NNMF CUDA

1.0.0

Generated by Doxygen 1.9.5

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 csv_reader Class Reference

`#include <csv_reader.h>`

Public Member Functions

- csv_reader ()
- ~csv_reader ()

Static Public Member Functions

- static int countLines (std::string filename)
- static int countColumns (std::string filename, const char delimiter)

3.1.1 Constructor & Destructor Documentation

3.1.1.1 csv_reader()

`csv_reader::csv_reader ( )`

3.1.1.2 ~csv_reader()

`csv_reader::~csv_reader ( )`

3.1.2 Member Function Documentation

3.1.2.1 countColumns()

```
int csv_reader::countColumns (
            std::string filename,
            const char delimiter )  [static]
```

3.1.2.2 countLines()

```
int csv_reader::countLines (
            std::string filename )  [static]
```

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

- include/io/csv_reader.h
- src/io/csv_reader.cpp

3.2 matrix Class Reference

```
#include <matrix.h>
```

Public Member Functions

- matrix ()
- ~matrix ()
- matrix (int nrows, int ncols)
- matrix (int nrows, int ncols, bool randomize)
- int get_num_rows ()
- int get_num_cols ()
- float at (int row, int col)
- void set (int row, int col, float val)

Static Public Member Functions

- static void mat_mult (matrix *A, matrix *B, matrix *out)

3.2.1 Constructor & Destructor Documentation

3.2.1.1 matrix() [1/3]

```
matrix::matrix ( )
```

3.2.1.2 ∼matrix()

```
matrix::∼matrix ( )
```

3.2.1.3 matrix() [2/3]

```
matrix::matrix (
            int nrows,
            int ncols )
```

3.2.1.4 matrix() [3/3]

```
matrix::matrix (
            int nrows,
            int ncols,
            bool randomize )
```

3.2.2 Member Function Documentation

3.2.2.1 at()

```
float matrix::at (
            int row,
            int col )
```

3.2.2.2 get_num_cols()

```
int matrix::get_num_cols ( )
```

3.2.2.3 get_num_rows()

```
int matrix::get_num_rows ( )
```

3.2.2.4 mat_mult()

```
void matrix::mat_mult (
            matrix * A,
            matrix * B,
            matrix * out )  [static]
```

3.2.2.5 set()

```
void matrix::set (
            int row,
            int col,
            float val )
```

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

- include/math/matrix.h
- src/math/matrix.cpp

3.3 nnmf Class Reference

```
#include <nnmf.h>
```

Public Member Functions

- nnmf ()
- ~nnmf ()
- bool Initialize ()

3.3.1 Constructor & Destructor Documentation

3.3.1.1 nnmf()

```
nnmf::nnmf ( )
```

3.3.1.2 ~nnmf()

```
nnmf::~nnmf ( )
```

3.3.2 Member Function Documentation

3.3.2.1 Initialize()

```
bool nnmf::Initialize ( )
```

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

- include/math/nnmf.h
- src/math/nnmf.cpp

Chapter 4

File Documentation

4.1 build/CMakeFiles/3.24.1/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- #define __has_include(x) 0
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define C_VERSION

Functions

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

Variables

- char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * info_language_standard_default
- const char * info_language_extensions_default

4.1.1 Macro Definition Documentation

4.1.1.1 __has_include

```
#define __has_include(
              x ) 0
```

4.1.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

4.1.1.3 C_VERSION

```
#define C_VERSION
```

4.1.1.4 COMPILER_ID

```
#define COMPILER_ID ""
```

4.1.1.5 DEC

```
#define DEC(
              n )
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

4.1.1.6 HEX

```
#define HEX(
              n )
```

Value:

```
  ('0' + ((n)>>28 & 0xF)), \
  ('0' + ((n)>>24 & 0xF)), \
  ('0' + ((n)>>20 & 0xF)), \
  ('0' + ((n)>>16 & 0xF)), \
  ('0' + ((n)>>12 & 0xF)), \
  ('0' + ((n)>>8  & 0xF)), \
  ('0' + ((n)>>4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

4.1.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

4.1.1.8 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

4.1.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

4.1.2 Function Documentation

4.1.2.1 main()

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

4.1.3 Variable Documentation

4.1.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

4.1.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

4.1.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
  "OFF"
"]"
```

4.1.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
=
  "INFO" ":" "standard_default[" C_VERSION "]"
```

4.1.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

4.2 build/CMakeFiles/3.24.1/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- #define __has_include(x) 0
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define CXX_STD __cplusplus

Functions

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

Variables

- char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * info_language_standard_default
- const char * info_language_extensions_default

4.2.1 Macro Definition Documentation

4.2.1.1 __has_include

```
#define __has_include(
              x ) 0
```

4.2.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

4.2.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

4.2.1.4 CXX_STD

```
#define CXX_STD __cplusplus
```

4.2.1.5 DEC

```
#define DEC(
              n )
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

4.2.1.6 HEX

```
#define HEX(
              n )
```

Value:

```
  ('0' + ((n)»28 & 0xF)), \
  ('0' + ((n)»24 & 0xF)), \
  ('0' + ((n)»20 & 0xF)), \
  ('0' + ((n)»16 & 0xF)), \
  ('0' + ((n)»12 & 0xF)), \
  ('0' + ((n)»8  & 0xF)), \
  ('0' + ((n)»4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

4.2.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

4.2.1.8 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

4.2.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

4.2.2 Function Documentation

4.2.2.1 main()

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

4.2.3 Variable Documentation

4.2.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

4.2.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

4.2.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
  "OFF"
"]"
```

4.2.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
= "INFO" ":" "standard_default["
  "98"
"]"
```

4.2.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

4.3 examples/nnmf.m File Reference

Typedefs

- using x = rand(100, 20) ∗rand(20, 50)

Functions

- A, K, 'PARAM1', val1, 'PARAM2', val2,... pairs:fields:NOTE:Examples: (default 1 following,[fields] the default PARALLELSTATS,[NOTE] depending on your installation and preferences TRUE,[Examples] meas, 2 nnmf)
- biplot (max(w(:)) ∗h', 'VarLabels',{ 'sl' 'sw' 'pl' 'pw'}, 'positive', true)
- axis ([0 12 0 12])
- legend (hLines) % % % Try a few iterations at several replicates using the % % multiplicative algorithm
- id Reference: (2007 varargin)
- end narginchk (2, Inf)
- if ∼isscalar (k)||∼isnumeric(k)||k< 1||k >min(m
- checkmatrices (a, w0, h0, k)
- if ∼isscalar (tries)||∼isnumeric(tries)||tries< 1||tries∼
- Special if K is full rank we know the answer if isempty (w0) &&isempty(h0) if k
- Suppress undesired warnings if usePool On workers and client pctRunOnAll internal stats parallel muteParallelStore ('rankDeficientMatrix',... warning('off', 'MATLAB:rankDeficientMatrix'))
- ... 'workerID', num2cell(1:poolsz distributeToPool ()
- Periodic reports behave differently in parallel than they do in serial computation(which is the baseline). % We advise the user of the difference. warning(message('stats Leave formatted by t UI strings untranslated fprintf (' worker\t rep\t iteration\t rms resid\t|delta x|\n')
- else if useParallel warning (message('stats:nnmf:displayParallel'))
- end fprintf (' rep\t iteration\t rms resid\t|delta x|\n')
- catch ME Revert warning setting for rankDeficientMatrix to value prior to nnmf if usePool On workers and on client pctRunOnAll warning (internal.stats.parallel.statParallelStore('rankDeficientMatrix').state, 'MATLAB↩ :rankDeficientMatrix')
- else On client warning (ws)
- end rethrow (ME)
- '%s\n', getString(message('stats:nnmf:FinalRMSResidual', sprintf('%g', normbest fprintf ()
- if any(hlen==0) warning(message('stats hlen (hlen==0)
- S getGlobalStream ()
- end if (∼isempty(h0) &&iter==1) whtry
- a numel ()
- Check for convergence if j if delta<=tolx break;elseif dnorm0-dnorm<=tolfun ∗max(1, dnorm0) break;elseif j==maxiter break end end if dispnum >2 % 'iter' if usePool fprintf(dispfmt, labindx, repnum, j, dnorm, delta);else fprintf(dispfmt, repnum, j, dnorm, delta);end end % Remember previous iteration results dnorm0=dnorm;w0=w;h0=h;endif dispnum > final or iter if usePool fprintf (dispfmt, labindx, repnum, j, dnorm, delta)
- else fprintf (dispfmt, repnum, j, dnorm, delta)
- a size ()

Variables

- function [wbest, hbest, normbest]
- hLines = gscatter(w(:,1),w(:,2),species)
- hold on
- hold off
- opt = statset('maxiter',5,'display','final')
- See also BIPLOT = nnmf(x,5,'w0',w,'h0',h,'opt',opt,'alg','als')
- See also PCA
- See also STATSET
- if n k
- end Process optional arguments pnames = {'algorithm' 'w0' 'h0' 'replicates' 'options'}
- dflts = {'als' [ ] [ ] 1 [ ] }
- Check optional arguments alg = internal.stats.getParamVal(alg,{'mult' 'als'},'ALGORITHM')
- ismult = strncmp('mult',alg,numel(alg))

- end defaultopt = statset('nnmf')
- tolx = statget(options,'TolX',defaultopt,'fast')
- tolfun = statget(options,'TolFun',defaultopt,'fast')
- maxiter = statget(options,'MaxIter',defaultopt,'fast')
- dispopt = statget(options,'Display',defaultopt,'fast')
- dispnum = dispnum - 1
- usePool = useParallel && poolsz>0
- Special case
- h0 = eye(k)
- end end Define the function that will perform one iteration of the loop inside smartFor loopbody = @loopBody
- else On client ws = warning('off','MATLAB:rankDeficientMatrix')
- end end try whbest
- end normbest = whbest{1}
- wbest = whbest{3}
- hbest = whbest{4}
- Then order by w [~, idx] = sort(sum(wbest.^2,1),'descend')
- end whtry is a temporary variable and hence needs to be reinitialized at start of each loop whtry = cell(4,1)
- end Perform a factorization [whtry{3}, whtry{4}, whtry{1}]
- cellout = whtry
- sqrteps = sqrt(eps)
- Display progress For parallel computing
- Display progress For parallel the replicate number will be displayed under the worker performing the replicate if dispnum final or iter if usePool labindx = internal.stats.parallel.workerGetValue('workerID')
- dispfmt = '%8d\t%8d\t%8d\t%14g\t%14g\n'
- end end for j
- h = max(0,h0 .* (numer ./ ((w0'*w0)*h0 + eps(numer))))
- numer = a*h'
- end Get norm of difference and max change in factors d = a - w*h
- dnorm = sqrt(sum(sum(d.^2))/nm)
- dw = max(max(abs(w-w0) / (sqrteps+max(max(abs(w0))))))
- dh = max(max(abs(h-h0) / (sqrteps+max(max(abs(h0))))))
- delta = max(dw,dh)

4.3.1 Typedef Documentation

4.3.1.1 x

```
using x = rand(100,20)*rand(20,50)
```

4.3.2 Function Documentation

4.3.2.1 axis()

```
axis ( )
```

4.3.2.2 biplot()

```
biplot (
             max(w(:))  *h' ,
             'VarLabels' ,
             { 'sl' 'sw' 'pl' 'pw'} ,
             'positive' ,
             true  )
```

4.3.2.3 checkmatrices()

```
checkmatrices (
             a ,
             w0 ,
             h0 ,
             k  )
```

4.3.2.4 distributeToPool()

```
...  'workerID', num2cell(1:poolsz distributeToPool ( )  [virtual]
```

4.3.2.5 fprintf() [1/5]

```
end fprintf (
             ' rep\t iteration\t rms resid\t|delta x|\n'  )
```

4.3.2.6 fprintf() [2/5]

```
Periodic reports behave differently in parallel than they do in serial computation(which is
the baseline).  % We advise the user of the difference.  warning(message( 'stats Leave formatted
by t UI strings untranslated fprintf (
             ' worker\t rep\t iteration\t rms resid\t|delta x|\n'  )
```

4.3.2.7 fprintf() [3/5]

```
'%s\n', getString(message( 'stats:nnmf:FinalRMSResidual', sprintf('%g', normbest fprintf ( )
[virtual]
```

4.3.2.8 fprintf() [4/5]

```
Check for convergence if j if delta<=tolx break; elseif dnorm0-dnorm<=tolfun *max(1, dnorm0)
break; elseif j==maxiter break end end if dispnum >2 % 'iter' if usePool fprintf(dispfmt,
labindx, repnum, j, dnorm, delta); else fprintf(dispfmt, repnum, j, dnorm, delta); end end %
Remember previous iteration results dnorm0=dnorm; w0=w; h0=h;endif dispnum > final or iter if
usePool fprintf (
             dispfmt ,
             labindx ,
             repnum ,
             j ,
             dnorm ,
             delta  )
```

4.3.2.9 fprintf() [5/5]

```
else fprintf (
             dispfmt ,
             repnum ,
             j ,
             dnorm ,
             delta  )
```

4.3.2.10 getGlobalStream()

```
S getGlobalStream ( )  [virtual]
```

4.3.2.11 hlen()

```
if any(hlen==0) warning(message( 'stats hlen (
             hlen  = =0 )
```

4.3.2.12 if()

```
end if (
             ~isempty(h0) && iter = =1 )
```

4.3.2.13 isempty()

```
Special if K is full rank we know the answer if isempty (
             w0  ) &&
```

4.3.2.14 legend()

```
legend (
             hLines  )
```

4.3.2.15 muteParallelStore()

```
Suppress undesired warnings if usePool On workers and client pctRunOnAll internal stats parallel
muteParallelStore (
             'rankDeficientMatrix' ,
             ...    warning 'off', 'MATLAB:rankDeficientMatrix' )
```

4.3.2.16 narginchk()

```
end narginchk (
             2 ,
             Inf  )
```

4.3.2.17 numel()

```
a numel ( )  [virtual]
```

4.3.2.18 pairs:fields:NOTE:Examples:()

```
A, K, 'PARAM1', val1, 'PARAM2', val2,...  pairs:fields:NOTE:Examples:  (
             default 1 following,
             [fields] the default PARALLELSTATS,
             [NOTE] depending on your installation and preferences TRUE,
             [Examples] meas ,
             2 nnmf )  [virtual]
```

4.3.2.19 Reference:()

```
id Reference:  (
             2007 varargin )  [virtual]
```

4.3.2.20 rethrow()

```
end rethrow (
             ME  )
```

4.3.2.21 size()

```
a size ( )  [virtual]
```

4.3.2.22 warning() [1/3]

```
end Revert warning setting for rankDeficientMatrix to value prior to nnmf if usePool On workers
and on client pctRunOnAll warning (
             internal.stats.parallel.statParallelStore('rankDeficientMatrix').  state,
             'MATLAB:rankDeficientMatrix'  )
```

4.3.2.23 warning() [2/3]

```
else if useParallel warning (
             message( 'stats:nnmf:displayParallel')  )
```

4.3.2.24 warning() [3/3]

```
else On client warning (
             ws  )
```

4.3.2.25 ∼isscalar() [1/2]

```
if ∼isscalar (
             k  )
```

4.3.2.26 ~isscalar() [2/2]

```
if ~isscalar (
             tries  )
```

4.3.3 Variable Documentation

4.3.3.1 alg

```
Check optional arguments alg = internal.stats.getParamVal(alg,{'mult' 'als'},'ALGORITHM')
```

4.3.3.2 BIPLOT

```
See also BIPLOT = nnmf(x,5,'w0',w,'h0',h,'opt',opt,'alg','als')
```

4.3.3.3 case

```
Special case
```

4.3.3.4 cellout

```
cellout = whtry
```

4.3.3.5 computing

```
Display progress For parallel computing
```

4.3.3.6 d

```
end Get norm of difference and max change in factors d = a - w*h
```

4.3.3.7 defaultopt

```
end defaultopt = statset('nnmf')
```

4.3.3.8 delta

```
delta = max(dw,dh)
```

4.3.3.9 dflts

```
dflts = {'als' [] [] 1 [] }
```

4.3.3.10 dh

```
dh = max(max(abs(h-h0) / (sqrteps+max(max(abs(h0))))))
```

4.3.3.11 dispfmt

```
else dispfmt = '%8d\t%8d\t%8d\t%14g\t%14g\n'
```

4.3.3.12 dispnum

```
dispnum = dispnum - 1
```

4.3.3.13 dispopt

```
dispopt = statget(options,'Display',defaultopt,'fast')
```

4.3.3.14 dnorm

```
dnorm = sqrt(sum(sum(d.^2))/nm)
```

4.3.3.15 dw

```
dw = max(max(abs(w-w0) / (sqrteps+max(max(abs(w0))))))
```

4.3.3.16 factorization

```
end Perform a factorization[whtry{3}, whtry{4}, whtry{1}]
```

Initial value:

```
= ...
            nnmf1(a,whtry{3},whtry{4},ismult,maxiter,tolfun,tolx,...
                                dispnum,iter,usePool)
```

4.3.3.17 function

```
function[wbest, hbest, normbest]
```

Initial value:

```
= nnmf(a,k,varargin)
%NNMF Non-negative matrix factorization.
%   [W,H] = NNMF(A,K) factors the N-by-M matrix A into non-negative factors
%   W (N-by-K) and H (K-by-M).  The result is not an exact factorization
```

4.3.3.18 h

```
else Alternating least squares h = max(0,h0 .* (numer ./ ((w0'*w0)*h0 + eps(numer))))
```

4.3.3.19 h0

```
h0 = eye(k)
```

4.3.3.20 hbest

```
hbest = whbest{4}
```

4.3.3.21 hLines

```
hLines = gscatter(w(:,1),w(:,2),species)
```

4.3.3.22 ismult

```
ismult = strncmp('mult',alg,numel(alg))
```

4.3.3.23 j

```
end end for j
```

Initial value:

```
=1:maxiter
    if ismult
        % Multiplicative update formula
        numer = w0'*a
```

4.3.3.24 k

```
else for k
```

Initial value:

```
=round(k)
    error(message('stats:nnmf:BadK'))
```

4.3.3.25 labindx

```
Display progress For parallel the replicate number will be displayed under the worker performing
the replicate if dispnum final or iter if usePool labindx = internal.stats.parallel.worker↩
GetValue('workerID')
```

4.3.3.26 loopbody

```
end end Define the function that will perform one iteration of the loop inside smartFor loopbody
= @loopBody
```

4.3.3.27 maxiter

```
maxiter = statget(options,'MaxIter',defaultopt,'fast')
```

4.3.3.28 normbest

```
end if normbest = whbest{1}
```

4.3.3.29 numer

```
numer = a*h'
```

4.3.3.30 off

```
hold off
```

4.3.3.31 on

```
hold on
```

4.3.3.32 opt

```
opt = statset('maxiter',5,'display','final')
```

4.3.3.33 PCA

```
See also PCA
```

4.3.3.34 pnames

```
end Process optional arguments pnames = {'algorithm' 'w0' 'h0' 'replicates' 'options'}
```

4.3.3.35 sqrteps

```
sqrteps = sqrt(eps)
```

4.3.3.36 STATSET

```
See also STATSET
```

4.3.3.37 tolfun

```
tolfun = statget(options,'TolFun',defaultopt,'fast')
```

4.3.3.38 tolx

```
tolx = statget(options,'TolX',defaultopt,'fast')
```

4.3.3.39 usePool

```
usePool = useParallel && poolsz>0
```

4.3.3.40 w

```
w = sort(sum(wbest.^2,1),'descend')
```

4.3.3.41 wbest

```
end wbest = whbest{3}
```

4.3.3.42 whbest

```
whbest
```

Initial value:

```
= internal.stats.parallel.smartForReduce(...
        tries, loopbody, useParallel, RNGscheme, 'argmin')
```

4.3.3.43 whtry

```
else whtry = cell(4,1)
```

4.3.3.44 ws

```
else On client ws = warning('off','MATLAB:rankDeficientMatrix')
```

4.4 examples/nnmf_Jessica.m File Reference

Functions

- NNMF finds nonnegative matrix W and nonnegative coefficient matrix H such that V ∼WH The algorithm solves the problem of minimizing (V-WH)^2 by varying W and H. % % % Multiplicative update rules developed by Lee and Seung were used to solve % optimization problem.(see reference below) % D. D. Lee and H. S. Seung. Algorithms for non-negative matrix % factorization. Adv. Neural Info. Proc. Syst. 13
- 2001 INPUTS:OUTPUTS:GTO:Created:modification:Modifications:JLA:JLA: (3=lab generated Lee and Seung code method,[OUTPUTS] V-WH and,[GTO] id,[Created] id Last,[modification] id conditions,[Modifications] id,[JLA] id normalization,[JLA] id flagStd)
- if length (err_save)>550 err_change
- id synergies ()
- W (:, i)
- elseif ind_cond (l)
- elseif ind (l)
- zeros (1, r)]
- W (k:end,:)]

Variables

- function [W, H, err]
- flagMethod = 1
- elseif nargin< 4 flagMethod=1;endV=V.*(V >0);% Any potential negative entry in data matrix will be set to zerotest=nansum(V, 2);% Any potential muscle channnel with only 0 's is not included in the iteration index=find(test~=0);ind=find(test==0);Vnew_m=V(index,:);test_cond=nansum(V, 1);% Any potential condition with only 0 's is not included in the iteration index_cond=find(test_cond~=0);ind_cond=find(test↩_cond==0);Vnew=Vnew_m(:, index_cond);%%%%% Scale the input data to have unit variance %%%%%%%%%if flagStd==1;stdev=std(Vnew');%scale the data to have unit variance of this data set elseif flagStd==2;global stdev % use this if you want to use the stdev(unit variance scaling) from a different data setendVnew=diag(1./stdev) *Vnew;if flagMethod==1 opts=statset('MaxIter', 1000, 'TolFun', 1e-6, 'TolX', 1e-4);[W, H, err]=nnmf(Vnew, r, 'alg', 'mult', 'rep', 50, 'options', opts);%[W, H, err]=nnmf(Vnew, r, 'alg', 'mult', 'rep', 50);elseif flagMethod==2[W, H, err]=nnmf(Vnew, r, 'alg', 'als', 'rep', 50);elseif flagMethod==3 % Initial conditions.[n, m]=size(Vnew);H=rand(r, m);W=rand(n, r);err=sum(sum((Vnew-W *H).^2));MAX_↩IT=100000;% Increased this from 10000 to 100000 % Error goal - the "err" quantity, as defined, is the squared error. If we % want a 1% mse, then, we want .01 *prod(size(V))=.01 *n *m. ERR_GOAL=.0001 *(n *m);% Update... For normed data, the max err is n x m err_save=[ ];while err > ERR_GOAL H_fac =W'*Vnew
- H =H.*H_fac
- W_fac =Vnew*H'
- W =W.*W_fac
- err =sum(sum((Vnew-W*H).^2))
- err_save =[err_save
- add in zero rows
- calculate final error undo the unit variance scaling so synergies are back out of unit variance space and in the same scaling as the input data was Vnew = diag(stdev)*Vnew
- Synergy vectors normailzation m =max(W)
- vector with max activation values for i
- end Set to NaN the columns or rows that were not included in the iteration [n_o, m_o] =size(V)
- Hnew =[ ]
- Wnew =[ ]
- for l
- else for k

4.4.1 Function Documentation

4.4.1.1 ind()

```
elseif ind (
            1  )
```

4.4.1.2 ind_cond()

```
elseif ind_cond (
            1  )
```

4.4.1.3 INPUTS:OUTPUTS:GTO:Created:modification:Modifications:JLA:JLA:()

```
2001 INPUTS:OUTPUTS:GTO:Created:modification:Modifications:JLA:JLA: (
             3  = lab generated Lee and Seung code method,
             [OUTPUTS] V-WH and,
             [GTO] id ,
             [Created] id Last,
             [modification] id conditions,
             [Modifications] id ,
             [JLA] id normalization,
             [JLA] id flagStd )  [virtual]
```

4.4.1.4 length()

```
if length (
             err_save  )
```

4.4.1.5 minimizing()

```
NNMF finds nonnegative matrix W and nonnegative coefficient matrix H such that V ~WH The algorithm
solves the problem of minimizing (
             V- WH )
```

4.4.1.6 synergies()

```
id synergies ( )  [virtual]
```

4.4.1.7 W() [1/2]

```
W (
             :  ,
             i  )
```

4.4.1.8 W() [2/2]

```
W (
             k:end ,
             :    )
```

4.4.1.9 zeros()

```
zeros (
            1 ,
            r  )
```

4.4.2 Variable Documentation

4.4.2.1 err

```
err =sum(sum((Vnew-W*H).^2))
```

4.4.2.2 err_save

```
err_save =[err_save
```

4.4.2.3 flagMethod

```
flagMethod = 1
```

4.4.2.4 function

```
function[W, H, err]
```

Initial value:

```
= nnmf_Jessica(V, r, flagStd, flagMethod)
%
% [W, H, err] = nnmf_Jessica(V, r, flagStd, flagMethod)
%
% NNMF: Given a nonnegative matrix V
```

4.4.2.5 H

```
H =H.*H_fac
```

4.4.2.6 H_fac

```
H_fac =W'*Vnew
```

4.4.2.7 Hnew

```
break else Hnew =[]
```

4.4.2.8 i

```
vector with max activation values for i
```

Initial value:

```
=1:r
    H(i,:)=H(i,:)*m(i)
```

4.4.2.9 iteration

```
end Set to NaN the columns or rows that were not included in the iteration[n_o, m_o] =size(V)
```

4.4.2.10 k

```
else for k
```

Initial value:

```
=1:m_o
            if ind_cond(l)==k
                Hnew=[H(:,1:k-1) zeros(r,1) H(:,k:end)]
```

4.4.2.11 l

```
end end end end for l
```

Initial value:

```
=1:length(ind_cond)
    if ind_cond(l)==1
        Hnew=[zeros(r,1) H]
```

4.4.2.12 m

```
Synergy vectors normailzation m =max(W)
```

4.4.2.13 rows

```
add in zero rows
```

4.4.2.14 Vnew

```
calculate final error undo the unit variance scaling so synergies are back out of unit variance
space and in the same scaling as the input data was Vnew = diag(stdev)*Vnew
```

4.4.2.15 W

```
W =W.*W_fac
```

4.4.2.16 W_fac

```
W_fac =Vnew*H'
```

4.4.2.17 Wnew

```
break else Wnew =[]
```

4.5 include/io/csv_reader.h File Reference

```
#include <fstream>
#include <string>
#include <iostream>
#include <sstream>
```

Classes

- class csv_reader

4.6 csv_reader.h

Go to the documentation of this file.

```
1 #ifndef CSV_READER_H
2 #define CSV_READER_H
3
4 #pragma once
5 #include <fstream>
6 #include <string>
7 #include <iostream>
8 #include <sstream>
9
10 class csv_reader
11 {
12 public:
13     csv_reader();
14     ~csv_reader();
15     static int countLines(std::string filename);
16     static int countColumns(std::string filename, const char delimiter);
17 private:
18
19 };
20
21 #endif
```

4.7 include/math/matrix.h File Reference

```
#include <random>
```

Classes

- class matrix

4.8 matrix.h

Go to the documentation of this file.

```
1 #ifndef MATRIX_H
2 #define MATRIX_H
3
4 #pragma once
5 #include <random>
6 class matrix
7 {
8 public:
9     matrix();
10     ~matrix();
11     matrix(int nrows, int ncols);
12     matrix(int nrows, int ncols, bool randomize);
13     int get_num_rows();
14     int get_num_cols();
15     float at(int row, int col);
16     void set(int row, int col, float val);
17     static void mat_mult(matrix* A, matrix* B, matrix* out);
18
19
20 private:
21     int numcols_, numrows_;
22     float *data;
23
24 };
25
26 #endif
```

4.9 include/math/nnmf.h File Reference

```
#include <string>
#include "math/matrix.h"
```

Classes

- class nnmf

4.10 nnmf.h

Go to the documentation of this file.

```
1 #ifndef NNMF_H
2 #define NNMF_H
3
4 #pragma once
5
6 #include <string>
7 #include "math/matrix.h"
8
9 class nnmf
10 {
11 public:
12     nnmf();
13     ~nnmf();
14     bool Initialize();
15
16 private:
17
18 };
19
20 #endif
```

4.11 src/io/csv_reader.cpp File Reference

```
#include "io/csv_reader.h"
```

4.12 src/main.cpp File Reference

```
#include <iostream>
#include <string>
#include "math/matrix.h"
#include "io/csv_reader.h"
```

Functions

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

4.12.1 Function Documentation

4.12.1.1 main()

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

4.13 src/math/matrix.cpp File Reference

```
#include "math/matrix.h"
```

4.14 src/math/nnmf.cpp File Reference

```
#include "math/nnmf.h"
```

4.15 tests/io/csv_tester.cpp File Reference

```
#include "io/csv_reader.h"
#include "math/matrix.h"
#include <string>
#include <iostream>
```

Functions

- int main ()

4.15.1 Function Documentation

4.15.1.1 main()

```
int main ( )
```

Index