Tutorial 7 – Torque Driven Treatment Optimization

The Treatment Optimization toolset generates predictive simulations of a patient's post-treatment movement function by optimizing specified treatment design parameters. The toolset consists of three tools designed using a "theme and variation" approach, where the tools are intended to be used in a specific order, with each tool serving a distinct purpose. Each tool uses the GPOPS-II direct collocation optimal control software for MATLAB and maintains a consistent structure for data inputs, problem design, cost function terms, constraint terms, and outputs, with variations.

This first section of the treatment optimization tutorial will cover the process of running Tracking Optimization (TO), Verification Optimization (VO), and Design Optimization (DO) in which joints are controlled by individual torque actuators. It will work through the design of settings files for all tools, along with analyzing the outputs of each tool with iterative problem-solving methods to generate good solutions.

The model used for this tutorial is the RCNL2024.osim model with the pelvis chosen as the root segment (i.e., the segment that connects the model to ground.) All segments are kept in the model and the reference motion is three-dimensional, but only the right hip flexion, knee angle, and ankle angle motions are allowed to change. Limiting the predicted motion to the sagittal plane of only the kicking leg significantly reduces computation time for tutorial purposes.

Since we use repeated inverse dynamics analyses to generate predictive simulations, the root segment in the model will have unrealistic residual forces and torques acting on it. For the kicking model, root segment residual loads will therefore be present on the left calcaneus. If the model and data were perfect, these residual loads would be zero during the first 0.13 seconds of Tracking Optimization since both feet are off the ground in that time window. Furthermore, if the model and data were perfect, these residual loads would equal the experimental ground reactions during from 0.13 to 0.31 seconds since the left foot is on the ground in that time window. However, neither root segment residual load situation will be satisfied due to errors in the model and data.

For each predictive simulation, the motion of each generalized coordinate in the model is either i) predicted (i.e., trajectory changeable as a function of time) or ii) prescribed (i.e., trajectory pre-defined as a function of time). If the coordinate name is included in the `<states_coordinate_list>` field, then motion of the coordinate will be predicted. If the coordinate name is not included in the `<states_coordinate_list>` field, then motion of the coordinate will be prescribed directly from the inverse kinematics data found in the `<tracked_quantities_directory>.`

For each generalized coordinate whose motion is predicted, if the coordinate name is also listed in the `<RCNLTorqueController>` field, then a force or torque controller is created to control that coordinate. For consistency with forces and torques calculated by inverse dynamics, the user should also add a `<kinetic_consistency>` constraint to ensure that the force or torque produced by the controller closely matches the corresponding force or torque calculated via inverse dynamics using the predicted motion and loads. Note, however, that a generalized coordinate can also have its motion predicted without having an associated torque controller.

Whether or not a torque controller should be created for a generalized coordinate depends on how the motion of the coordinate will be predicted. Generalized coordinate motion can be predicted by either i) tracking a reference coordinate motion, ii) tracking (or minimizing) a reference coordinate load (i.e., a control torque or inverse dynamics load), or iii) tracking both a reference coordinate motion and a reference coordinate load simultaneously. For Tracking Optimizations, both a reference coordinate motion and a reference coordinate load should be tracked simultaneously for each predicted coordinate, thereby producing dynamically consistent simulations that closely reproduce both inverse kinematics and inverse dynamics data simultaneously (i.e., spread out errors). Thus, both motion and load can be tracked simultaneously for the same coordinate during Tracking Optimizations. In contrast, for Verification and Design Optimizations, either motion or load should be tracked for the same coordinate. Thus, after Tracking Optimization is completed, the user should make a choice about how the motion of each generalized coordinate should be predicted, and then either track the reference coordinate motion (no `<RCNLTorqueController>` needed) or track/minimize the reference coordinate load `(<RCNLTorqueController>` required).

Section 1: Tracking Optimization

The Tracking Optimization (TO) tool uses a personalized model to produce a dynamically consistent movement simulation that closely reproduces all available experimental motion data, including joint motions, joint moments, ground reaction forces and moments, and muscle activations. To achieve a dynamically consistent motion, the tool spreads out matching errors between the different experimental quantities based on user-specified maximum allowable errors.

The tool accepts a post-JMP OpenSim model (.osim fle) and personalized NMSM Pipeline model (.osimx fle) along with experimental IK motions, ID loads, ground reactions, muscle–tendon lengths and velocities, muscle moment arms, and, if using synergy controls, NCP results for the trial of interest.

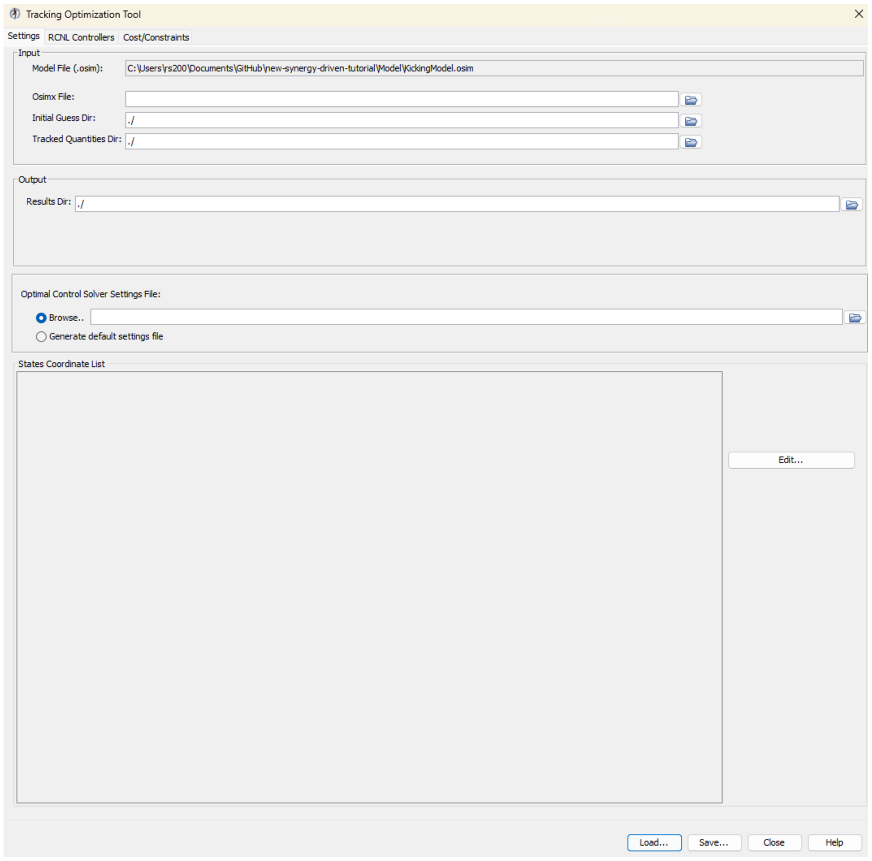
This section of the tutorial will only be using torque controls with no external forces, so the only inputs we will use are a post-JMP OpenSim model, experimental IK motions, and ID loads.

Before running Tracking Optimization:

1. Open the OpenSim model **KickingModel.osim** in the OpenSim GUI.
2. This model is a full three-dimensional gait model, but similar to the model personalization tutorials, this tutorial will only be using the right hip flexion, knee angle, and ankle angle coordinates.
3. This model uses a ground-pelvis joint, so inverse dynamics will yield unrealistic “residual” loads applied to the pelvis body.
4. With the model open in OpenSim, load the IK results in preprocessed (**preprocessed\IKData\drive_kick1.sto**) to visualize the motion.

Creating a Tracking Optimization settings file:

1. Activate the NMSM GUI in OpenSim by navigating to *Tools>User Plugins*, and click **rcnlPlugin.dll**
2. With **KickingModel.osim** selected in the OpenSim GUI, navigate to *Tools>Treatment Optimization >Tracking Optimization*
    a. The following window should be opened:


Tracking Optimization Tool

Settings | RCNL Controllers | Cost/Constraints

Input
Model File (.osim): C:\Users\rs200\Documents\GitHub\new-synergy-driven-tutorial\Model\KickingModel.osim
Osimx File:
Initial Guess Dir: ./
Tracked Quantities Dir: ./

Output
Results Dir: ./

Optimal Control Solver Settings File:
[✓ Browse..]
Generate default settings file

States Coordinate List

3. Leave the *Osimx file* field empty. An input OsimX file would be used if we wanted external loads or muscles to be used in the TO, but this run will not use either.
4. Set the *initial guess directory* to be **preprocessed**
5. Set the *tracked quantities directory* to be **preprocessed**
6. Set the *trial prefix* to be **drive_kick1**
7. Set the *results directory* to be **TorqueTOResults**
8. Set the *optimal control solver settings file* to be **gpopsSettings.xml**

9. Under *states coordinate list*, select (**hip_flexion_r, knee_angle_r, ankle_angle_r**)
    a. The states coordinate list specifies which coordinates the optimizer is allowed to change. <u>If a coordinate is not included in the states coordinate list, it is automatically prescribed.</u> Therefore, most of the coordinates in this tutorial are prescribed.
10. Click to the *RCNL Controllers* tab at the top.
11. Under *RCNL Torque Controller*, add (**hip_flexion_r, knee_angle_r, ankle_angle_r**) to the coordinate list.
    a. This creates individual torque actuators for each of the coordinates that we include in the states.
12. Click to the *Cost/Constraints* tab at the top.
    a. Hint for the upcoming section: When you click on the GUI drop down arrows to select cost term type, you can type the name of the cost term and the GUI will auto-navigate to the correct cost term.
13. **Cost terms:**
    a. Add a new cost term:
        i. Name: **Coordinate tracking**
        ii. Cost term type: **generalized_coordinate_tracking**
        iii. Component list: (**hip_flexion_r knee_angle_r ankle_angle_r**)
        iv. Max allowable error: **0.1 rad (5.7 deg)**
    b. Add a new cost term:
        i. Name: **Speed tracking**
        ii. Cost term type: **generalized_speed_tracking**
        iii. Component list: (**hip_flexion_r knee_angle_r ankle_angle_r**)
        iv. Max allowable error: **1**
    c. Add a new cost term:
        i. Name: **Load tracking**
        ii. Cost term type: **inverse_dynamics_load_tracking**
        iii. Component list: (**hip_flexion_r_moment knee_angle_r_moment ankle_angle_r_moment**)
        iv. Max allowable error: **50**

14. **Constraint terms:**
    a. Add a new constraint term:
        i. Name: **Kinetic consistency**
        ii. Constraint term type: **kinetic_consistency**
        iii. Component list: (**hip_flexion_r_moment knee_angle_r_moment ankle_angle_r_moment**)
        iv. Max error: **0.1**
        v. Min error: **-0.1**
15. Save this settings file as **TorqueTOSettings.xml**
16. Open **TorqueTOSettings.xml** in a text editor of your choice and explore it.

Running Tracking Optimization:

1. Open MATLAB and open **runTOTool.m** in your tutorial directory.
2. Open the project file (**Project.prj** inside your installation of nmsm-core.)
3. Run the MATLAB file **runTOTool.m**

Post TO Analysis:

1. The script will create 4 plots for you:
    a. Joint Angles: Joint angles for all model coordinates (including prescribed) as output by the TO run.
    b. Joint Velocities: Joint velocities for all states coordinates.
    c. Joint Loads: Joint loads for all model coordinates.
    d. Torque Controls: Torque controls for all coordinates actuated by a torque controller.
2. These plots are a valuable way to analyze the results of the TO run. RMSE values between the tracked data and the TO results are reported for every plot where applicable.
3. It is also valuable to visualize the motion in the OpenSim GUI
    a. With the model selected in OpenSim, load the newly created IK motion in your TO results directory. Ensure the motion looks as close to the experimental motion as possible.

4. Getting a good TO run is hard and often requires additional iteration after the first attempt. It is recommended to add/remove cost terms if you believe the problem would benefit, or change max allowable errors for cost terms to “nudge” the solution in a desired direction.

Alternative TO Formulations

A key function of Tracking Optimization is the ability to reduce root segment residual loads. This is important to ensure that your model accurately represents the real subject so that you can design a treatment for them. Reducing residual loads is not done in the main part of this tutorial because the optimization takes significantly longer to finish.

This model does not have external force data, so some residual loads when the left foot is in contact with the ground are to be expected. When both feet are out of contact with the ground at the start of the motion though, there ideally should be no residual loads acting on the pelvis. To demonstrate the residual load reduction functionality, we can reduce residual loads to zero while the model is in the air.

1. In the NMSM OpenSim GUI, load **TorqueTOSettings**
2. Change the *results directory* to be **TorqueTOSettingsResiduals**
3. Set the *states coordinate list* to be (**hip_flexion_r knee_angle_r ankle_angle_r hip_flexion_l knee_angle_l ankle_angle_l pelvis_tilt pelvis_tx pelvis_ty lumbar_extension arm_flex_r arm_flex_l**)
4. Add a new constraint term:
    a. Name: **Residual load reduction**
    b. Constraint term type: **root_segment_residual_load**
    c. Component list: (**pelvis_tilt_moment pelvis_tx_force pelvis_ty_force**)
    d. Max error: **1**
    e. Min error: **-1**
5. Save this settings file as **TorqueTOSettingsResiduals.xml**
6. Open **TorqueTOSettingsResiduals.xml** in a text editor.

7. Inside your `<RCNLConstraintTerm>` for **Residual load reduction**, copy and paste:

```
<time_ranges>0 0.41</time_ranges>
```

    a. This makes the constraint only apply when both feet are in the air. The time range is in units of normalized time (0 to 1)

Section 2: Verification Optimization

The Verification Optimization tool is used to perform a "dry run" Design Optimization without including any "design elements", the goal being to ensure a good initial guess and to verify the appropriateness of the optimal control problem formulation. The tool accepts the same inputs as the TO tool, but in general, the results directory of a TO run is used for both the initial guess and tracked quantities.

It is helpful to include any constraints you might want to use in later Design Optimizations into your Verification Optimization. When doing this, it's important that the constraints should already be satisfied by your Tracking Optimization solution, otherwise the Verification Optimization might struggle to converge.

Creating a Verification Optimization settings file:

1. Activate the NMSM GUI in OpenSim by navigating to *Tools>User Plugins*, and click **rcnlPlugin.dll**
2. With **KickingModel.osim** selected in the OpenSim GUI, go to *Tools>Treatment Optimization >Verification Optimization*
    a. A window should open that looks very similar to the Tracking Optimization window shown in section 1.
3. Leave the *Osimx file* field **empty**.
4. Set the *initial guess directory* to be **TorqueTOResults**
5. Set the *tracked quantities directory* to be **TorqueTOResults**
6. Set the *trial prefix* to be **drive_kick1**
7. Set the *results directory* to be **TorqueVOResults**
8. Set the *optimal control solver settings file* to be **gpopsSettings.xml**.
    a. Remove the file path in this field. It should only say "gpopsSettings.xml"
9. Under *states coordinate list*, select (**hip_flexion_r, knee_angle_r, ankle_angle_r**)
10. Click to the *RCNL Controllers* tab at the top.
11. Under *RCNL Torque Controller*, add (**hip_flexion_r, knee_angle_r, ankle_angle_r**) to the coordinate list.

12. Click to the *Cost/Constraints tab* at the top.
13. **Cost terms:**
    a. Add a new cost term:
        i. Name: **Controller tracking**
        ii. Cost term type: **controller_tracking**
        iii. Component list: (**hip_flexion_r knee_angle_r ankle_angle_r**)
        iv. Max allowable error: **50**
14. **Constraint terms:**
    a. Add a new constraint term:
        i. Name: **Kinetic consistency**
        ii. Constraint term type: **kinetic_consistency**
        iii. Component list: (**hip_flexion_r_moment knee_angle_r_moment ankle_angle_r_moment**)
        iv. Max error: **0.1**
        v. Min error: **-0.1**
    b. Add a new constraint term:
        i. Name: **Initial coordinate position deviation**
        ii. Constraint term type: **initial_generalized_coordinate_deviation**
        iii. Component list: (**hip_flexion_r knee_angle_r ankle_angle_r**)
        iv. Max error: **0.02**
        v. Min error: **-0.02**
    c. Add a new constraint term:
        i. Name: **Initial coordinate speed deviation**
        ii. Constraint term type: **initial_generalized_speed_deviation**
        iii. Component list: (**hip_flexion_r knee_angle_r ankle_angle_r**)
        iv. Max error: **0.2**
        v. Min error: **-0.2**
    d. Add a new constraint term:
        i. Name: **Final toe marker position deviation**
        ii. Constraint term type: **final_marker_position_deviation**
        iii. Component list: (**R_Toe**)

        iv. Max error: **0.001**
        v. Min error: **-0.001**
    e. Add a new constraint term:
        i. Name: **Final calcaneus orientation deviation**
        ii. Constraint term type: **final_body_orientation_deviation**
        iii. Component list: (**calcn_r**)
        iv. Max error: **0.02**
        v. Min error: **-0.02**
    f. Add a new constraint term:
        i. Name: **Limit ankle position bounds**
        ii. Constraint term type: **generalized_coordinate_value**
        iii. Component list: (**ankle_angle_r**)
        iv. Max error: **0.0475**
        v. Min error: **-0.3187**
    g. Add a new constraint term:
        i. Name: **Limit ankle velocity path**
        ii. Constraint term type: **generalized_speed_deviation**
        iii. Component list: (**ankle_angle_r**)
        iv. Max error: **0.2**
        v. Min error: **-0.2**

15. Save this settings file as **TorqueVOSettings.xml**
16. Open **TorqueVOSettings.xml** in a text editor of your choice and explore it. How does this settings file compare to your TO settings file?

Running Verification Optimization:

1. Open MATLAB and open **runVOTool.m** in your tutorial directory.
2. Open the project file (**Project.prj** inside your installation of nmsm-core.)
3. Run the MATLAB file **runVOTool.m**

Post VO Analysis:

1. The script will create the same 4 plots as the TO run.
2. Ideally, the VO solution should be nearly identical to the TO solution. If there are large differences, there is likely a problem with your VO problem formulation.

Section 3: Design Optimization

The Design Optimization tool predicts or optimizes how a planned treatment will affect a patient's post-treatment movement function. The tool accepts the same inputs as the other Treatment Optimization tools, but typically VO results are used as the initial guess. Similar to VO, DO allows for controller tracking and generalized coordinate tracking alongside other DO-specific cost function terms.

Unlike the other Treatment Optimization tools, the DO tool allows for both fixed and free final time problem formulations. For free final time problems, the time vector can be constrained to a user defined range and tracked control quantities are stretched or compressed in time to match the current time vector. In addition to all cost function and constraint terms available in the TO and VO tools, the DO tool includes a number of additional built-in cost function terms.

Three key features of the DO tool are support for model modification functions, user-defined cost function terms, and user-defined constraint terms. By employing model modification functions alongside GPOPS-II's static parameters feature, users can change the OpenSim musculoskeletal model, modify the neural control model, or adjust parameter values in an assistive device.

Creating a Design Optimization settings file:

Now that we have a good "dry run" VO run, we can incorporate some design aspects into our treatment optimization. This will be an iterative process, so first we will create a "base" DO settings file that we will build off of in subsequent runs.

1. Create a copy of **TorqueVOSettings.xml** and name it **TorqueDOSettingsTemp.xml**
    a. **Note**: You need to save this settings file as **TorqueDOSettingsTemp** because there is currently a bug with the GUI that may prevent you from opening a settings file and saving to a settings file with the same name. This way, we can save our final settings file as **TorqueDOSettings.xml** later.

2. Open **TorqueDOSettingsTemp.xml** in a text editor and at the top and bottom of the document, change `<VerificationOptimizationTool>` to `<DesignOptimizationTool>` and save the file.
3. Activate the NMSM GUI in OpenSim by navigating to *Tools>User Plugins*, and click **rcnlPlugin.dll**
4. With **KickingModel.osim** selected in the OpenSim GUI, *Tools>Treatment Optimization >Design Optimization*
    a. A window should open that looks very similar to the Tracking Optimization window shown in section 1.
5. Load your VO settings file **TorqueVOSettings.xml**.
6. Set the *initial guess directory* to be **TorqueVOResults**
7. Set the *tracked quantities directory* to be **TorqueVOResults**
8. Set the *results directory* to be **TorqueDOResults**
9. Because we loaded a previous synergy driven VO settings file, we don't need to add any extra cost or constraint terms here. We will add a user defined term, but that happens outside of the GUI.
10. Save this settings file as **TorqueDOSettings.xml**
11. Open **TorqueDOSettings.xml** in a text editor and explore it.
12. Design optimization allows for *free final time* problems, where the final time of the problem can change. This is useful for problems like this where speed is being changed because the new kicking motion will end up being shorter or longer than the original motion.
13. To enable free final time, underneath the `<trial_name>` field, copy and paste:

```
<final_time_range>0.26 0.36</final_time_range>
```

14. Increasing the final foot velocity is a great use of **user defined cost terms**.
15. A user defined cost term is premade in the tutorial directory named **footSpeedCost.m**
    a. This cost term minimizes deviations away from a desired final velocity.
16. **Note: If you are using user defined cost terms, the NMSM GUI in OpenSim is not recommended. If you load a settings file with a user defined cost term into the GUI,**

and then save a new settings file, the user defined cost term will not be saved properly.

17. To include this cost term in the settings file, copy the following lines into your `<RCNLCostTermSet>`

```
<RCNLCostTerm name="User defined">
   <type>user_defined</type>
   <function_name>footSpeedCost</function_name>
   <cost_term_type>discrete</cost_term_type>
   <is_enabled>true</is_enabled>
   <marker_name>R_Toe</marker_name>
   <target_speed>14.35</target_speed>
</RCNLCostTerm>
```

18. Open MATLAB and open **runDOTool.m** in your tutorial directory.
19. Open the project file (**Project.prj** inside your installation of nmsm-core.)
20. Run the MATLAB file **runDOTool.m**.
21. **Add alternate formulations**

Post DO Analysis:

1. Similar to the previous treatment optimization tools, we can generate a set of design optimization results plots.
2. The script will automatically generate plots of our DO results compared to our previous VO results.

Alternative DO Formulations:

There are multiple different ways to increase kicking speed in a Design Optimization without using user defined cost terms. Other methods presented in this section are to use built in constraint terms, and user defined constraint terms.

To increase kicking speed using built-in constraint terms, you can add a constraint term on the velocity of a foot marker such that the foot will speed up. Make sure when doing this, you only apply the constraint to the marker speed in the x-axis, otherwise the optimization will not be able to converge.

Alternatively, you can create your own constraints using our framework for user defined constraint terms. We have a pre-made user defined constraint term in the file **footSpeedConstraint.m.** To use this constraint term, copy and paste the following code into your `<RCNLConstraintTermSet>`

```
<RCNLConstraintTerm name="Toe marker final velocity">
       <is_enabled>true</is_enabled>
       <!--Type of constraint term.-->
       <type>user_defined</type>
       <!--Type of application of constraint term (path or terminal).-->
       <constraint_term_type>terminal</constraint_term_type>
       <!--Name of Matlab function defining constraint term.-->
       <function_name>footSpeedConstraint</function_name>
       <marker_name>R_Toe</marker_name>
       <!--Maximum error allowed by this constraint term.-->
       <max_error>14.4</max_error>
       <!--Minimum error allowed by this constraint term.-->
       <min_error>14.3</min_error>
</RCNLConstraintTerm>
```