Spring 2026

CS 5757: Optimization Methods for Robotics
Homework 2: Numerical Optimization

Due: Thursday, Mar. 5, 2026, at 11:59pm on Gradescope, Total Points: 50

Learning goals for this problem set:

- **Problem 1:** Build intuition around convex QPs, KKT systems, and how to solve them.
- **Problem 2:** Implement common least-squares costs (keypoint matching, smoothness, joint limits) in JAX.
- **Problem 3:** Implement basic algorithms for unconstrained optimization using automatic differentiation.
- **Problem 4:** Implement and tune a solver of your choice for a large-scale problem.

You are **strongly encouraged** to typeset your homework submissions using LaTeX (if you do not know LaTeX yet, now is as good a time as any to learn it—you will need it in grad school anyway). If you opt for handwritten submissions, make sure that they are legible—if we cannot read it, we cannot grade it!

2.1 Differential Inverse Kinematics

A variant of the inverse kinematics (IK) problem we discussed in class is "differential IK," which seeks joint velocities that achieve a desired end-effector twist. If the forward kinematics are $f\colon \mathbb{R}^n \to SE(3)$, the problem is

$$\begin{aligned} \text{find} \quad & \dot{\boldsymbol{q}} \in \mathbb{R}^n \\ \text{s.t.} \quad & \boldsymbol{J}(\boldsymbol{q})\,\dot{\boldsymbol{q}} = \boldsymbol{v}_{\text{des}}, \end{aligned}$$

where $\boldsymbol{v}_{\text{des}} \in \mathbb{R}^6$ is the desired twist.

(a) (1 pts) When the manipulator is redundant ($n > 6$), the system above is underdetermined and we can choose $\dot{\boldsymbol{q}}$ to optimize a secondary objective. A natural choice is to minimize a weighted norm of the joint velocities:

$$\begin{aligned} \min_{\dot{\boldsymbol{q}} \in \mathbb{R}^n} \quad & \frac{1}{2} \sum_{i=1}^{n} w_i\, \dot{q}_i^2 \\ \text{s.t.} \quad & \boldsymbol{J}(\boldsymbol{q})\,\dot{\boldsymbol{q}} = \boldsymbol{v}_{\text{des}}, \end{aligned}$$

where $w_1, \ldots, w_n > 0$ are positive weights. Show this problem can be written as the quadratic program (QP)

$$\begin{aligned} \min_{\dot{\boldsymbol{q}} \in \mathbb{R}^n} \quad & \frac{1}{2} \dot{\boldsymbol{q}}^\top \boldsymbol{W} \dot{\boldsymbol{q}} \\ \text{s.t.} \quad & \boldsymbol{J}\,\dot{\boldsymbol{q}} = \boldsymbol{v}_{\text{des}}, \end{aligned} \tag{1}$$

and explain why this QP is convex.

(b) ✎ (2 pts) Since this QP is convex, any point $(\dot{\boldsymbol{q}}^*, \boldsymbol{\lambda}^*)$ satisfying the KKT conditions is optimal. Show that the KKT conditions reduce to the linear system

$$\begin{bmatrix} \boldsymbol{W} & \boldsymbol{J}^\top \\ \boldsymbol{J} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{q}}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{v}_{\text{des}} \end{bmatrix}. \tag{2}$$

(c) ✎ (2 pts) Since $\boldsymbol{W} \succ 0$, we can eliminate $\dot{\boldsymbol{q}}^*$ from the first block row of (2) and substitute into the second. Show this yields the closed-form solution

$$\dot{\boldsymbol{q}}^* = \boldsymbol{W}^{-1}\boldsymbol{J}^\top \big(\boldsymbol{J}\boldsymbol{W}^{-1}\boldsymbol{J}^\top\big)^{-1} \boldsymbol{v}_{\text{des}}.$$

State a condition under which $\boldsymbol{J}\boldsymbol{W}^{-1}\boldsymbol{J}^\top$ is invertible. For a real robot arm, does this condition always hold?

(d) ✎ (2 pts) Substituting $\dot{\boldsymbol{q}}^*$ into the objective, we can write the optimal cost as a function of the desired twist alone: $f^*(\boldsymbol{v}_{\text{des}}) \triangleq \frac{1}{2}(\dot{\boldsymbol{q}}^*)^\top \boldsymbol{W}\, \dot{\boldsymbol{q}}^*$. Show that

$$\nabla_{\boldsymbol{v}_{\text{des}}} f^*(\boldsymbol{v}_{\text{des}}) = -\boldsymbol{\lambda}^*.$$

In a few words, explain: if we perturb the desired twist by a small $\delta\boldsymbol{v}$, what does $\boldsymbol{\lambda}^*$ tell us about the change in the minimum cost? (This is the *shadow price* interpretation of the Lagrange multipliers.)

(e) ✎ (2 pts) Now suppose we add joint velocity bounds:

$$\begin{aligned} \min_{\dot{\boldsymbol{q}} \in \mathbb{R}^n} \quad & \frac{1}{2}\dot{\boldsymbol{q}}^\top \boldsymbol{W}\, \dot{\boldsymbol{q}} \\ \text{s.t.} \quad & \boldsymbol{J}\, \dot{\boldsymbol{q}} = \boldsymbol{v}_{\text{des}} \\ & \dot{\boldsymbol{q}}_{\min} \le \dot{\boldsymbol{q}} \le \dot{\boldsymbol{q}}_{\max}. \end{aligned}$$

In a few words, explain why this problem no longer admits a closed-form solution.

Solution:

2.2 Retargeting Pt. 1: Residuals

Overview. In Problems 2–4 we build a motion retargeting pipeline: given a human motion-capture sequence, find a trajectory of joint angles and floating-base poses for a Unitree G1 humanoid that reproduces the motion. We formulate retargeting as a nonlinear least-squares problem

$$\min_{\boldsymbol{x}_{1:T}} \frac{1}{2} \left\| \boldsymbol{r}(\boldsymbol{x}_{1:T}) \right\|^2,$$

where each configuration is $\boldsymbol{x}_t = (\boldsymbol{q}_t, \boldsymbol{v}_t)$ with joint angles $\boldsymbol{q}_t \in \mathbb{R}^{29}$ and a log-space pose vector $\boldsymbol{v}_t \in \mathbb{R}^6$ for the floating base. The pose vector maps to a rigid-body transform via the exponential map:

$$\exp(\boldsymbol{v}_t^\wedge) = \boldsymbol{T}_t^{\text{base}} = \begin{bmatrix} \boldsymbol{R}_t & \boldsymbol{p}_t \\ \boldsymbol{0} & 1 \end{bmatrix}, \qquad \boldsymbol{R}_t \in SO(3),\ \boldsymbol{p}_t \in \mathbb{R}^3.$$

Provided helpers. The starter code provides:

- `robot_keypoints_single(cfg, pose_vec, robot, g1_indices)`: maps $(\boldsymbol{q}_t, \boldsymbol{v}_t)$ to world-frame keypoint positions $\in \mathbb{R}^{K\times 3}$ via the exponential map and forward kinematics.
- `pack_trajectory` / `unpack_trajectory`: convert between a flat decision variable $\boldsymbol{x} \in \mathbb{R}^{35T}$ and matrices `joint_angles` $\in \mathbb{R}^{T\times 29}$, `pose_vecs` $\in \mathbb{R}^{T\times 6}$.

Interface. Each residual function below takes `joint_angles` (shape `(T, 29)`), `pose_vecs` (shape `(T, 6)`), and a `RetargetingProblem` (which bundles the robot model, target keypoints, and joint limits). Each function returns a *flat* residual vector. The autograder checks shape and squared norm only, so reorderings of the residual entries are fine.

(a) Keypoint matching. (2 pts) For each frame t and keypoint k, define

$$\boldsymbol{r}_{t,k}^{\text{match}} = \text{FK}_k(\boldsymbol{q}_t, \boldsymbol{v}_t) - \boldsymbol{p}_{t,k}^{\text{target}} \ \in \ \mathbb{R}^3.$$

Implement `residuals_matching`, returning the flattened vector $\boldsymbol{r}^{\text{match}} \in \mathbb{R}^{3KT}$.

Hint: Consider using `jax.vmap`.

(b) Smoothness. (3 pts) Penalize frame-to-frame changes in joint angles and base pose. For $t = 1, \ldots, T-1$:

$$\boldsymbol{r}_t^{\text{vel}} = \boldsymbol{q}_{t+1} - \boldsymbol{q}_t \ \in \ \mathbb{R}^{29},$$
$$\boldsymbol{r}_t^{\text{twist}} = \log\big((\boldsymbol{T}_{t+1}^{\text{base}})^{-1}\, \boldsymbol{T}_t^{\text{base}}\big) \ \in \ \mathbb{R}^6.$$

Implement `residuals_smoothness`, returning the flattened vector

$$\boldsymbol{r}^{\text{smooth}} = \begin{bmatrix} \boldsymbol{r}_1^{\text{vel}} \\ \boldsymbol{r}_1^{\text{twist}} \\ \vdots \\ \boldsymbol{r}_{T-1}^{\text{vel}} \\ \boldsymbol{r}_{T-1}^{\text{twist}} \end{bmatrix} \ \in \ \mathbb{R}^{35(T-1)}.$$

(c) **Rest-pose regularization.** (3 pts) Penalize deviations from the zero configuration:

$$\boldsymbol{r}_t^{\text{rest}} = \boldsymbol{q}_t \in \mathbb{R}^{29}.$$

Implement `residuals_rest`, returning the flattened vector $\boldsymbol{r}^{\text{rest}} \in \mathbb{R}^{29T}$.

(d) **Joint limits.** (3 pts) Penalize violations of the joint limits $\underline{\boldsymbol{q}} \leq \boldsymbol{q}_t \leq \overline{\boldsymbol{q}}$:

$$\begin{aligned}\boldsymbol{r}_t^{\text{over}} &= \max(\boldsymbol{q}_t - \overline{\boldsymbol{q}},\ \mathbf{0}) \in \mathbb{R}^{29},\\ \boldsymbol{r}_t^{\text{under}} &= \max(\underline{\boldsymbol{q}} - \boldsymbol{q}_t,\ \mathbf{0}) \in \mathbb{R}^{29}.\end{aligned}$$

Implement `residuals_limits`, returning the flattened vector

$$\boldsymbol{r}^{\text{limit}} = \begin{bmatrix} \boldsymbol{r}_1^{\text{over}} \\ \boldsymbol{r}_1^{\text{under}} \\ \vdots \\ \boldsymbol{r}_T^{\text{over}} \\ \boldsymbol{r}_T^{\text{under}} \end{bmatrix} \in \mathbb{R}^{58T}.$$

(e) (2 pts) In a few words, explain how $\boldsymbol{r}_t^{\text{over}}$ and $\boldsymbol{r}_t^{\text{under}}$ penalize constraint violations. How does this approach differ from the hard constraints a QP solver would impose (as in question 1d).

Solution:

2.3 Retargeting Pt. 2: Solvers

We now implement several solvers for the retargeting problem. Each solver is a *step function*

$$\texttt{step\_fn(x, state) -> (x\_new, state\_new)},$$

where $\texttt{x} \in \mathbb{R}^{35T}$ is the flat decision variable and `state` is a `dict` of optimizer bookkeeping. The outer loop `run_optimizer` (provided) calls your step function repeatedly. Hyperparameters and objective functions are bound via `functools.partial`.

Two forms of the objective are available: a **scalar cost** $f\colon \mathbb{R}^n \to \mathbb{R}$ (`cost_fn`), and a **residual** $\boldsymbol{r}\colon \mathbb{R}^n \to \mathbb{R}^m$ (`residual_fn`), related by $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{x})\|^2$.

(a) 💻 (3 pts) Implement `gradient_descent_step(cost_fn, x, state, lr)`. Apply:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha\, \nabla f(\boldsymbol{x}_k),$$

where α is `lr`. Use `jax.grad` for the gradient. Return `state` unchanged.

(b) 💻 (3 pts) Implement `gauss_newton_step(residual_fn, x, state, damping)`. Compute the Jacobian $\boldsymbol{J} = \partial \boldsymbol{r}/\partial \boldsymbol{x} \in \mathbb{R}^{m\times n}$ using `jax.jacrev`, then solve the damped normal equations

$$\left(\boldsymbol{J}^\top \boldsymbol{J} + \lambda \boldsymbol{I}\right) \Delta \boldsymbol{x} = -\boldsymbol{J}^\top \boldsymbol{r}(\boldsymbol{x}_k)$$

for $\Delta \boldsymbol{x}$ using `jnp.linalg.solve`, and set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta \boldsymbol{x}$. The damping λ (`damping`) is the Levenberg–Marquardt regularizer. Return `state` unchanged.

(c) 💻 (3 pts) Implement `gradient_descent_line_search_step(cost_fn, x, state, candidate_alphas)`. Compute $\boldsymbol{g} = \nabla f(\boldsymbol{x}_k)$, then select the best step size from a provided array of candidates:

$$\alpha^\star = \arg\min_{\alpha_i}\ f(\boldsymbol{x}_k - \alpha_i\, \boldsymbol{g}), \qquad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha^\star\, \boldsymbol{g}.$$

Return `state` unchanged.

Hint: Use `jax.vmap` to evaluate all candidates in parallel.

(d) 💻 (3 pts) Implement `gauss_newton_matrix_free_step(residual_fn, x, state, damping, cg_tol, cg_maxiter)`. Solve the same system as part (b), but *without materializing* $\boldsymbol{J}$. Instead:

i. Implement $\boldsymbol{v} \mapsto \boldsymbol{J}\boldsymbol{v}$ via `jax.jvp` and $\boldsymbol{v} \mapsto \boldsymbol{J}^\top \boldsymbol{v}$ via `jax.vjp`.

ii. Define a function $\boldsymbol{v} \mapsto (\boldsymbol{J}^\top \boldsymbol{J} + \lambda \boldsymbol{I})\, \boldsymbol{v}$ by composing these products.

iii. Solve the system using `jax.scipy.sparse.linalg.cg`, which accepts a callable as its matrix argument: `cg(matvec_fn, b, tol=cg_tol, maxiter=cg_maxiter)`.

Hint: Make sure to pass the parameters `cg_tol`, `cg_maxiter` through to `cg` - we can use these to speed up the solver by only approximately solving the normal equations.

Solution:

2.4 Retargeting Pt. 3: Reflections and Custom Solver

The script `run_retargeting.py` runs your solvers on several trajectories from the CMU Mocap dataset [1].

(a) (1 pts) For simple walking trajectories, the basic solvers produce reasonable results. For more complex motions (e.g., `cartwheel.npz`), you will observe non-physical artifacts such as flips. What causes these artifacts? What other non-physical behavior do you observe?

(b) (1 pts) Run the dense Gauss–Newton solver on a longer trajectory. It will likely crash. What resource is the bottleneck, and why?

(c) (10 pts) Implement a custom optimizer in `custom_optimizer_step`. You may use any method (zero-, first-, or second-order). You may also use `custom_optimizer_init`, which has access to the cost/residual functions, the default initialization, and the `problem` object, to implement a custom optimizer state or initialization.
Some ideas:

- A momentum-based first-order method (e.g., Nesterov momentum, Adam). See *Algorithms for Optimization* [2], Ch. 5.
- A better initialization scheme for the orientations. We provide identity orientations as the initial guess, which can induce pretty bad local minima. You could instead find translations/rotations for each frame that align the torso points.
- A sampling-based method (Cross-Entropy Method, simulated annealing, etc.; Ch. 8 of [2]).

Creativity. You should implement a *non-trivial* improvement (either initialization scheme or algorithm) to improve the optimizer performance. We will only award partial credit to solutions that simply tune the hyperparameters of the methods we already implemented.

Leaderboard. On held-out trajectories, we will evaluate $\texttt{score} = \texttt{objective} + \beta \cdot \texttt{time}$, where `objective` is the final cost after 100 iterations, `time` is the wall-clock runtime in seconds, and β will be decided later. The **3** top-scoring students receives 1% extra credit toward their final grade. (Which is 4 more point on this homework)

(d) (2 pts) Briefly describe the solver you chose and why. What advantages did it have over Gauss-Newton? What limitations did it have?

Solution:

2.5 AI Usage Survey

✎ (2 pts) Please describe how you used AI (if at all) while completing this homework. Note that you will not be penalized for AI usage, regardless of how extensively you used it, as long as you acknowledge it. We are generally curious about how we can improve the course material and better help with your learning!

Solution:

References

[1] CMU Graphics Lab. Cmu graphics lab motion capture database. `http://mocap.cs.cmu.edu/`, 2003. Funding from NSF EIA-0196217.

[2] Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization*. MIT Press, Cambridge, MA, 2019.