Snob2

Risi Kondor

Oct 31, 2021

CONTENTS:

Snob2 is a C++ package with a Python interface for computing the representations of the symmetric group $\mathbb{S}_n$ and computing fast Fourier transforms on $\mathbb{S}_n$.

Snob2 is built on the **cnine** library which can be downloaded from https://github.com/risi-kondor/cnine. GPU functionality for the library is not yet available but is under development.

Snob2 is written by Risi Kondor at the University of Chicago and is released under the Mozilla public license v.2.0.

This document provides documentation for Snob2's Python interface. Not all features in the C++ library are available through this interface. The documentation of the C++ API can be found in pfd format in the package's `doc` directory.

CHAPTER

ONE

FEATURES

- Custom classes for combinatorial objects such as integer partitions and Young tableau implemented in C++ for efficiency.
- Classes for the symmetric group, conjugacy classes, quotient spaces, characters and irreducible representations of $\mathbb{S}_n$.
- Classes for functions on $\mathbb{S}_n$ and quotient spaces of $\mathbb{S}_n$.
- Classes implementing Clausen's FFT algorithm for the forward and backward Fourier transform on $\mathbb{S}_n$ and its quotient spaces.

CHAPTER TWO

INSTALLATION

Installing Snob2 requires the following:

1. A C++ installation with C++11 or higher
2. Python

To install Snob2 follow these steps:

1. Download the cnine and Snob2 packages.
2. Edit the file `options.txt`, in particular, make sure that `CNINE_ROOT` points to the root of the **cnine** package on your system.
3. Run `python setup.sty install` in the `pytorch` directory to compile the package and install it on your system.

To use Snob2, issue the command `import Snob2` in Python. This loads to the *Snob2* module and initializes the various static datastructures used by the package.

CHAPTER

THREE

DESIGN

The representation theory of $\mathbb{S}_n$ involves some data structures that are relatively expensive to compute but only needed to be computed once. Snob2's backend automatically caches these data. For example, the class `IntegerPartitions` returns all integer partitions of an integer n. The first time that an `IntegerPartitions` object is created for a given value of n, Snob2 constructs the integer partitions from the integer partitions of $m < n$ and stores the result in a static object so that on subsequent calls the same process does not need to be repeated.

CHAPTER

FOUR

CLASSES

4.1 Combinatorial classes

Snob provides specialized classes to represent various combinatorial objects involved in the representation theory of $\mathbb{S}_n$.

4.1.1 Permutations

A permutation π of n is a bijective map $\{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$.

```
>>> pi=Snob2.Permutation([2,3,1,5,4])
>>> print(pi)
[ 2 3 1 5 4 ]
>>> pi[3]
1
```

The product of two permutations τ and π is the permutation corresponding to the composition of maps $\tau \circ \pi$.

```
>>> tau=Snob2.Permutation([2,1,3,4,5])
>>> print(tau*pi)
[ 1 3 2 5 4 ]
```

The *inv* method returns the inverse of a permutation.

```
>>> print(pi.inv())
[ 3 1 2 5 4 ]
```

4.1.2 Integer partitions

An *integer partition* of a positive integer n is a vector of positive integers $\lambda = (\lambda_1, \ldots, \lambda_k)$ such that $\sum_{i=1}^{k} \lambda_i = n$ and $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_k$. The `IntegerPartition` class represents such vectors.

```
>>> a=Snob2.IntegerPartition([3,2,1])
>>> print(a)
[3,2,1]
>>> print(a[1])
2
```

The `IntegerPartitions` class returns an object that contains *all* integer partitions of math:*n*.

```
>>> Lambda=Snob2.IntegerPartitions(5)
>>> for i in range(len(Lambda)):
...      print(Lambda[i])
...
[5]
[4,1]
[3,2]
[3,1,1]
[2,2,1]
[2,1,1,1]
[1,1,1,1,1]
```

4.1.3 Young tableau

A Young tableau is a Young diagram whose cells are filled with integers. The default Young tableau of a given shape is the one where the numbers $1, 2, \ldots, n$ appear sequentially.

```
>>> T=Snob2.YoungTableau(Snob2.IntegerPartition([3,2,1]))
>>> print(T)
1 2 3
4 5
6
>>> print(T.shape())
[3,2,1]
>>>
```

A *standard Young tableau* is a Young tableau filled with the numbers $1, 2, \ldots, n$ in such a way that in any row the numbers increase from left to right and in any column the numbers increase top to bottom. The `StandardYoungTableaux` class returns an object that contains *all* standard Young tableaux of a given shape.

```
>>> lamb=Snob2.IntegerPartition([3,2])
>>> T=Snob2.StandardYoungTableaux(lamb)
>>> for i in range(len(T)):
...     print(T[i])
...
1 2 3
4 5

1 2 4
3 5

1 3 4
2 5

1 2 5
3 4

1 3 5
2 4
```

4.2 Symmetric group classes

The symmetric group $\mathbb{S}_n$, represented by the class Sn, is the group of all permutations of $\{1, 2, \dots, n\}$.

```
>>> G=Snob2.Sn(4)
>>> for i in range(len(G)):
...     print(G.element(i))
...
[ 1 2 3 4 ]
[ 2 1 3 4 ]
[ 1 3 2 4 ]
[ 2 3 1 4 ]
[ 3 1 2 4 ]
[ 3 2 1 4 ]
[ 1 2 4 3 ]
[ 2 1 4 3 ]
[ 1 3 4 2 ]
[ 2 3 4 1 ]
[ 3 1 4 2 ]
[ 3 2 4 1 ]
[ 1 4 2 3 ]
[ 2 4 1 3 ]
[ 1 4 3 2 ]
[ 2 4 3 1 ]
[ 3 4 1 2 ]
[ 3 4 2 1 ]
[ 4 1 2 3 ]
[ 4 2 1 3 ]
[ 4 1 3 2 ]
[ 4 2 3 1 ]
[ 4 3 1 2 ]
[ 4 3 2 1 ]
```

4.2.1 Group elements

The group elements of $\mathbb{S}_n$ are of type `SnElement`, which have the same methods as the `Permutation` class. Group elements are listed in a specific reverse insertion sort order that fits the structure of $\sigma\mathbb{S}_m$ cosets and hence is well adapted to Clausen-type fast Fourier transforms on $\mathbb{S}_n$. The following shows how to extract the i'th group element and how to get the index of a particular group element.

```
>>> pi=G[17]
>>> print(pi)
[ 3 4 2 1 ]
>>> print(G.index(pi))
17
```

4.2.2 Conjugacy classes

The conjugacy classes of $\mathbb{S}_n$ are in bijection with the integer partitions of n. Snob2 has a separate class `SnCClass` to represent conjugacy classes. `SnCClass` objects can be constructed either from the group object or directly from an integer partition.

```
>>> G=Snob2.Sn(5)
>>> mu=Snob2.IntegerPartition([3,2])
>>> cc=G.cclass(mu)
>>> print(cc)
SnCClass[3,2]
```

The conjugacy classes are ordered according to majorization order of their integer partitions. The `Sn.index` method returns the index of a given conjugacy class.

```
>>> G.index(cc)
2
```

4.2.3 Characters

The characters of $\mathbb{S}_n$ are also indexed by the integer partitions of n and can be accessed through the `character` method of `Sn`.

```
>>> G=Snob2.Sn(5)
>>> lambd=Snob2.IntegerPartition([3,2])
>>> chi=G.character(lambd)
>>> print(chi)
SnCClass[5] : 0
SnCClass[4,1] : -1
SnCClass[3,2] : 1
SnCClass[3,1,1] : -1
SnCClass[2,2,1] : 1
SnCClass[2,1,1,1] : 1
SnCClass[1,1,1,1,1] : 5
```

4.2.4 Irreducible representations

The irreducible representations (irreps) of $\mathbb{S}_n$ are captured by `SnIrrep` objects. For a given integer partition λ of n, the corresponding irrep can be constructed from the group object or directly from the integer partition.

```
>>> lambd=Snob2.IntegerPartition([3,1])
>>> rho=G.irrep(lambd)
>>> print(rho)
SnIrrep([3,1])

>>> lambd=Snob2.IntegerPartition([3,1])
>>> rho=Snob2.SnIrrep(lambd)
>>> print(rho)
SnIrrep([3,1])
```

The dimension of the irrep is accessible through the *get_dim()* method.

```
>>> print(rho.get_dim())
3
```

All irreps in Snob2 are expressed in Young's orthogonal representation. The representation matrices are easy to access.

```
>>> pi=Snob2.SnElement([3,2,1,4])
>>> print(rho[pi])
[ 1 0 0 ]
[ -0 -0.5 -0.866025 ]
[ -0 -0.866025 0.5 ]
```

4.2.5 Sn types

The *type* of a representation is an associative list of integer partitions and associated multiplicities describing what irreps a particular representation is composed of. The following shows how to set up an `SnType` object.

```
>>> tau=Snob2.SnType(Snob2.IntegerPartition([4,1]),2)
>>> tau.set(Snob2.IntegerPartition([3,2]),1)
>>> tau.set(Snob2.IntegerPartition([3,1,1]),1)
>>> print(tau)
([4,1]:2,[3,2]:1,[3,1,1]:1)
```

4.3 Sn-functions and Sn-vectors

4.3.1 Sn functions

The class `SnFunction` represents functions on $\mathbb{S}_n$. The following initializes a function on $\mathbb{S}_3$ with random Gaussian entries and prints it out.

```
>>> f=Snob2.SnFunction.gaussian(3)
>>> print(f)
[ 1 2 3 ] : -1.23974
[ 2 1 3 ] : -0.407472
[ 1 3 2 ] : 1.61201
[ 2 3 1 ] : 0.399771
[ 3 1 2 ] : 1.3828
[ 3 2 1 ] : 0.0523187
```

The value of `f` at specific group elements can be accessed via the `SnElement` object or just its index.

```
>>> f[Snob2.SnElement([1,3,2])]
1.6120094060897827
0.1949467808008194
>>> f[2]
1.6120094060897827
```

The *left-translate* of f by a permutation π is defined $g_1(\sigma) = f(\pi^{-1}\sigma)$.

```
>>> g1=f.left_translate(pi)
SnFunction moved
>>> print(g1)
[ 1 2 3 ] : -0.407472
[ 2 1 3 ] : -1.23974
[ 1 3 2 ] : 0.399771
[ 2 3 1 ] : 1.61201
[ 3 1 2 ] : 0.0523187
[ 3 2 1 ] : 1.3828
```

The *right-translate* of f by a permutation π is defined $g_1(\sigma) = f(\sigma\pi^{-1})$.

```
>>> g2=f.right_translate(pi)
SnFunction moved
>>> print(g2)
[ 1 2 3 ] : -0.407472
[ 2 1 3 ] : -1.23974
[ 1 3 2 ] : 1.3828
[ 2 3 1 ] : 0.0523187
[ 3 1 2 ] : 1.61201
[ 3 2 1 ] : 0.399771
```

The *inverse* of f is defined $f^{-1}(\sigma) = f(\sigma)$.

```
>>> f=Snob2.SnFunction.gaussian(3)
>>> finv=f.inv()
>>> print(finv)
[ 1 2 3 ] : -1.23974
[ 2 1 3 ] : -0.407472
[ 1 3 2 ] : 1.61201
[ 2 3 1 ] : 1.3828
[ 3 1 2 ] : 0.399771
[ 3 2 1 ] : 0.0523187
```

4.3.2 Sn/Sm functions

The class `SnOverSmFunction` represents functions on $\mathbb{S}_n/\mathbb{S}_m$. The following initializes a function on $\mathbb{S}_5/\mathbb{S}_4$ with random Gaussian entries and prints it out.

```
>>> f=Snob2.SnOverSmFunction.gaussian(5,4)
>>> print(f)
0.74589
-1.75177
-0.965146
-0.474282
-0.546571
```

4.3.3 Sn class functions

The class SnClassFunction represents functions on the conjugacy classes of $\mathbb{S}_n$. An important example of class functions are the characters of the group. The following initializes a class function on $\mathbb{S}_4$ with random Gaussian entries and prints it out.

```
>>> f=Snob2.SnClassFunction.gaussian(4)
>>> print(f)
SnCClass[4] : -1.23974
SnCClass[3,1] : -0.407472
SnCClass[2,2] : 1.61201
SnCClass[2,1,1] : 0.399771
SnCClass[1,1,1,1] : 1.3828
```

The value of f at specific conjugacy classes can be accessed via the corresponding SnCClass, IntegerPartition or just the index.

```
>>> f[Snob2.SnCClass([2,2])]
1.6120094060897827
>>> f[Snob2.IntegerPartition([2,2])]
1.6120094060897827
>>> f[Snob2.SnCClass(2)]
1.6120094060897827
```

4.3.4 Sn parts

An SnPart of type λ is a collection of m vectors on which acts by the irreducible representation ρ_λ. The SnPart is stored as a matrix $\mathbb{R}^{d_\lambda \times m}$.

```
>>>lambd=Snob2.IntegerPartition([3,2])
>>> p=Snob2.SnPart.gaussian(lambd,3)
>>> print(p)
Part [3,2]:
[ -1.23974 -0.407472 1.61201 ]
[ 0.399771 1.3828 0.0523187 ]
[ -0.904146 1.87065 -1.66043 ]
[ -0.688081 0.0757219 1.47339 ]
[ 0.097221 -0.89237 -0.228782 ]
```

4.3.5 Sn vectors

An Sn covariant vector or *Sn-vector* for short is a vector that transforms under the action of $\mathbb{S}_n$ by a combination of irreducible representations. Sn-vectors are stored as SnVec objects as a list of ``SnPart``s.

```
>>> tau=Snob2.SnType(Snob2.IntegerPartition([4,1]),2)
>>> tau.set(Snob2.IntegerPartition([3,2]),1)
>>> tau.set(Snob2.IntegerPartition([3,1,1]),1)
>>> v=Snob2.SnVec.gaussian(tau)
>>> print(v)
Part [4,1]:
[ -1.23974 -0.407472 ]
```

(continues on next page)

(continued from previous page)

```
[ 1.61201 0.399771 ]
[ 1.3828 0.0523187 ]
[ -0.904146 1.87065 ]

Part [3,2]:
[ -1.66043 ]
[ -0.688081 ]
[ 0.0757219 ]
[ 1.47339 ]
[ 0.097221 ]

Part [3,1,1]:
[ -0.228782 ]
[ 1.16493 ]
[ 0.584898 ]
[ -0.660558 ]
[ 0.534755 ]
[ -0.607787 ]
```

4.4 Sn Fourier transforms

The Fourier transform on $\mathbb{S}_n$ converts a function on $\mathbb{S}_n$ or a qutient space of $\mathbb{S}_n$ into an $\mathbb{S}_n$–vector. Snob2 uses Clausen's FFT to compute forward and backward Fourier transforms. Fourier transforms employs several internal data structures that can be reused on future transforms. Therefore before conducting a Fourier transform a corresponding `ClausenFFT` must be constructed.

4.4.1 FFTs on Sn

The following sets up an `ClausenFFT` object for Fourier transformation on $\mathbb{S}_4$ and defines a random function on the group.

```
>>> fft=Snob2.ClausenFFT(4)
>>> f=Snob2.SnFunction.gaussian(4)
>>> print(f)
[ 1 2 3 4 ] : -1.23974
[ 2 1 3 4 ] : -0.407472
[ 1 3 2 4 ] : 1.61201
[ 2 3 1 4 ] : 0.399771
[ 3 1 2 4 ] : 1.3828
[ 3 2 1 4 ] : 0.0523187
[ 1 2 4 3 ] : -0.904146
[ 2 1 4 3 ] : 1.87065
[ 1 3 4 2 ] : -1.66043
[ 2 3 4 1 ] : -0.688081
[ 3 1 4 2 ] : 0.0757219
[ 3 2 4 1 ] : 1.47339
[ 1 4 2 3 ] : 0.097221
[ 2 4 1 3 ] : -0.89237
[ 1 4 3 2 ] : -0.228782
```

(continues on next page)

(continued from previous page)

```
[ 2 4 3 1 ] : 1.16493
[ 3 4 1 2 ] : 0.584898
[ 3 4 2 1 ] : -0.660558
[ 4 1 2 3 ] : 0.534755
[ 4 2 1 3 ] : -0.607787
[ 4 1 3 2 ] : 0.74589
[ 4 2 3 1 ] : -1.75177
[ 4 3 1 2 ] : -0.965146
[ 4 3 2 1 ] : -0.474282
```

We can now use our fft object to take the Fourier transform of f.

```
>>> F=fft(f)
>>> print(F)
Part [4]:
[ -0.486197 ]

Part [3,1]:
[ 2.56166 1.21663 -0.41762 ]
[ 1.3139 -1.81861 3.2474 ]
[ 2.10957 -3.31125 -5.47569 ]

Part [2,2]:
[ -3.05059 -2.65296 ]
[ 1.56762 1.53786 ]

Part [2,1,1]:
[ -5.13609 4.39341 1.45563 ]
[ 3.59791 2.07342 -0.436283 ]
[ -3.65454 0.381513 -3.60564 ]

Part [1,1,1,1]:
[ 7.96084 ]
```

The inverse Fourier transform can be computed with the same FFT object and should return the original function `f`.

```
>>> fd=fft.inv(F)
>>> print(fd)
[ 1 2 3 4 ] : -1.23974
[ 2 1 3 4 ] : -0.407472
[ 1 3 2 4 ] : 1.61201
[ 2 3 1 4 ] : 0.399771
[ 3 1 2 4 ] : 1.3828
[ 3 2 1 4 ] : 0.0523185
[ 1 2 4 3 ] : -0.904147
[ 2 1 4 3 ] : 1.87065
[ 1 3 4 2 ] : -1.66043
[ 2 3 4 1 ] : -0.688081
[ 3 1 4 2 ] : 0.0757219
[ 3 2 4 1 ] : 1.47339
[ 1 4 2 3 ] : 0.097221
[ 2 4 1 3 ] : -0.89237
```

(continues on next page)

(continued from previous page)

```
[ 1 4 3 2 ] : -0.228782
[ 2 4 3 1 ] : 1.16493
[ 3 4 1 2 ] : 0.584898
[ 3 4 2 1 ] : -0.660558
[ 4 1 2 3 ] : 0.534755
[ 4 2 1 3 ] : -0.607787
[ 4 1 3 2 ] : 0.74589
[ 4 2 3 1 ] : -1.75177
[ 4 3 1 2 ] : -0.965146
[ 4 3 2 1 ] : -0.474282
```

4.4.2 FFTs on Sn/Sm

The `ClausenFFT` can also be used to compute FFTs on $\mathbb{S}_n/\mathbb{S}_m$.

```
>>> fft=Snob2.ClausenFFT(4,2)
>>> f=Snob2.SnOverSmFunction.gaussian(4,2)
>>> print(f)
-0.546571
-0.0384917
0.194947
-0.485144
-0.370271
-1.12408
1.73664
0.882195
-1.50279
0.570759
-0.929941
-0.934988

>>> F=fft(f)
>>> print(F)
Part [4]:
[ -2.54774 ]

Part [3,1]:
[ 0.329091 3.59416 ]
[ -1.78231 0.663375 ]
[ 1.96793 1.63815 ]

Part [2,2]:
[ -3.93037 ]
[ -1.41466 ]

Part [2,1,1]:
[ 0.290743 ]
[ -1.23415 ]
[ -1.32773 ]
```

```
>>> fd=fft.inv_snsm(F)
SnFunction moved
>>> print(fd)
[ 1 2 3 ] : -0.546571
[ 2 1 3 ] : -0.0384917
[ 1 3 2 ] : 0.194947
[ 2 3 1 ] : -0.485144
[ 3 1 2 ] : -0.37027
[ 3 2 1 ] : -1.12408
[ 32705 1 2 ] : 1.73664
[ 3 2 1 ] : 0.882195
[ 3 1 2 ] : -1.50279
[ 519242688 2 1 ] : 0.570759
[ 3 1 2 ] : -0.929941
[ 3 2 1 ] : -0.934988
```

CHAPTER

FIVE

REFERENCE

5.1 Combinatorial classes

class IntegerPartition(*parts*)

Class to represent an integer partition (p_1,p_2,. . . ,p_k) of n.

__getitem__(*self: Snob2.IntegerPartition*, *arg0: int*) → int

Return the the i'th part, p_i.

__init__(*self: Snob2.IntegerPartition*, *arg0: List[int]*) → None

Initialize from a list of integers.

__setitem__(*self: Snob2.IntegerPartition*, *arg0: int*, *arg1: int*) → None

Set the i'th part to x

__str__(*self: Snob2.IntegerPartition*, *indent: str = ''*) → str

Print the integer partition to string.

getn(*self: Snob2.IntegerPartition*) → int

Return n.

height(*self: Snob2.IntegerPartition*) → int

Return the number of parts, k.

class IntegerPartitions(*n*)

This object represents all integer partitions of a given integer n.

__getitem__(*self: Snob2.IntegerPartitions*, *arg0: int*) → *Snob2.IntegerPartition*

Return the i'th integer partition.

__init__(*self: Snob2.IntegerPartitions*, *arg0: int*) → None

Create an object to represent all integer partitions of n.

__len__(*self: Snob2.IntegerPartitions*) → int

Return the number of integer partition.

at(*self: Snob2.IntegerPartitions*, *arg0: int*) → *Snob2.IntegerPartition*

Return the i'th integer partition.

class YoungTableau

`__init__`(*self: Snob2.YoungTableau*, *arg0: Snob2.IntegerPartition*) → None
Return a tableau of the given shape filled with 1,. . . ,n

`__str__`(*self: Snob2.YoungTableau*, *indent: str = ''*) → str
Print the tableau to string.

`at`(*self: Snob2.YoungTableau*, *arg0: int*, *arg1: int*) → int
Return the integer at position (i,j) in the tableau.

`getk`(*self: Snob2.YoungTableau*) → int
Return the number of rows.

`shape`(*self: Snob2.YoungTableau*) → *Snob2.IntegerPartition*
Return the integer partition describing the shape of this tableau.

class Permutation

Class to represent a permutation sigma of (1,2,. . . ,n)

`__eq__`(*self: Snob2.Permutation*, *arg0: Snob2.Permutation*) → bool

`__getitem__`(*self: Snob2.Permutation*, *arg0: int*) → int
Return sigma(i)

`__imul__`(*self: Snob2.Permutation*, *arg0: Snob2.Permutation*) → *Snob2.Permutation*

`__init__`(**args*, ***kwargs*)
Overloaded function.

1. __init__(self: Snob2.Permutation, arg0: int) -> None
2. __init__(self: Snob2.Permutation, arg0: int, arg1: cnine::fill_raw) -> None
3. __init__(self: Snob2.Permutation, arg0: int, arg1: cnine::fill_identity) -> None

Return the identity permutation on n items

4. __init__(self: Snob2.Permutation, arg0: List[int]) -> None

Initialize the permutation from a list.

`__inv__`(*self: Snob2.Permutation*) → *Snob2.Permutation*

`__mul__`(*self: Snob2.Permutation*, *arg0: Snob2.Permutation*) → *Snob2.Permutation*

`__str__`(*self: Snob2.Permutation*, *indent: str = ''*) → *str*

`getn`(*self: Snob2.Permutation*) → int
Return n.

static `identity`(*arg0: int*) → *Snob2.Permutation*
Return the identity permutation of n.

`inv`(*self: Snob2.Permutation*) → *Snob2.Permutation*
Return the inverse permutation.

str(*self: Snob2.Permutation*, *indent: str = ''*) → *str*

5.2 Symmetric group classes

class Sn(*n*)

__getitem__(*self: Snob2.Sn*, *arg0: int*) → *Snob2.SnElement*

__init__(*self: Snob2.Sn*, *arg0: int*) → None

__len__(*self: Snob2.Sn*) → int

__str__(*self: Snob2.Sn*, *indent: str = ''*) → *str*

cclass(**args*, ***kwargs*)

Overloaded function.

1. cclass(self: Snob2.Sn, arg0: int) -> Snob2.SnCClass
2. cclass(self: Snob2.Sn, arg0: Snob2.IntegerPartition) -> Snob2.SnCClass

cclass_size(**args*, ***kwargs*)

Overloaded function.

1. cclass_size(self: Snob2.Sn, arg0: Snob2.IntegerPartition) -> int
2. cclass_size(self: Snob2.Sn, arg0: Snob2.SnCClass) -> int

character(*self: Snob2.Sn*, *arg0: Snob2.IntegerPartition*) → Snob2.SnClassFunction

element(*self: Snob2.Sn*, *arg0: int*) → *Snob2.SnElement*

getn(*self: Snob2.Sn*) → int

identity(*self: Snob2.Sn*) → *Snob2.SnElement*

index(**args*, ***kwargs*)

Overloaded function.

1. index(self: Snob2.Sn, arg0: Snob2.SnElement) -> int
2. index(self: Snob2.Sn, arg0: Snob2.IntegerPartition) -> int
3. index(self: Snob2.Sn, arg0: Snob2.SnCClass) -> int

irrep(*self: Snob2.Sn*, *arg0: Snob2.IntegerPartition*) → *Snob2.SnIrrep*

ncclasses(*self: Snob2.Sn*) → int

nchars(*self: Snob2.Sn*) → int

order(*self: Snob2.Sn*) → int

random(*self: Snob2.Sn*) → *Snob2.SnElement*

str(*self: Snob2.Sn*, *indent: str = ''*) → *str*

class SnElement

Class to represent symmetric group elements.

__eq__(*self: Snob2.SnElement*, *arg0: Snob2.Permutation*) → bool

__getitem__(*self: Snob2.SnElement*, *arg0: int*) → int

__imul__(*self: Snob2.SnElement*, *arg0: Snob2.Permutation*) → *Snob2.Permutation*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: Snob2.SnElement, arg0: int) -> None
2. __init__(self: Snob2.SnElement, arg0: int, arg1: cnine::fill_raw) -> None
3. __init__(self: Snob2.SnElement, arg0: int, arg1: cnine::fill_identity) -> None
4. __init__(self: Snob2.SnElement, arg0: List[int]) -> None
5. __init__(self: Snob2.SnElement, arg0: Snob2.Permutation) -> None

__inv__(*self: Snob2.SnElement*) → *Snob2.SnElement*

__mul__(*self: Snob2.SnElement*, *arg0: Snob2.SnElement*) → *Snob2.SnElement*

__str__(*self: Snob2.SnElement*, *indent: str = ''*) → str

getn(*self: Snob2.SnElement*) → int

Return n.

static identity(*arg0: int*) → *Snob2.SnElement*

Return the identity element of Sn.

inv(*self: Snob2.SnElement*) → *Snob2.SnElement*

class SnCharacter

Class to represent characters of Sn.

__init__(*self: Snob2.SnCharacter*, *arg0: Snob2.IntegerPartition*) → None

The character corresponding to the integer partition lambda.

__str__(*self: Snob2.SnCharacter*, *indent: str = ''*) → str

class SnIrrep

Class to represent the irreps of Sn.

__getitem__(*self: Snob2.SnIrrep*, *arg0: Snob2.SnElement*) → cnine::RtensorObj

`__init__`(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: Snob2.SnIrrep, arg0: int) -> None
2. __init__(self: Snob2.SnIrrep, arg0: Snob2.IntegerPartition) -> None

`__lt__`(*self: Snob2.SnIrrep*, *arg0: Snob2.SnIrrep*) → bool

`__str__`(*self: Snob2.SnIrrep*, *indent: str = ''*) → *str*

`get_dim`(*self: Snob2.SnIrrep*) → int

Return the dimension of the irrep

`str`(*self: Snob2.SnIrrep*, *indent: str = ''*) → *str*

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Y