EE364a Homework 5 solutions

1. A differentiable approximation of $\ell_1$-norm approximation. The function $\phi(u) = (u^2 + \epsilon)^{1/2}$, with parameter $\epsilon > 0$, is sometimes used as a differentiable approximation of the absolute value function $|u|$. To approximately solve the $\ell_1$-norm approximation problem

$$\text{minimize } \|Ax - b\|_1,$$

where $A \in \mathbb{R}^{m \times n}$, we solve instead the problem

$$\text{minimize } \sum_{i=1}^{m} \phi(a_i^T x - b_i),$$

where $a_i^T$ is the $i$th row of $A$. We assume $\text{rank } A = n$.

Let $p^\star$ denote the optimal value of the $\ell_1$-norm approximation problem (1). Let $\hat{x}$ denote the optimal solution of the approximate problem (2), and let $\hat{r}$ denote the associated residual, $\hat{r} = A\hat{x} - b$.

(a) Show that $p^\star \geq \sum_{i=1}^{m} \frac{\hat{r}_i^2}{\hat{r}_i^2 + \epsilon}^{1/2}$.

(b) Show that

$$\|A\hat{x} - b\|_1 \leq p^\star + \sum_{i=1}^{m} |\hat{r}_i| \left(1 - \frac{|\hat{r}_i|}{(\hat{r}_i^2 + \epsilon)^{1/2}}\right).$$

(By evaluating the righthand side after computing $\hat{x}$, we obtain a bound on how suboptimal $\hat{x}$ is for the $\ell_1$-norm approximation problem.)

\textbf{Solution.} One approach is based on duality. The point $\hat{x}$ minimizes the differentiable convex function $\sum_{i=1}^{m} \phi(a_i^T x - b_i)$, so its gradient vanishes:

$$\sum_{i=1}^{m} \phi'(\hat{r}_i)a_i = \sum_{i=1}^{m} \hat{r}_i(\hat{r}_i^2 + \epsilon)^{-1/2}a_i = 0.$$

Now, the dual of the $\ell_1$-norm approximation problem is

$$\text{maximize } \sum_{i=1}^{m} b_i \lambda_i$$

subject to $|\lambda_i| \leq 1$, $i = 1, \ldots, m$

$$\sum_{i=1}^{m} \lambda_i a_i = 0.$$

Thus, we see that the vector

$$\lambda_i = -\frac{\hat{r}_i}{(\hat{r}_i^2 + \epsilon)^{-1/2}}, \quad i = 1, \ldots, m,$$

is dual feasible. It follows that its dual function value,

$$\sum_{i=1}^{m} -b_i \lambda_i = -b_i \hat{r}_i \frac{\hat{r}_i}{(\hat{r}_i^2 + \epsilon)^{-1/2}},$$
provides a lower bound on \( p^* \). Now we use the fact that \( \sum_{i=1}^{m} \lambda_i a_i = 0 \) to obtain

\[
p^* \geq \sum_{i=1}^{m} -b_i \lambda_i \\
= \sum_{i=1}^{m} (a_i^T \hat{x} - b_i) \lambda_i \\
= \sum_{i=1}^{m} \hat{r}_i \lambda_i \\
= \frac{\hat{r}_i^2}{(\hat{r}_i^2 + \epsilon)^{1/2}}.
\]

Now we establish part (b). We start with the result above,

\[
p^* \geq \sum_{i=1}^{m} \hat{r}_i^2 / (\hat{r}_i^2 + \epsilon)^{1/2},
\]

and subtract \( \| A\hat{x} - b \|_1 = \sum_{i=1}^{m} |\hat{r}_i| \) from both sides to get

\[
p^* - \| A\hat{x} - b \|_1 \geq \sum_{i=1}^{m} \left( \frac{\hat{r}_i^2}{(\hat{r}_i^2 + \epsilon)^{1/2}} - |\hat{r}_i| \right).
\]

Re-arranging gives the desired result,

\[
\| A\hat{x} - b \|_1 \leq p^* + \sum_{i=1}^{m} |r_i| \left( 1 - \frac{|r_i|}{(r_i^2 + \epsilon)^{1/2}} \right).
\]

6.6 *Duals of some penalty function approximation problems.* Derive a Lagrange dual for the problem

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{m} \phi(r_i) \\
\text{subject to} & \quad r = Ax - b,
\end{align*}
\]

for the following penalty functions \( \phi : \mathbb{R} \to \mathbb{R} \). The variables are \( x \in \mathbb{R}^n, r \in \mathbb{R}^m \).

(b) *Huber penalty* (with \( M = 1 \)),

\[
\phi(u) = \begin{cases} 
    u^2 & |u| \leq 1 \\
    2|u| - 1 & |u| > 1.
\end{cases}
\]

(c) *Log-barrier* (with limit \( a = 1 \)),

\[
\phi(u) = -\log(1 - u^2), \quad \text{dom} \phi = (-1, 1).
\]
Solution. We first derive a dual for general penalty function approximation. The
Lagrangian is
\[
L(x, r, \lambda) = \sum_{i=1}^{m} \phi(r_i) + \nu^T (Ax - b - r).
\]
The minimum over \(x\) is bounded if and only if \(A^T \nu = 0\), so we have
\[
g(\nu) = \begin{cases}
-\nu^T b + \sum_{i=1}^{m} \inf_{r_i} (\phi(r_i) - \nu r_i) & A^T \nu = 0 \\
-\infty & \text{otherwise}.
\end{cases}
\]
Using
\[
\inf_{r_i} (\phi(r_i) - \nu r_i) = -\sup_{r_i} (\nu r_i - \phi(r_i)) = -\phi^*(\nu_i),
\]
we can express the general dual as
\[
\begin{align*}
\text{maximize} & \quad -\nu^T b - \sum_{i=1}^{m} \phi^*(\nu_i) \\
\text{subject to} & \quad A^T \nu = 0 \\
& \quad \|\nu\|_{\infty} \leq 2.
\end{align*}
\]
Now we’ll work out the conjugates of the given penalty functions.

(b) **Huber penalty.**

\[
\phi^*(z) = \begin{cases}
z^2/4 & |z| \leq 2 \\
\infty & \text{otherwise}
\end{cases}
\]
so we get the dual problem
\[
\begin{align*}
\text{maximize} & \quad -(1/4)\|\nu\|_2^2 - b^T \nu \\
\text{subject to} & \quad A^T \nu = 0 \\
& \quad \|\nu\|_\infty \leq 2.
\end{align*}
\]

(c) **Log-barrier.** The conjugate of \(\phi\) is

\[
\phi^*(z) = \sup_{|x| < 1} \left( xz + \log(1 - x^2) \right) = -1 + \sqrt{1 + z^2} + \log(-1 + \sqrt{1 + z^2}) - 2 \log |z| + \log 2.
\]
We can also derive the duals without using conjugates, by directly working out expressions for \(\inf_r (\phi(r) - \nu r)\).

(b) **Huber penalty.** We have

\[
\inf_r (\phi(r) - \nu r) = \begin{cases}
-\nu^2/4 & |\nu| \leq 1 \\
-\infty & \text{otherwise}
\end{cases}
\]
It is easy to verify graphically that the infimum is minus infinity if \(|\nu| > 1\). If \(-1 \leq \nu \leq 1\), we can set the derivative equal to zero and find that \(r = \nu/2\) is the minimizer. We get the dual problem
\[
\begin{align*}
\text{maximize} & \quad -(1/4)\|\nu\|_2^2 - b^T \nu \\
\text{subject to} & \quad A^T \nu = 0 \\
& \quad \|\nu\|_\infty \leq 2.
\end{align*}
\]
(c) *Log-barrier.* We evaluate
\[ \inf_{|r| < 1} (-\log(1 - r^2) - \nu r). \]
by setting the derivative equal to zero. This gives a quadratic equation \( \nu r^2 + 2r = \nu \). For \( \nu \neq 0 \) there are two roots \( r = (-1 \pm \sqrt{1 + \nu^2})/\nu \). Only the solution \( r = (-1 + \sqrt{1 + \nu^2})/\nu \) is in the interval \([-1, 1]\). Hence
\[ r = \begin{cases} 
(-1 + \sqrt{1 + \nu^2})/\nu & \nu \neq 0 \\
0 & \nu = 0.
\end{cases} \]
and
\[ \inf_{|r| < 1} (-\log(1 - r^2) - \nu r) = 1 - \sqrt{1 + \nu^2} - \log(-1 + \sqrt{1 + \nu^2}) + 2 \log |\nu| - \log 2. \]

The dual problem is
\[
\begin{align*}
\text{maximize} & \quad -b^T \nu - \sum_{i=1}^m \Big( \sqrt{1 + \nu_i^2} + \log(-1 + \sqrt{1 + \nu_i^2}) - 2 \log |\nu_i| \Big) + m - m \log 2 \\
\text{subject to} & \quad A^T \nu.
\end{align*}
\]

8.16 *Maximum volume rectangle inside a polyhedron.* Formulate the following problem as a convex optimization problem. Find the rectangle
\[ \mathcal{R} = \{ x \in \mathbb{R}^n \mid l \leq x \leq u \} \]
of maximum volume, enclosed in a polyhedron \( \mathcal{P} = \{ x \mid Ax \leq b \} \). The variables are \( l, u \in \mathbb{R}^n \). Your formulation should not involve an exponential number of constraints.

**Solution.** A straightforward, but very inefficient, way to express the constraint \( \mathcal{R} \subseteq \mathcal{P} \) is to use the set of \( m2^n \) inequalities \( Av^i \leq b \), where \( v^i \) are the \( (2^n) \) corners of \( \mathcal{R} \). (If the corners of a box lie inside a polyhedron, then the box does.) Fortunately it is possible to express the constraint in a far more efficient way. Define
\[ a_{ij}^+ = \max\{a_{ij}, 0\}, \quad a_{ij}^- = \max\{-a_{ij}, 0\}. \]
Then we have \( \mathcal{R} \subseteq \mathcal{P} \) if and only if
\[ \sum_{j=1}^n (a_{ij}^+ u_j - a_{ij}^- l_j) \leq b_i, \quad i = 1, \ldots, m, \]
The maximum volume rectangle is the solution of
\[
\begin{align*}
\text{maximize} & \quad (\prod_{i=1}^n (u_i - l_i))^{1/n} \\
\text{subject to} & \quad \sum_{j=1}^n (a_{ij}^+ u_j - a_{ij}^- l_j) \leq b_i, \quad i = 1, \ldots, m,
\end{align*}
\]
with implicit constraint \( u \geq l \). Another formulation can be found by taking the log of the objective, which yields
\[
\begin{align*}
\text{maximize} & \quad \sum_{i=1}^n \log(u_i - l_i) \\
\text{subject to} & \quad \sum_{j=1}^n (a_{ij}^+ u_j - a_{ij}^- l_j) \leq b_i, \quad i = 1, \ldots, m.
\end{align*}
\]
Minimum possible maximum correlation. Let $Z$ be a random variable taking values in $\mathbb{R}^n$, and let $\Sigma \in S_+^n$ be its covariance matrix. We do not know $\Sigma$, but we do know the variance of $m$ linear functions of $Z$. Specifically, we are given nonzero vectors $a_1, \ldots, a_m \in \mathbb{R}^n$ and $\sigma_1, \ldots, \sigma_m > 0$ for which
\[
\text{var}(a_i^T Z) = \sigma_i^2, \quad i = 1, \ldots, m.
\]
For $i \neq j$ the correlation of $Z_i$ and $Z_j$ is defined to be
\[
\rho_{ij} = \frac{\Sigma_{ij}}{\sqrt{\Sigma_{ii} \Sigma_{jj}}}.
\]
Let $\rho_{\text{max}} = \max_{i \neq j} |\rho_{ij}|$ be the maximum (absolute value) of the correlation among entries of $Z$. If $\rho_{\text{max}}$ is large, then at least two components of $Z$ are highly correlated (or anticorrelated).

(a) Explain how to find the smallest value of $\rho_{\text{max}}$ that is consistent with the given information, using convex or quasiconvex optimization. If your formulation involves a change of variables or other transformation, justify it.

(b) The file correlation_bounds_data.* contains $\sigma_1, \ldots, \sigma_m$ and the matrix $A$ with columns $a_1, \ldots, a_m$. Find the minimum value of $\rho_{\text{max}}$ that is consistent with this data. Report your minimum value of $\rho_{\text{max}}$, and give a corresponding covariance matrix $\Sigma$ that achieves this value. You can report the minimum value of $\rho_{\text{max}}$ to an accuracy of 0.01.

Solution.

(a) Using the formula for the variance of a linear function of a random variable, we have that
\[
\text{var}(a_i^T Z) = a_i^T \text{var}(Z) a_i = a_i^T \Sigma a_i,
\]
which is a linear function of $\Sigma$. We can find the minimum value of the maximum correlation among components of $Z$ that is consistent with the data by solving the following optimization problem.
\[
\begin{align*}
\text{minimize} & \quad \max_{i \neq j} |\rho_{ij}| \\
\text{subject to} & \quad a_i^T \Sigma a_i = \sigma_i^2, \quad i = 1, \ldots, m \\
\end{align*}
\]

Observe that
\[
|\rho_{ij}| = \frac{|\Sigma_{ij}|}{\sqrt{\Sigma_{ii} \Sigma_{jj}}}
\]
is a quasiconvex function of $\Sigma$: the numerator is a nonnegative convex function of $\Sigma$, and the denominator is a positive concave function of $\Sigma$ (it is the composition
of the geometric mean and a linear function of $\Sigma$). Since the maximum of quasi-convex functions is quasiconvex, the objective in the optimization problem above is quasiconvex. Thus, we can find the smallest value of $\rho_{\text{max}}$ by solving a quasi-convex optimization problem. In particular, we have that $\rho_{\text{max}} \leq t$ is consistent with the data if and only if the following convex feasibility problem is feasible:

$$ |\Sigma_{ij}| \leq t \sqrt{\Sigma_{ii} \Sigma_{jj}}, \quad i \neq j $$

$$ a_i^T \Sigma a_i = \sigma_i^2, \quad i = 1, \ldots, m $$

$$ \Sigma \succeq 0 $$

We can find the minimum value of $t$ for which this problem is feasible using bisection search starting with $t = 0$ and $t = 1$.

(b) The following Matlab code solves the problem.

```matlab
% find the minimum possible maximum correlation
lb = 0;
ub = 1;
Sigma_opt = nan(n,n);
while ub-lb > 1e-3
    t = (lb+ub)/2;
    cvx_begin sdp czz
    variable Sigma(n,n) symmetric
    for i = 1:(n-1)
        for j = (i+1):n
            abs(Sigma(i,j)) <= t * geo_mean([Sigma(i,i), Sigma(j,j)])
        end
    end
    for i = 1:m
        A(:,i)' * Sigma * A(:,i) == sigma(i)^2
    end
    Sigma >= 0
    cvx_end
    if strcmp(cvx_status, 'Solved')
        ub = t;
        Sigma_opt = Sigma;
    else
        lb = t;
    end
end
```

6
% print the results and check the correlation matrix

\[
Sigma = Sigma_opt \\
C = \text{diag}(1./\sqrt{\text{diag}(Sigma)}); \\
R = C \cdot Sigma \cdot C; \\
rho_max = \max(\max(R - \text{diag}(\text{diag}(R))))
\]

The following Python code solves the problem.

```python
import cvxpy as cvx
from math import sqrt
from correlation_bounds_data import *

Sigma = cvx.Semidef(n)
t = cvx.Parameter(sign='positive')
rho_cons = []
for i in range(n - 1):
    for j in range(i + 1, n):
        Sij = cvx.vstack(Sigma[i, i], Sigma[j, j])
        rho_cons += [cvx.abs(Sigma[i, j]) <= t * cvx.geo_mean(Sij)]
var_cons = [A[:, i].T * Sigma * A[:, i] == sigma[i]**2 for i in range(m)]
problem = cvx.Problem(cvx.Minimize(0), rho_cons + var_cons)

lb, ub = 0.0, 1.0
Sigma_opt = None
while ub - lb > 1e-3:
    t.value = (lb + ub) / 2
    problem.solve()
    if problem.status == cvx.OPTIMAL:
        ub = t.value
        Sigma_opt = Sigma.value
    else:
        lb = t.value

print('rho_max =', t.value)
print('Sigma =', Sigma_opt)

# compute the correlation matrix
C = np.diag([1 / sqrt(Sigma_opt[i, i]) for i in range(n)])
R = C * Sigma_opt * C
print('R =', R)
```
The following Julia code solves the problem.

using Convex, SCS
set_default_solver(SCSSolver(verbose=false))

include("correlation_bounds_data.jl")

lb = 0; ub = 1
t = (lb+ub)/2
Sigma_opt = []
while ub-lb > 1e-3
    t = (lb+ub)/2
    Sigma = Semidefinite(n)
    problem = satisfy()
    for i = 1:(n-1)
        for j = (i+1):n
            problem.constraints += (abs(Sigma[i,j]) <= t*geomean(Sigma[i,i],Sigma[j,j]))
        end
    end
    for i = 1:m
        problem.constraints += (A[:,i]' * Sigma * A[:,i] == sigma[i]^2)
    end
    solve!(problem)
    if problem.status == :Optimal
        ub = t
        Sigma_opt = Sigma.value
    else
        lb = t
    end
end
println("t = $(round(t,4))")
println("Sigma = $(round(Sigma_opt,4))")

# compute the correlation matrix
C = diagm(Float64[1/sqrt(Sigma_opt[i,i]) for i in 1:n])
R = C * Sigma_opt * C
println("R = $(round(R,4))")

We find that the minimum value of the maximum correlation that is consistent
with the data is \( \rho^{\text{max}} = 0.62 \). A corresponding covariance matrix is

\[
\Sigma = \begin{bmatrix}
3.78 & 0.30 & 1.46 & 1.47 & 0.24 \\
0.30 & 2.91 & -1.28 & 1.29 & 0.86 \\
1.46 & -1.28 & 1.46 & 0.09 & 0.27 \\
1.47 & 1.29 & 0.09 & 1.48 & 0.49 \\
0.24 & 0.09 & 0.27 & 0.49 & 0.42
\end{bmatrix}.
\]

Although we did not ask you to do so, it is a good idea to compute the correlation matrix to this value of \( \Sigma \), and check that the maximum correlation is equal to \( \rho^{\text{max}} \):

\[
R = \begin{bmatrix}
1.00 & 0.09 & 0.62 & 0.62 & 0.19 \\
0.09 & 1.00 & -0.62 & 0.62 & 0.08 \\
0.62 & -0.62 & 1.00 & 0.06 & 0.34 \\
0.62 & 0.62 & 0.06 & 1.00 & 0.62 \\
0.19 & 0.08 & 0.34 & 0.62 & 1.00
\end{bmatrix}.
\]

10.1 Some famous inequalities. The Cauchy-Schwarz inequality states that

\[ |a^T b| \leq \|a\|_2 \|b\|_2 \]

for all vectors \( a, b \in \mathbb{R}^n \) (see page 633 of the textbook).

(a) Prove the Cauchy-Schwarz inequality.

*Hint.* A simple proof is as follows. With \( a \) and \( b \) fixed, consider the function \( g(t) = \|a + tb\|_2^2 \) of the scalar variable \( t \). This function is nonnegative for all \( t \). Find an expression for \( \inf_t g(t) \) (the minimum value of \( g \)), and show that the Cauchy-Schwarz inequality follows from the fact that \( \inf_t g(t) \geq 0 \).

(b) The 1-norm of a vector \( x \) is defined as \( \|x\|_1 = \sum_{k=1}^{n} |x_k| \). Use the Cauchy-Schwarz inequality to show that

\[ \|x\|_1 \leq \sqrt{n} \|x\|_2 \]

for all \( x \).

(c) The harmonic mean of a positive vector \( x \in \mathbb{R}^n_{++} \) is defined as

\[ \left( \frac{1}{n} \sum_{k=1}^{n} \frac{1}{x_k} \right)^{-1}. \]

Use the Cauchy-Schwarz inequality to show that the arithmetic mean \( (\sum_k x_k)/n \) of a positive \( n \)-vector is greater than or equal to its harmonic mean.

*Solution.*
(a) First note that the inequality is trivially satisfied if \( b = 0 \). Assume \( b \neq 0 \). We write \( g \) as

\[
g(t) = (a - tb)^T(a - tb) = \|a\|^2_2 + 2ta^Tb + t^2\|b\|^2_2.
\]

Setting the derivative equal to zero gives \( t = -(a^Tb)/\|b\|^2 \). Therefore

\[
\inf_t g(t) = \|a\|^2_2 - \frac{(a^Tb)^2}{\|b\|^2}.
\]

The Cauchy-Schwarz inequality now follows from \( \inf g(t) \geq 0 \).

(b) Define \( a_k = |x_k|, b_k = 1 \). Then

\[
\|x\|_1 = a^Tb \leq \|a\|_2\|b\|_2 = \sqrt{n}\|x\|_2.
\]

(c) Define \( a_k = \sqrt{x_k/n} \) and \( b_k = 1/\sqrt{nx_k} \). Then

\[
1 = (a^Tb)^2 \leq \|a\|^2_2 \|b\|^2_2 = \left(\frac{1}{n} \sum_{k=1}^n x_k \right) \left(\sum_{k=1}^n \frac{1}{nx_k} \right).
\]

Hence,

\[
\frac{1}{n} \sum_{k} x_k \geq \frac{n}{\sum_k 1/x_k}.
\]

14.8 **Optimal spacecraft landing.** We consider the problem of optimizing the thrust profile for a spacecraft to carry out a landing at a target position. The spacecraft dynamics are

\[
m\ddot{p} = f - mge_3,
\]

where \( m > 0 \) is the spacecraft mass, \( p(t) \in \mathbb{R}^3 \) is the spacecraft position, with 0 the target landing position and \( p_3(t) \) representing height, \( f(t) \in \mathbb{R}^3 \) is the thrust force, and \( g > 0 \) is the gravitational acceleration. (For simplicity we assume that the spacecraft mass is constant. This is not always a good assumption, since the mass decreases with fuel use. We will also ignore any atmospheric friction.) We must have \( p(T^d) = 0 \) and \( \dot{p}(T^d) = 0 \), where \( T^d \) is the touchdown time. The spacecraft must remain in a region given by

\[
p_3(t) \geq \alpha\|(p_1(t), p_2(t))\|_2,
\]

where \( \alpha > 0 \) is a given minimum glide slope. The initial position \( p(0) \) and velocity \( \dot{p}(0) \) are given.

The thrust force \( f(t) \) is obtained from a single rocket engine on the spacecraft, with a given maximum thrust; an attitude control system rotates the spacecraft to achieve any desired direction of thrust. The thrust force is therefore characterized by the constraint

\[
\|f(t)\|_2 \leq F^{\max}.
\]

The fuel use rate is proportional to the thrust force magnitude, so the total fuel use is

\[
\int_0^{T^d} \gamma\|f(t)\|_2 \, dt,
\]
where $\gamma > 0$ is the fuel consumption coefficient. The thrust force is discretized in time, i.e., it is constant over consecutive time periods of length $h > 0$, with $f(t) = f_k$ for $t \in [(k-1)h, kh)$, for $k = 1, \ldots, K$, where $T^{td} = Kh$. Therefore we have

$$\begin{align*}
v_{k+1} &= v_k + \frac{h}{m}f_k - hge_3, \\
p_{k+1} &= p_k + \frac{h}{2}(v_k + v_{k+1}),
\end{align*}$$

where $p_k$ denotes $p((k-1)h)$, and $v_k$ denotes $\dot{p}((k-1)h)$. We will work with this discrete-time model. For simplicity, we will impose the glide slope constraint only at the times $t = 0, h, 2h, \ldots, Kh$.

(a) **Minimum fuel descent.** Explain how to find the thrust profile $f_1, \ldots, f_K$ that minimizes fuel consumption, given the touchdown time $T^{td} = Kh$ and discretization time $h$.

(b) **Minimum time descent.** Explain how to find the thrust profile that minimizes the touchdown time, i.e., $K$, with $h$ fixed and given. Your method can involve solving several convex optimization problems.

(c) Carry out the methods described in parts (a) and (b) above on the problem instance with data given in spacecraft_landing_data.*. Report the optimal total fuel consumption for part (a), and the minimum touchdown time for part (b). The data files also contain plotting code (commented out) to help you visualize your solution. Use the code to plot the spacecraft trajectory and thrust profiles you obtained for parts (a) and (b).

**Hints.**

- In Julia, the plot will come out rotated.

**Remarks.** If you’d like to see the ideas of this problem in action, watch these videos:

- [http://www.youtube.com/watch?v=2t15vP1PyoA](http://www.youtube.com/watch?v=2t15vP1PyoA)
- [https://www.youtube.com/watch?v=orUjSkc2pG0](https://www.youtube.com/watch?v=orUjSkc2pG0)
- [https://www.youtube.com/watch?v=1B6oiLNyKKI](https://www.youtube.com/watch?v=1B6oiLNyKKI)
- [https://www.youtube.com/watch?v=ZCBE8oc0kAQ](https://www.youtube.com/watch?v=ZCBE8oc0kAQ)

**Solution.**

(a) To find the minimum fuel thrust profile for a given $K$, we solve

$$\begin{align*}
\text{minimize} & \quad \sum_{k=1}^{K} \|f_k\|_2 \\
\text{subject to} & \quad v_{k+1} = v_k + \frac{h}{m}f_k - hge_3, \\
& \quad p_{k+1} = p_k + \frac{h}{2}(v_k + v_{k+1}), \\
& \quad \|f_k\|_2 \leq F^{\text{max}}, \\
& \quad (p_k)_3 \geq \alpha \|(p_k)_1, (p_k)_2\|_2, \\
& \quad p_{K+1} = 0, \\
& \quad v_{K+1} = 0, \\
& \quad p_1 = p(0), \\
& \quad v_1 = \dot{p}(0),
\end{align*}$$

with variables $p_1, \ldots, p_{K+1}$, $v_1, \ldots, v_{K+1}$, and $f_1, \ldots, f_K$. This is a convex optimization problem.
(b) We can solve a sequence of convex feasibility problems to find the minimum touchdown time. For each $K$ we solve

$$\begin{align*}
\text{minimize} & \quad 0 \\
\text{subject to} & \quad v_{k+1} = v_k + (h/m)f_k - hge_3, \quad p_{k+1} = p_k + (h/2)(v_k + v_{k+1}), \\
& \quad \|f_k\|_2 \leq F_{\text{max}}, \quad (p_k)_3 \geq \alpha \|(p_k)_1, (p_k)_2\|_2, \quad k = 1, \ldots, K \\
& \quad p_{K+1} = 0, \quad v_{K+1} = 0, \quad p_1 = p(0), \quad v_1 = \dot{p}(0),
\end{align*}$$

with variables $p_1, \ldots, p_{K+1}, v_1, \ldots, v_{K+1}$, and $f_1, \ldots, f_K$. If the problem is feasible, we reduce $K$, otherwise we increase $K$. We iterate until we find the smallest $K$ for which a feasible trajectory can be found. (In fact, the problem is quasiconvex as long as $(1/m)F_{\text{max}} \geq g$, so we can use bisection to speed up our search.)

(c) For part (a) the optimal total fuel consumption is 193.0. For part (b) the minimum touchdown time is $K = 25$. The following plots show the trajectories we obtain. The blue line shows the position of the spacecraft, the black arrows show the thrust profile, and the colored surface shows the glide slope constraint.

Here is the minimum fuel trajectory for part (a). Notice that for a portion of the trajectory the thrust is exactly equal to zero (which we would expect, given our cost function).

![Minimum fuel trajectory for part (a)](image1)

Here is a minimum time trajectory for part (b).
The following code solves the problem. In Matlab:

```matlab
spacecraft_landing_data;

% solve part (a) (find minimum fuel trajectory)
cvx_solver sdpt3;
cvx_begin
    variables p(3,K+1) v(3,K+1) f(3,K)
    v(:,2:K+1) == v(:,1:K)+(h/m)*f-h*g*repmat([0;0;1],1,K);
    p(:,2:K+1) == p(:,1:K)+(h/2)*(v(:,1:K)+v(:,2:K+1));
    p(:,1) == p0; v(:,1) == v0;
    p(:,K+1) == 0; v(:,K+1) == 0;
    p(3,:) >= alpha*norms(p(1:2,:));
    norms(f) <= Fmax;
    minimize(sum(norms(f)))
cvx_end
min_fuel = cvx_optval*gamma*h;
p_minf = p; v_minf = v; f_minf = f;

% solve part (b) (find minimum K)
% we will use a linear search, but bisection is faster
Ki = K;
while(1)
    cvx_begin
    variables p(3,Ki+1) v(3,Ki+1) f(3,Ki)
    v(:,2:Ki+1) == v(:,1:Ki)+(h/m)*f-h*g*repmat([0;0;1],1,Ki);
```
\[ p(:,2:Ki+1) = p(:,1:Ki)+(h/2)*(v(:,1:Ki)+v(:,2:Ki+1)); \]
\[ p(:,1) = p0; v(:,1) = v0; \]
\[ p(:,Ki+1) = 0; v(:,Ki+1) = 0; \]
\[ p(3,:) \geq \alpha*\|\|p(1:2,:);\|\|; \]
\[ \|\|f\|\| \leq F_{\text{max}}; \]
\[ \text{minimize} \left( \sum \|\|f\|\| \right) \]
\text{cvx_end}

if(strcmp(cvx_status,'Infeasible') == 1)
    Kmin = Ki+1;
    break;
end

Ki = Ki-1;
p_mink = p; v_mink = v; f_mink = f;
end

%% plot the glide cone
x = linspace(-40,55,30); y = linspace(0,55,30);
[X,Y] = meshgrid(x,y);
Z = \alpha*\sqrt{X.^2+Y.^2};
figure; colormap autumn; surf(X,Y,Z);
axis([-40,55,0,55,0,105]); grid on; hold on;

%% plot minimum fuel trajectory for part (a)
plot3(p_minf(1,:),p_minf(2,:),p_minf(3,:),'b','linewidth',1.5);
quiver3(p_minf(1,1:K),p_minf(2,1:K),p_minf(3,1:K),...
f_minf(1,:),f_minf(2,:),f_minf(3,:),0.3,'k','linewidth',1.5);
print('-depsc','spacecraft_landing_a.eps');

%% plot minimum time trajectory for part (b)
figure; colormap autumn; surf(X,Y,Z);
axis([-40,55,0,55,0,105]); grid on; hold on;
plot3(p_mink(1,:),p_mink(2,:),p_mink(3,:),'b','linewidth',1.5);
quiver3(p_mink(1,1:Kmin),p_mink(2,1:Kmin),p_mink(3,1:Kmin),...
f_mink(1,:),f_mink(2,:),f_mink(3,:),0.3,'k','linewidth',1.5);
print('-depsc','spacecraft_landing_b.eps');

In Python:

```python
import numpy as np
import cvxpy as cvx
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
```
h = 1.  
g = 0.1  
m = 10.  
Fmax = 10.  
p0 = np.matrix('50;50;100')  
v0 = np.matrix('-10;0;-10')  
alpha = 0.5  
gamma = 1.  
K = 35  

# Solution begins
#--------------------------------------------------------------

e3 = np.matrix('0; 0; 1')

p = cvx.Variable(3, K+1)  
v = cvx.Variable(3, K+1)  
f = cvx.Variable (3, K)

fuel_use = h*gamma*sum([cvx.norm(f[:,i]) for i in range(K)])

const = [v[:,i+1] == v[:,i] + (h/m)*f[:,i]-h*g*e3 for i in range(K)]
const += [p[:,i+1] == p[:,i] + h/2*(v[:,i]+v[:,i+1]) for i in range(K)]
const += [p[:,0]==p0, v[:,0]==v0]
const += [p[:,K]==0, v[:,K]==0]
const += [p[2,i] >= alpha*cvx.norm(p[0:2,i]) for i in range(K+1)]
const += [cvx.norm(f[:,i]) <= Fmax for i in range(K)]

prob = cvx.Problem(cvx.Minimize(fuel_use), const)
prob.solve()
print 'Minimum fuel use is %.2f' % fuel_use.value

# Minimum fuel trajectory and glide cone
fig = plt.figure()  
ax = fig.gca(projection='3d')

X = np.linspace(-40, 55, num=30)  
Y = np.linspace(0, 55, num=30)  
X, Y = np.meshgrid(X, Y)  
Z = alpha*np.sqrt(X**2+Y**2);  
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm, linewidth=0)
# For minimum time descent, we do a linear search but bisection would be faster.

K = 1
while True:
    p = cvx.Variable(3, K+1)
    v = cvx.Variable(3, K+1)
    f = cvx.Variable(3, K)

    const = [v[:,i+1] == v[:,i] + (h/m)*f[:,i]-h*g*e3 for i in range(K)]
    const += [p[:,i+1] == p[:,i] + h/2*(v[:,i]+v[:,i+1]) for i in range(K)]
    const += [p[:,0]==p0, v[:,0]==v0]
    const += [p[:,K]==0, v[:,K]==0]
    const += [p[2,i] >= alpha*cvx.norm(p[0:2,i]) for i in range(K+1)]
    const += [cvx.norm(f[:,i]) <= Fmax for i in range(K)]

    prob = cvx.Problem(cvx.Minimize(0), const)
    prob.solve()

    if prob.status=='optimal':
        break
    K += 1

print 'The minimum touchdown time is', K

# Minimum time trajectory and glide cone
fig = plt.figure()
ax = fig.gca(projection='3d')
X = np.linspace(-40, 55, num=30)
Y = np.linspace(0, 55, num=30)
Z = alpha*np.sqrt(X**2+Y**2);
In Julia:

```julia
include("spacecraft_landing_data.jl");
using Convex, SCS
solver = SCSSolver(max_iters=20000, verbose=false);

# solve part (a) (find minimum fuel trajectory)
p = Variable(3, K+1);
v = Variable(3, K+1);
f = Variable(3, K);
constraints = [];
constraints += v[:,2:K+1] == v[:,1:K] + (h/m)*f - h*g*repmat([0,0,1], 1, K);
constraints += p[:,2:K+1] == p[:,1:K] + (h/2)*(v[:,1:K] + v[:,2:K+1]);
constraints += p[:,1] == p0;
constraints += v[:,1] == v0;
constraints += p[:,K+1] == 0;
constraints += v[:,K+1] == 0;
for i = 1:K+1
    constraints += p[3,i] >= alpha*norm(p[1:2,i]);
end
objective = 0;
for i = 1:K
    constraints += norm(f[:,i]) <= Fmax;
    objective += norm(f[:,i]);
end
problem = minimize(objective, constraints);
solve!(problem, solver);
min_fuel = problem.optval*gamma*h;
p_minf = p.value'; v_minf = v.value'; f_minf = f.value';

# solve part (b) (find minimum K)
# we will use a linear search, but bisection is faster
Kmin = K;
p_mink = nothing;
v_mink = nothing;
f_mink = nothing;
while true
    p = Variable(3, Kmin+1);
v = Variable(3, Kmin+1);
f = Variable(3, Kmin);
constraints = [];
constraints += v[:,2:Kmin+1] == v[:,1:Kmin] + (h/m)*f - h*g*repmat([0,0,1], 1, K);
constraints += p[:,2:Kmin+1] == p[:,1:Kmin] + (h/2)*(v[:,1:Kmin] + v[:,2:Kmin+1]);
constraints += p[:,1] == p0;
constraints += v[:,1] == v0;
constraints += p[:,Kmin+1] == 0;
constraints += v[:,Kmin+1] == 0;
for i = 1:Kmin+1
    constraints += p[3,i] >= alpha*norm(p[1:2,i]);
end
objective = 0;
for i = 1:Kmin
    constraints += norm(f[:,i]) <= Fmax;
    objective += norm(f[:,i]);
end
problem = minimize(objective, constraints);
solve!(problem, solver);
min_fuel = problem.optval*gamma*h;
p_mink = p.value'; v_mink = v.value'; f_mink = f.value';

17
```
constraints += p[:,1] == p0;
constraints += v[:,1] == v0;
constraints += p[:,Kmin+1] == 0;
constraints += v[:,Kmin+1] == 0;
for i = 1:Kmin+1
    constraints += p[3,i] >= alpha*norm(p[1:2,i]);
end
objective = 0;
for i = 1:Kmin
    constraints += norm(f[:,i]) <= Fmax;
    objective += norm(f[:,i]);
end
problem = minimize(objective, constraints);
solve!(problem, solver);
if problem.status != :Optimal
    Kmin += 1;
    break;
end
Kmin -= 1;
p_mink = p.value'; v_mink = v.value'; f_mink = f.value';
end

# plot the glide cone
using PyPlot
x = linspace(-40,55,30); y = linspace(0,55,30);
X = repmat(x', length(x), 1);
Y = repmat(y, 1, length(y));
Z = alpha*sqrt(X.^2+Y.^2);
figure();
grid(true);
hold(true);
surf(X, Y, Z, cmap=get_cmap("autumn"));
xlim([-40, 55]);
ylim([0, 55]);
zlim([0, 105]);

# plot minimum fuel trajectory for part (a)
plot(p_minf[:,1], p_minf[:,2], p_minf[:,3]);
heads = p_minf[1:K, :] + f_minf;
quiver(heads[:,1], heads[:,2], heads[:,3],
        f_minf[:,1], f_minf[:,2], f_minf[:,3], length=10);
# plot minimum time trajectory for part (b)
figure();
grid(true);
hold(true);
surf(X, Y, Z, cmap=get_cmap("autumn"));
xlim([-40, 55]);
ylim([0, 55]);
zlim([0, 105]);
plot(p_mink[:,1], p_mink[:,2], p_mink[:,3]);
heads_k = p_mink[1:Kmin,] + f_mink;
quiver(heads_k[:,1], heads_k[:,2], heads_k[:,3],
       f_mink[:,1], f_mink[:,2], f_mink[:,3], length=10);