Quasi-Newton methods with application to penalty and barrier problems

August Borna 19981022-0070
Björn Elwin 19960406-2811
Daniel Larsson 19980523-2551

November 2022

1 Introduction

Minimizing functions can sometimes be a tedious procedure, which is why there is a need for numerical algorithms to find optimal points. In this two different algorithms were used, the DFP Quasi-Newton and the BFGS Quasi-Newton, where DFP is short for Davidon-Fletcher-Powell and BFGS short for Broyden-Fletcher-Goldfarb-Shanno. The difference between these methods is how we update a matrix D, where the matrix D is part of how a descent directions is computed.

2 Who has done what

All group members have participated in all parts of the project, however some division of labor was done to ensure time optimization. Daniel was more responsible for derivative approximation, Björn took care of the algorithm for updating the D-matrix, and August for the general Quasi-Newton. The report was also split up a bit, but all group members are aware and in agreement of everything written.

3 The optimal point(s) and the optimal function value for each problem.

Function $f(\mathbf{x})$	$\overline{\mathbf{x}}$	$f(\overline{\mathbf{x}})$
Rosenbrock, i.e. $100 * (x_2 - x_1^2)^2 + (1 - x_1)^2$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	0
$e^{x_1 x_2 x_3 x_4 x_5}$ subject to $\begin{cases} x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 = 10 \\ x_2 x_3 = 5 x_4 x_5 \\ x_1^3 + x_3^3 = -1 \end{cases}$	$\begin{bmatrix} -1.7172 \\ 1.8272 \\ 1.5957 \\ -0.7636 \\ -0.7636 \end{bmatrix}$	0.0539498

Table 1: Functions f(x), point $\overline{\mathbf{x}}$ minimizing the function and its corresponding functionvalue $f(\overline{\mathbf{x}})$.

4 A motivation for your choices of line search method(s) and stop criterion.

Armijo's Rule was chosen for line search since it finds a suitable lambda whilst avoiding the use of second derivatives. The only inputs needed are a function and a starting point. Another reason for the choice of line search was that no initial interval is needed when starting the line search, thus avoiding the issue of a bad initial interval. For the stopping criterion two conditions were used. The first one being when we are close to a stationary point and the second one when the relative distance moved in the last step is small. These were done by checking if the norm of the gradient is less than a tolerance and the norm of the distance between two points divided by the norm of the first point, respectively. The conditions can be seen below;

$$||\nabla f(x_k)|| \leq \epsilon \tag{1}$$

$$\frac{||x_{k+1} - x_k||}{||x_k||} \leq \epsilon \tag{2}$$

The first condition was chosen initially by its own but in some cases the second condition was needed since the distance moved was very small and the number of iterations could be decreased by adding it.

5 A short discussion on the consistency in the program behaviour and how you have tested it.

Consistency of the program was tested by choosing many different starting points for the different functions. The global minima is found for the Rosenbrock function for most starting points, but the number of iterations needed to find the minima differs depending on the starting point. With restart the minima is always found but without restart convergence is not achieved for some starting points. A local minima is always found when analysing the exponential function of table 3, here the number of iterations also differs on the starting point, when starting at some points, ex. the point given in the project description $x = [-2, 2, 2, -1, -1]$ the global minima shown in table 3 is found.

6 A table for each problem and a short discussion on how the convergence and the solution are affected by different initial points for a given tolerance.

In table 6 below the results for the Rosenbrock function can be seen. As can be seen in general using restart results in faster convergence. For the starting points when we do not achieve convergence one could see that the step size quickly becomes zero and we therefore get stuck without being at a minima. For the ones were we get convergence one could see that in general the BFGS method achieves faster convergence.

starting point	tolerance	algorithm	restart	convergence iterations
[200; 200]	1e-6	DFP	yes	34
[200; 200]	1e-6	BFGS	yes	26
[200; 200]	1e-6	DFP	no	did not converge
[200; 200]	1e-6	BFGS	no	70
[200; 10]	1e-6	DFP	yes	84
[200; 10]	1e-6	BFGS	yes	82
[200; 10]	1e-6	DFP	no	4116
[200; 10]	1e-6	BFGS	no	56
[1000; 1000]	1e-6	DFP	yes	908
[1000; 1000]	1e-6	BFGS	yes	904
[1000; 1000]	1e-6	DFP	no	did not converge
[1000; 1000]	1e-6	BFGS	no	did not converge

Table 2: Convergence results for the Rosenbrock function for different starting points and methods.

starting point	tolerance	algorithm	restart	iter	f(x)
$[-2;2;2;-1;-1]$	1e-6	DFP	yes	25	0.053
$[-2;2;2;-1;-1]$	1e-6	BFGS	yes	25	0.053
$[-2;2;2;-1;-1]$	1e-6	DFP	no	20	0.053
$[-2;2;2;-1;-1]$	1e-6	BFGS	no	15	0.053
$[1;1;1;1;1]$	1e-6	DFP	yes	50	0.053
$[1;1;1;1;1]$	1e-6	BFGS	yes	50	0.053
$[1;1;1;1;1]$	1e-6	DFP	no	35	0.479
$[1;1;1;1;1]$	1e-6	BFGS	no	35	0.053
$[-1;-1;-1;-1;-1]$	1e-6	DFP	yes	30	0.413
$[-1;-1;-1;-1;-1]$	1e-6	BFGS	yes	25	0.413
$[-1;-1;-1;-1;-1]$	1e-6	DFP	no	95	0.413
$[-1;-1;-1;-1;-1]$	1e-6	BFGS	no	30	0.413

Table 3: Convergence results for the second problem described in 3, for different starting points and methods.

7 A comparison of the two quasi-Newton methods with/without restart.

Using the restart parameter lets the program re-estimate the D matrix after every n (=number of dimension) iterations which helps the function converge faster,(in most cases). Without the restart parameter convergence may be very slow with short steps for each estimation, but can be fast when the hessian is quadratic(since a quadratic function theoretically should converge after n iterations using a Quasi-Newton method).

8 A strategy of choosing initial points and the sequence of penalty parameter values in problem (pen).

When choosing initial points, one must take into consideration local minima. If we choose a poor initial point, the algorithm might converge to a local minimum instead of a global one, and thus not finding an optimal point. This can be seen in table 6, where the algorithm fails to find a global minimum for some starting points or configurations. The penalty parameter is also a selection of value where one must tread carefully. If chosen too big, or too small, convergence might fail. This is why a more adaptive approach is taken. To start off, a small μ is used, and it is then increased as the algorithm works it's way towards the minimum. We found that one set of values that worked well was; $\mu = [10^0, 10^1, 10^2, 10^3]$. These values can possibly be refined to further increase performance, however,

they performed satisfactorily.

9 A printout of a minimization with all information (starting point, tolerance, method, etc.).

```
iteration  x        stepsize    f(x)    norm(grad)   ls iters
    lambda
    1      -1.938,     0.53971   5.559 63.55161684     13
    0.007707
            1.784
            1.599
           -0.800
           -0.800
    2      -1.679,     0.33155   1.356 13.49079541     13
    0.007707
            1.667
            1.541
           -0.687
           -0.687
    3      -1.681,     0.09082   1.240 12.21039211      7
    0.087791
            1.746
            1.543
           -0.655
           -0.655
    4      -1.500,     0.69135   0.447 8.85622787      1   1.000000
            2.335
            1.281
           -0.778
           -0.778
    5      -1.649,     0.47834   0.119 4.02478307      2   0.666667
            1.933
            1.492
           -0.759
           -0.759
    6      -1.638,     0.03102   0.075 2.60580534     13   0.007707
            1.943
            1.517
           -0.765
           -0.765
    7      -1.648,     0.01860   0.060 1.36676254     12   0.011561
            1.953
            1.524
           -0.773
           -0.773
    8      -1.647,     0.00310   0.060 1.31373179      9   0.039018
            1.951
            1.523
           -0.774
           -0.774
    9      -1.657,     0.01657   0.055 0.07526376      1   1.000000
            1.939
            1.525
           -0.771
```

```
          -0.771
  10      -1.707,     0.11845   0.054 0.20602142      5   5.062500
           1.850
           1.585
          -0.769
          -0.769
  11      -1.707,     0.00357   0.053 0.12287229     11   0.017342
           1.849
           1.583
          -0.767
          -0.767
  12      -1.706,     0.00031   0.053 0.08642855     14   0.005138
           1.848
           1.583
          -0.767
          -0.767
  13      -1.707,     0.00039   0.053 0.08359886      9   0.039018
           1.849
           1.584
          -0.767
          -0.767
  14      -1.710,     0.00671   0.053 0.04444516      2   1.500000
           1.844
           1.587
          -0.768
          -0.768
  15      -1.711,     0.00484   0.053 0.02652498      3   2.250000
           1.840
           1.589
          -0.767
          -0.767
  16      -1.711,     0.00031   0.053 0.01958879     12   0.011561
           1.840
           1.590
          -0.767
          -0.767
  17      -1.712,     0.00011   0.053 0.00474627     13   0.007707
           1.840
           1.590
          -0.767
          -0.767
  18      -1.712,     0.00026   0.053 0.00833185      7   0.087791
           1.840
           1.590
          -0.767
          -0.767
  19      -1.720,     0.01887   0.053 0.02552195      6   7.593750
           1.826
           1.599
          -0.766
          -0.766
  20      -1.717,     0.00595   0.053 0.00202424      2   1.500000
           1.830
           1.596
          -0.767
          -0.767
  21      -1.717,     0.00004   0.053 0.00204201     11   0.017342
```

```
            1.830
            1.596
           -0.767
           -0.767
   22      -1.717,     0.00001    0.053  0.00107597     13    0.007707
            1.830
            1.596
           -0.767
           -0.767
   23      -1.717,     0.00014    0.053  0.00257203      4    0.296296
            1.830
            1.596
           -0.767
           -0.767
   24      -1.718,     0.00210    0.053  0.00454123      5    5.062500
            1.828
            1.597
           -0.767
           -0.767
   25      -1.718,     0.00018    0.053  0.00058376      3    2.250000
            1.828
            1.597
           -0.767
           -0.767
>>
```

10 Appendix

Armijo's rule line search function

```
function [lambda, ls_iters ] = armijo(f, epsilon, alpha,deriv0)

    delta = sqrt(eps);

    %first = (f(delta/2) - f(-delta/2))/delta; %derivative at the
    point lambda = 0
    t = @(x) f(0) + epsilon * x * deriv0;

    lambda = 1;
    ls_iters = 1;

    while ~((f(lambda) <= t(lambda)) && (f(alpha * lambda) >= t(
    alpha * lambda)))
        if f(lambda) > t(lambda)
            lambda = lambda / alpha;
        elseif (f(alpha * lambda) < t(alpha * lambda))
            lambda = alpha * lambda;
        end
        ls_iters = ls_iters + 1;
    end

    if f(lambda)>f(0)
        lambda=0;
    end
    if isnan(f(lambda)) || f(lambda)>f(0)
```

```
        error('Bad job of the line search!')
    end
end
```

Nonlinearmin

```
function [x,no_its,normg] = nonlinearmin(f,x0,method,tol,restart,
    printout)
%clc
if printout
  fprintf('iteration \t\t x \t     stepsize \t f(x) \t    norm(grad
    ) \t\t  ls iters\tlambda \n');
end
dim=length(x0);

%ls_iters=0;
no_its = 1;
yk=x0;
x =x0;
normg = norm(grad(f,x0));
D=eye(dim);
ygranit = yk - 20000000*tol;
while(normg > tol && abs(f(yk)-f(ygranit))/abs(f(ygranit)) > tol*10
      )
    if restart
        D=eye(dim);
    end

    for i=1:dim
        gradientyk = grad(f,yk);
        d=-D*gradientyk;
        F=@(lambda) f(yk+lambda*d);
        deriv0 = gradientyk'*d;
        [lambda, ls_iters]=armijo(F,0.2,2,deriv0);
        %ls_iters=ls_iters+temp_iters;
        yk1=yk+lambda*d;


        stepsize = norm(yk1-yk);
        gradientyk1 = grad(f,yk1);
        D=updateD(D,lambda,d,method,gradientyk,gradientyk1);
        ygranit = yk;
        yk=yk1;
        normg = norm(gradientyk1);

        if printout
            fprintf('%5.0f  %12.3f,  %12.5f %12.8f %12.8f %6.0f
    %16.6f\n',no_its,yk1(1),stepsize,f(yk1),normg, ls_iters,lambda)
    ;
            for j=2:dim
                fprintf('%5.0f  %12.3f  %12.5f %12.8f %12.8f %6.0f
    %16.6f\n','',yk1(j),'','','', '','');
            end


        end
```

```
45
46      no_its = no_its +1;
47      end
48
49
50
51  end
52  x=yk1;
53
54  end
```

Update D

```
1   function D1 = updateD(D,lambda,d,method,gradyk,gradyk1)
2
3   p = lambda * d;
4   q = gradyk1 - gradyk;
5
6   if method == "DFP"
7       D1 = D + (p*p')/(p'*q) - D*q*q'*D/(q'*D*q);
8   end
9
10  if method == "BFGS"
11      D1 = D + (1+q'*D*q/(p'*q))*(p*p')/(p'*q) - (p*q'*D + D*q*p')/(p
        '*q);
12  end
13  end
```