Documentation for MAGMA functions

This file contains documentation for the most important functions in the MAGMA file accompanying the thesis.

ComputeDegree(*clustervar, grading, rank)*

Returns the degree of the cluster variable *clustervar* given grading vector *grading* (a sequence of integers). The integer *rank* should be set to the rank of the cluster algebra.

BipartiteB(n)/ BipartiteC(n)

Generate the Cartan matrix B_n in bipartite form.

A3(a,b,c)

Returns the skew-symmetric matrix with sub-diagonal entries given by the integers a, b and c.

MAExMat(k, l)/MAinitCl(k, l)/MAInitGr(k, l)

Generate the exchange matrix/initial cluster/initial grading vector for the cluster algebra structure on $\mathcal{O}((M(k,l))$.

GAExMat(k,l)/GAInitGr(k, l)

Does the same as above but for the Grassmannian ring (input (k,l) for $Gr(k,k+l)$).

MatrixPathMutation(*rank, B, path*)

Given exchange matrix B, return the matrix obtained by applying the mutation path *path* to B.

Optional parameters:

- *mutable_cols:=rank* The number of mutable columns of B.

IteratedClusterMutation(*clusterplusmat,mutationlist, rank*)

Returns the seed (or degree seed along with list of degrees found) obtained from the initial seed *clusterplusmat* (which needs to be a list (sequence) with the cluster as the first entry and matrix as the second) under the mutation path *mutationlist*. The cluster is in *clusterplusmat* is allowed to be a grading vector. The seed is returned as a list with entries of the form [*cluster*, *matrix*] or as [*degree cluster*, *matrix*, *degrees*].

Optional parameters:

- *multi:=false* Whether a multi-grading is being input as the degree cluster.
- *fast:=false* If this is true, computations are done using degree vectors rather than with variables.
- *mutable_cols:=rank* The number of mutable columns of the exchange matrix in *clusterplusmat*.

DvectorPathMutation(*rank, B, path)*

Returns the denominator cluster obtained after applying the mutation path *path* using initial exchange matrix B.

PathMutation(*rank, B, path, grading)*

Depending on the optional parameters, prints information about the variables, degrees, exchange matrix, cluster, denominator cluster, degree cluster, etc. obtained after applying the mutation path *path* (a sequence of integers) with the initial exhchange matrix B and grading vector *grading*. Can also return the objects themselves.

Optional parameters:

- *multi:=false* This needs to be set to true if the grading is a multi-grading.
- *fulloutput:=false* Whether to compute and output full cluster variables (otherwise, denominator vectors, or just grading vectors, will be used instead).
- *compute_dvectors:=true* Whether to use denominator vectors when calculating paths. If this is false, only the degree cluster will be calculated (unless *fulloutput* is true).
- *stringoutput:=true* Whether to return a string (easier to read) or return a list containing the cluster and matrix after the mutation path
- *show_degreecluster:=true* Whether to display the degree cluster obtained.
- *show_matrix:=true* If using *fulloutput:=true*, whether to show the matrix obtained.
- *show_heading:=true* Whether to show a heading to make output easier to interpret.
- *show_deglist:=false* Whether to show the list of all degrees we have obtained under the mutation path.

- *custom_initclust:=[]* Here a custom initial cluster can be input to use instead of the standard initial cluster $(X_1, \ldots, X_n)$, which will be used by default.
- *mutable_cols:=rank* The number of mutable directions if there are frozen variables. (If this is set to k, the frozen directions are assumed to be the last k.)

Example usage:

- PathMutation(3, A3(-2,-1,1), [1], [1,1,2] : fulloutput:=true);

MAPathMutation/GAPathMutation(*k, l, path*)

These functions call PathMutation with inputs appropriate for the cluster algebras corresponding to $\mathcal{O}(M(k,l))$ and $\mathcal{O}(Gr(k,k+l))$. There are optional parameters that are mostly the same as in PathMutation (but *custom_initclust* is no longer an optional parameter since it is being set in PathMutation).

RandPath(*rank, length*)

Return a random repetition-free mutation path of lenght *length* with entries in $\{1, \ldots, rank\}$.

Optional parameters:

- *exclude:={}* A set of mutation directions that will not be selected.

AllClusterVariables(*rank, initexchmat, CAtype*)

For a finite-type cluster algebra, finds all cluster variables. The type of the cluster algebra needs to be specified in *CAtype* as a one-letter string (e.g. *CAtype:="A"*). Returns a list whose first entry is the list of cluster variables found, second entry is the list of mutation paths used and third entry is the time taken.

Optional parameters:

- *grading:=[0..rank]* This can be set to an initial grading vector. If it is changed from the default, the function will replace each cluster variable returned with a pair whose first entry is the variable and second its degree.

ACVRadius(*rank, initexchmat, radius*)

Acts in the same way as AllClusterVariables (including the optional parameter) but only only tries mutation paths whose lengths are at most *radius*. In other words, finds all cluster variables of a cluster algebra obtainable within a specified mutation radius.

PrintACV/PrintACVR/BriefACVR(*rank, initexchmat, CAtype)*

Calls AllClusterVariables/ACVRadius and outputs relevant information in a readable format (rather than returning lists of objects etc.).

Optional parameters:

- *grading:=[0..rank]* Works as in AllClusterVariables.

MAFindBigMats(*k, l, length*)

In the cluster algebra for $\mathcal{O}(M(k,l)$, searches for and prints mutation paths that lead to a double (or larger) arrow by using all paths up to length *length*. This is used to prove that infinitely mant degrees occur in this graded cluster algebra.

Optional parameters:

- *minlength:=1* The minimum length of paths to try.
- *custom_paths:=[]* If this is changed form the default, a custom list of paths to try will be used.
- *minentry:=2* The minimum arrow weight (or minimum entry of the exchange matrix, in absolute value) to search for.

MAFindBigMatsRand(*k, l, length, num_paths*)

This does the same as MAFindBigMats, but instead of exhausting all paths, *num_paths* random paths of length *length* are attempted.

Optional parameters:

- *minentry:=2* Works as in MAFindBigMats.

Quiver(*Q*)

Given a skew-symmetric matrix Q, returns the corresponding quiver represented as a simple graph with labelled vertices encoding the edge weights and directions. This allows us to use the IsIsomorphic function, which takes advantage of the nauty package for fast graph isomorphism detection.

FindQuiverClass(*A*)

Given the skew-symmetric matrix A, returns the list of matrices corresponding to quivers that are mutation equivalent.

IsEsseqDegreeQuiver(*A, gradA, B, gradB*)

For skew-symmetric matrices A and B, checks if the corresponding degree quivers (with degrees of vertices defined by the grading vectors *gradA* and *gradB*, to be input as sequences of integers) are essentially equivalent. Returns true/false as well as the permutation such that the first degree quiver is essentially equivalent to the second.

Optional parameters:

- *allow_neg_grad:=false* If this is set to *true*, the function will return *true* if the degree quivers are essentially equivalent up to sign of the grading vectors.
- *only_allow_neg_grad:=false* If this is set to *true*, the function will only return *true* if the degree quivers are essentially equivalent after negating one of the grading vectors.
- *custom_isom:=false* This should be set to *true* if *given_isom* is going to be changed from the default (see below).
- *given_isom:=0* When *custom_isom* is set to *true*, this can be set to a specific isomorphism. If so, the function returns true if the two degree quivers are essentially equivalent by that specific isomorphism. The isomorphism should be a mapping between vertices of graphs (as is returned by IsIsomorphic, for example).
- *multi:=false* If this is *true*, multi-gradings can be used in *gradA* and *gradB*. This should be a list of basis vectors for the grading (sequence of sequences, e.g. [[3,1,2],[1,-1,1]]).

Example usage:

- X_123:=MatrixPathMutation(7, X_7, [1,2,3]);
  X_123g:=IteratedClusterMutation([*[1,1,2,1,1,1,1], X_7*], [1,2,3], 7 : fast:=true)[1];
  IsEsseqDegreeQuiver(X_7, [1,1,2,1,1,1,1], X_123, X_123g);

FindDegreeQuivers(*A, grading*)

Attempts to find all degree quivers essentially equivalent the degree quiver corresponding to the skew-symmetric matrix A with vertex degrees defined by *grading*. Returns a list of matrices for the quivers and a corresponding list of gradings.

Optional parameters:

- *multi:=false* Works as in previous functions.

Example usage:

- FindDegreeQuivers(E_7_11, [[0,1,0,-1,0,0,0,0,1],[0,0,1,2,1,0,0,0,0],[0,0,0,-1,0,1,0,1,0]] : multi:=true);

FDQ_NewSession(*file_prefix, A, grading*)

For degree quivers with large mutation classes. Starts performing FindDegreeQuivers on A with the grading *grading* and saves progress to a file. Saves associated files with a prefix as set by the string *file_prefix*. Once a session is running, its state will be saved automatically. Interrupting the function or closing or disconnecting from the MAGMA session should not cause any problems as the function can pick up from where it left off using RestoreSession below.

Optional parameters:

- *multi:=false* Works as above.
- *save_frequency:=100* How frequently to save the state.

Example usage:

- FDQ_NewSession("E_aff_6", E_aff_6, [1,0,1,0,1,0,1] : save_frequency:=200);

FDQ_RestoreSession(*file_prefix*)

Continues the computation of FindDegreeQuivers for the set of files created by FDQ_NewSession with the prefix *file_prefix*.

Optional parameters:

- *save_frequency:=100* How frequently to save the state.

Example usage:

- FDQ_RestoreSession("E_aff_6");

FindEsseqDegSeeds:=function(*Q, grading, attempts*)

Searches for and prints out paths that result in a degree quiver essentially equivalent to the one defined by the skew-symmetric matrix Q with grading given by *grading*. Stops after trying a number of paths equal to *attempts*. Returns true or false (depending on whether any successful paths were found), a list of successful paths and a list of corresponding permutations.

Optional parameters:

- *exclude_dirs:={}* A set of mutation directions to exclude from paths that are attempted.
- *randomise:=false* Whether to randomise the paths that are attempted.
- *min_length:=1* The minimum length of paths that are to be attempted.
- *allow_var_esseq:=true* Whether to allow clusters that have the same variables to count as successes.
- *allow_neg_grad:=false* Whether to allow degree seeds that are essentially equivalent up to negating the degree cluster to count as successes.
- *only_allow_neg_grad:=false* As above, but to only allow such degree seeds to count as successes.
- *min_successes:=10* Stop the function once this many successful paths have been found.
- *multi:=false* Whether the grading is a multi-grading.
- *display_messages:=true* Whether to print a message about the paths found.

Example usage:

- found, paths, perms:=FindEsseqDegSeeds(E_aff_6,grad, 100 : randomise:=true, min_length:=2, min_successes:=1, allow_var_esseq:=false);

FindNonVarDegLoop(*Q, grading, attempts*)

Searches for and prints paths that give a degree seed essentially equivalent to the one defined by the matrix Q and grading vector *grading*, but with cluster variables that appear to grow on repeated application of the path. (This is useful for finding paths that can be used to prove the existence of infinitely many variables of a certain degree.) Returns the list of successful paths, the list of corresponding permutations, and the weight of each path after a certain number of repetitions (which is a measure of the size of the corresponding denominator cluster, or how fast the cluster variables are growing). Returns

Optional parameters:

- *exclude_dirs:={}* Mutation directions to exclude from attempted paths.
- *depth:=200* How many times to repeat each mutation path (repeated paths will have entries permuted as per essential equivalence).
- *min_dvec_weight:=Ceiling(depth/4)* The minimum weight that the resulting denominator cluster needs to be for a path to be considered successful.
- *search_strength:=Ncols(Q)*100* The number of paths to try per attempt.
- *min_length:=Round(Ncols(Q)/2)* The minimum length of paths that are to be attempted.
- *multi:=false,allow_neg_grad:=false* Works as in above functions.
- *allow_neg_grad:=false* Works as in above functions.
- *only_allow_neg_grad:=false* Works as in above functions.

Example usage:

- paths,perms,weights:=FindNonVarDegLoop(E_7_11, [[0,1,0,-1,0,0,0,0,1],[0,0,1,2,1,0,0,0,0],[0,0,0,-1,0,1,0,1,0]], 100 : depth:=15, multi:=true, min_length:=5);

RepeatPermedPathMut(*reps, Q, grad, path, perm*)

Given a path which gives and degree seed that is essentially equivalent to the one defined by the skew-symmetric matrix Q and grading vector *grad* by the permutation perm, apply the path n times, permuting *path* by *perm* each time.

Optional parameters:

- *multi:=false* Works as in a above functions.
- *initial_path:=[]* If this is changed from the default, an additional path will be applied before applying the repeated paths.
- *terminal_path:=[]* As above, but with a path applied after the repeated paths.
- *subpath_length:=0* If this is greater than 0, the last repetition of the path will be replaces by a subpath of length *subpath_length*.
- *compute_dvectors:=true* Works as in PathMutation, which is called by this function.
- *stringoutput:=true* Works as in PathMutation.
- *show_deglist:=true* Works as in PathMutation.

Example usage:

- RepeatPermedPathMut(5, E_aff_7, [[-1,0,-1,0,1,0,1,0],[-1,0,-1,0,0,0,0,1]], path, perm : subpath_length:=1, multi:=true);