EE 553 Homework 3
Solution

Fall 2023

Problem 1 (10+10+10 points)

Assume that all three of the thermal units described below are running. Find the economic dispatch schedule to serve the total demand of 450 MW. Report either computer code with results OR the corresponding algebraic solution.

Input/output curve (MBtu/hr vs. MW)	$P_{\min}(MW)$	$P_{\max}$ (MW)	Fuel cost ($/MBtu)
$225 + 8.4P_1 + 0.0025P_1^2$	45	350	0.8
$729 + 6.3P_2 + 0.0081P_2^2$	45	350	1.02
$400 + 7.5P_3 + 0.0025P_3^2$	47.5	450	0.9

Solution. The fuel cost curves are the product of the input/output curve and the fuel cost, which are

$$\begin{aligned} C_1(P_1) &= 0.002P_1^2 + 6.72P_1 + 180 \\ C_2(P_2) &= 0.008262P_2^2 + 6.426P_2 + 743.58 \\ C_3(P_3) &= 0.00225P_3^2 + 6.75P_3 + 360 \end{aligned}$$ ■

a) Use the lambda iteration method with generation limits ignored.

Solution. We are trying to find the optimal $[P_1, P_2, P_3]$ that solves

$$\min \quad \sum_{i=1}^{3} C_i(P_i) \tag{1a}$$

$$\text{s.t.} \quad \sum_{i=1}^{3} P_i = 450 \tag{1b}$$

With the lambda iteration method, we form the Lagrangian

$$\mathcal{L}(P_1, P_2, P_3, \lambda) = \sum_{i=1}^{3} C_i(P_i) + \lambda(450 - \sum_{i=1}^{3} P_i).$$

The first order necessary condition is

$$\frac{\partial \mathcal{L}}{\partial P_i} = 0, \quad i = 1, 2, 3, \tag{2a}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = 0, \tag{2b}$$

from which we have

$$\begin{aligned}
0.004P_1 + 6.72 = \lambda \quad &\Longrightarrow \quad P_1 = \frac{\lambda - 6.72}{0.004} \\
0.016524P_2 + 6.426 = \lambda \quad &\Longrightarrow \quad P_2 = \frac{\lambda - 6.426}{0.016524} \\
0.0045P_3 + 6.75 = \lambda \quad &\Longrightarrow \quad P_3 = \frac{\lambda - 6.75}{0.0045} \\
P_1 + P_2 + P_3 = 450 &
\end{aligned}$$

Plugging P_i as functions of λ into (2b), we have

$$\frac{\lambda - 6.72}{0.004} + \frac{\lambda - 6.426}{0.016524} + \frac{\lambda - 6.75}{0.0045} = 450.$$

This is a linear equation in λ so the solution can be readily solved for. However, using lambda iteration implemented by the MATLAB code below, we obtain

$$\lambda^* = 7.5439.$$

So the optimal solution is

$$\boxed{P_1^* = \frac{\lambda^* - 6.72}{0.004} = 205.97,}$$

$$\boxed{P_2^* = \frac{\lambda^* - 6.426}{0.016524} = 67.65,}$$

$$\boxed{P_3^* = \frac{\lambda^* - 6.75}{0.0045} = 176.42.}$$

The optimal cost is therefore 4485.62.

```
eps = 1e-4;

f = @(x) (x - 6.72)/0.004 + (x - 6.426)/0.016524 + (x - 6.75)/0.0045;

lambda_h = 10;
lambda_l = 5;
iter = 0;

while lambda_h - lambda_l >= eps
    lambda_m = 0.5 * (lambda_h + lambda_l);
    if f(lambda_m) > 450
        lambda_h = lambda_m;
    else
        lambda_l = lambda_m;
    end
    iter = iter + 1;
end

iter
lambda_m
```

The output is

```
    iter = 16
    lambda_m = 7.5439
```

■

b) Use the steepest descent method (Lecture 11, slide 13) with generation limits ignored. You are free to choose any step size rule.

Solution. Steepest descent is used to solve an unconstrained optimization problem. Since problem (1) has an equality constraint, we need to eliminate the constraint first. This can be done by solving for one of the P_i's in (1b), and plugging it in the objective function (1a). For example, we can solve for P_1. In this case problem (1) becomes

$$\min_{P_2,P_3} \quad C_1(450 - P_2 - P_3) + C_2(P_2) + C_3(P_3). \tag{3}$$

It is easy to see problems (1) and (3) are equivalent.

In steepest descent, we update $[P_2, P_3]$ iteratively along the negative gradient direction of the objective function in (3) using

$$\begin{bmatrix} P_2^{(k+1)} \\ P_3^{(k+1)} \end{bmatrix} = \begin{bmatrix} P_2^{(k)} \\ P_3^{(k)} \end{bmatrix} - \mu \begin{bmatrix} 0.0205P_2 + 0.004P_3 - 2.094 \\ 0.004P_2 + 0.0085P_3 - 1.77 \end{bmatrix}$$

where μ is the step size. The following MATLAB code implements steepest descent algorithm to solve (3) with a constant step size of $\mu = 10$ with initial value $[P_2^{(0)}, P_3^{(0)}] = [225, 225]$.

```
eps = 1e-4;
mu = 10;

% gradient
grad = @(x) [0.0205*x(1) + 0.004*x(2) - 2.094; ...
             0.004*x(1) + 0.0085*x(2) - 1.77];

x = [225; 225];
err = eps + 1;
iter = 0;

while err > eps
    xnew = x - mu * grad(x);
    err = norm(xnew - x);
    x = xnew;
    iter = iter+1;
end

x
grad(x)
iter
```

The output is

```
x = 67.7342
    176.3614
ans = 1.0e-05 *
    -0.2650
     0.8757
iter = 88
```

The optimal solution is $\underline{P^* = [205.91, 67.73, 176.36]}$. The optimal cost is $\underline{4485.64}$. ■

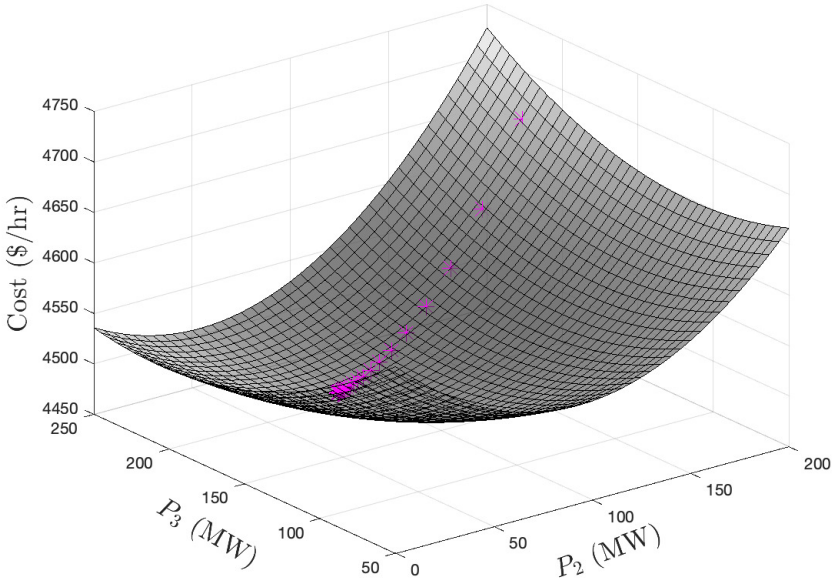

Figure 1: Visualization of the steepest descent iteration in Problem 1(b).

c) Use the lambda iteration method with generation limits enforced.

Solution. The solution method differs from 1(a) in how `f` is evaluated in the third line of the MATLAB code. The line needs to be changed to

```
pmin = [45; 45; 47.5];
pmax = [350; 350; 450];

f = @(x) min(max((x - 6.72)/0.004, pmin(1)), pmax(1)) + ...
         min(max((x - 6.426)/0.016524, pmin(2)), pmax(2)) + ...
         min(max((x - 6.75)/0.0045, pmin(3)), pmax(3));
```

The solution stays the same. ■

Problem 2 (20 points)

Design a program to repeat the Newton OPF problem in Lecture 13, slide 23, but now with impedance of the line set to $0.06 + j0.08$ p.u. Solve the problem to optimality with tolerance $\epsilon = 10^{-5}$.

Solution. We can formulate the OPF problem as

$$\begin{aligned} \min \quad & 200P_{G1}^2 + 2000P_{G1} + 100P_{G2}^2 + 2200P_{G2} \\ \text{s.t.} \quad & 6 - 6\cos\theta_2 - 8\sin\theta_2 - P_{G1} = 0 \\ & 11 - 6\cos\theta_2 + 8\sin\theta_2 - P_{G2} = 0 \end{aligned}$$

The Newton OPF uses the method of Lagrange multiplier to solve the above problem. The Lagrangian is

$$\begin{aligned} \mathcal{L}(\theta_2, P_{G1}, P_{G2}, \lambda) = 200P_{G1}^2 + 2000P_{G1} + 100P_{G2}^2 + 2200P_{G2} + \\ \lambda_1(6 - 6\cos\theta_2 - 8\sin\theta_2 - P_{G1}) + \lambda_2(11 - 6\cos\theta_2 + 8\sin\theta_2 - P_{G2}) \end{aligned}$$

The first order necessary condition of the minimum is

$$\nabla\mathcal{L} = \begin{bmatrix} \partial\mathcal{L}/\partial\theta_2 \\ \partial\mathcal{L}/\partial P_{G1} \\ \partial\mathcal{L}/\partial P_{G2} \\ \partial\mathcal{L}/\partial\lambda_1 \\ \partial\mathcal{L}/\partial\lambda_2 \end{bmatrix} = \begin{bmatrix} 6(\lambda_1+\lambda_2)\sin\theta_2 - 8(\lambda_1-\lambda_2)\cos\theta_2 \\ 2000 + 400P_{G1} - \lambda_1 \\ 2200 + 200P_{G2} - \lambda_2 \\ 6 - 6\cos\theta_2 - 8\sin\theta_2 - P_{G1} \\ 11 - 6\cos\theta_2 + 8\sin\theta_2 - P_{G2} \end{bmatrix} = 0$$

To find the solution to the equations, we apply Newton's method. The Hessian matrix is

$$\begin{aligned} \nabla^2\mathcal{L} &= \begin{bmatrix} \frac{\partial^2\mathcal{L}}{\partial\theta_2^2} & \frac{\partial^2\mathcal{L}}{\partial\theta_2\partial P_{G1}} & \frac{\partial^2\mathcal{L}}{\partial\theta_2\partial P_{G2}} & \frac{\partial^2\mathcal{L}}{\partial\theta_2\partial\lambda_1} & \frac{\partial^2\mathcal{L}}{\partial\theta_2\partial\lambda_2} \\ \frac{\partial^2\mathcal{L}}{\partial P_{G1}\partial\theta_2} & \frac{\partial^2\mathcal{L}}{\partial P_{G1}^2} & \frac{\partial^2\mathcal{L}}{\partial P_{G1}\partial P_{G2}} & \frac{\partial^2\mathcal{L}}{\partial P_{G1}\partial\lambda_1} & \frac{\partial^2\mathcal{L}}{\partial P_{G1}\partial\lambda_2} \\ \frac{\partial^2\mathcal{L}}{\partial P_{G2}\partial\theta_2} & \frac{\partial^2\mathcal{L}}{\partial P_{G2}\partial P_{G1}} & \frac{\partial^2\mathcal{L}}{\partial P_{G2}^2} & \frac{\partial^2\mathcal{L}}{\partial P_{G2}\partial\lambda_1} & \frac{\partial^2\mathcal{L}}{\partial P_{G2}\partial\lambda_2} \\ \frac{\partial^2\mathcal{L}}{\partial\lambda_1\partial\theta_2} & \frac{\partial^2\mathcal{L}}{\partial\lambda_1\partial P_{G1}} & \frac{\partial^2\mathcal{L}}{\partial\lambda_1\partial P_{G2}} & \frac{\partial^2\mathcal{L}}{\partial\lambda_1^2} & \frac{\partial^2\mathcal{L}}{\partial\lambda_1\partial\lambda_2} \\ \frac{\partial^2\mathcal{L}}{\partial\lambda_2\partial\theta_2} & \frac{\partial^2\mathcal{L}}{\partial\lambda_2\partial P_{G1}} & \frac{\partial^2\mathcal{L}}{\partial\lambda_2\partial P_{G2}} & \frac{\partial^2\mathcal{L}}{\partial\lambda_2\partial\lambda_1} & \frac{\partial^2\mathcal{L}}{\partial\lambda_2^2} \end{bmatrix} \\ &= \begin{bmatrix} 6(\lambda_1+\lambda_2)\cos\theta_2 + 8(\lambda_1-\lambda_2)\sin\theta_2 & 0 & 0 & 6\sin\theta_2 - 8\cos\theta_2 & 6\sin\theta_2 + 8\cos\theta_2 \\ 0 & 400 & 0 & -1 & 0 \\ 0 & 0 & 200 & 0 & -1 \\ 6\sin\theta_2 - 8\cos\theta_2 & -1 & 0 & 0 & 0 \\ 6\sin\theta_2 + 8\cos\theta_2 & 0 & -1 & 0 & 0 \end{bmatrix} \end{aligned}$$

and the Newton step is

$$\begin{bmatrix} \theta_2^{(k+1)} \\ P_{G1}^{(k+1)} \\ P_{G2}^{(k+1)} \\ \lambda_1^{(k+1)} \\ \lambda_2^{(k+1)} \end{bmatrix} = \begin{bmatrix} \theta_2^{(k)} \\ P_{G1}^{(k)} \\ P_{G2}^{(k)} \\ \lambda_1^{(k)} \\ \lambda_2^{(k)} \end{bmatrix} - \nabla^{-2}\mathcal{L}(\theta_2^{(k)}, P_{G1}^{(k)}, P_{G2}^{(k)}, \lambda^{(k)})\nabla\mathcal{L}(\theta_2^{(k)}, P_{G1}^{(k)}, P_{G2}^{(k)}, \lambda^{(k)}).$$

The iterative process stops when $\left\|\nabla\mathcal{L}(\theta_2^{(k)}, P_{G1}^{(k)}, P_{G2}^{(k)}, \lambda^{(k)})\right\|_2 \leq \epsilon$ for some given $\epsilon > 0$.

The MATLAB code is

```
eps = 1e-5;
x = [0; 0; 5; 0; 0];
f = @(x) [6*(x(4)+x(5))*sin(x(1)) - 8*(x(4)-x(5))*cos(x(1))
          2000 + 400*x(2) - x(4)
          2200 + 200*x(3) - x(5)
          6 - 6*cos(x(1)) - 8*sin(x(1)) - x(2)
          11 - 6*cos(x(1)) + 8*sin(x(1)) - x(3)];
J = @(x) [6*(x(4)+x(5))*cos(x(1)) + 8*(x(4)+x(5))*sin(x(1)), 0, 0 , ...
                6*sin(x(1)) - 8*cos(x(1)), 6*sin(x(1)) + 8*cos(x(1));
          0, 400, 0 -1, 0;
          0, 0, 200, 0, -1;
          6*sin(x(1)) - 8*cos(x(1)), -1, 0, 0, 0;
          6*sin(x(1)) + 8*cos(x(1)), 0, -1, 0, 0];
iter = 0;

s1 = "Iteration %2i: mismatch = %11.6f, theta2 = %7.4f, P1 = %7.4f, ";
s2 = "P2 = %7.4f, lambda1 = %7.4f, lambda2 = %7.4f \n";

while norm(f(x)) > eps
    % Newton-Raphson iteration
    xnew = x - J(x) \ f(x);

    iter = iter + 1;
    fprintf(s1+s2, iter, norm(f(xnew)), xnew(1), xnew(2), xnew(3), ...
        xnew(4), xnew(5));

    % update variables
    x = xnew;
end
```

The program output is

```
Iteration  1: mismatch = 8312.773035, theta2 = -0.2500, P1 =  2.0000, P2 =
3.0000, lambda1 = 2800.0000, lambda2 = 2800.0000
Iteration  2: mismatch = 1524.111011, theta2 = -0.1134, P1 =  0.9044, P2 =
4.0632, lambda1 = 2361.7447, lambda2 = 3012.6421
Iteration  3: mismatch =  127.916993, theta2 = -0.1355, P1 =  1.1343, P2 =
3.9728, lambda1 = 2453.7036, lambda2 = 2994.5597
Iteration  4: mismatch =   12.485426, theta2 = -0.1335, P1 =  1.1186, P2 =
3.9882, lambda1 = 2447.4372, lambda2 = 2997.6460
Iteration  5: mismatch =    1.185236, theta2 = -0.1337, P1 =  1.1202, P2 =
3.9869, lambda1 = 2448.0886, lambda2 = 2997.3840
Iteration  6: mismatch =    0.112770, theta2 = -0.1337, P1 =  1.1201, P2 =
3.9870, lambda1 = 2448.0271, lambda2 = 2997.4092
Iteration  7: mismatch =    0.010727, theta2 = -0.1337, P1 =  1.1201, P2 =
3.9870, lambda1 = 2448.0329, lambda2 = 2997.4068
Iteration  8: mismatch =    0.001020, theta2 = -0.1337, P1 =  1.1201, P2 =
3.9870, lambda1 = 2448.0324, lambda2 = 2997.4070
Iteration  9: mismatch =    0.000097, theta2 = -0.1337, P1 =  1.1201, P2 =
3.9870, lambda1 = 2448.0324, lambda2 = 2997.4070
Iteration 10: mismatch =    0.000009, theta2 = -0.1337, P1 =  1.1201, P2 =
3.9870, lambda1 = 2448.0324, lambda2 = 2997.4070
```

The optimal solution is $P_{G1}^* = 1.1201$ p.u. and $P_{G2}^* = 3.9870$ p.u., the optimal cost is 12,852 \$/hr. ■

Problem 3 (20+15+15 points)

A three-generator system is serving a total load of 5 p.u.. The fuel cost curves of the generators are

$$C_1(P_{g1}) = P_{g1}^2$$
$$C_2(P_{g2}) = 3P_{g2}^2$$
$$C_3(P_{g3}) = 4P_{g3}^2$$

Generator 1 can generate up to 3 p.u., generator 2 up to 2 p.u., and due to power transfer capability constraint, generator 2 and 3 can only generate up to 5 p.u. in total. To determine the minimum cost to serve the load, the following problem is formulated (where, for notational brevity, we use x_1, x_2, x_3 in place of P_{g1}, P_{g2}, and P_{g3}):

$$\begin{aligned} \min \quad & x_1^2 + 3x_2^2 + 4x_3^2 \\ \text{s.t.} \quad & x_1 + x_2 + x_3 = 5 \\ & x_1 \leq 3 \\ & x_2 \leq 2 \\ & x_2 + x_3 \leq 5 \end{aligned}$$

In the following, use different algorithms to solve this problem. Report either computer code with results OR the corresponding algebraic solution. For all Newton iterations, use ℓ_2-norm with tolerance $\epsilon = 10^{-5}$ as the stopping criterion.

a) Lagrange multiplier method with equality constraint. *Hint: you can always rewrite an inequality constraint $a^\top x \leq b$ as a equality constraint $a^\top x + s^2 = b$ with slack variable s.*

Solution. Let's introduce three slack variables and rewrite the optimization problem as

$$\begin{aligned} \min \quad & x_1^2 + 3x_2^2 + 4x_3^2 \\ \text{s.t.} \quad & x_1 + x_2 + x_3 = 5 \\ & x_1 + s_1^2 = 3 \\ & x_2 + s_2^2 = 2 \\ & x_2 + x_3 + s_3^2 = 5 \end{aligned}$$

The Lagrangian is

$$\begin{aligned} \mathcal{L}(x) = x_1^2 + 3x_2^2 + 4x_3^2 + \lambda_1(x_1 + x_2 + x_3 - 5) + \\ \lambda_2(x_1 + s_1^2 - 3) + \lambda_3(x_2 + s_2^2 - 2) + \lambda_4(x_2 + x_3 + s_3^2 - 5), \end{aligned}$$

where $x = [x_1, x_2, x_3, s_1, s_2, s_3, \lambda_1, \lambda_2, \lambda_3, \lambda_4]^\top$. The first order necessary condition of the minimum is

$$\nabla \mathcal{L}(x) = \begin{bmatrix} \partial\mathcal{L}/\partial x_1 \\ \partial\mathcal{L}/\partial x_2 \\ \partial\mathcal{L}/\partial x_3 \\ \partial\mathcal{L}/\partial s_1 \\ \partial\mathcal{L}/\partial s_2 \\ \partial\mathcal{L}/\partial s_3 \\ \partial\mathcal{L}/\partial \lambda_1 \\ \partial\mathcal{L}/\partial \lambda_2 \\ \partial\mathcal{L}/\partial \lambda_3 \\ \partial\mathcal{L}/\partial \lambda_4 \end{bmatrix} = \begin{bmatrix} 2x_1 + \lambda_1 + \lambda_2 \\ 6x_2 + \lambda_1 + \lambda_3 + \lambda_4 \\ 8x_3 + \lambda_1 + \lambda_4 \\ 2\lambda_2 s_1 \\ 2\lambda_3 s_2 \\ 2\lambda_4 s_3 \\ x_1 + x_2 + x_3 - 5 \\ x_1 + s_1^2 - 3 \\ x_2 + s_2^2 - 2 \\ x_2 + x_3 + s_3^2 - 5 \end{bmatrix} = 0$$

To find the solution to the equations, we apply Newton's method. The Hessian matrix is

$$\nabla^2\mathcal{L} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 8 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2\lambda_2 & 0 & 0 & 0 & 2s_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\lambda_3 & 0 & 0 & 0 & 2s_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2\lambda_4 & 0 & 0 & 0 & 2s_3 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2s_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2s_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 2s_3 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and the Newton step is

$$x^{(k+1)} = x^{(k)} - \nabla^{-2}\mathcal{L}(x^{(k)})\nabla\mathcal{L}(x^{(k)}).$$

The iterative process stops when $\left\|\nabla\mathcal{L}(x^{(k)})\right\|_2 \leq \epsilon$ for some given $\epsilon > 0$.

The MATLAB code is

```
eps = 1e-5;

% initial guess of x
x1_0 = 1;
x2_0 = 1;
x3_0 = 5 - x1_0 - x2_0;
s1_0 = sqrt(3 - x1_0);
s2_0 = sqrt(2 - x2_0);
s3_0 = sqrt(5 - x2_0 - x3_0);
x = [x1_0; x2_0; x3_0; s1_0; s2_0; s3_0; 1; 1; 1; 1];

f = @(x) [2*x(1) + x(7) + x(8)
          6*x(2) + x(7) + x(9) + x(10)
          8*x(3) + x(7) + x(10)
          2*x(8)*x(4)
          2*x(9)*x(5)
          2*x(10)*x(6)
          x(1) + x(2) + x(3) - 5
          x(1) + x(4)^2 - 3
          x(2) + x(5)^2 - 2
          x(2) + x(3) + x(6)^2 - 5];

J = @(x) [2, 0, 0, 0,       0,       0,        1, 1,       0,       0
          0, 6, 0, 0,       0,       0,        1, 0,       1,       1
          0, 0, 8, 0,       0,       0,        1, 0,       0,       1
          0, 0, 0, 2*x(8), 0,       0,        0, 2*x(4), 0,       0
          0, 0, 0, 0,       2*x(9), 0,        0, 0,       2*x(5), 0
          0, 0, 0, 0,       0,       2*x(10), 0, 0,       0,       2*x(6)
          1, 1, 1, 0,       0,       0,        0, 0,       0,       0
          1, 0, 0, 2*x(4), 0,       0,        0, 0,       0,       0
          0, 1, 0, 0,       2*x(5), 0,        0, 0,       0,       0
          0, 1, 1, 0,       0,       2*x(6),  0, 0,       0,       0];

while norm(f(x)) > eps
    x = x - J(x) \ f(x);
end
```

Figure 2 shows the initial guesses that converges to the global optimal solution $\underline{x^* = [3, 8/7, 6/7]^\top}$.

Note that not all initial guesses lead to the global optimal solution. Newton's algorithm is not to blame, since for the other solutions, the first order necessary conditions are always met when the

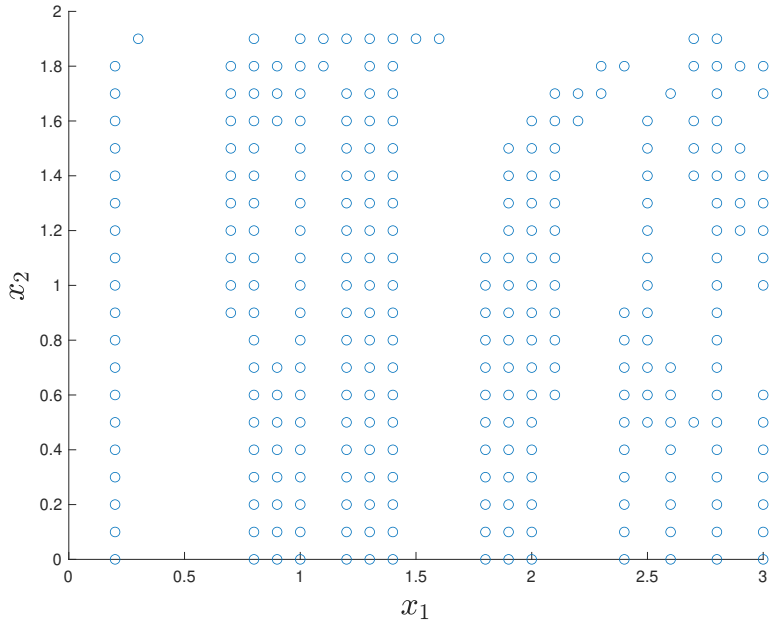

Figure 2: Initial guess of x_1, x_2 that converge to the global optimal solution.

algorithm converges. The problem is that the first order necessary conditions are not sufficient for this problem. The original inequality-constrained problem is actually convex. However, the introduction of the slack variables destroys the convexity structure of the problem. So, although the equality-constrained problem is an equivalent reformulation, the problem structure is changed from convex to nonconvex, and the necessary condition is no longer sufficient. This problem illustrates the disadvantage of the reformulation method.

The KKT condition, which is an extension of the first order necessary condition for inequality-constrained problems, resolves the issue and guarantees sufficiency of global optimality for inequality-constrained problems (under mild technical condition). Our textbook covers KKT conditions, please check it if you are interested. ■

b) Active set method (Lecture 13, slide 35). In the second step in the while loop, release the inequality constraint as long as the corresponding optimal Lagrange multiplier is negative.

Solution. First, we solve the problem assuming no inequality constraints are binding. The problem is

$$\begin{aligned} \min \quad & x_1^2 + 3x_2^2 + 4x_3^2 && \text{(4a)} \\ \text{s.t.} \quad & x_1 + x_2 + x_3 = 5 && \text{(4b)} \end{aligned}$$

The optimal solution is $x^* = [60/19, 20/19, 15/19]^\top$.

We see that the inequality constraint $x_1 \leq 3$ is violated, so it is added as an equality constraint in (5). The optimization problem becomes

$$\min \quad x_1^2 + 3x_2^2 + 4x_3^2 \qquad \text{(5a)}$$

$$\text{s.t.} \quad x_1 + x_2 + x_3 = 5 \tag{5b}$$
$$x_1 = 3 \tag{5c}$$

The corresponding Lagrangian is

$$\mathcal{L} = x_1^2 + 3x_2^2 + 4x_3^2 + \lambda_1(x_1 + x_2 + x_3 - 5) + \lambda_2(x_1 - 3).$$

$\lambda_2^* = 6/7 > 0$ so the constraint should not be released. No inequality constraints are violated. The active set method terminates and the optimal solution is $\underline{x^* = [3, 8/7, 6/7]^\top}$. ■

c) Penalty function method (Lecture 14, slide 24). Start with $c_1 = 1$ and gradually increase it in each iteration with $c_{k+1} = \beta c_k$ with $\beta \in [4, 10]$. Experiment with different β and use the optimal $x^{(k)}$ from the previous iteration to avoid ill-conditioning of the Hessian matrix. Before you start, answer the following questions:

- Does the gradient of the penalty function exist for all values of c and x?
- Does the Hessian of the penalty function exist for all values of c and x?

Solution. Given the penalty factor $c_k > 0$, the penalty function is

$$f_{c_k}(x) = x_1^2 + 3x_2^2 + 4x_3^2 + c_k \left((x_1 + x_2 + x_3 - 5)^2 + p(x_1 - 3) + p(x_2 - 2) + p(x_2 + x_3 - 5)\right)$$

where $p(x) := (\max\{0, x\})^2$. The gradient of the penalty function exists for all values of c and x since $\mathrm{d}p(x)/\mathrm{d}x = \max\{0, 2x\}$. But the Hessian does not exist when $x_1 = 3$ or $x_2 = 2$ or $x_2 - x_3 = 5$ since the derivative of the gradient at these points are not defined.

For each iteration, we need to solve the unconstrained optimization problem

$$\min \quad f_{c_k}(x) \tag{6}$$

This can be solved using a number of algorithms, including steepest descent and Newton's method. Both methods are fine, but just be careful that if you choose Newton's method, the Hessian matrix needs to be approximated at points where it is not defined.

As an example, we use steepest descent algorithm to solve the minimization problem. Each iteration is

$$\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \end{bmatrix} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{bmatrix} - \mu_{c_k} \nabla f_{c_k}(x)$$

where

$$\nabla f_{c_k}(x) =$$
$$\begin{bmatrix} 2x_1^{(k)} + 2c_k \left(x_1^{(k)} + x_2^{(k)} + x_3^{(k)} - 5\right) + c_k \max\left\{0, 2x_1^{(k)} - 6\right\} \\ 6x_2^{(k)} + c_k \left(2x_1^{(k)} + 2x_2^{(k)} + 2x_3^{(k)} - 10 + \max\left\{0, 2x_2^{(k)} - 4\right\} + \max\left\{0, 2x_2^{(k)} + 2x_3^{(k)} - 10\right\}\right) \\ 8x_3^{(k)} + c_k \left(2x_1^{(k)} + 2x_2^{(k)} + 2x_3^{(k)} - 10 + \max\left\{0, 2x_2^{(k)} + 2x_3^{(k)} - 10\right\}\right) \end{bmatrix}$$

After the algorithm converges, the optimal solution ${x^{(k)}}^*$ is recorded and $\left\|{x^{(k)}}^* - {x^{(k-1)}}^*\right\|$ is evaluated. The algorithm terminates when it is sufficiently small. Otherwise, problem (6) is solved again with $c_{k+1} > c_k$.

The overall algorithm implementation in MATLAB is shown below:

```
    eps = 1e-5;
    mu = 0.1;

    % gradient
    grad = @(x,c) [2*x(1) + 2*c*(x(1)+x(2)+x(3)-5) + c*max(0, 2*x(1)-6)
                   6*x(2) + 2*c*(x(1)+x(2)+x(3)-5) + c*max(0, 2*x(2)-4) + ...
                                                     c*max(0, 2*x(2)+2*x(3)-10)
                   8*x(3) + 2*c*(x(1)+x(2)+x(3)-5) + c*max(0, 2*x(2)+2*x(3)-10)];

    x = [3; 2; 0];
    err = eps + 1;
    beta = 10;
    c = 1 / beta;
    iter = 0;

    s = "Iteration %i: mismatch = %8.6f, x1 = %6.4f, x2 = %6.4f, x3 = %6.4f \n";

    while err > eps
        c = beta * c;
        xnew = descent(x, grad, c);
        iter = iter + 1;
        err = norm(xnew - x);

        fprintf(s, iter, norm(err), xnew(1), xnew(2), xnew(3));

        x = xnew;
    end

    function x = descent(x, grad, c)
    % steepest descent algorithm
        eps = 1e-10;
        mu = 1 / (40*c); % adjust step size to avoid divergence
        err = eps + 1;

        while err > eps
            xnew = x - mu * grad(x,c);
            err = norm(xnew - x);
            x = xnew;
        end
    end
```

The program output is

```
    Iteration 1: mismatch = 1.789669, x1 = 1.9355, x2 = 0.6452, x3 = 0.4839
    Iteration 2: mismatch = 1.121048, x1 = 2.9703, x2 = 0.9901, x3 = 0.7426
    Iteration 3: mismatch = 0.167678, x1 = 3.0036, x2 = 1.1216, x3 = 0.8412
    Iteration 4: mismatch = 0.024079, x1 = 3.0004, x2 = 1.1407, x3 = 0.8555
    Iteration 5: mismatch = 0.002498, x1 = 3.0000, x2 = 1.1426, x3 = 0.8570
    Iteration 6: mismatch = 0.000248, x1 = 3.0000, x2 = 1.1428, x3 = 0.8572
    Iteration 7: mismatch = 0.000025, x1 = 3.0000, x2 = 1.1428, x3 = 0.8572
    Iteration 8: mismatch = 0.000002, x1 = 3.0000, x2 = 1.1428, x3 = 0.8572
```

■