 0 & 1 \\ 0 & 0.4 & 0 \\ 1 & 0 & 2 \end{pmatrix}$$

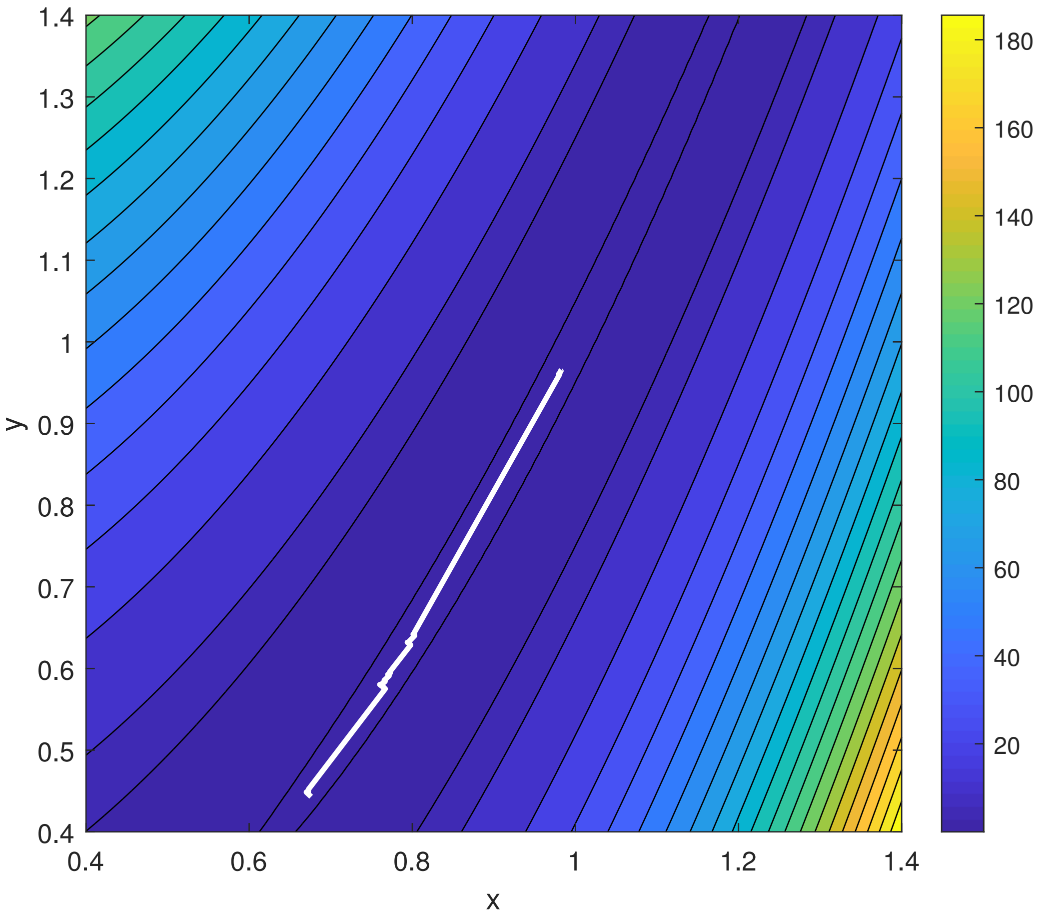


Figure 7: Contour plot showing 29 iterations of the Conjugate Gradients algorithm for f_5 starting at $(0.676, 0.443)$

As you can see from the outputs of the DFP algorithm when $\lambda^* = 0.3942, 2.5522, 4.2202$ (shown in the Appendix C.1), we can see that the final value for H is

$$\begin{aligned} H &= \begin{pmatrix} 3.333333316446757 & 0.000000189544994 & -1.666666787457326 \\ 0.000000189544994 & 2.499997844071311 & 0.000001369445411 \\ -1.666666787457326 & 0.000001369445411 & 1.333332462771204 \end{pmatrix} \\ &\approx \begin{pmatrix} \frac{10}{3} & 0 & -\frac{5}{3} \\ 0 & 2.5 & 0 \\ -\frac{5}{3} & 0 & \frac{4}{3} \end{pmatrix} \\ &= \nabla^2 f_6^{\ -1} \end{aligned}$$

verifying H tends to the inverse Hessian matrix.

Question 7

Using the contour map for the function f_4, and the fact that near the minimum the f_4 is convex we can draw an approximately elliptical contour at the final value of f_4 we get via DFP, and the minima must lie inside this contour. The iterations of the DFP algorithm, as well as the contour described above, can be seen in Figure 8. In this figure the gray ellipse cannot be seen indicating that the minima found is extremely close to the true minima.

From Figure 9 and the program output (shown in the Appendix C.4), we can see that the algorithm tends to the minimum value of f_4 from above and is approaching a value that is approximately equal to -1.095275394488075. We can be very confident this is close to the actual minimum of the f_4 as the value of $f_4(\mathbf{x}_n)$ did not change for several iterations (this value was found after 7 iterations).

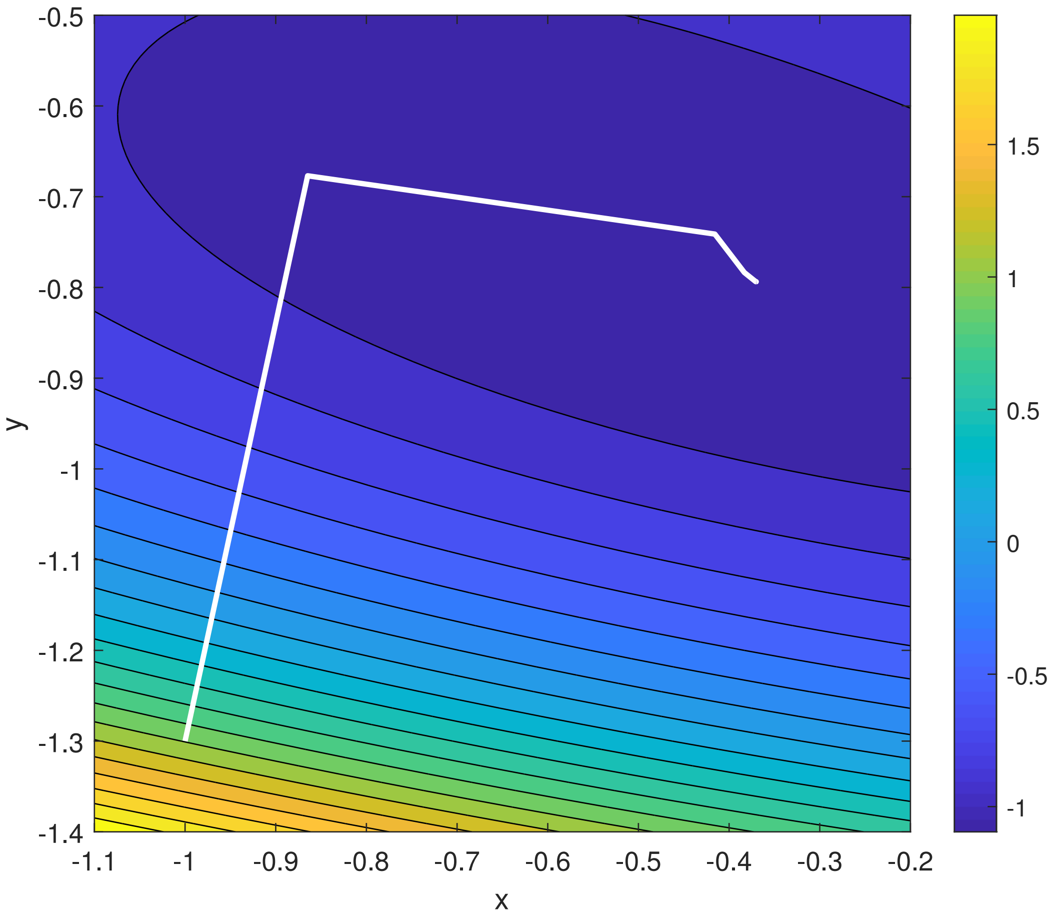


Figure 8: Contour plot showing 10 iterations of the DFP algorithm for f_4 starting at $(-1.0, -1.3)$

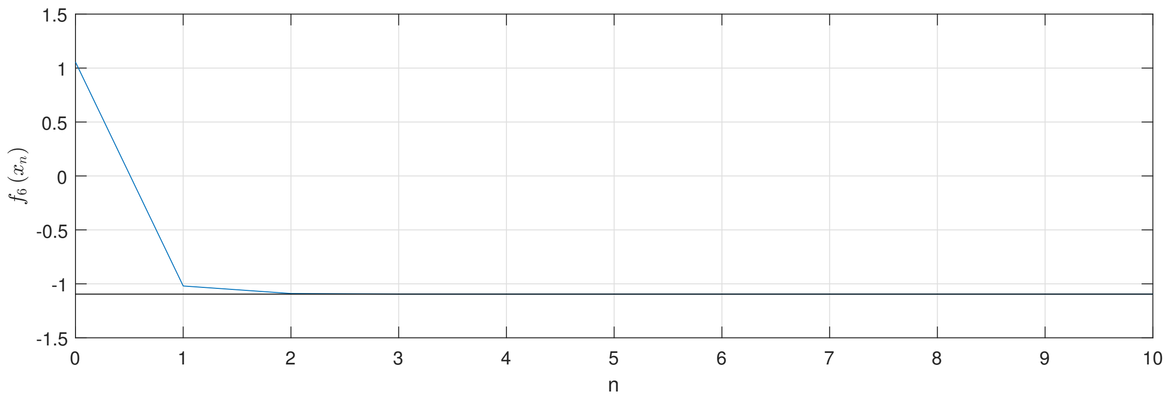


Figure 9: Plot showing the $f_4(x_n)$ against n for the DFP algorithm

Calculating the Hessian at the minima $(x^*, y^*) = \left(2^{-\frac{2}{3}} - 1, -2^{-\frac{1}{3}}\right) \approx (-0.3700, -0.7944,)$ we get

$$\nabla^2 f_4\left(2^{-\frac{2}{3}} - 1, -2^{-\frac{1}{3}}\right) = \begin{pmatrix} 1 & 2^{\frac{2}{3}} \\ 2^{\frac{2}{3}} & 5 \cdot 2^{\frac{1}{3}} \end{pmatrix}$$

So

$$\nabla^2 f_4\left(2^{-\frac{2}{3}} - 1, -2^{-\frac{1}{3}}\right)^{-1} = \frac{1}{3 \cdot 2^{\frac{1}{3}}} \begin{pmatrix} 5 \cdot 2^{\frac{1}{3}} & -2^{\frac{2}{3}} \\ -2^{\frac{2}{3}} & 1 \end{pmatrix}$$
$$\approx \begin{pmatrix} 1.6666 & -0.4200 \\ -0.4200 & 0.2646 \end{pmatrix} \approx H$$

This is true during iterations 7 and 8 (as seen in the Appendix C.4) but after this H seems to move away from the inverse Hessian, which may may have occurred due to rounding errors while working with $\mathbf{p}$ and $\mathbf{q}$ which will have a small magnitude as we get closer to the minima which can cause errors during division.

Question 8

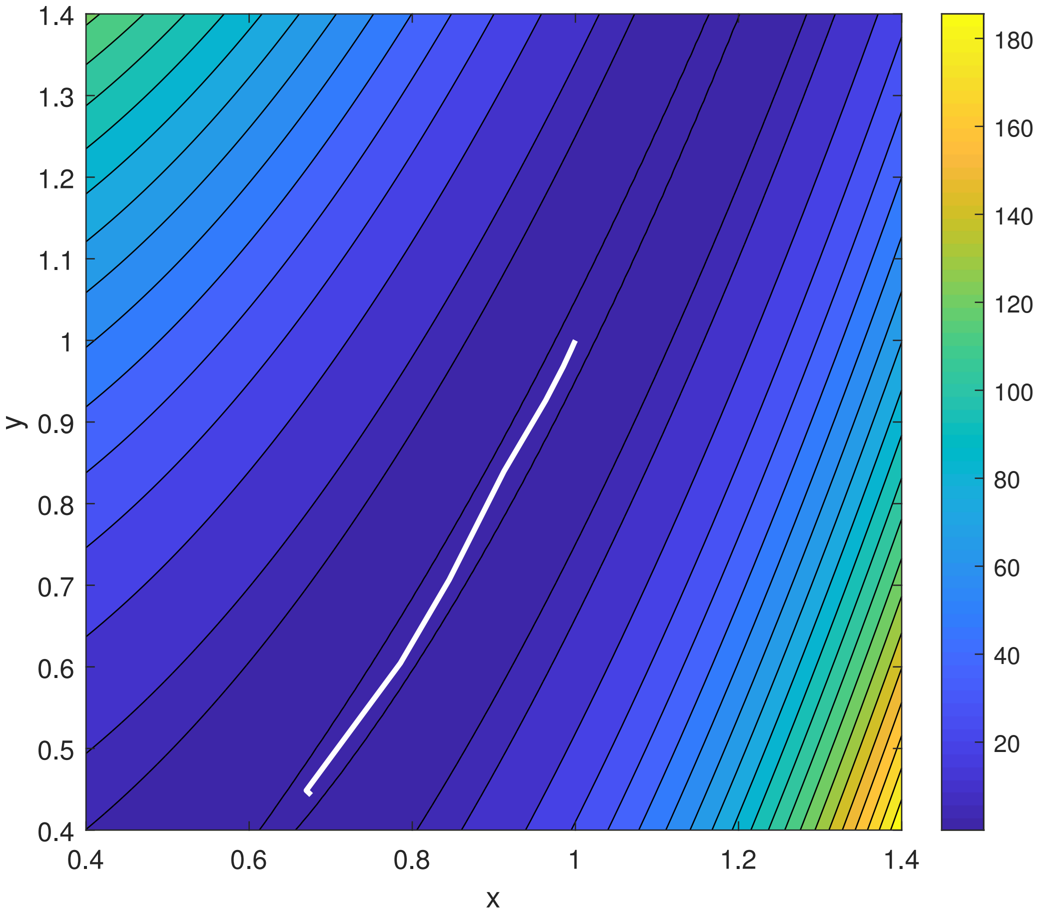


Figure 10: Contour plot showing 9 iterations of the DFP algorithm for f_5 starting at $(0.676, 0.443)$

We can see in Figure 10 that the DFP algorithm is able to converge quickly to the minima, $(1, 1)$, of f_5.

Calculating the Hessian at the minima $(x^*, y^*) = (1, 1)$ we get

$$\nabla^2 f_5(1, 1) = \begin{pmatrix} 642 & -320 \\ -320 & 160 \end{pmatrix}$$

So

$$\begin{aligned}\nabla^2 f_5(1,1)^{-1} &= \begin{pmatrix} \frac{1}{2} & 1 \\ 1 & \frac{321}{160} \end{pmatrix} \\ &\approx H\end{aligned}$$

And we can see that H does tend to the inverse Hessian (shown in the Appendix C.5).

Question 9

In general we see that DFP converges in fewer steps than Conjugate Gradients which in turn converges in fewer steps that Steepest Descents, however both Conjugate Gradients and DFP require extra calculations to take place to find the search direction which leads to each iteration taking longer.

Now consider the time complexity of each algorithm for a single iteration. Given the gradients can be calculated in $O(n)$, Steepest Descents and Conjugate Gradients are $O(n)$ as they only require a constant number of vector operations per iteration (i.e. scalar products and vector addition) whereas DFP is $O\left(n^2\right)$ due to the need to use matrix multiplication , where n is the number of inputs for our function.

Now consider the space complexity of each algorithm. For all of the algorithms, a given function, a way to calculate its gradient, and the value of x_i must all be store somewhere.

- For Steepest Descents, no extra data must be stored.
- DFP requires the storage of H an $n \times n$ matrix as well as the calculation of $\mathbf{p}$ and $\mathbf{q}$ both n dimensional column vectors so the algorithm in total needs $O\left(n^2\right)$ cells of extra storage.
- Conjugate Gradients requires the previous gradient, as well as the previous search direction, requiring $O(n)$ extra cells of storage being needed for Conjugate gradients.

This means Steepest Descents is the most space effective, followed by Conjugate Gradients, with DFP requiring the most space per iteration.

We should note that DFP makes the assumption that the function we are minimising is twice differentiable about the minima, if this is not true then DFP may fail to converge. Steepest Descents and Conjugate Gradients do not make this extra assumption as they simply assume the function is once differentiable about the minima. This means that there are situations where Steepest Descents and Conjugate Gradients may converge, but DFP will not.

Appendices

A Steepest Descents

A.1 Output for the method of Steepest Descents for minimising f_4

```
Iteration 0
f(x_0) =  1.0561

Iteration 1
lambda is 0.08
f(x_1) = -1.019763408140815
f(x_1) - f(x_0) = -2.075863408140815

Iteration 2
lambda is 0.64
f(x_2) = -1.053793240896909
f(x_2) - f(x_1) = -0.03402983275609417

Iteration 3
lambda is 0.23
f(x_3) = -1.072642172715154
f(x_3) - f(x_2) = -0.01884893181824476

Iteration 4
lambda is 0.61
f(x_4) = -1.082233600550578
f(x_4) - f(x_3) = -0.009591427835423882

Iteration 5
lambda is 0.21
f(x_5) = -1.087745445813394
f(x_5) - f(x_4) = -0.00551184526281645

Iteration 6
lambda is 0.52
f(x_6) = -1.090571187063349
f(x_6) - f(x_5) = -0.002825741249954605

Iteration 7
lambda is 0.2
f(x_7) = -1.092335980210098
f(x_7) - f(x_6) = -0.001764793146748822

Iteration 8
lambda is 0.5700000000000001
f(x_8) = -1.093492115253709
f(x_8) - f(x_7) = -0.001156135043611339

Iteration 9
lambda is 0.19
f(x_9) = -1.094196396245041
f(x_9) - f(x_8) = -0.0007042809913322401
```

```
Iteration 10
lambda is 0.5700000000000001
f(x_10) = -1.094620369419527
f(x_10) - f(x_9) = -0.0004239731744859476
```

B Conjugate Gradients

B.1 Output for the Conjugate Gradients algorithm for minimising f_4

```
Iteration 0
f(x_0) =  1.0561

Iteration 1
lambda is 0.08
f(x_1) = -1.019763408140815
f(x_1) - f(x_0) = -2.075863408140815

Iteration 2
lambda is 0.5700000000000001
f(x_2) = -1.050869389535873
f(x_2) - f(x_1) = -0.03110598139505782

Iteration 3
lambda is 0.24
f(x_3) = -1.069774656933543
f(x_3) - f(x_2) = -0.01890526739767062

Iteration 4
lambda is 1.39
f(x_4) = -1.095185648940472
f(x_4) - f(x_3) = -0.02541099200692831

Iteration 5
lambda is 0.15
f(x_5) = -1.095274551266079
f(x_5) - f(x_4) = -8.890232560765376e-05

Iteration 6
lambda is 0.43
f(x_6) = -1.095274833683071
f(x_6) - f(x_5) = -2.824169915704999e-07

Iteration 7
lambda is 0.2
f(x_7) = -1.095275014887243
f(x_7) - f(x_6) = -1.81204171889604e-07

Iteration 8
lambda is 1.32
f(x_8) = -1.09527539407085
f(x_8) - f(x_7) = -3.791836074018562e-07

Iteration 9
```

```
lambda is 0.15
f(x_9) = -1.095275394481108
f(x_9) - f(x_8) = -4.102576056652651e-10

Iteration 10
lambda is 1.1
f(x_10) = -1.095275394485695
f(x_10) - f(x_9) = -4.586775403936372e-12
```

C DFP Algorithm

C.1 Output for the DFP algorithm for minimising f_6 with $\lambda^* = 0.3942, 2.5522, 4.2202$

```
DFP(f6, [1 1 1], 'auto', false, 'printIteration', true, 'printH', true, 'maxLambda', 5);
Iteration 0
H is
     1     0     0
     0     1     0
     0     0     1

f(x_0) =  2.6

Iteration 1
Enter value for lambda (inf to exit): 0.3942
H is
   0.858352274194042   0.014072944031701  -0.258119970316233
   0.014072944031701   1.004768770136763   0.022660657086842
  -0.258119970316233   0.022660657086842   0.531080379880792

f(x_1) = 0.15595116992
f(x_1) - f(x_0) = -2.44404883008

Iteration 2
Enter value for lambda (inf to exit): 2.5522
H is
   0.940491198655168   0.381837100905275  -0.312416389405326
   0.381837100905275   2.439068453539567  -0.216104101774114
  -0.312416389405326  -0.216104101774114   0.566883349269534

f(x_2) = 0.002299764462646786
f(x_2) - f(x_1) = -0.1536514054573532

Iteration 3
Enter value for lambda (inf to exit): 4.2202
H is
   3.333333316446757   0.000000189544994  -1.666666787457326
   0.000000189544994   2.499997844071311   0.000001369445411
  -1.666666787457326   0.000001369445411   1.333332462771204

f(x_3) = 1.187842101580304e-09
f(x_3) - f(x_2) = -0.002299763274804684

Iteration 4
Enter value for lambda (inf to exit): inf
```

C.2 Output for the DFP algorithm for minimising f_6 with $\lambda^* = 0.394, 2.552, 4.220$

```
DFP(f6, [1 1 1], 'auto', false, 'printIteration', true, 'printH', true, 'maxLambda', 5);
Iteration 0
H is
     1     0     0
     0     1     0
     0     0     1

f(x_0) =  2.6

Iteration 1
Enter value for lambda (inf to exit): 0.394
H is
   0.858352274194042   0.014072944031701  -0.258119970316233
   0.014072944031701   1.004768770136763   0.022660657086842
  -0.258119970316233   0.022660657086842   0.531080379880792

f(x_1) = 0.155951808
f(x_1) - f(x_0) = -2.444048192

Iteration 2
Enter value for lambda (inf to exit): 2.552
H is
   0.940919560103759   0.381844461546537  -0.312183367054798
   0.381844461546537   2.439054657237847  -0.216187523449194
  -0.312183367054798  -0.216187523449194   0.566461134734899

f(x_2) = 0.00230130274367788
f(x_2) - f(x_1) = -0.1536505052563221

Iteration 3
Enter value for lambda (inf to exit): 4.220
H is
   3.324564137603769   0.000110000416139  -1.674235290951988
   0.000110000416138   2.499998556653221   0.000094959393648
  -1.674235290951988   0.000094959393648   1.326800907255224

f(x_3) = 1.533328035901804e-05
f(x_3) - f(x_2) = -0.002285969463318862

Iteration 4
Enter value for lambda (inf to exit): inf
```

C.3 Output for the DFP algorithm for minimising f_6 with $\lambda^* = 0.39, 2.55, 4.22$

```
DFP(f6, [1 1 1], 'auto', false, 'printIteration', true, 'printH', true, 'maxLambda', 5);
Iteration 0
H is
     1     0     0
     0     1     0
     0     0     1

f(x_0) =  2.6
```

```
Iteration 1
Enter value for lambda (inf to exit): 0.39
H is
   0.858352274194042   0.014072944031701  -0.258119970316233
   0.014072944031701   1.004768770136763   0.022660657086842
  -0.258119970316233   0.022660657086842   0.531080379880792

f(x_1) = 0.1562288
f(x_1) - f(x_0) = -2.4437712

Iteration 2
Enter value for lambda (inf to exit): 2.55
H is
   0.947195294118883   0.380312710163843  -0.309851698582830
   0.380312710163843   2.439338638540589  -0.215616906381142
  -0.309851698582830  -0.215616906381142   0.552875478310989

f(x_2) = 0.002966889204507186
f(x_2) - f(x_1) = -0.1532619107954928

Iteration 3
Enter value for lambda (inf to exit): 4.22
H is
   1.114926952632663   0.297200972706512  -0.782237765005345
   0.297200972706512   2.459749686453476  -0.118466284144752
  -0.782237765005345  -0.118466284144752   0.980730329670877

f(x_3) = 0.006353069946297814
f(x_3) - f(x_2) = 0.003386180741790628

Iteration 4
Enter value for lambda (inf to exit): inf
```

C.4 Output for the DFP algorithm for minimising f_4

```
Iteration 0
f(x_0) =  1.0561

Iteration 1
lambda is 0.08
H is
   0.973333954817419  -0.154625728680082
  -0.154625728680082   0.107941705224738

f(x_1) = -1.019763408140815
f(x_1) - f(x_0) = -2.075863408140815

Iteration 2
lambda is 1.47
H is
   1.492033781484901  -0.288012182776520
  -0.288012182776520   0.131615311060360
```

```
f(x_2) = -1.089684845505132
f(x_2) - f(x_1) = -0.06992143736431711

Iteration 3
lambda is 2
H is
   1.614948909031376  -0.432221520448885
  -0.432221520448885   0.288217132836756

f(x_3) = -1.095079094194513
f(x_3) - f(x_2) = -0.005394248689380765

Iteration 4
lambda is 0.92
H is
   1.612482851390598  -0.418718101882532
  -0.418718101882532   0.267834145823137

f(x_4) = -1.095275280000317
f(x_4) - f(x_3) = -0.000196185805803939

Iteration 5
lambda is 1.05
H is
   1.671504360430613  -0.424260081550387
  -0.424260081550387   0.268352206161323

f(x_5) = -1.095275394486938
f(x_5) - f(x_4) = -1.144866217384077e-07

Iteration 6
lambda is 1
H is
   1.667938277904089  -0.421949792770784
  -0.421949792770784   0.267638123951814

f(x_6) = -1.095275394488075
f(x_6) - f(x_5) = -1.13642428800631e-12

Iteration 7
lambda is 0.99
H is
   1.667537779985913  -0.420064154862705
  -0.420064154862704   0.264576239842747

f(x_7) = -1.095275394488075
f(x_7) - f(x_6) = 0

Iteration 8
lambda is 1
H is
   1.667254187643883  -0.420168628733312
  -0.420168628733312   0.264631780297620
```

```
f(x_8) = -1.095275394488075
f(x_8) - f(x_7) = 0

Iteration 9
lambda is 0.04
H is
   1.863454320569237  -0.546163131113046
  -0.546163131113046   0.277917293622589

f(x_9) = -1.095275394488075
f(x_9) - f(x_8) = 0

Iteration 10
lambda is 0.02
H is
   2.111596449654974  -0.698111361227302
  -0.698111361227302   0.318137484382235

f(x_10) = -1.095275394488075
f(x_10) - f(x_9) = 0
```

C.5 Output for the Conjugate Gradients algorithm for minimising f_5

```
Iteration 0
f(x_0) =  0.12060228608

Iteration 1
H is
   0.352850109954132   0.476370300268338
   0.476370300268338   0.649391422409122


Iteration 2
H is
   0.218447197469747   0.310874003422232
   0.310874003422232   0.446706558488486


Iteration 3
H is
   0.225978516544820   0.353795869111774
   0.353795869111773   0.565472840022352


Iteration 4
H is
   0.268546073052791   0.484256989526351
   0.484256989526350   0.878203642430776


Iteration 5
H is
   0.317039014503000   0.572875738069541
```

```
   0.572875738069556   1.038985165729992


Iteration 6
H is
   0.356206183028290   0.688588452612676
   0.688588452612676   1.333787183574219


Iteration 7
H is
   0.416338567396304   0.822227239385273
   0.822227239385259   1.630267356822878


Iteration 8
H is
   0.465082489675924   0.927568727644885
   0.927568727644858   1.856197844035878


Iteration 9
H is
   0.498262429235778   0.994909648939278
   0.994909648939276   1.992621564019540
```

Program Listingseeeee

Steepest Descents

Function that performs the method of Steepest Descents

```
function [x_list, f_list] = SD(func, x0, varargin) % max_iter, auto
% Use the Steepest Descent Algorithm

    p = inputParser;
    addRequired(p,'func',@(f) isa(f, 'symfun'));
    addRequired(p,'x0',@isnumeric);
    addParameter(p,'maxIter',100);
    addParameter(p,'auto',false);
    addParameter(p,'stationaryTolerance', 1e-4, @(x) isnumeric(x) && (x
        >=0));
    addParameter(p,'functionTolerance', 1e-4, @(x) isnumeric(x) && (x>=0)
        );
    addParameter(p,'precision', 1e-2, @(x) isnumeric(x) && (x>0));
    addParameter(p,'maxLambda', 2, @(x) isnumeric(x) && (x>0));
    addParameter(p,'printIteration', false, @islogical);
    parse(p, func, x0, varargin{:})

    auto = p.Results.auto;
    func = p.Results.func;
    func_tol = p.Results.functionTolerance;
    max_iter = p.Results.maxIter;
    stat_tol = p.Results.stationaryTolerance;
    x0 = p.Results.x0;
    prec = p.Results.precision;
    print_iter = p.Results.printIteration;
    maxl = p.Results.maxLambda;

    inputs = argnames(func);
    n = size(inputs);
    n = n(2);
    grad_sym = symfun.empty(n,0);
    grad_sym = grad_sym(1);
    for j = 1:n
        grad_sym(j) = diff(func, inputs(j));
    end
    g = symfun(grad_sym, inputs);

    x_list = zeros(max_iter, n);
    x_list(1,:) = x0;
    temp_x = num2cell(x0);
    f_list = zeros(1, max_iter);
    f_list(1) = double(func(temp_x{:}));

    if print_iter
        disp("Iteration 0")
        disp(join([" f(x_0) = " num2str(f_list(1),'%.16g')]))
    end

    iteration = 1;
```

```
while iteration <= max_iter
    last_x = x_list(iteration, :);
    temp_x = num2cell(last_x);
    s = -double(g(temp_x{:}));
    if auto && (norm(s) < stat_tol)
        break
    end
    if print_iter
        disp(' ');
        disp(join(['Iteration ',num2str(iteration,'%.16g')]));
    end
    % get lambda l
    l_list = 0:prec:maxl;
    ordinates = cell(n,1);
    for j = 1:1:n
        ordinates{j} = last_x(j) + l_list*s(j);
    end
    f_l_list = func(ordinates{:});
    if ~auto
        % manual - display graph of l
        clf('reset')
        figure(1)
        plot(l_list, f_l_list)
        xlabel('\lambda');
        ylabel('f(x+\lambda s)')
        grid on
        % manual - user input value of l
        l = input('Enter value for lambda (inf to exit): ');
        if l == inf
            break
        end
    else
        % auto - check exit conditions
        l = min_l(l_list, f_l_list);
        if print_iter
            disp(join(['lambda is ', num2str(l,'%.16g')]))
        end
    end

    new_x = last_x + l*s;
    temp_x = num2cell(new_x);
    new_f = double(func(temp_x{:}));
    last_f = f_list(iteration);
    if print_iter
        disp(join(['f(x_' num2str(iteration,'%.16g')...
            ') = ' num2str(new_f,'%.16g')]))
        disp(join(['f(x_' num2str(iteration,'%.16g') ') - f(x_'...
            num2str(iteration-1,'%.16g') ') = '...
            num2str(new_f-last_f,'%.16g')]))
    end

    x_list(iteration+1,:) = new_x;
    f_list(iteration+1) = new_f;
```

```
        if auto
            last_f = f_list(iteration);
            if abs(new_f-last_f) < func_tol
                break
            end
        end

        iteration = iteration + 1;
    end
    x_list = x_list(1:iteration,:);
    f_list = f_list(1:iteration);
end
```

Code that generates Figure 1

```
f = figure(1);
f.Position = [20, 1, 1000, 500];
f.Visible = 'off';
X = linspace(-1.5, 1.5, 100);
Y = linspace(-1.5, 1.5, 100);
[X,Y] = meshgrid(X,Y);
Z4 = double(f4(X,Y));
subplot(1,2,1);
contourf(X,Y,Z4,20);
daspect([1, 1, 1])
xlabel('x')
ylabel('y')
colorbar

subplot(1,2,2);
s = surf(X,Y,Z4);
s.EdgeColor = 'none';
xlim([-1.5 1.5])
ylim([-1.5 1.5])
xticks(-1.5:0.5:1.5)
yticks(-1.5:0.5:1.5)
daspect([1, 1, 3])
xlabel('x')
ylabel('y')
zlabel('f(x,y)')

print("contour1","-depsc");

f = figure(1);
f.Position = [20, 350, 1000, 500];
f.Visible = 'off';
Z5 = double(f5(X,Y));
subplot(1,2,1);
[~, h] = contourf(X,Y,Z5,20);
h.LevelList = [0 0.2 5 h.LevelList];
daspect([1, 1, 1])
xlabel('x')
ylabel('y')
colorbar
```

```
subplot(1,2,2);
s = surf(X,Y,Z5);
s.EdgeColor = 'none';
xlim([-1.5 1.5])
ylim([-1.5 1.5])
xticks(-1.5:0.5:1.5)
yticks(-1.5:0.5:1.5)
daspect([1, 1, 350])
xlabel('x')
ylabel('y')
zlabel('f(x,y)')
print("contour2","-depsc");
```

Code that generates Figure 2

```
f = figure(1);
f.Visible = 'off';
X = linspace(-1.1, -0.2, 100);
Y = linspace(-1.4, -0.5, 100);
[X,Y] = meshgrid(X,Y);
Z4 = double(f4(X,Y));
contourf(X,Y,Z4,20);
hold on
[~, h] = contour(X,Y,Z4,[fs1(end) fs1(end)]);
h.LineWidth = 1;
h.LineColor = [0.75, 0.75, 0.75];
hold off
line(xs1(:,1),xs1(:,2), 'color', 'white', 'lineWidth', 2)
xlim([-1.1, -0.2]);
ylim([-1.4, -0.5]);
daspect([1, 1, 1])
xlabel('x')
ylabel('y')
colorbar
print("contour3","-depsc");
```

Code that generates Figure 3

```
f = figure(1);
f.Visible = 'off';
plot(0:1:10,fs1)
grid on
daspect([1 1 1])
ylim([-1.5 1.5])
xlim([0 10])
yticks(-1.5:0.5:1.5)
line([-10 10], [-1.1 -1.1], 'color', 'black')
xlabel('n')
ylabel('$f_4\left(x_n\right)$', 'Interpreter', 'latex')
print("flimit1","-depsc");
```

Code that generates Figure 4

```
f = figure(1);
f.Visible = 'off';
X = linspace(0.4, 1.4, 100);
Y = linspace(0.4, 1.4, 100);
[X,Y] = meshgrid(X,Y);
Z5 = double(f5(X,Y));
[~, h] = contourf(X,Y,Z5,20);
h.LevelList = [0 0.2 5 h.LevelList(2:end)];
line(xs2(:,1),xs2(:,2), 'color', 'white', 'lineWidth', 2)
daspect([1, 1, 1])
xlabel('x')
ylabel('y')
colorbar
print("contour4","-depsc");
```

Conjugate Gradients

Function that performs the Conjugate Gradients Algorithm

```
function [x_list, f_list] = CG(func, x0, varargin) % max_iter, auto
% Use the Conjugate Gradients Algorithm

    p = inputParser;
    addRequired(p,'func',@(f) isa(f, 'symfun'));
    addRequired(p,'x0',@isnumeric);
    addParameter(p,'maxIter',100);
    addParameter(p,'auto',false);
    addParameter(p,'stationaryTolerance', 1e-4, @(x) isnumeric(x) && (x
        >=0));
    addParameter(p,'functionTolerance', 1e-4, @(x) isnumeric(x) && (x>=0)
        );
    addParameter(p,'precision', 1e-2, @(x) isnumeric(x) && (x>0));
    addParameter(p,'maxLambda', 2, @(x) isnumeric(x) && (x>0));
    addParameter(p,'printIteration', false, @islogical);
    parse(p, func, x0, varargin{:})

    auto = p.Results.auto;
    func = p.Results.func;
    func_tol = p.Results.functionTolerance;
    max_iter = p.Results.maxIter;
    stat_tol = p.Results.stationaryTolerance;
    x0 = p.Results.x0;
    prec = p.Results.precision;
    print_iter = p.Results.printIteration;
    maxl = p.Results.maxLambda;

    inputs = argnames(func);
    n = size(inputs);
    n = n(2);
    grad_sym = symfun.empty(n,0);
    grad_sym = grad_sym(1);
    for j = 1:n
        grad_sym(j) = diff(func, inputs(j));
    end
```

```
g = symfun(grad_sym, inputs)';

x_list = zeros(max_iter, n);
x_list(1,:) = x0;
temp_x = num2cell(x0);
f_list = zeros(1, max_iter);
f_list(1) = double(func(temp_x{:}));

if print_iter
    disp("Iteration 0")
    disp(join(["f(x_0) = " num2str(f_list(1),'%.16g')]))
end

g_list = zeros(n,n-1);
s_list = zeros(n,n-1);
iteration = 1;
while iteration <= max_iter
    last_x = x_list(iteration, :);
    temp_x = num2cell(last_x);

    % calculate search function
    grad = double(g(temp_x{:}));
    s = -grad;
    cycle = mod(iteration, n);
    if cycle > 1 % cycle ~= 1 or 0
        g_prev = g_list(:,cycle-1);
        s_prev = s_list(:,cycle-1);
        b = (grad'*grad)/(g_prev'*g_prev);
        s = -grad + b*s_prev;
    elseif cycle == 0
        g_prev = g_list(:,n-1);
        s_prev = s_list(:,n-1);
        b = (grad'*grad)/(g_prev'*g_prev);
        s = -grad + b*s_prev;
    end
    if cycle ~= 0
        g_list(:,cycle) = grad;
        s_list(:,cycle) = s;
    end

    if auto && (norm(s) < stat_tol)
        break
    end
    if print_iter
        disp(' ');
        disp(join(['Iteration ',num2str(iteration,'%.16g')]));
    end
    % get lambda l
    l_list = 0:prec:maxl;
    ordinates = cell(n,1);
    for j = 1:1:n
        ordinates{j} = last_x(j) + l_list*s(j);
    end
```

```
        f_l_list = func(ordinates{:});
        if ~auto
            % manual - display graph of l
            clf('reset')
            figure(1)
            plot(l_list, f_l_list)
            xlabel('\lambda');
            ylabel('f(x+\lambda s)')
            grid on
            % manual - user input value of l
            l = input('Enter value for lambda (inf to exit): ');
            if l == inf
                break
            end
        else
            % auto - check exit conditions
            l = min_l(l_list, f_l_list);
            if print_iter
                disp(join(['lambda is ', num2str(l,'%.16g')]))
            end
        end

        new_x = last_x + l*s';
        temp_x = num2cell(new_x);
        new_f = double(func(temp_x{:}));
        last_f = f_list(iteration);
        if print_iter
            disp(join(['f(x_' num2str(iteration,'%.16g')...
                ') = ' num2str(new_f,'%.16g')]))
            disp(join(['f(x_' num2str(iteration,'%.16g') ') - f(x_'...
                num2str(iteration-1,'%.16g') ') = '...
                num2str(new_f-last_f,'%.16g')]))
        end

        x_list(iteration+1,:) = new_x;
        f_list(iteration+1) = new_f;
        if auto
            last_f = f_list(iteration);
            if abs(new_f-last_f) < func_tol
                break
            end
        end

        iteration = iteration + 1;
    end
    x_list = x_list(1:iteration,:);
    f_list = f_list(1:iteration);
end
```

Code that generates Figure 5

```
f = figure(1);
f.Visible = 'off';
X = linspace(-1.1, -0.2, 100);
```

```
Y = linspace(-1.4, -0.5, 100);
[X,Y] = meshgrid(X,Y);
Z4 = double(f4(X,Y));
contourf(X,Y,Z4,20);
hold on
[~, h] = contour(X,Y,Z4,[fs3(end) fs3(end)]);
h.LineWidth = 1;
h.LineColor = [0.75, 0.75, 0.75];
hold off
line(xs3(:,1),xs3(:,2), 'color', 'white', 'lineWidth', 2)
xlim([-1.1, -0.2]);
ylim([-1.4, -0.5]);
daspect([1, 1, 1])
xlabel('x')
ylabel('y')
colorbar
print("contour5","-depsc");
```

Code that generates Figure 6

```
f = figure(1);
f.Visible = 'off';
plot(0:1:10,fs3)
grid on
daspect([1 1 1])
ylim([-1.5 1.5])
xlim([0 10])
yticks(-1.5:0.5:1.5)
line([-10 10], [-1.09527539 -1.09527539], 'color', 'black')
xlabel('n')
ylabel('$f_5\left(x_n\right)$', 'Interpreter', 'latex')
print("flimit2","-depsc");
```

Code that generates Figure 7

```
f = figure(1);
f.Visible = 'off';
X = linspace(0.4, 1.4, 100);
Y = linspace(0.4, 1.4, 100);
[X,Y] = meshgrid(X,Y);
Z5 = double(f5(X,Y));
[~, h] = contourf(X,Y,Z5,20);
h.LevelList = [0 0.2 5 h.LevelList(2:end)];
line(xs4(:,1),xs4(:,2), 'color', 'white', 'lineWidth', 2)
daspect([1, 1, 1])
xlabel('x')
ylabel('y')
colorbar
print("contour6","-depsc");
```

DFP

Function that performs the DFP Algorithm

```matlab
function [x_list, f_list] = DFP(func, x0, varargin) % max_iter, auto
% Use the DFP Algorithm

    p = inputParser;
    addRequired(p,'func',@(f) isa(f, 'symfun'));
    addRequired(p,'x0',@isnumeric);
    addParameter(p,'maxIter',100);
    addParameter(p,'auto',false);
    addParameter(p,'stationaryTolerance', 1e-4, @(x) isnumeric(x) && (x
        >=0));
    addParameter(p,'functionTolerance', 1e-4, @(x) isnumeric(x) && (x>=0)
        );
    addParameter(p,'precision', 1e-2, @(x) isnumeric(x) && (x>0));
    addParameter(p,'maxLambda', 2, @(x) isnumeric(x) && (x>0));
    addParameter(p,'printIteration', false, @islogical);
    addParameter(p,'printH', false, @islogical);
    parse(p, func, x0, varargin{:})

    auto = p.Results.auto;
    func = p.Results.func;
    func_tol = p.Results.functionTolerance;
    max_iter = p.Results.maxIter;
    stat_tol = p.Results.stationaryTolerance;
    x0 = p.Results.x0;
    prec = p.Results.precision;
    print_iter = p.Results.printIteration;
    print_H = p.Results.printH;
    maxl = p.Results.maxLambda;

    inputs = argnames(func);
    n = size(inputs);
    n = n(2);
    grad_sym = symfun.empty(n,0);
    grad_sym = grad_sym(1);
    for j = 1:n
        grad_sym(j) = diff(func, inputs(j));
    end
    g = symfun(grad_sym, inputs)';

    x_list = zeros(max_iter, n);
    x_list(1,:) = x0;
    temp_x = num2cell(x0);
    f_list = zeros(1, max_iter);
    f_list(1) = double(func(temp_x{:}));
    H = eye(n);

    if print_iter || print_H
        disp("Iteration 0")
    end
    if print_H
        disp('H is')
        disp(H)
    end
```

```
if print_iter
    disp(join([" f(x_0) = " num2str(f_list(1),'%.16g')]))
end


iteration = 1;
while iteration <= max_iter
    last_x = x_list(iteration, :);
    temp_last_x = num2cell(last_x);

    % calculate search function
    grad = double(g(temp_last_x{:}));
    s = -H*grad;

    if auto && (norm(s) < stat_tol)
        break
    end
    if print_iter || print_H
        disp(' ');
        disp(join(['Iteration ',num2str(iteration,'%.16g')]));
    end
    % get lambda l
    l_list = 0:prec:maxl;
    ordinates = cell(n,1);
    for j = 1:1:n
        ordinates{j} = last_x(j) + l_list*s(j);
    end
    f_l_list = func(ordinates{:});
    if ~auto
        % manual - display graph of l
        clf('reset')
        figure(1)
        plot(l_list, f_l_list)
        xlabel('\lambda');
        ylabel('f(x+\lambda s)')
        grid on
        % manual - user input value of l
        l = input('Enter value for lambda (inf to exit): ');
        if l == inf
            break
        end
    else
        % auto - check exit conditions
        l = min_l(l_list, f_l_list);
        if print_iter
            disp(join(['lambda is ', num2str(l,'%.16g')]))
        end
    end

    new_x = last_x + l*s';
    temp_new_x = num2cell(new_x);
    new_f = double(func(temp_new_x{:}));
```

```
        p = double(g(temp_new_x{:}))-double(g(temp_last_x{:}));
        q = l*s;


        H = H - (H*(p*p')*H)/(p'*H*p) + (q*q')/(p'*q);


        if print_H
            disp('H is')
            disp(H)
        end

        last_f = f_list(iteration);
        if print_iter
            disp(join(['f(x_' num2str(iteration,'%.16g')...
                ') = ' num2str(new_f,'%.16g')]))
            disp(join(['f(x_' num2str(iteration,'%.16g') ') - f(x_'...
                num2str(iteration-1,'%.16g') ') = '...
                num2str(new_f-last_f,'%.16g')]))
        end

        x_list(iteration+1,:) = new_x;
        f_list(iteration+1) = new_f;
        if auto
            last_f = f_list(iteration);
            if abs(new_f-last_f) < func_tol
                break
            end
        end

        iteration = iteration + 1;
    end
    x_list = x_list(1:iteration,:);
    f_list = f_list(1:iteration);
end
```

Code that generates Figure 8

```
f.Visible = 'off';
X = linspace(-1.1, -0.2, 100);
Y = linspace(-1.4, -0.5, 100);
[X,Y] = meshgrid(X,Y);
Z4 = double(f4(X,Y));
contourf(X,Y,Z4,20);
hold on
[~, h] = contour(X,Y,Z4,[fs5(end) fs5(end)]);
h.LineWidth = 1;
h.LineColor = [0.75, 0.75, 0.75];
hold off
line(xs5(:,1),xs5(:,2), 'color', 'white', 'lineWidth', 2)
xlim([-1.1, -0.2]);
ylim([-1.4, -0.5]);
daspect([1, 1, 1])
xlabel('x')
```

```
ylabel('y')
colorbar
print("contour7","-depsc");
```

Code that generates Figure 9

```
f = figure(1);
f.Visible = 'off';
plot(0:1:10, fs5)
grid on
daspect([1 1 1])
ylim([-1.5 1.5])
xlim([0 10])
yticks(-1.5:0.5:1.5)
line([-10 10], [-1.095275394488075 -1.095275394488075], 'color', 'black')
xlabel('n')
ylabel('$f_6\left(x_n\right)$', 'Interpreter', 'latex')
print("flimit3","-depsc");
```

Code that generates Figure 10

```
f = figure(1);
f.Visible = 'off';
X = linspace(0.4, 1.4, 100);
Y = linspace(0.4, 1.4, 100);
[X,Y] = meshgrid(X,Y);
Z5 = double(f5(X,Y));
[~, h] = contourf(X,Y,Z5,20);
h.LevelList = [0 0.2 5 h.LevelList(2:end)];
line(xs6(:,1),xs6(:,2), 'color', 'white', 'lineWidth', 2)
daspect([1, 1, 1])
xlabel('x')
ylabel('y')
colorbar
print("contour8","-depsc");
```

Other

Function to automatically find λ^*

```
function l = min_l(l_list, f_list)
    n = size(f_list);
    n = n(2);
    if f_list(1) <= f_list(2)
        l = l_list(2)/2; % assume l_list(1)=0, and don't want l=0 else
        % go nowhere but l_list(2) too large, so try half way point
    elseif f_list(n) <= f_list(n-1)
        l = l_list(n);
    else
        for j = 2:1:n-1
            if (f_list(j) <= f_list(j-1))&&(f_list(j) <= f_list(j+1))
                l = l_list(j);
                break
            end
```

```
            end
        end
    end
```