This project studies the following **unconstrained optimisation** methods, all using the **line-search** strategy: **Newton-CG**; **nonlinear conjugate gradients**; **quasi-Newton**. It consists of programming some Matlab functions and running them on some examples, commenting on the results. The references below are for the book *Numerical Optimization*, 2nd ed. by Nocedal and Wright.

Note: you may discuss issues with each other, but you have to produce your own solutions for every part.

I Matlab functions

Write the following Matlab functions, **using the templates provided** and following strictly the convention given for the input and output arguments:

1. `Olincg`: linear conjugate gradient (algorithm 5.2) to solve a positive-definite linear system $\mathbf{Ax} = \mathbf{b}$.
2. `Olincg1`: modification of `Olincg` that stops iterating when encountering negative or zero curvature directions. This is intended to be called from `Onewtoncg`.
3. `Onewtoncg`: line-search Newton-CG (algorithm 7.1), using `linesearch` and `Olincg1`. Use the same convergence criterion as in `Osteepdesc`, i.e., stop when $\|\nabla f(\mathbf{x}_k)\| \leq$ `tol` OR $k \geq$ `maxit` (tolerance achieved or maximum number of iterations achieved). The argument `convcrit` determines what forcing sequence to pass to `Olincg1`: $\eta_k = 0.5$ for `'linear'`, $\eta_k = \min(0.5, \sqrt{\|\nabla f(\mathbf{x}_k)\|})$ for `'superlinear'`, and $\eta_k = \min(0.5, \|\nabla f(\mathbf{x}_k)\|)$ for `'quadratic'`.
4. `Oprcg`: three nonlinear conjugate gradients methods: Polak-Ribière, Fletcher-Reeves (algorithm 5.4) and Hestenes-Stiefel.
5. `Obfgs`: three quasi-Newton methods: BFGS (algorithm 6.1), DFP and SR1.
6. For each problem tested in the evaluation, write one driver file that sets up the problem, solves it and displays the results. As an example, see `driver0.m`. Since the combinations of different `tol`, different forcing sequence, etc. result in several problems, just send me 4 representative drivers, one for each of `Olincg`, `Onewtoncg`, `Oprcg` and `Obfgs`.

Important notes:

- Sometimes, these methods may compute directions that are not descent, or may have some other problem. Explain how you solve such issues.
- **Ensure your code (and drivers) is fast**. It should solve any of the problems we try (which are small) in a few seconds.

Programming advice:

- Write the functions in the order above. Use my functions `fcontours`, `plotseq`, `Fquad`, `Osteepdesc`, `linesearch`, `convseq`, etc.
- Make sure you understand how `Osteepdesc` (and `linesearch`) works because your `Onewtoncg`, etc. programs should look very similar to it. Follow the convention in `Osteepdesc` for:
  - default values for arguments
  - passing as an argument an arbitrary function handle and its parameters: `f, paramf`
  - obtaining the value of the function, gradient and Hessian: `[ff g H] = f(x',paramf:);`
- To get more decimals in Matlab do `format long`. Use `set(gca,'DataAspectRatio',[1 1 1]);` to avoid distorted plots.
- Program thinking of n dimensions, not 2. It's more general and usually easier.
- Avoid fancy features I don't ask for: error-checking of arguments, informative messages, etc. This is not a programming course.
- You may find useful the following Matlab construct to solve linear systems: `x0 = A \ b;` will solve `A*x = b`. The corresponding error `A*x0 - b` should be around 10^{-13} or less unless `A` is ill-conditioned.
- The function `numgradhess.m` is useful to check numerically whether the expressions for your gradient and Hessian are correct.

II Evaluation

Once you have correctly programmed the algorithms, explore them as follows.

1. `Olincg`. Test it with a system where $\mathbf{A}$ is the Hilbert matrix of order n ($a_{ij} = \frac{1}{i+j-1}$ for $i, j = 1 \ldots, n$), $\mathbf{b} = (1, \ldots, 1)^T$ and with initial point $\mathbf{x}_0 = \mathbf{0}$. Try dimensions $n = 3,\ 5,\ 8,\ 12,\ 20$. Use the following Matlab functions: `hilb`, `cond`.

   (a) Report the condition number of $\mathbf{A}$ and the number of iterations required to reduce the norm of the residual $\mathbf{r}_k = \mathbf{A}\mathbf{x}_k - \mathbf{b}$ below 10^{-6}. Discuss the results in view of theorem 5.1.

   (b) Plot f_k and $\|\mathbf{r}_k\|$ (the latter using `semilogy`) as a function of the iteration index k. At each iteration, does the value of f_k decrease? How about the value of $\|\mathbf{r}_k\|$?

2. `Onewtoncg`. Apply it to the Rosenbrock function in 2 dimensions (eq. (2.22) in the book) from two initial points $\mathbf{x}_0 = (1.2, 1.2)$ and $\mathbf{x}_0 = (-1.2, 1)$ (cf. exercise 3.1).

   (a) Plot the iterates over the function contours (use `fcontours` and `plotseq`). Plot $\|\mathbf{x}_k - \mathbf{x}^*\|$ (where $\mathbf{x}^*$ is the exact minimiser) and $f(\mathbf{x}_k)$ as a function of k. Report the number of Newton iterations, and the number of CG iterations ran inside each Newton iteration. How large is the condition number of $\nabla^2 f(\mathbf{x}^*)$?

   (b) Explore the effects of the following and comment on the results:

      i. The value of `tol`, e.g. try 10^{-6} and 10^{-10}.

      ii. The line search: use different values of the initial step length, e.g. $\overline{\alpha} = 1,\ 0.5,\ 0.2$. Also, try not using a line search at all and fixing the step length to $\alpha = 1$.

      iii. The forcing sequence (use `convseq` to estimate empirically the convergence rate).

   (c) Repeat everything for `Osteepdesc` and compare the results with those of `Onewtoncg`, in terms on convergence rate, number of iterations and (roughly) in terms of *actual* computational cost.

   (d) Consider now the Rosenbrock function in n dimensions (see exercise 7.1 in the book). Try the previous algorithms for $n = 100$ from the initial point $\mathbf{x} = (-1, -1, \ldots, -1)^T$. How do your results change? You can plot the first 2 dimensions of $\mathbf{x}_k$ to get an idea of the algorithms' progress.

   (e) Finally, run `driver0.m`, which runs `Onewtoncg` with function `F6_1b` (n dimensions). Comment on the results.

3. `Oprcg`, `Obfgs`: repeat a similar evaluation as for `Onewtoncg`. In particular, estimate the convergence rate empirically.

In the evaluation, do not simply report results and above all do not flood me with plots or tables without explanation. You should run your algorithms under various conditions such as those mentioned above and ensure you understand their behaviour and how it compares with their theoretical properties. After you have done that exploration on your own, summarise it clearly and concisely in the report. A few representative plots are enough. Have the 4 driver files recreate figures or tables as necessary (e.g. by calling your `Obfgs` with appropriate arguments) and refer to them in the report (so you don't have to include them in the report).

III What you have to submit

Follow the following instructions strictly. Email me the following packed into a **single** file (`project1.tar.gz` or `project1.zip`) and with email subject `[EECS260] Project 1`:

1. Your code for the Matlab functions (`Olincg.m`, `Olincg1.m`, `Onewtoncg.m`, `Oprcg.m`, `Obfgs.m`), and the 4 representative driver files (`driver1.m`, `driver2.m`, `driver3.m`, `driver4.m`). Do not include any other functions (e.g. `fcontours.m`).

2. Your evaluation of the methods in a single PDF file (`evaluation.pdf`). Do not include there the methods' actual results (figures, tables), instead have the driver files reproduce them (I will run the driver files and check it against your evaluation).

Use file names as above and do not include any other files besides those. If your project was done by a group of 2 students or more, send me only one file and briefly describe in the evaluation what each member did for the project.