NCA Tutorial

Yingbo Ma

30nd November 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
```

1.3 Example

First, let's load the example NCA data inside `PuMaS.jl`.

```
using PuMaS, CSV

root = joinpath(dirname(pathof(PuMaS)), "..")
const example_nca_data = CSV.read("$root/examples/nca_test_data/dapa_IV.csv")
concs(i) = Float64.(example_nca_data[:CObs])[16(i-1)+1:16*i]
times(i) = Float64.(example_nca_data[:TIME])[16(i-1)+1:16*i]
```

1.3.1 AUC and AUMC

We can compute the area under the curve (AUC) from the first observation time to infinity

```
auc(concs(1), times(1))
auc(concs(1), times(1), 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 `NCAdata` object first

```
nca = NCAdata(concs(1), times(1))
auc(nca)
```

```
263.792662196049
```

to create many `NCAdata` object at once, one can use Julia's broadcasting machinery

```
ncas = @. NCAdata(concs(1:24), times(1:24));
```

To compute `AUClast`, one can do

```
ncas = @. NCAdata(concs(1:24), times(1:24), dose=5000.)
auc(ncas[1], auctype=:AUClast)
```

```
(246.932184, 0.0493864368)
```

Note that if `dose` is provided, `auc` will return a tuple, which is in the form of (AUC, normalized AUC).

To compute `AUClast` for every subject. Again, it is done by Julia's broadcasting machinery.

```
@. auc(ncas, auctype=:AUClast)
```

```
24-element Array{Tuple{Float64,Float64},1}:
 (246.932184, 0.0493864368)
 (302.24594, 0.060449188)
 (288.579555, 0.05771591100000001)
 (333.80397999999997, 0.066760796)
 (129.06118400000003, 0.025812236800000006)
 (291.9510375, 0.0583902075)
 (333.99432249999995, 0.06679886449999999)
 (259.9668375, 0.0519933675)
 (233.64285700000002, 0.0467285714)
 (242.71946250000002, 0.048543892500000005)
 ⋮
 (196.6721735, 0.039334434700000004)
 (319.29748499999994, 0.06385949699999999)
 (185.39912850000002, 0.037079825700000006)
```

```
(403.216205, 0.080643241)
(202.639537, 0.0405279074)
(222.76995, 0.04455399)
(364.75571750000006, 0.07295114350000001)
(265.6632825, 0.05313265649999999)
(156.80645950000002, 0.031361291900000005)
```

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

```
auc(ncas[1], interval=(10,Inf))
```

```
(27.824427196048966, 0.005564885439209793)
```

One can get extrapolation percentage by

```
auc_extrap_percent(ncas[1])
```

```
6.391564517256502
```

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

```
aumc(ncas[1])
aumc_extrap_percent(ncas[1])
```

```
59.47554145363463
```

1.3.2 λz

To compute λz, one can issue

```
lambdaz(ncas[1])
```

```
(0.03876710923615265, 5, 0.7444781209375907)
```

`lambdaz` returns a tuple in the form of (λz, the number of data points used, r^2). By default, it checks last 10 or less data points, one can change it by providing the keyword `threshold`, e.g.

```
lambdaz(ncas[1], threshold=15)
```

```
(0.03876710923615265, 5, 0.7444781209375907)
```

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

```
lambdaz(ncas[1], idx=[10, 15, 16])
```

```
(0.03876710923615265, 5, 0.7444781209375907)
```

1.3.3 Simple functions

`T_max` and `C_max`:

```
tmax(ncas[1])
cmax(ncas[1])
cmax(ncas[1], interval=(20, 24))
```

```
(0.653632, 0.0001307264)
```

note that `cmax` returns `C_max` and normalized `C_max` if `dose` is provided.

`T_last` and `C_last`:

```
tlast(ncas[1])
clast(ncas[1])
```

```
0.653632
```

`T_half`:

```
thalf(ncas[1])
```

```
17.879774742490834
```

Interpolation/extrapolation:

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

```
0.911367
```

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