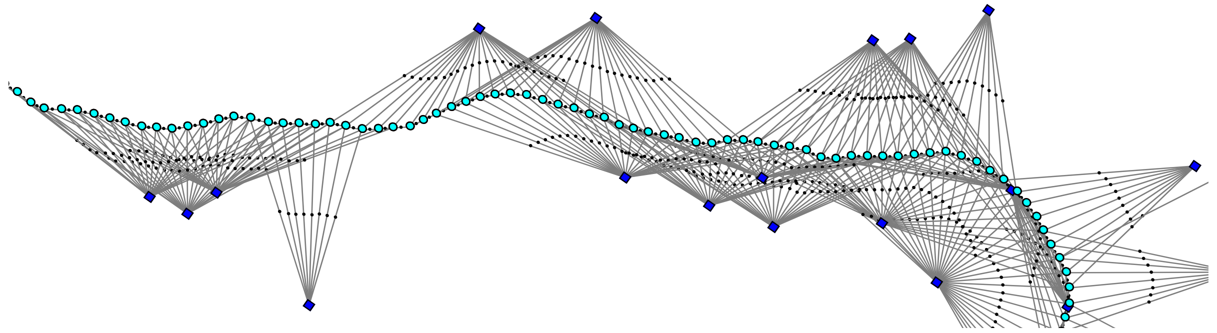
CERES SOLVER



Brief Explanation and Tutorial

Andrew Torgesen

CERES SOLVER: OVERVIEW

- Open-source C++ library for solving
  - nonlinear least squares problems with bounds constraints
  - general unconstrained optimization problems

$$\min_{\boldsymbol{x}} \boldsymbol{r}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{r}(\boldsymbol{x})$$

$$l_i \leq x_i \leq u_i$$

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

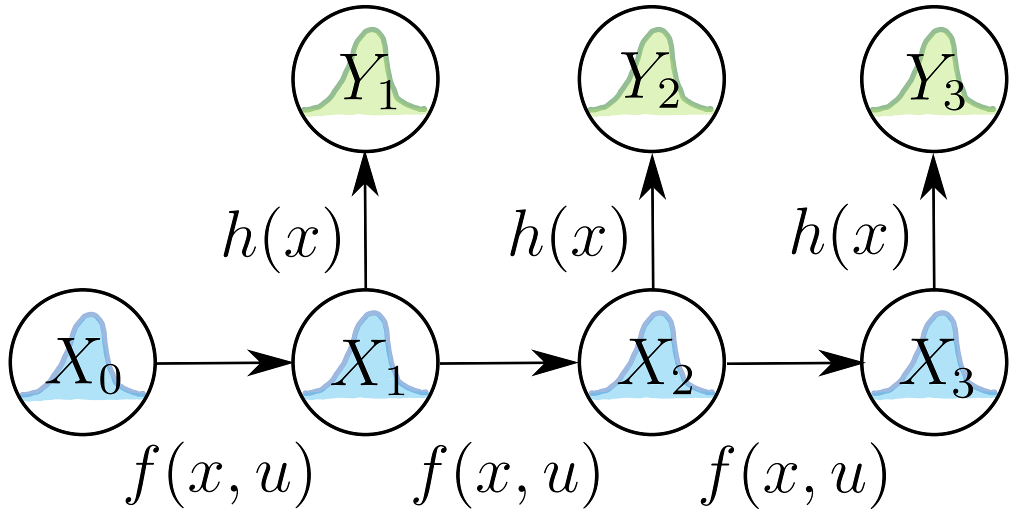
- Solves large-scale estimation problems (like GTSAM)
- Made as a bundle adjustment backend for Google
  - Google maps
  - Android AR / panorama stitching
  - Blender
  - etc.
- *Useful for your problem?*

ESTIMATION AS NONLINEAR OPTIMIZATION

- Bayesian inference: maximize joint probability

$$\max_{\boldsymbol{x}} P(\boldsymbol{x}_k, \ldots | \boldsymbol{y}_k, \ldots)$$

$$\rightarrow \max_{\boldsymbol{x}} \prod_k \exp(-\boldsymbol{r}_k^\top \boldsymbol{Q} \boldsymbol{r}_k)$$



Filtering:

$$\hat{x}_k^- = \int P(X_k | X_{k-1} = x) \hat{x}_{k-1}^+ (X_{k-1} = x) dx$$

$$\hat{x}_k^+ = \eta P(Y_k | X_k) \hat{x}_k^-$$

Nonlinear Optimization (Smoothing):

$$\min_{\boldsymbol{x}} \boldsymbol{r}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{r}(\boldsymbol{x})$$

ESTIMATION AS NONLINEAR OPTIMIZATION

- Estimation/bundle adjustment problem reduces to solving **nonlinear least-squares problem** over all residuals
- Generally solved by a variation on Gauss-Newton local search:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{J}_{\boldsymbol{r}}^{\dagger} \boldsymbol{r}(\boldsymbol{x}_k)$$

$$\boldsymbol{J}_{\boldsymbol{r}} = \begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \cdots & \frac{\partial r_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_m}{\partial x_1} & \cdots & \frac{\partial r_m}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

- Requires computation of *lots* of derivatives!

NONLINEAR OPTIMIZATION ON THE MANIFOLD

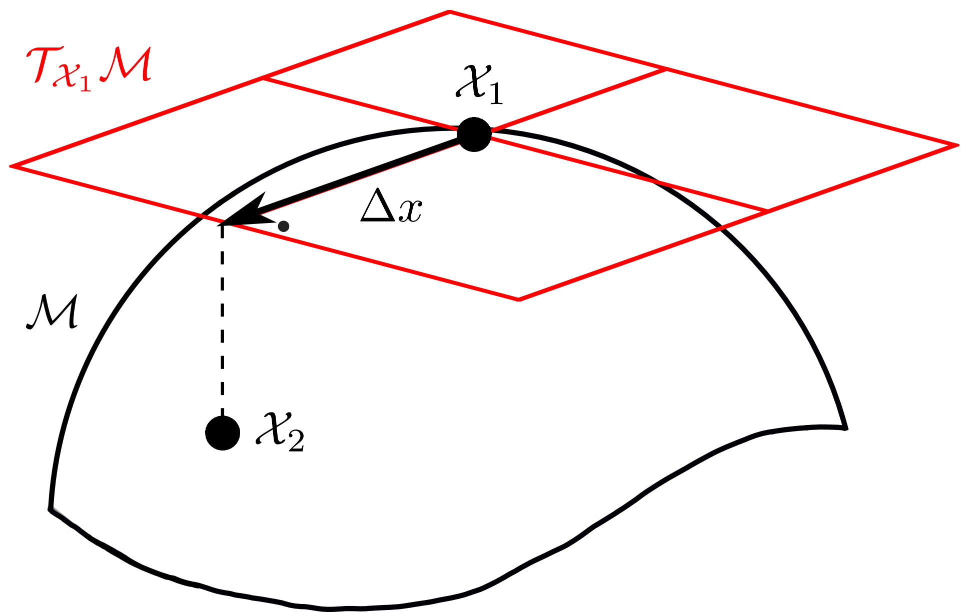
- $\boldsymbol{x}$ stored as vector, but may not be a vector!
  - Examples: rotation averaging, PGO, etc.
- Need to define geodesic maps (or retractions):

$$(\boldsymbol{a} \oplus \boldsymbol{b}) \in \mathcal{M} \quad \boldsymbol{a} \in \mathcal{M}, \boldsymbol{b} \in \mathbb{R}^m \cong \mathfrak{m}$$

$$(\boldsymbol{a} \ominus \boldsymbol{b}) \in \mathbb{R}^m \cong \mathfrak{m} \quad \boldsymbol{a}, \boldsymbol{b} \in \mathcal{M}$$

- …and (the hard part) re-define your Jacobians:

$$\frac{{}^{\mathcal{X}}\partial f(\mathcal{X})}{\partial \mathcal{X}} \triangleq \lim_{\boldsymbol{\tau} \to 0} \frac{f(\mathcal{X} \oplus \boldsymbol{\tau}) \ominus f(\mathcal{X})}{\boldsymbol{\tau}} \in \mathbb{R}^{n \times m}$$



CERES SOLVER: FEATURES

- Variety of local search **solver choices**, like trust region and line search
  - **Fast** and **more accurate** than other nonlinear least squares solvers
- Ability to specify retractions for optimization on the manifold using **local parameterizations**
- Robust **loss functions** for rejecting data outliers
- Built-in **covariance estimation** of posterior solutions
- → ***Auto-Differentiation*** that's probably *just as fast, if not faster,* than your analytic derivatives ←
  - Utilization of dual numbers (“Jet” data type)

CERES VS GTSAM

Ceres Advantages

- Supported by Google, not just one research lab
- Has an awesome automatic differentiation system–no time wasted computing complicated derivatives
- Generalizes well beyond robotics applications, or even exotic robotics applications that don't yet have pre-programmed tools

GTSAM Advantages

- Made specifically for robotics applications, so comes with a lot of useful tools, such as:
- iSAM2 incremental optimization
- Marginalization support (e.g., for fixed-lag smoothing)

TOY PROBLEM: PGO

$$J = \sum_{(i,j)\in\mathcal{E}} ||\left(\hat{\boldsymbol{T}}_i^{-1}\hat{\boldsymbol{T}}_j\right) \ominus \boldsymbol{T}_{ij}||_{\boldsymbol{\Sigma}}^2,$$

$$\boldsymbol{T} \in SE(3).$$

- Going to solve with Ceres (using Python wrappers and the manif-geom-cpp library) by first defining:
  - A *local parameterization* for $\boldsymbol{T}$.
  - A *cost function* for front-end (e.g., VIO) and loop closure measurement residuals.
- Ceres will give us the Jacobians of the above for free!

TOY PROBLEM: LOCAL PARAMETERIZATION

- Define the $\oplus$ operator, with C++ templating.

```
// boxplus operator for both doubles and jets
  template<typename T>
  bool operator()(const T* x, const T* delta, T* x_plus_delta) const
  {
    SE3<T> X(x);
    Eigen::Map<const Eigen::Matrix<T, 6, 1>> dX(delta);
    Eigen::Map<Eigen::Matrix<T, 7, 1>> Yvec(x_plus_delta);

    Yvec << (X + dX).array();

    return true;
  }
```

TOY PROBLEM: RESIDUAL DEFINITION

- Residual is $\hat{\boldsymbol{T}}_{ij} \ominus \boldsymbol{T}_{ij}$, weighted by (inverted) covariance.

```
// templated residual definition for both doubles and jets
// basically a weighted implementation of boxminus using Eigen templated types
template<typename T>
bool operator()(const T* _Xi_hat, const T* _Xj_hat, T* _res) const
{
  SE3<T> Xi_hat(_Xi_hat);
  SE3<T> Xj_hat(_Xj_hat);
  Map<Matrix<T,6,1>> r(_res);
  r = Q_inv_ * (Xi_hat.inverse() * Xj_hat - Xij_.cast<T>());
  return true;
}
```

TOY PROBLEM: DYNAMICS AND COVARIANCES

```
# delta pose/odometry between successive nodes in graph (tangent space representation as a local perturbation)
dx = np.array([1.,0.,0.,0.,0.,0.1])

# odometry covariance
odom_cov = np.eye(6)
odom_cov[:3,:3] *= odom_cov_vals[0] # delta translation noise
odom_cov[3:,3:] *= odom_cov_vals[1] # delta rotation noise
odom_cov_sqrt = np.linalg.cholesky(odom_cov)

# loop closure covariance
lc_cov = np.eye(6)
lc_cov[:3,:3] *= lc_cov_vals[0] # relative translation noise
lc_cov[3:,3:] *= lc_cov_vals[1] # relative rotation noise
lc_cov_sqrt = np.linalg.cholesky(lc_cov)
```

TOY PROBLEM: TRUE STATE + ODOM SIMULATION

```
# optimization problem
problem = ceres.Problem()

# create odometry measurements
for k in range(num_steps):
    if k == 0:
        # starting pose
        xhat.append(SE3.identity().array())
        x.append(SE3.identity().array())

        # add (fixed) prior to the graph
        problem.AddParameterBlock(xhat[k], 7, factors.SE3Parameterization())
        problem.SetParameterBlockConstant(xhat[k])
    else:
        # time step endpoints
```

TOY PROBLEM: ADD LOOP CLOSURE MEASUREMENTS

```
loop_closures = list()
for l in range(num_lc):
    i = np.random.randint(low=0, high=num_steps-1)
    j = np.random.randint(low=0, high=num_steps-2)
    if j == i:
        j = num_steps-1
    loop_closures.append((i,j))

# create loop closure measurements
for l in range(num_lc):
    i = loop_closures[l][0]
    j = loop_closures[l][1]

    # noise-less loop closure measurement
    xi = SE3(x[i])
```

TOY PROBLEM: SOLVE!

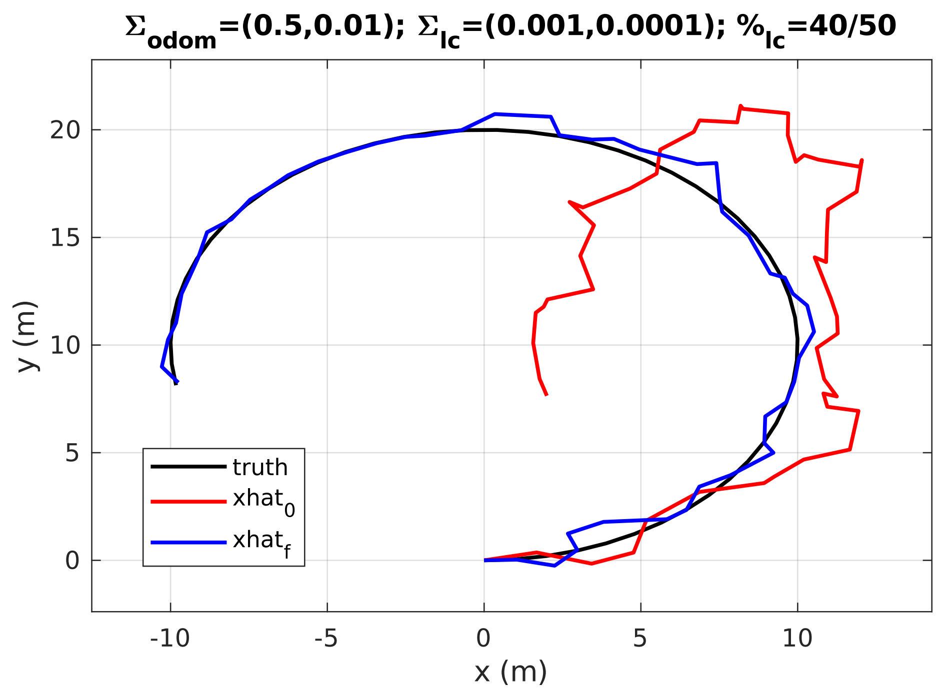
```
# set solver options
options = ceres.SolverOptions()
options.max_num_iterations = 25
options.linear_solver_type = ceres.LinearSolverType.SPARSE_NORMAL_CHOLESKY
options.minimizer_progress_to_stdout = True

# solve!
summary = ceres.Summary()
ceres.Solve(options, problem, summary)
```

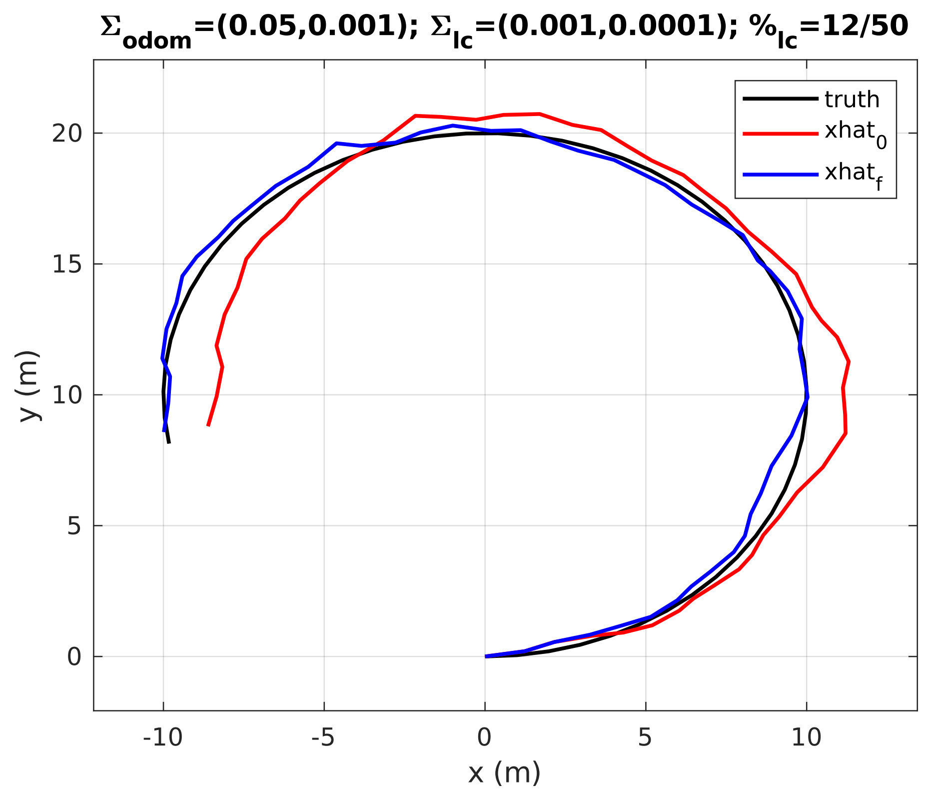
TOY PROBLEM: RESULTS

- Python-wrapped solver completes in one-hundredth of a second.

```
iter      cost      cost_change  |gradient|   |step|    tr_ratio  tr_radius  ls_iter  iter_time  total_time
   0  2.643133e+09    0.00e+00    1.91e+00   0.00e+00   0.00e+00  1.00e+04        0    1.62e-03    2.14e-03
   1  2.872315e+08    2.36e+09    2.86e+02   1.03e+02   8.95e-01  1.97e+04        1    1.98e-03    4.16e-03
   2  9.150302e+05    2.86e+08    4.99e+02   3.11e+01   9.97e-01  5.90e+04        1    1.51e-03    5.69e-03
   3  1.600187e+04    8.99e+05    9.67e+01   5.52e+00   1.00e+00  1.77e+05        1    1.59e-03    7.29e-03
   4  1.536517e+04    6.37e+02    2.95e+01   5.41e+00   1.00e+00  5.31e+05        1    1.48e-03    8.78e-03
   5  1.530853e+04    5.66e+01    3.66e+00   3.38e+00   1.00e+00  1.59e+06        1    1.39e-03    1.02e-02
   6  1.530598e+04    2.55e+00    6.84e-01   8.33e-01   1.00e+00  4.78e+06        1    1.33e-03    1.15e-02
   7  1.530597e+04    1.73e-02    4.78e-02   7.25e-02   1.00e+00  1.43e+07        1    1.31e-03    1.28e-02
Average error of optimized poses: 31.432182 -> 0.040357
```

TOY PROBLEM: RESULTS





RESOURCES

- **ceres-solver.org**
- https://notes.andrewtorgesen.com/doku.php?id=public:ceres
    - **Links to libraries with installation instructions.**
    - 1D SLAM
    - Quaternion averaging
    - PGO
    - PGO with range measurements