The next 2 questions will refer to the following matlab code:

```matlab
function S = gaussianSampler(mu, Sigma, N)
    L = chol(Sigma);
    S = repmat(mu,N,1) + randn(N,length(mu))*L;
```

1 Sampling from a Gaussian (10 Points)

The function `gaussianSampler` takes a vector of means, `mu`, a covariance matrix `Sigma`, and a number of samples `N`. It then calls the function `randn` to generate `N` samples from a mean 0 multivariate Gaussian with standard deviation 1 and uses these to generate samples from the desired distribution.

Draw 10,000 samples from the $2 \times 2$ Gaussian with mean $[1 2]$ and covariance matrix:

\[
\begin{pmatrix}
1.0 & 0.5 \\
0.5 & 1.0
\end{pmatrix}
\]

and compute the maximum likelihood mean and covariance matrix. Since this is easily done on the command line in matlab, just paste in what you did. There is no need to submit a program. Don’t forget to put a semicolon after your call to `gaussianSampler`.

2 Sampling from a Gaussian II (10 points)

The code above uses the matlab function `chol` to compute the Cholesky decomposition of the covariance matrix. Do `help chol` (you might also try `doc chol`) to learn more about what the Cholesky decomposition returns.

To prove that `gaussianSampler` does the right thing, prove that if $L$ is a square, upper triangular matrix and $x$ is a row vector of independent mean 0 Gaussian random variables with standard deviation 1, then $y = \mu + xL$ has mean $\mu$ and covariance $L^T L$.

Nota bene: For the next two questions (and in general) don’t forget that, unlike C and many other languages, Matlab indices start from 1 instead of 0.
3 Marginals of Gaussians (10 points)

Implement a function: `gaussMarginal(mu, Sigma, b)`, which takes a Gaussian distribution specified by mean `mu` and covariance matrix `Sigma`, and a vector of `indices b`. It should return two values. The first should be mean of the Gaussian with the variables indexed by `b` marginalized out, and the second should be the covariance matrix of the Gaussian with variables indexed by `b` marginalized out.

Here is an example of correct output:

```matlab
>> [newmu, newsigma] = gaussMarginal([1 2 3], [1 0.5 0.25; 0.5 2 0.75; 0.25 0.75 5.0], [2])

newmu =
    1   3

newsigma =
    1.0000   0.2500
    0.2500   5.0000
```

4 Conditionals of Guassians (10 points)

Implement a function: `gaussConditional(mu, Sigma, b, val)`, which takes a Gaussian distribution specified by mean `mu` and covariance matrix `Sigma`, a vector of `indices b`, and vector of corresponding values `val`. It should return two values. The first should be mean of the Gaussian conditioned on the variables indexed by `b` with the corresponding values in `val` the second should be the resulting covariance matrix.

Here is an example of correct output:

```matlab
[newmu, newsigma] = gaussConditional([0 0]', [1.0 0.98; 0.98 1.0], 1, 0.7)

newmu =
    0.6860

newsigma =
    0.0396
```
5 Ordinary Least Squares, The Silly Way (10 points)

Implement ordinary least squares regression in matlab using quadprog by explicitly including the sum of squared differences from the training data in your objective function.

You should implement a function \texttt{regress(Phi, t)} which takes an \( m \times n \) design matrix \( \Phi \), and an \( m \times 1 \) column vector \( t \) with target values. It should return a \( n \times 1 \) column vector of the weights of the least squares solution.

Note that this is a terribly inefficient way to approach ordinary least squares, so it is entirely an exercise in familiarizing yourself with quadprog. Your code should, of course, return the same answer as the more efficient methods. Also, we suggest you pass an options argument to quadprog instructing it to use an interior point method. You do this by setting an options variable and then passing it to quadprog per the documentation. The way to set the options argument for interior point is:

\[
\text{options} = \text{optimset('Algorithm','interior-point-convex');}
\]

Finally, you will get some diagnostic information from quadprog while it is running. You don’t need to worry about this unless it reports something suggesting that it has failed to find a solution.

6 Lasso Implementation (10 points)

Modify your solution to the previous problem to use quadprog to implement \( L_1 \) regularized least squares regression, solving the following objective function:

\[
w^* = \arg\min_w \lambda \|w\|_1 + \sum_{i=1}^{m} (\phi^{(i)} w - t_i)^2
\]

for constant \( \lambda \). Note that \( \phi^{(i)} \) refers to the \( i^{th} \) row of \( \Phi \). Your function should be \texttt{L1regress(Phi, lambda, t)}; for \( m \times n \) design matrix \( \Phi \), \( m \times 1 \) target vector \( t \) and scalar regularization parameter \( \lambda \).

7 Using Lasso (10 points)

Plot the performance of your Lasso code with a degree 9 polynomial trained on the noisy sin function used in class and in the textbook. The training data can be downloaded from the book web page. Here’s a direct link:

\[
\text{http://research.microsoft.com/en-us/um/people/cmbishop/PRML/webdatasets/curvefitting.txt}
\]

For your test data, use the function: \( y = \sin(2\pi x) \) for \( x \in [0,1] \), sampling at steps of 0.05, starting at 0 and ending at 1, i.e., 21 points. Plot the sum of squared errors on the test set:

\[
\sum_{i=1}^{21} (\phi^{(i)} w - y^i)^2
\]

for \( \lambda = 10^i \) for \( -10 \leq i \leq 5 \) in steps of 1. Note that here \( \phi^{(i)} \) refers to the features of the \( i^{th} \) test point.