7. Napp のモデルの作成法

7.1. 概要

何らかの性質を解析の対象として調べようとするときに、その性質をうまく表す数式が分かっていると非常に役立ちます。例えば、薬の効果や安全性を考えるためにその血中濃度を予測しようとすれば、血中濃度を表す何らかの数式が必要となるでしょう。このように何らかの性質の変化、あるいは他との関係を説明する数学的表現を、ここではモデルと呼ぶことにします。Napp の基本的な機能はモデルを使って対象の性質を解析することです。単純に数式と言わずに数学的表現としたのは、計算に特殊なテクニックを要するものが含まれるからです。Napp は多様なモデルに対応しており、またモデルの新規作成、修正が容易にできるように工夫されています。

この文章で説明する Napp のモデルは比較的単純に数式のように記述するもので、これをインタープリター型のモデルと呼ぶことにします。インタープリター型のモデルはプログラミングの知識がなくても作成できます。Napp にはもう 1 つのモデルの形態があり、これはプログラムを作成するもので、実行速度がやや速い、自由に記述できるとのメリットがありますが、作成するには専門的な知識が必要なので、ここでは取り扱いません。このモデルはバンドル型と呼びます。特殊なモデルや実行速度を極めたい場合は、バンドル型のモデルを考慮すべきかもしれませんので、著者まで御相談下さい。

専門的には...

Napp ではインタープリター型、バンドル型に関わらず、どのモデルでも通常のシミュレーションや非線形最小二乗法によるパラメータ推定ができるだけでなく、ポピュレーション解析（拡張最小二乗法）、およびベイズ推定を行うこともできます。

なお、バンドル型のモデルはプログラミング言語 Objective-C でソースコードを作成し、これをコンパイルしてバンドルを作成し、最終的にダイナミックリンクします。非線形偏微分方程式のモデルはこの方法で作る必要があります。バンドルとはアプリケーションのリソースの形式の 1 つで、アップルのコンピューターの専門用語です。

7.2. モデルの種類

Napp では通常の解析式の他に微分方程式、ラプラス変換式、偏微分方程式のモデルを扱うことができます。以下ではこれらの意味について説明します。

7.2.1. 解析式

解析式のモデルは明示的なもので、例えば x の値を与えた時に y の値を返す数式です。この場合に、x は自由に決められるので独立変数、y は x に従って決まるので、従属変数と呼ばれます。例えば時間 t に従って血漿中濃度を定義するのならば、t が独立変数となります。単純な静脈内投与後の 1-コンパートメントのモデルは以下となります。

$$y = \frac{Dose}{Vd} e^{-Ke\,t} \qquad (1)$$

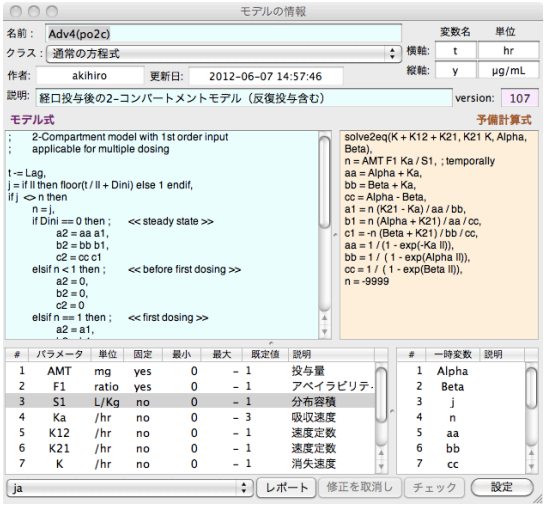
Dose は投与量、Vd は分布容積、Ke は排泄速度です。分布容積や排泄速度の意味については薬物速度論の教科書を参照下さい。ここで Dose, Vd, Ke はいずれも数式の性質を定めるのでパラメータと呼びます。その値を変えて y をシミュレーションしたり、その逆に y の実測値から、適切なパラメータの値を計算することができます。ただし、数式からも明らかなように、Dose と Vd はどちらかの値を決めないと他方も決まりません。この場合は Dose は既知のことが多いので、これを定数とするのが一般的でしょう。式(1)をこのままの形でコンピュータに入力するのは、書式の設定が煩雑ですので以下のように記述します。

y = Dose / Vd exp(-Ke t)　　(2)

記述の文法についてはほとんど自明なのでここでは省略しますが、細かくは後の章で述べることにします。なお、(2)はここでの説明用の式の番号ですので、実際には入力しません。モデルによっては従属変数が複数定義される場合もあります。例えば(2)式で排泄量についても記述しようとすれば、y1 を血漿中濃度、y2 を排泄量として、

y1 = Dose / Vd exp(-Ke t)
y2 = Dose (1 - exp(-Ke t))　　(3)

と定義できます。Napp では従属変数の数に制限はなく、(3)式の場合、y1 はコンパートメント 1、y2 はコンパートメント 2 の値と解釈されます。ここでy1、y2 の 1、2 はインデックスであり、下付きであると解釈すると分かりやすいでしょう。



7.2.2. 解析式

簡単な式であれば、モデル式を素直に入力すればモデルの設定は終了です。例えば、単純な 1-コンパートメントモデルであれば、先に述べた式(2)をそのまま入力します。ただし、独立変数に依存しない計算は、その部分を予備計算式として入力する方が効率が良いので、以下のように設定することも可能です。（実際にはこの場合の計算効率の向上は無視しうる程度ですが、説明の例として挙げます。）

予備計算式

a = Dose / Vd

モデル式

y = a exp(-Ke t)

ここで、変数 a はパラメータ Dose, Vd により一意的に決まり、独立変数 t には関わりません。このような変数はモデルの計算中に一時的に用いられるだけですので、パラメータには含めず一時変数と呼びます。一時変数の値を解析後に知りたい場合は、その頭文字を大文字とすればその値がレポートに出力されます。

予備計算式およびモデルの計算式を入力した後にチェックボタンを押すと、定義されたパラメータあるいは一時変数名が自動的に判別されてリストが表れるので、矛盾がないかを確認します。数式に明らかな定義のエラーがあるとエラーメッセージが現れ、エラーの生じている行が選択されますので修正して下さい。エラーメッセージが出なくても、意図しないパラメータあるいは一時変数名がリストに表れるのは、数式の記述の文法がおかしい場合ですので注意して下さい。また、パラメータと変数の区別が適切であるかも確認して下さい。

パラメータと一時変数のリストにより、それぞれ初期値、とりうる値の制限などが設定可能です。ただし、これらの設定は実際に使用する時にまた自由に変更

できますので、最初のうちは無視していても構いません。ここでパラメータの固定とは、当てはめ計算の推定の対象とするかどうかであり、上記のモデルの場合、Dose は既知でなので固定とするのが普通でしょう。リストの左端の数字を入力することで、順番を変更することができます。リストの説明はモデルや数式の意味がわかりやすいように、必要に応じて適宜入力して下さい。リストの入力時にリターンキーを押すとカーソルは下に移動し、タブキーを押すと右に移動します。

以下にモデル式で認められる数式の文法の詳細を記述します。モデル式は単純な数式としての表現に加えて、必要な場合には、複数の数式を組み合わせ、多少プログラミング的な if ~ then ~ などの条件分岐を行うことができます。もし、複雑なモデルを想定しないのであれば、後者の機能とその文法についてはスキップしても良いでしょう。

A. 数式的記述の文法

i. 空白、改行

数式を複数行にわたって書くことができます。空白や改行は無視されます。

ii. コメント

数式の中に自由にメモとしてコメントを書くことができます。コメントの書き方には２種類あり、セミコロン「;」以降、その行の終わりまでのもの、もう１つは{}でくくられたものです。コメントを適切に入れて、モデルや数式の意味を分かりやすくするのは良い習慣です。

iii. 演算子

加減乗除は+ - * / で表現します。ただし、乗算記号は省略可能で半角スペースで代用できます。半角スペースが必ず必要であること、全角スペースを使わないことに注意して下さい。累乗は^と**の両方を使うことができます。

iv. 計算順序

関数の計算は乗除優先でなされます。^および**は加減乗除に優先して計算され、負を表す演算子の-の場合はさらに優先されます。すなわち、-1^2 は (-1)^2 = 1 と解釈されます。この場合などは分かりにくいので関数の計算順序を明確に()で指定するのが良いでしょう。()は入れ子で用いることができ、その数に制限はありません。計算順序を指定する目的で[], {}を用いることはできません。

v. 代入

代入は=を使います。従って、a = a + 1 は a に 1 を加えるとの意味になります。この表現のかわりに a += 1 を用いることもできます（これはプログラミング言語 C や Java と同じ文法です）。同様に -=, *=, /=, ^= を用いることができます。なお、等価の判定には後述するように==を使います。

vi. 定数

定数は通常の実数で表現します。また、1.23-04e, -.23+18e などの指数表現も可能です。定数はすべて浮動小数点の実数として評価されます。

vii. パラメータおよび一時変数名

パラメータおよび一時変数名は基本的に任意の文字列で定義でき、漢字などの全角文字も使用可能です。ただし、上つきなどの書式は設定できません。英字の大文字と小文字は区別され、従って ke と Ke は別の変数と見なされます。長さに特段の制限はありませんが、あまり長い変数名だとシート上に全てが表示されないので不便です。

半角の記号文字（!"|$%&'()=~-^¥`{+*}<>?@[;:],./）は変数名の一部としては使用できません。アンダーライン_は使えます。また#は定義済みの定数、#PI と#E（円周率π、指数 e）としてのみ使えます。数字は２

文字目以降には含められますが頭文字には使えません。従属変数名には後ろに数字を加えてコンパートメントの番号を表しますので、名前本体に数字は含めないようにします。

予約語（if, then, else, elsif, endif, not, and, or, xor, for, to, while, loop, endloop, break, continue、ただし現在のバージョンでは予約語の全ては機能してはいません）、および以下であげる定義済み関数名は避けて下さい。誤って定義済みの関数名をパラメータ名に使用すると、パラメータ名としては登録されないので分かります。一時変数名についても同様です。

viii. 従属変数名

従属変数名は基本的に頭文字を小文字とすることを勧めます。これは特にラプラス変換のモデルの場合には必要条件となります。従属変数名に数字が引き続くと（例えば y1, y2...）数字に該当するコンパートメントの値とみなされます。ただし、数字が飛んだ場合（例えば y0, y3...）、コンパートメント番号は飛ばずに 1 から順に設定されます。

ix. 定義済み関数

定義済みの関数は以下があります。なお、関数名は小文字で記述して下さい（開発の初期バージョンからは変更されています）。

二乗根、三乗根：sqrt(x), cbrt(x)
自然対数：ln(x)あるいは log(x)
常用対数： log10(x)
指数：exp(x)
切り上げ、切り捨て、四捨五入：ceil(x), floor(x), rint(x)
三角関数：sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), atan2(a, b), sinh(x), cosh(x), tanh(x) asinh(x), acosh(x), atanh(x)
特殊関数
gamma(x): ガンマ関数
lgamma(x): ガンマ関数の自然対数
signgam(0): ガンマ関数の符号
igamma(a, b): 不完全ガンマ関数
ibeta(a, b): 不完全ベータ関数
erf(x): 誤差関数
erfc(x): 相補誤差関数
rand(0): [0, 1)の一様乱数
normrand(mean, sd): 正規乱数
lnormrand(mean, sd): 対数正規乱数
tdist(a, n): 自由度 n の t 分布の分布関数
strange(a): 無限大、無限小、不定の判別
import(a,b)、export(a,b): 前後のシートとの情報交換

特殊関数は複素数に対応していません。従ってラプラス変換式のモデルでは使わないで下さい。上の例のなかで、引数の 0 には数値としての意味はありません。

x. 線形方程式の解の公式

Napp は 4 次方程式までの解の公式により解くことができます。これらの解法は数値計算ではありませんので、高速で比較的誤差が少ないものです。ただし、重解に極めて近いなどの特殊なケースでは誤差を生ずることがありますので注意が必要です。また、引数として複素数を用いることはできません。なお、5 次以上の方程式の解の公式は、得られないことが証明されています。

solve2eq(b, c, x1, x2)
2 次方程式 y = x^2 + b x + c を解いて解を x1, x2 に代入します。関数本体は実数解の数、すなわち 0 か 2 を返します。これが 0 の場合の虚数解は x1 ± x2 i となります。

solve3eq(b, c, d, x1, x2, x3)
3 次方程式 y = x^3 + b x^2 + c x + d をカルダノの公式により解いて解を x1, x2, x3 に代入し

ます。関数本体は実数解の数、すなわち 1 か 3 を返します。これが 1 の場合の実数解は x1、虚数解は x2 ± x3 i となります。

solve4eq(b, c, d, e, x1, x2, x3, x4)

4 次方程式 y = x^4 + b x^3 + c x^2 + d x + e をフェラーリの公式により解いて解を x1, x2, x3, x4 に代入します。関数本体は実数解の数、すなわち 0 か 2 か 4 を返します。これが 0 の場合の解は x1 ± x2 i と x3 ± x4 i、2 の場合は x1, x2 と x3 ± x4 i となります。

x. ニュートン-ラフソン法による方程式の解法

ニュートン-ラフソン法により初期値を与えることで、どのような方程式でも数値的に解くことができます。なお、この方法は虚数解には対応していません。

newton_solve(equation, init, x)

equation が 0 になる変数 x の値を返します。equation として変数 x により値の変化する任意の数式を用いることができます。init は x の初期値です。

newton_min(min)

探索する変数 x の最小値を設定します。初期状態では最小値は設定されていません。

newton_max(max)

探索する変数 y の最大値を設定します。初期状態では最大値は設定されていません。

newton_free(0)

探索する変数の範囲の制限を解除します。引数 0 に意味はありません。

newton_init(loopMax)

変数の探索回数の上限を設定するとともに、ニュートン-ラフソン法に関する探索回数の上限以外の全ての計算の設定を初期状態に戻します。探索回数の上限の初期値は 3000 です。

newton_abs(flag)

変数を探索する時に、相対的に変化させるか絶対的に変化させるかを設定します。変数の変化が 0 をまたがない場合には、相対的変化の方が結果は安定しています。相対的とする場合は、flag を 0、絶対的とする場合は flag を 1 とします。初期状態では相対的です。

newton_delta(delta)

変数を探索するときに、変数を変化させる差分の初期値を設定します。差分は自動調節されますが、その初期値を定めます。初期値は 0.0001 です。

newton_criteria(criteria)

変数の探索の終了は、関数値の改善が criteria 以下になった場合と定義されています。初期値は 0.0000001 です。

newton_error(errorValue)

変数の探索に失敗した場合に返す値を設定します。初期値は 0 です。

B. 複数の式を組み合わせる場合の文法

i. 複数の式を組み合わせ

式と式の境目はカンマ「,」で表します。ただし、式の最後には不要です。

ii. 条件分岐

以下の形式で条件の判断により式を選択して実行することが可能です。

if t < Lag then

```
        y = 0
elsif t < T then
        y = Dose A / Vd
else
        y = Dose / Vd Exp(- Ke (t - Lag))
endif
```

if を使う場合に then と endif は省略できません。これに対し、elsif, else は必要な場合だけ使います。カンマ「,」は式と式の区切りに必要ですが、制御語である then, elsif, else, endif の直前、あるいは直後には不要であることに注意して下さい。この場合に、if から endif までが一つの数式と見なされます。なお、上の例は以下のようにも書けます。（すなわち if 文は数式として値を持ちます。）

```
y =    if t < Lag then     0
       elsif t < T then    Dose A / Vd
       else                Dose/Vd Exp(-Ke(t - Lag))
       endif
```

iii. ループ

以下の形式で、条件判断によるループが可能です。

```
c = 0,
while a < b and c < 100 loop
    a += d,
    c += 1
endloop
```

while を用いる場合には loop、endloop を置くことが必要です。ループは条件を適切に設定しないと、無限に実行を続けコントロール不能に陥りますので十分注意して下さい。上の例で c によりループの実行回数を確認し、上限に達したらループを出るようにしているのは、このような危険を避けるためです。現バージョンでは、例えばプログラミング言語 C の for や do ~ while に相当する構造はありません。また、break, continue に相当する機能もサポートされていません。

iv. 条件

条件判断としては、==, <, >, <=, >=, <> が使えます。a > b は a は b より大きい、a >= b は a は b と等しいか大きい、a <> b は a と b は等しくない、との意味になります。もし if a then...と条件のところに変数をおくと、if a <> 0 then...と同じ意味になります。if a ==b then....と書くと a と b が等しければ成立するとの意味ですが、これを誤って if a = b then....と書くと、a に b を代入してその値が 0 でなければ成立すると解釈されますので注意して下さい。

v. 条件の結合

条件の結合としては and、or と xor があります。and は A と B の両方が成立する場合で、A が成立しなければ B は評価されません。or は A か B のどちらかが成立する場合で、A が成立すると B は評価されません。xor は A か B のどちらかだけが成立する場合（排他的論理和）で、必ず A, B の両方が評価されます。

7.2.3. 微分方程式

微分方程式における式の定義の基本は解析式の項を参照下さい。ここでは微分方程式に特有の記述について述べます。

A. 微分の表現

微分方程式では従属変数を y とすれば、その微分を定義すること、および y の初期値を与えることが必要です。微分は従属変数名に「'」をつけて表現しますので、基本的には(5)~(7)式の形式で入力します。

B. @文

初期値を与えるためには、時間 0 の時にだけ実行す

る数式のブロックを定める必要があります。このように、時間がある条件をみたす場合だけ実行する数式のブロックを区切るために、以下の@文があります。@文は@で始まり:で終わります。@文が有効なブロックは@文から@文、あるいは@文から最後までとなります。

i. @At T:

時間が T に等しい場合のみ実行されます。従って、初期条件は@At 0:になります。また、時間 T に瞬時投与が起こる場合などは、以下で表現できます。

```
@At T:
  y  += dose
```

@At 文のブロック中では微分値(y')の値を参照することも代入することもできません。なお、@At 文で初期値を与えない場合、すべてのコンパートメントの初期値は 0 と仮定されます。

ii. @From T1 to T2:

時間が T1 から T2 の間の場合だけ実行されます。[to T2]は必要無ければ省略できます。@From:ブロックでは、基本的には従属変数(y)の値を参照して、その微分値(y)を定義することになります。ただし、PK-PD モデルなどでは従属変数の値そのものを定義することも可能です。例えば、コンパートメント 1 、 2 をそれぞれ血中、薬効部位のコンパートメントとし、コンパートメント 1 の濃度は普通の解析式で与えておき、コンパートメント 2 への移行は非線形の微分方程式に従うなどの定義が可能です。

7.2.4. ラプラス変換方程式

ラプラス変換方程式における式の定義の基本は解析式の項を参照下さい。ここでは特有の記述について述べます。

A. 補助変数 (s)

ラプラス変換では、解析式の独立変数(例えば時間 t)が変換を受けるので方程式の中には直接あらわれなくなります。その代わりに変換された補助変数(例えば s)が使われます。すなわちラプラス変換方程式とは、見かけの独立変数 s の値によって従属変数(y)のラプラス変換(Y)を定義するものとなります。ここで逆ラプラス変換計算を行う時には、s は複素数として扱われますが、Napp は複素数への変換、複素数の関数計算を自動的に行いますので、ユーザーがこれを意識することはほとんどありません。ただし、定義済みの関数で特殊なものは複素数では実行不能です。通常の四則計算、指数、対数、三角関数は問題ありません。

ラプラス変換式では以下の特殊関数が使えます。

dispersionModelClosed(Dn, s)

拡散定数 Dn の拡散モデルを定義します。境界条件は closed conditions (Dankwerts)となります。s を s + K とすると、K は消失速度定数になります。

B. @文

ラプラス変換方程式の中で直接もともとの独立変数(時間 t など)を参照することはできません。しかし、時にはこの値によって定義する関数を変えたい場合が考えられます。持続投与が終わった場合、消失速度が変わった場合などが相当します。この場合は微分方程式の項で述べたのと同様に@From ~ to ~ :によるブロック定義が使えます。

また、ラプラス変換方程式でラグタイムを考える場合には次のように記述できます。

```
@From T1 to T2 with time -= Lag:
```

これは多少込み入った問題になるので深入りしませんが、一般に逆ラプラス変換はラグタイムを含むモデルの計算が不得意で、時間がかかる上に計算精度も悪くなります。これを避けるため、Napp では上記の

ように記述すると、ラグタイムなしで逆ラプラス変換を行い、その後でラグタイムを考慮して結果を調整します。

なお、逆ラプラス変換の計算には、時間０の値は計算できないとの欠点があります。この計算エラーを避けるため、Napp ではデフォルトでは時間０の値は０が返るようになっています。モデルによってはこれが不都合で時間０の値を別に定義したい場合は、@At 0: ブロックで従属変数(y)の値を与えます。