AstraZeneca

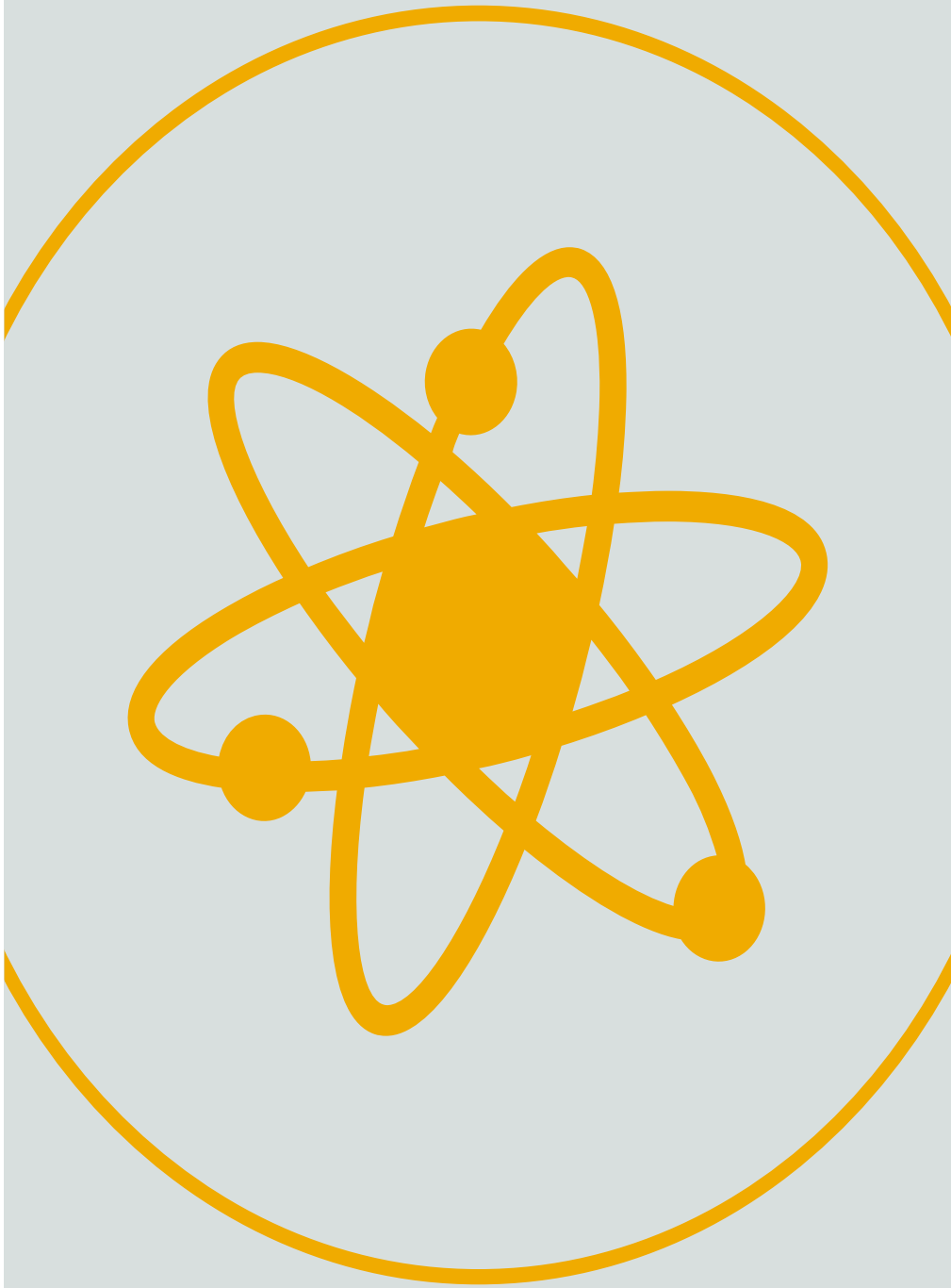
Unlocking R's Core Strengths: A Fresh Take on Statistical Programming in Clinical Trials

Kamil Kmita

Senior Statistical Programmer, R&I Biometrics

PHUSE EMEA SDE – Frankfurt

29.04.2026



Agenda

1. Warm-up
2. SAS vs R: a key difference
3. Example 1 (simple)
4. Example 2 (complex)
   a) SAS implementation
   b) R implementation
5. Conclusions
6. *Appendix (if time allows)

- This work was funded by AstraZeneca.
- Kamil Kmita is an employee of AstraZeneca.

1. Warm-up

“

We will introduce R’s features that allow different, intuitive ways of processing statistical information that SAS’s structured procedures can’t match. This isn't just a matter of preference; **it transforms how programmers can express ideas (…)**

”

Abstract of this presentation

SAS and R as calculators

SAS

```
%put %sysevalf(2 + 2);
```

4 *(in the log)*

```
data _null_;
x = put(2 + 2, 1.);
put x;
run;
```

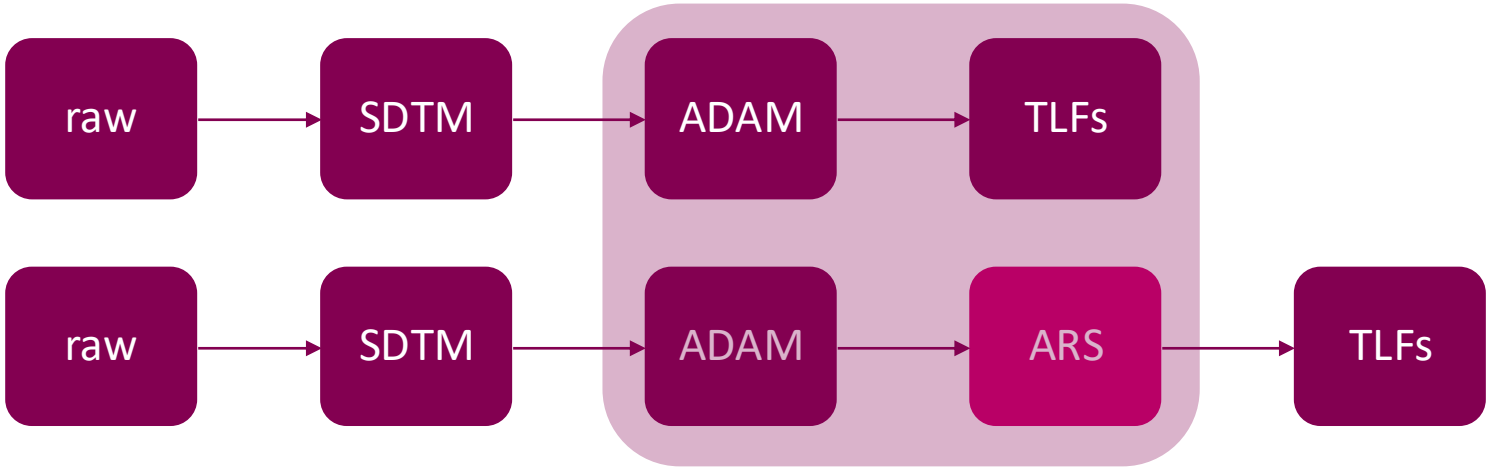
4 *(in the log)*

R

```
> 2 + 2
```

```
[1] 4
```

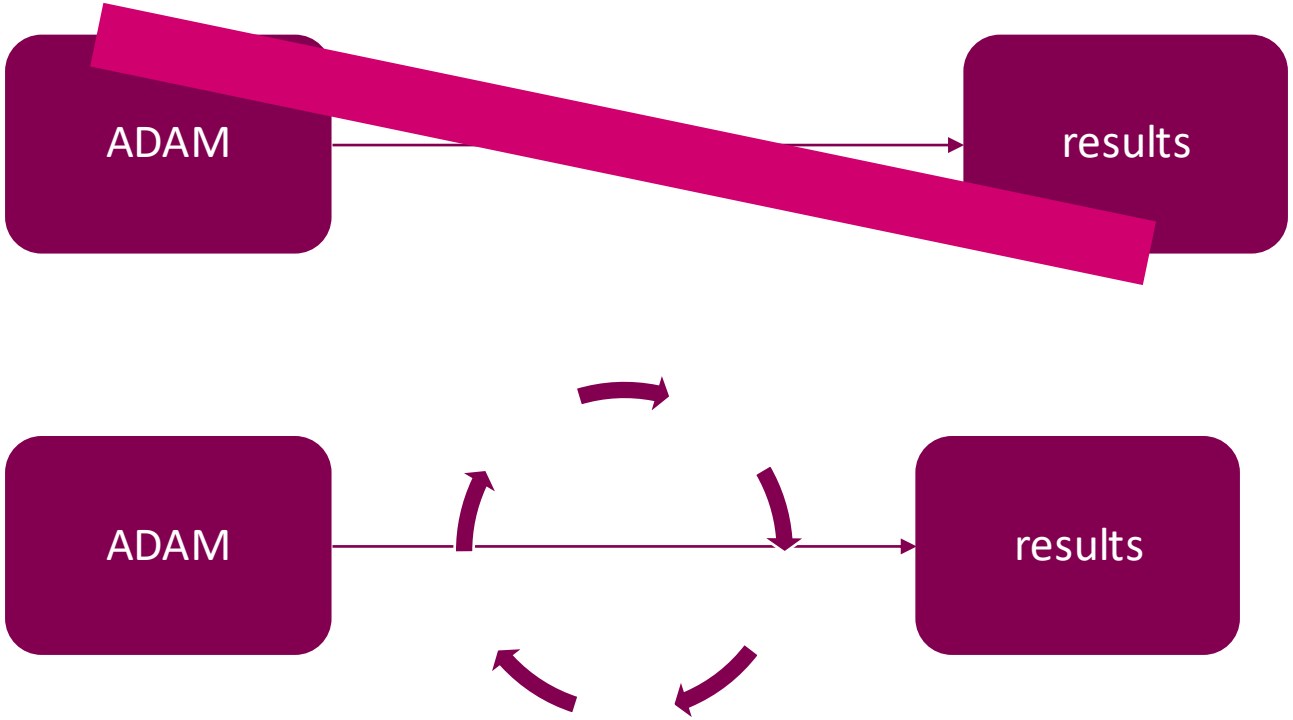

Scope: creating statistical results from ADAM



- We focus on creating **statistical** results, i.e., model-based, from ADAM
- In SAS, predefined procedures (`PROC REG` and `PROC MIXED` in this presentation)
- In R, packages: `stats::lm` shipped with R and open-source community-driven `mmrm::mmrm`

This presentation explores the real edge R holds over SAS in statistical programming – its core language features, an aspect often overlooked in favor of its packaging protocol, open-source ecosystem and visualization capabilities.

The process is not linear, it is adaptive



- Introducing new data patterns
- Changing modelling from additive changes to multiplicative
- Introducing subgroups
- Or “simply” programming from scratch a complicated procedure (tipping point, bootstrap, etc.)

2. SAS vs R: a key difference

Overview of Base SAS Software (from SAS® 9.4 Language Reference: Concepts, Sixth Edition)

"The core of SAS is Base SAS software, which consists of the following:

- **DATA step**: a programming language that you use to manipulate and manage your data.
- **SAS procedures**: software tools for data analysis and reporting. A type of SAS language element that refers to *a self-contained program for performing a specific task*, such as to produce reports, to manage files, or to analyze data.
- **macro facility**: a tool for extending and customizing SAS software programs and for reducing text in your programs."
- …

Overview of R (from https://www.r-project.org/about.html)

- “R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language (…)
- **R**, like S, **is designed around a true computer language**, and it allows users to add additional functionality by defining new functions.”

Comparison[a]

SAS

- Base SAS language is primarily a procedural, domain-specific language for data step and PROC-based analytics
- DATA step: to manipulate datasets
- PROC: to analyze data

R

- **R**, like S, **is designed around a true computer language**
- Thus, R language has native support for object-oriented programming

a: based on references from the previous slide

SAS vs R:
a key difference is the support of object-oriented programming (OOP) paradigm

Object-oriented programming – the shortest intro

"Object-oriented programming (OOP) is a programming paradigm based on objects – software entities that encapsulate data [attributes] and function(s) [methods]"[a].

Definition of a class `Person` in R

```
Person <- function(name) {
  obj <- list(name = name)   # attribute `name`
  class(obj) <- "Person"
  return(obj)
}

say_hello <- function(x) UseMethod("say_hello")

say_hello.Person <- function(x) {
  cat(sprintf("My name's %s\n", x$name))
}
```

Creating an instance of a class `Person` - a concrete entity, and invoking the method `say_hello`

```
alice = Person("Alice")
say_hello(alice)
# Output: My name's Alice
```


a: https://en.wikipedia.org/wiki/Object-oriented_programming

Comparison

SAS	R

• Model is a PROCEDURE
• E.g. *"the MIXED procedure fits a variety of mixed linear models to data and enables you to use these fitted models to make statistical inferences about the data."*[a]
• All SAS procedures produce ***output objects*** that ODS delivers to various ***ODS destinations***
• OUTPUT destination, for saving results to SAS data sets
	• Model is an object
	`> fit <- lm(y ~ x)`
	`> class(fit)`
	`[1] "lm"`
	• Model has attributes
	`> names(fit)`
	`[1] "coefficients" "residuals" …`
	• Model has methods
	`> methods(class = "lm")`
	`[1] add1 alias anova case.names …`

a: https://documentation.sas.com/doc/en/statug/latest/statug_mixed_overview.htm

3. Example 1 – simple

extracting estimated coefficient from linear regression

SAS

```
/* recreate dataset `df` based on `x, y`*/

ods output
  ParameterEstimates = estimates;

proc reg data = df;
model y = x;
run;
```

Model	Dependent	Variable	DF	Estimate	StdErr	tValue
MODEL1	y	Intercept	1	-0.9890	0.33564	-0.29
MODEL1	y	x	1	0.17013	0.46402	0.37

```
data _null_;
set estimates;
where upcase(VARIABLE) eq “X”;
call symput(“xbeta”, put(estimate, best.));
run;

%put [&xbeta.];
[0.1701271681]
```

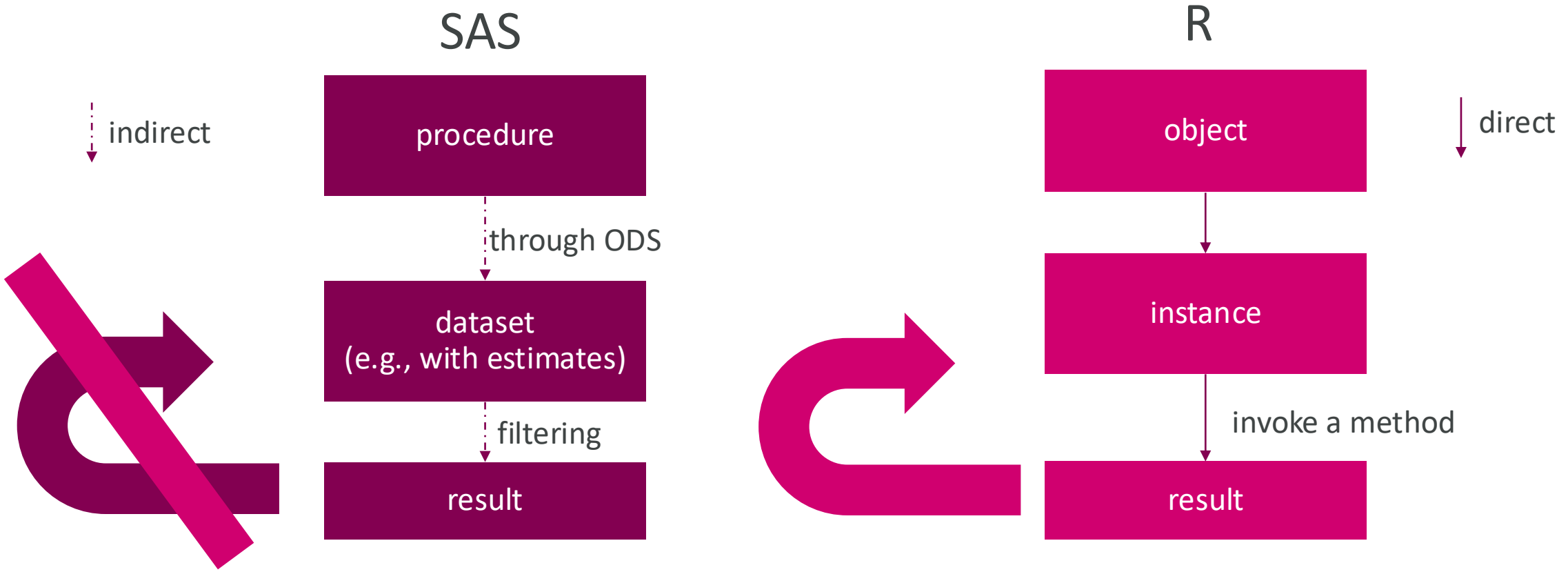
R

```
> x <- rnorm(10); y <- rnorm(10);
> fit <- lm(y ~ x)

> coefficients(fit)
(Intercept)           x
-0.0989         0.1701

> (xbeta <- coefficients(fit)[2])
        x
0.1701272
```

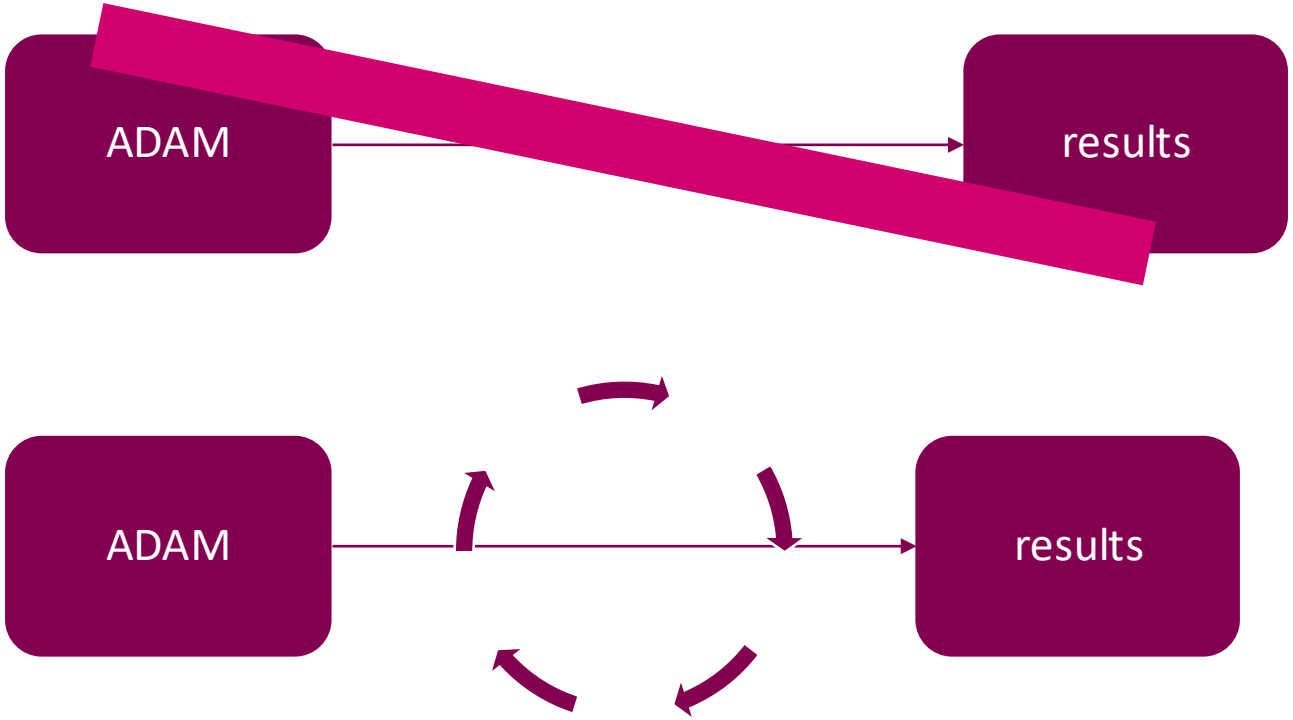
The true issue: traceability & reusability



- The “model” is gone (since it’s a procedure that generates output based on input and finishes) – you only have datasets that you stored

- Want to inspect the model further? You can!
- Want to write some custom method, change the results? You can do it directly based on the **instance** of a model

Recall: the ADAM -> results process is not linear, it is adaptive



Limited traceability and reusability hurts!

4. Example 2 - complex

Scenario

- Mixed models for repeated measures analyze continuous outcomes collected from the same participants at multiple, discrete time points
- In clinical trials, repeated measures arise naturally - for example, measuring FEV1 in 100 participants across 5 visits, with 50 randomized to investigational product and 50 to placebo
- The fixed effects are typically the primary focus - especially the treatment effect (difference between investigational product and placebo over time)
- The within-subject correlation structure (how repeated measurements from the same participant relate to each other) influences estimates and inference, even though it's not the target of inference
- **Pre-specify a sequence of covariance structures** in the Statistical Analysis Plan; start with the preferred structure, and if the model doesn't converge, proceed to the next pre-specified alternative

SAS

```
%macro run_mixed(ctype);
  ods select none;
  ods output SolutionF = estimated_coefs;  /*need to prespecify ODS Table Name*/

  proc mixed data = have method = reml;
  class PTS TRT;
  model FEV1 = TRT TIME / solution;
  repeated TIME / type = &ctype. Subject = PTS;
  run;

  ods select all;
%mend;

%run_mixed(UN);  /* if fails, try %run_mixed(CSH) */
```

R

```
run_mixed <- function(cv) {
  out <- mmrm(fev1 ~ trt + time, covariance = cov_struct(cv, "time", "pts"), data = have)
  return(out)
}

fit <- run_mixed("us")
estimated_coefs <- coefficients(fit)  # no need to prespecify
```


SAS: **proc mixed** implements mmrm

R: **mmrm** package implements mmrm, and provides dummy data to compare various approaches

Extension of the scenario (iterative nature)

- Recall: a model may fail to converge under certain covariance structure
- Current approach: we manually set the covariance option and then check convergence
  - UN/US: unstructured,
  - CSH: Heterogeneous Compound Symmetry

<u>The request</u>

- Implement an automatic check of the pre-defined covariance-structure sequence, fitting the model iteratively and reporting which structure(s) converged
- Run a second model that adds a new binary covariate (dummy), repeating the same automated sequence and providing a side-by-side summary of convergence

4. a) SAS implementation

SAS step 1: extend *run_mixed* macro

```
%macro run_mixed(covariates, ctypes = UN CSH);

  %do i=1 %to %sysfunc(countw(&ctypes.));
    %global iterType;
    %let iterType = %scan(&ctypes., &i.);

    ods output SolutionF = estimated_coefs convergenceStatus = convergence_status;

    proc mixed data = have method = reml;
    class PTS TRT;
    model CHG = &covariates. / solution;
    repeated TIME / type = &iterType. Subject = PTS;
    run;

    data _null_;    /*separate data step to get convergence to handle programmatically*/
    set convergence_status;
    call symputx("convergenceStatus", STATUS);
    run;

    %if convergenceStatus. eq 0 %then %goto %proceed;   /*break the loop*/
  %end;

%proceed:
%mend;
```


SAS step 2: store info about covariance structure

```
%macro store_covariance(id, ctype);

  data _summary_covariance;
  /*implement creation-updating of a single dataset approach*/
  run;

%mend;



%macro for_model(id, covariates);

  %run_mixed(&covariates.);
  %store_covariance(&id., &iterType.);  /*&iterType. is global from %run_mixed*/

%mend;



%for_model(id = 1, covariates = TRT TIME);
%for_model(id = 2, covariates = TRT TIME DUMMY);   /*include DUMMY covariate*/

proc print data = _summary_covariance;   /*report covariances used*/
```


Summary: SAS approach

- The link between the procedure's outputs and the information in the **_summary_covariance** dataset is indirect
- To revisit a model based on findings from **_summary_covariance**, you must reconstruct the procedure call with the appropriate covariance parameters and options
- This becomes non-trivial when working within nested (outer–inner) macro structures, where parameters propagate through multiple layers and increase the risk of mismatch

```
%let my_covariates = ...;
%let my_type = ...;

ods output SolutionF = estimated_coefs; /*possibly more outputs to be request*/

proc mixed data = have method = reml;
class PTS TRT;
model CHG = &my_covariates. / solution;
repeated TIME / type = &my_type. Subject = PTS;
run;
```


4. b) R implementation

R: step 1 extend *run_mixed* function

```
run_mixed <- function(.formula, cvs = c("us", "csh")) {
  for (cv in cvs) {
    fit <- try(
      mmrm(.formula, covariance = cov_struct(cv, "time", "pts"), data = have),
      silent = TRUE)

  if (!inherits(fit, "try-error")) { return(fit) }  # lack of convergence throws error
}

formulas <- list(
  formula(fev1 ~ trt + time),
  formula(fev1 ~ trt + time + dummy)
)

models <- lapply(formulas, run_mixed)   # vectorization: core feature of R
```


R: **no need to** store info about covariance structure

You can directly access the covariance type of a given **model**

```
> models[[1]]$formula_parts$cov_type
[1] “csh”
```

You can retrieve the formula, by invoking **`formula`** method

```
> formula(models[[1]])
fev1 ~ trt + time
```

The report is a simple **for** loop

```
> for (i in seq_along(models)) {
+   curm <- models[[i]]
+   covtype <- models[[i]]$formula_parts$cov_type
+   print(paste0(“[“, i, “] model: “, deparse(formula(curm)), “, cov: “, covtype}

[1] “[1] model: fev1 ~ trt + time, covariance: csh”
[2] “[2] model: fev1 ~ trt + time + dummy, covariance = csh”
```


Summary: R approach

- The link between the model's output and the information about covariance type used is direct
- No need to reconstruct the model call to revisit a model – you can work directly with the instance **model**
- You can create your own *generics* and *methods*[a]

```
> names(models[[1]])
  [1] "cov"          "beta_est"     "beta_vcov"     "beta_vcov_inv_L"   "beta_vconv_inv_D"
  [6] "theta_est"    "theta_vcov"   "neg_log_lik"   …

> methods(class = "mmrm")
[1] anova    augment    confint    glance    summary    tidy
```


a: see Appendix

5. Conclusions

- The real edge R holds over SAS in statistical programming is its core language features: R supports Object-Oriented Programming paradigm, enabling rich results with user-friendly displays and developer-friendly APIs.
- This isn't just a matter of preference; it transforms how programmers can express ideas and increases traceability and reusability of the programs.
- In our example, R retains all model components within a single object, preserving direct links among data, estimates, and covariance details. SAS manages these links indirectly, with relationships materialized through procedure outputs and multiple macros.
- In R, you can query the model immediately from the object itself. In SAS, reproducing the same view often requires reconstructing procedure state—using targeted macro calls to rebuild what’s needed in the workspace.

Thank you.

*Appendix

Recall

"Object-oriented programming (OOP) is a programming paradigm based on objects – software entities that encapsulate data [attributes] and function(s) [methods]"[a].

Definition of a class `Person` in R

```
Person <- function(name) {
  obj <- list(name = name)    # attribute `name`
  class(obj) <- "Person"
  return(obj)
}

say_hello <- function(x) UseMethod("say_hello")

say_hello.Person <- function(x) {
  cat(sprintf("My name's %s\n", x$name))
}
```

Creating an instance of a class `Person` - a concrete entity, and invoking the method `say_hello`

```
alice = Person("Alice")
say_hello(alice)
# Output: My name's Alice
```


a: https://en.wikipedia.org/wiki/Object-oriented_programming

S3 OOP system in R

"S3 classes provide a lightweight object-orientated programming (OOP) approach for automated dispatching calls to *generics* of the type **print**(y) to concrete *methods* such as **print.data.frame**(y), based on the *class* of the object they are invoked on."[a]

```
print.Person <- function(x) {    # we are working with a predefined generic `print`
  sprintf("This is custom printing of Person's class instance %s\n", x$name)
}

> print(alice)
[1] "This is custom printing of Person's class instance Alice\n"
```

```
> print(data.frame(x = c(1, 2), y = c("A", "B"))
   x  y
1 1 A
2 2 B
```


a: https://deepr.gagolewski.com/chapter/220-s3.html#object-type-vs-class

R: **no need to** store info about covariance structure

```
> models[[1]]$formula_parts$cov_type
[1] "csh"
```

A nicer way: create your own generic & method!

```
> class(models[[1]])
[1] "mmrm" "mmrm_fit" "mmrm_tmb"
```

Create a generic, and then create a method for "mmrm" class

```
covname <- function(x, ...) { UseMethod("covname") }   # register a generic method

covname.mmrm <- function(x, ...) {
  x$formula_parts$cov_type
}
```

Use the method on the models

```
> for (i in seq_along(models)) {
+   curm <- models[[i]]
+   print(paste0("[", i, "] model: ", deparse(formula(curm)), ", cov: ", covname(curm))}

[1] "[1] model: fev1 ~ trt + time, covariance: csh"
[2] "[2] model: fev1 ~ trt + time + dummy, covariance = csh"
```


AstraZeneca PLC, 1 Francis Crick Avenue, Cambridge Biomedical Campus, Cambridge, CB2 0AA, UK
+44(0)203 749 5000
www.astrazeneca.com