*Math*DAMP

Mathematica package for Differential Analysis of Metabolite Profiles
version 1.0.0

*Math*DAMP package facilitates the differential analysis of metabolite profiles. A visual approach is employed: (i) Individual datasets' retention/migration times and signal intensities are normalized. (ii) Arithmetic operations are applied to to all corresponding datapoints in the datasets to highlight differences or patterns of interest. (iii) The results are displayed on annotated density plots. Additionally, sets of overlaid chromatograms/electropherograms in the vicinities of the most significant differences from selected results are displayed in descending order of significance for visual confirmation.
The text notes throughout the code in this notebook are intended to provide brief comments on the implementation of particular functions. Actual reference documentation for each function is provided via the `usage` messages defined in the *Usage Description Section*. This information can be displayed by executing ?FunctionName (where FunctionName stands for the name of the function of interest) in any notebook after loading the *Math*DAMP package.
The package provides basic building blocks for performing differential analysis along with assemblies of core functions to perform common tasks. Specific examples about using the package for different types of differential analysis (comparing two datasets, detecting outliers within a group of datasets, comparing two groups of replicate datasets, comparing multiple groups of replicate datasets) along with additional information can be found at http://www.iab.keio.ac.jp/software/mathdamp/


```
BeginPackage["MathDAMP`"]

Needs["Graphics`MultipleListPlot`"]
Needs["Statistics`NonlinearFit`"]
Needs["Statistics`DescriptiveStatistics`"]
```

Usage Description Section

This section contains a list of explanations for MathDAMP's public functions. This information is shown upon executing the ?FunctionName in any notebook after loading the *Math*DAMP package.

```
DAMPStrInternalFormat = "{matrix of signal intensities, list of
   m/z values, list of timepoints, additional information (list of rules)}"
DAMPStrResolution = "specifies the resolution to which the
   data will be binned along the m/z axis"

DAMPImportMS::usage =
 "DAMPImportMS[filename] reads and processes an Agilent ChemStation MS file
    specified by filename into a MathDAMP format " <> DAMPStrInternalFormat <>
  ". This function is universal for SIM and scan mode datafiles. The
    data are binned to 1 m/z unit resolution by averaging."

DAMPImportCSV::usage =
 "DAMPImportCSV[filename,samplename,options] reads and processes a csv file generated by
    Analyst QS software for Agilent TOFMS and specified by filename into a MathDAMP format " <>
  DAMPStrInternalFormat <> ". Since the csv file does not contain information
    about the sample name, this has to be specified as the second
    parameter (samplename). \nOptions:\nResolution - " <> DAMPStrResolution <>
  " (default: 1)\nVerbose - if set to True, prints a notification along with current
    time every 100 processed chromatograms/electropherograms (default: True)"

DAMPImportMZXML::usage =
 "DAMPImportMZXML[filename,samplename,options] reads and processes the first level MS
    scans from an mzXML data file specified by filename into a MathDAMP format " <>
  DAMPStrInternalFormat <> ". The sample name is specified via the second
    parameter (samplename). \nOptions:\nResolution - " <> DAMPStrResolution <>
  " (default: 1)\nImportMode - determines the m/z dimension elements. If set to Sequential,
    the m/z dimension is represented by a discrete range of values determined by the smallest
    and the largest m/z value in the imported dataset (rounded according to the resolution).
    Resolution determines the stepsize of the discrete range. If set to Selective, the
    m/z dimension is represented by only those m/z values which are present in the
    imported dataset (rounded according to the resolution). Sequential mode is recommended
    for scan data, Selective mode is recommended for SIM data. (default: Sequential)"
```

```
DAMPImportCDF::usage =
 "DAMPImportCDF[filename,options] reads and processes a CDF file specified
    by filename into a MathDAMP format " <> DAMPStrInternalFormat <>
  ". This function was not thoroughly tested with CDF files generated by
    different software and may not be universal.  \nOptions:\nResolution - " <>
  DAMPStrResolution <> " (default: 1)"

DAMPImportBDT::usage =
 "DAMPImportBDT[filename,samplename] reads and processes a binary BDT file
    specified by filename into a MathDAMP format " <> DAMPStrInternalFormat <>
  ". The BDT is a binary datafile used internally by The Institute for Advanced
    Biosciences (Keio University). Separate in house software for preprocessing
    of the Analyst QS generated csv files stores the data in the BDT format.
    Since the original csv file does not contain information about the sample
    name, this has to be specified as a second parameter (samplename)."

DAMPExportBDT::usage =
 "DAMPExportBDT[filename,msdata] exports data in a MathDAMP format " <> DAMPStrInternalFormat <>
  " into a BDT format file specified by filename. The m/z values in the
    data are not saved one by one but as a range specified by min, max and
    step. Because of this, only data having a regular list of m/z values
    which can be converted to the range representation should be exported."

DAMPAppendToSampleName::usage =
 "DAMPAppendToSampleName[msdata4,string] appends string to the SampleName in msdata4. msdata4
   is expected to be a list of rules (format as the fourth element in the msdata format).
   The added string is enclosed inside two vertical bars. If previous modifiers are
   present, the newly added string will be separated from these by a single vertical
   bar.\nOptions:\nShareBars - if set to False, the newly added string is separated
   from previous modifiers by two vertical bars with a space in between (default: True)"

DAMPLoadAnnotationTable::usage =
 "DAMPLoadAnnotationTable[filename] loads a list of annotation labels for plots
   from a csv file. The organization of columns in the csv file is assumed to be as
   follows: (1) m/z value, (2) retention/migration time, (3) short compound name/id (
   density plot label text), (4) full compound name (chromatogram/electropherogram
   label text), and (5) label text position on the density plots relative to
   the label (1 - right, 2 - top, 3 - left, 4 - bottom, 1.5 - top right, etc.)."

DAMPGetIntensities::usage = "DAMPGetIntensities[msdata,mz]
   returns a list of signal intensities from msdata corresponding to m/z mz."

DAMPGetChromatogram::usage =
 "DAMPGetChromatogram[msdata,mz] returns a chromatogram/electropherogram (a list
   of {time,signal intensity} elements) from msdata corresponding to m/z mz."

DAMPCrop::usage =
 "DAMPCrop[msdata,options] reduces the msdata dataset to datapoints falling within the
   timerange and m/z value range specified by options:\nmzRange - two element list specifying
   the cropping m/z value range (default: All)\nTimeRange - two element
   list specifying the cropping time range (default: All)."

DAMPSelectMZs::usage = "DAMPSelectMZs[msdata,mzs] reduces the msdata
   datased to datapoints corresponding to m/z values specified in the mzs list."

DAMPDropMZs::usage = "DAMPDropMZs[msdata,mzs] reduces the msdata datased by
   eliminating datapoints corresponding to m/z values specified in the mzs list."

DAMPPlotChromatogram::usage =
 "DAMPPlotChromatogram[{msdata,msdata,...},mz,options] plots multiple overlaid chromatograms/
   electropherograms from the list of msdatas corresponding to m/z mz (mz may be a list
   of m/z values as well). The DAMPGenColors function is used by default to assign colors
   to individual chromatograms/electropherograms. Custom colors may be specified via the
   PlotOptions option (and the enclosed PlotStyle option). \nOptions:\nPlotOptions - list
   of options for the MultipleListPlot function which is used internally for plotting (
   default: see the output of Options[DAMPPlotChromatogram]),\nAnnotationTable - annotation
   table for the labeling of the chromatogram/electropherogram (default: {})\nResolution -
   resolution to which the msdata were binned along the m/z dimension. The annotation
   table passed through the AnnotationTable option will be rounded accordingly to ensure
   the appropriate appearance of annotation labels on the chromatogram/electropherogram (
   default: 1)\nLegendData - a list of legend elements {{color,label},...}. If Automatic
   is specified, the SampleName from each msdata is used as a label (default: Automatic)"

DAMPGradient::usage =
 "DAMPGradient[pos1,pos2,color1,color2,backgroundgraylevel] returns a function which
   calculates the parameters for the Hue function of a specific point within a
   gradient. The relative location <0;1> of the point of interest within the gradient
   is the only parameter of the returned function. The color gradient consists of
   three subgradients defined by the parameters of the DAMPGradient function. The
   gradient starts with backgroundgraylevel <0;1> at relative position 0, then
   proceeds to color1 (numeric value corresponding to hue specification <0;1>) at pos1,
   and color2 at pos2, and finally to darkened color2 at the relative position 1."
```

```
DAMPGradientPalette::usage =
 "DAMPGradientPalette[options] - generates a color palette for assigning colors to intensity
   values on density plots. It uses the DAMPGradient function to create the gradients
   for both positive and negative ranges. Both gradients consist of three subgradients (
   execute ?DAMPGradient for details).\nOptions:\nColorPositions - list with two
   elements specifying the relative location within the gradient of endpoints of the
   middle subgradient (default: {.075,.4})\nPositiveColors - hue color specification
   of the endpoints of the middle subgradient in the positive range color gradient (
   default: {1/6,0})\nNegativeColors - hue color specification of the endpoints of
   the middle subgradient in the negative range color gradient (default: {1/2,2/3})\n
   BackgroundGrayLevel - gray level of the density plot's background area (default: .7)\n
   PaletteSize - number of palette entries (subdivisions) of both positive and negative
   range color gradients, palette entry for value 0 is shared between the two (default: 100)"

DAMPDrawAnnotation::usage =
 "DAMPDrawAnnotation[annotationtable,options] draws the annotation for density plots. (For
   details about the format of the annotationtable, execute ?DAMPLoadAnnotationTable.)\n
   Options:\nTimeRange - time range for which the annotation should be drawn (default:
   {0,50})\nMZs - list of m/z values for which the annotation should be drawn. The order
   of m/z values in the mzs list determines the positioning of annotation labels along
   the m/z scale (default: Range[50.,1000.,1.])\nResolution - resolution to which the
   underlying data were binned. The m/z values from the annotation table will be rounded
   accordingly to ensure the appropriate appearance of the annotation labels (default:
   1)\nScaleCoefficients - coefficients used to convert the scales (currently used for
   timescale only, the second item is 1 by default) of the annotation table to the scales
   of the ListDensityPlot (default: {62.2488,1})\nTextStyle - style options for the
   annotation label's text (default: Join[{FontSize->4},DAMPTextStyle])\nLabelStyle -
   style options for the annotation label's mark (default: {AbsoluteThickness[.25]})
   \nLabelShape - pure function to draw the annotation label's mark. Two parameters
   are passed to the function, first determining the position {xpos,ypos}, and
   second determining the size {xsize,ysize}. (default: Circle[#1,#2]&)\nLabelSize -
   size of the annotation label's mark (relative units). When relative size {1,1}
   is used by the Circle function as radius, an ellipse is drawn which appears to
   appropriately capture the dimensions of peaks on the ListDensityPlot (default: {1,1})"

DAMPDensityPlot::usage =
 "DAMPDensityPlot[msdata,options] plots the msdata using the ListDensityPlot function. A
   gradient palette is used for representing the signal intensities and the plot may be
   annotated to allow easier identification of peaks.\nOptions:\nMaxScale - determines
   the extent of the signal intensity scale (default: Automatic)\nLogScale - determines
   whether the signal intensities should be displayed using a logarithmic scale (default:
   False)\nFrameTickFreqs - frequencies at which tickmarks are placed on the time axis (
   frequency in minutes) and on the m/z axis (in terms of the number of elements in the
   msdata's mz list) (default: {1,1})\nFrameTickOffsets - determines the positions with
   respect to the origin of msdata's dimensions at which the tickmarks are started to
   be placed (default: {0,0})\nmzTickShift - shift of tickmark positioning on the m/z
   axis (default: -0.5)\nmzFrameTicks - a list of custom frame tickmarks to be shown
   on the m/z axis (default: Automatic)\nPalette - list of color specifications to be
   used for representing the signal intensity values (default: DAMPGradientPalette[])\n
   mzGridLineFreq - frequency of horizontal gridlines in terms of the number of elements
   in the msdata's m/z value list. Enter a list of values to place the gridlines at
   certain specified positions (default: 5)\nmzGridLineStyle - style options for horizontal
   gridlines (default: {{AbsoluteThickness[0.25],GrayLevel[0.5]},{AbsoluteThickness[0.25],
   GrayLevel[0.5],Dashing[{0.01,0.01}]}})\nAnnotationTables - list of annotation tables to
   be overlaid on the plot (default: None)\nPlotOptions - options for the ListDensityPlot
   function (default: {Mesh->False,ImageSize->930,AspectRatio->0.35,FrameLabel->{\"Time (
   min)\",\"m/z\"},TextStyle->DAMPTextStyle})\nAnnotationOptions - lists of options for
   the DAMPDrawAnnotation function for each annotation table. The number of lists of
   options must correspond to the number of passed annotation tables (default: Automatic)"

DAMPParallelPlot::usage =
 "DAMPParallelPlot[msdatas,options] plots msdatas (a list of datasets) on a density plot
   in a parallel format, so that chromatograms/electropherograms from the datasets
   corresponding to the same m/z value appear next to each other. The datasets do not
   have to have chromatograms corresponding to an identical set of m/z values, neither
   do the datasets have to be aligned. In the latter case, the time axis is labeled
   according to the first dataset in msdatas (please note that the time axis may be
   misleading for the remaining datasets in this case). The options are passed directly
   to the DAMPDensityPlot function which is used internally for plotting the data."

DAMPApplyFunctionToSingle::usage =
 "DAMPApplyFunctionToSingle[msdata,function,options] applies a pure function function to
   msdata.\nOptions:\nSampleNameSuffix - string to be added to the SampleName from
   the msdata to keep track of modifications performed on the dataset (default: \"\")"

DAMPBinChromatogram::usage =
 "DAMPBinChromatogram[chromatogram,binsize,options] bins the chromatogram
   by dissecting it into intervals of size binsize (minutes) and applying the
   binning function passed as an option to signal intensities and timepoints
   in every interval.\nOptions:\nBinningFunction - function to be applied
```

```
   to signal intensities and timepoints in every interval (default: Mean)"

DAMPBinChromatogramFast::usage =
 "DAMPBinChromatogramFast[chromatogram,binsize] bins the chromatogram by dissecting it
   into intervals of approximate size binsize (in minutes) and averaging the signal
   intensities and timepoints in every interval. This function is compiled and intended
   for fast binning. The time interval to which the data are binned may differ slightly
   from the one which is passed as the binsize parameter. The binsize is adjusted
   so that every interval contains equal number of datapoints. Please refer to the
   function DAMPBinChromatogram for a more flexible (and slower) way to bin the data."

DAMPRobustPolynomialFit::usage =
 "DAMPRobustPolynomialFit[chromatogram,options] fits a polynomial to chromatogram
   data by robust nonlinear regression and returns a dataset in which the signal
   intensity is calculated from the polynomial for each timepoint. The function
   is intended for baseline fitting. Please refer to the MathDAMP.nb notebook for
   details about implementation.\nOptions:\nPolynomialDegree - degree of polynomial
   which is going to be fitted to the data (default: 1)\nConvergenceCriteria -
   convergence criteria for nonlinear regression (default: 0.001)\nMaxIterations -
   maximum number of outer (recalculating the weights) iterations (default: 10)\nb -
   parameter for calculating weights (default: 4.05)\nBinningFunction - function to
   bin the chromatogram data before polynomial fitting. This may increase the speed
   considerably without considerable loss of precision. Use BinningFunction->None not
   to bin the data for polynomial fitting (default: (DAMPBinChromatogramFast[#,0.25]&))"

DAMPSubtractBaselines::usage =
 "DAMPSubtractBaselines[msdata,options] subtracts baselines from all chromatograms/
   electropherograms in msdata dataset and returns the new processed dataset.\nOptions:\n
   BaselineFittingFunction - pure function which is expected to return the baseline points
   for every timepoint in a chromatogram/electropherogram passed as an argument (default:
   DAMPRobustPolynomialFit)\nSampleNameSuffix - string to be added to the SampleName from
   the msdata to keep the track of modifications performed on the dataset (default: \"bs\")"

DAMPRemoveNoise::usage =
 "DAMPRemoveNoise[msdata,options] removes noise from every chromatogram/electropherogram
   in msdata by leveling to 0 all signal intensities, absolute values of which are
   smaller than a threshold. The threshold is calculated as a specific multiple of
   the standard deviation of signal intensities from a specified time range of every
   chromatogram/electropherogram.\nOptions:\nTimeRange - time range (in minutes)
   from which to calculate the standard deviation of signal intensities (default:
   {1,3})\nSDThreshold - specifies the multiple of the standard deviation of signal
   intensities from the selected time range to be used as the noise discrimination
   threshold (default: 5)\nLevelNegativeSignals - if set to True, all negative signal
   intensities in every chromatogram/electropherogram will be leveled to 0 (default:
   True)\nSampleNameSuffix - string to be added to the SampleName from the msdata
   to keep the track of modifications performed on the dataset (default: \"nr\")"

DAMPRemoveSpikes::usage =
 "DAMPRemoveSpikes[msdata,options] levels to 0 all signal intensity values both neighbours
   of which have 0 signal intensity value in every chromatogram/electropherogram in
   msdata.\nOptions:\nSampleNameSuffix - string to be added to the SampleName from the
   msdata to keep the track of modifications performed on the dataset (default: \"sr\")"

DAMPThreshold::usage =
 "DAMPThreshold[msdata,threshold,options] levels to 0 all signal intensity
   values in msdata which are within ±threshold.\nOptions:\nSampleNameSuffix -
   string to be added to the SampleName from the msdata to keep
   track of modifications performed on the dataset (default: \"t\")"

DAMPMovingAverageFast::usage =
 "DAMPMovingAverageFast[intensities,windowsize] applies a moving average filter to a list of
   signal intensities from a chromatogram/electropherogram. The new value for a signal
   intensity is calculated by averaging it with a number of preceding and subsequent signal
   intensity values given by Round[(windowsize-1)/2]. This means the calculated new value
   is in the middle of the smoothing window with equal number of preceding and subsequent
   signal intensity values to be averaged with. Only the available signal intensities are
   used for the calculation of the average at the beginning or at the end for the list."

DAMPSmooth::usage =
 "DAMPSmooth[msdata,options] applies a smoothing function to all chromatograms/
   electropherograms in msdata.\nOptions:\nSmoothingFunction - pure function to be applied
   to the list of signal intensities from a chromatogram/electropherogram (default:
   DAMPMovingAverageFast[#,9]&)\nSampleNameSuffix - string to be added to the SampleName
   from the msdata to keep track of modifications performed on the dataset (default: \"s\")"

DAMPPickChromatogramPeaks::usage =
 "DAMPPickChromatogramPeaks[chromatogram,options] picks peaks from chromatogram and returns
   them as list {{Retention Time,Signal Intensity},{Retention Time,Signal Intensity},...}.
   \nOptions:\nThreshold - minimum vertical distance of a new strategic point from the
   line defined by the two neighbouring strategic points. Please refer to the MathDAMP.
   nb notebook for a more detailed description of the peak picking algorithm (default:
   1000)\nNodeRestrictedVicinity - neigbourhood of a strategic point (in minutes) from
```

```
    which new strategic points can not be picked (default: 0.05)\nCentroidVicinity –
    determines the time interval (in minutes) around the peak-top strategic point
    for the calculation of centroided retention/migration time (default: 0.05)"

DAMPPickPeaks::usage =
 "DAMPPickPeaks[msdata,options] picks peaks from all chromatograms/electropherograms in
    msdata and returns them in a list {{m/z value,peaklist},{m/z value,peaklist},...}. The
    function does not have any default options, the options passed to the DAMPPickPeaks
    function are passed further to the DAMPPickChromatogramPeaks function which is
    used internally to pick peaks from individual chromatograms/electropherograms."

DAMPPeakListToAnnotationTable::usage =
 "DAMPPeakListToAnnotationTable[peaklist,options] converts the peaklist returned
    by DAMPPickPeaks to an annotation table format for plots. The peak names are
    assigned from sequential numbering.\nOptions:\nTextLabelPosition – specifies the
    position of the text label relative to the annotation label mark (default: 1.5)"

DAMPSelectRepresentativePeaks::usage =
 "DAMPSelectRepresentativePeaks[peaklist,options] reduces the peaklist returned
    by the DAMPPickPeaks function to a selected number of highest peaks in every
    chromatogram/electropherogram and a selected number of highest peaks in every
    time interval of selected size.\nOptions:\nPeaksPerChromatogram – number of
    highest peaks to select from every chromatogram/electropherogram (default: All)
    \nPeaksPerInterval – number of highest peaks to select from every time interval (
    default: All)\nIntervalSize – size of the time interval (in minutes) for selection
    of highest peaks when PeaksPerInterval is not set to All (default: 1)\nTimeRange –
    select peaks from this retention/migration time range only (default: All)"

DAMPPlotPeakLayout::usage =
 "DAMPPlotPeakLayout[peaklists,options] plots the positions of peaks from a list of
    peaklists which are expected to have a format as returned by DAMPPickPeaks. Symbols'
    colors are by default assigned using the DAMPGenColors function. Custom colors
    may be passed via the PlotOptions option (and the enclosed SymbolStyle option).\n
    Options:\nmzGridLineFreq – frequency of horizontal gridlines on plot (default:
    1)\nmzTickFreq – frequency of labeled tickmarks on the m/z axis (default: 1)\n
    LegendItems – list of strings to be shown as legend labels (default: Automatic)\n
    PlotOptions – list of options to be passed to the MultipleListPlot function used
    internally to generate the plot (default: {TextStyle→DAMPTextStyle,Frame→True,
    ImageSize→600,SymbolShape→{PlotSymbol[Diamond,5,Filled→False],PlotSymbol[Box,5,
    Filled→False],PlotSymbol[Diamond,3,Filled→False],PlotSymbol[Box,3,Filled→False]}})"

DAMPDPScore::usage =
 "DAMPDPScore[peaklist1,peaklist2,options] calculates the dynamic programming (
    DP) score between two peaklists having format as returned by the DAMPPickPeaks
    function.\nOptions:\nGapPenalty – gap penalty (in minutes) for DP (default: 0.5)"

DAMPFitShiftFunction::usage =
 "DAMPFitShiftFunction[peaklist1,peaklist2,options] optimizes the parameters of a retention/
    migration time shift function so that when applied to peaklist2 the optimum peak
    alignment (as measured by DAMPDPScore) is achieved.\nOptions:\nShiftFunction – pure
    function to be used as a retention/migration time shift function (default: (1/(1/(α #)+
    γ/2)&))\nShiftFunctionParameters – list of parameters for optimization. If Automatic
    is specified, these are extracted automatically from ShiftFunction. The parameters
    may be also specified explicitly with seek ranges (default: {{α,0.8,1.2},{γ,-0.04,
    0.04}})\nGapPenalty – gap penalty value to be passed to the DAMPDPScore function for
    the scoring of the alignment. A list of gap penalty values may be passed to perform
    the fitting of the retention/migration time shift function iteratively (default:
    {3,0.5})\nNMinimizeOptions – list of options to be passed to the NMinimize function
    used for optimization (default: {MaxIterations->1000})\nTimeRange – specifies the
    selection time range of peaks from peaklist1 to be used for scoring (default: {0,∞})"

DAMPAlignPeakList::usage =
 "DAMPAlignPeakList[peaklist,timeshiftfunction] applies the retention/migration
    time shift function timeshiftfunction to all retention/migration times in the
    peaklist (expected to have format as returned by the function DAMPPickPeaks)"

DAMPAnnotationTableToPeakList::usage =
 "DAMPAnnotationTableToPeakList[annotationtable,options] converts an annotation table (
    execute ?DAMPLoadAnnotation for format description) to a peaklist with format as
    returned by the function DAMPPickPeaks.\nOptions:\nResolution – determines to which
    resolution the m/z values in the annotation table will be rounded (default: 1)"

DAMPAlignAnnotationTable::usage =
 "DAMPAlignAnnotationTable[peaklist,annotationtable,options] aligns annotationtable to
    peaklist and returns the new annotation table. Options for DAMPFitShiftFunction, which
    is used internally, may be passed directly via options. Additional option Resolution
    determines the rounding of m/z values in the annotation table. If the resulting annotation
    table is intended to be used on a density plot where the underlying data were binned
    to 0.1 m/z resolution, the resolution option should be set to 0.1 as well (default: 1)"

DAMPAlign::usage =
 "DAMPAlign[msdata,shiftfunction,timepoints,options] aligns msdata according to timeshift
```

```
   function shiftfunction and selects (by interpolation) time points identical to the ones in
   the timepoints list.\nOptions:\nSampleNameSuffix – string to be added to the SampleName
   from the msdata to keep track of modifications performed on the dataset (default: \"a\")"

DAMPNormalize::usage =
 "DAMPNormalize[msdata,coefficient,options] multiplies the signal intensities
   in msdata (msdata[[1]]) with coefficient.\nOptions:\nSampleNameSuffix – string
   to be added to the SampleName from the msdata to keep the track
   of modifications performed on the dataset (default: \"n\")"

DAMPIntegrate::usage =
 "DAMPIntegrate[chromatogram,options] calculates the area below the signal intensities of a
   chromatogram/electropherogram within the retention/migration time range specified by
   the option TimeRange (default: {0,∞}). Baseline may be calculated as an average of
   signal intensities within the timerange specified by the options BaselineFromTimeRange (
   default: None). If set to None, baseline is set to signal intensity value 0."

DAMPFilter::usage =
 "DAMPFilter[msdata,criteriamsdata,threshold,options] sets to 0 those signal intensities
   in msdata for which the aboslute values of corresponding signal intensities in
   the criteriamsdata dataset are not equal to or greater than threshold.\nOptions:\n
   SampleNameSuffix – string to be added to the SampleName from the msdata to keep track
   of modifications performed on the dataset (default: \"f\")\n\nDAMPFilter[msdata,
   criteriamsdata,filterfunction,options] pure function filterfunction is applied to
   the signal intensity matrix of criteriamsdata. For zero signal intensities in the
   result, the corresponding signal intensities in the msdata dataset are set to 0 as
   well.\nOptions:\nSampleNameSuffix – string to be added to the SampleName from the
   msdata to keep the track of modifications performed on the dataset (default: \"f\")"

DAMPTrendFilter::usage =
 "DAMPTrendFilter[group1msdatas,group2msdatas,tofiltermsdata,filtnum,options] filters out
   those signal intensities from a result (tofiltermsdata) of two averaged groups of
   msdatas where at least a certain number of corresponding signal intensities from the
   individual datasets specified by filtnum follows the same trend as their averages.\n
   Options:\nSampleNameSuffix – string to be added to the SampleName from the tofiltermsdata
   to keep the track of modifications performed on the dataset (default: \"tf\")"

DAMPApplyFunctionToGroup::usage =
 "DAMPApplyFunctionToGroup[msdatas,function,options] applies function to either all
   corresponding signal intensities in the datasets or to the msdatas (decided by option).
   \nOptions:\nApplyToIntensitiesOnly – if set to true, the function is applied to the
   corresponding signal intentisites in the msdatas (instead of to the whole msdatas
   list) (default: True)\nResultSampleName – string to be set as the SampleName of the
   resulting dataset. If set to Automatic, the SampleName of the first dataset from msdatas
   is used (default: Automatic)\nSampleNameSuffix – string to be added to the SampleName
   to keep the track of the modifications performed on the dataset (default: \"\")"

DAMPCheckDir::usage = "DAMPCheckDir[directory] checks if directory exists, creates it if
   it does not. The parent directory must exist, error message is shown otherwise."

DAMPGenColors::usage =
 "DAMPGenColors[number] generates a list of color specifications with a number of elements
   determined by the parameter number. If number<=6, the colors are assigned from
   the following sequence: blue, red, green, yellow, cyan, magenta. If number>6, the
   hue range is split proportionally and the colors are assigned from this sequence."

DAMPNormalizeGroup::usage =
 "DAMPNormalizeGroup[msdatas,options] aligns msdatas (a list of datasets) and normalizes
   them according to the areas of the peaks of the internal standard and external
   normalization coefficients (optional). The results are returned as a list of rules:
   {NormalizedDatasets->...,AlignedAnnotationTables->...}\nOptions:\nReference – position
   of the reference dataset within msdatas to which the remaining msdatas will be
   normalized (default: 1)\nAlignmentTimeRange – peak picks from the reference dataset
   falling within this timerange only (specified as {starttime,endtime} in minutes)
   will be used for alignment (default: All)\nRepresentativePeakOptions – options
   to be passed to the DAMPSelectRepresentativePeaks function to filter the initial
   peak picks (default: {PeaksPerChromatogram->5,PeaksPerInterval->8,IntervalSize->
   .5})\nPeakPickingOptions – options to be passed to the DAMPPickPeaks function (
   default: {Threshold->5000})\nPeakLayoutPlotOptions – options to be passed to the
   DAMPPlotPeakLayout function (default: {})\nFitShiftFunctionOptions – options to be
   passed to the DAMPFitShiftFunction function (default: {})\nAnnotationTables – a
   list of annotation tables to be aligned to the reference msdata (default: None)\n
   OutputTimeRange – time range to which the resulting normalized datasets should be
   cropped (default: All)\nExternalNormalizationCoefficients – list of coefficients
   by which the signal intensities in msdatas will be multiplied. The number of
   coefficients in the list must equal the number of datasets in msdatas (default: None)
   \nResolution – resolution to which the datasets were binned along the m/z dimension.
   The annotation tables passed through the AnnotationTables option will be binned the
   same way (default: 1)\nInternalStandard – the internal standard for signal intensity
   normalization may be specified in one of two ways: 1) short name (3rd column) from
   the first annotation table in the list passed via the AnnotationTables option.
   In this case the position of the internal standard will be extrapolated from the
```

```
    aligned annotation table and the vicinity blindly integrated 2) specifying the m/z
    and integration time range (as in the reference dataset) explicitly: {mz,{starttime,
    endtime}}. (default: None)\nAutoISIntegrationVicinity - if the location of the internal
    standard is extrapolated from the aligned annotation table, this option determines
    the vicinity (in minutes) of the predicted retention/migration time to be blindly
    integrated (default: {-.25,.25})\nSaveMemory - if set to true, signal intensities
    are rounded to integers in internal calculations and results (default: True)"

DAMPTwoDatasets::usage =
  "DAMPTwoDatasets[msdata1,msdata2,options] generates datasets representing the absolute,
    relative, and absolute×relative differences between msdata2 and msdata1 and returns
    them along with the normalized datasets and aligned annotation tables as a list of
    rules: {NormalizedDatasets->...,Absolute->...,Relative->...,AbsoluteRelative->...,
    AlignedAnnotationTables->...}\nOptions:\nNormalizeGroupOptions - a list of options
    for the DAMPNormalizeGroup function which is used internally to normalize the
    datasets (default: {})\nThresholdForRelative - relative difference in the relative
    result will be set to 0 if neither of the two corresponding signal intensities
    from msdata1 and msdata2 are equal to or greater than this threshold (default: 0)"

DAMPOutliers::usage =
  "DAMPOutliers[msdatas,options] highlights the presence of outlying signal intensities in
    msdatas (a list of datasets). Two types of result datasets are generated: a z-score
    map and a quartile-based result. These datesets show the result for the most outlying
    signal intensity from withing the set of corresponding signal intensities. Optionally,
    the results may be generated individually for every dataset in msdatas. The results
    are returned as a list of rules: {NormalizedDatasets->...,AveragedDatasets->...,
    ZScores->...,ZScoresIndividual->...,QuartileResult->...,QuartileResultIndividual->...,
    AlignedAnnotationTables->...}\nOptions:\nNormalizeGroupOptions - a list of options
    for the DAMPNormalizeGroup function which is used internally to normalize the
    datasets (default: {})\nOutliersToDrop - specifies the number of outliers to be
    dropped from the set of corresponding signal intensities prior to calculating
    the mean and standard deviation for the z-score calculation (default: 1)\n
    IndividualZs - if set to true, z-score map will be calculated for every dataset
    in msdatas in addition to the overall result (default: False)\nIndividualQs -
    if set to true, results based on the interquartile range will be generated for
    every dataset in msdatas in addition to the overall result (default: False)"

DAMPTwoGroups::usage =
  "DAMPTwoGroups[msdatas1,msdatas2,options] generates datasets representing the absolute,
    relative, and absolute×relative differences between the averaged datasets of msdatas2
    datasets and msdatas1 datasets. Additionally, a dataset representing t-scores between
    the groups of corresponding signal intensities from msdatas1 and msdatas2 datasets
    is generated. The results are returned as a list of rules: {NormalizedDatasets->...,
    AveragedGroup1->...,AveragedGroup2->...,Absolute->...,Relative->...,AbsRel->...,
    FilteredAbsRel->...,TScores->...,AlignedAnnotationTables->...,GroupCounts->...,
    GroupNames->...}\nOptions:\nNormalizeGroupOptions - a list of options for the
    DAMPNormalizeGroup function which is used internally to normalize the datasets (
    default: {})\nThresholdForRelative - relative difference in the relative result will
    be set to 0 if neither of the two corresponding signal intensities (from the averaged
    datasets) is equal to or greater than this threshold (default: 0)\nGroupNames -
    names to assign to groups (will be combined into the SampleName of the results).
    If set to Automatic, the SampleName of the first dataset from every group is used (
    default: Automatic)\nAbsRelTrendFilter - determines the minimum number of individual
    corresponding signal intensities from every group which must follow the same trend
    as their averages to remain in the absolute×relative result. This is intended to
    filter out results originating from individual spikes or outliers (default: 2)"

DAMPMultiGroups::usage =
  "DAMPMultiGroups[msdatas,replicates,options] performs the comparison of multiple
    groups of datasets containing identical number of replicates determined by the
    replicates parameter. msdatas is expected to contain an ungrouped list which
    will be later grouped according to the number of replicates. The results are
    returned as a list of rules {NormalizedDatasets->...,AlignedAnnotationTables->...,
    FRatios->...,GroupNames->...}\nOptions:\nNormalizeGroupOptions - a list of options
    for the DAMPNormalizeGroup function which is used internally to normalize the
    datasets (default: {})\nGroupNames - names to assign to groups (will be combined
    into the SampleName of the resulting FRatios dataset). If set to Automatic, the
    SampleName of the first dataset from every group is used (default: Automatic)"

DAMPPlotCandidates::usage =
  "DAMPPlotCandidates[msdatas,criteriamsdata,options] plots a ranked list of overlaid
    chromatograms/electropherograms from msdatas datasets in the vicinities of the most
    significant signals in the criteriamsdata dataset.\nOptions:\nPlotCount - number of
    chromatograms/electropherograms to plot (default: 12)\nTimeRange - the most significant
    signals will be searched in the criteriamsdata dataset in this retention/migration time
    range only (default: All)\nPlotVicinity - retention/migration time range to show on the
    chromatogram/electropherogram around the candidate difference in minutes (default: {-.6,
    .6})\nDropVicinity - after a particular datapoint in the criteriamsdata is selected
    as the most significant signal, the vicinity of this datapoint determined by this
    option (in minutes) will no longer be searched in subsequent iterations (default: .2)
    \nPlotChromatogramOptions - options to be passed to the DAMPPlotChromatogram function
    which is used internally to plot the chromatograms/electropherograms (default: {})"
```

```
Clear[DAMPStrInternalFormat]
Clear[DAMPStrResolution]
```

Public Variables

`DAMPTextStyle` contains the default text formatting specifiers for plots. Since different font sizes seem to be suitable for printing and on screen inspection, the font settings are central (and have `Global‘` context) and can be easily overridden in particular notebooks. Redefining the `DAMPTextStyle` in the analysis notebook makes all subsequently generated plots use the settings. Text settings can be redefined on an individual basis for each plot via the `TextStyle` option in the `PlotOptions` option for the particular plot. `DAMPCETOFMSDensityPlotOptions`, `DAMPCETOFMSPeakLayoutOptions`, `DAMPCEQMSDensityPlotOptions`, and `DAMPCEQMSPeakLayoutOptions` are lists of options for optimal display of data from different sources. Since the data dimensions of CE−TOFMS and CE−QMS datasets differ significantly, different settings for density plots and peak layout plots are desirable (tickmark/gridline frequency, plot size, etc.) These lists can then be specified as a whole or parts overridden by joining a list with overriding/additional options with the list represented by any of these global variables. The lists are empty for the CE−QMS data since the options for underlying function (`DAMPPlotDensityPlot`, `DAMPPlotPeakLayout`) are set by default to the optimum for CE−QMS data.

```
Global‘DAMPTextStyle = {FontFamily → "Helvetica", FontSize → 8};

Global‘DAMPCETOFMSDensityPlotOptions = {Global‘MaxScale → 20000, Global‘FrameTickFreqs → {1, 10},
   Global‘Palette → DAMPGradientPalette[Global‘BackgroundGrayLevel → 1],
   Global‘mzGridLineFreq → 10, Global‘PlotOptions → {ImageSize → 940, AspectRatio → 1.43}};
Global‘DAMPCETOFMSPeakLayoutOptions = {Global‘mzGridLineFreq → 50, Global‘mzTickFreq → 50};

Global‘DAMPCEQMSDensityPlotOptions = {};
Global‘DAMPCEQMSPeakLayoutOptions = {};
```

Begin Private Section

```
Begin["‘Private‘"]
```

Data Import/Export

Imported datasets are processed into a nested list (further referred to as `msdata`). The first element is a matrix containing ion signal intensities, rows corresponding to individual chromatograms/electropherograms. The second element is a list of m/z values and the third element is a list of scan timepoints (in minutes). The dimensions of the signal intensity matrix are determined by the length of the lists containing the m/z values and timepoints. The fourth element is a list of rules containing additional information about the dataset. Currently, it is used to store the sample name only.

Sample dataset with 2 m/z values and 6 timepoints in *Math*DAMP internal representation:
`{{{0,0,1,3,1,0},{0,0,0,0,0,0}},{100.,150.},{1,2,3,4,5,6},{SampleName→"Dummy Profile"}}`

The datasets may be imported in two modes: sequential and selective. The modes determine the way in which the m/z values are generated (along with the matrix of signal intensities). In the sequential mode, the list of m/z values will contain an ordered list of m/z values in which the difference between two neighboring m/z values (step size) will be fixed. The values of the first and the last element in the m/z list are determined by the minimum and maximum m/z value in the imported dataset (rounded according to specified resolution as described in the next paragraph). In the selective mode, the list of m/z values will contain only those m/z values, which are present in the imported dataset (and rounded according to the selected resolution as described in the next paragraph). Selective mode import is recommended for datasets acquired by selected ion monitoring (SIM) for instance. Selective mode works for scan data also, using sequential mode leads to faster data import. Additionally, using selective mode for scan data may lead to an omission of some chromatograms (leading to a non−sequential m/z scale) if no signal intensities are present within a particular m/z interval (as determined by resolution).

The raw data is binned along the m/z axis during import. The binning interval is determined by the option `Resolution`. All signal intensities falling within the binning interval at a particular timepoint are averaged. For example, if resolution is set to `1`, the signal intensities at a particular timepoint falling within an interval `<mz−0.5,mz+0.5)` will be averaged (`mz` represents a whole number in this case). For resolution `0.5`, the binning interval would be `<mz−0.25,mz+0.25)` (with `mz` representing a whole number multiple of `0.5`), etc...

■ Scan binning

The functions `DAMPBinScanSequential` and `DAMPBinScanSelective` facilitate the binning along the m/z dimension as described above. The functions are used internally by functions importing datasets in different data formats.

```
DAMPBinScanSequential =
 Compile[{{rec, _Real, 2}, {dims, _Real, 1}}, Module[{tbl, cnts, tmp2 = 0., tmpcnt = 0.},
   tbl = cnts = Table[0., {Round[1 + (dims[[2]] - dims[[1]]) / dims[[3]]}];
   (tmp2 = #[[2]]; (tbl[[#]] += tmp2; cnts[[#]] += 1) &[Round[1 + (#[[1]] - dims[[1]]) / dims[[3]]]]) & /@ rec;
   cnts = If[# == 0., 1., #] & /@ cnts;
   tbl / cnts]]

DAMPBinScanSelective = Compile[{{data, _Real, 2}, {allmzs, _Real, 1}}, Module[{tbl, cnttbl, pos},
    tbl = cnttbl = Table[0., {Length[allmzs]}];
    (pos = Position[allmzs, #[[1]]][[1, 1]]; tbl[[pos]] += #[[2]]; cnttbl[[pos]] += 1.) & /@ data;
    cnttbl = If[# == 0., 1., #] & /@ cnttbl;
    tbl / cnttbl
   ]];
```

■ Agilent ChemStation MS format

The `DAMPImportMS` function imports the raw data from an Agilent ChemStation *.MS file and processes it into the *Math*DAMP internal format. Although the description of the file format is not available (?) and the format was reverse−engineered, the imported data appear to perfectly match the data in a csv file exported using the ChemStation software.
Part of the code from the `DAMPImportMS` function was moved to a compiled function `DAMPProcessDataMS[mzintensitypairs,scanpositions]` to speed up the data processing.
The signal intensities are binned to 1 m/z unit resolution upon loading.

```
DAMPProcessDataMS = Compile[{{data, _Integer, 1}, {positions, _Integer, 2}},
  Module[{procdata = {{{}}}, allmzs = {}, mzslen, findata, tbl = {}, cnttbl = {}, pos},
   procdata = ({Round[#[[1]] / 20 - .2], N[If[#[[2]] > 49152, 512 (#[[2]] - 49152),
            If[#[[2]] > 32768, 64 (#[[2]] - 32768), If[#[[2]] > 16384, 8 (#[[2]] - 16384), #[[2]]]]]} & /@ #) & /
     (Partition[Take[data, {#[[1]], #[[1]] + 2 #[[2]] - 1}], 2] & /@ positions);
   allmzs = Union[Flatten[procdata[[All, All, 1]]]];
   mzslen = Length[allmzs];
   findata = (tbl = cnttbl = Table[0, {mzslen}];
       (pos = Position[allmzs, #[[1]]][[1, 1]]; tbl[[pos]] += #[[2]]; cnttbl[[pos]] += 1) & /@ #;
       cnttbl = If[# == 0., 1., #] & /@ cnttbl; tbl / cnttbl) & /@ procdata;
   Join[{allmzs}, findata]
  ]]

DAMPImportMS[filename_] :=
 Module[{data, msnr, pos, positions, trheaders, timepoints, findata, mainhead},
  data = BinaryReadList[filename, "UnsignedInteger16", ByteOrdering → 1];
  msnr = data[[141]];
  pos = 0;
  positions =
   Table[(If[pos == 0, pos = 384, pos = pos + 2 data[[pos]] + 14]; {pos + 3, data[[pos]]}), {msnr}];
  trheaders = Transpose[Join[Take[data, {#[[1]] - 10, #[[1]] - 1}],
       Take[data, {#[[1]] + 2 #[[2]], #[[1]] + 2 #[[2]] - 1 + 4}]] & /@ positions];
  timepoints = (trheaders[[3]] 2^16 + trheaders[[4]]) / 60000;
  findata = DAMPProcessDataMS[data, positions];
  mainhead = Take[data, {21, 28}];
  {Transpose[Drop[findata, 1]], 1. findata[[1]], timepoints, {Global`SampleName → FromCharacterCode
      Select[Drop[Flatten[Transpose[#[mainhead, 2^8] & /@ {Quotient, Mod}]], 1], # ≠ 0 &]]}}
 ]
```

■ Analyst QS CSV format

`DAMPImportCSV` function imports data from a csv file generated by Analyst QS software (for Agilent TOFMS) and processes it into *Math*DAMP internal format. Since the csv file does not contain information about the sample name, the name has to be specified via the second parameter `samplename`. The option `Resolution` determines the binning of the data along the `m/z` dimension.

```
Options[DAMPImportCSV] = {Global`Resolution → 1, Verbose → True};

DAMPImportCSV[filename_, samplename_, opts___] :=
 Module[{verbs, dims, allmzs, timepoints, strm, i, recnum, findata, nrpairs, rec},
   verbs = Verbose /. {opts} /. Options[DAMPImportCSV];
   timepoints = {};
   strm = OpenRead[filename];
   Table[Skip[strm, String], {3}];
   dims =
    Join [ToExpression[Table[Read[strm, Record, RecordSeparators -> {",", "\n"}], {3}][[{2, 3}]]],
     {Global`Resolution /. {opts} /. Options[DAMPImportCSV]}];
   allmzs = N[Table[i, {i, dims[[1]], dims[[2]], dims[[3]]}]];
   Table[Skip[strm, String], {3}];
   recnum = ToExpression[Table[Read[strm, Record, RecordSeparators -> {",", "\n"}], {2}][[2]]];
   Table[Skip[strm, String], {4}];
   findata = Table[(If[Mod[i, 100] == 0 && verbs, Print[Join[{{i}}, Date[]]]];
      AppendTo[timepoints, Read[strm, Number]];
      Table[Read[strm, Record, RecordSeparators -> {",", "\n"}], {5}];
      nrpairs = ToExpression[Read[strm, Record, RecordSeparators -> {",", "\n"}]];
      rec = Partition[N[Round[1/dims[[3]]
            ToExpression["{" <> StringTake[Read[strm, String], {2, -3}] <> "}"]] dims[[3]]], 2];
      DAMPBinScanSequential[rec, dims]), {i, 1, recnum}];
   Close[strm];
   {Transpose[findata], allmzs, timepoints, {Global`SampleName -> samplename}}
  )]
```

■ mzXML format

`DAMPImportMZXML` imports the first level MS scans from an mzXML datafile. The import may be performed in either sequential or selective mode (see beginning of this section for explanation).
The function `DAMP6bitTo8bit` facilitates the conversion of base64 encoded m/z and signal intensity pairs to binary data.

```
base64enctbl = #[[1]] → #[[2]] & /@ Transpose[{Join[CharacterRange["A", "Z"],
      CharacterRange["a", "z"], CharacterRange["0", "9"], {"+", "/", "="}], Range[0, 64]}];
DAMP6bitTo8bit = Compile[{{data, _Integer, 1}}, Module[{todrop},
   todrop = Length[Cases[data, 64]];
   Round[Drop[Flatten[{#[[1]] 2^2 + (#[[2]] - Mod[#[[2]], 2^4])/2^4, Mod[#[[2]], 2^4] 2^4 + (#[[3]] - Mod[#[[3]], 2^2])/2^2,
         Mod[#[[3]], 2^2] 2^6 + #[[4]]} & /@ Partition[data, 4], -todrop]]
  ]]
```

```
Options[DAMPImportMZXML] = {Global`Resolution → 1., Global`ImportMode → Sequential};

DAMPImportMZXML[filename_, samplename_, opts___] :=
 Module[{resol, md, scans, prec, todrop, strm, rslt, rsltitm, dims},
  resol = Global`Resolution /. {opts} /. Options[DAMPImportMZXML];
  md = Global`ImportMode /. {opts} /. Options[DAMPImportMZXML];
  scans = {"retentionTime" /. #[[1]], #[[2]] /. {___,
         XMLElement["peaks", {___, "precision" → prec_, ___}, encdat_], ___} → {prec, encdat}} & /@
    Cases[Import[filename, "XML"], XMLElement["scan", atts_, elems_] → {atts, elems}, 5];
  rslt = (
      prec = ToString[#[[1]]];
      decchars = Characters[#[[2, 1]]] /. base64enctbl;
      strm = StringToStream[FromCharacterCode[DAMP6bitTo8bit[decchars]]];
      rsltitm = Partition[BinaryReadList[strm, "Real" <> prec, ByteOrdering → 1], 2];
      Close[strm];
      rsltitm[[All, 1]] = Round[rsltitm[[All, 1]] / resol] resol;
      rsltitm
     ) & /@ scans[[All, 2]];
  {Sequence @@ If[md === Sequential,
     dims = {Min[#], Max[#], resol} &[Flatten[rslt[[All, All, 1]]]];
     {Transpose[DAMPBinScanSequential[#, dims] & /@ rslt], Range[Sequence @@ dims]}
     ,
     allmzs = Union[Flatten[rslt[[All, All, 1]]]];
     {Transpose[DAMPBinScanSelective[#, allmzs] & /@ rslt], allmzs}
    ], (ToExpression[StringReplace[#, {"PT" → "", "S" → ""}]] & /@ scans[[All, 1]]) / 60,
   {Global`SampleName → samplename}}
 ]
```

■ CDF format

`DAMPImportCDF[filename,options]` function imports NetCDF data and processes it into *Math*DAMP internal format. The function was not tested thoroughly with NetCDF format files from different sources and may therefore not be generic. The option `Resolution` specifies the data binning along the `m/z` axis.

```
CDFnctypesizes = {"Byte" → 1, "Character8" → 1, "Integer16" → 2, "Real64" → 8, _ → 4};
CDFReadPadded[st_, nctp_, len_] := Module[{rslt},
  rslt = BinaryRead[st, Table[nctp, {len}], ByteOrdering → 1];
  Skip[st, "Byte", If[# == 4, 0, #] &[4 - Mod[len (nctp /. CDFnctypesizes), 4]]];
  rslt
 ]

CDFReadString[st_] := Module[{strlen},
  strlen = BinaryRead[st, "UnsignedInteger32", ByteOrdering → 1];
  StringJoin @@ CDFReadPadded[st, "Character8", strlen]
 ]

CDFReadDimArrayElem[st_] := Module[{},
  {CDFReadString[st], BinaryRead[st, "UnsignedInteger32", ByteOrdering → 1]}
 ]

CDFnctypes = {1 → "Byte", 2 → "Character8", 3 → "Integer16", 4 → "Integer32", 5 → "Real32", 6 → "Real6

CDFReadAttArrayElem[st_] := Module[{nm, nctype, nelems},
  nm = CDFReadString[st];
  nctype = BinaryRead[st, "Integer32", ByteOrdering → 1] /. CDFnctypes;
  nelems = BinaryRead[st, "UnsignedInteger32", ByteOrdering → 1];
  {nm, nctype, CDFReadPadded[st, nctype, nelems]}
 ]

CDFReadAttArray[st_] := Module[{attarraymagic, attarraynelems},
  attarraymagic = BinaryRead[st, "Integer32", ByteOrdering → 1];
  If[attarraymagic ≠ 12, If[attarraymagic == 0, Skip[st, "Byte", 4]]; {},
   attarraynelems = BinaryRead[st, "UnsignedInteger32", ByteOrdering → 1];
   Table[CDFReadAttArrayElem[st], {attarraynelems}]
  ]
 ]

CDFReadVarArrayElem[st_] := Module[{nm, nelems, dimids, vattarray},
  nm = CDFReadString[st];
  nelems = BinaryRead[st, "UnsignedInteger32", ByteOrdering → 1];
  dimids = Table[BinaryRead[st, "UnsignedInteger32", ByteOrdering → 1], {nelems}];
  vattarray = CDFReadAttArray[st];
  nctype = BinaryRead[st, "Integer32", ByteOrdering → 1] /. CDFnctypes;
  {nm, nelems, dimids, vattarray, nctype, BinaryRead[st, "UnsignedInteger32", ByteOrdering → 1],
   BinaryRead[st, "UnsignedInteger32", ByteOrdering → 1]}
 ]
```

```
Options[DAMPImportCDF] = {Global`Resolution → 1.}

DAMPImportCDF[filename_, opts___] :=
 Module[{resol, strm, magic, versionbyte, numrecs, dimarraymagic,
   dimarraynelems, dimarray, gattarray, vararraymagic, vararraynelems, vararray,
   scannumber, vars, itm, info, pairs, allmzs, mzdims, counter, newpairs, rslt},
  resol = Global`Resolution /. {opts} /. Options[DAMPImportCDF];
  strm = OpenRead[filename, BinaryFormat → True];
  magic = BinaryRead[strm, Table["Character8", {3}]];
  If[magic ≠ {"C", "D", "F"}, Print["Error: Does not seem to be a CDF file..."],
   versionbyte = BinaryRead[strm, "Byte"];
   numrecs = BinaryRead[strm, "UnsignedInteger32", ByteOrdering → 1];
   dimarraymagic = BinaryRead[strm, "Integer32", ByteOrdering → 1];
   If[dimarraymagic ≠ 10,
    If[dimarraymagic == 0, Skip[st, "Byte", 4]]; Print["Warning: Header info missing?"],
    dimarraynelems = BinaryRead[strm, "UnsignedInteger32", ByteOrdering → 1];
    dimarray = Table[CDFReadDimArrayElem[strm], {dimarraynelems}];
   ];
   gattarray = CDFReadAttArray[strm];
   vararraymagic = BinaryRead[strm, "Integer32", ByteOrdering → 1];
   If[vararraymagic ≠ 11,
    If[vararraymagic == 0, Skip[st, "Byte", 4]]; Print["Warning: Variable array missing?"],
    vararraynelems = BinaryRead[strm, "UnsignedInteger32", ByteOrdering → 1];
    vararray = Table[CDFReadVarArrayElem[strm], {vararraynelems}];
   ];
   scannumber = Select[dimarray, #[[1]] == "scan_number" &][[1, 2]];
   vars = (
       itm = #;
       info = Select[vararray, #[[1]] == itm[[1]] &][[1]];
       SetStreamPosition[strm, info[[-1]]];
       Table[BinaryRead[strm, info[[-3]], ByteOrdering → 1], {itm[[2]]}]
      ) & /@ {{"scan_acquisition_time", scannumber},
      {"point_count", scannumber}, {"mass_values", 2 numrecs}};
   pairs = Partition[vars[[3]], 2];
   mzdims = {40., Select[gattarray, #[[1]] == "global_mass_max" &][[1, 3, 1]], resol};
   allmzs = Range[Sequence @@ mzdims];
   counter = 0;
   newpairs = Take[pairs, {counter + 1, counter += #}] & /@ vars[[2]];
   rslt = {Transpose[DAMPBinScanSequential[#, mzdims] & /@ newpairs],
     allmzs, vars[[1]] / 60, {Global`SampleName →
       StringJoin @@ Drop[Select[gattarray, #[[1]] == "experiment_title" &][[1, 3]], -1]}};
  ];
  Close[strm];
  rslt
 ]
```

■ BDT format

`DAMPImportBDT[filename,samplename]` function imports data from a BDT file and processes it into *Math*DAMP internal format. The BDT is a binary datafile used internally by The Institute for Advanced Biosciences (Keio University). Separate in house software for preprocessing of the Analyst QS generated csv files stores the data in the BDT format. The organization of data in the BDT format is similar to the internal MathDAMP organization what allows a fast data import into *Mathematica*. Since the csv file does not contain the information about the sample name, this has to be specified via the second parameter `samplename`.

```
DAMPImportBDT[filename_, samplename_] := Module[{tmp},
   tmp = BinaryReadList[filename, "Real64", ByteOrdering → -1];
   {Partition[Drop[tmp, 4 + tmp[[4]]], Round[tmp[[4]]]],
    Range[Sequence @@ Take[tmp, 3]], Take[tmp, {5, 4 + tmp[[4]]}], {Global`SampleName → samplename}}
  ];
```

`DAMPExportBDT[filename,msdata]` function exports data in *Math*DAMP internal format into a BDT file format specified by `filename`. Since the m/z values in the data are not saved one by one but as a range specified by min, max and step, only data having a regular list of m/z values which can be converted to the range representation should be exported.

```
DAMPExportBDT[filename_, msdata_] := Module[{strm},
  strm = OpenWrite[filename, BinaryFormat → True];
  BinaryWrite[strm, {Min[#], Max[#], (Max[#] - Min[#])/(Length[#] - 1)} &[msdata[[2]], "Real64", ByteOrdering → -1];
  BinaryWrite[strm, Length[msdata[[3]]], "Real64", ByteOrdering → -1];
  BinaryWrite[strm, msdata[[3]], "Real64", ByteOrdering → -1];
  BinaryWrite[strm, Flatten[msdata[[1]]], "Real64", ByteOrdering → -1];
  Close[strm];
 ]
```

Core Functionality

■ Modifying the SampleName in msdata

When `msdata` is processed by some functions (baseline subtraction, smoothing, etc) it may be desirable to keep track of modifications performed on the dataset by appending a short string to the `SampleName` of the original dataset. The functions use the private function `DAMPAppendToSampleName` for this purpose. The function adds vertical bars to separate multiple modification identifiers. The vertical bars are shared between multiple modifiers unless the `ShareBars` option is set to `False`.

```
DAMPAppendToSampleName[msdata4_, str_, opts___] := Module[{sname},
  sname = Global`SampleName /. msdata4;
  (msdata4 /. {sname → sname <> # <> str <> "|"}) &[
   If[StringLength[sname] > 0 && StringTake[sname, -1] == "|" && Global`ShareBars /. {opts} /.
     {Global`ShareBars → True}, "", " |"]]
 ]
```

■ Annotation table manipulation

The `DAMPLoadAnnotationTable` function loads a compound annotation table from a csv file and selects records having relevant retention/migration times and m/z values. The organization of the records is assumed to be as follows: 1) m/z value, 2) retention/migration time, 3) short compound name/id (density plot label text), 4) full compound name (chromatogram/electropherogam annotation text), and 5) label's text position on the density plots relative to the label (1 − right, 2 − top, 3 − left, 4 − bottom, 1.5 − top right, etc). The text label can be placed anywhere around the label on the density plot by using real numbers like 1.5 (position at 45 degrees from the right position).

```
DAMPLoadAnnotationTable[filename_] :=
  Select[Import[filename], (NumericQ[#[[1]]] && NumericQ[#[[2]]] && #[[1]] > 0 && #[[2]] > 0) &];
```

■ Extracting chromatograms/electropherograms from data

`DAMPGetIntensities[msdata,mz]` returns a list of signal intensities from `msdata` corresponding to m/z `mz`.
`DAMPGetChromatogram[msdata,mz]` returns a chromatogram/electropherogram (list of `{time,signal intensity}`) from `msdata` corresponding to m/z `mz`. Since the m/z values in `msdata` are of type `Real`, the `mz` parameter is multiplied by `1.` to convert it to a `Real` type value (for cases when it is passed as an `Integer`).
The `_?(Chop[1. mz-#]==0&)` construct is the argument of the `Position` function to match values slightly different due to roundoff or truncation errors.

```
DAMPGetIntensities[msdata_, mz_] := msdata[[1, Position[msdata[[2]], _?(Chop[1. mz - #] == 0 &)][[1, 1]]]]
DAMPGetChromatogram[msdata_, mz_] := Transpose[{msdata[[3]], DAMPGetIntensities[msdata, mz]}]
```

■ Dataset cropping and particular m/z(s) dropping/selection

`DAMPCrop[msdata,options]` selects datapoints from `msdata` falling within the m/z and time ranges specified by options `mzRange` and `TimeRange`.
`DAMPSelectMZs[msdata,mzs]` reduces the `msdata` to those datapoints only, which correspond to m/z values specified in the `mzs` list.
`DAMPDropMZs[msdata,mzs]` drops datapoints from `msdata` corresponding to m/z values specified in the `mzs` list.
The `mzs` parameters are multiplied by `1.` to make sure they are of `Real` type. This is necesarry for looking up the m/z values in the datasets' lists of m/z values which are of type `Real`.

```
Options[DAMPCrop] = {Global`TimeRange → All, Global`mzRange → All}

DAMPCrop[msdata_, opts___] :=
 Module[{timerange, mzrange, selmzposs, seltpposs, newints, newmzs, newtimepoints},
  timerange = Global`TimeRange /. {opts} /. Options[DAMPCrop];
  mzrange = Global`mzRange /. {opts} /. Options[DAMPCrop];
  newints = msdata[[1]];
  newmzs = msdata[[2]];
  newtimepoints = msdata[[3]];
  If[mzrange =!= All, newmzs = Select[msdata[[2]], mzrange[[1]] ≤ # ≤ mzrange[[2]] &];
   selmzposs = Join @@ (Join @@ Position[msdata[[2]], #] & /@ newmzs); newints = newints[[selmzposs]]];
  If[timerange =!= All, newtimepoints = Select[msdata[[3]], timerange[[1]] ≤ # ≤ timerange[[2]] &];
   seltpposs = Join @@ (Join @@ Position[msdata[[3]], #] & /@ newtimepoints);
   newints = Transpose[Transpose[newints][[seltpposs]]];
  {newints, newmzs, newtimepoints, msdata[[4]]}
 ]

DAMPSelectMZs[msdata_, mzs_] := Module[{newmzs, selmzposs},
  newmzs = Intersection[msdata[[2]], 1. mzs];
  {msdata[[1, Join @@ ((mz = #; Join @@ Position[msdata[[2]], _?(Chop[1. mz - #] == 0 &)]) & /@ newmzs)]],
   newmzs, msdata[[3]], msdata[[4]]}
 ]

DAMPDropMZs[msdata_, mzs_] := Module[{newmzs, selmzposs},
  newmzs = Complement[msdata[[2]], 1. mzs];
  {msdata[[1, Join @@ ((mz = #; Join @@ Position[msdata[[2]], _?(Chop[1. mz - #] == 0 &)]) & /@ newmzs)]],
   newmzs, msdata[[3]], msdata[[4]]}
 ]
```

■ Plotting chromatograms/electropherograms

`DAMPPlotChromatogram[msdatas,mz,options]` plots overlaid chromatograms/electropherograms from `msdatas` (a list of `msdata` datasets) corresponding to m/z `mz`. `mz` may be a list of m/z values as well. The chromatograms/electropherograms may be annotated. Annotation table is passed via the `AnnotationTable` option. The `Resolution` option specifies to which resolution the data were binned along the m/z dimension. The m/z values in the annotation table are then rounded accordingly to ensure the appropriate annotation labels are selected for annotating the chromatogram/electropherogram.
The legend is drawn explicitly. For an unknown reason, this is considerably faster than *Mathematica's* `PlotLegend` function. Due to this, significant amount of time can be saved when plotting numerous chromatograms/electropherograms (using the `DAMPPlotCandidates` function for example).
Legend elements can be specified either explicitly (assigning a list in the form {{color,label},...} to the `LegendData` option) or created automatically (`LegendData->Automatic`). In the latter case, `SampleName` from each `msdata` is used as a legend label.
`MultipleListPlot` is used for plotting the chromatograms/electropherograms. Options for the `MultipleListPlot` function can be passed via the `PlotOptions` option. These options then override the options determined by options passed to the `DAMPPlotChromatogram` function. This approach of using the `PlotOptions` option to pass options to the function used for plotting the data is used throughout the *Math*DAMP package. It can also be found in functions like `DAMPDensityPlot` and `DAMPPlotPeakLayout`.

```
Options[DAMPPlotChromatogram] =
 {Global`Resolution → 1, Global`AnnotationTable → {}, Global`LegendData → Automatic,
  Global`PlotOptions → {Frame → True, PlotRange → All, Axes → None, PlotJoined → True,
    ImageSize → 670, SymbolShape → None, AspectRatio → .23, TextStyle :→ Global`DAMPTextStyle}}
```

```
DAMPPlotChromatogram[msdatas_ /; Depth[N[msdatas]] == 5, mz_, opts___] :=
 Module[{annottab, resol, styles, plotoptions, legdat, snames, textstl, dispfunc, ar, is,
   timerange, chroms, mzitm, maxint, minint, bgcoef, plrange, dtplot, annotgr, legstep, leggr},
  annottab = Global`AnnotationTable /. {opts} /. Options[DAMPPlotChromatogram];
  resol = Global`Resolution /. {opts} /. Options[DAMPPlotChromatogram];
  annottab[[All, 1]] = Round[annottab[[All, 1]] / resol] resol;
  styles = DAMPGenColors[Length[msdatas] Length[Flatten[{mz}]]];
  plotoptions = Join[Global`PlotOptions /. {opts} /. Options[DAMPPlotChromatogram],
    Global`PlotOptions /. Options[DAMPPlotChromatogram],
    {PlotStyle -> styles, SymbolStyle -> styles}];
  legdat = Global`LegendData /. {opts} /. Options[DAMPPlotChromatogram];
  If[legdat == Automatic,
   legdat = Transpose[{PlotStyle /. plotoptions, If[Length[mz] > 1, snames = #; Join @@
          ((mzitm = #; (# <> " (" <> ToString[mzitm] <> ")") & /@ snames) & /@ mz), #] &[
       (Global`SampleName /. #[[4]]) & /@ msdatas]}]];
  textstl = TextStyle /. plotoptions /. {TextStyle :> Global`DAMPTextStyle};
  dispfunc = DisplayFunction /. plotoptions /. {DisplayFunction → $DisplayFunction};
  ar = AspectRatio /. plotoptions;
  is = ImageSize /. plotoptions;
  timerange = PlotRange /. plotoptions;
  If[Length[timerange] > 0 && VectorQ[timerange[[1]], NumberQ],
   timerange = timerange[[1]], timerange = {Min[#], Max[#]} &[msdatas[[All, 3]]]];
  chroms = Select[#, timerange[[1]] <= #[[1]] <= timerange[[2]] &] & /@
    Join @@ ((mzitm = #; DAMPGetChromatogram[#, 1. mzitm] & /@ msdatas) & /@ Flatten[{mz}]);
  maxint = Max[chroms[[All, All, 2]]];
  minint = Min[chroms[[All, All, 2]]];
  bgcoef = .2 670 / (ar is);
  plrange =
   {All, {minint - .05 bgcoef (maxint - minint), minint + (1 + .15 bgcoef) (maxint - minint)}};
  If[(# =!= All && #[[2]] =!= All) &[PlotRange /. plotoptions], plrange = PlotRange /. plotoptions];
  dtplot = MultipleListPlot[chroms, PlotRange → plrange, AspectRatio → 1.2 ar / (0.113012975608684 53` + ar)
    Sequence @@ plotoptions, PlotLabel -> "m/z : " <> ToString[mz],
    FrameLabel → {"Time (min)", None}, DisplayFunction → Identity];
  annotgr = If[annottab != {}, Graphics[{AbsoluteThickness[.25],
       Line[{{#[[1]], minint + (1 + .02 bgcoef) (maxint - minint)}, {#[[1]], maxint}}]}], Text[StyleForm[
         #[[2]], Sequence @@ textstl], {#[[1]], minint + (1 + .05 bgcoef) (maxint - minint)}]] & /@
    Select[annottab, ((Times @@ Chop[Flatten[{mz}] - #[[1]]] == 0) &&
        timerange[[1]] <= #[[2]] <= timerange[[2]]) &][[All, {2, 4}]], {}];
  legstep = 1 / (Length[legdat]);
  leggr = Show[Graphics[{GrayLevel[1], Line[{{0, 0}, {1, 1}}]}], (MapIndexed[Graphics[
       {{#1[[1]], AbsoluteThickness[.25], Line[{{0.02, #2[[1]] legstep}, {.98, #2[[1]] legstep}}]},
        Text[StyleForm[#1[[2]], Sequence @@ textstl], {0.01, #2[[1]] legstep + .01 bgcoef}, {-1, 1}]}] &,
     Reverse[legdat]]), DisplayFunction → Identity, AspectRatio → ar 9];
  Show[Graphics[Rectangle[{0, 0}, {1, 1}, Show[dtplot, annotgr]]], Graphics[
    Rectangle[{1, 0}, {1.1, 1}, leggr]], AspectRatio → ar, ImageSize → is, DisplayFunction → dispfu
 ]

DAMPPlotChromatogram[msdatas_ /; Depth[N[msdatas]] == 4, mz_, opts___] :=
 DAMPPlotChromatogram[{msdatas}, mz, opts]
```

■ Plotting Density Plots

The functions in this section facilitate the visualization of `msdata` on density plots. The axes represent the retention/migration time and m/z values. The peaks appear as colored spots. The color is assigned to a signal intensity value from a color gradient. The color is not calculated for each intensity during the generation of the plot but taken from a pre−generated palette instead. In other words, the pure function passed via the `ColorFunction` option of the `ListDensityPlot` function does not compute the color value but picks it from a pre−generated palette. This speeds up the process considerably.
`DAMPGradient` returns a function which calculates the color parameters for the `Hue` directive corresponding to a relative position within the color gradient (the returned function takes this position as the only parameter `<0;1>`) . The gradient consists of three subgradients: `backgroundgraylevel` (at relative position 0) to `color1` (at `pos1`), `color1` (at `pos1`) to `color2` (at `pos2`), and `color2` (at `pos2`) to darkened `color2` (at relative position 1). The `pos1` and `pos2` parameters represent the relative positions of the endpoints of the middle gradient `(0;1)`; `color1` and `color2` parameters represent the hue specification `<0;1>` of the endpoints of the middle gradient; and the `backgroundgraylevel` parameter specifies the color (graylevel) of the background area `<0;1>` (black to white scale). The returned function is probably compiled for historical reasons only. Faster possibilities for the plot generation were investigated by increasing the speed of the function which was used as a color function in the `ListDensityPlot` function. The speed was not satisfactory, so a switch to pre−generating the color palette was made so that now the colorfunction just picks the appropriate color from the palette and does not calculate the color every time. The `DAMPGradient` still returns a compiled function although it does not make a significant difference in the current context.
`DAMPGradientPalette` uses the `DAMPGradient` function to generate a palette of two adjacent gradients (intended for positive and negative range). The palette will be used by the colorfunction of the `ListDensityPlot` to assign appropriate colors to signal intensi−ties and to draw a legend. The option `ColorPositions` determines the relative position of gradient endpoints and is used for both positive and negative ranges (default: {.075,.4}). `PositiveColors` and `NegativeColors` hold the hue color specifications for the endpoints of the middle gradients in both positive and negative ranges (defaults: `{1/6,0}` − yellow to red; `{1/2,2/3}` − cyan to blue). `BackgroundGrayLevel` specifies the graylevel of the background area (default: `0.7` − 30% gray). `PaletteSize` specifies the number of palette entries (subdivisions) for both positive and negative range color gradients, entry for value `0` is shared between the two ranges.

```
DAMPGradient[pos1_, pos2_, color1_, color2_, backgroundgraylevel_] :=
 Compile[{col}, {If[col < pos1, color1, If[col < pos2, color1 + (col - pos1) (color2 - color1)/(pos2 - pos1), color2
   If[col < pos1, col/pos1, 1], If[col < pos1,
    backgroundgraylevel + (1 - backgroundgraylevel) col/pos1, If[col < pos2, 1, 0.2 + 0.8 (1 - col)/(1 - pos2)]]}]

Options[DAMPGradientPalette] =
 {Global`ColorPositions → {0.075, 0.4}, Global`PositiveColors → {1/6, 0},
  Global`NegativeColors → {1/2, 2/3}, Global`BackgroundGrayLevel → .7, Global`PaletteSize → 200}

DAMPGradientPalette[opts___] := Module[{bgl, palsize, colposs, negfunc, posfunc},
  bgl = Global`BackgroundGrayLevel /. {opts} /. Options[DAMPGradientPalette];
  palsize = Global`PaletteSize /. {opts} /. Options[DAMPGradientPalette];
  colposs = Global`ColorPositions /. {opts} /. Options[DAMPGradientPalette];
  posfunc = DAMPGradient[Sequence @@ colposs,
    Sequence @@ (Global`PositiveColors /. {opts} /. Options[DAMPGradientPalette]), bgl];
  negfunc = DAMPGradient[Sequence @@ colposs,
    Sequence @@ (Global`NegativeColors /. {opts} /. Options[DAMPGradientPalette]), bgl];
  Hue[Sequence @@ #] & /@ Join[Reverse[negfunc[#] & /@ Drop[#, 1]], posfunc[#] & /@ #] &[
   Range[0, 1, 1/palsize]]
 ]
```

`DAMPDrawAnnotation[atdata,options]` is used by the `DAMPDensityPlot` function to annotate the density plots of datasets and allow easier identification of peaks. The annotation table is passed via the `atdata` parameter. For the annotation table format, please refer to the description preceding the `DAMPLoadAnnotationTable` in the *Annotation Table Manipulation* subsection of the *Core Functionality* section. The `TimeRange` and `mzs` options determine the selection of annotation table items and their proper positioning on the density plots. The `Resolution` option specifies the m/z dimension coverage of each electropherogram on the density plot. To ensure the proper selection and display of the annotation labels, the m/z values in the annotation table are rounded to the resolution. Usually, the data are binned to `1` m/z resolution, so the default value for the `Resolution` option is set to 1. The option `ScaleCoefficients` ensures the correct positioning of the labels and their size reproducibility. The `ScaleCoefficients` are used for the scale conversion to the scale of the `ListDensityPlot` where the time scale is numbered sequentially for every datapoint. The

positioning along the m/z dimension is done according to the respective m/z value's position within the `mzs` list, so the second element in the `ScaleCoefficients` is not relevant at the moment and is set to `1` by default. The scale conversion value for the time dimension is calculated in the `DAMPDensityPlot` function according to the `msdata`'s timepoints. The option `TextStyle` determines the appearance of the annotation label's text, it is set to `DAMPTextStyle` by default, the `FontSize` is overriden to a smaller size (4) to fit better into the crowded density plots. The `LabelStyle` and `LabelSize` options determine the label mark's appearance and size. The size units are relative, {1,1} by default. The `LabelShape` option specifies a pure function for drawing the annotation label symbol. Two parameters are passed to the function, the first is the position on the `ListDensityPlot`'s coordinate system `{retention/migration time,m/z value}` and the second the size of the size `{xsize,ysize}` in the same units. The scale conversion is set so, that the default relative size of the annotation label (`LabelSize->{1,1}`) draws an ellipse which appears to properly mark the peaks on the density plot. The `{xsize,ysize}` correspond to the ellipse's radii in this case.

```
Options[DAMPDrawAnnotation] =
  {Global`TimeRange → {0, 50}, Global`MZs → Range[50., 1000., 1.], Global`Resolution → 1,
   Global`ScaleCoefficients → {62.2488, 1}, TextStyle :→ Join[{FontSize → 4}, Global`DAMPTextStyle]
   Global`LabelStyle → {AbsoluteThickness[.25]},
   Global`LabelShape → (Circle[#1, #2] &), Global`LabelSize → {1, 1}};
DAMPDrawAnnotation[atdata_, opts___] := Module[{at, timerange, mzrange, resol,
   scalecoefs, labelstyle, labelshape, labelsize, annottoshow, tmpmz, tmpgr, position},
  timerange = Global`TimeRange /. {opts} /. Options[DAMPDrawAnnotation];
  mzlist = Global`MZs /. {opts} /. Options[DAMPDrawAnnotation];
  resol = Global`Resolution /. {opts} /. Options[DAMPDrawAnnotation];
  textstl = Join[TextStyle /. {opts} /. Options[DAMPDrawAnnotation],
    TextStyle /. Options[DAMPDrawAnnotation]];
  at = atdata;
  at[[All, 1]] = Round[at[[All, 1]]/resol] resol 1.;
  annottoshow = Select[at, timerange[[1]] ≤ #[[2]] ≤ timerange[[2]] &];
  annottoshow = Join @@ MapIndexed[(tmpmz = {#1, #2[[1]]};
      Join[{tmpmz[[2]]}, Drop[#, 1]] & /@ Select[annottoshow, #[[1]] == tmpmz[[1]] &]) &, mzlist];
  scalecoefs = Global`ScaleCoefficients /. {opts} /. Options[DAMPDrawAnnotation];
  labelstyle = Global`LabelStyle /. {opts} /. Options[DAMPDrawAnnotation];
  labelshape = Global`LabelShape /. {opts} /. Options[DAMPDrawAnnotation];
  labelsize = Global`LabelSize /. {opts} /. Options[DAMPDrawAnnotation];
  tmpgr = (position = {#[[2]] - timerange[[1]], #[[1]] - .5} scalecoefs;
      {{Text[StyleForm[#[[3]], Sequence @@ textstl],
         position + {.2, .8} labelsize scalecoefs {Cos[#], Sin[#]} &[π/2 (#[[5]] - 1)],
         -{Cos[#], Sin[#]} &[π/2 (#[[5]] - 1)]]}, Join[labelstyle,
        {labelshape[position, {.2, .8} labelsize scalecoefs]}]}) & /@ annottoshow;
  tmpgr]
```

`DAMPDensityPlot` plots `msdata` using the `ListDensityPlot` function. The axes are intended to represent the migration/retention time and m/z. Signal intensities are encoded by colors. The color palette is generated using the `DAMPGradientPalette` function. The `ColorFunction` of the `ListDensity` plot then picks the closest palette entry for a given signal intensity. This implementation improves the speed of the plot generation (compared to calculating the color for every signal intensity by the color function). The plot legend is drawn explicitly and placed below the plot. Options for the `ListDensityPlot` may be passed via the option `PlotOptions`. The option `MaxScale` determines the size of the signal intensity scale. It is `Automatic` by default meaning the signal intensity scale is determined by the dataset's biggest positive or negative signal intensity value. The option `FrameTickFreq` determines the frequency of frameticks on the time and m/z axes. The value for the time axis specifies the frequency of tickmarks in minutes, for the m/z axis the frequency in the number actual m/z values from the dataset's m/z list. Default is {1,1} meaning a frametick will be placed for every minute on the time axis and for every single m/z value from the `msdata`'s m/z list. `FrameTickOffsets` option specifies from which position to start to place the tickmarks on both axes. The `mzTickShift` option specifies the shift of every tickmark along the m/z axis. `ListDensityPlot` places the tickmarks at the beginning of the colored rectangle not in its middle. The −.5 shift places the tickmark in the middle of the colored rectangle. It may seem redundant with the `FrameTickOffsets`, but the functions of the two are different. `mzTickShift` can be used to appropriately place the tickmarks on the parallel plots where it is desirable to place the tickmark in the middle of the multiple parallel lanes corresponding to the same m/z. For this reason, the shifting on the m/z axis is not fixed to −.5 as is the case for the tickmarks on the time axis. The options `mzGridLineFreq` and `mzGridLineStyle` specify the distribution and appearance of horizontal gridlines. The horizontal gridlines are intended for better orientation on the plots in terms of estimating the m/z values of peaks/spots. Solid gridlines alternate with dashed ones by default. More complex patterns are possible by entering multiple style specifications in the `mzGridLineStyle` option. `AnnotationTables` option specifies the list of annotation tables and `AnnotationOptions` can be used to pass lists of options to the `DAMPDrawAnnotation` function and control the appearance of annotation labels for each annotation table. The signal intensities can be displayed using a logarithmic scale by setting the `LogScale` option to `True`.

`mzFrameTicks` option was added to facilitate easier display of asymmetric tickmarks on the m/z axis. This is used by the

`DAMPParallelPlot` function if datasets of non−perfect overlap in terms of their m/z values are displayed. `mzTickShift` may be becoming obsolete due to this.

```
Options[DAMPDensityPlot] =
 {Global`MaxScale → Automatic, Global`FrameTickFreqs → {1, 1}, Global`FrameTickOffsets → {0, 0},
  Global`mzTickShift → -.5, Global`mzFrameTicks → Automatic, Global`Palette → DAMPGradientPalette
  Global`mzGridLineFreq → 5, Global`mzGridLineStyle → {{AbsoluteThickness[.25], GrayLevel[.5]},
    {AbsoluteThickness[.25], GrayLevel[.5], Dashing[{0.01, 0.01}]}},
  Global`AnnotationTables → None, Global`PlotOptions → {Mesh → False, ImageSize → 930,
    AspectRatio → .35, FrameLabel → {"Time (min)", "m/z"}, TextStyle :→ Global`DAMPTextStyle},
  Global`AnnotationOptions → Automatic, Global`LogScale → False}

DAMPDensityPlot[msdata_, opts___] :=
 Module[{tmax, gridfreq, palette, pallen, frametickfreq, frametickoffset, mzticks, frameticks,
   tmp, plotoptions, annopts, logscale, ints, ldp, gp, fp, ar, is, atbl, legtmax, legend, tmppp},
  legtmax = Global`MaxScale /. {opts} /. Options[DAMPDensityPlot];
  If[legtmax === Automatic, legtmax = Max[Abs[Flatten[msdata[[1]]]]]];
  gridfreq = Global`mzGridLineFreq /. {opts} /. Options[DAMPDensityPlot];
  palette = Global`Palette /. {opts} /. Options[DAMPDensityPlot];
  pallen = Length[palette];
  frametickfreq = Global`FrameTickFreqs /. {opts} /. Options[DAMPDensityPlot];
  frametickoffset = Global`FrameTickOffsets /. {opts} /. Options[DAMPDensityPlot];
  mzticks = Global`mzFrameTicks /. {opts} /. Options[DAMPDensityPlot];
  frameticks = {tmp = Length[msdata[[3]]] - 1 / (msdata[[3, -1]] - msdata[[3, 1]]);
     Table[{i tmp - msdata[[3, 1]] tmp - .5, i},
     {i, Floor[msdata[[3, 1]] + frametickoffset[[1]], Ceiling[msdata[[3, -1]], frametickfreq[[1]]}],
    #[[Table[i, {i, Max[1, 1 + frametickoffset[[2]]], Length[#], frametickfreq[[2]]}]]] &[
     If[mzticks === Automatic, MapIndexed[{#2[[1]] + Global`mzTickShift /. {opts} /.
          Options[DAMPDensityPlot], #1} &, msdata[[2]]], mzticks]]};
  plotoptions = Join[Global`PlotOptions /. {opts} /. Options[DAMPDensityPlot],
    Global`PlotOptions /. Options[DAMPDensityPlot]];
  ar = AspectRatio /. plotoptions;
  is = ImageSize /. plotoptions;
  atbl = Global`AnnotationTables /. {opts} /. Options[DAMPDensityPlot];
  annopts = Global`AnnotationOptions /. {opts} /. Options[DAMPDensityPlot];
  logscale = Global`LogScale /. {opts} /. Options[DAMPDensityPlot];
  If[annopts === Automatic, annopts = Table[{}, {Length[atbl]}]];
  If[Length[annopts] ≠ Length[atbl],
   Print[StyleForm["Error: Number of annotation tables (AnnotationTables
       option) does not correspond to the number of their rendering
       styles (AnnotationStyles option).", FontColor → Hue[0]]]; Return[{}]];
  If[logscale == True, ints = (Log[10, Abs[msdata[[1]]] /. {-∞ → 0., Indeterminate → 0.})
     Sign[msdata[[1]]]; tmax = Log[10, legtmax], ints = msdata[[1]]; tmax = legtmax];
  ldp = ListDensityPlot[ints, AspectRatio → 1.1 ar - .083, Sequence @@ plotoptions, PlotLabel →
     (Global`SampleName /. msdata[[4]]), ColorFunction → (palette[[Round[(pallen - 2) #] + 1]] &),
    PlotRange → {All, All, {-tmax, tmax}}, FrameTicks → frameticks, DisplayFunction → Identity];
  If[gridfreq === None, fp = ldp, gp = Plot[Evaluate[
      If[VectorQ[gridfreq], gridfreq, Join @@ Table[{i}, {i, 0, Length[msdata[[2]]], gridfreq}]]],
     {x, 0, Length[msdata[[3]]]}, DisplayFunction → Identity,
     PlotStyle → (Global`mzGridLineStyle /. {opts} /. Options[DAMPDensityPlot])];
   fp = Show[ldp, gp, DisplayFunction → Identity]];
  If[atbl =!= None,
   fp = Show[fp, Graphics[DAMPDrawAnnotation[#[[1]], Sequence @@ #[[2]], Global`MZs → msdata[[2]],
         Global`TimeRange → msdata[[3, {1, -1}]], Global`ScaleCoefficients →
          {Length[#] - 1 / (#[[-1]] - #[[1]]) &[msdata[[3]]], 1}]] & /@ Transpose[{atbl, annopts}]]];
  legend = Graphics[Join[{MapIndexed[{#1, Rectangle[Sequence @@
             Transpose[{.7 / pallen {#2[[1]] - 1, #2[[1]]} + .15, {0, 1}}]]} &, palette],
     {Text[ToString[legtmax], {.86, .5}, {-1, 0}], Text[ToString[-legtmax], {.14, .5}, {1, 0}],
      {PointSize[0], Point[{1, 1}]}}}]];
  tmppp = Show[legend, DisplayFunction → Identity, AspectRatio → .015, TextStyle →
     (Join[{FontSize → (FontSize /. # /. {FontSize → 7}) - 1}, #] &[TextStyle /. plotoptions])];
  Show[Graphics[{Rectangle[{0, 0.015 / ar}, {1, 1}, fp], Rectangle[{0, 0}, {1, .015 / ar}, tmppp]}],
   AspectRatio → ar, ImageSize → is, DisplayFunction → (If[Length[#] > 0, #[[1]], #] &[
     (DisplayFunction /. {plotoptions} /. {DisplayFunction → $DisplayFunction})])]
 ]
```

`DAMPParallelPlot` plots multiple `msdatas` in a parallel format. The datasets are interlaced into one dataset so that chromatograms/electropherograms corresponding to the same m/z value appear next to each other on the density plot. The

chromatograms/electropherograms do not have to correspond to an identical set of m/z values in every dataset. The corresponding chromatograms/electropherograms are paired automatically and horizontal gridlines separate lanes of parallel chromatograms/electropherograms corresponding to the same m/z value. Some lanes may be narrower/wider depending on the presence of data corresponding to the particular m/z value in all datasets. Although aligned datasets are most appropriate to plot using `DAMPParallelPlot`, unaligned datasets may be plotted as well. In this case, the time axis is labeled according to the first dataset in `msdatas` and the chromatograms/electropherograms from the remaining datasets are simply concatenated or extended with 0 signal intensities to accommodate them to the dimensions of the first dataset. Please note that the time axis may be misleading in this case. The possibility to plot unaligned datasets is granted in spite of this to allow quick parallel visualizations of initial data prior to processing.

```
DAMPParallelPlot[msdatas_, opts___] :=
 Module[{mzs, tplen, mzitm, msdataitm, mesodata, pardata, mesosum, mzticks, mzgridlines},
  mzs = Union[Join @@ msdatas[[All, 2]]];
  tplen = Length[msdatas[[1, 3]]];
  mesodata =
   (mzitm = #; {mzitm, Length[#], If[Length[#] < tplen, Join[#, Table[0, {tplen - Length[#]}]],
             Take[#, tplen]]} & /@ #) &[Select[
        Join @@ (msdataitm = #; msdataitm[[1, #]] & /@ (Join @@ Position[msdataitm[[2]], mzitm])) & /@
         msdatas, Length[#] > 0 &]]) & /@ mzs;
  pardata = {Join @@ mesodata[[All, 3]], Join @@ (Table[#[[1]], {#[[2]]}] & /@ mesodata[[All, {1, 2}]]),
    msdatas[[1, 3]], {Global`SampleName ->
      "Parallel Plot " <> (" : " <> (Global`SampleName /. #)) & /@ msdatas[[All, 4]])}};
  mesosum = 0;
  mzgridlines = {0};
  mzticks = ((mesosum += #[[2]]; AppendTo[mzgridlines, mesosum]; {mesosum - (#[[2]] / 2), #[[1]]}) & /@
     mesodata[[All, {1, 2}]]);
  DAMPDensityPlot[pardata, opts, Global`mzFrameTicks → mzticks,
   Global`mzGridLineFreq → mzgridlines]
 ]
```

■ Applying a function to a single dataset

`DAMPApplyFunctionToSingle` applies a pure function `function` to the signal intensities of `msdata` (`msdata[[1]]`). This generic function could be wrapped by functions like `DAMPSubtractBaselines`, `DAMPThreshold`, `DAMPRemoveSpikes`, `DAMPSmooth`, etc. However, it is not done so for simplicity (at the expense of having redundant code though).

```
Options[DAMPApplyFunctionToSingle] = {Global`SampleNameSuffix → ""}

DAMPApplyFunctionToSingle[msdata_, func_, opts___] := Module[{tmpdat},
  tmpdat = func[msdata];
  tmpdat[[4]] = DAMPAppendToSampleName[tmpdat[[4]],
    Global`SampleNameSuffix /. {opts} /. Options[DAMPApplyFunctionToSingle]];
  tmpdat
 ]
```

■ Baseline subtraction / noise removal / smoothing

Currently, only robust nonlinear regression to a polynomial is implemented as a baseline fitting algorithm. Any custom baseline fitting function may be passed as an option to the `DAMPSubtractBaselines` function which subtracts baselines from all electropherograms in the passed `msdata`.

The implementation of `DAMPFitBaselineRobRegr` was inspired by Ruckstuhl et al [1]. The weights are calculated in the same way, although the regression is global. Binning chromatograms speeds up the baseline fitting. For this purpose, two functions were implemented, a versatile `DAMPBinChromatogram` for which the binning function can be passed as an options (default: `Mean`) and which bins precisely to the time intervals specified (in minutes), and a fast compiled version `DAMPBinChromatogramFast` with limitations in the two mentioned factors. The function for binning the data for baseline estimation can be passed to the `DAMPRobustNonlinearFit` function as an option. It is set to `(DAMPBinChromatogramFast[#,0.25]&)` by default.

```
Options[DAMPBinChromatogram] = {Global`BinningFunction → Mean};
DAMPBinChromatogram[chrom_, timeint_, opts___] := Module[{binfunc, predat},
  binfunc = Global`BinningFunction /. {opts} /. Options[DAMPBinChromatogram];
  predat = Table[ Select[chrom, a ≤ #[[1]] < a + timeint &], {a, 0, chrom[[-1, 1]], timeint}];
  binfunc[#] & /@ Transpose[#] & /@ predat
 ]

DAMPBinChromatogramFast = Compile[{{chrom, _Real, 2}, {timeint, _Real}}, Module[{},
   (Plus @@ # / Length[#] & /@ Transpose[#]) & /@ Partition[chrom, Round[(Length[chrom] - 1) / (chrom[[-1, 1]] - chrom[[1, 1]]) timeint]]
  ]]

Options[DAMPRobustPolynomialFit] =
  {Global`BinningFunction → (DAMPBinChromatogramFast[#, .25] &), Global`PolynomialDegree → 1,
   Global`ConvergenceCriteria → .001, MaxIterations → 10, Global`b → 4.05};
DAMPRobustPolynomialFit[chrom_, opts___] := Module[{binfunc, poldeg, cc, maxit,
   bval, ssqprev, ssqlast, counter, rgr, pol, chromtofit, ws, pars, gis, σ},
  binfunc = Global`BinningFunction /. {opts} /. Options[DAMPRobustPolynomialFit];
  poldeg = Global`PolynomialDegree /. {opts} /. Options[DAMPRobustPolynomialFit];
  cc = Global`ConvergenceCriteria /. {opts} /. Options[DAMPRobustPolynomialFit];
  maxit = MaxIterations /. {opts} /. Options[DAMPRobustPolynomialFit];
  bval = Global`b /. {opts} /. Options[DAMPRobustPolynomialFit];
  ssqlast = ssqprev = ∞;
  counter = 0;
  pars = ToExpression[Take[CharacterRange["a", "z"], poldeg + 1]];
  pol = Plus @@ (pars Table[x^i, {i, 0, poldeg}]);
  chromtofit = If[binfunc === None, chrom, binfunc[chrom]];
  ws = Table[1, {Length[chromtofit]}];
  While[(ssqlast == ∞ || ssqprev == ∞ || Abs[ssqlast - ssqprev] > ssqprev cc) && counter < maxit,
   counter++;
   ssqprev = ssqlast;
   rgr = NonlinearRegress[chromtofit, pol,
     x, pars, Weights → ws , RegressionReport → {BestFit, ANOVATable}];
   ssqlast = (ANOVATable /. rgr)[[1, 1, 2]];
   gis = (BestFit /. rgr) /. x → chromtofit[[All, 1]];
   σ = (Median[Abs[chromtofit[[All, 2]] - gis]] / .6745) /. {0 → 1, 0. → 1};
   ws = Max[1 - (# / bval)^2, 0]^2 & /@ (chromtofit[[All, 2]] - gis) / σ;
  ];
  Transpose[{chrom[[All, 1]], (BestFit /. rgr) /. x → chrom[[All, 1]]}]
 ]

Options[DAMPSubtractBaselines] =
 {Global`BaselineFittingFunction → DAMPRobustPolynomialFit, Global`SampleNameSuffix → "bs"}

DAMPSubtractBaselines[msdata_, opts___] := Module[{bfunc},
  bfunc = Global`BaselineFittingFunction /. {opts} /. Options[DAMPSubtractBaselines];
  {(# - bfunc[Transpose[{msdata[[3]], #}]][[All, 2]]) & /@ msdata[[1]],
   msdata[[2]], msdata[[3]], DAMPAppendToSampleName[msdata[[4]],
    Global`SampleNameSuffix /. {opts} /. Options[DAMPSubtractBaselines]]}
 ]
```

Noise removal is done by calculating the standard deviation of signal intensities in the selected area of each chromatogram/electropherogram (where no signals are expected) and leveling to `0` of all signal intensities within a specified multiple of the calculated standard deviation. The chromatograms/electropherograms should be baseline subtracted before applying this noise removal. Some noise related signals may exceed the threshold and stay in the dataset. If no negative peaks are expected (or they are artifacts without significance), all negative signal intensities may be leveled to `0` by setting the `LevelNegativeSignals` option to `True` (default). Also, applying the `DAMPRemoveSpikes` function on the noise reduced data may remove additional noise−related signals which exceeded the threshold during noise removal. `DAMPThreshold` may be applied to `msdata` to level to `0` all signal intensities within ±`threshold`.

```
Options[DAMPRemoveNoise] = {Global`TimeRange → {1, 3}, Global`SDThreshold → 5,
  Global`LevelNegativeSignals → True, Global`SampleNameSuffix → "nr"}

DAMPRemoveNoise[msdata_, opts___] :=
 Module[{timerange, sdthreshold, levelnegs, posrange, newsigs},
  timerange = Global`TimeRange /. {opts} /. Options[DAMPRemoveNoise];
  sdthreshold = Global`SDThreshold /. {opts} /. Options[DAMPRemoveNoise];
  levelnegs = Global`LevelNegativeSignals /. {opts} /. Options[DAMPRemoveNoise];
  posrange = {Min[#], Max[#]} &[
    Flatten[Position[msdata[[3]], #] & /@ Select[msdata[[3]], timerange[[1]] ≤ # ≤ timerange[[2]] &]]];
  newsigs = Chop[1. #, sdthreshold StandardDeviation[Take[#, posrange]]] & /@ msdata[[1]];
  If[levelnegs == True, newsigs = newsigs (Sign[newsigs] + 1)/2];
  {newsigs, msdata[[2]], msdata[[3]], DAMPAppendToSampleName[
    msdata[[4]], Global`SampleNameSuffix /. {opts} /. Options[DAMPRemoveNoise]]}
 ]

DAMPRemoveSpikesFromInts = Compile[{{ints, _Real, 1}}, Module[{},
   Join[{ints[[1]]}, Table[If[ints[[i]] ≠ 0 && ints[[i - 1]] == 0 && ints[[i + 1]] == 0, 0, ints[[i]]],
     {i, 2, Length[ints] - 1}], {ints[[-1]]}]
  ]]

Options[DAMPRemoveSpikes] = {Global`SampleNameSuffix → "sr"}

DAMPRemoveSpikes[msdata_, opts___] := Module[{},
  {DAMPRemoveSpikesFromInts[#] & /@ msdata[[1]], msdata[[2]], msdata[[3]], DAMPAppendToSampleName[
    msdata[[4]], Global`SampleNameSuffix /. {opts} /. Options[DAMPRemoveSpikes]]}
 ]

Options[DAMPThreshold] = {Global`SampleNameSuffix → "t"}

DAMPThreshold[msdata_, threshold_, opts___] := Module[{},
  {Chop[1. msdata[[1]], threshold], msdata[[2]], msdata[[3]], DAMPAppendToSampleName[
    msdata[[4]], Global`SampleNameSuffix /. {opts} /. Options[DAMPThreshold]]}
 ]
```

Chromatogram/electropherogram smoothing is done via the `DAMPSmooth` function which takes the smoothing function as one of its parameters. It is set to `DAMPMovingAverageFast[#,9]&` by default. `DAMPMovingAverageFast` smoothes the list of signal intensities by averaging every signal intensity with an equal number of preceding and succeeding signal intensities. The signal intensities at the beginning and at the end of the list are averaged with the available intensities within the window only. This ensures the smoothed list to have an equal number of elements as the original list.

```
DAMPMovingAverageFast =
 Compile[{{ints, _Real, 1}, {winsize, _Integer}}, Module[{vicdist, intslen},
   vicdist = Round[(winsize - 1)/2];
   intslen = Length[ints];
   Table[(Plus @@ #)/Length[#] &[
     Take[ints, {Max[1, i - vicdist], Min[i + vicdist, intslen]}]], {i, 1, intslen}]
  ]]

Options[DAMPSmooth] =
 {Global`SmoothingFunction → (DAMPMovingAverageFast[#, 9] &), Global`SampleNameSuffix → "s"}

DAMPSmooth[msdata_, opts___] := Module[{smoothingfunc},
  smoothingfunc = Global`SmoothingFunction /. {opts} /. Options[DAMPSmooth];
  {smoothingfunc[#] & /@ msdata[[1]], msdata[[2]], msdata[[3]],
   DAMPAppendToSampleName[msdata[[4]], Global`SampleNameSuffix /. {opts} /. Options[DAMPSmooth]]}
 ]
```

■ Peak picking

Peak picking is performed to find a set of peaks in every dataset and use these to align the datasets (find the parameters of the time shift function, subsection below). Thanks to the robustness of the alignment approach, even few corresponding peaks between the datasets are sufficient to find the best parameters for the alignment. For this reason, the demands on the quality of peak−picking algorithm are low. The peak lists may contain many peaks without corresponding counterparts across the lists or erroneous peak picks as well.

The implementation of the peak picking algorithm was inspired by Wallace's et al. [2] use of the Douglas−Peucker algorithm. In their approach, the chromatogram/electropherogram is segmented into portions defined by strategic points. First, the endpoints of the chromatogram/electropherogram are the only two strategic points. Additional strategic points are found by recursively finding a point with the biggest orthogonal distance from the line connecting two neighboring strategic points (and the distance being above a thresh−old). The difference in the current implementation is that vertical (not orthogonal) distance of a datapoint from a line connecting two strategic points is used as a criteria for new strategic points. The advantage of this approach is that it is not necessary to normalize either the signal intensity or time dimension. The threshold may be therefore specified in the units of the signal intensity scale. Using the vertical distance, however, may lead to too many datapoints fulfilling the criteria of a strategic point (having the distance from the line connecting two adjacent strategic points above the threshold). This is partially overcome by setting the minimal distance between the strategic points. In other words, a new strategic point cannot be selected in the specified vicinity of an existing strategic point. A good setting for this parameter is the 1/4 of expected peak width at its base, so that the restricted areas around the peak−top strategic point and peak base strategic points partially or almost overlap. The peaks are then selected from the list of strategic points according to criteria, that peak−top strategic point must have a higher signal intensity than the two neighboring strategic points. This approach may sometimes lead to artifacts − e.g. selecting a high point in the baseline as a peak, however, these do not cause harm to the purpose thanks to the robustness of the alignment algorithm as discussed above (and in the next subsection). Applying additional quality control on the selected peaks was therefore not implemented to save computational time. The peaks' retention/migration times are then calculated as a centroid in the vicinity of the peak−top strategic points. Smoothing the data before peak picking could prove beneficial in some cases.

The function `DAMPFindNodesFast` selects strategic points from a chromatogram/electropherogram. The function returns a list of strategic points' retention times and signal intensities. If a new strategic point can not be found between two other strategic points (the restricted areas of the two overlap) or the distance is not above threshold, `{{0,0}}` is returned. This is to comply with the restrictions set on compiled functions in *Mathematica*. The `{0,0}` values are then filtered out in the `DAMPPickChromatogramPeaks` function.

The `DAMPPickChromatogramPeaks` function takes the minimal distance between two strategic points (in minutes) via the `NodeRestrictedVicinity` option and converts it to approximate number of datapoints for the `DAMPFindNodesFast`. The conver−sion is done to allow faster processing.

`DAMPGetPeakCentroid` calculates the peak centroid from datapoints around the specified retention time. The selection of datapoints for calculation is determined by the option `CentroidVicinity` (in minutes, datapoints within this distance in both directions are selected). Note that the retention/migration time values coming from the compiled function `DAMPFindNodesFast` may differ from the values in the original chromatogram, since the compiled function converts the retention time values, which may be rational, to real. Therefore, it is necessary to use the `Select` function in `DAMPGetCentroid` instead of trying to find the position of the retention time value (using the `Position` function) and selecting a certain number of datapoints in the neighborhood.

`DAMPPickPeaks[msdata,options]` picks peaks from all chromatograms/electropherograms in `msdata` and returns them in a list `{{mz,peaklist},{mz,peaklist},...}`. The function does not have any default options, all options passed to `DAMPPickPeaks` are passed further to `DAMPPickChromatogramPeaks`.

```
DAMPFindNodesFast = Compile[{{chrom, _Real, 2}, {thresh, _Real}, {nrv, _Integer}},
  Module[{chromlen, vdists = {}, tmp = {}, newnodepos},
   chromlen = Length[chrom];
   If[chromlen > 2 (nrv + 1),
    vdists = Table[
      Abs[chrom[[i, 2]] - (chrom[[1, 2]] + (chrom[[1, 2]] - chrom[[chromlen, 2]])/(chrom[[1, 1]] - chrom[[chromlen, 1]]) (chrom[[i, 1]] - chrom[[1, 1]]))],
      {i, 1 + nrv, chromlen - nrv}];
    newnodepos = If[# > thresh, Position[vdists, #][[1, 1]] + nrv, 0] &[Max[vdists]];
    If[newnodepos == 0, {{0, 0}},
     Join[DAMPFindNodesFast[Take[chrom, {1, newnodepos}], thresh, nrv], {chrom[[newnodepos]]},
      DAMPFindNodesFast[Take[chrom, {newnodepos, chromlen}], thresh, nrv]]
    ],
    {{0, 0}}
   ]
  ], {{DAMPFindNodesFast[_, _, _], _Real, 2}}
 ]

Options[DAMPGetPeakCentroid] = {Global`CentroidVicinity → .05}

DAMPGetPeakCentroid[chrom_, rt_, opts___] := Module[{vicinity, selpoints},
  vicinity = Global`CentroidVicinity /. {opts} /. Options[DAMPGetPeakCentroid];
  selpoints = Select[chrom, rt - vicinity ≤ #[[1]] ≤ rt + vicinity &];
  (Plus @@ (#[[1]] #[[2]] & /@ selpoints))/(Plus @@ selpoints[[All, 2]])
 ]

Options[DAMPPickChromatogramPeaks] =
 {Global`Threshold → 1000, Global`NodeRestrictedVicinity → .05, Global`CentroidVicinity → .05}

DAMPPickChromatogramPeaks[chrom_, opts___] := Module[{threshold, nrv, centvic, nodes},
  threshold = Global`Threshold /. {opts} /. Options[DAMPPickChromatogramPeaks];
  nrv = Round[(Length[chrom] - 1)/(chrom[[-1, 1]] - chrom[[1, 1]])
     (Global`NodeRestrictedVicinity /. {opts} /. Options[DAMPPickChromatogramPeaks])];
  centvic = Global`CentroidVicinity /. {opts} /. Options[DAMPPickChromatogramPeaks];
  nodes =
   Join[{chrom[[1]]}, Select[DAMPFindNodesFast[chrom, threshold, nrv], # ≠ {0, 0} &], {chrom[[-1]]}];
  If[centvic === None, #,
     {DAMPGetPeakCentroid[chrom, #[[1]], Global`CentroidVicinity → centvic], #[[2]]} & /@ #] &[
   Select[Partition[nodes, 3, 1], Sign[#[[2, 2]] - {#[[1, 2]], #[[3, 2]]}] == {1, 1} &][[All, 2]]]
 ]

DAMPPickPeaks[msdata_, opts___] := Module[{},
  {#, DAMPPickChromatogramPeaks[DAMPGetChromatogram[msdata, #], opts]} & /@ msdata[[2]]
 ]
```

The peaklists may be used to annotate the density plots. The function `DAMPPeakListToAnnotationTable` converts the peaklist in a format returned by `DAMPPickPeaks` to an annotation table format. The names (both short and long) are assigned to peaks from a sequence of integers (starting with `1`).

```
Options[DAMPPeakListToAnnotationTable] = {Global`LabelTextPosition → 1.5}

DAMPPeakListToAnnotationTable[peaks_, opts___] := Module[{ltpos, mz, tmplist, atlen},
  ltpos = Global`LabelTextPosition /. {opts} /. Options[DAMPPeakListToAnnotationTable];
  tmplist = Join @@ ((mz = #[[1]]; {mz, #[[1]]} & /@ #[[2]]) & /@ peaks);
  atlen = Length[tmplist];
  Transpose[
   Join[Transpose[tmplist], {#, #} &[ToString[#] & /@ Range[atlen]], {Table[ltpos, {atlen}]}]]
 ]
```

A stack of peaks with different m/z values may be present at certain retention/migration times. This may be due to post−source fragmentation of a single compound, isotopic peaks, or due to a group of multiple compounds reaching the detector at the same time. If there is an overwhelming number of stacked peaks, these may have significant influence on the dataset alignment (implemented in the next subsection) and bias the alignment. This is undesirable, especially in the case of CE analysis, where there are numerous peaks stacked at migration time corresponding the position of EOF peak. These peaks often do not seem to follow the migration time shift trend of the remaining compounds. Their influence may be removed by (manually) specifying a time range, which does not contain the stacked

peaks, for fitting the migration time shifts. Another possibility to limit their influence (and to avoid manual specification of the time range) is to automatically select a set of representative peaks: a certain number of highest peaks from every chromatogram/electropherogram and a certain number of highest peaks from every time interval of specified size. This may provide peaks more evenly distributed on the retention/migration time vs m/z plane and eliminate most bias−causing stacked peaks. Given the robustness of the alignment method, a minority of bias−causing peaks should not hijack the alignment. `DAMPSelectRepresentativePeaks` can also be used to select peak picks from a certain time interval only.

```
Options[DAMPSelectRepresentativePeaks] = {Global`PeaksPerChromatogram → All,
  Global`PeaksPerInterval → All, Global`IntervalSize → 1, Global`TimeRange → All}

DAMPSelectRepresentativePeaks[peaklist_, opts___] :=
 Module[{perchrom, perint, intsize, timerange, newlist, tmplist, mz, inters},
  perchrom = Global`PeaksPerChromatogram /. {opts} /. Options[DAMPSelectRepresentativePeaks];
  perint = Global`PeaksPerInterval /. {opts} /. Options[DAMPSelectRepresentativePeaks];
  intsize = Global`IntervalSize /. {opts} /. Options[DAMPSelectRepresentativePeaks];
  timerange = Global`TimeRange /. {opts} /. Options[DAMPSelectRepresentativePeaks];
  newlist = peaklist;
  If[timerange =!= All,
   newlist = {#[[1]], Select[#[[2]], timerange[[1]] ≤ #[[1]] ≤ timerange[[2]] &]} & /@ newlist;
  ];
  If[perchrom =!= All,
   newlist = {#[[1]], Take[#, Min[perchrom, Length[#]]] &[Sort[#[[2]], #1[[2]] > #2[[2]] &]]} & /@ newlist;
  ];
  If[perint =!= All,
   tmplist = Join @@ ((mz = #[[1]]; {mz, #} & /@ #[[2]]) & /@ newlist);
   inters = Range[0, Ceiling[Max[newlist[[All, 2, All, 1]]], intsize];
   tmplist =
    Join @@ ((interitem = #; Take[#, Min[perint, Length[#]]] &[Sort[Select[tmplist, interitem[[1]] <
               #[[2, 1]] ≤ interitem[[2]] &], #1[[2, 2]] > #2[[2, 2]] &]]) & /@ Partition[inters, 2, 1]);
   newlist = (mz = #; {mz, Select[tmplist, #[[1]] == mz &][[All, 2]]}) & /@ newlist[[All, 1]];
  ];
  newlist
 ]
```

The layouts of peaks from different datasets picked by the `DAMPPickPeaks` function may be plotted both prior to and after the align−ment for visual inspection.
`Graphics::realu` was turned off since it started appearing for higher number of peaklists (probably `PlotLegend` related). The result does not seem to be affected when the error is present.

```
Options[DAMPPlotPeakLayout] =
  {Global`mzGridLineFreq → 1, Global`mzTickFreq → 1, Global`LegendItems → Automatic,
   Global`PlotOptions → {TextStyle :→ Global`DAMPTextStyle, Frame → True, ImageSize → 930, Axes → Nor
     SymbolShape → {PlotSymbol[Diamond, 5, Filled → False], PlotSymbol[Box, 5, Filled → False],
       PlotSymbol[Diamond, 3, Filled → False], PlotSymbol[Box, 3, Filled → False]}}};

DAMPPlotPeakLayout[peaklists_, opts___] :=
 Module[{mzgridfreq, mztickfreq, leglabels, plotoptions, styles, mzs, pl, mzi, toplot, rslt},
  mzgridfreq = Global`mzGridLineFreq /. {opts} /. Options[DAMPPlotPeakLayout];
  mztickfreq = Global`mzTickFreq /. {opts} /. Options[DAMPPlotPeakLayout];
  legitems = Global`LegendItems /. {opts} /. Options[DAMPPlotPeakLayout];
  If[legitems === Automatic, legitems = Range[Length[peaklists]]];
  plotoptions = Join[Global`PlotOptions /. {opts} /. Options[DAMPPlotPeakLayout],
    Global`PlotOptions /. Options[DAMPPlotPeakLayout]];
  styles = DAMPGenColors[Length[peaklists]];
  mzs = Union[Join @@ peaklists[[All, All, 1]]];
  toplot = (pl = #; Join @@ MapIndexed[(mzi = #2[[1]];
          {#[[1]], mzi} & /@ (Join @@ Select[pl, #[[1]] == mzs[[mzi]] &][[All, 2]])) &, mzs]) & /@ peaklists;
  Off[Graphics::"realu"];
  rslt = MultipleListPlot[Sequence @@ toplot, Sequence @@ plotoptions, SymbolStyle → styles,
    GridLines → {None, Table[{i, {GrayLevel[.7]}}, {i, 1, Length[mzs], mzgridfreq}]},
    FrameTicks → {Automatic, {#, mzs[[#]]} & /@ Range[1, Length[mzs], mztickfreq]},
    PlotLegend → (StyleForm[ToString[#], Sequence @@ Global`DAMPTextStyle] & /@ legitems),
    LegendShadow → None, LegendSize → {.2, .8}, LegendPosition → {1, -.4}];
  On[Graphics::"realu"];
  rslt
 ]
```

■ Dataset alignment

A combination of global optimization and dynamic programming (DP) is used for dataset alignment. DP score serves as a measure of goodness of the alignment between two datasets (their peaklists). Parameters of a mathematical function, which is assumed to be able to fit the time shifts (as a function of retention/migration time in one of the samples) of corresponding peaks between two datasets, are optimized to achieve the lowest DP score.
For a discussion on using DP in *Mathematica*, please refer to [3].

```
Options[DAMPDPChromatogramScore] = {Global`GapPenalty → .5};

DAMPDPChromatogramScore[pl1_, pl2_, opts___] := Module[{mpos, gappenalty},
  gappenalty = Global`GapPenalty /. {opts} /. Options[DAMPDPChromatogramScore];
  Clear[mpos];
  mpos[0, 0] = 0;
  mpos[i_, 0] := mpos[i, 0] = i gappenalty;
  mpos[0, j_] := mpos[0, j] = j gappenalty;
  mpos[i_, j_] := mpos[i, j] = Min[mpos[i - 1, j - 1] + Abs[pl1[[i, 1]] - pl2[[j, 1]]],
     mpos[i - 1, j] + gappenalty, mpos[i, j - 1] + gappenalty];
  mpos[Length[pl1], Length[pl2]]
 ]

Options[DAMPDPScore] = {Global`GapPenalty → .5};

DAMPDPScore[pls1_, pls2_, opts___] := Module[{gappenalty, itm},
  gappenalty = (Global`GapPenalty /. {opts} /. Options[DAMPDPScore]);
  Plus @@ ((itm = #;
       (If[Length[itm[[2]]] > 0 && Length[#] > 0 && Length[#[[1, 2]]] > 0, DAMPDPChromatogramScore[itm[[2]
            #[[1, 2]], Global`GapPenalty → gappenalty], 0]) &[Select[pls2, #[[1]] == itm[[1]] &]]) & /@ pls
 ]
```

`DAMPFitShiftFunction` optimizes the parameters of a retention/migration time shift function to find the optimum alignment between two peak lists. The parameters are optimized (using the `NMinimize` function), the timescale on one of the peaklists is modified accordingly, and the goodness of the alignment evaluated using `DAMPDPScore`. The `DAMPDPScore` is used as the objective function to be minimized.
The default function to fit the migration time shifts is the one derived by Reijenga [4] for normalizing migration times in capillary electrophoresis. The default choice is influenced by the predominant use of CE−based approaches in the authors' institute (Institute for Advanced Biosciences, Keio University). Any custom migration time shift function can be passed to the `DAMPFitShiftFunction` via the `ShiftFunction` option. Parameters for the retention/migration time shift function with lower and upper bounds for the optimization range can be passed via the `ShiftFunctionParameters` option. If `ShiftFunctionParameters` is set to `Automatic`, the parameter names are extracted from the shift function automatically and are assigned no bounds for the optimization range. The peaks to be used for fitting the time shift function may be limited to those (for the peaklist `pls1`) falling within a specific time range set by the `TimeRange` option.
The `GapPenalty` option may hold a list of gap penalty values to be used iteratively for fitting the time shift function. This may be desirable when a small gap penalty value is required for good alignment. If a big gap penalty value is used, noncorresponding peaks from the two peaklists which are close enough to fall within the gap penalty value may negatively affect the alignment. However, if only a small gap penalty value is used and time shifts between the two peaklists are significant, `NMinimize` may not find the region of convergence to the global minimum. Subproblem scores are assigned to gap penalty values for a wider range of shift function parameters so the objective function does not change (if all scores are assigned the gap penalty values) or may converge to a local minimum (if noncorresponding peaks are within the gap penalty distance). A reasonable approach in cases, where time shifts are significant, appears to be a two−step fitting of the time shift function. First, the fitting is performed with a big gap penalty value to find an approximate alignment. Then, second fitting is performed with a small gap penalty value with initial regions of parameters set to the neighborhood of optimized parameter values from the first fitting.

```
DAMPAlignPeakList[peaklist_, shiftfunc_] := Module[{},
  Transpose[{peaklist[[All, 1]], Transpose[{shiftfunc[#[[All, 1]], #[[All, 2]]}] & /@ peaklist[[All, 2]]}]
 ]

Options[DAMPFitShiftFunction] =
 {Global`ShiftFunction → (1/(1/Global`α# + Global`γ/2) &), Global`ShiftFunctionParameters →
   {{Global`α, .8, 1.2}, {Global`γ, -.04, .04}}, Global`GapPenalty → .5,
  Global`NMinimizeOptions → {MaxIterations → 1000}, Global`TimeRange → {0, ∞}}

DAMPFitShiftFunction[pls1_, pls2_, opts___] :=
 Module[{shiftfunc, params, gappenalty, autoparams, timeinterval, steprslt},
  shiftfunc = Global`ShiftFunction /. {opts} /. Options[DAMPFitShiftFunction];
  params = Global`ShiftFunctionParameters /. {opts} /. Options[DAMPFitShiftFunction];
  gappenalty = Global`GapPenalty /. {opts} /. Options[DAMPFitShiftFunction];
  autoparams = Select[Variables[shiftfunc[DAMPDummyVariable]], # =!= DAMPDummyVariable &];
  If[params === Automatic, params = autoparams];
  If[VectorQ[gappenalty],
   (steprslt = DAMPFitShiftFunction[pls1, pls2,
        Global`GapPenalty → #, Global`ShiftFunctionParameters → params, opts]; params =
       {#, (Sequence @@ {# - .02 Abs[#], # + .02 Abs[#]}) &[# /. (Global`BestFitPars /. steprslt)]} & /@
        autoparams) & /@ gappenalty;
   steprslt,
   timeinterval = Global`TimeRange /. {opts} /. Options[DAMPFitShiftFunction];
   {Global`Score → #[[1]], Global`BestFitFunc → shiftfunc /. #[[2]], Global`BestFitPars → #[[2]]} &[
    NMinimize[DAMPDPScore[Transpose[
       {pls1[[All, 1]], Select[#, timeinterval[[1]] ≤ #[[1]] ≤ timeinterval[[2]] &] & /@ pls1[[All, 2]]}],
      DAMPAlignPeakList[pls2, shiftfunc], Global`GapPenalty → gappenalty], params,
     Sequence @@ Join[Global`NMinimizeOptions /. {opts} /. Options[DAMPFitShiftFunction],
       Global`NMinimizeOptions /. Options[DAMPFitShiftFunction]]]]
 ]
```

Annotation table may be converted to a peak list format for displaying it on the peak layout plots and also for purposes of aligning the annotation table onto a reference dataset. There are no default options for the `DAMPAlignAnnotationTable`, all options passed to the `DAMPAlignAnnotationTable` are passed further to the `DAMPFitShiftFunction`, which is used internally.

```
Options[DAMPAnnotationTableToPeakList] = {Global`Resolution → 1}

DAMPAnnotationTableToPeakList[atbl_, opts___] := Module[{resol, tmpannot, mz},
  resol = Global`Resolution /. {opts} /. Options[DAMPAnnotationTableToPeakList];
  tmpannot = atbl;
  tmpannot[[All, 1]] = Round[tmpannot[[All, 1]]/resol] resol 1.;
  (mz = #; {mz, {#[[2]], 0}} & /@ Select[tmpannot, #[[1]] == mz &]) & /@ Union[tmpannot[[All, 1]]]
 ]

DAMPAlignAnnotationTable[peaklist_, atbl_, opts___] := Module[{atpl, shiftfunc, newatbl},
  atpl = DAMPAnnotationTableToPeakList[atbl, opts];
  shiftfunc = Global`BestFitFunc /. DAMPFitShiftFunction[peaklist, atpl, opts];
  newatbl = atbl;
  newatbl[[All, 2]] = shiftfunc[newatbl[[All, 2]]];
  newatbl
 ]
```

Sample dataset is aligned to a reference dataset by applying the migration time shift function to the timepoints of the sample dataset, interpolating all resulting chromatograms in the sample dataset, and selecting timepoints (from this interpolation) identical to time−points in the reference dataset. This will lead to datasets having corresponding datapoints. Direct datapoint−by−datapoint arithmetic operation will therefore be possible. Signal intensities are adjusted to compensate for peak broadening/compression and conserve peaks' areas. Due to interpolation and the discrete nature of the data, minor differences in the areas before and after rescaling are observed. These seemed to stay within 2% for our datasets, what could be acceptable for visual approach.

```
Options[DAMPAlign] = {Global`SampleNameSuffix → "a"}

DAMPAlign[msdata_, shiftfunc_, timepoints_, opts___] := Module[{compensationcoefs, ifunc, rslt},
  compensationcoefs = Join[{#[[1]]}, #] &[
    (#[[2]] - #[[1]]) / (shiftfunc[#[[2]]] - If[#[[1]] == 0., 0., shiftfunc[#[[1]]]]) & /@ Partition[msdata[[3]], 2, 1]];
  Off[InterpolatingFunction::dmval];
  rslt = {(ifunc = Interpolation[Transpose[{Join[{If[msdata[[3, 1]] == 0, 0, shiftfunc[msdata[[3, 1]]]]},
           shiftfunc[Drop[msdata[[3]], 1]]], #}]]; ifunc[#] & /@ timepoints) & /@
     Transpose[Transpose[msdata[[1]]] compensationcoefs], msdata[[2]], timepoints,
    DAMPAppendToSampleName[msdata[[4]], Global`SampleNameSuffix /. {opts} /. Options[DAMPAlign]]};
  On[InterpolatingFunction::dmval];
  rslt
 ]
```

■ Dataset normalization

Datasets are normalized by multiplying all the signal intensities by a normalization coefficient. The coefficient value may come from some external source (e.g. from sample weights) or from integrating the peaks of internal standard in the datasets.
A simple integration was implemented (used for the peaks of internal standards only). No peak detection is performed. The area below a linearly interpolated part of interest of a chromatogram/electropherogram is calculated. The part of interest is specified as a time range. Optionally, the baseline may be calculated as an average of signal intensities from a time range where no peaks are expected. Please note that both the peak integration and baseline estimation is blind in the case of the `DAMPIntegrate` function. Therefore, the chromatograms of integrated peaks as well as the integration results are displayed for visual inspection during automated differential analysis located in the *Assemblies of Core Functions for Common Tasks* section.

```
Options[DAMPNormalize] = {Global`SampleNameSuffix → "n"}

DAMPNormalize[msdata_, coef_, opts___] := Module[{},
  DAMPApplyFunctionToSingle[msdata,
   ({coef #[[1]], #[[2]], #[[3]], #[[4]]} &), Join[opts, Options[DAMPNormalize]]]
 ]

Options[DAMPIntegrate] := {Global`TimeRange → {0, ∞}, Global`BaselineFromTimeRange → None}

DAMPIntegrate[chrom_, opts___] := Module[{bsln, bslnfrotr, timerange},
  bsln = 0;
  timerange = Global`TimeRange /. {opts} /. Options[DAMPIntegrate];
  bslnfromtr = Global`BaselineFromTimeRange /. {opts} /. Options[DAMPIntegrate];
  If[bslnfromtr =!= None,
   bsln = Mean[Select[chrom, bslnfromtr[[1]] ≤ #[[1]] <= bslnfromtr[[2]] &][[All, 2]]]];
  Plus @@ ((((#[[1, 2]] + #[[2, 2]]) / 2 - bsln) (#[[2, 1]] - #[[1, 1]])) & /@
     Partition[Select[chrom, timerange[[1]] ≤ #[[1]] ≤ timerange[[2]] &], 2, 1])
 ]
```

■ Dataset filtering

The `DAMPFilter` function uses one dataset as a criteria to filer out datapoints from a different dataset (although the two could be identical). A pure function to process the signal intensity matrix of the criteria dataset is passed as a parameter. The signal intensity values in the processed criteria dataset determine the fate of signal intensities in the filtered dataset. Signal intensities in the filtered dataset for which the corresponding signal intensities in the processed criteria dataset are zero are leveled to zero as well (or left unchanged if the corresponding signal intensities from the criteria dataset are non−zero).
A simpler version of the `DAMPFilter` function which accepts a numeric threshold value instead of the processing function is defined. Signal intensities in the filtered dataset are leveled to `0` if the absolute values of the corresponding signal intensities in the criteria dataset do not exceed the threshold value.
The dataset to be filtered and the criteria dataset must have identical dimensions (in other words, be aligned or identical).

```
Options[DAMPFilter] = {Global`SampleNameSuffix → "f"}

DAMPFilter[msdata_, criteriamsdata_, threshold_?NumericQ, opts___] := Module[{},
  DAMPFilter[msdata, criteriamsdata, Chop[1. #, threshold] &, opts]
 ]

DAMPFilter[msdata_, criteriamsdata_, filtfunc_, opts___] := Module[{},
  {msdata[[1]] (# / (# /. {0 → ∞, 0. → ∞})) &[filtfunc[criteriamsdata[[1]]], msdata[[2]], msdata[[3]],
   DAMPAppendToSampleName[msdata[[4]], Global`SampleNameSuffix /. {opts} /. Options[DAMPFilter]]}
 ]
```

The `DAMPTrendFilter` function filters out those datapoints from the result of averages of two groups, where at least a certain number of signal intensities from both groups does not follow the same trend (greater or smaller than the average of the other group) as the averages of the groups.

```
Options[DAMPTrendFilter] = {Global`SampleNameSuffix → "tf"}

DAMPTrendFilter[msdatas1_, msdatas2_, tofiltermsdata_, filtnum_, opts___] :=
 Module[{tmpset, ctrlvals, smplvals, ctrlmean, smplmean},
  tmpset = Table[({ctrlvals, smplvals} = {msdatas1[[All, 1, i, j]], msdatas2[[All, 1, i, j]]};
     {ctrlmean, smplmean} = Mean[#] & /@ {ctrlvals, smplvals};
     If[(Length[Select[ctrlvals, # > smplmean &]] ≥ filtnum &&
         Length[Select[smplvals, # < ctrlmean &]] ≥ filtnum && tofiltermsdata[[1, i, j]] < 0) ||
       (Length[Select[smplvals, # > ctrlmean &]] ≥ filtnum &&
         Length[Select[ctrlvals, # < smplmean &]] ≥ filtnum && tofiltermsdata[[1, i, j]] > 0), 1, 0]),
    {i, 1, Length[msdatas1[[1, 2]]]}, {j, 1, Length[msdatas1[[1, 3]]]}];
  {tofiltermsdata[[1]] tmpset, tofiltermsdata[[2]], tofiltermsdata[[3]], DAMPAppendToSampleName[
    tofiltermsdata[[4]], Global`SampleNameSuffix /. {opts} /. Options[DAMPTrendFilter]]}
 ]
```

■ Applying a function to a group of datasets

The `DAMPApplyFunctionToGroup` function applies a pure function function to all corresponding signal intensities of `msdatas` (a list of `msdata` datasets).

```
Options[DAMPApplyFunctionToGroup] = {Global`ApplyToIntensitiesOnly → True,
  Global`SampleNameSuffix → "", Global`ResultSampleName → Automatic}

DAMPApplyFunctionToGroup[msdatas_, func_, opts___] := Module[{rsltsamplename},
  rsltsamplename = Global`ResultSampleName /. {opts} /. Options[DAMPApplyFunctionToGroup];
  If[rsltsamplename === Automatic, rsltsamplename = Global`SampleName /. msdatas[[1, 4]]];
  If[(Global`ApplyToIntensitiesOnly /. {opts} /. Options[DAMPApplyFunctionToGroup]) == True,
   {func[#] & /@ Transpose[#] & /@ Transpose[msdatas[[All, 1]], msdatas[[1, 2]],
    msdatas[[1, 3]], DAMPAppendToSampleName[{Global`SampleName → rsltsamplename},
     Global`SampleNameSuffix /. {opts} /. Options[DAMPApplyFunctionToGroup]]}
   ,
   tmpdat = func[msdata];
   tmpdat[[4]] = DAMPAppendToSampleName[tmpdat[[4]],
     Global`SampleNameSuffix /. {opts} /. Options[DAMPApplyFunctionToGroup]];
   tmpdat
  ]
 ]
```

■ Miscellaneous functions

`DAMPCheckDir` checks if a directory dir exists and creates it, if it does not exist. Parent directory must exist, error message is displayed otherwise. This function is used in some template notebooks to create directories for the saving/export of the results.

```
DAMPCheckDir[dir_] := Module[{curdir, createrslt},
  curdir = Directory[];
  If[dir == curdir || dir == curdir <> "/", Return[]];
  Off[SetDirectory::"cdir"];
  If[SetDirectory[dir] == curdir,
   Off[CreateDirectory::"nffil"];
   createrslt = CreateDirectory[dir];
   If[createrslt === $Failed,
    Print[StyleForm["Error: Failed to create directory: " <> dir, FontColor → Hue[0]]];
    ,
    Print["Created directory: " <> createrslt];
   ];
   On[CreateDirectory::"nffil"];
   ,
   SetDirectory[curdir];
  ];
  On[SetDirectory::"cdir"];
 ]
```

`DAMPGenColors` generates a list of color specifications having the a specified number of elements. If the number is 6 or less elements, the colors are assigned from the following sequence: blue, red, green, yellow, cyan, magenta. If the number is more than 6, the hue range is split proportionally and the colors are assigned from this sequence. The function is mostly used to assign comprehensive colors to overlaid chromatogram/electropherogram plots.

```
DAMPGenColors[nr_] := Module[{},
  If[nr ≤ 6,
   Take[{Hue[2/3], Hue[0], Hue[1/3], Hue[1/6], Hue[1/2], Hue[5/6]}, nr], Table[Hue[i/nr], {i, 0, nr - 1}]]
 ]
```

`DAMPCalcRelativeDifference` calculates a relative difference between two corresponding signal intensities. The difference between the signal intensities is divided by the larger of the two (or an absolute value of their difference if this is bigger than any of the two, when one of the signal intensities is negative).

```
DAMPCalcRelativeDifference[ref_, smpl_, threshold_: 0] :=
 N[(smpl - ref) / (Chop[1. Max[smpl, ref, Abs[smpl - ref]], threshold] /. {0 → ∞, 0. → ∞})]
```

Assemblies of Core Functions for Common Tasks

■ Normalizing multiple datasets

The `DAMPNormalizeGroup` function aligns `msdatas` datasets to a reference dataset (one picked from `msdatas` specified by the `Reference` option) and normalizes them according to the area of the peak of the internal standard. Additional normalization coeffi−cients may be passed to `DAMPNormalizeGroup` via the `ExternalNormalizationCoefficients` options (to account for different sample weights, etc.). Annotation tables may be passed to the function via the `AnnotationTables` option. These are aligned to the reference dataset as well. For automatic internal standard localization, its short name in the first annotation table must be specified by the `InternalStandard` option. The annotation table is aligned to the reference dataset and the vicinity of the location of the peak of the selected internal standard, as extrapolated from the aligned annotation table, is integrated. This integration is blind, overlaid chromatograms/electropherograms with the integrated area are shown for visual confirmation. The location of the peak of the internal standard in the reference dataset may also be specified explicitly. Also, peak picks, representative peak selection, aligned peaks and aligned annotation table layouts are shown for visual confirmation.
`RepresentativePeakOptions` were set to rather low values for the peak numbers to allow fast alignment with default options.
With many groups, *Mathematica* may run into memory problems. To limit this, signal intensities in internal calculations may be converted to integers to save memory. This is determined by the `SaveMemory` option (default: `True`).

```
Options[DAMPNormalizeGroup] =
 {Global`Reference → 1, Global`AlignmentTimeRange → All, Global`RepresentativePeakOptions →
   {Global`PeaksPerChromatogram → 5, Global`PeaksPerInterval → 8, Global`IntervalSize → .5},
  Global`PeakPickingOptions → {Global`Threshold → 5000}, Global`PeakLayoutPlotOptions → {},
  Global`FitShiftFunctionOptions → {}, Global`AnnotationTables → None,
  Global`OutputTimeRange → All, Global`ExternalNormalizationCoefficients → None,
  Global`Resolution → 1, Global`InternalStandard → None,
  Global`AutoISIntegrationVicinity → {-.25, .25}, Global`SaveMemory → True}
```

```
DAMPNormalizeGroup[msdatas_, opts___] :=
 Module[{reference, aligntr, rpopts, ppopts, fsfopts, annotbls, rslttr,
   extcoefs, resol, is, isin, savemem, msdatalen, samplenames, peakpicks,
   selpeakpicks, fitfuncs, alignedpeaklists, alignedannotbls, refdataset,
   reftimepoints, aligneddatasets, toremovevar, coefs, ismz, istime, istr},
  reference = Global`Reference /. {opts} /. Options[DAMPNormalizeGroup];
  aligntr = Global`AlignmentTimeRange /. {opts} /. Options[DAMPNormalizeGroup];
  rpopts = Global`RepresentativePeakOptions /. {opts} /. Options[DAMPNormalizeGroup];
  ppopts = Global`PeakPickingOptions /. {opts} /. Options[DAMPNormalizeGroup];
  plopts = Global`PeakLayoutPlotOptions /. {opts} /. Options[DAMPNormalizeGroup];
  fsfopts = Global`FitShiftFunctionOptions /. {opts} /. Options[DAMPNormalizeGroup];
  annotbls = Global`AnnotationTables /. {opts} /. Options[DAMPNormalizeGroup];
  rslttr = Global`OutputTimeRange /. {opts} /. Options[DAMPNormalizeGroup];
  extcoefs = Global`ExternalNormalizationCoefficients /. {opts} /. Options[DAMPNormalizeGroup];
  resol = Global`Resolution /. {opts} /. Options[DAMPNormalizeGroup];
  is = Global`InternalStandard /. {opts} /. Options[DAMPNormalizeGroup];
  isin = Global`AutoISIntegrationVicinity /. {opts} /. Options[DAMPNormalizeGroup];
  savemem = Global`SaveMemory /. {opts} /. Options[DAMPNormalizeGroup];
  msdatalen = Length[msdatas];
  samplenames = (Global`SampleName /. #) & /@ msdatas[[All, 4]];
  peakpicks = DAMPPickPeaks[#, Sequence @@ ppopts] & /@ msdatas;
  If[aligntr =!= All,
   peakpicks[[reference]] =
     {#[[1]], Select[#[[2]], aligntr[[1]] <= #[[1]] <= aligntr[[2]] &]} & /@ peakpicks[[reference]];
  ];
  DAMPPlotPeakLayout[peakpicks, Sequence @@ plopts,
   Global`LegendItems -> samplenames, Global`PlotOptions -> {PlotLabel -> "Picked Peaks"}];
  If[Length[#] > 0, Print[StyleForm["Error: No peaks found in dataset(s): " <>
        ToString[#] <> ". Please modify the peak picking options or perform the
          analysis without this(these) dataset(s).", FontColor → Hue[0]]]; Return[]] &[
   Join @@ Position[Length[Join @@ #[[All, 2]]] & /@ peakpicks, 0, 1]];
  selpeakpicks = DAMPSelectRepresentativePeaks[#, Sequence @@ rpopts] & /@ peakpicks;
  DAMPPlotPeakLayout[selpeakpicks, Sequence @@ plopts, Global`LegendItems -> samplenames,
   Global`PlotOptions -> {PlotLabel -> "Representative Peaks"}];
  fitfuncs = DAMPFitShiftFunction[selpeakpicks[[reference]], #, Sequence @@ fsfopts] & /@
    Drop[selpeakpicks, {reference}];
  alignedpeaklists = DAMPAlignPeakList[Sequence @@ #] & /@
    Transpose[{Drop[peakpicks, {reference}], (Global`BestFitFunc /. #) & /@ fitfuncs}];
  DAMPPlotPeakLayout[Join[Take[alignedpeaklists, reference - 1], {peakpicks[[reference]]},
    Take[alignedpeaklists, {reference, msdatalen - 1}]], Sequence @@ plopts,
   Global`LegendItems -> samplenames, Global`PlotOptions -> {PlotLabel -> "Aligned Peaks"}];
  alignedannotbls = None;
  If[annotbls =!= None,
   alignedannotbls =
    DAMPAlignAnnotationTable[selpeakpicks[[reference]], #, Sequence @@ fsfopts] & /@ annotbls;
   DAMPPlotPeakLayout[Join[{peakpicks[[reference]]}, DAMPAnnotationTableToPeakList[#] & /@
      alignedannotbls], Sequence @@ plopts, Global`LegendItems -> Join[
      {samplenames[[reference]]}, ("Annot " <> ToString[#]) & /@ Range[Length[alignedannotbls]]],
    Global`PlotOptions -> {PlotLabel -> "Aligned Annotation Tables"}];
  ];
  If[rslttr === All, refdataset = msdatas[[reference]];,
   refdataset = DAMPCrop[msdatas[[reference]], Global`TimeRange -> rslttr];];
  If[savemem, refdataset = {Round[refdataset[[1]]], refdataset[[2]],
     refdataset[[3]], refdataset[[4]]}];
  aligneddatasets = (reftimepoints = refdataset[[3]];
      If[savemem, {Round[#[[1]]], #[[2]], #[[3]], #[[4]]}, #] &[DAMPAlign[Sequence @@ #, reftimepoints]] &,
       Transpose[{Drop[#, {reference}], (Global`BestFitFunc /. #) & /@ fitfuncs}]) &[msdatas];
  aligneddatasets = Join[Take[aligneddatasets, reference - 1], {refdataset},
    Take[aligneddatasets, {reference, msdatalen - 1}]];
  If[is =!= None,
   If[Length[is] == 0 && alignedannotbls =!= None,
     {ismz, istime} = {Round[#[[1]] / resol] resol 1., #[[2]]} &[
       alignedannotbls[[1, Position[alignedannotbls[[1]], {_, _, is, ___}][[1, 1]]]];
     istr = isin + istime;
     ,
     If[Length[is] == 2, {ismz, istr} = is, Print["Warning: IS specified, but either
          has wrong format or annotation table was not specified."]; is = None];
    ];
  ];
  If[is === None,
   coefs = Table[1, {msdatalen}];
   Print["No internal standard specified."];
   ,
   isareas =
    DAMPIntegrate[DAMPGetChromatogram[#, ismz], Global`TimeRange -> istr] & /@ aligneddatasets;
```

```
  DAMPPlotChromatogram[aligneddatasets, ismz, Global`PlotOptions ->
    {PlotRange -> {Mean[istr] + {-1, 1}, All}}, Global`AnnotationTable ->
    {{ismz, istr[[1]], "", "integration START", 2}, {ismz, istr[[2]], "", "integration END", 2}}];
  coefs = isareas[[reference]] / isareas;
  Print["IS normalization coefficients : " <> ToString[1. coefs]];
 ];
 If[extcoefs =!= None,
  extcoefs = Flatten[extcoefs];
  Print["External normalization coefficients : " <> ToString[1. extcoefs]];
  coefs *= extcoefs;
  Print["Overall normalization coefficients : " <> ToString[1. coefs]];
 ];
 Table[If[coefs[[i]] ≠ 1, aligneddatasets[[i, 1]] *= coefs[[i]];]; If[savemem,
   aligneddatasets[[i, 1]] = Round[aligneddatasets[[i, 1]]];, {i, Length[aligneddatasets]}];
 {Global`NormalizedDatasets → aligneddatasets,
  Global`AlignedAnnotationTables -> alignedannotbls}
]
```

■ Comparing two datasets

The `DAMPTwoDatasets` function uses the `DAMPNormalizeGroup` function to align and normalize two datasets. Datasets representing the absolute, relative, and absolute×relative differences are returned along with the normalized datasets and aligned annotation tables as a list of rules.

```
Options[DAMPTwoDatasets] = {Global`NormalizeGroupOptions → {}, Global`ThresholdForRelative → 0}

DAMPTwoDatasets[msdata1_, msdata2_, opts___] :=
 Module[{normrslt, ndat, annottbls, combsname, absdif, reldif, absreldif},
  normrslt = DAMPNormalizeGroup[{msdata1, msdata2},
    Sequence @@ (Global`NormalizeGroupOptions /. {opts} /. Options[DAMPTwoDatasets])];
  {ndat, annottbls} = {Global`NormalizedDatasets, Global`AlignedAnnotationTables} /. normrslt;
  combsname = (Global`SampleName /. ndat[[2, 4]]) <> " vs " <> (Global`SampleName /. ndat[[1, 4]]);
  absdif = {ndat[[2, 1]] - ndat[[1, 1]], ndat[[1, 2]],
    ndat[[1, 3]], {Global`SampleName → "Absolute difference : " <> combsname}};
  reldif = {N[(ndat[[2, 1]] - ndat[[1, 1]]) / (Chop[1. (Max[Join[Abs[#], {Abs[#[[1]] - #[[2]]}]] & /@
              Transpose[#] & /@ Transpose[{ndat[[2, 1]], ndat[[1, 1]]}]),
        Global`ThresholdForRelative /. {opts} /. Options[DAMPTwoDatasets]] /. {0 → ∞, 0. → ∞})],
    ndat[[1, 2]], ndat[[1, 3]], {Global`SampleName → "Relative difference : " <> combsname}};
  absreldif = {absdif[[1]] Abs[reldif[[1]], ndat[[1, 2]], ndat[[1, 3]],
    {Global`SampleName → "Absolute×Relative difference : " <> combsname}};
  {Global`NormalizedDatasets → ndat, Global`Absolute → absdif, Global`Relative → reldif,
   Global`AbsoluteRelative → absreldif, Global`AlignedAnnotationTables → annottbls}
 ]
```

■ Looking for outliers within multiple datasets

The `DAMPOutliers` function uses the `DAMPNormalizeGroup` function to align and normalize a group of datasets and highlights outlying signals. Two types of results are generated: a *z*−score map and a quartile−based result. Both types of results may be generated for every single dataset (optional). Overall results are generated by default. In this case, only one resulting dataset is generated (for both types of approaches) by using the most outlying datapoint from every set of corresponding signal intensities for result calculation. This simplifies the generation of chromatograms/electropherograms for candidates as these do not have to be generated for each dataset's results.

`DAMPDropOutliers` is used internally by the function `DAMPOutliers` to remove a selected number of outliers from a set of values. The function is employed in *z*−score calculation. The presence of an outlier(s) influences the values of both the mean and the standard deviation of a set. This may lead to a relatively low *z*−score value for an outlier. It is therefore desirable to calculate the mean and the standard deviation (which are used for calculating the *z*−scores) from a set of values not containing an outlier(s). The `DAMPOutliers` function's option `OutliersToDrop` determines the number of outliers to drop from every set of corresponding signal intensities. When the number is set to more than one, the outlier dropping is performed iteratively by calculating the mean and removing the single most distant value from the set. The same specified number of values is dropped from every set regardless of whether the values would be classified as outliers.
The result for a quartile−based calculation is set to `0` if the tested signal intensity is between the first and the third quartile of the set of corresponding signal intensities from all datasets (no 'outliers' are dropped in this case in contrast to the *z*−score calculation). If the tested signal intensity is greater than the third quartile, the result is calculated as the difference between the tested signal intensity and the third quartile and this difference is divided by the interquartile range (the difference between the third and the first quartile). If the tested signal intensity is less than the first quartile, the result is calculated as the difference between the first quartile and the tested signal intensity value and this difference is divided by the interquartile range. A negative sign is assigned to the latter result to indicate a 'negative' outlier.

```
DAMPDropOutliers = Compile[{{dt, _Real, 1}, {nr, _Integer}}, Module[{avg = 0., rslt = {}},
    avg = Plus @@ dt / Length[dt];
    rslt = dt;
    If[nr > 0, rslt = Drop[dt, {(Position[#, Max[#]][[1, 1]]) &[Abs[dt - avg]]}]];
    If[nr > 1, rslt = DAMPDropOutliers[rslt, nr - 1]];
    rslt
   ]];
```

```
Options[DAMPOutliers] = {Global`NormalizeGroupOptions → {},
  Global`OutliersToDrop → 1, Global`IndividualZs → False, Global`IndividualQs → False}

DAMPOutliers[msdatas_, opts___] :=
 Module[{iz, iq, outlierstodrop, normrslt, ndat, annottbls, avgs, stds, minmaxs, qs, rz, rq},
  iz = Global`IndividualZs /. {opts} /. Options[DAMPOutliers];
  iq = Global`IndividualQs /. {opts} /. Options[DAMPOutliers];
  outlierstodrop = Global`OutliersToDrop /. {opts} /. Options[DAMPOutliers];
  normrslt = DAMPNormalizeGroup[msdatas,
    Sequence @@ (Global`NormalizeGroupOptions /. {opts} /. Options[DAMPOutliers])];
  If[Length[normrslt] == 0, Return[{}]];
  {ndat, annottbls} = {Global`NormalizedDatasets, Global`AlignedAnnotationTables} /. normrslt;
  avgs =
   DAMPApplyFunctionToGroup[ndat, ((1. Plus @@ #/Length[#]) &[DAMPDropOutliers[#, outlierstodrop]]) &,
    Global`ResultSampleName → "Averaged datasets"];
  stds = DAMPApplyFunctionToGroup[ndat,
    (StandardDeviation[DAMPDropOutliers[#, outlierstodrop]]) &,
    Global`ResultSampleName -> "Standard deviations"];
  minmaxs = DAMPApplyFunctionToGroup[ndat, {Min[#], Max[#]} &,
    Global`ResultSampleName → "Minimums and maximums"];
  qs = DAMPApplyFunctionToGroup[ndat, Drop[Quartiles[#], {2}] &,
    Global`ResultSampleName → "First and third quartiles"];
  rz = {(If[0 ≤ #[[1]] ≥ #[[2]], -#[[1]], If[0 ≤ #[[2]] ≥ #[[1]], #[[2]], 0]] &[N[{#[[2]] - #[[1, 1]], #[[1, 2]] - #[[2]]}/#[[3]] /. {0 → 1, 0. → 1}]]) &/@
      Transpose[{minmaxs[[1]], avgs[[1]], stds[[1]]}],
    avgs[[2]], avgs[[3]], {Global`SampleName → "Z-scores - most significant from" <>
      (((" : " <> (Global`SampleName /. #)) & /@ ndat[[All, 4]]))};
  iz = If[iz == True, {(If[0 ≤ #[[1]] ≥ #[[2]], -#[[1]], If[0 ≤ #[[2]] ≥ #[[1]], #[[2]], 0]] &[
        N[{#[[2]] - #[[1]], #[[1]] - #[[2]]}/#[[3]] /. {0 → 1, 0. → 1}]]) &/@ Transpose[#] &/@
      Transpose[{#[[1]], avgs[[1]], stds[[1]]}], avgs[[2]], avgs[[3]],
     {Global`SampleName → "Z scores : " <> (Global`SampleName /. #[[4]])}} &/@ ndat, None];
  rq = {(If[0 ≤ #[[1]] ≥ #[[2]], -#[[1]], If[0 ≤ #[[2]] ≥ #[[1]], #[[2]], 0]] &[
        N[{#[[2, 1]] - #[[1, 1]], #[[1, 2]] - #[[2, 2]]}/(#[[2, 2]] - #[[2, 1]]) /. {0 → 1, 0. → 1}]]) &/@ Transpose[#] &/@
     Transpose[{minmaxs[[1]], qs[[1]]}], qs[[2]], qs[[3]], {Global`SampleName →
      "Quartile based outliers - most significant from" <>
       (((" : " <> (Global`SampleName /. #)) & /@ ndat[[All, 4]]))};
  iq = If[iq == True, {(If[0 ≤ #[[1]] ≥ #[[2]], -#[[1]], If[0 ≤ #[[2]] ≥ #[[1]], #[[2]], 0]] &[
        N[{#[[2, 1]] - #[[1]], #[[1]] - #[[2, 2]]}/(#[[2, 2]] - #[[2, 1]]) /. {0 → 1, 0. → 1}]]) &/@ Transpose[#] &/@
      Transpose[{#[[1]], qs[[1]]}], qs[[2]], qs[[3]], {Global`SampleName →
       "Quartile based outliers : " <> (Global`SampleName /. #[[4]])}} &/@ ndat, None];
  {Global`NormalizedDatasets → ndat, Global`AveragedDatasets → avgs, Global`ZScores → rz,
   Global`ZScoresIndividual → iz, Global`QuartileResult → rq,
   Global`QuartileResultIndividual → iq, Global`AlignedAnnotationTables → annottbls}
 ]
```

■ Comparing two groups of replicates

The `DAMPTwoGroups` function uses the `DAMPNormalizeGroup` function to align and normalize the datasets of the two groups of replicates to be compared. One approach to the comparison is to average the datasets in every group and continue the comparison as with two datasets. Absolute, relative, and absolute×relative differences are generated for the averaged datasets. These results can be further filtered with `DAMPTrendFilter` to remove those results, where at least a certain number of signal intensities in both groups does not follow the trend of the averages of the groups. In the current implementation, this is done for the absolute×relative result only. Additionally to the comparison of the averages of the groups, *t*−scores are calculated for the groups of corresponding signal intensities.

```
Options[DAMPTwoGroups] = {Global`NormalizeGroupOptions → {},
  Global`ThresholdForRelative → 0, Global`GroupNames → Automatic, Global`AbsRelTrendFilter → 2}
```

```
DAMPTwoGroups[ctrls_, smpls_, opts___] :=
 Module[{relthreshold, groupnames, ctrllen, smpllen, finaldatasets,
   alignedannotbls, averagedctrls, averagedsmpls, ctrlname, smplname, absolute,
   relative, absrel, filtabsrel, sdsctrls, sdssmpls, tscores, matscores},
  relthreshold = Global`ThresholdForRelative /. {opts} /. Options[DAMPTwoGroups];
  groupnames = Global`GroupNames /. {opts} /. Options[DAMPTwoGroups];
  {ctrllen, smpllen} = {Length[ctrls], Length[smpls]};
  {finaldatasets, alignedannotbls} = {Global`NormalizedDatasets,
     Global`AlignedAnnotationTables} /. DAMPNormalizeGroup[Join[ctrls, smpls],
     Sequence @@ (Global`NormalizeGroupOptions /. {opts} /. Options[DAMPOutliers])];
  averagedctrls = DAMPApplyFunctionToGroup[Take[finaldatasets, ctrllen],
    Mean, Global`SampleNameSuffix → "avg"];
  averagedsmpls = DAMPApplyFunctionToGroup[Take[finaldatasets, -smpllen],
    Mean, Global`SampleNameSuffix → "avg"];
  If[groupnames === Automatic,
   ctrlname = Global`SampleName /. averagedctrls[[4]];
   smplname = Global`SampleName /. averagedsmpls[[4]];
   ,
   {ctrlname, smplname} = groupnames;
  ];
  absolute = {averagedsmpls[[1]] - averagedctrls[[1]], averagedsmpls[[2]], averagedsmpls[[3]],
    {Global`SampleName → "Absolute Difference : " <> smplname <> " - " <> ctrlname}};
  relative = {(averagedsmpls[[1]] - averagedctrls[[1]]) /
     (Chop[Max[Abs[#]] & /@ Transpose[#] & /@ Transpose[{averagedsmpls[[1]], averagedctrls[[1]]}],
        relthreshold] /. {0 → ∞, 0. → ∞}), averagedsmpls[[2]], averagedsmpls[[3]],
    {Global`SampleName → "Relative Difference : " <> smplname <> " - " <> ctrlname}};
  absrel = {absolute[[1]] Abs[relative[[1]]], averagedsmpls[[2]], averagedsmpls[[3]],
    {Global`SampleName → "Absolute×Relative Difference : " <> smplname <> " - " <> ctrlname}};
  filtabsrel = DAMPTrendFilter[Take[finaldatasets, ctrllen], Take[finaldatasets, -smpllen],
    absrel, Global`AbsRelTrendFilter /. {opts} /. Options[DAMPTwoGroups]];
  sdsctrls = DAMPApplyFunctionToGroup[Take[finaldatasets, ctrllen],
    StandardDeviation, Global`SampleNameSuffix → "SD"];
  sdssmpls = DAMPApplyFunctionToGroup[Take[finaldatasets, -smpllen],
    StandardDeviation, Global`SampleNameSuffix → "SD"];
  tscores = {(averagedsmpls[[1]] - averagedctrls[[1]]) /
     Sqrt[sdsctrls[[1]]^2/ctrllen + sdssmpls[[1]]^2/smpllen] /. {0. → ∞, 0 → ∞},
    averagedsmpls[[3]], {Global`SampleName → "t scores : " <> smplname <> " - " <> ctrlname}};
  {Global`NormalizedDatasets → finaldatasets, Global`AveragedGroup1 → averagedctrls,
   Global`AveragedGroup2 → averagedsmpls, Global`Absolute → absolute,
   Global`Relative → relative, Global`AbsRel → absrel, Global`FilteredAbsRel → filtabsrel,
   Global`TScores → tscores, Global`AlignedAnnotationTables → alignedannotbls,
   Global`GroupCounts → {ctrllen, smpllen}, Global`GroupNames → {ctrlname, smplname}}
 ]
```

■ Comparing multiple groups of replicates

One−way ANOVA is used to locate the differences among multiple groups of datasets. To reduce the amount of memory necessary to perform the calculations, the F ratio is calculated for each corresponding group of signal intensities separately (instead of doing it in a whole dataset fashion by transposing them appropriately − this approach proves to be memory demanding). A compiled function `DAMPCalcFRatioFast` for calculating the F ratio was implemented. It is used instead of *Mathematica*'s `ANOVA` function to gain speed. However, the groups must contain an identical number of replicates as mentioned below.
Possibility to use an analogy of the absolute×relative plot (average of control datasets vs. the most distant average) was investigated, but the results did not meet the expectations for the datasets tested. This approach is therefore not used.

`DAMPCalcFRatioFast` calculates an F−ratio (one−way ANOVA) for `data`. `Data` is expected to contain a flat list of values. These are then grouped into groups each containing equal number of values determined by `replicates`. Every group must contain an equal number of replicates − a constraint enforced by this F ratio calculation procedure. This function is used by the `DAMPMultiGroups` function.

```
DAMPCalcFRatioFast = Compile[{{data, _Real, 1}, {replicates, _Integer}},
   Module[{groups, groupings, interm, tmp, totals, d, bms},
    groups = Length[data] / replicates;
    groupings = {# + 1, # + replicates} & /@ Range[0, Length[data] - 1, replicates];
    interm =
     (tmp = #[[1]]^2 / #[[2]]; {#[[1]], #[[2]], #[[1]] / #[[2]], #[[3]], tmp, #[[3]] - tmp}) &[{Plus @@ #, Length[#], Plus @@ #^2}] &
       Take[data, #]] & /@ groupings;
    totals = Plus @@ interm;
    d = totals[[1]]^2 / totals[[2]];
    bms = totals[[5]] - d;
    If[# == 0, 0, bms / (groups - 1) / #] &[totals[[-1]] / (groups (replicates - 1))]
   ]
  ];

Options[DAMPMultiGroups] = {Global`NormalizeGroupOptions → {}, Global`GroupNames → Automatic}

DAMPMultiGroups[msdatas_, replicates_, opts___] :=
 Module[{normopts, groupnames, rslt, findata, annotbls, samplenames, fratios},
  normopts = Join[Global`NormalizeGroupOptions /. {opts} /. Options[DAMPMultiGroups],
    Global`NormalizeGroupOptions /. Options[DAMPMultiGroups]];
  groupnames = Global`GroupNames /. {opts} /. Options[DAMPMultiGroups];
  rslt = DAMPNormalizeGroup[msdatas, Sequence @@ normopts];
  {findata, annotbls} = {Global`NormalizedDatasets, Global`AlignedAnnotationTables} /. rslt;
  If[groupnames === Automatic,
   groupnames = Global`SampleName /. # & /@ findata[[Range[1, Length[findata], replicates], 4]];
  {Global`NormalizedDatasets → findata, Global`AlignedAnnotationTables → annotbls,
   Global`FRatios → {Table[DAMPCalcFRatioFast[#, replicates] &[findata[[All, 1, i, j]]],
      {i, 1, Length[findata[[1, 1]]]}, {j, 1, Length[findata[[1, 1, 1]]]}], findata[[1, 2]], findata[[1, 3
    {Global`SampleName → ("F ratio " <> (StringJoin @@ ((" : " <> #) & /@ groupnames)))}},
   Global`GroupNames → groupnames}
 ]
```

■ Overlaid chromatograms/electropherograms for candidate differences

For the confirmation of signals (and rejection of false positives) from plots highlighting differences between selected datasets overlaid chromatograms/electropherograms may be plotted. Normalized original datasets along with a dataset representing the differences between them (result dataset) are passed to the function `DAMPPlotCandidates` and overlaid chromatograms/electropherograms of the most significant differences are generated. To find the most significant differences, the result dataset is searched for a highest absolute signal intensity value. This datapoint, along with its neighborhood (of specified size in the retention/migration time dimension) is set to `0` and the search for additional significant differences continues iteratively.

```
Options[DAMPPlotCandidates] = {Global'PlotCount → 12, Global'TimeRange → All,
  Global'PlotVicinity → {-.6, .6}, Global'DropVicinity → .2, Global'PlotChromatogramOptions → {}}

DAMPPlotCandidates[datamsdatas_, critmsdata_, opts___] :=
 Module[{plotcount, timerange, plotvicin, dropvicin,
   chromopts, plotopts, tmpmsdata, dlen, pos, tmpplot},
  plotcount = Global'PlotCount /. {opts} /. Options[DAMPPlotCandidates];
  chromopts =
   Join[Global'PlotChromatogramOptions /. {opts} /. {Global'PlotChromatogramOptions → {}},
    Global'PlotChromatogramOptions /. Options[DAMPPlotCandidates]];
  plotopts = Global'PlotOptions /. chromopts /. {Global'PlotOptions → {}};
  timerange = Global'TimeRange /. {opts} /. Options[DAMPPlotCandidates];
  plotvicin = Global'PlotVicinity /. {opts} /. Options[DAMPPlotCandidates];
  dropvicin = Global'DropVicinity /. {opts} /. Options[DAMPPlotCandidates];
  dropvicin = Round[(Length[critmsdata[[3]]] - 1)/(critmsdata[[3, -1]] - critmsdata[[3, 1]]) dropvicin];
  tmpmsdata = DAMPCrop[critmsdata, Global'TimeRange → timerange];
  tmpmsdata[[1]] = Abs[tmpmsdata[[1]]];
  dlen = Length[tmpmsdata[[3]]];
  resplots = {};
  Do[pos = Position[tmpmsdata[[1]], Max[tmpmsdata[[1]]]];
   tmpplot = DAMPPlotChromatogram[datamsdatas, tmpmsdata[[2, pos[[1, 1]]]], Global'PlotOptions →
      {Sequence @@ plotopts, PlotRange → {tmpmsdata[[3, pos[[1, 2]]]] + plotvicin, All},
       Axes → True, AxesOrigin → {tmpmsdata[[3, pos[[1, 2]]]], Automatic}, AxesStyle →
        {{GrayLevel[1]}, {AbsoluteDashing[{5, 5}], GrayLevel[.5]}}}, Sequence @@ chromopts];
   tmpmsdata[[1, pos[[1, 1]], Range[Max[1, pos[[1, 2]] - dropvicin], Min[dlen, pos[[1, 2]] + dropvicin]]] =
   AppendTo[resplots, tmpplot];
   , {Global'PlotCount /. {opts} /. Options[DAMPPlotCandidates]}];
  resplots
 ]
```

References

[1] Ruckstuhl, A. F., Jacobson, M. P., Field, R. W., Dodd, J. A. (2001) Baseline subtraction using robust local regression estimation. *J. Quant. Spectrosc. Radiat. Transfer*, **68**, 179−193

[2] Wallace, W. E., Kearsley, A. J., Guttman, C. M. (2004) An Operator−Independent Approach to Mass Spectral Peak Identification and Integration. *Anal. Chem.*, **76**, 2446−2452

[3] Wagner, D. B. (1995) Dynamic Programming. *Mathematica J.*, **5**, 42−51

[4] Reijenga, J. C., Martens, J. H. P. A., Giuliani, A., Chiari, M. (2002) Pherogram normalization in capillary electrophoresis and micellar electrokinetic chromatography analyses in cases of sample matrix−induced migration time shifts. *J. Chromatogr. B*, **770**, 45−51

Epilog

```
End[]

Print["MathDAMP version 1.0.0 loaded (2006/04/26)"]
Print[
 "This program 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."]

EndPackage[]
```