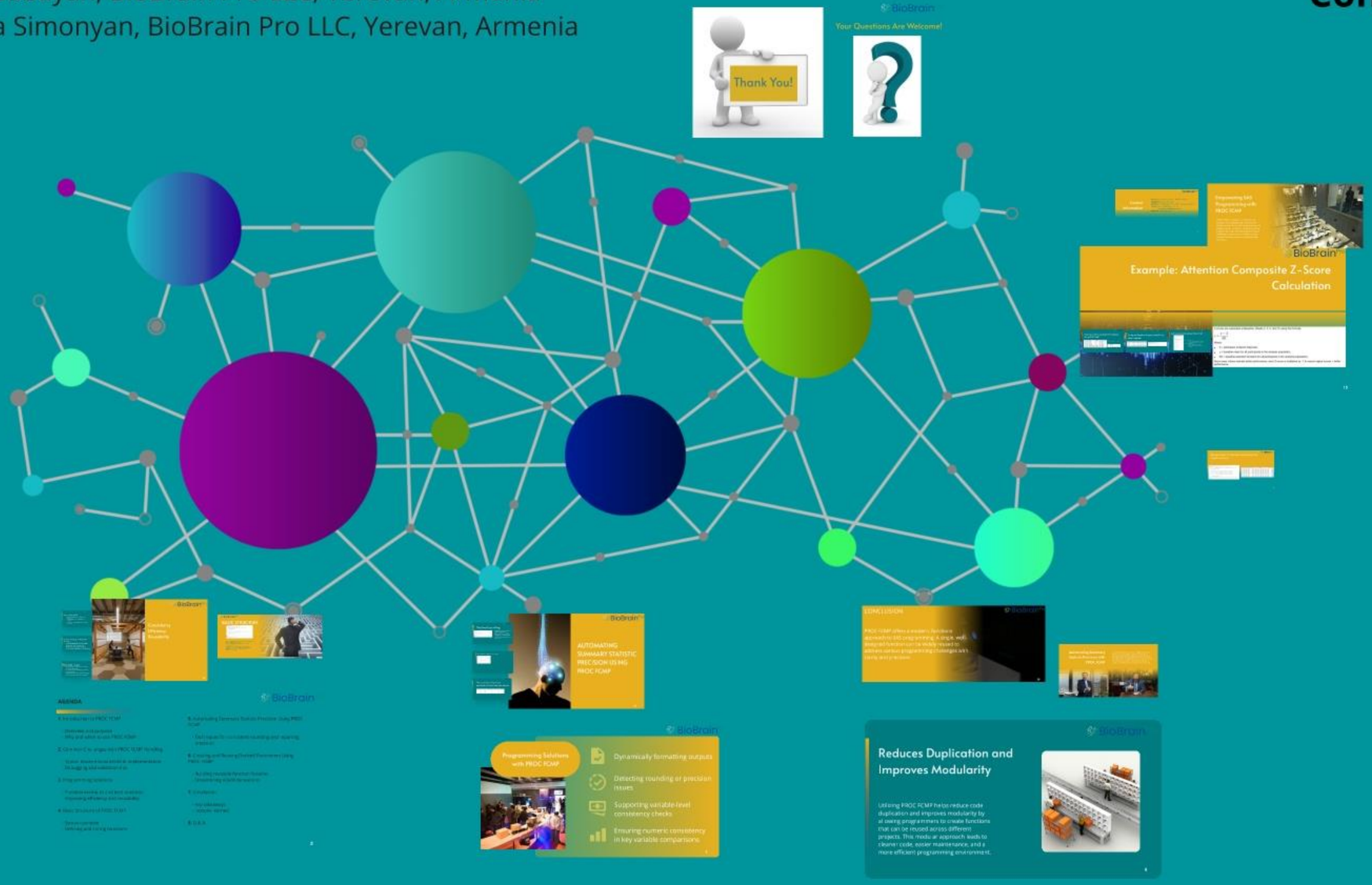
Paper CT15
Building Reusable Precision Logic in SAS Using PROC FCMP

Mariam Babayan, BioBrain Pro LLC, Yerevan, Armenia
Syuzanna Simonyan, BioBrain Pro LLC, Yerevan, Armenia

PHUSE EU Connect 2025



AGENDA

1. Introduction to PROC FCMP

- Overview and purpose
- Why and when to use PROC FCMP

2. Common Challenges with PROC FCMP Handling

- Typical issues encountered in implementation
- Debugging and validation tips

3. Programming Solutions

- Practical examples and best practices
- Improving efficiency and reusability

4. Basic Structure of PROC FCMP

- Syntax overview
- Defining and calling functions

5. Automating Summary Statistic Precision Using PROC FCMP

- Techniques for consistent rounding and reporting precision

6. Creating and Reusing Derived Parameters Using PROC FCMP

- Building reusable function libraries
- Streamlining ADaM derivations

7. Conclusion

- Key takeaways
- Lessons learned

8. Q & A

BioBrain

What Is PROC FCMP?

- Enables creation of custom SAS functions & subroutines
- Integrates logic directly into DATA steps and procedures
- Works across multiple SAS programs — no macro calls needed

BioBrain

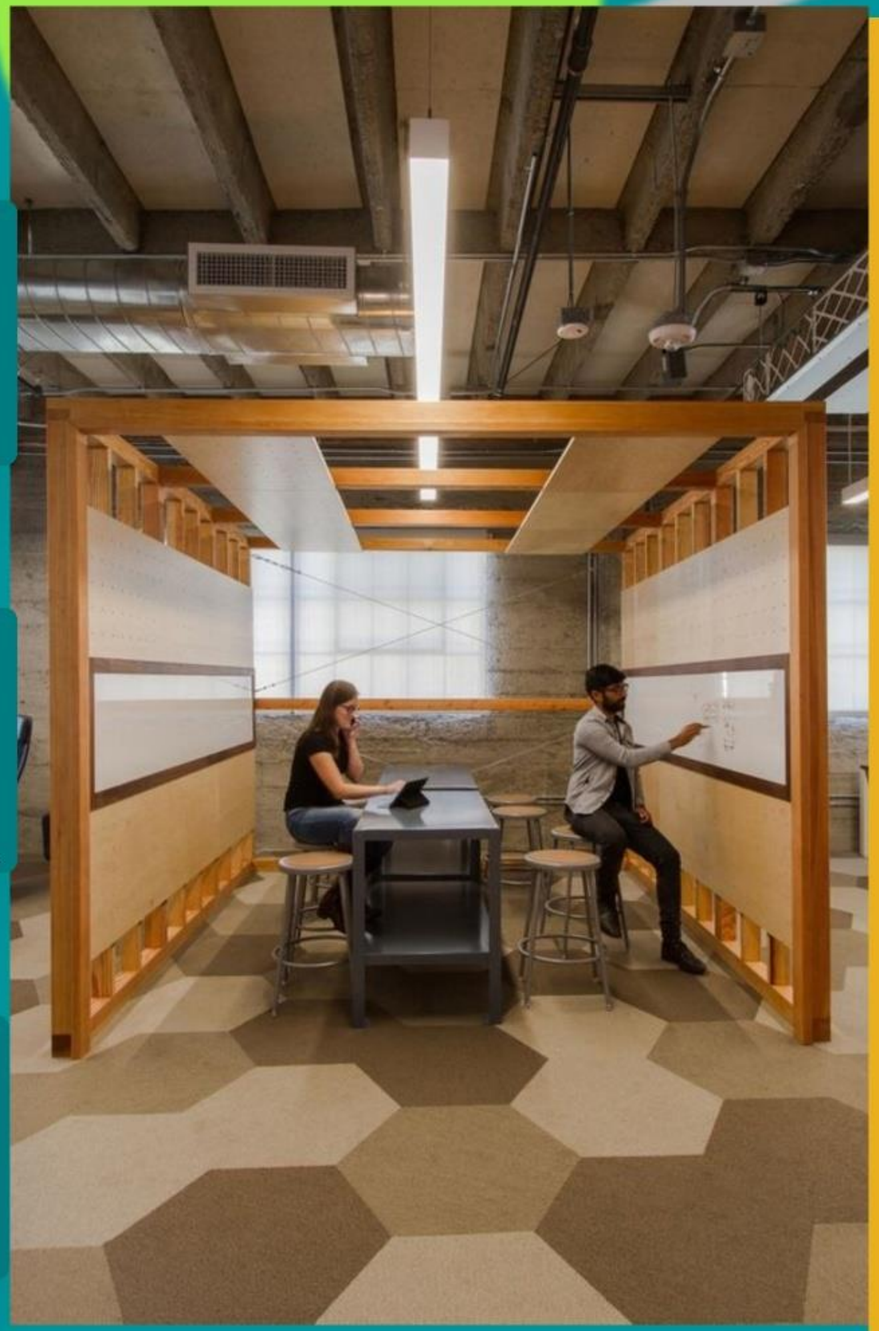
Common Challenges in PROC FCMP Handling

- Inconsistent dataset rules or formats
- Managing reporting precision
- Floating-point comparison issues
- Inconsistent validation or truncation logic

BioBrain

PROC FCMP in Action

- Create and reuse custom functions across multiple SAS programs
- Centralize logic for formatting, validation, and calculations
- Standardize rounding or date-handling logic
- Reduce redundant code across programs



BioBrain Pro

Consistency
Efficiency
Reusability

What Is PROC FCMP?

- Enables creation of custom SAS functions & subroutines
- Integrates logic directly into DATA steps and procedures
- Works across multiple SAS programs — no macro calls needed

BioBrainPro

Common Challenges in PROC FCMP Handling

- Inconsistent dataset rules or formats
- Managing reporting precision
- Floating-point comparison issues
- Inconsistent validation or truncation logic

PROC FCMP in Action

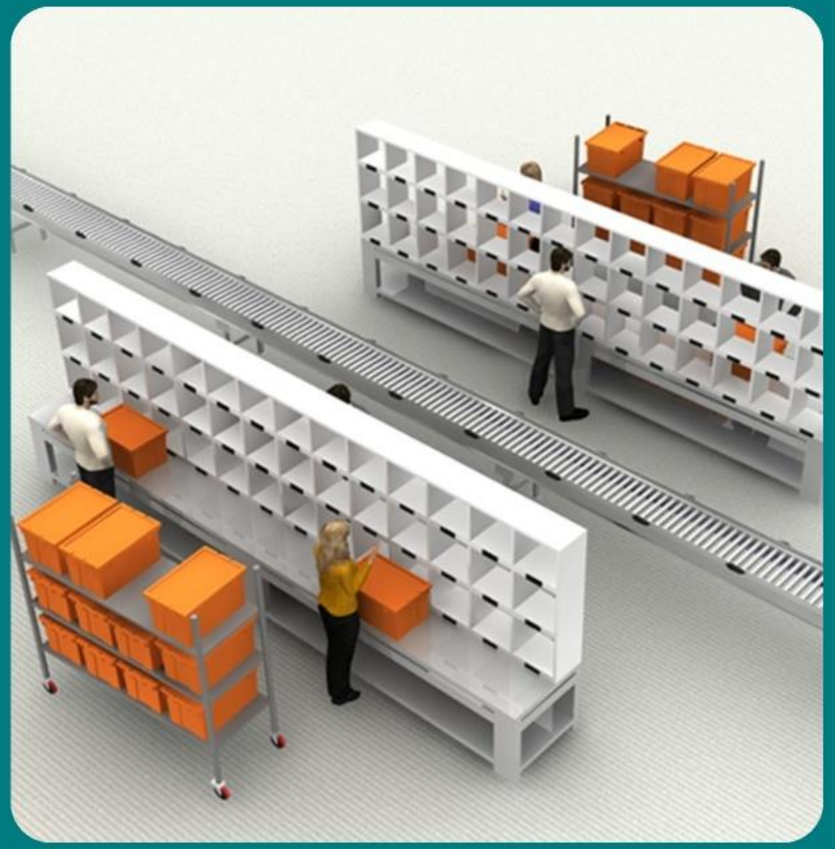
- Create and reuse custom functions across multiple SAS programs
- Centralize logic for formatting, validation, and calculations
- Standardize rounding or date-handling logic
- Reduce redundant code across programs

Programming Solutions with PROC FCMP

- Dynamically formatting outputs
- Detecting rounding or precision issues
- Supporting variable-level consistency checks
- Ensuring numeric consistency in key variable comparisons

Reduces Duplication and Improves Modularity

Utilizing PROC FCMP helps reduce code duplication and improves modularity by allowing programmers to create functions that can be reused across different projects. This modular approach leads to cleaner code, easier maintenance, and a more efficient programming environment.



BioBrainPro

BASIC STRUCTURE

```
proc fcmp outlib=<libref.dataset.package>;

    /* Function: Returns a single value */
    function function_name(arg1, arg2, ...) return_type;
        /* Logic goes here */
        return(result);
    endsub;

    /* Subroutine: Returns multiple values */
    subroutine subroutine_name(arg1, arg2, ..., output1, output2, ...);
        outargs output1, output2, ...;
        /* Logic goes here */
    endsub;

quit;

options cmplib=libref.dataset;
```

10

- OUTLIB: Specifies the three-level name of an output package where functions and routines are stored.
- Function: Defines a custom function you want to use later.
- Return_type: Optional for numeric.
- Return: Specifies the value that is returned by the function.
- Outargs: Required in subroutines to define output variables
- ENDSUB: Closes the function or subroutine block.
- QUIT: Exits the procedure.

```
proc fcmp outlib=<libref.dataset.package>;

    /* Function: Returns a single value */
    function function_name(arg1, arg2, ...) return_type;
        /* Logic goes here */
        return(result);
    endsub;

    /* Subroutine: Returns multiple values */
    subroutine subroutine_name(arg1, arg2, ..., output1, output2, ...);
        outargs output1, output2, ...;
        /* Logic goes here */
    endsub;

quit;

options cmplib=libref.dataset;
```

BioBrainPro

AUTOMATING SUMMARY STATISTIC PRECISION USING PROC FCMP

Decimal counting

- Minimum and Maximum – same level of precision as the raw data.
- Mean and Median – to one additional decimal place beyond the raw data.
- Standard Deviation / Standard Error – two additional decimal places beyond the raw data.

Round values with adjustable decimal precision

The same decimal precision applied in the final analysis outputs

Decimal counting

```
proc fcmp outlib=myfuncs.dec.utils;
   function count_decimals(num);
        length str $32;
        s2 = strip(num);
        if s2 = '' then return(0);
        if index(s2, '.') = 0 then return(0);
        decpart = strip(scan(s2, 2, '.'));          /* part after the dot */
        return(length(decpart));

   endsub;
quit;

options cmplib=myfuncs.dec;
```

- Minimum and Maximum - same level of precision as the raw data.
- Mean and Median - to one additional decimal place beyond the raw data.
- Standard Deviation / Standard Error - two additional decimal places beyond the raw data.

Round values with adjustable decimal precision

```
proc fcmp outlib=myfuncs.round_dec.utils;
function format_min(val, decimals, min);
   length result $20;
   if min = 0 then factor = 10**decimals;
   else if min = 1 then factor = 10**(decimals+1);
   else if min = 2 then factor = 10**(decimals+2);
   max_dec = 10**3;
   factor_min = min(factor,max_dec);
   rounded = round(val, 1/factor_min);
   result = strip(put(rounded, best32.));
   return(result);
endsub;
quit;

options cmplib=myfuncs.round_dec;
```

Inputs:

- Val – the statistic value.
- Decimals- the maximum number of decimals observed for the parameter.
- Stattype – the statistic type:
  - 0 = Min/Max -> same precision as data.
  - 1 = Mean/Median -> one extra decimal place.
  - 2 = SD/SE -> two extra decimal places.

The same decimal precision applied in the final analysis outputs

Visit/ Timepoint	Statistics	Actual Value	Change from Baseline	Actual Value	Change from Baseline	Actual Value	Change from Baseline
Baseline	n	12		12		12	
	Mean (SD)	72.4 (15.42)		67.0 (8.27)		70.8 (11.21)	
	SE	4.45		2.39		3.24	
	Median	70.0		68.0		69.5	
	Q1, Q3	58.0, 83.0		62.0, 72.5		63.0, 81.5	
	Min, Max	55, 101		51, 82		52, 86	

consistency and precision throughout their workflows.

BioBrainPro

Example: Attention Composite Z-Score Calculation

Z-scores are calculated at Baseline, Weeks 2, 4, 6, and 10 using this formula:

$$z = \frac{x - \bar{x}}{SD}$$

Where:

- X = participant endpoint response,
- $\bar{x}$ = baseline mean for all participants in the analysis population,
- SD = baseline standard deviation for all participants in the analysis population).

Since lower values indicate better performance, each Z-score is multiplied by -1 to ensure higher scores = better performance.

Creating and Reusing Derived Parameters Using PROC FCMP

Obs	SUBJID	PARAM	PARAMCD	AVAL	AVISIT	AVISITN
1	003-005	Detection	DET	2.68907	Baseline	1
2	003-005	Detection	DET	2.65156	Week 2	2
6	003-005	Identification	IDN	2.82111	Baseline	1
7	003-005	Identification	IDN	2.85957	Week 2	2
11	003-005	One Back Tests - Speed	ONBSPD	3.02192	Baseline	1
12	003-005	One Back Tests - Speed	ONBSPD	3.13437	Week 2	2

The following endpoints will be used to create the Attention Composite Z-score endpoint:

- CTB: Detection
- CTB: Identification
- CTB: One Back – Speed of performance

Z-scores are calculated at Baseline, Weeks 2, 4, 6, and 10 using this formula:

$$z = \frac{x - \bar{x}}{SD}$$

Where:

- X = participant endpoint response,
- $\bar{x}$ = baseline mean for all participants in the analysis population,
- SD = baseline standard deviation for all participants in the analysis population).

Since lower values indicate better performance, each Z-score is multiplied by -1 to ensure higher scores = better performance.

Compute Baseline means and SDs for each endpoint

Obs	SUBJID	AVISITN	AVISIT	_NAME_	_LABEL_	DET	IDN	ONBSPD
1	003-005	1	Baseline	AVAL	Analysis Value	2.68907	2.82111	3.02192
2	003-005	2	Week 2	AVAL	Analysis Value	2.65156	2.85957	3.13437
3	003-005	4	Week 4	AVAL	Analysis Value	2.65226	2.86338	3.09768
4	003-005	6	Week 6	AVAL	Analysis Value	2.62272	2.85730	3.14096
5	003-005	10	Week 10	AVAL	Analysis Value	2.75089	2.85943	3.03945

```
proc means data=cognitive_data noprint;
    where aVisit = "Baseline";
    var Det idn Onbspd;
    output out=baseline_stats mean=mean_det mean_id mean_ob
                              std=std_det std_id std_ob;
run;
```

IMPLEMENTATION USING PROC FCMP

```
proc fcmp outlib=work.cognitive.utils;

    /* Z-score with reversed directionality */
    function zscore_reverse(x, mean, sd);
        if missing(x) or sd = 0 then return(.);
        z = (x - mean) / sd;

        return (z);
    endsub;

    /* Compute Attention Composite Score */
    function attention_composite(z1, z2, z3);
        count = 0;
        sum = 0;
        if not missing(z1) then do;
            sum + z1;
            count + 1;
        end;
        if not missing(z2) then do;
            sum + z2;
            count + 1;
        end;
        if not missing(z3) then do;
            sum + z3;
            count + 1;
        end;

        if count >= 2 then do;
            z4 = sum / count;
            return (-1*z4);
        end;
        else return(.);
    endsub;

quit;

options cmplib=work.cognitive;
```

zscore_reverse()

- calculates a Z-score and maintains the logic for reversing directionality (higher is better).

attention_composite()

- computes the average of the available Z-scores, only if at least two of the three are non-missing.

Merging Baseline Statistics and Calculating Cognitive Scores

```
/* Merge baseline stats into main dataset */
data cognitive_with_stats;
    if _n_ = 1 then set baseline_stats(keep=mean_: std_:);
    set cognitive_data;

    /* Compute Z-scores with direction reversed */
    z_det = zscore_reverse(Det, mean_det, std_det);
    z_id  = zscore_reverse(Idn, mean_id, std_id);
    z_ob  = zscore_reverse(Onbspd, mean_ob, std_ob);

    /* Compute composite */
    attn_composite = attention_composite(z_det, z_id, z_ob);
run;
```

Obs	mean_det	std_det	SUBJID	AVISITN	AVISIT	DET	IDN	ONBSPD	z_det	z_id	z_ob	attn_composite
1	2.58302	0.10096	003-005	1	Baseline	2.68907	2.82111	3.02192	1.05034	1.14177	1.34805	-1.18005
2	2.58302	0.10096	003-005	2	Week 2	2.65156	2.85957	3.13437	0.67887	1.69536	2.61220	-1.66214
3	2.58302	0.10096	003-005	4	Week 4	2.65226	2.86338	3.09768	0.68580	1.75020	2.19973	-1.54524
4	2.58302	0.10096	003-005	6	Week 6	2.62272	2.85730	3.14096	0.39322	1.66268	2.68628	-1.58073
5	2.58302	0.10096	003-005	10	Week 10	2.75089	2.85943	3.03945	1.66270	1.69334	1.54512	-1.63372
6	2.58302	0.10096	003-008	1	Baseline	2.55735	2.75765	2.98786	-0.25430	0.22834	0.96510	-0.31304
7	2.58302	0.10096	003-008	2	Week 2	2.58883	2.78907	2.95831	0.05755	0.68059	0.63296	-0.45703
8	2.58302	0.10096	003-008	4	Week 4	2.78243	2.81833	2.91833	1.97509	1.10176	0.18351	-1.08679
9	2.58302	0.10096	003-008	6	Week 6	2.61752	2.80574	2.98588	0.34171	0.92054	0.94290	-0.73505
10	2.58302	0.10096	003-008	10	Week 10	2.57955	2.79379	2.90943	-0.03437	0.74853	0.08346	-0.26587

CONCLUSION

PROC FCMP offers a modern, functional approach to SAS programming. A single, well-designed function can be widely reused to address various programming challenges with clarity and precision.

Contact Information

Authors: Syuzanna Simonyan, Mariam Babayan
Company: Biobrain Pro LLC
Address: 2/2 A Mikoyan Street, 0054, Yerevan, Armenia
Emails: syuzie.ss@biobrainpro.net,
marbabayan@biobrainpro.net
Website: biobrainpro.net

BioBrainPro

Your Questions Are Welcome!