This project studies the quadratic-penalty method for constrained optimisation (chapter 17 of the book *Numerical Optimization* by Nocedal and Wright; read carefully pages 491–500). 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. 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 (`steepdesc.m`).
2. Polak-Ribière conjugate gradient (`prcg.m`).
3. Quasi-Newton BFGS (`bfgs.m`).
4. Newton-CG (`newtoncg.m`).

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

I Paper exercises

1. Consider the quadratic-penalty function $Q(\mathbf{x};\mu)$ of eq. (17.5). 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 4 unconstrained optimisation methods.
3. Consider the Lagrange multiplier estimates

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

   given in eq. (17.8) for the equality-constrained case. Guess how they may be extended to the inequality-constrained case. Hint: what happens with eq. (17.8) 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 $\|\nabla f(\mathbf{x}_k)\| < \tau_k$ for some tolerance sequence $\tau_k \to 0$. 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

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

   where $\dot{\mathbf{x}}$ is the path tangent (see p. 506 for the version of this for the log-barrier function). 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, following strictly the convention given for the input and output arguments. **Use the templates in the web page**.

1. `negpart.m`: absolute value of the negative part, i.e., $[x]^- = \max(-x, 0)$.

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

3. `qpenalty.m`: quadratic-penalty method. Follow the algorithmic framework 17.1 (p. 494).

   (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).

4. Unconstrained optimisation functions: you will use `steepdesc.m` and `newtoncg.m` (which are given as examples in the web page). Optionally, write `prcg.m` and `bfgs.m` (in the same style of input and output arguments) by modifying your code from homework #2 (`lincg.m`, `bfgs.m`).

   A particular unconstrained optimisation solver, say `newtoncg`, will be passed to `qpenalty.m` as in this example (see `qpdriver.m`):

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

   The template `qpenalty.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 `newtoncg` directly:

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

   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(:)]`.

5. Program several functions (one for f and one for each c_i) to compute the function, gradient and Hessian for exercise 18.2. Follow the same style as in `quadf`. 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 noting that there are several quadratic constraints (so you can use `quadf`).

It will be most efficient if you write the functions in the order above and test `Q` and `qpenalty` with combinations of `quadf`, visualising the results with `fcontours`. I have provided a driver file that sets up a constrained optimisation problem, solves it and displays the result in `qpdriver.m`.

III Evaluation

The objective is to evaluate your implementation `qpenalty` of the quadratic-penalty method on some test problems. You will report your results on exercise 18.2 and on any 2 problems of the following (already solved in the homework, and easily coded by calling `quadf`): 12.19, 12,20, 12,21, 16.1, 16.8, hw3–e3, hw3–e4, hw4–e2(i).

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: `steepdesc`, `prcg`, `bfgs`, `newtoncg`.

For each problem, write a different driver file[1] (based on `qpdriver.m`) that sets it up, calls `qpenalty` and collects and plots/tabulates the following information[2]:

1. Plots the contours of $Q(\mathbf{x};\mu_k)$, the constraints and the minimiser $\mathbf{x}_k$ (call `fcontours` from within `qpenalty` after each unconstrained minimisation of Q).
2. Plots the contours of f and the constraints, and the sequence of minimisers $\mathbf{x}_k$ (call `fcontours` and `plotseq` from `qpdriver.m` after `qpenalty` has finished).
3. Tabulates or plots the following information (obtained from the output arguments of `qpenalty`) as a function of μ_k: $f(\mathbf{x}_k)$; $\|\nabla f(\mathbf{x}_k)\|$; cond $\left(\nabla^2_{\mathbf{xx}} Q\right)$; value of your convergence criterion; number of iterations of the unconstrained optimisation solver.
4. For exercise 18.2, 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 `qpenalty` 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): $\mathbf{x}_0$ and $\mathbf{x}^*$ are swapped.

Here are some specific questions to answer:

1. Comment on your results in view of our theoretical understanding:
   (a) Applicability: does the particular combination work (e.g. `newtoncg` with inequality constraints)? Why or why not?
   (b) Efficiency: computational cost.
2. What can you say about the *shape* of the contours of Q around the boundary of the feasible set? (for equality constraints, and for inequality constraints).
3. Consider problems where f and all the constraints c_i are quadratic:
   (a) What can you say (in theory and given the experiments) about the performance of the 4 unconstrained optimisation methods *with the quadratic-penalty function* Q? (i.e., not for the constrained problem, just for Q).
   (b) For quadratic-programming problems (when the constraints are linear), how does the quadratic-penalty method compare with other methods for quadratic programming (active-set, gradient-projection, interior-point methods)?
4. What happens if the problem is infeasible? (try some experiment).

What you have to submit:

1. In paper, your solutions to part I and your evaluation of the methods.
2. By email, your code for the Matlab functions: the modified templates in the web page, and one driver file per problem tested.

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

[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 `quadf`.

[2] **Do not print or email me the tables or plots**, just email me the driver file so I can run it and replicate your results.