1. The Matlab function `linesearch.m` provided on the class web page with this assignment has the following header:

   ```matlab
   function x = linesearch(Grad, x0, p)
   ```

This function takes three arguments:

- The name `Grad` of a function with header
  ```matlab
  function g = Grad(x)
  ```
  that evaluates the gradient of a function \( f(x) \) at \( x \).
- A starting point \( x_0 \) for line search.
- A search direction \( p \) that need not be normalized to unit length.

The idea is that if you want to minimize a function \( f(x) \) with some method that requires knowing the gradient \( g(x) \) of \( f(x) \), then you just provide a routine to compute \( g(x) \). The function `linesearch` is different from what we saw in class: instead of finding a minimum of \( f(x) \), it finds a zero of \( g(x) \) after checking that \( g(x_0)^T p \) is indeed negative. This is why the version of line search given above does not take \( f \) itself as argument. In fact, \( f(x) \) itself is never needed for this homework. All you need is its gradient.

(a) Write a Matlab function that converges towards the minimum of a function \( f(x) \) by steepest descent. Your function should have the following header:

   ```matlab
   function [x,allx] = steepest(Grad,x,maxits,deltax)
   ```

   where

   - `Grad` is the name of a function that computes the gradient of \( f(x) \), as discussed above.
   - The input argument \( x \) is a starting point.
   - `maxits` is the maximum number of iterations for steepest descent.
   - `deltax` is a termination bound: your function will stop either after `maxits` iterations, or when the last step \( x_k - x_{k-1} \) was shorter than `deltax`, whichever comes first.
   - The output argument \( x \) contains the solution point.
   - The output `allx` contains all the steps \( x_k \) that `steepest` took to reach the minimum. This is useful to see what your minimization function does.

   Hint: when `steepest` needs to evaluate `Grad` at \( x \), it calls a Matlab function called `feval` as follows: \( g = feval(Grad, x) \).

(b) The function

   \[
   f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2
   \]

   is known as Rosenbrock's function. It has a unique minimum at \( x^* = (1, 1)^T \). Write an expression for the gradient of this function.

(c) Write a Matlab function \( g = rosengrad(x) \) that computes the gradient \( g \) of Rosenbrock's function at \( x \). Turn in your code.

(d) Call `steepest` with `rosengrad` as follows:

   ```matlab
   [x,allx] = steepest(‘rosengrad’, [-1.2 1], 100, 1.0e-4); 
   ```

   and display your results. To do this, do
numsteps = size(allx,2)
drawrosenbrock(allx);

where the function drawrosenbrock is also supplied on the class web page. This plots the first 100 steps taken by your algorithm when it is started at $x_0 = (-1.2, 1.0)^T$. Show the resulting plot.

(e) Does your algorithm converge within 100 steps? If not, what is $\delta_{x}$ when it gives up?

(f) How far was the solution $x$ from $x^*$ upon termination?

(g) Arm yourself with patience. How many steps does it take for your steepest descent code to converge within the given $\delta_{x}$?

(h) How far was the solution $x$ from $x^*$ upon termination?

(i) Write a Matlab function or script that minimizes a function by the conjugate gradients method. Your function should have the following header:

```
function [x, allx] = conjugate(Grad, x, maxits, deltax)
```

(j) Test your routine on Rosenbrock’s function by doing the following:

```
[x,allx] = conjugate(’rosengrad’, [-1.2 1], 100, 1.0e-4);
numsteps = size(allx,2)
drawrosenbrock(allx);
```

and show the resulting plot.

(k) Does your algorithm converge within 100 steps? If not, what is $\delta_{x}$ when it gives up?

(l) How far was the solution $x$ from $x^*$ upon termination?

(m) (No credit) Which local minimization method is preferable, steepest descent or conjugate gradients?