Pumas.jl Workshop Solutions

August 11, 2020

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, Random
Random.seed!(0)
```

```
MersenneTwister(UInt32[0x00000000], Random.DSFMT.DSFMT_state(Int32[74839879
7, 1073523691, -1738140313, 1073664641, -1492392947, 1073490074, -162528183
9, 1073254801, 1875112882, 1073717145  ...  943540191, 1073626624, 1091647724
, 1073372234, -1273625233, -823628301, 835224507, 991807863, 382, 0]), [0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0  ...  0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0], UInt128[0x00000000000000000000000000000000, 0x000
00000000000000000000000000000, 0x00000000000000000000000000000000, 0x000000
00000000000000000000000000, 0x00000000000000000000000000000000, 0x000000000
00000000000000000, 0x00000000000000000000000000000000, 0x000000000000
00000000000000000, 0x00000000000000000000000000000000, 0x000000000000000
0000000000000000  ...  0x00000000000000000000000000000000, 0x000000000000000
0000000000000000, 0x00000000000000000000000000000000, 0x00000000000000000
00000000000000, 0x00000000000000000000000000000000, 0x0000000000000000000
00000000, 0x00000000000000000000000000000000, 0x00000000000000000000000000
00000], 1002, 0)
```

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

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

to build a single subject

```
s1 = Subject(id=1, events=single_dose_regimen, covariates=(Wt=70,))
```

```
Subject
  ID: 1
  Events: 1
  Covariates: Wt
```

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 it to generate a population of subjects with a random weight generated from the covariate function above

```
pop = Population(map(i -> Subject(id = i, events=single_dose_regimen,
covariates=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
```

```
Pumas.ConstantCovar{NamedTuple{(:Wt,),Tuple{Int64}}}((Wt = 74,))
```

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])
    Vc  = tvv * (Wt/70) * exp(η[2])
```

```
    Ka = tvka * exp(η[3])
  end
  @covariates Wt

  @dynamics Depots1Central1
    #@dynamics begin
    #    Depot' =  -Ka*Depot
    #    Central' =  Ka*Depot - (CL/V)*Central
    #end

  @derived begin
      cp = @. 1000*(Central / Vc)
      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, tvka = 1, Ω = PDMats.PDiagMat{Float64,Array{Float64,
1}}(3, [0.09, 0.09, 0.09], [11.11111111111111, 11.11111111111111, 11.111111
11111111]), 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	Float64	Float64?	Float64?	Float64	Int8	Int64?	Float64	Int64
1	1	0.0	*missing*	*missing*	100.0	1	1	0.0	74
2	1	0.0	0.0	0.0	0.0	0	*missing*	0.0	74
3	1	1.0	2136.14	1928.23	0.0	0	*missing*	0.0	74
4	1	2.0	2803.41	3018.93	0.0	0	*missing*	0.0	74
5	1	3.0	2977.33	3186.01	0.0	0	*missing*	0.0	74
6	1	4.0	2986.35	2872.35	0.0	0	*missing*	0.0	74

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"
```

```
1776-element Array{String,1}:
 "ev"
 "ev"
 "ev"
 "ev"
 "ev"
 "ev"
 "ev"
 "ev"
 "ev"
 "ev"
  ⋮
 "ev"
 "ev"
 "ev"
 "ev"
 "ev"
 "ev"
 "ev"
 "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
```

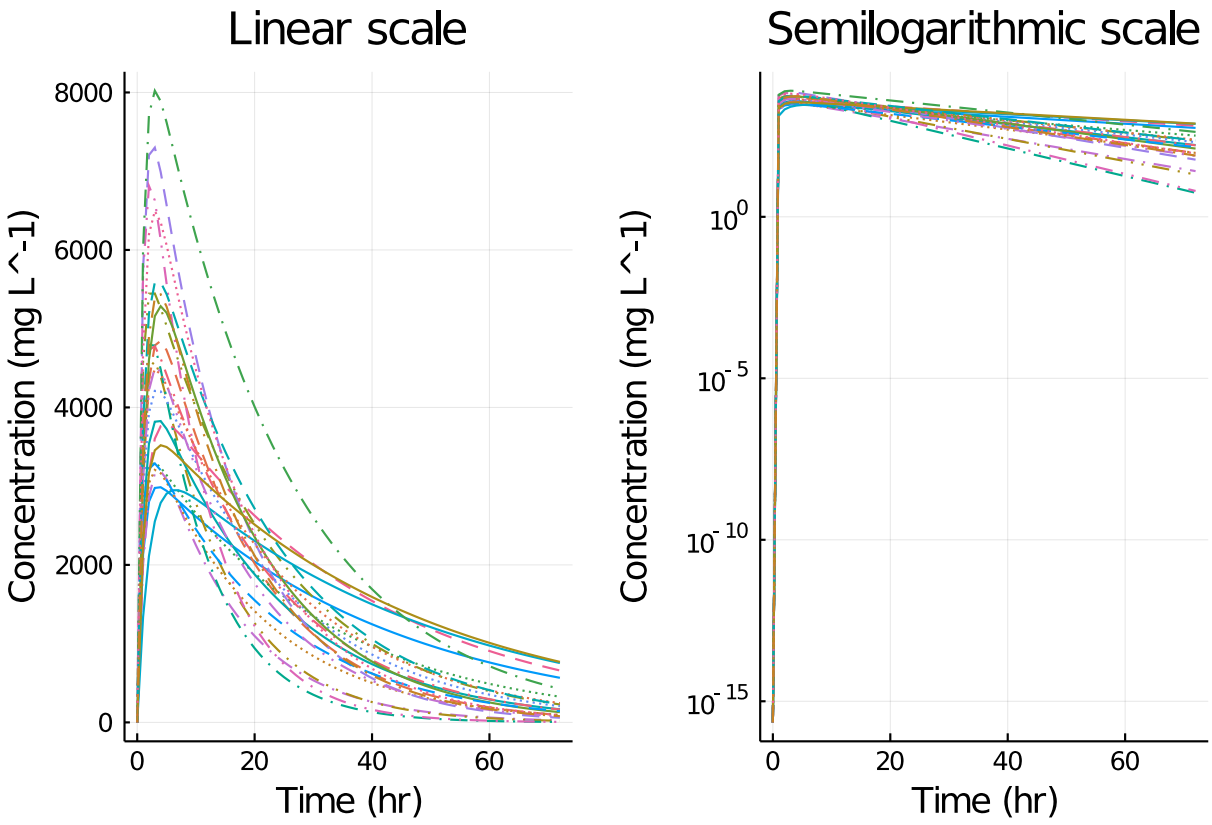
```
ncadf = read_nca(simdf, id=:id, time=:time, conc=:cp, amt=:amt,
    route=:route,timeu=timeu, concu=concu, amtu=amtu, lloq=0.4concu)
```

```
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	Quantit…
1	1	132198.0 mg hr L^-1
2	2	102079.0 mg hr L^-1
3	3	106989.0 mg hr L^-1
4	4	57458.7 mg hr L^-1
5	5	71282.8 mg hr L^-1
6	6	96739.3 mg hr L^-1
7	7	161621.0 mg hr L^-1
8	8	72576.2 mg hr L^-1
9	9	61923.5 mg hr L^-1
10	10	132283.0 mg hr L^-1
11	11	157841.0 mg hr L^-1
12	12	124295.0 mg hr L^-1
13	13	109570.0 mg hr L^-1
14	14	110130.0 mg hr L^-1
15	15	83332.2 mg hr L^-1
16	16	118001.0 mg hr L^-1
17	17	81171.9 mg hr L^-1
18	18	119144.0 mg hr L^-1
19	19	210948.0 mg hr L^-1
20	20	90731.3 mg hr L^-1
21	21	169402.0 mg hr L^-1
22	22	139547.0 mg hr L^-1
23	23	124435.0 mg hr L^-1
24	24	108008.0 mg hr L^-1

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

	id	cmax
	String	Quantit…
1	1	2986.35 mg L^-1
2	2	4845.71 mg L^-1
3	3	3300.26 mg L^-1
4	4	3188.57 mg L^-1
5	5	4587.95 mg L^-1
6	6	3826.88 mg L^-1
7	7	3786.89 mg L^-1
8	8	3206.47 mg L^-1
9	9	4795.85 mg L^-1
10	10	5447.36 mg L^-1
11	11	2948.35 mg L^-1
12	12	7299.77 mg L^-1
13	13	4210.48 mg L^-1
14	14	4776.76 mg L^-1
15	15	6813.03 mg L^-1
16	16	5285.48 mg L^-1
17	17	3288.68 mg L^-1
18	18	4481.94 mg L^-1
19	19	8022.22 mg L^-1
20	20	4503.52 mg L^-1
21	21	3520.06 mg L^-1
22	22	5590.86 mg L^-1
23	23	6493.77 mg L^-1
24	24	5435.34 mg L^-1

Or generate the entire NCA report using

```
report = NCAReport(ncadf)
first(report,6)
```

	id	doseamt	tlag	tmax	cmax	tlast	clast	clast_
	String	Quantit…	Quantit…	Quantit…	Quantit…	Quantit…	Quantit…	Quan
1	1	100.0 mg	0.0 hr	4.0 hr	2986.35 mg L^-1	72.0 hr	566.15 mg L^-1	566.267
2	2	100.0 mg	0.0 hr	4.0 hr	4845.71 mg L^-1	72.0 hr	93.876 mg L^-1	94.0492
3	3	100.0 mg	0.0 hr	3.0 hr	3300.26 mg L^-1	72.0 hr	326.361 mg L^-1	326.425
4	4	100.0 mg	0.0 hr	4.0 hr	3188.57 mg L^-1	72.0 hr	26.0065 mg L^-1	26.0723
5	5	100.0 mg	0.0 hr	3.0 hr	4587.95 mg L^-1	72.0 hr	20.3659 mg L^-1	20.3848
6	6	100.0 mg	0.0 hr	4.0 hr	3826.88 mg L^-1	72.0 hr	166.316 mg L^-1	166.438

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	Float64	Float64?	Float64?	Float64	Int8	Int64	Float64	Int64	String
1	1	0.0	*missing*	*missing*	100.0	1	1	0.0	74	ev
2	1	1.0	2136.14	1928.23	0.0	0	2	0.0	74	ev
3	1	2.0	2803.41	3018.93	0.0	0	2	0.0	74	ev
4	1	3.0	2977.33	3186.01	0.0	0	2	0.0	74	ev
5	1	4.0	2986.35	2872.35	0.0	0	2	0.0	74	ev
6	1	5.0	2941.06	2521.28	0.0	0	2	0.0	74	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, covariates=[:Wt], observations=[:dv])
```

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

where

- `covariates` is a symbol that maps to the columns of covariates
- `observations` is a symbol that maps to the columns of 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())
```

```
Iter     Function value   Gradient norm
     0     1.134964e+04     1.928833e+01
 * time: 4.1961669921875e-5
     1     1.134928e+04     3.577934e+01
 * time: 0.15595006942749023
     2     1.134867e+04     2.016275e+01
```

```
 * time: 0.254626989364624
     3     1.134846e+04     1.467966e+01
 * time: 0.3941769599914551
     4     1.134813e+04     2.651939e+00
 * time: 0.540241003036499
     5     1.134812e+04     2.339406e+00
 * time: 0.7207441329956055
     6     1.134809e+04     1.934372e+00
 * time: 0.9149031639099121
     7     1.134805e+04     3.034282e-01
 * time: 0.997305154800415
     8     1.134805e+04     2.558618e-01
 * time: 1.044020175933838
     9     1.134804e+04     8.600182e-02
 * time: 1.0940351486206055
    10     1.134804e+04     2.772043e-02
 * time: 1.1434760093688965
    11     1.134804e+04     9.824259e-03
 * time: 1.1958811283111572
    12     1.134804e+04     4.457464e-03
 * time: 1.27817702293396
    13     1.134804e+04     1.025930e-03
 * time: 1.323249101638794
    14     1.134804e+04     8.285445e-05
 * time: 1.367711067199707
FittedPumasModel

Successful minimization:                      true

Likelihood approximation:              Pumas.FOCEI
Log-likelihood value:                   -11348.038
Number of subjects:                             24
Number of parameters:         Fixed      Optimized
                                  0              5
Observation records:         Active        Missing
    dv:                        1728              0
    Total:                     1728              0

---------------------
           Estimate
---------------------
tvcl        0.96606
tvv        20.124
tvka        0.89507
Ω_1,_1        0.10253
Ω_2,_2        0.077902
Ω_3,_3        0.074778
σ_prop      0.038981
---------------------
```

3.3 Part 3: Infer the results

infer provides the model inference

```
infer(res)
```

```
Calculating: variance-covariance matrix. Done.
Asymptotic inference results
```

```
Successful minimization:                      true

Likelihood approximation:              Pumas.FOCEI
Log-likelihood value:                   -11348.038
Number of subjects:                             24
Number of parameters:         Fixed      Optimized
                                  0              5
Observation records:         Active        Missing
    dv:                        1728              0
    Total:                     1728              0

-----------------------------------------------------------------
          Estimate           SE                      95.0% C.I.
-----------------------------------------------------------------
tvcl       0.96606         0.063144         [ 0.8423   ;  1.0898  ]
tvv       20.124           1.1628           [17.845    ; 22.403   ]
tvka       0.89507         0.066005         [ 0.7657   ;  1.0244  ]
Ω_1,_1       0.10253         0.026178         [ 0.05122  ;  0.15384 ]
Ω_2,_2       0.077902        0.020065         [ 0.038576 ;  0.11723 ]
Ω_3,_3       0.074778        0.04165          [-0.0068549;  0.15641 ]
σ_prop     0.038981        0.0015704        [ 0.035903 ;  0.042059]
-----------------------------------------------------------------
```

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	evid	dv	dv_pred	dv_ipred	Wt
	String	Float64	Int64	Float64	Float64	Float64	Int64
1	1	1.0	0	1928.23	2705.81	1956.5	74
2	1	2.0	0	3018.93	3686.23	2724.75	74
3	1	3.0	0	3186.01	3967.48	2997.53	74
4	1	4.0	0	2872.35	3968.57	3064.41	74
5	1	5.0	0	2521.28	3860.47	3046.41	74
6	1	6.0	0	3474.89	3712.77	2994.1	74

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

	id	time	dv_wres	dv_iwres	wres_approx	Wt
	String	Float64	Float64	Float64	FOCEI…	Int64
1	1	1.0	-0.411365	-0.073188	FOCEI()	74
2	1	2.0	0.187917	0.546827	FOCEI()	74
3	1	3.0	-0.0281296	0.318467	FOCEI()	74
4	1	4.0	-0.642131	-0.317451	FOCEI()	74
5	1	5.0	-1.17404	-0.87309	FOCEI()	74
6	1	6.0	0.535575	0.813335	FOCEI()	74

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

	Array…
1	[-0.290432, 0.320516, -0.0259105]
2	[0.151995, 0.00669468, -0.0850562]
3	[0.0315156, 0.407506, -0.0865642]
4	[0.688058, 0.303218, -0.362252]
5	[0.334693, -0.183283, 0.16122]
6	[0.0829534, 0.123602, -0.17294]

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)
```