# OpenSim Moco Cheat Sheet for the Matlab Interface

## MocoStudy

### MocoProblem

Access the MocoProblem from the study.

```
problem = study.updProblem();
```

**Set the model.**

```
problem.setModel(Model('model_file.osim'));
```

**Set variable bounds.**
Set initial time to 0; final time between 0.5 and 1.5 s.

```
problem.setTimeBounds(MocoInitialBounds(0),
    MocoFinalBounds(0.5, 1.5));
```

The coordinate value must be between 0 and $\pi$ over the phase, and its initial value is 0 and its final value is $\pi/2$.

```
problem.setStateInfo('/jointset/j0/q0/value',
    [0, pi], 0, pi/2);
```

The control for actuator '/tau0' must be within [-50, 50] over the phase.

```
problem.setControlInfo('/tau0', [-50, 50]);
```

**Optimize static model properties.**
Create parameter 'myparam' to optimize the mass of Body '/bodyset/b0' within [0.1, 0.5].

```
problem.addParameter(MocoParameter('myparam',
    '/bodyset/b0', 'mass', MocoBounds(0.1, 0.5)));
```

**Add goals to the problem.**

Control · ControlTracking · FinalTime
StateTracking · MarkerTracking · TranslationTracking
OrientationTracking · JointReaction

Minimize the sum of squared controls with weight 1.5.

```
problem.addGoal(MocoControlGoal('effort', 1.5));
```

**Add path constraints to the problem.**
Define time-dependent bounds for controls.

```
pathCon = MocoControlBoundConstraint();
problem.addPathConstraint(pathCon);
```

### MocoSolver

**Initialize the CasADi or Tropter solver.**

```
solver = study.initCasADiSolver();
% alternative: solver = study.initTropterSolver();
```

**Settings for Tropter and CasADi solvers.**
Solve the problem on a grid of 50 mesh intervals.

```
solver.set_num_mesh_intervals(50);
```

Transcribe the optimal control problem with the Hermite-Simpson scheme (alternative: 'trapezoidal').

```
solver.set_transcription_scheme('hermite-simpson');
```

Loosen the convergence and constraint tolerances.

```
solver.set_optim_convergence_tolerance(1e-3);
solver.set_optim_constraint_tolerance(1e-3);
```

Stop optimization after 500 iterations.

```
solver.set_optim_max_iterations(500);
```

By default, the Hessian is approximated from first derivatives. Set to 'exact' to use an exact Hessian.

```
solver.set_optim_hessian_approximation('exact');
```

Create a guess, randomize it, then set the guess.

```
guess = solver.createGuess(); guess.randomizeAdd();
solver.setGuess(guess);
```

Set the guess from a MocoTrajectory or MocoSolution file.

```
solver.setGuessFile('previous_solution.sto');
```

**Settings for only CasADi solver.**
By default, CasADi uses 'central' finite differences; 'forward' differences are faster but less accurate.

```
solver.set_finite_difference_scheme('forward');
```

Turn off parallel calculations.

```
solver.set_parallel(0);
```

Monitor solver progress by writing every 10th iterate to file.

```
solver.set_output_interval(10);
```

Solve the study and obtain a MocoSolution.

```
solution = study.solve();
```

Visualize the solution.

```
study.visualize(solution);
```

Compute outputs from the solution.

```
outputs = StdVectorString();
outputs.add('.*active_force_length_multiplier');
table = study.analyze(solution, outputs);
```

## MocoTrajectory and MocoSolution

Create a MocoTrajectory.

```
traj = MocoTrajectory('MocoStudy_solution.sto');
traj = MocoTrajectory.createFromStatesControlsTables(
    states, controls);
```

Get time information.

```
traj.getNumTimes();
traj.getInitialTime(); trajectory.getFinalTime();
traj.getTimeMat();
```

Get names of variables.

```
traj.getStateNames(); traj.getControlNames();
traj.getMultiplierNames(); traj.getParameterNames();
```

Get the trajectory/value for a single variable by name.

```
traj.getStateMat(name); traj.getControlMat(name);
traj.getMultiplierMat(name); traj.getParameter(name);
```

Get the trajectories/values for all variables of a given type.

```
traj.getStatesTrajectoryMat();
traj.getControlsTrajectoryMat();
traj.getMultipliersTrajectoryMat();
traj.getParametersMat();
```

Change the number of times in the trajectory.

```
traj.resampleWithNumTimes(150);
```

Set variable values.

```
traj.setTime(times)
traj.setState(stateTraj); traj.setControl(controlTraj);
traj.setParameter(value);
traj.setStatesTrajectory(statesTraj);
traj.insertStatesTrajectory(subsetStates);
```

Randomize the variable values.

```
traj.randomizeAdd();
```

Export the trajectory.

```
traj.write('mocotrajectory.sto');
traj.exportToStatesTable()
traj.exportToStatesTrajectory(mocoProblem)
```

Compare two trajectories.

```
traj.isNumericallyEqual(otherTraj);
traj.compareContinuousVariablesRMS(otherTraj);
traj.compareParametersRMS(otherTraj);
```

Functions on only MocoSolution.

```
solution.success(); solution.getStatus();
solution.getObjective(); solution.getNumIterations();
solution.getSolverDuration();
solution.unseal(); % Access a failed solution.
```

## The Moco Optimal Control Problem

| | | |
|---|---|---|
| minimize | $\sum_j w_j J_j\left(t_0, t_f, y_0, y_f, x_0, x_f, \lambda_0, \lambda_f, p, \int_{t_0}^{t_f} s_{c,j}(t,y,x,\lambda,p)\,dt\right)$ | `problem.addGoal()` |
| subject to | $\dot{q} = u$ | |
| | $M(q,p)\dot{u} + G(q,p)^T\lambda = f_{\mathrm{app}}(t,y,x,p) - f_{\mathrm{bias}}(q,u,p)$ | |
| | $\dot{z}_{\mathrm{ex}}(t) = f_{\mathrm{aux,ex}}(t,y,x,\lambda,p) \qquad 0 = f_{\mathrm{aux,im}}(t,y,\dot{z}_{\mathrm{im}},x,\lambda,p)$ | `problem.setModel()` |
| | $0 = \phi(q,p)$ | |
| | $0 = \nu(q,u,p)$ | |
| | $0 = \alpha(q,u,\dot{u},p)$ | |
| | $g_L \leq g(t,y,x,\lambda,p) \leq g_U$ | `problem.addPathConstraint()` |
| | $V_{L,k} \leq V_k\left(t_0, t_f, y_0, y_f, x_0, x_f, \lambda_0, \lambda_f, p, \int_t s_{e,k}(t,y,x,\lambda,p)\,dt\right) \leq V_{U,k}$ | `problem.addGoal()` |
| | $y_{0,L} \leq y_0 \leq y_{0,U} \qquad y_{f,L} \leq y_f \leq y_{f,U}$ | `problem.setStateInfo()` |
| | $x_{0,L} \leq x_0 \leq x_{0,U} \qquad x_{f,L} \leq x_f \leq x_{f,U}$ | `problem.setControlInfo()` |
| with respect to | $y \in [y_L, y_U] \qquad x \in [x_L, x_U]$ | |
| | $t_0 \in [t_{0,L}, t_{0,U}] \qquad t_f \in [t_{f,L}, t_{f,U}]$ | `problem.setTimeBounds()` |
| | $p \in [p_L, p_U]$ | `problem.addParameter()` |

| | | | | | |
|---|---|---|---|---|---|
| $t$ | time | $w_j$ | weight for the $j$-th cost | $\phi$ | position-level constraints |
| $q(t)$ | generalized coordinates | $J_j$ | the $j$-th cost | $\nu$ | velocity-level constraints |
| $u(t)$ | generalized speeds | $s_{c,j}$ | integrand used in the $j$-th cost | $\alpha$ | acceleration-level constraints |
| $z(t)$ | auxiliary states | $M$ | mass matrix | $g$ | path constraints |
| $y(t)$ | $(q(t), u(t), z(t))$ | $f_{\mathrm{bias}}$ | centripetal and coriolis forces | $V_k$ | the $k$-th endpoint constraint |
| $x(t)$ | controls | $G$ | kinematic constraint Jacobian | $K$ | number of endpoint constraints |
| $p$ | constant parameters | $f_{\mathrm{app}}$ | applied forces | $s_{e,k}$ | integrand in $k$-th endpoint con. |
| $\lambda$ | kinematic constraint multipliers | $f_{\mathrm{aux}}$ | auxiliary dynamics (explicit & implicit) | $(\ )_U$ | an upper bound |
| | | | | $(\ )_L$ | a lower bound |