An Open Source Package Suite in Julia to Facilitate QSP Modeling and Simulation

Timothy Knab[1], Eric Jordie[1], Matthew Riggs[1] Ahmed Elmokadem[1]
[1]*Metrum Research Group, Tariffville, CT, USA*

Introduction

Objectives: Systems of ordinary differential equations are frequently used to mathematically characterize complex systems and are the foundation of quantitative systems pharmacology modeling (QSP). Simulations using these models often involve multiple parameter sets and initial conditions, while simultaneously incorporating complex dosing schemes and covariate information, which can be difficult and time-consuming to account for with every new model. Furthermore, it may be beneficial to develop a system whereby parameter, state, and dosing information is incorporated into a single model object. Here, we introduce a pair of packages ***PMParameterized.jl*** which simplifies specification of parameters and initial conditions for Ordinary Differential Equation models, and ***PMSimulator.jl*** which enables complex dosing, inputs and events. Together these packages result in stateful, reproducible models and simulation for a quantitative systems pharmacology and pharmacometrics workflow in the Julia language.
Methods: A domain specific language (DSL) for QSP and pharmacometrics (PM) was developed utilizing acausal and symbolic modeling functionality from ModelingToolkit[1] and the SciML ecosystem and exposed via PMParameterized.jl Solvers from DifferentialEquations.jl[2] and SciMLSensitivity.jl[3] were extended in PMSimulator.jl to solve ordinary differential equation and sensitivity problems defined with models from PMParameterized.jl. Complex dosing events defined in NONMEM®-like datasets as well as events constructed from tools made available in PMSimulator.jl are supported.

PMParameterized

Specify Model

```
mdl = @model mod begin
    @IVs t [unit = u"hr", description = "time", tspan = (0.0, 100.0)]
    D = Differential(t)
    @constants day_to_h = 24.0, [unit=u"hr/d",
                                 description = "Convert days to hours"]
    @parameters begin
        (CL_ADC = 0.0043), [unit = u"L/d",
                            description = "central clearance"]
        CLD_ADC = 0.014, [unit = u"L/d",
                          description = "intercompartmental clearance"]
        V1_ADC = 0.034,       [unit = u"L",
                               description = "central volume"]
        V2_ADC = 0.04,        [unit = u"L",
                               description = "peripheral volume"]
    end
    CL_ADC = CL_ADC/day_to_h
    CLD_ADC = CLD_ADC/day_to_h
    @variables begin
        (X1_ADC(t) = 0.0), [unit = u"nmol",
                            description = "ADC amount in Compartment 1"
        (X2_ADC(t) = 0.0), [unit = u"nmol",
                            description = "ADC amount in Compartment 2"
    end
    @observed begin
        C_X1 = X1_ADC/V1_ADC
        C_X2 = X2_ADC/V2_ADC
    end
    @eq D(X1_ADC) ~ -(CL_ADC/V1_ADC)*X1_ADC -
                     (CLD_ADC/V1_ADC)*X1_ADC +
                     (CLD_ADC/V2_ADC)*X2_ADC

    @eq D(X2_ADC) ~ (CLD_ADC/V1_ADC)*X1_ADC -
                     (CLD_ADC/V2_ADC)*X2_ADC
end;
```

Aims

- Simple and intuitive updates of constants, parameters, and variables
- Stateful i.e changed values persist for lifetime of model
- Fully self-contained
- Direct access to states and observed values in solution

Direct Update of Parameters and Initial Conditions

```
# Define simulation conditions
MW = 148781.0       # [g/mol] T-DM1 molecular weight
BW = 70             # [kg] human body weight
dose_in_mgkg = 3.6  # mg/kg
dose = (dose_in_mgkg * 1e-3 * BW) / (MW / 1e9)  # nmol

# Set initial condition in first compartment
mdl.states.X1_ADC_nmol = dose
mdl.parameters.V1_ADC = 0.05

# Update simulation tspan
mdl.tspan = (0.0, 21.0*24.0);
```

Query Model

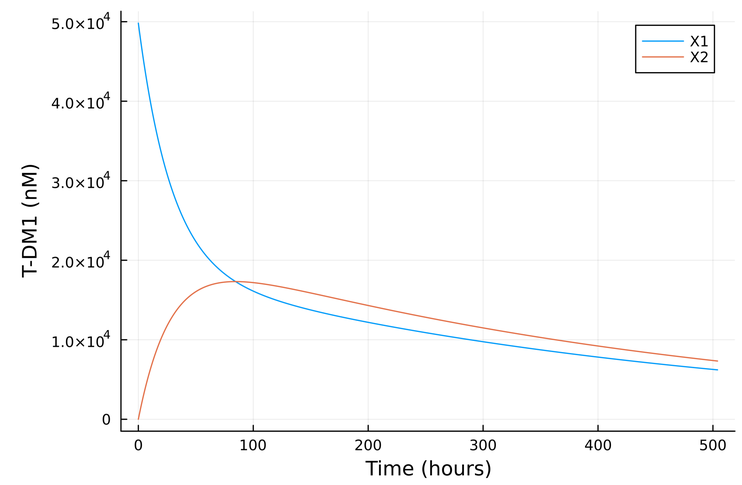
```
detDefault(mdl.parameters, :CL_ADC)
0.0043
mdl.parameters.CL_ADC
0.05
getUnit(mdl.states, :X1_ADC_nmol)
L d⁻¹
getDescription(mdl.states, :V1_ADC)
"central volume"
```

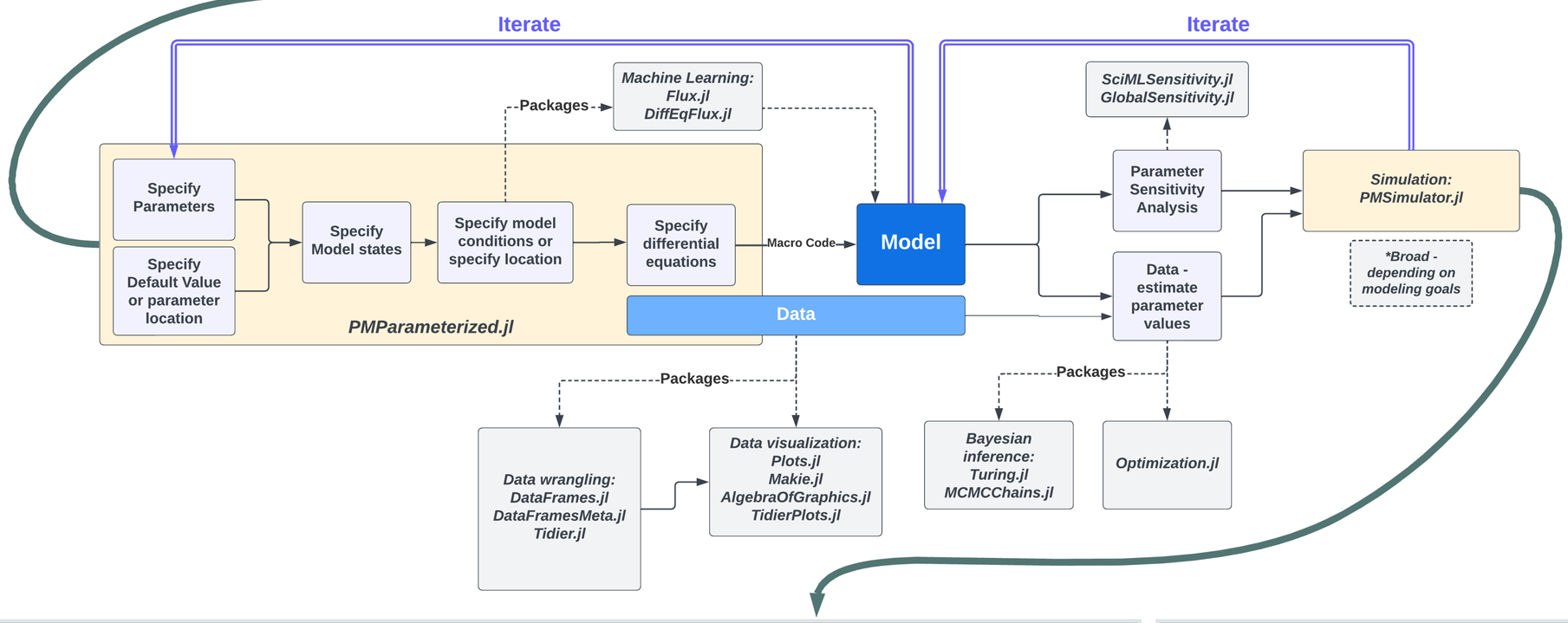
Simulate

```
using DifferentialEquations
sol = solve(mdl, AutoTsit5(Rosenbrock23), saveat=0.5)

plot(sol.t, sol.C_X1, label = "X1",
     xlabel="Time (hours)", ylabel="T-DM1 (nM)")

plot!(sol.t, sol.C_X2, label = "X2")
```

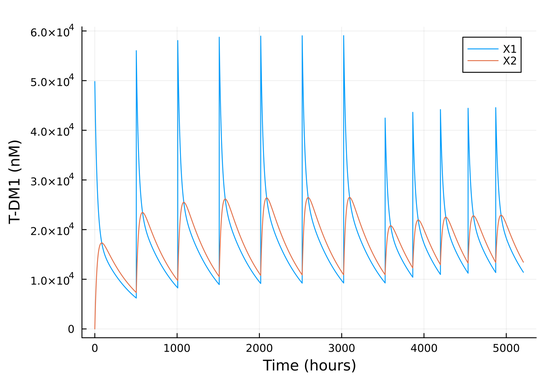




PMSimulator

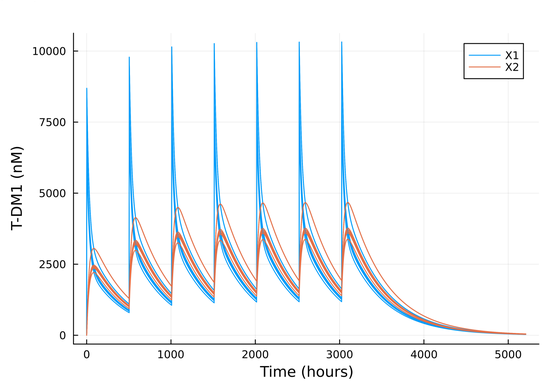
Dosing and Update Events

```
# Function to Convert dose in mg/kg to nmol
mgkg_to_nmol(dose_mgkg, BW, MW)=(dose_mgkg * 1e-3 * BW) / (MW / 1e9)
# Create Input Events at t = 0 and t = 3528
ev1 = PMInput(time = 0.0, amt = mgkg_to_nmol(3.6, BW, MW),
              input = :X1_ADC, ii = 504, addl = 6);
ev2 = PMInput(time = 7*24*21, amt = mgkg_to_nmol(2.4, BW, MW),
              input = :X1_ADC, ii = 336, addl = 4);
# Combine events into vector
evs = [ev1, ev2]
# Solve model with events
sol = solve(mdl,evs,AutoTsit5(Rosenbrock23()),saveat=1.0);
# Plot the solution
plot(sol.t, sol.C1_X1, label="X1",
       xlabel="Time (hours)", ylabel="T-DM1 (nmol)")
plot!(sol.t, sol.C_X2, label="X2")
```



NONMEM®-Like Data and Populations

```
#Use an example dataset
data = DataFrame(CSV.File("tdm1.csv"))
# Modify to set infusion compartment
data = @chain data begin
    @mutate(input = ifelse(CMT == 1, :X1_ADC, :nothing))
end;
# Solve using NONMEM-like dataframe
sol = solve(mdl,data,AutoTsit5(Rosenbrock23()),saveat=1.0);
# Plot results
plt = plot(xlabel="Time (hours)", ylabel="T-DM1 (nmol)", dpi=600)
[plot!(sol[i].t, sol[i].C_X1,color=get_color_palette(:auto,17)[1],
       label=i ==1 ? "X1" : nothing) for i in keys(sol)];
[plot!(sol[i].t, sol[i].C_X2,color=get_color_palette(:auto,17)[2],
       label=label=i==1 ? "X2" : nothing) for i in keys(sol)];
display(plt)
```



Summary

PMParameterized.jl

- Developed a DSL based on ModelingToolkit.jl[1] that builds stateful models
- Simplifies specification and update of parameters and initial conditions
- Supports local sensitivity analysis, structural identifiability

PMSimulator.jl

- Supports events such as bolus or infusion delivery of exogenous inputs
  - single or multiple IDs
- Event interface: `PMInput` , `PMUpdate`
- Automatic conversion of NONMEM®-like datasets to events

See Our Other Posters

References

[1] Ma, Y., Gowda, S., Anantharaman, R., Laughman, C., Shah, V. and Rackauckas, C. ModelingToolkit: A Composable Graph Transformation System For Equation-Based Modeling (2021).

[2] Rackauckas, C. and Nie, Q. Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software* **5** (2017):15.

[3] Ma, Y., Dixit, V., Innes, M.J., Guo, X. and Rackauckas, C. A Comparison of Automatic Differentiation and Continuous Sensitivity Analysis for Derivatives of Differential Equation Solutions. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)* (2021), pages 1–9.

Presented at the American Conference on Pharmacometrics (ACoP14); Abstract M002; 6 November 2023