You must turn in two things. (1) Your code, including a script called `hmwk7` that runs all the experiments (and generates the plots) asked for below. (2) A short writeup (pdf) containing the plots asked for below, and your answers to the questions in problem 5.

1. Write a method with signature

   ```
   function x_sol, res = grad_descent(f,grad,x0)
   ```

   The inputs `f` and `grad` are function handles. The function `f`: $\mathbb{R}^N \to \mathbb{R}$ is an arbitrary objective function, and `grad`: $\mathbb{R}^N \to \mathbb{R}^N$ is its gradient. The method should minimize f using gradient descent, and terminate when the gradient of f is small. I suggest stopping when

   $$\|\nabla f(x^k)\| < \|\nabla f(x^0)\| * tol$$

   where x^0 is an initial guess and *tol* is a small tolerance parameter (a typical value would be 10^{-4}).

   Use a backtracking line search to guarantee convergence. The stepsize should be monotonically decreasing. Each iteration should begin by trying the stepsize that was used on the previous iteration, and then backtrack until the Armijo condition holds:

   $$f(x^{k+1}) \leq f(x^k) + \alpha\langle x^{k+1} - x^k, \nabla f(x^k)\rangle,$$

   where $\alpha \in (0,1)$, and $\alpha = 0.1$ is suggested.

   The function returns the solution vector `x_sol`, and also a vector `res` containing the norm of the residual (i.e., the norm of the gradient) at each iteration.

   You can obtain the initial stepsize by estimating a Lipschitz constant for `f` using the formula:

   $$L \approx \frac{\|\nabla f(x) - \nabla f(y)\|}{\|x - y\|}$$

   where $x = x^0$ and y is obtained by adding a small random perturbation to x. The initial stepsize is then $\tau = \frac{2}{L}$. Use a vector of zeros as the initial guess x^0.

2. Modify your solution from Question 1 to create the new function

   ```
   function D, c = grad_descent_bb(f,grad,x0)
   ```

   This method is the same as the method from problem 1. However, instead of using a monotonically decreasing stepsize, begin each linesearch with a Barzilai-Borwein stepsize of magnitude

   $$\tau = \frac{\langle x^{k+1} - x^k, x^{k+1} - x^k\rangle}{\langle x^{k+1} - x^k, \nabla f(x^{k+1}) - \nabla f(x^k)\rangle}.$$

   This time, you can start with a smaller initial stepsize to avoid unnecessary backtracking.

3. Modify your solution from Question 1 to create the new function

   ```
   function D, c = grad_descent_nesterov(f,grad,x0)
   ```

This function should implement Nesterov's method:

$$x^k = y^k - \tau \nabla f(y^k)$$
$$\delta^{k+1} = \frac{1 + \sqrt{1 + 4(\delta^k)^2}}{2}$$
$$y^{k+1} = x^k + \frac{\delta^k - 1}{\delta^{k+1}}(x^k - x^{k-1}).$$

The method is initialized with $x^0 = y^1$ and $\delta^1 = 1$, and the first iteration has index $k = 1$. Use the monotone line search from Question 1. For the Nesterov problem, the backtracking condition is

$$f(x^k) \leq f(y^k) + \alpha \langle x^k - y^k, \nabla f(y^k) \rangle.$$

Unlike standard gradient methods, Nestorov's method requires $\alpha \in [1/2, 1)$. I suggest using $\alpha = 1/2$.

4. Test your three solvers using the logistic regression classification problem from homework 4. Remember, you already have code for generating this objective function and gradient. You handed these functions to your gradient checker.

   Your test should use a classification problem with 200 feature vectors, 20 features per vector, and condition number $\kappa = 1$. Then use your solvers to minimize `logreg_objective` using the gradient function `logreg_grad`. Use a stopping condition with $tol = 10^{-8}$ to verify that the methods converge to a high degree of precision. Plot the residuals for the different methods using a *logarithmic* y-axis. For example, in Matlab you could call

```
[D, c] = create_classification_problem(200, 20, 1);
[x_sol, res] = grad_descent( @(x) logreg_objective(x,D,c),
                             @(x) logreg_grad(x,D,c), x0);
semilogy(res)
```

   Now, re-run the experiment with condition number $\kappa = 100$.

   Turn in two plots: one plot with all three residual curves for $\kappa = 1$, and one plot for all three residual curves using $\kappa = 100$. Make sure the plots have titles and legends.

5. Answer the following questions. Keep your answers short (but complete).

   a Why is the stopping condition in problem 1 based on the residual? Why don't we just terminate when the objective function gets small?

   b Why is the stopping condition suggested in problem 1 better than just terminating when the residual is small, i.e. when $\|\nabla f(x^k)\| < tol$.

   c Which gradient method was best?

   d What effect does the condition number have on the convergence speed?

6. Now use your Barzilai-Borwein solver to train a neural net on MNIST. Using your code from previous homeworks, build the objective function and gradient operator for a net with 4 hidden layers of sizes 50, 40, 30, and 20. Run your gradient solver on this problem for 200 iterations.

   Calculate the accuracy (percentage of correctly classified digits) of your solution on the training data. Now, calculate the accuracy on the test data. **Report these accuracies in your pdf, and plot the converge curve.**

   You won't be able to plug your gradient and objective functions directly into your gradient solver because your neural net routines expect a list (or cell array in Matlab) of weight matrices, while your gradient solver expects a vector. To solve this problem, use the `list_to_vec` and `vec_to_list` routines (Python) or `cell_to_vec` and `vec_to_cell` routines (Matlab) to convert between these formats. These routines are provided on the course webpage.

   Using these routines, it should take only a few lines of code to produce gradient and objective functions that operate on vectors rather than lists of matrices. The gradient function should unpack its argument (i.e., convert a vector into a list of matrices), hand it to your original backprop routine, and then convert the result back into a vector.

   Note: I suggest you subtract the average from each column of the data matrix so that each input feature has mean zero. This will make the network train much faster. I you do this, you also need to subtract the same values from the test data.