SAS2131-2018

Customizing the Aggregation of Risk Data

Scott Gray and Stacey Christian, SAS Institute Inc.

ABSTRACT

You asked for real-time and on-the-fly queries of risk data, and SAS answered with SAS® High-Performance Risk. Now with new and changing regulations such as the FRTB (Fundamental Review of the Trading Book) and SIMM (Standard Initial Margin Model), you need customizable queries. Adding to its established evaluation-stage actions, SAS High-Performance Risk now offers you control over how your query aggregates data. You can customize exposure calculations, weight aggregations, perform correlated aggregations, and calculate other nonlinear measures.

INTRODUCTION

Financial risk analysis involves calculating risk metrics at various levels of a portfolio's hierarchical structure. Behind the risk metrics is the aggregation of either simulated or stressed values for the positions in the portfolio or for position-level risk factor sensitivities. The aggregation and computation of risk metrics is computed on demand when possible to make risk-aware decisions and to provide an interactive, real-time view of the risk in a portfolio. Usually the aggregation step is quite fast—even on large volumes of data—because the aggregation methodology is a simple sum. However, many cases exist in which the aggregation of positions into portfolios and sub-portfolios is much more complex than a sum (for example, multilevel netting in counterparty credit risk). SAS High-Performance Risk enables you to write complex aggregation methods that run in a distributed, in-memory environment and deliver advanced, customizable risk calculations on demand.

AGGREGATING VALUES

Suppose your portfolio has two positions—A and B—and each has its own value—Value_A and Value_B. Summing often expresses the aggregated value in a simple, natural, and useful way:

$$\text{Value}_{\text{Portfolio}} = \text{Value}_A + \text{Value}_B$$

However, summing is not a universally appropriate way to aggregate. In particular, summing is not appropriate for rates. Suppose each position has an internal rate of return (IRR)—IRR_A and IRR_B. You would expect the IRR of the portfolio to be between IRR_A and IRR_B. In this case, calculating the mean might be more accurate:

$$\text{IRR}_{\text{Portfolio}} = (\text{IRR}_A + \text{IRR}_B)/2$$

In fact, calculating a weighted mean might be even better:

$$\text{IRR}_{\text{Portfolio}} = (\text{Value}_A{*}\text{IRR}_A + \text{Value}_B{*}\text{IRR}_B)/ (\text{Value}_A + \text{Value}_B)$$

Further, you typically derive the exposure of the portfolio from the position's values, augmented to keep it nonnegative. Varying concerns could justify either of the following calculations of the portfolio's exposure:

$$\text{Exposure}_{\text{Portfolio}} = \text{Max}(\text{Value}_A, 0) + \text{Max}(\text{Value}_B, 0)$$

$$\text{NettedExposure}_{\text{Portfolio}} = \text{Max}(\text{Value}_A + \text{Value}_B, 0)$$

Other advanced calculations might justify tracking the maximum exposure or minimum return across traders. Further, regulatory calculations such as FRTB and SIMM might justify correlated aggregation of the data. Each of these use cases requires a non-summable process to accurately aggregate the position-level values. To accommodate these and other non-summable results, SAS High-Performance Risk introduces custom aggregations.

CUSTOM AGGREGATION TERMS

To generalize the concepts described in the previous section, consider an example that has more than two positions. Figure 1 shows a custom aggregation of a portfolio that contains 18 positions.

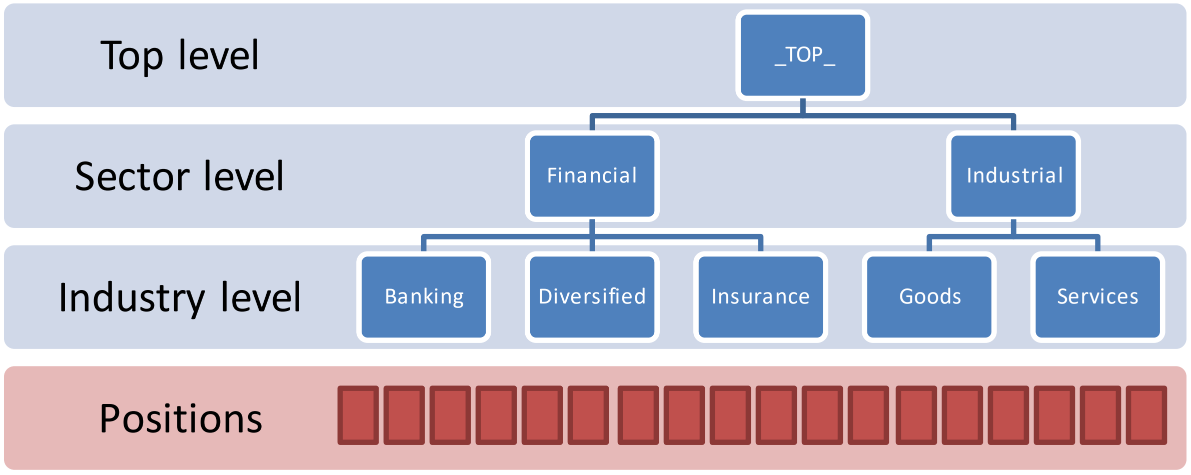


Figure 1. Demonstration of Custom Aggregation

The following terms are useful when describing larger custom aggregations:

- *Aggregation*: a collection of positions in the portfolio in which a classification variable equals a certain value—for example, the positions where Sector="Financial" or where Industry="Goods". Figure 1 demonstrates eight aggregations: _TOP_ at the Top level, Financial and Industrial at the Sector level, and five at the Industry level. The positions are not an aggregation.
- *Aggregation level*: the collection of aggregations for various values of a classification variable—for example, the collection of aggregations where Sector="Financial" and Sector="Industrial". Figure 1 contains three aggregation levels: Top, Sector, and Industry. The positions are not an aggregation level.
- *Roll-up*: the process of collecting positions into an aggregation level. Figure 1 demonstrates three roll-ups: Positions to Industry, Industry to Sector, and Sector to Top.
- *Simple summing*: the default roll-up process. This is distinct from custom aggregation and involves summing the values of the positions in the aggregation.
- *Child level and parent level*: aggregation levels relative to the current roll-up. The child level is the aggregation level from which you roll up. The parent level is the aggregation level to which you roll up. For the roll-up from Industry to Sector, Industry is the child level and Sector is the parent level. However, for the roll-up from Sector to Top, Sector is the child level.

SPECIFYING A CUSTOM AGGREGATION

To perform a custom aggregation, you use roll-up methods (which you specify in the COMPILE procedure) and a hierarchy (which you specify in the HPRISK procedure). The roll-up methods and hierarchy enable you to control the rolling up of each output variable at each aggregation level. The examples in this section create a custom aggregation for the portfolio (the collection of positions) and classification variables that are shown in Figure 1. Appendix 1 contains the SAS code that demonstrates the concepts and syntax that are outlined in this section.

HIERARCHY

In the HIERARCHY statement in PROC HPRISK, you specify a list of classification variables (with the variables in parentheses) and a data set (in the ROLLUP_DATA= option):

```
   hierarchy mySectorIndustryHierarchy (Sector Industry)
        rollup_data = mylib.myRollupData;
```

The classification variable list specifies the order of the aggregation levels. This example creates three aggregation levels: Industry, Sector, and _TOP_ (always implicitly there). More importantly for this discussion, it creates three roll-ups: Positions to Industry, Industry to Sector, and Sector to _TOP_.

The roll-up data set specifies which roll-up method to run at each roll-up in the hierarchy's classification variable list. Table 1 shows appropriate columns and values for the ROLLUP_DATA data set that is specified in the hierarchy mySectorIndustryHierarchy.

CCVar	**myValue**	**myExposure**	**myNettedExposure**	**myDuration**
TOP	_SUM_	_SUM_	_SUM_	weightSum
Sector	_SUM_	_SUM_	netExposures	weightSum
Industry	sumExposures	_NONE_	_MISSING_	weightSum

Table 1. Roll-Up Data Set

This data set contains a column named CCVar (cross-classification variable) and a column for each output variable in the analysis that requires custom aggregation. The values of the CCVar variable are _TOP_ and the variables in the hierarchy's classification variable list. The values of the output variables indicate the roll-up method to run when aggregating to that classification variable. In this example, the roll-up method netExposures will aggregate the output variable myNettedExposure from Industry to Sector. PROC HPRISK offers three system-defined roll-up methods:

- _SUM_ sums the values from the child level. If all roll-up methods are _SUM_, the custom aggregation would calculate the same results as simple summing.
- _NONE_ skips the setting of that output variable to that aggregation level. A value of _NONE_ suggests that some other roll-up method at this level will set these values. In this example, the method sumExposures will set myExposure. If no method sets the value of myExposure, then its value will be zero at this level.
- _MISSING_ indicates that the aggregated values should be set to missing. A value of _MISSING_ suggests that the current output variable is not relevant to the current aggregation level. In this example, myNettedExposure is _MISSING_ at the Industry level because myNettedExposure is out of context at that level.

The order of rows in the roll-up data set is irrelevant because the classification variable list in the HIERARCHY statement specifies the order. The roll-up method always aggregates to the value of CCVar—either from the next class variable in the hierarchy's classification variable list or from the positions if it is the last classification variable in the hierarchy.

If you have an output variable that is not in the data set, the query uses _SUM_ at every aggregation level. Similarly, if you specify a classification variable in the hierarchy but do not give it a row in the ROLLUP_DATA= data set, the query uses _SUM_ for each output variable.

By default, PROC HPRISK does not report results at the position level. If you want to report results at the position level, add InstId to the end of the classification variable list.

ROLL-UP METHODS

The COMPILE procedure enables you to specify a roll-up method. The roll-up methods typically follow these steps:

1. Get the child-level values.
2. Calculate the parent-level value from the child-level values.
3. Set the parent-level value.

The _ROLLUP_ object is a system-defined object that enables you to get and set the values of output variables in the roll-up's child and parent levels. With it, you have access to the following accessor routines:

- *getInputValues(outvar, array)* fills *array* with values of *outvar* from the child level. If you specify no *outvar*, this routine returns values for the current output variable.
- *setValue(outvar, value)* sets the value of *outvar* to *value* for the parent level. If you specify no *outvar*, this routine sets the value for the current output variable.

For more advanced usage, you can also access the names and values of classification variables in the child level and parent level of the roll-up. For more information, see the section "More Accessor Methods."

The following sections demonstrate rollup methods that use the _ROLLUP_ object to get and set output variable values.

Sum Values

The following roll-up method demonstrates a user-defined version of _SUM_:

```
method sumValues kind=rollup;
   /* get the child-level values */
   call _rollup_.getInputValues(., childOVValues);
   nChildValues = dim(childOVValues);

   /* calculate the parent-level value from the child-level values */
   myValueTmp = 0;
   do i = 1 to nChildValues;
      myValueTmp = myValueTmp + childOVValues[i];
   end;

   /* set the parent-level value */
   call _rollup_.setValue(., myValueTmp);
endmethod;
```

Note that the first argument to *getInputValues* and *setValue* is missing (.). A missing value prompts the method to get and set the values of the current output variable.

Net Exposure

The following roll-up method demonstrates how to use child values from one output variable to set a different output variable—in this case using myValue to set myNettedExposure:

```
method netExposures kind=rollup;
   call _rollup_.getInputValues("myValue", childOVValues);
   nChildValues = dim(childOVValues);

   myValueTmp = 0;
   do i = 1 to nChildValues;
      myValueTmp = myValuetmp + childOVValues[i];
   end;
```

```
      myNettedExposureTmp = max(myValueTmp, 0);

      call _rollup_.setValue("myNettedExposure", myNettedExposureTmp);
   endmethod;
```

Sum Exposure

The following roll-up method demonstrates how to set multiple output variables in one method:

```
   method sumExposures kind=rollup;
      call _rollup_.getInputValues("myValue", childOVValues);
      nChildValues = dim(childOVValues);

      myValueTmp = 0;
      myExposureTmp = 0;
      do i = 1 to nChildValues;
         myValueTmp = myValueTmp + childOVValues[i];
         if childOVValues[i] > 0 then
            myExposureTmp = myExposureTmp + childOVValues[i];
      end;

      call _rollup_.setValue("myValue",myValueTmp);
      call _rollup_.setValue("myExposure",myExposureTmp);
   endmethod;
```

Weight Sums

This section demonstrates two roll-up methods that weight sums.

The following roll-up method demonstrates how to calculate a weighted sum:

```
   method weightSum kind=rollup;
      call _rollup_.getInputValues("myDuration", childDurValues);
      nChildValues = dim(childDurValues);
      call _rollup_.getInputValues("myValue", childValues);

      durationNum = 0;
      durationDenom = 0;
      do i = 1 to nChildValues;
         weight = abs(childValues[i]);
         durationNum   = durationNum + weight*childDurValues[i];
         durationDenom = durationDenom + weight;
      end;
      if durationDenom ne . and durationDenom ne 0 then
         durationTmp = durationNum / durationDenom;
      else
         durationTmp = .;

      call _rollup_.setValue("myDuration", durationTmp);
   endmethod;
```

The following roll-up method demonstrates an optimization that calculates the intermediate value myValueTmp, which is then used to set both myValue and myDuration. Note that this method also weights myDuration with myValue instead of with abs(myValue):

```
method weightSums2 kind=rollup;
   call _rollup_.getInputValues(“myDuration”, childDurValues);
   nChildValues = dim(childDurValues);
   call _rollup_.getInputValues(“myValue”, childValues);
```

```
myValueTmp = 0;
durationNum = 0;
do i = 1 to nChildValues;
   myValueTmp = myValueTmp + childValues[i];
   durationNum = durationNum + childValues[i]*childDurValues[i];
end;
if myValueTmp ne . and myValueTmp ne 0 then
   myDurationTmp = durationNum / myValueTmp;
else
   myDurationTmp = .;

call _rollup_.setValue("myValue",myValueTmp);
call _rollup_.setValue("myDuration", myDurationTmp);
endmethod;
```

ADVANCED CUSTOM AGGREGATION

Roll-ups for regulations such as FRTB and SIMM involve matrix multiplication of the form **Z'CZ** where **Z** is the vector of child-level values and **C** is a correlation matrix that relates those child-level aggregations. This matrix multiplication requires that the values being aggregated be matched with the proper correlations. You can calculate aggregations such as these by storing the correlations in a parameter matrix and using advanced accessor methods.

You can load the correlations into a parameter matrix that has named rows and columns. Doing so enables you to take advantage of live parameter matrices in roll-up methods, even updating parameters (such as collateral values or margin requirements in Counterparty Credit Risk, or correlation shocks in FRTB) on the fly.

MORE ACCESSOR METHODS

Advanced custom aggregations require access to more than just the output variable values. They often require the names or values of the child-level and parent-level classification variables. To fill those needs, the _ROLLUP_ object offers advanced accessor routines such as in the following list. The examples that are shown assume that aggregations with Industry="Banking", "Diversified", and "Equity" are rolling up into Sector="Financial":

- *getInputClassName()* returns the child-level classification variable name. The following function would return "Industry":

  ```
  _ROLLUP_.getInputClassName()
  ```

- *getInputClassValue (i, charValue, numValue*) sets *charValue* if the child-level classification variable has character values and sets *numValue* if the child-level classification variable has numeric values. The value it sets is from the *i*th child-level aggregation. The following call would set industryName="Diversified":

  ```
  Call _ROLLUP_.getInputClassValue(2, industryName, unusedVar) ;
  ```

- *getClassName()* returns the name of the parent-level classification variable. The following function would return "Sector"*:*

  ```
  _ROLLUP_.getClassName()
  ```

- *getClassValue(ccvar, charValue, numValue)* sets the value of the current parent-level classification variable. The following call would set sectorName="Financial":

  ```
  Call _ROLLUP_.getClassValue("Sector", sectorName, unusedVar) ;
  ```

- *getOutvarName()* returns current output variable name.

CORRELATED AGGREGATION

If you store the correlations in a parameter matrix, you can use the accessor routine *getInputClassValue* to access correlations for the child aggregations. The following roll-up method demonstrates how you can use this approach to perform correlated aggregation:

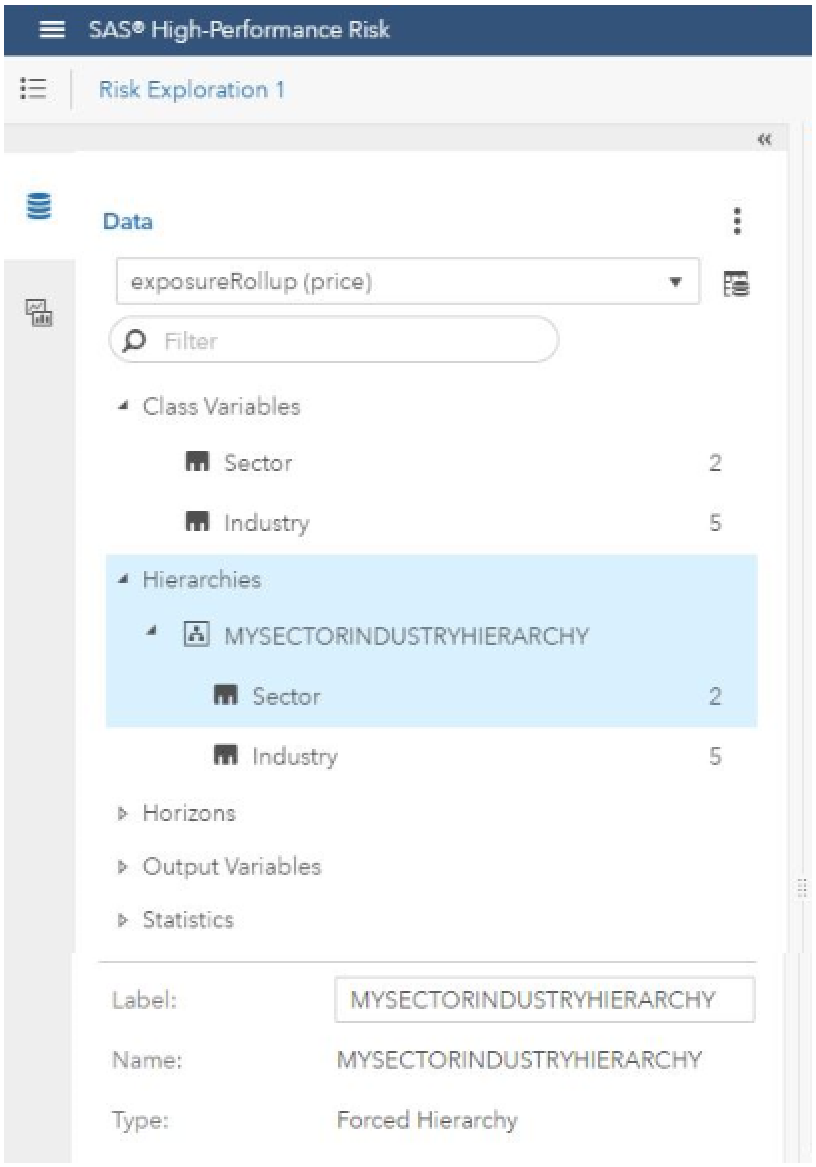
```
method weightSensitivities kind=rollup;
   length charValue $ 32;
   length ivalue jvalue $ 32;
   call _rollup_.getClassValue("Sector", charValue, unusedNum);
   call _rollup_.getInputValues("weightedSensitivity", childSensitivities);
   nChildValues = dim(childSensitivities);

   /* calculate sqrt(Z’CZ) */
   sumTmp = 0;
   do i = 1 to nChildValues;
      call _rollup_.getInputClassValue (i, iValue, unusedNum);
      do j = 1 to nChildValues;
         call _rollup_.getInputClassValue (j, jValue, unusedNum);
         call pmxelem(FinCovMat, iValue, jValue, corrValue);
         sumTmp = sumTmp +
childSensitivities[i]*childSensitivities[j]*corrValue;
      end;
   end;
   sumTmp = sqrt(sumTmp);

   call _rollup_.setValue("weightedSensitivity", sumTmp);
endmethod;
```

Appendix 2 contains SAS code that demonstrates the concepts and syntax that are outlined in this section.

EXPLORING IN THE USER INTERFACE

The SAS High-Performance Risk user interface offers analytic and exploration capabilities that include filtering, side-by-side comparison, and exploration through plots and tables. All of these capabilities are available when you explore the results of custom aggregation.

Figure 2 shows the Data pane for the results from Appendix 1.



Label: MYSECTORINDUSTRYHIERARCHY

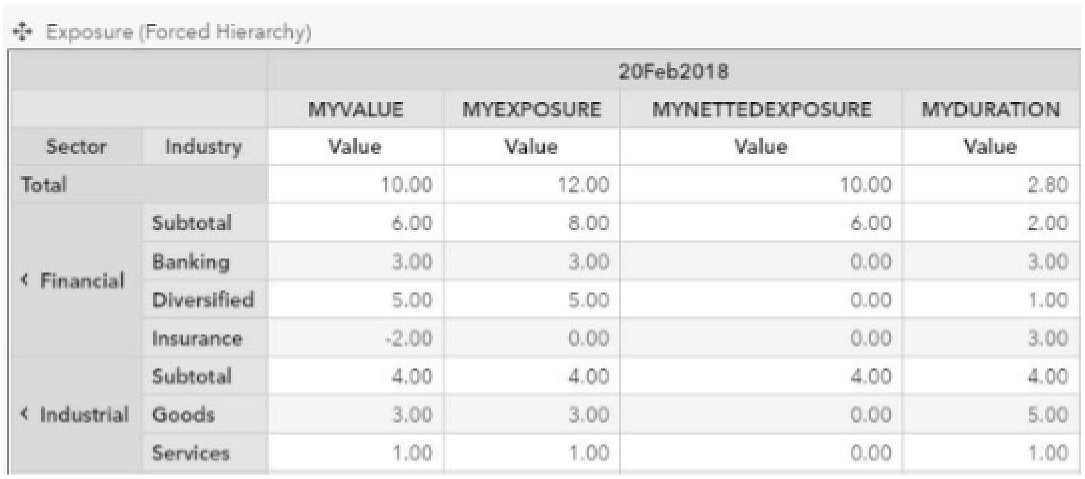
Name: MYSECTORINDUSTRYHIERARCHY

Type: Forced Hierarchy

Figure 2. Data Pane for Custom Aggregation

The Data pane displays the classification variables and how they contribute to the shape of the hierarchy *mySectorIndustryHierarchy*. The type **Forced Hierarchy** indicates that the aggregation levels are fixed and cannot be changed. You can click **Output Variables** (or expand the arrow beside it) to display the list of output variables that you can view in the crosstabulation table (crosstab).

Figure 3 shows the crosstabulation table (crosstab) for the results from Appendix 1.



Exposure (Forced Hierarchy)

		20Feb2018			
		MYVALUE	MYEXPOSURE	MYNETTEDEXPOSURE	MYDURATION
Sector	Industry	Value	Value	Value	Value
Total		10.00	12.00	10.00	2.80
Financial	Subtotal	6.00	8.00	6.00	2.00
	Banking	3.00	3.00	0.00	3.00
	Diversified	5.00	5.00	0.00	1.00
	Insurance	-2.00	0.00	0.00	3.00
Industrial	Subtotal	4.00	4.00	4.00	4.00
	Goods	3.00	3.00	0.00	5.00
	Services	1.00	1.00	0.00	1.00

Figure 3. Crosstabulation Table for Custom Aggregation

The pricing method sets the output variables MYVALUE, MYEXPOSURE, MYNETTEDEXPOSURE, and MYDURATION. You can see the effect of the roll-up method *sumExposures* by observing MYEXPOSURE where Industry=Insurance.

$$\text{myExposure}_{\text{Insurance}} = \text{Max}(–2.00, 0.00) = 0.00$$

You can see the effect of the roll-up method *netExposures* by observing MYNETTEDEXPOSURE for the subtotal where Sector=Financial. This demonstrates the effect of myValue=–2.00 from Industry=Insurance, which is lost in the calculation of MYEXPOSURE at the same level.

$$\begin{aligned}\text{myNettedExposure}_{\text{Financial}} &= \text{myValue}_{\text{Banking}} + \text{myValue}_{\text{Diversified}} + \text{myValue}_{\text{Insurance}} \\ &= 3.00 + 5.00 + –2.00 \\ &= 6.00\end{aligned}$$

Finally, you can most easily see the effect of the roll-up method *weightSums* by observing how MYDURATION for Industry=Goods and Services rolls up to Subtotal for Sector=Industrial.

$$\begin{aligned}\text{myDuration}_{\text{Industrial}} &= \frac{\text{myValue}_{\text{Goods}} * \text{myDuration}_{\text{Goods}} + \text{myValue}_{\text{Services}} * \text{myDuration}_{\text{Services}}}{\text{myValue}_{\text{Goods}} + \text{myValue}_{\text{Services}}} \\ &= \frac{3.00 * 5.00 + 1.00 * 1.00}{3.00 + 1.00} \\ &= \frac{16.00}{4.00} \\ &= 4.00\end{aligned}$$

Figure 4 shows the plots for the results from Appendix 1.

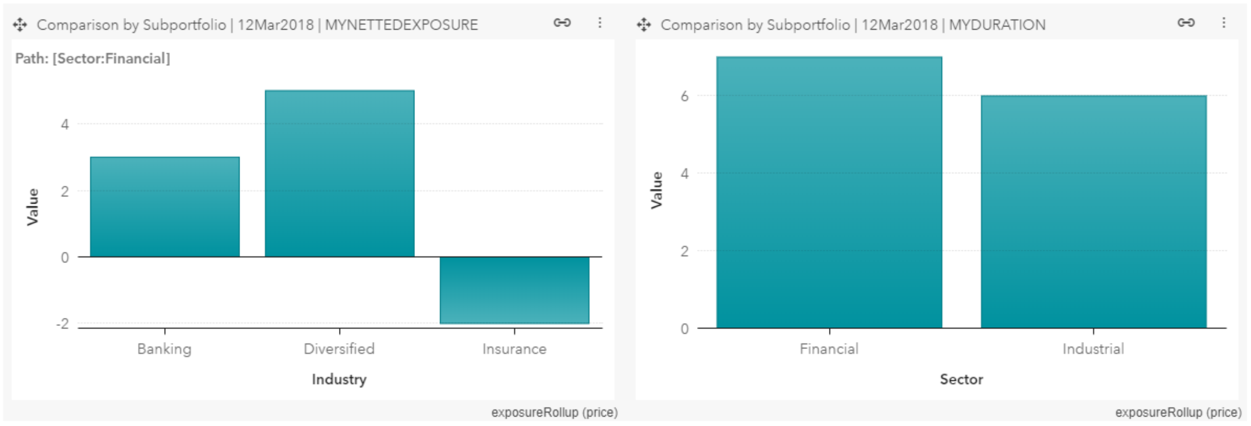


Figure 4. Plots Showing Results of Custom Aggregation

The left plot displays values of MYNETTEDEXPOSURE for Industry for Sector=Financial. Similarly, the right plot displays values of MYDURATION for values of Sector.

PERFORMANCE

Custom aggregations typically run slower than either summing with _SUM_ or simple summing. One reason is that nonlinear calculations are typically more complex. Also, querying _TOP_ with simple summing would calculate results only at the top level and would skip calculations at the Industry and Sector levels. Custom aggregations cannot skip any level of a hierarchy that has a custom roll-up method defined. However, strategies exist that can help reduce run time and memory usage.

Some output variables share intermediate values. For example, the *WeightSums2* method (which is specified in the section "Weight Sums") calculates myValueTmp once and uses it to set both myValue and myDuration. Calculating the intermediate value once and using it to set both output variables in a single roll-up method saves run time over calculating each output variable separately. You could also take advantage of how regulatory and internal capital requirements share intermediate values by calculating those intermediate values once and setting multiple output variables in a single roll-up method.

One feature of High-Performance Risk is that it distributes the valuation of positions across nodes, distributing the load to reduce run time. However, queries require that these values be aggregated on a single node. To reduce excessive data movement—which slows run time and consumes more memory—PROC HPRISK sums query results within each node when possible before it transfers data to a different node. During custom aggregation, _SUM_ can also take advantage of this optimization, but custom roll-up methods cannot. Custom roll-up methods cannot sum before transferring, so PROC HPRISK must transfer the child-level aggregations to a single node. To better understand this issue, consider a portfolio that has 1,000,000 positions and no classification variable on 100 nodes. To run a custom roll-up method, PROC HPRISK must copy all 1,000,000 positions to a single node. However, if your positions sum to an intermediate classification variable before a custom aggregation needs to be performed, adding that classification variable to the hierarchy would prompt summing on each of the 100 nodes before transferring data to the single node that performs the custom aggregation. To minimize data movement and thereby maximize the effect of this optimization, the classification variable you add should minimize the cardinality of the new aggregation level. This strategy is particularly effective when you aggregate from the position level, because the cardinality of the position level is the largest.

CONCLUSION

Given the demand for complex, nonlinear risk metrics, particularly to the degree seen in recent risk regulation, the flexibility to control the aggregation of values and maintain fast performance is essential to modern risk management software. The new custom aggregation feature in SAS High-Performance Risk enables you to make risk-aware and capital-aware decisions, by providing an interactive, real-time view of the risk of a portfolio. SAS High-Performance Risk runs in a distributed, in-memory environment, so you you receive the most advanced, customizable risk calculations on demand.

APPENDIX 1

The following SAS High-Performance Risk code demonstrates custom aggregation:

```
libname myLib "&data_path";

/**************************************************************************/
/* DECLARE VARIABLES
/**************************************************************************/
proc risk;
   env new=myLib.myRollupEnv;
   setoptions nooptimize;
   declare instvars=(MtMValue num var,
                     MtMDuration num var,
                     Sector char 32 var,
                     Industry char 32 var);
   declare riskfactors=(rf1 num var);
   declare outvars=(myValue num computed,
                    myExposure num computed comptype=exposure,
                    myNettedExposure num computed comptype=exposure,
                    myDuration num computed);

  env save;
run;

/**************************************************************************/
/* CREATE METHODS
/**************************************************************************/
proc compile env=myLib.myRollupEnv outlib=myLib.myRollupEnv;
   method Price1 kind=price;
      _value_ = rf1;
      myValue = MtmValue;
          myExposure = .;
          myNettedExposure = .;
      myDuration = MtMDuration;
   endmethod;

   method sumExposures kind=rollup;
      call _rollup_.getInputValues("myValue", child0VValues);
      nChildValues = dim(child0VValues);

      myValueTmp = 0;
      myExposureTmp = 0;
      do i = 1 to nChildValues;
         myValueTmp = myValueTmp + child0VValues[i];
         if child0VValues[i] > 0 then
            myExposureTmp = myExposureTmp + child0VValues[i];
      end;
      call _rollup_.setValue("myValue",myValueTmp);
      call _rollup_.setValue("myExposure",myExposureTmp);
   endmethod;

   method netExposures kind=rollup;
      call _rollup_.getInputValues("myValue", childOVValues);
      nChildValues = dim(childOVValues);

      myValueTmp = 0;
      do i = 1 to nChildValues;
         myValueTmp = myValuetmp + childOVValues[i];
      end;
      myNettedExposureTmp = max(myValueTmp, 0);

      call _rollup_.setValue("myNettedExposure", myNettedExposureTmp);
   endmethod;

   method weightSums kind=rollup;
      call _rollup_.getInputValues("myDuration", childDurValues);
      nChildValues = dim(childDurValues);
      call _rollup_.getInputValues("myValue", childValues);

      durationNum = 0;
      durationDenom = 0;
      do i = 1 to nChildValues;
         weight = abs(childValues[i]);
         durationNum   = durationNum + weight*childDurValues[i];
         durationDenom = durationDenom + weight;
      end;
      if durationDenom ne . and durationDenom ne 0 then
         durationTmp = durationNum / durationDenom;
      else
         durationTmp = .;

      call _rollup_.setValue("myDuration", durationTmp);
   endmethod;
run;

/***************************************************************************/
/* CREATE DATA
/***************************************************************************/
data hierData.SGFdemo_RollupData;
   length CCVar myValue myExposure myNettedExposure myDuration $32;
   input  CCVar $ myValue $ myExposure $ myNettedExposure $ myDuration $;
   datalines;
_TOP_     _SUM_          _SUM_    _SUM_          weightSums
Sector    _SUM_          _SUM_    netExposures   weightSums
Industry  sumExposures   _NONE_   _MISSING_      weightSums
;
run;

data myLib.instdata;
   length Insttype Instid Sector Industry$32.;
   input InstType $ InstId $ Sector $ Industry $ MtMValue MtMDuration;
   datalines;
Inst1 i1 Financial  Banking       3 3
Inst1 i2 Financial  Diversified   5 1
```

```
Inst1 i3 Financial  Insurance    -2 3
Inst1 i4 Industrial Goods         3 5
Inst1 i5 Industrial Services      1 1
run;

data myLib.mktdata;
   rf1=1;
run;

/***************************************************************************/
/* CREATE PROJECT
/***************************************************************************/
proc risk;
   env open=myLib.myRollupEnv;

   marketdata Market
      file = myLib.mktdata
      type = current;

   instrument Inst1
      variables = (MtMValue MtMDuration Sector Industry)
      methods = (Price Price1);
   instdata Port
      file = myLib.instdata
      format = simple;
   sources Source (Port);
   read sources = Source out=Read;

   crossclass SectorIndustry (Sector Industry);

   project Project
      data = ( Market )
      portfolio = Read
      crossclass = SectorIndustry
      rollupmethods = (sumExposures netExposures weightSums);

  env save;
run;

proc hpexport
   expprojlib = myLib
   env = myLib.myRollupEnv;
run;

/***************************************************************************/
/* CREATE CUBE
/***************************************************************************/
proc hprisk
   expprojlib = myLib
   task = runproject
   project = Project
   cube = "&cube_path\exposureRollup"
   filesprefix = "&hprfilesprefix/exposureRollup"
   priceby = positions;

   outvars replace=(myValue myExposure myNettedExposure myDuration);
   crossclassvars Sector Industry;
   hierarchy mySectorIndustryHierarchy (Sector Industry)
      rollup_data = hierdata.SGFdemo_RollupData;
run;
quit;

/**************************************************************************/
/* QUERY CUBE
/**************************************************************************/
proc hprisk
   task = query
   cube = "&cube_path\exposureRollup"
   tracelog = yes;

   output summary = myLib.hprSummary
      hierarchy = mySectorIndustryHierarchy;
   query;
   query Sector;
   query Sector Industry;
run;
quit;

proc print
   data = myLib.hprSummary(keep=Sector Industry NInst myValue myExposure
myNettedExposure myDuration);
run;

proc hprisk
   task = clean
   cube = "&cube_path/exposureRollup";
run;
```

APPENDIX 2

The following SAS High-Performance Risk code demonstrates correlated aggregation:

```
libname myLib "&data_path";

data myLib.FinancialSectorCovMat;
length _name_ _type_ $32;
_type_= "COV";
input _name_ $ Banking Diversified Insurance Equity;
datalines;
Banking      .23  .05  .03  .10
Diversified  .05  .15  .05  .04
Insurance    .03  .05  .20  .02
Equity       .10  .04  .02  .20
;
run;
data myLib.IndustrialSectorCovMat;
length _name_ _type_ $32;
_type_= "COV";
input _name_ $ Goods Services;
datalines;
Goods     .21  .06
Services  .06  .14
;
run;
```

```
/***************************************************************************/
/* DECLARE VARIABLES
/***************************************************************************/
proc risk;
   env new=myLib.myRollupEnv;
   setoptions nooptimize;
   declare instvars=(MtMValue num var,
                    Sector char 32 var,
                    Industry char 32 var);
   declare riskfactors=(rf1 num var);
   declare outvars=(weightedSensitivity num computed);

   marketdata FinCovData
      file=myLib.FinancialSectorCovMat
      type=parameter;
   marketdata IndCovData
      file=myLib.IndustrialSectorCovMat
      type=parameter;

   parameter FinCovMat data=FinCovData
      Rows=(Banking Diversified Insurance Equity)
          Columns=(Banking Diversified Insurance Equity);
   parameter IndCovMat data=IndCovData
      Rows=(Goods Services)
          Columns=(Goods Services);

  env save;
run;

/***************************************************************************/
/* CREATE METHODS
/***************************************************************************/
proc compile env=myLib.myRollupEnv outlib=myLib.myRollupEnv;
   method Price1 kind=price;
      _value_ = MtmValue;
      weightedSensitivity = _value_;
   endmethod;

   method weightSensitivities kind=rollup;
      /* select parameter matrix containing correlations */
      length charValue $ 32;
      length ivalue jvalue $ 32;
      call _rollup_.getClassValue("Sector", charValue, unusedNum);
      call _rollup_.getInputValues("weightedSensitivity",
childSensitivities);
      nChildValues = dim(childSensitivities);

      /* calculate sqrt(Z'CZ) */
      sumTmp = 0;
      if ( charValue eq "Financial" ) then do;
         do i = 1 to nChildValues;
            call _rollup_.getInputClassValue (i, iValue, unusedNum);
            do j = 1 to nChildValues;
               call _rollup_.getInputClassValue (j, jValue, unusedNum);
               call pmxelem(FinCovMat, iValue, jValue, covValue);
```

```
                sumTmp = sumTmp +
childSensitivities[i]*childSensitivities[j]*covValue;
             end;
          end;
       end;
       else do;
          do i = 1 to nChildValues;
             call _rollup_.getInputClassValue (i, iValue, unusedNum);
             do j = 1 to nChildValues;
                call _rollup_.getInputClassValue (j, jValue, unusedNum);
                call pmxelem(IndCovMat, iValue, jValue, covValue);
                sumTmp = sumTmp +
childSensitivities[i]*childSensitivities[j]*covValue;
             end;
          end;
       end;
       sumTmp = sqrt(sumTmp);

       call _rollup_.setValue("weightedSensitivity", sumTmp);
   endmethod;
run;

/**************************************************************************/
/* CREATE DATA
/**************************************************************************/
data myLib.myRollupData;
   length CCVar weightedSensitivity $32;
   input  CCVar weightedSensitivity $;
   datalines;
_TOP_     _SUM_
Sector    weightSensitivities
Industry  _SUM_
;
run;

data myLib.instdata;
   length Insttype Instid Sector Industry$32.;
   input InstType $ InstId $ Sector $ Industry $ MtMValue;
   datalines;
Inst1 i1 Financial  Banking      3
Inst1 i2 Financial  Diversified  5
Inst1 i3 Financial  Insurance   -2
Inst1 i4 Industrial Goods        3
Inst1 i5 Industrial Services     1
run;

data myLib.mktdata;
   rf1=1;
run;

/**************************************************************************/
/* CREATE PROJECT
/**************************************************************************/
proc risk;
   env open=myLib.myRollupEnv;

   crossclass SectorIndustry (Sector Industry);
   marketdata Market
      file = myLib.mktdata
      type = current;

   instrument Inst1
      variables = (MtMValue Sector Industry)
      methods = (Price Price1);
   instdata Port
      file = myLib.instdata
      format = simple;
   sources Source (Port);
   read sources = Source out=Read;

   project Project
      portfolio = Read
      crossclass = SectorIndustry
      data = (Market)
      rollupmethods = (weightSensitivities);

  env save;
run;

proc hpexport
   expprojlib = myLib
   env = myLib.myRollupEnv
   project = Project;
run;

/****************************************************************************/
/* CREATE CUBE
/****************************************************************************/
proc hprisk
   expprojlib = myLib
   task = runproject
   project = Project
   cube = "&cube_path/mtmCube"
   filesprefix = "&hprfilesprefix/mtmCube"
   priceby = positions;

   outvars replace=(weightedSensitivity);
   crossclassvars Sector Industry;
   hierarchy mySectorIndustryHierarchy (Sector Industry)
      rollup_data = myLib.myRollupData;
run;
quit;

/****************************************************************************/
/* QUERY CUBE
/****************************************************************************/
proc hprisk
   task = query
   cube = "&cube_path/mtmCube"
   tracelog = yes;

   output summary = myLib.hprSummary
      hierarchy = mySectorIndustryHierarchy;
   query;
   query Sector;
   query Sector Industry;
run;
quit;

proc hprisk
   task = clean
   cube = "&cube_path/mtmCube";
run;

proc print
   data = myLib.hprSummary(keep=Sector Industry NInst weightedSensitivity);
run;
```

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Scott Gray
SAS Institute Inc.
scott.gray@sas.com

Stacey Christian
SAS Institute Inc.
stacey.christian@sas.com