This project studies the **quadratic-penalty method for constrained optimisation** (chapter 17 of the book *Numerical Optimization* by Nocedal and Wright, 2nd. ed.; read carefully section 17.1). It consists of some problems to be solved analytically (in paper); programming some Matlab functions; and running them on some examples, commenting on the results.

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

We will consider the general constrained optimisation problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \text{ subject to } \begin{cases} c_i(\mathbf{x}) = 0, & i \in \mathcal{E} \\ c_i(\mathbf{x}) \geq 0, & i \in \mathcal{I} \end{cases}$$

using the following unconstrained optimisation methods to solve the subproblem:

1. Steepest descent (`Osteepdesc.m`).
2. Newton-CG (`Onewtoncg.m`).

I Paper exercises

1. Consider the quadratic-penalty function $Q(\mathbf{x};\mu)$ of eq. (17.7):

$$Q(\mathbf{x};\mu) = f(\mathbf{x}) + \frac{\mu}{2}\sum_{i\in\mathcal{E}} c_i^2(\mathbf{x}) + \frac{\mu}{2}\sum_{i\in\mathcal{I}} ([c_i(\mathbf{x})]^-)^2.$$

   Write the expressions for Q, $\nabla_{\mathbf{x}} Q$ and $\nabla^2_{\mathbf{xx}} Q$ in terms of f, ∇f, $\nabla^2 f$, c_i, ∇c_i, $\nabla^2 c_i$.

2. Assuming f and all the constraints have continuous second derivatives:
   (a) Show that $\nabla_{\mathbf{x}} Q(\mathbf{x};\mu)$ is continuous $\forall \mathbf{x} \in \mathbb{R}^n$, $\mu > 0$.
   (b) Give a condition for $\nabla^2_{\mathbf{xx}} Q(\mathbf{x};\mu)$ to be continuous $\forall \mathbf{x} \in \mathbb{R}^n$, $\mu > 0$. Apply it to the case where $c_i(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}_i\mathbf{x} + \mathbf{b}_i^T\mathbf{x} + c_i$ (quadratic constraints).

   Given this, comment on the applicability of the unconstrained optimisation methods.

3. Consider the Lagrange multiplier estimates

$$\lambda_i^{(k)} = -\mu_k c_i(\mathbf{x}_k)$$

   given in eq. (17.10) for the equality-constrained case. Guess how they may be extended to the inequality-constrained case. Hint: what happens with eq. (17.10) for an inequality constraint c_i which is *inactive* at a KKT point $\mathbf{x}^*$?

4. The convergence criterion (to a stationary point) that we used for unconstrained optimisation is $\|\nabla f(\mathbf{x}_k)\| < \tau$ for some tolerance τ. Write an appropriate convergence criterion to a KKT point (take into account the result of I.3).

5. (Optional.) The equation for the tangent to the path $\mathbf{x}(\mu)$ is (see the course notes)

$$\frac{d\nabla_{\mathbf{x}} Q(\mathbf{x}(\mu);\mu)}{d\mu} = \nabla^2_{\mathbf{xx}} Q(\mathbf{x};\mu)\,\dot{\mathbf{x}} + \frac{\partial \nabla_{\mathbf{x}} Q(\mathbf{x};\mu)}{\partial \mu} = \mathbf{0} \qquad \dot{\mathbf{x}} = \frac{d\mathbf{x}}{d\mu} \in \mathbb{R}^n$$

   where $\dot{\mathbf{x}}$ is the path tangent. This gives a linear system $\mathbf{H}\dot{\mathbf{x}} = \mathbf{b}$ for $\dot{\mathbf{x}}$. Write the expressions for $\mathbf{H}$ and $\mathbf{b}$.

II Matlab functions

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

1. `Q.m`: quadratic-penalty function. Use your expressions from I.1. See `fcontours.m` for passing functions and their arguments as arguments of `Q`.

2. `Oqpenalty.m`: quadratic-penalty method. Follow the algorithmic framework 17.1.

   (a) Decide yourself on how to implement the updates for μ_k and τ_k.

   (b) To compute the new starting point $\mathbf{x}^s_{k+1}$, implement two versions:
   - $\mathbf{x}^s_{k+1} \leftarrow \mathbf{x}^s_k$, i.e., we simply use the minimiser of $Q(\mathbf{x};\mu_k)$ as the starting point for minimising $Q(\mathbf{x};\mu_{k+1})$.
   - (Optional) $\mathbf{x}^s_{k+1} \leftarrow \mathbf{x}_k + (\mu_{k+1}-\mu_k)\dot{\mathbf{x}}$, i.e., we use the value predicted by the path tangent. For this, write a function `Qinit.m` that implements your solution to I.5.

   (c) For the "final convergence test" use your criterion of I.4 and also a limit `maxit` on the number of iterations (essential to debug the code, and to avoid infinite loops with e.g. unbounded problems).

3. Unconstrained optimisation functions: you will use `Osteepdesc.m` and `Onewtoncg.m`. A particular unconstrained optimisation solver, say `Onewtoncg`, will be passed to `Oqpenalty.m` as in this example (see `qpdriver0.m`):

   ```
   [X,LE,...] = Oqpenalty(f,paramf,E,paramE,I,paramI,@Onewtoncg,{convcrit},x0,mu0,tol,maxit)
   ```

   The template `Oqpenalty.m` already contains a line to call the unconstrained optimisation solver (passed in the input arguments `O` and `paramO`):

   ```
   [X1,F1] = O(@Q,{f,paramf,E,paramE,I,paramI,mu},x',tau,maxit2,paramO{:});.
   ```

   which mimics the way we would call `Onewtoncg` directly:

   ```
   [X,F] = Onewtoncg(f,paramf,x0,tol,maxit,convcrit).
   ```

   Don't change the input or output arguments in `[X1,F1] = O(@Q,...)` To see more examples of how to call functions of functions in Matlab using function handles, see `fcontours.m`. This receives as input arguments an objective function and several equality and inequality constraints and plots them. Note e.g. the use of `E{i}([X(:) Y(:)],paramE{i}{:})` to apply the `i`th equality constraint to the points in `[X(:) Y(:)]`.

4. Consider exercise 18.3 in the book. Program a driver file `qpdriver18_3.m`; and two functions (`F18_3f.m` for f and `F18_3c3` for c_3) to compute the necessary functions, gradients and Hessians (note that for c_1 and c_2 you can simply use `Fquad.m`). Follow the same style as in `Fquad`. Hint: you can simplify the task by using these identities:

   $$e^f \text{ has gradient } e^f\nabla f \text{ and Hessian } e^f(\nabla^2 f + \nabla f \nabla f^T)$$
   $$f^2 \text{ has gradient } 2f\nabla f \text{ and Hessian } 2(f\nabla^2 f + \nabla f \nabla f^T)$$

   and checking the output of your `F18_3f.m` and `F18_3c3.m` with `numgradhess.m`. See also the sample output provided in the file `qpdriver18_3.txt`.

Important notes:

- Ensure your code (and drivers) is fast. It should solve any of the problems we try (which are small) in a few seconds.
- Ensure your code runs the provided `qpdriver0.m` without errors.
- Don't modify the input and output arguments or the expected dimension of vectors or arrays given in the templates provided, such as `x0` or `X`.

It will be most efficient if you (1) do the paper exercises first, (2) write the functions in the order above, and (3) test `Q` and `Oqpenalty` with combinations of `Fquad`, visualising the results with `fcontours` (see also the programming advice in project #1). I have provided a sample driver file `qpdriver0.m` that sets up a constrained optimisation problem, solves it and displays the result; see also the iterates' path in `qpdriver0.png` and the Q function in the animation `qpdriver0.gif`.

III Evaluation

The objective is to evaluate your implementation `Oqpenalty` of the quadratic-penalty method on some test problems. You will report your results on: the problem in the provided file `qpdriver0.m`; exercise 12.19; exercise 16.1 part (b), already solved in the homework; and exercise 18.3 (in $n = 5$ dimensions).

To gain understanding of the method's performance, use different combinations of:

- Starting point $\mathbf{x}_0$.
- Updates for μ_k, τ_k.
- Starting point $\mathbf{x}_k^s$ for the unconstrained optimisation solver ($\mathbf{x}_k$ or the path tangent extrapolation).
- Unconstrained optimisation solver: `Osteepdesc`, `Onewtoncg`.

For each problem, write a different driver file[1] (based on `qpdriver0.m`) that sets it up, calls `Oqpenalty` and collects and plots/tabulates the following information:

1. Plots the contours of f and the constraints, and the sequence of minimisers $\mathbf{x}_k$. Call `fcontours` and `plotseq` from `qpdriver0.m` after `Oqpenalty` has finished, as in `qpdriver0.m`.
2. Plots the contours of $Q(\mathbf{x}; \mu_k)$, the constraints and the minimiser $\mathbf{x}_k(\mu_k)$, as a loop over the μ_k values after `Oqpenalty` has finished. Use `fcontours` and `plotseq` as in `qpdriver0.m`. Don't plot anything from within `Oqpenalty`.
3. Tabulates or plots the following information (obtained from the output arguments of `Oqpenalty`) as a function of μ_k: $f(\mathbf{x}_k)$; $\|\nabla f(\mathbf{x}_k)\|$; $\operatorname{cond}\left(\nabla^2_{\mathbf{xx}} Q\right)$; value of your convergence criterion; number of iterations of the unconstrained optimisation solver.
4. For exercise 18.3, do not plot anything, simply report at the end: the solution $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$; and an estimate of the computational cost it took to solve the problem (e.g. the number of iterations of `Oqpenalty` and the total number of iterations of the unconstrained optimisation solver; or the CPU time). Note there is an erratum in this exercise (see the errata list for the first edition of the book): $\mathbf{x}_0$ and $\mathbf{x}^*$ are swapped.

Here are some specific questions to answer, based on your theoretical understanding and your experiments:

1. Comment on your results in view of our theoretical understanding:
   (a) Applicability: does the particular combination work (e.g. `Onewtoncg` with inequality constraints)? Why or why not?
   (b) Efficiency: computational cost.
2. What can you say about the *shape* of the contours of Q near the boundary of the feasible set? (for equality constraints, and for inequality constraints). Hint: remember I.2.
3. Consider problems where f is linear or quadratic and all the constraints c_i are linear (i.e., LP and QP):
   (a) What can you say (in theory and given the experiments) about the performance of the different unconstrained optimisation methods *with the quadratic-penalty function Q*? (i.e., not for the constrained problem, just for Q). Consider the particular cases of: LP with equalities only; LP with inequalities only.
   (b) How does the quadratic-penalty method compare with other methods for LP or QP (active-set, gradient-projection, interior-point methods)?
4. What happens if the problem: has several solutions; or is unbounded; or is infeasible? (try some experiment).
5. What is the convergence rate of the *constrained* method? How can we accelerate it? Does it depend on the unconstrained optimizer? Use `convseq` on the last few iterates to investigate this question.

Optional: you may check or compare your results with the Matlab Optimization Toolbox.

Hints: debug your `Q.m` and `Oqpenalty.m` functions using the problem of `qpdriver0.m`, which is quite simple, and then move to the other, more difficult problems. Make sure you understand well the problem of `qpdriver0.m`, e.g. the number of solutions and how to converge to them. It is very helpful to plot $Q(\mathbf{x}; \mu)$ for several μ *before trying to run* `Oqpenalty.m` to find out about good $\mathbf{x}_0$ and μ_0.

[1] Note that when the objective function or the constraints are quadratic or linear you need not program them, just set up values for $\mathbf{A}$, $\mathbf{b}$ and c and use `Fquad`.

IV What you have to submit

Follow the following instructions strictly. Email me the following packed into a **single** file (`project2.tar.gz` or `project2.zip`) and with email subject `[EECS260] Project 2`:

1. Your code for the Matlab functions (`Q.m`, `Oqpenalty.m`); one driver file per problem tested (`qpdriver12_19.m`, `qpdriver16_1.m`, `qpdriver18_3.m`); and Matlab functions for the objective and constraints in those problems, if they can't be coded with e.g. `Fquad` (`F18_3f.m`, `F18_3c3.m`, possibly some more). Do not include any other functions (e.g. `fcontours.m`).

2. Your evaluation of the methods and your solutions to part I 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.