```
#!/usr/bin/perl
#
# Implementation Approach:
#
# 1. Read first matrix
#    Checks:
#    Matrix quadratic?
#    Risk factor order left->right (top row) == top->bottom (leftmost
column)?
#    Diagonals == 1 (warning)?
#    No NC category (warning if there is)?
#    Matrix symmetric: M(i,j) == M(j,i) for all i,j?
#    [Not for DC files because not given there.]
#
# 2. Read second matrix
#    Checks identical to above
#
# 3. Risk factors in both matrices identical?
#    Warn about risk factors which are in first matrix but not in second and
vice versa
#    Highlight outliers per category
#    Highlight outliers per ccy
#
# Version Date       Author         Comment
# 1.17    24/03/2014 Bernd Plumhoff Link to RMLinks.cfg in $ALGO_TOP/cfg
my $version = "'1.17,24/03/2014";

use strict;
use warnings;
use List::Util qw[min max];
use feature "switch";
use Getopt::Std;

#################################################################
#                                                               #
#                   Process parameters                          #
#                                                               #
#################################################################

our $opt_b; # b - breaches: do not report differences between the two input
matrices
      #     but breaches beyond tolerances
our $opt_d; # d - debug ["level"] gives debugging information at detail
level "level"
      #     level 1: -
      #     level 2: -
      #     level 3: Print all elements of matrices 1 and 2
our $opt_f; # f - read deviation file [-f needs to be followed by a valid
filename]
      #     Reads min and max values for all slices for differences which
      should
      #     be ignored during comparison. See option -w to get format
      example.
our $opt_h; # h - help: list parameters and their explanation
our $opt_i; # i - ignore risk factors in a given file [-i needs to be
followed by
      #     a valid filename]
our $opt_m; # m - set max rank index [default is 6 (=return highest 3
      #     and lowest 3 of each slice); m needs to be even and >= 4 !
our $opt_n; # n - tolerate risk factor category NC
our $opt_r; # r - set Algo risk factor category file [default is
```

```
"./RMLinks.cfg"]
our $opt_s; # s - summarize findings, no detailed warnings or error messages
our $opt_t; # t - read tolerance file and apply tolerance check
our $opt_v; # v - print version
our $opt_w; # w - write deviation file with min and max values of all
slices.
      #     This file is ","-separated to be easily readable via Excel.
      #     It can be amended and used with option -f later
      #     [-w needs to be followed by a valid filename,
      #     preferrably ending with ".csv"]
our $opt_x; # x - read translation table [-x needs to be followed by a valid
filename].
      #     Risk factor names of matrix 1 will be translated by second name
      in
      #     comma-separated row

getopts('bd:f:hi:m:nr:st:vw:x:');

if (defined $opt_h) {
  print "$0 parameters:\n" .
    "b - breaches: do not report differences between the two input
    matrices\n" .
    "    but breaches beyond tolerances\n" .
    "d - debug [\"level\"] gives debugging information at detail level
    \"level\"\n" .
    "    level 1: -\n" .
    "    level 2: -\n" .
    "    level 3: Print all elements of matrices 1 and 2\n" .
    "f - read deviation file [-f needs to be followed by a valid
    filename]\n" .
    "    Reads min and max values for all slices for differences which
    should\n" .
    "    be ignored during comparison. See option -w to get format
    example.\n" .
    "h - help: list parameters and their explanation\n" .
    "i - ignore risk factors in a given file [-i needs to be followed by\n"
    .
    "    a valid filename]\n" .
    "m - set max rank index [default is 6 (=return highest 3\n" .
    "    and lowest 3 of each slice); m needs to be even and >= 4 !\n" .
    "n - tolerate risk factor category NC\n" .
    "r - set Algo risk factor category file [default is
    \"./RMLinks.cfg\"]\n" .
    "s - summarize findings, no detailed warnings or error messages\n" .
    "t - read file with tolerated changes for each matrix element and
    apply\n" .
        "    tolerance check\n" .
    "v - print version\n" .
    "w - write deviation file with min and max values of all slices.\n" .
    "    This file is \",\"-separated to be easily readable via Excel.\n" .
    "    It can be amended and used with option -f later\n" .
    "    [-w needs to be followed by a valid filename,\n" .
    "    preferrably ending with \".csv\"]\n" .
    "x - read translation table [-x needs to be followed by a valid
    filename].\n" .
    "    Risk factor names of matrix 1 will be translated by second name
    in\n" .
    "    comma-separated row\n";
  exit(0);
}
```

```
if (defined $opt_v) {
  print "$0\tVersion: $version\n";
  exit(0);
}

$opt_m ||= 6;
if ($opt_m < 4 || $opt_m % 2 != 0) {
  die "$0: parameter -m needs to be followed by an even number >= 4!\n";
}

# Initialize risk factor categories from file
# Please note that you will find info to fill that file in $ALGO_TOP\cfg\
$opt_r ||= "./RMLinks.cfg";

our $doutfile;
if (defined $opt_w) {
  if (defined $opt_f) {
    die "$0: Do not use option -w together with option -f!\n";
  }
  open($doutfile, ">",  $opt_w) || die "$0: Can't open $opt_w: $!";
  $opt_b ||= "";
  print $doutfile "$0,$version,Min,Max,Total,Tolerance Breach Down,Tolerance
  Breach Up," .
    "$opt_b,Do not change cells on the left." .
    " They are used by an implicit format check.\n";
}

our $epsilon = 0.000001;  # Tolerance for floating number rounding errors.
Please note that the
        # value 1e-6 has been chosen carefully: it coincides with the
        precision
        # of printf's format %f (see sub printarr below, please).

################################################################
#                                                              #
#                   Read deviation file                        #
#                                                              #
################################################################

our $dinfile;
our $line;
our @fields=();
our @devglobal=();    # Stores global deviations as read by -f option if any
our %devrfcat=();   # Stores deviations per category if any
our %devccy=();     # Stores deviations per ccy if they exist
our %devrfcpair=();   # Stores deviations per category pair if they exist
if (defined $opt_f) {
  open($dinfile, "<",  $opt_f) || die "$0: Can't open $opt_f: $!";
  @fields = split(",", <$dinfile>);
  if ($0 ne $fields[0] || $version ne $fields[1] . ";" . $fields[2]) {
    print STDERR "$0: Warning: $0,$version expected but $fields[0]," .
        "$fields[1],$fields[2] found.\n";
  }
  if (! $opt_b && $fields[5] || $opt_b && ! $fields[5]) {
    $opt_b ||= "";
    die "$0: Option -b setting ($opt_b) does not match deviation file
    setting ($fields[5])";
  }
  while($line = <$dinfile>) {
    @fields = split(";", substr($line,0,length($line)-2));
    given($fields[0]) {
```

```
      when ("GLOBAL") {
        $devglobal[0] = $fields[3];
        $devglobal[1] = $fields[4];
      }
      when ("CATEGORY") {
        $devrfcat{$fields[1]}[0] = $fields[3];
        $devrfcat{$fields[1]}[1] = $fields[4];
      }
      when ("CURRENCY") {
        $devccy{$fields[1]}[0] = $fields[3];
        $devccy{$fields[1]}[1] = $fields[4];
      }
      when ("CATEGORYPAIR") {
        $devrfcpair{$fields[1] . "," . $fields[2]}[0] = $fields[3];
        $devrfcpair{$fields[1] . "," . $fields[2]}[1] = $fields[4];
      }
      default {
        die "$0: I don't know what to do with deviation definition
        $fields[0]";
      }
    }
  }
  close $dinfile;
}

################################################################
#                                                              #
#   Read file with riskfactors which should get ignored        #
#                                                              #
################################################################

our %rfignore=(); # Stores risk factors to be ignored
if (defined $opt_i) {
  open($dinfile, "<",  $opt_i) || die "$0: Can't open $opt_i: $!";
  while($line = <$dinfile>) {
    $line =~ s/[\r\n]+//g;
    $rfignore{$line} = 1;
  }
  close $dinfile;
}

################################################################
#                                                              #
#    Read file with translation table for risk factors         #
#                                                              #
################################################################

our %rftrans=();  # Stores risk factors to be translated
if (defined $opt_x) {
  open($dinfile, "<",  $opt_x) || die "$0: Can't open $opt_x: $!";
  while($line = <$dinfile>) {
    $line =~ s/[\r\n]+//g;
    @fields = split(",", $line);
    $rftrans{$fields[0]} = $fields[1];
  }
  close $dinfile;
}

################################################################
#                                                              #
#                  Read tolerance matrix                       #
```

```
#                                                              #
################################################################

our $tinfile;
our @tnames=();   # risk factor names in tolerance matrix
our %hashtnames=(); # hash table of risk factor names and their position
our @tol=();    # array with tolerances (per risk factor)
our $i;     # row index for loops
our $j;     # column index in loops
if (defined $opt_t) {
  open($tinfile, "<",  $opt_t) || die "$0: Can't open $opt_t: $!";
  $line = <$tinfile>;
  $line =~ s/[\r\n]+//g;
  @tnames = split(",", $line);
  for ($i=1; $i<scalar(@tnames); $i++) {
    $hashtnames{$tnames[$i]} = $i;
  }
  while($line = <$tinfile>) {
    $line =~ s/[\r\n]+//g;
    @fields=split(",", $line);

    # Matrix quadratic?
    if (scalar(@tnames) != scalar(@fields)) {
      print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" .
      @tnames .
          ") != # of fields in row $. (" . @fields . ")!\n";
      exit(1);
    }

    # Risk factor order left->right (top row) == top->bottom (leftmost
    column)?
    if ($fields[0] ne @tnames[$. - 1]) {
      print STDERR "$ARGV[0], Row $.: risk factor \"$fields[0]\" does not
      match" .
        " corresponding column header \"@tnames[$. - 1]\".\n";
    }

    # No NC category (warning if there is)?
    ! $opt_n && ! $opt_s && get_rf_category($fields[0]) eq "NC" &&
      print STDERR "$ARGV[0], Row $.: risk factor $fields[0] is linked to
      category NC.\n";

    # Please note that we only fill triangle above diagonal since we check
    for symmetry
    for ($i=$. - 2; $i<scalar(@tnames) - 1; $i++) {
      $tol[$. - 2][$i] = $fields[$i + 1];
      defined $opt_d && $opt_d > 2 && print "Tolerance[" . ($. - 2) .
      "][$i]=" .
        $fields[$i + 1] . "\n";
    }
    for ($i=0; $i<$. - 1; $i++) {
      # Matrix symmetric: M(i,j) == M(j,i) for all i,j?
      if ($fields[$i + 1] != $tol[$i][$. - 2]) {
        print STDERR "$ARGV[0], Row $.: M[" . ($. - 2) . "][" . $i ."] !=
        M[" . $i .
            "][" . ($. - 2) . "]: " . $fields[$i + 1] . "!=" .
            $tol[$i][$. - 2] . ".\n";
      }
    }
  }
```

```
  # Matrix quadratic?
  if (scalar(@tnames) != $.) {
    print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" .
      scalar(@tnames) . ") != # of rows ($.)!\n";
    exit(1);
  }

  close $tinfile;

}
our %rfcatpat=(); # links search pattern to risk factor category
our %rfcat=();    # speeds up get_rf_category
our $rfcatcount=0;  # counts risk factor categories in order to build
@rfcatrank

our @m1=();     # First matrix
our @m2=();     # Second matrix
our $m2val;   # Temporary value of second matrix
our @m3=();     # Diff matrix: Second minus First
our @names1=();   # risk factor names in matrix 1
our @names2=();   # risk factor names in matrix 2
our %rfnames1=(); # to store row number of risk factor names in matrix 1
our %rfnames2=(); # to store row number of risk factor names in matrix 2
our $ccy;
our $t;     # tolerance

# Variables to process DC file
our @hnames=();   # helper array to split up risk factor names in DC file
our $rf1;
our $rf2;
our $old_rf;    # for row change comparisons of the DC matrices
our $all_rfnames_read;  # Boolean
our $start_next_row;  # Boolean
our $is_upper_right_triangle; # Boolean
# Variables to rank outliers detected
our @diagnotone1=();  # ranks diagonal values <> 1 of matrix 1
our $dno1idx = 0; # number of elements in @diagnotone1
our @diagnotone2=();  # ranks diagonal values <> 1 of matrix 2
our $dno2idx = 0; # number of elements in @diagnotone2
our @totaldiff=();  # ranks all differences of matrix 2 minus matrix 1
our $tdidx = 0;   # number of elements in @totaldiff
our %rfcatrank=();
our %rfcatidx=(); # number of elements in @{$rfcatrank{$rfcatidx{rfname}}}
our %rfcccyrank=();
our %rfcccyidx=();  # number of elements in
@{$rfccyrank{$rfcccyidx{rfname}}}
our %rfcpairrank=();
our %rfcpairidx=(); # number of elements in
@{$rfcpairrank{$rfcpairidx{rfname}}}

init_rf_category($opt_r);

#################################################################
#                                                               #
#                   1. Read first matrix                        #
#                                                               #
#################################################################

open (FILE, "<", $ARGV[0]) || die "$0: Cannot open $ARGV[0]: $!";
$line = <FILE>;
$line =~ s/[\r\n]+//g;
```

```
if (substr($line,0,1) eq ",") {

  # We read a CSV file

  @names1 = split(",", $line);
  for ($i=1; $i<@names1; $i++) {
    $rftrans{$names1[$i]} ||= $names1[$i];
    $rfnames1{$rftrans{$names1[$i]}} = $i;
  }

  while($line = <FILE>) {
    $line =~ s/[\r\n]+//g;
    @fields=split(",", $line);

    # Matrix quadratic?
    if (scalar(@names1) != scalar(@fields)) {
      print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" .
      @names1 .
          ") != # of fields in row $. (" . @fields . ")!\n";
      exit(1);
    }

    # Risk factor order left->right (top row) == top->bottom (leftmost
    column)?
    if ($fields[0] ne @names1[$. - 1]) {
      print STDERR "$ARGV[0], Row $.: risk factor \"$fields[0]\" does not
      match" .
        " corresponding column header \"@names1[$. - 1]\".\n";
    }

    # No NC category (warning if there is)?
    ! $opt_n && ! $opt_s && get_rf_category($fields[0]) eq "NC" &&
      print STDERR "$ARGV[0], Row $.: risk factor $fields[0] is linked to
      category NC.\n";

    # Diagonals == 1 (warning)?
    if (abs($fields[$. - 1] - 1) > $epsilon) {
      ($dno1idx, @diagnotone1) = rankinsert($fields[$. - 1],
              "$ARGV[0], Row $.: $fields[0]",
                2, -2, 0, $dno1idx, @diagnotone1)
                unless $rfignore{$fields[0]};
      ! $opt_s && print STDERR "$ARGV[0], Row $.: Diagonal element is not
      equal to 1: M[" .
            ($. - 1) . "," . ($. - 1) . "] == " . $fields[$. - 1] . ".\n";
    }

    # Please note that we only fill triangle above diagonal since we check
    for symmetry
    for ($i=$. - 2; $i<scalar(@names1) - 1; $i++) {
      $m1[$. - 2][$i] = $fields[$i + 1];
      defined $opt_d && $opt_d > 2 && print "Matrix1[" . ($. - 2) . "][$i]="
      .
        $fields[$i + 1] . "\n";
    }
    for ($i=0; $i<$. - 1; $i++) {
      # Matrix symmetric: M(i,j) == M(j,i) for all i,j?
      if ($fields[$i + 1] != $m1[$i][$. - 2]) {
        print STDERR "$ARGV[0], Row $.: M[" . ($. - 2) . "][" . $i ."] !=
        M[" . $i .
            "][" . ($. - 2) . "]: " . $fields[$i + 1] . "!=" .
            $m1[$i][$. - 2] . ".\n";
```

```
      }
    }
  }

  # Matrix quadratic?
  if (scalar(@names1) != $.) {
    print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" .
      scalar(@names1) . ") != # of rows ($.)!\n";
    exit(1);
  }

} elsif (substr($line,0,1) eq "\*") {

  # We read a DC file

  $all_rfnames_read = 0;  # We have not identified all risk factor names yet
  $i = 1;
  $is_upper_right_triangle = 0;
  $old_rf = "";

  while($line = <FILE>) {
    $line = substr($line,0,length($line)-2);
    next if (($line  =~ /^\*/));
    next if (($line  =~ /^$/));

    @fields = split(",", $line);
    @hnames = split(/\./, $fields[0]);
    if ($is_upper_right_triangle) {
      $rf1 = $hnames[0] . "." . $hnames[1];
      $rf2 = $hnames[2] . "." . $hnames[3];
    } else {
      $rf2 = $hnames[0] . "." . $hnames[1];
      $rf1 = $hnames[2] . "." . $hnames[3];
    }

    if ($. == 4) {
      if ($rf1 eq $old_rf) {
        # DC file is of upper right triangle form
        $is_upper_right_triangle = 1;
        $rf2 = $rf1;
        $rf1 = $hnames[0] . "." . $hnames[1];
      }
    }

    if ($rf2 ne $old_rf) {
      $start_next_row = 1;
    } else {
      $start_next_row = 0;
    }
    $old_rf = $rf2;

    if ($start_next_row) {
      if ($. > 3) {
        $all_rfnames_read = 1;
      }
      if ($rf1 ne $rf2) {
        print STDERR "$ARGV[0], Row $.: matrix is not quadratic: After
        change of 2." .
            " risk factor name the next diagonal element has" .
            " to have (name 1 == name 2)!\n";
        exit(1);
```

```
      }
      # Diagonals == 1 (warning)?
      if (abs($fields[1] - 1) > $epsilon) {
        ($dno1idx, @diagnotone1) = rankinsert($fields[1],
                "$ARGV[0], Row $.: $rf1",
                  2, -2, 0, $dno1idx, @diagnotone1)
                  unless $rfignore{$rf1};
        ! $opt_s && print STDERR "$ARGV[0], Row $.: Diagonal element is not
        equal to 1: M[" .
            ($rfnames1{$rftrans{$rf1}}) . "," . ($rfnames1{$rftrans{$rf2}})
            .
            "] == " . $fields[1] . ".\n";
      }
    }

    if (! $all_rfnames_read) {
      $rftrans{$rf1} ||= $rf1;
      $names1[$i] = $rftrans{$rf1};
      if (! defined $rfnames1{$rftrans{$rf1}}) {
        $rfnames1{$rftrans{$rf1}} = $i++;
      }
    } else {
      # Risk factor order left->right (top row) == top->bottom (leftmost
      column)?
      ! defined $rftrans{$rf1} &&
        print STDERR "$ARGV[0], Row $.: risk factor \"$rf1\" does not
        exist.\n";
      ! defined $rftrans{$rf2} &&
        print STDERR "$ARGV[0], Row $.: risk factor \"$rf2\" does not
        exist.\n";
    }

    # No NC category (warning if there is)?
    ! $opt_n && ! $opt_s && get_rf_category($rf1) eq "NC" &&
      print STDERR "$ARGV[0], Row $.: risk factor $rf1 is linked to category
      NC.\n";
    ! $opt_n && ! $opt_s && get_rf_category($rf2) eq "NC" &&
      print STDERR "$ARGV[0], Row $.: risk factor $rf2 is linked to category
      NC.\n";

    # Please note that we only fill triangle above diagonal
    $m1[$rfnames1{$rftrans{$rf2}} - 1][$rfnames1{$rftrans{$rf1}} - 1] =
    $fields[1];
    defined $opt_d && $opt_d > 2 && print "Matrix1[" .
    $rfnames1{$rftrans{$rf2}} .
              "][$rfnames1{$rftrans{$rf1}}]=" . $fields[1] . "\n";
  }

  # Matrix quadratic?
  if (scalar(@names1) != sqrt(2*$. - 3.75) + 0.5) {
    print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" .
    scalar(@names1) .
        ") != # of rows (" . (sqrt(2*$. - 3.75) + 0.5) . ")!\n";
    # exit(1);
  }

}

close(FILE);

! $opt_s && $dno1idx && printarr("Diagonal in matrix 1 not 1:", $dno1idx,
```

```
@diagnotone1);

#################################################################
#                                                               #
#                    2. Read second matrix                      #
#                                                               #
#################################################################
our $m1name = $ARGV[0];
shift;
$all_rfnames_read = 0;
open (FILE, "<", $ARGV[0]) || die "Cannot open $ARGV[0]: $!";

$line = <FILE>;
$line =~ s/[\r\n]+//g;
if (substr($line,0,1) eq ",") {

  # We read a CSV file

  @names2 = split(",", $line);
  for ($i=1; $i<@names2; $i++) {
    $rfnames2{$names2[$i]} = $i;
  }

  while($line = <FILE>) {
    $line =~ s/[\r\n]+//g;
    @fields=split(",", $line);

    # Matrix quadratic?
    if (scalar(@names2) != scalar(@fields)) {
      print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" .
      @names2 .
          ") != # of fields in row $. (" . @fields . ")!\n";
      exit(1);
    }

    # Risk factor order left->right (top row) == top->bottom (leftmost
    column)?
    if ($fields[0] ne @names2[$. - 1]) {
      print STDERR "$ARGV[0], Row $.: risk factor \"$fields[0]\" does not
      match" .
        " corresponding column header \"@names2[$. - 1]\".\n";
    }

    # Warn about risk factors which are in first matrix but not in second
    #if (! (defined $rfnames1{$rftrans{$fields[0]}})) {
    # print STDERR "$ARGV[0], Row $.: risk factor \"$fields[0]\" does not
    exist in " .
    #    "$m1name" . "\n";
    #}

    # No NC category (warning if there is)?
    ! $opt_n && ! $opt_s && get_rf_category($fields[0]) eq "NC" &&
      print STDERR "$ARGV[0], Row $.: risk factor $fields[0] is linked to
      category NC.\n";

    # Diagonals == 1 (warning)?
    if (abs($fields[$. - 1] - 1) > $epsilon) {
      ($dno2idx, @diagnotone2) = rankinsert($fields[$. - 1],
              "$ARGV[0], Row $.: $fields[0]",
                2, -2, 0, $dno2idx, @diagnotone2)
                unless $rfignore{$fields[0]};
```

```
      ! $opt_s && print STDERR "$ARGV[0], Row $.: Diagonal element is not
      equal to 1: M[" .
            ($. - 1) . "," . ($. - 1) . "] == " . $fields[$. - 1] . ".\n";
    }

    # Please note that we only fill triangle above diagonal since we check
    for symmetry
    for ($i=$. - 2; $i<scalar(@names2) - 1; $i++) {
      $m2[$. - 2][$i] = $fields[$i + 1];
      defined $opt_d && $opt_d > 2 && print "Matrix2[" . ($. - 2) . "][$i]="
      .
         $fields[$i + 1] . "\n";
    }
    for ($i=0; $i<$. - 1; $i++) {
      # Matrix symmetric: M(i,j) == M(j,i) for all i,j?
      if ($fields[$i + 1] != $m2[$i][$. - 2]) {
        print STDERR "$ARGV[0], Row $.: M[" . ($. - 2) . "][" . $i ."] !=
        M[" . $i .
            "][" . ($. - 2) . "]: " . $fields[$i + 1] . "!=" .
            $m2[$i][$. - 2] . ".\n";
      }
    }
  }

  # Matrix quadratic?
  if (scalar(@names2) != $.) {
    print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" .
    scalar(@names2) .
       ") != # of rows ($.)!\n";
    exit(1);
  }

} elsif (substr($line,0,1) eq "\*") {

  # We read a DC file

  $all_rfnames_read = 0;  # We have not identified all risk factor names yet
  $i = 1;
  $is_upper_right_triangle = 0;
  $old_rf = "";

  while($line = <FILE>) {
    $line = substr($line,0,length($line)-2);
    next if (($line  =~ /^\*/));
    next if (($line  =~ /^$/));

    @fields = split(",", $line);
    @hnames = split(/\./, $fields[0]);
    if ($is_upper_right_triangle) {
      $rf1 = $hnames[0] . "." . $hnames[1];
      $rf2 = $hnames[2] . "." . $hnames[3];
    } else {
      $rf2 = $hnames[0] . "." . $hnames[1];
      $rf1 = $hnames[2] . "." . $hnames[3];
    }

    if ($. == 4) {
      if ($rf1 eq $old_rf) {
        # DC file is of upper right triangle form
        $is_upper_right_triangle = 1;
        $rf2 = $rf1;
```

```
      $rf1 = $hnames[0] . "." . $hnames[1];
    }
  }

  if ($rf2 ne $old_rf) {
    $start_next_row = 1;
  } else {
    $start_next_row = 0;
  }
  $old_rf = $rf2;

  if ($start_next_row) {
    if ($. > 3) {
      $all_rfnames_read = 1;
    }
    if ($rf1 ne $rf2) {
      print STDERR "$ARGV[0], Row $.: matrix is not quadratic: After
      change of 2." .
        " risk factor name the next diagonal element has to be" .
        " given (name 1 == name 2)!\n";
      exit(1);
    }
    # Diagonals == 1 (warning)?
    if (abs($fields[1] - 1) > $epsilon) {
      ($dno2idx, @diagnotone2) = rankinsert($fields[1],
              "$ARGV[0], Row $.: $rf1",
                2, -2, 0, $dno2idx, @diagnotone2)
                unless $rfignore{$rf1};
      ! $opt_s && print STDERR "$ARGV[0], Row $.: Diagonal element is" .
          " not equal to 1: M[" .
          ($rfnames2{$rf1}) . "," . ($rfnames2{$rf2}) . "] == " .
          $fields[1] . ".\n";
    }
  }

  if (! $all_rfnames_read) {
    $names2[$i] = $rf1;
    if (defined $rfnames2{$rf1}) {
      print STDERR "$ARGV[0], Row $.: risk factor \"$rf1\" already
      exists.\n";
    } else {
      $rfnames2{$rf1} = $i++;
    }
  } else {
    # Risk factor order left->right (top row) == top->bottom (leftmost
    column)?
    if (! (defined $rfnames2{$rf1})) {
      print STDERR "$ARGV[0], Row $.: risk factor \"$rf1\" does not
      exist.\n";
    }
    if (! (defined $rfnames2{$rf2})) {
      print STDERR "$ARGV[0], Row $.: risk factor \"$rf2\" does not
      exist.\n";
    }
  }

  # No NC category (warning if there is)?
  ! $opt_n && ! $opt_s && get_rf_category($rf1) eq "NC" &&
    print STDERR "$ARGV[0], Row $.: risk factor $rf1 is linked to category
    NC.\n";
  ! $opt_n && ! $opt_s && get_rf_category($rf2) eq "NC" &&
```

```
      print STDERR "$ARGV[0], Row $.: risk factor $rf2 is linked to category
      NC.\n";

    # Please note that we only fill triangle above diagonal
    $m2[$rfnames2{$rf2} - 1][$rfnames2{$rf1} - 1] = $fields[1];
    defined $opt_d && $opt_d > 2 && print "Matrix2[" . $rfnames2{$rf2} .
              "][$rfnames2{$rf1}]=" . $fields[1] . "\n";
  }

  # Matrix quadratic?
  if (scalar(@names2) != sqrt(2*$. - 3.75) + 0.5) {
    print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" .
    scalar(@names2) .
        ") != # of rows (" . (sqrt(2*$. - 3.75) + 0.5) . ")!\n";
    # exit(1);
  }

}

close(FILE);

! $opt_s && $dno2idx && printarr("Diagonal in matrix 2 not 1:", $dno2idx,
@diagnotone2);

# Warn about risk factors which are in first matrix but not in second
foreach (keys %rfnames1) {
  if (! defined $rfnames2{$_}) {
    print STDERR "Risk factors in $m1name but not in $ARGV[0]\n";
    last;
  }
}
foreach (keys %rfnames1) {
  if (! defined $rfnames2{$_}) {
    if ($_ eq $rftrans{$_}) {
      print STDERR $rftrans{$_} . "\n";
    } else {
      print STDERR $rftrans{$_} . ",$_\n";
    }
  }
}

# Warn about risk factors which are in second matrix but not in first
foreach (keys %rfnames2) {
  if (! defined $rfnames1{$_}) {
    print STDERR "Risk factors in $ARGV[0] but not in $m1name\n";
    last;
  }
}
foreach (keys %rfnames2) {
  if (! defined $rfnames1{$_}) {
    print STDERR $_ . "\n";
  }
}

################################################################
#                                                              #
#                     3. Compare Matrices                      #
#                                                              #
################################################################

# Calculate differences
```

```
defined $opt_d && $opt_d > 2 && print "#Rows=" . scalar(@names1) . "\n";
for ($i=0; $i<scalar(@names1); $i++) {
  defined $opt_d && $opt_d > 2 && print "Row," . $i . ",Start\n";
  defined $opt_d && $opt_d > 2 && print "#Columns=" . scalar(@names1) .
  "\n";
  for ($j=$i; $j<scalar(@names1); $j++) {
    defined $opt_d && $opt_d > 2 && print "Row," . $i . ",Column," . $j .
    ",Start\n";
    if (defined $names1[$i + 1] && defined $names1[$j + 1] &&
      defined $rfnames2{$names1[$i + 1]} && defined $rfnames2{$names1[$j +
      1]}) {
      $m2val = $m2[min($rfnames2{$names1[$i + 1]} - 1,
              $rfnames2{$names1[$j + 1]} - 1)][max($rfnames2{$names1[$i +
              1]} - 1,
          $rfnames2{$names1[$j + 1]} - 1)];
      $m3[$i][$j] = $m2val - $m1[$i][$j];
      if (defined $hashtnames{$names1[$i + 1]} && defined
      $hashtnames{$names1[$j + 1]}) {
        $t = $tol[min($hashtnames{$names1[$i + 1]} - 1,
          $hashtnames{$names1[$j + 1]} - 1)][max($hashtnames{$names1[$i +
          1]} - 1,
          $hashtnames{$names1[$j + 1]} - 1)];
      } else {
        $t = 0;
      }
      defined $opt_d && $opt_d > 2 && print "Matrix3[" . $i .
              "][$j]=" . $m3[$i][$j] . ",Tol=$t\n";
      ($tdidx, @totaldiff) = rankinsert($m3[$i][$j],
        get_rf_category($names1[$i + 1]) . ",$names1[$i + 1]," .
        get_rf_category($names1[$j + 1]) . ",$names1[$j + 1]," .
        $m1[$i][$j] . "," . $m2val,
        $devglobal[0], $devglobal[1], $t,
        $tdidx, @totaldiff)
        unless $rfignore{$names1[$i + 1]} || $rfignore{$names1[$j + 1]};
      # One complex statement intentionally omitted
      if (get_rf_category($names1[$i + 1]) ne get_rf_category($names1[$j +
      1])) {
        ($rfcatidx{get_rf_category($names1[$j + 1])},
          @{$rfcatrank{get_rf_category($names1[$j + 1])}}) =
          rankinsert($m3[$i][$j],
          get_rf_category($names1[$i + 1]) . ",$names1[$i + 1]," .
          get_rf_category($names1[$j + 1]) . ",$names1[$j + 1]," .
          $m1[$i][$j] . "," . $m2val,
          $devrfcat{get_rf_category($names1[$j + 1])}[0],
          $devrfcat{get_rf_category($names1[$j + 1])}[1], $t,
          $rfcatidx{get_rf_category($names1[$j + 1])},
          @{$rfcatrank{get_rf_category($names1[$j + 1])}})
          unless $rfignore{$names1[$i + 1]} || $rfignore{$names1[$j + 1]};
      }
      $rfcpairidx{get_rf_category($names1[$i + 1]) . "," .
        get_rf_category($names1[$j + 1])} ||= 0;
      ($rfcpairidx{get_rf_category($names1[$i + 1]) . "," .
        get_rf_category($names1[$j + 1])},
        @{$rfcpairrank{get_rf_category($names1[$i + 1]) . "," .
        get_rf_category($names1[$j + 1])}}) = rankinsert($m3[$i][$j],
        get_rf_category($names1[$i + 1]) . ",$names1[$i + 1]," .
        get_rf_category($names1[$j + 1]) . ",$names1[$j + 1]," .
        $m1[$i][$j] . "," . $m2val,
        $devrfcpair{get_rf_category($names1[$i + 1]) . "," .
        get_rf_category($names1[$j + 1])}[0],
        $devrfcpair{get_rf_category($names1[$i + 1]) . "," .
```

```
        get_rf_category($names1[$j + 1])}[1], $t,
        $rfcpairidx{get_rf_category($names1[$i + 1]) . "," .
        get_rf_category($names1[$j + 1])},
        @{$rfcpairrank{get_rf_category($names1[$i + 1]) . "," .
        get_rf_category($names1[$j + 1])}})
        unless $rfignore{$names1[$i + 1]} || $rfignore{$names1[$j + 1]};
      $ccy = substr($names1[$i + 1], 0, 3);
      $rfcccyidx{$ccy} ||= 0;
      ($rfcccyidx{$ccy}, @{$rfcccyrank{$ccy}}) = rankinsert($m3[$i][$j],
        get_rf_category($names1[$i + 1]) . ",$names1[$i + 1]," .
        get_rf_category($names1[$j + 1]) . ",$names1[$j + 1]," .
        $m1[$i][$j] . "," . $m2val,
        $devccy{$ccy}[0], $devccy{$ccy}[1], $t,
        $rfcccyidx{$ccy}, @{$rfcccyrank{$ccy}})
        unless $rfignore{$names1[$i + 1]} || $rfignore{$names1[$j + 1]};
      if ($ccy ne substr($names1[$j + 1], 0, 3)) {
        $ccy = substr($names1[$j + 1], 0, 3);
        $rfcccyidx{$ccy} ||= 0;
        ($rfcccyidx{$ccy}, @{$rfcccyrank{$ccy}}) = rankinsert($m3[$i][$j],
          get_rf_category($names1[$i + 1]) . ",$names1[$i + 1]," .
          get_rf_category($names1[$j + 1]) . ",$names1[$j + 1]," .
          $m1[$i][$j] . "," . $m2val,
          $devccy{$ccy}[0], $devccy{$ccy}[1], $t,
          $rfcccyidx{$ccy}, @{$rfcccyrank{$ccy}})
          unless $rfignore{$names1[$i + 1]} || $rfignore{$names1[$j + 1]};
      }
    }
    defined $opt_d && $opt_d > 2 && print "Row," . $i . ",Column," . $j .
    ",End\n";
  }
  defined $opt_d && $opt_d > 2 && print "Row," . $i . ",End\n";
}

#################################################################
#                                                               #
#               Now standard output to Stdout                   #
#                                                               #
#################################################################

print "$0,$version,";
print "Ignored risk factors in " . $opt_i if defined $opt_i;
print "\nSlice,[Category/Currency],[Category],\n";
print "Min/max difference";
print " beyond tolerance" unless ! $opt_b;
print ",Category 1,Risk Factor 1,Category 2,Risk Factor2,Old value,New
value\n\n";

$tdidx && printarr("GLOBAL,,", $tdidx, @totaldiff);
our $rfk;
foreach my $rfk (sort keys %rfcatidx) {
  $rfcatidx{$rfk} && printarr("CATEGORY,$rfk,", $rfcatidx{$rfk},
  @{$rfcatrank{$rfk}});
}
foreach $rfk (sort keys %rfcccyidx) {
  $rfcccyidx{$rfk} && printarr("CURRENCY,$rfk,", $rfcccyidx{$rfk},
  @{$rfcccyrank{$rfk}});
}
foreach $rfk (sort keys %rfcpairidx) {
  $rfcpairidx{$rfk} && printarr("CATEGORYPAIR,$rfk", $rfcpairidx{$rfk},
  @{$rfcpairrank{$rfk}});
}
```

```
$opt_w && close $doutfile;

exit(0);

sub printarr {
# Print array. With option -w write min & max into deviation file.
#
# Version Date       Author         Comment
# 1.0     23/03/2014 Bernd Plumhoff Create
#
  my ($title, $idx, @arr) = @_;
  my $i;

  print "$title\n";
  for ($i=0; $i<$idx; $i++) {
    printf STDOUT "%7.4f,%s,\n", $arr[$i][0], $arr[$i][1];
  }
  print "\n";

  if (defined $opt_w && $idx > 0) {
    # If we have min and/or max write them into deviation file.
    # Please note that in this version of Perl (5.12.4) the standard
    # precision of the %f format is 6 digits - which coincides with
    # our definition of $epsilon above.
    $arr[$opt_m][0] ||= 0;
    $arr[$opt_m + 1][0] ||= 0;
    $arr[$opt_m + 2][0] ||= 0;
    printf $doutfile "%s,%.6f,%.6f,%u,%u,%u\n", $title, $arr[0][0],
      $arr[$idx - 1][0],$arr[$opt_m][0],$arr[$opt_m + 1][0],
      $arr[$opt_m + 2][0];
  }
}

sub init_rf_category {
# Initialize risk factor hashtable with recognition patterns
#
# Synopsis: init_rf_category(RF_Category_filename)
#
# Example:  init_rf_category("./RF_Categories.csv");
#
# Version Date       Author         Comment
# 1.0     23/03/2014 Bernd Plumhoff Create
#
  open (FILE, "<", $_[0]);
  while (my $rfline = <FILE>) {
    $rfline =~ s/[\r\n]+//g;
    if (substr($rfline,0,2) ne '//' and 4 < length($rfline)) {
      my @rffields = split(",", $rfline);
      my $matchpattern = $rffields[0];
      if (substr($matchpattern,0,1) eq '#') {
        $matchpattern = '^(.*)' .
        substr($matchpattern,1,length($matchpattern) - 1) . '\(T[0-9]+\)$';
      } else {
        $matchpattern = '^(.*)\.' . $matchpattern . '$';
      }
      $rfcatpat{$matchpattern} = $rffields[3];
    }
  }
  close (FILE);
}
```

```
sub get_rf_category {
# Return risk factor category (Super Category - SubCategory) for a
recognized pattern
# or "NC-NC" if no pattern matched.
#
# Synopsis: get_rf_category(Risk_Factor)
#
# Example:  get_rf_category("USD.#SPREAD-CORP-BB(T1825)");
#           will result in "MR-CS-CORP".
#
# Version Date       Author         Comment
# 1.0     23/03/2014 Bernd Plumhoff Create
#
  defined $rfcat{$_[0]} && return $rfcat{$_[0]};
  my $searchpat = $_[0];
  my $success = "NC";
  $searchpat =~ s/\.\#/-/; # We have to substitute ".#" by "-" in order to
  match the patterns
  foreach my $pat (keys %rfcatpat) {
    # my $temp = $searchpat =~ /$pat/;
    # print $searchpat . " =~ \"" . $pat . "\" -> " . $temp . "\n";
    # Please note that if this approach is too slow we can try to apply
    # the map, study and eval commands to speed this up later.
    # See http://perldoc.perl.org/functions/study.html, for example. [Bernd
    25/01/2012]
    # General hints to speed perl up:
    http://www.ccl4.org/~nick/P/Fast_Enough/
    # [Bernd 08/03/2012]
    $success = $rfcatpat{$pat}, last if $searchpat =~ /$pat/;
  }
  if ($success eq "NC") {
    # This is a trick because the RAI regex are sometimes comparing
    # against $ when there still is a bracket term.
    # For example: CZK.#SWAP(T30)
    # In this case we omit the bracket term and look up CZK.#SWAP.
    $searchpat = substr($searchpat, 0, index($searchpat, "("));
    foreach my $pat (keys %rfcatpat) {
      $success = $rfcatpat{$pat}, last if $searchpat =~ /$pat/;
    }
  }
  $rfcat{$_[0]} = $success; # next call will be quicker
  $rfcatidx{$success} = 0;
  return $success;
}

sub rankinsert {
# Ranks and inserts value into array and stores text with it.
#
# Synopsis: rankinsert(value, text, deviation_lower_bound,
deviation_upper_bound,
#   tolerance, arrayindex, array[][])
#
# Example:  rankinsert($fields[$. - 1], "$ARGV[0], Row $.: $fields[0]", 2,
-2, 0,
#                                       $dno1idx, @diagnotone1);
#
# This function has intentionally been omitted
```