ISoP Webinar Series

Introduction to PMXStan

AN R PACKAGE TO FACILITATE BAYESIAN PKPD MODELING WITH STAN

YUAN XIONG AND WENPING WANG

MAY 2016

Stan is by far one of the most efficient tools for Bayesian statistical modeling and inference

"Sampling Through Adaptive Neighborhoods"

- a probabilistic programming language implementing full Bayesian statistical inference
  - Similar (and probably improved) modeling language as BUGS
  - Uses Hamiltonian Monte Carlo algorithm (a variant of MCMC) for sampling (from a probability distribution)
  - Implements penalized maximum likelihood estimation with optimization

Composite of a Stan program

Blocks for model building

- Essential: data, parameters, model
  - **`data`**: reads input data and declares associated variables
  - **`parameters`**: specifies parameters to be estimated
  - **`model`**: defines priors and sampling statements
- If needed: **`transformed data`**, **`transformed parameter`**, **`generated quantities`**
- All data, parameters, and variables need to be declared and type-specified following Stan requirements.

```
data {
  int<lower=0> N;
  vector[N] x;
  vector[N] y;
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model {
  for (n in 1:N)
    y[n] ~ normal(alpha + beta * x[n], sigma);
}
```

Steps to run a Stan program

- The Stan code for a specific model is translated to a C++ program
- The C++ program is compiled to a self-contained platform-specific executable
- Run the Stan executable for the model: read in data and then generate samples

Current hurdles and challenges in practical PK/PD modeling with Stan

Large amount of C-like codes need to be written for various tasks not directly related to PK/PD modeling

- Strict rules of declaring and transforming data, variables, and parameters
- Lack of convenient ways to handle data in batch, often requires loops within/after loops

Input data can be challenging to format from a standard PK/PD data set

- Input data format can depend on model specification, parameterization, etc.
- Sometimes model code needs to be changed for the same model but different data sources
- Inputs involving discrete time events such as dosing, confounded by various routes, can be difficult to be integrated to the model

Current Stan release has not yet fully developed to handle generic ODE models

- Some model-specific ODE solver was tailored for rather narrow industry applications
- Formal Stan release has an ODE solver but deemed non-workable for a simple 2-compartment PK model using simulated data (PAGE2015 poster5486)
- A recently developed generic ODE solver is not readily prepared for handling PKPD and has not been tested for real PKPD data (https://github.com/stan-dev/stan/wiki/ODE-Integrator-Support)

No “standard” procedure for PK/PD model qualification has been clearly defined under a Bayesian setting

- Different interpretation for parameter estimation and model comparison with Stan output compared to NONMEM output
- Various ways to generate goodness-of-fit plots and make inferences using posterior samples for pharmacometrics uses

Are there ways to tackle some challenges or bypass some hurdles?

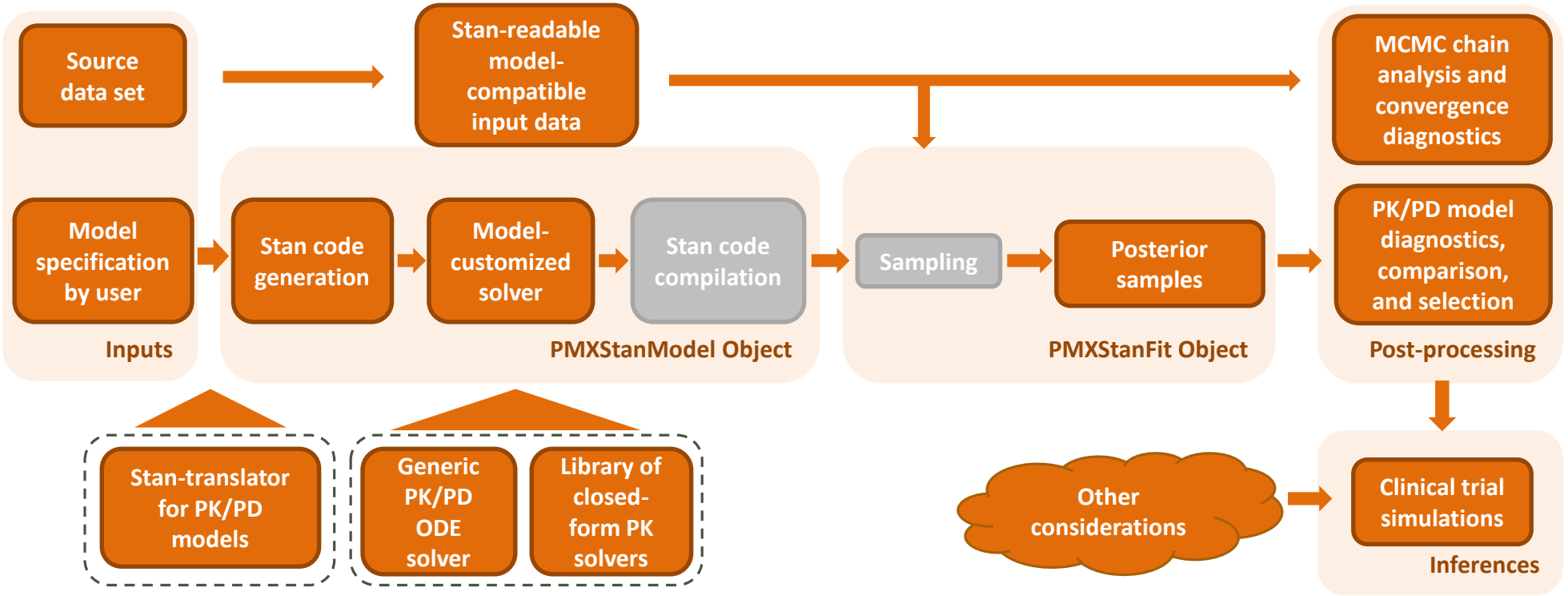
In terms of coding PK/PD models in Stan:

- Can we ***modularize*** some processes of model specification, building, fitting, and diagnostics for common pharmacometrics practice?
- Can we ***standardize*** certain parts of Stan codes as well as the input data formats, so that they can be naturally compatible with each other?

In terms of solving PK/PD systems:

- For ***dosing events with various routes***, can we design a mechanism to integrate them seamlessly into the dynamic system?
- For standard compartmental PK models, can we code in the ***closed-form solutions*** that are readily called by the users?
- For more generic ODE-based PK/PD models, can we develop ***an MCMC sampler-compatible ODE solver*** that can handle most PK/PD data sufficiently well?

We developed an R package to facilitate Stan-based model building, diagnosis, and simulation for pharmacometrics uses



Model building, fitting, and diagnostics process in PMXStan for a population PK model

```
m <- PMXStanModel(path = "pk_abc123",
                  route = "IV_infusion",
                  compile = T
                  )

dat <- prepareInputData(data.file="data_poppk_abc123.csv",
                        model = m
                       )

fit <- PMXStanFit(m, dat, iter = 400, chains = 4)

traces(fit)
waic(fit)
gofplot(fit)
```

Model specification by user

a 2-cmpt population PK model for IV infusion, parameterized by clearance-volume, solved by closed form solution, and compiled during initialization

Automated modeling process

- Stan code generation
- Data preparation
- Invoke NUTS (No-U-Turn-Sampler) for sampling

Post-processing

- Trace plots to check convergence
- Bayesian model specific diagnostics
- PK/PD specific goodness-of-fit for model diagnostics

PMXStan provides a NUTS-compatible LSODA solver to handle PKPD models with generic ODEs

All the user needs to do is to provide a set of ODEs

- A customized ODE solver (called an ODE extension) is then generated specifically for the input ODE system.
- System parameters are recognized and output for the convenience of model specification by users.

```
> ode <- "
      C2 = centr/V;
      d/dt(depot) =-ka*depot;
      d/dt(centr) = ka*depot - ke*centr;
      d/dt(eff) = (1+Emax*C2/(C2+EC50))*Kin - Kout*eff;
 "
> instant.stan.extension(ode)
A new ODE extension for Stan has been created.
System parameters are: V ka ke Emax EC50 Kin Kout
```

Advantages

- High efficiency since written in C++ directly
- Seamlessly handling dosing events of various routes and flexible dosing schedules
- Capacity to fit multiple endpoints simultaneously

Model building, fitting, and diagnostics process in PMXStan for a population PKPD model

```
ode <- "
   C2 = centr/V;
   d/dt(depot) =-ka*depot;
   d/dt(centr) = ka*depot - ke*centr;
   d/dt(eff) = (1+Emax*C2/(C2+EC50))*Kin -
Kout*eff;
"
instant.stan.extension(ode)

m5 <- PMXStanModel(type = "PKPD",
      path = "pkpd_m5", ode = ode,
      theta= c("Emax","EC50",
               "Kin","Kout"),
      eta = c("Emax","Kout"),
      fixed = c(V=1, ka=0.5, ke=0.4),
      obs.state = 3)
compile(m5)
```

```
dat5 <- prepareInputData(data.file =
        "./datasets/pkpd_abc123.csv",
        model = m5, inits = "BSL",
        covar = c("BMK1","BMK2"))

f5 <- PMXStanFit(m5, dat5,
      iter = 500, chains = 4)

print(f5, on.screen = F)
save(m5, dat5, f5,
     file = file.path("pkpd_m5",
     "ModelFit.RData"))

traces(f5)
waic(f5)
gofplot(f5)

# additional diagnostics by covariates ...
```

Model specifications according to target model type

Specification	Specification Variable	Options
Common for both model types		
Model type	type	**PK** | PKPD
File path for the model	path	*Input path to store model and fitting result*
Drug administration	route	**1st_order_abs** | IV_bolus | IV_infusion
Compile with specification	compile	TRUE | **FALSE**
For PK models only		
PK model structure	pk.struct	1-cmpt | **2-cmpt** | 3-cmpt
PK model parameterization	pk.param	**CL_V** | micro_rate
PK model solver	solver	**closed_form** | ODE
For PKPD models only		
Index of observed state variable	obs.state	*An integer*
Parameters to be estimated	theta	*Choose from parameter list*
Between-subject random effects	eta	*Choose from the variable theta*
Parameters not to be estimated	fixed	*Input values of constant parameters*
Specification of ODE system	ode	*Input string of ODEs*

Stan codes are automatically and dynamically generated from model specifications

```
> m <- PMXStanModel(path = "pk_abc123",
                    route = "IV_infusion",
                    compile = T
                   )
Model type is set to "PK" by default.
PK model struct is set to "2-cmpt" by default.
Solver for the PK model is set to "closed_form" by
default.
PK model parameterization is set to "CL_V" by default.
```

A model-specific Stan source code named "*popPK_2cmpt_ivinfs_clearance_cls.stan*" is then generated under the given directory "*pk_abc123*".

```
data{
    int<lower=0> NSUB;
    int<lower=0> NOBS[NSUB];
    int<lower=0> NDOSE[NSUB];
    vector[sum(NOBS)] conc;
    ...
}
parameters{
    vector<lower=-5.0, upper=5.0>[4] theta;
...
}
transformed parameters{
    ...
    {   ...
        for(i in 1:NSUB){
            ...
            g <- linear_cmpt_iv_infusion(...);
            ...
        }
    }
}
model{
    for(k in 1:4){
        ...
        theta[k] ~ normal(0.,1000.);
        sigma_eta[k] ~ normal(0.,1000.);
    }
    sigma ~ normal(0.,1000.);
    conc ~ normal(y_pred, sigma);
}
...
```

Input data structures required by Stan are automatically prepared from existing NONMEM datasets

	A	B	C	D	E	F	G
1	ID	CMT	DV	EVID	AMT	TIME	RATE
2	1	2	0	1	4.02	0	2.01
3	1	2	0.74	0	0	0	0
4	1	2	2.84	0	0	0.25	0
5	1	2	6.57	0	0	0.57	0
6	1	2	10.5	0	0	1.12	0
7	1	2	9.66	0	0	2.02	0
8	1	2	8.58	0	0	3.82	0
9	1	2	8.36	0	0	5.1	0
10	1	2	7.47	0	0	7.03	0
11	1	2	6.89	0	0	9.05	0
12	1	2	5.94	0	0	12.12	0
13	1	2	3.28	0	0	24.37	0
14	2	2	0	1	4.4	0	2.2
15	2	2	0	0	0	0	0
16	2	2	1.72	0	0	0.27	0
17	2	2	7.91	0	0	0.52	0
18	2	2	8.31	0	0	1	0
19	2	2	8.33	0	0	1.92	0
20	2	2	6.85	0	0	3.5	0

```
data{
    int<lower=0> NSUB;
    int<lower=0> NOBS[NSUB];
    int<lower=0> NDOSE[NSUB];
    vector[sum(NOBS)] conc;
    vector<lower=0>[sum(NOBS)] obs_time;
    vector<lower=0>[sum(NDOSE)] dose_time;
    vector<lower=0>[sum(NDOSE)] dose_amt;
    vector<lower=0>[sum(NDOSE)] inf_time;
}
```

```
> dat
$NSUB
[1] 12

$NOBS
 [1] 11 11 11 11 11 11 11 11 11 11 11 11

$obs_time
  [1]  0.00  0.25  0.57  1.12  2.02  3.82  5.10  7.03  9.05 12.12
 [11] 24.37  0.00  0.27  0.52  1.00  1.92  3.50  5.02  7.03  9.00
 [21] 12.00  24.30 0.00  0.27  0.58  1.02  2.02  3.62  ...

$conc
  [1]  0.74  2.84  6.57 10.50  9.66  8.58  8.36 7.47  6.89  5.94
 [11]  3.28  0.00  1.72  7.91 8.31  8.33  6.85  6.08  5.40  4.55
 [21]  3.01 0.90  0.00  4.40  6.90  8.20  7.80  7.50  ...

$NDOSE
 [1] 1 1 1 1 1 1 1 1 1 1 1 1

$dose_amt
 [1] 4.02 4.40 4.53 4.40 5.86 4.00 4.95 4.53 3.10 5.50 4.92 5.30

$dose_time
 [1] 0 0 0 0 0 0 0 0 0 0 0 0

$inf_time
 [1] 2 2 2 2 2 2 2 2 2 2 2 2
```

Model fitting and sampling process is tracked and reported by Stan

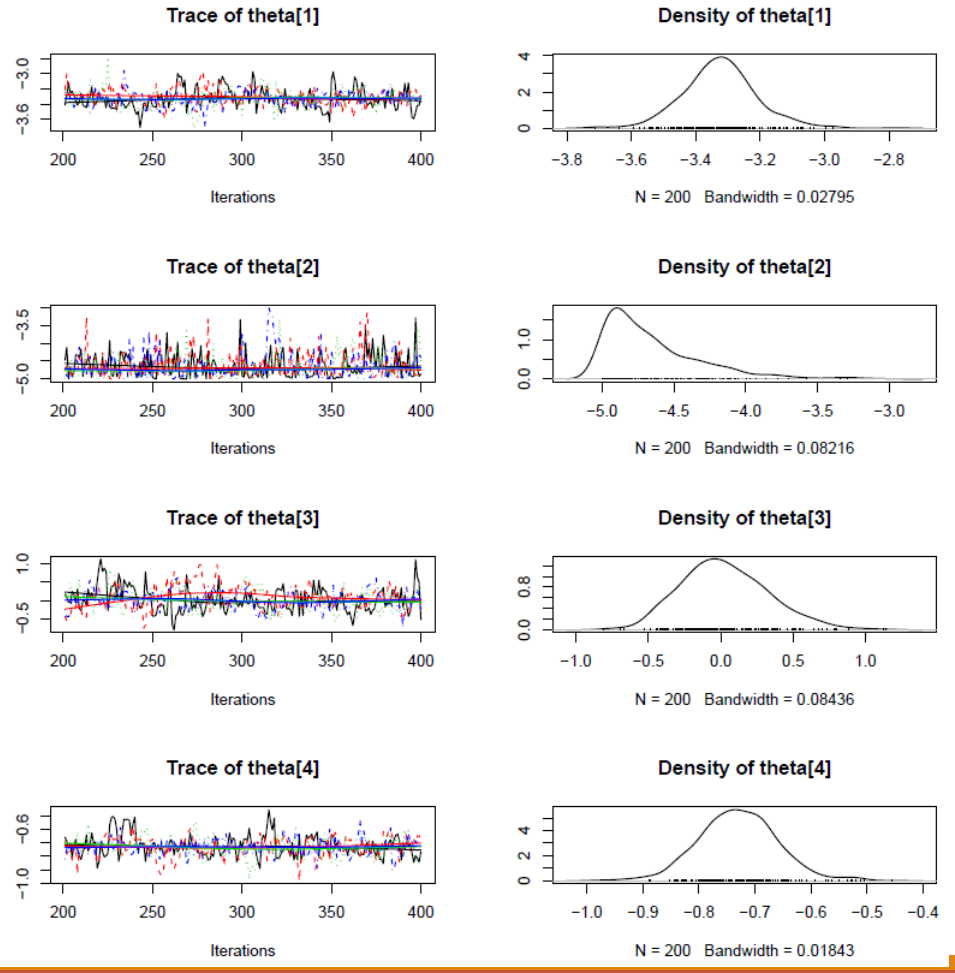
```
fit <- PMXStanFit(m, dat, iter = 400, chains = 4)
```

- Implicitly invoked: `.fit = sampling(.stanmodel, data = .standata, ...)`

```
SAMPLING FOR MODEL 'popPK_2cmpt_ivinfs_clearance_cls' NOW (CHAIN 1).

Chain 1, Iteration:   1 / 400 [  0%]  (Warmup)
Chain 1, Iteration:  40 / 400 [ 10%]  (Warmup)
Chain 1, Iteration:  80 / 400 [ 20%]  (Warmup)
Chain 1, Iteration: 120 / 400 [ 30%]  (Warmup)
Chain 1, Iteration: 160 / 400 [ 40%]  (Warmup)
Chain 1, Iteration: 200 / 400 [ 50%]  (Warmup)
Chain 1, Iteration: 201 / 400 [ 50%]  (Sampling)
Chain 1, Iteration: 240 / 400 [ 60%]  (Sampling)
Chain 1, Iteration: 280 / 400 [ 70%]  (Sampling)
Chain 1, Iteration: 320 / 400 [ 80%]  (Sampling)
Chain 1, Iteration: 360 / 400 [ 90%]  (Sampling)
Chain 1, Iteration: 400 / 400 [100%]  (Sampling)#

#  Elapsed Time: 7.13 seconds (Warm-up)
#                1.448 seconds (Sampling)
#                8.578 seconds (Total)
#
.....
```

Convergence of the MCMC chains and distributions of the posterior samples can be checked

```
> traces(fit)
```



Bayesian model specific diagnostics are conveniently provided for model comparison and selection

Currently implemented

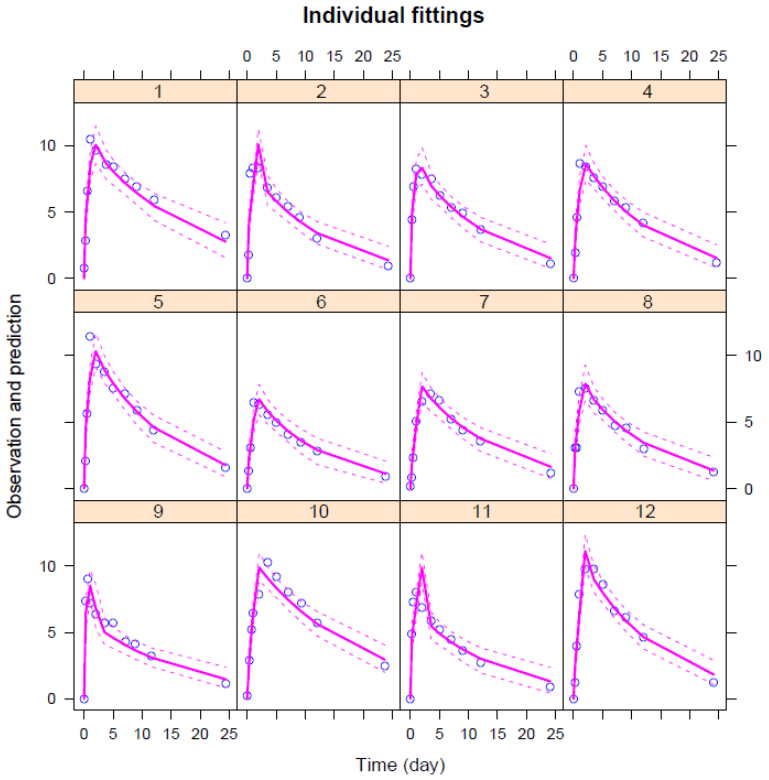
- Watanabe-Akaike Information Criterion (WAIC)
- Leave-one-out cross-validation (LOO-CV)

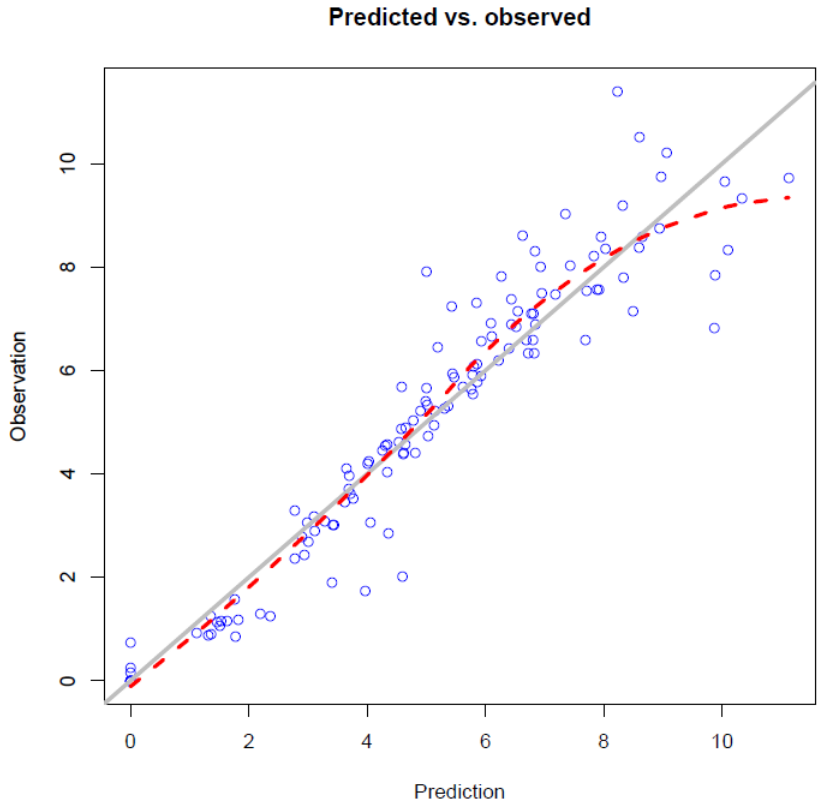
```
> waic(fit)

$total
      waic        lpd     p_waic  elpd_waic      p_loo   elpd_loo
 419.01256 -173.52813   35.97815 -209.50628   36.74860 -210.27673

$se
     waic       lpd    p_waic elpd_waic     p_loo  elpd_loo
28.349963  7.569561  6.693495 14.174982  6.461927 13.939331
```

Commonly used overall goodness-of-fit plots are automatically generated

```
> gofplot(fit)
```

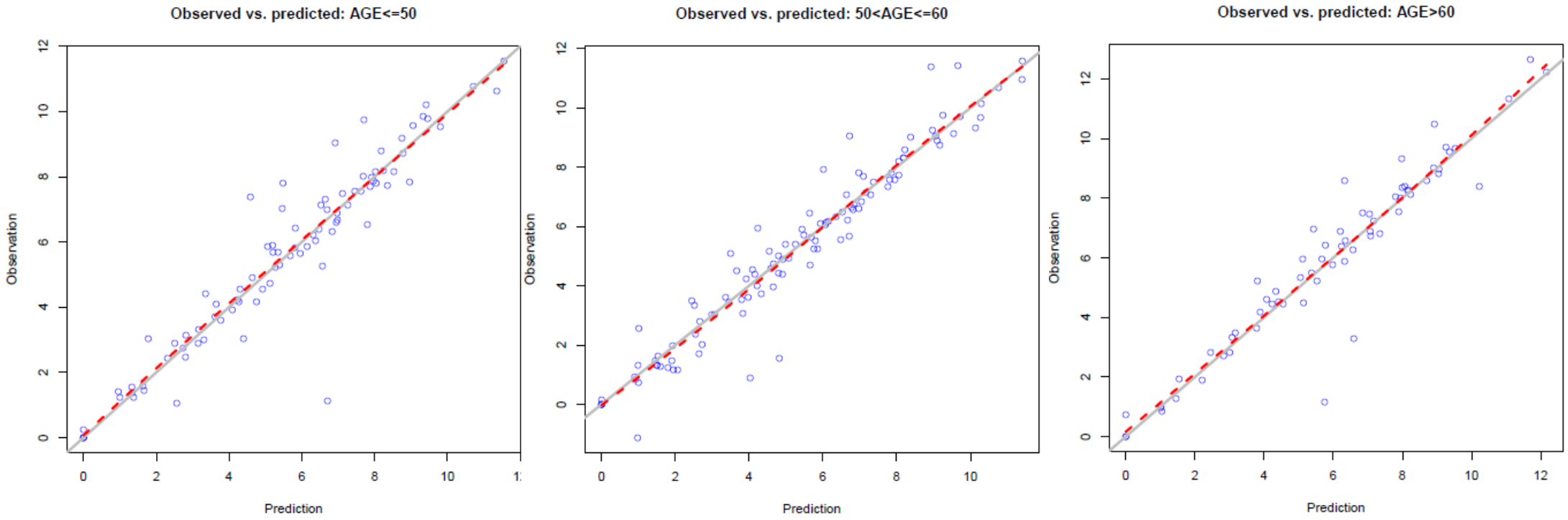
- Currently implemented: individual fittings, prediction vs. observation, residual plot





Some types of subgroup analysis by covariates are also implemented

```
> obs.vs.pred(fit, by.cov = "AGE", type = "continuous", cutoff = c(50, 60),
              filename = "obs_pred_by_age.pdf")
```



Building a two-compartment population PK model in Stan: a contrast before and after using PMXStan

Before: requiring rather long Stan coding for variables, parameters, and model specifications, followed by non-trivial R codes for post processing

- Model building in Stan: ~90 lines
- Transforming input data for Stan: ~50 lines when reading from a well-prepared NONMEM dataset
- Traces checks and basic GoF plots: ~20 lines

Advantages

- Frees users from intimidating coding that are NOT commonly used in PMX, thus allows them to focus more on model building
- Minimizes the possibility of errors and facilitates practical Bayesian PKPD modeling practice
- Fully extensible and customizable by providing everything conveniently accessible by users

After: modulized PMX-friendly interface with minimal coding in R (~10 lines)

```
library(PMXStan)

m <- PMXStanModel(path = "pk_m1", pk.struct = "1-cmpt", compile = T)

dat <- prepareInputData(data.file = "./datasets/poppk_1orderabs_theo.csv", model = m1, covar = c("AGE","GENDER"))

fit <- PMXStanFit(m1, dat, iter=500, chains=4)

traces(fit)

waic(fit)

gofplot(fit)

obs.vs.pred(fit, by.cov = "AGE", type = "continuous", cutoff = c(50, 60), filename = "obs_pred_by_age.pdf")

rsd.vs.pred(fit, by.cov = "GENDER", type = "categorical", filename = "rsd_pred_by_gender.pdf")
```

Summary and future work

Currently implemented: individual and population models

- PK models
  - 1/2/3-compartment
  - Drug administration: 1st-order absorption, IV-bolus, IV-infusion
  - Parameterization: clearance-volume, micro-rate constants
- Generic PKPD models that can be written in ODE forms

Work in progress within current capacity of PMXStan

- Fit with multiple endpoints
- Expand goodness-of-fit functionality
- Implement a simulation module
- Explore interface with ShinyStan

Future work

- Other forms of PK/PD models
- More general statistical models that are frequently used in PMX
- Whatever requested most by users!!

We would like to thank colleagues from the pharmacometrics community for their inspiring discussions and constant encouragement.

Also thanks to the Stan team for their helpful discussions and inputs.

Last but not least, we are grateful to the organizers of the ISoP Webinar Series for providing this opportunity of presenting, and helping all the way to make this event possible!

References

- The Stan Development Team. *Stan Modeling Language User's Guide and Reference Manual.* http://mc-stan.org
- Andrew Gelman's blog about our poster on ACoP6: http://andrewgelman.com/2015/10/05/pmxstan-an-r-package-to-facilitate-bayesian-pkpd-modeling-with-stan (Note that substantial improvements have been made since then)
- Sumio Watanabe. *Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory.* http://www.jmlr.org/papers/volume11/watanabe10a/watanabe10a.pdf.
- Aki Vehtari and Andrew Gelman. *WAIC and cross-validation in Stan.* http://www.stat.columbia.edu/~gelman/research/unpublished/waic_stan.pdf

Contact information

Yuan Xiong*: yxiong@wequantify.com*

Wenping Wang*: wenping.wang@novartis.com*