Numerical Optimisation: Assignment 6

Student Number:13064947

February 2018

1 Exercise 1

Submitted via Cody Coursework

2 Exercise 2

You are given an adaptation of the trust region SR-1 function in trustRegionLS.m and the 2D sub- space in solverCM2dSubspaceExtLS.m. Point out and explain the relevant modifications in the solver.

2.1 trustRegionLS.m

Although we are asked to point out the differences in the solver for the 2d subspace method, I would also point to a change illustrated by the snippet of code below. The original code provided in the prior exercises was not really suitable for a larger scale implementation as the Hessian was constructed explicitly. The code below shows a more scalable implementation where we now evaluate the Hessian by its approximation at some point multiplied by a vector.

```
% SR1 quasi Newton: Hessian update
% Efficient large scale implementation:
% - implement Hessian as their action on a vector
%     F.d2f(x, q) ... evaluates the Hessian (approximation) at
%     'x'and multiplies it with vector 'q'.
% - use iterative solvers
if sr1
% Residual vector of the secant equation.
% B_k_1 * p = F.d2f(x_k_1, p).
rSec = y_k - F.d2f(x_k_1, p); % recall notation s_k = p_k,

% Update Hessian approximation if condition holds, otherwise skip
    update.
if (abs(rSec'*p) >= r*norm(p)*norm(rSec))
  % B_k = B_k_1 + (rSec*rSec')/(rSec'*p);
  F.d2f = @(x, q) F.d2f(x_k_1, q) + (rSec'*q)/(rSec'*p)*rSec;
  % this update is only valid at x_k, F.d2f = F.d2f(x_k, q)
  %multiplies q with a matrix.
end
```

2.2 solverCM2dSubspaceExtLS.m

If we recall the original implementation of the 2d-subspace method, we were given a positive definite matrix B. Thus the original problem swept out a two dimensional region between the Cauchy point (the direction of steepest descent) and the Newton point (where we take into account the curvature). If the Newton point was inside the trust region then we were done. If not then we would hit the boundary in our two dimensional subspace, ensuring we did at least as well as the Cauchy point or the dogleg method. The only other case we needed to check was collinearity of the Cauchy and Newton point.

The original problem was given by:

$$\min_p m(p) = f + g^T p + \frac{1}{2} p^T B p \text{ s.t. } ||p|| \leq \Delta, p \in \text{span}[g, B^{-1}g]$$

The implementation given in this assignment is an extension in the sense that we are now dealing with the case where B may be indefinite. Thus the 2 dimensional subspace is changed to:

$$\text{span}[g, (B + \alpha I)^{-1} g], \text{ with } \alpha \in (-\lambda_1, -2\lambda_1)$$

Here λ_1 is the most negative eigenvalue of $\mathbf{B}$, and where α makes $B + \alpha I$ positive definite.

In the code given we have three potential cases. First negative eigenvalues, second zero eigenvalues and three positive eigenvalues.

2.2.1 Positive eigenvalues

The easiest is of course where we have positive eigenvalues, which should be identical to the problem we originally solved. The implementation given is philosophically the same, but after checking the code, the key difference is the new code is more scalable because it uses an iterative solver (in mcg.m), to approximate B \ g. As per the original implementation the question is whether the full Newton step is inside the trust region or not, otherwise it is a standard 2d subspace problem.

2.2.2 Zero eigenvalues

In the code given, when there are no negative eigenvalues but there are zero eigenvalues the step taken is the Cauchy point step.

2.2.3 Negative eigenvalues

In the case where we have negative eigenvalues, the code given implements the methodology given by (4.18) and (4.19) of Nocedal and the modified problem above is solved. That is we modify $\mathbf{B}$ with α, such that $B + \alpha I$ is positive semi-definite. If $||B + \alpha I|| \leq \Delta$, that is inside the trust region. Then we don't need to do the subspace search and take the step $p = -(B + \alpha I)^{-1}g + v$, with v being a vector such that $v^T(B + \alpha I)^{-1}g \leq 0$. The relevant code snippet implementing this is given below. (Note again the use of an iterative solver mcg() for scalability).

```
alpha = -1.5*lambdaB1; % shift ensuring that B + alpha*I is p.d.

B = @(q) B(q) + alpha*q; % modify to spd matrix (B + alpha I)
% unconstraint solution with modified Hessian
[pNewt, flag, relres, iter, resvec, pNewts] = mcg(B, g, tol, maxit,
    @(x) x);
% iterative solve to approximate B\g;

if norm(pNewt) <= Delta
npNewt = pNewt/norm(pNewt);
v = randn(size(x_k));
v = v/norm(v);
v = -0.1*npNewt + 0.1*(v - npNewt*npNewt'*v); % v: v'*pNewt <= 0
p = -pNewt + v;  % ensure ||p|| >= ||pNewt||
```

The final case is where the modified Newton point is not in the trust region. In this case we cannot use the unconstrained step and continue as in the original 2 dimensional subspace methodology, albeit using the span between the Cauchy point and the modified Newton point.

In the last part of the code given, there is a further projection ensuring that the optimisation is constrained to the correct subspace.

3 Exercise 3

Compute and plot a minimal area surface function boundary conditions:

- Bottom edge: $B_b(t) = sin(\pi t), t \in [0, 1]$.
- Left edge: $B_L(t) = sin(\pi t + \pi), t \in [0, 1]$.
- Top edge: $B_T(t) = sin(3\pi t), t \in [0, 1]$.
- Right edge: $B_R(t) = sin(3\pi + \pi), t \in [0, 1]$.

Discretization with q = 14 yields a problem of reasonable size. Provide any parameters and explanations that you consider relevant in the minimisation.

Using the discretisation given we have grid points $x_{(i-1)(q+1)+1}, \ldots, x_{i(q+1)}$ for $i = 1, 2, \ldots, q + 1$. The surface area is given by

$$f_j(\mathbf{x}) = \int\int_{(x,y)\in A_j} \sqrt{1 + \left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2}\, dxdy$$

Again using the equations and discretisation given, using finite differences we have the following surface function to minimise:

$$f_j(\mathbf{x}) = \frac{1}{q^2}\left[1 + \frac{q^2}{2}[(z(x_j) - z(x_{j+q+2}))^2 + (z(x_{j+1}) - z(x_{j+q+1}))^2]\right]^{\frac{1}{2}}$$

3.1 Using the LS-Newton-CG algorithm

For the LS-Newton-CG and BFGS algorithms I chose the following parameters:

- Delta = 1e-02 (for computing the numerical gradient).
- Maximum iterations 200
- Tolerance 1e-04 (I increased the tolerance after some initial problems with the line search, of course I could run for more iterations to achieve whatever tolerance I wish).
- Alpha0 = 1 (for line search)
- Line Search optimisation Parameters c1 = 1e-04, c2=0.5

The LS-Newton-CG code was identical to that provided in the Cody coursework submission. The basic intuition is to find a search direction by applying conjugate gradients to the newton equations (where we have a linear system deriving an approximation to the newton step). Because conjugate gradients require a positive definite system and this may not be the case here then using this method we end the CG iterations the moment we hit negative curvature. Thus, the method is also known as the truncated Newton method. By truncating, we ensure the step is in a direction of descent, but we also get the additional speedy convergence of the Newton method.

With the parameters given above, I achieved convergence in only ten steps. The figure showing the surface plot is given in fig 1, part (b).

3.2 Using the BFGS algorithm (provided)

The BFGS Quasi-Newton algorithm, its implementation and convergence was described in detail in a previous coursework. With the same parameters as the LS-Newton-CG algorithm, the implementation given required 71 iterations, with the figure showing the surface plot given in fig 1, part (a). We can see from the plots that we have essentially converged to the same surface plot (albeit with more iterations for BFGS). I also checked this was the case by comparing the norms of the differences of the final iterations.

3.3 Using the Trust Region-SR1 algorithm

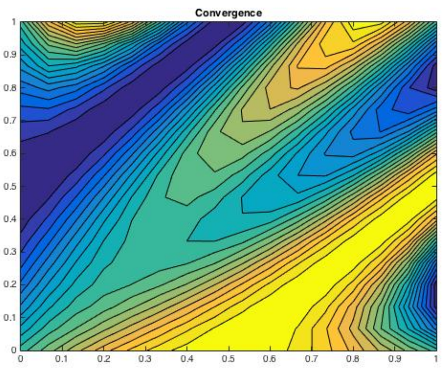
For the Trust Region-SR1 algorithm, I used the following parameters:

- Maximum iterations 100.
- Tolerance 1e-04.
- Eta = 0.1. (Step acceptance, relative progress threshold).
- Delta = 1 (Trust region radius).

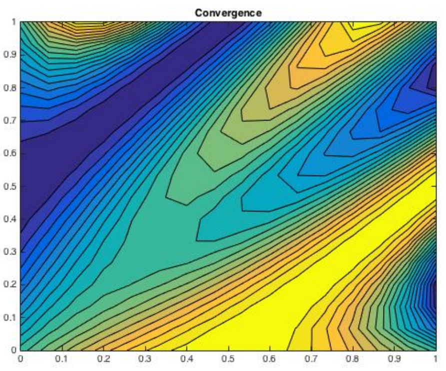
The original trust region algorithm was implemented and discussed in a previous coursework, so I won't repeat here. The key differences in the code provided in this assignment being discussed in question 2.

With the hyper-parameters given above, the algorithm converged in 75 iterations. The surface plot is shown in fig 2, part a). Clearly the surface plot is very similar to the solutions from the other algorithms. The trust region method took the most iterations of the three algorithms to converge, with a similar number of iterations as BFGS, but with the LS-Newton-cg converging far faster than the other two.

For completeness I have also included a final plot of the boundary conditions given for this question in fig 2, part b).

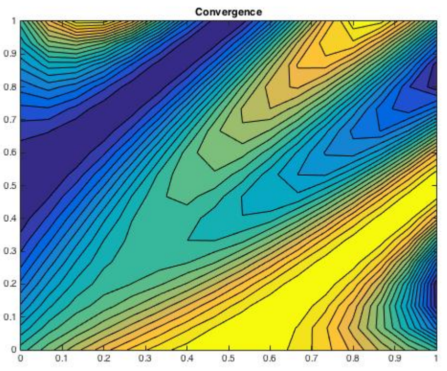


(a) BFGS Surface plot

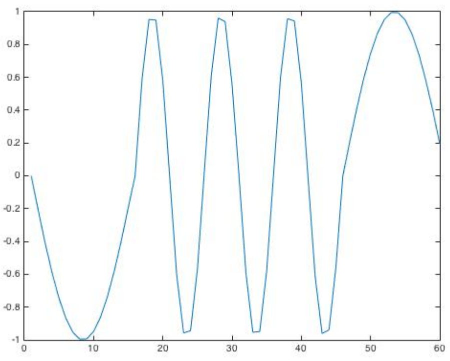


(b) Newton-cg Surface Plot

Figure 1: BFGS versus Newton Conjugate Gradient



(a) Trust Region - surface plot



(b) Boundary conditions

Figure 2: Trust Region and Boundary conditions