SNEG

Mathematica package for calculations with non-commuting operators of the second quantization algebra

MANUAL

0.1 Introduction

The purpose of this manual is to describe the notations and conventions used in the SNEG library. The manual is rather abstract and concise, it focuses more on the design of the library than on real-world examples; it is not a tutorial. The examples in directory `examples/` should thus be studied in conjunction with reading this text. For details, one should consult the library source code, which is the only true reference.

0.2 Non-commuting multiplication

The cornerstone of SNEG is the non-commuting multiplication function `nc`. *Mathematica*'s built-in multiplication function `Times` has `Orderless` attribute, which corresponds to the mathematical property of commutativity, therefore it is unsuitable. Another built-in function, `NonCommutativeMultiply`, has `Flat` attribute, which corresponds to the mathematical property of associativity. Non-commuting multiplication should indeed be associative; unfortunately, `Flat` attribute also affects pattern matching, which proved to be undesirable for our purposes due to decreased computational performance.

An operator string is a sequence of second-quantization operators, such as $c^\dagger_{k\uparrow} c^\dagger_{k\downarrow} c_{k\downarrow} c_{k\uparrow}$. In SNEG, it would be represented as `nc[c[CR,k,UP], c[CR,k,DO], c[AN,k,DO], c[AN,k,UP]]` (see Section **??** on operators).

Function `nc` has the following basic properties:

- Associativity: for example, $\mathrm{nc}[a, \mathrm{nc}[b, c]] = \mathrm{nc}[a, b, c]$, where a, b and c are operators.
- Linearity: for example, $\mathrm{nc}[\alpha a + \beta b, c] = \alpha \mathrm{nc}[a, c] + \beta \mathrm{nc}[b, c]$, where α and β are numbers, and a, b and c are operators. Linearity implies distributivity. Furthermore, numerical objects are factored out from operator strings.
- Product of zero terms equals one, $\mathrm{nc}[] = 1$.
- Product of one term equals the term itself, $\mathrm{nc}[a] = a$.

In addition, fermionic operators are automatically normal ordered by anti-commuting the creation operators to the left and annihilation operators to the right using the default (canonical) or user-defined anti-commutation relations (see Section **??** on anti-commutation relations and operator ordering). This produces equivalent results for equivalent strings of operators and enables automatic expression simplifications. This property is the `nc` equivalent of argument sorting in the commuting multiplication function `Times`.

Dirac bras and kets (see Section **??** on Dirac notation) can also appear in `nc` multiplication expressions. They are automatically shifted to the right-most side of operator expressions. The default behavior is that bras and kets commute with operators, i.e. it is assumed that they belong to a different Hilbert space than the one where the second quantization operators operate.

0.3 Operators

In SNEG, operators are indexed objects, for example `c[t, sigma]`. The expression head, `c`, is the operator name, while `t, sigma` are indeces which indicate, for example, if the operator is a creation or an annihilation operator, and the associated degrees of freedom such as spin. The operator character of `c` must be declared using the `snegfermionoperators` function: for example `snegfermionoprators[c]`.

By convention, the first index indicates if the operator creates or annihilates a particle. Two constants are predefined for this purpose: `CR=0` and `AN=1`. Another convention is that the spin index in the case of $S = 1/2$ operators is on the last position. For convenience, two further constants are predefined: `UP=1` and `DO=0`.

For a single orbital, the creation operators are thus `c[CR, UP]` and `c[CR, DO]`, while the corresponding annihilation operators are `c[AN, UP]` and `c[AN, DO]`. (Conjugation can be performed using the function `conj`.)

0.4 Numbers and numeric expressions

SNEG must be able to differentiate operators from numbers and other numeric expressions in order to factor out non-operator parts of operator expressions. Using `snegrealconstants[x1, x2,...]`, x1, x,... are defined to be real constants (invariant under conjugation). Using `snegcomplexconstants[z1, z2,...]`, z1, z2,... are defined to be complex constants. Finally, using `snegfreeindexes[k, sigma,...]` we notify SNEG that `k`, `sigma`,... are indexes: this is required to factor out numerical expressions such as `KroneckerDelta[k1,k2]` out of operator strings.

0.5 Anti-commutation relations and operator ordering

Two properties of operators must be known to SNEG to perform automatic reorderings and simplifications. These are anti-commutation relations and the vacuum state. The first affects how the transpositions of operators are performed, while the latter determines the conventional ordering of operators in operator strings.

The default anti-commutation relation for fermionic operators are the usual cannonical anti-commutation relations, $\{c^\dagger_\alpha, c_\beta\} = \delta_{\alpha\beta}$, $\{c^\dagger_\alpha, c^\dagger_\beta\} = 0$, and $\{c_\alpha, c_\beta\} = 0$. Operators with different heads simply anti-commute. The value of the $\{c^\dagger_\alpha, c_\beta\}$ anti-commutator is defined by the funciton `acmt[c, {indexes1}, {indexes2}]`. The default behaviour is to literally compare indexes1 and indexes2: 1 is returned if they are equal, and 0 otherwise.

`ordering[c]=SEA`, `ordering[c]=EMPTY`.

0.6 Expression-building functions

SNEG includes several functions that can be used to build operator expressions such as occupation number, spin operator, spin-spin scalar product, charge-charge repulsion, etc.

Unless otherwise specified, parameters to such functions are operators with their type (`CR` or `AN`) and spin indeces removed, for example `c[]`. To build the occupancy (number) operator for orbital c, we therefore call `number[c[]]`, which returns an expression equivalent to $\sum_\sigma c^\dagger_\sigma c_\sigma$. The sum over spin is automatically performed. Additional indexes are given as arguments to `c[]`, for example `number[c[k]]` gives an expression corresponding to $\sum_\sigma c^d a g_{k\sigma} c_{k\sigma}$, i.e. the occupancy operator for a state with wavenumber k.

Functions in this group are:

- `number[c[], sigma]` - number of particles with spin `sigma` in orbital c, i.e. $n_\sigma = c^\dagger_\sigma c_\sigma$.
- `number[c[]]` - charge density (number of particles), i.e. $n = n_\uparrow + n_\downarrow$.
- `spinx[c[]]`, `spiny[c[]]`, `spinz[c[]]` - spin density, i.e. $\mathbf{S} = \sum_{\alpha\beta} c^\dagger_\alpha (1/2\boldsymbol{\sigma}) c_\beta$, where α and β are spin indexes and $\boldsymbol{\sigma}$ is the vector of Pauli matrices.
- `spinplus[c[]]` and `spinminus[c[]]` - spin raising and spin lowering operators, $S^+ = S_x + IS_y$ and $S^- = S_x - IS_y$.
- `spinss[c[]]` - the total spin operator squared, $\mathbf{S}^2$.
- `spinspin[c[1], c[2]]` - scalar product of two spin operators for different (or equal) orbitals, $\mathbf{S}_1 \cdot \mathbf{S}_2$. Transverse and longitudinal parts of the scalar product are obtained using `spinspinxy[c[1], c[2]]` and `spinspinz[c[1], c[2]]`.
- `hubbard[c[]]` - Hubbard's local electron-electron repulstion operator, $n_\uparrow n_\downarrow$.
- `nambu[c[]]` - Nambu spinor, $\eta = \{c^\dagger_\uparrow, (-1)^n c_\downarrow\}$.
- `isospin[c[]]` - isospin operator, $\mathbf{I} = \sum_{\alpha\beta} \eta_\alpha (1/2\boldsymbol{\sigma}) \eta_\beta$, where η is the Nambu spinor, α and β are isospin indexes and $\boldsymbol{\sigma}$ is the vector of Pauli matrices.

- `hop[c[1], c[2]]` - electron hopping operator $\sum_\sigma (c^\dagger_{1\sigma} c_{2\sigma} + c^\dagger_{2\sigma} c_{1\sigma})$.
- `twohop[c[1], c[2]]` - two-electron hopping operator $c^\dagger_{1\downarrow} c^\dagger_{1\uparrow} c_{2\downarrow} c_{2\uparrow}$ + H.c..
- `projection[c[], pr]` - projection operator. `pr = PROJ0 | PROJUP | PROJDO | PROJ2 | PROJ1 | PROJ02`; `PROJ0` projects to zero-occupancy states, $(1-n_\uparrow)(1-n_\downarrow)$, `PROJUP` to spin-up states, $n_\uparrow(1-n_\downarrow)$, `PROJDO` to spin-down states, $n_\downarrow(1-n_\uparrow)$, `PROJ2` to double-occupancy states, $n_\uparrow n_\downarrow$, `PROJ1` to single-occupancy states, and `PROJ02` to zero- and double-occupancy states.

0.7 Expression manipulation functions

Function `invertspin` inverts the spins of all operators appearing in an expression. The convention that the spin index appears at the last position must be followed.

Function `conj` calculates the Hermitian conjugate of an expression.

Function `vev` calculates the vacuum expectation value of an operator expression; `vevwick` does the same using Wick's theorem.

Function `wick` writes an operator string using normal ordered strings of operators and contractions (Wick's theorem).

`komutator[a,b]` calculates the commutator $[a, b]$, while `antikomutator[a,b]` calculates the the anti-commutator $\{a, b\}$.

0.8 Dirac's bra and ket notation

Dirac's bras $\langle i, j, \ldots |$ and kets $|i, j, \rangle$ are represented by expressions `bra[i,j,...]` and `ket[i,j,...]`, where `i`, `j`,... denote the quantum numbers that determine the corresponding state.

By default, the values of brakets are determined using the Kronecker's delta, i.e. $\langle i_1, j_1, \ldots | i_2, j_2, \ldots = \delta_{i_1 i_2} \delta_{j_1 j_2} \ldots$.

Bras and kets may appear in `nc` strings. They do not commute, i.e. the braket $\langle i|j\rangle$ is different from the projector $|j\rangle\langle i|$.

Different quantum numbers correspond to different argument positions. `Null` placeholders can be used in place of unspecified quantum numbers. Neighboring bras and kets are "concatenated" if they have compatible patterns of `Null` placeholders. For example: `nc[ket[i, Null], ket[Null,j]]=ket[i,j]`. From the mathematical point of view, this corresponds to a direct (tensor) product of states.

0.9 VACUUM vector

Keyword `VACUUM` corresponds to a vacuum state. By default, if an annihilation operator is applied to `VACUUM` from the left, the result is 0. Correspondingly, if a creation operator is applied to `conj[VACUUM]` from the right, the result is 0.

By definition, `nc[conj[VACUUM], VACUUM]=1`.

0.10 State vectors: operator and occupation number representations

A state can be represented either as the effect of string of creation operators on a (unspecified) vacuum state, or as a set of occupation numbers. If the vacuum is known, a one-to-one mapping between the two representations can be established.

In the creation operator representation, a state is just an operator expression. For example, a one-particle state $1/\sqrt{2}(c^\dagger_{1\uparrow} + c^\dagger_{2\uparrow})|0\rangle$ would be described as `1/Sqrt[2] (c[CR,1,UP]+c[CR,2,UP])`, a two-particle state $c^\dagger_{1\uparrow} c^\dagger_{1\downarrow}$ as `nc[c[CR,1,UP], c[CR,1,DO]]`, etc.

In the occupation number representation, a state is given by a vector of occupation numbers, `vc[0,1,0,0,1,0,...]`. Each position in the vector corresponds to some orbital with a given spin.

A mapping between operators and positions in the vector is established using `makebasis` function. For two orbitals, for example, the mapping is defined by `makebasis[c[1], c[2]]`. Position 1 then corresponds to $c^\dagger_{1\uparrow}$, position 2 to $c^\dagger_{1\downarrow}$, position 3 to $c^\dagger_{2\uparrow}$, and position 4 to $c^\dagger_{2\downarrow}$. The mapping can be accessed as the global variable `BASIS`, or using `op2ndx` and `ndx2op` mapping functions.

The zero-particle vacuum can be obtained using `vacuum`. Operator expressions can be applied to occupation number vectors using `ap`. One can go from creation operator representation to occupation number representation and back using `ops2vc` and `vc2ops` mapping functions.

Scalar products of vectors are given by `scalarproductop[a,b]` and `scalarproductvc[a,b]`. Norms can be calculated using `normop[a]` and `normvc[a]`.

An operator given by a second-quantisation operator expression `O` can be transformed using `matrixrepresentationop[O, l]` and `matrixrepresentationvc[O, l]` to its matrix representation in a subspace spanned by states given in a list `l`.

A vector can be decomposed in a given subspace using `decomposeop[v, l]` and `decomposevc[v, l]`.

A set of vectors can be orthogonalized in a given subspace using `orthogop[vs, l]` and `orthogvc[vs, l]`, where `vs` are the vectors and `l` is a list of vectors spanning the subspace.

0.11 Basis sets

In SNEG, basis sets are lists of subspace basis sets. A subspace basis set consists of a list of invariant quantum numbers that characterize the invariant subspace and a list of states in either creation operator or occupation number representation.

Function `qszbasis[l]` constructs a basis set with subspaces with well-defined charge, Q, and spin projection, S_z, quantum numbers. List `l` is the list of operators that defined the orbitals; it is usually the same list as the one passed as the argument to `makebasis`.

Function `qsbasis[l]` constructs a basis set with subspaces with well-defined charge, Q, and total spin, S, quantum numbers.

Conversions of basis sets from creation operator to occupation number representation can be performed using `bzvc2bzop` and `bzop2bzvc` functions.

0.12 Symbolic sums

`sum[expr, {k, sigma,...}]`