Pumas.jl Workshop Solutions

Chris Rackauckas, Vijay Ivaturi

February 17, 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)
```


```
Random.MersenneTwister(UInt32[0x00000000], Random.DSFMT.DSFMT_state(Int32[7
48398797, 1073523691, -1738140313, 1073664641, -1492392947, 1073490074, -16
25281839, 1073254801, 1875112882, 1073717145  …  943540191, 1073626624, 109
1647724, 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
, 0x00000000000000000000000000000000, 0x00000000000000000000000000000000, 0
x00000000000000000000000000000000, 0x00000000000000000000000000000000, 0x00
000000000000000000000000000000, 0x00000000000000000000000000000000, 0x00000
000000000000000000000000000, 0x00000000000000000000000000000000, 0x00000000
000000000000000000000000  …  0x00000000000000000000000000000000, 0x00000000
000000000000000000000000, 0x00000000000000000000000000000000, 0x00000000000
000000000000000000000, 0x00000000000000000000000000000000, 0x00000000000000
000000000000000000, 0x00000000000000000000000000000000, 0x00000000000000000
000000000000000, 0x00000000000000000000000000000000, 0x00000000000000000000
000000000000], 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 sinlge subject

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

```
Subject
  ID: 1
  Events: 1
  Covariates: (Wt = 70,)
```

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 = 55,)
```

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 Depots1Central1
    #@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, 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	Int64	Float64	Float64	Float64	Int8	Int64	Float64	Int64
1	1	0	0.0	0.0	100.0	1	1	0.0	55
2	1	0	0.0	0.0	0.0	0		0.0	55
3	1	1	5993.99	6699.63	0.0	0		0.0	55
4	1	2	6866.37	6459.5	0.0	0		0.0	55
5	1	3	6662.28	7583.15	0.0	0		0.0	55
6	1	4	6253.33	6887.26	0.0	0		0.0	55

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
```

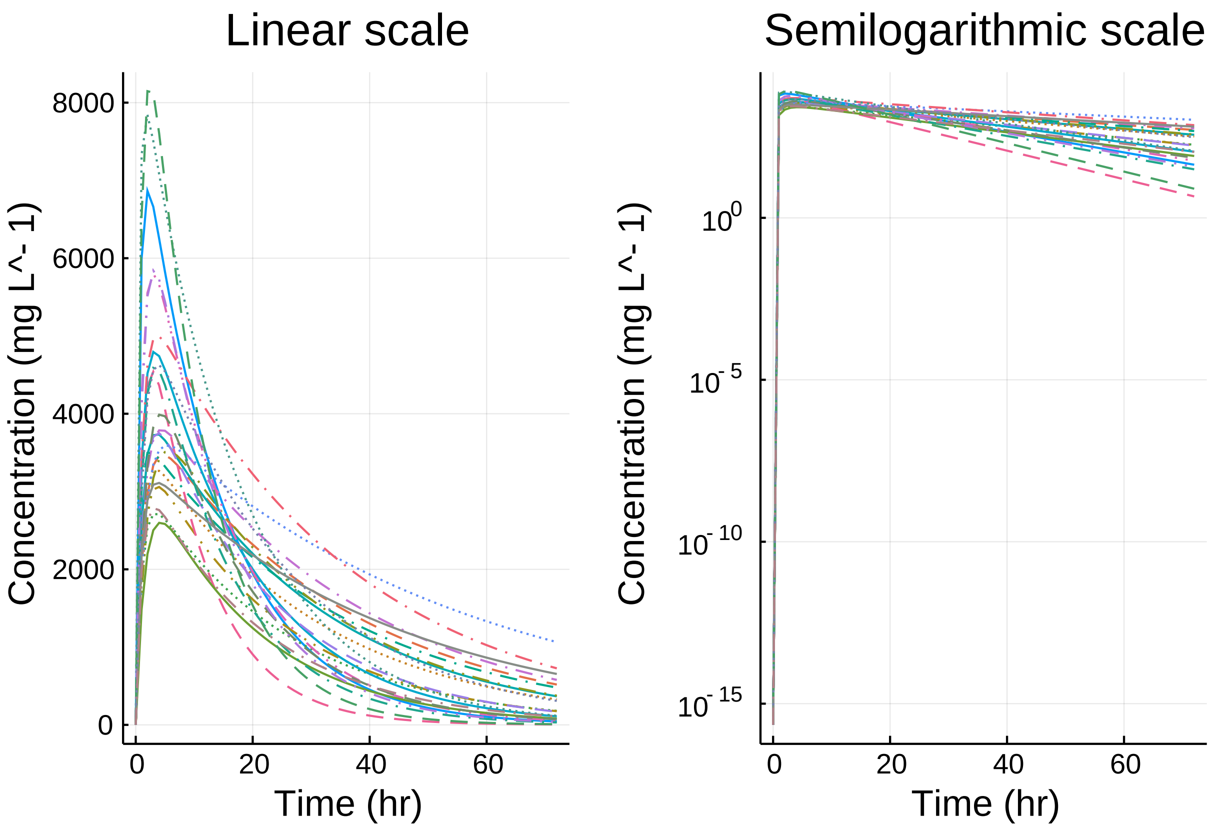
```
ncadf = read_nca(simdf, id=:id, time=:time, conc=:cp, amt=:amt,
    route=:route,timeu=timeu, concu=concu, amtu=amtu, lloq=0.4concu)
```

```
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	Unitful...
1	1	108523.0 mg hr L^-1
2	2	1.37362e5 mg hr L^-1
3	3	77472.5 mg hr L^-1
4	4	1.51225e5 mg hr L^-1
5	5	84021.7 mg hr L^-1
6	6	1.2242e5 mg hr L^-1
7	7	59389.0 mg hr L^-1
8	8	1.07756e5 mg hr L^-1
9	9	130008.0 mg hr L^-1
10	10	1.22717e5 mg hr L^-1
11	11	1.02494e5 mg hr L^-1
12	12	96274.4 mg hr L^-1
13	13	2.12502e5 mg hr L^-1
14	14	1.93418e5 mg hr L^-1
15	15	1.04152e5 mg hr L^-1
16	16	61961.3 mg hr L^-1
17	17	84879.5 mg hr L^-1
18	18	1.43681e5 mg hr L^-1
19	19	79255.8 mg hr L^-1
20	20	98306.3 mg hr L^-1
21	21	1.46144e5 mg hr L^-1
22	22	1.03207e5 mg hr L^-1
23	23	1.33058e5 mg hr L^-1
24	24	67737.9 mg hr L^-1

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

	id	cmax
	String	Unitful...
1	1	6866.37 mg L^-1
2	2	3469.66 mg L^-1
3	3	2710.07 mg L^-1
4	4	3786.48 mg L^-1
5	5	3059.08 mg L^-1
6	6	3733.26 mg L^-1
7	7	4540.05 mg L^-1
8	8	3265.54 mg L^-1
9	9	3496.55 mg L^-1
10	10	3511.88 mg L^-1
11	11	4795.13 mg L^-1
12	12	3749.97 mg L^-1
13	13	3590.13 mg L^-1
14	14	4993.79 mg L^-1
15	15	5804.5 mg L^-1
16	16	2597.4 mg L^-1
17	17	3987.81 mg L^-1
18	18	7845.09 mg L^-1
19	19	4590.03 mg L^-1
20	20	5839.47 mg L^-1
21	21	3110.65 mg L^-1
22	22	8147.68 mg L^-1
23	23	4624.67 mg L^-1
24	24	2785.83 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...	Unitfu
1	1	100.0 mg	0.0730531 hr^-1	9.48826 hr	2 hr	0 hr	6866.37 mg L^-1	43.6488 m
2	2	100.0 mg	0.028807 hr^-1	24.0618 hr	5 hr	0 hr	3469.66 mg L^-1	518.298 m
3	3	100.0 mg	0.0402099 hr^-1	17.2382 hr	3 hr	0 hr	2710.07 mg L^-1	180.311 m
4	4	100.0 mg	0.0283742 hr^-1	24.4288 hr	4 hr	0 hr	3786.48 mg L^-1	578.314 m
5	5	100.0 mg	0.0426007 hr^-1	16.2708 hr	4 hr	0 hr	3059.08 mg L^-1	176.002 m
6	6	100.0 mg	0.0344511 hr^-1	20.1197 hr	4 hr	0 hr	3733.26 mg L^-1	366.833 m

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	55	ev
2	1	1	5993.99	6699.63	0.0	0	2	0.0	55	ev
3	1	2	6866.37	6459.5	0.0	0	2	0.0	55	ev
4	1	3	6662.28	7583.15	0.0	0	2	0.0	55	ev
5	1	4	6253.33	6887.26	0.0	0	2	0.0	55	ev
6	1	5	5825.83	5759.5	0.0	0	2	0.0	55	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
Deviance:                              19742.767
```

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

---------------------
           Estimate
---------------------
tvcl        0.95056
tvv        20.923
tvka        0.8943
Ω_1,_1        0.11207
Ω_2,_2        0.08522
Ω_3,_3        0.1822
σ_prop      0.042688
---------------------
```

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
Deviance:                          19742.767
Total number of observation records:    1728
Number of active observation records:   1728
Number of subjects:                       24

-------------------------------------------------------------
          Estimate       RSE             95.0% C.I.
-------------------------------------------------------------
tvcl       0.95056      6.8651       [ 0.82266 ;  1.0785  ]
tvv       20.923        6.0007       [18.462   ; 23.384   ]
tvka       0.8943      10.371        [ 0.71253 ;  1.0761  ]
Ω_1,_1       0.11207     28.011        [ 0.050543;  0.1736  ]
Ω_2,_2       0.08522     29.941        [ 0.03521 ;  0.13523 ]
Ω_3,_3       0.1822      35.976        [ 0.053727;  0.31067 ]
σ_prop     0.042688     3.2558       [ 0.039964;  0.045412]
-------------------------------------------------------------
```

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	dv_pred	dv_ipred	pred_approx
	String	Float64	Int64	Float64	Float64	Pumas...
1	1	1.0	55	3251.17	5535.34	FOCEI()
2	1	2.0	55	4842.08	6469.77	FOCEI()
3	1	3.0	55	5289.64	6334.61	FOCEI()
4	1	4.0	55	5288.21	5969.44	FOCEI()
5	1	5.0	55	5139.73	5572.61	FOCEI()
6	1	6.0	55	4948.98	5189.74	FOCEI()

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

	id	time	Wt	dv_wres	dv_iwres	wres_approx
	String	Float64	Int64	Float64	Float64	Pumas...
1	1	1.0	55	1.57625	1.01804	FOCEI()
2	1	2.0	55	-0.346817	-0.00768109	FOCEI()
3	1	3.0	55	0.471787	0.95396	FOCEI()
4	1	4.0	55	0.0874083	0.744159	FOCEI()
5	1	5.0	55	-0.491189	0.162325	FOCEI()
6	1	6.0	55	-1.94495	-1.57674	FOCEI()

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

	id	time	Wt	ebe_1	ebes_approx
	String	Float64	Int64	Array...	Pumas...
1	1	1.0	55	[0.178176, -0.221974, 0.4724]	FOCEI()
2	1	2.0	55	[0.178176, -0.221974, 0.4724]	FOCEI()
3	1	3.0	55	[0.178176, -0.221974, 0.4724]	FOCEI()
4	1	4.0	55	[0.178176, -0.221974, 0.4724]	FOCEI()
5	1	5.0	55	[0.178176, -0.221974, 0.4724]	FOCEI()
6	1	6.0	55	[0.178176, -0.221974, 0.4724]	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)
```