dMod – dynamical modeling in R

ICSB2022 Satelite Event

Severin Bang

University of Freiburg

October 7, 2022

- What
- How
- How practically
- Why
- Where

What

- **d**ynamical **Mod**eling
- Framework/Toolbox for ODE modeling in R
- Developed by Daniel Kaschek[1]

[1] D. Kaschek et al., J. Stat. Softw. **88**, 10.18637/jss.v088.i10 (2019)

How

- Symbolic equations
- Define observation function: connect model parameters to observed quantities
- Translate into system of ODEs, compile as C code
- Transformation: (condition dependent) definition of parameters
- Optional: add prior to objective function
- Multi start trust-region fit utilizing gradient and Hessian
- Prediction function composed of modular functions
- Profile likelihood to calculate confident intervals

How practically

1. Get data:

	time	name	value	condition
1	0.00000	B_obs	1.0430899	reference
2	10.34483	B_obs	1.8257833	reference
3	20.68966	B_obs	1.8324904	reference
4	31.03448	B_obs	1.7451547	reference
5	41.37931	B_obs	1.7168632	reference

How practically

1. Get data:

	time	name	value	condition
1	0.00000	B_obs	1.0430899	reference
2	10.34483	B_obs	1.8257833	reference
3	20.68966	B_obs	1.8324904	reference
4	31.03448	B_obs	1.7451547	reference
5	41.37931	B_obs	1.7168632	reference

2. Build `equation list` EL:

```
el <- NULL
el <- addReaction(el, from = "A", to = "B", rate = "k_AB * A", description = "A to B")
el <- addReaction(el, from = "B", to = "C", rate = "k_BC * B", description = "B to C")
```

How practically

1. Get data:

	time	name	value	condition
1	0.00000	B_obs	1.0430899	reference
2	10.34483	B_obs	1.8257833	reference
3	20.68966	B_obs	1.8324904	reference
4	31.03448	B_obs	1.7451547	reference
5	41.37931	B_obs	1.7168632	reference

2. Build `equation list` EL:

```
el <- NULL
el <- addReaction(el, from = "A", to = "B", rate = "k_AB * A", description = "A to B")
el <- addReaction(el, from = "B", to = "C", rate = "k_BC * B", description = "B to C")
```

3. define `observables` and `error model`:

```
observables <- eqnvec(B_obs = "scale_B * B + offset_B")
errors <- c(B_obs = "sigma_B_obs")
```

How practically

4. Compile Models:

```
model <- odemodel(f = el) #ode model
x <- Xs(odemodel = model) #compiled ode model
g <- Y(g = observables, f = el) #observation function
e <- Y(g = errors, f = c(as.eqnvec(el), observables), states = names(observables)) #error model
```

How practically

4. Compile Models:

```
model <- odemodel(f = el) #ode model
x <- Xs(odemodel = model) #compiled ode model
g <- Y(g = observables, f = el) #observation function
e <- Y(g = errors, f = c(as.eqnvec(el), observables), states = names(observables)) #error model
```

5. Set up `transformation`:

```
> trafo
Idx       Inner <- Outer
  1           A <- A
  2           B <- B
  3           C <- C
  4        k_AB <- k_AB
  5        k_BC <- k_BC
  7    offset_B <- offset_B
  6     scale_B <- scale_B
  8 sigma_B_obs <- sigma_B_obs
```

How practically

4. Compile Models:

```
model <- odemodel(f = el) #ode model
x <- Xs(odemodel = model) #compiled ode model
g <- Y(g = observables, f = el) #observation function
e <- Y(g = errors, f = c(as.eqnvec(el), observables), states = names(observables)) #error model
```

5. Set up `transformation`:

```
> trafo
Idx       Inner <- Outer
  1           A <- A
  2           B <- B
  3           C <- C
  4        k_AB <- k_AB
  5        k_BC <- k_BC
  7    offset_B <- offset_B
  6     scale_B <- scale_B
  8 sigma_B_obs <- sigma_B_obs
```

6. Cond. depending paramters, scales, inits:

```
> trafoList
$reference
Idx       Inner <- Outer
  1           A <- 10^(A)
  2           B <- 0
  3           C <- 0
  4        k_AB <- 10^(k_AB)
  5        k_BC <- 10^(k_BC_reference)
  7    offset_B <- 10^(offset_B)
  6     scale_B <- 10^(scale_B)
  8 sigma_B_obs <- 10^(sigma_B_obs)

$treatment
Idx       Inner <- Outer
  1           A <- 10^(A)
  2           B <- 0
  3           C <- 0
  4        k_AB <- 10^(k_AB)
  5        k_BC <- 10^(k_BC_treatment)
  7    offset_B <- 10^(offset_B)
  6     scale_B <- 10^(scale_B)
  8 sigma_B_obs <- 10^(sigma_B_obs)
```

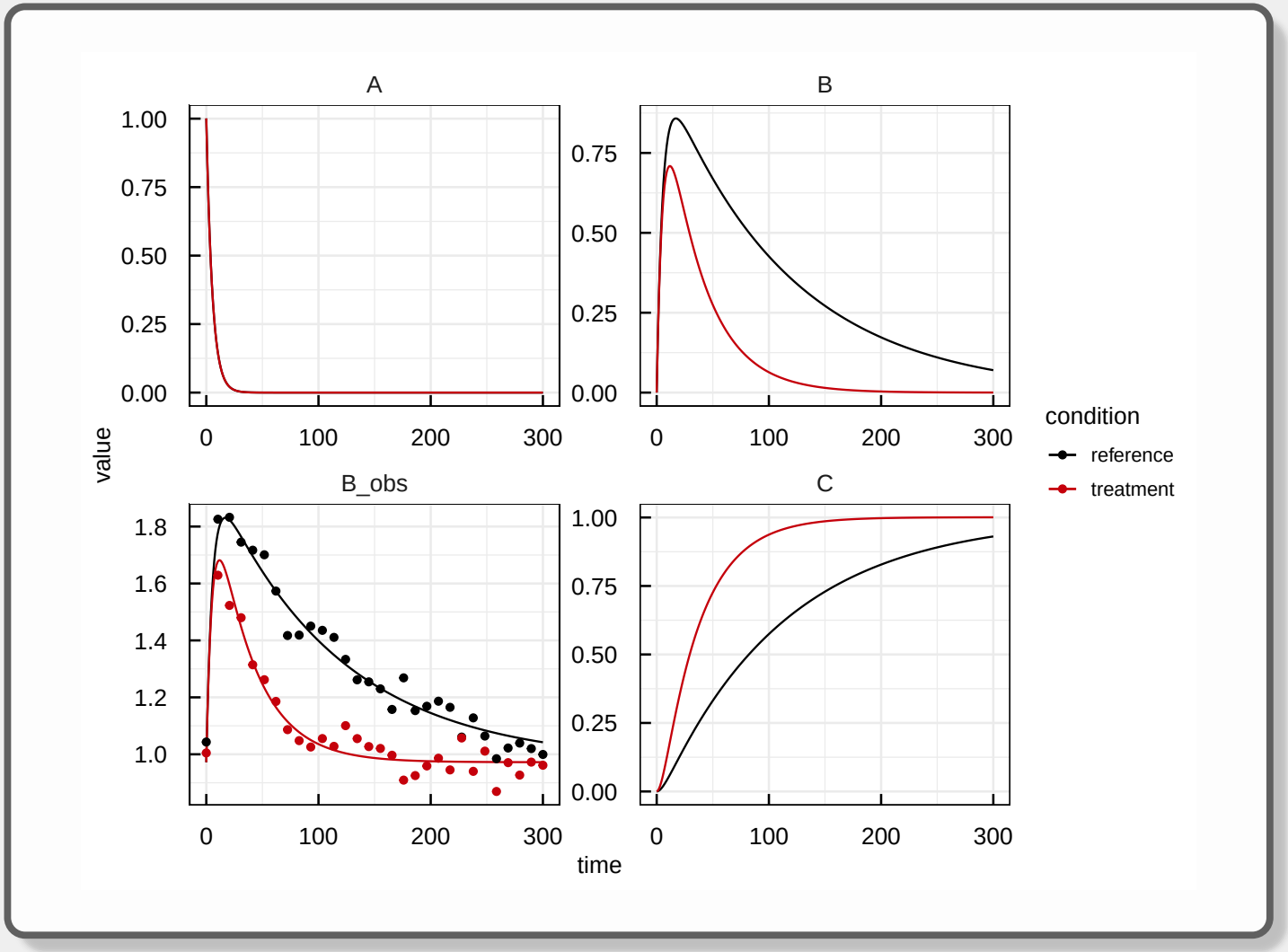
How practically

7. Fit:

```
fitOutput <- mstrust(
  objfun=obj,
  center=dMod::msParframe(paramOuter, n = 20, seed=1),
  studyname="fits",
  fits = 20,
  cores = 1
)
```

How practically

7. Fit:



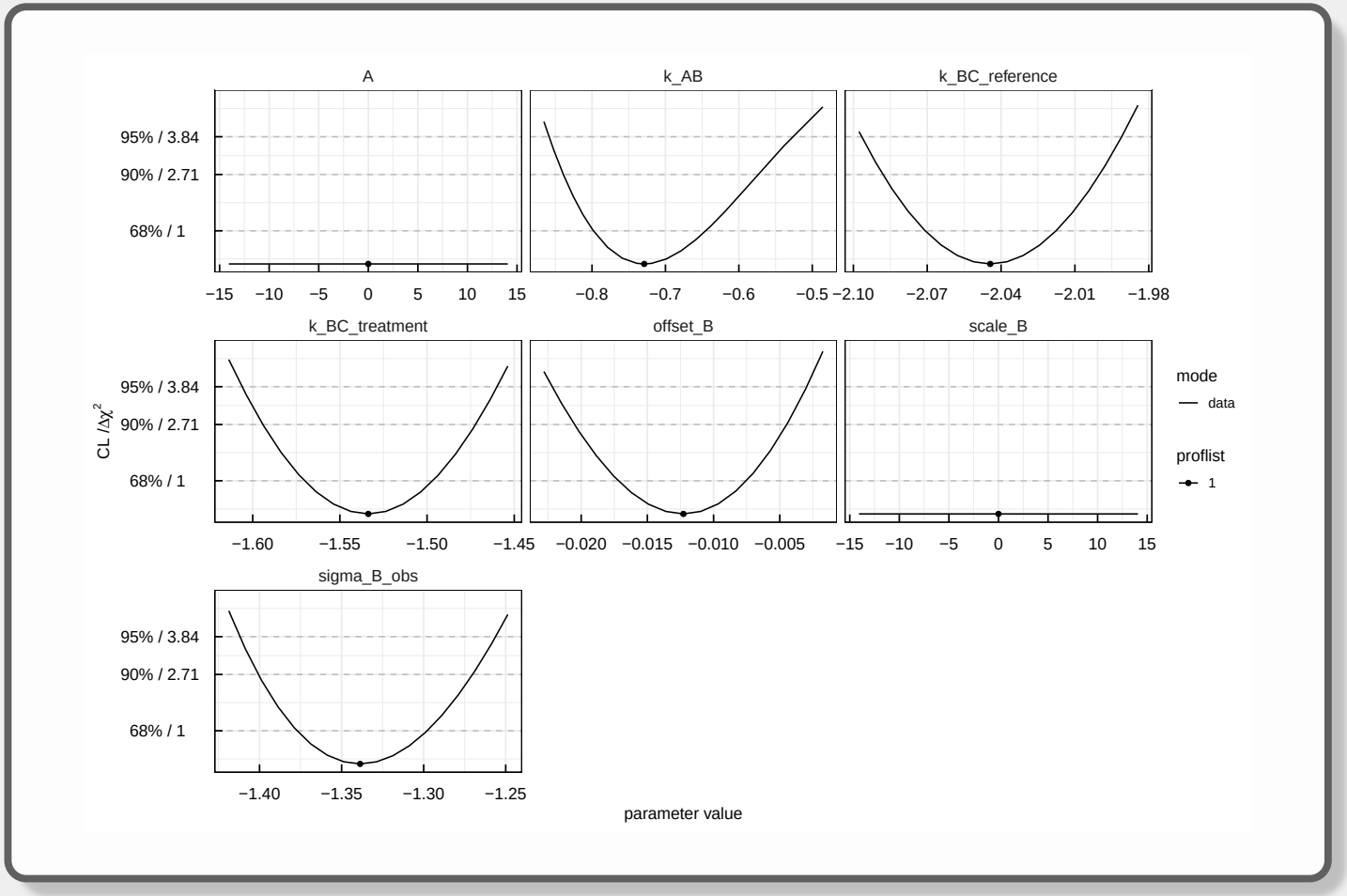
How practically

8. Profiles:

```
profiles <- profile(
  obj = obj,
  pars =  bestFit,
  whichPar = names(paramOuter),
  cores = 1,
  method = "optimize",
  optControl = list(iterlim = 50)
)
```

How practically

8. Profiles:



Why

- Open source
- Modular
  - Prediction function build by linking model, observation and transformation
  - Objective function with ”organically” added prior
- Transformation function: focus on handling different treatment/conditions
- `slurm`–ready
- Reads and speaks PEtab (under development)

Where

- dMod on CRAN:
- dMod on github:
- example model:

- Recap:
  - Open source model framework in R
  - Optimization based on gradient and hessian
  - Optional prior for obj. function
  - CI from profile likelihood
  - PEtab integration
- Paper:
- Further questions?
  severin.bang@physik.uni-freiburg.de