MAE 5930 - Optimization

Fall 2019

Homework 7

Jared Hansen

Due: 11:59 PM, Tuesday November 26, 2019

A-number: `A01439768`

e-mail: `jrdhansen@gmail.com`

Purpose: the problems assigned help develop your ability to

- solve optimization problems using optimality conditions.
- formulate and solve larger problems in MATLAB.

NOTE: please write or type your formulations clearly so that a reader can understand what you are doing. You are welcome to use the equivalent functions in Python.

1. Solve the following problem numerically and analytically using `fmincon`.

$$\begin{array}{ll} \text{minimize} & x_1^2 + x_2^2 \\ \text{subject to} & x_1 + x_2 - 2 \leq 0 \\ & x_1^2 - x_2 - 4 \leq 0 \end{array}$$

```
%============================================================================
%== PROBLEM 1
%============================================================================

% Initial guess (we know x* = [0,0], so we pick a close point.)
x0 = [0.5; 0.3];

% Solve the problem using fmincon.
[x,f] = fmincon(@obj,x0,[],[],[],[],[],[],@con)

% Problem 1 Objective function
function J = obj(x)
x1 = x(1);
x2 = x(2);
J  = (x1)^2 + (x2)^2;
end
% Problem 1 Constraint functions
function [cin,ceq] = con(x)
x1 = x(1);
x2 = x(2);
cin = x1 + x2 - 2;
cin = (x1)^2 - x2 - 4;
ceq = [];
end
```

Figure 1: Here is my MATLAB code that uses `fmincon` to solve the problem.

```
<stopping criteria details>

x =

   1.0e-06 *

    0.0247
    0.2348


f =

   5.5761e-14
```

Figure 2: Output for the code gets very very close to the correct answer of $x^* = [x_1 = 0, x_2 = 0]^T$. As the initial guess x_0 was made closer and closer to (0,0) the `fmincon` function got closer and closer to (0,0). I believe that the output is giving the answer $(x_1 = 0.0247 \times 10.0^{-6}, x_2 = 0.2348 \times 10.0^{-6}) \approx (x_1 = 0, x_2 = 0)$

- First, let's do some thinking that will help us see if an answer exists & what it might be: we know $x_1^2 \geq 0$ and $x_2^2 \geq 0$ $\Rightarrow$ the lowest $x_1^2 + x_2^2$ can be (unconstrained) is $0+0=0$, achieved by $(x_1=0, x_2=0.)$
- Is $(x_1=0, x_2=0)$ feasible? $0+0-2 \leq 0$ and $0^2-0-4 \leq 0$, so yes, it is.
- Therefore, our process should lead us to $(x_1^*=0$ and $x_2^*=0)$.
- Also, note that this is a convex program. $\nabla_x f = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix} \rightarrow \nabla_x^2 f = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$ which is PD $\Rightarrow$ objective is convex. We can see $g_1(x)$ is linear, and $\nabla_x^2 g_2 = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$ which is PSD $\Rightarrow g_2(x)$ is convex. $h(x)=0$ is trivially affine.
- Since this is a convex program, if we find a solution candidate such that the optimality conditions system is satisfied, these are also sufficient (for CP's) so it is guaranteed to be a global minimum.
- $L(x) = \lambda_0 x_1^2 + \lambda_0 x_2^2 + \lambda_1(x_1+x_2-2) + \lambda_2(x_1^2 - x_2 - 4)$
- $\boxed{\nabla_x L \overset{set}{=} 0} \rightarrow$ $\partial L/\partial x_1 = 2\lambda_0 x_1 + \lambda_1 + 2\lambda_2 x_1 \overset{set}{=} 0$ $\boxed{eq. 1}$

  $\partial L/\partial x_2 = 2\lambda_0 x_2 + \lambda_1 - \lambda_2 \overset{set}{=} 0$ $\boxed{eq. 2}$
- Complimentarity: $\lambda_1(x_1+x_2-2)=0$ and $\lambda_2(x_1^2-x_2-4)=0$
- CASE 1: $\lambda_0 = 0$
  - $\boxed{eq 1}$ becomes $0 + \lambda_1 + 2\lambda_2 x_1 = 0 \Rightarrow \lambda_1 = -2\lambda_2 x_1$
  - $\boxed{eq 2}$ becomes $0 + \lambda_1 - \lambda_2 = 0 \Rightarrow \lambda_1 = \lambda_2$
  - Thus $\boxed{eq 1}$ now becomes $\lambda_1 = -2\lambda_1 x_1 \rightarrow x_1 = -1/2$
  - Now solve for x_2: $\lambda_1(-1/2 + x_2 - 2) = 0$ and $\lambda_1((-1/2)^2 - x_2 - 4) = 0$

    $\Rightarrow -5/2 + x_2 = -x_2 - 15/4 \Rightarrow x_2 = -5/8$
  - To satisfy compl. cond's, since $(x_1+x_2-2) \neq 0 \Rightarrow \lambda_1 = 0$ and $(x_1^2-x_2-4) \neq 0 \Rightarrow \lambda_2 = 0$
  - This can't work since it requires $\lambda_0 = \lambda_1 = \lambda_2 = 0$ which violates the non-triviality condition. Therefore, if $\exists$ a solution, $\lambda_0 = 1$.

- CASE 2: $\lambda_0 = 1$ with $\lambda_1 = 0$

- $\boxed{eq1}$ becomes $2x_1 + 2\lambda_2 x_1 = 0 \longrightarrow 2x_1(1+\lambda_2) = 0 \Rightarrow x_1 = 0$ or $\lambda_2 = -1$, but since $\lambda \geq \underset{\sim}{0} \Rightarrow \lambda_2 \neq -1 \Rightarrow x_1 = 0$ here.

- $\boxed{eq2}$ becomes $2x_2 - \lambda_2 = 0 \Rightarrow \lambda_2 = 2x_2$ or $x_2 = \lambda_2/2$

- To satisfy complim. conds. $\lambda_2(x_1^2 - x_2 - 4) = 0 \longrightarrow$ sub in $\lambda_2 = 2x_2$ and $x_1 = 0$, giving $2x_2(-x_2 - 4) = 0 \Rightarrow$ either $x_2 = 0$ or $x_2 = -4$. If $x_2 = 0 \Rightarrow \lambda_2 = 0$, if $x_2 = -4 \Rightarrow \lambda_2 = -8$ which isn't allowed due to $\lambda \geq 0$ constraint.

- Therefore we have a candidate minimizer of:

$x_1 = 0,\ x_2 = 0,\ \lambda_0 = 1,\ \lambda_1 = 0,\ \lambda_2 = 0$

- Since we showed above that this is a convex program, we know that a candidate that satisfies the optimality condition system is a global minimizer. Also, we showed that our intuition is correct, verifying that $[x_1 = 0, x_2 = 0]$ with $\lambda_0 = 1$ and $\lambda^T = [\lambda_1 = 0, \lambda_2 = 0]$ satisfies the opt. conditions and achieves the unconstrained optimal objective value of 0.

Since $(0,0)$ is a unique point & uniquely achieves $f(x) = 0$ & satisfies the conditions, we can say that $\begin{bmatrix} x_1^* = 0 \\ x_2^* = 0 \end{bmatrix}$ globally minimizes

$f(x) = x_1^2 + x_2^2$ for the constraints $g_1(x) \leq 0$ and $g_2(x) \leq 0$.

2. Solve the following problem numerically and analytically using `fmincon`.

$$\begin{array}{ll} \text{minimize} & x_1^2 + x_2^2 \\ \text{subject to} & x_1 - 10 \leq 0 \\ & x_1 - x_2^2 - 4 \geq 0 \end{array}$$

```
%========================================================================
%== PROBLEM 2
%========================================================================

% Initial guess (we know x* = [4,0], so we pick a close point.)
x0 = [4.2; 0.3];

% Solve the problem using fmincon.
[x,f] = fmincon(@obj,x0,[],[],[],[],[],[],@con)

% Problem 2 Objective function
function J = obj(x)
x1 = x(1);
x2 = x(2);
J  = (x1)^2 + (x2)^2;
end
% Problem 2 Constraint functions
function [cin,ceq] = con(x)
x1 = x(1);
x2 = x(2);
cin = x1 -10;
cin = -x1 + (x2)^2 +4;
ceq = [];
end
```

Figure 3: Here is my MATLAB code that uses `fmincon` to solve the problem.

```
<stopping criteria details>

x =

    4.0000
    0.0000


f =

   16.0000
```

Figure 4: Output for the code achieves exactly the correct answer of $x^* = [x_1 = 4, x_2 = 0]^T$. As with problem 1, if x_0 isn't close enough to x^* `fmincon` won't quite converge to (4,0), but a guess sufficiently close to (4,0) will converge almost exactly.

- First let's do some analysis of the problem. Is it convex? Yes, it is.
  = We know from problem #1 that $f(x) = x_1^2 + x_2^2$ is convex.
  = $g_1(x) = x_1 - 10$ is linear and thus convex, $g_2(x) = -x_1 + x_2^2 + 4$ has $\nabla^2 g_2(x) = \begin{bmatrix} 0 & 0 \\ 0 & 2 \end{bmatrix}$ which is PSD making $g_2(x)$ convex also. So the inequality constraints are convex.
  = $h(x) = 0$ is trivially affine.
  = Thus, if $\exists$ a candidate that satisfies optimality conditions it is a guaranteed global min.
- $L(x) = \lambda_0 x_1^2 + \lambda_0 x_2^2 + \lambda_1(x_1 - 10) + \lambda_2(-x_1 + x_2^2 + 4)$
- $\partial L/\partial x_1 = 2\lambda_0 x_1 + \lambda_1 - \lambda_2 \overset{set}{=} 0$ $\boxed{eq1}$

  $\partial L/\partial x_2 = 2\lambda_0 x_2 + 2\lambda_2 x_2 \overset{set}{=} 0$ $\boxed{eq2}$
- Complimentarity: $\lambda_1(x_1 - 10) = 0$ and $\lambda_2(-x_1 + x_2^2 + 4) = 0$

- CASE 1: $\lambda_0 = 0$
- eq1 becomes $\lambda_1 = \lambda_2$, eq2 becomes $2\lambda_2 x_2 = 0 \Rightarrow$ either $\lambda_2 = 0$ or $x_2 = 0$. We can't let $\lambda_2 = 0$ since that makes $\lambda_1 = 0$ which gives $\lambda_0 = \lambda_1 = \lambda_2 = 0$ which is trivial.
  Thus let $x_2 = 0$.
- With $x_2 = 0$ let's look at complimentarity: $\lambda_1(x_1 - 10) = 0$ and $\lambda_2(-x_1 + 4) = 0 \Rightarrow$ $\lambda_1(x_1 - 10) = 0$ and $\lambda_2(-x_1 + 4) = 0$. Since λ_1 and λ_2 cannot $= 0$ this means that $x_1 - 10 = 0 \Rightarrow x_1 = 10$ and $-x_1 + 4 = 0 \Rightarrow x_1 = 4$. Since x_1 cannot $= 4$ and 10 we know that $\lambda_0 = 0$ won't work, and any feasible candidates must be normal ($\lambda_0 = 1$).

- CASE 2: $\lambda_0 = 1$
- eq1 becomes: $2x_1 + \lambda_1 - \lambda_2 = 0$, eq2 becomes: $2x_2 + 2\lambda_2 x_2 = 0 \rightarrow 2x_2(1 + \lambda_2) = 0$ means either $x_2 = 0$ or $\lambda_2 = -1$, but $\lambda \geq 0$ is required, so we must let $x_2 = 0$.
- Now the complimentarity conditions: $\lambda_2(-x_1 + 0^2 + 4) = 0$ and $\lambda_1(x_1 - 10) = 0$
  These conditions will give us 4 sub-cases to check when $\lambda_0 = 1$:

  (A) $x_1 = 4$, (B) $x_1 = 10$, (C) $\lambda_1 = 0$, (D) $\lambda_2 = 0$

- CASE 2A: $(\lambda_0 = 1)$, $x_1 = 4$

- If $x_1 = 4 \Rightarrow x_1 \neq 10 \Rightarrow \lambda_1 = 0$ (from complimentarity)
- Using eq1: $2x_1 + \lambda_1 - \lambda_2 = 0 \rightarrow 2(4) + 0 = \lambda_2 \rightarrow \lambda_2 = 8$
- This candidate, $(x_1 = 4,\ x_2 = 0,\ \lambda_0 = 1,\ \lambda_1 = 0,\ \lambda_2 = 8)$ satisfies all conditions. Since the problem is convex, we know $\begin{bmatrix} x_1^* = 4 \\ x_2^* = 0 \end{bmatrix}$ is a global optimizer of f in this case. We'll check the remaining 3 cases to see if $\exists$ any other optima that achieve $f([4,0]) = 16$ optimum given the constraints.

- CASE 2B: $(\lambda_0 = 1)$, $x_1 = 10$

- If $x_1 = 10 \Rightarrow x_1 \neq 4 \Rightarrow \lambda_2 = 0$ (from complimentarity). Using eq1: $2x_1 + \lambda_1 = 0 \Rightarrow 20 = -\lambda_1$ $\Rightarrow \lambda_1 = -20$ which violates $\lambda \geq 0 \Rightarrow x_1 \neq 10$.

- CASE 2C: $(\lambda_0 = 1)$, $\lambda_1 = 0$

- If $\lambda_1 = 0$, eq1 says $2x_1 = \lambda_2 \rightarrow$ plugged into compl. conds gives $2x_1(-x_1 + 4) \Rightarrow x_1$ is either 0 or 4. We've already looked at $x_1 = 4$ (see CASE 2A), so let $x_1 = 0$.
- If $x_1 = 0$, in the other compl. cond we have $\lambda_2\ (0+4) = 0 \Rightarrow \lambda_2 = 0$. But is $x_1 = x_2 = 0$ feasible? No. Consider $g_2(x) = x_1 - x_2^2 - 4 \geq 0 \rightarrow g_2([0,0]) = -4 \not\geq 0 \Rightarrow \lambda_1 = 0$ isn't feasible here.

- CASE 2D: $(\lambda_0 = 1)$, $\lambda_2 = 0$

- If $\lambda_2 = 0$, eq1 gives $2x + \lambda_1 = 0 \Rightarrow \lambda_1 = -2x_1$. Plugging into compl. cond 1 gives: $-2x_1(x_1 - 10) = 0$. From CASE 2C we know $x_1 = x_2 = 0$ isn't feasible, so let $x_1 = 10$ to satisfy this.
- As above in CASE 2B, if $x_1 = 10$ we have $\lambda_1 = -2(10) = -20$, and $\lambda_1 = -20$ violates $\lambda \geq 0$.

- FINAL: based on all of these cases, and the fact that our problem is convex, we can guarantee that the point $[(x_1^* = 4,\ x_2^* = 0)]$ is a global optimizer of $f(x) = x_1^2 + x_2^2$ for given constraints. This x^* gives $f(x^*) = 16$ as the global optimum, and satisfies optimality conditions with $\lambda_0 = 1$, $\lambda^T = [\lambda_1 = 0,\ \lambda_2 = 8]$.

3. Solve the following problem numerically and analytically using `fmincon`.

$$\begin{aligned}
\text{minimize} \quad & x_1^2 + x_2^2 \\
\text{subject to} \quad & 4 - x_1 - x_2^2 \leq 0 \\
& 3x_2 - x_1 \leq 0 \\
& -3x_2 - x_1 \leq 0
\end{aligned}$$

```
%=========================================================================
%== PROBLEM 3
%=========================================================================

% Initial guess (we know x* = [3,1], so we pick a close point.)
% NOTE: not even giving the correct answer could get fmincon to give the
%       right answer.
x0 = [3.0; 1.0];

% Solve the problem using fmincon.
[x,f] = fmincon(@obj,x0,[],[],[],[],[],[],@con)

% Problem 3 Objective function
function J = obj(x)
x1 = x(1);
x2 = x(2);
J  = (x1)^2 + (x2)^2;
end
% Problem 3 Constraint functions
function [cin,ceq] = con(x)
x1 = x(1);
x2 = x(2);
cin = 4 -x1 - (x2)^2 ;
cin = 3*x2 -x1 ;
cin = -3*x2  -x1 ;
ceq = [];
end
```

Figure 5: Here is my MATLAB code that uses `fmincon` to solve the problem.

```
<stopping criteria details>

x =

   1.0e-03 *

    0.1573
    0.4719


f =

   2.4748e-07
```

Figure 6: We can see from the output that `fmincon` falls flat on this problem. Even when we make x_0 the analytical solution the function can't converge to $x^* = (3, 1)$, but instead returns a point that is actually infeasible. The same held true for initial guesses x_0 further from x^*. Here `fmincon` is returning $(x_1 = 0.1573 \times 10.0^{-3}, x_2 = 0.4719 \times 10.0^{-3}) \approx (x_1 = 0, x_2 = 0)$ which is obviously not close to $x^* = (3, 1)$.

- First, is this problem a convex program like #1 and #2 were? No. Look at $g_1(x) \leq 0 \rightarrow 4 - x_1 - x_2^2 \leq 0$. Is $g_1(x)$ convex? No. $\nabla_x g_1 = \begin{bmatrix} -1 \\ -2x_2 \end{bmatrix}$ and $\nabla_x^2 g_1 = \begin{bmatrix} 0 & 0 \\ 0 & -2 \end{bmatrix} \Rightarrow \nabla_x^2 g_1$ is not PSD (since evals $\{0,-2\} \not\geq 0$).

- Since not all inequality constraints are convex, we won't have the strong guarantee of any optimality-condition-satisfying points being global minimizers.

- $L(x) = \lambda_0 x_1^2 + \lambda_0 x_2^2 + \lambda_1(4 - x_1 - x_2^2) + \lambda_2(-x_1 + 3x_2) + \lambda_3(-x_1 - 3x_2)$

$\partial L / \partial x_1 = 2\lambda_0 x_1 - \lambda_1 - \lambda_2 - \lambda_3 \overset{set}{=} 0$ $\boxed{eq1}$

$\partial L / \partial x_2 = 2\lambda_0 x_2 - 2\lambda_1 x_2 + 3\lambda_2 - 3\lambda_3 \overset{set}{=} 0$ $\boxed{eq2}$

- <u>Complimentarity:</u> $\lambda_1(4 - x_1 - x_2^2) = 0$ $\boxed{comp1}$, $\lambda_2(-x_1 + 3x_2)$ $\boxed{comp2}$, $\lambda_3(-x_1 - 3x_2)$ $\boxed{comp3}$

- <u>CASE 1: $\lambda_0 = 0$:</u>
  - This yields eq1: $-\lambda_1 - \lambda_2 - \lambda_3 = 0 \rightarrow \lambda_1 + \lambda_2 + \lambda_3 = 0$. Since $\underset{\sim}{\lambda} \geq 0 \ \forall \lambda_i \Rightarrow \lambda_1 + \lambda_2 + \lambda_3 = 0$ only holds if $\lambda_1 = \lambda_2 = \lambda_3 = 0$. BUT we can't have this, since it gives $\lambda_0 = 0$, $\underset{\sim}{\lambda} = \underset{\sim}{0}$ which violates the non-triviality condition.
  - Therefore, if $\exists$ a solution it must be "normal", i.e. $\lambda_0 = 1$.
- <u>When $\lambda_0 = 1$:</u> we have eq1: $x_1 = \dfrac{\lambda_1 + \lambda_2 + \lambda_3}{2}$, eq2: $x_2 = \dfrac{3}{2}\left(\dfrac{-\lambda_2 + \lambda_3}{1 - \lambda_1}\right)$
  - We will have numerous cases to check, generated by $\boxed{comp\ i}$ conditions. We can start by checking $\lambda_1 = 0$, $\lambda_2 = 0$, $\lambda_3 = 0$, also checking when the other terms in the $\boxed{comp\ i}$ conditions are $= 0$, allowing $\lambda_1, \lambda_2, \lambda_3$ to be some value > 0.
- Since λ_0 must $= 1$, we now must check all combos of $\lambda_1, \lambda_2, \lambda_3$ being $= 0$.

- <u>CASE 2: $\lambda_0 = 1$, while $\lambda_1 = \lambda_2 = \lambda_3 = 0$</u>
  - Leaves eq1: $x_1 = (0+0+0)/2 \rightarrow x_1 = 0$, eq2: $x_2 = \frac{3}{2}\left(\frac{0}{1}\right) = 0$, but is $(x_1 = x_2 = 0)$ feasible?

  <u>No</u>: see $g_1(x) \not\leq 0 \longrightarrow g_1([0,0]) = 4 - 0 - 0 = 4 \not\leq 0 \Rightarrow \underset{\sim}{\lambda} \neq \underset{\sim}{0}$

- CASE 3: $\lambda_0 = 1$, while $\lambda_2 = \lambda_3 = 0$ and $(4 - x_1 - x_2^2) = 0$

- Eq1 is now: $2x_1 - \lambda_1 - 0 - 0 = 0 \Rightarrow x_1 = \lambda_1/2$
- Eq2 is now: $2x_2 - 2\lambda_1 x_2 + 0 - 0 = 0 \Rightarrow x_2(2 - 2\lambda_1) = 0 \Rightarrow x_2 = 0$ or $\lambda_1 = 1$
- If $x_2 = 0$, $\boxed{\text{comp1}}$ is $(4 - x_1 - 0^2) = 0 \Rightarrow x_1 = 4 \Rightarrow \lambda_1 = 8$. Is $(x_1 = 4, x_2 = 0)$ feasible?

  (i) $g_1(x) = 4 - 4 - 0 = 0 \leq 0$ ✓

  (ii) $g_2(x) = 3(0) - 4 = -4 \leq 0$ ✓

  (iii) $g_3(x) = -3(0) - 4 = -4 \leq 0$ ✓

  > So $(x_1 = 4, x_2 = 0)$ with $\lambda_0 = 1$, $\lambda^T = [\lambda_1 = 8, \lambda_2 = \lambda_3 = 0]$ is feasible & satisfies optimality conditions. (Not necessarily a local or global min though.)

- What if $\lambda_1 = 1$? Eq1 is: $2x_1 - 1 = 0 \rightarrow x_1 = 1/2$. Then, to satisfy $\boxed{\text{comp 1}}$ we need $(4 - x_1 - x_2^2) = 0 \rightarrow \left(\frac{8}{2} - \frac{1}{2} = x_2^2\right) \Rightarrow x_2 = \pm\sqrt{7/2}$. Is $\left(x_1 = \frac{1}{2}, x_2 = \pm\sqrt{\frac{7}{2}}\right)$ feasible?

  No for both points. It obviously satisfies $g_1(x) \leq 0$ by construction, but we can't satisfy $g_2(x) \leq 0$ and $g_3(x) \leq 0$.

  $g_2\left(\left[x_1 = 1/2, x_2 = \sqrt{\frac{7}{2}}\right]\right) = 3\left(\sqrt{\frac{7}{2}}\right) - \frac{1}{2} = 5.11 \nleq 0 \Rightarrow \left(x_1 = \frac{1}{2}, x_2 = \sqrt{\frac{7}{2}}\right)$ isn't feasible.

  $g_3\left(\left[x_1 = 1/2, x_2 = -\sqrt{\frac{7}{2}}\right]\right) = {+}{+}\,3\left(\sqrt{\frac{7}{2}}\right) - \frac{1}{2} = 5.11 \nleq 0 \Rightarrow \left(x_1 = \frac{1}{2}, x_2 = -\sqrt{\frac{7}{2}}\right)$ isn't feasible.

- CASE 4: $\lambda_0 = 1$, while $\lambda_1 = \lambda_2 = 0$ and $(-x_1 - 3x_2 = 0)$

- Eq1 is now: $2x_1 - \lambda_3 = 0 \rightarrow x_1 = \lambda_3/2$, Eq2 is now: $2x_2 - 3\lambda_3 \rightarrow x_2 = \frac{3}{2}\lambda_3$
- Sub in to $\boxed{\text{comp3}}$ cond: $\left(-\frac{\lambda_3}{\cancel{2}} = 3\left(\frac{3}{\cancel{2}}\lambda_3\right)\right) \rightarrow -\lambda_3 = 9\lambda_3 \rightarrow$ iff $\lambda_3 = 0 \Rightarrow x_1 = x_2 = 0$
- But $x_1 = x_2 = 0$ violates $g_1(x) \leq 0 \rightarrow 4 - 0 - 0^2 = 4 \nleq 0 \Rightarrow$ this is infeasible.

- CASE 5: $\lambda_0 = 1$, while $\lambda_1 = \lambda_3 = 0$ and $(-x_1 + 3x_2 = 0)$

- Eq1 is now: $2x_1 - \lambda_2 = 0$ and Eq2 is now: $2x_2 + 3\lambda_2 = 0 \rightarrow x_1 = \frac{\lambda_2}{2}$, $x_2 = \frac{-3\lambda_2}{2}$
- Plugging in to $\boxed{\text{comp2}}$ condition gives: $\frac{\lambda_2}{\cancel{2}} = 3\left(\frac{-3\lambda_2}{\cancel{2}}\right) \rightarrow \lambda_2 = -9\lambda_2 \rightarrow$ iff $\lambda_2 = 0$

  $\Rightarrow x_1 = x_2 = 0$, which again violates $g_1(x) \leq 0 \rightarrow 4 - 0 - 0^2 = 4 \nleq 0$.

  Thus, this route is infeasible.

- CASE 6: $\lambda_0 = 1,\ \lambda_1 = 0,\ (-x_1 + 3x_2 = 0),\ (-x_1 - 3x_2 = 0)$

- We can see that $(-x_1 + 3x_2$ must $= -x_1 - 3x_2) \rightarrow -x_1 + x_1 = 3x_2 ++ 3x_2$ $\rightarrow 6x_2 = 0 \rightarrow$ iff $x_2 = 0 \Rightarrow x_1 = 0$, but we know $x_1 = x_2 = 0$ violates $g_1(x) \leq 0$, since $4 - 0 - 0^2 = 4 \neq 0$. Thus, this solution can't work.

- CASE 7: $\lambda_0 = 1,\ \lambda_2 = 0,\ (4 - x_1 - x_2^2 = 0),\ (-x_1 - 3x_2) = 0$

- Using the last equation in the CASE, $\Rightarrow x_1 = -3x_2$. Plugging in to the other CASE equation gives: $(4 + + 3x_2 - x_2^2 = 0) \equiv x_2^2 - 3x_2 - 4 = 0 \rightarrow (x_2 - 4)(x_2 + 1) = 0$ This gives $x_2 = 4$ and $x_2 = -1 \rightarrow (x_1 = 3, x_2 = -1),\ (x_1 = -12, x_2 = 4)$. Feasible?
- $(x_1 = 3, x_2 = -1)$: this is a feasible point. Looking at eq 1: $-2 - \lambda_1 - \lambda_3 = 0$ and eq 2: $-2 + 2\lambda_1 - 3\lambda_3 = 0$. Gives $\lambda_1 = -2 - \lambda_3 \rightarrow$ into eq 2: $-2 - 4 - 2\lambda_3 - 3\lambda_3 = 0$ $\rightarrow -5\lambda_3 = 6 \rightarrow \lambda_3 = -6/5$ which violates $\lambda_i \geq 0\ \forall i \in \{1,2,3\}$, so this solution won't work. In short, $(x_1 = 3, x_2 = -1)$ is feasible, but doesn't satisfy FJ conditions here.
- $(x_1 = -12, x_2 = 4)$: this is a feasible point. However, it has a much greater objective value than the candidate from CASE 3, so it's not worth finding λ_1 & λ_3.

- CASE 8: $\lambda_0 = 1,\ \lambda_3 = 0,\ (4 - x_1 - x_2^2 = 0),\ (-x_1 + 3x_2 = 0)$

- Using the equations in the case: $x_1 = 3x_2 \rightarrow [4 - 3x_2 - x_2^2 = 0] \equiv [x_2^2 + 3x_2 - 4 = 0]$ $\equiv [(x_2 + 4)(x_2 - 1) = 0] \Rightarrow$ either $(x_2 = 1, x_1 = 3)$ or $(x_2 = -4, x_1 = -12)$.
- $(x_1 = -12, x_2 = -4)$: doesn't matter, much larger objective value than case 3 candidate.
- $(x_1 = 3, x_2 = 1)$: feasible point. Our new eq 1: $6 - \lambda_1 - \lambda_2 = 0$, and our new eq 2: $2 - 2\lambda_1 + 3\lambda_2 = 0$. $\lambda_1 = 6 - \lambda_2 \rightarrow 2 - 2(6 - \lambda_2) + 3\lambda_2 = 0 \rightarrow 5\lambda_2 = 10 \rightarrow \lambda_2 = 2$ $\Rightarrow \lambda_1 = 4$

Thus, $(x_1 = 3, x_2 = 1)$ with $\lambda_0 = 1$, $\lambda^T = [\lambda_1 = 4, \lambda_2 = 2, \lambda_3 = 0]$ is our best candidate with $f(x) = 3^2 + 1^2 = 10$. We can't guarantee it to be a global max, but it is the best solution using the FJ conditions. $(x_1 = 3, x_2 = -1)$ also achieves $f(x) = 10$, but doesn't satisfy FJ conditions (although it IS a feasible point).

4. Solve the following problem numerically and analytically using `fmincon`.

$$\begin{aligned} \text{minimize} \quad & x_1 x_2 \\ \text{subject to} \quad & x_1 + x_2 \geq 2 \\ & x_2 \geq x_1 \end{aligned}$$

```
%=========================================================================
%== PROBLEM 4
%=========================================================================

% Initial guess...answer DNE, so doesn't really matter what we put in.
x0 = [-10; 14];

% Solve the problem using fmincon.
[x,f] = fmincon(@obj,x0,[],[],[],[],[],[],@con)

% Problem 4 Objective function
function J = obj(x)
x1 = x(1);
x2 = x(2);
J  = x1 * x2 ;
end
% Problem 4 Constraint functions
function [cin,ceq] = con(x)
x1 = x(1);
x2 = x(2);
cin = -x1 - x2 + 2 ;
cin = x1 - x2 ;
ceq = [];
end
```

Figure 7: Here is my MATLAB code that uses `fmincon` to solve the problem.

```
<stopping criteria details>

x =

   1.0e+12 *

   -1.5362
    1.4928


f =

  -2.2931e+24
```

Figure 8: As we would expect, `fmincon` can't find the correct answer since this problem does not have an answer (see analytical examination below; min DNE since it continually approaches $-\infty$). But, we can see that the function behaves as we've predicted, making x_2 very large and positive and making x_1 very large and negative in order to achieve a very large, negative value of $f(x)$. Here the function achieves ($x_1 = -1.5362 \times 10^{12}, x_2 = 1.4928 \times 10^{12}$) and $f(x) = -2.2931 \times 10^{24}$. I'd bet that as we continue to make guesses of very negative x_1 and very positive x_2 for x_0 then $f(x) \longrightarrow -\infty$.

- First, is the problem convex? <u>No</u>. $\nabla_x f = \begin{bmatrix} x_2 \\ x_1 \end{bmatrix} \rightarrow \nabla_x^2 f = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ which has eigvals $= -1, 1$ which aren't both $\geq 0 \Rightarrow \nabla_x^2 f$ not PSD $\rightarrow f$ not convex
  $\Rightarrow$ this is not a convex program.
- Let's build some additional intuition before jumping in to the math.
  In general, we'll minimize $x_1 x_2$ if one of $\{x_1, x_2\} > 0$ and the other of $\{x_1, x_2\} < 0$, with $|x_1|$ or $|x_2|$ a very large value, e.g. a positive times a negative is a negative, and the larger in magnitude the values x_1 & x_2 the larger in (absolute) magnitude their product, which will be negative. We know $x_2 \geq x_1 \Rightarrow x_2$ will be the positive and x_1 will be the negative. Then we just need $x_1 + x_2 \geq 2 \Rightarrow |x_2| \geq |x_1| + 2$.
- For example: Let $x_2 = 12$ and $x_1 = -10$. This satisfies the constraints and gives $f(x) = (-10)(12) = -120$, a pretty low (minimal) objective!
- But what if $x_2 = 102$ and $x_1 = -100$. This also satisfies the constraints but gives $f(x) = (-100)(102) = -10{,}200$ an even lower (more min.) objective.

• We can see this trend can continue interminably. Therefore we know a min of $f(x) = x_1 x_2$ DNE (it continually approaches $-\infty$ as x_1 and x_2 grow in magnitude.)

- Now let's use more formal math to substantiate our intuition and line of reasoning that min $f(x) = x_1 x_2$ DNE.
- $L(x) = \lambda_0 x_1 x_2 + \lambda_1(-x_1 - x_2 + 2) + \lambda_2(x_1 - x_2)$

$$\partial L / \partial x_1 = \lambda_0 x_2 - \lambda_1 + \lambda_2 \overset{set}{=} 0 \longleftarrow \boxed{eq\ 1}$$

$$\partial L / \partial x_2 = \lambda_0 x_1 - \lambda_1 - \lambda_2 \overset{set}{=} 0 \longleftarrow \boxed{eq\ 2}$$

- Complimentarity: $\lambda_1(-x_1-x_2+2)=0 \leftarrow \boxed{\text{comp }1}$, $\boxed{\text{comp}2} \rightarrow \lambda_2(x_1-x_2)=0$

- Case 1: $\lambda_0=0$
  - Eq1 becomes: $-\lambda_1+\lambda_2=0 \Rightarrow \lambda_1=\lambda_2$, Eq2 is now: $-\lambda_1=\lambda_2$. For both to hold, we must have $\lambda_1=\lambda_2=0$, BUT $\lambda_0=0$ and $\lambda^T=[\lambda_1=0, \lambda_2=0]$ violates the non-trivality condition. Thus, λ_0 must $=1$ for any potential solutions.

- Case 2: $\lambda_0=1$, $\lambda_1=0$, $\lambda_2=0$
  - Eq1 becomes: $x_2=0$, Eq2 becomes $x_1=0$. Is $(0,0)$ feasible? No, it's not. Consider $x_1+x_2\geq 2 \rightarrow 0+0=0 \not\geq 2 \Rightarrow$ this case isn't feasible.

- Case 3: $\lambda_0=1$, $\lambda_1=0$, $(x_1-x_2=0)$
  - If $x_1-x_2=0 \Rightarrow x_1=x_2$. Eq1 becomes: $x_2+\lambda_2=0$, Eq2 becomes: $x_1-\lambda_2=0$.
  - Now we have $x_2=-\lambda_2$ and $x_2=\lambda_2$ which only works if $\lambda_2=0$ which then makes $x_2=0 \Rightarrow x_1=0$ and we just showed $(0,0)$ to be infeasible in Case 2. Thus this case, case 3, is also infeasible.

- Case 4: $\lambda_0=1$, $\lambda_2=0$, $(-x_1-x_2+2=0)$
  - Eq1 becomes: $x_2-\lambda_1=0$, Eq2 becomes: $x_1-\lambda_1=0 \Rightarrow x_1=x_2=\lambda_1$
  - Rewriting $(x_1+x_2=2) \rightarrow x_1+x_1=2 \rightarrow x_1=1 \Rightarrow x_2=1 \Rightarrow \lambda_1=1$.
  - The candidate $x=[x_1=1, x_2=1]$ is feasible and satisfies optimality conditions with $\lambda_0=1$, $\lambda^T=[\lambda_1=1, \lambda_2=0]$. But we've already shown the pt $(-100, 102)$ has a much more minimal objective value, thus $(1,1)$ is not a minimizer.

- Case 5: $\lambda_0=1$, $(x_1-x_2=0)$, $(-x_1-x_2+2=0)$
  - $x_1-x_2=0 \Rightarrow x_1=x_2 \rightarrow$ into other eqn: $-x_1-x_1=-2 \Rightarrow x_1=1$, $x_2=1$, same as case 4.

- Thus, $\nexists$ a minimizer of $f(x)=x_1x_2$ by using FJ optimality conditions, and our exploration at the beginning is upheld: $\nexists$ a minimizer for this problem.

5. Solve the following problem numerically and analytically using `fmincon`.

$$
\begin{aligned}
\text{minimize} \quad & -x_1 \\
\text{subject to} \quad & -x_1 \leq 0 \\
& -x_2 \leq 0 \\
& x_2 + (x_1 - 1)^3 \leq 0
\end{aligned}
$$

```
%==========================================================================
%== PROBLEM 5
%==========================================================================

% Initial guess. We know x* = [1,0] so we'll pick x0 close to that.
% NOTE: even giving an initial guess that is the answer can't get
%       fmincon to converge to the correct answer.
%       However, I accidentally made g3(x) equality instead of ineq
%       and it did give the correct answer.
x0 = [1.0; 0.0];

% Solve the problem using fmincon.
[x,f] = fmincon(@obj,x0,[],[],[],[],[],[],@con)

% Problem 5 Objective function
function J = obj(x)
x1 = x(1);
x2 = x(2);
J  = -x1 + 0*x2 ;
end
% Problem 5 Constraint functions
function [cin,ceq] = con(x)
x1 = x(1);
x2 = x(2);
cin = -x1 ;
cin = -x2 ;
cin = x2 + (x1 - 1)^3 ;
ceq = [] ;
end
```

Figure 9: Here is my MATLAB code that uses `fmincon` to solve the problem.

```
<stopping criteria details>

x =

   1.0e+19 *

    0.0000
   -4.4751


f =

  -3.5503e+06
```

Figure 10: We can see that `fmincon` gets this problem very wrong even though we gave it the exactly-correct answer as x_0. It is giving an infeasible solution since $-x_2 \nleq 0$. Also, it is very far from the correct answer of $x^* = (1, 0)$ instead getting $x = (x_1 = 0.0, x_2 = -4.48 \times 10^{19})$.

- First, is the problem convex? No.
- $\nabla_x g_3 = \begin{bmatrix} 3(x_1-1)^2 \\ 1 \end{bmatrix} \longrightarrow \nabla_x^2 g_3 = \begin{bmatrix} 6(x_1-1) & 0 \\ 0 & 0 \end{bmatrix}$ Is this a PSD matrix? No. Let $x_1=0$, gives eigvals $\{-6,0\} \not\geq 0 \Rightarrow \nabla_x^2 g_3$ is not PSD $\Rightarrow$ Ineq. constraint $g_3(x)$: $x_2 + (x_1-1)^3 \leq 0$ is not convex $\Rightarrow$ this problem as a whole is not convex.
- Since this isn't a CP we can't make guarantees on candidate points being global optima or not.
- Let's solve the problem without using the FJ conditions to make sure we get the right answer when we do so:
- In general, to minimize $-x_1$ we must make x_1 a positive value as large as possible, relative to the given constraints.
- $g_1(x)$: $-x_1 \leq 0$ This fits into the desire to make $-x_1$ as small (x_1 as large) as possible. BUT may leave x_1 unbounded? (Taken care of in constraint $g_3(x)$.)
- $g_2(x)$: $-x_2 \leq 0$ We know that $x_2 \geq 0$.
- $g_3(x)$: $x_2 + (x_1-1)^3 \leq 0 \longrightarrow (x_1-1)^3 \leq -x_2$ and $g_2(x)$ stipulates $-x_2 \leq 0$ $\Rightarrow (x_1-1)^3 \leq 0$ How do we satisfy this? With some $x_1 \leq 1$. BUT, our means of minimizing $-x_1$ is to make x_1 as large as possible. Therefore, if it must be that $x_1 \leq 1$ we'd choose $x_1 = 1$ to minimize $-x_1$.
- So using a rough approach (that won't work in many cases, but does here) we see $(x_1^* = 1,\ x_2^* = 0)$ which we'll verify using FJ-conditions. (Technically, we may only be showing that $(x_1=1,\ x_2=0)$ is the best candidate that comes from applying the FJ conditions. Since the program is non-convex, we can't mathematically guarantee it's the global optimum. Rather, our deductive method helps us with this.)

- $L(x) = -\lambda_0 x_1 - \lambda_1 x_1 - \lambda_2 x_2 + \lambda_3(x_2 + (x_1-1)^3)$

  $\partial L/\partial x_1 = -\lambda_0 - \lambda_1 + 3\lambda_3(x_1-1)^2 \overset{set}{=} 0$ $\boxed{eq\ 1}$

  $\partial L/\partial x_2 = -\lambda_2 + \lambda_3 = 0$ $\boxed{eq\ 2}$ $\Rightarrow \lambda_3 = \lambda_2$

- Complimentarity conditions:

  $\boxed{c1}$: $-\lambda_1 x_1 = 0 \longrightarrow$ either $\lambda_1 = 0$ or $x_1 = 0$

  $\boxed{c2}$: $-\lambda_2 x_2 = 0 \longrightarrow$ either $\lambda_2 = 0$ or $x_2 = 0$

  $\boxed{c3}$: $\lambda_3(x_2 + (x_1-1)^3) = 0 \rightarrow$ either $\lambda_3 = 0$ or $(x_1-1)^3 = -x_2$

- Since $\boxed{eq\ 2}$ requires that $\lambda_3 = \lambda_2$ we need only check cases where $\lambda_3 = \lambda_2$.

- CASE 1: $\lambda_0 = 0$, $\lambda_1 \neq 0$, $\lambda_2 = \lambda_3 = 0$

- Eq 1: $-\lambda_1 + 0 + 0(x_1-1)^2 = 0 \Rightarrow \lambda_1 = 0 \Rightarrow \lambda_0 = 0$ and $\underset{\sim}{\lambda} = \underset{\sim}{0}$ which violates the non-triviality condition. Therefore, this case is not feasible.

- CASE 2: $\lambda_0 = 0$, $\lambda_1 = 0$, $\lambda_2 \neq 0$, $\lambda_3 \neq 0$

- Eq 1: $3\lambda_3(x_1-1)^2 = 0$, λ_3 must be $\neq 0 \Rightarrow (x_1 - 1) = 0 \Rightarrow x_1 = 0$
- $\boxed{c2}$: since $\lambda_2 \neq 0$ here $\Rightarrow \lambda_2 x_2 = 0$ iff $x_2 = 0$
- $\boxed{c3}$: since $\lambda_3 \neq 0$ we have $\lambda_3(0 + (1-1)^3) = 0 \rightarrow \lambda_3(0) = 0 \Rightarrow \lambda_3 \in \mathbb{R}$

  and since $\lambda_3 \in \mathbb{R}$ and $\lambda_3 = \lambda_2 \Rightarrow \boxed{\lambda_2 \in \mathbb{R}}$

- Thus, $(x_1 = 1,\ x_2 = 0)$ with $\lambda^T = [\lambda_1 = 0,\ \lambda_2 \geq 0,\ \lambda_3 \geq 0]$ with $\lambda_2 = \lambda_3$ is a candidate according to the FJ conditions. Strictly speaking, we can't declare this to be a global minimizer, but from our procedural reasoning above (at problem's outset) we know that this is a best solution (there may exist others, we'll continue checking other cases).

- **CASE 3:** $\lambda_0 = 1$, $\underset{\sim}{\lambda} = \underset{\sim}{0}$ e.g. $\lambda_1 = \lambda_2 = \lambda_3 = 0$

- Eq 1: $-1 - 0 + 0 = 0 \Rightarrow -1 = 0$ which isn't true. Therefore, having these λ values fails to satisfy FJ conditions ("FJ infeasible").

- **CASE 4:** $\lambda_0 = 1$, $\lambda_1 \neq 0$, $\lambda_2 = \lambda_3 = 0$

- Eq 1: $-1 - \lambda_1 + 0 = 0 \Rightarrow \lambda_1 = -1$ which can't be, since all elements of $\underset{\sim}{\lambda}$ must be ≥ 0, $\Rightarrow$ this case isn't feasible under the FJ conditions.

- **CASE 5:** $\lambda_0 = 1$, $\lambda_1 = 0$, $\lambda_2 \neq 0$, $\lambda_3 \neq 0$, ($\lambda_2 = \lambda_3$ still required by $\boxed{\text{eq } 2}$)

- Eq 1: $-1 - 0 + 3\lambda_3 (x_1 - 1)^2 = 0 \rightarrow 3\lambda_3 (x_1 - 1)^2 = 1$
- c2: Since we've said $\lambda_2 \neq 0$, for $\lambda_2 x_2 = 0 \Rightarrow x_2 = 0$
- c3: Since we've said $\lambda_3 \neq 0$, $\lambda_3 (x_1 - 1)^2 = 0 \Rightarrow x_1 - 1 = 0 \Rightarrow x_1 = 1$
- Back to Eq 1: if $3\lambda_3 (x_1 - 1)^2 = 1 \rightarrow 3\lambda_3 (1 - 1)^2 = 1 \Rightarrow 3\lambda_3 (0) = 0 = 1$ which isn't possible $\Rightarrow$ no feasible solution under FJ conditions for $\lambda_0 = 1$, $\lambda_1 = 0$, $[\lambda_2 = \lambda_3] \neq 0$.

- **CONCLUSION:** we have demonstrated the existence of one viable candidate, $(x_1 = 1, x_2 = 0)$ with $\lambda_0 = 0$ and $\lambda^T = [\lambda_1 = 0, \lambda_2 \geq 0, \lambda_3 \geq 0]$ and $\lambda_2 = \lambda_3$. Since the program isn't convex, we can't declare this to be a global minimizer based on the FJ conditions being satisfied. However, based on the procedural explanation at the problem's outset, we concluded the best solution is $(x_1 = 1, x_2 = 0)$. (Done via some logic and combination of constraints $g_2(x)$ and $g_3(x)$.) This is substantiated by our findings in CASE 2, showing this candidate has a satisfactory formulation for the FJ conditions. Thus, based on some not-typical reasoning, we can conclude $x^* = (x_1^* = 1, x_2^* = 0)$ achieving $f(x^*) = -1$.

6. Solve the following problem numerically and analytically using `fmincon`.

$$\begin{array}{ll} \text{minimize} & 2x_1^2 - x_2^2 \\ \text{subject to} & x_1^2 x_2 - x_2^3 = 0 \end{array}$$

```
%=========================================================================
%== PROBLEM 6
%=========================================================================

% Initial guess. We know x* = [0,0] so we'll pick x0 close to that.
x0 = [0.005; 0.005];

% Solve the problem using fmincon.
[x,f] = fmincon(@obj,x0,[],[],[],[],[],[],@con)

% Problem 6 Objective function
function J = obj(x)
x1 = x(1);
x2 = x(2);
J  = 2*(x1^2) - (x2^2) ;
end
% Problem 6 Constraint functions
function [cin,ceq] = con(x)
x1 = x(1);
x2 = x(2);
cin = (x1^2)*x2 - (x2^3) ;
ceq = [];
end
```

Figure 11: Here is my MATLAB code that uses `fmincon` to solve the problem.

```
<stopping criteria details>

x =

   1.0e+10 *

    0.2810
    2.3552


f =

  -5.3892e+20
```

Figure 12: Just as above in problem 5, `fmincon` gets this problem very wrong, even when we give it almost the exactly-correct answer for x_0. It is giving an infeasible solution since $x_1^2 x_2 - x_2^3 \neq 0$ but instead for the outputted x_1 and x_2 we get $x_1^2 x_2 - x_2^3 = 1.8597 \times 10^{29}$. Also, it is very far from the correct answer of $x^* = (0, 0)$ instead getting $x = (x_1 = 0.2810 \times 10^{10}, x_2 = 2.3552 \times 10^{10})$ but getting close to the correct optimal objective value of $f(x) = 0$.

- First, is the problem convex? <u>No</u>. $\nabla_x^2 f(x) = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix} \Rightarrow$ eigvals $= -1, 2$, since $-1 \not\geq 0 \Rightarrow$ not all positive eigvals $\Rightarrow \nabla_x^2 f(x)$ not PSD $\Rightarrow f(x)$ objective is not convex $\Rightarrow$ this is not a convex program.

- Let's see if we can solve the problem without using the FJ conditions. This will help us know if candidates are good, correct, lend intuition.
- Intuition, to minimize $2x_1^2 - x_2^2$ we should make x_1^2 as small as possible (occurs when $x_1 = 0$) and make $|x_2|$ as large as possible. Thus we'll have $0 -$ (some big number).
- If $x_1 = 0 \Rightarrow h(x) = x_1^2 x_2 - x_2^3 = 0 \rightarrow 0 x_2 - x_2^3 = 0 \rightarrow x_2^3 = 0 \Rightarrow x_2 = 0$.

Is $(x_1 = x_2 = 0)$ the best we can do? (obj value $= 0$)
- From $h(x)$ we see $(x_1^2)(\cancel{x_2}) = (\cancel{x_2})(x_2^2)$. $\Rightarrow \pm x_1 = \pm x_2$. But since both x_1 and x_2 are squared in $f(x)$, let's just pick $[x_1 = x_2] > 0$ for ease.

Say $x_1 = x_2 = 2 \longrightarrow$ obj: $f(x) = 2(2^2) - (2^2) = 8 - 4 = 4 \not\leq 0$ (best obj. so far).
- Intuitively, since x_1^2 must $= x_2^2$, in the objective function the first term will always be double the second term. Since the terms are always non-negative, the best objective value is when the $x_1^2 = x_2^2 = 0$, which occurs when $x_1 = x_2 = 0$.

- Thus, our best objective value is $f(x) = 0$ at $[x_1 = 0, x_2 = 0]$. We didn't arrive here through rigorous proofs, but rather through intuition, logical deduction, and some simple math. (Thus, can't declare global optimality.) Let's see if use of the FJ conditions substantiates this finding.

- $L(x) = \lambda_0 (2x_1^2 - x_2^2) + \lambda_1 (x_1^2 x_2 - x_2^3)$

$\partial L/\partial x_1 = 4\lambda_0 x_1 + 2\lambda_1 x_1 \overset{set}{=} 0 \rightarrow x_1(2\lambda_0 + \lambda_1) = 0 : \boxed{eq1}$

$\partial L/\partial x_2 = -2\lambda_0 x_2 + \lambda_1 x_1^2 - 3\lambda_1 x_2^2 \overset{set}{=} 0 : \boxed{eq2}$

- Complimentarity:

$\boxed{c1}$: $\lambda_1 (x_1^2 x_2 - x_2^3) = 0 \Rightarrow$ either $\lambda_1 = 0$ or $(x_1^2 x_2 - x_2^3 = 0)$

- CASE 1: $\lambda_0 = 0, \lambda_1 \neq 0$

- $\boxed{Eq1}$ becomes: $x_1(\lambda_1) = 0 \Rightarrow x_1 = 0$ since we specified $\lambda_1 \neq 0$ in case.
- From $\boxed{c1}$, $\lambda_1 \neq 0 \Rightarrow x_1^2 x_2 - x_2^3 = 0$ with $x_1 = 0 \rightarrow x_2^3 = 0 \Rightarrow x_2 = 0$
- From $\boxed{eq2}$: $-0 + \lambda_1 (0^2) - 3\lambda_1(0^2) = 0 \rightarrow 0 + 0 - 0 = 0$ holds $\forall \lambda_1 \in \mathbb{R}$.

- Thus, the candidate $[x_1 = 0, x_2 = 0]$ with $\lambda_0 = 0$ and $\lambda_1 > 0$ (to avoid violation of non-triviality condition) is viable per the optimality conditions. From our logic at the beginning of the problem, we know this is the best solution we can achieve. Let's be thorough and check the remaining cases.

- CASE 2: $\lambda_0 = 1, \lambda_1 = 0$

- $\boxed{c1}$ satisfied since $\lambda_1 = 0$. $\boxed{Eq1}$ becomes $x_1(2\lambda_0) = 0 \rightarrow x_1(2) = 0 \Rightarrow x_1 = 0$
- $\boxed{Eq2}$ becomes $-2\lambda_0 x_2 + 0 - 0 = 0 \rightarrow -2x_2 = 0 \Rightarrow x_2 = 0$

- Once again, $(x_1 = 0, x_2 = 0)$ is a viable candidate, this time with $\lambda_0 = 1$ and $\lambda_1 = 0$.

- CASE 3: $\lambda_0 = 1$, $\lambda_1 > 0$

- To satisfy $\boxed{c1}$ we have $x_1^2 x_2 - x_2^3 = 0 \longrightarrow (x_1^2)(\cancel{x_2}) = (\cancel{x_2})(x_2^2) \Rightarrow x_1^2 = x_2^2$

$\Rightarrow$ either $x_1 = x_2$ or $-x_1 = x_2$ (equivalent to $x_1 = -x_2$).

- When $x_1 = x_2$: $\boxed{eq1}$ is now $x_1(2+\lambda_1) = 0 \Rightarrow x_1 \text{ must } = 0 \Rightarrow x_2 = 0$

  - Does $\boxed{eq2}$ hold? $-2x_2 + \lambda_1 x_2^2 - 3\lambda_1 x_2^2 = 0 \longrightarrow -2x_2 - 2\lambda_1 x_2^2 = 0$

$\rightarrow -2(x_2 - \lambda_1 x_2^2) = 0 \longrightarrow x_2(1 - \lambda_1 x_2) = 0$, which works when $x_2 = 0$, which we're explored. So let's explore $1 - \lambda_1 x_2 = 0 \rightarrow x_2 = 1/\lambda_1$ and $\lambda_1 = 1/x_2$

  - In this case: objective $= 2\left(\frac{1}{\lambda_1}\right)^2 - \left(\frac{1}{\lambda_1}\right)^2 = \frac{1}{\lambda_1}$. Since $\lambda_1 > 0$ we can take $\lim_{\lambda_1 \to \infty} \frac{1}{\lambda_1} = 0$.

  - Thus, the candidate $\left(x_1 = 1/\lambda_1,\ x_2 = 1/\lambda_1\right)$, with $\lambda_0 = 1$ and $\lambda_1 \to \infty$ is a viable candidate, but never quite achieves $f(x) = 0$, just $f(x) \to 0$ as $\lambda_1 \to \infty$.

- When $-x_1 = x_2$: $\boxed{eq2}$ becomes $-2x_2 + \lambda_1(-x_1)^2 - 3\lambda_1(x_2^2) = 0$

$\rightarrow -2x_2 + \lambda_1 x_2^2 - 3\lambda_1(x_2^2) = 0 \longrightarrow -2x_2 - 2\lambda_1 x_2^2$, which we run into above.

Thus, there are no new λ_1 values to explore.

- Conclusion: we have explored all possible non-trivial cases, and found candidates $(x_1 = 0, x_2 = 0)$ and $(x_1 = x_2 = 1/\lambda_1$ as $\lambda_1 \to \infty)$. The best objective value from these is $f(x) = 0$. Since this isn't a CP, we can't guarantee $(0,0)$ to be a/the global optimizer. However, we know it's a viable candidate according to optimality conditions (both when $\lambda_0 = 0, \lambda_1 > 0$ and when $\lambda_0 = 1$ and $\lambda_1 = 0$). Also, based on our reasoning at the beginning of the problem, it is very likely a global optimizer (though rigorous proof would need to hold to guarantee this formally.)

7. Solve the traveling salesman problem using MATLAB's genetic algorithm solver `ga`. Use the MTZ formulation below.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i,j} d_{i,j} x_{i,j} \\
\text{subject to} \quad & \sum_{i} x_{i,j} = 1, \forall j \\
& \sum_{j} x_{i,j} = 1, \forall i \\
& u_1 = 1 \\
& 2 \le u_i \le n, \forall i \neq 1 \\
& u_i - u_j + 1 \le (n-1)(1 - x_{i,j}), \forall i \neq 1, \forall j \neq 1 \\
& x_{i,j} \in \{0,1\} \text{ and } u_i \in \mathbb{R}
\end{aligned}
$$

Generate random x and y locations for the cities using random numbers as

```
» x = 20*rand(1,n);
» y = 20*rand(1,n);
```

See how large you can make `n` and still solve the problem. Recall that you could solve about 40 cities using the integer programming approach.

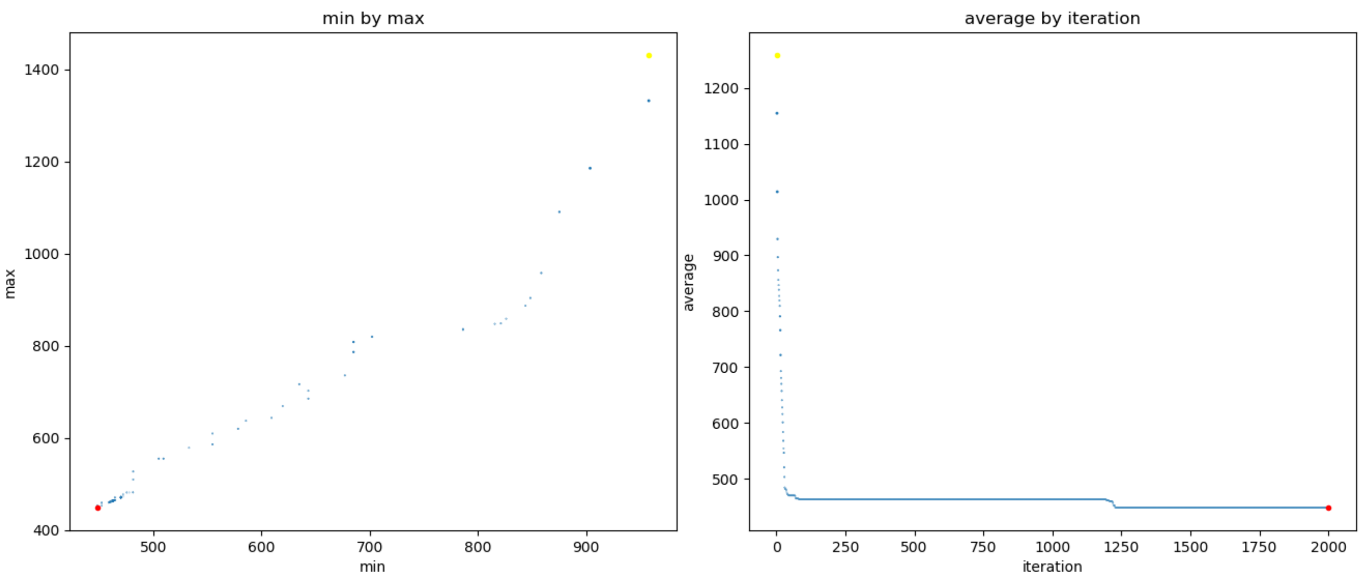

Figure 13: Here are plots (made in Python using `matplotlib`) showing the progression of the GA, e.g. it "learning" over iterations for 25 randomly-generated cities. This is telling us that the final solution achieved by the GA has a path length of about 450 which it achieved after roughly the 1250^{th} iteration mark (see the right plot). We can see the actual cities and the plot of the final solution in the image directly below.

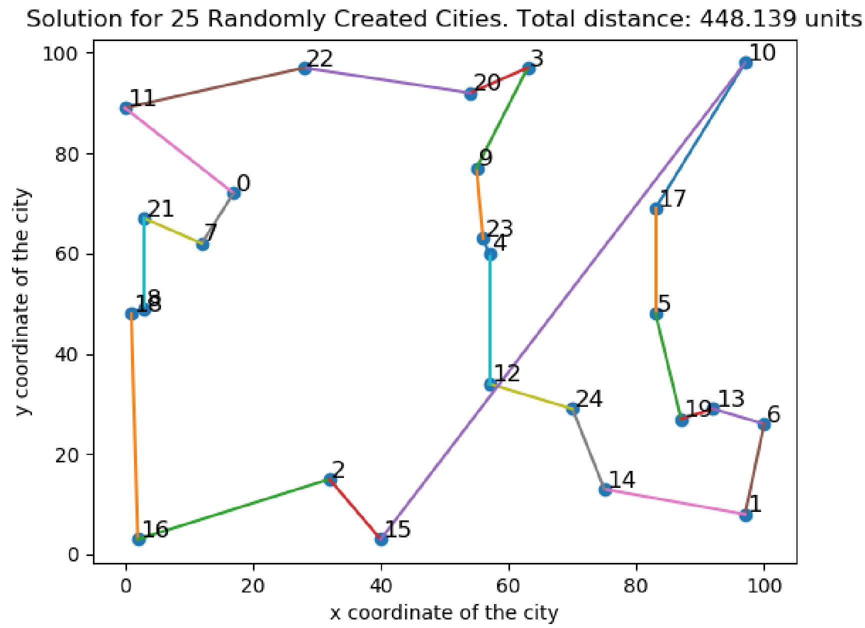

Figure 14: This plot (made in Python using `matplotlib`) shows the final path decided upon by the genetic algorithm for 25 cities in the traveling salesman problem. I used the `DEAP (Distributed Evolutionary Algorithms in Python)` package to do the solving. We can see that this isn't quite an ideal solution; at least from what I saw in the integer programming solutions, we'd expect the most efficient path to be "totally hollow", whereas this is certainly not hollow. I think if it went from 15 over to 12, then kept 12 to 24, and so on up to 10 and then had 10 connect to 4 it would be a more "ideal-looking" solution, but this certainly isn't too far off.

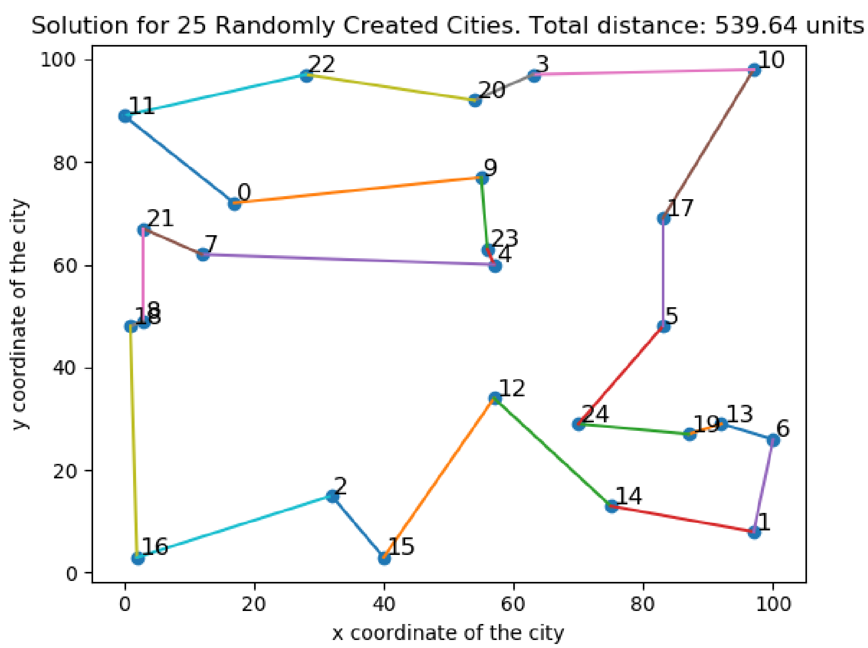

Figure 15: For comparison, I ran the script with a different seed and obtained this plot of cities and ideal route as obtained by the genetic algorithm. Here we can see that this **looks** like a more ideal route, as it is perfectly hollow and traces the outer edge to connect all the cities, but it is not an/the ideal route! Comparing it to the plot above: we can see that all the cities are in the same place, so it is the same map. But there is a big difference between the 448-unit-length path in Figure 14 and the 539-unit-length path in this plot. Looks can be deceiving!

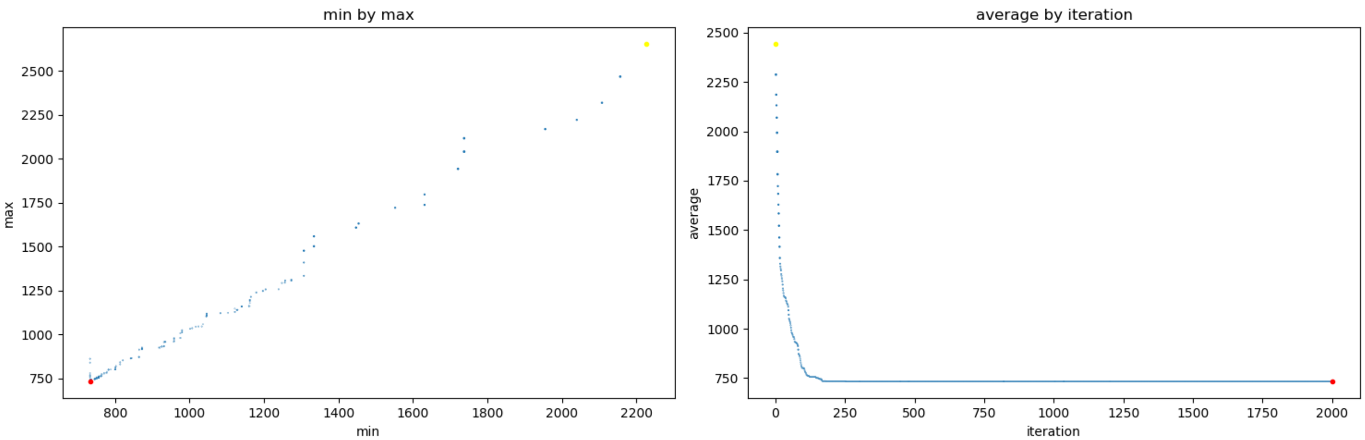

Figure 16: Here are plots (made in Python using `matplotlib`) showing the progression of the GA, e.g. it “learning” over iterations for 50 randomly-generated cities. This is telling us that the final solution achieved by the GA has a path length of about 750 which it achieved after only about 250 iterations (see the right plot). Surprisingly, this is almost half as many iterations as it took the 25-city version to converge; I honestly don’t have a great idea as to why this is. We can see the actual cities and the plot of the final solution in the image directly below.

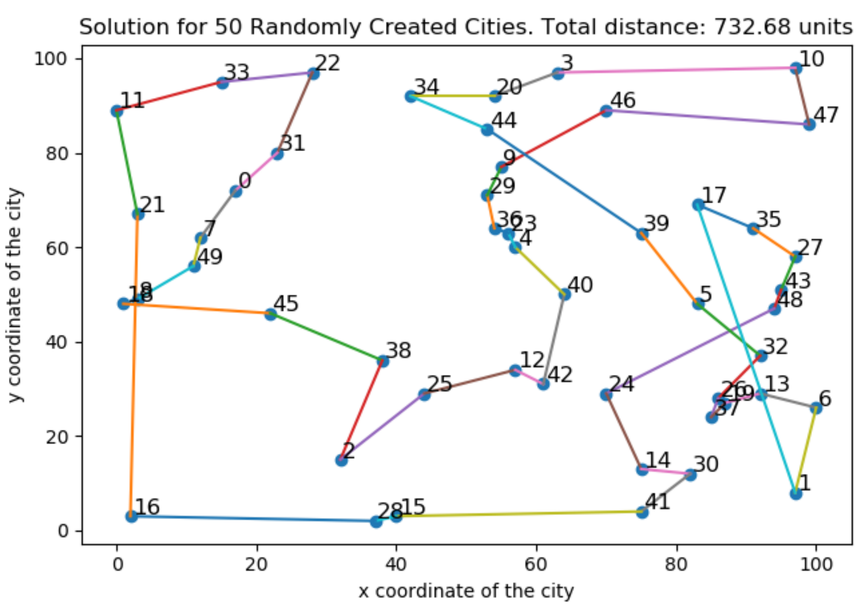

Figure 17: This plot (made in Python using `matplotlib`) shows the final path decided upon by the genetic algorithm for 50 cities in the traveling salesman problem. I used the `DEAP (Distributed Evolutionary Algorithms in Python)` package to do the solving. As with the 25-city version, we can see that this isn’t quite an ideal solution; we’d expect the most efficient path to be “totally hollow”, whereas this is certainly not hollow (the path crosses itself several times). All-in-all though, I wouldn’t say it’s a terrible solution. Certainly not as good as the integer version I got with Gurobi, but probably better than an average human could draw or come up with (especially in a matter of minutes).

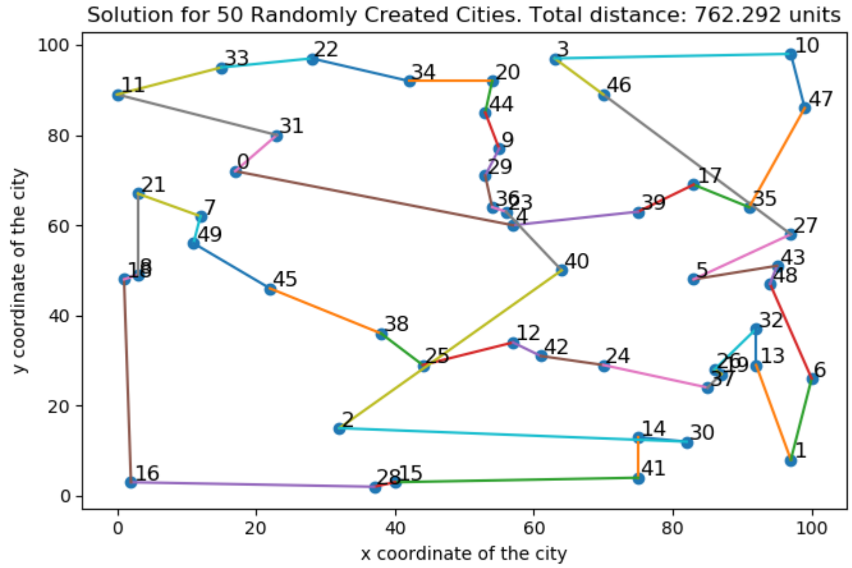

Figure 18: When we did the 25-city version we were able to set a different seed and generate a problem that seemed to yield an ideal solution with the GA solver (although it wasn't actually ideal). I was curious if we could do the same with the 50-city problem. I tried several different seeds, but never could come up with an ideal-looking solution as we did with the 25-city problem. This is an example of another one of the 50-city versions that doesn't quite look ideal when solved with the GA, but it is nice since it converges to roughly the same path length as the route in Figure 17.

Below is my code for this problem.

```
'''
File name      : hw7_prob7_TSP_ga.py
Author         : Jared Hansen
Date created   : 11/23/2019
Python version : 3.7.3

DESCRIPTION: The purpose of this script is to solve the TSP (Traveling
             Salesman Problem) using a genetic algorithm.

'''



#==============================================================================
#==============================================================================
#==== IMPORT STATEMENTS
#==============================================================================
#==============================================================================
import copy
import datetime as dt
import math
import matplotlib.pyplot as plt
import numpy as np
import random
import pylab as pl
from matplotlib import collections as mc
from pulp import *
from datetime import datetime
from deap import base, creator, tools
import string
from deap import base, creator, tools


#==============================================================================
#==============================================================================
#==== Runner CLASS : uses the DEAP toolbox to set up the GA, track performance
#====                metrics, and output the final population (population is
#====                composed of "individuals", e.g. paths around the cities).
#==============================================================================
#==============================================================================
class Runner:

    def __init__(self, toolbox):
        # Use the default toolbox (from DEAP).
        self.toolbox = toolbox
        # Set defaults for the parameters (modified in the method below).
        self.set_params(10, 5, 2)

    def set_params(self, pop_size, iters, num_matings):
        # Specifying population size, number of iterations, and number of
        # matings for our genetic algorithm.
        self.iters = iters
        self.pop_size = pop_size
        self.num_matings = num_matings

    def set_fitness(self, population):
        # Create a list of fitnesses: evaluate each individual in the
        # population based on the calc_path_len function below (a fitter
        # individual has a shorter path length).
        fitnesses = [
            (individual, self.toolbox.calc_path_len(individual))
            for individual in population
        ]
        for individual, fitness in fitnesses:
            individual.fitness.values = (fitness,)

    def get_offspring(self, population):
        # After an iteration of the GA has completed, use the old population
```

```
        # to create the new population (offspring).
        n = len(population)
        for _ in range(self.num_matings):
            i1, i2 = np.random.choice(range(n), size=2, replace=False)
            offspring1, offspring2 = \
                self.toolbox.mate(population[i1], population[i2])
            yield self.toolbox.mutate(offspring1)[0]
            yield self.toolbox.mutate(offspring2)[0]

    @staticmethod
    def return_stats(population, iteration=1):
        # Returns a dictionary that specifies performance metrics (mean,
        # std dev, max, min) for the passed-in population of individual paths.
        fitnesses = [ individual.fitness.values[0] for individual in population ]
        return {
            'i': iteration,
            'mu': np.mean(fitnesses),
            'std': np.std(fitnesses),
            'max': np.max(fitnesses),
            'min': np.min(fitnesses)
        }

    def Run(self):
        # Runs the GA. Sets the population, calculates performance metrics
        # ("stats" list below) for each iteration, creates offspring, and so on
        # until we've reached the specified number of iterations (num_iters
        # below).
        population = self.toolbox.population(n=self.pop_size)
        self.set_fitness(population)
        stats = []
        # Iteratively creates parent population, evaluates, uses to create
        # offspring population which becomes the next parent population, and so on.
        for iteration in list(range(1, self.iters + 1)):
            current_population = list(map(self.toolbox.clone, population))
            offspring = list(self.get_offspring(current_population))
            for child in offspring:
                current_population.append(child)
            self.set_fitness(current_population)
            population[:] = self.toolbox.select(current_population, len(population))
            stats.append(
                Runner.return_stats(population, iteration))
        return stats, population


#===============================================================================
#===============================================================================
#==== SPECIFYING AND RUNNING THE GA, PLOTTING FINAL RESULTS
#===============================================================================
#===============================================================================
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)
random.seed(11);
np.random.seed(121);
INDIVIDUAL_SIZE = NUMBER_OF_CITIES = n = 25
pop_size = 200
num_iters = 1000
num_matings = 50
'''
pop_size = 400
num_iters = 2000
num_matings = 100
'''

# Create labeled cities
cities = []
for i in range(n):
    # Append chars '0', '1', '2', ..., 'n' to the cities list
```

```python
    cities.append(str(i))

# Create coordinates for each city
random.seed(1)
points = [(random.randint(0,100),random.randint(0,100)) for i in range(n)]
cities_x = np.array(points)[:,0]
cities_y = np.array(points)[:,1]

# Plot the points that we've generated, with cities labeled by number
fig, tsp_img = plt.subplots()
tsp_img.scatter(cities_x, cities_y)
for i, cityNum in enumerate(cities):
    tsp_img.annotate(cityNum, xy=(cities_x[i], cities_y[i]), xytext=(1,1),
                    textcoords='offset points',
                    fontsize=9)
# Function for calculating the distance between two cities.
def calc_dist(city, to_city):
    dist = math.sqrt((points[city][0] - points[to_city][0])**2 +
                    (points[city][1] - points[to_city][1])**2)
    return(dist)

# Calculate the distances between each city.
distances = np.zeros((n, n))
for city in range(n):
    for to_city in [i for i in range(n) if not i == city]:
        distances[to_city][city] = distances[city][to_city] = calc_dist(city, to_city)

# Set additional GA specifications: toolbox, permutation regimen, and population
# setup.
toolbox = base.Toolbox()
toolbox.register("indices", random.sample, range(INDIVIDUAL_SIZE), INDIVIDUAL_SIZE)
toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.indices)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# Define the calc_path_len function to calculate the length of the individual path
# e.g. how long to travel along the specified cities path.
def calc_path_len(individual):
    summation = 0
    start = individual[0]
    for i in range(1, len(individual)):
        end = individual[i]
        summation += distances[start][end]
        start = end
    return summation
# Specifying some additional genetic algorithm settings.
toolbox.register("calc_path_len", calc_path_len)
toolbox.register("mate", tools.cxOrdered)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.01)
toolbox.register("select", tools.selTournament, tournsize=10)

# Actually run the genetic algorithm to get an answer.
a = Runner(toolbox)
a.set_params(pop_size, num_iters, num_matings)
stats, population = a.Run()

# Plot the "learning" results of the algorithm.
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
_ = plt.scatter([ s['min'] for s in stats ], [ s['max'] for s in stats ], marker='.', s=[ (s['std'] + 1) / 20 for s in stats ])
_ = plt.title('min by max')
_ = plt.xlabel('min')
_ = plt.ylabel('max')
_ = plt.plot(stats[0]['min'], stats[0]['max'], marker='.', color='yellow')
_ = plt.plot(stats[-1]['min'], stats[-1]['max'], marker='.', color='red')
plt.subplot(1,2,2)
_ = plt.scatter([ s['i'] for s in stats ], [ s['mu'] for s in stats ], marker='.', s=[ (s['std'] + 1) / 20 for s in stats ])
_ = plt.title('average by iteration')
```

```python
_ = plt.xlabel('iteration')
_ = plt.ylabel('average')
_ = plt.plot(stats[0]['i'], stats[0]['mu'], marker='.', color='yellow')
_ = plt.plot(stats[-1]['i'], stats[-1]['mu'], marker='.', color='red')
plt.tight_layout()
plt.show()

# See how long the best route took.
fitnesses = sorted([
    (i, toolbox.calc_path_len(individual))
    for i, individual in enumerate(population)
], key=lambda x: x[1])
best_fit = np.round(fitnesses[:1][0][1], 3)
calc_path_len(population[0])
best_path = list(population[0])

# Plot the best path over top of the points
fig, tsp_img = plt.subplots()
tsp_img.scatter(cities_x, cities_y)
for i, cityNum in enumerate(cities):
    tsp_img.annotate(cityNum, xy=(cities_x[i], cities_y[i]), xytext=(1,1),
                    textcoords='offset points',
                    fontsize=12)
for i in range(len(best_path)):
    if(i < len(best_path)-1):
        plt.plot([cities_x[best_path[i]], cities_x[best_path[i+1]]],
                [cities_y[best_path[i]], cities_y[best_path[i+1]]])
    else:
        plt.plot([cities_x[best_path[-1]], cities_x[best_path[0]]],
                [cities_y[best_path[-1]], cities_y[best_path[0]]])

plt.title('Solution for ' + str(NUMBER_OF_CITIES) + ' Randomly Created Cities. Total distance: ' + str(best_fit) + ' units')
#plt.title('Solution for ' + str(n) + ' part-B Cities. Total distance: ' + str(total_dist) + ' units')
plt.xlabel('x coordinate of the city')
plt.ylabel('y coordinate of the city')
plt.show()
```

8. The powered descent phase of a planar planetary landing is commonly posed as the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \sum_i ||u_i|| \\ \text{subject to} \quad & x_{i+1} = Ax_i + Bu_i + g, \forall i = 1, \ldots, N \\ & 0 < \rho_1 \leq ||u_i|| \leq \rho_2, \forall i = 1, \ldots, N \\ & x_1 = a, x_{N+1} = b \end{aligned}$$

The decision variables are $x_i \in \mathbb{R}^4 (i = 1, \ldots, N+1)$ and $u_i \in \mathbb{R}^2 (i = 1, \ldots, N)$. The x's represent the lander's state vector: horizontal range, vertical attitude, horizontal velocity, and vertical velocity. The u's represent the lander's control vector coming from the thruster in the horizontal and vertical directions.
The objective minimizes the amount of fuel consumed in the descent. The discrete difference equation approximates the differential equations describing the dynamics. There are no nonlinear terms (such as drag) because the thrust force between lower and upper bounds, i.e. the thruster cannot be turned off and it cannot provide infinitely large thrusts. Lastly, the lander is starting at a point $a \in \mathbb{R}^4$ and landing at a point on the surface $b \in \mathbb{R}^4$.

```
A = [1,0,0.1303,0; 0,1,0,0.1303; 0,0,1,0; 0,0,0,1];
B = [0.0085,0; 0,0.0085; 0.1303,0; 0,0.1303];
g = [0;-0.0832;0;-1.2779];
N = 499;
a = [1000;1500;-25;0];
b = [0;0;0;0];
rho1 = 4;
rho2 = 12;
```

Solve the problem using `fmincon`, `ga`, and `patternsearch`. Plot the state and control trajectories as a function of time.

- Before going into the (incorrect) results that I was able to get for this problem I'm going to describe the things that I did to try and solve it (mostly to give an idea of effort, as opposed to my actual output, which is poor.)
- One thing I spent quite a lot of time on was finding a good Python optimizer. My competence with Matlab is relegated to straightforward use of built-in functions like `linprog`, `fminunc`, `fmincon`, etc. Thus it made sense for me to devote some time to finding a good solver to use in Python (since I know Python fairly well).
- The one I landed on, and tried for quite awhile, is called `GEKKO`. It is built and maintained by a group at BYU. You can find it here if interested: https://apmonitor.com/wiki/index.php/Main/GekkoPythonOptimization It seemed like a good candidate since it can handle non-linear, non-convex problems like this one.
- The drawback of using this solver is all of the specific syntax used for setting up problems. What I mainly tried to do was use a couple of their examples. See the bottom two examples "Model Predictive Control" and "Optimization of Multiple Linked Phases" at this site https://gekko.readthedocs.io/en/latest/examples.html. I was never actually able to get code that ran and gave some kind of intelligible output, hence I've omitted that code from my submission.
- One thing I (kind of) got to work was using the `CVXPY` package in Python. Credit where credit is due, Ronak explained to me (after Dr. Harris explained to him) how this problem can be made convex by using an additional variable γ. Then I was able to use this package to try and solve the problem. (Again credit to Ronak, he helped me quite a lot in writing this code.)
- **Problems with convex formulation solution:** as can be seen in the figures below, the solution to which I've arrived is incorrect. This is most easily discerned in the plot of range VS altitude. Many of the value (for both range and altitude) are negative, which is nonsensical; this would mean that the aircraft is somehow below the planet's surface. I tried several things to fix this: modifying the objective (to take the norm instead of just sum of u_i values), and especially modifying the constraints. The most obvious way to fix this is to introduce constraints that specify that the first two elements of $x_i \in \mathbb{R}^4$ be values ≥ 0. I tried this (commented-out in the code below), but didn't have any luck. Since I'd already spent quite awhile on this problem and problem 7 (probably 6-7 hours between the two of them) I had to call it good here in order to work on homework for other classes.
- Plots below show the (incorrect) solution to which I arrived.

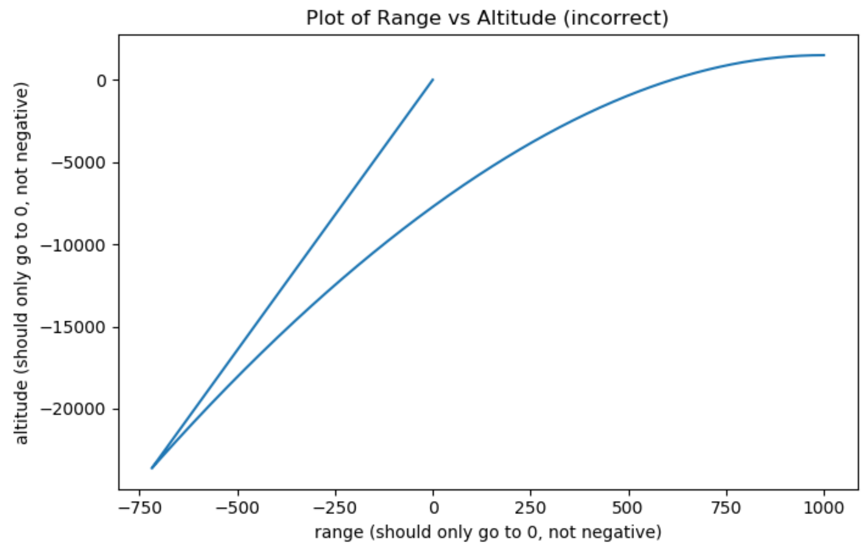

Figure 19: This plot, as mentioned above several times, shows that our solution is incorrect. Something in the formulation doesn't prevent negative values for range or altitude, which leads the solver to determine that going very negative back to (0,0) is the optimal solution. If we're looking hard for any correctness in the plot, at least it starts at the right point (1000,1500) and ends at the right point (0,0). (See later bullet points above for more detailed discussion on how I tried to fix this.)

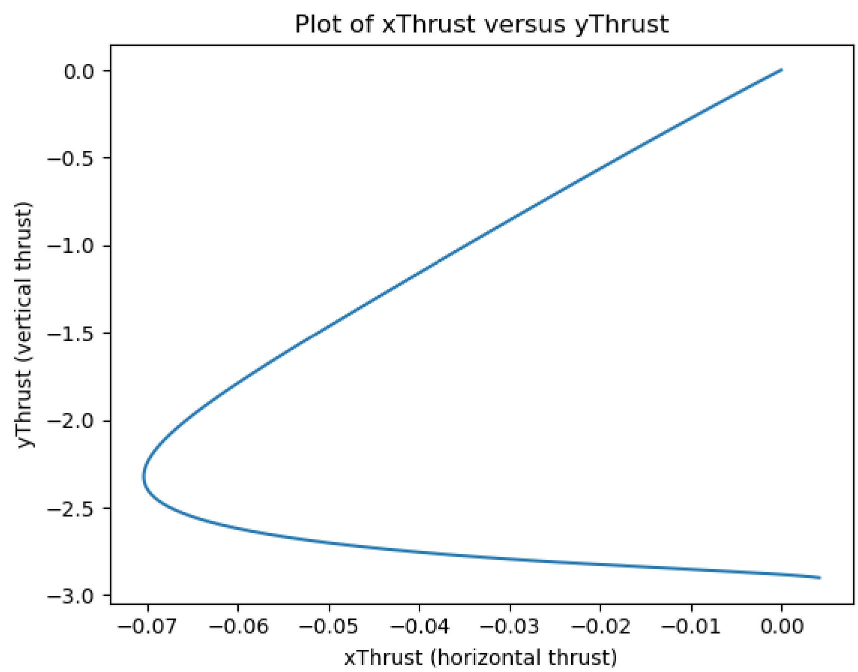

Figure 20: This plot shows horizontal thrust (xThrust) versus (yThrust). It also seem to be pretty incorrect, as we'd expect to see only positive values for yThrust, e.g. propelling the aircraft upward against gravitational forces such that the craft can land with a vertical velocity of 0 when it reaches (0,0).

Below is my code for this problem.

```python
'''
File name      : hw7_prob8.py
Author         : Jared Hansen
Date created   : 11/26/2019
Python version : 3.7.3

DESCRIPTION: The purpose of this script is to solve the planetary landing
             problem posed in question 8 in homework 7. Something in the
             formulation is off, since the plot obviously is wrong. It
             satisfies the constraints of x_1 = a, x_{N+1} = b, and starts at
             the correct x and u, but it somehow manages to go negative (e.g.
             crashing far into the ground before coming back up)

'''



#==============================================================================
#==== IMPORT STATEMENTS
#==============================================================================
import cvxpy as cp
import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as npla




#==============================================================================
#==== DEFINING CONSTANTS, GIVEN INFORMATION
#==============================================================================
# Dimensions of individual x and u vectors, number of time points.
n = 4
m = 2
T = 500
# Given values: A, B, g, N, a, b, rho1, rho2
A = np.matrix([[1, 0, 0.1303, 0     ],
               [0, 1, 0,      0.1303],
               [0, 0, 1,      0     ],
               [0, 0, 0,      1     ]])
B = np.matrix([[0.0085, 0     ],
               [0     , 0.0085],
               [0.1303, 0     ],
               [0     , 0.1303]])
g = np.array([0, -0.0832, 0, -1.2779])#.reshape(4,1)
a = np.array([1000, 1500, -25, 0])#.reshape(4,1)
b = np.array([0, 0, 0, 0])#.reshape(4,1)
rho1 = 4;
rho2 = 12;

# Defining variables for the convex solver. We introduce a new variable, gamma,
# to make the problem convex.
x = cp.Variable((n, T))
u = cp.Variable((m, T))
gamma = cp.Variable((1,T))


# Initialize the cost to 0, make constraints an empty list, then populate.
cost = 0
constr = []
for t in range(T-2):
    # Incrementing the cost function for each successive u vector
    cost += cp.sum((gamma[:,t])**2)
    # Defining the new constraints...basically these constraints are enforcing
    # that the states are connected, and that the magnitude of our thrust
    # vector is within the bounds of rho1 and rho2
    constr += [x[:,t+1] == A@x[:,t] + B@u[:,t] + g,
               cp.norm(u[:,t], 'inf') <= gamma[:,t],
               gamma[:,t] <= rho2,
```

```
            -gamma[:,t] <= -rho1,
            #gamma[:,t] >= rho1,
            #np.array([0,0,0,0]) <= -x[:,t],
            #np.array([0,0]) <= -u[:,t]
            ]
    '''
    # This is where the problem lies: there is nothing telling the algorithm
    # that we can't have negative thrust or negative position values. When I
    # try and implement these constraints the solver says that the objective
    # value of the solution is infinite, so this doesn't work with this solver.
    constr += [ -x[:,t] <= -np.array([0,0,0,0]),
              -u[:,t] <= np.array([0,0])  ]
    '''
# Adding the boundary constraints (must start at "a" and end at "b")
constr += [x[:,499] == b, x[:,0] == a]
# Define the problem with our completed cost and constraints list.
problem = cp.Problem(cp.Minimize(cost), constr)
# Have the solver determine optimal values for x and u.
problem.solve(solver=cp.ECOS)


#==============================================================================
#==== PLOTTING (INCORRECT) RESULTS
#==============================================================================
# Store the x's and u's in a nicer format (as 2-dimensional numpy arrays).
x_mat = x.value
u_mat = u.value
# Grab range and altitude for plotting.
x_range = x_mat[0,:]
x_altitude = x_mat[1,:]
# Grab the thrusts for plotting.
u_xThrust = u_mat[0,:]
u_yThrust = u_mat[1,:]

# Plot of range versus altitude
plt.plot(x_range, x_altitude)
plt.title("Plot of Range vs Altitude (incorrect)")
plt.xlabel('range (should only go to 0, not negative)')
plt.ylabel('altitude (should only go to 0, not negative)')

# Plot of xThrust versus yThrust
plt.plot(u_xThrust, u_yThrust)
plt.title('Plot of xThrust versus yThrust')
plt.xlabel('xThrust (horizontal thrust)')
plt.ylabel('yThrust (vertical thrust)')
```