MATH 467: Project 1

Asif Zubair

November 7, 2016

Function

The function, gradient and hessian are provided below:

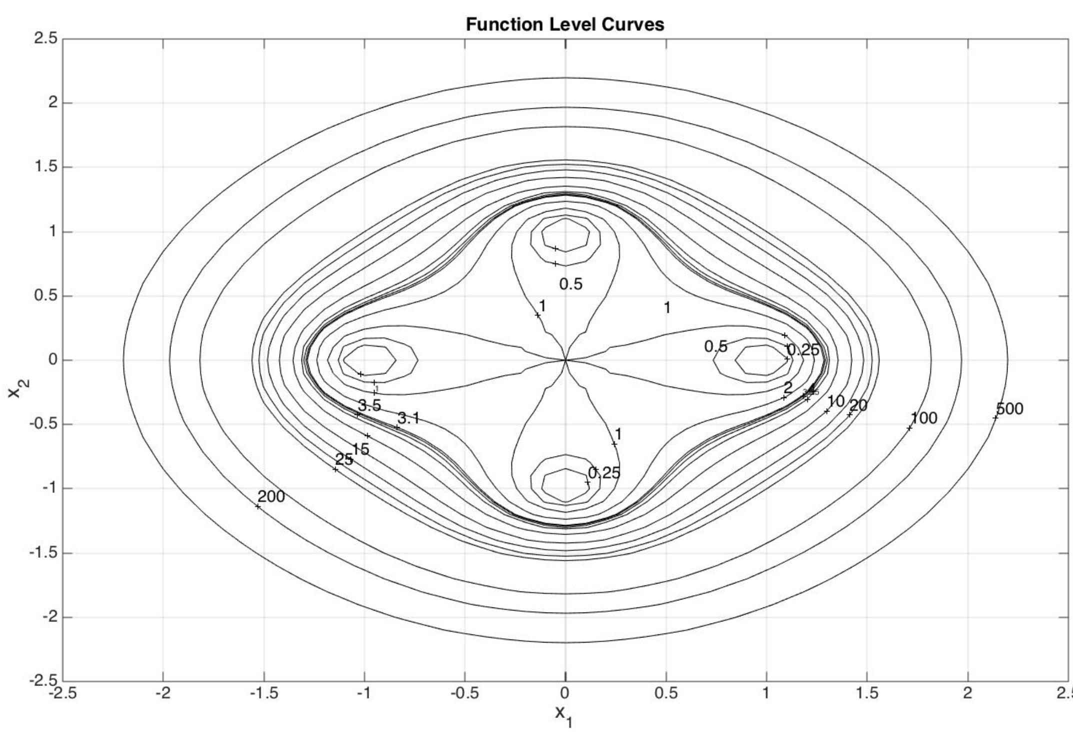
$$f([x,y]) = (x^4 + y^4 - 6x^2y^2 - 1)^2 + (4x^3y - 4xy^3)^2$$

$$\triangledown f([x,y]) = \begin{bmatrix} 8x^7 - 8x^3 + 24x^3y^4 + 24x^5y^2 + 8xy^6 + 24xy^2 \\ 8y^7 - 8y^3 + 24x^4y^3 + 8x^6y + 24x^2y^5 + 24x^2y \end{bmatrix}$$

$$F([x,y]) = \begin{bmatrix} 56x^6 - 24x^2 + 72x^2y^4 + 120x^4y^2 + 8y^6 + 24y^2 & 48xy(1 + (x^2 + y^2)^2) \\ 48xy(1 + (x^2 + y^2)^2) & 56y^6 - 24y^2 + 72x^4y^2 + 120x^2y^4 + 8x^6 + 24x^2 \end{bmatrix}$$

Analysis of function minimia

We sketch the contour plot of the function below:



From the contour plot, we can say the function has four minimizers at $(1, 0), (-1, 0), (0, 1), (0, -1)$ and a local maxima at $(0, 0)$. The gradient evaluated at the points $(1, 0), (-1, 0), (0, 1), (0, -1)$ is $\mathbf{0}$. The Hessian computes at these points is $\begin{bmatrix} 32 & 0 \\ 0 & 32 \end{bmatrix}$ which is a positive definite matrix. This implies that these points are minimizers of the function. We can also see that the value of the function at these points is $\mathbf{0}$. Since the function is non-negative, this implies that these points are also global minima of the function. We also see that the gradient is zero at $(0, 0)$. However, it is easy to see from the contour plot, that $(1, 1)$ is not a local minima of the function (It is in fact a local maxima).

Code design

Three methods were implemented in MATLAB:
- fixed-step
- newton's method with backtracking
- conjugate gradient method with Fletcher-Reeves formula

The actual implementation are attached at the end of the report. Here we briefly discuss the code design. The function, gradient and hessian were implemented separately as functions. The optimization algorithms were written such that they can accept a function (and its gradient and hessian) as an argument. The iterative algorithms terminate if the either the norm of the gradient ($norm(\bigtriangledown f(x_k))$ is below a tolerance value (tol) or the number of iterations have exceeded a pre-defined limit (MAX_TOL).

The algorithms also return a return a flag indicating if the algorithm has converged or not. In addition, to checking tolearance and number of iterations as indicated above, we also say that an algorithm has not converged to a local minima if $norm(x_k) < tol$. In this, case the algorithm has converged to a local maxima at $\mathbf{0}$.

Validation on quadratic function

We first provide validation of the implementation for a quadratic function define as follows:

$$f(x) = \frac{1}{2}x' \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} x + \begin{bmatrix} 1 & \frac{1}{2} \end{bmatrix} x + 3$$

It easy to see that this function has a minimizer at $\begin{bmatrix} -1 \\ -\frac{1}{4} \end{bmatrix}$.

Below, we provide step-wise computations for the different methods as implemented on the quadratic function. Here we used the initial point x_o as $\begin{bmatrix} -1 \\ 0 \end{bmatrix}$ for all the methods. A stepsize of 0.5 was chosen for the fixed-step method. Convergence was tested by comparing the norm of the gradient with a tolerance of $1e-5$.

k	x_k	y_k	α_k	d_1	d_2	$f(x_k)$
1	-1.000	-0.000	0.500	-0.000	-0.500	2.500
2	-1.000	-0.250	0.500	-0.000	-0.000	2.438
3	-1.000	-0.250	0.500	-0.000	-0.000	2.438

Table 1: Fixed Step

k	x_k	y_k	α_k	d_1	d_2	$f(x_k)$
1	-1.000	-0.000	0.2500	-0.000	-0.500	2.500
2	-1.000	-0.250	0.000	-0.000	-0.000	2.438

Table 2: Newton's method with backtracking

k	x_k	y_k	α_k	d_1	d_2	$f(x_k)$
1	-1.000	-0.000	0.2500	-0.000	-0.500	2.500
2	-1.000	-0.250	0.000	-0.000	-0.000	2.438

Table 3: Conjugate Gradient

We see that all three methods converge to the minimizer. The fixed step method takes three steps, the netwon method and conjugate gradient method take two steps to find the minimizer.

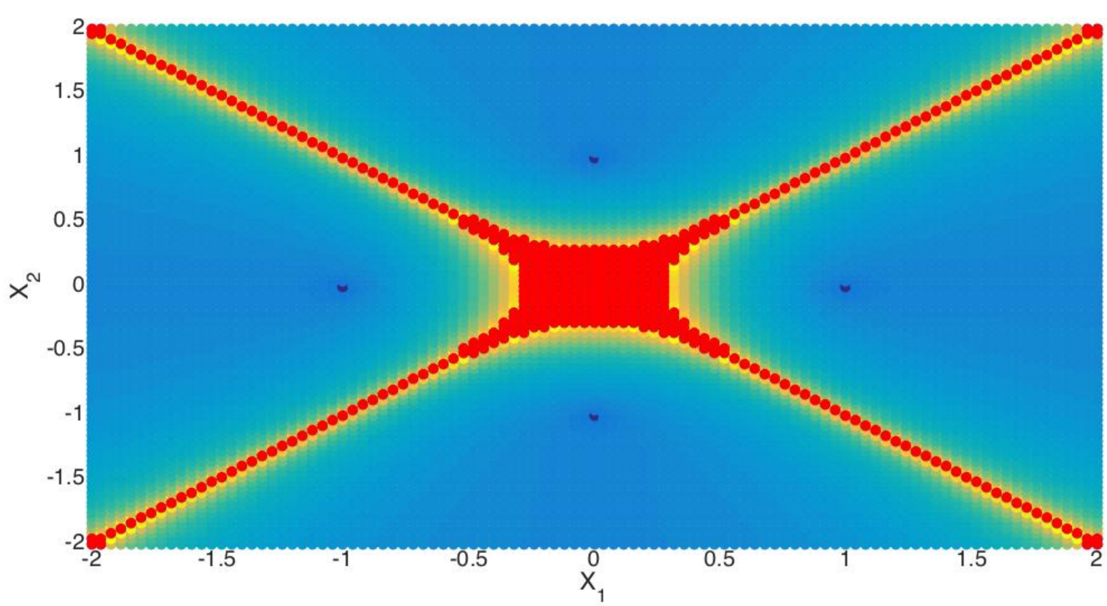
Performance of algorithms

For all the algorithms we used a tolerance of $1e-7$ and a maximum number of iterations of 10000. Algorithms were considered to have converged if the limit point of the iterative algorithms was one of the global minima - $(1,0),(-1,0),(0,1),(0,-1)$. Algorithms were deemed to have not converged if they didn't meet the convergence criterion $(norm(\bigtriangledown f(x_k)) < tol)$ within the maximum number of iterations or if they converged to the local maxima at $(0,0)$.

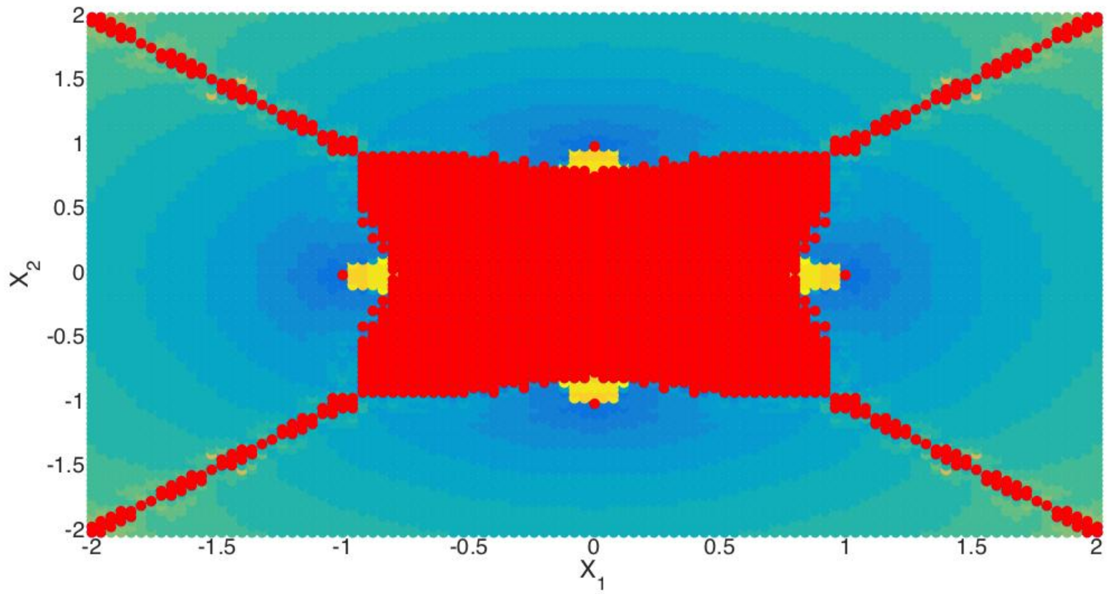
We plotted the grid of start points $x_k = 2 + 4k/100$ and $y_j = 2 + 4j/100$ for $k, j = 0, \cdots, 100$ and each point was colored to indicate which points converged. In addition, the color of the grid points also indicates how many iterations were required to achieve the minima.

In the plots below, all red points are start points that either didn't converge within the maximum number of iterations or converged to the local maxima at $\mathbf{0}$. The yellow points are start points which take more number of iterations to converge. The darker blue a point is, the lesser number of steps required for convergence.

Fixed-step



Newton's method



Conjugate Gradient

We implemented the conjugate gradient method for non-quadratic problems using the Fletcher-Reeves approximation. However, none of the start points converged in the maximum number of allowed iterations. A possible fix to this is to restart the conjugate gradient method after a few iterations. This is because some iterations the Q-conjugacy criteria will not hold because of the approximation. So, restarting the algorithm would help convergence properties.

Analysis of Convergence

As we can see in the convergence plots above, Newton's method converges faster to the correct minima than the fixed-step method. However, the fixed-step method finds the correct minima from a lot more start points than the newton's method. This may be because of the requirement for positive definite of the Hessian is violated. A possible solution to this problem would be to use a quasi-newton approach.

It is also seen that convergence is not achieved for start points of the form $z^0 = x^0 \pm ix^0$ for both the algorithms. Near-by points to the $y = x$ line are harder to converge for fixed-step algorithm than for the Newton's method.

MATLAB algorithm implementation

```
function z = vfunc(t)
%% Function evaluation at (x,y)
    x = t(1);
    y = t(2);
    z = (x^4 + y^4 - 6*x^2*y^2 - 1)^2 + (4*x^3*y - 4*x*y^3)^2;
end

function z = gfunc(t)
%% Gradient evaluation at (x,y)
    x = t(1);
    y = t(2);
    z = [8*x^7 - 8*x^3 + 24*x^3*y^4 + 24*x^5*y^2 + 8*x*y^6 + 24*x*y^2;
         8*y^7 - 8*y^3 + 24*x^4*y^3 + 8*x^6*y + 24*x^2*y^5 + 24*x^2*y];
end

function z = hfunc(t)
%% Hessian evaluation at (x,y)
    x = t(1);
    y = t(2);
    f1 = 56*x^6 - 24*x^2 + 72*x^2*y^4 + 120*x^4*y^2 + 8*y^6 + 24*y^2;
    f2 = 56*y^6 - 24*y^2 + 72*x^4*y^2 + 120*x^2*y^4 + 8*x^6 + 24*x^2;
    f12 = 48*x*y*((x^2 + y^2)^2 + 1);
    z = [f1, f12; f12, f2];
end

function [xs, v, g, itr] = fixed(vfunc, gfunc, alpha, x0, TOL, MAX_ITR)
%% Fixed-step algorithm implementation
    itr = 1;
    xk = x0;
    dk = -gfunc(xk);
    xk1 = xk + alpha*dk;
    while(norm(xk1 - xk)/norm(xk) > TOL && norm(dk) > TOL && itr < MAX_ITR)
        xk = xk1;
        dk = -gfunc(xk);
        xk1 = xk + alpha*dk;
        itr = itr + 1;
    end
    xs = xk1;
    v = vfunc(xs);
    g = gfunc(xs);
end

function t = backtrack(vfunc, gfunc, xk, dk, T)
%% Backtracking algorithm
    ALPHA = 0.5;
    BETA= 0.5;
    MAX_ITR = 100;
    R = -(gfunc(xk)'*dk)/(norm(gfunc(xk)));
    t = T;
    g1 = vfunc(xk + t*dk);
    h1 = vfunc(xk) - R*ALPHA*t*norm(gfunc(xk));
    itr = 1;
    while (g1 > h1 && itr < MAX_ITR)
        t = BETA*t;
```

```
        g1 = vfunc(xk + t*dk);
        h1 = (vfunc(xk) - R*ALPHA*t*norm(gfunc(xk)));
        itr = itr + 1;
    end
end

function [xs, v, g, itr] = newton(vfunc, gfunc, hfunc, x0, TOL, MAX_ITR)
%%  Newton's method algorithm implementation
    xk = x0;
    dk = -inv(hfunc(xk))*gfunc(xk);
    ndk = norm(dk);
    tk = backtrack(vfunc, gfunc, xk, dk/ndk, ndk);
    xk1 = xk + tk*dk/ndk;
    itr = 1;
    while (norm(xk1 - xk)/norm(xk) > TOL && norm(gfunc(xk1)) > TOL && itr < MAX_ITR)
        xk = xk1;
        dk = -inv(hfunc(xk))*gfunc(xk);
        ndk = norm(dk);
        tk = backtrack(vfunc, gfunc, xk, dk/ndk, ndk);
        xk1 = xk + tk*dk/ndk;
        itr = itr +1;
    end
    xs = xk1;
    v = vfunc(xs);
    g = gfunc(xs);
end

function [xs, v, g, itr] = conjugate(vfunc, gfunc, x0, TOL, MAX_ITR)
%% Conjugate gradient algorithm implementation
    xk = x0;
    if norm(gfunc(xk)) < TOL
        return
    else
        itr = 1;
        dk = -gfunc(xk);
        ndk = norm(dk);
        ak = backtrack(vfunc, gfunc, xk, dk/ndk, ndk);
        xk1 = xk +ak*dk/ndk;
        while (norm(gfunc(xk1)) > TOL && itr < MAX_ITR)
            bk = (gfunc(xk1)'*gfunc(xk1))/(gfunc(xk)'*gfunc(xk));
            dk = -gfunc(xk1) + bk*dk;
            ndk = norm(dk);
            ak = backtrack(vfunc, gfunc, xk, dk/ndk, ndk);
            xk = xk1;
            xk1 = xk1 + ak*dk/ndk;
            itr = itr + 1;
        end
    end
    xs = xk1;
    v = vfunc(xs);
    g = gfunc(xs);
end

%% Driver implementation
e2 = 1e-2;
e7 = 1e-7;
TOL = e7;
```

```
MAX_ITR = 10000;
ALPHA = 0.0001;

FIXED_FILE=fopen('fixed.txt','w');
NEWTON_FILE=fopen('newton.txt','w');
CONJUGATE_FILE=fopen('conjugate.txt','w');

fprintf(FIXED_FILE,'count\tX0(1)\tX0(2)\tXs(1)\tXs(2)\tf(Xs)\tItrs\tConv.\n');
fprintf(NEWTON_FILE,'count\tX0(1)\tX0(2)\tXs(1)\tXs(2)\tf(Xs)\tItrs\tConv.\n');
fprintf(CONJUGATE_FILE,'count\tX0(1)\tX0(2)\tXs(1)\tXs(2)\tf(Xs)\tItrs\tConv.\n');

count = 1;
for ii=0:100
    for jj=0:100
        x = [-2 + 4*ii/100; -2 + 4*jj/100];

        [xf, vf, gf, itf] = fixed(@vfunc, @gfunc, ALPHA, x, TOL, MAX_ITR);
        [xn, vn, gn, itn] = newton(@vfunc, @gfunc, @hfunc, x, TOL, MAX_ITR);
        [xc, vc, gc, itc] = conjugate(@vfunc, @gfunc, x, TOL, MAX_ITR);

        fprintf(FIXED_FILE,'%5i\t%4.3f\t%4.3f\t%4.3f\t%4.3f\t%4.3f\t%5i\t%i\n', ...
          count, x(1), x(2), xf(1), xf(2), vf, itf, itf < MAX_ITR && norm(xf) >
e2);
        fprintf(NEWTON_FILE, '%5i\t%4.3f\t%4.3f\t%4.3f\t%4.3f\t%4.3f\t%5i\t%i\n',
...
          count, x(1), x(2), xn(1), xn(2), vn, itn, itn < MAX_ITR && norm(xn) >
e2);
        fprintf(CONJUGATE_FILE, '%5i\t%4.3f\t%4.3f\t%4.3f\t%4.3f\t%4.3f\t%5i\t%i\n',
...
          count, x(1), x(2), xc(1), xc(2), vc, itc, itc < MAX_ITR && norm(xc) >
e2);
        count = count + 1;
    end
end
```