Pumas.jl Workshop Solutions

Chris Rackauckas, Vijay Ivaturi

July 21, 2019

1 Problem 1: Simulate a first-order absorption model with linear elimination after a 100 mg oral dose in 24 subjects

Parameters are: `Ka = 1 hr-1`, `CL = 1 L/hr`, `V = 20 L/hr`.

1.1 Part 1: Setup the population

```
using Pumas, Plots, CSV

single_dose_regimen = DosageRegimen(100, time=0)
first(single_dose_regimen.data)
```

	time	cmt	amt	evid	ii	addl	rate	ss
	Float64	Int64	Float64	Int8	Float64	Int64	Float64	Int8
1	0.0	1	100.0	1	0.0	0	0.0	0

to build a sinlge subject

```
s1 = Subject(id=1, evs=single_dose_regimen,cvs=(Wt=70,))
```

```
Subject
  ID: 1
  Events: 1
```

let's first define a function to choose body weight randomly

```
choose_covariates() = (Wt = rand(55:80),)
```

```
choose_covariates (generic function with 1 method)
```

Then, we use generate a population of subjects with a random weight generated from the covariate function above

```
pop = Population(map(i -> Subject(id = i,evs = single_dose_regimen, cvs =
    choose_covariates()),1:24))
```

```
Population
  Subjects: 24
  Covariates: Wt
```

You can view the generated population using by calling a random subject by index and look at the subject's

- covariates
- events
- id numbers
- observations
- time

Let us us peek at the first subject's covariates

```
pop[1].covariates
```

```
(Wt = 67,)
```

1.2 Part 2: Write the model

```
mymodel = @model begin
  @param   begin
    tvcl ∈ RealDomain(lower=0, init = 1.0)
    tvv ∈ RealDomain(lower=0, init = 20)
    tvka ∈ RealDomain(lower = 0, init= 1)
    Ω ∈ PDiagDomain(init=[0.09,0.09, 0.09])
    σ_prop ∈ RealDomain(lower=0,init=0.04)
  end

  @random begin
    η ~ MvNormal(Ω)
  end

  @pre begin
    CL = tvcl * (Wt/70)^0.75 * exp(η[1])
    V  = tvv * (Wt/70) * exp(η[2])
    Ka = tvka * exp(η[3])
  end
  @covariates Wt

  @dynamics OneCompartmentModel
    #@dynamics begin
    # Depot' = -Ka*Depot
    # Central' = Ka*Depot - (CL/V)*Central
    #end

  @derived begin
      cp = @. 1000*(Central / V)
      dv ~ @. Normal(cp, sqrt(cp^2*σ_prop))
    end
end
```

```
PumasModel
  Parameters: tvcl, tvv, tvka, Ω, σ_prop
  Random effects: η
  Covariates: Wt
```

```
Dynamical variables: Depot, Central
Derived: cp, dv
Observed: cp, dv
```

Note that above, we are using the analytical solution in `@dynamics`. You can switch to using the differential equation system if you prefer.

1.3 Part 3: Simulate

Let's first extract the model parameters

```
param = init_param(mymodel)
```

```
(tvcl = 1.0, tvv = 20.0, tvka = 1.0, Ω = PDMats.PDiagMat{Float64,Array{Floa
t64,1}}(3, [0.09, 0.09, 0.09], [11.1111, 11.1111, 11.1111]), σ_prop = 0.04)
```

Then using the `simobs` function, carry out the simulation and visualize the simulation output

```
obs = simobs(mymodel, pop, param, obstimes=0:1:72)
plot(obs)
```

where

- `mymodel` is the model setup in the Part 2,
- `pop` is the population of subjects that was setup in Part 1
- `param` is the specified set of model parameters
- `obstimes` specifies the simulation time period.

2 Problem 2: Peform Non-compartmental analysis

We will start by generating a dataframe of the resuls from the simulation step

```
simdf = DataFrame(obs)
first(simdf, 6)
```

	id	time	cp	dv	amt	evid	cmt	rate	Wt
	String	Int64	Float64	Float64	Float64	Int8	Int64	Float64	Int64
1	1	0	0.0	0.0	100.0	1	1	0.0	67
2	1	0	0.0	0.0	0.0	0		0.0	67
3	1	1	4378.18	4408.47	0.0	0		0.0	67
4	1	2	5662.54	6838.12	0.0	0		0.0	67
5	1	3	5889.9	4976.36	0.0	0		0.0	67
6	1	4	5763.36	6322.58	0.0	0		0.0	67

For the purpose of NCA, let us use the `cp` (output without residual error) as our observed value

To prepare the dataset for NCA analysis, let us use the `read_nca` function. The NCA datasets in Pumas requires a `route` specification which can either be `iv` or `ev`. Since this is an oral drug administration, lets add that to the `simdf`.

```
simdf.route = "ev"
```

```
"ev"
```

Next we can define time, concentration and dose units so the report includes the units for the pharmacokinetic parameters. The general syntax for units are `u` followed by the unit in quotes `""`.

```
timeu = u"hr"
concu = u"mg/L"
amtu  = u"mg"
```

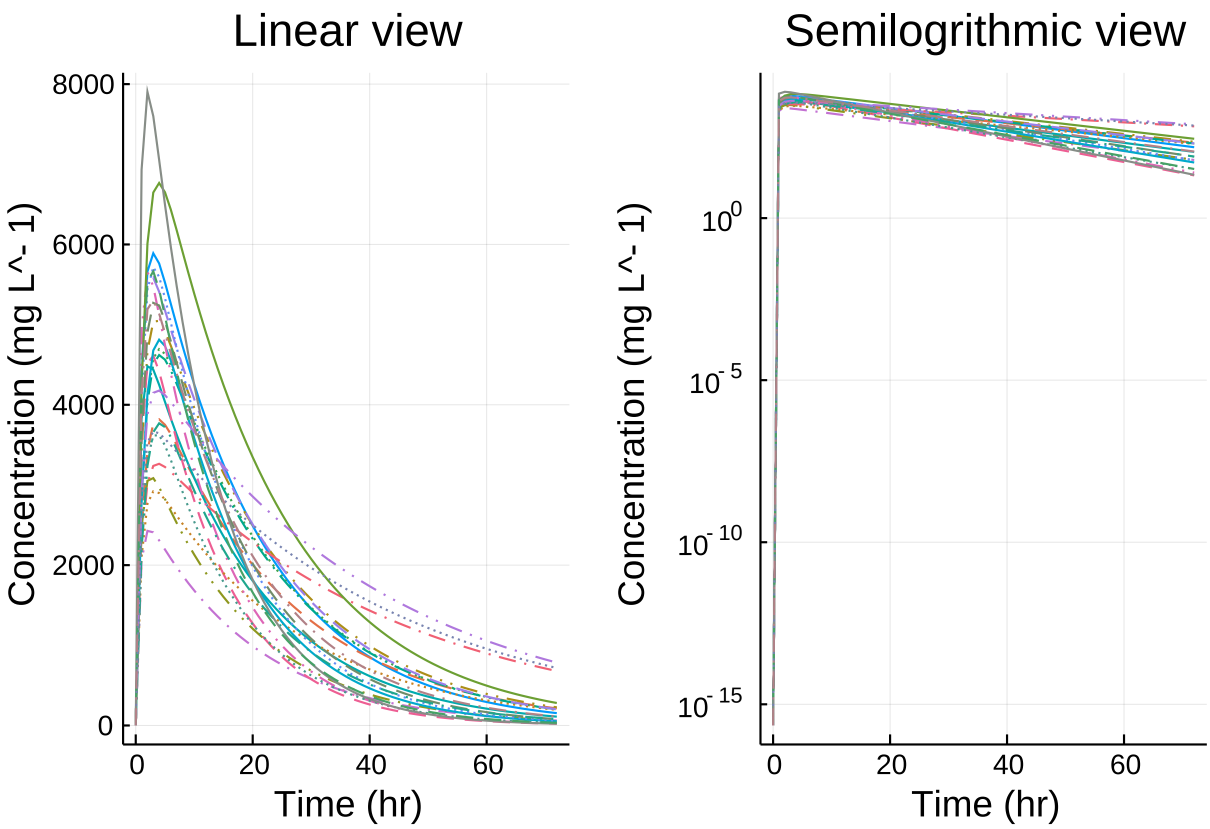
```
mg
```

```
ncadf = read_nca(simdf, id=:id, time=:time, conc=:cp, amt=:amt,
    route=:route,timeu=timeu, concu=concu, amtu=amtu, lloq=0.4concu)
```

```
NCAPopulation (24 subjects):
  ID: ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13",
 "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24"]
    concentration: mg L^-1
    time:          hr
    auc:           mg hr L^-1
    aumc:          mg hr^2 L^-1
    λz:            hr^-1
    dose:          mg
```

You can view the concentration-time plots by doing

```
plot(ncadf)
```



You can then generate `cmax` and `auc` for each subject

```
auc = NCA.auc(ncadf)
```

	id	auc
	String	Unitful
1	1	1.28297e5 mg hr L^-1
2	2	1.04044e5 mg hr L^-1
3	3	1.19677e5 mg hr L^-1
4	4	51584.5 mg hr L^-1
5	5	1.28688e5 mg hr L^-1
6	6	94164.2 mg hr L^-1
7	7	71832.4 mg hr L^-1
8	8	82991.9 mg hr L^-1
9	9	1.18227e5 mg hr L^-1
10	10	62836.1 mg hr L^-1
11	11	92371.9 mg hr L^-1
12	12	1.30654e5 mg hr L^-1
13	13	1.04288e5 mg hr L^-1
14	14	1.52377e5 mg hr L^-1
15	15	84398.4 mg hr L^-1
16	16	1.70218e5 mg hr L^-1
17	17	1.03906e5 mg hr L^-1
18	18	65952.3 mg hr L^-1
19	19	81889.8 mg hr L^-1
20	20	1.8396e5 mg hr L^-1
21	21	1.09155e5 mg hr L^-1
22	22	91688.2 mg hr L^-1
23	23	1.64625e5 mg hr L^-1
24	24	1.09528e5 mg hr L^-1

```
cmax = NCA.cmax(ncadf)
```

	id	cmax
	String	Unitful
1	1	5889.9 mg L-1
2	2	3817.94 mg L-1
3	3	4693.79 mg L-1
4	4	2427.58 mg L-1
5	5	5047.79 mg L-1
6	6	4480.84 mg L-1
7	7	4612.52 mg L-1
8	8	2920.35 mg L-1
9	9	4620.57 mg L-1
10	10	3085.18 mg L-1
11	11	4812.18 mg L-1
12	12	5586.06 mg L-1
13	13	5717.52 mg L-1
14	14	3263.94 mg L-1
15	15	5659.32 mg L-1
16	16	6766.67 mg L-1
17	17	5271.27 mg L-1
18	18	3620.71 mg L-1
19	19	3769.13 mg L-1
20	20	4176.95 mg L-1
21	21	7904.46 mg L-1
22	22	5664.0 mg L-1
23	23	3668.06 mg L-1
24	24	5305.01 mg L-1

Or generate the entire NCA report using

```
report = NCAReport(ncadf)
report = NCA.to_dataframe(report)
first(report,6)
```

	id	doseamt	lambda_z	half_life	tmax	tlag	cmax	clast
	String	Unitful	Unitful	Unitful	Unitful	Unitful	Unitful	Unitful
1	1	100.0 mg	0.0535472 hr-1	12.9446 hr	3 hr	0 hr	5889.9 mg L-1	153.834 mg L
2	2	100.0 mg	0.0431487 hr-1	16.0642 hr	4 hr	0 hr	3817.94 mg L-1	212.74 mg L
3	3	100.0 mg	0.0473535 hr-1	14.6377 hr	4 hr	0 hr	4693.79 mg L-1	201.072 mg L
4	4	100.0 mg	0.0532171 hr-1	13.0249 hr	2 hr	0 hr	2427.58 mg L-1	62.1136 mg L
5	5	100.0 mg	0.0461713 hr-1	15.0125 hr	4 hr	0 hr	5047.79 mg L-1	226.384 mg L
6	6	100.0 mg	0.0537639 hr-1	12.8924 hr	2 hr	0 hr	4480.84 mg L-1	110.14 mg L

3 Problem 3: Estimate using Non-linear mixed effects

We can use the simulated dataset in the Problem 1 for our estimation. We need a couple of data manipulation steps

1. missing `cmt` should be converted to 2 to reflect central compartment
2. data rows where `time = 0`, and `cp=0` should be removed

```
simdf.cmt = ifelse.(ismissing.(simdf.cmt), 2, simdf.cmt)
est_df = simdf[.!((simdf.dv .== 0.0) .& (simdf.cmt .==2)),:]
first(est_df,6)
```

	id	time	cp	dv	amt	evid	cmt	rate	Wt	route
	String	Int64	Float64	Float64	Float64	Int8	Int64	Float64	Int64	String
1	1	0	0.0	0.0	100.0	1	1	0.0	67	ev
2	1	1	4378.18	4408.47	0.0	0	2	0.0	67	ev
3	1	2	5662.54	6838.12	0.0	0	2	0.0	67	ev
4	1	3	5889.9	4976.36	0.0	0	2	0.0	67	ev
5	1	4	5763.36	6322.58	0.0	0	2	0.0	67	ev
6	1	5	5525.25	4881.75	0.0	0	2	0.0	67	ev

3.1 Part 1: Read datasets for NLME estimation

We can use the `read_pumas` function to prepare the dataset for NLME estimation

```
data = read_pumas(est_df ,cvs = [:Wt], dvs=[:dv])
```

```
Population
  Subjects: 24
  Covariates: Wt
  Observables: dv
```

where

- `cvs` takes an array of covariates
- `dvs` takes an array of the dependent variables
- since the dataframe has `time` as the variable, the function does not need a specific input

3.2 Part 2: Perform a model fit

We now use the

- `mymodel` model that we wrote earlier
- the set of parameters specified in `param` as initial estimates
- `data` that was read in using the `read_pumas` function

to fit the model.

```
res = fit(mymodel,data,param,Pumas.FOCEI())
```

```
FittedPumasModel

Successful minimization:                      true

Likelihood approximation:              Pumas.FOCEI
Objective function value:                  11220.7
```

```
Total number of observation records:    1728
Number of active observation records:   1728
Number of subjects:                       24

-----------------
       Estimate
-----------------
tvcl    1.019
tvv    19.913
tvka    0.94629
Ω_1,_1     0.084256
Ω_2,_2     0.054302
Ω_3,_3     0.13733
σ_prop  0.036728
-----------------
```

3.3 Part 3: Infer the results

infer provides the model inference

```
infer(res)
```

```
Calculating: variance-covariance matrix. Done.
FittedPumasModelInference

Successful minimization:                true

Likelihood approximation:        Pumas.FOCEI
Objective function value:            11220.7
Total number of observation records:    1728
Number of active observation records:   1728
Number of subjects:                       24

-------------------------------------------------
       Estimate      RSE         95.0% C.I.
-------------------------------------------------
tvcl    1.019       6.0073 [ 0.899    ;  1.139    ]
tvv    19.913       4.8287 [18.029    ; 21.798    ]
tvka    0.94629     8.6658 [ 0.78556 ;  1.107    ]
Ω_1,_1     0.084256  28.054   [ 0.037928;  0.13059 ]
Ω_2,_2     0.054302  23.117   [ 0.029698;  0.078905]
Ω_3,_3     0.13733   40.91    [ 0.027217;  0.24744 ]
σ_prop  0.036728   2.0633 [ 0.035243;  0.038213]
-------------------------------------------------
```

3.4 Part 4: Inspect the results

inspect gives you the

- model predictions
- residuals
- Empirical Bayes estimates

```
preds = DataFrame(predict(res))
first(preds, 6)
```

	id	time	Wt	pred	ipred	pred_approx
	String	Float64	Int64	Float64	Float64	Pumas
1	1	1.0	67	2917.25	4311.8	FOCEI()
2	1	2.0	67	3952.07	5615.73	FOCEI()
3	1	3.0	67	4226.08	5863.35	FOCEI()
4	1	4.0	67	4201.7	5747.16	FOCEI()
5	1	5.0	67	4065.71	5512.53	FOCEI()
6	1	6.0	67	3892.21	5246.04	FOCEI()

```
resids = DataFrame(wresiduals(res))
first(resids, 6)
```

	id	time	Wt	wres	iwres	wres_approx
	String	Float64	Int64	Float64	Float64	Pumas
1	1	1.0	67	0.952997	0.116996	FOCEI()
2	1	2.0	67	1.37441	1.1358	FOCEI()
3	1	3.0	67	-0.583026	-0.789355	FOCEI()
4	1	4.0	67	0.798189	0.522428	FOCEI()
5	1	5.0	67	-0.32281	-0.597079	FOCEI()
6	1	6.0	67	0.8688	0.598683	FOCEI()

```
ebes = DataFrame(empirical_bayes(res))
first(ebes, 6)
```

	id	time	Wt	ebe_1	ebe_2	ebe_3	ebes_approx
	String	Float64	Int64	Float64	Float64	Float64	Pumas
1	1	1.0	67	-0.227424	-0.274124	0.0897472	FOCEI()
2	1	2.0	67	-0.227424	-0.274124	0.0897472	FOCEI()
3	1	3.0	67	-0.227424	-0.274124	0.0897472	FOCEI()
4	1	4.0	67	-0.227424	-0.274124	0.0897472	FOCEI()
5	1	5.0	67	-0.227424	-0.274124	0.0897472	FOCEI()
6	1	6.0	67	-0.227424	-0.274124	0.0897472	FOCEI()

There is an `inspect` function that provides all the results at once

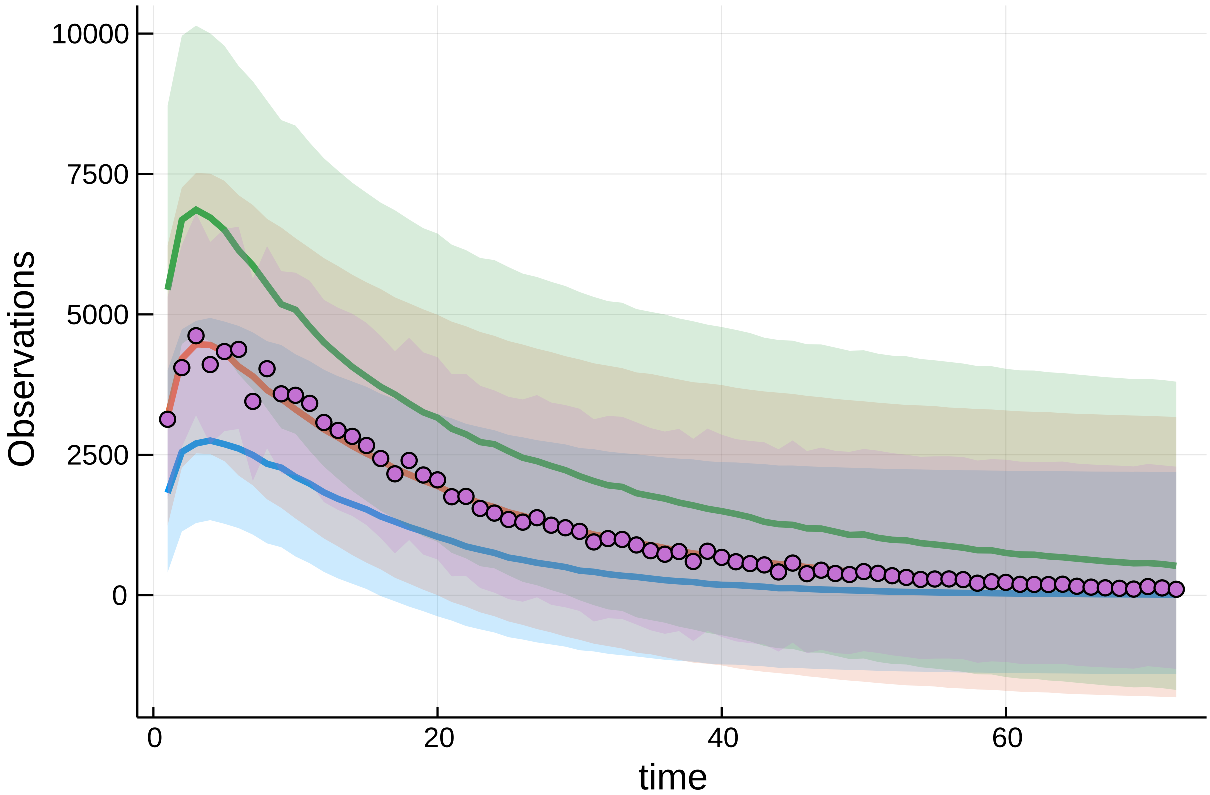
Note that this function below fails to convert into a dataframe due to a bug. Will be fixed soon

```
resout = DataFrame(inspect(res))
first(resout, 6)
```

4 Problem 4: Validate your model

Finally validate your model with a visual predictive check

```
vpc(res,200) |> plot
```



or stratify it based on bodyweight

```
vpc(res,200, stratify_on=[:Wt]) |> plot
```