# Practical Course: Vision-based Navigation
# SS 2018

# Lecture 3. State Estimation

Dr. Jörg Stückler, Dr. Xiang Gao
Vladyslav Usenko, Prof. Dr. Daniel Cremers

# Contents

- From state estimation to least square
- Batch least square
- Application: estimate camera pose by iterative method

# Contents

- From state estimation to least square
- Batch least square
- Application: estimate camera pose by iterative method

# 1. From state estimation to least square

- Recall the motion model and observation model

$$\begin{cases} \boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{w}_k\right) \\ \boldsymbol{z}_{k,j} = h\left(\boldsymbol{y}_j, \boldsymbol{x}_k, \boldsymbol{v}_{k,j}\right) \end{cases} .$$

- How to estimate the unknown variables given the observation data?

# 1. Batch state estimation

- Batch approach
  - Give all the measurements
  - To estimate all the state variables
- State variables:

$$x = \{x_1, \ldots, x_N, y_1, \ldots, y_M\}.$$

Observation and input:

$$u = \{u_1, u_2, \cdots\}, z = \{z_{k,j}\}$$

- Our purpose:

$$P(x|z, u).$$

- Bayes' Rule:

Likehood Priori

$$p(x|u, z) = \frac{P(z|x, u)p(x|u)}{P(z|u)}$$

Posteriori

# 1. From state estimation to least square

- It is usually hard to write out the full distribution of Bayes' formula, but we can:
- MAP: Maximum A Posteriori

$$x_{MAP} = \underset{x}{\operatorname{argmax}} P(x|u,z) = \operatorname{argmax} \frac{P(z|x,u)P(x|u)}{P(z|u)}$$
$$= \operatorname{argmax} P(z|x)P(x|u)$$

Drop denominator because it is not relevant with x

Drop u because z is not relevant with u

- "In which state it is most like to produce such measurements"

# 1. From state estimation to least square

- From MAP to batch least square
- We assume the noise variables are independent, so that the joint pdf can be factorized:

$$\boxed{P(z|x) = \prod_{k=0}^{K} P(z_k|x_k)}$$

- Let's consider a single observation: $\boldsymbol{z}_{k,j} = h\left(\boldsymbol{y}_j, \boldsymbol{x}_k\right) + \boldsymbol{v}_{k,j},$
  - Affected by white Gaussian noise: $v_{k,j} \sim N\left(0, Q_{k,j}\right)$

- The observation model gives us a conditional pdf:

$$P(\boldsymbol{z}_{j,k}|\boldsymbol{x}_k, \boldsymbol{y}_j) = N\left(h(\boldsymbol{y}_j, \boldsymbol{x}_k), \boldsymbol{Q}_{k,j}\right).$$

- Then how to compute the MAP of x,y given z?

# 1. From state estimation to least square

- Gaussian distribution (matrix form)

$$\boxed{P\left(\boldsymbol{x}\right) = \frac{1}{\sqrt{\left(2\pi\right)^{N}\det(\boldsymbol{\Sigma})}}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{T}\boldsymbol{\Sigma}^{-1}\left(\boldsymbol{x}-\boldsymbol{\mu}\right)\right).}$$

- Take minus logarithm at both sides:

$$-\ln\left(P\left(\boldsymbol{x}\right)\right) = \frac{1}{2}\ln\left(\left(2\pi\right)^{N}\det\left(\boldsymbol{\Sigma}\right)\right) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{T}\boldsymbol{\Sigma}^{-1}\left(\boldsymbol{x}-\boldsymbol{\mu}\right).$$

Constant w.r.t x

Mahalanobis distance (sigma-norm)

- Maximum of P(x) is equivalent to minimum of –ln(P(x))

# 1. From state estimation to least square

- Take this into the MAP:

Max: $P(\boldsymbol{z}_{j,k}|\boldsymbol{x}_k, \boldsymbol{y}_j) = N\left(h(\boldsymbol{y}_j, \boldsymbol{x}_k), \boldsymbol{Q}_{k,j}\right).$

Information matrix

$$\Rightarrow \quad x_k, y_j = \operatorname{argmin}\left(\left(z_{k,j} - h(y_j, x_k)\right)^T Q_{j,k}^{-1} \left(z_{k,j} - h(y_j, x_k)\right)\right)$$

Error or residual of single observation

- We turn a MAP problem into a least square problem

# 1. From state estimation to least square

- Batch least square
- Original problem

$$\begin{cases} \boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{w}_k\right) \\ \boldsymbol{z}_{k,j} = h\left(\boldsymbol{y}_j, \boldsymbol{x}_k, \boldsymbol{v}_{k,j}\right) \end{cases}.$$

$$x_{MAP} = \text{argmax} P(z|x)P(x|u)$$

Least square

Define the errors(residuals)

$$\begin{aligned} \boldsymbol{e}_{v,k} &= \boldsymbol{x}_k - f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k\right) \\ \boldsymbol{e}_{y,j,k} &= \boldsymbol{z}_{k,j} - h\left(\boldsymbol{x}_k, \boldsymbol{y}_j\right), \end{aligned}$$

- Sum of the squared residuals:

min

$$J(\boldsymbol{x}) = \sum_k e_{v,k}^T \boldsymbol{R}_k^{-1} e_{v,k} + \sum_k \sum_j e_{y,k,j}^T \boldsymbol{Q}_{k,j}^{-1} e_{y,k,j}.$$

# 1. From state estimation to least square

$$J(\boldsymbol{x}) = \sum_k \boldsymbol{e}_{v,k}^T \boldsymbol{R}_k^{-1} \boldsymbol{e}_{v,k} + \sum_k \sum_j \boldsymbol{e}_{y,k,j}^T \boldsymbol{Q}_{k,j}^{-1} \boldsymbol{e}_{y,k,j}.$$

- Some notes:
  - Because of noise, when we take the estimated trajectory and map into the models, they won't fit perfectly
  - Then we adjust our estimation to get a better estimation (minimize the error)
  - The error distribution is affected by noise distribution (information matrix)
- Structure of the least square problem
  - Sum of many squared errors
  - The dimension of total state variable maybe high
  - But single error item is easy (only related to two states in our case)
  - If we use Lie group and Lie algebra, then it's a non-constrained least square

# Contents

- From state estimation to least square
- Batch least square
- Application: estimate camera pose by iterative method

# 2. Batch least square

- How to solve a least square problem?
  - Non-linear, discrete time, non-constrained
- Let’s start from a simple example

- Consider minimizing a squared error:

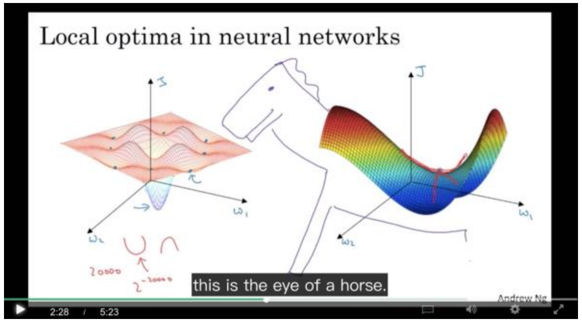
$$\boxed{\min_{x} \frac{1}{2} \| f\ (x) \|_2^2 .}$$

$$x \in \mathbb{R}^n$$

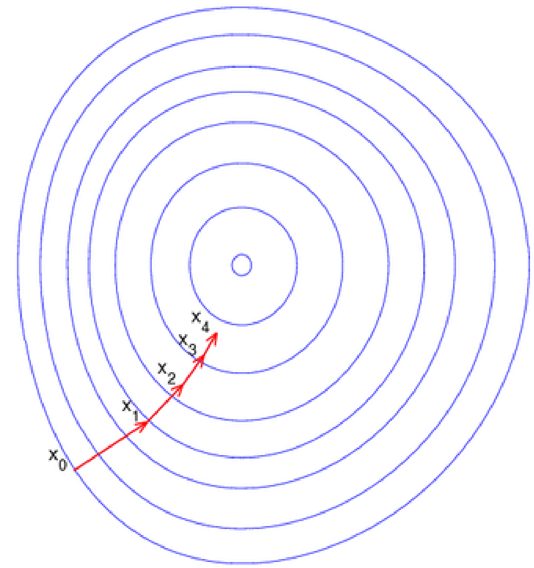
- When f is simple, just solve:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = 0.$$

- And we will find the maxima/minima/saddle points



# 2. Batch least square




- When f is a complicated function:
  - df/dx=0 is hard to solve
  - We use iterative methods
- Iterative methods
  1. Start from a initial estimation $x_0$
  2. At iteration $k$ , we find a incremental $\Delta x_k$ to make $\|f(x_k + \Delta x_k)\|_2^2$ become smaller
  3. If $\Delta x_k$ is small enough, stop (converged)
  4. If not, set $x_{k+1} = x_k + \Delta x_k$ and return to step 2.

# 2. Batch least square

- How to find the incremental part?
- By the gradient
- Taylor expansion of the object function:

$$\|f(\boldsymbol{x}+\Delta\boldsymbol{x})\|_2^2 \approx \|f(\boldsymbol{x})\|_2^2 + \boldsymbol{J}(\boldsymbol{x})\,\Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^T\boldsymbol{H}\Delta\boldsymbol{x}.$$

Jacobian Hessian

- First order methods and second order methods
- First order: (Steepest descent)

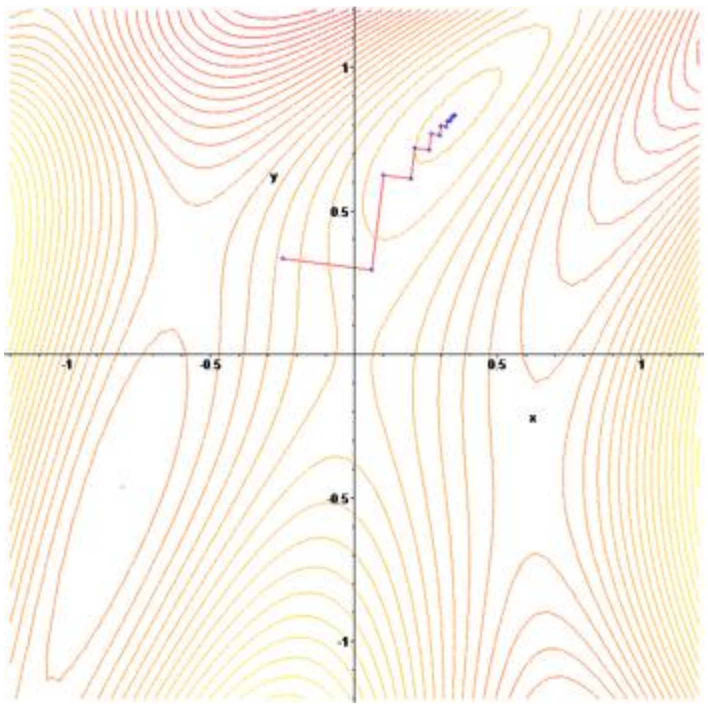
$$\min_{\Delta x}\|f(x)\|_2^2 + J\Delta x$$

Incremental will be: $\Delta\boldsymbol{x}^* = -\boldsymbol{J}^T(\boldsymbol{x}).$

Usually we need a step size

# 2. Batch least square

- Zig-zag in steepest descent




Other shortcomings

- Slow convergence speed
- Slow when close to the minimum

# 2. Batch least square

- Second order methods

$$\boxed{\|f(\boldsymbol{x}+\Delta\boldsymbol{x})\|_2^2 \approx \|f(\boldsymbol{x})\|_2^2 + \boldsymbol{J}(\boldsymbol{x})\,\Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^T \boldsymbol{H}\,\Delta\boldsymbol{x}.}$$

- Solve an increment to minimize it:

$$\Delta\boldsymbol{x}^* = \arg\min \|f(\boldsymbol{x})\|_2^2 + \boldsymbol{J}(\boldsymbol{x})\,\Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^T \boldsymbol{H}\,\Delta\boldsymbol{x}.$$

- Let the derivative to $\Delta x$ be zero, then we get: $\boldsymbol{H}\Delta\boldsymbol{x} = -\boldsymbol{J}^T.$

- This is called Newton's method

# 2. Batch least square

- Second order method converges more quickly than first order methods
- But the Hessian matrix maybe hard to compute: $H\Delta x = -J^T.$

- Can we avoid the Hessian matrix and also keeps second order's convergence speed?
  - Gauss-Newton
  - Levenberg-Marquardt

# 2. Batch least square

- Gauss-Newton
  - Taylor expansion of f(x): $f(\boldsymbol{x}+\Delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}.$
  - Then the squared error becomes:

$$\begin{aligned}\frac{1}{2}\|f(\boldsymbol{x})+\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}\|^2 &= \frac{1}{2}\left(f(\boldsymbol{x})+\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}\right)^T\left(f(\boldsymbol{x})+\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}\right)\\ &= \frac{1}{2}\left(\|f(\boldsymbol{x})\|_2^2 + 2f(\boldsymbol{x})^T\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x} + \Delta\boldsymbol{x}^T\boldsymbol{J}(\boldsymbol{x})^T\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}\right).\end{aligned}$$

  - Also let its derivative with $\Delta x$ be zero:

$$2\boldsymbol{J}(\boldsymbol{x})^T f(\boldsymbol{x}) + 2\boldsymbol{J}(\boldsymbol{x})^T\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x} = \boldsymbol{0}.$$

$$\underbrace{\boldsymbol{J}(\boldsymbol{x})^T\boldsymbol{J}(\boldsymbol{x})}_{H}\Delta\boldsymbol{x} = \underbrace{-\boldsymbol{J}(\boldsymbol{x})^T f(\boldsymbol{x})}_{g}.$$

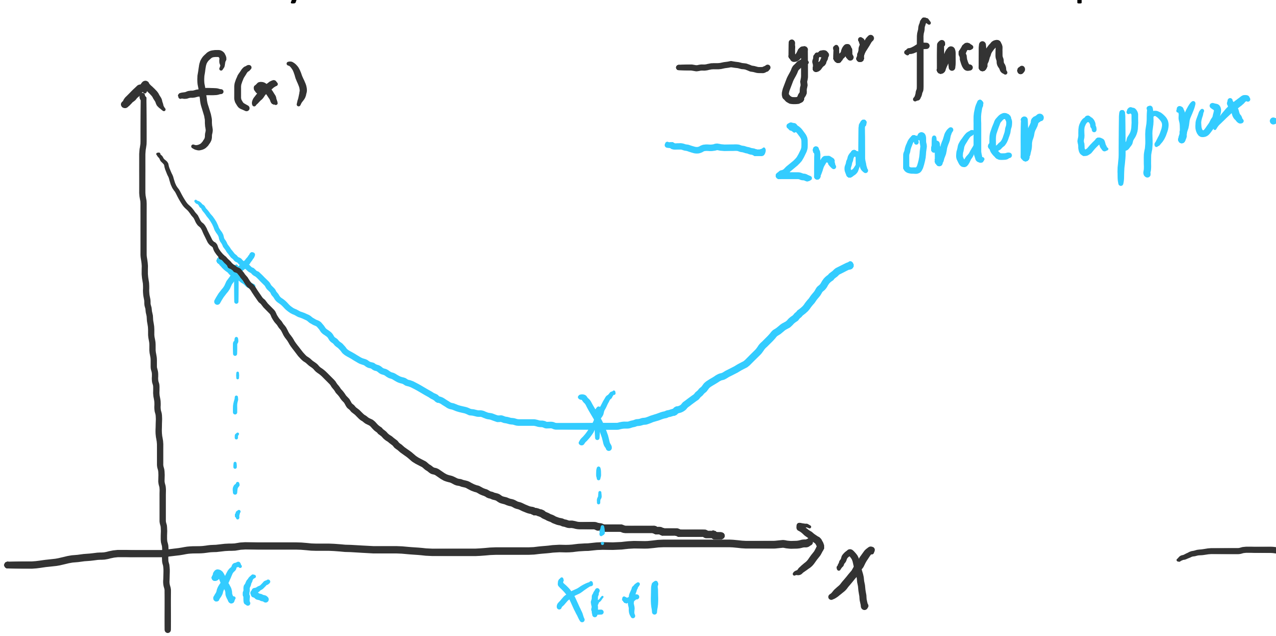
$$\boldsymbol{H}\Delta\boldsymbol{x} = \boldsymbol{g}.$$

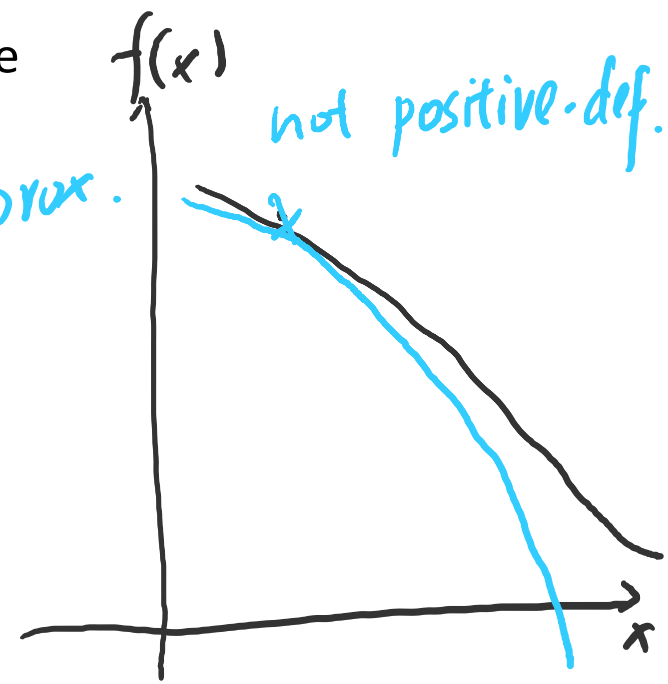
# 2. Batch least square

$$\boldsymbol{J}(\boldsymbol{x})^T \boldsymbol{J}(\boldsymbol{x}) \Delta \boldsymbol{x} = -\boldsymbol{J}(\boldsymbol{x})^T f(\boldsymbol{x}) .$$

- Gauss-Newton use $J(x)^T J(x)$ as an approximation of the Hessian
  - Therefore avoiding the computation of H in the Newton's method

- But $J(x)^T J(x)$ is only semi-positive definite
  - H maybe irreversible when J^T J has null space



# 2. Batch least square

- Levernberg-Marquardt method
  - Trust region approach: approximation is only valid in a region
  - Evaluate if the approximation is good:

$$\rho = \frac{f(\boldsymbol{x} + \Delta\boldsymbol{x}) - f(\boldsymbol{x})}{\boldsymbol{J}(\boldsymbol{x})\,\Delta\boldsymbol{x}}.$$

Real descent/approx. descent

  - If rho is large, increase the region
  - If rho is small, decrease the region

- LM optimization: $\min_{\Delta x_k} \frac{1}{2} \|f(x_k) + J(x_k)\Delta x_k\|^2, s.t. \|\Delta x_k\|^2 \leq \mu$
  - Assume the approximation is only good within a ball

## 2. Batch least square

- Use Lagrange multipliers:

$$\min_{\Delta x_k} \frac{1}{2} \|f(x_k) + J(x_k)\Delta x_k\|^2, s.t. \|\Delta x_k\|^2 \leq \mu$$

$$\Rightarrow \quad \min \frac{1}{2} \|f(x_k) + J(x_k)\Delta x_k\|^2 + \frac{\lambda}{2} \|\Delta x\|^2$$

- Expand it just like in G-N's case, the incremental will be:

$$\left(J(x_k)^T J(x_k) + \lambda I\right)\Delta x_k = g$$

- This $\lambda I$ increase the semi-positive definite property of the Hessian
  - Also balancing the first-order and second-order items

# 2. Batch least square

- Other methods
  - Dog-leg method
  - Conjugate gradient method
  - Quasi-Newton’s method
  - Pseudo-Newton’s method
  - …

- You can find more in optimization books if you are interested

- In SLAM, we use G-N or L-M to solve camera’s motion, pixel’s movement, optical-flow, etc.

# 2. Batch least square

- Problem in the Practical Assignment
- Curve fitting: find best parameters a,b,c from the observation data:

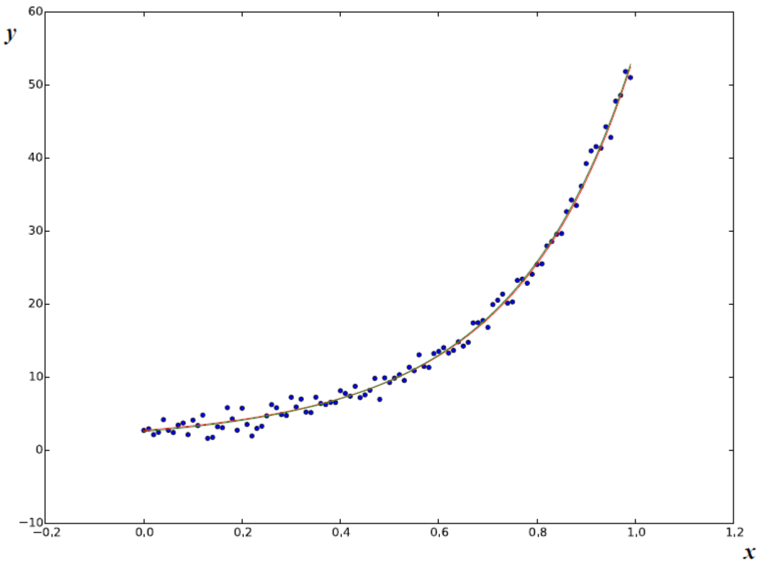
  Curve function: $y = \exp(ax^2 + bx + c) + w,$

- Error:

  $$e_i = y_i - \exp\left(ax_i^2 + bx_i + c\right)$$

- Least square problem:

  $$a, b, c = \operatorname{argmin} \sum_{i=1}^{N} \|y_i - \exp\left(ax_i^2 + bx_i + c\right)\|^2$$



# 2. Batch least square

- You are asked to solve this problem with a hand-written Gauss-Newton's method and use optimization libraries.
- Libraries:
  - Google Ceres Solver http://ceres-solver.org/
  - G2O: https://github.com/RainerKuemmerle/g2o
- You can choose one of them to implement your estimation

# 2. Batch least square

- Google Ceres
  - An optimization library for solving least square problems
  - Tutorial: [http://ceres-solver.org/tutorial.html](http://ceres-solver.org/tutorial.html)
  - Define your residual class as a functor (overload the () operator)

```
struct ExponentialResidual {
  ExponentialResidual(double x, double y)
      : x_(x), y_(y) {}

  template <typename T>
  bool operator()(const T* const m, const T* const c, T* residual) const {
    residual[0] = T(y_) - exp(m[0] * T(x_) + c[0]);
    return true;
  }

 private:
  // Observations for a sample.
  const double x_;
  const double y_;
};
```

# 2. Batch least square

- Build the optimization problem:

```
double m = 0.0;
double c = 0.0;

Problem problem;
for (int i = 0; i < kNumObservations; ++i) {
  CostFunction* cost_function =
       new AutoDiffCostFunction<ExponentialResidual, 1, 1, 1>(
           new ExponentialResidual(data[2 * i], data[2 * i + 1]));
  problem.AddResidualBlock(cost_function, NULL, &m, &c);
}
```

- With auto-diff, Ceres will compute the Jacobians for you

# 2. Batch least square

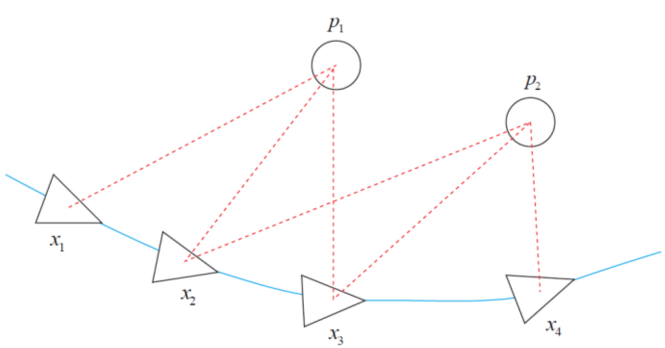
- Finally solve it by calling the Solve() function and get the result summary
- You can set some parameters like number of iterations, stop conditions or the linear solver type.

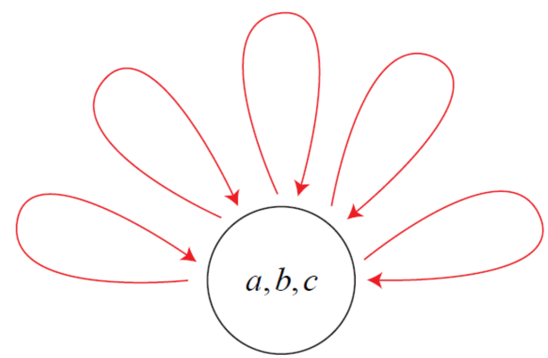
```
Solver::Options options;
options.max_num_iterations = 25;
options.linear_solver_type = ceres::DENSE_QR;
options.minimizer_progress_to_stdout = true;

Solver::Summary summary;
Solve(options, &problem, &summary);
```

# 2. Batch least square

- G2O
  - General Graph Optimization
  - Need to convert the least square problem into a graph
- Graph Optimization
  - State variables are vertices
  - Residuals/Errors are edges connecting those vertices
  - Edges can be unary/binary/multiple



# 2. Batch least square

- Use g2o to solve your least square problem
  - Define your vertices and edges (or use the built-in vertices and edges in g2o)
  - Build the problem by adding vertices and edges into it
  - Set the optimization parameters (linear solver type, iterations, etc.)
  - Call solve function
  - Fetch the results from the graph

# 2. Batch least square

- Tutorial of g2o
  - http://ais.informatik.uni-freiburg.de/publications/papers/kuemmerle11icra.pdf
  - Doc/ in the github repo: https://github.com/RainerKuemmerle/g2o
  - Examples

# 2. Batch least square

- Summary
  - In the batch estimation, we estimate all the status variable given all the measurements and input
  - The batch estimation problem can be formulated into a least square problem, after solving it we get a MAP estimation
  - The least square problem can be solved by iterative methods like gradient descent, Newton's method, Gauss-Newton or Levernberg-Marquardt method
  - The least square problem can also be represented by a graph and forms a (factor) graph optimization problem

# Contents

- From state estimation to least square
- Batch least square
- Application: estimate camera pose by iterative method

# 3. Application: estimate camera pose

- Suppose we want to estimate the camera pose
- We have several observations from the projection function

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_i = K(RP_i + t) = KTP_i$$

- Minimizing the reprojection error:

$$(R, t)^* = T^* = \operatorname{argmin} \frac{1}{2} \sum_{i=1}^{N} \| u_i - \pi(RP_i + t) \|_2^2$$

  - Where $\pi(\cdot)$ is the projection equation (observation model)

# 3. Application: estimate camera pose

- Linearize the error: $e_i(x \oplus \Delta x) \approx e_i(x) + J(x)\Delta x$
- Derivative is defined by SE(3) disturb model:

$$\frac{\partial e}{\partial T} = \lim_{\delta\xi \to 0} \frac{e(\delta\xi \oplus T) - e(T)}{\delta\xi}$$
$$= \lim_{\delta\xi \to 0} \frac{\frac{1}{Z}K(\delta\xi \oplus T)P - \frac{1}{Z}KTP}{\delta\xi}$$

- Let $P' = TP$ then use chain rule: $\frac{\partial e}{\partial T} = \frac{\partial e}{\partial P'}\frac{\partial P'}{\partial T}$
- For $P'$ we have:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix}.$$

$$\Rightarrow \quad u = f_x \frac{X'}{Z'} + c_x, \quad v = f_y \frac{Y'}{Z'} + c_y.$$

$$\frac{\partial e}{\partial \boldsymbol{P}'} = -\begin{bmatrix} \frac{\partial u}{\partial X'} & \frac{\partial u}{\partial Y'} & \frac{\partial u}{\partial Z'} \\ \frac{\partial v}{\partial X'} & \frac{\partial v}{\partial Y'} & \frac{\partial v}{\partial Z'} \end{bmatrix} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} \end{bmatrix}.$$

# 3. Application: estimate camera pose

- The second item: $\frac{\partial(TP')}{\partial T} = \begin{bmatrix} I & -P'^{\wedge} \\ 0^T & 0^T \end{bmatrix}$ See Lecture 2.
- Remove the homogeneous part:

$$\frac{\partial(TP')}{\partial T} = \begin{bmatrix} I & -P'^{\wedge} \end{bmatrix}$$

- Put them together:

$$\frac{\partial e}{\partial T} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} & -\frac{f_x X' Y'}{Z'^2} & f_x + \frac{f_x X^2}{Z'^2} & -\frac{f_x Y'}{Z'} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} & -f_y - \frac{f_y Y'^2}{Z'^2} & \frac{f_y X' Y'}{Z'^2} & \frac{f_y X'}{Z'} \end{bmatrix}.$$

# 3. Application: estimate camera pose

- If we want to take the derivative of Point P

$$s\begin{bmatrix}u\\v\\1\end{bmatrix}_i = K(RP_i + t) = KTP_i$$

$$\frac{\partial e}{\partial P} = \frac{\partial e}{\partial P'}\frac{\partial P'}{\partial P} = -\begin{bmatrix} f_x/Z' & 0 & -f_xX'/Z'^2 \\ 0 & f_y/Z' & -f_yY'/Z'^2 \end{bmatrix} R$$

- P is not relevant to translation t

# 3. Application: estimate camera pose

- We can also compute these Jacobians in SO(3)
- With Jacobian in manifold it will be easy to perform Gauss-Newton iterations to solve the camera's motion iteratively

# Any Questions?