Write a Matlab program using a feasible direction method to solve linear programming problems

$$\max_{\mathbf{x}} \mathbf{b}^T \mathbf{x}$$

$$\mathbf{A}^T \mathbf{x} \geq \mathbf{c}$$

where \( \mathbf{x} \in \mathbb{R}^n \) and \( \mathbf{c} \in \mathbb{R}^m \) with \( n \leq m \). Assume a constraint qualification.

Write a Matlab function \( \mathbf{x}_{\text{opt}} = \text{lpfeasdir}(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{x}) \). The parameters to your feasible direction algorithm are \( \mathbf{A}, \mathbf{b}, \mathbf{c}, \) an initial feasible point \( \mathbf{x} \).

- Use \text{qrupdate}, \text{qrinsert}, \text{qrdelete} (instead of the \( \mathbf{B} \) and \( \mathbf{N} \) method in the notes) to update a factorization of the matrix \( \hat{\mathbf{A}} \) corresponding to the currently active constraints.

- At each iteration, \( \hat{\mathbf{A}} \) gains one row, and it may also lose one: if there is no feasible downhill direction, remove the constraint corresponding to the most negative (estimated) Lagrange multiplier.

- The next point is \( \mathbf{x} + \alpha \mathbf{p} \), where \( \mathbf{p} \) is determined from solving the system involving a column of the identity matrix, and \( \alpha \) defines the longest step that is possible without violating any of the constraints. The constraint that we hit becomes the added one.

- Stop when there is no feasible downhill direction.

- You must apply the feasible direction approach to the problem as written above, not to the dual problem.

Find one linear programming problem on which to test your algorithm.

Grading: 30 points total.

- 20 points for the efficient implementation of the algorithm as a bug-free Matlab function, with good documentation for the calling sequence and the algorithm. “Efficient” means not using an order of magnitude more computation than necessary.

- 10 points for the script that tests the algorithm.

Note. Let \( \mathbf{A} \) and \( \mathbf{B} \) be matrices, and let \( \mathbf{c} \) be a vector. Make sure you understand why the statements \( \mathbf{A} \*(\mathbf{B} \* \mathbf{c}) \) and \( \mathbf{A} \backslash (\mathbf{B} \* \mathbf{c}) \) take much less time than \( \mathbf{A} \* \mathbf{B} \* \mathbf{c} \) and \( \mathbf{A} \backslash \mathbf{B} \* \mathbf{c} \), and then use this knowledge in your programming.