Lecture 6: CS395T Numerical Optimization for Graphics and AI — Line Search Applications

Qixing Huang
The University of Texas at Austin
huangqx@cs.utexas.edu

1 Disclaimer

This note is adapted from

- Section 3 of *Numerical Optimization* by Jorge Nocedal and Stephen J. Wright. Springer series in operations research and financial engineering. Springer, New York, NY, 2. ed. edition, (2006)
- Geometry and convergence analysis of algorithms for registration of 3D shapes. H Pottmann, QX Huang, YL Yang, SM Hu - International Journal of Computer Vision, 2006.
- http://www.scholarpedia.org/article/Policy_gradient_methods.

2 Rigid Alignment of Depth Scans

This section covers http://www.geometrie.tuwien.ac.at/geom/ig/papers/tr117.pdf.

Given a surface Φ and a point cloud $\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n$ (which can be considered as sampling from Φ followed by rotation and translation), our goal is to solve the following optimization problem to recover the underlying rotation $R^\star$ and translation $\boldsymbol{t}^\star$:

$$(R^\star, \boldsymbol{t}^\star) = \operatorname*{argmin}_{R,\boldsymbol{t}} \sum_{i=1}^{n} d^2(R\boldsymbol{x}_i + \boldsymbol{t}, \Phi). \tag{1}$$

2.1 The squared distance function of a surface

To solve (1) we will have to know the second-order approximation of $d^2(\boldsymbol{x}, \Phi)$, which is given by

Proposition 1. *The second order Taylor approximant of the squared distance function of a surface Φ at a point $\boldsymbol{p} \in \mathbb{R}^3$ is expressed in the principal frame at its normal foot point $\boldsymbol{s} \in \Phi$ via*

$$F_d(x_1, x_2, x_3) = \frac{d}{d - \rho_1} x_1^2 + \frac{d}{d - \rho_2} x_2^2 + x_3^2. \tag{2}$$

Let us look at two important special cases.

- For $d = 0$ we obtain
$$F_0(x_1, x_2, x_3) = x_3^2.$$
This means that the second order approximant to d^2 at a surface point $\boldsymbol{p}$ is the same for the surface Φ and for its tangent plane at $\boldsymbol{p}$. Thus, if we are close to the surface, the squared distance function to the tangent plane at the closest point to the surface is a very good approximant.

- For $d = \infty$, we obtain
$$F_\infty(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2.$$
This is the squared distance to the foot point on the surface. We see that distances to normal foot points are just good if we are in the 'far field' of the surface Φ. In the near field it is much better to use other local quadratic approximants. The simplest one is the squared distance to the tangent plane at the normal foot point.

2.2 Gradient-Based Method

Please refer to Section 4 of Pottmann et al. 06 for more details.

The directional derivative of F in the direction $C = (\boldsymbol{c}, \overline{\boldsymbol{c}})$ is given by

$$\frac{\partial F}{\partial C} = \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{y}_i)^T \boldsymbol{v}(\boldsymbol{x}_i) = \sum_{i=1}^{n} (\boldsymbol{f}_i^T \overline{\boldsymbol{c}} + (\boldsymbol{x}_i \times \boldsymbol{f}_i)^T \boldsymbol{c}).$$

So any local minimizer satisfies

$$\sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{y}_i) = 0, \quad \sum_{i=1}^{n} \boldsymbol{x}_i \times (\boldsymbol{x}_i - \boldsymbol{y}_i) = 0. \tag{3}$$

To compute the gradient we need the following normalization

$$\sum_{i=1}^{n} \|\overline{\boldsymbol{c}} + \boldsymbol{c} \times x_i\|^2 = 1,$$

which gives rise to

$$\boldsymbol{c}^T M_e \boldsymbol{c} = 1.$$

So the actual gradient is given by

$$\nabla_e F = M_e^{-1} \begin{pmatrix} \sum_i \boldsymbol{x}_i \times (\boldsymbol{x}_i - \boldsymbol{y}_i) \\ \sum_i (\boldsymbol{x}_i - \boldsymbol{y}_i) \end{pmatrix}$$

2.3 ICP Revisited

Please refer to Section 5 of Pottmann et al. 06 for more details.

2.4 Quadratically Convergent Registration Algorithms

The rule for Newton-method:

$$\boldsymbol{x}_+ = \boldsymbol{x}_c - (\nabla^2 F(\boldsymbol{x}_c))^{-1} \nabla F(\boldsymbol{x}_c).$$

Using first-order motion approximation:

$$\boldsymbol{v}(\boldsymbol{x}) = \overline{\boldsymbol{c}} + \boldsymbol{c} \times \boldsymbol{x}.$$

The approximation to the objective function is given by

$$F_2 = \sum_i \sum_{j=1}^{2} \alpha_{ij} [\boldsymbol{n}_{i,j}^T (\overline{\boldsymbol{c}} + \boldsymbol{c} \times \boldsymbol{x}_i)]^2 + \hat{F}_2,$$

where

$$\hat{F}_2 = \sum_i (\boldsymbol{n}_i^T (\overline{\boldsymbol{c}} + \boldsymbol{c} \times \boldsymbol{x}_i) + d_i)^2.$$

This leads to a linear system. Note that for Gauss-Newton method, we set $\alpha_{i,j} = 0$.

To obtain quadratically convergent algorithms, we replace $\boldsymbol{v}(\boldsymbol{x}) = \overline{\boldsymbol{c}} + \boldsymbol{c} \times \boldsymbol{x}$ by

$$\boldsymbol{v}(\boldsymbol{x}) = \overline{\boldsymbol{c}} + \boldsymbol{c} \times \boldsymbol{x} + \frac{1}{2} (\boldsymbol{c} \times \overline{\boldsymbol{c}} + (\boldsymbol{c}\boldsymbol{c}^T - \|\boldsymbol{c}\|^2 I_3) \boldsymbol{x}_i).$$

3 Policy Gradient Methods

3.1 Assumptions and Notations

We assume that we can model the control system in a discrete-time manner and we will denote the current time step k. To model the system, we use $\boldsymbol{x}_k$ to denote the state vector at iteration k and use $\boldsymbol{u}_k$ to denote the action at iteration k. The state vector at the current iteration k and the next iteration $k+1$ follow from a distribution

$$\boldsymbol{x}_{k+1} \sim p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k).$$

We further assume that actions are generated by a policy $\boldsymbol{u}_k \sim \pi_\theta(\boldsymbol{u}_k|\boldsymbol{x}_k)$ which is modeled as a probability distribution in order to incorporate exploratory actions; for some special problems, the optimal solution to a control system is actually a stochastic controller. The policy is assumed to be parameterized by K policy parameters $\theta \in \mathbb{R}^K$.

At each instant of time, the system receives a reward denoted by $r_k = r(\boldsymbol{x}_k, \boldsymbol{u}_k) \in \mathbb{R}$. The general goal of policy optimization in reinforcement learning is to optimize the policy parameters $\theta \in \mathbb{R}^K$ so that the expected return

$$J(\theta) = E\{\sum_{k=0}^{H} a_k r_k\}$$

is optimized where a_k denote time-step dependent weighting factors, often set to $a_k = \gamma^k$ for some constant $\gamma \in (0, 1)$ or $a_k = \frac{1}{H}$, if we are interested in the average reward.

When optimizing $J(\theta)$, policy gradient methods which follow the steepest descent on the expected return are the method of choice. These methods update the policy parameterization according to the gradient update rule

$$\theta_{h+1} = \theta_h + \alpha_h \nabla_\theta J|_{\theta=\theta_h},$$

here $\nabla_\theta J|_{\theta=\theta_h}$ denotes the gradient information, and α_h denotes the step-size. Note that the update is given by $+\alpha_h \nabla_\theta J|_{\theta=\theta_h}$ since we want to maximize the reward. If the gradient estimate is unbiased and learning rates fulfill $\sum_{h=0}^{\infty} \alpha_h > 0$ and $\sum_{h=0}^{\infty} \alpha_h^2 = const$, the learning progress is guaranteed to converge to a local minimum (something we covered last lecture).

Note that the main problem in policy gradient methods is to estimate $\nabla_\theta J|_{\theta=\theta_h}$ accurately. In the following, we will talk about three main approaches. Optimization is fun because even though the number of optimization strategies is relatively fixed, the implementation of each optimization strategy varies significantly across different applications. Policy gradient methods provide one such instance. The rigid alignment problem we just covered is another problem.

3.2 Finite-Difference Methods

Finite-difference methods are one of the most widely used methods for calculating numerical gradients. The nice feature of finite-difference methods is that they only require computing the values of objective functions. The application in policy gradient methods is straight-forward: the policy parameterization is varied I times by small increments $\delta\theta_i, i = 1, 2, \cdots, I$ and for each policy parameter variation $\theta_h + \delta\theta_i$ roll-outs (or trajectory) are performed which generate estimates $\delta\hat{J}_i \approx J(\theta_h + \delta\theta_i) - J_{ref}$ of the expected return. There are different ways of choosing the reference value J_{ref}, e.g., forward-difference estimators with $J_{ref} = J(\theta_h)$ and central-difference estimators with $J_{ref} = J(\theta_h - \delta\theta_i)$. The policy gradient estimate can be obtained by regression:

$$\min_{\boldsymbol{x}} \sum_{i=1}^{I} \|\delta\hat{J}_i - \boldsymbol{x}^T \delta\theta_i\|^2$$

The choice of number of roll-outs can be essentially, empirically setting I to be twice as the number of parameters gives accurate gradient estimation.

Advantages.

- Easy to implement

Disadvantages.

- Perturbation of the parameters is tricky.
- The error decreases slowly in the presence of noise.
- Performance depends highly on the chosen policy parameterization.

3.2.1 A Few Words about Re-parameterization

Consider a function given by $f(\boldsymbol{g}(\boldsymbol{x}))$, where $\boldsymbol{g}(\boldsymbol{x}) = (g_1(\boldsymbol{x}), \cdots, g_m(\boldsymbol{x}))$. The first order derivatives give rise to

$$\frac{\partial f(\boldsymbol{g}(\boldsymbol{x}))}{\partial \boldsymbol{x}} = [\nabla g(\boldsymbol{x})]^T \nabla f(g(\boldsymbol{x})).$$

The second-order derivative gives

$$\frac{\partial^2 f(g(\boldsymbol{x}))}{\partial^2 \boldsymbol{x}} = [\nabla \boldsymbol{g}(\boldsymbol{x})]^T \nabla^2 f(\boldsymbol{g}(\boldsymbol{x}))[\nabla \boldsymbol{g}(\boldsymbol{x})] + \sum_{i=1}^{m} \frac{\partial^2 g_i(\boldsymbol{x})}{\partial^2 \boldsymbol{x}} \cdot \frac{\partial f(\boldsymbol{g}(\boldsymbol{x}))}{\partial x_i}$$

It is easy to see that, at any critical point $g(\overline{\boldsymbol{x}}^\star)$, the Hessian matrix is given by

$$\frac{\partial^2 f(g(\boldsymbol{x}^\star))}{\partial^2 \boldsymbol{x}} = [\nabla \boldsymbol{g}(\boldsymbol{x}^\star)]^T \nabla^2 f(\boldsymbol{g}(\boldsymbol{x}^\star))[\nabla \boldsymbol{g}(\boldsymbol{x}^\star)].$$

In other words, re-parameterization can effectively change the condition number of the Hessian-matrix, which could lead to a huge impact on convergence behavior. The question is can we do this for neural networks, e.g., via neural network re-parametrization.

3.3 Likelihood Ratio Methods and REINFORCE

Likelihood ratio methods are driven by a different important insight. Assume that trajectories τ are generated from a system by roll-outs, i.e., $\tau \sim p_\theta(\tau) = p(\tau|\theta)$ with return $r(\tau) = \sum_{k=0}^{H} a_k r_k$ which leads to $J(\theta) = E[r(\tau)] = \int_T p_\theta(\tau) r(\tau) d\tau$ by using

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \log p_\theta(\tau).$$

This means

$$\nabla_\theta J(\theta) = \int_{\mathcal{T}} \nabla_\theta p_\theta(\tau) r(\tau) d(\tau) = E[\nabla_\theta \log p_\theta(\tau) r(\tau)].$$

This expectation can be computed by sampling, and the remaining task is to compute $\nabla_\theta \log p_\theta(\tau)$. Note that,

$$p_\theta(\tau) = p(\boldsymbol{x}_0) \prod_{k=0}^{H} p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k) \pi_\theta(\boldsymbol{u}_k|\boldsymbol{x}_k)$$

We will spend one lecture on stochastic methods.

Advantages.

- Theoretically faster convergence rate.
- No-need to generate policy parameter variations.
- This approach has yielded the most real-world robotics results.

Disadvantages.

- One has to maintain a system model for deterministic policy.

3.4 Natural Policy Gradients

The KL-divergence between two probability distributions $p_\theta(\tau)$ and $p_{\theta+\delta\theta}(\tau)$ is given by

$$d_{KL}(p_\theta, p_{\theta+\delta\theta}) \approx (\delta\theta)^T F_\theta \delta\theta,$$

where

$$F_\theta = \int_{\mathcal{T}} p_\theta(\tau) \nabla \log p_\theta(\tau) [\log p_\theta(\tau)]^T d\tau$$

is known as the Fisher-information matrix.

So the natural policy gradient is given by

$$\max_{\delta\theta} \quad (\delta\theta)^T \nabla_\theta J \quad s.t. \quad [\delta\theta]^T F_\theta \delta\theta = \epsilon.$$

The solution to this program is given by

$$\delta \propto F_\theta^{-1} \nabla_\theta J.$$

Advantages.

- Natural policy gradients can be an order of magnitude faster than the regular gradient. They also profit from most other advantages of the regular policy gradients.

Disadvantages.

- Matrix inverse is numerically brittle.
- Natural policy gradient estimators are often much harder to implement.