# Yael

v300

Generated by Doxygen 1.7.3

Thu Sep 13 2012 11:21:49

# Contents

# Chapter 1

# Module Index

## 1.1 Modules

Here is a list of all modules:

# Chapter 2

# Data Structure Index

## 2.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 3

# Module Documentation

## 3.1 Binheap

This structure is used, in particular, to find the maxk smallest elements of a possibly unsized stream of values.

### Data Structures

- struct **fbinheap_s**

  *Binary heap used as a maxheap.*

### Typedefs

- typedef struct **fbinheap_s fbinheap_t**

### Functions

- struct **fbinheap_s** ∗ **fbinheap_new** (int maxk)

  *create the maxheap structure for maxk elements (maximum)*

- size_t **fbinheap_sizeof** (int maxk)

  *return the size of a maxheap structure*

- void **fbinheap_init** (**fbinheap_t** ∗bh, int maxk)

  *A binheap can be stored in an externally allocated memory area of fbinheap_sizeof(maxk) bytes.*

- void **fbinheap_delete** (**fbinheap_t** ∗bh)

  *free allocated memory*

- void **fbinheap_reset** (**fbinheap_t** ∗bh)

  *remove all the elements from the heap*

- void **fbinheap_add** (**fbinheap_t** ∗bh, int label, float val)

  *insert an element on the heap (if the value val is small enough)*

- void **fbinheap_pop** (**fbinheap_t** ∗bh)

  *remove largest value from binheap (low-level access!)*

- void **fbinheap_addn** (**fbinheap_t** ∗bh, int n, const int ∗labels, const float ∗v)

  *add n elements on the heap (the values are added only if they are small enough compared to the other elements)*

- void **fbinheap_addn_label_range** (**fbinheap_t** ∗bh, int n, int label0, const float ∗v)

  *add n elements on the heap, using the set of labels starting at label0*

- void **fbinheap_sort_labels** (**fbinheap_t** ∗bh, int ∗perm)

  *output the labels in increasing order of associated values*

- void **fbinheap_sort_values** (**fbinheap_t** ∗bh, float ∗v)

  *output the sorted values*

- void **fbinheap_sort** (**fbinheap_t** ∗bh, int ∗labels, float ∗v)

  *output both sorted results: labels and corresponding values*

- void **fbinheap_sort_per_labels** (**fbinheap_t** ∗bh, int ∗labels, float ∗v)

  *sort by increasing labels, ouptput sorted labels & associated values*

- void **fbinheap_display** (**fbinheap_t** ∗bh)

  *show the heap content*

### 3.1.1 Detailed Description

This structure is used, in particular, to find the maxk smallest elements of a possibly unsized stream of values.

### 3.1.2 Function Documentation

#### 3.1.2.1 struct fbinheap_s∗ fbinheap_new ( int *maxk* ) [read]

create the maxheap structure for maxk elements (maximum)

**Parameters**

| | |
|---|---|
| *maxk* | maximum number of elements to be stored in the heap |

References fbinheap_init(), and fbinheap_sizeof().

Referenced by merge_ordered_sets().

#### 3.1.2.2 size_t fbinheap_sizeof ( int *maxk* )

return the size of a maxheap structure

**Parameters**

| | |
|---|---|
| *maxk* | the maximum number of elements that the structure will receive |

Referenced by fbinheap_new(), and knn_full().

#### 3.1.2.3 void fbinheap_init ( **fbinheap_t** * *bh*, int *maxk* )

A binheap can be stored in an externally allocated memory area of fbinheap_sizeof(maxk) bytes.

The **fbinheap_init()** (p. 7) function is used to initialize this memory area

References fbinheap_s::k, fbinheap_s::label, fbinheap_s::maxk, and fbinheap_s::val.

Referenced by fbinheap_new(), and knn_full().

#### 3.1.2.4 void fbinheap_addn ( **fbinheap_t** * *bh*, int *n*, const int * *labels*, const float * *v* )

add n elements on the heap (the values are added only if they are small enough compared to the other elements)

**Parameters**

| | |
|---|---|
| *bh* | the maxheap structure |
| *n* | the number of elements to be added |
| *labels* | The identifiers for the values to be added |
| *v* | the set of vectors to be added |

References fbinheap_pop(), fbinheap_s::k, fbinheap_s::maxk, and fbinheap_s::val.

#### 3.1.2.5 void fbinheap_sort_labels ( **fbinheap_t** * *bh*, int * *perm* )

output the labels in increasing order of associated values

**Parameters**

| | |
|---|---|
| *bh* | the maxheap structure perm the array that receive the output permutation order (pre-allocated) |

References fvec_sort_index(), fbinheap_s::k, fbinheap_s::label, and fbinheap_s::val.

## 3.2 Gmm

Gaussian Mixture Model implementation and Fisher Kernels, as defined in: F.

### Data Structures

- struct **gmm_s**

  *Gaussian Mixture Model (GMM) implementation.*

### Defines

- #define **GMM_FLAGS_W** 1

  *during computation of probabilities: take weights into account*

- #define **GMM_FLAGS_NO_NORM** 2

  *do not normalize probabilities (bad!)*

- #define **GMM_FLAGS_1SIGMA** 4

  *during learning: compute a single value for the sigma diagonal*

- #define **GMM_FLAGS_PURE_KMEANS** 32

  *during gmm learning: just do a kmeans*

- #define **GMM_FLAGS_SIGMA** 8

  *dp_dlambda: include mu and sigma in derivatives*

- #define **GMM_FLAGS_MU** 16

### Typedefs

- typedef struct **gmm_s gmm_t**

  *Gaussian Mixture Model (GMM) implementation.*

### Functions

- **gmm_t** ∗ **gmm_learn** (int d, int n, int k, int niter, const float ∗v, int nt, int seed, int nredo, int flags)

  *Estimate the Gaussian mixture in two stages:*
  - *standard kmeans,*
  - *EM to find parameters.*

- void **gmm_print** (const **gmm_t** ∗g)

*Describe to stdout.*

- void **gmm_delete** (**gmm_t** ∗g)

  *free a GMM structure*

- void **gmm_compute_p** (int n, const float ∗v, const **gmm_t** ∗g, float ∗p, int flags)

  *compute probabilities of centroids for each vector p(c_i|x).*

- void **gmm_fisher** (int n, const float ∗v, const **gmm_t** ∗g, int flags, float ∗fisher_vector_out)

  *Fisher descriptor.*

- size_t **gmm_fisher_sizeof** (const **gmm_t** ∗g, int flags)
- void **gmm_write** (const **gmm_t** ∗g, FILE ∗f)

  *write the GMM structure parameter into a file*

- **gmm_t** ∗ **gmm_read** (FILE ∗f)

  *read the GMM from a file*

- void **gmm_compute_p_thread** (int n, const float ∗v, const **gmm_t** ∗g, float ∗p, int flags, int n_thread)

  *Threaded version of gmm_compute_p.*

### 3.2.1 Detailed Description

Gaussian Mixture Model implementation and Fisher Kernels, as defined in: F. Perronnin and C. Dance, Fisher kernels on visual vocabularies for image categorization, CVPR 2006.

### 3.2.2 Function Documentation

#### 3.2.2.1 gmm_t∗ gmm_learn ( int *d*, int *n*, int *k*, int *niter*, const float ∗ *v*, int *nt*, int *seed*, int *nredo*, int *flags* )

Estimate the Gaussian mixture in two stages:

- standard kmeans,
- EM to find parameters.

**Parameters**

| | |
|---|---|
| *d,k* | see gmm_t structure |
| *n* | nb of learning points |
| *niter* | nb of iterations (same for both stages) |

| | |
|---|---|
| *v(d,n)* | input vectors |
| *nt* | nb of threads |
| *seed* | usedd by kmeans to initialize random number generator |
| *nredo* | number of "redo" (launch several kmeans) |
| *flags* | see GMM_∗ flags. Typically, use GMM_FLAGS_W (to estimate weights). |

References fvec_new(), fvec_new_0(), fvec_set(), fvec_sum(), gmm_compute_p_thread(), GMM_FLAGS_PURE_KMEANS, ivec_new(), ivec_print(), kmeans(), gmm_s::mu, gmm_s::sigma, and gmm_s::w.

#### 3.2.2.2 void gmm_compute_p ( int *n*, const float ∗ *v*, const gmm_t ∗ *g*, float ∗ *p*, int *flags* )

compute probabilities of centroids for each vector p(c_i|x).

**Parameters**

| | |
|---|---|
| *v(d,n)* | v(:,i) is c_i |
| *p(k,n)* | output probability values |

References gmm_s::d, fvec_new(), GMM_FLAGS_W, gmm_s::k, gmm_s::mu, gmm_s::sigma, and gmm_s::w.

Referenced by gmm_fisher().

#### 3.2.2.3 void gmm_fisher ( int *n*, const float ∗ *v*, const gmm_t ∗ *g*, int *flags*, float ∗ *fisher_vector_out* )

Fisher descriptor.

Compute

nabla_lambda p(x, lambda)

where

lambda = (w, mu, sqrt(sigma))

**Parameters**

| | |
|---|---|
| *v(d,n)* | vectors where to compute descriptor |
| *flags* | combination of GMM_FLAGS_∗. Typically, use yael.GMM_FLAGS_MU (only interested in the derivative wrt mu) |
| *fisher_vector_out(dd)* | output descriptor. The output descriptor size dd is given by gmm_fisher_sizeof(flags) |

References gmm_s::d, fmat_mul_tr(), fvec_new(), gmm_compute_p(), GMM_FLAGS_NO_NORM, GMM_FLAGS_SIGMA, gmm_s::k, and gmm_s::w.

## 3.3 Kmlsh

K-means LSH is an implementation of the technique described in the following paper "Locality sensitive hashing: a comparison of hash function types and querying mechanisms", by L.

### Data Structures

- struct **nnlist_s**

  *A structure to handle the list of KNN.*

- struct **kmlsh_s**

  *The structure that contains the parameters of the KM-LSH.*

- struct **kmlsh_idx_s**

  *A structure containing the pre-processed data (tables of quantized indexes) for a set of vectors.*

### Defines

- #define **KMLSH_NT** 0x000000ff
- #define **KMLSH_QUIET** 0x00010000
- #define **KMLSH_WRITE_INTER_NHASH** 0x00020000
- #define **KMLSH_BLOCK_SIZE** 256
- #define **KMLSH_NB_ITER_MAX** 8
- #define **KMLSH_VECTYPE_FVEC** 0
- #define **KMLSH_VECTYPE_BVEC** 1

### Typedefs

- typedef struct **nnlist_s nnlist_t**
- typedef struct **kmlsh_s kmlsh_t**
- typedef struct **kmlsh_idx_s kmlsh_idx_t**

### Functions

- **nnlist_t** ∗ **nnlist_new** (int n, int k)
- **nnlist_t** ∗ **nnlist_new_noalloc** (int n, int k)
- void **nnlist_delete** (**nnlist_t** ∗l)
- void **nnlist_addn** (**nnlist_t** ∗l, int lno, int n, int ∗idx, float ∗dis)
- **kmlsh_t** ∗ **kmlsh_new** (int nhash, int nclust, int d)
- void **kmlsh_delete** (**kmlsh_t** ∗lsh)
- void **kmlsh_learn_xvec** (**kmlsh_t** ∗lsh, int n, int nlearn, const void ∗v, int flags, int vec_type)

- **kmlsh_t** ∗ **kmlsh_new_learn_bvec** (int nhash, int nclust, int d, int n, int nlearn, const unsigned char ∗v, int flags)
- **kmlsh_t** ∗ **kmlsh_new_learn_fvec** (int nhash, int nclust, int d, int n, int nlearn, const float ∗v, int flags)
- **nnlist_t** ∗ **kmlsh_match_xvec** (const **kmlsh_t** ∗lsh, const **kmlsh_idx_t** ∗lshidx_b, const void ∗vb, int nb, const **kmlsh_idx_t** ∗lshidx_q, const void ∗vq, int nq, int k, int nt, int vec_type)

  *A function that performs the match assuming that the codes are pre-computed.*

- **nnlist_t** ∗ **kmlsh_match_bvec** (const **kmlsh_t** ∗lsh, const **kmlsh_idx_t** ∗lshidx_b, const unsigned char ∗vb, int nb, const **kmlsh_idx_t** ∗lshidx_q, const unsigned char ∗vq, int nq, int k, int nt)
- **nnlist_t** ∗ **kmlsh_match_fvec** (const **kmlsh_t** ∗lsh, const **kmlsh_idx_t** ∗lshidx_b, const float ∗vb, int nb, const **kmlsh_idx_t** ∗lshidx_q, const float ∗vq, int nq, int k, int nt)
- **nnlist_t** ∗ **kmlsh_ann_xvec** (const void ∗vb, int nb, const void ∗vq, int nq, int d, int k, int nhash, int flags, int vec_type)
- **nnlist_t** ∗ **kmlsh_ann_bvec** (const unsigned char ∗vb, int nb, const unsigned char ∗vq, int nq, int d, int k, int nhash, int flags)
- **nnlist_t** ∗ **kmlsh_ann_fvec** (const float ∗vb, int nb, const float ∗vq, int nq, int d, int k, int nhash, int flags)
- **kmlsh_idx_t** ∗ **kmlsh_idx_new** (const **kmlsh_t** ∗lsh, int n)
- void **kmlsh_idx_delete** (**kmlsh_idx_t** ∗lshidx)
- **kmlsh_idx_t** ∗ **kmlsh_idx_new_compile_xvec** (const **kmlsh_t** ∗lsh, const void ∗v, int n, int flags, int vec_type)
- **kmlsh_idx_t** ∗ **kmlsh_idx_new_compile_bvec** (const **kmlsh_t** ∗lsh, const unsigned char ∗v, int n, int flags)
- **kmlsh_idx_t** ∗ **kmlsh_idx_new_compile_fvec** (const **kmlsh_t** ∗lsh, const float ∗v, int n, int flags)
- int **kmlsh_idx_get_nvec** (const **kmlsh_idx_t** ∗lshidx, int h, int c)
- int **kmlsh_idx_get_maxincell** (const **kmlsh_idx_t** ∗lshidx, int h)
- int ∗ **kmlsh_idx_get_vecids** (const **kmlsh_idx_t** ∗lshidx, int h, int c)
- void **kmeans_cohash_xvec** (const **kmlsh_t** ∗lsh, int h, const void ∗v, int n, int ∗perm, int ∗boundaries, int flags, int vec_type)
- void **kmeans_cohash_bvec** (const **kmlsh_t** ∗lsh, int h, const unsigned char ∗v, int n, int ∗perm, int ∗boundaries, int flags)
- void **kmeans_cohash_fvec** (const **kmlsh_t** ∗lsh, int h, const float ∗v, int n, int ∗perm, int ∗boundaries, int flags)
- void **kmlsh_write** (const char ∗filename, const **kmlsh_t** ∗lsh)
- void **kmlsh_read** (const char ∗filename, const **kmlsh_t** ∗lsh)
- void **kmlsh_idx_write** (const char ∗filename, const **kmlsh_idx_t** ∗lshidx)
- void **kmlsh_idx_read** (const char ∗filename, **kmlsh_idx_t** ∗lshidx)

### 3.3.1 Detailed Description

K-means LSH is an implementation of the technique described in the following paper "Locality sensitive hashing: a comparison of hash function types and querying mechanisms", by L. Pauleve, H. Jegou and L. Amsaleg, Pattern Recognition Letters, August 2010

Only the regular LSH (multiple hash functions, no multi-probe, no query adaptive) is provided. This implementation is not intended for a classical query/database scenario, although it could be used for (with relatively low efficienty). Instead, it is optimized towards batch processing of large amounts of queries.

## 3.4 Matrix

Matrix functions.

### Data Structures

- struct **pca_online_s**

### Typedefs

- typedef struct **pca_online_s** **pca_online_t**

### Functions

- float ∗ **fmat_new** (int nrow, int ncol)

  *Allocate a new nrow x ncol matrix.*

- float ∗ **fmat_new_0** (int nrow, int ncol)
- void **fmat_mul_full** (const float ∗left, const float ∗right, int m, int n, int k, const char ∗transp, float ∗result)

  *Matrix multiplication.*

- void **fmat_mul_full_nonpacked** (const float ∗left, const float ∗right, int m, int n, int k, const char ∗transp, int ld_left, int ld_right, float ∗result, int ld_result)

  *same as fmat_mul_full, matrices may be non-packed (yes, this is close to sgemm)*

- float ∗ **fmat_new_mul_full** (const float ∗left, const float ∗right, int m, int n, int k, const char ∗transp)

  *same as fmat_mul, allocates result*

- void **fmat_mul** (const float ∗left, const float ∗right, int m, int n, int k, float ∗mout)

  *same as fmat_mul_full, all in standard order*

- void **fmat_mul_tl** (const float ∗left, const float ∗right, int m, int n, int k, float ∗mout)

  *same as fmat_mul_full, left(k,m) transposed*

- void **fmat_mul_tr** (const float ∗left, const float ∗right, int m, int n, int k, float ∗mout)

  *same as fmat_mul_full, right(n,k) transposed*

- void **fmat_mul_tlr** (const float ∗left, const float ∗right, int m, int n, int k, float ∗mout)

  *same as fmat_mul_full, left(k,m) and right(n,k) transposed*

- float ∗ **fmat_new_mul** (const float ∗left, const float ∗right, int m, int n, int k)
- float ∗ **fmat_new_mul_tl** (const float ∗left, const float ∗right, int m, int n, int k)
- float ∗ **fmat_new_mul_tr** (const float ∗left, const float ∗right, int m, int n, int k)
- float ∗ **fmat_new_mul_tlr** (const float ∗left, const float ∗right, int m, int n, int k)
- int **fmat_solve_ls_t** (int m, int n, const float ∗a, const float ∗b, float ∗x)

  *solve the linear squares system a∗x = b with n unknowns and m equations.*

- void **fmat_print** (const float ∗a, int nrow, int ncol)

  *display the matrix in matlab-parsable format*

- void **fmat_print_tranposed** (const float ∗a, int nrow, int ncol)

  *same as fmat_print but matrix is in row-major order*

- float ∗ **fmat_get_submatrix** (const float ∗a, int nrow, int nrow_out, int ncol)

  *Extract a submatrix.*

- int ∗ **imat_get_submatrix** (const int ∗a, int nrow, int nrow_out, int ncol)
- float ∗ **fmat_new_get_columns** (const float ∗a, int nrow, int ncolout, const int ∗cols)

  *return the submatrix defined by a list of columns*

- void **fmat_get_columns** (const float ∗a, int d, int ncolout, const int ∗cols, float ∗out)
- void **fmat_get_rows_cols** (const float ∗a, int d, int n_row, const int ∗rows, int n_col, const int ∗cols, float ∗out)

  *return the matrix defined by*

- void **fmat_shuffle_columns** (float ∗a, int nrow, int ncol)
- float ∗ **fmat_new_get_row** (const float ∗a, int nrow, int ncol, int row)

  *produce a vector by taking a particular row of a matrix*

- float ∗ **fmat_new_get_rows** (const float ∗a, int d, int n, int nrowout, const int ∗rows)

  *produce a matrix composed of the rows indicated by the vector rows*

- void **fmat_sum_columns** (const float ∗a, int nrow, int ncol, float ∗sums)

  *per-column sum of matrix elements.*

- float ∗ **fmat_new_sum_columns** (const float ∗a, int nrow, int ncol)
- void **fmat_sum_rows** (const float ∗a, int nrow, int ncol, float ∗sums)

  *per-row sum of matrix elements*

- float ∗ **fmat_new_sum_rows** (const float ∗a, int nrow, int ncol)
- float ∗ **fmat_new_vstack** (const float ∗a, int da, const float ∗b, int db, int n)
- float ∗ **fmat_new_transp** (const float ∗a, int ncol, int nrow)

  *Matrix transposition.*

- void **fmat_splat_separable** (const float ∗a, int nrow, int ncol, const int ∗row_assign, const int ∗col_assign, int k, float ∗accu)

  *RM a is ncol-by-nrow accu is k-by-k.*

- int ∗ **imat_joint_histogram** (int n, int k, int ∗row_assign, int ∗col_assign)
- int **fmat_remove_0_columns** (float ∗a, int d, int n)

  *removes 0-filled columns of a matrix.*

- void **fmat_normalize_columns_l2sqr_pow** (float ∗a, int d, int n, float pw)

  *replaces each column with colmn ∗ (norm2sqr of column) ^ pw*

- float ∗ **fmat_new_rand_gauss** (int nrow, int ncol)

  *RM produce a new matrix of size nrow x ncol, filled with gaussian values.*

- float ∗ **random_orthogonal_basis** (int d)

  *produce a random orthogonal basis matrix of size d∗d*

- float ∗ **hadamard** (int d)

  *Construct a Hadamard matrix of dimension d using the Sylvester construction.*

- float ∗ **fmat_center_columns** (int d, int n, float ∗v)
- void **fmat_subtract_from_columns** (int d, int n, float ∗m, const float ∗avg)
- void **fmat_add_to_columns** (int d, int n, float ∗m, const float ∗avg)
- void **fmat_rev_subtract_from_columns** (int d, int n, float ∗m, const float ∗avg)
- float ∗ **fmat_new_covariance** (int d, int n, const float ∗v, float ∗avg, int assume_centered)

  *Compute covariance of a set of vectors.*

- float ∗ **fmat_new_pca** (int d, int n, const float ∗v, float ∗singvals)

  *Perform the Principal Component Analysis of a set of vectors.*

- float ∗ **fmat_new_pca_part** (int d, int n, int nev, const float ∗v, float ∗singvals)

  *same as fmat_pca, but return only a few vectors*

- int **fmat_svd_partial** (int d, int n, int ns, const float *a, float *singvals, float *u, float *v)

  *Compute SVD decomposition of a matrix: a = u * diag(singvals) * v'.*

- int **fmat_svd_partial_full** (int n, int m, int nev, const float *a, int a_transposed, float *s, float *vout, float *uout, int nt)

  *with additionnal options*

- float * **fmat_new_pca_from_covariance** (int d, const float *cov, float *singvals)

  *Compute the PCA eigenvalues and eigenvectors from covariance matrix.*

- void **fmat_pca_from_covariance** (int d, const float *cov, float *singvals, float *pcamat)
- **pca_online_t** * **pca_online_new** (int d)

  *Construct the online PCA structure.*

- void **pca_online_delete** (struct **pca_online_s** *pca)

  *Free memory associated with the online PCA structure.*

- void **pca_online_accu** (struct **pca_online_s** *pca, const float *v, long n)

  *Accumulate information for PCA for n input vectors.*

- void **pca_online_cov** (struct **pca_online_s** *pca)

  *compute the mean and covariance matrix*

- void **pca_online_complete** (struct **pca_online_s** *pca)

  *Online PCA: compute the mean and the eigenvectors.*

- void **pca_online_complete_part** (struct **pca_online_s** *pca, int nev)

  *Same function as pca_online_complete but for compute only the eigenvectors associated with the nev largest eigenvalues.*

- void **pca_online_project** (const **pca_online_t** *pca, const float *v, float *vo, int d, long n, int dout)

  *Project some vectors according to a PCA structure.*

### 3.4.1 Detailed Description

Matrix functions. All matrices are stored in column-major order (like Fortran and Matlab) and indexed from 0 (like C, unlike Fortran). The declaration:

a(m, n)

means that element a(i, j) is accessed with a[ i * m + j ] where

0 <= i < m and 0 <= j < n

WARNING some matrix functions assume row-major storage! (noted with RM)

### 3.4.2 Function Documentation

#### 3.4.2.1 void fmat_mul_full ( const float * *left,* const float * *right,* int *m,* int *n,* int *k,* const char * *transp,* float * *result* )

Matrix multiplication.

This function maps to the BLAS sgemm, assuming all matrices are packed

**Parameters**

| | |
|---|---|
| *left(m,k)* | left operand |
| *right(k,n)* | right perand |
| *result(m,n)* | result matrix |
| *m* | nb of rows of left matrix and of result |
| *n* | nb of columns of right matrix and of result |
| *k* | nb of columns of left matrix and nb of rows of right matrix |
| *transp* | transp[0] (resp. transp[1]) should be set to 'N' if the left (resp. right) matrix is in column-major (Fortran) order and to 'T' if it is row-major order (C order). The result is always in column-major order |

References fmat_mul_full_nonpacked().

Referenced by fmat_mul(), fmat_mul_tl(), fmat_mul_tlr(), fmat_mul_tr(), fmat_new_-mul_full(), and pca_online_project().

#### 3.4.2.2 int fmat_solve_ls_t ( int *m,* int *n,* const float * *a,* const float * *b,* float * *x* )

solve the linear squares system a*x = b with n unknowns and m equations.

**Parameters**

| | |
|---|---|
| *m* | number of unkowns |
| *n* | number of equations |
| *a(m,n)* | transposed matrix of the system |
| *b(n)* | right-hand side of the equation |
| *x(m)* | solution |

**Returns**

0 if ok, else an error code (see sgels doc)

References fvec_new().

#### 3.4.2.3 float* fmat_get_submatrix ( const float * *a,* int *nrow,* int *nrow_out,* int *ncol* )

Extract a submatrix.

**Parameters**

| | |
|---|---|
| *a* | the matrix (at least nrow by ncol) |
| *nrow* | nb of rows of input matrix |
| *nrow_out* | nb of rows of output matrix |
| *ncol* | nb of columns of output matrix |

**Returns**

the extracted submatrix

References fmat_new().

#### 3.4.2.4 void fmat_get_rows_cols ( const float * *a*, int *d*, int *n_row*, const int * *rows*, int *n_col*, const int * *cols*, float * *out* )

return the matrix defined by

out(i, j) = a(rows[i], cols[j])

#### 3.4.2.5 void fmat_sum_columns ( const float * *a*, int *nrow*, int *ncol*, float * *sums* )

per-column sum of matrix elements.

Output is a vector of length ncol

References fvec_0().

#### 3.4.2.6 float* fmat_new_transp ( const float * *a*, int *ncol*, int *nrow* )

Matrix transposition.

**Parameters**

| | |
|---|---|
| *a* | the matrix (nrow by ncol |
| *ncol* | number of columns of original matrix |
| *nrow* | number of rows of original matrix |

**Returns**

transposed copy of the matrix (and void for the inplace version)

References fvec_new().

#### 3.4.2.7 void fmat_splat_separable ( const float * *a*, int *nrow*, int *ncol*, const int * *row_assign*, const int * *col_assign*, int *k*, float * *accu* )

RM a is ncol-by-nrow accu is k-by-k.

for i=0..ncol-1,j=0..nrow-1, do accu(row_assign[i],col_assign[j]) += a(i,j)

#### 3.4.2.8 int fmat_remove_0_columns ( float * a, int d, int n )

removes 0-filled columns of a matrix.

Returns new number of columns

References fvec_all_0().

#### 3.4.2.9 float* hadamard ( int d )

Construct a Hadamard matrix of dimension d using the Sylvester construction.

d should be a power of 2

References fvec_new(), and hadamard().

Referenced by hadamard().

#### 3.4.2.10 float* fmat_new_covariance ( int d, int n, const float * v, float * avg, int assume_centered )

Compute covariance of a set of vectors.

**Parameters**

| | |
|---|---|
| *v(d,n)* | vectors to compute covariance |
| *avg(d)* | on output, average vector (can be NULL) |
| *assume_centered* | assumes the data is centered (avg not used) |

**Returns**

(d,d) covariance matrix

References fvec_0(), fvec_new(), and fvec_new_0().

Referenced by fmat_new_pca().

#### 3.4.2.11 float* fmat_new_pca ( int d, int n, const float * v, float * singvals )

Perform the Principal Component Analysis of a set of vectors.

**Parameters**

| | |
|---|---|
| *v(d,n)* | vectors to perform the PCA on. The vectors are assumed to be centered already! |
| *singvals(d)* | corresponding singular values (may be NULL) |

**Returns**

(d,d) matrix of eigenvectors (column-stored).

References fmat_new_covariance(), fmat_new_pca_from_covariance(), fvec_all_finite(), and fvec_new().

#### 3.4.2.12 float* fmat_new_pca_part ( int *d*, int *n*, int *nev*, const float * *v*, float * *singvals* )

same as fmat_pca, but return only a few vectors

**Parameters**

| | |
|---|---|
| *v(d,n)* | vectors to perform the PCA on. The vectors are assumed to be centered already! |
| *singvals(nev)* | corresponding singular values (may be NULL) |

**Returns**

(d,nev) matrix of eigenvectors. To transform a vector a low-dimension space, multiply by the d2<nev first lines of the matrix

References count_cpu(), fmat_new(), and fmat_svd_partial_full().

#### 3.4.2.13 int fmat_svd_partial ( int *d*, int *n*, int *ns*, const float * *a*, float * *singvals*, float * *u*, float * *v* )

Compute SVD decomposition of a matrix: a = u * diag(singvals) * v'.

**Parameters**

| | |
|---|---|
| *d* | nb of rows of matrix a |
| *n* | nb of columns of matrix a |
| *ns* | nb of singular values to compute |
| *a(d,n)* | matrix to compute singular vals for |
| *singvals(ns)* | output singular values (may be NULL) |
| *u(d,ns)* | left orthogonal matrix (may be NULL) |
| *v(n,ns)* | right orthogonal matrix (may be NULL) |

References count_cpu(), and fmat_svd_partial_full().

#### 3.4.2.14 void pca_online_complete ( struct pca_online_s * *pca* )

Online PCA: compute the mean and the eigenvectors.

Also compute the covariance matrix if not already done. The output is stored in the structure itself

References pca_online_cov().

Referenced by pca_online_complete_part().

## 3.5 Knearestneighbors

Nearest-neighbor (NN) functions.

### Functions

- void **knn_full** (int distance_type, int nq, int nb, int d, int k, const float ∗b, const float ∗q, const float ∗b_weights, int ∗assign, float ∗dis)

  *Finds nearest neighbors of vectors in a base.*

- void **knn_full_thread** (int distance_type, int nq, int nb, int d, int k, const float ∗b, const float ∗q, const float ∗b_weights, int ∗assign, float ∗dis, int n_thread)

  *multi-threaded version*

- double **nn** (int n, int nb, int d, const float ∗b, const float ∗v, int ∗assign)

  *single NN, returns sum of squared distances*

- double **nn_thread** (int n, int nb, int d, const float ∗b, const float ∗v, int ∗assign, int n_thread)

  *single NN, multithread*

- float ∗ **knn** (int n, int nb, int d, int k, const float ∗b, const float ∗v, int ∗assign)

  *also returns distances to centroids (alloc'ed with malloc)*

- float ∗ **knn_thread** (int nq, int nb, int d, int k, const float ∗b, const float ∗v, int ∗assign, int n_thread)
- void **knn_reorder_shortlist** (int n, int nb, int d, int k, const float ∗b, const float ∗v, int ∗idx, float ∗dis)

  *Re-order a short-list based on exact distances.*

- void **knn_recompute_exact_dists** (int n, int nb, int d, int k, const float ∗b, const float ∗v, int label0, int ∗kp, const int ∗idx, float ∗dis)

  *same as knn_reorder_shortlist for a partial base matrix (eg.*

- void **compute_cross_distances** (int d, int na, int nb, const float ∗a, const float ∗b, float ∗dist2)

  *Computes all distances between 2 sets of vectors.*

- void **compute_cross_distances_nonpacked** (int d, int na, int nb, const float ∗a, int lda, const float ∗b, int ldb, float ∗dist2, int ldd)

  *compute_cross_distances for non-packed matrices*

- void **compute_cross_distances_thread** (int d, int na, int nb, const float ∗a, const float ∗b, float ∗dist2, int nt)

  *compute_cross_distances with threads*

- void **compute_cross_distances_alt** (int distance_type, int d, int na, int nb, const float ∗a, const float ∗b, float ∗dist2)

  *Like compute_cross_distances with alternative distances.*

- void **compute_cross_distances_alt_nonpacked** (int distance_type, int d, int na, int nb, const float ∗a, int lda, const float ∗b, int ldb, float ∗dist2, int ldd)

  *compute_cross_distances_alt with non-packed input and output*

- void **compute_cross_distances_alt_thread** (int distance_type, int d, int na, int nb, const float ∗a, const float ∗b, float ∗dist2, int nt)
- void **compute_distances_1** (int d, int nb, const float ∗a, const float ∗b, float ∗dist2)

  *version of compute_cross_distances where na==1*

- void **compute_distances_1_nonpacked** (int d, int nb, const float ∗a, const float ∗b, int ldb, float ∗dist2)
- void **compute_distances_1_thread** (int d, int nb, const float ∗a, const float ∗b, float ∗dist2, int n_thread)
- void **compute_distances_1_nonpacked_thread** (int d, int nb, const float ∗a, const float ∗b, int ldb, float ∗dist2, int n_thread)

### 3.5.1 Detailed Description

Nearest-neighbor (NN) functions. All matrices are stored in column-major order (like Fortran) and indexed from 0 (like C, unlike Fortran). The declaration:

a(m, n)

means that element a(i, j) is accessed with a[ i ∗ m + j ] where

0 <= i < m and 0 <= j < n

### 3.5.2 Function Documentation

#### 3.5.2.1 void knn_full ( int *distance_type*, int *nq*, int *nb*, int *d*, int *k*, const float ∗ *b*, const float ∗ *q*, const float ∗ *b_weights*, int ∗ *assign*, float ∗ *dis* )

Finds nearest neighbors of vectors in a base.

**Parameters**

| | |
|---|---|
| *distance_-type* | 2 = L2 distance (see compute_cross_distances_alt for distance_type's) |
| *nq* | number of query vectors |
| *nb* | number of base vectors to assign to |
| *k* | number of neighbors to return |
| *q(d,n)* | query vectors |
| *b(d,nb)* | base vectors |

| | |
|---|---|
| *assign(k,n)* | on output, the NNs of vector i are assign(:, i) (sorted by increasing distances) |
| *b_weights(nb)* | multiply squared distances by this for each base vector (may be NULL) |
| *dis(k,n)* | squared distances of i to its NNs are dis(0, i) to dis(k-1, i) |

References compute_cross_distances(), compute_cross_distances_alt(), fbinheap_addn_label_range(), fbinheap_init(), fbinheap_sizeof(), fbinheap_sort(), fvec_new(), and fbinheap_s::k.

Referenced by knn(), knn_full_thread(), and nn().

#### 3.5.2.2 void knn_reorder_shortlist ( int *n*, int *nb*, int *d*, int *k*, const float * *b*, const float * *v*, int * *idx*, float * *dis* )

Re-order a short-list based on exact distances.

**Parameters**

| | |
|---|---|
| *n* | nb of query vectors |
| *nb* | nb of database vectors |
| *d* | dimension of vectors |
| *k* | nb of nearest-neighbors per query vector |
| *b(d,nb)* | database vector matrix |
| *v(d,nb)* | query vector matrix |
| *idx(k,nq)* | - input: idx(:,q) is the array of nearest neighbor indices to rerank<br>• output: idx(:,q) is a permutation of the input array, such that the NNs are ordered by increasing exact distance |
| *dis(k,nq)* | on output, dis(i,j) contains the exact squared L2 distance to the i^th NN of query j. |

References compute_distances_1(), fvec_new(), fvec_sort_index(), ivec_new(), and ivec_sort().

#### 3.5.2.3 void knn_recompute_exact_dists ( int *n*, int *nb*, int *d*, int *k*, const float * *b*, const float * *v*, int *label0*, int * *kp*, const int * *idx*, float * *dis* )

same as knn_reorder_shortlist for a partial base matrix (eg.

because b does not fit in memory)

**Parameters**

| | |
|---|---|
| *label0* | label of b(:,0) in the idx array |
| *idx(k,nq)* | idx(i, q) is the index of the i^th neighbor of query j. The array is sorted: |

idx(0, q) < idx(1, q) < ... < idx(k-1, q)

all distances for i st.

label0 <= idx(i,q) < label0 + nb

will be recomputed.

**Parameters**

| | |
|---|---|
| *kp(nq)* | index array of labels for which the distance must be recomputed.<br>• input: kp[q] is the smallest i st. label0 <= idx(i,q)<br>• output: kp[q] is the smallest i st. label0 + nb <= idx(i,q) |

References fvec_distance_L2sqr().

#### 3.5.2.4 void compute_cross_distances ( int *d*, int *na*, int *nb*, const float * *a*, const float * *b*, float * *dist2* )

Computes all distances between 2 sets of vectors.

**Parameters**

| | |
|---|---|
| *a(d,na)* | set of vectors |
| *b(d,nb)* | set of vectors |
| *dist2(na,nb)* | distances between all vectors of a and b. On output, |

dist2(i, j) = || a(:, i) - b(:, j) ||^2 = dist2[ i + na * j ]

where 0 <= i < na and 0 <= j < nb

References compute_cross_distances_nonpacked().

Referenced by compute_cross_distances_thread(), and knn_full().

#### 3.5.2.5 void compute_cross_distances_nonpacked ( int *d*, int *na*, int *nb*, const float * *a*, int *lda*, const float * *b*, int *ldb*, float * *dist2*, int *ldd* )

compute_cross_distances for non-packed matrices

**Parameters**

| | |
|---|---|
| *lda* | size in memory of one vector of a |
| *ldb* | size in memory of one vector of b |
| *ldd* | size in memory of one vector of dist2 |

Referenced by compute_cross_distances(), and compute_cross_distances_alt_nonpacked().

#### 3.5.2.6 void compute_cross_distances_alt ( int *distance_type,* int *d,* int *na,* int *nb,* const float * *a,* const float * *b,* float * *dist2* )

Like compute_cross_distances with alternative distances.

**Parameters**

| | |
|---|---|
| *distance_type* | type of distance to compute:<br>• 1: L1<br>• 2: L2 (use 12 for optimized version)<br>• 3: symmetric chi^2<br>• 4: symmetric chi^2 with absolute value<br>• 5: histogram intersection (sum of min of vals)<br>• 6: dot prod (use 16 for optimized version) |

References compute_cross_distances_alt_nonpacked().

Referenced by knn_full().

## 3.6 Sorting

Various sorting functions + a few simple array functions that can be called from python efficiently.

### Functions

- void **fvec_k_max** (const float *v, int n, int *maxes, int k)

  *Find the maximum elements of an array.*

- void **fvec_k_min** (const float *v, int n, int *mins, int k)

  *Find the minimum elements of an array.*

- void **fvec_ranks_of** (const float *tab, int n, const float *vals, int nval, int *minranks, int *maxranks)

  *finds the ranks of a few values in a large set.*

- void **fvec_ranks_inc_of** (const float *tab, int n, const float *vals, int nval, int *minranks, int *maxranks)

  *idem but ranks in increasing array*

- void **find_labels** (const int *labels, int nres, int *ilabels, int nilabels)

  *Replace ilabels[i] with the location of ilabels[i] in the table labels.*

- float **fvec_min** (const float *f, long n)

  *return the smallest value of a vector*

- int **ivec_min** (const int *f, long n)
- float **fvec_max** (const float *f, long n)

  *return the largest value of a vector*

- int **ivec_max** (const int *f, long n)
- int **fvec_arg_min** (const float *f, long n)

  *return the position of the smallest element of a vector.*

- int **fvec_arg_max** (const float *f, long n)

  *return the position of the largest elements of a vector.*

- float **fvec_median** (float *f, int n)

  *computes the median of a float array.*

- float **fvec_median_const** (const float *f, int n)
- float **fvec_quantile** (float *f, int n, int q)

  *find quantile.*

- void **ivec_sort** (int *tab, int n)

  *in-place sort*

- void **ivec_sort_index** (const int *tab, int n, int *perm)

  *return permutation to sort an array.*

- void **ivec_invert_perm** (const int *perm, int n, int *iperm)

  *fill-in iperm so that iperm[perm[i]]=i for i=0..n-1*

- void **fvec_sort** (float *v, int n)

  *in-place sort*

- void **fvecs_sort** (float *v, int d, int n)

  *in-place sort for several vectors*

- void **fvec_sort_index** (const float *tab, int n, int *perm)

  *return permutation to sort an array.*

- void **ivec_sort_by_permutation** (int *v, const int *order, int n)

  *Apply a permutation to a vector.*

- void **fvec_sort_by_permutation** (float *v, const int *order, int n)
- int **ivec_sorted_count_occurrences** (const int *v, int n, int val)

  *count occurrences of val in sorted vector*

- int **ivec_sorted_find** (const int *v, int n, int val)

  *find index of highest value <= val (=-1 if all values are > val)*

- int **ivec_sorted_count_unique** (const int ∗v, int n)

  *count the number of distinct values in the input fvector*

- int **ivec_sorted_count_occurrences_multiple** (const int ∗v, int n, const int ∗vals, int nval)

  *count the number of occurrences of several values*

- int **merge_ordered_sets** (const int ∗∗labels, const float ∗∗vals, const int ∗sizes, int k, int ∗∗labels_out, float ∗∗vals_out)

  *merge several sorted sets*

- int **compress_labels_by_disratio** (int ∗labels, const float ∗vals, int n, float ratio)

  *remove largest values from an array*

### 3.6.1 Detailed Description

Various sorting functions + a few simple array functions that can be called from python efficiently.

### 3.6.2 Function Documentation

#### 3.6.2.1 void fvec_k_max ( const float ∗ *v*, int *n*, int ∗ *maxes*, int *k* )

Find the maximum elements of an array.

**Parameters**

| | |
|---|---|
| *v(n)* | array to search |
| *maxes(k)* | largest values, on output: |

v[maxes[0]] >= v[maxes[1]] >= ... >= v[maxes[k-1]] >= v[i] for all i not in maxes.

#### 3.6.2.2 void fvec_k_min ( const float ∗ *v*, int *n*, int ∗ *mins*, int *k* )

Find the minimum elements of an array.

See find_k_max.

References fvec_arg_min().

#### 3.6.2.3 void fvec_ranks_of ( const float ∗ *tab*, int *n*, const float ∗ *vals*, int *nval*, int ∗ *minranks*, int ∗ *maxranks* )

finds the ranks of a few values in a large set.

Finds the ranks the values would have if the set was sorted by ∗decreasing∗ order.

**Parameters**

| | |
|---|---|
| *tab(n)* | unsorted table in which ranks are to be found |
| *vals(nval)* | values whose ranks are to be found |
| *min-ranks(nval)* | minimum rank of each value (may be NULL) |
| *maxranks(nva* | maximum rank of each value + 1 (may be NULL) |

On ouput, for value 0 <= i < nval,

- minranks[i]-1 is the highest index of values > vals[i]
- maxranks[i] is the lowest index of values < vals[i]
- if vals[i] is in the array, elements minranks[i] to maxranks[i]-1 have value vals[i]

The algorithm is in O(n ∗ log(nval)). If nval is larger, it is more efficient to sort the array.

#### 3.6.2.4 void find_labels ( const int ∗ *labels*, int *nres*, int ∗ *ilabels*, int *nilabels* )

Replace ilabels[i] with the location of ilabels[i] in the table labels.

**Parameters**

| | |
|---|---|
| *labels(nres)* | |
| *il-abels(nilabels* | |

On output ilabels is modified:

labels[ilabels_out[i]] = ilabels[i] for 0<=i<nilabels or -1 if there is none

#### 3.6.2.5 int fvec_arg_min ( const float ∗ *f*, long *n* )

return the position of the smallest element of a vector.

First position in case of ties, n should be >0.

Referenced by fvec_k_min().

#### 3.6.2.6 int fvec_arg_max ( const float ∗ *f*, long *n* )

return the position of the largest elements of a vector.

First position in case of ties, n should be >0.

#### 3.6.2.7 float fvec_median ( float * f, int n )

computes the median of a float array.

Array modified on output!

#### 3.6.2.8 float fvec_quantile ( float * f, int n, int q )

find quantile.

**Returns**

value v such that q elements are <= v

References fvec_max(), and fvec_min().

#### 3.6.2.9 void ivec_sort_index ( const int * tab, int n, int * perm )

return permutation to sort an array.

**Parameters**

| | |
|---|---|
| *tab(n)* | table to sort |
| *perm(n)* | output permutation that sorts table |

On output,

tab[perm[0]] <= tab[perm[1]] <= ... <= tab[perm[n-1]]

Is stable.

Referenced by fbinheap_sort_per_labels().

#### 3.6.2.10 void fvec_sort_index ( const float * tab, int n, int * perm )

return permutation to sort an array.

See ivec_sort_index.

Referenced by eigs_reorder(), fbinheap_sort(), fbinheap_sort_labels(), and knn_reorder_-shortlist().

#### 3.6.2.11 void ivec_sort_by_permutation ( int * v, const int * order, int n )

Apply a permutation to a vector.

The permutation is typically generated using the ivec_sort_index function. In that case the function outputs a sorted array.

#### 3.6.2.12 int merge_ordered_sets ( const int ** *labels*, const float ** *vals*, const int * *sizes*, int *k*, int ** *labels_out*, float ** *vals_out* )

merge several sorted sets

There are k sets. Set 0 <= i < k has size sizes[i]. Element j of set i is vals[i][j], and the arbitrary associated index is lists[i][j]. The set is ordered, so on input

vals[i][0] <= vals[i][1] <= ... <= vals[i][sizes[i]-1]

On ouput, *labels_out and *vals_out contain an ordered set with all values.

**Returns**

total number of elements (=sum(sizes[i],i=0..k-1))

References fbinheap_add(), fbinheap_delete(), fbinheap_new(), fbinheap_pop(), fvec_new(), ivec_new(), fbinheap_s::k, fbinheap_s::label, and fbinheap_s::val.

#### 3.6.2.13 int compress_labels_by_disratio ( int * *labels*, const float * *vals*, int *n*, float *ratio* )

remove largest values from an array

Finds the smallest value m of vals, compresses array labels by removing labels[i] for which vals[i] < m * ratio returns new size of labels array.

NB that on output, vals[i] does not correspond to labels[i] any more!

## 3.7 Vector

Vectors are represented as C arrays of basic elements.

### Functions

- float * **fvec_new** (long n)

  *Alloc a new aligned vector of floating point values -- to be de-allocated with free.*

- int * **ivec_new** (long n)

  *Alloc an int array -- to be de-allocated with free.*

- unsigned char * **bvec_new** (long n)

  *Alloc an byte array -- to be de-allocated with free.*

- long long * **lvec_new** (long n)

  *Alloc a long array -- to be de-allocated with free.*

- double * **dvec_new** (long n)

  *Alloc an int array -- to be de-allocated with free.*

- float ∗ **fvec_new_0** (long n)
  *create a vector initialized with 0's.*

- double ∗ **dvec_new_0** (long n)
- int ∗ **ivec_new_0** (long n)
  *create a vector initialized with 0's.*

- unsigned char ∗ **bvec_new_0** (long n)
- long long ∗ **lvec_new_0** (long n)
  *create a vector initialized with 0's.*

- float ∗ **fvec_new_nan** (long n)
  *create a vector initialized with NaN (to trace errors)*

- float ∗ **fvec_new_set** (long n, float val)
  *create a vector initialized with a specified value.*

- int ∗ **ivec_new_set** (long n, int val)
  *create a vector initialized with a specified value.*

- float ∗ **fvec_new_rand** (long n)
  *create a vector initialized with uniformly drawn samples in [0,1)*

- float ∗ **fvec_new_randn** (long n)
  *create a vector initialized with gaussian samples*

- void **fvec_randn_r** (float ∗v, long n, unsigned int seed)
  *same as fvec_randn, with seed for thread-safety*

- float ∗ **fvec_new_rand_r** (long n, unsigned int seed)
  *same as fvec_new_rand, with seed for thread-safety*

- float ∗ **fvec_new_randn_r** (long n, unsigned int seed)
  *same as fvec_new_randn, with seed for thread-safety*

- int ∗ **ivec_new_range** (long a, long b)
  *new vector [a,a+1,...b-1]*

- int ∗ **ivec_new_cpy** (const int ∗v, long n)
  *new vector initialized with another vector*

- float ∗ **fvec_new_cpy** (const float ∗v, long n)
  *new vector initialized with another vector*

- int ∗ **ivec_new_random_perm** (int n)

*random permutation of 0..n-1*

- int ∗ **ivec_new_random_idx** (int n, int k)

  *select k random indexes among n (without repetition)*

- int ∗ **ivec_new_random_perm_r** (int n, unsigned int seed)

  *same as ivec_new_random_perm, thread-safe, with a random seed*

- int ∗ **ivec_new_random_idx_r** (int n, int k, unsigned int seed)

  *same as ivec_new_random_idx, thread-safe with a random seed*

- float ∗ **fvec_resize** (float ∗v, long n)

  *resize a vector (realloc).*

- int ∗ **ivec_resize** (int ∗v, long n)

  *resize a vector (realloc).*

- int ∗ **ivec_new_histogram** (int k, const int ∗v, long n)

  *count occurrences*

- int ∗ **ivec_new_histogram_clip** (int k, int ∗v, long n)

  *same as ivec_new_histogram, but values falling out of range are clipped (counted in the nearest bin)*

- int ∗ **fvec_new_histogram_clip** (float vmin, float vmax, int k, float ∗v, long n)

  *count occurrences: maps [vmin,vmax) to 0..k-1*

- int **ivec_hash** (const int ∗v, long n)

  *compute a hash value for the vector*

- void **ivec_replace** (int ∗v, long n, int val, int replace_val)

  *all occurences of a value by another in a vector*

- long **ivec_count_occurrences** (const int ∗v, long n, int val)

  *count occurences of a value in the vector*

- long **fvec_count_occurrences** (const float ∗v, long n, float val)
- long **fvec_count_lt** (const float ∗v, long n, float val)

  *count the number of values below a threshold*

- long **ivec_count_lt** (const int ∗v, long n, int val)
- long **fvec_count_gt** (const float ∗v, long n, float val)

  *count number of values above a threshold*

- long **ivec_count_gt** (const int ∗v, long n, int val)
- long **fvec_count_inrange** (const float ∗v, long n, float vmin, float vmax)

*count number of values in a range (min <= x < max)*

- long **ivec_count_inrange** (const int *v, long n, int vmin, int vmax)
- long **fvec_count_nan** (const float *v, long n)

  *count the number of nan values*

- long **fvec_count_nonfinite** (const float *v, long n)
- long **fvec_count_0** (const float *val, long n)
- long **fvecs_fsize** (const char *fname, int *d_out, int *n_out)

  *Read the number of vectors in a file and their dimension (vectors of same size).*

- long **ivecs_fsize** (const char *fname, int *d_out, int *n_out)
- long **bvecs_fsize** (const char *fname, int *d_out, int *n_out)
- long **lvecs_fsize** (const char *fname, int *d_out, int *n_out)
- int **ivec_fwrite** (FILE *f, const int *v, int d)

  *write a vector into an open file*

- int **fvec_fwrite** (FILE *f, const float *v, int d)
- int **ivec_fwrite_raw** (FILE *f, const int *v, long d)

  *write a vector without the dimension header*

- int **fvec_fwrite_raw** (FILE *f, const float *v, long d)
- int **bvec_fwrite_raw** (FILE *f, const unsigned char *v, long d)
- int **ivec_write_raw** (const char *fname, const int *v, long d)
- int **fvec_write_raw** (const char *fname, const float *v, long d)
- int **bvec_write_raw** (const char *fname, const unsigned char *v, long d)
- int **ivecs_fwrite** (FILE *f, int d, int n, const int *v)

  *write a set of vectors into an open file*

- int **fvecs_fwrite** (FILE *fo, int d, int n, const float *vf)
- int **ivecs_write** (const char *fname, int d, int n, const int *v)

  *several integer vectors of identical length into an file*

- int **ivecs_write_txt** (const char *fname, int d, int n, const int *v)
- int **fvecs_write** (const char *fname, int d, int n, const float *vf)
- int **fvecs_write_txt** (const char *fname, int d, int n, const float *vf)
- int **fvecs_new_read** (const char *fname, int *d_out, float **vf)

  *load float vectors from file.*

- int **fvecs_new_fread_max** (FILE *f, int *d_out, float **vf, long nmax)
- int **fvecs_new_mmap** (const char *fname, int *d_out, float **vf)

  *mmap vectors from a file.*

- int **ivecs_new_mmap** (const char *fname, int *d_out, int **vi)
- int **bvecs_new_read** (const char *fname, int *d_out, unsigned char **v_out)
- int **lvecs_new_read** (const char *fname, int *d_out, long long **v_out)

- int **b2fvecs_new_read** (const char *fname, int *d_out, float **v_out)
  *load a file of byte vectors, and convert them to float on-the-fly*
- int **fvecs_new_read_sparse** (const char *fname, int d, float **vf_out)
  *reads sparse vectors and return them as dense.*
- int **bvecs_new_from_siftgeo** (const char *fname, int *d_v_out, unsigned char **v_out, int *d_meta_out, float **meta_out)
  *read siftgeo, return as bvecs + metadata as fvecs*
- int **fvecs_read** (const char *fname, int d, int n, float *v)
  *load float vector without allocating memory*
- int **b2fvecs_read** (const char *fname, int d, int n, float *v)
  *read some vector from a bvec file and put them into a fvec vector*
- int **fvecs_read_txt** (const char *fname, int d, int n, float *v)
  *read a vector from a text file (one line per vector)*
- int **fvec_read** (const char *fname, int d, float *a, int o_f)
  *read a single vector from a file*
- int **fvec_fread** (FILE *f, float *v, int d_alloc)
  *load float vectors from an open file.*
- int **fvec_fread_raw** (FILE *f, float *v, long n)
- int **ivec_fread_raw** (FILE *f, int *v, long n)
- int **bvec_fread_raw** (FILE *f, unsigned char *v, long n)
- float * **fvec_new_fread_raw** (FILE *f, long n)
- int * **ivec_new_fread_raw** (FILE *f, long d)
- unsigned char * **bvec_new_fread_raw** (FILE *f, long n)
- float * **fvec_new_read_raw** (const char *fname, long d)
- int * **ivec_new_read_raw** (const char *fname, long d)
- unsigned char * **bvec_new_read_raw** (const char *fname, long d)
- long **fvecs_fread** (FILE *f, float *v, long n, int d_alloc)
  *load a set of n vectors from an open file.*
- long **ivecs_fread** (FILE *f, int *v, long n, int d_alloc)
- long **bvecs_fread** (FILE *f, unsigned char *v, long n, int d_alloc)
- long **lvecs_fread** (FILE *f, long long *v, long n, int d_alloc)
- long **b2fvecs_fread** (FILE *f, float *v, long n)
- int * **ivec_new_read** (const char *fname, int *d_out)
  *read and allocate a an integer vector file*
- int **ivec_fread** (FILE *f, int *v, int d_alloc)
  *read an integer vector file from an open file and return the dimension*

- int **bvec_fread** (FILE ∗f, unsigned char ∗v, int d_alloc)
  *read a byte vector file from an open file and return the dimension*

- int **b2fvec_fread** (FILE ∗f, float ∗v)
- int **lvec_fread** (FILE ∗f, long long ∗v, int d_alloc)
  *read an long vector file from an open file and return the dimension*

- int **ivecs_new_read** (const char ∗fname, int ∗d_out, int ∗∗vi)
  *read several integer vectors from an ivec file.*

- int **ivecs_new_fread_max** (FILE ∗f, int ∗d_out, int ∗∗vi, long nmax)
  *load a few of ivecs*

- void **fvec_print** (const float ∗v, int n)
  *display a float vector*

- void **fvec_fprintf** (FILE ∗f, const float ∗v, int n, const char ∗fmt)
- void **ivec_print** (const int ∗v, int n)
  *display an integer vector*

- void **ivec_fprintf** (FILE ∗f, const int ∗v, int n, const char ∗fmt)
- long **ivec_index** (const int ∗v, long n, int val)
  *find first index of val (return -1 if not found)*

- float ∗ **ivec2fvec** (const int ∗v, long n)
  *cast a vector of int into a (new) vector of floats*

- float ∗ **bvec2fvec** (const unsigned char ∗v, long n)
  *cast a vector of int into a (new) vector of floats*

- void **bvectofvec** (const unsigned char ∗v, float ∗vb, long n)
  *cast a vector of int into a vector of floats.*

- void **fvectodvec** (const float ∗a, double ∗b, long n)
- void **fvec_0** (float ∗v, long n)
  *Set all the components of the vector v to 0.*

- void **ivec_0** (int ∗v, long n)
- void **fvec_nan** (float ∗v, long n)
- void **fvec_rand** (float ∗v, long n)
  *Set all the components of the vector to random values in [0,1[.*

- void **fvec_randn** (float ∗v, long n)
  *Set all the components of the vector to gaussian values.*

- int **fvec_all_0** (const float ∗v, long n)

  *are all values 0?*

- int **ivec_all_0** (const int ∗v, long n)
- int **fvec_all_ge0** (const float ∗v, long n)

  *are all vals >= 0?*

- int **ivec_all_ge0** (const int ∗v, long n)
- int **fvec_all_finite** (const float ∗v, long n)

  *are all values finite?*

- void **fvec_set** (float ∗v, long n, float val)

  *Set all the components of the vector v to the value val.*

- void **ivec_set** (int ∗v, long n, int val)
- void **ivec_cpy** (int ∗vdest, const int ∗vsource, long n)

  *copy the vector from v2 to v1*

- void **fvec_cpy** (float ∗vdest, const float ∗vsource, long n)
- void **bvec_cpy** (unsigned char ∗vdest, const unsigned char ∗vsource, long n)
- void **fvec_incr** (float ∗v, long n, double scal)

  *Increment or decrement a vector by a scalar value.*

- void **fvec_decr** (float ∗v, long n, double scal)
- void **ivec_incr** (int ∗v, long n, int scal)
- void **ivec_decr** (int ∗v, long n, int scal)
- void **fvec_mul_by** (float ∗v, long n, double scal)

  *Multiply or divide a vector by a scalar.*

- void **fvec_div_by** (float ∗v, long n, double scal)
- void **fvec_rdiv_by** (float ∗v, long n, double scal)
- void **fvec_add** (float ∗v1, const float ∗v2, long n)

  *Add or subtract two vectors.*

- void **fvec_sub** (float ∗v1, const float ∗v2, long n)
- void **fvec_rev_sub** (float ∗v1, const float ∗v2, long n)

  *v1 := v2-v1*

- void **fvec_add_mul** (float ∗v1, const float ∗v2, long n, double scal)

  *v1 := v1 + v2 ∗ scal*

- void **fvec_mul** (float ∗v1, const float ∗v2, long n)

  *Component-wise multiplication or division of two vectors (result in v1)*

- void **fvec_div** (float ∗v1, const float ∗v2, long n)
- double **fvec_normalize** (float ∗v, long n, double norm)

*Normalize the vector for the given Minkowski norm.*

- int **fvecs_normalize** (float ∗v, long n, long d, double norm)

  *This function normalize a set of n d-dimensional vectors.*

- void **fvec_round** (float ∗v, long n)
- void **fvec_sqrt** (float ∗v, long n)
- void **fvec_sqr** (float ∗v, long n)
- void **fvec_exp** (float ∗v, long n)
- void **fvec_log** (float ∗v, long n)
- void **fvec_neg** (float ∗v, long n)
- void **fvec_ssqrt** (float ∗v, long n)

  *signed square-root: y = sign(x)∗sqrt(abs(x))*

- void **fvec_spow** (float ∗v, long n, double scal)

  *signed power: y = sign(x) ∗ pow(abs(x), scal)*

- void **fvec_normalize_2stage** (float ∗v, long n, double scal)

  *2-stage normalization (like Lowe's SIFT normalization)*

- void **ivec_add** (int ∗v1, const int ∗v2, long n)
- void **ivec_sub** (int ∗v1, const int ∗v2, long n)
- void **ivec_mul_by** (int ∗v1, long n, int scal)
- void **ivec_mod_by** (int ∗v1, long n, int scal)
- void **ivec_add_scalar** (int ∗v, long n, int scal)
- void **fvec_add_scalar** (float ∗v, long n, float scal)
- int **fvec_purge_nans** (float ∗v, long n, float replace_value)

  *Replace the "Not a number" values by a given value.*

- int **fvec_purge_nonfinite** (float ∗v, long n, float replace_value)
- long **fvec_shrink_nonfinite** (float ∗v, long n)

  *Shrink the vector, removing "Not a number" and inf values.*

- long **fvec_index_nonfinite** (float ∗v, long n)

  *find 1st occurrence of a non-finite element*

- double **fvec_sum** (const float ∗v, long n)

  *compute the sum of the value of the vector*

- long long **ivec_sum** (const int ∗v, long n)
- void **fvec_cumsum** (float ∗v, long n)

  *cumulative sum*

- void **ivec_cumsum** (int ∗v, long n)
- void **fvec_cumdiff** (float ∗v, long n)

  *opposite of cumsum: v[i] := v[i] - v[i-1]*

- void **ivec_cumdiff** (int ∗v, long n)
- double **fvec_product** (const float ∗v, long n)

  *compute the sum of the product of the vector*

- long long **ivec_product** (const int ∗v, long n)
- double **fvec_sum_sqr** (const float ∗v, long n)

  *sum of squared components*

- long long **ivec_sum_sqr** (const int ∗v, long n)
- double **fvec_mean** (const float ∗v, long n)

  *compute the sum of the value of the vector*

- long long **ivec_mean** (const int ∗v, long n)
- double **fvec_norm** (const float ∗v, long n, double norm)

  *compute the norm of a given vector (norm=-1 => infinty norm)*

- double **fvec_norm2sqr** (const float ∗v, long n)

  *compute squared norm 2*

- long **fvec_nz** (const float ∗v, long n)

  *count the number of non-zeros elements*

- long **ivec_nz** (const int ∗v, long n)
- int **fvec_find** (const float ∗v, int n, int ∗∗nzpos_out)

  *compute the positions of the non-null positions.*

- int **ivec_find** (const int ∗v, int n, int ∗∗nzpos_out)
- void **ivec_shuffle** (int ∗v, long n)

  *perform a random permutation on the elements of the vector*

- double **fvec_entropy** (const float ∗pmf, int n)

  *entropy of the probability mass function represented by the vector*

- double **binary_entropy** (double p)

  *entropy of a binary variable*

- double **ivec_unbalanced_factor** (const int ∗hist, long n)
- long **ivec_distance_hamming** (const int ∗v1, const int ∗v2, long n)

  *Return the Hamming distance (i.e., the number of different elements)*

- double **fvec_distance_L2** (const float ∗v1, const float ∗v2, long n)

  *Return the L2 distance between vectors.*

- double **fvec_distance_L1** (const float ∗v1, const float ∗v2, long n)

  *Return the L1 distance between vectors.*

- double **fvec_distance_L2sqr** (const float ∗v1, const float ∗v2, long n)
  *Return the square L2 distance between vectors.*
- double **fvec_inner_product** (const float ∗v1, const float ∗v2, long n)
  *inner product between two vectors*
- int **fvec_to_spfvec** (float ∗v, int n, int ∗∗idx_out, float ∗∗v_out)
  *convert a vector to a sparse vector.*
- int **ivec_to_spivec** (int ∗v, int n, int ∗∗idx_out, int ∗∗v_out)
- float ∗ **spfvec_to_fvec** (int ∗idx, float ∗v, int nz, int n)
  *convert a sparse vector into a full vector*
- int ∗ **spivec_to_ivec** (int ∗idx, int ∗v, int nz, int n)
- float **spfvec_inner_product** (int ∗idx1, float ∗val1, int nz1, int ∗idx2, float ∗val2, int nz2)
  *inner product between two sparse vectors*
- void **ivec_accumulate_slices** (const int ∗v, int ∗sl, int n)
  *on output,*
- void **fvec_map** (const float ∗src, const int ∗map, int n, float ∗dest)
  *mapping operator: dest[i]:=src[map[i]] for i=0..n-1*
- void **ivec_map** (const int ∗src, const int ∗map, int n, int ∗dest)
  *mapping operator: dest[i]:=src[map[i]] for i=0..n-1*
- void **fvec_imap** (const float ∗src, const int ∗imap, int n, float ∗dest)
  *inverse mapping operator: dest[imap[i]]:=src[i] for i=0..n-1*
- void **fvec_splat_add** (const float ∗a, int n, const int ∗assign, float ∗accu)
  *for i=0..n-1, do accu[assign[i]] += a[i]*
- void **fvec_isplat_add** (const float ∗a, int n, const int ∗assign, float ∗accu)
  *for i=0..n-1, do accu[i] += a[assign[i]]*
- int ∗ **ivec_repeat_with_inc** (const int ∗a, int n, int nrepeat, int inc)
  *return input vector duplicated n times, with a value added each time*
- void **fvec_cpy_subvectors** (const float ∗v, int ∗idx, int d, int nout, float ∗vout)
  *Copy the set of nout vectors in v (seen as a set of vectors), indexed by idx, in vout.*
- void **b2fvec_cpy_subvectors** (const unsigned char ∗v, int ∗idx, int d, int nout, float ∗vout)
- void **ivec_to_fvec** (const int ∗v, float ∗f, long n)
  *simple type conversion*

### 3.7.1 Detailed Description

Vectors are represented as C arrays of basic elements. Functions operating on them are prefixed with:

ivec_: basic type is int

fvec_: basic type is float

Vector sizes are passed explicitly, as long int's to allow for large arrays on 64 bit machines. Vectors can be free'd with free().

Arrays of vectors are stored contiguously in memory. An array of n float vectors of dimension d is

float *fv

The i'th element of vector j of vector array vf, where 0 <= i < d and 0 <= j < n is

vf[ j * d + i ]

It can also be seen as a column-major matrix of size d, n.

### 3.7.2 Function Documentation

#### 3.7.2.1 float* fvec_new ( long *n* )

Alloc a new aligned vector of floating point values -- to be de-allocated with free.

Some operations may be faster if input arrays are allocated with this function (data is suitably aligned).

References memalign().

Referenced by b2fvecs_new_read(), bvec2fvec(), eigs_reorder(), fmat_new(), fmat_new_covariance(), fmat_new_get_row(), fmat_new_pca(), fmat_new_pca_from_covariance(), fmat_new_transp(), fmat_solve_ls_t(), fvec_new_cpy(), fvec_new_nan(), fvec_new_rand(), fvec_new_rand_r(), fvec_new_randn(), fvec_new_randn_r(), fvec_to_spfvec(), fvecs_new_read_sparse(), gmm_compute_p(), gmm_fisher(), gmm_learn(), hadamard(), ivec2fvec(), kmeans(), kmlsh_match_xvec(), knn(), knn_full(), knn_reorder_shortlist(), merge_ordered_sets(), nn(), nn_thread(), pca_online_accu(), pca_online_cov(), and pca_online_new().

#### 3.7.2.2 int* ivec_new ( long *n* )

Alloc an int array -- to be de-allocated with free.

Referenced by eigs_reorder(), fvec_find(), fvec_to_spfvec(), fvecs_new_read_sparse(), gmm_learn(), ivec_new_cpy(), ivec_new_random_idx_r(), ivec_new_range(), ivec_new_set(), ivec_repeat_with_inc(), kmeans(), kmlsh_match_xvec(), knn_reorder_shortlist(), and merge_ordered_sets().

#### 3.7.2.3 unsigned char* bvec_new ( long *n* )

Alloc an byte array -- to be de-allocated with free.

References memalign().

#### 3.7.2.4 double* dvec_new ( long *n* )

Alloc an int array -- to be de-allocated with free.

References memalign().

#### 3.7.2.5 float* fvec_new_0 ( long *n* )

create a vector initialized with 0's.

Referenced by fmat_new_covariance(), gmm_learn(), pca_online_new(), and spfvec_-to_fvec().

#### 3.7.2.6 int* ivec_new_0 ( long *n* )

create a vector initialized with 0's.

Referenced by fvec_new_histogram_clip(), ivec_new_histogram(), and ivec_new_histogram_-clip().

#### 3.7.2.7 long long* lvec_new_0 ( long *n* )

create a vector initialized with 0's.

#### 3.7.2.8 float* fvec_new_set ( long *n*, float *val* )

create a vector initialized with a specified value.

#### 3.7.2.9 int* ivec_new_set ( long *n*, int *val* )

create a vector initialized with a specified value.

References ivec_new().

#### 3.7.2.10 float* fvec_resize ( float * *v*, long *n* )

resize a vector (realloc).

Usage: v = fvec_resize (v, n).

#### 3.7.2.11 int* ivec_resize ( int * v, long n )

resize a vector (realloc).

Usage: v = fvec_resize (v, n).

#### 3.7.2.12 int* ivec_new_histogram ( int k, const int * v, long n )

count occurrences

**Parameters**

| | |
|---|---|
| *k* | is the range of the values that may be encountered (assuming start at 0). Values outside the range trigger an assertion! |
| *v* | is the vector of values to be histrogramized, of length n |

References ivec_new_0().

#### 3.7.2.13 int* fvec_new_histogram_clip ( float vmin, float vmax, int k, float * v, long n )

count occurrences: maps [vmin,vmax) to 0..k-1

**Parameters**

| | |
|---|---|
| *vmin* | min val of range |
| *vmax* | max val of range |
| *k* | nb of bins |
| *v* | is the vector of values to be histrogramized, of length n |

References ivec_new_0().

#### 3.7.2.14 long fvecs_fsize ( const char * fname, int * d_out, int * n_out )

Read the number of vectors in a file and their dimension (vectors of same size).

Output the number of bytes of the file.

#### 3.7.2.15 int fvecs_new_read ( const char * fname, int * d_out, float ** vf )

load float vectors from file.

Returns nb of vectors read, or <0 on error

#### 3.7.2.16 int fvecs_new_mmap ( const char * fname, int * d_out, float ** vf )

mmap vectors from a file.

The returned memory area is read-only.

WARNING, the i'th element of vector j of vector array vf, where 0 <= i < d and 0 <= j < n is

vf[ j ∗ (d + 1) + i ]

(mind the d+1) the file remains open and there is no deallocation function (yet)

#### 3.7.2.17 int fvecs_new_read_sparse ( const char ∗ *fname,* int *d,* float ∗∗ *vf_out* )

reads sparse vectors and return them as dense.

d must be known

References fvec_new(), ivec_new(), and spfvec_to_fvec().

#### 3.7.2.18 int fvecs_read ( const char ∗ *fname,* int *d,* int *n,* float ∗ *v* )

load float vector without allocating memory

Fills n∗d array with as much vectors read from fname as possible. Returns nb of vectors read, or <0 on error.

#### 3.7.2.19 int fvec_read ( const char ∗ *fname,* int *d,* float ∗ *a,* int *o_f* )

read a single vector from a file

Fill a with a single float vector from fname offset o_f into file a Returns <0 on error

References fvec_fread().

#### 3.7.2.20 int fvec_fread ( FILE ∗ *f,* float ∗ *v,* int *d_alloc* )

load float vectors from an open file.

Return the dimension

Referenced by fvec_read().

#### 3.7.2.21 long fvecs_fread ( FILE ∗ *f,* float ∗ *v,* long *n,* int *d_alloc* )

load a set of n vectors from an open file.

Return the number of vectors that have been read.

#### 3.7.2.22 int ivecs_new_read ( const char ∗ *fname,* int ∗ *d_out,* int ∗∗ *vi* )

read several integer vectors from an ivec file.

Return number read

References ivecs_new_fread_max().

Referenced by ivec_new_read().

#### 3.7.2.23 void bvectofvec ( const unsigned char * *v*, float * *vb*, long *n* )

cast a vector of int into a vector of floats.

No internal allocation.

#### 3.7.2.24 void fvec_add ( float * *v1*, const float * *v2*, long *n* )

Add or subtract two vectors.

The result is stored in v1.

Referenced by pca_online_accu().

#### 3.7.2.25 double fvec_normalize ( float * *v*, long *n*, double *norm* )

Normalize the vector for the given Minkowski norm.

The function return the norm of the original vector. If the vector is all 0, it will be filled with NaNs. This case can be identified when the return value is 0. Infinty norm can be obtained with norm=-1

References fvec_mul_by(), and fvec_norm().

Referenced by fmat_svd_partial_full(), and fvec_normalize_2stage().

#### 3.7.2.26 int fvecs_normalize ( float * *v*, long *n*, long *d*, double *norm* )

This function normalize a set of n d-dimensional vectors.

It returns the number of vectors whose norms was 0 (for which the normalization has put some NaN values).

References fvec_mul_by(), and fvec_norm().

#### 3.7.2.27 long fvec_shrink_nonfinite ( float * *v*, long *n* )

Shrink the vector, removing "Not a number" and inf values.

Returns new size

#### 3.7.2.28 int fvec_find ( const float * *v*, int *n*, int ** *nzpos_out* )

compute the positions of the non-null positions.

return the number of non-zeros positions.

References fvec_nz(), and ivec_new().

#### 3.7.2.29 int fvec_to_spfvec ( float * v, int n, int ** idx_out, float ** v_out )

convert a vector to a sparse vector.

Return the number of non-zeros positions

References fvec_new(), fvec_nz(), and ivec_new().

#### 3.7.2.30 void ivec_accumulate_slices ( const int * v, int * sl, int n )

on output,

sl_out[0] = v[ 0] + ... + v[sl[0]-1]

sl_out[i] = sl_out[i-1] + v[sl[i-1]] + ... + v[sl[i]-1] for 0<i<n

#### 3.7.2.31 int* ivec_repeat_with_inc ( const int * a, int n, int nrepeat, int inc )

return input vector duplicated n times, with a value added each time

**Parameters**

| | |
|---|---|
| *nrepeat* | nb of times to repeat input vector |
| *inc* | inc*i is added to all elements of i^th repeated vector |

**Returns**

vector of size n*nrepeat

References ivec_cpy(), and ivec_new().

#### 3.7.2.32 void fvec_cpy_subvectors ( const float * v, int * idx, int d, int nout, float * vout )

Copy the set of nout vectors in v (seen as a set of vectors), indexed by idx, in vout.

**Parameters**

| | |
|---|---|
| *v* | the set of input vectors |
| *idx* | the indexes of the vectors to be copied |
| *d* | vectors' dimensionality |
| *nout* | number of vectors copied |
| *vout* | output vector (must be allocated externally with size nout) |

Referenced by kmlsh_match_xvec().

## 3.8 Linearalgebra

### Typedefs

- typedef struct **arpack_eigs_t arpack_eigs_t**
  *thin wrapper around the partial eigenvalue Arpack function*

### Functions

- int **eigs_sym** (int d, const float ∗m, float ∗eigval, float ∗eigvec)
  *Compute the eigenvalues and eigvectors of a symmetric matrix m.*
- int **geigs_sym** (int d, const float ∗a, const float ∗b, float ∗eigval, float ∗eigvec)
  *Solve a generalized eigenvector problem.*
- void **eigs_reorder** (int d, float ∗eigval, float ∗eigvec, int criterion)
  *Re-ordering of the eigenvalues and eigenvectors for a given criterion.*
- int **eigs_sym_part** (int d, const float ∗m, int nev, float ∗eigval, float ∗eigvec)
  *same as eigs_sym, but returns only part of the vectors*
- **arpack_eigs_t** ∗ **arpack_eigs_begin** (int n, int nev)
  *begin partial eigenvalue computation -- user should have a matrix multiplication function at hand*
- int **arpack_eigs_step** (**arpack_eigs_t** ∗, float ∗∗x_out, float ∗∗y_out)
  *one iteration*
- int **arpack_eigs_end** (**arpack_eigs_t** ∗, float ∗sout, float ∗vout)
  *result and cleanup*

### 3.8.1 Function Documentation

#### 3.8.1.1 int eigs_sym ( int *d*, const float ∗ *m*, float ∗ *eigval*, float ∗ *eigvec* )

Compute the eigenvalues and eigvectors of a symmetric matrix m.

**Parameters**

| | |
|---|---|
| *d* | dimension of the square matrix m |
| *m(d,d)* | the matrix (first elements are the first row) |
| *eigval(d)* | on output the eigenvalues (unsorted) |
| *eigvec(d,d)* | on output, eigenvector j is eigvec(:,j) |

**Returns**

=0 for success, else an error code (see info in lapack's dsygv documentation)

the vectors eigval and eigvec must be allocated externally

References memalign().

#### 3.8.1.2 void eigs_reorder ( int *d,* float * *eigval,* float * *eigvec,* int *criterion* )

Re-ordering of the eigenvalues and eigenvectors for a given criterion.

**Parameters**

| | |
|---|---|
| *criterion* | equal to 0 for ascending order, descending otherwise |

References fvec_new(), fvec_sort_index(), and ivec_new().

#### 3.8.1.3 int eigs_sym_part ( int *d,* const float * *m,* int *nev,* float * *eigval,* float * *eigvec* )

same as eigs_sym, but returns only part of the vectors

**Parameters**

| | |
|---|---|
| *nev* | nb of eigenvectors/values to return |
| *eigval(nev)* | the n eigenvalues |
| *eigvec(d,nev)* | eigenvector j is eigvec(:,j) |

**Returns**

=0 for success, else an error code (see info in ssaupd or ierr in sseupd from arpack's documentation)

References arpack_eigs_begin(), arpack_eigs_end(), arpack_eigs_step(), and count_cpu().

Referenced by pca_online_complete_part().

#### 3.8.1.4 arpack_eigs_t* arpack_eigs_begin ( int *n,* int *nev* )

begin partial eigenvalue computation -- user should have a matrix multiplication function at hand

**Parameters**

| | |
|---|---|
| *n* | dimension of the square matrix |
| *nev* | nb of eigenvectors/values to return |

Referenced by eigs_sym_part(), and fmat_svd_partial_full().

#### 3.8.1.5 int arpack_eigs_step ( arpack_eigs_t * , float ** *x_out*, float ** *y_out* )

one iteration

**Parameters**

| | |
|---|---|
| *x* | *x_out is the array that should be multiplied (size n) |
| *y* | *y_out is result of the multiplication (size n) |

**Returns**

>0 compute y := A * x 0: stop iteration <0, error (call arpack_eigs_end for cleanup)

Referenced by eigs_sym_part(), and fmat_svd_partial_full().

#### 3.8.1.6 int arpack_eigs_end ( arpack_eigs_t * , float * *sout*, float * *vout* )

result and cleanup

**Parameters**

| | |
|---|---|
| *sout* | eigenvalues |
| *vout* | eigenvectors |

**Returns**

nb of filled-in eigenvals and eigenvecs (may be below nev if some did not converge)

Referenced by eigs_sym_part(), and fmat_svd_partial_full().

## 3.9 Clustering

### Defines

- #define **KMEANS_QUIET** 0x10000
- #define **KMEANS_INIT_BERKELEY** 0x20000
- #define **KMEANS_NORMALIZE_CENTS** 0x40000
- #define **KMEANS_INIT_RANDOM** 0x80000
- #define **KMEANS_INIT_USER** 0x100000
- #define **KMEANS_L1** 0x200000
- #define **KMEANS_CHI2** 0x400000

### Functions

- float **kmeans** (int d, int n, int k, int niter, const float *v, int flags, long seed, int redo, float *centroids, float *dis, int *assign, int *nassign)

*Compute the k-means centroids.*

- float * **clustering_kmeans** (int n, int d, const float *points, int k, int nb_iter_max, double normalize)

  *simplified call*

- float * **clustering_kmeans_assign** (int n, int d, const float *points, int k, int nb_iter_max, double normalize, int **clust_assign_out)

  *Same as kmeans, but generate in addition the assignment performed on the input set.*

- float * **clustering_kmeans_assign_with_score** (int n, int d, const float *points, int k, int nb_iter_max, double normalize, int n_thread, double *score_out, int **clust_assign_out)
- double **spectral_clustering** (int d, int n, int k, double sigma, int niter, const float *v, int nt, int seed, int nredo, int *assign, int *nassign)

### 3.9.1 Function Documentation

#### 3.9.1.1 float kmeans ( int *d*, int *n*, int *k*, int *niter*, const float * *v*, int *flags*, long *seed*, int *redo*, float * *centroids*, float * *dis*, int * *assign*, int * *nassign* )

Compute the k-means centroids.

**Parameters**

| | |
|---|---|
| *v(d,n)* | vectors to cluster |
| *centroids(d,k)* | output centroids (input centroids here if KMEANS_INIT_USER) |
| *flags* | a set of computation parameters:<br>• flags & 0xffff : use this many threads to compute<br>• flags & KMEANS_QUIET: suppress kmeans output<br>• flags & KMEANS_INIT_RANDOM: random initialization<br>• flags & KMEANS_NORMALIZE_CENTS: normalize centroids to L2=1 after they are computed<br>• flags & KMEANS_INIT_USER: the user gives the initialization<br>• flags & KMEANS_L1: L1 distance kmeans<br>• flags & KMEANS_CHI2: chi-squared distance kmeans -> provided by user with parameter centroids_out) |
| *seed* | random seed for intialization (used only if !=0) |
| *redo* | perform clustering this many times and keep clusters with smallest quantization error |
| *dis(n)* | squared distance to assigned centroid of each input vector (may be NULL) |
| *assign(n)* | index of assigned centroid in 0..k-1 (may be NULL) |
| *nassign(k)* | nb of vectors assigned to each centroid (may be NULL) |

**Returns**

final quantization error

References fvec_new(), and ivec_new().

Referenced by gmm_learn().

## 3.10 Machinedep

### Data Structures

- struct **malloc_stats_t**

  *trace all mallocs between two function calls.*

### Functions

- int **count_cpu** (void)

  *Return the number of cores.*

- double **log2** (double x)
- void * **memalign** (size_t ignored, size_t nbytes)

  *allocate memory such that the pointer is aligned*

- void **malloc_stats_begin** (void)
- **malloc_stats_t malloc_stats_end** (void)
- double **getmillisecs** ()

  *return a timestamp, which is useful to measure elapsed time*

- void **compute_tasks** (int n, int nthread, void(*task_fun)(void *arg, int tid, int i), void *task_arg)

  *exectutes a set of tasks in parallel using a thread pool*

### 3.10.1 Function Documentation

#### 3.10.1.1 int count_cpu ( void )

Return the number of cores.

Referenced by clustering_kmeans(), clustering_kmeans_assign(), eigs_sym_part(), fmat_new_pca_part(), and fmat_svd_partial().

#### 3.10.1.2 void compute_tasks ( int *n,* int *nthread,* void(∗)(void ∗arg, int tid, int i) *task_fun,* void ∗ *task_arg* )

exectutes a set of tasks in parallel using a thread pool

**Parameters**

| | |
|---|---|
| *n* | number of tasks to execute |
| *nthread* | number of threads that will run the tasks |
| *task_fun* | this callback will be called with<br>• arg = task_arg<br>• tid = identifier of the thread in 0..nthread-1<br>• i = call number in 0..n-1 |

Referenced by compute_cross_distances_thread(), gmm_compute_p_thread(), and knn_-full_thread().

# Chapter 4

# Data Structure Documentation

## 4.1 fbinheap_s Struct Reference

Binary heap used as a maxheap.

`#include <binheap.h>`

### Data Fields

- float * **val**

  *valid values are val[1] to val[k]*

- int * **label**

  *idem for labels*

- int **k**

  *number of elements stored*

- int **maxk**

  *maximum number of elements*

### 4.1.1 Detailed Description

Binary heap used as a maxheap. Element (label[1],val[1]) always contains the maximum value of the binheap.

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

- /Users/hjegou/tmp/yael_v300/yael/binheap.h

## 4.2 gmm_s Struct Reference

Gaussian Mixture Model (GMM) implementation.

`#include <gmm.h>`

### Data Fields

- int **d**
  *vector dimension*
- int **k**
  *number of mixtures*
- float * **w**
  *weights of the mixture elements (size k)*
- float * **mu**
  *centroids (d-by-k)*
- float * **sigma**
  *diagonal of the covariance matrix (d-by-k)*

### 4.2.1 Detailed Description

Gaussian Mixture Model (GMM) implementation.

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

- /Users/hjegou/tmp/yael_v300/yael/gmm.h

## 4.3 hkm_s Struct Reference

the structure used for the quantization

`#include <hkm.h>`

### Data Fields

- int **nlevel**
- int **bf**
- int **k**
- int **d**
- float ** **centroids**

### 4.3.1 Detailed Description

the structure used for the quantization

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

- /Users/hjegou/tmp/yael_v300/yael/hkm.h

## 4.4 kmlsh_idx_s Struct Reference

A structure containing the pre-processed data (tables of quantized indexes) for a set of vectors.

`#include <kmlsh.h>`

### Data Fields

- int **nhash**
- int **n**
- int **nclust**
- int * **perm**
- int * **boundaries**

### 4.4.1 Detailed Description

A structure containing the pre-processed data (tables of quantized indexes) for a set of vectors.

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

- /Users/hjegou/tmp/yael_v300/yael/kmlsh.h

## 4.5 kmlsh_s Struct Reference

The structure that contains the parameters of the KM-LSH.

`#include <kmlsh.h>`

### Data Fields

- int **nhash**
- int **d**
- int **nclust**
- float ** **centroids**

### 4.5.1 Detailed Description

The structure that contains the parameters of the KM-LSH.

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

- /Users/hjegou/tmp/yael_v300/yael/kmlsh.h

## 4.6 malloc_stats_t Struct Reference

trace all mallocs between two function calls.

`#include <machinedeps.h>`

### Data Fields

- int **n_alloc**
- int **n_free**
- int **n_realloc**
- size_t **delta_alloc**
- size_t **max_alloc**
- int **n_untracked_frees**

### 4.6.1 Detailed Description

trace all mallocs between two function calls. Intended to replace struct mallinfo that does not seem to work. Implemented only for Linux. Includes inefficient code that should not be relied on while profiling.

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

- /Users/hjegou/tmp/yael_v300/yael/machinedeps.h

## 4.7 nnlist_s Struct Reference

A structure to handle the list of KNN.

`#include <kmlsh.h>`

### Data Fields

- long **n**
- long **k**
- int * **idx**
- float * **dis**

### 4.7.1 Detailed Description

A structure to handle the list of KNN.

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

- /Users/hjegou/tmp/yael_v300/yael/kmlsh.h

## 4.8 pca_online_s Struct Reference

### Data Fields

- int **n**
- int **d**
- float * **mu**
- float * **cov**
- float * **eigvec**
- float * **eigval**

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

- /Users/hjegou/tmp/yael_v300/yael/matrix.h

# Index