NCA Tutorial

Yingbo Ma

November 30, 2018

1 Introduction

This is an introduction to `NCA.jl`, a software for noncompartmental analysis (NCA). In this tutorial we will show how to use `NCA.jl` to analysis data.

1.1 Installation

Currently, `NCA.jl` is a submodule in `PuMaS.jl`, so you only need to install `PuMaS.jl`, and everything will be ready to go.

1.2 Getting Started

To load the package, use

```
using PuMaS.NCA
```

First, let's load the example NCA data inside `PuMaS.jl`. This data have 24 individuals, and each of them has 16 data points.

```
using PuMaS, CSV

file = PuMaS.example_nmtran_data("nca_test_data/dapa_IV")
data = CSV.read(file)
```

here is what the dataset looks like

```
first(data, 6) # take first 6 rows
```

	ID	TIME	TAD	CObs	AMT_IV	AMT_ORAL	Formulation
	Int64	Float64	Float64	Float64	Float64	Float64	String
1	1	0.0	0.0	157.021	5000.0	0.0	IV
2	1	0.05	0.05	141.892	0.0	0.0	IV
3	1	0.35	0.35	116.228	0.0	0.0	IV
4	1	0.5	0.5	109.353	0.0	0.0	IV
5	1	0.75	0.75	66.4814	0.0	0.0	IV
6	1	1.0	1.0	74.7532	0.0	0.0	IV

2 Efficient Computation of Multiple NCA Diagnostics

2.1 AUC and AUMC

We can compute the area under the curve (AUC) from the first observation time to infinity. Below we are accessing the concentration and corresponding time array for the first individual.

```
NCA.auc(data[:CObs][1:16], data[:TIME][1:16])
```

```
263.792662196049
```

```
NCA.auc(data[:CObs][1:16], data[:TIME][1:16], method=:linuplogdown)
```

```
257.8586273987722
```

the keyword argument `method` can be `:linear`, `:linuplogdown`, or `:linlog`, and it defaults to `:linear`. This is a simple interface, however it is not efficient if you want to compute many quantities. The recommended way is to create an `NCASubject` or an `NCAPopulation` object first and then call the respective NCA diagnostic on the data object. To parse data to an `NCAPopulation` object one can call the `parse_ncadata` function and give column names of `id`, `time`, `conc` (concentration), `amt` (dosage), `formulation`, `iv` (IV bolus name). Note that, by default, the lower limit of quantization (LLQ) is 0, and concentrations that are below LLQ (BLQ) are dropped.

```
pop = parse_ncadata(data, id=:ID, time=:TIME, conc=:CObs, amt=:AMT_IV,
    formulation=:Formulation, iv="IV", llq=0)
pop[1]
```

```
NCASubject:
  ID: 1
    concentration: Float64
    time:          Float64
    auc:           Float64
    aumc:          Float64
```

```
λz:             Float64
dose:           Float64
```

Here, each element of `pop` has the type `NCASubject`. Tt is a lazy data structure and actual computations are not performed. When we are instantiating `NCASubject`, it only performs data checking and cleaning. To calculate AUC, one can do:

```
NCA.auc(pop)
```

	id	auc
	Int64	Float64
1	1	263.793
2	2	323.253
3	3	339.848
4	4	373.361
5	5	132.145
6	6	303.86
7	7	380.275
8	8	279.126
9	9	239.831
10	10	260.862
11	11	146.864
12	12	359.489
13	13	522.905
14	14	262.988
15	15	378.993
16	16	206.926
17	17	341.551
18	18	195.925
19	19	433.443
20	20	214.27
21	21	232.537
22	22	471.515
23	23	292.413
24	24	170.305

`AUClast` is the area under the curve from the first observation to the last observation. To compute `AUClast` on the second individual, one would do:

```
NCA.auc(pop[2], auctype=:last)
```

```
302.24594
```

Or to compute the AUC on every individual, one would do:

```
NCA.auc(pop, auctype=:last)
```

	id	auc
	Int64	Float64
1	1	246.932
2	2	302.246
3	3	288.58
4	4	333.804
5	5	129.061
6	6	291.951
7	7	333.994
8	8	259.967
9	9	233.643
10	10	242.719
11	11	141.435
12	12	311.005
13	13	427.174
14	14	246.329
15	15	311.131
16	16	196.672
17	17	319.297
18	18	185.399
19	19	403.216
20	20	202.64
21	21	222.77
22	22	364.756
23	23	265.663
24	24	156.806

One can also compute AUC on a certain interval. To compute AUC on the interval $[10, \infty]$ on the first individual

```
NCA.auc(pop[1], interval=(10,Inf))
```

```
27.82442719604898
```

One can also specify multiple intervals

```
NCA.auc(pop[1], interval=[(10,Inf), (10, 15)])
```

```
2-element Array{Float64,1}:
 27.82442719604898
  4.6593795
```

In many cases, the AUC commands may need to extrapolate in order to cover the desired interval. To see the percentage of extrapolation ($\frac{\text{extrapolated AUC}}{\text{Total AUC}} \cdot 100$), you can use the command:

```
NCA.auc_extrap_percent(pop[1])
```

```
6.391564517256502
```

Area under the first moment of the concentration (AUMC) is

$$\int_{t_0}^{t_1} t \cdot \text{concentration}(t) dt. \tag{1}$$

The interface of computing AUMC is exactly the same with AUC, and one needs to change `auc` to `aumc` for calculating AUMC or related quantities. For instance,

```
NCA.aumc_extrap_percent(pop[1])
NCA.aumc(pop[1])
```

```
1411.6198735770834
```

2.2 Terminal Rate Constant (λz)

The negative slope for concentration vs time in log-linear scale is the terminal rate constant, often denoted by λz. To compute λz, one can call

```
NCA.lambdaz(pop[1])
```

```
0.03876710923615261
```

To get the coefficient of determination (r^2), the adjusted coefficient of determination ($adjr^2$), the y-intercept, the first time point used, and the number of points used while computing λz, one can do:

```
NCA.lambdazr2(pop)
NCA.lambdazadjr2(pop)
NCA.lambdazintercept(pop)
NCA.lambdaztimefirst(pop)
NCA.lambdaznpoints(pop)
```

	id	lambdaznpoints
	Int64	Int64
1	1	5
2	2	3
3	3	5
4	4	6
5	5	5
6	6	10
7	7	4
8	8	4
9	9	7
10	10	4
11	11	6
12	12	5
13	13	4
14	14	6
15	15	3
16	16	6
17	17	4
18	18	5
19	19	7
20	20	3
21	21	5
22	22	3
23	23	4
24	24	3

By default, λz calculation checks last 10 or less data points, one can change it by providing the keyword `threshold`, e.g.

```
NCA.lambdaz(pop[1], threshold=2)
```

```
0.02987746793176597
```

One can also specify the exact data points by passing their indices

```
NCA.lambdaz(pop[1], idxs=[10, 15, 16])
```

```
0.1061738895705389
```

You can also pass their time points

```
NCA.lambdaz(pop[1], slopetimes=[1,2,3])
```

```
0.5387479621404712
```

2.3 Simple functions

`T_max` is the time point at which the maximum concentration (`C_max`) is observed, and they can be computed by:

```
NCA.tmax(pop[1])
NCA.cmax(pop[1])
NCA.cmax(pop[1], interval=(20, 24))
NCA.cmax(pop[1], interval=[(20, 24), (10, 15)])
```

```
2-element Array{Float64,1}:
 0.653632
 1.10532
```

Note that `cmax` returns `C_max` and normalized `C_max` if `dose` is provided. If `dose` is provided in the `NCASubject`, that `dose` will be used by all computations where dose can be used.

`T_last` is the time of the last observed concentration value above the lower limit of quantization (LLQ), and the corresponding concentration value is (`C_last`). They can be computed by the command

```
NCA.tlast(pop[1])
NCA.clast(pop[1])
```

```
0.653632
```

The half-life can be computed by:

```
NCA.thalf(pop[1])
```

```
1.2865889604594303
```

One may need to interpolate or to extrapolate the concentration-time data. For example, if you wanted to interpolate the concentration at $t = 12$ using linear interpolation, you would do:

```
NCA.interpextrapconc(pop[1], 12., interpmethod=:linear)
```

```
0.911367
```

`interpmethod` can be `:linear`, `:linuplogdown`, or `:linlog`.