Tutorial: “In the wild” movement analysis using dynamic simulations

Section 0: Getting the code to run

Install the BioMAC-Sim-Toolbox following the instructions here: https://github.com/mad-lab-fau/BioMAC-Sim-Toolbox. Fork and clone (recommended) or download and unzip this tutorial from: <ADD LINK>.

After opening MATLAB, navigate to the Toolbox in the *Current* Folder and then add the folder *src* to the path, with all its subfolders. Then, navigate back to the Tutorial folder. The tutorial is based on the script *scriptTutorial.m*. In this script, you are supposed to add code for different tasks.

>> TODO 1: Add the path to the location of the BioMAC-Sim-Toolbox in line 24

scriptTutorial.m contains all the required scripts, but it relies on other functions that define the problems to be solved. We use the function *standing2D* to define our standing initial guess (not covered in the tutorial), and *move2D_IMU* to define our movement problems. You need to first find a good standing solution, in which the model is standing on flat feet.

>> TODO 2: (re)run the code up to line 73 until you get a solution standing on flat feet

You might get an error if you have unzipped instead of cloned the git folder. A solution is provided at the end of this PDF.

You will create different problems by varying the input to the function. In line 27 of *move2D_IMU*, you can see that there are 4 fixed inputs and a variable one. The fixed inputs are the model (always the same in the tutorial), the name of the result file (variable in the tutorial), the data that is tracked (always the same in the tutorial), and the initial guess (variable in the tutorial). The variable input can be left empty, which means that the default settings are used, as shown in line 29-33. Otherwise, it can be a struct defining the objective weights (line 37) or a double to define the number of nodes (line 45).

In the tutorial, we aim to create simulations of walking by tracking IMU signals of seven sensors located on the lower body and minimizing effort. We also always add a regularization term to promote smoothness of the movement, which helps the optimization algorithm. The data that we are tracking come from [1] and contain the mean and variance of a single participant, who walked 10 times through a capture volume at normal speed. The data consists of gyroscope signals, accelerometer signals, joint angles, joint moments and ground reaction forces, as well as speed and duration of the gait cycle. The full dataset can be found here: https://doi.org/10.5281/zenodo.11522050.

Section 1: Getting familiar with the result object

Our first step is to get familiar with the output of our code, to get to know where to find what information. We use the example result object in the file Section1Example.mat. Based on the file, you should be able to figure out exactly how this simulation was created. In this section, we will go through how you can find the information in the result object. You can either use the Variables (recommended) or the Command Window in MATLAB.

>> TODO 3: load the result Section1Example.mat and open the result object

You will see 12 properties in the object. The first three properties of the object are related to the simulation and its creation, the next two are related to the output of the optimization algorithm and the rest saves information about where and how the result was created, such as the time, the filename, the name of the computer, and the related git folder.

First, let's take a look at the properties related to the optimization algorithm, IPOPT. The related properties are "converged" and "info".

- Q1. What does it mean when converged is equal to 1? Hint: use the toolbox documentation
- Q2. What was the final objective value?
- Q3. How long did the optimization take?

Now, let's go into the simulation. The related properties are *problem*, *solver*, and *X*. *problem* saves the optimization problem that was solved. *solver* describes the solver that was used and summarizes the important settings. We normally use default values but make the tolerances tighter for standing simulations. *X* contains the optimal value for all optimization variables.

X is a very long vector, so it can be challenging to figure out which variable means what. To analyse simulation results, we use the *report* function, which outputs the relevant biomechanical variables in a table based on *X*. However, it is important to understand *X* when you want to use or set up your own objective or constraint functions. To do so, we should go into *problem*. There, the fifth property is called *idx*. Here, you find an overview of where you can find what information of *X*.

- Q4. What variables are part of the optimization variables in this problem?

Besides *idx*, *problem* contains many other relevant properties, such as the discretization method (property *Euler*), the number of nodes, the initial guess, the lower and upper bounds of *X* as well as the constraints, and the objective and constraint terms that were added, among others. The required properties are the upper and lower bounds, the initial guess, and the property *name*, which states which function has been used to create the problem.

Two important properties are *constraintTerms* and *objectiveTerms*, which define optimization's constraints and objective. The first column of *constraintTerms* shows the name, the second their values, the third the value history, and the last column shows any variable inputs. The first column of *objectiveTerms* also shows the name, the second their weights, the third their values, the fourth the value history, and the last column again shows any variable inputs. These variable inputs can be, for example, the data that was tracked, or any other additional information that is required. In case of the periodicity constraint, this is an input as to whether we assume symmetry or not. For the effort objective, we add the power, and whether or not the muscles are weighed by their volume.

- Q5. What was the full objective that was solved for this problem?
- Q6. Did we assume left-right symmetry in the problem? Hint: you can find this in the problem itself and in the constraintTerms.

From the model object, we can still look into *X* in more detail. From *idx*, we know what the optimization variables are and the order in which they are defined. This order can be different, it is not required to first define the states. The *states* and *controls* properties in *idx* are both matrices of indices with the number of rows equal to the number of states (66) or controls (16) and the columns equal to the number of time nodes plus an extra one for the periodicity constraint (101). We also need to know what these states and controls are. To find this out, we go to *model*, which is a property of *problem*. The model object contains, among others, the properties *states* and *controls*, which show information about the model states and controls. For the controls, you can see that we use stimulation *u* as input, which has a minimum of 0 and a maximum of 5. Note that we often do not constrain the muscle simulation at 100%, because we are solving minimum effort simulations, meaning that they should automatically be as low as possible.

- Q7. What are the different states in the model state?
- Q8. How many degrees of freedom does the skeletal model have?

More information about the states, so the degrees of freedom, muscles, and contact points can be found in the properties *dofs*, *joints*, *muscles*, and *CPs*. Furthermore, the property *torques* can include additional torques that are used to propel the system. We use for example that in 3D full-body movements, where the muscles for the upper limbs are not defined. Torques can also replace muscles entirely in a skeletal model.

Section 2: Running and comparing simulations

In this section, we will focus on different choices that you can make when creating the simulations and compare simulations to investigate their effect. Specifically, we will run simulations with a different weighting (lines 101-140), a different initial guess (lines 142-178), and different number of nodes (lines 180-224). All simulations that are used are also provided in the folder Results. The comments at the beginning of each section describe the respective names. Especially a large number of nodes will make the optimization rather slow and is likely not feasible during the tutorial.

Objective weighting

First, we will investigate different weightings. In *move2D_IMU*, you can see the standard objective weights in lines 30-32. The objectives are added in lines 85-91. Setting objective weights is often more of an art than a science. In our case, we have three objectives, which means that we need to set two objective weights with respect to a reference one. The value of the weights matter, since IPOPT, the algorithm that we use to solve the optimizations, likes to have values between around 0.1 and 1000. Higher or lower weights can make the optimization slower, so keep that in mind. The regularization weight is normally set to be 1000x smaller than the tracking weight. Setting the ratio between the tracking and effort weight is a bit trickier and depends on the type of data and simulation purpose. Now, we will play with these weights so that you can see the effect. Given the tracking weight of 1, we will use effort weights of 1, 50, 100, 600 (base value), 5000, and 10,000. For now, you can run optimizations with two of these weights and use the provided solutions for the other simulations. Note that the effort weight of 1 is rather slow.

>> TODO 4: copy the results that you do not want to run to your result folder. See line 30 in *scriptTutorial* to find the result folder.

>> TODO 5-8: set up the simulations in lines 105, 107, 112, and 123

To find a good weighting, we look at different variables. The ground reaction forces should not have large impact peaks. The activations should be smooth. The tracked signals should be tracked well. The code will plot these variables, as well as the joint angles and joint moments. The legend is plotted in a separate figure. It is not possible to automatically add a description to a legend, but the highest line corresponds to the first item in the simVarTables cell, and the lowest to the last one.

- Q9. Would you discard a solution as unrealistic when looking at the ground reaction forces and activations?
- Q10. How much difference do you see in the joint angles and joint moments of the acceptable solutions?

Initial guess

Second, we will investigate the initial guess. So far, we have used a standing initial guess. However, let's try using a previous simulation, as well as the midpoint between the upper and lower bound. You can find how to use the midpoint in the function *makeinitialguess* (from line 505 in \src\problem\@Collocation\Collocation.m or its documentation. Since we have just solved walking with different effort weightings, we will use one of those as the previous simulation, in this case the one with an effort weight of 5000, as well as the midpoint.

>> TODO 9-10: set up the simulations in lines 147 and 159

- Q11: Do you see any differences between the three simulations?

Number of Nodes

Third, let's take a look at the number of nodes. We will run a convergence analysis and investigate the effect of the number of nodes on the metabolic cost. We vary the nodes between 5, 25, 50, 75, 100, 250, and 500. It will take a long time to run the simulations with many nodes. To speed up, we are using the previous simulations as initial guess, but you can also again use the standing solution. For now, do not run more than 50 nodes (or even less depending on the time).

>> TODO 11: set up the simulations in line 194

>> TODO 12: create a convergence plot showing the metabolic cost as a function of the number of nodes in line 223

- Q12. What does the convergence plot tell you about a suitable number of nodes?
- Q13: Based on the joint angles and joint moments, from which number of nodes do you not see a clear difference between the different simulations anymore?

Note that due to the collocation scheme that we are using, the metabolic cost estimations will still decrease. Backward Euler is known to introduce damping and thus remove energy, and this effect is reduced with a larger number of nodes. It can be solved by using a different, energy-conserving method, so-called symplectic integrators like semi-implicit Euler. When making comparisons between two simulations, it is important that the number of nodes (so the time step) is consistent!

Section 3: Create your own constraint function

A very important aspect when setting up an optimization is to define new objectives and constraints. When doing so, I advise to use an existing similar one as template and adjust it. Furthermore, it is imperative to ensure that the derivative (gradient for objective and Jacobian for constraints) matches the actual function, otherwise the optimization algorithm will not be able to find the correct solution.

This small example, adding a constraint to track speed, aims to cover several vital topics that will help create new objectives and constraint functions. This constraint function should ensure that the difference between the actual and target speed is equal to 0. *speedConstraint_IMU* is located in the tutorial folder as a template, but the lines defining the constraint and its Jacobian need to be added. Currently, we use strict bounds when creating the optimization variable for speed in line 83 of *move2D_IMU*.

>> TODO optional 1: give the function *move2D_IMU* a new name in line 27 and save it accordingly with the new name.

>> TODO optional 2: comment line 83 and uncomment line 82 in the new function to remove the strict bounds on the optimization variable

>> TODO optional 3-4: add the output and Jacobian according to the comments in line 30 and 33 of *speedConstraint_IMU*.

To be able to use the constraint function, it should be located in the folder \src\Problem\@Collocation\, where all functions related to *Collocation* objects should be stored.

>> TODO optional 5: move this function from the tutorial folder into the *@Collocation* folder

It might be that you need to update the path. You can do this by first removing the folder *problem* and its subfolders from the path (right-click on *problem* and select "Remove from Path" and then "Selected Folders and Subfolders") and then again adding it to the path (right-click on *problem* and select "Add to Path" and again "Selected Folders and Subfolders").

>> TODO optional 6: ensure that the file is added to the path

Next, we need to add this constraint to the new version of the function *move2D_IMU* to use it in the optimization. This requires us to add one line of code to constrain the speed, which will be similar to line 94 and 95, using the function *addConstraint* (from line 464 in \src\problem\@Collocation\Collocation.m or its documentation). This function has at least three inputs, the first being a handle to the constraint, the second being the lower bound, the third the upper bound, and then any other inputs to the function. In this case, the fourth would be the *targetSpeed* that should be achieved, which is defined in line 62.

>> TODO optional 7: add the code in line 96 of the new *move2D_IMU* to ensure that *speedConstraint_IMU* is used.

Before running the optimization, we will check if the Jacobian is correct using the function *derivativetest* in *Problem* (starting on line 65 of \src\Problem\Problem.m). This function uses finite differences (a slow, numerical approach) and compare the output of the finite difference method to the output from the (analytical) functions, which are much faster.

>> TODO optional 8: create a problem with a small number of nodes, e.g., 4, to do the derivative test in line 234 of *scriptTutorial*.

>> TODO optional 9. Add code to run the derivative test function in line 235

This is an example output of the derivative test, generated from a random input.

```
Checking derivatives: 99.4% done...
Checking gradient...
Max. difference: 100.2565627098 at 8283 1 (-506490920.0087790489 vs. -506490819.7522163391)
Checking Jacobian...
Max. difference:   0.0026012746 at 3166 3232 (107.4744429296 vs. 107.4718416549)
```

It is important to check that the maximum difference is small with respect to the value of the numbers in the brackets. In this case, the largest difference in the gradient is 100, which seems large, but is fine given the size of the numbers in the brackets with 9 digits.

If you have such a large difference, it makes sense to run the test a couple of times, because there might be a smaller structural error somewhere else (e.g., any nonzero where there should be a zero or vice versa). The numbers are large because we use a random input. It is also possible to define an input used for the derivative test and avoid large numbers that way.

- Q14: Do you think that the constraint function is correct?

As soon as you have verified that your code is correct, you can comment/delete the lines to test the derivatives.

>> TODO optional 10. Run the new optimization

- Q15: Do you get a different result than when the speed is set using the bounds?

Unzipping problem

If you download and unzip the tutorial, it might happen that you will receive an error related to `"gait2dc_dyn_al_raw.c" doesn't exist`. This error happens because we check if changes are made to the model by comparing the date and time when the functions are created. Depending on the order of unzipping, it might therefore be that the model file is created slightly after, triggering this check. If that happens, please change line 145 in \src\model\gait2dc\@Gait2dc to the following. This allows there to be a small difference in the timing between the functions.

```
if mdate(dependencies{i}) - mdate(file) > 1e-5 * 5
```

References

1. Dorschky, E., Nitschke, M., Seifer, A. K., Van Den Bogert, A. J., & Eskofier, B. M. (2019). Estimation of gait kinematics and kinetics from inertial sensor data using optimal control of musculoskeletal models. *Journal of biomechanics*, *95*, 109278.