

1 Convexity of a NC function

This chapter describes commands which do two things. One is compute the "region" on which a noncommutative function is matrix convex. The other is take a noncommutative quadratic function variables H_1, H_2 , etc and give a Gram representation for it, that is, represent it as

$$V[H]^T M V[H]$$

a "vector" with the H_j entering linearly and M a matrix. Other commands are described here but they are subservient to **NCConvexityRegion**[*afunction*,*alist*,*options*] and would not be used independently of it. The commands in this chapter are not listed alphabetically but are listed in the presumed order of importance.

1.1 NCConvexityRegion[afunction,alist,options]

Aliases: **None**.

Description: **NCConvexityRegion**[*afunction*,*alist*] performs three main operations. First it computes the Hessian with respect to *alist* (see **NCHessian**). Then, using **NCMatrixOfQuadratic**, the Hessian is factored into the form $v^t M v$. Finally **NCAAllPermutationLDU** is called to compute the *LDU* factorization of M . A list of the diagonal elements of D is returned. Options permit the user to select a range of different permutation matrices, thereby producing several possibly distinct diagonal matrices D .

Arguments: *afunction* is a function whose variables are listed in *alist*, where *alist* should be of the form $\{x_1, x_2, \dots, x_n\}$.

For example, **NCConvexityRegion**[$x**y**x, \{x, y\}$] gives:

MIDDLE MATRIX IS SIZE 2 X 2

AT MOST 2 PERMUTATIONS POSSIBLE.

{1}

{{{0, 2}}}

Here, **NCHessian**[$x**y + y**x, \{x, h\}, \{y, k\}$] gives

$2h**k + 2k**h,$

NCMatrixOfQuadratic[$2h**k + 2k**h, \{h, k\}$] gives

{{{h, k}}, {{0, 2}, {2, 0}}, {{h}, {k}}},

and **NCAAllPermutationLDU**[{{{0, 2}, {2, 0}}] gives

{{{{1, 0}, {0, 1}}, {{0, 2}, {2, 0}}, {{1, 0}, {0, 1}}, {{1, 0}, {0, 1}}},

{{{{1, 0}, {0, 1}}, {{0, 2}, {2, 0}}, {{1, 0}, {0, 1}}, {{1, 0}, {0, 1}}}}.

The default options for **NCConvexityRegion** are: {**NCSimplifyDiagonal** \rightarrow **False**, **DiagonalSelection** \rightarrow **False**,

ReturnPermutation \rightarrow **False**, **ReturnBorderVector** \rightarrow **False**}.

NCSimplifyDiagonal is an option geared toward a similar option used in **NCLDUDecomposition**. This will make sure that the pivots (or diagonal entries) are all first simplified with **NCSimplifyRational** before they

are used to check that the pivots are all nonzero. Simplifying the pivots using `NCSimplifyRational` can be quite time consuming, so by default we commute everything and then use Mathematica simplification commands. We do this only to convince ourselves that the pivot is nonzero. If all the pivots are zero using `CommuteEverything` we then revert to using `NCSimplifyRational` to verify our suspicions. Setting `NCSimplifyDiagonal` \rightarrow `True` will bypass the commute everything step. (Note: Either way, the unsimplified form of the pivot is returned unless it is equal to zero.)

Different permutations return different diagonals. Some diagonals are simpler to work with than others. Because of this, we allow the user to select a sampling of different permutations. The total number of permutations will not be known until M is computed. After M is computed, the total number of possible permutations will be printed on the screen. `DiagonalSelection` \rightarrow $\{n\}$ returns the diagonals resulting from the first n permutations. `DiagonalSelection` \rightarrow $\{k,n\}$ returns the diagonals resulting from the k^{th} through n^{th} permutations. Since the total number of permutations is assumed to be unknown by the user, if n is too high, then n is replaced by the total number of permutations which are possible to compute in a modest period of time. Also, not all of the permutations are permissible. Because of this, `NCLDUdecomposition` automatically pivots if an invalid permutation is used for a particular step. This means it is possible that not all the diagonals returned result from different permutations. For this reason there is the option `ReturnPermutation` which if entered as `True` returns the permutations used for each resulting factorization. Finally, the user may wish to analyze the border vectors and may do so by setting `ReturnBorderVector` to `True`. This will cause `NCConvexityRegion` to return the border vectors v from the $v^t M v$ factorization of the hessian. Now v^t will have the form

$$[L_{11}H_1, L_{12}H_1, \dots, L_{1k_1}H_1, \dots, L_{n1}H_n, L_{n2}H_n, \dots, L_{nk_n}H_n]$$

So what will actually be returned is a list of the form

$$\{\{L_{11}, \dots, L_{1k_1}\}, \dots, \{L_{n1}, \dots, L_{nk_n}\}\}.$$

This vector will be formed using a call to `NCBorderVectorGather`. Also, a call will be made to `NCIndependenceCheck` to determine, if possible, whether or not the elements of the above list are independent. The results of this check will be printed to the screen.

Comments / Limitations: None.

1.2 NCMatrixOfQuadratic[$Q, \{H_1, H_2, \dots, H_n\}$]

Aliases: **None**.

Description: `NCMatrixOfQuadratic[Q, {H1, H2, ..., Hn}]` gives a vector matrix factorization of a symmetric quadratic function Q in noncommutative variables $\vec{H} = \{H_1, H_2, \dots, H_n\}$ and their transposes. `NCMatrixOfQuadratic[Q, {H1, H2, ..., Hn}]`, generates the list `{left border vector, coefficient matrix, right border vector}`. That is, Q is factored into the vector-matrix-vector product $V[\vec{H}]^T M_Q V[\vec{H}]$. The vector $V[\vec{H}]$ is linear in \vec{H} and is called a *border vector of the quadratic function* Q . The matrix M_Q is called the *coefficient matrix of the quadratic function* Q .

Arguments: Each term of Q is assumed to be a quadratic expression in terms of the variables H_1, H_2, \dots, H_n and their transposes (Q is homogeneous). For example, suppose that $Q = 3tp[x] ** y + 3tp[y] ** x$ and that $\vec{H} = \{x, y\}$. Then, `NCMatrixOfQuadratic[Q, \vec{H}]` gives

$$\{\{\{tp[x], tp[y]\}\}, \{\{0, 3\}, \{3, 0\}\}, \{\{x\}, \{y\}\}\}.$$

In `MatrixForm`, this looks like

$$(tp[x] \quad tp[y]) * \begin{pmatrix} 0 & 3 \\ 3 & 0 \end{pmatrix} * \begin{pmatrix} x \\ y \end{pmatrix}.$$

In general, suppose Q is a quadratic function of two variables, $\vec{H} = \{H, K\}$, with all transpose elements H^T, K^T occurring before all non-transpose elements. Then `NCMatrixOfQuadratic` will return the *left border vector* $V[\vec{H}]^T$, the matrix M_Q , and the *right vector* $V[\vec{H}]$ where

$$M_Q := \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1,\ell_1} & A_{1,\ell_1+1} & \cdots & A_{1,n} \\ A_{12}^T & A_{22} & \cdots & A_{2,\ell_1} & A_{2,\ell_1+1} & \cdots & A_{2,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ A_{1,\ell_1}^T & A_{2,\ell_1}^T & \cdots & A_{\ell_1,\ell_1} & A_{\ell_1,\ell_1+1} & \cdots & A_{\ell_1,n} \\ A_{1,\ell_1+1}^T & A_{2,\ell_1+1}^T & \cdots & A_{\ell_1+1,\ell_1+1}^T & A_{\ell_1+1,\ell_1+1} & \cdots & A_{\ell_1+1,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ A_{1,n}^T & A_{2,n}^T & \cdots & A_{\ell_1,n}^T & A_{\ell_1+1,n}^T & \cdots & A_{n,n} \end{pmatrix}$$

$$\text{and } V[\vec{H}] := \begin{pmatrix} HL_1^1 \\ HL_2^1 \\ \cdots \\ HL_{\ell_3}^1 \\ KL_1^2 \\ \cdots \\ KL_{\ell_2}^2 \end{pmatrix}$$

for some $L_i^j, i = 1, \dots, \ell_j$. The $L_i^j, i = 1, \dots, \ell_j$ are called the *coefficients of the border vector*. The L_i^1 corresponding to H are distinct and only one may be the identity matrix (equivalently for the L_i^2 corresponding to K). The border vector V is the vector composed of H, K and L_i^j . The matrix M_Q is the matrix with $A_{i,j}$ entries.

Noncommutative quadratics which are *not hereditary* have a similar representation (which takes more space to write) for such a quadratic in H, K . For example, the border vector for a quadratic in H, H^T, K, K^T has the form

$$V[H, K] = (V_1 \quad V_2)$$

where we have

$$V_1 = ((L_1^1)^T H^T, \dots, (L_{\ell_1}^1)^T H^T, (L_1^2)^T K^T, \dots, (L_{\ell_2}^2)^T K^T)$$

and

$$V_2 = (\tilde{L}_1^1 H, \dots, \tilde{L}_{\ell_1}^1 H, \tilde{L}_1^2 K, \dots, \tilde{L}_{\ell_2}^2 K).$$

We should emphasize that the size of the M_Q representation of a noncommutative quadratic functions $Q[H_1, \dots, H_k]$ depends on the particular quadratic and not only on the number of arguments of the quadratic. There are noncommutative quadratic functions in one variable which have a representation with M_Q a 102×102 matrix.

The basic idea of `NCMatrixOfQuadratic` is that it searches for terms of form

$$Left ** X ** Middle ** Y ** Right$$

where $X = H_i$ or H_i^T and $Y = H_j$ or H_j^T for $1 \leq (i, j) \leq n$. Terms of the form $Left ** X$ and $Y ** Right$ are used to form the left and right vectors. Each time the search finds a unique *Right* (*Left*) term causes the length of the right (left) border vector to be increased by one. The term *Middle* becomes the entries in the matrix M_Q .

Comments / Limitations: `NCMatrixOfQuadratic` will try to symmetrize the resulting matrix M_Q . If `NCMatrixOfQuadratic` is unable to do this, an error message will be printed and `{leftvector, matrix, rightvector}` will be returned, where `matrix` is not symmetric and `leftvector` is not necessarily the transpose of `rightvector`. The vector-matrix-vector product should still be equal to the original quadratic expression.

1.3 NCIndependenceCheck[aListOfLists, variable]

Aliases: **None**.

Description: `NCIndependenceCheck[aListOfLists, variable]` is aimed at verifying whether or not a given set of polynomials are independent or not. It analyzes each list of polynomials in *aListOfLists* separately. There are three possible types of outputs for each list in *aListOfLists*. Two of them correspond to `NCIndependenceCheck` successfully determining whether or not the list of polynomials is independent. The third type of output corresponds to an unsuccessful attempt at determining dependence or independence. If a particular list is determined to be independent, *True* will be returned. If a list is determined to be dependent, a list beginning with *False* containing a set of coefficients which demonstrate independence will

be returned. Finally, if **NCIndependenceCheck** cannot determine dependence or independence, it returns a list beginning with *Undetermined* containing other information which is illustrated below and described further in Comments/Limitations.

Arguments: *aListofLists* is a list containing a list of the polynomials which are suspected of being dependent. The argument *variable* will be subscripted and used to return the coefficient dependencies for each list. Below is an example of a list of four lists. The first two are dependent, the third is independent, and the fourth is undetermined.

Suppose you have four lists:

$$\begin{aligned} List1 &= \{7, 6a, a, abd^2, d, b, 12a, d, 4a^2d, a^2, 5a^2, b^2, b\} \\ List2 &= \{50, 8a, a, abd^2, d, b, 12a, d, 4a^2d, a^2, 16a^2, 40b^2, b\} \\ List3 &= \{4a, 5b + c, c\} \\ List4 &= \{x ** y, y ** x\} \end{aligned}$$

Then **NCIndependenceCheck[List1, List2, List3, List4, λ]** returns $\{NewList1, NewList2, NewList3, NewList4\}$ where:

$$\begin{aligned} NewList1 &= \{False, \\ &\quad \{0, -\frac{\lambda_3}{6} - 2\lambda_7, \lambda_3, 0, -\lambda_8, -\lambda_{13}, \lambda_7, \lambda_8, -\frac{5\lambda_{11}}{4}, 0, \lambda_{11}, 0, \lambda_{13}\}\} \\ NewList2 &= \{False, \\ &\quad \{0, -\frac{\lambda_3}{8} - \frac{3\lambda_7}{2}, \lambda_3, 0, -\lambda_8, -\lambda_{13}, \lambda_7, \lambda_8, -4\lambda_{11}, 0, \lambda_{11}, 0, \lambda_{13}\}\} \\ NewList3 &= True \\ NewList4 &= \{Undetermined, -\lambda_2 x ** y + \lambda_2 y ** x, \{-\lambda_2, \lambda_2\}\} \end{aligned}$$

In particular, what the above says is that $List1.Newlist1[[2]] = 0$, and $List2.Newlist2[[2]] = 0$ (where “.” refers to the vector dot product). Therefore, the set of polynomials in *List1* and *List2* are dependent. *List3* is independent. Note that *List4* is clearly independent in the noncommuting case, and dependent in the commuting case. When such phenomena occur, **NCIndependenceCheck** is unable to determine whether or not the list of polynomials is independent. However, it does return to the user, a set of dependencies for the λ_i 's which must hold in order for the polynomials to sum to zero.

Comments / Limitations: *IndependenceCheck* first uses the *CommuteEverything* command to make the problem feasible. Therefore it is possible that polynomials are dependent if variables commute, and independent if not. So in this case, or when the the expression does not collapse to zero when using the commuting coefficients with the non commuting polynomials,

then the list $\{Undetermined, expression, list\}$ is returned. The list element *expression* is the sum of the polynomials with their corresponding λ 's. And finally, *list* yields a list of the dependencies for the coefficients.

1.4 NCBorderVectorGather[alist,varlist]

Aliases: **None**.

Description: `NCBorderVectorGather[alist,varlist]` can be used to gather the polynomial coefficients preceding the elements given in *varlist* whenever they occur in *alist*. That is to say, *alist* is a vector with variable entries. Each entry should end with some term from *varlist* (or the transpose of some term from *varlist*). Then for each element of *varlist* the coefficients that appear in front of that element in *alist* are gathered together and placed inside a list. The list returned will be a list of lists, each entry a list of the coefficients corresponding to the respective entries in *varlist* and their transposes if they occur.

Arguments: The first argument *alist* is a list of polynomials, all of which end in terms from elements of the second argument, *varlist*, or in their transpose. *alist* need not be ordered in a particular way with respect to *varlist*. The preceding is best explained in the following example.

Suppose *List* =

$$\{A**B**k, B**B**tp[h], B**tp[A]**k, B**C**tp[h], A**tp[h], B**h, C**h\}$$

Then `NCBorderVectorGather[List,{k,h}]` returns the following list

$$\{\{A**B, B**tp[A]\}, \{B, C\}, \{B**B, B**C, A\}\}$$

Note that the vectors are gather in the pattern $k, tp[k], h, tp[h]$. This pattern will be the same despite the length of *avarlist*.

Comments / Limitations: None.