props

Generated by Doxygen 1.6.1

Sun Apr 11 07:00:44 2010

Contents

Chapter 1

PROPS

1.1 Introduction

This is the technical documentation for PROPS, generated from the source code by Doxygen. For a more didactic and general introduction, see the `PROPS main page`.

1.2 Installation

If all the dependencies are installed (Python 2.6, IT++, SWIG, numpy), typing 'make' or 'make python' should build PROPS and its Python wrapper code. Typing 'make test' should build all the object files and C++ tests. Typing 'make dist' will build a tarball suitable for distribution. Typing 'python test.py' will run a series of unit tests.

1.3 Overview

1.4 Libraries

`IT++` is employed for linear algebra and matrix manipulation. In order to efficiently construct numpy matrices out of IT++ matrices, it helps to know how IT++ stores matrices internally. There does not appear to be clear documentation on this, but PROPs assumes a dense column-major format which is apparently correct.

`SWIG` is used to generate python wrappers for the C++ code.

`numpy` is used in all Python code for matrix manipulation. Matrices are transferred from IT++ to numpy with the help of SWIG and numpy's C interface.

1.5 Coding Style

The basic coding style is taken from the `Google C++ style guide`. In particular, operator overloading is used only where it is obvious what the operators should do, and constructors are only used where there is an obvious choice of initialization parameters; appropriately named static methods are used in other cases. Exceptions are used to signal errors, which automatically become Python exceptions under SWIG.

All code is documented with doxygen. Method descriptions are placed in header files (except for static methods). Documentation of static methods is included for work on internals, even though those methods are not accessible from Python.

Chapter 2

Todo List

Member bluetooth_obs which one?

test this

Member char_prod Check ch0.size() == ch1.size()

Member direct_sum is there a built-in way to do this?

throw error

Class FourierFunc work out how band-limiting will operate

fix Matrix workaround

Member FourierFunc::at(const Partition &part) const build a hash table for this?

Member FourierFunc::inverse_fft() const think about running time of this

Member FourierFunc::inverse_fourier() const fix next line for band-limiting

Member FourierFunc::marginals(Partition part) precompute intertwining operators

what if not all the dominating partitions are being kept?

Member FourierFunc::shift(const Permutation &p0, const Permutation &p1) const test right multiplication

fix this for band-limiting

Member FourierFunc::zero(std::vector< Partition > ind_parts) do this with band-limiting instead

provide a way to construct an uninitialized FourierFunc

Class Func add min, max

Member Func::convolve(const Func &other) check dimensions

need a real test of this

Member Func::fft() const list running time

disallow 0-element FourierFunc construction

Member Func::fourier() const list running time

Member Func::join(Func ∗d0, Func ∗d1) want arbitrary indices for d0, d1

Member Func::operator∗(const Func &other) check dimensions

Member Func::restrict_to(int i) const more general restriction?

Member gz_irrep_adj_swap_eles Currently this is Theta(irrep dimension), which does not take advantage of sparseness. Reimplement with map for Theta(lg d) or with hash_map for Theta(1)

Member intertwine check size consistency

Member kostka should we check here that the Kostka number is nonzero?

Member Matrix::Matrix(int r, int c) Fix encapsulation

Member Matrix::operator+=(const Matrix &m) check dimensions

Member mix_pair Check i, j, p

Member mix_subset check k <= n
check that subset makes sense

Member mult_irrep right multiplication in wrong order
test this for identity

Member mult_irrep_left make sure above is true

Member multitrack_obs test this

Member Partition::minus() const what should this be called?
check that this is not a partition of 1

Member Partition::operator()(int i) const error checking

Member Partition::Partition(int p, int ∗parts) have the constructor sort _parts

Member Permutation::cont_cycle(int n, int i, int j) check j >= i

Member Permutation::from_inverse(int n, const int inv_map[]) error checking

Member Permutation::swap(int n, int i, int j) Check i, j

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 DimensionMismatch Class Reference

Inherits std::exception.

Public Member Functions

- const char * **what** () const throw ()

The documentation for this class was generated from the following file:

- matrix.h

5.2 FourierFunc Class Reference

A function on S_n stored as (band-limited) Fourier coefficients.

```
#include <fourierfunc.h>
```

Public Member Functions

- Func inverse_fourier () const

  Compute the "naive" inverse Fourier transform.

- Func inverse_fft () const

  Compute the inverse Fourier transform using Clausen's FFT.

- FourierFunc shift (const Permutation &p0, const Permutation &p1) const

  Shift this function in the Fourier domain.

- Matrix marginals (Partition part)

  Compute the matrix of marignals defined by the tabloids on partition part.

- std::string str () const

  A string representation of this function as coefficient matrices.

- Matrix at (const Partition &part) const

  The matrix of Fourier coefficients at the partition part.

- void set (int i, Matrix m)

  Set the ith stored matrix of coefficients to m.

- void set (const Partition &part, Matrix m)

  Set the matrix of coefficients at part to m.

- int num_elements () const

  Return n where this a function over S_n.

- std::vector< Partition > ind_parts ()

  Return the partitions at which Fourier coefficients are stored.

Static Public Member Functions

- static FourierFunc zero (std::vector< Partition > ind_parts)

  Create a zero function with coefficients stored at ind_parts.

- static FourierFunc zero (int nelements)

  Create a full (not band-limited) zero function on S_n.

Friends

- FourierFunc * uniform_fourier (int n)

  The uniform distribution in the Fourier domain on S_n.

5.2.1 Detailed Description

A function on S_n stored as (band-limited) Fourier coefficients.

Todo

work out how band-limiting will operate
fix Matrix workaround

5.2.2 Member Function Documentation

5.2.2.1 Matrix FourierFunc::at (const Partition & *part*) const

The matrix of Fourier coefficients at the partition part.

Todo

build a hash table for this?

5.2.2.2 Func FourierFunc::inverse_fft () const

Compute the inverse Fourier transform using Clausen's FFT.

Todo

think about running time of this

5.2.2.3 Func FourierFunc::inverse_fourier () const

Compute the "naive" inverse Fourier transform. Thus is a direct translation of the definition, i.e., a sum over "matrix dot products", and so operates in Theta(n!^2). This exists for demonstration; in practice inverse_fft should be used instead

Todo

fix next line for band-limiting

5.2.2.4 Matrix FourierFunc::marginals (Partition *part*)

Compute the matrix of marignals defined by the tabloids on partition part.

Todo

precompute intertwining operators
what if not all the dominating partitions are being kept?

5.2.2.5 FourierFunc FourierFunc::shift (const Permutation & *p0*, const Permutation & *p1*) const

Shift this function in the Fourier domain. If this is the Fourier transform of f, the result is the Fourier transform of f(p) = f(p0^-1 p p1)

Todo

test right multiplication

Todo

fix this for band-limiting

5.2.2.6 FourierFunc FourierFunc::zero (std::vector< Partition > *ind_parts*) `[static]`

Create a zero function with coefficients stored at ind_parts.

Todo

do this with band-limiting instead
provide a way to construct an uninitialized FourierFunc

The documentation for this class was generated from the following files:

- fourierfunc.h
- fourierfunc.cpp

5.3 Func Class Reference

Real-valued explicitly-stored functions on permutations.

`#include <func.h>`

Public Types

- typedef double **val_t**

Public Member Functions

- **Func** (const Func &f)
- Func & **operator=** (const Func &f)
- Func & operator+= (const Func &f)

    Add another function to this one.

- std::string str ()

    A string representation of this function.

- val_t operator() (const Permutation &sigma) const

    The actual mapping S_n -> R.

- Func ∗ operator∗ (const Func &other)

    Pointwise product of functions.

- Func ∗ convolve (const Func &other)

    Convolution.

- Func ∗ shift (const Permutation &p0, const Permutation &p1)

    Compute a shifted function f so that f(p) = this(p0^-1 p p1).

- void set (const Permutation &p, val_t v)

    Set the val of this function at permutation p.

- FourierFunc fourier () const

    Compute the "naive" Fourier transform.

- FourierFunc fft () const

    Compute the Fourier transform using Clausen's FFT.

- Func restrict_to (int i) const
- Func **embed** (int i) const
- int num_elements () const

    Return n where this a function over S_n.

- val_t marginalize (Partition part, Permutation p0, Permutation p1)

    Compute the probability that the tabloid (part, p0) maps to the tabloid (part, p1).

- Matrix marginals (Partition part)

Compute a matrix of marginals defined by the tabloids on partition part.

- val_t ordered (int i, int j)

  Compute the sum of f(p) over permutations p such that p(i) < p(j) (for a distribution, compute P(p(i) < p(j)).

- Matrix orderings ()

  Compute the matrix of order marginals: orderings()->(i,j) = ordered(i,j).

- void normalize ()

  Normalize this distribution; public functions should always maintain normalization.

- std::pair< Func *, Func * > split (int n)

  Split this distribution into two first-order independent distributions.

Static Public Member Functions

- static Func zero (int n)

  Construct a new function on S_n that is identically zero.

- static Func * join (Func *d0, Func *d1)

  Construct the distribution formed by joining d0 and d1 as first-order independent.

Protected Member Functions

- Func (int n)

  Construct uninitialized func on S_n.

Protected Attributes

- val_t * func

  The explicit function: {0, ..., n! - 1} -> R.

- int nelements

  This is a function over S_nelements.

- int funclen

  Number of values stored in func, i.e., nelements!

Static Protected Attributes

- static const int MAX_PRINT_ELEMENTS = 6

  The maximum number of values displayed by the output of str.

Friends

- Func ∗ uniform (int n)

  Construct the uniform distribution on S_n.

- Func ∗ mix_subset (int n, int k, Permutation::perm_val_t ∗subset)

  Construct the k-subset mixing model.

- Func ∗ mix_pair (int n, int i, int j, val_t p)

  Construct the pairwise mixing model.

- Func ∗ insertion_mix (int n)

  Construct the insertion mixing model.

- Func ∗ mallows (Permutation p0, double c)

  The Mallows model.

- Func ∗ multitrack_obs (int n, int k, int ∗tracks, int ∗obs, Func::val_t p)

  The multi-track observation model.

- Func ∗ singletrack_obs (int n, int i, int j, Func::val_t pi)

  The single track observation model.

- Func ∗ bluetooth_obs (int n, int k, int ∗tracks, int ∗obs, Func::val_t pi)

  The bluetooth observation model.

- Func ∗ pair_rank_obs (int n, int i, int j, Func::val_t pi)

  The pairwise ranking observation model.

5.3.1 Detailed Description

Real-valued explicitly-stored functions on permutations.

Todo

add min, max

5.3.2 Member Function Documentation

5.3.2.1 Func ∗ Func::convolve (const Func & *other*)

Convolution.

Todo

check dimensions

Todo

need a real test of this

5.3.2.2 FourierFunc Func::fft () const

Compute the Fourier transform using Clausen's FFT.

Todo

list running time

Todo

disallow 0-element FourierFunc construction

5.3.2.3 FourierFunc Func::fourier () const

Compute the "naive" Fourier transform. This is a direct translation of the definition; it exists only for testing/demonstrating; in practice fft should be used instead

Todo

list running time

5.3.2.4 Func * Func::join (Func * *d0*, Func * *d1*) `[static]`

Construct the distribution formed by joining d0 and d1 as first-order independent.

Todo

want arbitrary indices for d0, d1

5.3.2.5 Matrix Func::marginals (Partition *part*)

Compute a matrix of marginals defined by the tabloids on partition part. The (i,j) entry of the matrix is the probability that tabloid i maps to tabloid j, where tabloids are enumerated in the order given by tabloid_list

5.3.2.6 Func * Func::operator* (const Func & *other*)

Pointwise product of functions.

Todo

check dimensions

5.3.2.7 Func Func::restrict_to (int *i*) const

Todo

more general restriction?

5.3.2.8 pair< Func *, Func * > Func::split (int *n*)

Split this distribution into two first-order independent distributions. The first distribution will be a mapping over the indices is in the original distribution, and the second over {1, ..., n} \ is

5.3.3 Friends And Related Function Documentation

5.3.3.1 Func* bluetooth_obs (int *n*, int *k*, int * *tracks*, int * *obs*, Func::val_t *pi*) `[friend]`

The bluetooth observation model.

Todo

which one?
test this

5.3.3.2 Func* mallows (Permutation *p0*, double *c*) `[friend]`

The Mallows model. The distribution defined by P(p) = exp(-c * d(p, p0)) where d is the Kendall's tau distance: the number of adjacent swaps to transform p^-1 into p0^-1

5.3.3.3 Func* mix_pair (int *n*, int *i*, int *j*, val_t *p*) `[friend]`

Construct the pairwise mixing model.

Todo

Check i, j, p

5.3.3.4 Func* mix_subset (int *n*, int *k*, Permutation::perm_val_t * *subset*) `[friend]`

Construct the k-subset mixing model.

Todo

check k <= n

Todo

check that subset makes sense

5.3.3.5 Func* multitrack_obs (int *n*, int *k*, int * *tracks*, int * *obs*, Func::val_t *p*) `[friend]`

The multi-track observation model.

Todo

test this

5.3.3.6 Func* pair_rank_obs (int *n*, int *i*, int *j*, Func::val_t *pi*) [friend]

The pairwise ranking observation model. The pairwise ranking model is defined as pi for permutations p where p(i) < p(j) and 1 - pi otherwise

5.3.4 Member Data Documentation

5.3.4.1 val_t* Func::func [protected]

The explicit function: {0, ..., n! - 1} -> R. The indexing used is that defined by Permutation::index

The documentation for this class was generated from the following files:

- func.h
- func.cpp

5.4 IndexOutOfBounds Class Reference

Inherits std::exception.

Public Member Functions

- const char $*$ **what** () const throw ()

The documentation for this class was generated from the following file:

- permutation.h

5.5 InvalidPartition Class Reference

Inherits std::exception.

Public Member Functions

- const char * **what** () const throw ()

The documentation for this class was generated from the following file:

- partition.h

5.6 InvalidPermutation Class Reference

Inherits std::exception.

Public Member Functions

- const char * **what** () const throw ()

The documentation for this class was generated from the following file:

- permutation.h

5.7 Matrix Class Reference

A basic explicitly-stored matrix.

`#include <matrix.h>`

Public Types

- typedef double mat_val_t

    The type used for matrix entries.

Public Member Functions

- **Matrix** (const Matrix &m)
- Matrix & **operator=** (const Matrix &m)
- Matrix **operator*** (mat_val_t s) const
- Matrix operator+= (const Matrix &m)
- mat_val_t **ele** (int i, int j) const
- mat_val_t matrix_dot (const Matrix &m) throw (DimensionMismatch)

    *Compute the matrix dot product, trace(this^T*m).*

- void set (int r, int c, mat_val_t val)

    Set the (r,c) element to val.

- Matrix **submat** (int r0, int c0, int r, int c) const
- int **num_rows** () const
- int **num_cols** () const
- mat_val_t trace () const

    The trace of this matrix, i.e., the sum of the diagonal elements.

- std::string **str** () const
- Matrix (int r, int c)

Static Public Member Functions

- static Matrix identity (int n)

    Construct an n x n identity matrix.

- static Matrix zero (int r, int c)

    Construct an r x c zero matrix.

- static Matrix **direct_sum** (std::vector< Matrix > ms)

Public Attributes

- mat_val_t * mat

  The matrix stored in row major order.

- int rows

  Number of rows in matrix.

- int cols

  Number of columns in matrix.

5.7.1 Detailed Description

A basic explicitly-stored matrix.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 Matrix::Matrix (int *r*, int *c*)

Todo

Fix encapsulation

Construct an uninitialized r x c matrix

5.7.3 Member Function Documentation

5.7.3.1 Matrix::mat_val_t Matrix::matrix_dot (const Matrix & *m*) throw (DimensionMismatch)

Compute the matrix dot product, trace(this^T*m). Note that this is Theta(nm) while computing trace(this^T*m) explictly is O((nm)^2)

5.7.3.2 Matrix Matrix::operator+= (const Matrix & *m*)

Todo

check dimensions

The documentation for this class was generated from the following files:

- matrix.h
- matrix.cpp

5.8 Partition Class Reference

Partitions of integers represented explicitly.

`#include <partition.h>`

Public Member Functions

- Partition (int p, int ∗parts) throw (InvalidPartition)

    Create a new partition of sum(parts) consisting of p parts.

- Partition (const Partition &p)

    Copy constructor.

- Partition & **operator=** (const Partition &p)
- int operator() (int i) const

    Return the size of the ith greatest element of this partition.

- int num () const

    The number that this partition partitions.

- int num_parts () const

    The number of parts of this partition.

- int hook_len (int i, int j) const

    The hook length from the box at component i, position j.

- std::vector< Partition > minus () const
- Partition decr (int i) const

    Form a new partition by decrementing the ith component of this one.

- Partition **subpart** (int start, int end) const
- Partition **concat** (const Partition &part) const
- bool operator== (const Partition &p) const

    Test two partitions for equality.

- bool operator!= (const Partition &p) const

    Opposite of ==.

- bool dominates (const Partition &p) const

    Return true iff this partition dominates p.

- std::string str () const

    A string representation of the partition.

- std::string **repr** () const

Static Public Member Functions

- static std::vector< Partition > list (int n)

  Enumerate partitions in lexicographic order.

- static std::vector< Partition > **list_up_to** (int n, unsigned int size)

5.8.1 Detailed Description

Partitions of integers represented explicitly.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 Partition::Partition (int *_p*, int ∗ *_parts*) throw (InvalidPartition)

Create a new partition of sum(parts) consisting of p parts.

Todo

have the constructor sort _parts

5.8.3 Member Function Documentation

5.8.3.1 bool Partition::dominates (const Partition & *p*) const

Return true iff this partition dominates p. Domination is taken in the weak sense, so p.dominates(p) returns true

5.8.3.2 vector< Partition > Partition::minus () const

Todo

what should this be called?

Todo

check that this is not a partition of 1

5.8.3.3 int Partition::operator() (int *i*) const

Return the size of the ith greatest element of this partition.

Todo

error checking

The documentation for this class was generated from the following files:

- partition.h
- partition.cpp

5.9 Permutation Class Reference

Permutations on n elements.

`#include <permutation.h>`

Public Types

- typedef int perm_val_t

  The type that this permutation maps.

Public Member Functions

- Permutation (int n, const perm_val_t _map[ ]) throw (InvalidPermutation)

  Construct a permutation given the explicit mapping.

- Permutation (const Permutation &sigma)

  Copy constructor.

- Permutation & **operator=** (const Permutation &other)
- int min_cycle_ele () const

  The minimum i that is not mapped to i.

- perm_val_t operator() (perm_val_t i) throw (IndexOutOfBounds)

  The result of applying this permutation to i.

- Permutation of (const Permutation &sigma) const throw (PermutationMismatch)

  Composition of permutations, i.e., this o sigma.

- Permutation inverse () const

  The inverse permutation, this^-1.

- Permutation direct_sum (const Permutation &sigma) const

  The direct sum of permutations.

- int index () const

  Implements a bijection from S_n to {0, ..., n! - 1}.

- int num_elements () const

  The number of elements permuted.

- bool setmapsto (int k, int ∗is, int ∗js)
- std::string str ()

  Produce a "one line" explicit string representation.

- std::string **repr** ()
- std::vector< Permutation > factor_swaps ()

  Factor this permutation into swaps (transpositions).

- std::vector< Permutation > factor_adj_swaps ()

  Factor this permutation into adjacent swaps (transpositions of the form (i,i+1)).

- Permutation ()

  For constructing by static methods.

Static Public Member Functions

- static Permutation identity (int n)

  Construct the identity permutation on n elements.

- static Permutation swap (int n, int i, int j)

  Construct the transposition of i and j.

- static Permutation from_inverse (int n, const int inv_map[ ]) throw (InvalidPermutation)

  Construct a permutation from {0, ..., n-1} permuted, i.e., the inverse mapping.

- static Permutation cont_cycle (int n, int i, int j)
- static Permutation from_index (int n, int i) throw (IndexOutOfBounds)

  Implements a bijection from {0, ..., n! - 1} to S_n.

5.9.1 Detailed Description

Permutations on n elements. A permutation is a mapping from {0,...,n-1} to {0,...,n-1}, i.e., an element of S_n. Permutations are immutable.

5.9.2 Member Function Documentation

5.9.2.1 Permutation Permutation::cont_cycle (int *n*, int *i*, int *j*) `[static]`

Todo

check j >= i

5.9.2.2 Permutation Permutation::direct_sum (const Permutation & *sigma*) const

The direct sum of permutations. If p0 is in S_n, p1 is in S_m, the direct sum p0 (+) p1 = p is defined so that p(i) = p0(i) when i < n or p1(i - n) + n when i >= n

5.9.2.3 Permutation Permutation::from_index (int *n*, int *i*) throw (IndexOutOfBounds) `[static]`

Implements a bijection from {0, ..., n! - 1} to S_n. This is the inverse operation from index

5.9.2.4 Permutation Permutation::from_inverse (int *n*, const int *inv_map*[ ]) throw (InvalidPermutation) `[static]`

Construct a permutation from {0, ..., n-1} permuted, i.e., the inverse mapping.

Todo

error checking

5.9.2.5 int Permutation::index () const

Implements a bijection from S_n to {0, ..., n! - 1}. This is the inverse operation from from_index

5.9.2.6 int Permutation::min_cycle_ele () const

The minimum i that is not mapped to i. In other words, the first element of the first cycle comprising this permutation. In particular, if this permutation is (i, i+1), returns i. If this is the identity permutation, returns num_elements().

5.9.2.7 bool Permutation::setmapsto (int *k*, int * *is*, int * *js*)

Decide if for every i in is there is some j in js such that p(i) = j, i.e, if the set of indices is maps unordered to the values js.

k should be the number of values pointed to by is and js. This function is useful for testing tabloid equivalence.

5.9.2.8 Permutation Permutation::swap (int *n*, int *i*, int *j*) `[static]`

Construct the transposition of i and j.

Todo

Check i, j

The documentation for this class was generated from the following files:

- permutation.h
- permutation.cpp

5.10 PermutationMismatch Class Reference

Inherits std::exception.

Public Member Functions

- const char * **what** () const throw ()

The documentation for this class was generated from the following file:

- permutation.h

5.11 Tableau Class Reference

(standard) Young tableau

`#include <partition.h>`

Public Member Functions

- Tableau (const Tableau &t)

  Copy constructor.

- std::string str () const

  This tableau, written explicitly.

- std::pair< int, int > **pos** (int i) const
- int axial_dist (int i, int j) const

  The axial distance, or (signed) number of steps required to get from i to j by moving in four adjacent directions.

- Tableau swap (int i) const

  Compute (i, i + 1) o (this tabloid).

- bool operator== (const Tableau &t)

  Test two tableau for equality.

Static Public Member Functions

- static std::vector< Tableau > list (const Partition &part)

  Enumerate all standard tableau for a particular partition.

- static int count (const Partition &part)

  Count the number of tableau of a particular shape using the hook formula.

5.11.1 Detailed Description

(standard) Young tableau

The documentation for this class was generated from the following files:

- partition.h
- partition.cpp

Chapter 6

File Documentation

6.1 dist.cpp File Reference

Some functions for constructing distributions on permutations. `#include "dist.h"`

`#include "util.h"`

`#include <cmath>`

Functions

- Func ∗ uniform (int n)

  Construct the uniform distribution on S_n.

- Func ∗ mix_subset (int n, int k, Permutation::perm_val_t ∗subset)

  Construct the k-subset mixing model.

- Func ∗ mix_pair (int n, int i, int j, Func::val_t p)

  Construct the pairwise mixing model.

- Func ∗ insertion_mix (int n)

  Construct the insertion mixing model.

- Func ∗ mallows (Permutation p0, double c)

  The Mallows model.

6.1.1 Detailed Description

Some functions for constructing distributions on permutations.

6.1.2 Function Documentation

6.1.2.1 Func∗ mallows (Permutation *p0*, double *c*)

The Mallows model. The distribution defined by P(p) = exp(-c ∗ d(p, p0)) where d is the Kendall's tau distance: the number of adjacent swaps to transform p^-1 into p0^-1

6.1.2.2 Func∗ mix_pair (int *n*, int *i*, int *j*, Func::val_t *p*)

Construct the pairwise mixing model.

Todo

Check i, j, p

6.1.2.3 Func∗ mix_subset (int *n*, int *k*, Permutation::perm_val_t ∗ *subset*)

Construct the k-subset mixing model.

Todo

check k <= n

Todo

check that subset makes sense

6.2 dist.h File Reference

Definition of Func class for (explicit) distributions on permutations. `#include "func.h"`
`#include "partition.h"`

Functions

- Func ∗ uniform (int n)

  Construct the uniform distribution on S_n.

- Func ∗ mix_subset (int n, int k, Permutation::perm_val_t ∗subset)

  Construct the k-subset mixing model.

- Func ∗ mix_pair (int n, Permutation::perm_val_t i, Permutation::perm_val_t j, Func::val_t p)

  Construct the pairwise mixing model.

- Func ∗ insertion_mix (int n)

  Construct the insertion mixing model.

- Func ∗ mallows (Permutation p0, double c)

  The Mallows model.

6.2.1 Detailed Description

Definition of Func class for (explicit) distributions on permutations.

6.2.2 Function Documentation

6.2.2.1 Func∗ mallows (Permutation *p0*, double *c*)

The Mallows model. The distribution defined by P(p) = exp(-c ∗ d(p, p0)) where d is the Kendall's tau distance: the number of adjacent swaps to transform p^-1 into p0^-1

6.2.2.2 Func∗ mix_pair (int *n*, Permutation::perm_val_t *i*, Permutation::perm_val_t *j*, Func::val_t *p*)

Construct the pairwise mixing model.

Todo

Check i, j, p

Todo

Check i, j, p

6.2.2.3 Func∗ mix_subset (int *n*, int *k*, Permutation::perm_val_t ∗ *subset*)

Construct the k-subset mixing model.

Todo

check k <= n

Todo

check that subset makes sense

Todo

check k <= n

Todo

check that subset makes sense

6.3 fourier.cpp File Reference

Implementation of Fourier-domain functions for the symmetric group. `#include "fourier.h"`

```
#include "util.h"
#include <vector>
#include <cmath>
#include <iostream>
#include <itpp/itbase.h>
```

Defines

- #define **SQ**(x) ((x)*(x))
- #define **Matrix** mat

Functions

- static vector< pair< int, double > > gz_irrep_adj_swap_eles (vector< Tableau > tabs, int i, int j)

    Find the nonzero elements in row (or column) j of the irrep at (i, i+1) in the gz basis given by the specified tableaux.

- Matrix gz_irrep (Partition part, Permutation p)

    Construct the irreducible representation matrix in the Gel'fand-Tsetlin basis at the given permutation.

- static Matrix mult_irrep (Partition part, Permutation p, Matrix m, bool left_mult)
- Matrix mult_irrep_left (Partition part, Permutation p, Matrix m)

    Multiply a matrix by the partition part irrep at the permutation p.

- Matrix mult_irrep_right (Partition part, Permutation p, Matrix m)

    Like mult_irrep_left, but perform right multiplication instead.

- vector< int > character (Partition part)

    The character of the irrep at partition part.

- int char_prod (vector< int > ch0, vector< int > ch1)

    The inner product of characters of two irreps.

- Matrix marginal_rep (Partition part, Permutation p)

    Compute the the marginal representation of S_n at the permutation p for the partition part.

- Matrix **direct_sum** (Matrix m0, Matrix m1)
- Matrix direct_sum (vector< Matrix > blocks)

    *Compute the direct sum a given vector of matrices */.*

- Matrix multi_direct_sum (Matrix m, int n)

    Compute the multiple direct sum m (+) ... (+) m (n times).

- Matrix **kron_factor_id** (Matrix p, int d)

- Matrix **normalize_rows** (Matrix m)
- static Matrix intertwine (Matrix X0, Matrix X1, Matrix Y0, Matrix Y1, int z)

  Compute part of an orthogonal intertwining operator between two orthogonal representations.

- static Matrix concat_vertical (vector< Matrix > ms)

  Vertically concatenate given vector of matrices.

- Matrix marg_intertwine (Partition part, vector< Partition > dom_parts)

  Compute an orthogonal intertwining operator between the matrix of marginals at partition part and the matrices of GZ basis irreps at the partitions dom_parts.

6.3.1 Detailed Description

Implementation of Fourier-domain functions for the symmetric group.

6.3.2 Function Documentation

6.3.2.1 int char_prod (vector< int > *ch0*, vector< int > *ch1*)

The inner product of characters of two irreps.

Todo

Check ch0.size() == ch1.size()

6.3.2.2 Matrix direct_sum (vector< Matrix > *blocks*)

Compute the direct sum a given vector of matrices ∗/.

Todo

is there a built-in way to do this?

Todo

throw error

6.3.2.3 static vector<pair<int, double> > gz_irrep_adj_swap_eles (vector< Tableau > *tabs*, int *i*, int *j*) `[static]`

Find the nonzero elements in row (or column) j of the irrep at (i, i+1) in the gz basis given by the specified tableaux. This function exists to allow efficient "sparse" computation with these irreps

Todo

Currently this is Theta(irrep dimension), which does not take advantage of sparseness. Reimplement with map for Theta(lg d) or with hash_map for Theta(1)

6.3.2.4 static Matrix intertwine (Matrix *X0*, Matrix *X1*, Matrix *Y0*, Matrix *Y1*, int *z*) `[static]`

Compute part of an orthogonal intertwining operator between two orthogonal representations. X0 and X1 should be elements of a representation that correspond to two different elements that generate S_n (such as a transposition and a complete cycle). Y0 and Y1 should be elements of a different representation that correspond to the same elements of S_n. z should be the multiplicity of the X representation in the Y representation (the dimension of the Y representation should be at least the dimension of the X representation).

In typical use, X is an irrep, Y is a marginal rep, and z is a Koskta number.

The matrix returned has orthonormal rows and forms an intertwining operator when stacked with other such matrices generated from X representations that from a decomposition of the representation Y.

Todo

check size consistency

6.3.2.5 Matrix marg_intertwine (Partition *part*, std::vector< Partition > *dom_parts*)

Compute an orthogonal intertwining operator between the matrix of marginals at partition part and the matrices of GZ basis irreps at the partitions dom_parts. The partitions from dom_parts are used with multiplicities given by the Kostka numbers; any that do not dominate part will not be used. Commonly, dom_parts will be all partitions above (and including part) in the dominance ordering.

6.3.2.6 Matrix marginal_rep (Partition *part*, Permutation *p*)

Compute the the marginal representation of S_n at the permutation p for the partition part. The (i,j) entry of the resulting matrix is 1 if p(tabloid_j) = tabloid_i, where tabloids are indexed in the order given by tabloid_list, composition means applying the map p to each element of the tabloid, and equality means tabloid equivalance. Note the "swiched" order of indices, which is necessary for the composition order of matrices to follow that of permutations.

6.3.2.7 static Matrix mult_irrep (Partition *part*, Permutation *p*, Matrix *m*, bool *left_mult*) `[static]`

Todo

right multiplication in wrong order

Todo

test this for identity

6.3.2.8 Matrix mult_irrep_left (Partition *part*, Permutation *p*, Matrix *m*)

Multiply a matrix by the partition part irrep at the permutation p. This function is Theta(nd), where d is the dimension of the matrix and n is the number of adjacent transposition factors of the irrep

Todo

make sure above is true

6.4 fourier.h File Reference

Declaration of Fourier-domain functions for the symmetric group. `#include "partition.h"`

`#include <vector>`

`#include <itpp/itbase.h>`

Defines

- #define **Matrix** itpp::mat

Functions

- Matrix gz_irrep (Partition part, Permutation p)

  Construct the irreducible representation matrix in the Gel'fand-Tsetlin basis at the given permutation.

- Matrix mult_irrep_left (Partition part, Permutation p, Matrix m)

  Multiply a matrix by the partition part irrep at the permutation p.

- Matrix mult_irrep_right (Partition part, Permutation p, Matrix m)

  Like mult_irrep_left, but perform right multiplication instead.

- std::vector< int > character (Partition part)

  The character of the irrep at partition part.

- int char_prod (std::vector< int > ch0, std::vector< int > ch1)

  The inner product of characters of two irreps.

- Matrix direct_sum (std::vector< Matrix > ms)

  *Compute the direct sum a given vector of matrices */.*

- Matrix multi_direct_sum (Matrix m, int n)

  Compute the multiple direct sum m (+) ... (+) m (n times).

- Matrix marg_intertwine (Partition part, std::vector< Partition > dom_parts)

  Compute an orthogonal intertwining operator between the matrix of marginals at partition part and the matrices of GZ basis irreps at the partitions dom_parts.

- Matrix marginal_rep (Partition part, Permutation p)

  Compute the the marginal representation of S_n at the permutation p for the partition part.

6.4.1 Detailed Description

Declaration of Fourier-domain functions for the symmetric group.

6.4.2 Function Documentation

6.4.2.1 int char_prod (vector< int > *ch0*, vector< int > *ch1*)

The inner product of characters of two irreps.

Todo

Check ch0.size() == ch1.size()

6.4.2.2 Matrix direct_sum (vector< Matrix > *blocks*)

Compute the direct sum a given vector of matrices */.

Todo

is there a built-in way to do this?

Todo

throw error

6.4.2.3 Matrix marg_intertwine (Partition *part*, std::vector< Partition > *dom_parts*)

Compute an orthogonal intertwining operator between the matrix of marginals at partition part and the matrices of GZ basis irreps at the partitions dom_parts. The partitions from dom_parts are used with multiplicities given by the Kostka numbers; any that do not dominate part will not be used. Commonly, dom_parts will be all partitions above (and including part) in the dominance ordering.

6.4.2.4 Matrix marginal_rep (Partition *part*, Permutation *p*)

Compute the the marginal representation of S_n at the permutation p for the partition part. The (i,j) entry of the resulting matrix is 1 if p(tabloid_j) = tabloid_i, where tabloids are indexed in the order given by tabloid_list, composition means applying the map p to each element of the tabloid, and equality means tabloid equivalance. Note the "swiched" order of indices, which is necessary for the composition order of matrices to follow that of permutations.

6.4.2.5 Matrix mult_irrep_left (Partition *part*, Permutation *p*, Matrix *m*)

Multiply a matrix by the partition part irrep at the permutation p. This function is Theta(nd), where d is the dimension of the matrix and n is the number of adjacent transposition factors of the irrep

Todo

make sure above is true

6.5 fourierfunc.cpp File Reference

Implementation of FourierFunc class for manipulating band-limited functions on S_n. `#include <vector>`

`#include <iostream>`

`#include <sstream>`

`#include "fourierfunc.h"`

`#include "partition.h"`

`#include "func.h"`

`#include "util.h"`

`#include "fourier.h"`

Defines

- #define **Matrix** itpp::mat

6.5.1 Detailed Description

Implementation of FourierFunc class for manipulating band-limited functions on S_n.

6.6 fourierfunc.h File Reference

Declaration of FourierFunc class for band-limited functions on S_n stored as Fourier coefficients.

```
#include "partition.h"
#include <itpp/itbase.h>
```

Classes

- class FourierFunc

  A function on S_n stored as (band-limited) Fourier coefficients.

Defines

- #define **Matrix** itpp::mat

6.6.1 Detailed Description

Declaration of FourierFunc class for band-limited functions on S_n stored as Fourier coefficients.

6.7 func.cpp File Reference

Implementation of explicit real-valued functions on permutations. #include <sstream>

#include <cstring>

#include <iostream>

#include <assert.h>

#include "func.h"

#include "permutation.h"

#include "partition.h"

#include "fourier.h"

#include "util.h"

#include "fourierfunc.h"

Defines

- #define **Matrix** itpp::mat

6.7.1 Detailed Description

Implementation of explicit real-valued functions on permutations.

6.8 func.h File Reference

Definition of class for real-valued functions on permutations. `#include <string>`

`#include "partition.h"`

`#include "permutation.h"`

`#include <itpp/itbase.h>`

Classes

- class Func

    Real-valued explicitly-stored functions on permutations.

Defines

- #define **Matrix** itpp::mat

6.8.1 Detailed Description

Definition of class for real-valued functions on permutations.

6.9 matrix.cpp File Reference

Implementation of Matrix class. `#include "matrix.h"`

`#include <cstring>`

`#include <sstream>`

`#include <iomanip>`

Defines

- #define **MIN**(x, y) (((x) < (y)) ? (x) : (y))

6.9.1 Detailed Description

Implementation of Matrix class.

6.10 matrix.h File Reference

Declaration of an explicitly-stored matrix class. `#include <exception>`

`#include <string>`

`#include <vector>`

`#include <itpp/itbase.h>`

Classes

- class DimensionMismatch
- class Matrix

  A basic explicitly-stored matrix.

Functions

- itpp::mat **direct_sum** (std::vector< itpp::mat > blocks)

6.10.1 Detailed Description

Declaration of an explicitly-stored matrix class.

6.11 models.cpp File Reference

Implementation of various Fourier-domain probabilistic models. `#include "models.h"`

Defines

- #define **Matrix** itpp::mat

Functions

- FourierFunc ∗ uniform_fourier (int n)
  The uniform distribution in the Fourier domain on S_n.

6.11.1 Detailed Description

Implementation of various Fourier-domain probabilistic models.

6.12 models.h File Reference

Various Fourier-domain models. `#include "fourierfunc.h"`

Functions

- FourierFunc ∗ uniform_fourier (int n)

  The uniform distribution in the Fourier domain on S_n.

6.12.1 Detailed Description

Various Fourier-domain models.

6.13 obs.cpp File Reference

Implementation of various observation models. `#include "obs.h"`
`#include "util.h"`

Functions

- Func ∗ multitrack_obs (int n, int k, int ∗tracks, int ∗obs, Func::val_t pi)
  The multi-track observation model.
- Func ∗ singletrack_obs (int n, int i, int j, Func::val_t pi)
  The single track observation model.
- Func ∗ bluetooth_obs (int n, int k, int ∗tracks, int ∗obs, Func::val_t pi)
  The bluetooth observation model.
- Func ∗ pair_rank_obs (int n, int i, int j, Func::val_t pi)
  The pairwise ranking observation model.

6.13.1 Detailed Description

Implementation of various observation models.

6.13.2 Function Documentation

6.13.2.1 Func∗ bluetooth_obs (int *n*, int *k*, int ∗ *tracks*, int ∗ *obs*, Func::val_t *pi*)

The bluetooth observation model.

Todo

which one?
test this

6.13.2.2 Func∗ multitrack_obs (int *n*, int *k*, int ∗ *tracks*, int ∗ *obs*, Func::val_t *p*)

The multi-track observation model.

Todo

test this

6.13.2.3 Func∗ pair_rank_obs (int *n*, int *i*, int *j*, Func::val_t *pi*)

The pairwise ranking observation model. The pairwise ranking model is defined as pi for permutations p where p(i) < p(j) and 1 - pi otherwise

6.14 obs.h File Reference

Various observation models. `#include "func.h"`

Functions

- Func ∗ multitrack_obs (int n, int k, int ∗tracks, int ∗obs, Func::val_t p)
  The multi-track observation model.
- Func ∗ singletrack_obs (int n, int i, int j, Func::val_t pi)
  The single track observation model.
- Func ∗ bluetooth_obs (int n, int k, int ∗tracks, int ∗obs, Func::val_t pi)
  The bluetooth observation model.
- Func ∗ pair_rank_obs (int n, int i, int j, Func::val_t pi)
  The pairwise ranking observation model.

6.14.1 Detailed Description

Various observation models.

6.14.2 Function Documentation

6.14.2.1 Func∗ bluetooth_obs (int *n*, int *k*, int ∗ *tracks*, int ∗ *obs*, Func::val_t *pi*)

The bluetooth observation model.

Todo

which one?
test this

6.14.2.2 Func∗ multitrack_obs (int *n*, int *k*, int ∗ *tracks*, int ∗ *obs*, Func::val_t *p*)

The multi-track observation model.

Todo

test this

6.14.2.3 Func∗ pair_rank_obs (int *n*, int *i*, int *j*, Func::val_t *pi*)

The pairwise ranking observation model. The pairwise ranking model is defined as pi for permutations p where p(i) < p(j) and 1 - pi otherwise

6.15 partition.cpp File Reference

Implementation of partitions and related things. `#include "partition.h"`

`#include "util.h"`

`#include <cstring>`

`#include <sstream>`

`#include <iostream>`

`#include <assert.h>`

Defines

- #define **MIN**(x, y) ((x) < (y) ? (x) : (y))

Functions

- bool tabloid_equal (const Partition &part, const Permutation &p0, const Permutation &p1)

    Determine if two Young's tabloids are equal (as tabloids).

- static vector< vector< int > > **ord_subsets** (vector< int > list, unsigned int n)
- static vector< int > **ord_minus** (vector< int > a, vector< int > b)
- static vector< vector< int > > **make_tabloid_list** (const Partition &part, vector< int > avail)
- static int ∗ **new_list_from_vector** (vector< int > v)
- vector< Permutation > tabloid_list (const Partition &part)

    List all the (tabloid_equal-distinct) Young's tabloids over a partition.

- static vector< Partition > **fill_ss_tableau** (const Partition &part, int n, int right)
- int kostka (const Partition &m_part, const Partition &i_part)

    Calculate the multiplicity of an irrep in a given marginal rep.

6.15.1 Detailed Description

Implementation of partitions and related things.

6.15.2 Function Documentation

6.15.2.1 int kostka (const Partition & *m_part*, const Partition & *i_part*)

Calculate the multiplicity of an irrep in a given marginal rep. This is the Kostka number, calculated according to Young's rule. m_part is the partition defining the marginal rep, and i_part is the partition defining the irrep

Todo

should we check here that the Kostka number is nonzero?

6.15.2.2 bool tabloid_equal (const Partition & *part*, const Permutation & *p0*, const Permutation & *p1*)

Determine if two Young's tabloids are equal (as tabloids). The tabloids are both over the partition part, and defined to contain the elements p0(0), p0(1), ... filled in from left to right, largest component to smallest component

6.15.2.3 vector<Permutation> tabloid_list (const Partition & *part*)

List all the (tabloid_equal-distinct) Young's tabloids over a partition. The tabloids are returned as permutations, as described in tabloid_equal

6.16 partition.h File Reference

Declarations of partitions and related things: tabloids and tableaux. `#include "permutation.h"`
`#include <vector>`

Classes

- class InvalidPartition
- class Partition

  Partitions of integers represented explicitly.

- class Tableau

  (standard) Young tableau

Functions

- bool tabloid_equal (const Partition &part, const Permutation &p0, const Permutation &p1)

  Determine if two Young's tabloids are equal (as tabloids).

- std::vector< Permutation > tabloid_list (const Partition &part)

  List all the (tabloid_equal-distinct) Young's tabloids over a partition.

- int kostka (const Partition &m_part, const Partition &i_part)

  Calculate the multiplicity of an irrep in a given marginal rep.

6.16.1 Detailed Description

Declarations of partitions and related things: tabloids and tableaux.

6.16.2 Function Documentation

6.16.2.1 int kostka (const Partition & *m_part*, const Partition & *i_part*)

Calculate the multiplicity of an irrep in a given marginal rep. This is the Kostka number, calculated according to Young's rule. m_part is the partition defining the marginal rep, and i_part is the partition defining the irrep

Todo

should we check here that the Kostka number is nonzero?

6.16.2.2 bool tabloid_equal (const Partition & *part*, const Permutation & *p0*, const Permutation & *p1*)

Determine if two Young's tabloids are equal (as tabloids). The tabloids are both over the partition part, and defined to contain the elements p0(0), p0(1), ... filled in from left to right, largest component to smallest component

6.16.2.3 std::vector<Permutation> tabloid_list (const Partition & *part*)

List all the (tabloid_equal-distinct) Young's tabloids over a partition. The tabloids are returned as permutations, as described in tabloid_equal

6.17 permutation.cpp File Reference

Implementation of Permutation class. `#include <sstream>`

`#include <iostream>`

`#include "permutation.h"`

`#include "util.h"`

Functions

- int * new_seq (int i, int len)

    Return a newly allocated array of length len that counts from i.

6.17.1 Detailed Description

Implementation of Permutation class.

6.17.2 Function Documentation

6.17.2.1 int* new_seq (int *i*, int *len*)

Return a newly allocated array of length len that counts from i. The returned array should be freed with delete.

6.18 permutation.h File Reference

Definition of Permutation class. `#include <string>`

`#include <vector>`

Classes

- class IndexOutOfBounds
- class InvalidPermutation
- class PermutationMismatch
- class Permutation

  Permutations on n elements.

Functions

- int * new_seq (int i, int len)

  Return a newly allocated array of length len that counts from i.

6.18.1 Detailed Description

Definition of Permutation class.

6.18.2 Function Documentation

6.18.2.1 int* new_seq (int *i*, int *len*)

Return a newly allocated array of length len that counts from i. The returned array should be freed with delete.

6.19 props.h File Reference

Include all PROPS header files. `#include "func.h"`
`#include "fourierfunc.h"`

6.19.1 Detailed Description

Include all PROPS header files.

6.20 speedtest.cpp File Reference

A simple way of testing the speed of the FFT. `#include <cstdlib>`

`#include <iostream>`

`#include "props.h"`

Functions

- int **main** (int argc, char *argv[ ])

6.20.1 Detailed Description

A simple way of testing the speed of the FFT.

6.21 util.cpp File Reference

Utility functions for props.

Functions

- double **fact** (double n)

6.21.1 Detailed Description

Utility functions for props.

6.22 util.h File Reference

Prototypes of utility functions for props.

Defines

- #define **RND**(x) ((x) < 0 ? (int)((x) - 0.5) : (int)((x) + 0.5))

Functions

- double **fact** (double n)

6.22.1 Detailed Description

Prototypes of utility functions for props.

Index