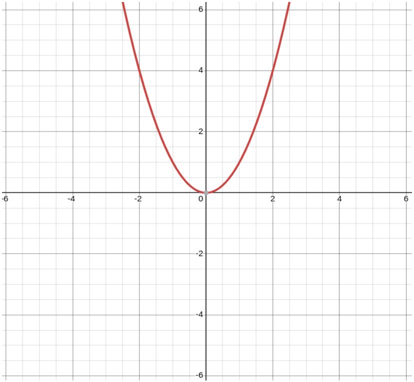
Relaxation for Optimization

Northeastern University Directed Reading Program Spring 2024

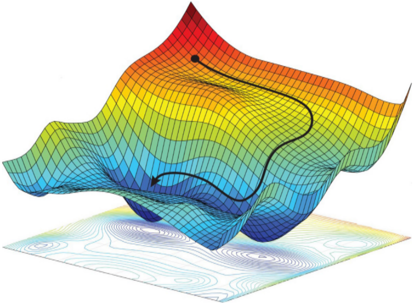
Daniel Ma
Mentored by: Forrest Miller

April 11^{th}, 2024

What Is Optimization?



(a) $f(x) = x^2$



(b) 3-D optimization

Figure 1: Optimization seeks to find the extrema of functions

Convex Optimization

General Optimization Problem

$$\min_{\theta \in \mathbb{R}^d} \{c(\theta) | h_i(\theta) = 0, \ i = 1, \ldots, N\}$$

Convex Optimization

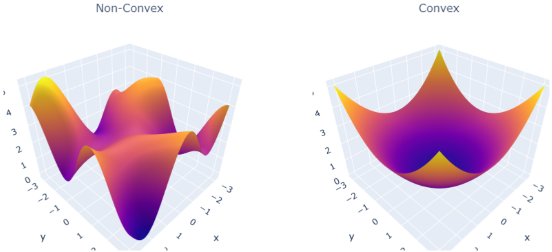
General Optimization Problem

$$\min_{\theta \in \mathbb{R}^d} \{c(\theta) | h_i(\theta) = 0,\ i = 1, \ldots, N\}$$

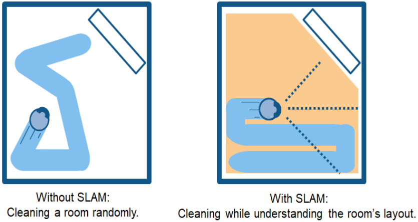
Definition

A function is convex if, $\forall x, y \in f(x)$, f(x) $\leq$ the line between the x and y.

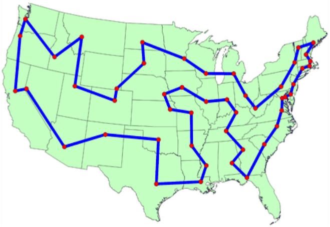
$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y),\ \lambda \in [0, 1]$$



Real World Examples



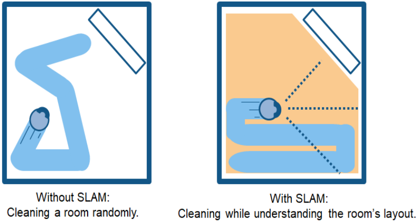
(a) Robot Vacuum Cleaner



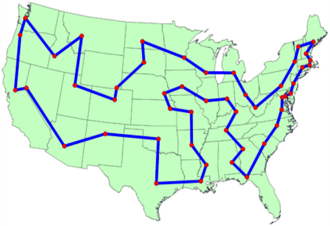
(b) Travelling Salesman

Figure 2: Some application areas of optimization

Real World Examples



(a) Robot Vacuum Cleaner



(b) Travelling Salesman

Figure 2: Some application areas of optimization

Real world problems are rarely convex and computationally simple

Autotight Algorithm

Problem: Nonconvex optimization is hard (in general)

Autotight Algorithm

Problem: Nonconvex optimization is hard (in general)
Solution: Relax nonconvex problems into semidefinite programs (SDP)

$$\min_{\theta \in \mathbb{R}^d} \{c(\theta) | h_i(\theta) = 0, \ i \in [N_h]\} \Rightarrow \min_{X \in S_+^N} \{\langle Q, X \rangle | \langle A_i, X \rangle = 0, i \in [N_A]\},$$

Remarkably effective for real world problems

Autotight Algorithm

Problem: Nonconvex optimization is hard (in general)
Solution: Relax nonconvex problems into semidefinite programs (SDP)

$$\min_{\theta \in \mathbb{R}^d} \{c(\theta) | h_i(\theta) = 0,\ i \in [N_h]\} \Rightarrow \min_{X \in S_+^N} \{\langle Q, X \rangle | \langle A_i, X \rangle = 0, i \in [N_A]\},$$

Remarkably effective for real world problems
New Problem: Gap between nonconvex solution and relaxed SDP solution is not tight and finding appropriate A_i is hard

Autotight Algorithm

Problem: Nonconvex optimization is hard (in general)
Solution: Relax nonconvex problems into semidefinite programs (SDP)

$$\min_{\theta \in \mathbb{R}^d} \{c(\theta) | h_i(\theta) = 0,\ i \in [N_h]\} \Rightarrow \min_{X \in S_+^N} \{\langle Q, X \rangle | \langle A_i, X \rangle = 0, i \in [N_A]\},$$

Remarkably effective for real world problems
New Problem: Gap between nonconvex solution and relaxed SDP solution is not tight and finding appropriate A_i is hard

Definition

Autotight is a recently proposed procedure which automatically tightens an SDP's solution to its nonrelaxed counterpart.

So, to solve a nonconvex problem,

1. Lift original problem to a SDP, getting a lower bound to solution
2. Utilize Autotight to automatically tighten the SDP's solution to the original problem's solution

Example Problem

$$\min_{\theta} \sum_{i=1}^{N} \left(\frac{1}{\theta - a_i} \right)^2 \tag{1}$$

$$\min_{\theta \in \mathbb{R}^d} \{ c(\theta) | h_i(\theta) = 0,\ i = 1, \ldots, N \} \tag{2}$$

Example Problem

$$\min_{\theta} \sum_{i=1}^{N} \left(\frac{1}{\theta - a_i} \right)^2 \tag{1}$$

$$\min_{\theta \in \mathbb{R}^d} \{ c(\theta) | h_i(\theta) = 0,\ i = 1, \ldots, N \} \tag{2}$$

Lift to intermediate optimization problem:

$$\min_{x \in \mathbb{R}^N} \{ f(x) | g_i(x) = 0, i \in [N_h] \}, \tag{3}$$

where f and g_i are quadratic in the lifted vector x, and where the lifted vector is given by

$$x^T = \begin{bmatrix} 1 & \theta & z_1 & \ldots & z_{N_l} \end{bmatrix}$$

where $z_i = l_i(\theta) := \frac{1}{\theta - a_i}$.

Example Problem (cont.)

$$\min_{x \in \mathbb{R}^N} \{x^T Q x | x^T A_i x = 0, i \in [N_A]\}, \tag{4}$$

where Q is the cost matrix and A_i, $i \in [N_A]$ are the constraint matrices. $Q_{i,\,i} = 1$ for $i = 3, \ldots, N+2$ and $A_{i,\,1,\,2+i} = A_{i,\,2+i,\,1} = -a_i$ and $A_{i,\,2,\,2+i} = A_{i,\,2+i,\,1} = 1$.

$$Q = \begin{bmatrix} 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ 0 & 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \end{bmatrix},\ A_i = \begin{bmatrix} 0 & \ldots & -a_i & \ldots & 0 \\ \vdots & \ddots & 1 & & \vdots \\ -a_i & 1 & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ 0 & \ldots & \ldots & \ldots & 0 \end{bmatrix}$$

Intermediate Problem to SDP

Definition

The inner product $\langle A, B \rangle$, where A and B are n by n matrices, is defined as tr(AB).

Equation 4 is still hard due to it needing a *rank 1* solution. To get an SDP, *relax* this requirement. Let X $:= xx^T$, where $X \succcurlyeq 0$. We can solve the following equation.

$$\min_{X \in S_+^N} \{\langle Q, X \rangle | \langle A_i, X \rangle = 0, i \in [N_A]\}, \tag{5}$$

Equation 5 is the SDP relaxation of the original problem.

Illustrative Code

```python
#Pedagogical problem before Autotight
n = 3
a_array = np.array([4, 15, 2])
np.random.seed(5)

#Q matrix (cost)
Q_mat = np.zeros([n+2, n+2])
for i in range(2, n+2):
  Q_mat[i, i] = 1

#A_i matrices (constraint)
A = np.zeros([n, n+2, n+2])

for i in range(n):
  A[i, 0, 2+i] = A[i, 2+i, 0] = -a_array[i]
  A[i, 1, 2+i] = A[i, 2+i, 1] = 1

#defining decision variable, objective, and constraints
X = cp.Variable((n+2, n+2), symmetric=True)
constraints = [X >> 0]
constraints += [
        cp.trace(A[i] @ X) == 0 for i in range(n)
        ]

prob = cp.Problem(cp.Minimize(cp.trace(Q_mat @ X)),
                  constraints)
prob.solve(solver=cp.MOSEK, verbose = False)

eigenvalues = sorted(np.linalg.eigvals(X.value), reverse = True)
print("The optimal value is", prob.value)
print("A solution X is\n", X.value)
print("using X, the sum is", np.sum([(1 / math.sqrt((X.value[1,1])) - a_array[i]) ** 2 for i in range(n)]))
print("Ratio between two largest eigenvalues of X*: ", eigenvalues[0] / eigenvalues[1])
```

Output

```
The optimal value is 0.0
A solution X is
 [[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
using X, the sum is 206.0
Ratio between two largest eigenvalues of X*:  1.0
```

This is NOT tight ($\text{rank}(X^\star) \neq 1$)

Improvement

Definition

We define a solution $X^\star$ to be *rank tight* if the rank of $X^\star$ is 1 and if the ratio between the two largest eigenvalues of $X^\star$ is numerically large. We can deduce strong duality (on the relaxation) and that $X^\star$ is an optimal solution of the SDP.

- Currently not rank tight
- Tighten gap by finding redundant constraints
- Tedious manual process!

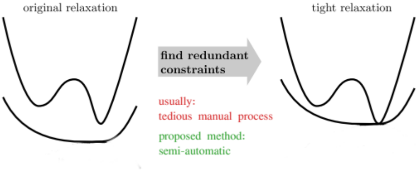


Figure 3

Autotight

Procedure

1. *Generate feasible points* $\theta^{(s)}$
2. *Create lifted vectors* $\mathcal{X} = \{x^{(1)}, \dots, x^{(N_s)}\}$
3. *Formulate data matrix* $Y = [vech((xx^\top)^{(1)}) \dots vech(xx^\top)^{N_s})] \in \mathbb{R}^{nxN_s}$
4. *Find nullspace basis of* Y
5. *Add vectors in the nullspace into our constraints*

Main idea: $\langle A_i, X \rangle = 0 \implies A_i \in \text{null}(\text{span}(\{\mathcal{X}\}))$

Code

```python
# Autotight Algorithm

# half vectorization function
def vech(matrix):
    if len(matrix) != len(matrix[0]):
        raise ValueError("not a square matrix")
    rows = int((len(matrix)*(len(matrix)+1))/2)
    newVec = np.zeros([rows, 1])
    index = 0
    for col in range(len(matrix[0])):
        for row in range(col + 1):
            if row == col:
                newVec[index][0] = matrix[row][col] / math.sqrt(2)
                index += 1
            else:
                newVec[index][0] = matrix[row][col]
                index += 1
    return newVec

# inverse half vectorization, which creates a symmetric matrix
def inv_vech(vech):
    vech_flat = vech.flatten()
    n = len(vech_flat)
    mat_size = int(math.sqrt(2 * n + (1/4)) - (1/2))

    A = populateUpperTri(mat_size, vech_flat)
    A = A + A.T
    np.fill_diagonal(A, np.diagonal(A) / 2)
    return A
```

```python
def populateUpperTri(mat_size, vec):
    A = np.zeros([mat_size, mat_size])
    index = 0
    for col in range(mat_size):
        for row in range(col + 1):
            if row == col:
                A[row][col] = vec[index] * math.sqrt(2)
                index += 1
            else:
                A[row][col] = vec[index]
                index += 1
    return A

# formulates the Y data matrix
def formulateY(N):
    vech_size = int(N)
    total_pts = int(1.2 * N)
    Y = np.empty([vech_size, 0])
    for i in range(total_pts):
        theta = np.random.rand(1)
        z_vals = 1 / (theta - a_array)
        x_feas = np.hstack(([1], theta, z_vals)) # creates
    the lifted vector
        Y = np.hstack((Y, vech(x_feas[None, :].T @ x_feas[
    None, :])))
    return Y
```

```python
big_N = (n+2)*(n+3)/2
Y = formulateY(big_N)

# QR factorization
Q, R = sp.linalg.qr(Y)
learned_constraints = Q[:, np.linalg.matrix_rank(Y) + 1:]

constraints.clear()
constraints += [X >> 0]
constraints += [
        cp.trace(inv_vech(learned_constraints[:, i]) @ X) ==
     0 for i in range(len(learned_constraints[0]))
        ]
constraints += [
        X[0][0] == 1
        ]
prob = cp.Problem(cp.Minimize(cp.trace(Q_mat @ X)),
                  constraints)
prob.solve(solver=cp.MOSEK, verbose = False)

# Print result.
print("The optimal value is", prob.value)
print("A solution X is\n", X.value)
print("using X, the sum is", np.sum([(1 / (X.value[0,1] -
    a_array[i])) ** 2 for i in range(n)]))

#checking for rank tightness by comparing greatest two
    eigenvalues
eigenvalues = sorted(np.linalg.eigvals(X.value), reverse =
    True)
print("Ratio between two largest eigenvalues of X*: ",
    eigenvalues[0] / eigenvalues[1])
```

Output

```
The optimal value is 0.0003153607991512305
A solution X is
```

$$\begin{bmatrix} 1.0\times10^{0} & -9.9\times10^{1} & -7.2\times10^{-3} & -5.5\times10^{-3} & -3.7\times10^{-3} \\ -9.9\times10^{1} & 1.5\times10^{4} & 9.7\times10^{-1} & 9.2\times10^{-1} & 9.9\times10^{-1} \\ -7.2\times10^{-3} & 9.7\times10^{-1} & 8.7\times10^{-5} & 2.1\times10^{-5} & 2.6\times10^{-5} \\ -5.5\times10^{-3} & 9.2\times10^{-1} & 2.1\times10^{-5} & 1.1\times10^{-4} & 1.1\times10^{-4} \\ -3.7\times10^{-3} & 9.9\times10^{-1} & 2.6\times10^{-5} & 1.1\times10^{-4} & 1.1\times10^{-4} \end{bmatrix}$$

```
using X, the sum is 0.0002693883667950298
Ratio between two largest eigenvalues of X*:  42194.0209114485
```

Results and Future

Conclusions:

- Eigenvalue ratio sufficiently large
- Solution is rank tight
- Lower bound equal to the global minimum of the original problem

Results and Future

Conclusions:

- Eigenvalue ratio sufficiently large
- Solution is rank tight
- Lower bound equal to the global minimum of the original problem

In the future,

- Vectorize code
- Implement Autotemplate for scalability
- Continue learning about optimization!

References

[1] Frederike Dümbgen, Connor Holmes, Ben Agro, and Timothy Barfoot. *Toward Globally Optimal State Estimation Using Automatically Tightened Semidefinite Relaxations.* Sept. 2023. URL: https://arxiv.org/pdf/2308.05783.pdf (visited on 03/27/2024).

[2] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* 2014. URL: https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf.

[3] Sheldon Axler. *Linear Algebra Done Right.* Mar. 2024. URL: https://linear.axler.net/LADR4e.pdf (visited on 03/26/2024).

[4] Heena Rijhwani. *Optimization.* Analytics Vidhya, Nov. 2020. URL: https://medium.com/analytics-vidhya/optimization-acb996a4623c.

[5] Robert A. Leffler. *Low Discrepancy Sequence Initialization for NonConvex Optimization.* Medium, Dec. 2021. URL: https://medium.com/@robert.a.leffler/low-discrepancy-sequence-initialization-for-nonconvex-optimization-dfcc35d5c0dd (visited on 04/01/2024).

[6] *What Is SLAM (Simultaneous Localization and Mapping) – MATLAB & Simulink.* www.mathworks.com. URL: https://www.mathworks.com/discovery/slam.html.