Convex Optimization, DCP, and the Smallest Enclosing Ball Problem

Derryl Sayo

```
import numpy as np
import cvxpy as cp
import matplotlib.pyplot as plt
```

```
(CVXPY) Apr 15 12:04:09 PM: Encountered unexpected exception importing solver GLOP:
RuntimeError('Unrecognized new version of ortools (9.9.3963). Expected < 9.8.0. Please open a
(CVXPY) Apr 15 12:04:09 PM: Encountered unexpected exception importing solver PDLP:
RuntimeError('Unrecognized new version of ortools (9.9.3963). Expected < 9.8.0. Please open a
```

Convex Optimization

In previous entries, we have been primarily using CVXPY (see Diamond and Boyd 2016), a *convex optimization* package for Python, to solve a variety of optimization problems.

The keyword in CVXPY is *convex optimization* and attempting to solve an unsupported problem will throw an error:

```
try:
  x = cp.Variable(1)
  objective = cp.Minimize(x**4 - 3*x**2 + 2)
  problem = cp.Problem(objective)
  problem.solve()
except Exception as err:
  print(err)
```

```
Problem does not follow DCP rules. Specifically:
The objective is not DCP. Its following subexpressions are not:
power(var1, 4.0) + -3.0 @ power(var1, 2.0) + 2.0
```

In this entry, we will take a look into the definitions of convex optimization problems, Disciplined Convex Programming (DCP) rules, and see how we can verify that a problem's components are solvable by CVXPY by hand to better understand its capabilities.

What is Convex Optimization?

A **convex optimization** problem is a problem consisting on minimizing a *convex function* over a *convex set* ("Chapter 8: Convex Optimization," n.d.). Convex problems take the standard form:

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t} \quad & g_i(x) \leq 0, \quad i = 1, 2, ..., m \\ & h_j(x) = 0, \quad j = 1, 2, ..., p \end{aligned}$$

where $f, g_i : \mathbb{R}^n \to \mathbb{R}$ are convex functions representing the objective and inequality constraints and $h_i : \mathbb{R}^n \to \mathbb{R}$ are affine functions representing any equality constraints.

Convex optimization problems are a superset of problems involving linear programming, integer programming, and quadratic programming which is why these can all be solved by CVXPY. Note that all of these problems follow the above standard form with specific objective and constraints; they all aim to min/max some objective f subject to some set of constraints which form a convex set.

One very important and handy property of convex problems is that **all local optimum are also *global* optimum** which we have experienced empirically through LP and QP problems. There is less of a worry to verify the global optimality of convex optimization solutions as any solution is the best.

We will now define some more of the intricacies of the definition above, namely what are *convex functions* and *convex sets*.

Convex Sets

A set $C \subseteq \mathbb{R}^n$ is called *convex* if

$$\forall x, y \in C, \ \lambda \in [0, 1], \ \lambda x + (1 - \lambda) y \in C$$

In other words, we can take any two points inside some set and the line drawn between the two points are also contained in the set ("Chapter 6: Convex Sets," n.d.).

Some examples of convex sets include circles, ellipses, squares and other polyhedra, and even infinite sets like $\mathbb{R}, \mathbb{R}^+$ while examples of sets that are not convex are stars and bananas since you can find a pair of points where the line between them are not inside the set.

An important feature of convex sets are that all other points are visible from any location inside the set. This follows from the line intuition detailed above since all lines of sight are contained in the set which also gives an intuitive representation that any locally optimal point is globally optimal; it's like seeing all other feasible solutions and being able to verify that there are no hidden points that could have a better value than the current optimum.

Convex Functions

A function $f : C \to \mathbb{R}$ defined on a convex set $C \subseteq \mathbb{R}^n$ is *convex* if

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

This can be seen as drawing a line between two points on the set (the RHS of the inequality) and if all function values between the two points are underneath that line ("Chapter 7: Convex Functions," n.d.).

One very special property of convex functions is that there are a variety of operations that preserve convexity including but not limited to multiplication by non-negative scalars (ie. αf), summation of convex functions (ie. $f_1 + f_2 + \cdots + f_n$), and composition with a nondecreasing convex function (ie. $g(f(x))$). This will come in handy later.

Examples of convex functions include lines and quadratics which allows the above definition for convex optimization to hold for LP and QP objective functions and constraints.

Disciplined Convex Programming (DCP)

Before CVXPY can employ methods to solve the input problem, it must first verify that it is convex. In CVXPY, the problem is constructed from an objective function and a list of constraints (ie. follows the semantic definition for a convex problem above).

The objective and constraint functions will be broken into sub-components which we can use **composition rules for convex functions as stated above** to determine the *curvature* of each component. DCP labels each component as one of constant, affine, convex, concave, or unknown.

Afterward, the entire problem can be labelled as convex if it follows the following rules:

- The problem objective must be either convex (minimize) or concave (maximize)
- The constraints are one of `affine == affine`, `convex <= concave`, or `concave >= convex`

If the problem has components that satisfy these rules, then it can be **guaranteed** solvable by CVXPY (even if it takes an exponential amount of time).

More information behind the theory of DCP in general can be found in (Grant, Boyd, and Ye 2006).

Smallest Enclosing Ball/Chebyshev Center Problem

We will illustrate some convex analysis with a simple problem:

> Given a set of m points $P = \{p_1, ..., p_m\},\ p_i \in \mathbb{R}^n$ in some n-dimensional space $\mathbb{R}^n$, how can we enclose all points in a n-dimensional ball?

We can describe this problem in the standard form

$$\begin{aligned} \min_{r} \quad & r \\ \text{s.t} \quad & ||x - p_i|| \leq r,\ i = 1, 2, ..., m \end{aligned}$$

where we try to find both a center point $x \in \mathbb{R}^n$ and the radius $r \in \mathbb{R}$.

Below we will go through the problem to determine if it follows DCP rules and then solve it using Python.

Objective Function

In order to follow DCP rules, the objective $f(x) = r$ must be convex.

Proof. We show that $f(x) = r$ is convex. Let $x, y \in \mathbb{R}^n$ and $\lambda \in [0, 1]$. Then,

$$f(\lambda x + (1 - \lambda)y) = r \leq r = \lambda r + (1 - \lambda)r = \lambda f(x) + (1 - \lambda)f(y)$$

□

In fact, a constant function is affine and affine functions are both convex and convex (ie. $r = r \iff r \geq r \wedge r \leq r$). Hence, the objective function for the smallest enclosing ball problem passes DCP rules.

Constraints

In order to follow DCP rules, we show that the constraint LHS $||x - p_i||$ is convex (the RHS is concave since it is a constant).

Proof. Like in how DCP is actually applied in CVXPY, we break down $g(x) = ||x - p_i||$ into smaller components. The interior $x - p_i$ is the summation of affine functions which is both concave and convex.

So, we show that the 2-norm $h(z) = ||z||$ is convex. Let $x, y \in \mathbb{R}^n$ and $\lambda \in [0, 1]$. Then,

$$\begin{aligned} h(\lambda x + (1-\lambda)y) = ||\lambda x + (1-\lambda)y|| &\leq ||\lambda x|| + ||(1-\lambda)y|| \\ &= \lambda||x|| + (1-\lambda)||y|| \\ &= \lambda h(x) + (1-\lambda)h(y) \end{aligned}$$

by the triangle inequality.

Hence, norms are convex and the constraint $||x - p_i|| \leq r$ is convex $\leq$ concave so this constraint follows DCP rules. □

Since now we have mathematically shown that all components of this problem follow DCP rules, we can confidently use CVXPY below to guarantee a solution is able to be obtained.

Solutions using Python

We write and solve the above problem using CVXPY:

```
def chebyshev_center(P):
  """
  Find the smallest enclosing ball for a set of points P

  Params:
    - P: numpy matrix of points

  Returns:
    A tuple (x, r) representing the center and radius of the ball
  """
  m = np.shape(P)[0]
  n = np.shape(P)[1]

  r = cp.Variable(1)
  X = cp.Variable(n)

  objective = cp.Minimize(r)
  constraints = [cp.norm(X - P[i]) <= r for i in range(m)]

  problem = cp.Problem(objective, constraints)
```

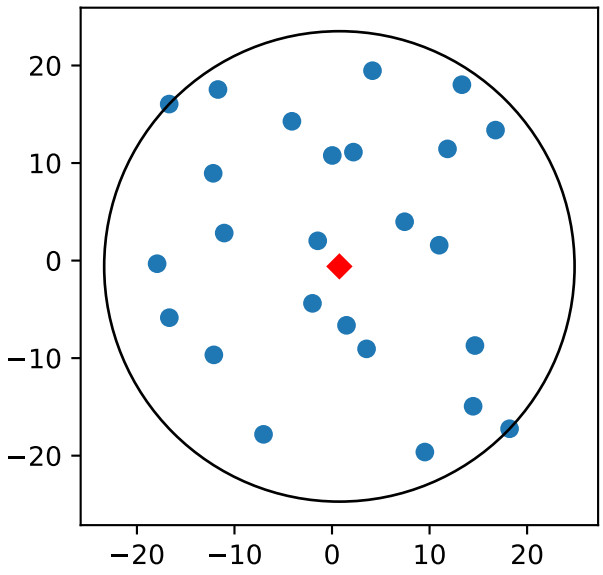
```
  problem.solve()

  return objective.value, X.value

def generate_points(num_points, coord_range):
  """
  Generate num_points in R^2

  Params:
    - num_points: number of points to generate (m)
    - coord_range: tuple (min, max) to generate points within

  Returns:
    num_points x 2 matrix of points
  """
  range_min, range_max = coord_range

  x_coords = np.random.uniform(range_min, range_max, num_points)
  y_coords = np.random.uniform(range_min, range_max, num_points)

  points = np.column_stack((x_coords, y_coords))

  return points

def display_circle(r, c, points):
  """
  Display the smallest enclosing circle solution using plt (2D case only)

  Params:
    - r: radius of circle
    - c: center of circle [x_1, x_2]
    - points: m x 2 matrix of points
  """
  fig, ax = plt.subplots()
  ax.set_aspect("equal")

  # Plot points
  ax.scatter(points[:,0], points[:,1])

  # Plot circle and center
  ax.plot(c[0], c[1], "D", color="r")
  cir = plt.Circle((c[0], c[1]), r, fill=False)
```

```
  ax.add_patch(cir)

  plt.show()
```

We generate points and solve the smallest enclosing ball problem:

```
points = generate_points(25, (-20, 20))
radius, center = chebyshev_center(points)
display_circle(radius, center, points)
```

```
/home/derry/.local/share/virtualenvs/MATH441-vlJmoJsg/lib/python3.8/site-packages/cvxpy/redu
    Your problem is being solved with the ECOS solver by default. Starting in
    CVXPY 1.5.0, Clarabel will be used as the default solver instead. To continue
    using ECOS, specify the ECOS solver explicitly using the ``solver=cp.ECOS``
    argument to the ``problem.solve`` method.

  warnings.warn(ECOS_DEPRECATION_MSG, FutureWarning)
```



Conclusion

Convex optimization is the set of problems that requires convex objective and contraint functions. In order to verify the feasibility of the problem, convex optimization packages such as

CVXPY leverage the properties of convex functions to determine whether it follows a set of rules. Although there is much more going on in the backend to verify function convexity (eg. all of the CVXPY built in functions are documented with curvature and signs), it follows the structured nature derived from the mathematical definitions of convexity.

As a result, if we can determine that we are optimizing over convex functions then we can solve a greater variety of problems including geometry using norms as constraints and all subsets of convex optimization like linear and integer programming.

References

"Chapter 6: Convex Sets." n.d. In *Introduction to Nonlinear Optimization: Theory, Algorithms, and Applications with Python and MATLAB, Second Edition*, 113–34. https://doi.org/10.1137/1.9781611977622.ch6.

"Chapter 7: Convex Functions." n.d. In *Introduction to Nonlinear Optimization: Theory, Algorithms, and Applications with Python and MATLAB, Second Edition*, 135–69. https://doi.org/10.1137/1.9781611977622.ch7.

"Chapter 8: Convex Optimization." n.d. In *Introduction to Nonlinear Optimization: Theory, Algorithms, and Applications with Python and MATLAB, Second Edition*, 171–206. https://doi.org/10.1137/1.9781611977622.ch8.

Diamond, Steven, and Stephen Boyd. 2016. "CVXPY: A Python-Embedded Modeling Language for Convex Optimization." *Journal of Machine Learning Research* 17 (83): 1–5.

Grant, Michael, Stephen Boyd, and Yinyu Ye. 2006. "Disciplined Convex Programming." In *Global Optimization: From Theory to Implementation*, edited by Leo Liberti and Nelson Maculan, 155–210. Boston, MA: Springer US. https://doi.org/10.1007/0-387-30528-9_7.