Optimization workshop

Sicheng He, Daning Huang

MTech, Umich Aero

February 21, 2017

Why optimization?

For Aerospace and Mechanical industry...

- Optimize aircraft to give good fuel burn saving ▸ MDOlab work
- Optimize the automobile to give stiff structure
- ...

Introduction

Method:

Gradient based optimization

Contents:

- Basic:
  - Unconstrained optimization
  - Constrained optimization
- Advanced:
  - KS aggregation
  - Fortran wrapper

Basic: Unconstrained optimization

Solve for the optimality condition with a quadratic approximation

Problem setup:

$$\min f(x)$$
$$s.t.\ x \in X$$

The optimality condition for the unconstrained optimization problem is:

$$\nabla f(x) = 0$$

To solve it, we construct a quadratic approximation $\bar{f}$ of the function of f and optimize the g,

$$\bar{f}(x) := f(x_k) + \nabla f(x_k)(x - x_k) + \frac{1}{2}(x - x_k)^T H(x_k)(x - x_k)$$

Then the optimality condition can be written as,

$$\begin{aligned} &\nabla f(x) = 0 \approx \nabla \bar{f}(x) = 0 \Leftrightarrow \nabla f(x_k) = H(x_k)(x - x_k) \\ &\Rightarrow \Delta x = H^{-1}(x_k)\nabla f(x_k),\ x_{k+1} = \alpha_k \Delta x + x_k \end{aligned} \tag{1}$$

What info you need to give for an optimizer

Recall (1):

$$\nabla f(x) = 0 \approx \nabla \bar{f}(x) = 0 \Leftrightarrow \nabla f(x_k) = H(x_k)(x - x_k)$$
$$\Rightarrow \Delta x = H^{-1}(x_k)\nabla f(x_k), x_{k+1} = \alpha_k \Delta x + x_k$$

To solve an optimization problem we generally need to give it:

- initial point: x_0
- original function: $f(x)$ (helps to determine the step size)
- function helps to calculate the gradient $\nabla f(x_k)$
- function helps to calculate the Hessian? (too expensive, we will use low order updates instead – Broyden–Fletcher–Goldfarb–Shanno (BFGS) method)

More thorough treatment: Prof.Epelman's note

Basic: Constrained optimization

Optimality condition for constrained optimization

For the constrained optimization, we use the SLSQP (Sequential Least Square Quadratic Programming) method. The optimization problem is,

$$\begin{aligned} \min\ & f(x) \\ \text{s.t. } & c(x) = 0 \\ & g(x) \leq 0 \end{aligned}$$

The optimal condition (necessary not sufficient) is enforced through the Karush–Kuhn–Tucker (KKT) conditions:

$$\begin{aligned} \nabla_x L(x^*, \lambda^*, \nu^*) &= 0 \\ c(x^*) &= 0 \\ g(x^*) &\leq 0 \\ {\lambda^*}^T g(x^*) &= 0 \\ \lambda^* &\geq 0 \end{aligned}$$

where $L(x, \lambda, \nu) := f(x^*) + {\nu^*}^T c(x^*) + {\lambda^*}^T g(x^*)$ is the

Solving for KKT condition with Newton method 1

Linearize KKT condition, we have,

$$
\begin{aligned}
\nabla^2 L(x_k, \lambda_k, \nu_k)\Delta x + \nabla c(x_k)\Delta \nu + \nabla g(x_k)\Delta \lambda &= -\nabla L(x_k, \lambda_k, \nu_k) \\
\nabla c(x_k)^T \Delta x &= -c(x_k) \\
\nabla g(x_k)^T \Delta x &\leq -g(x_k) \\
g(x_k)^T \Delta \lambda + \lambda_k^T \nabla g(x_k)\Delta x &= -\lambda_k g(x_k) \\
\Delta \lambda &\geq -\lambda_k
\end{aligned}
$$

Rewrite it in matrix form,

$$
\begin{pmatrix}
\nabla^2 L(x_k, \lambda_k, \nu_k) & \nabla c(x_k) & \nabla g(x_k) \\
\nabla c(x_k)^T & 0 & 0 \\
\lambda_k^T \nabla g(x_k) & g(x_k)^T & 0
\end{pmatrix}
\begin{pmatrix}
\Delta x \\ \Delta \lambda \\ \Delta \nu
\end{pmatrix}
=
\begin{pmatrix}
-\nabla L(x_k, \lambda_k, \nu_k) \\ -c(x_k) \\ -\lambda_k g(x_k)
\end{pmatrix}
$$

Solving for KKT condition with Newton method 2

Solving the following problem will give a d^* equals Δx in the previous problem. And the following problem is actually what is solved in a optimizer.

$$\begin{aligned} \min\ & f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla^2 L(x_k, \lambda_k, \nu_k) d \\ \text{s.t. } & c(x_k) + \nabla c(x_k)^T d = 0 \\ & g(x_k) + \nabla g(x_k)^T d \leq 0 \end{aligned}$$

Advanced: KS aggregation

KS aggregation

Sometimes we will have so many constraints, it will make optimization slow. There is one clever trick to deal with it: For optimization problem with the following constraints,

$$g_j \leq 0, j \in \{1, 2, ..., N\}$$

It can be approximated as:

$$\frac{1}{\rho} \log \left(\sum_{i=1}^{N} e^{\rho g_i} \right) \leq 0, \rho \to \infty$$

Why? As $\rho \to \infty$, the part in the bracket in the LHS goes to $e^{\rho g_j(x)}$ where $j = argmax_i g_i(x)$. So we have,

$$\frac{1}{\rho} \log(e^{\rho g_j}) \approx g_j \leq 0, j = argmax_i g_i(x), \rho \to \infty$$

KS aggregation: Implementation suggestion

It is better to scale the constraint in case of large number overflow,

$$\frac{1}{\rho}\log\left(\sum_{i=1}^{N} e^{\rho g_i}\right) = \frac{1}{\rho}\log\left(e^{\rho g_j}\left(\sum_{i=1}^{N} e^{\rho(g_i-g_j)}\right)\right)$$

$$= g_j + \frac{1}{\rho}\log\left(\sum_{i=1}^{N} e^{\rho(g_i-g_j)}\right), j = argmax_i g_i(x)$$

So finally we use:

$$g_j + \frac{1}{\rho}\log\left(\sum_{i=1}^{N} e^{\rho(g_i-g_j)}\right), j = argmax_i g_i(x)$$

to substitute the large number of variables.

Advanced: f2py

f2py: Calling Fortran through Python

Motivation

Python is easy to use, but slow. Fortran is fast but harder to use. So how about combining them?!

Procedures

Write a Fortran code; compile it with f2py to a module which can be imported by python; use python call it!