GAMES-105 Lecture 3 **Character Kinematics**

Q：What is **Kinematics**?

A：The study of the motion of bodies **without reference to mass or force**

1. Forward Kinematics

1.1. Kinematics of a chain

1.1.1. Orientation

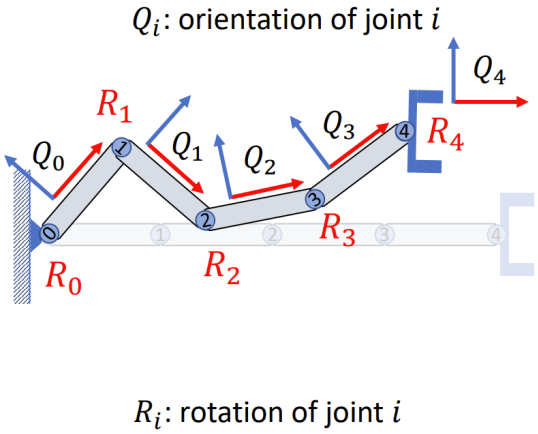


Figure 1: The rotation of a robot arm

Q: How to calculate the orientation of these jointss ? (The global coordinate is $\boldsymbol{I}$)

A:

$$
\begin{aligned}
Q_0 &= R_0 \\
Q_1 &= R_0 R = Q_0 R_1 \\
Q_2 &= R_0 R_1 R = Q_1 R_2 \\
Q_3 &= R_0 R_1 R_2 R = Q_2 R_3 \\
Q_4 &= R_0 R_1 R_2 R_3 R_4 = Q_3 R_4
\end{aligned}
$$

Summary:

- From rotation to orientation: $Q_i = Q_{i-1} R_i$
- From orientation to rotation: $R_i = Q_{i-1}^T Q_i$, extension: $R_i^j = Q_j^T Q_i$
- Relative rotation: $R_4^1 = Q_1^T Q_4$

1.1.2. Coordinate

Global coordinate of local origin:

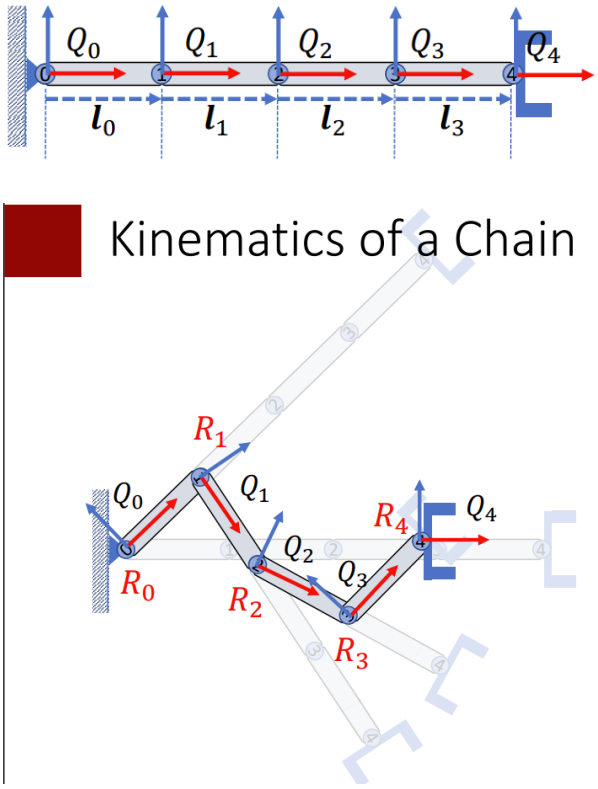


Figure 2: Calculate the **global** coordinate p of each local frame origin

$$
\begin{aligned}
p_0 &= \text{origin} \\
p_1 &= p_0 + Q_0 l_0 \\
p_2 &= p_1 + Q_1 l_1 \\
p_3 &= p_2 + Q_2 l_2 \\
p_4 &= p_3 + Q_3 l_3
\end{aligned}
$$

Global coordinate of relative point in local frame:

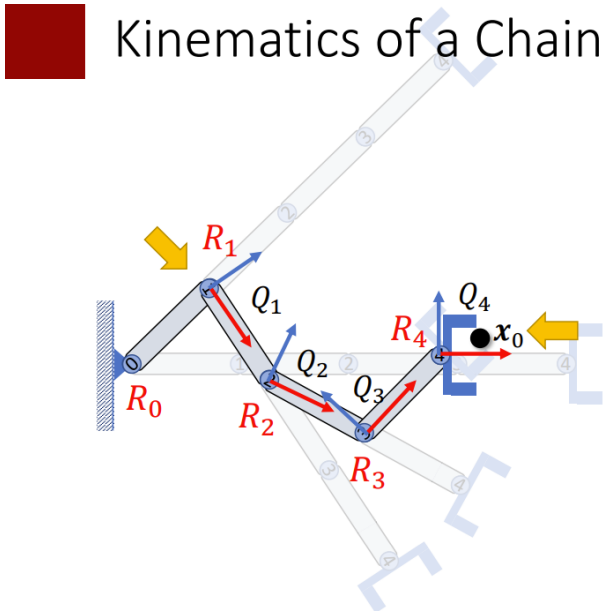


Figure 3: x_0 is the local coordinate in Q_4

$$
\begin{aligned}
x &= p_4 + R_0R_1R_2R_3R_4x_0 \\
&= p_4 + Q_4x_0 \\
&= p_3 + Q_3l_3 + Q_3R_4x_0 \\
&= p_3 + Q_3\underbrace{(l_3 + R_4x_0)}_{x} \\
&= p_2 + Q_2(l_2 + R_3(l_3 + R_4x_0))
\end{aligned}
$$

Summary:

Given the rotations of all joints R, find the coordinate of x_0 in global frame x:

❖ From root to leaf:
- Iterate 0→k: $Q_i = Q_{i-1}R_i;\ p_{i+1} = p_i + Q_il_i$
- $x = p_i + Q_ix_0$
- Explanation:
  * First, derive the orientation Q and origin global coordinate p of each local frame from 0 to k iteratively.
  * Then, Q_ix_0 calculates the global coordinate of x_0 corresponding to origin global coordinate p_i

❖ From leaf to root:
- set $x = x_0$
- Iterate k→0: $x = l_{i-1}$(relative coordinate of origin) $+ R_ix$
- Explanation:
  * First, from last local frame Q_k, revert local coordinate x_0 to the previous local frame, $x = l_{k-1} + R_kx_0$

* Then, repeat first step and set $x = l_{i-1} + R_i x$ until global frame.

1.2. Kinematics of a Character

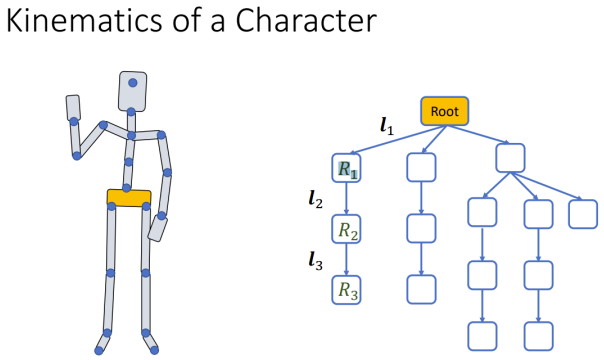


Figure 4: Kinematics of a character is a Tree.

1.3. Types of Joints and Degrees of Freedom (DOF)

DOF: Number of independent parameters that define the configuration or state of a mechanical system

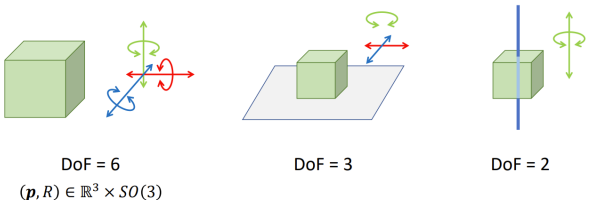


Figure 5: DoF of different cases, for DoF=6 is inclusive of translation and rotation of three axes.

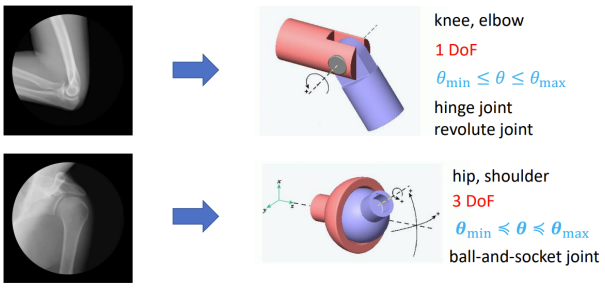


Figure 6: two prevalent joint type and their DoF with joint limits $[\theta_{min}, \theta_{max}]$.

1.4. Pose Parameters

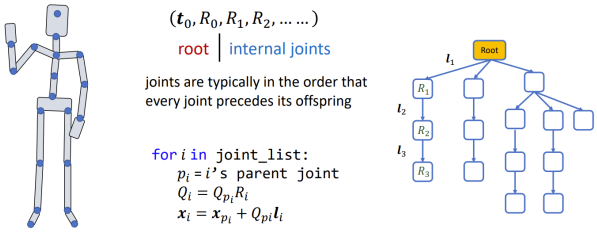


Figure 7: DoF of root t_0(6 dim) + internal joints' relative rotation.

Q1: If we know the orientations of all the joints Q_i, how to compute joint rotations?

A1: $R_{i\to j} = Q_i^T Q_j = (R_0 R_1 \cdots R_i)^T (R_0 R_1 \cdots R_i)(R_{i+1} R_{i+2} \cdots R_j)$

Q2: How should we allow stretchable bones?

A2: Add an additional term to represent translation of stretchable bones.

Motion data in a file

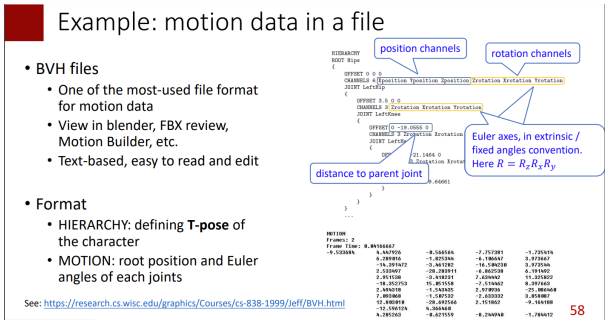


Figure 8: motion data in a file.

2. Inverse Dynamics

Inverse dynamics (IK) is an optimization process. Given the target pose x, search one or more solutions θ (a set of rotations) that can drive the joint to post $x = f(\theta)$

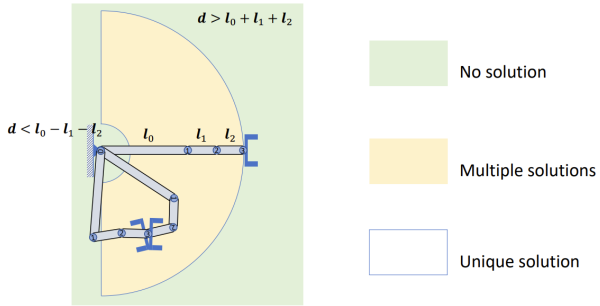


Figure 9: IK examle

The distance bewteen target point and total length of the robot arm is d.

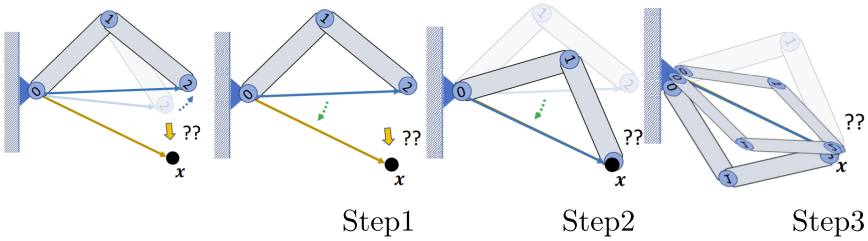
For green area, $d > l_0 + l_1 + l_2$ & $d < l_0 - l_1 - l_2$, no solution can reach that point.

For yellow area, $l_0 - l_1 - l_2 \leqslant d \leqslant l_0 + l_1 + l_2$, there are at least two solutions can reach target point.

For white area, only solution is the straight line pose of arm.

2.1. Example: Two-Joint IK

A simple solution to two-joint IK problem:



Step1: Rotate joint 1 and satisfy $\|l_{0\to 2}\| = \|l_{0\to x}\|$

How? Δ_{012}, we know l_{0-2}, l_{0-1} and l_{1-2}, can derive θ_{012} via the law of cosines.

Step2: Rotate joint 0 and let $l_{0\to 2}$ aligns $l_{0\to x}$: $l_{0\to 2} = l_{0\to x}$

How? rotate axis: $l_{0\to 2} \times l_{0\to x}$, rotate angle: $l_{0\to 2} \cdot l_{0\to x}$

Step3: Rotate joint 0 around $l_{0\to x}$ if necessary

2.2. General case: IK as an Optimization Problem

2.2.1. Problem statement

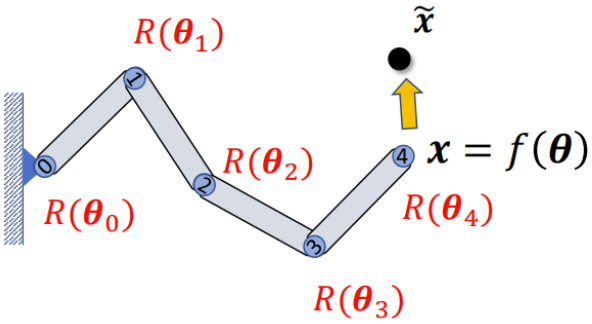


Figure 10: A general case of IK

Target: Find θ such that: $f(\theta) - x = 0$, this can be formulated as an optimization problem:

Find θ to optimize: $\min_\theta \frac{1}{2}\|f(\theta) - x\|_2^2$

2.2.2. Iterative Optimization Approach —— Coordinate Descent

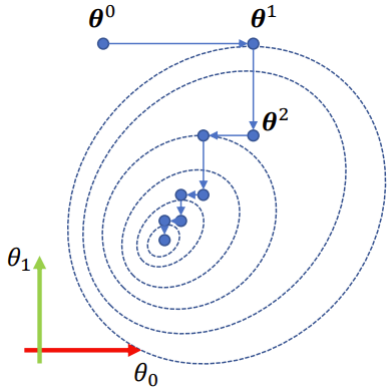
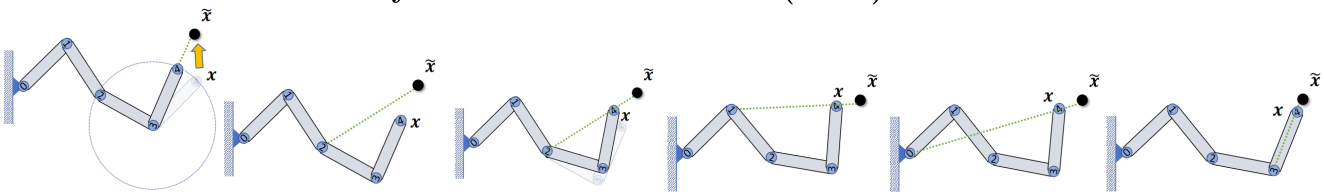


Figure 11: Coordinate Descent.(A little bit similar to conjugate gradient in TRPO)

Step1: Update parameters along each axis of the coordinate system.

Step2: Iterate cyclically through all axes.

The illustration of Cyclic Coordinate Descent (CCD) IK



(1) (2) (3) (4) (5)

>

(1) Rotate joint 3 such that $l_{3\to 4}$ points towards $\boldsymbol{x}$

$$\min_{\theta_3} f(\theta) = \min_{\theta_3} \|f(\theta_0, \theta_1, \theta_2, \theta_3) - x\|_2^2 \tag{1}$$

(2) Rotate joint 2 such that $l_{2\to 4}$ points towards $\boldsymbol{x}$

(3) Rotate joint 1 such that $l_{1\to 4}$ points towards $\boldsymbol{x}$

(4) Rotate joint 0 such that $l_{0\to 4}$ points towards $\boldsymbol{x}$

(5) Rotate joint 3 such that $l'_{3\to 4}$ points towards $\boldsymbol{x}$

.....

Iteratively rotation each joint to make the end-effector align with vector between the joint and the target. Easy to implement, very fast. The "first" joint moves more than the others May take many iterations to converge. Result can **be sensitive to** the initial solution.

Note: the reverse order of is also fine.

(1) Rotate joint 0 such that $l_{0\to 4}$ points towards $\boldsymbol{x}$

.....

The iteration order depends on the task itself.

2.3. Gradient Descent

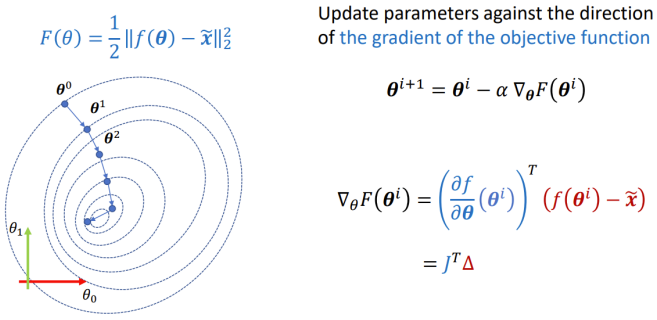


Figure 12: Leverage gradient descent to resolve IK.

$$\begin{aligned} \theta &: \text{n dim} \\ x &: \text{3 dim} \\ f(\theta) - x &: \text{3 dim} \\ \frac{\partial f}{\partial \theta}(\theta) &: \text{3-n dim} \\ \nabla_\theta f(\theta) &: \text{n dim} \end{aligned}$$

Q: How to compute the Jacobian matrix?

A: just enjory modern *autograd* framework like: jax, pytorch ...

Q: any other method?

A: Yeah, see the next section.

2.4. Finite Differencing

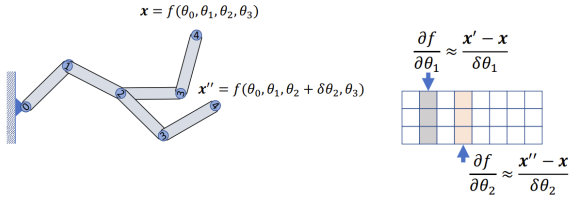


Figure 13: Finite Differencing

2.5. Geometric Approach

If dismiss modern *autograd* framework, there is an easy way to calculate Jacobian Matrix.

Case1: Assuming all joints are hinge joint

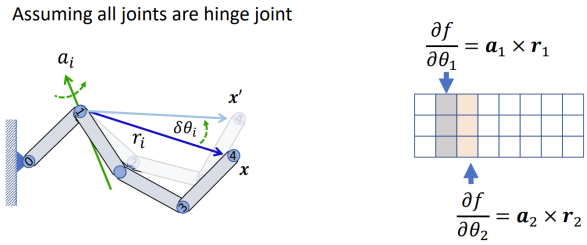


Figure 14: Geometric approach for hinge joint.

We conduct a rotation $\delta\theta_i$to joint ①， according to Rodrigues' rotation formula, there exists:

$$\begin{aligned} x' &= x + \sin\delta\theta_i(a_i \times x) + (1 - \cos\delta\theta_i)(a_i \times (a_i \times x)) \\ x' - x &= \sin\delta\theta_i(a_i \times x) + (1 - \cos\delta\theta_i)(a_i \times (a_i \times x)) \\ \lim_{\delta\theta_i \to 0} \frac{(x' - x)}{(\delta\theta_i)} &= \frac{\sin\delta\theta_i}{\delta\theta_i}(a_i \times x) + \frac{(1 - \cos\delta\theta_i)}{\delta\theta_i}(a_i \times (a_i \times x)) = a_i \times x \\ \frac{\partial f}{\partial \theta_i} &= a_i \times x \end{aligned} \tag{2}$$

Hence, we can derive the Jacobian term by cross product of rotation axis and vector between rotated joint and the end joint point.

Case2: How about ball joints ?

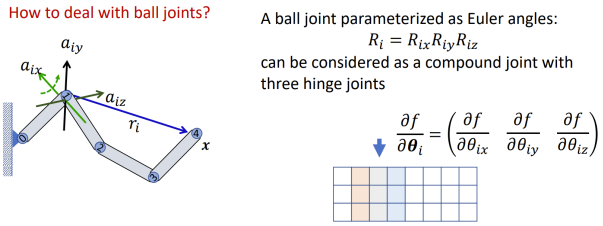


Figure 15: Geometric approach for ball joint.

The rotation of a ball joint i can be parameterized as Euler angles:
Intrinsic form: $x \rightarrow y \rightarrow z$ in local frame
Explicit form: $z \rightarrow y \rightarrow x$ in globale frame

$$R_i = R_{ix}R_{iy}R_{iz}$$

So for a ball joint which can be considerd as three hinge joints, there are three columns in Jacobian Matrix (corresponding to orange, grey, and blue column in Figure 15).

$$\frac{\partial f}{\partial \theta_{i*}} = a_{i*} \times r_i; \ * \in [x, y, z] \tag{3}$$

Caution: For intrinsic Euler angles, each axis rotation is conducted in the previous local frame. But cross product requires the coordinate in global frame, so we need to **convert each rotation axes from local frame e_x, e_y, e_z to global frame a_x, a_y, a_z**:

$$\begin{aligned} a_x &= Q_{i-1}e_x (e_x = [1, 0, 0]) \\ a_y &= Q_{i-1}e_x e_y \\ a_z &= Q_{i-1}e_x e_y e_z \end{aligned}$$

2.6. Gaussian-Newton Method

Apart from first-order optimization approach, newton method, a second-order optimization method, is an alternative for IK. Specifically, consider the following optimization problem:

$$\min_\theta F(\theta) = \frac{1}{2}\|f(\theta) - \widetilde{x}\|_2^2; \tag{4}$$

where $f(\theta)$: FK function, $\widetilde{x}$: IK target

Newton Method: consider the second-order approximation of original function $F(\theta)$, we can expand $F(\theta)$ at point θ_k using Talyor Formula:

$$\begin{aligned} F(\theta) &= F(\theta_k) + J(\theta_k)\Delta\theta + \frac{1}{2}\Delta\theta^T H(\theta_k)\Delta\theta \ \left(\text{Jacobian Matrix version}\right) \\ &= F(\theta_k) + g(\theta_k)^T\Delta\theta + \frac{1}{2}\Delta\theta^T H(\theta_k)\Delta\theta \ \left(\text{Gradient Vector version}\right) \\ \Delta\theta &= \theta - \theta_k \end{aligned} \tag{5}$$

We desire that, at the next point of $\theta = \theta_{k+1}$, the first-order derivative of $F(\theta_{k+1})$ is zero:

$$
\begin{aligned}
\nabla_\theta F(\theta_{k+1}) &= 0 \\
\nabla_\theta F(\theta_{k+1}) &= J(\theta_k) + \frac{1}{2}\Delta\theta^T 2H(\theta_k) \\
&\qquad \text{Hessian Matrix is symmetric} \\
\Delta\theta^T H(\theta_k) &= J(\theta_k) \\
H(\theta_k)\Delta\theta &= J^T(\theta_k) \\
\Delta\theta &= H(\theta_k)^{-1} J^T(\theta_k) \\
\theta_{k+1} &= \theta_k + H(\theta_k)^{-1} J^T(\theta_k)
\end{aligned}
$$

However, Newton method requires the inverse of Hessian Matrix which is extremely expensive to calculate. By the way, it also has superconvergence property and can be very efficient.

Gaussian-Newton Method is a variant of Newton method and is designed for least squares problem. IK is a typical least squares question, so we can leverage GN to solve IK. Specifically, GN approximates the regret function $f(\theta) - \widetilde{x}$ with first-order talyor formula. For simplicity, we intergate $\widetilde{x}$ in $f(\theta)$ and IK loss can be denoted as:

$$
\min_\theta F(\theta) = \frac{1}{2}\|f(\theta)\|_2^2
$$

We expand $f(\theta)$ with firsrt-order talyor formula at point x_k:

$$
f(\theta) = f(\theta_k) + J(\theta_k)\Delta\theta
$$

So, our new optimization problem is converted into the following formula:

$$
\begin{aligned}
\min_\theta F(\theta) &= \frac{1}{2}\|f(\theta)\|_2^2 \\
\min_{\Delta\theta} F(\theta) &= \frac{1}{2}\|f(\theta_k) + J(\theta_k)\Delta\theta\|_2^2
\end{aligned}
$$

The new object function is a linear function, so we can calculate the derivative and set zero to get the optimal solution.

$$
\begin{aligned}
F(\theta) &= \frac{1}{2}\|f(\theta_k) + J(\theta_k)\Delta\theta\|_2^2 \\
&= \frac{1}{2}(f(\theta_k) + J(\theta_k)\Delta\theta)^T(f(\theta_k) + J(\theta_k)\Delta\theta) \\
&= \frac{1}{2}\Big[\|f(\theta_k)\|^2 + f^T(\theta_k)J(\theta_k)\Delta\theta + \Delta\theta^T J^T(\theta_k)f(\theta_k) + \Delta\theta^T J^T(\theta_k)J(\theta_k)\Delta\theta\Big] \\
\nabla_{\Delta\theta}F(\theta) &= \frac{1}{2}\Big[f^T(\theta_k)J(\theta_k) + f^T(\theta_k)J(\theta_k) + \Delta\theta^T\Big(\big(J^T(\theta_k)J(\theta_k)\big)^T + \big(J^T(\theta_k)J(\theta_k)\big)\Big)\Big] \\
&= \frac{1}{2}\Big[2f^T(\theta_k)J(\theta_k) + \Delta\theta^T\Big(\big(J^T(\theta_k)J(\theta_k)\big) + \big(J^T(\theta_k)J(\theta_k)\big)\Big)\Big] \\
&= \frac{1}{2}\Big[2f^T(\theta_k)J(\theta_k) + 2\Delta\theta^T J^T(\theta_k)J(\theta_k)\Big] \\
&= f^T(\theta_k)J(\theta_k) + \Delta\theta^T J^T(\theta_k)J(\theta_k)
\end{aligned}
\tag{6}
$$

Tip: $\dfrac{d(x^TAx)}{dx} = x^T(A^T + A)$

Tip: $J^T(\theta_k)J(\theta_k)$ is a n x n symmetric matrix

Tip: $\dfrac{d(x^Ta)}{dx} = \dfrac{d(a^Tx)}{dx} = a^T$

(derivation ref: 1, 2, 3, 4)

We set the gradient vector to zero:

$$
\begin{aligned}
f^T(\theta_k)J(\theta_k) + \Delta\theta^T J^T(\theta_k)J(\theta_k) &= 0 \\
\text{Inverse the above equation} \\
J^T(\theta_k)f(\theta_k) + \big(J^T(\theta_k)J(\theta_k)\big)^T\Delta\theta &= 0 \\
J^T(\theta_k)f(\theta_k) + J^T(\theta_k)J(\theta_k)\Delta\theta &= 0 \\
\underbrace{J^T(\theta_k)J(\theta_k)}_{\text{Approx of Hessian}}\Delta\theta &= -J^T(\theta_k)f(\theta_k)
\end{aligned}
$$

If $J^T(\theta_k)J(\theta_k)$ is invertible, $J_{m\times n}(\theta_k),\ m \geqslant n$ may be column full rank:

$$
\begin{aligned}
\Delta\theta &= -\big(J^T(\theta_k)J(\theta_k)\big)^{-1}J^T(\theta_k)f(\theta_k) \\
\theta &= \theta_k - \underbrace{\big(J^T(\theta_k)J(\theta_k)\big)^{-1}J^T(\theta_k)}_{\text{Left Pseudoinverse of } J(\theta_k)}f(\theta_k)
\end{aligned}
\tag{7}
$$

Else if $J(\theta_k)J^T(\theta_k)$ is invertible, $J_{m\times n}(\theta_k),\ m \leqslant n$ may be row full rank:

$$
\begin{aligned}
J(\theta_k)J^T(\theta_k)J(\theta_k)\Delta\theta &= -J(\theta_k)J^T(\theta_k)f(\theta_k) \\
J(\theta_k)\Delta\theta &= -f(\theta_k) \\
\underbrace{J(\theta_k)J(\theta_k)_L^{-1}}_{I}\Delta\theta &= -J(\theta_k)_L^{-1}f(\theta_k) \\
\Delta\theta &= -J(\theta_k)_L^{-1}f(\theta_k) \\
\theta &= \theta_k - \underbrace{J^T(\theta_k)\Big(J(\theta_k)J^T(\theta_k)\Big)^{-1}f(\theta_k)}_{\text{Right Pseudoinverse of } J(\theta_k)}
\end{aligned}
\tag{8}
$$

2.7. Levenberg-Marquardt Method (LM)

The problem of GN is the assumption that $J^T(\theta_k)J(\theta_k)$ shoule be invertible (singular). However, this product is only semipositive definite. That is to say, when GN is used, $J^T(\theta_k)J(\theta_k)$ may be a singular matrix. At this time, the incremental stability is poor, causing the algorithm to not converge. More seriously, if the step size $\Delta\theta$ is too large, the local approximation will not be accurate enough.

Although GN has these shortcomings, it is still worth learning, because in nonlinear optimization, a lot of algorithms can be attributed to variations of GN. These algorithms all use the ideas of GN and correct the shortcomings of GN. For example, some line search methods , such improvements add scalars α， After determining the step size $\Delta\theta$, further find out α. Minimize $\|f(\theta+\alpha\Delta\theta)\|^2$, not as simple as GN $\alpha=1$.

LM method corrects these problems to some extent and is generally considered more robust than GN. Although its convergence rate may be slower than GN, it is called the damped Newton method Since the approximate second-order Taylor expansion adopted in GN can only have a good approximation effect near the expansion point, we naturally thought that we should give Δx. Add a Trust Region, which cannot be too approximate and inaccurate. Nonlinear optimization has a series of such methods, which are also called Trust Region Method.

In the trust region, we believe that the approximation is effective. If the region is out, the approximation may go wrong. How to confirm the scope of this trust region? A better method is to determine according to the difference between our approximate model and the actual function. If the difference is small enough, let's make the scope as large as possible. If the difference is large, we will narrow the approximate range.

Therefore, consider using $\rho=\dfrac{f(x+\Delta x)-f(x)}{J(x)\Delta x}$ to determine whether Taylor approximation is good enough. The denominator is the value that approximates the decline of the model.

If ρ Close to 1, the approximation is good.

If ρ If it is too small, it means that the actual reduction is far less than the approximate reduction, which means that the approximation is poor.

If ρ is relatively large, it means that the actual decline is greater than expected. We can enlarge the approximate range.

Therefore, we can build a constrained version of GN:

$$\begin{aligned} \min_{\Delta\theta} &= \frac{1}{2}\|f(\theta_k) + J(\theta_k)\Delta\theta\|_2^2 \\ \min_{\Delta x_k} \frac{1}{2}\|f(\theta_k) + J(\theta_k)\Delta\theta\|_2^2, &\text{ s.t. } \|D\Delta\theta\|^2 \leqslant \mu \end{aligned} \tag{9}$$

The above equation of the object of LM method.

Here, the threshold μ of the expanded approximate range are empirical values, which can be replaced by other values. In Eq.(9), we limit the increment to a radius of μ. Only in this ball can it be effective. With D, the sphere can be regarded as an ellipsoid. In the optimization method proposed by Levenberg, taking D as the unit matrix I is equivalent to directly taking $\Delta\theta$ is constrained within a sphere. Then Margaudt proposed to take D as a non negative diagonal matrix. In practice, the square root of the diagonal element of $J^T J$ is commonly used to make the constraint range on the dimension with small gradient larger.

To solve Eq.(9), we use the Lagrange multiplier to transform it into an unconstrained optimization problem:

$$\min_{\Delta x_k} \frac{1}{2}\|f(\theta_k) + J(\theta_k)\Delta\theta\|_2^2 + \lambda\|D\Delta\theta\|^2$$

Similar to GN, after expanding the above equation, we find that the core of the problem is still the linear equation for calculating increments:

$$\big(H + \lambda D^T D\big)\Delta x = g \tag{10}$$

If $D = I$, then

$$(H + \lambda I)\Delta x = g \tag{11}$$

We can see that when the parameter λ is small, H is dominant, which shows that the quadratic approximation model is better in this range, and the LM method is closer to the GN method.

When large λ, LM is more close to the first step descent method, which indicates that the nearby quadratic approximation is not good enough.

The solution method of LM can avoid the nonsingular and ill conditioned problem of coefficient matrix of linear equations to a certain extent, and provide more stable and accurate increment Δ x.