Documentation of the LSRMTools

René Dahms (rene.dahms at math.ethz.ch)

February 17, 2017

Contents

1 General

1.1 Introduction

In this documentation we will describe the functionality of the LSRMTools provided by the files

- LSRMTools.dll (ActiveX component) or
- LSRM_Dll.dll (function based dynamic link library),

both in a 32bit and 64bit version, but the latter have not been tested very much up to now (because I don't have a 64bit version of Excel; but I got responds from others that the 64bit versions work for them). Note, which version, 32bit or 64bit, you should use depends on your Excel version and not on the version of Windows. The functionality can be used by any program that supports ActiveX elements or dynamic link libraries. In order to use it within Excel we provide corresponding sample VBA-interfaces

- LSRM_Tools_ActiveX.xla and
- LSRM_Tools_Dll.xla,

respectively.
Essentially, both ways provide the same functionality, see Figure 1.1. The main difference is that the AxtiveX way provides an object that has to be registered with administrator rights, see Section 1.3.1, whereas the the other provides functions, which only have to be "declared" before they could be used, see Section 1.3.2.
If you want to use the reference implementation for EXCEL we refer to Section 5.2 together with the included examples.

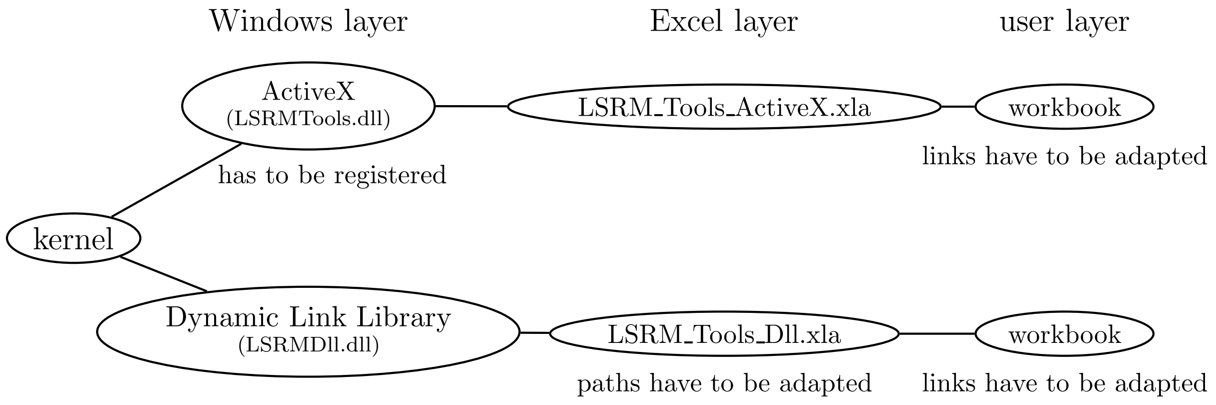

Figure 1: Overview of the program architecture

Note: The shortcut Ctrl + Shift + i is very important, since it (re)initialises *LSRMs*.
Note: If you specify the exposures $R_{i,k}^{m}$ and $R_{i,k}^{m_1,m_2}$ by formulas, see Section 2.2, you may have to change the regional settings to English (UK) or use the property **Delimiters** (if you use the ActiveX component, otherwise it is the function **LSRM_SetDelimiters**), see Section 2.21 in order to change the delimiters. The reason is, that by default the decimal delimiter is taken from your regional settings, which may cause a conflict with the default formula delimiter (;) or the default parameter delimiter (,).

1.2 License

```
LSRMTools is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

LSRMTools is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with LSRMTools.  If not, see <http://www.gnu.org/licenses/>.
```

1.3 Installation

1.3.1 Installation of the ActiveX component

Before the functionality of the ActiveX component can be used it has to be registered. Therefore, you (or someone with the appropriate rights) have to start a command prompt (cmd.exe) with administrator rights. Then you have to switch to the folder where the file LSRMTools.dll is located and run the following command:

regsvr32.exe LSRMTools.dll

In order to unregister the tool you can use the command prompt command

regsvr32.exe /u LSRMTools.dll

Note, if you have a 32bit Excel version you must use the 32bit version of the ActiveX component, and if you have a 64bit Excel version it has to be the 64bit version of the ActiveX component.
Moreover, in order to access the LSRMTools from an Excel macro you have to add a reference within the VBA environment of this Excel file, i.e. go to menu→references and either select the tools from the provided list or search for the file LSRMTools.dll.
We provide a sample Excel interface within the file LSRM_Tools_ActiveX.xla. In order to use it you may have to change the reference to the ActiveX component LSRMTools.dll within the VBA environment of the file LSRM_Tools_ActiveX.xla.

1.3.2 Installation of the function based dynamic link library

Before the functionality of the function based dynamic library can be used you have to "declared" each of its functions. Within Excel this is done by the VBA directive Declare, see Microsoft help pages for more information.
Within the Excel file LSRM_Tools_Dll.xla such declarations have been made within the module DDLFunction_Declaration. In order to use it you have to replace each path with the path of the folder where your copy of the file LSRM_Dll.dll is located.

Note: Recently we have observed some problems relating to saving changes of xla files (the save button in the VBA environment may not save changes at all). Therefore, we have added the procedure SaveXLA to the module DDLFunction_Declaration, which does the saving correctly.

2 Functionality of LSRMTools

Both, the ActiveX component and the function based dynamic library, provides the same functionality. Only the way of accessing is different. The ActiveX component provides an object named *LSRM* that itself provides functions, procedures and (read-only) properties. In order to use them you have to create a new *LSRM* object. In VBA this might be done by

```
Dim MyLSRM as New LSRM
```

For each function, procedure or property the LSRM_Dll.dll provides a corresponding function or procedure. Therefore, we will only document the ActiveX component and use a Pascal syntax to specify functions, procedures and properties.

2.1 Function About(): *Variant*

Returns a string with some information about the version of the tool, the licence and the author.

2.2 Function Init(LSRMName: *Variant*; What, M, I, J, FuturePeriods: *Long*; gamma1, gamma2: *Variant*; S, w, f, sigma, alpha, InflationRate: *Double**): *Variant*

This function (re)initialises a *LSRM* with the provided parameters. If something went wrong the function returns a corresponding error message otherwise an empty string. The parameters are as follows.
Note: The exposure parameters **gamma1** and **gamma2** may or may not depend on accident periods. If both depend on accident periods we call the *LSRM* a StandardLSRM and if both do not depend on accident periods we call it SimplifiedLSRM. **Mixed cases are not allowed!**

LSRMName: Specifies the name of the *LSRM* that should be initialised. If there does not exist a *LSRM* of that name a new *LSRM* will be created.

What: Specifies a field of bits that indicates which parameter has to be (re)initialised. The possible values are shown in Table 1. Moreover, within **What** some calculation flags can be specified, see Table 2.

M: Specifies the number of claim properties.

Constant	Value	Description
cLSRMInitGeometry	2^0	initialises number of claim properties **M**, accident periods **I**, development periods **J**, and tail periods **FuturePeriods**, see Section 3.2
cLSRMInitGamma1	2^1	initialises exposure parameter $\gamma_{i,k,h,j}^{m,l}$
cLSRMInitGamma2	2^2	initialises exposure parameter $\gamma_{i,k,h,j}^{m_1,m_2,l}$
cLSRMInitS	2^3	initialises claim properties $S_{i,k}^m$
cLSRMInitw	2^4	initialises user defined weights $w_{i,k}^{m,I}$ and $w_{i,k}^{m,I+1}$
cLSRMUseDefaultw	2^5	indicates that default weights, see [1, formula (3.3)], should be used
cLSRMInitf	2^6	initialises user defined development factors f_k^m
cLSRMUseDefaultf	2^7	indicates that default development factors, see [1, formula (3.1)], should be used
cLSRMInitsigma	2^8	initialises user defined covariance parameter $\sigma_k^{m_1,m_2}$
cLSRMUseDefaultsigma	2^9	indicates that default covariance parameter, see [1, formula (4.2)], should be used
cLSRMInitalpha	2^{10}	initialises user defined mixing weights α_i^m
cLSRMUseDefaultalpha	2^{11}	indicates that default mixing weights should be used, i.e. $\alpha_i^m \equiv 1$
cLSRMInitInflationRate	2^{12}	initialises inflation rates, see Section 3.1

Table 1: Initialisation constants

I: Specifies the number of accident periods.

J: Specifies the number of development periods without tail periods.

FuturePeriods: Specifies the number of tail development periods. In the following we will refer to the number of future periods by K.

gamma1 Specifies the exposure parameter $\gamma_{i,k,h,j}^{m,l}$ for $0 \le m,\ l \le M$, $0 \le i \le I$ and $0 \le k,\ j \le J - 1 + \textbf{FuturePeriods}$. There are the following three different ways to do that:

- **gamma1** is a two dimensional array, where the second dimension contains the following 7 columns

$$m,\ l,\ i,\ k,\ h,\ j \text{ and } \gamma_{i,k,h,j}^{m,l}.$$

  Zero values for $\gamma_{i,k,h,j}^{m,l}$ do not have to be specified.

- If the exposure parameter $\gamma_{i,k,h,j}^{m,l}$ are independent of i and h those columns can be omitted. Hence **gamma1** is a two dimensional array, where the second dimension contains the following 5 columns

$$m,\ l,\ k,\ j \text{ and } \gamma_{i,k,h,j}^{m,l}.$$

Zero values for $\gamma^{m,l}_{i,k,h,j}$ do not have to be specified.

- If the exposures $R^m_{i,k}$ can be calculated as "linear combination of the triangles $S^m_{i,k}$" then **gamma1** can be specified by a string that contains a formula for each exposure $R^m_{i,k}$. Those formulas have to be separated by semicolons. The following syntax elements are supported:
  - Signs $+$ and $-$
  - Operators $+$, $-$, $*$ and $/$, where the calculation is done element by element
  - Constant real numbers, representing a constant triangle
  - Collection of **J** + **FuturePeriods** real numbers separated by commata and enclosed within square brackets, representing a triangle with constant columns
  - Claim property: $S(m)$, where $0 \leq m \leq \mathbf{M}$
  - Pairs of round brackets to specify the order of calculation
  - Functions:
    - $Cum(A)$: cumulates triangle A
    - $Inc(A)$: takes the increments of A
    - $Column(k, A)$: sets all columns of A to zero, except for the k-th column

  Examples for formulas of $R^m_{i,k}$ are:
  - "S(0)": $R^m_{i,k} := S^0_{i,k}$
  - "2*S(0)-S(1)": $R^m_{i,k} := 2 * S^0_{i,k} - S^i_{i,k}$
  - "Cum(S(0))": $R^m_{i,k} := \sum_{j=0}^{k} S^0_{i,j}$, i.e. the Chain Ladder exposures
  - "Cum(S(1))-Cum(S(0))": $R^m_{i,k} := \sum_{j=0}^{k} S^1_{i,j} - S^0_{i,j}$, if $S^1_{i,k}$ are reported amounts and $S^0_{i,k}$ are payments then $R^m_{i,k}$ represent the corresponding case reserves.
  - "[1,0.5,0,0]*Cum(S(1))-[1,0.5,0,0]*Cum(S(0))+[0,0.5,1,1]*Cum(S(1))": If the total number of development periods $\mathbf{J} + \mathbf{FuturePeriods} + 1$ equals 4 and if $S^1_{i,k}$ are reported amounts and $S^0_{i,k}$ are payments then the formula represents the following exposure:
    - case reserves for the first development period
    - a $50\% - 50\%$ mixture of case reserves and reported amounts for the second development period
    - reported amounts for the last two development periods (Chain-Ladder on reported amounts)

    Note: The default decimal delimiter (defined by the regional settings), the default formula delimiter (;) and the default parameter delimiter (,) can be changed by the property Delimiters, see Section 2.21.

If cLSRMInitGamma1 has been excluded from the parameters **What** the corresponding model parameter **gamma1** will not be reinitialised.

Note: The specifications of **gamma1** and **gamma2** have to be consistent. That means if you specify **gamma1** independent on accident periods the specification of **gamma2** has to be independent of accident periods, too.

gamma2: Specifies the exposures $R_{i,k}^{m_1,m_2}$. There are the following three different ways to do that:

- **gamma2** is a two dimensional array, where the second dimension contains the following 8 columns

$$m_1,\ m_2,\ l,\ i,\ k,\ h,\ j \text{ and } \gamma_{i,k,h,j}^{m,l},$$

where $0 \leq m_1,\ m_2,\ l \leq M$, $0 \leq i \leq I$ and $0 \leq k,\ j \leq J-1+K$. Zero values for $\gamma_{i,k,h,j}^{m_1,m_2,l}$ do not have to be specified.

- If the exposure parameter $\gamma_{i,k,h,j}^{m_1,m_2,l}$ are independent of i and h those columns can be omitted. Hence **gamma2** is a two dimensional array, where the second dimension contains the following 6 columns

$$m_1, m_2,\ l,\ k,\ j \text{ and } \gamma_{i,k,h,j}^{m,l}.$$

Zero values for $\gamma_{i,k,h,j}^{m_1,m_2,l}$ do not have to be specified.

- If the exposures $R_{i,k}^{m_1,m_2}$ can be calculated as "linear combination of the triangles" $S_{i,k}^{m}$ then **gamma2** can be specified by a string that contains a formula for each exposure $R_{i,k}^{m_1,m_2}$. In addition to the specification of formulas for **gamma1** the following syntax elements are supported:
  - Power operator ˆ
  - Functions:
    - *Abs*(A): taking the absolute value of each entry of the triangle A
    - *Sqrt*(A): taking the square root of each entry of the triangle A
    - *Min*(A,B) and *Max*(A,B): element by element minimum and maximum of the triangles A and B

If cLSRMInitGamma2 has been excluded from the parameter **What** the corresponding model parameter **gamma2** will not be reinitialised.

Note: The specification of **gamma1** and **gamma2** have to be consistent. That means if you specify **gamma2** independent on accident periods the specification of **gamma1** has to be independent of accident periods, too.

S: Represents the first entry of a three dimensional array (index by $0 \leq m \leq M$, $0 \leq i \leq I$ and $0 \leq k \leq J$) of float values that defines the claim properties $S_{i,k}^{m}$. Note, the number of elements of each dimension will be taken from the parameters **M**, **I** and **J**. If no claim property has to be reinitialised you may take a dummy variable of type float and exclude cLSRMInitS from the parameter **What**.

w: Represents the first entry of a four dimensional array (index by $\leq m \leq M$, $0 \leq i \leq I$ and $0 \leq k \leq J-1+\mathbf{1}_{K>0}$) of float values that defines the not normalised weights $w_{i,k}^{m,I}$ and $w_{i,k}^{m,I+1}$. By not normalised we mean that the final weights $w_{i,k}^{m,I}$ and $w_{i,k}^{m,I+1}$ are taken from the specified array and are normalised such that the sum over k always equals one. Moreover, during the normalization we set a weight $w_{i,k}^{m,I}$ and $w_{i,k}^{m,I+1}$ to zero if the corresponding exposure $R_{i,k}^{m}$ is zero. Note, the number of elements of each dimension will be taken from the parameters **M**, **I**, **J** and **FuturePeriods**. If no weight has to be reinitialised you may take a dummy variable of type float and exclude cLSRMInitw from the parameter **What**. If cLSRMUseDefaultw is included within the parameter **What** the specified weights will be ignored and the standard weights will be taken, see [1, formula (3.3)].

f: Represents the first entry of a two dimensional array (index by $m \leq M$ and $0 \leq k \leq J - 1 + K$) of float values that defines the development factors f_k^m. Note, the number of elements of each dimension will be taken from the parameters **M** and **J** + **FuturePeriods** − 1. If no development factor has to be reinitialised you may take a dummy variable of type float and exclude cLSRMInitf from the parameter **What**. If cLSRMUseDefaultf is included within the parameter **What** the specified development factors will be ignored and the standard weighted estimators, using the active weights, will be taken, see [1, formula (3.1)].

sigma: Represents the first entry of a three dimensional array (index by $0 \leq m_1, m_2 \leq M$ and $0 \leq k \leq J - 1 + \mathbf{1}_{K>0}$) of float values that defines the covariance parameters $\sigma_k^{m_1,m_2}$. Note, the number of elements of each dimension will be taken from the parameters **M**, **J** and **FuturePeriods**. If no covariance parameter has to be reinitialised you may take a dummy variable of type float and exclude cLSRMInitsigma from the parameter **What**. If cLSRMUseDefaultsigma is included within the parameter **What** the specified variance parameters will be ignored and the standard unbiased estimators, using the active weights, will be taken, see [1, formula (4.2)].

alpha: Represents the first entry of a two dimensional array (index by $0 \leq m \leq M$ and $0 \leq i \leq I$) of float values that defines the mixing weights α_i^m. Note, the number of elements of each dimension will be taken from the parameters **M** and **I**. If no mixing weight has to be reinitialised you may take a dummy variable of type float and exclude cLSRMInitalpha from the parameter **What**. If cLSRMUseDefaultalpha is included within the parameter **What** the specified mixing weights will be ignored and we take $\alpha_i^m \equiv 1$.

InflationRate Represents the first entry of a three dimensional array (index by $0 \leq m \leq M$, $0 \leq i \leq I$ and $0 \leq k \leq J + K$) of float values that defines inflation rates. Note, the number of elements of each dimension will be taken from the parameters **M**, **I** and **J** + **FuturePeriods**. If no inflation rate has to be reinitialised you may take a dummy variable of type float and exclude cLSRMInitInflationRate from the parameter **What**. For technical details see, Section 3.1.

2.3 Procedure Reset(LSRMName: *Variant*)

Deletes, if found, the *LSRM* with the specified name **LSRMName**. If **LSRMName** is an empty string all *LSRMs* will be deleted.

2.4 Property alpha(LSRMName: *Variant*; m, i: *Long*): *Variant*; read-only

Returns the value of the used mixing weight α_i^m for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

2.5 Property CalcInfo(LSRMName: *Variant*; Step, InfoType: *Long*): *Variant*; read-only

Returns if the internal calculation step **Step** has been done and how long it took, for more information about the internal calculation engine see Section 4. The parameter **InfoType** corresponds to:

0: The name of the step.

1: TRUE or FALSE for whether or not the calculation step has been calculated.

2: The time for the calculation step in milliseconds.

If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

2.6 Property CalculationFlags(LSRMName: *Variant*): *Variant*; read-only

Returns whether or not a calculation flag, see Table 2, has been set. The *LSRM* with addressed by its name **LSRMName**. If there does not exist such a *LSRM* or if something went wrong a corresponding error message will be returned.

2.7 Property CDR(LSRMName: *Variant*; m, i, Proxy, Part, Tail: *Long*): *Variant*; read-only

Returns the value of the one year uncertainty of the claims development result (CDR) for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

LSRMName: Specifies the *LSRM*.

m: Specifies the claim property. If $\mathbf{m} \notin \{0, \dots, M\}$ the overall CDR uncertainty weighted by the mixing weights α_i^m will be returned.

i: Specifies the accident period. If $\mathbf{i} \notin \{0, \dots, I\}$ the total CDR uncertainty of claim property **m** will be returned.

Proxy: Specifies whether (**Proxy** $= 1$) or not (**Proxy** $= 0$) the approximation of the operator $\mathbf{H}()$ should be taken.

Part: Specifies which part of the CDR uncertainty should be taken, i.e. the process variance (**Part** =0), the parameter error (**Part** =1) or the sum of both (**Part** = 2).

Tail: Specifies how to deal with the tail uncertainty of the CDR. If **Tail** = 0 it will be ignored, if **Tail** = 1 it will be taken into account and if **Tail** = 2 only the tail uncertainty will be returned.

2.8 Property CDRCov(LSRMName: *Variant*; m1, m2, Proxy, Part, Tail: *Long*): *Variant*; read-only

Returns the (**m1**,**m2**) entry of the covariance matrix associated with the one period uncertainty of the CDR, i.e. the corresponding addend of the summation in [1, Estimator 5.6]. For more detailed information about this topic see Section 3.3. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

LSRMName: Specifies the *LSRM*.

m1 and **m2:** Specify the addend of the summation.

Proxy: Specifies whether (**Proxy** = 1) or not (**Proxy** = 0) the approximation of the operator **H**() should be taken.

Part: Specifies which part of the CDR uncertainty should be taken, i.e. the process variance (**Part** =0), the parameter error (**Part** =1) or the sum of both (**Part** = 2).

Tail: Specifies whether (**Tail** = 1) or not (**Tail** = 0) tail uncertainty of the CDR should be taken into account.

2.9 Property CDRCovEigenvalue(LSRMName: *Variant*; m, Proxy, Part, Tail: *Long*): *Variant*; read-only

Returns the **m**-th largest eigenvalue of the covariance matrix associated with the one period uncertainty of the CDR, see Section 3.3. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

LSRMName: Specifies the *LSRM*.

m: Specifies which eigenvalue should be returned. Eigenvalues are arranged in ascending order and numbering starts with zero, i.e. **m** = 0 corresponds to the smallest eigenvalue.

Proxy: Specifies whether (**Proxy** = 1) or not (**Proxy** = 0) the approximation of the operator **H**() should be taken.

Part: Specifies which part of the CDR uncertainty should be taken, i.e. the process variance (**Part** =0), the parameter error (**Part** =1) or the sum of both (**Part** = 2).

Tail: Specifies whether (**Tail** = 1) or not (**Tail** = 0) the tail uncertainty of the CDR should be taken into account.

2.10 Property CDRCovEigenvector(LSRMName: *Variant*; m, l, Proxy, Part, Tail: *Long*): *Variant*; read-only

Returns the **l**-th coordinate of the eigenvector corresponding to the **m**-th largest eigenvalue of the covariance matrix associated with the one period uncertainty of the CDR, see Section 3.3. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

LSRMName: Specifies the *LSRM*.

m: Specifies the eigenvector that should be returned (numberings starts with zero). The eigenvectors are in the same order like the corresponding eigenvalues, which are arranged in ascending order. For example **m** =3 stands for the eigenvector of the fourth smallest eigenvalue.

l: Specifies which coordinate of the eigenvector that should be returned.

Proxy: Specifies whether (**Proxy** = 1) or not (**Proxy** = 0) the approximation of the operator **H**() should be taken.

Part: Specifies which part of the CDR uncertainty should be taken, i.e. the process variance (**Part** =0), the parameter error (**Part** =1) or the sum of both (**Part** = 2).

Tail: Specifies whether (**Tail** = 1) or not (**Tail** = 0) the tail uncertainty of the CDR should be taken into account.

2.11 Property CDRDetail(LSRMName: *Variant*; m1, m2, i1, i2: *Long*; Proxy, Part, Tail: *Long*): *Variant*; read-only

Returns the addends of the summation in [1, Estimator 5.6], i.e. the formula for the calculation of the overall uncertainty of the CDR. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

LSRMName: Specifies the *LSRM*.

m1, m2, i1 and **i2:** Specify the addend of the summation.

Proxy: Specifies whether (**Proxy** = 1) or not (**Proxy** = 0) the approximation of the operator **H**() should be taken.

Part: Specifies which part of the CDR uncertainty should be taken, i.e. the process variance (**Part** =0) or the parameter error (**Part** =1).

Tail: Specifies how to deal with tail uncertainty of the CDR. If **Tail** = 0 it will be ignored, if **Tail** = 1 it will be taken into account and if **Tail** = 2 only the tail uncertainty will be returned.

2.12 Property CoorF(LSRMName: *Variant*; m, l, i, k, h, j: *Long*): *Variant*; read-only

Returns the coordinate $\left(F^m_{i,k}\right)^l_{h,j}$ of the operator $F^m_{i,k}$, see [1, formula (3.5)], for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

2.13 Property Defaultalpha(LSRMName: *Variant*; m, i: *Long*): *Variant*; read-only

Returns the value of the default mixing weights $\alpha^m_i \equiv 1$ for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

2.14 Property DefaultalphaUsed(LSRMName: *Variant*): *Variant*; read-only

Returns whether or not default mixing weights α^m_i are used for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM* or if something went wrong a corresponding error message will be returned.

2.15 Property Defaultf(LSRMName: *Variant*; m, k: *Long*): *Variant*; read-only

Returns the value of the default development factor f^m_k, see [1, formula (3.1)], for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

2.16 Property DefaultfUsed(LSRMName: *Variant*): *Variant*; read-only

Returns whether or not default development factor f^m_k, see [1, formula (3.1)], are used for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM* or if something went wrong a corresponding error message will be returned.

2.17 Property Defaultsigma(LSRMName: *Variant*; m1, m2, k: *Long*): *Variant*; **read-only**

Returns the value of the default covariance parameter $\sigma_k^{m_1,m_2}$, see [1, formula (4.2)], for the *LSRM* with name **LSRMName**. If the calculation flag cLSRMUseWeightsForDefaultSigma has not been set, see function **Init** in Section 2.2, the default covariance parameters will be calculated by formula [1, formula (4.2)], but with $R_{i,k}^m$ instead of $w_{i,k}^m$ (the normalising constant will be adapted accordantly). If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

2.18 Property DefaultsigmaUsed(LSRMName: *Variant*): *Variant*; **read-only**

Returns whether or not default covariance parameter $\sigma_k^{m_1,m_2}$, see [1, formula (4.2)], are used for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM* or if something went wrong a corresponding error message will be returned.

2.19 Property Defaultw(LSRMName: *Variant*; n, m, i, k: *Long*): *Variant*; **read-only**

Returns the value of the default weight $w_{i,k}^{m,I}$ or $w_{i,k}^{m,I+1}$ for the *LSRM* with name **LSRMName**.
If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.
The values of **n** have the following meaning:

n $= 0$**:** Not normalised weights are returned, i.e. $w_{i,k}^m = R_{i,k}^m$ for $i + k \leq I$.

n $= 1$**:** Normalised weights $w_{i,k}^{m,I}$ for observing period I are returned, see [1, formula (3.3)].

n $= 2$**:** Normalised weights $w_{i,k}^{m,I+1}$ for observing period $I+1$ are returned, see [1, Assumption 5.1].

2.20 Property DefaultwUsed(LSRMName: *Variant*): *Variant*; **read-only**

Returns whether or not default weights, see [1, formula (3.3) and Assumption 5.1], are used for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM* or if something went wrong a corresponding error message will be returned.

2.21 Property Delimiters(): *Variant*; **read-write**

A string that contains the decimal delimiter, the formula delimiter and the parameter delimiter in this order. By default the regional setting is used to set the decimal delimiter. The default values for the formula delimiter and the parameter delimiter are semicolon and comma, respectively.

All delimiters have to be different from each other. Letters and digits are not allowed. Moreover the following characters are forbidden: +-*/ˆ()[].

2.22 Property Eigenvalue(LSRMName: *Variant*; i, k, m: *Long*): *Variant*; read-only

Returns the **m**-th largest eigenvalue of the covariance matrix $\left(\sigma_k^{m_1,m_2} R_{i,k}^{m_1,m_2}\right)_{m_1,m_2}$ for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

2.23 Property Eigenvector(LSRMName: *Variant*; i, k, m, l: *Long*): *Variant*; read-only

Returns the **l**-th coordinate of the eigenvector corresponding to the **m**-th largest eigenvalue of the covariance matrix $\left(\sigma_k^{m_1,m_2} R_{i,k}^{m_1,m_2}\right)_{m_1,m_2}$ for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

2.24 Property f(LSRMName: *Variant*; m, k: *Long*): *Variant*; read-only

Returns the used development factor f_k^m for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

2.25 Property I(LSRMName: *Variant*): *Variant*; read-only

Returns the number of accident periods I for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM* or if something went wrong a corresponding error message will be returned.

2.26 Property InitialisedParameters(LSRMName: *Variant*): *Variant*; read-only

Returns a bit field that indicates which parameters have been initialised for the *LSRM* with name **LSRMName**. All entries of Table 1 except for cLSRMUseDefaultAlpha, cLSRMUseDefaultf, cLSRMUseDefaultsigma and cLSRMUseDefaultw are possible flags. If there does not exist such a *LSRM* or if something went wrong a corresponding error message will be returned.

2.27 Property J(LSRMName: *Variant*): *Variant*; read-only

Returns the number of development periods J for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM* or if something went wrong a corresponding error message will be returned.

2.28 Property K(LSRMName: *Variant*): *Variant*; **read-only**

Returns the number of future development (tail) periods K for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM* or if something went wrong a corresponding error message will be returned.

2.29 Property MaxVarCoef1(): *Double*; **read-write**

Specifies the variable **MaxVarCoef1** that is used for the estimation of the default covariance parameters $\sigma_k^{m_1,m_2}$, see Section 3.4.
Note: Changing this value will affect the (re-)estimation of the covariance parameters $\sigma_k^{m_1,m_2}$ of every *LSRM*.

2.30 Property MaxVarCoef2(): *Double*; **read-write**

Specifies the the variable **MaxVarCoef2** that is used for the estimation of the default covariance parameters, see Section 3.4.
Note: Changing this value will affect the (re-)estimation of the covariance parameters $\sigma_k^{m_1,m_2}$ of every *LSRM*.

2.31 Property M(LSRMName: *Variant*): *Variant*; **read-only**

Returns the number of claim properties M for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM* or if something went wrong a corresponding error message will be returned.

2.32 Property ModelType(LSRMName: *Variant*): *Variant*; **read-only**

Returns the type of the *LSRM* with name **LSRMName**. Values are as follows strings:

- "Normal LSRM": The exposure parameter $\gamma_{i,k,h,j}^{m,l}$ and $\gamma_{i,k,h,j}^{m_1,m_2,l}$ might depend on accident periods i and h.
- "Simplified LSRM without inflation": The exposure parameter $\gamma_{i,k,h,j}^{m,l}$ and $\gamma_{i,k,h,j}^{m_1,m_2,l}$ do not depend on accident periods i and h and we do not have any inflation.
- "Simplified LSRM with inflation": The exposure parameter $\gamma_{i,k,h,j}^{m,l}$ and $\gamma_{i,k,h,j}^{m_1,m_2,l}$ do not depend on accident periods i and h and we might have inflation.

For more detailed information see Section 4. If there does not exist such a *LSRM* or if something went wrong a corresponding error message will be returned.

2.33 Property MSEP(LSRMName: *Variant*; m, i, Proxy, Part, Tail: *Long*): *Variant*; read-only

Returns the value of the ultimate uncertainty, i.e. the mean squared error of prediction (MSEP), for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

LSRMName: Specifies the *LSRM*.

m: Specifies the claim property. If $\mathbf{m} \notin \{0, \dots, M\}$ the overall MSEP weighted by the mixing weights α_i^m will be returned.

i: Specifies the accident period. If $\mathbf{i} \notin \{0, \dots, I\}$ the total MSEP of claim property **m** will be returned.

Proxy: Specifies whether (**Proxy** = 1) or not (**Proxy** = 0) the approximation of the operator **H**() should be taken.

Part: Specifies which part of the MSEP should be taken, i.e. the process variance (**Part** = 0), the parameter error (**Part** = 1) or the sum of both (**Part** = 2).

Tail: Specifies how to deal with the tail uncertainty. If **Tail** = 0 it will be ignored, if **Tail** = 1 it will be taken into account and if **Tail** = 2 only the tail uncertainty will be returned.

2.34 Property MSEPCov(LSRMName: *Variant*; m1, m2, Proxy, Part, Tail: *Long*): *Variant*; read-only

Returns the (**m1**,**m2**) entry of the covariance matrix associated with the ultimate uncertainty, i.e. the corresponding addend of the summation in [1, Estimator 4.11]. For more detailed information about this topic see Section 3.3. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

LSRMName: Specifies the *LSRM*.

m1 and **m2:** Specify the addend of the summation.

Proxy: Specifies whether (**Proxy** = 1) or not (**Proxy** = 0) the approximation of the operator **H**() should be taken.

Part: Specifies which part of the MSEP should be taken, i.e. the process variance (**Part** =0), the parameter error (**Part** =1) or the sum of both (**Part** = 2).

Tail: Specifies whether (**Tail** = 1) or not (**Tail** = 0) the tail uncertainty should be taken into account.

2.35 Property MSEPCovEigenvalue(LSRMName: *Variant*; m, Proxy, Part, Tail: *Long*): *Variant*; read-only

Returns the **m**-th largest eigenvalue of the covariance matrix associated with the ultimate uncertainty, see Section 3.3. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

LSRMName: Specifies the *LSRM*.

m: Specifies which eigenvalue should be returned. Eigenvalues are arranged in ascending order and numbering starts with zero, i.e. **m** = 0 corresponds to the smallest eigenvalue.

Proxy: Specifies whether (**Proxy** = 1) or not (**Proxy** = 0) the approximation of the operator **H**() should be taken.

Part: Specifies which part of the MSEP should be taken, i.e. the process variance (**Part** =0), the parameter error (**Part** =1) or the sum of both (**Part** = 2).

Tail: Specifies whether (**Tail** = 1) or not (**Tail** = 0) the tail uncertainty should be taken into account.

2.36 Property MSEPCovEigenvector(LSRMName: *Variant*; m, l, Proxy, Part, Tail: *Long*): *Variant*; read-only

Returns the **l**-th coordinate of the eigenvector corresponding to the **m**-th largest eigenvalue of the covariance matrix associated with the ultimate uncertainty, see Section 3.3. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

LSRMName: Specifies the *LSRM*.

m: Specifies the eigenvector that should be returned (numberings starts with zero). The eigenvectors are in the same order like the corresponding eigenvalues, which are arranged in ascending order. For example **m** =3 stands for the eigenvector of the fourth smallest eigenvalue.

l: Specifies which coordinate of the eigenvector should be returned.

Proxy: Specifies whether (**Proxy** = 1) or not (**Proxy** = 0) the approximation of the operator **H**() should be taken.

Part: Specifies which part of the MSEP should be taken, i.e. the process variance (**Part** =0), the parameter error (**Part** =1) or the sum of both (**Part** = 2).

Tail: Specifies whether (**Tail** = 1) or not (**Tail** = 0) the tail uncertainty should be taken into account.

2.37 Property MSEPDetail(LSRMName: *Variant*; m1, m2, i1, i2: *Long*; Proxy, Part, Tail: *Long*): *Variant*; read-only

Returns the addends of the summation in [1, Estimator 4.11], i.e. the formula for the calculation of the overall ultimate uncertainty. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

LSRMName: Specifies the *LSRM*.

m1, m2, i1 and **i2:** Specify the addend of the summation.

Proxy: Specifies whether (**Proxy** = 1) or not (**Proxy** = 0) the approximation of the operator **H**() should be taken.

Part: Specifies which part of the MSEP should be taken, i.e. the process variance (**Part** =0) or the parameter error (**Part** =1).

Tail: Specifies how to deal with tail uncertainty. If **Tail** = 0 it will be ignored, if **Tail** = 1 it will be taken into account and if **Tail** = 2 only the tail uncertainty will be returned.

2.38 Property R1(LSRMName: *Variant*; m, i, k: *Long*): *Variant*; read-only

Returns the value of the exposure $R^{m}_{i,k}$ for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

2.39 Property R2(LSRMName: *Variant*; m, i, k: *Long*): *Variant*; read-only

Returns the value of the exposure $R^{m_1,m_2}_{i,k}$ for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

2.40 Property S(LSRMName: *Variant*; m, i, k: *Long*): *Variant*; read-only

Returns the value of the claim property $S^{m}_{i,k}$ for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

2.41 Property sigma(LSRMName: *Variant*; m1, m2, k: *Long*): *Variant*; read-only

Returns the value of the used covariance parameter $\sigma_k^{m_1,m_2}$ for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

2.42 Property Ultimate(LSRMName: *Variant*; m, i, Tail: *Long*): *Variant*; read-only

Returns the value of the ultimate outcome, i.e. $\sum_j S_{i,j}^m$, for the *LSRM* with name **LSRMName**. If $\mathbf{i} \notin \{0, \ldots, M\}$ the total ultimate over all accident periods will be returned. If $\mathbf{m} \notin \{0, \ldots, M\}$ the mixing weights α_i^m are used to get the overall ultimate of all claim properties. The parameter **Tail** specifies whether (**Tail** $= 1$) or not (**Tail** $= 0$) any tail development should be taken into account. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.

2.43 Property w(LSRMName: *Variant*; n, m, i, k: *Long*): *Variant*; read-only

Returns the value of the used weight $w_{i,k}^{m,I}$ or $w_{i,k}^{m,I+1}$ for the *LSRM* with name **LSRMName**. If there does not exist such a *LSRM*, if something went wrong or if some indices are out of range a corresponding error message will be returned.
The values of **n** have the following meaning:

n $= 0$**:** Not normalised weights are returned, i.e. $w_{i,k}^m = R_{i,k}^m$ for $i + k \leq I$.

n $= 1$**:** Normalised weights $w_{i,k}^{m,I}$ for observing period I are returned.

n $= 2$**:** Normalised weights $w_{i,k}^{m,I+1}$ for observing period $I + 1$ are returned.

3 Technical notes

The implementation of LSRMTools is based on the paper [1]. Unfortunately, it contains some errors. They are not critical for the main results but for the implementation. Therefore, we added a revised version of the paper.
Moreover, in the implementation we added three features. The first deals with inflation, see Section 3.1, and the second with tails, see Section 3.2, and the third with covariances of the estimated uncertainties, see Section 3.3. Moreover, we put a bit more effort in the calculation of exposures, see Section 3.6, in the estimation of the standard covariance parameters $\sigma_k^{m_1,m_2}$, see Section 3.4, and in the estimation of uncertainties, see Section 3.5.

3.1 *LSRMs* and inflation

It is well known, that inflation of payments does not go along with many reserving methods. In order to handle them anyhow one usually makes an assumption on the inflation rates, deflate the payments, projects them and inflate the resulting cash flows again. This works for the estimation of IBNR, reserves or ultimates. But how to inflate the estimated uncertainties?
In the following we will present how this can be solved with *LSRMs*. Therefore, assume the claim properties $S^m_{i,k}$ satisfy the assumptions of a *LSRM* with development factors f^m_k, covariance parameters σ^{m_1,m_2}_k and exposures $R^m_{i,k} = \Gamma^m_{i,k}\mathbf{S}^{i+k}$ and $R^{m_1,m_2}_{i,k} = \Gamma^{m_1,m_2}_{i,k}\mathbf{S}^{i+k}$. Then we have:

$$\begin{aligned}\mathrm{E}\big[S^m_{i,k+1}\big|\mathcal{D}_k\big] &= f^m_k R^m_{i,k} \qquad \text{and}\\ \mathrm{Cov}\big[S^{m_1}_{i,k+1}, S^{m_2}_{i,k+1}\big|\mathcal{D}_k\big] &= \sigma^{m_1,m_2}_k R^{m_1,m_2}_{i,k}.\end{aligned}$$

Now assume we know the inflation $r^m_{i,k}$. That means we want to study the inflated claim properties

$$\widetilde{S}^m_{i,k} := r^m_{i,k} S^m_{i,k}.$$

Then we get

$$\mathrm{E}\Big[\widetilde{S}^m_{i,k+1}\Big|\mathcal{D}_k\Big] = r^m_{i,k+1}\mathrm{E}\big[S^m_{i,k+1}\big|\mathcal{D}_k\big] = f^m_k r^m_{i,k+1}\Gamma^m_{i,k}\frac{\mathbf{S}^{i+k}}{\mathbf{r}^{i+k}},$$

where the quotient of the two vectors $\mathbf{S}^{i+k}$ and $\mathbf{r}^{i+k}$ is taken component by component.
In the same way we conclude

$$\mathrm{Cov}\Big[\widetilde{S}^{m_1}_{i,k+1}, \widetilde{S}^{m_2}_{i,k+1}\Big|\mathcal{D}_k\Big] = \sigma^{m_1,m_2}_k r^{m_1}_{i,k+1} r^{m_2}_{i,k+1}\Gamma^{m_1,m_2}_{i,k}\frac{\mathbf{S}^{i+k}}{\mathbf{r}^{i+k}}.$$

Therefore, the inflated claim properties satisfy the assumptions on a *LSRM* with exposures

$$\widetilde{R}^m_{i,k} := r^m_{i,k+1}\Gamma^m_{i,k}\frac{\mathbf{S}^{i+k}}{\mathbf{r}^{i+k}} \qquad \text{and} \qquad \widetilde{R}^{m_1,m_2}_{i,k} := r^{m_1}_{i,k+1} r^{m_2}_{i,k+1}\Gamma^{m_1,m_2}_{i,k}\frac{\mathbf{S}^{i+k}}{\mathbf{r}^{i+k}}$$

and the same model parameters f^m_k and $\sigma^{m_1 m_2}_k$ like the original *LSRM*.
We implemented inflations according to those formulas.

3.2 *LSRMs* with Tails

We implemented the handling of tails. Therefore, we introduce the number of future (or tail) periods K, which are specified within the parameter **FuturePeriods** of the function **Init**, see Section 2.2. In case of $K > 0$ we

- implement the projection of claim properties for the whole time horizon of the development, i.e. $S^m_{i,k}$, for all $0 \le k \le J + K$.
- estimate the ultimate and the one period uncertainties for one additional tail period.

The projection of claim properties only requires tail information within the exposure parameters $\gamma_{i,k,h,j}^{m,l}$ and within the development factors f_k^m. The first is usually given by the choice of the reserving method. But for the development parameters we need manual inputs via the parameters **f** of the function **Init**, see Section 2.2. If no such manual development factors are specified zero is taken as default value.
The estimation of uncertainties is a bit more tricky. The main problem is how to estimate the covariance parameters $\sigma_k^{m_1,m_2}$ for all tail periods $J \le k < J + K$. One possible way to try this is to follow the suggestion of T. Mack and to take an exponential decay. But without some more technical effort this only works for variances and not for covariance matrices. Therefore, we decided to "collect" the whole tail uncertainty within one additional development period and take by default

$$\sigma_J^{m_1,m_2} := \sigma_{J-1}^{m_1,m_2}.$$

Manual covariance parameters can be specified by the parameter **sigma** of the function **Init**, see Section 2.2.
Moreover, the parameters $\varrho_J^{*\,m_1,m_2}$ are not well defined by [1, formula (4.6)]. Therefore, we use the following approximation:

$$\varrho_J^{*\,m_1,m_2} := \varrho_{J-1}^{*\,m_1,m_2} \frac{\sqrt{\sigma_J^{m_1,m_1}\sigma_J^{m_2,m_2}R_{0,J}^{m_1,m_1}R_{0,J}^{m_2,m_2}}R_{0,J-1}^{m_1}R_{0,J-1}^{m_2}}{\sqrt{\sigma_{J-1}^{m_1,m_1}\sigma_{J-1}^{m_2,m_2}R_{0,J-1}^{m_1,m_1}R_{0,J-1}^{m_2,m_2}}R_{0,J}^{m_1}R_{0,J}^{m_2}}.$$

Unfortunately, this approximation may lead to larger solvency uncertainty than ultimate uncertainty. By setting the flag cLSRMUseMSEPifTailCDRisLarger (function **Init**) we can bound the tail solvency uncertainty by the corresponding ultimate uncertainty.
Then the tail uncertainties are estimated by additional addends for the period J in the approximated formula [1, Estimator 4.11] (or [1, Estimator 5.6]).
Note: We only adapted the formulas of the non-tail case to get some estimation for the tail uncertainties.

3.3 Estimation of resulting covariance matrices

If you cancel the summation over m_1 and m_2 in the formulas for the estimation of uncertainties (see [1, Estimator 4.11] and [1, Estimator 5.6]) you automatically get a matrix that can be interpreted as covariance matrix of the corresponding uncertainties. This is the way we implemented the properties **CDRCov** and **MSEPCov**. In order to analyse non-negative definiteness of those matrices we added the corresponding properties **CDRCovEigenvalue**, **CDRCovEigenvector**, **MSEPCovEigenvalue** and **MSEPCovEigenvector**.

3.4 Estimation of the default covariance parameters

In order to estimate the development parameters f_k^m we often use manual weights $w_{i,k}^m$ to exclude some observed strange development. The question now is if we should use those manual weights for the estimation of the covariance parameters, too? If the calculation

flag cLSRMUseWeightsForDefaultSigma is set within the parameter **What** of the function **Init** we will use the same weights as for the estimation of the development factors f_k^m otherwise we will always use the default weights.
In general we have the problem that our model is statistically over-parametrised for late development periods if we do not have enough accident periods. Moreover, this problem gets bigger and bigger the more claim properties we have. We do not see a standard method to compensate for this.
Nevertheless, we implemented a "smooth" way of estimating standard covariance parameters $\sigma_k^{m_1,m_2}$ for cases where the formula [1, formula (4.2)] is not well defined. We use the "exponential decay" approach of T. Mack to estimate the variances (i.e. the diagonals of the covariance matrices) and projected the other covariance entries proportional to the decay of the corresponding variances:

$$\sigma_{k+1}^{m_1,m_2} := \sigma_k^{m_1,m_2} \sqrt{\frac{\sigma_{k+1}^{m_1,m_1}\sigma_{k+1}^{m_2,m_2}}{\sigma_k^{m_1,m_1}\sigma_k^{m_2,m_2}}}, \qquad \text{for } m_1 \neq m_2.$$

Moreover, computing the default covariance parameters according to [1, formula (4.2)] we may get normalising constants Z which should be equal to zero but differ slightly from zero, because of numerical artefacts. Therefore we introduced two global variables **MaxVarCoef1** and **MaxVarCoef2** and set the variance

$$\widehat{\sigma}_k^{m_1,m_2} := 0$$

if

$$|\widehat{\sigma}_k^{m_1,m_2}| > \left| \widehat{f}_k^{m_1} \widehat{f}_k^{m_2} \max_{0 \leq i \leq I-k-1} \left\{ \max(R_{i,k}^{m_1}, R_{i,k}^{m_1}) \right\} \right| \mathbf{MaxVarCoef1} + \mathbf{MaxVarCoef2}.$$

The default values for both variables are $100'000'000$. They can be changed by the properties **MaxVarCoef1** and **MaxVarCoef1**.

3.5 Estimation of uncertainties

In some cases, for instance a Chain Ladder projection of reported amounts that decrease rapidly at the end, you may observe that the estimated uncertainties decrease if you add another development period. In order to explain this behaviour we take a look at the approximation of the uncertainties, see [1, Estimator 4.11 and Estimator 5.6]. Here addends may be negative, because the product of the coordinates of the operator **F** is negative.
In order to avoid such decrease in uncertainties we introduced the flags cLSRMVirtual-ParaErrDiversification, cLSRMVirtualProcVarDiversification and cLSR-MVirtualTailDiversification, which can be specified within the parameter **What** of the function **Init**. If the flag is set such negative addends will be ignored within the approximated estimation of the parameter error, process variance and tail uncertainties, respectively.
Note: Although for the tail uncertainties the estimators with and without approximation are the same the flag cLSRMVirtualTailDiversification only applies to the estimator with approximation, see Section 3.5.

3.6 Calculation of exposures

Sometimes inconsistent data or estimation errors lead to exposures $R_{i,k}^{m}$ and $R_{i,k}^{m_1,m_2}$ that are negative (or positive) instead or zero. We introduced the flags cLSRMPreventFutureRFromChangingSign and cLSRMPreventPastRFromChangingSign, which can be specified within the parameter **What** of the function **Init**. If such a flag is set we stop the decrease (or increase) of known or projected exposures, respectively, at zero. That means if $R_{i,k}^{m}$ (or $R_{i,k}^{m_1,m_2}$) has a different sign than the first $R_{i,j}^{m} \neq 0$, $j \leq k$, (or $R_{i,j}^{m_1,m_2}$), we set it to zero.
Note: Doing so we leave the theoretical framework of *LSRMs* (at least for the exposures $R_{i,k}^{m}$), since such a maximum (or minimum) with zero is not a linear function of claim properties!

4 Notes about the implementation

LSRMTools have been programmed within Delphi. We have tried to speed up calculations. Therefore, we broke down the calculation into small steps and implemented an algorithm that ensures that recalculation of a step will only occur if necessary. In order to do so we have to keep provisional results in memory, which lead to restrictions of the complexity of reserving models. If the parameter **What** of the function **Init** contains the flag cLSRMReduceMemSize we do not store all of the provisional results in memory at the cost of that only approximated uncertainties can be estimated.
Although most of the parameters and results are indexed by more than one index we use one dimensional arrays of float values in order to store them. This speeds up calculations in comparison to using multidimensional arrays. But we are sure that the calculation can be speeded up much more, for instance by using multiple cores and GPUs instead of CPUs.
Section 4.1 contains short descriptions about the calculation steps and [2] gives an overview about their dependencies.

4.1 Initialisation steps

Init_alpha: Initialises user defined mixing weights α_i^m.

Init_CDR: Calculates the total one period uncertainty without tail.

Init_CDRCov: Calculates the covariance matrix of the one period uncertainty.

Init_CDRCovEigenvalue: Calculates eigenvalues and eigenvectors of the covariance matrix of the one period uncertainty.

Init_CDRCovEigenvalueParaErr: Calculates eigenvalues and eigenvectors of the covariance matrix of the one period parameter error.

Init_CDRCovEigenvalueParaErrProxy: Calculates eigenvalues and eigenvectors of the covariance matrix of the approximated one period parameter error.

Init_CDRCovEigenvalueParaErrTail: Calculates eigenvalues and eigenvectors of the covariance matrix of the one period parameter error with tail.

Init_CDRCovEigenvalueParaErrTailProxy: Calculates eigenvalues and eigenvectors of the covariance matrix of the approximated one period parameter error with tail.

Init_CDRCovEigenvalueProcVar: Calculates eigenvalues and eigenvectors of the covariance matrix of the one period process variance.

Init_CDRCovEigenvalueProcVarProxy: Calculates eigenvalues and eigenvectors of the covariance matrix of the approximated one period process variance.

Init_CDRCovEigenvalueProcVarTail: Calculates eigenvalues and eigenvectors of the covariance matrix of the one period process variance with tail.

Init_CDRCovEigenvalueProcVarTailProxy: Calculates eigenvalues and eigenvectors of the covariance matrix of the approximated one period process variance with tail.

Init_CDRCovEigenvalueProxy: Calculates eigenvalues and eigenvectors of the covariance matrix of the approximated one period uncertainty.

Init_CDRCovEigenvalueTail: Calculates eigenvalues and eigenvectors of the covariance matrix of the one period uncertainty with tail.

Init_CDRCovEigenvalueTailProxy: Calculates eigenvalues and eigenvectors of the covariance matrix of the approximated one period uncertainty with tail.

Init_CDRCovParaErr: Calculates the covariance matrix of the one period parameter error.

Init_CDRCovParaErrProxy: Calculates the covariance matrix of the approximated one period parameter error.

Init_CDRCovParaErrTail: Calculates the covariance matrix of the one period parameter error with tail.

Init_CDRCovParaErrTailProxy: Calculates the covariance matrix of the approximated one period parameter error with tail.

Init_CDRCovProcVar: Calculates the covariance matrix of the one period process variance.

Init_CDRCovProcVarProxy: Calculates the covariance matrix of the approximated one period process variance.

Init_CDRCovProcVarTail: Calculates the covariance matrix of the one period process variance with tail.

Init_CDRCovProcVarTailProxy: Calculates the covariance matrix of the approximated one period process variance with tail.

Init_CDRCovProxy: Calculates the covariance matrix of the approximated one period uncertainty.

Init_CDRCovTail: Calculates the covariance matrix of the one period uncertainty with tail.

Init_CDRCovTailProxy: Calculates the covariance matrix of the approximated one period uncertainty with tail.

Init_CDRParaErr: Calculates the one period parameter error.

Init_CDRParaErrDetail: Estimates the one period parameter error detail.

Init_CDRParaErrProxy: Calculates the approximated one period parameter error.

Init_CDRParaErrProxyDetail: Estimates the approximated one period parameter error detail.

Init_CDRParaErrTail: Calculates the one period tail parameter error.

Init_CDRParaErrTailDetail: Estimates one period tail parameter error detail.

Init_CDRParaErrTailProxy: Calculates the approximated one period tail parameter error.

Init_CDRParaErrTailProxyDetail: Estimates the approximated one period tail parameter error detail.

Init_CDRProcVar: Calculates the one period process variance.

Init_CDRProcVarDetail: Estimates the one period process variance detail.

Init_CDRProcVarProxy: Calculates the approximated one period process variance .

Init_CDRProcVarProxyDetail: Calculates the approximated one period process variance detail.

Init_CDRProcVarTail: Calculates the one period tail process variance.

Init_CDRProcVarTailDetail: Estimates the one period tail process variance detail.

Init_CDRProcVarTailProxy: Calculates the approximated one period tail process variance.

Init_CDRProcVarTailProxyDetail: Estimates the approximated one period tail process variance detail.

Init_CDRProxy: Calculates the approximated total one period uncertainty.

Init_CDRTail: Calculates the total one period uncertainty with tail.

Init_CDRTailProxy: Calculates the approximated total one period uncertainty with tail.

Init_CoorF: Calculates the coordinates of operator $\mathbf{F}$. This is one of the most time consuming steps for the calculation of approximated uncertainties.

Init_Defaultalpha: Calculates the default mixing weights α_i^m.

Init_Defaulttf: Calculates the default development factors f_k^m.

Init_Defaultsigma: Calculates the default variance parameters $\sigma_k^{m_1,m_2}$.

Init_Defaulttw: Calculates the default weights $w_{i,k}^{m,I}$ and $w_{i,k}^{m,I+1}$.

Init_Eigenvalues: Calculates eigenvalues and eigenvectors of the covariance matrices $\left(\sigma_k^{m_1,m_2} R_{i,k}^{m_1,m_2}\right)_{m_1,m_2}$.

Init_f: Initialises user defined development factors f_k^m.

Init_Gamma1: Initialises the exposure parameters $\gamma_{i,k,h,j}^{m,l}$.

Init_Gamma2: Initialises the exposure parameters $\gamma_{i,k,h,j}^{m_1,m_2,l}$.

Init_Geometry: Initialises the number of claim properties M, the number of accident periods I, the number of development periods J and the number of future (tail) development periods.

Init_InflationRate: Initialises inflation rates.

Init_MSEP: Calculates the total ultimate uncertainty without tail.

Init_MSEPCov: Calculates the covariance matrix of the ultimate uncertainty.

Init_MSEPCovEigenvalue: Calculates eigenvalues and eigenvectors of the covariance matrix of the ultimate uncertainty.

Init_MSEPCovEigenvalueParaErr: Calculates eigenvalues and eigenvectors of the covariance matrix of the ultimate parameter error.

Init_MSEPCovEigenvalueParaErrProxy: Calculates eigenvalues and eigenvectors of the covariance matrix of the approximated ultimate parameter error.

Init_MSEPCovEigenvalueParaErrTail: Calculates eigenvalues and eigenvectors of the covariance matrix of the ultimate parameter error with tail.

Init_MSEPCovEigenvalueParaErrTailProxy: Calculates eigenvalues and eigenvectors of the covariance matrix of the approximated ultimate parameter error with tail.

Init_MSEPCovEigenvalueProcVar: Calculates eigenvalues and eigenvectors of the covariance matrix of the ultimate process variance.

Init_MSEPCovEigenvalueProcVarProxy: Calculates eigenvalues and eigenvectors of the covariance matrix of the approximated ultimate process variance.

Init_MSEPCovEigenvalueProcVarTail: Calculates eigenvalues and eigenvectors of the covariance matrix of the ultimate process variance with tail.

Init_MSEPCovEigenvalueProcVarTailProxy: Calculates eigenvalues and eigenvectors of the covariance matrix of the approximated ultimate process variance with tail.

Init_MSEPCovEigenvalueProxy: Calculates eigenvalues and eigenvectors of the covariance matrix of the approximated ultimate uncertainty.

Init_MSEPCovEigenvalueTail: Calculates eigenvalues and eigenvectors of the covariance matrix of the ultimate uncertainty with tail.

Init_MSEPCovEigenvalueTailProxy: Calculates eigenvalues and eigenvectors of the covariance matrix of the approximated ultimate uncertainty with tail.

Init_MSEPCovParaErr: Calculates the covariance matrix of the ultimate parameter error.

Init_MSEPCovParaErrProxy: Calculates the covariance matrix of the approximated ultimate parameter error.

Init_MSEPCovParaErrTail: Calculates the covariance matrix of the ultimate parameter error with tail.

Init_MSEPCovParaErrTailProxy: Calculates the covariance matrix of the approximated ultimate parameter error with tail.

Init_MSEPCovProcVar: Calculates the covariance matrix of the ultimate process variance.

Init_MSEPCovProcVarProxy: Calculates the covariance matrix of the approximated ultimate process variance.

Init_MSEPCovProcVarTail: Calculates the covariance matrix of the ultimate process variance with tail.

Init_MSEPCovProcVarTailProxy: Calculates the covariance matrix of the approximated ultimate process variance with tail.

Init_MSEPCovProxy: Calculates the covariance matrix of the approximated ultimate uncertainty.

Init_MSEPCovTail: Calculates the covariance matrix of the ultimate uncertainty with tail.

Init_MSEPCovTailProxy: Calculates the covariance matrix of the approximated ultimate uncertainty with tail.

Init_MSEPParaErr: Calculates the ultimate parameter error.

Init_MSEPParaErrDetail: Estimates the ultimate parameter error detail.

Init_MSEPParaErrProxy: Calculates the approximated ultimate parameter error.

Init_MSEPParaErrProxyDetail: Estimates the approximated ultimate parameter error detail.

Init_MSEPParaErrTail: Calculates the ultimate tail parameter error.

Init_MSEPParaErrTailDetail: Estimates the ultimate tail parameter error detail.

Init_MSEPParaErrTailProxy: Calculates the approximated ultimate tail parameter error.

Init_MSEPParaErrTailProxyDetail: Estimates the ultimate tail parameter error detail.

Init_MSEPProcVar: Calculates the ultimate process variance.

Init_MSEPProcVarDetail: Estimates the ultimate process variance detail. This is one of the most time consuming step for the calculation of the ultimate process variance (without approximation).

Init_MSEPProcVarProxy: Calculates the approximated ultimate process variance.

Init_MSEPProcVarProxyDetail: Estimates the approximated ultimate process variance detail. This is one of the most time consuming steps for the calculation of the approximated ultimate process variance.

Init_MSEPProcVarTail: Calculates the ultimate tail process variance.

Init_MSEPProcVarTailDetail: Estimates the ultimate tail process variance detail.

Init_MSEPProcVarTailProxy: Calculates the approximated ultimate tail process variance.

Init_MSEPProcVarTailProxyDetail: Estimates the approximated ultimate tail process variance detail.

Init_MSEPProxy: Calculates the approximated total ultimate uncertainty.

Init_MSEPTail: Calculates the total ultimate uncertainty with tail.

Init_MSEPTailProxy: Calculates the approximated total ultimate uncertainty with tail.

Init_R1: Calculates the known values of the exposures $R_{i,k}^{m}$.

Init_R2: Calculates the known values of the exposures $R_{i,k}^{m_1,m_2}$.

Init_R2Full: Estimates the future values of the exposures $R_{i,k}^{m_1,m_2}$.

Init_rhoCDR: Calculates $\varrho_{i_1,i_2,k}^{*1\,m_1,m_2}$, $\varrho_{i_1,i_2,k}^{*2\,m_1,m_2}$, $\varrho_{i_1,i_2,k}^{*12\,m_1,m_2}$ and $\bar{\varrho}_{i_1,i_2,k}^{m_1,m_2}$.

Init_rhoMSEP: Calculates $\varrho_{i_1,i_2,k}^{*\,m_1,m_2}$.

Init_rhoStar: Calculates $\varrho_k^{m_1,m_2}$.

Init_S: Initialises the known values of the claim properties $S_{i,k}^m$.

Init_sigma: Initialises user defined variance parameters $\sigma_k^{m_1,m_2}$.

Init_SxS0: Calculates the operator $\mathbf{H}(0)$.

Init_SxSCDRParaErr: Calculates $\mathbf{H}(\varrho_{i_1,i_2,k}^{*1\,m_1,m_2})$, $\mathbf{H}(\varrho_{i_1,i_2,k}^{*2\,m_1,m_2})$, and $\mathbf{H}(\varrho_{i_1,i_2,k}^{*12\,m_1,m_2})$. This is the most time consuming step for the calculation of the one period parameter error (without approximation).

Init_SxSCDRProcVar: Calculates $\mathbf{H}(\bar{\varrho}_{i_1,i_2,k}^{m_1,m_2})$. This is the most time consuming step for the calculation of the one period process variance (without approximation).

Init_SxSMSEP: Calculates $\mathbf{H}(\varrho_{i_1,i_2,k}^{*\,m_1,m_2})$. This is one of the most time consuming steps for the calculation of the ultimate parameter error (without approximation).

Init_Ultimate: Calculates the total α_i^m weighted ultimates.

Init_UltimateDetail: Estimates future values of the claim properties $S_{i,k}^m$ and the exposures $R_{i,k}^m$.

Init_w: Initialises user defined weights $w_{i,k}^{m,I}$ and $w_{i,k}^{m,I+1}$

4.2 *LSRM* types

Since for all classical *LSRMs*, like the Chain Ladder Method and the Complementary Loss Ratio Method, the exposure parameter do not depend on accident periods we introduced three kinds of *LSRM* implementations.

StandardLSRM: Represent the most general implementation of *LSRMs* (with linear exposures $R_{i,k}^{m_1,m_2}$) at the cost of calculation speed. Such a *LSRM* will be created if the exposure parameters **gamma1** and **gamma2** within the function **Init** are two dimensional array of float values where the second dimension has 7 and 8 columns, respectively. In order to speed up calculations we only sum over pairs (m, l) of claim property indices (and pairs (i, h) of accident periods) for which at least one exposure parameters $\gamma_{i,k,h,j}^{m,l}$ is not equal to zero. For instance, in the case of two Chain Ladder Models we sum only over $m = l$ and over $i = h$.

SimplifiedLSRM (without inflation): A *LSRM* with exposures $R_{i,k}^m$ and $R_{i,k}^{m_1,m_2}$ that depend only on claim properties of the same accident period i. Moreover these dependencies have to be independent on the accident period i. For the exposure parameters $\gamma_{i,k,h,j}^{m,l}$ and $\gamma_{i,k,h,j}^{m_1,m_2,l}$ this means

$$\begin{aligned}
\gamma_{i_1,k,h_1,j}^{m,l} &= \gamma_{i_2,k,h_2,j}^{m,l}, && \text{for all } i_1,\ i_2,\ h_1 \text{ and } h_2,\\
\gamma_{i,k,h,j}^{m,l} &= 0, && \text{for all } i \neq h,\\
\gamma_{i_1,k,h_1,j}^{m_1,m_2,l} &= \gamma_{i_2,k,h_2,j}^{m_1,m_2,l}, && \text{for all } i_1,\ i_2,\ h_1 \text{ and } h_2,\\
\gamma_{i,k,h,j}^{m_1,m_2,l} &= 0, && \text{for all } i \neq h.
\end{aligned}$$

Such *LSRMs* will be created by the function **Init** if no inflation rate has been specified and if the exposure parameters **gamma1** and **gamma2** are either strings or two dimensional array of float values where the second dimension has 5 and 6 columns, respectively.

For a SimplifiedLSRM we can skip all addends with $i = h$ in summations containing the exposure parameters $\gamma_{i,k,h,j}^{m,l}$. Moreover we only sum over pairs (m, l) of claim property indices for within at least one exposure parameters $\gamma_{i,k,h,j}^{m,l}$ is not equal to zero.

SimplifiedLSRM (with inflation): A *LSRM* with inflation where the corresponding deflated version is a SimplifiedLSRM without inflation. In this case we cannot skip summation over accident years (since the inflation may depend on the accident period), but we still only sum over pairs (m, l) of claim property indices for within at least one exposure parameters $\gamma_{i,k,h,j}^{m,l}$ is not equal to zero.

Such *LSRMs* will be created by the function **Init** if inflation rates have been specified and if the exposure parameters **gamma1** and **gamma2** are either strings or two dimensional array of float values where the second dimension has 5 and 6 columns, respectively.

Note: If the exposures are specified by formulas, see function **Init** in Section 2.2, the exposures $R_{i,k}^{m_1,m_2}$ may depend on $\mathbf{S}^{i+k}$ in a non-linear way!

5 *LSRMs* Excel interface

In this section we describe the provided VBA interface of LSRMTools, which can be found in the files LSRM_Tools_ActiveX.xla and LSRM_Tools_Dll.xla. The first is an interface that is based on the ActiveX component provided by the file LSRMTools.dll and the second is based on the function based dynamic link library LSRM_Dll.dll, see Figure 1.1. In all other aspects both interfaces are the same. For more information about installation of those files see Section 1.3.
The Excel interface provides

- Excel functions for each property of LSRMTools, see Section 5.1. Those function can be used like normal Excel functions. They can also be accessed via the category **LSRM Tools** of the Excel menu "insert function". The implementation of those functions can be found within the VBA module "LSRM_Functions".
- A concept based on "Excel names" that allows you to (re)initialise *LSRMs* more comfortable, see Section 5.2.

Without loss of generality we assume that accident and development periods are associated with Excel rows and columns, respectively.
Note: If you specify the exposures $R_{i,k}^{m}$ and $R_{i,k}^{m_1,m_2}$ by formulas, see Section 2.2, you may have to change the regional settings to English (UK) or use the property **Delimiters** (if you use the ActiveX component, otherwise it is the function **LSRM_SetDelimiters**), see Section 2.21 in order to change the delimiters. The reason is, that by default the decimal delimiter is taken from your regional settings, which may cause a conflict with the default formula delimiter (;) or the default parameter delimiter (,).

5.1 Excel functions

5.1.1 Default values of parameters

All parameter of each described function in this section are optional and of type Variant. If not otherwise noted default values are listed in Table 3.

5.1.2 Function **LSRMalpha**(**LSRMName**, **m**, **i**) as *Variant*

The interface for the property **alpha**, see Section 2.4.

5.1.3 Function **LSRMalphaInitialised**(**LSRMName**) as *Variant*

Returns if user defined mixing weights α_i^m have been initialised. If something went wrong a corresponding error message will be returned.
The function is based on the property **InitialisedParameters**, see Section 2.26.

5.1.4 Function **LSRMCalcInfo**(**LSRMName**, **Step**, **InfoType**) as *Variant*

The interface for the property **CalcInfo**, see Section 2.5.
Default value for the parameter **InfoType** is 2, i.e. calculation time. If the parameter **Step** is missing the function returns:

InfoType =0: the string “Total”

InfoType =1: an empty string

InfoType =2: the total calculation time

5.1.5 Function **LSRMCDR**(**LSRMName**, **m**, **i**, **Proxy**, **Part**, **Tail**) as *Variant*

The interface for the property **CDR**, see Section 2.7.
The default value for **Part** is 2, i.e. the sum of process variance and parameter error. Moreover, if **m** or **i** are missing or if they are not numeric we take $M + 1$ and $I + 1$, respectively.

5.1.6 Function **LSRMCDRCov**(**LSRMName**, **m1**, **m2**, **Proxy**, **Part**, **Tail**) as *Variant*

The interface for the property **CDRCov**, see Section 2.8.
The default value for **Part** is 2, i.e. the sum of process variance and parameter error.

5.1.7 Function LSRMCDRCovEigenvalue(LSRMName, m, Proxy, Part, Tail) as *Variant*

The interface for the property **CDRCovEigenvalue**, see Section 2.9.
The default value for **Part** is 2, i.e. the sum of process variance and parameter error.

5.1.8 Function LSRMCDRCovEigenvector(LSRMName, m, l, Proxy, Part, Tail) as *Variant*

The interface for the property **CDRCovEigenvector**, see Section 2.10.
The default value for **Part** is 2, i.e. the sum of process variance and parameter error.

5.1.9 Function LSRMCDRDetail(LSRMName, m1, m2, i1, i2 as *Long*, Proxy, Part, Tail) as *Variant*

The interface for the property **CDRDetail**, see Section 2.11.

5.1.10 Function LSRMCoorF(LSRMName, m, l, i, k, h, j) as *Variant*

The interface for the property **CoorF**, see Section 2.12.

5.1.11 Function LSRMDefaultalpha(LSRMName, m, i) as *Variant*

The interface for the property **Defaultalpha**, see Section 2.13.

5.1.12 Function LSRMDefaultalphaUsed(LSRMName) as *Variant*

The interface for the property **DefaultalphaUsed**, see Section 2.14.

5.1.13 Function LSRMDefaultf(LSRMName, m, k) as *Variant*

The interface for the property **Defaultf**, see Section 2.15.

5.1.14 Function LSRMDefaultfUsed(LSRMName) as *Variant*

The interface for the property **DefaultfUsed**, see Section 2.16.

5.1.15 Function LSRMDefaultsigma(LSRMName, m1, m2, k) as *Variant*

The interface for the property **Defaultsigma**, see Section 2.17.

5.1.16 Function LSRMDefaultsigmaUsed(LSRMName) as *Variant*

The interface for the property **DefaultsigmaUsed**, see Section 2.18.

5.1.17 Function LSRMDefaultw(LSRMName, n, m, i, k) as *Variant*

The interface for the property **Defaultw**, see Section 2.19.
The dafault value for the parameter **n** is 0, i.e. not normalised weights.

5.1.18 Function LSRMDefaultwUsed(LSRMName) as *Variant*

The interface for the property **DefaultwUsed**, see Section 2.20.

5.1.19 Function LSRMDelimiters() as *Variant*

The interface for the property **Delimiters**, see Section 2.21.

5.1.20 Function LSRMEigenvalue(LSRMName, i, k, m) as *Variant*

The interface for the property **Eigenvalue**, see Section 2.22.

5.1.21 Function LSRMEigenvector(LSRMName, i, k, m, l) as *Variant*

The interface for the property **Eigenvector**, see Section 2.23.
The default value for the parameter **l** is 0.

5.1.22 Function LSRMf(LSRMName, m, k) as *Variant*

The interface for the property **f**, see Section 2.24.

5.1.23 Function LSRMfInitialised(LSRMName) as *Variant*

Returns if user defined development factors f_k^m have been initialised. If something went wrong a corresponding error message will be returned.
The function is based on the property **InitialisedParameters**, see Section 2.26.

5.1.24 Function LSRMgamma1Initialised(LSRMName) as *Variant*

Returns if the exposure parameters $\gamma_{i,k,h,j}^{m,l}$ have been initialised. If something went wrong a corresponding error message will be returned.
The function is based on the property **InitialisedParameters**, see Section 2.26.

5.1.25 Function LSRMgamma2Initialised(LSRMName) as *Variant*

Returns if the exposure parameters $\gamma_{i,k,h,j}^{m_1,m_2,l}$ have been initialised. If something went wrong a corresponding error message will be returned.
The function is based on the property **InitialisedParameters**, see Section 2.26.

5.1.26 Function LSRMI(LSRMName) as *Variant*

The interface for the property **I**, see Section 2.25.

5.1.27 Function LSRMInflationInitialised(LSRMName) as *Variant*

Returns if inflation rates have been initialised. If something went wrong a corresponding error message will be returned.
The function is based on the property **InitialisedParameters**, see Section 2.26.

5.1.28 Function LSRMJ(LSRMName) as *Variant*

The interface for the property **J**, see Section 2.27.

5.1.29 Function LSRMK(LSRMName) as *Variant*

The interface for the property **K**, see Section 2.28.

5.1.30 Function LSRMM(LSRMName) as *Variant*

The interface for the property **M**, see Section 2.31.

5.1.31 Function LSRMModelType(LSRMName) as *Variant*

The interface for the property **ModelType**, see Section 2.32.

5.1.32 Function LSRMMSEP(LSRMName, m, i, Proxy, Part, Tail) as *Variant*

The interface for the property **MSEP**, see Section 2.33.
The default value for **Part** is 2, i.e. the sum of process variance and parameter error. Moreover, if **m** or **i** are missing or if they are not numeric we take $M + 1$ and $I + 1$, respectively.

5.1.33 Function LSRMMSEPCov(LSRMName, m1, m2, Proxy, Part, Tail) as *Variant*

The interface for the property **MSEPCov**, see Section 2.34.
The default value for **Part** is 2, i.e. the sum of process variance and parameter error.

5.1.34 Function LSRMMSEPCovEigenvalue(LSRMName, m, Proxy, Part, Tail) as *Variant*

The interface for the property **MSEPCovEigenvalue**, see Section 2.35.
The default value for **Part** is 2, i.e. the sum of process variance and parameter error.

5.1.35 Function LSRMMSEPCovEigenvector(LSRMName, m, l, Proxy, Part, Tail) as *Variant*

The interface for the property **MSEPCovEigenvector**, see Section 2.36.
The default value for **Part** is 2, i.e. the sum of process variance and parameter error.

5.1.36 Function LSRMMSEPDetail(LSRMName, m1, m2, i1, i2 as *Long*, Proxy, Part, Tail) as *Variant*

The interface for the property **MSEPDetail**, see Section 2.37.

5.1.37 Function LSRMPreventFutureExposureFromChangingSign(LSRMName) as *Variant*

Returns if we forbid future exposures to change sign, see Section 3.6. If something went wrong a corresponding error message will be returned.
The function is based on the property **CalculationFlags**, see Section 2.6.

5.1.38 Function LSRMPreventPastExposureFromChangingSign(LSRMName) as *Variant*

Returns if we forbid past (known) exposures to change sign, see Section 3.6. If something went wrong a corresponding error message will be returned.

The function is based on the property **CalculationFlags**, see Section 2.6.

5.1.39 Function LSRMR1(LSRMName, m, i, k) as *Variant*

The interface for the property **R1**, see Section 2.38.

5.1.40 Function LSRMR2(LSRMName, m1, m2, i, k) as *Variant*

The interface for the property **R2**, see Section 2.39.

5.1.41 Function LSRMReduceMemory(LSRMName) as *Variant*

Returns if some calculations should be disabled in oder to save the size of used memory, see Section 4. If something went wrong a corresponding error message will be returned.
The function is based on the property **CalculationFlags**, see Section 2.6.

5.1.42 Function LSRMS(LSRMName, m, i, k) as *Variant*

The interface for the property **S**, see Section 2.40.

5.1.43 Function LSRMsigma(LSRMName, m1, m2, k) as *Variant*

The interface for the property **sigma**, see Section 2.41.

5.1.44 Function LSRMsigmaInitialised(LSRMName) as *Variant*

Returns if user defined variance parameters $\sigma_k^{m_1,m_2}$ have been initialised. If something went wrong a corresponding error message will be returned.
The function is based on the property **InitialisedParameters**, see Section 2.26.

5.1.45 Function LSRMSInitialised(LSRMName) as *Variant*

Returns if claim properties $S_{i,k}^m$ have been initialised. If something went wrong a corresponding error message will be returned.
The function is based on the property **InitialisedParameters**, see Section 2.26.

5.1.46 Function LSRMUltimate(LSRMName, m, i, Tail) as *Variant*

The interface for the property **Ultimate**, see Section 2.42.

If **m** or **i** are missing or if they are not numeric we take $M + 1$ and $I + 1$, respectively. That means we take to corresponding total ultimate.

5.1.47 Function LSRMUseMSEPifTailCDRisLarger(LSRMName) as *Variant*

Returns if the solvency uncertainty should be bound by the ultimate uncertainty in the tail, see Section 3.2. If something went wrong a corresponding error message will be returned.
The function is based on the property **CalculationFlags**, see Section 2.6.

5.1.48 Function LSRMVirtualParaErrDiversification(LSRMName) as *Variant*

Returns if virtual diversification in the estimation of parameter errors should be allowed, see Section 3.5. If something went wrong a corresponding error message will be returned.
The function is based on the property **CalculationFlags**, see Section 2.6.

5.1.49 Function LSRMVirtualProcVarDiversification(LSRMName) as *Variant*

Returns if virtual diversification in the estimation of process variances should be allowed, see Section 3.5. If something went wrong a corresponding error message will be returned.
The function is based on the property **CalculationFlags**, see Section 2.6.

5.1.50 Function LSRMVirtualTailDiversification(LSRMName) as *Variant*

Returns if virtual diversification in the estimation of tail uncertainties should be allowed, see Section 3.5. If something went wrong a corresponding error message will be returned.
The function is based on the property **CalculationFlags**, see Section 2.6.

5.1.51 Function LSRMw(LSRMName, n, m, i, k) as *Variant*

The interface for the property **w**, see Section 2.43.
The dafault value for the parameter **n** is 0, i.e. not normalised weights.

5.1.52 Function LSRMWeightsUsedForDefaultsigma(LSRMName) as *Variant*

Returns if weights should be used for the calculation of the standard covariance parameters, see Section 3.4. If something went wrong a corresponding error message will be returned.
The function is based on the property **CalculationFlags**, see Section 2.6.

5.1.53 Function LSRMwInitialised(LSRMName) as *Variant*

Returns if user defined weights $w_{i,k}^{m,I}$ and $w_{i,k}^{m,I+1}$ have been initialised. If something went wrong a corresponding error message will be returned.
The function is based on the property **InitialisedParameters**, see Section 2.26.

5.2 Excel user interface for *LSRMs*

In order to make the usage of *LSRMs* within Excel more comfortable we added a menu, see Section 5.2.1, and the procedure **LSRMInit**, which can be accessed by the shortcut CTRL + SHIFT + I. The inputs for this procedure are based on a special Excel names convention, see Sections 5.2.2, 5.2.3 and 5.2.4.

5.2.1 *LSRM* menu in Excel

The Excel interfaces LSRM_TOOLS_ACTIVEX.XLA and LSRM_TOOLS_DLL.XLA add the menu LSRM TOOLS to Excel with the following entries:

Initialise: (Re)initialises the *LSRM* of the active worksheet, see Section 5.2.3. This procedure can also be accessed by the shortcut CTRL + SHIFT + I.

Recalculate Selected Range Recalculates the selected range of the active worksheet. This may be necessary, since Excel does not know if some parts of the *LSRM* have been recalculated. If only a singe cell is selected the whole worksheet will be recalculated. You can also accesses this functionality by the shortcut CTRL + SHIFT + R.

Recalculate Sheet Recalculates the active worksheet.

Free Memory Removes all *LSRMs* from memory.

About Displays a short message about the version and the license of LSRMTOOLS.

5.2.2 Supported Excel names

The initialisation, see Section 5.2.3, and the implemented checks, see Section 5.2.4, are based on Excel names. In this section we will describe those Excel names.

LSRM_ALPHA: Specifies the range of the user defined mixing weights α_i^m. It has to consists of M areas (one for each claim property, the order has to be the same as for LSRM_S) that are ranges of I rows and 1 column. If no such Excel name exists no user defined mixing weights can be (re)initialised.

LSRM_CALCINFO: Refers to the range that contains the informations about the calculated steps. If this name exists the corresponding range will be recalculated at the end of a (re)initialisation of a *LSRM*.

LSRM_F: Specifies the range of the user defined development factors f_k^m. It has to consists of M areas (one for each claim property, the order has to be the same as for LSRM_S) that are ranges of 1 row and $J-1+K$ columns. If no such Excel name exists no user defined development factors can be (re)initialised.

LSRM_GAMMA1: Specifies the range of the exposure parameter $\gamma_{i,k,h,j}^{m,l}$. In case of a formula based specification of those exposure parameters it has to be a single cell that contains the corresponding formula. Otherwise, the range has to consist of the following 7 columns (or 5 columns for a SIMPLIFIEDLSRM) in the specified order: m, l, i, k, h, j and $\gamma_{i,k,h,j}^{m,l}$ (for a SIMPLIFIEDLSRM the columns i and h have to be omitted). If no such Excel name exists no exposure parameters $\gamma_{i,k,h,j}^{m,l}$ can be (re)initialised.

LSRM_GAMMA2: Specifies the range of the exposure parameter $\gamma_{i,k,h,j}^{m_1,m_2,l}$. In case of a formula based specification of those exposure parameters it has to be a single cell that contains the corresponding formula. Otherwise, the range has to consist of the following 8 columns (or 6 columns for a SIMPLIFIEDLSRM) in the specified order: m_1, m_2, l, i, k, h, j and $\gamma_{i,k,h,j}^{m_1,m_2,l}$ (for a SIMPLIFIEDLSRM the columns i and h have to be omitted). If no such Excel name exists no exposure parameters $\gamma_{i,k,h,j}^{m_1,m_2,l}$ can be (re)initialised.

LSRM_GAMMABYFORMULA: Refers to a cell that specifies if the exposure parameters are specified by formulas. If the Excel name does not exist or if the corresponding cell does not contain FALSE we assume that the exposure parameters are specified by formulas.

LSRM_INITALPHA: Refers to a cell that specifies if user defined mixing weights α_i^m should be (re)initialised. If the Excel name does not exist or if the corresponding cell does not contain TRUE no user defined mixing weights will be (re)initialised.

LSRM_INITF: Refers to a cell that specifies if user defined development factors f_k^m should be (re)initialised. If the Excel name does not exist or if the corresponding cell does not contain TRUE no user defined development factors will be (re)initialised.

LSRM_INFLATION: Specifies the range of the inflation rates. It has to consists of M areas (one for each claim property, the order has to be the same as for LSRM_S) that are ranges of I row and $J + K$ columns. If no such Excel name exists no inflation rates can be (re)initialised.

LSRM_INITGAMMA1: Refers to a cell that specifies if exposure parameters $\gamma_{i,k,h,j}^{m,l}$ should be (re)initialised. The only case in which exposure parameters will not be (re)initialised is if they are already initialised and the cell contains FALSE.

LSRM_INITGAMMA2: Refers to a cell that specifies if exposure parameters $\gamma_{i,k,h,j}^{m_1,m_2,l}$ should be (re)initialised. The only case in which exposure parameters will not be (re)initialised is if they are already initialised and the cell contains FALSE.

LSRM_INITINFLATION: Refers to a cell that specifies if inflation rates should be (re)initialised. If the Excel name does not exist or if the corresponding cell does not contain TRUE no inflation rates will be (re)initialised.

LSRM_INITS: Refers to a cell that specifies if claim properties should be (re)initialised. The only case in which claim properties will not be (re)initialised is if they are already initialised and the cell contains FALSE.

LSRM_Initsigma: Refers to a cell that specifies if user defined variance parameters $\sigma_k^{m_1,m_2}$ should be (re)initialised. If the Excel name does not exist or if the corresponding cell does not contain True no used defined variance parameter will be (re)initialised.

LSRM_Initw: Refers to a cell that specifies if user defined (not normalised) weights $w_{i,k}^{m,I}$ and $w_{i,k}^{m,I+1}$ should be (re)initialised. If the Excel name does not exist or if the corresponding cell does not contain True no used defined weights will be (re)initialised.

LSRM_InputChanges: Refers to a range that are used to check if input parameters have been changed since the last (re)initialisation, see Section 5.2.4.

LSRM_K: Refers to a cell that specifies the number of tail periods K. If the Excel name does nor exists or if the cell does not contain a positive integer zero tail periods are taken.

LSRM_Name: Refers to a cell that specifies the name of the *LSRM*. If the Excel name does not exist we take an empty string.

LSRM_Overwrite: Refers to a cell that specifies if an already *LSRM* should be overwritten. If the Excel name does not exist or if the corresponding cell does not contain True the *LSRM* will only be reinitialised, in the case it already exists.

LSRM_PreventFutureExposureFromChangingSign: Refers to a cell that specifies if future (projected) exposures should be prevented from changing signs, see Section 3.6. If the Excel name does not exist or if the corresponding cell does not contain True future exposures will not be prevented from changing signs.

LSRM_PreventPastExposureFromChangingSign: Refers to a cell that specifies if past (tail) exposures should be prevented from changing signs, see Section 3.6. If the Excel name does not exist or if the corresponding cell does not contain True past exposures will not be prevented from changing signs.

LSRM_ReduceMemSize: Refers to a cell that specifies if the used memory size should be reduced at the cost of some functionality, see Section 4. If the Excel name does not exist or if the corresponding cell does not contain True the full functionality of LSRMTools will be accessible.

LSRM_S: Specifies the range of the claim properties. This range also defines the number of claim properties M and its order (equal to the number of areas), the number of accident periods I (equal to the number of rows of the first area) and the number of development periods J (equal to the number of columns of the first area). If no such Excel name exists no claim properties can be (re)initialised.

LSRM_sigma: Specifies the range of the user defined variance parameters $\sigma_k^{m_1,m_2}$. It has to consists of M time M areas (ordered like $(0,0)$, $(0,1), \ldots$, $(0,M)$, $(1,0), \ldots$, (M,M)) that are ranges of $J-1+\mathbf{1}_{K>0}$ columns. If no such Excel name exists no user defined covariance parameters can be (re)initialised.

LSRM_UseDefaultalpha: Refers to a cell that specifies if default mixing weights α_i^m should be used. If the Excel name does not exist, if the corresponding cell does not contain False or if no user defined mixing weights are initialised the default ones will be used.

LSRM_UseDefaultf: Refers to a cell that specifies if default development factors f_k^m should be used. If the Excel name does not exist, if the corresponding cell does not contain False or if no user defined development factors are initialised the default ones will be used.

LSRM_UseDefaultsigma: Refers to a cell that specifies if default covariance parameters $\sigma_k^{m_1,m_2}$ should be used. If the Excel name does not exist, if the corresponding cell does not contain False or if no user defined covariance parameters are initialised the default ones will be used.

LSRM_UseDefaultw: Refers to a cell that specifies if default weights $w_{i,k}^{m,I}$ and $w_{i,k}^{m,I+1}$ should be used. If the Excel name does not exist, if the corresponding cell does not contain False or if no user defined weights are initialised the default ones will be used.

LSRM_UseInflation: Refers to a cell that specifies if inflation rates should be used. If inflation rates have been specified we will use them except for the case where the cell contains False.

LSRM_UseWeightsForDefaultSigma: Refers to a cell that specifies if weights should be used for the calculation of the covariance parameters $\sigma_k^{m_1,m_2}$, see Section 3.4. If the Excel name does not exist or if the corresponding cell does not contain False we will use the same weights like in the calculation of the development factors f_k^m.

LSRM_VirtualParaErrDiversification: Refers to a cell that specifies if virtual diversification for parameter error estimations should be allowed, see Section 3.6. If the Excel name does not exist or if the corresponding cell does not contain True we forbid such virtual diversification.

LSRM_VirtualProcVarDiversification: Refers to a cell that specifies if virtual diversification for process variance estimations should be allowed, see Section 3.6. If the Excel name does not exist or if the corresponding cell does not contain True we forbid such virtual diversification.

LSRM_VirtualTailDiversification: Refers to a cell that specifies if virtual diversification for tail uncertainty estimations should be allowed, see Section 3.6. If the Excel name does not exist or if the corresponding cell does not contain True we forbid such virtual diversification.

LSRM_w: Specifies the range of the user defined (not normalised) weights $w_{i,k}^{m,I}$ and $w_{i,k}^{m,I+1}$. It has to consists of M areas (one for each claim property, the order has to be the same as for LSRM_S) that are ranges of I rows and $J-1+\mathbf{1}_{K>0}$ columns. If no such Excel name exists no user defined weights can be (re)initialised.

5.2.3 Reinitialising of ***LSRMs***

We implemented a mechanism based on predefined Excel names, see Section 5.2.2. It is accessible by the shortcut CTRL + SHIFT + I and does the following:

1. Look for the name of the active *LSRM* within the cell the Excel name LSRM_NAME refers to.

2. Look within the Excel Name LSRM_OVERWRITE if an already existing *LSRM* should be overwritten.

3. Look for exposure parameters $\gamma_{i,k,h,j}^{m,l}$ within the Excel name LSRM_GAMMA1 and if they should be (re)initialised within the Excel name LSRN_INITGAMMA1.

4. Look for exposure parameters $\gamma_{i,k,h,j}^{m_1,m_2,l}$ within the Excel name LSRM_GAMMA2 and if they should be (re)initialised within the Excel name LSRN_INITGAMMA2.

5. Look for claim properties $S_{i,k}^{m}$ within the Excel name LSRM_S and if they should be (re)initialised within the Excel name LSRN_INITS. **Note**, the corresponding range also defines the number and order of claim properties M (the number and order of areas), the number of accident periods I (number of rows of the first area) and the number of development periods J (number of columns of the first area).

6. Look for manual weights $w_{i,k}^{m,I}$ and $w_{i,k}^{m,I+1}$ within the Excel name LSRM_W and if they should be (re)initialised within the Excel name LSRN_INITW. **Note**, the found values will be normalised by the program such that the sum over all accident periods equals one. Therefore, it is possible to specify both, $w_{i,k}^{m,I}$ and $w_{i,k}^{m,I+1}$, within one range.

7. Look for manual development factors f_k^m within the Excel name LSRM_F and if they should be (re)initialised within the Excel name LSRN_INITF.

8. Look for manual covariance parameters $\sigma_k^{m_1,m_2}$ within the Excel name LSRM_SIGMA and if they should be (re)initialised within the Excel name LSRN_INITSIGMA.

9. Look for manual mixing weights α_i^m within the Excel name LSRM_ALPHA and if they should be (re)initialised within the Excel name LSRN_INITALPHA.

10. Look for inflation rates within the Excel name LSRM_INFLATION and if they should be (re)initialised within the Excel name LSRN_INITINFLATION.

11. Look within the Excel Name LSRM_USEDEFAULTW if default weights $w_{i,k}^{m,I}$ and $w_{i,k}^{mI+1}$ should be used. **Note**, if no manual weights are initialised default once will always be used.

12. Look within the Excel Name LSRM_USEDEFAULTF if default development factors f_k^m should be used. **Note**, if no manual development factors are initialised default once will always be used.

13. Look within the Excel Name LSRM_UseDefaultsigma if default covariance parameters $\sigma_k^{m_1,m_2}$ should be used. **Note**, if no manual covariance parameters are initialised default once will always be used.

14. Look within the Excle name LSRM_UseWeightsForDefaultSigma if the same weights should be used for the calculation of the default covariance parameters $\sigma_k^{m_1,m_2}$ and for the development factors f_k^m. If not the default weights are used for the calculation of the default covariance parameters $\sigma_k^{m_1,m_2}$.

15. Look within the Excel Name LSRM_UseDefaultalpha if default mixing weights α_i^m should be used. **Note**, if no manual mixing weights are initialised default once will always be used.

16. Look within the Excel Name LSRM_ReduceMemSize if the used memory size should be kept smaller at the cost of some functionality.

17. Look within the Excel Name LSRM_VirtualProcVarDiversification if "virtual diversification" in process variance estimations should be allowed, see Section 3.5.

18. Look within the Excel Name LSRM_VirtualParaErrDiversification if "virtual diversification" in parameter error estimations should be allowed, see Section 3.5.

19. Look within the Excel Name LSRM_VirtualTailDiversification if "virtual diversification" in tail uncertainty estimations should be allowed, see Section 3.5.

20. Look within the Excel Name LSRM_PreventPastExposureFromChangingSign if past (known) exposures should be allowed to change sign, see Section 3.6.

21. Look within the Excel Name LSRM_PreventFutureExposureFromChangingSign if future (projected) exposures should be allowed to change sign, see Section 3.6.

22. Look within the Excel Name LSRM_K for the number of future (tail) periods.

23. Load claim properties $S_{i,k}^m$, if they should be (re)initialised.

24. Look within the Excel Name LSRM_GammaByFormula if the exposures are specified by an array or a string.

25. Load the formula for the exposures $R_{i,k}^m$ or the exposure parameters $\gamma_{i,k,h,j}^{m,l}$, if they should be (re)initialised. For that matter the value of Excel name LSRM_GammaByFormula is used.

26. Load the formula for the exposures $R_{i,k}^{m_1,m_2}$ or the exposure parameters $\gamma_{i,k,h,j}^{m_1,m_2,l}$, if they should be (re)initialised. For that matter the value of Excel name LSRM_GammaByFormula is used.

27. Load manual weights $w_{i,k}^{m,I}$ and $w_{i,k}^{mI+1}$, if they should be (re)initialised.

28. Load manual development factors f_k^m, if they should be (re)initialised.

29. Load manual covariance parameters $\sigma_k^{m_1,m_2}$, if they should be (re)initialised.

30. Load manual mixing weights α_i^m, if they should be (re)initialised.

31. Load inflation rates, if they should be (re)initialised.

32. (Re)initialise the *LSRM* with the specified parameters.

33. Recalculate all formulas of the active worksheet and afterwards all cells of the range specified by the Excel name LSRM_CalcInfo. This is necessary since Excel does not know about the right order of calculation.

5.2.4 Implemented controls

Since the *LSRM* does not know if some input parameters, that are specified within Excel ranges, may have been changed we implement a simple process to keep track of such changes. Therefore, we use the function **LSRMInputChanged** on each of those input ranges. This function returns TRUE if we are not in an update mode. In order to remember the state of such cells when executing **LSRMInit**, **LSRMCalculateRange** or **LSRMCalculateSheet** the collection of those cells have to be associated with the Excel name LSRM_InputChanges. Internally we use the procedures **FRememberChanges** and **FResetChanges**.
Note, this will only work if Application.Calculation is set to xlCalculationAutomatic.

References

[1] Dahms, René (2012). Linear Stochastic Reserving Methods. ASTIN Bulletin.

[2] Dependencies.xls

Flag	Value	Description
cLSRMReduceMemSize	2^{13}	disable some calculations in order to reduce the used memory size
cLSRMOverwriteExistingLSRM	2^{14}	an already existing *LSRM* of the same name will be overwritten
cLSRMVirtualTailDiversification	2^{15}	allow virtual tail diversification within tail error estimators, see Section 3.5
cLSRMVirtualProcVarDiversification	2^{16}	allow virtual diversification within process variance estimators, see Section 3.5
cLSRMVirtualParaErrDiversification	2^{17}	allow virtual diversification within parameter error estimators, see Section 3.5
cLSRMPreventPastRFromChangingSign	2^{18}	stop decrease (or increase) of known exposures at zero, see Section 3.6
cLSRMPreventFutureRFromChangingSign	2^{19}	stop decrease (or increase) of known exposures at zero, see Section 3.6
cLSRMUseWeightsForDefaultSigma	2^{20}	indicates that weights should be used for the calculation of the covariance parameters $\sigma_k^{m_1,m_2}$
cLSRMUseMSEPifTailCDRisLarger	2^{21}	the used tail approximation may lead to larger solvency uncertainty (CDR) than ultimate uncertainty (MSEP). If this flag is set then the MSEP is taken in cases where the tail CDR is larger.

Table 2: calculation flags

Parameter	Default value
LSRMName	the value of the Excel name "LSRM_Name" if such a name exists, otherwise an empty string
claim property indices like m, m_1, m_2 and l	0
accident periods like i, i_1, i_2 and h	0
development periods like k and j	0
Proxy	1 (approximated estimations)
Part	0 (process variance)
Tail	1 (with tail)

Table 3: Default values of Excel functions