Question 1.

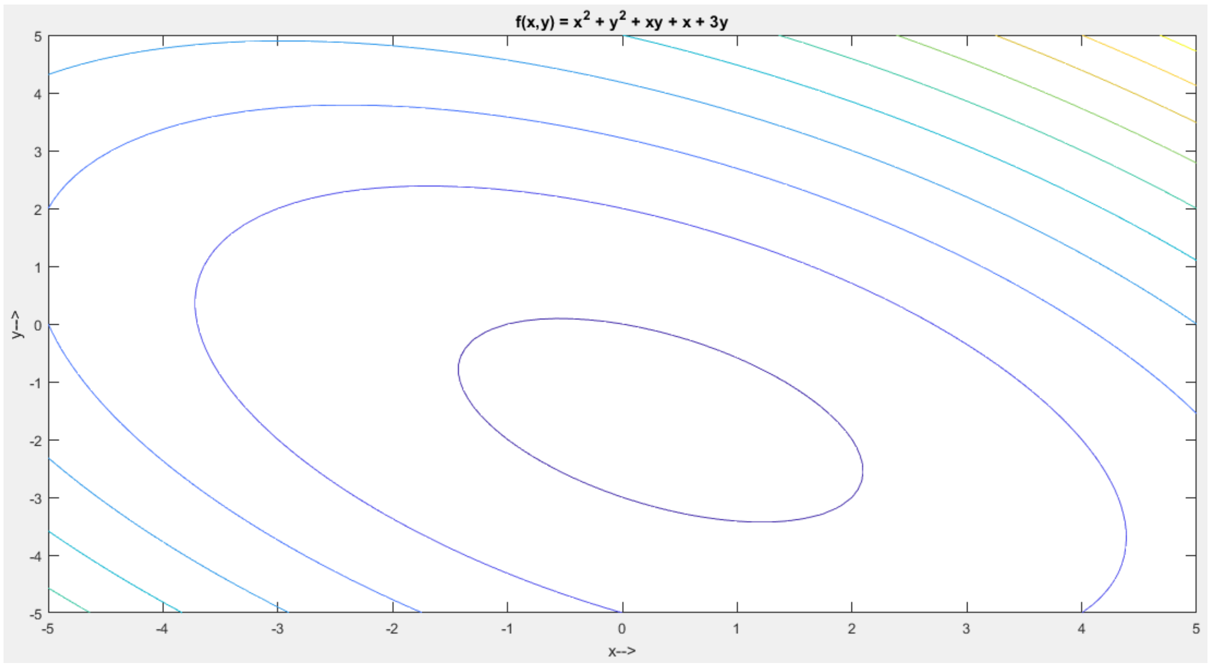
The two main iterative methods for solving least-squares are done by recasting the sym+def linear equations as an optimization program. Thus, they reduce to minimization of a quadratic convex function. The two methods are *Steepest Descent* and *Method of Conjugate Gradients.* The steepest descent algorithm can be inefficient because it can move in essentially the same direction many times. Conjugate gradient method avoids this by ensuring that each step is orthogonal to all previous steps that have been taken. CG can thus get an approximate solution using much less computation than solving the system directly. Thus, it significantly outperforms steepest descent algorithm. We then talked about convergence guarantees and found the bounds for the number of iterations for these two methods.

We shifted our focus from static problems to streaming problems, where at each time, we want to perform the best estimate from the observations seen up to that point. This can also be adapted to the kernel regression framework. We discussed the matrix inversion lemma which was crucial to the analysis of *online least squares*. We analysed the updated terms and concluded that the computation time was significantly lesser when matrix inversion lemma was used.

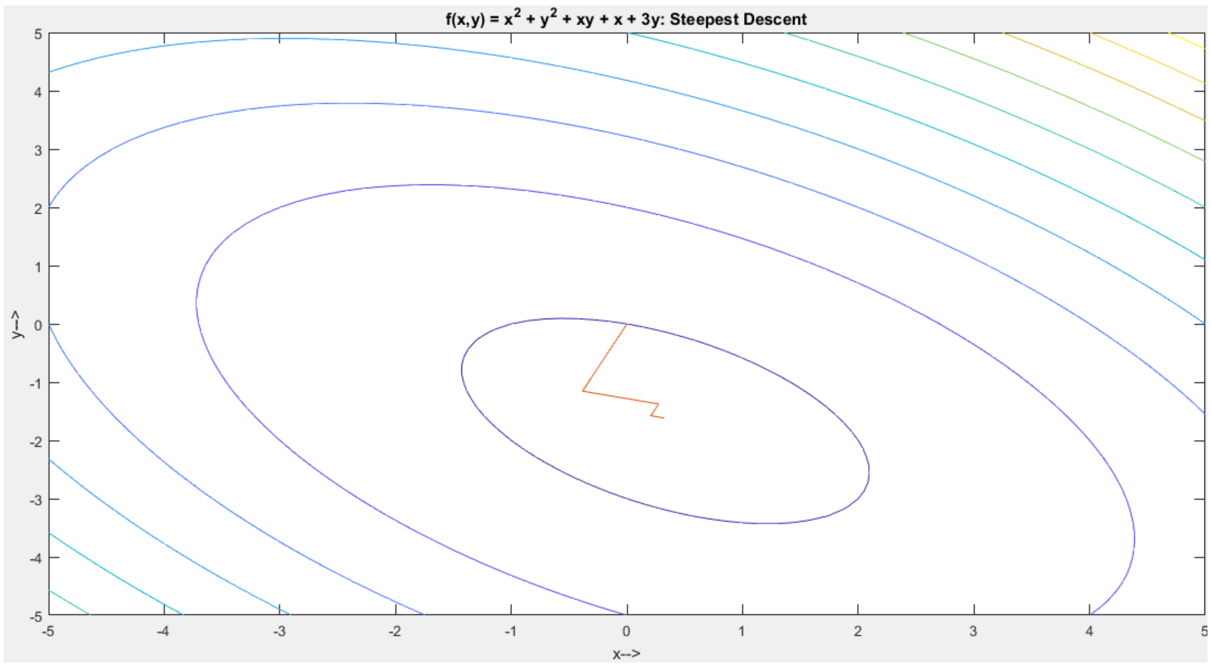
Till now, we had considered a very specific estimation framework: the unknown variables were subjected to a linear operator (matrix), perturbed and then observed. But what if we estimate unknown variables using observations that are samples of a random variable, whose distribution is controlled by some unknown parameter? Thus, we discussed about the Best Linear Unbiased Estimator (BLUE). We briefly reviewed probability concepts that were needed to analyse BLUE. Properties of BLUE were discussed and we observed that there can be significant gains from loosening the restriction on it being unbiased. We concluded that BLUE is an overall optimal estimator. We then delved into the *minimum mean-square estimation*. Multivariate Gaussian and the method of Gaussian estimation was discussed in detail. A brief look into Gaussian graphical models made us realize the importance of the inverse covariance matrix. We shall look more into this in the coming classes.

Question 2.

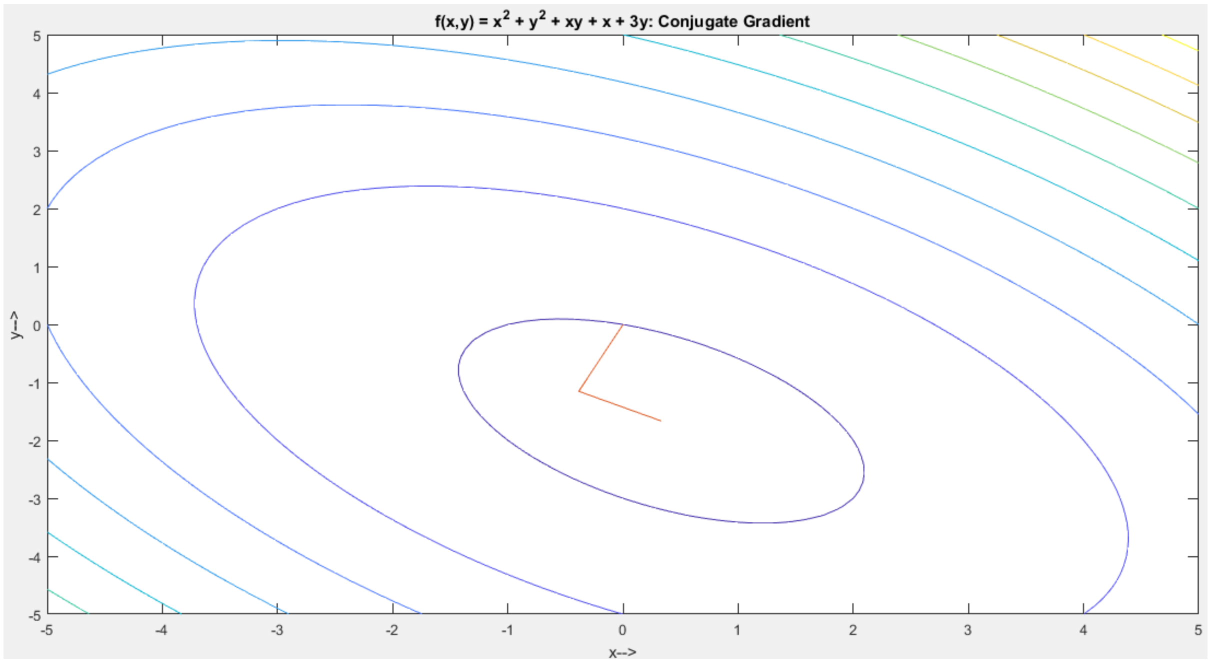
(c) Contour plot of f(x) around its minimizer.



(d) First four steps of gradient descent algorithm starting at x = 0.



(e) First two steps of conjugate gradient algorithm starting at x = 0.



Code:

```
H = [2 1; 1 2];
b = [-1; -3];
syms x1 x2
f(x1,x2) = x1^2 + x2^2 + x1*x2 + x1 + 3*x2;
figure;
fcontour(f);
title('f(x,y) = x^2 + y^2 + xy + x + 3y');
xlabel('x-->');
ylabel('y-->');

figure;
fcontour(f);
hold on;
title('f(x,y) = x^2 + y^2 + xy + x + 3y: Steepest Descent');
xlabel('x-->');
ylabel('y-->');
[xsd, itersd] = sdsolve(H, b, 10^-6, 4);
plot(xsd(1,:),xsd(2,:));

figure;
fcontour(f);
hold on;
title('f(x,y) = x^2 + y^2 + xy + x + 3y: Conjugate Gradient');
xlabel('x-->');
ylabel('y-->');
[xcg, itercg] = cgsolve(H, b, 10^-6, 2);
plot(xcg(1,:),xcg(2,:));
```

Question 3.

The number of iterations needed for convergence: **190**
tol = 10^{-6}.
norm(b) = 30.6123.
norm(r)/norm(b) = 9.4480e-07, which is lesser than tol.
Also, norm(Hx - b) = 2.8923e-05 < 3.0612e-05 (specified tolerance of b).
Thus, verified that Hx is within the specified tolerance of b.

Code:

```
%More efficient version 2.
function [x, iter] = sdsolve(H, b, tol, maxiter)
k = 1;
x(:,1) = zeros(1000,1);
% x(:,1) = zeros(2,1);
r(:,1) = b - H*x(:,1);
while (((norm(r(:,k))/norm(b)) > tol) && (k <= maxiter))
    q = H*r(:,k);
    a(:,k) = ((r(:,k)')*r(:,k))/((r(:,k)')*q);
    x(:,k+1) = x(:,k) + (a(:,k)*r(:,k));
    r(:,k+1) = r(:,k) - (a(:,k)*q);
    k = k + 1;
end
iter = k-1;
```

Question 4.

The number of iterations needed for convergence: **40**

tol = 10^{-6}.
norm(b) = 30.6123.
norm(r)/norm(b) = 8.5093e-07, which is lesser than tol.
Also, norm(Hx - b) = 2.6049e-05 < 3.0612e-05 (specified tolerance of b).
Thus, verified that Hx is within the specified tolerance of b.

Code:

```
function [x, iter] = cgsolve(H, b, tol, maxiter)
k = 1;
ss = 1000;
x(:,1) = zeros(1000,1);
% x(:,1) = zeros(2,1);
r(:,1) = b - H*x(:,1);
d(:,1) = r(:,1);
while (((norm(r(:,k))/norm(b)) > tol) && (k <= maxiter) && (k ~= (ss+1)))
    q = H*r(:,k);
    a(:,k) = ((r(:,k)')*r(:,k))/((d(:,k)')*H*d(:,k));
    x(:,k+1) = x(:,k) + (a(:,k)*d(:,k));
    r(:,k+1) = r(:,k) - (a(:,k)*H*d(:,k));
    beta(:,k+1) = ((r(:,k+1)')*r(:,k+1))/((r(:,k)')*(r(:,k)));
    d(:,k+1) = r(:,k+1) + beta(:,k+1)*d(:,k);
    k = k + 1;
end
iter = k-1;
```