kaldi-matrix.h

Go to the documentation of this file.

```
// matrix/kaldi-matrix.h

// Copyright 2009-2011  Ondrej Glembek;  Microsoft Corporation;  Lukas Burget;
//                      Saarland University;  Petr Schwarz;  Yanmin Qian;
//                      Karel Vesely;  Go Vivace Inc.;  Haihua Xu

// See ../../COPYING for clarification regarding multiple authors
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//  http://www.apache.org/licenses/LICENSE-2.0
//
// THIS CODE IS PROVIDED *AS IS* BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED
// WARRANTIES OR CONDITIONS OF TITLE, FITNESS FOR A PARTICULAR PURPOSE,
// MERCHANTABLITY OR NON-INFRINGEMENT.
// See the Apache 2 License for the specific language governing permissions and
// limitations under the License.

#ifndef KALDI_MATRIX_KALDI_MATRIX_H_
#define KALDI_MATRIX_KALDI_MATRIX_H_ 1

#include "matrix/matrix-common.h"

namespace kaldi {


template<typename Real>
Real TraceMatMat(const MatrixBase<Real> &A, const MatrixBase<Real> &B,
                 MatrixTransposeType trans = kNoTrans);


template<typename Real>
class MatrixBase {
 public:
  // so this child can access protected members of other instances.
  friend class Matrix<Real>;
  // friend declarations for CUDA matrices (see ../cudamatrix/)
  friend class CuMatrixBase<Real>;
  friend class CuMatrix<Real>;
  friend class CuSubMatrix<Real>;
  friend class CuPackedMatrix<Real>;

  friend class PackedMatrix<Real>;

  inline MatrixIndexT  NumRows() const { return num_rows_; }

  inline MatrixIndexT NumCols() const { return num_cols_; }

  inline MatrixIndexT Stride() const {  return stride_; }

  size_t  SizeInBytes() const {
    return static_cast<size_t>(num_rows_) * static_cast<size_t>(stride_) *
        sizeof(Real);
  }

  inline const Real* Data() const {
    return data_;
  }

  inline Real* Data() { return data_; }

  inline  Real* RowData(MatrixIndexT i) {
    KALDI_ASSERT(static_cast<UnsignedMatrixIndexT>(i) <
                 static_cast<UnsignedMatrixIndexT>(num_rows_));
    return data_ + i * stride_;
```

```
  }

  inline const Real* RowData(MatrixIndexT i) const {
    KALDI_ASSERT(static_cast<UnsignedMatrixIndexT>(i) <
                 static_cast<UnsignedMatrixIndexT>(num_rows_));
    return data_ + i * stride_;
  }

  inline Real&  operator() (MatrixIndexT r, MatrixIndexT c) {
    KALDI_PARANOID_ASSERT(static_cast<UnsignedMatrixIndexT>(r) <
                          static_cast<UnsignedMatrixIndexT>(num_rows_) &&
                          static_cast<UnsignedMatrixIndexT>(c) <
                          static_cast<UnsignedMatrixIndexT>(num_cols_));
    return *(data_ + r * stride_ + c);
  }
  Real &Index (MatrixIndexT r, MatrixIndexT c) {  return (*this)(r, c); }

  inline const Real operator() (MatrixIndexT r, MatrixIndexT c) const {
    KALDI_PARANOID_ASSERT(static_cast<UnsignedMatrixIndexT>(r) <
                          static_cast<UnsignedMatrixIndexT>(num_rows_) &&
                          static_cast<UnsignedMatrixIndexT>(c) <
                          static_cast<UnsignedMatrixIndexT>(num_cols_));
    return *(data_ + r * stride_ + c);
  }

  /*   Basic setting-to-special values functions. */

  void SetZero();
  void Set(Real);
  void SetUnit();
  void SetRandn();
  void SetRandUniform();

  /*  Copying functions.  These do not resize the matrix! */


  template<typename OtherReal>
  void CopyFromMat(const MatrixBase<OtherReal> & M,
                   MatrixTransposeType trans = kNoTrans);

  void CopyFromMat(const CompressedMatrix &M);

  template<typename OtherReal>
  void CopyFromSp(const SpMatrix<OtherReal> &M);

  template<typename OtherReal>
  void CopyFromTp(const TpMatrix<OtherReal> &M,
                  MatrixTransposeType trans = kNoTrans);

  template<typename OtherReal>
  void CopyFromMat(const CuMatrixBase<OtherReal> &M,
                   MatrixTransposeType trans = kNoTrans);

  void CopyRowsFromVec(const VectorBase<Real> &v);

  void CopyRowsFromVec(const CuVectorBase<Real> &v);

  template<typename OtherReal>
  void CopyRowsFromVec(const VectorBase<OtherReal> &v);

  void CopyColsFromVec(const VectorBase<Real> &v);

  void CopyColFromVec(const VectorBase<Real> &v, const MatrixIndexT col);
  void CopyRowFromVec(const VectorBase<Real> &v, const MatrixIndexT row);
  void CopyDiagFromVec(const VectorBase<Real> &v);

  /* Accessing of sub-parts of the matrix. */

  inline const SubVector<Real> Row(MatrixIndexT i) const {
    KALDI_ASSERT(static_cast<UnsignedMatrixIndexT>(i) <
                 static_cast<UnsignedMatrixIndexT>(num_rows_));
    return SubVector<Real>(data_ + (i * stride_), NumCols());
  }

  inline SubVector<Real> Row(MatrixIndexT i) {
```

```
189       KALDI_ASSERT(static_cast<UnsignedMatrixIndexT>(i) <
190                    static_cast<UnsignedMatrixIndexT>(num_rows_));
191       return SubVector<Real>(data_ + (i * stride_), NumCols());
192     }
193
195     inline SubMatrix<Real> Range(const MatrixIndexT row_offset,
196                                  const MatrixIndexT num_rows,
197                                  const MatrixIndexT col_offset,
198                                  const MatrixIndexT num_cols) const {
199       return SubMatrix<Real>(*this, row_offset, num_rows,
200                              col_offset, num_cols);
201     }
202     inline SubMatrix<Real> RowRange(const MatrixIndexT row_offset,
203                                     const MatrixIndexT num_rows) const {
204       return SubMatrix<Real>(*this, row_offset, num_rows, 0, num_cols_);
205     }
206     inline SubMatrix<Real> ColRange(const MatrixIndexT col_offset,
207                                     const MatrixIndexT num_cols) const {
208       return SubMatrix<Real>(*this, 0, num_rows_, col_offset, num_cols);
209     }
210
211     /* Various special functions. */
213     Real Sum() const;
215     Real Trace(bool check_square = true) const;
216     // If check_square = true, will crash if matrix is not square.
217
219     Real Max() const;
221     Real Min() const;
222
224     void MulElements(const MatrixBase<Real> &A);
225
227     void DivElements(const MatrixBase<Real> &A);
228
230     void Scale(Real alpha);
231
233     void Max(const MatrixBase<Real> &A);
234
237     void MulColsVec(const VectorBase<Real> &scale);
238
241     void MulRowsVec(const VectorBase<Real> &scale);
242
246     void MulRowsGroupMat(const MatrixBase<Real> &src);
247
249     Real LogDet(Real *det_sign = NULL) const;
250
254     void Invert(Real *log_det = NULL, Real *det_sign = NULL,
255                 bool inverse_needed = true);
260     void InvertDouble(Real *LogDet = NULL, Real *det_sign = NULL,
261                       bool inverse_needed = true);
262
264     void InvertElements();
265
269     void Transpose();
270
276     void CopyCols(const MatrixBase<Real> &src,
277                   const std::vector<MatrixIndexT> &indices);
278
284     void CopyRows(const MatrixBase<Real> &src,
285                   const std::vector<MatrixIndexT> &indices);
286
288     void ApplyFloor(Real floor_val);
289
291     void ApplyCeiling(Real ceiling_val);
292
294     void ApplyLog();
295
297     void ApplyExp();
298
300     void ApplyPow(Real power);
301
306     void ApplyPowAbs(Real power, bool include_sign=false);
307
312     void ApplyHeaviside();
313
328     void Eig(MatrixBase<Real> *P,
```

```
329             VectorBase<Real> *eigs_real,
330             VectorBase<Real> *eigs_imag) const;
331
339   bool Power(Real pow);
340
352   void DestructiveSvd(VectorBase<Real> *s, MatrixBase<Real> *U,
353                       MatrixBase<Real> *Vt);  // Destroys calling matrix.
354
359   void Svd(VectorBase<Real> *s, MatrixBase<Real> *U,
360            MatrixBase<Real> *Vt) const;
362   void Svd(VectorBase<Real> *s) const { Svd(s, NULL, NULL); }
363
364
366   Real MinSingularValue() const {
367     Vector<Real> tmp(std::min(NumRows(), NumCols()));
368     Svd(&tmp);
369     return tmp.Min();
370   }
371
372   void TestUninitialized() const; // This function is designed so that if any element
373   // if the matrix is uninitialized memory, valgrind will complain.
374
377   Real Cond() const;
378
380   bool IsSymmetric(Real cutoff = 1.0e-05) const;  // replace magic number
381
383   bool IsDiagonal(Real cutoff = 1.0e-05) const;  // replace magic number
384
389   bool IsUnit(Real cutoff = 1.0e-05) const;     // replace magic number
390
392   bool IsZero(Real cutoff = 1.0e-05) const;     // replace magic number
393
396   Real FrobeniusNorm() const;
397
400   bool ApproxEqual(const MatrixBase<Real> &other, float tol = 0.01) const;
401
403   bool Equal(const MatrixBase<Real> &other) const;
404
406   Real LargestAbsElem() const;  // largest absolute value.
407
413   Real LogSumExp(Real prune = -1.0) const;
414
417   Real ApplySoftMax();
418
420   void Sigmoid(const MatrixBase<Real> &src);
421
423   void SoftHinge(const MatrixBase<Real> &src);
424
427   void GroupPnorm(const MatrixBase<Real> &src, Real power);
428
429
436   void GroupPnormDeriv(const MatrixBase<Real> &input, const MatrixBase<Real> &output,
437                        Real power);
438
439
441   void Tanh(const MatrixBase<Real> &src);
442
443   // Function used in backpropagating derivatives of the sigmoid function:
444   // element-by-element, set *this = diff * value * (1.0 - value).
445   void DiffSigmoid(const MatrixBase<Real> &value,
446                    const MatrixBase<Real> &diff);
447
448   // Function used in backpropagating derivatives of the tanh function:
449   // element-by-element, set *this = diff * (1.0 - value^2).
450   void DiffTanh(const MatrixBase<Real> &value,
451                 const MatrixBase<Real> &diff);
452
460   void SymPosSemiDefEig(VectorBase<Real> *s, MatrixBase<Real> *P,
461                         Real check_thresh = 0.001);
462
463   friend Real kaldi::TraceMatMat<Real>(const MatrixBase<Real> &A,
464       const MatrixBase<Real> &B, MatrixTransposeType trans);  // tr (A B)
465
```

```
466   // so it can get around const restrictions on the pointer to data_.
467   friend class SubMatrix<Real>;
468
470   void Add(const Real alpha);
471
473   void AddToDiag(const Real alpha);
474
476   template<typename OtherReal>
477   void AddVecVec(const Real alpha, const VectorBase<OtherReal> &a,
478                  const VectorBase<OtherReal> &b);
479
481   template<typename OtherReal>
482   void AddVecToRows(const Real alpha, const VectorBase<OtherReal> &v);
483
485   template<typename OtherReal>
486   void AddVecToCols(const Real alpha, const VectorBase<OtherReal> &v);
487
489   void AddMat(const Real alpha, const MatrixBase<Real> &M,
490               MatrixTransposeType transA = kNoTrans);
491
495   void SymAddMat2(const Real alpha, const MatrixBase<Real> &M,
496                   MatrixTransposeType transA, Real beta);
497
500   void AddDiagVecMat(const Real alpha, VectorBase<Real> &v,
501                      const MatrixBase<Real> &M, MatrixTransposeType transM,
502                      Real beta = 1.0);
503
506   void AddMatDiagVec(const Real alpha,
507                      const MatrixBase<Real> &M, MatrixTransposeType transM,
508                      VectorBase<Real> &v,
509                      Real beta = 1.0);
510
512   void AddMatMatElements(const Real alpha,
513                          const MatrixBase<Real>& A,
514                          const MatrixBase<Real>& B,
515                          const Real beta);
516
518   template<typename OtherReal>
519   void AddSp(const Real alpha, const SpMatrix<OtherReal> &S);
520
521   void AddMatMat(const Real alpha,
522                  const MatrixBase<Real>& A, MatrixTransposeType transA,
523                  const MatrixBase<Real>& B, MatrixTransposeType transB,
524                  const Real beta);
525
527   void AddMatMatDivMat(const MatrixBase<Real>& A,
528                        const MatrixBase<Real>& B,
529                       const MatrixBase<Real>& C);
530
533   void AddMatSmat(const Real alpha,
534                   const MatrixBase<Real>& A, MatrixTransposeType transA,
535                   const MatrixBase<Real>& B, MatrixTransposeType transB,
536                   const Real beta);
537
540   void AddSmatMat(const Real alpha,
541                   const MatrixBase<Real>& A, MatrixTransposeType transA,
542                   const MatrixBase<Real>& B, MatrixTransposeType transB,
543                   const Real beta);
544
546   void AddMatMatMat(const Real alpha,
547                     const MatrixBase<Real>& A, MatrixTransposeType transA,
548                     const MatrixBase<Real>& B, MatrixTransposeType transB,
549                     const MatrixBase<Real>& C, MatrixTransposeType transC,
550                     const Real beta);
551
553   // This and the routines below are really
554   // stubs that need to be made more efficient.
555   void AddSpMat(const Real alpha,
556                 const SpMatrix<Real>& A,
557                 const MatrixBase<Real>& B, MatrixTransposeType transB,
558                 const Real beta) {
559     Matrix<Real> M(A);
560     return AddMatMat(alpha, M, kNoTrans, B, transB, beta);
561   }
563   void AddTpMat(const Real alpha,
```

```
564                 const TpMatrix<Real>& A, MatrixTransposeType transA,
565                 const MatrixBase<Real>& B, MatrixTransposeType transB,
566                 const Real beta) {
567     Matrix<Real> M(A);
568     return AddMatMat(alpha, M, transA, B, transB, beta);
569   }
571   void AddMatSp(const Real alpha,
572                 const MatrixBase<Real>& A, MatrixTransposeType transA,
573                 const SpMatrix<Real>& B,
574                 const Real beta) {
575     Matrix<Real> M(B);
576     return AddMatMat(alpha, A, transA, M, kNoTrans, beta);
577   }
579   void AddSpMatSp(const Real alpha,
580                   const SpMatrix<Real> &A,
581                   const MatrixBase<Real>& B, MatrixTransposeType transB,
582                   const SpMatrix<Real>& C,
583                 const Real beta) {
584     Matrix<Real> M(A), N(C);
585     return AddMatMatMat(alpha, M, kNoTrans, B, transB, N, kNoTrans, beta);
586   }
588   void AddMatTp(const Real alpha,
589                 const MatrixBase<Real>& A, MatrixTransposeType transA,
590                 const TpMatrix<Real>& B, MatrixTransposeType transB,
591                 const Real beta) {
592     Matrix<Real> M(B);
593     return AddMatMat(alpha, A, transA, M, transB, beta);
594   }
595
597   void AddTpTp(const Real alpha,
598                const TpMatrix<Real>& A, MatrixTransposeType transA,
599                const TpMatrix<Real>& B, MatrixTransposeType transB,
600                const Real beta) {
601     Matrix<Real> M(A), N(B);
602     return AddMatMat(alpha, M, transA, N, transB, beta);
603   }
604
606   // This one is more efficient, not like the others above.
607   void AddSpSp(const Real alpha,
608                const SpMatrix<Real>& A, const SpMatrix<Real>& B,
609                const Real beta);
610
612   void CopyLowerToUpper();
613
615   void CopyUpperToLower();
616
621   void OrthogonalizeRows();
622
625   // Will throw exception on failure.
626   void Read(std::istream & in, bool binary, bool add = false);
628   void Write(std::ostream & out, bool binary) const;
629
630   // Below is internal methods for Svd, user does not have to know about this.
631 #if !defined(HAVE_ATLAS) && !defined(USE_KALDI_SVD)
632   // protected:
633   // Should be protected but used directly in testing routine.
634   // destroys *this!
635   void LapackGesvd(VectorBase<Real> *s, MatrixBase<Real> *U,
636                    MatrixBase<Real> *Vt);
637 #else
638  protected:
639   // destroys *this!
640   bool JamaSvd(VectorBase<Real> *s, MatrixBase<Real> *U,
641                MatrixBase<Real> *V);
642
643 #endif
644  protected:
645
647   explicit MatrixBase(Real *data, MatrixIndexT cols, MatrixIndexT rows,
      MatrixIndexT stride) :
648     data_(data), num_cols_(cols), num_rows_(rows), stride_(stride) {
649     KALDI_ASSERT_IS_FLOATING_TYPE(Real);
650   }
651
654   explicit MatrixBase(): data_(NULL) {
```

```
655       KALDI_ASSERT_IS_FLOATING_TYPE(Real);
656     }
657
658     // Make sure pointers to MatrixBase cannot be deleted.
659     ~MatrixBase() { }
660
666     inline Real*  Data_workaround() const {
667       return data_;
668     }
669
671     Real*   data_;
672
675     MatrixIndexT    num_cols_;
676     MatrixIndexT    num_rows_;
677
679     MatrixIndexT    stride_;
680    private:
681     KALDI_DISALLOW_COPY_AND_ASSIGN(MatrixBase);
682  };
683
685  template<typename Real>
686  class Matrix : public MatrixBase<Real> {
687   public:
688
690     Matrix();
691
694     Matrix(const MatrixIndexT r, const MatrixIndexT c,
695            MatrixResizeType resize_type = kSetZero):
696         MatrixBase<Real>() { Resize(r, c, resize_type); }
697
700     template<typename OtherReal>
701     explicit Matrix(const CuMatrixBase<OtherReal> &cu,
702                     MatrixTransposeType trans = kNoTrans);
703
704
706     void Swap(Matrix<Real> *other);
707
709     void Swap(CuMatrix<Real> *mat);
710
713     explicit Matrix(const MatrixBase<Real> & M,
714                     MatrixTransposeType trans = kNoTrans);
715
717     Matrix(const Matrix<Real> & M);  //  (cannot make explicit)
718
720     template<typename OtherReal>
721     explicit Matrix(const MatrixBase<OtherReal> & M,
722                       MatrixTransposeType trans = kNoTrans);
723
726     template<typename OtherReal>
727     explicit Matrix(const SpMatrix<OtherReal> & M) : MatrixBase<Real>() {
728       Resize(M.NumRows(), M.NumRows(), kUndefined);
729       this->CopyFromSp(M);
730     }
731
733     explicit Matrix(const CompressedMatrix &C);
734
736     template <typename OtherReal>
737     explicit Matrix(const TpMatrix<OtherReal> & M,
738                     MatrixTransposeType trans = kNoTrans) : MatrixBase<Real>() {
739       if (trans == kNoTrans) {
740         Resize(M.NumRows(), M.NumCols(), kUndefined);
741         this->CopyFromTp(M);
742       } else {
743         Resize(M.NumCols(), M.NumRows(), kUndefined);
744         this->CopyFromTp(M, kTrans);
745       }
746     }
747
749     // Unlike one in base, allows resizing.
750     void Read(std::istream & in, bool binary, bool add = false);
751
753     void RemoveRow(MatrixIndexT i);
754
757     void Transpose();
758
```

```
760   ~Matrix() { Destroy(); }
761
769   void Resize(const MatrixIndexT r,
770               const MatrixIndexT c,
771               MatrixResizeType resize_type = kSetZero);
772
774   Matrix<Real> &operator = (const MatrixBase<Real> &other) {
775     if (MatrixBase<Real>::NumRows() != other.NumRows() ||
776         MatrixBase<Real>::NumCols() != other.NumCols())
777       Resize(other.NumRows(), other.NumCols(), kUndefined);
778     MatrixBase<Real>::CopyFromMat(other);
779     return *this;
780   }
781
783   Matrix<Real> &operator = (const Matrix<Real> &other) {
784     if (MatrixBase<Real>::NumRows() != other.NumRows() ||
785         MatrixBase<Real>::NumCols() != other.NumCols())
786       Resize(other.NumRows(), other.NumCols(), kUndefined);
787     MatrixBase<Real>::CopyFromMat(other);
788     return *this;
789   }
790
791
792  private:
794   void Destroy();
795
800   void Init(const MatrixIndexT r,
801             const MatrixIndexT c);
802
803 };
805
808
811 struct HtkHeader {
813   int32    mNSamples;
815   int32    mSamplePeriod;
817   int16    mSampleSize;
819   uint16   mSampleKind;
820 };
821
822 // Read HTK formatted features from file into matrix.
823 template<typename Real>
824 bool ReadHtk(std::istream &is, Matrix<Real> *M, HtkHeader *header_ptr);
825
826 // Write (HTK format) features to file from matrix.
827 template<typename Real>
828 bool WriteHtk(std::ostream &os, const MatrixBase<Real> &M, HtkHeader htk_hdr);
829
830 // Write (CMUSphinx format) features to file from matrix.
831 template<typename Real>
832 bool WriteSphinx(std::ostream &os, const MatrixBase<Real> &M);
833
835
843 template<typename Real>
844 class SubMatrix : public MatrixBase<Real> {
845  public:
846   // Initialize a SubMatrix from part of a matrix; this is
847   // a bit like A(b:c, d:e) in Matlab.
848   // This initializer is against the proper semantics of "const", since
849   // SubMatrix can change its contents.  It would be hard to implement
850   // a "const-safe" version of this class.
851   SubMatrix(const MatrixBase<Real>& T,
852             const MatrixIndexT ro,  // row offset, 0 < ro < NumRows()
853             const MatrixIndexT r,   // number of rows, r > 0
854             const MatrixIndexT co,  // column offset, 0 < co < NumCols()
855             const MatrixIndexT c);   // number of columns, c > 0
856
857   // This initializer is mostly intended for use in CuMatrix and related
858   // classes.  Be careful!
859   SubMatrix(Real *data,
860             MatrixIndexT num_rows,
861             MatrixIndexT num_cols,
862             MatrixIndexT stride);
863
864   ~SubMatrix<Real>() {}
865
```

```
  SubMatrix<Real> (const SubMatrix &other):
  MatrixBase<Real> (other.data_, other.num_cols_, other.num_rows_,
                    other.stride_) {}

 private:
  SubMatrix<Real> &operator = (const SubMatrix<Real> &other);
};


// Some declarations.  These are traces of products.


template<typename Real>
bool ApproxEqual(const MatrixBase<Real> &A,
                 const MatrixBase<Real> &B, Real tol = 0.01) {
  return A.ApproxEqual(B, tol);
}

template<typename Real>
inline void AssertEqual(const MatrixBase<Real> &A, const MatrixBase<Real> &B,
                        float tol = 0.01) {
  KALDI_ASSERT(A.ApproxEqual(B, tol));
}

template <typename Real>
double TraceMat(const MatrixBase<Real> &A) { return A.Trace(); }


template <typename Real>
Real TraceMatMatMat(const MatrixBase<Real> &A, MatrixTransposeType transA,
                    const MatrixBase<Real> &B, MatrixTransposeType transB,
                    const MatrixBase<Real> &C, MatrixTransposeType transC);

template <typename Real>
Real TraceMatMatMatMat(const MatrixBase<Real> &A, MatrixTransposeType transA,
                       const MatrixBase<Real> &B, MatrixTransposeType transB,
                       const MatrixBase<Real> &C, MatrixTransposeType transC,
                       const MatrixBase<Real> &D, MatrixTransposeType transD);









template<typename Real> void SortSvd(VectorBase<Real> *s, MatrixBase<Real> *U,
                                     MatrixBase<Real>* Vt = NULL,
                                     bool sort_on_absolute_value = true);

template<typename Real>
void CreateEigenvalueMatrix(const VectorBase<Real> &real, const VectorBase<Real>
 &imag,
                            MatrixBase<Real> *D);

template<typename Real>
bool AttemptComplexPower(Real *x_re, Real *x_im, Real power);




template<typename Real>
std::ostream & operator << (std::ostream & Out, const MatrixBase<Real> & M);

template<typename Real>
std::istream & operator >> (std::istream & In, MatrixBase<Real> & M);

// The Matrix read allows resizing, so we override the MatrixBase one.
template<typename Real>
std::istream & operator >> (std::istream & In, Matrix<Real> & M);


template<typename Real>
bool SameDim(const MatrixBase<Real> &M, const MatrixBase<Real> &N) {
  return (M.NumRows() == N.NumRows() && M.NumCols() == N.NumCols());
}


```

```
973
974 }  // namespace kaldi
975
976
977
978 // we need to include the implementation and some
979 // template specializations.
980 #include "matrix/kaldi-matrix-inl.h"
981
982
983 #endif  // KALDI_MATRIX_KALDI_MATRIX_H_
```