# Introduction to Pumas-QSP

Pumas for Quantitative Pharmacology Workshop - December 5 and December 6

## Workshop description

---

In this workshop, we show you how to use PumasQSP on JuliaHub to

- generate a Virtual Population (VP) trained on multiple clinical conditions
- subsample from a VP
- use this VP to test it's behaviour in new virtual, clinical conditions
- and perform a Global Sensitivity Analysis (GSA) of the model
- parallelize VP generation over multiple nodes.

---

## Starting point and material

---

We start with the clinical data provided in CSV files, the model description given in MATLAB, and the meta information in a .jl file.

- State of data: three clinical trial data sets stored in CSV files.
- State of model: their model is coded in MATLAB.
- State of meta data: the setup of the clinical conditions is described in a .jl file.

Along side this PDF you should have received the following material in the ZIP folder "Workshop material". This contains

- the "model.m" MATLAB file: the model description in MATLAB
- the "data" folder : here is the experimental data of three clinical conditions given in 3 csv files
  - "data1.csv"
  - "data2.csv"
  - "data3.csv"
- a MetaData.jl file,
- an empty analysis folder,
- an empty visualization folder.

Whilst working though the exercises of this tutorial, we will create files labeled "exercise_X.jl" in the analysis folder and visualize results in the visualization folder.

---

## Exercise 1: Data exploration

1. Look at the data. Open the CSV files (data1.csv, data2.csv, data3.csv). How many states are observed? What do the time dimensions look like? Are there missing values or other abnormalities you should be aware of?

2. Visualize the data. For this, first read the CSV file into a DataFrame in your Julia environment with the DataFrames.jl package and then use the package Plots.jl to visualize the data.

- *If you need help in this step, have a look at the documentation of the DataFrames.jl and Plots.jl package.*
- *Helpful functions/constructors in this step are: CSV.read, Matrix, scatter*

With some formatting, the result should look similar to this:

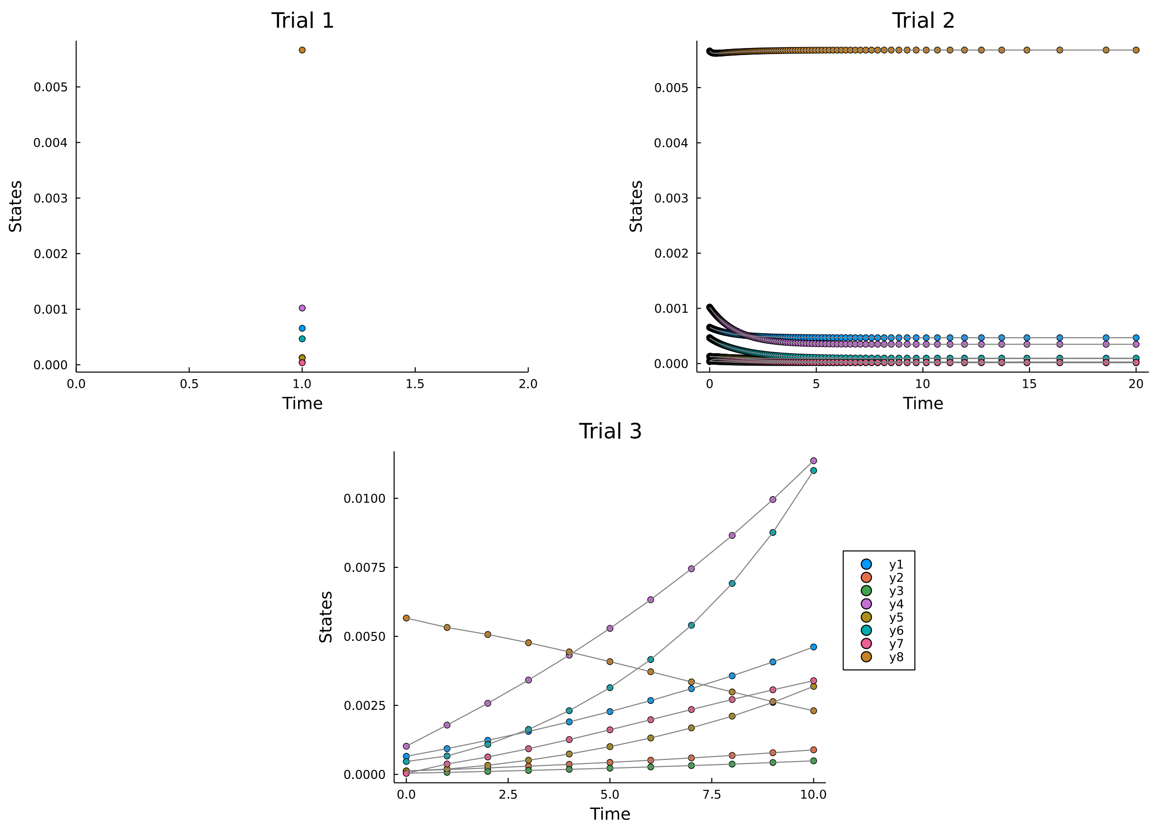

Figure 1: Expected outcome of exercise 1.

## Exercise 2: Model exploration

1. Have a look through the MATLAB code describing the model (model.m) and recognise the states from the data. Now translate the MATLAB model to Julia by formulating the model as a ODESystem of ModelingToolkit.jl. For this, you will need

   - the states as @variables
   - the parameters as @parameters
   - the equations via a Differential
   - default values for the initial conditions for all states and parameters:
     y1 = 1.0, y2 = 0.0, y3 = 0.0, y4 = 0.0, y5 = 0.0, y6 = 0.0, y7 = 0.0, y8 = 0.0057,
     k1 = 1.71, k2 = 280.0, k3 = 8.32, k4 = 0.69, k5 = 0.3, k6 = 1.81

2. After you defined your model as an ODESystem, we want to do a test simulation of the model over time. For this, we need to convert it to an ODEProblem of the package DifferentialEquations.jl. Here we will need some additional information:

   - For the ODEProblem, a simulation time span:
     (0.0, 10.0)
   - For the solve, the solver specifications:
     For steady state: DynamicSS(QNDF()), for ODE: Rosenbrock23()

3. Use again the Plots.jl package to visualize the simulation results.

- *If you need help in this step, have a look at the documentation of the ModelingToolkit.jl and DifferentialEquations.jl package.*
- *Helpful constructors/functions in this step are: ODESystem, ODEProblem, solve, plot*

With some formatting, the result should look similar to this:

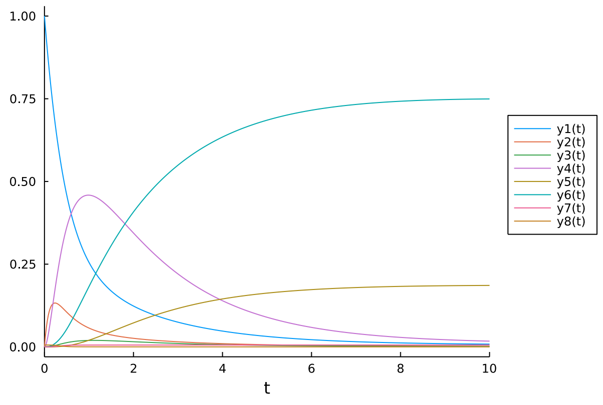

Figure 2: Expected outcome of exercise 2.

## Exercise 3: Construction of Trial Structure

1. Now we want to construct Trial objects which represent fundamental roles in the API of PumasQSP. For each observed clinical condition, we combine the observed data and the associated model configuration. For the first case (data_1.csv), we use the SteadyStateTrial object of PumasQSP. For case two and three (data_2.csv, data_3.csv), we use the Trial object of PumasQSP. We want to construct the trial 2 and 3 in a way that they are based on trial 1, use the "forward_u0" for this. The associated model configuration need for this step are

   - Trial 1: SteadyStateTrial with parameters changed as k6 = 20.0
   - Trial 2: Trial with parameters changed as k3 = 4.0, k4 = 0.4 and k6 = 8.0, and a timespan of (0.0, 20.0)
   - Trial 3: Trial with parameters changed as k3 = 15.0, k4 = 0.6 and k6 = 2.0 and a timespan of (0.0, 10.0)

2. After you have created the three objects for the three given clinical conditions, combine them in a SteadyStateTrials object.

- *If you need help in this step, have a look at the documentation of the PumasQSP.jl package.*
- *Helpful constructors/functions in this step are: Trial, SteadyStateTrial, SteadyStateTrialCollection*

## Exercise 4: Generation of a Virtual Population

1. Define your InverseProblem object which depends on the SteadyStateTrialCollection and the model. Additionally, it requires the search_space which has not been defined yet. Firstly, this variable sets the parameters which are varied in order to create the diversity of the population to be created. Secondly, it also defines the ranges of the parameters in which new virtual patients are created. Use the following parameters and ranges
   - search_space for patients as k1 = (0.0, 5.0), k2 = (200.0, 300.0) and y1 = (0.0, 3.0).
2. Once the problem is defined, generate a Virtual Population of size 10 using PumasQSP.jl. Additional specifications are required which are
   - the optimizer and a set number of maximal iterations StochGlobalOpt(maxiters = 750)
   - the number of patients you want to generate, set this to 10 for the beginning.
3. Visualize the virtual population with Plots.jl for the clinical condition two and three together with the respective training data using the keyword argument "show_data".
4. Save your VP in a CSV file.

- *If you need help in this step, have a look at the documentation of the PumasQSP.jl package.*
- *Helpful constructors/functions in this step are: InverseProblem, vpop, plot, CSV.write*

With some formatting, the result should look similar to this:

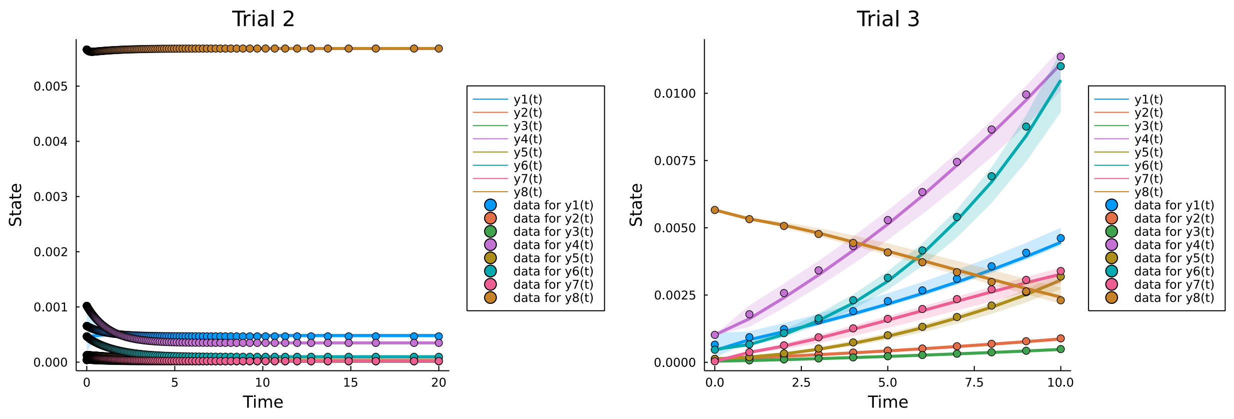

Figure 3: Expected outcome of exercise 4.

## Exercise 5: Subsampling from a Virtual Population

1. Remind yourself of trial 3 by plotting the trial for the whole virtual population.
2. Perform a subsampling of the population based of constraints for states four and six using the Discretized Density Sampling (DDS) algorithm.
   - dist4 = TransformedBeta(Beta = Beta(2, 2), lb = 0.011, ub = 0.013)
   - dist6 = TransformedBeta(Beta = Beta(2, 5), lb = 0.01, ub = 0.015)
   - n = 100
   - nbins = 20
3. Visualize your results.

- *If you need help in this step, have a look at the documentation of the PumasQSP.jl package.*
- *Helpful constructors/functions in this step are: Beta, TransformedBeta, subsample, DSS*

With some formatting, the result should look similar to this:

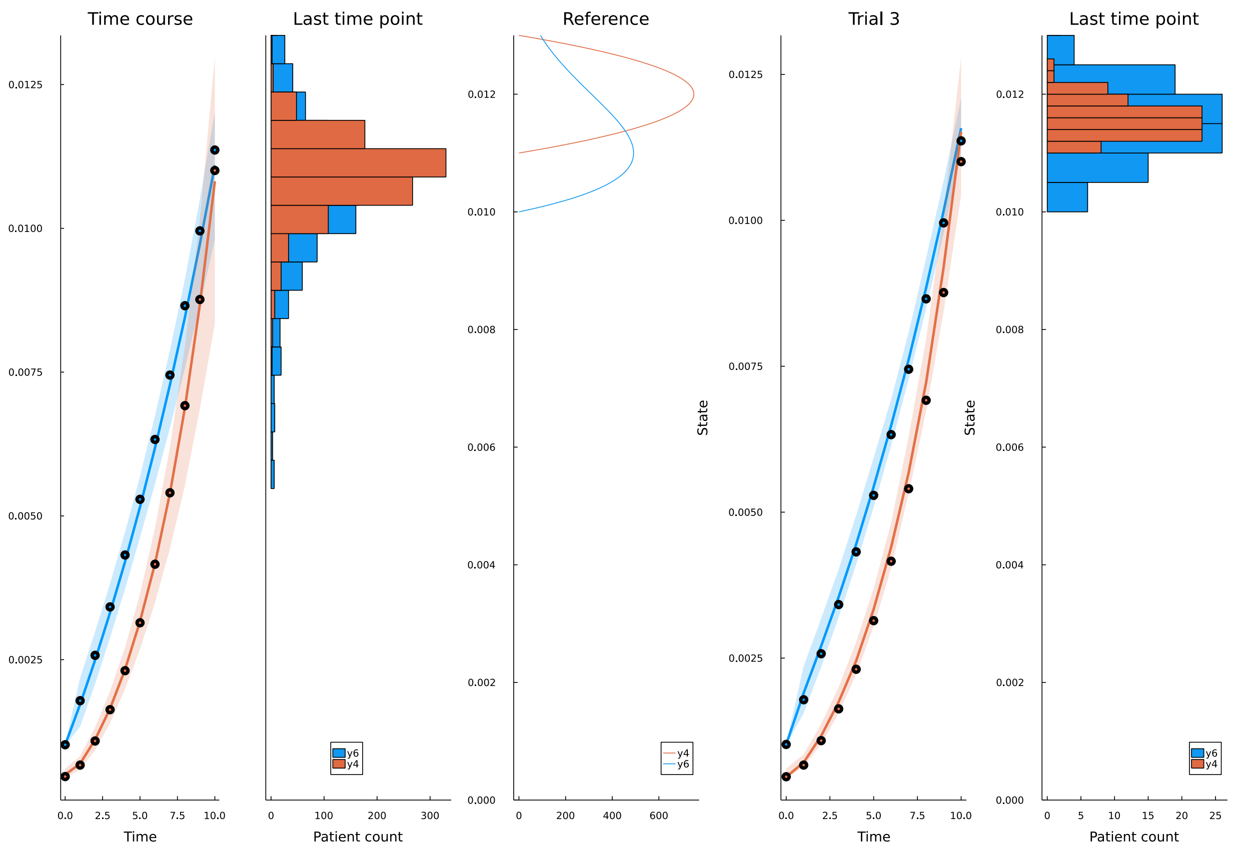

Figure 4: Expected outcome of exercise 5.

## Exercise 6: Exploration of New Clinical Conditions

1. Define a new clinical condition based on trial 3, via the virtual_trial object where the parameters k3 and k4 are changed.
    - Change them to k3 = 1.0 and k4 = 1.7.
2. Again, visualize the behaviour of the virtual population for this new virtual_trial.

- *If you need help in this step, have a look at the documentation of the PumasQSP.jl package.*
- *Helpful constructors/functions in this step are: virtual_trial, plot*

With some formatting, the result should look similar to this:

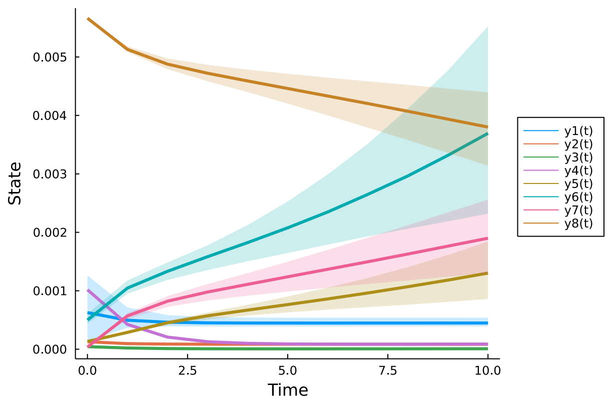

Figure 5: Expected outcome of exercise 6.

## Exercise 7: Global Sensitivity Analysis

1. Generate a SensitivityProblem object with a parameter space of k1 = (0.0, 5.0), k2 = (200, 300), and k3 = (5., 10.).
2. Perform a global sensitivity analysis (GSA) with a GSA method of your choice, e.g. Morris or Sobol.

- *If you need help in this step, have a look at the documentation of the PumasQSP.jl package.*
- *Helpful constructors/functions in this step are: SensitivityProblem, gsa.*

With some formatting, the result should look similar to this:

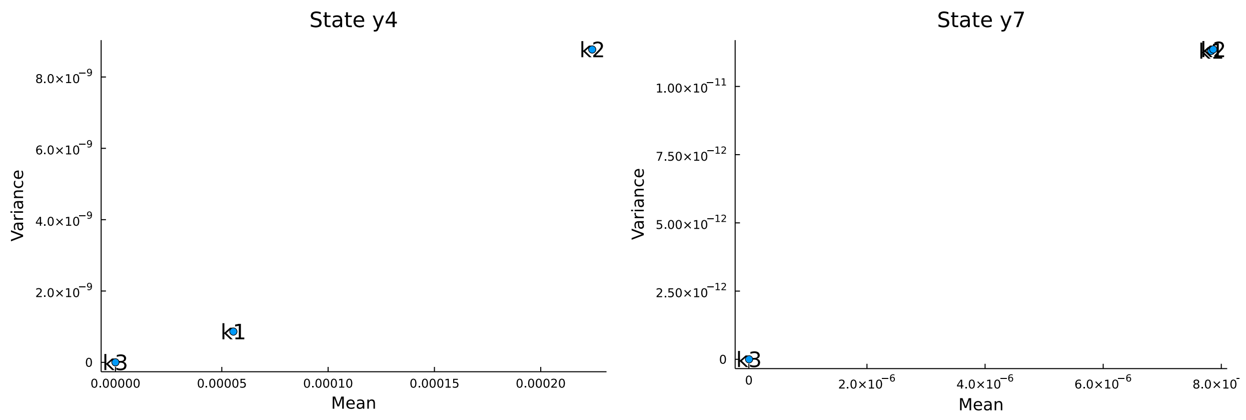

Figure 6: Expected outcome of exercise 7.

## Exercise 8: Polish and test your code thoroughly

1. Restructure and organize your code in a manner that makes rigorous testing easy. One option is to write your own tests using the Test.jl package.
    - Test the size/length of the read in data.
    - Test if it matches the simulation of trials.
    - Test the size of the VP.
    - Test the quality of the VP.
2. The other option is to adapt your code to a package structure, write similar tests and run the automatically when updating code in you version control system.

- *If you need help in this step, have a look at the documentation of the Test.jl package.*
- *Helpful macros in this step are: @test, @testset.*

## Exercise 9: Running code over multiple nodes on JuliaHub

So far, we used the interactive mode on JuliaHub. Now we want to submit parallelized jobs directly to JuliaHub.

1. Run the same VP generation over multiple nodes.
2. Scale the VP generation to 10.000 patients.

- *If you need help in this step, have a look at the documentation of the Distributed.jl package.*
- *Helpful macros/functions in this step are: @everywhere, EnsembleDistributed*

## Further resources

Helpful links to more material on Pumas-QSP

1. Pumas-QSP Website: `https://about.juliahub.com/products/pumas-qsp/`
2. Pumas-QSP Documentation: `https://help.juliahub.com/pumasqsp/dev/`
3. Pumas-QSP Webinar: `https://www.youtube.com/watch?v=K5cWJg4mh60`
4. Pumas-QSP Introduction course: `https://labs.pumas.ai/courses/pumas-QSP`

*Contact:* Elisabeth Roesch, PhD (Pumas-QSP Sales engineer) – email: elisabeth.roesch@juliahub.com