FFLAS-FFPACK

Generated on Fri Nov 17 2023 00:00:00 for FFLAS-FFPACK by Doxygen 1.9.8

Fri Nov 17 2023 00:00:00

Chapter 1

FFLAS-FFPACK Documentation.

1.1 Introduction

FFLAS-FFPACK is a LGPL-2.1+ source code library for basic linear algebra operations over a finite field. It is inspired by BLAS interface (Basic Linear Algebra Subprograms) and the LAPACK library for numerical linear algebra, and shares part of their design. Yet it differs in many aspects due to the specifities of computing over a finite field:

- it is generic with respect to the finite field, so as to accomodate a large variety of field sizes and implementations;
- it is a pure source code library, to be included and compiled in the user's software. Its build system is only used for tests and benchmarks.

1.2 Goals

1.3 Design

1.4 Using FFLAS-FFPACK.

- Copying and Licence.
- Tutorial. This is a brief introduction to FFLAS-FFPACK capabilities.
- Configuring and Installing FFLAS-FFPACK. Explains how to configure/install from sources or from the latest svn version.
- Architecture of the library.. Describes how FFLAS-FFPACK is organized
- `Documentation for Users`. If everything around is blue, then you are reading the lighter, user-oriented, documentation.
- `Documentation for Developers`. If everything around is green, then you can get to everything (not necessarily yet) documented.

1.5 Contributing to fflas-ffpack, getting assistance.

Version

2.5.0

1.6 Copying and Licence

The FFLAS-FFPACK library is licensed under the terms of the GNU LGPL v2.1 or later.

See https://www.gnu.org/licenses/lgpl-2.1.html

1.7 Tutorial

no doc.

1.8 Configuring and Installing FFLAS-FFPACK

FFLAS-FFPACK is a header-only package.

Hower configuration process can be tweaked a lot. Configure looks for BLAS routines and Givaro library which are both mandatory dependencies. See the output of `./configure -help` for information about the LAPACK/↩ BLAS discovering strategies.

1.9 Architecture of the library.

no doc.

Chapter 2

Bug List

Global DOUBLE_TO_FLOAT_CROSSOVER

to be benchmarked.

Global FFPACK::buildMatrix (const Field &F, typename Field::ConstElement_ptr E, typename Field::Const↩ Element_ptr C, const size_t lda, const size_t *B, const size_t *T, const size_t me, const size_t mc, const size_t lambda, const size_t mu)

is this :

Global FFPACK::Invert2 (const Field &F, const size_t M, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr X, const size_t ldx, int &nullity)

not tested.

Chapter 3

Bibliography

Global FFPACK::LeadingSubmatrixRankProfiles (const size_t M, const size_t N, const size_t R, const size_t LSm, const size_t LSn, const size_t *P, const size_t *Q, size_t *RRP, size_t *CRP)

Dumas J-G., Pernet C., and Sultan Z. *Simultaneous computation of the row and column rank profiles* , ISSAC'13.

Global FFPACK::LUdivine (const Field &F, const FFLAS::FFLAS_DIAG Diag, const FFLAS::FFLAS_TRANSPOSE trans, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t *P, size_t *Qt, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive, const size_t cutoff=__FFLASFFPACK_LUDIVINE_THRESHOLD)

Jeannerod C-P, Pernet, C. and Storjohann, A. *Rank-profile revealing Gaussian elimination and the CUP matrix decomposition* , J. of Symbolic Comp., 2013

- Pernet C, Brassel M *LUdivine, une divine factorisation LU*, 2002

Global FFPACK::PLUQ (const Field &F, const FFLAS::FFLAS_DIAG Diag, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t *P, size_t *Q)

Dumas J-G., Pernet C., and Sultan Z. *Simultaneous computation of the row and column rank profiles* , ISSAC'13, 2013

Global FFPACK::productBruhatxTS (const Field &Fi, size_t N, size_t s, size_t r, size_t t, const size_t *P, const size_t *Q, typename Field::ConstElement_ptr Xu, size_t ldu, size_t NbBlocksU, const size_t *Ku, const size_t *Tu, const size_t *MU, typename Field::ConstElement_ptr Xl, size_t ldl, size_t NbBlocksL, const size_t *Kl, const size_t *Tl, const size_t *ML, typename Field::Element_ptr B, size_t ldb, const typename Field::Element beta, typename Field::Element_ptr D, size_t ldd)

Pernet C. and Storjohann A. *Time and space efficient generators for quasiseparable matrices* , JSC (85), 2018, doi:10.1016/j.jsc.2017.07.010

Global FFPACK::Protected::GaussJordan (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, const size_t colbeg, const size_t rowbeg, const size_t colsize, size_t *P, size_t *Q, const FFPACK::FFPACK_LU_TAG LuTag)

Algorithm 2.8 of A. Storjohann Thesis 2000,

- Algorithm 11 of Jeannerod C-P., Pernet, C. and Storjohann, A. *Rank-profile revealing Gaussian elimination and the CUP matrix decomposition* , J. of Symbolic Comp., 2013

Class ftrsmLeftUpperNoTransNonUnit< Element >

Dumas, Giorgi, Pernet 06, arXiv:cs/0601133.

Chapter 4

Todo List

File debug.h

we should put vector printing elsewhere.

Global FFPACK::getTriangular (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_↩ DIAG diag, const size_t M, const size_t N, const size_t R, typename Field::Element_ptr A, const size_t lda)

just one triangular fzero+fassign ?

Global FFPACK::getTriangular (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_↩ DIAG diag, const size_t M, const size_t N, const size_t R, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr T, const size_t ldt, const bool OnlyNonZeroVectors=false)

just one triangular fzero+fassign ?

Global FFPACK::Invert2 (const Field &F, const size_t M, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr X, const size_t ldx, int &nullity)

this init is not all necessary (done after ftrtri)

Global FFPACK::LUdivine (const Field &F, const FFLAS::FFLAS_DIAG Diag, const FFLAS::FFLAS_↩ TRANSPOSE trans, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t ∗P, size_t ∗Q, const FFPACK::FFPACK_LU_TAG LuTag, const size_t cutoff)

std::swap ?

Global FFPACK::Protected::RandomKrylovPrecond (const PolRing &PR, std::list< typename PolRing::↩ Element > &completedFactors, const size_t N, typename PolRing::Domain_t::Element_ptr A, const size_t lda, size_t &Nb, typename PolRing::Domain_t::Element_ptr &B, size_t &ldb, typename PolRing↩ ::Domain_t::RandIter &g, const size_t degree=__FFLASFFPACK_ARITHPROG_THRESHOLD)

don't assing K2 c∗noc x N but only mas (c,noc) x N and store each one after the other

swap to save space ??

Module field

biblio

Module MMalgos

biblio

Module simd

biblio

Chapter 5

Topic Index

5.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 6

Namespace Index

6.1 Namespace List

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

Chapter 7

Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 8

File Index

8.1 File List

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

Chapter 9

Topic Documentation

9.1 CHECKER

Class CHECKER provides functions to verify computations in FFLAS and FFPACK.

Class CHECKER provides functions to verify computations in FFLAS and FFPACK.

9.2 FFLAS-FFPACK

the FFLAS FFPACK library

Modules

- FFLAS
  The C-style wrapper of BLAS for finite field linear algebra.
- Interfaces
  Intefaces for FFLAS-FFPACK.

9.2.1 Detailed Description

the FFLAS FFPACK library

C++ header library for fast exact dense linear algebra

See also

FFLAS

FFPACK

9.2.2 FFLAS

The C-style wrapper of BLAS for finite field linear algebra.

The C-style wrapper of BLAS for finite field linear algebra.

FFLAS, Finite Field Linear Algebra Subroutines, provide basic linear algebra subroutines based on the BLAS interface. Therefore, the specifications are in C style; only the field given as a template parameter requires `C++`.

As much as possible, these routines use `ATLAS/BLAS` computations and achieve therefore high efficiency.

9.2.3 Interfaces

Intefaces for FFLAS-FFPACK.

Intefaces for FFLAS-FFPACK.

C interface in folder

See also

libs

9.3 Matrix Multiplication Algorithms

Matrix Multiplication (level 3) algorithms.

Files

- file schedule_bini.inl
  Bini implementation.

9.3.1 Detailed Description

Matrix Multiplication (level 3) algorithms.

Todo biblio

9.4 SIMD wrapper

wraps SIMD functions Supporst SSE4.1, AVX, AVX2.

wraps SIMD functions Supporst SSE4.1, AVX, AVX2.

Todo biblio

9.5 FFPACK

Class FFPACK provides functions using fflas much as Lapack uses BLAS.

Namespaces

- namespace FFPACK

  ***F**inite **F**ield **PACK** Set of elimination based routines for dense linear algebra.*

9.5.1 Detailed Description

Class FFPACK provides functions using fflas much as Lapack uses BLAS.

9.6 FFLAS-FFPACK fields

fields in the FFLAS-FFPACK library

Files

- file rns-double-elt.h

  rns elt structure with double support
- file rns-double.h

  rns structure with double support
- file rns-integer-mod.h

  representation of $\mathbb{Z}/p\mathbb{Z}$ using RNS representation (note: fixed precision)
- file rns-integer.h

  representation of $\mathbb{Z}$ using RNS representation (note: fixed precision)
- file rns.h

9.6.1 Detailed Description

fields in the FFLAS-FFPACK library

Unparametric/Random elements

Todo biblio

9.7 RNS

just include them all

just include them all

Chapter 10

Namespace Documentation

10.1 FFLAS::FieldCategories Namespace Reference

Traits and categories will need to be placed in a proper file later.

Data Structures

- struct GenericTag

  generic ring.
- struct ModularTag

  This is a modular field like e.g. `Modular<T>` *or* `ModularBalanced<T>`
- struct UnparametricTag

  If the field uses a representation with infix operators.

10.1.1 Detailed Description

Traits and categories will need to be placed in a proper file later.

10.2 FFLAS::ModeCategories Namespace Reference

Specifies the mode of action for an algorithm w.r.t.

Data Structures

- struct ConvertTo

  Force conversion to appropriate element type of ElementCategory T.
- struct DefaultBoundedTag

  Use standard field operations, but keeps track of bounds on input and output.
- struct DefaultTag

  No specific mode of action: use standard field operations.
- struct DelayedTag

  Performs field operations with delayed mod reductions. Ensures result is reduced.
- struct LazyTag

  Performs field operations with delayed mod only when necessary. Result may not be reduced.

10.2.1 Detailed Description

Specifies the mode of action for an algorithm w.r.t.

its field

10.3 FFLAS::ParSeqHelper Namespace Reference

ParSeqHelper for both fgemm and ftrsm.

10.3.1 Detailed Description

ParSeqHelper for both fgemm and ftrsm.

ParSeqHelper for both fgemm and ftrsm

10.4 FFLAS::StructureHelper Namespace Reference

StructureHelper for ftrsm.

10.4.1 Detailed Description

StructureHelper for ftrsm.

10.5 FFPACK Namespace Reference

Finite **F**ield **PACK** Set of elimination based routines for dense linear algebra.

Data Structures

- class Failure
  A precondtion failed.

Functions

- void **LAPACKPerm2MathPerm** (size_t *MathP, const size_t *LapackP, const size_t N)

  Conversion of a permutation from LAPACK format to Math format.
- void **MathPerm2LAPACKPerm** (size_t *LapackP, const size_t *MathP, const size_t N)

  Conversion of a permutation from Maths format to LAPACK format.
- template<class Field >
  void applyP (const Field &F, const FFLAS::FFLAS_SIDE Side, const FFLAS::FFLAS_TRANSPOSE Trans, const size_t M, const size_t ibeg, const size_t iend, typename Field::Element_ptr A, const size_t lda, const size_t *P)

  Computes P1 x Diag (I_R, P2) where P1 is a LAPACK and P2 a LAPACK permutation and store the result in P1 as a LAPACK permutation.
- template<class Field >
  void MonotonicApplyP (const Field &F, const FFLAS::FFLAS_SIDE Side, const FFLAS::FFLAS_↩ TRANSPOSE Trans, const size_t M, const size_t ibeg, const size_t iend, typename Field::Element_ptr A, const size_t lda, const size_t *P, const size_t R)

  Apply a R-monotonically increasing permutation P, to the matrix A.
- template<class Field >
  void fgetrs (const Field &F, const FFLAS::FFLAS_SIDE Side, const size_t M, const size_t N, const size_t R, typename Field::Element_ptr A, const size_t lda, const size_t *P, const size_t *Q, typename Field::Element↩ _ptr B, const size_t ldb, int *info)

  Solve the system $AX = B$ or $XA = B$.
- template<class Field >
  Field::Element_ptr fgetrs (const Field &F, const FFLAS::FFLAS_SIDE Side, const size_t M, const size↩ _t N, const size_t NRHS, const size_t R, typename Field::Element_ptr A, const size_t lda, const size_t *P, const size_t *Q, typename Field::Element_ptr X, const size_t ldx, typename Field::ConstElement_ptr B, const size_t ldb, int *info)

  Solve the system A X = B or X A = B.
- template<class Field >
  size_t fgesv (const Field &F, const FFLAS::FFLAS_SIDE Side, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr B, const size_t ldb, int *info)

  Square system solver.
- template<class Field >
  size_t fgesv (const Field &F, const FFLAS::FFLAS_SIDE Side, const size_t M, const size_t N, const size_t NRHS, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr X, const size_t ldx, typename Field::ConstElement_ptr B, const size_t ldb, int *info)

  Rectangular system solver.
- template<class Field >
  void ftrtri (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG Diag, const size_t N, typename Field::Element_ptr A, const size_t lda, const size_t threshold=__FFLASFFPACK_FTRTRI_↩ THRESHOLD)

  Compute the inverse of a triangular matrix.
- template<class Field >
  void ftrtrm (const Field &F, const FFLAS::FFLAS_SIDE side, const FFLAS::FFLAS_DIAG diag, const size_t N, typename Field::Element_ptr A, const size_t lda)

  Compute the product of two triangular matrices of opposite shape.
- template<class Field >
  void ftrstr (const Field &F, const FFLAS::FFLAS_SIDE side, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diagA, const FFLAS::FFLAS_DIAG diagB, const size_t N, typename Field::Const↩ Element_ptr A, const size_t lda, typename Field::Element_ptr B, const size_t ldb, const size_t threshold=64)

  Solve a triangular system with a triangular right hand side of the same shape.
- template<class Field >
  void ftrssyr2k (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diagA, const size_t N, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr B, const size↩ _t ldb, const size_t threshold=64)

Solve a triangular system in a symmetric sum: find B upper/lower triangular such that A^ T B + B^ T A = C where C is symmetric.

- template<class Field >
  bool fsytrf (const Field &F, const FFLAS::FFLAS_UPLO UpLo, const size_t N, typename Field::Element_ptr A, const size_t lda, const size_t threshold=__FFLASFFPACK_FSYTRF_THRESHOLD)

  Triangular factorization of symmetric matrices.

- template<class Field >
  bool fsytrf_nonunit (const Field &F, const FFLAS::FFLAS_UPLO UpLo, const size_t N, typename Field::↩ Element_ptr A, const size_t lda, typename Field::Element_ptr D, const size_t incD, const size_t threshold=↩ __FFLASFFPACK_FSYTRF_THRESHOLD)

  Triangular factorization of symmetric matrices.

- template<class Field >
  size_t PLUQ (const Field &F, const FFLAS::FFLAS_DIAG Diag, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t *P, size_t *Q)

  Compute a PLUQ factorization of the given matrix.

- template<class Field >
  size_t LUdivine (const Field &F, const FFLAS::FFLAS_DIAG Diag, const FFLAS::FFLAS_TRANSPOSE trans, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t *P, size_t *Qt, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive, const size_t cutoff=__FFLASFFPACK_LUDIVINE↩ _THRESHOLD)

  Compute the CUP or PLE factorization of the given matrix.

- template<class Field >
  size_t ColumnEchelonForm (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t *P, size_t *Qt, bool transform=false, const FFPACK_LU_TAG LuTag=FfpackSlab↩ Recursive)

  Compute the Column Echelon form of the input matrix in-place.

- template<class Field >
  size_t RowEchelonForm (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t *P, size_t *Qt, const bool transform=false, const FFPACK_LU_TAG LuTag=Ffpack↩ SlabRecursive)

  Compute the Row Echelon form of the input matrix in-place.

- template<class Field >
  size_t ReducedColumnEchelonForm (const Field &F, const size_t M, const size_t N, typename Field::↩ Element_ptr A, const size_t lda, size_t *P, size_t *Qt, const bool transform=false, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Compute the Reduced Column Echelon form of the input matrix in-place.

- template<class Field >
  size_t ReducedRowEchelonForm (const Field &F, const size_t M, const size_t N, typename Field::Element↩ _ptr A, const size_t lda, size_t *P, size_t *Qt, const bool transform=false, const FFPACK_LU_TAG Lu↩ Tag=FfpackSlabRecursive)

  Compute the Reduced Row Echelon form of the input matrix in-place.

- template<class Field >
  Field::Element_ptr Invert (const Field &F, const size_t M, typename Field::Element_ptr A, const size_t lda, int &nullity)

  Invert the given matrix in place or computes its nullity if it is singular.

- template<class Field >
  Field::Element_ptr Invert (const Field &F, const size_t M, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr X, const size_t ldx, int &nullity)

  Invert the given matrix or computes its nullity if it is singular.

- template<class Field >
  Field::Element_ptr Invert2 (const Field &F, const size_t M, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr X, const size_t ldx, int &nullity)

  Invert the given matrix or computes its nullity if it is singular.

- template<class PolRing >
  std::list< typename PolRing::Element > & CharPoly (const PolRing &R, std::list< typename PolRing::↩ Element > &charp, const size_t N, typename PolRing::Domain_t::Element_ptr A, const size_t lda, typename PolRing::Domain_t::RandIter &G, const FFPACK_CHARPOLY_TAG CharpTag=FfpackAuto, const size_↩ t degree=__FFLASFFPACK_ARITHPROG_THRESHOLD)

  Compute the characteristic polynomial of the matrix A.
- template<class PolRing >
  PolRing::Element & CharPoly (const PolRing &R, typename PolRing::Element &charp, const size_t N, typename PolRing::Domain_t::Element_ptr A, const size_t lda, typename PolRing::Domain_t::RandIter &G, const FFPACK_CHARPOLY_TAG CharpTag=FfpackAuto, const size_t degree=__FFLASFFPACK_ARITHPROG↩ _THRESHOLD)

  Compute the characteristic polynomial of the matrix A.
- template<class PolRing >
  PolRing::Element & CharPoly (const PolRing &R, typename PolRing::Element &charp, const size_t N, typename PolRing::Domain_t::Element_ptr A, const size_t lda, const FFPACK_CHARPOLY_TAG Charp↩ Tag=FfpackAuto, const size_t degree=__FFLASFFPACK_ARITHPROG_THRESHOLD)

  Compute the characteristic polynomial of the matrix A.
- template<class Field , class Polynomial >
  Polynomial & MinPoly (const Field &F, Polynomial &minP, const size_t N, typename Field::ConstElement_ptr A, const size_t lda)

  Compute the minimal polynomial of the matrix A.
- template<class Field , class Polynomial , class RandIter >
  Polynomial & MinPoly (const Field &F, Polynomial &minP, const size_t N, typename Field::ConstElement_ptr A, const size_t lda, RandIter &G)

  Compute the minimal polynomial of the matrix A.
- template<class Field , class Polynomial >
  Polynomial & MatVecMinPoly (const Field &F, Polynomial &minP, const size_t N, typename Field::Const↩ Element_ptr A, const size_t lda, typename Field::ConstElement_ptr v, const size_t incv)

  Compute the minimal polynomial of the matrix A and a vector v, namely the first linear dependency relation in the Krylov basis $(v, Av, ..., A^N v)$.
- template<class Field >
  size_t Rank (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda)

  Computes the rank of the given matrix using a PLUQ factorization.
- template<class Field >
  bool IsSingular (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda)

  Returns true if the given matrix is singular.
- template<class Field >
  Field::Element & Det (const Field &F, typename Field::Element &det, const size_t N, typename Field::↩ Element_ptr A, const size_t lda, size_t *P=NULL, size_t *Q=NULL)

  Returns the determinant of the given square matrix.
- template<class Field >
  Field::Element_ptr Solve (const Field &F, const size_t M, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr x, const int incx, typename Field::ConstElement_ptr b, const int incb)

  Solves a linear system AX = b using PLUQ factorization.
- template<class Field >
  *void RandomNullSpaceVector (const Field &F, const FFLAS::FFLAS_SIDE Side, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr X, const size_t incX)

  Solve L X = B or X L = B in place.
- template<class Field >
  size_t NullSpaceBasis (const Field &F, const FFLAS::FFLAS_SIDE Side, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr &NS, size_t &ldn, size_↩ t &NSdim)

  Computes a basis of the Left/Right nullspace of the matrix A.

- template<class Field >
  size_t RowRankProfile (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t *&rkprofile, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Computes the row rank profile of A.
- template<class Field >
  size_t ColumnRankProfile (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t *&rkprofile, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Computes the column rank profile of A.
- void RankProfileFromLU (const size_t *P, const size_t N, const size_t R, size_t *rkprofile, const FFPACK_↩ LU_TAG LuTag)

  Recovers the column/row rank profile from the permutation of an LU decomposition.
- size_t LeadingSubmatrixRankProfiles (const size_t M, const size_t N, const size_t R, const size_t LSm, const size_t LSn, const size_t *P, const size_t *Q, size_t *RRP, size_t *CRP)

  Recovers the row and column rank profiles of any leading submatrix from the PLUQ decomposition.
- template<class Field >
  size_t RowRankProfileSubmatrixIndices (const Field &F, const size_t M, const size_t N, typename Field::↩ Element_ptr A, const size_t lda, size_t *&rowindices, size_t *&colindices, size_t &R)

  RowRankProfileSubmatrixIndices.
- template<class Field >
  size_t ColRankProfileSubmatrixIndices (const Field &F, const size_t M, const size_t N, typename Field::↩ Element_ptr A, const size_t lda, size_t *&rowindices, size_t *&colindices, size_t &R)

  *Computes the indices of the submatrix r*r X of A whose columns correspond to the column rank profile of A.*
- template<class Field >
  size_t RowRankProfileSubmatrix (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr &X, size_t &R)

  *Computes the r*r submatrix X of A, by picking the row rank profile rows of A.*
- template<class Field >
  size_t ColRankProfileSubmatrix (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr &X, size_t &R)

  Compute the $r \times r$ submatrix X of A, by picking the row rank profile rows of A.
- template<class Field >
  void getTriangular (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diag, const size_t M, const size_t N, const size_t R, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr T, const size_t ldt, const bool OnlyNonZeroVectors=false)

  Extracts a triangular matrix from a compact storage A=L\U of rank R.
- template<class Field >
  void getTriangular (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diag, const size_t M, const size_t N, const size_t R, typename Field::Element_ptr A, const size_t lda)

  Cleans up a compact storage A=L\U to reveal a triangular matrix of rank R.
- template<class Field >
  void getEchelonForm (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diag, const size_t M, const size_t N, const size_t R, const size_t *P, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr T, const size_t ldt, const bool OnlyNonZeroVectors=false, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Extracts a matrix in echelon form from a compact storage A=L\U of rank R obtained by RowEchelonForm or Column↩ EchelonForm.
- template<class Field >
  void getEchelonForm (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diag, const size_t M, const size_t N, const size_t R, const size_t *P, typename Field::Element_ptr A, const size_t lda, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Cleans up a compact storage A=L\U obtained by RowEchelonForm or ColumnEchelonForm to reveal an echelon form of rank R.

- template<class Field >
  void getEchelonTransform (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diag, const size_t M, const size_t N, const size_t R, const size_t *P, const size_t *Q, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr T, const size_t ldt, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Extracts a transformation matrix to echelon form from a compact storage A=L\U of rank R obtained by RowEchelonForm or ColumnEchelonForm.
- template<class Field >
  void getReducedEchelonForm (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const size_t M, const size_t N, const size_t R, const size_t *P, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr T, const size_t ldt, const bool OnlyNonZeroVectors=false, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Extracts a matrix in echelon form from a compact storage A=L\U of rank R obtained by ReducedRowEchelonForm or ReducedColumnEchelonForm with transform = true.
- template<class Field >
  void getReducedEchelonForm (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const size_t M, const size_t N, const size_t R, const size_t *P, typename Field::Element_ptr A, const size_t lda, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Cleans up a compact storage A=L\U of rank R obtained by ReducedRowEchelonForm or ReducedColumnEchelonForm with transform = true.
- template<class Field >
  void getReducedEchelonTransform (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const size_t M, const size_t N, const size_t R, const size_t *P, const size_t *Q, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr T, const size_t ldt, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Extracts a transformation matrix to echelon form from a compact storage A=L\U of rank R obtained by RowEchelonForm or ColumnEchelonForm.
- void **PLUQtoEchelonPermutation** (const size_t N, const size_t R, const size_t *P, size_t *outPerm)

  Auxiliary routine: determines the permutation that changes a PLUQ decomposition into a echelon form revealing PLUQ decomposition.
- template<class Field >
  size_t LTBruhatGen (const Field &Fi, const FFLAS::FFLAS_DIAG diag, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t *P, size_t *Q)

  LTBruhatGen Suppose A is Left Triangular Matrix This procedure computes the Bruhat Representation of A and return the rank of A.
- template<class Field >
  void getLTBruhatGen (const Field &Fi, const size_t N, const size_t r, const size_t *P, const size_t *Q, typename Field::Element_ptr R, const size_t ldr)

  GetLTBruhatGen This procedure Computes the Rank Revealing Matrix based on the Bruhta representation of a Matrix.
- template<class Field >
  void getLTBruhatGen (const Field &Fi, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diag, const size_t N, const size_t r, const size_t *P, const size_t *Q, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr T, const size_t ldt)

  GetLTBruhatGen This procedure computes the matrix L or U f the Bruhat Representation Suppose that A is the bruhat representation of a matrix.
- size_t LTQSorder (const size_t N, const size_t r, const size_t *P, const size_t *Q)

  LTQSorder This procedure computes the order of quasiseparability of a matrix.
- template<class Field >
  size_t CompressToBlockBiDiagonal (const Field &Fi, const FFLAS::FFLAS_UPLO Uplo, size_t N, size_t s, size_t r, const size_t *P, const size_t *Q, typename Field::Element_ptr A, size_t lda, typename Field::Element_ptr X, size_t ldx, size_t *K, size_t *M, size_t *T)

  CompressToBlockBiDiagonal This procedure compress a compact representation of a row echelon form or column echelon form.
- template<class Field >
  void ExpandBlockBiDiagonalToBruhat (const Field &Fi, const FFLAS::FFLAS_UPLO Uplo, size_t N, size_t s, size_t r, typename Field::Element_ptr A, size_t lda, typename Field::Element_ptr X, size_t ldx, size_t NbBlocks, size_t *K, size_t *M, size_t *T)

ExpandBlockBiDiagonal This procedure expand a compact representation of a row echelon form or column echelon form.

- void Bruhat2EchelonPermutation (size_t N, size_t R, const size_t *P, const size_t *Q, size_t *M)

  Bruhat2EchelonPermutation (N,R,P,Q) Compute M such that LM or MU is in echelon form where L or U are factors of the Bruhat Rpresentation.

- template<class Field >
  void **productBruhatxTS** (const Field &Fi, size_t N, size_t s, size_t r, const size_t *P, const size_t *Q, const typename Field::Element_ptr Xu, size_t ldu, size_t NbBlocksU, size_t *Ku, size_t *Tu, size_t *MU, const typename Field::Element_ptr Xl, size_t ldl, size_t NbBlocksL, size_t *Kl, size_t *Tl, size_t *ML, typename Field::Element_ptr B, size_t t, size_t ldb, typename Field::Element_ptr C, size_t ldc)

  productBruhatxTS Comput the product between the CRE compact representation of a matrix A and B a tall matrix

- template<class Field >
  Field::Element_ptr LQUPtoInverseOfFullRankMinor (const Field &F, const size_t rank, typename Field::↩
  Element_ptr A_factors, const size_t lda, const size_t *QtPointer, typename Field::Element_ptr X, const size↩
  _t ldx)

  LQUPtoInverseOfFullRankMinor.

- template<class Field >
  void RandomNullSpaceVector (const Field &F, const FFLAS::FFLAS_SIDE Side, const size_t M, const size↩
  _t N, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr X, const size_t incX)

  Solve L X = B or X L = B in place.

- template<class Field >
  Field::Element_ptr **expandLCRE** (const Field &Fi, size_t N, size_t s, size_t r, size_t *R, size_t i, typename Field::ConstElement_ptr Xu, size_t ldu, size_t NbBlocksU, const size_t *Ku, const size_t *Tuinv, typename Field::ConstElement_ptr Xl, size_t ldl, size_t NbBlocksL, const size_t *Kl, const size_t *Tlinv, typename Field::Element_ptr CRE, size_t ldcre)

  Expands an anti-diagonal block of a left triangular matrix from its compact Bruhat representation.

- template<class Field >
  void productBruhatxTS (const Field &Fi, size_t N, size_t s, size_t r, size_t t, const size_t *P, const size_t *Q, typename Field::ConstElement_ptr Xu, size_t ldu, size_t NbBlocksU, const size_t *Ku, const size_t *Tu, const size_t *MU, typename Field::ConstElement_ptr Xl, size_t ldl, size_t NbBlocksL, const size_t *Kl, const size_t *Tl, const size_t *ML, typename Field::Element_ptr B, size_t ldb, const typename Field::Element beta, typename Field::Element_ptr D, size_t ldd)

  Compute the product of a left-triangular quasi-separable matrix A, represented by a compact Bruhat generator, with a dense rectangular matrix B: $C \leftarrow A \times B + betaC$.

- template<class Field >
  Field::Element_ptr buildMatrix (const Field &F, typename Field::ConstElement_ptr E, typename Field::↩
  ConstElement_ptr C, const size_t lda, const size_t *B, const size_t *T, const size_t me, const size_t mc, const size_t lambda, const size_t mu)

- template<class Field >
  size_t fsytrf_UP_RPM (const Field &Fi, const size_t N, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr Dinv, const size_t incDinv, size_t *P, size_t BCThreshold)

- template<class Field >
  size_t LUdivine (const Field &F, const FFLAS::FFLAS_DIAG Diag, const FFLAS::FFLAS_TRANSPOSE trans, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t *P, size_t *Q, const FFPACK::FFPACK_LU_TAG LuTag, const size_t cutoff)

- void composePermutationsLLL (size_t *P1, const size_t *P2, const size_t R, const size_t N)

  Computes P1 x Diag (I_R, P2) where P1 is a LAPACK and P2 a LAPACK permutation and store the result in P1 as a LAPACK permutation.

- void composePermutationsLLM (size_t *MathP, const size_t *P1, const size_t *P2, const size_t R, const size_t N)

  Computes P1 x Diag (I_R, P2) where P1 is a LAPACK and P2 a LAPACK permutation and store the result in MathP as a MathPermutation format.

- void composePermutationsMLM (size_t *MathP1, const size_t *P2, const size_t R, const size_t N)

  Computes MathP1 x Diag (I_R, P2) where MathP1 is a MathPermutation and P2 a LAPACK permutation and store the result in MathP1 as a MathPermutation format.

- template<class Field , class RandIter >
  Field::Element_ptr NonZeroRandomMatrix (const Field &F, size_t m, size_t n, typename Field::Element_ptr A, size_t lda, RandIter &G)

  Random non-zero Matrix.
- template<class Field , class RandIter >
  Field::Element_ptr NonZeroRandomMatrix (const Field &F, size_t m, size_t n, typename Field::Element_ptr A, size_t lda)

  Random non-zero Matrix.
- template<class Field , class RandIter >
  Field::Element_ptr RandomMatrix (const Field &F, size_t m, size_t n, typename Field::Element_ptr A, size_t lda, RandIter &G)

  Random Matrix.
- template<class Field >
  Field::Element_ptr RandomMatrix (const Field &F, size_t m, size_t n, typename Field::Element_ptr A, size_t lda)

  Random Matrix.
- template<class Field , class RandIter >
  Field::Element_ptr RandomTriangularMatrix (const Field &F, size_t m, size_t n, const FFLAS::FFLAS_↩ UPLO UpLo, const FFLAS::FFLAS_DIAG Diag, bool nonsingular, typename Field::Element_ptr A, size_t lda, RandIter &G)

  Random Triangular Matrix.
- template<class Field >
  Field::Element_ptr RandomTriangularMatrix (const Field &F, size_t m, size_t n, const FFLAS::FFLAS_UPLO UpLo, const FFLAS::FFLAS_DIAG Diag, bool nonsingular, typename Field::Element_ptr A, size_t lda)

  Random Triangular Matrix.
- template<class Field , class RandIter >
  Field::Element_ptr RandomSymmetricMatrix (const Field &F, size_t n, bool nonsingular, typename Field::↩ Element_ptr A, size_t lda, RandIter &G)

  Random Symmetric Matrix.
- template<class Field , class RandIter >
  Field::Element_ptr RandomMatrixWithRank (const Field &F, size_t m, size_t n, size_t r, typename Field::↩ Element_ptr A, size_t lda, RandIter &G)

  Random Matrix with prescribed rank.
- template<class Field >
  Field::Element_ptr RandomMatrixWithRank (const Field &F, size_t m, size_t n, size_t r, typename Field::↩ Element_ptr A, size_t lda)

  Random Matrix with prescribed rank.
- size_t * RandomIndexSubset (size_t N, size_t R, size_t *P)

  Pick uniformly at random a sequence of R distinct elements from the set $\{0, \ldots, N-1\}$ using Knuth's shuffle.
- size_t * RandomPermutation (size_t N, size_t *P)

  Pick uniformly at random a permutation of size N stored in LAPACK format using Knuth's shuffle.
- void RandomRankProfileMatrix (size_t M, size_t N, size_t R, size_t *rows, size_t *cols)

  Pick uniformly at random an R-subpermutation of dimension M x N : a matrix with only R non-zeros equal to one, in a random rook placement.
- void RandomSymmetricRankProfileMatrix (size_t N, size_t R, size_t *rows, size_t *cols)

  Pick uniformly at random a symmetric R-subpermutation of dimension N x N : a symmetric matrix with only R non-zeros, all equal to one, in a random rook placement.
- template<class Field , class RandIter >
  Field::Element_ptr RandomMatrixWithRankandRPM (const Field &F, size_t M, size_t N, size_t R, typename Field::Element_ptr A, size_t lda, const size_t *RRP, const size_t *CRP, RandIter &G)

  Random Matrix with prescribed rank and rank profile matrix Creates an m x n matrix with random entries and rank r.
- template<class Field >
  Field::Element_ptr RandomMatrixWithRankandRPM (const Field &F, size_t M, size_t N, size_t R, typename Field::Element_ptr A, size_t lda, const size_t *RRP, const size_t *CRP)

Random Matrix with prescribed rank and rank profile matrix Creates an m x n matrix with random entries and rank r.

- template<class Field , class RandIter >
  Field::Element_ptr RandomSymmetricMatrixWithRankandRPM (const Field &F, size_t N, size_t R, typename Field::Element_ptr A, size_t lda, const size_t *RRP, const size_t *CRP, RandIter &G)

  Random Symmetric Matrix with prescribed rank and rank profile matrix Creates an n x n symmetric matrix with random entries and rank r.
- template<class Field >
  Field::Element_ptr RandomSymmetricMatrixWithRankandRPM (const Field &F, size_t M, size_t N, size_t R, typename Field::Element_ptr A, size_t lda, const size_t *RRP, const size_t *CRP)

  Random Symmetric Matrix with prescribed rank and rank profile matrix Creates an n x n symmetric matrix with random entries and rank r.
- template<class Field , class RandIter >
  Field::Element_ptr RandomMatrixWithRankandRandomRPM (const Field &F, size_t M, size_t N, size_t R, typename Field::Element_ptr A, size_t lda, RandIter &G)

  Random Matrix with prescribed rank, with random rank profile matrix Creates an m x n matrix with random entries, rank r and with a rank profile matrix chosen uniformly at random.
- template<class Field >
  Field::Element_ptr RandomMatrixWithRankandRandomRPM (const Field &F, size_t M, size_t N, size_t R, typename Field::Element_ptr A, size_t lda)

  Random Matrix with prescribed rank, with random rank profile matrix Creates an m x n matrix with random entries, rank r and with a rank profile matrix chosen uniformly at random.
- template<class Field , class RandIter >
  Field::Element_ptr RandomSymmetricMatrixWithRankandRandomRPM (const Field &F, size_t N, size_t R, typename Field::Element_ptr A, size_t lda, RandIter &G)

  Random Symmetric Matrix with prescribed rank, with random rank profile matrix Creates an n x n matrix with random entries, rank r and with a rank profile matrix chosen uniformly at random.
- template<class Field >
  Field::Element_ptr RandomSymmetricMatrixWithRankandRandomRPM (const Field &F, size_t N, size_t R, typename Field::Element_ptr A, size_t lda)

  Random Symmetric Matrix with prescribed rank, with random rank profile matrix Creates an n x n matrix with random entries, rank r and with a rank profile matrix chosen uniformly at random.
- template<class Field >
  Field::Element_ptr RandomMatrixWithDet (const Field &F, size_t n, const typename Field::Element d, typename Field::Element_ptr A, size_t lda)

  Random Matrix with prescribed det.
- template<class Field , class RandIter >
  Field::Element_ptr RandomMatrixWithDet (const Field &F, size_t n, const typename Field::Element d, typename Field::Element_ptr A, size_t lda, RandIter &G)

  Random Matrix with prescribed det.

10.5.1 Detailed Description

Finite **F**ield **PACK** Set of elimination based routines for dense linear algebra.

This namespace enlarges the set of BLAS routines of the class FFLAS, with higher level routines based on elimination.

10.5.2 Function Documentation

10.5.2.1 applyP()

```
template<class Field >
void applyP (
```

```
            const Field & F,
            const FFLAS::FFLAS_SIDE Side,
            const FFLAS::FFLAS_TRANSPOSE Trans,
            const size_t M,
            const size_t ibeg,
            const size_t iend,
            typename Field::Element_ptr A,
            const size_t lda,
            const size_t * P )  [inline]
```

Computes P1 x Diag (I_R, P2) where P1 is a LAPACK and P2 a LAPACK permutation and store the result in P1 as a LAPACK permutation.

Parameters

in,out	*P1*	a LAPACK permutation of size N
	P2	a LAPACK permutation of size N-R

Applies a permutation P to the matrix A. Apply a permutation P, stored in the LAPACK format (a sequence of transpositions) between indices ibeg and iend of P to (iend-ibeg) vectors of size M stored in A (as column for NoTrans and rows for Trans). Side==FFLAS::FflasLeft for row permutation Side==FFLAS::FflasRight for a column permutation Trans==FFLAS::FflasTrans for the inverse permutation of P

Parameters

F	base field
Side	decides if rows (FflasLeft) or columns (FflasRight) are permuted
Trans	decides if the matrix is seen as columns (FflasTrans) or rows (FflasNoTrans)
M	size of the elements to permute
ibeg	first index to consider in P
iend	last index to consider in P
A	input matrix
lda	leading dimension of A
P	permutation in LAPACK format
psh	(optional): a sequential or parallel helper, to choose between sequential or parallel execution

Warning

not sure the submatrix is still a permutation and the one we expect in all cases... examples for iend=2, ibeg=1 and P=[2,2,2]

10.5.2.2 MonotonicApplyP()

```
template<class Field >
void MonotonicApplyP (
            const Field & F,
            const FFLAS::FFLAS_SIDE Side,
            const FFLAS::FFLAS_TRANSPOSE Trans,
            const size_t M,
            const size_t ibeg,
            const size_t iend,
```

```
            typename Field::Element_ptr A,
            const size_t lda,
            const size_t * P,
            const size_t R )  [inline]
```

Apply a R-monotonically increasing permutation P, to the matrix A.

MonotonicApplyP Apply a permutation defined by the first R entries of the vector P (the pivots).

The permutation represented by P is defined as follows:

- the first R values of P is a LAPACK reprensentation (a sequence of transpositions)
- the remaining iend-ibeg-R values of the permutation are in a monotonically increasing progression Side==FFLAS::FflasLeft for row permutation Side==FFLAS::FflasRight for a column permutation Trans==FFLAS::FflasTrans for the inverse permutation of P

Parameters

F	base field
Side	selects if it is a row (FflasLeft) or column (FflasRight) permutation
Trans	inverse permutation (FflasTrans/NoTrans)
M	
ibeg	
iend	
A	input matrix
lda	leading dimension of A
P	LAPACK permuation
R	first values of P

The non pivot elements, are located in montonically increasing order.

10.5.2.3 fgetrs() [1/2]

```
template<class Field >
void fgetrs (
            const Field & F,
            const FFLAS::FFLAS_SIDE Side,
            const size_t M,
            const size_t N,
            const size_t R,
            typename Field::Element_ptr A,
            const size_t lda,
            const size_t * P,
            const size_t * Q,
            typename Field::Element_ptr B,
            const size_t ldb,
            int * info )
```

Solve the system $AX = B$ or $XA = B$.

Solving using the PLUQ decomposition of A already computed inplace with PLUQ (FFLAS::FflasNonUnit). Version for A square. If A is rank deficient, a solution is returned if the system is consistent, Otherwise an info is 1

Parameters

F	base field
Side	Determine wheter the resolution is left (FflasLeft) or right (FflasRight) looking.
M	row dimension of `B`
N	col dimension of `B`
R	rank of `A`
A	input matrix
lda	leading dimension of `A`
P	row permutation of the `PLUQ` decomposition of `A`
Q	column permutation of the `PLUQ` decomposition of `A`
B	Right/Left hand side matrix. Initially stores `B`, finally stores the solution `X`.
ldb	leading dimension of `B`
info	Success of the computation: 0 if successfull, >0 if system is inconsistent

10.5.2.4 fgetrs() [2/2]

```
template<class Field >
Field::Element_ptr fgetrs (
             const Field & F,
             const FFLAS::FFLAS_SIDE Side,
             const size_t M,
             const size_t N,
             const size_t NRHS,
             const size_t R,
             typename Field::Element_ptr A,
             const size_t lda,
             const size_t * P,
             const size_t * Q,
             typename Field::Element_ptr X,
             const size_t ldx,
             typename Field::ConstElement_ptr B,
             const size_t ldb,
             int * info )
```

Solve the system A X = B or X A = B.

Solving using the PLUQ decomposition of A already computed inplace with PLUQ(FFLAS::FflasNonUnit). Version for A rectangular. If A is rank deficient, a solution is returned if the system is consistent, Otherwise an info is 1

Parameters

F	base field
Side	Determine wheter the resolution is left (FflasLeft) or right (FflasRight) looking.
M	row dimension of A
N	col dimension of A
NRHS	number of columns (if Side = FFLAS::FflasLeft) or row (if Side = FFLAS::FflasRight) of the matrices X and B
R	rank of A
A	input matrix
lda	leading dimension of A
P	row permutation of the PLUQ decomposition of A

Parameters

Q	column permutation of the PLUQ decomposition of A
X	solution matrix
ldx	leading dimension of X
B	Right/Left hand side matrix.
ldb	leading dimension of B
info	Succes of the computation: 0 if successfull, >0 if system is inconsistent

10.5.2.5 fgesv() [1/2]

```
template<class Field >
size_t fgesv (
             const Field & F,
             const FFLAS::FFLAS_SIDE Side,
             const size_t M,
             const size_t N,
             typename Field::Element_ptr A,
             const size_t lda,
             typename Field::Element_ptr B,
             const size_t ldb,
             int * info )
```

Square system solver.

Parameters

F	The computation domain
Side	Determine wheter the resolution is left (FflasLeft) or right (FflasRight) looking
M	row dimension of B
N	col dimension of B
A	input matrix
lda	leading dimension of A
B	Right/Left hand side matrix. Initially contains B, finally contains the solution X.
ldb	leading dimension of B
info	Success of the computation: 0 if successfull, >0 if system is inconsistent

Returns

the rank of the system

Solve the system A X = B or X A = B. Version for A square. If A is rank deficient, a solution is returned if the system is consistent, Otherwise an info is 1

10.5.2.6 fgesv() [2/2]

```
template<class Field >
size_t fgesv (
             const Field & F,
```

```
            const FFLAS::FFLAS_SIDE Side,
            const size_t M,
            const size_t N,
            const size_t NRHS,
            typename Field::Element_ptr A,
            const size_t lda,
            typename Field::Element_ptr X,
            const size_t ldx,
            typename Field::ConstElement_ptr B,
            const size_t ldb,
            int * info )
```

Rectangular system solver.

Parameters

F	The computation domain
Side	Determine wheter the resolution is left (FflasLeft) or right (FflasRight) looking
M	row dimension of A
N	col dimension of A
NRHS	number of columns (if Side = FFLAS::FflasLeft) or row (if Side = FFLAS::FflasRight) of the matrices X and B
A	input matrix
lda	leading dimension of A
B	Right/Left hand side matrix. Initially contains B, finally contains the solution X.
ldb	leading dimension of B
X	
ldx	
info	Success of the computation: 0 if successfull, >0 if system is inconsistent

Returns

the rank of the system

Solve the system A X = B or X A = B. Version for A square. If A is rank deficient, a solution is returned if the system is consistent, Otherwise an info is 1

10.5.2.7 ftrtri()

```
template<class Field >
void ftrtri (
            const Field & F,
            const FFLAS::FFLAS_UPLO Uplo,
            const FFLAS::FFLAS_DIAG Diag,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            const size_t threshold = __FFLASFFPACK_FTRTRI_THRESHOLD )
```

Compute the inverse of a triangular matrix.

Parameters

F	base field
Uplo	whether the matrix is upper or lower triangular
Diag	whether the matrix is unit diagonal (FflasUnit/NoUnit)
N	input matrix order
A	the input matrix
lda	leading dimension of A

10.5.2.8 ftrtrm()

```
template<class Field >
void ftrtrm (
             const Field & F,
             const FFLAS::FFLAS_SIDE side,
             const FFLAS::FFLAS_DIAG diag,
             const size_t N,
             typename Field::Element_ptr A,
             const size_t lda )
```

Compute the product of two triangular matrices of opposite shape.

Product UL or LU of the upper, resp lower triangular matrices U and L stored one above the other in the square matrix A.

Parameters

F	base field
Side	set to FflasLeft to compute the product UL, FflasRight to compute LU
diag	whether the matrix U is unit diagonal (FflasUnit/NoUnit)
N	input matrix order
A	the input matrix
lda	leading dimension of A

10.5.2.9 ftrstr()

```
template<class Field >
void ftrstr (
             const Field & F,
             const FFLAS::FFLAS_SIDE side,
             const FFLAS::FFLAS_UPLO Uplo,
             const FFLAS::FFLAS_DIAG diagA,
             const FFLAS::FFLAS_DIAG diagB,
             const size_t N,
             typename Field::ConstElement_ptr A,
             const size_t lda,
             typename Field::Element_ptr B,
             const size_t ldb,
             const size_t threshold = 64 )  [inline]
```

Solve a triangular system with a triangular right hand side of the same shape.

Parameters

F	base field
Side	set to FflasLeft to compute U1^-1∗U2 or L1^-1∗L2, FflasRight to compute U1∗U2^-1 or L1∗L2^-1
Uplo	whether the matrix A is upper or lower triangular
diag1	whether the matrix U1 or L2 is unit diagonal (FflasUnit/NoUnit)
diag2	whether the matrix U2 or L2 is unit diagonal (FflasUnit/NoUnit)
N	order of the input matrices
A	the input matrix to be inverted (U1 or L1)
lda	leading dimension of A
B	the input right hand side (U2 or L2)
ldb	leading dimension of B

10.5.2.10 ftrssyr2k()

```
template<class Field >
void ftrssyr2k (
             const Field & F,
             const FFLAS::FFLAS_UPLO Uplo,
             const FFLAS::FFLAS_DIAG diagA,
             const size_t N,
             typename Field::ConstElement_ptr A,
             const size_t lda,
             typename Field::Element_ptr B,
             const size_t ldb,
             const size_t threshold = 64 )  [inline]
```

Solve a triangular system in a symmetric sum: find B upper/lower triangular such that A^T B + B^T A = C where C is symmetric.

C is overwritten by B.

Parameters

	F	base field
	Side	set to FflasLeft to compute U1^-1∗U2 or L1^-1∗L2, FflasRight to compute U1∗U2^-1 or L1∗L2^-1
	Uplo	whether the matrix A is upper or lower triangular
	diagA	whether the matrix A is unit diagonal (FflasUnit/NoUnit)
	N	order of the input matrices
	A	the input matrix
	lda	leading dimension of A
in,out	*B*	the input right hand side where the output is written
	ldb	leading dimension of B

10.5.2.11 fsytrf()

```
template<class Field >
bool fsytrf (
```

```
            const Field & F,
            const FFLAS::FFLAS_UPLO UpLo,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            const size_t threshold = __FFLASFFPACK_FSYTRF_THRESHOLD )
```

Triangular factorization of symmetric matrices.

Parameters

	F	The computation domain
	UpLo	Determine wheter to store the upper (FflasUpper) or lower (FflasLower) triangular factor
	N	order of the matrix A
in,out	*A*	input matrix
	lda	leading dimension of A

Returns

false if the A does not have generic rank profile, making the computation fail.

Compute the a triangular factorization of the matrix A: $A = L \times D \times L^T$ if UpLo = FflasLower or $A = U^T \times D \times U$ otherwise. D is a diagonal matrix. The matrices L and U are unit diagonal lower (resp. upper) triangular and overwrite the input matrix A. The matrix D is stored on the diagonal of A, as the diagonal of L or U is known to be all ones. If A does not have generic rank profile, the LDLT or UTDU factorizations is not defined, and the algorithm returns false.

10.5.2.12 fsytrf_nonunit()

```
template<class Field >
bool fsytrf_nonunit (
            const Field & F,
            const FFLAS::FFLAS_UPLO UpLo,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            typename Field::Element_ptr D,
            const size_t incD,
            const size_t threshold = __FFLASFFPACK_FSYTRF_THRESHOLD )
```

Triangular factorization of symmetric matrices.

Parameters

	F	The computation domain
	UpLo	Determine wheter to store the upper (FflasUpper) or lower (FflasLower) triangular factor
	N	order of the matrix A
in,out	*A*	input matrix
in,out	*D*	
	lda	leading dimension of A

Returns

false if the A does not have generic rank profile, making the computation fail.

Compute the a triangular factorization of the matrix A: $A = L \times Dinv \times L^T$ if UpLo = FflasLower or $A = U^T \times D \times U$ otherwise. D is a diagonal matrix. The matrices L and U are lower (resp. upper) triangular and overwrite the input matrix A. The matrix D need to be stored separately, as the diagonal of L or U are not unit. If A does not have generic rank profile, the LDLT or UTDU factorizations is not defined, and the algorithm returns false.

10.5.2.13 PLUQ()

```
template<class Field >
size_t PLUQ (
             const Field & F,
             const FFLAS::FFLAS_DIAG Diag,
             const size_t M,
             const size_t N,
             typename Field::Element_ptr A,
             const size_t lda,
             size_t * P,
             size_t * Q )  [inline]
```

Compute a PLUQ factorization of the given matrix.

Return its rank. The permutations P and Q are represented using LAPACK's convention.

Parameters

F	base field
Diag	whether U should have a unit diagonal (FflasUnit) or not (FflasNoUnit)
M	matrix row dimension
N	matrix column dimension
A	input matrix
lda	leading dimension of A
P	the row permutation
Q	the column permutation

Returns

the rank of A

Bibliography

- Dumas J-G., Pernet C., and Sultan Z. *Simultaneous computation of the row and column rank profiles* , ISSAC'13, 2013

10.5.2.14 LUdivine() [1/2]

```
template<class Field >
size_t LUdivine (
             const Field & F,
             const FFLAS::FFLAS_DIAG Diag,
```

```
            const FFLAS::FFLAS_TRANSPOSE trans,
            const size_t M,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            size_t * P,
            size_t * Qt,
            const FFPACK_LU_TAG LuTag = FfpackSlabRecursive,
            const size_t cutoff = __FFLASFFPACK_LUDIVINE_THRESHOLD )
```

Compute the CUP or PLE factorization of the given matrix.

Using a block algorithm and return its rank. The permutations P and Q are represented using LAPACK's convention.

Parameters

F	base field
Diag	whether the transformation matrix (U of the CUP, L of the PLE) should have a unit diagonal (FflasUnit) or not (FflasNoUnit)
trans	whether to compute the CUP decomposition (FflasNoTrans) or the PLE decomposition (FflasTrans)
M	matrix row dimension
N	matrix column dimension
A	input matrix
lda	leading dimension of A
P	the factor of CUP or PLE
Q	a permutation indicating the pivot position in the echelon form C or E in its first r positions
LuTag	flag for setting the earling termination if the matrix is singular
cutoff	threshold to basecase

Returns

the rank of A

Bibliography

- Jeannerod C-P, Pernet, C. and Storjohann, A. *Rank-profile revealing Gaussian elimination and the CUP matrix decomposition* , J. of Symbolic Comp., 2013
- Pernet C, Brassel M *LUdivine, une divine factorisation LU*, 2002

10.5.2.15 ColumnEchelonForm()

```
template<class Field >
size_t ColumnEchelonForm (
            const Field & F,
            const size_t M,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            size_t * P,
            size_t * Qt,
            bool transform = false,
            const FFPACK_LU_TAG LuTag = FfpackSlabRecursive )  [inline]
```

Compute the Column Echelon form of the input matrix in-place.

If LuTag == FfpackTileRecursive, then after the computation A = [M \ V] such that AU = C is a column echelon decomposition of A, with U = P^T [V] and C = M + Q [Ir] [0 In-r] [0] If LuTag == FfpackTileRecursive then A = [N \ V] such that the same holds with M = Q N

Qt = Q^T If transform=false, the matrix V is not computed. See also test-colechelon for an example of use

Parameters

	F	base field
	M	number of rows
	N	number of columns
in	*A*	input matrix
	lda	leading dimension of A
	P	the column permutation
	Qt	the row position of the pivots in the echelon form
	transform	decides whether V is computed
	LuTag	chooses the elimination algorithm. SlabRecursive for LUdivine, TileRecursive for PLUQ

10.5.2.16 RowEchelonForm()

```
template<class Field >
size_t RowEchelonForm (
            const Field & F,
            const size_t M,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            size_t * P,
            size_t * Qt,
            const bool transform = false,
            const FFPACK_LU_TAG LuTag = FfpackSlabRecursive )  [inline]
```

Compute the Row Echelon form of the input matrix in-place.

If LuTag == FfpackTileRecursive, then after the computation A = [L \ M] such that X A = R is a row echelon decomposition of A, with X = [L 0] P and R = M + [Ir 0] Q^T [In-r] If LuTag == FfpackTileRecursive then A = [L \ N] such that the same holds with M = N Q^T Qt = Q^T If transform=false, the matrix L is not computed. See also test-rowechelon for an example of use

Parameters

	F	base field
	M	number of rows
	N	number of columns
in	*A*	the input matrix
	lda	leading dimension of A
	P	the row permutation
	Qt	the column position of the pivots in the echelon form
	transform	decides whether L is computed
	LuTag	chooses the elimination algorithm. SlabRecursive for LUdivine, TileRecursive for PLUQ

10.5.2.17 ReducedColumnEchelonForm()

```
template<class Field >
size_t ReducedColumnEchelonForm (
```

```
            const Field & F,
            const size_t M,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            size_t * P,
            size_t * Qt,
            const bool transform = false,
            const FFPACK_LU_TAG LuTag = FfpackSlabRecursive )  [inline]
```

Compute the Reduced Column Echelon form of the input matrix in-place.

After the computation A = [V] such that AX = R is a reduced col echelon [M 0] decomposition of A, where X = P^T [V] and R = Q [Ir] [0 In-r] [M 0] Qt = Q^T If transform=false, the matrix X is not computed and the matrix A = R

Parameters

	F	base field
	M	number of rows
	N	number of columns
in	*A*	input matrix
	lda	leading dimension of A
	P	the column permutation
	Qt	the row position of the pivots in the echelon form
	transform	decides whether X is computed
	LuTag	chooses the elimination algorithm. SlabRecursive for LUdivine, TileRecursive for PLUQ

10.5.2.18 ReducedRowEchelonForm()

```
template<class Field >
size_t ReducedRowEchelonForm (
            const Field & F,
            const size_t M,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            size_t * P,
            size_t * Qt,
            const bool transform = false,
            const FFPACK_LU_TAG LuTag = FfpackSlabRecursive )  [inline]
```

Compute the Reduced Row Echelon form of the input matrix in-place.

After the computation A = [V1 M] such that X A = R is a reduced row echelon [V2 0] decomposition of A, where X = [V1 0] P and R = [Ir M] Q^T [V2 In-r] [0] Qt = Q^T If transform=false, the matrix X is not computed and the matrix A = R

Parameters

	F	base field
	M	number of rows
	N	number of columns
in	*A*	input matrix

Parameters

	lda	leading dimension of A
	P	the row permutation
	Qt	the column position of the pivots in the echelon form
	transform	decides whether X is computed
	LuTag	chooses the elimination algorithm. SlabRecursive for LUdivine, TileRecursive for PLUQ

10.5.2.19 Invert() [1/2]

```
template<class Field >
Field::Element_ptr Invert (
             const Field & F,
             const size_t M,
             typename Field::Element_ptr A,
             const size_t lda,
             int & nullity )
```

Invert the given matrix in place or computes its nullity if it is singular.

An inplace $2n^3$ algorithm is used.

Parameters

	F	The computation domain
	M	order of the matrix
in,out	*A*	input matrix ($M \times M$)
	lda	leading dimension of A
	nullity	dimension of the kernel of A

Returns

pointer to A and $A \leftarrow A^{-1}$

10.5.2.20 Invert() [2/2]

```
template<class Field >
Field::Element_ptr Invert (
             const Field & F,
             const size_t M,
             typename Field::ConstElement_ptr A,
             const size_t lda,
             typename Field::Element_ptr X,
             const size_t ldx,
             int & nullity )
```

Invert the given matrix or computes its nullity if it is singular.

Precondition

X is preallocated and should be large enough to store the $m \times m$ matrix A.

Parameters

	F	The computation domain
	M	order of the matrix
in	*A*	input matrix ($M \times M$)
	lda	leading dimension of A
out	*X*	this is the inverse of A if A is invertible (non NULL and nullity $= 0$). It is untouched otherwise.
	ldx	leading dimension of X
	nullity	dimension of the kernel of A

Returns

pointer to $X = A^{-1}$

10.5.2.21 Invert2()

```
template<class Field >
Field::Element_ptr Invert2 (
            const Field & F,
            const size_t M,
            typename Field::Element_ptr A,
            const size_t lda,
            typename Field::Element_ptr X,
            const size_t ldx,
            int & nullity )
```

Invert the given matrix or computes its nullity if it is singular.

An $2n^3 f$ algorithm is used. This routine can be % faster than FFPACK::Invert but is not totally inplace.

Precondition

X is preallocated and should be large enough to store the $m \times m$ matrix A.

Warning

A is overwritten here !

Bug not tested.

Parameters

	F	the computation domain
	M	order of the matrix
in,out	*A*	input matrix ($M \times M$). On output, A is modified and represents a "psycological" factorisation LU.
	lda	leading dimension of A
out	*X*	this is the inverse of A if A is invertible (non NULL and nullity $= 0$). It is untouched otherwise.
	ldx	leading dimension of X
	nullity	dimension of the kernel of A

Returns

pointer to $X = A^{-1}$

Todo this init is not all necessary (done after ftrtri)

Todo this init is not all necessary (done after ftrtri)

10.5.2.22 CharPoly() [1/3]

```
template<class PolRing >
std::list< typename PolRing::Element > & CharPoly (
             const PolRing & R,
             std::list< typename PolRing::Element > & charp,
             const size_t N,
             typename PolRing::Domain_t::Element_ptr A,
             const size_t lda,
             typename PolRing::Domain_t::RandIter & G,
             const FFPACK_CHARPOLY_TAG CharpTag = FfpackAuto,
             const size_t degree = __FFLASFFPACK_ARITHPROG_THRESHOLD )  [inline]
```

Compute the characteristic polynomial of the matrix A.

Parameters

	R	the polynomial ring of charp (contains the base field)
out	*charp*	the characteristic polynomial of as a list of factors
	N	order of the matrix A
in	*A*	the input matrix ($N \times N$) (could be overwritten in some algorithmic variants)
	lda	leading dimension of A
	CharpTag	the algorithmic variant
	G	a random iterator (required for the randomized variants LUKrylov and ArithProg)

10.5.2.23 CharPoly() [2/3]

```
template<class PolRing >
PolRing::Element & CharPoly (
             const PolRing & R,
             typename PolRing::Element & charp,
             const size_t N,
             typename PolRing::Domain_t::Element_ptr A,
             const size_t lda,
             typename PolRing::Domain_t::RandIter & G,
             const FFPACK_CHARPOLY_TAG CharpTag = FfpackAuto,
             const size_t degree = __FFLASFFPACK_ARITHPROG_THRESHOLD )  [inline]
```

Compute the characteristic polynomial of the matrix A.

Parameters

	R	the polynomial ring of charp (contains the base field)

Parameters

out	*charp*	the characteristic polynomial of as a single polynomial
	N	order of the matrix A
in	*A*	the input matrix ($N \times N$) (could be overwritten in some algorithmic variants)
	lda	leading dimension of A
	CharpTag	the algorithmic variant
	G	a random iterator (required for the randomized variants LUKrylov and ArithProg)

10.5.2.24 CharPoly() [3/3]

```
template<class PolRing >
PolRing::Element & CharPoly (
            const PolRing & R,
            typename PolRing::Element & charp,
            const size_t N,
            typename PolRing::Domain_t::Element_ptr A,
            const size_t lda,
            const FFPACK_CHARPOLY_TAG CharpTag = FfpackAuto,
            const size_t degree = __FFLASFFPACK_ARITHPROG_THRESHOLD )  [inline]
```

Compute the characteristic polynomial of the matrix A.

Parameters

	R	the polynomial ring of charp (contains the base field)
out	*charp*	the characteristic polynomial of as a single polynomial
	N	order of the matrix A
in	*A*	the input matrix ($N \times N$) (could be overwritten in some algorithmic variants)
	lda	leading dimension of A
	CharpTag	the algorithmic variant

10.5.2.25 MinPoly() [1/2]

```
template<class Field , class Polynomial >
Polynomial & MinPoly (
            const Field & F,
            Polynomial & minP,
            const size_t N,
            typename Field::ConstElement_ptr A,
            const size_t lda )  [inline]
```

Compute the minimal polynomial of the matrix A.

The algorithm is randomized probabilistic, and computes the minimal polynomial of the Krylov iterates of a random vector: (v, Av, .., A^kv)

Parameters

	F	the base field

Parameters

out	*minP*	the minimal polynomial of A
	N	order of the matrix A
in	*A*	the input matrix ($N \times N$)
	lda	leading dimension of A

10.5.2.26 MinPoly() [2/2]

```
template<class Field , class Polynomial , class RandIter >
Polynomial & MinPoly (
             const Field & F,
             Polynomial & minP,
             const size_t N,
             typename Field::ConstElement_ptr A,
             const size_t lda,
             RandIter & G )  [inline]
```

Compute the minimal polynomial of the matrix A.

The algorithm is randomized probabilistic, and computes the minimal polynomial of the Krylov iterates of a random vector: (v, Av, .., A^kv)

Parameters

	F	the base field
out	*minP*	the minimal polynomial of A
	N	order of the matrix A
in	*A*	the input matrix ($N \times N$)
	lda	leading dimension of A
	G	a random iterator

10.5.2.27 MatVecMinPoly()

```
template<class Field , class Polynomial >
Polynomial & MatVecMinPoly (
             const Field & F,
             Polynomial & minP,
             const size_t N,
             typename Field::ConstElement_ptr A,
             const size_t lda,
             typename Field::ConstElement_ptr v,
             const size_t incv )  [inline]
```

Compute the minimal polynomial of the matrix A and a vector v, namely the first linear dependency relation in the Krylov basis $(v, Av, ..., A^N v)$.

Parameters

	F	the base field

Parameters

out	*minP*	the minimal polynomial of A and v
	N	order of the matrix A
in	*A*	the input matrix ($N \times N$)
	lda	leading dimension of A
	K	an $N \times (N+1)$ matrix containing the vector v on its first row
	ldk	leading dimension of K
	P	[out] (optional) the permutation used in the elimination of the Krylov matrix K

10.5.2.28 Rank()

```
template<class Field >
size_t Rank (
             const Field & F,
             const size_t M,
             const size_t N,
             typename Field::Element_ptr A,
             const size_t lda )
```

Computes the rank of the given matrix using a PLUQ factorization.

The input matrix is modified.

Parameters

	F	base field
	M	row dimension of the matrix
	N	column dimension of the matrix
in	*A*	input matrix
	lda	leading dimension of A
	psH	(optional) a ParSeqHelper to choose between sequential and parallel execution

10.5.2.29 IsSingular()

```
template<class Field >
bool IsSingular (
             const Field & F,
             const size_t M,
             const size_t N,
             typename Field::Element_ptr A,
             const size_t lda )
```

Returns true if the given matrix is singular.

The method is a block elimination with early termination

using LQUP factorization with early termination. If M != N, then the matrix is virtually padded with zeros to make it square and it's determinant is zero.

Warning

The input matrix is modified.

Parameters

	F	base field
	M	row dimension of the matrix
	N	column dimension of the matrix.
in,out	*A*	input matrix
	lda	leading dimension of A

10.5.2.30 Det()

```
template<class Field >
Field::Element & Det (
             const Field & F,
             typename Field::Element & det,
             const size_t N,
             typename Field::Element_ptr A,
             const size_t lda,
             size_t * P = NULL,
             size_t * Q = NULL )  [inline]
```

Returns the determinant of the given square matrix.

The method is a block elimination using PLUQ factorization. The input matrix A is overwritten.

Warning

The input matrix is modified.

Parameters

	F	base field
out	*det*	the determinant of A
	N	the order of the square matrix A.
in,out	*A*	input matrix
	lda	leading dimension of A
	psH	(optional) a ParSeqHelper to choose between sequential and parallel execution
	P,Q	(optional) row and column permutations to be used by the PLUQ factorization. randomized checkers (see cherckes/checker_det.inl) need them for certification

10.5.2.31 Solve()

```
template<class Field >
Field::Element_ptr Solve (
             const Field & F,
             const size_t M,
             typename Field::Element_ptr A,
             const size_t lda,
             typename Field::Element_ptr x,
             const int incx,
```

```
            typename Field::ConstElement_ptr b,
            const int incb )  [inline]
```

Solves a linear system AX = b using PLUQ factorization.

@oaram F base field @oaram M matrix order

Parameters

in	*A*	input matrix
	lda	leading dimension of A
out	*x*	output solution vector
	incx	increment of x
	b	input right hand side of the system
	incb	increment of b

10.5.2.32 RandomNullSpaceVector() [1/2]

```
template<class Field >
*void RandomNullSpaceVector (
            const Field & F,
            const FFLAS::FFLAS_SIDE Side,
            const size_t M,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            typename Field::Element_ptr X,
            const size_t incX )
```

Solve L X = B or X L = B in place.

L is M∗M if Side == FFLAS::FflasLeft and N∗N if Side == FFLAS::FflasRight, B is M∗N. Only the R non trivial column of L are stored in the M∗R matrix L Requirement : so that L could be expanded in-place Computes a vector of the Left/Right nullspace of the matrix A.

Parameters

	F	The computation domain
	Side	decides whether it computes the left (FflasLeft) or right (FflasRight) nullspace
	M	number of rows
	N	number of columns
in,out	*A*	input matrix of dimension M x N, A is modified to its LU version
	lda	leading dimension of A
out	*X*	output vector
	incX	increment of X

10.5.2.33 NullSpaceBasis()

```
template<class Field >
size_t NullSpaceBasis (
```

```
            const Field & F,
            const FFLAS::FFLAS_SIDE Side,
            const size_t M,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            typename Field::Element_ptr & NS,
            size_t & ldn,
            size_t & NSdim )
```

Computes a basis of the Left/Right nullspace of the matrix A.

return the dimension of the nullspace.

Parameters

	F	The computation domain
	Side	decides whether it computes the left (FflasLeft) or right (FflasRight) nullspace
	M	number of rows
	N	number of columns
in,out	*A*	input matrix of dimension M x N, A is modified
	lda	leading dimension of A
out	*NS*	output matrix of dimension N x NSdim (allocated here)
out	*ldn*	leading dimension of NS
out	*NSdim*	the dimension of the Nullspace (N-rank(A))

10.5.2.34 RowRankProfile()

```
template<class Field >
size_t RowRankProfile (
            const Field & F,
            const size_t M,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            size_t *& rkprofile,
            const FFPACK_LU_TAG LuTag = FfpackSlabRecursive )  [inline]
```

Computes the row rank profile of A.

Parameters

	F	base field
	M	number of rows
	N	number of columns
in	*A*	input matrix of dimension M x N
	lda	leading dimension of A
out	*rkprofile*	return the rank profile as an array of row indexes, of dimension r=rank(A)
	LuTag	chooses the elimination algorithm. SlabRecursive for LUdivine, TileRecursive for PLUQ

A is modified rkprofile is allocated during the computation.

Returns

R

10.5.2.35 ColumnRankProfile()

```
template<class Field >
size_t ColumnRankProfile (
            const Field & F,
            const size_t M,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            size_t *& rkprofile,
            const FFPACK_LU_TAG LuTag = FfpackSlabRecursive )  [inline]
```

Computes the column rank profile of A.

Parameters

	F	base field
	M	number of rows
	N	number of columns
in	*A*	input matrix of dimension
	lda	leading dimension of A
out	*rkprofile*	return the rank profile as an array of row indexes, of dimension r=rank(A)
	LuTag	chooses the elimination algorithm. SlabRecursive for LUdivine, TileRecursive for PLUQ

A is modified rkprofile is allocated during the computation.

Returns

R

10.5.2.36 RankProfileFromLU()

```
void RankProfileFromLU (
            const size_t * P,
            const size_t N,
            const size_t R,
            size_t * rkprofile,
            const FFPACK_LU_TAG LuTag )  [inline]
```

Recovers the column/row rank profile from the permutation of an LU decomposition.

Works with both the CUP/PLE decompositions (obtained by LUdivine) or the PLUQ decomposition. Assumes that the output vector containing the rank profile is already allocated.

Parameters

	P	the permutation carrying the rank profile information
	N	the row/col dimension for a row/column rank profile
	R	the rank of the matrix
out	*rkprofile*	return the rank profile as an array of indices
	LuTag	chooses the elimination algorithm. SlabRecursive for LUdivine, TileRecursive for PLUQ

10.5.2.37 LeadingSubmatrixRankProfiles()

```
size_t LeadingSubmatrixRankProfiles (
              const size_t M,
              const size_t N,
              const size_t R,
              const size_t LSm,
              const size_t LSn,
              const size_t * P,
              const size_t * Q,
              size_t * RRP,
              size_t * CRP )  [inline]
```

Recovers the row and column rank profiles of any leading submatrix from the PLUQ decomposition.

Only works with the PLUQ decomposition Assumes that the output vectors containing the rank profiles are already allocated.

Parameters

P	the permutation carrying the rank profile information
M	the row dimension of the initial matrix
N	the column dimension of the initial matrix
R	the rank of the initial matrix
LSm	the row dimension of the leading submatrix considered
LSn	the column dimension of the leading submatrix considered
P	the row permutation of the PLUQ decomposition
Q	the column permutation of the PLUQ decomposition
RRP	return the row rank profile of the leading submatrix

Returns

the rank of the LSm x LSn leading submatrix

A is modified

Bibliography

- Dumas J-G., Pernet C., and Sultan Z. *Simultaneous computation of the row and column rank profiles* , ISSAC'13.

10.5.2.38 RowRankProfileSubmatrixIndices()

```
template<class Field >
size_t RowRankProfileSubmatrixIndices (
              const Field & F,
              const size_t M,
              const size_t N,
              typename Field::Element_ptr A,
              const size_t lda,
              size_t *& rowindices,
              size_t *& colindices,
              size_t & R )
```

RowRankProfileSubmatrixIndices.

Computes the indices of the submatrix r∗r X of A whose rows correspond to the row rank profile of A.

Parameters

	F	base field
	M	number of rows
	N	number of columns
in	*A*	input matrix of dimension
	rowindices	array of the row indices of X in A
	colindices	array of the col indices of X in A
	lda	leading dimension of A
out	*R*	list of indices

rowindices and colindices are allocated during the computation. A is modified

Returns

R

10.5.2.39 ColRankProfileSubmatrixIndices()

```
template<class Field >
size_t ColRankProfileSubmatrixIndices (
            const Field & F,
            const size_t M,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            size_t *& rowindices,
            size_t *& colindices,
            size_t & R )
```

Computes the indices of the submatrix r∗r X of A whose columns correspond to the column rank profile of A.

Parameters

	F	base field
	M	number of rows
	N	number of columns
in	*A*	input matrix of dimension
	rowindices	array of the row indices of X in A
	colindices	array of the col indices of X in A
	lda	leading dimension of A
out	*R*	list of indices

rowindices and colindices are allocated during the computation.

Warning

A is modified

Returns

R

10.5.2.40 RowRankProfileSubmatrix()

```
template<class Field >
size_t RowRankProfileSubmatrix (
            const Field & F,
            const size_t M,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            typename Field::Element_ptr & X,
            size_t & R )
```

Computes the r∗r submatrix X of A, by picking the row rank profile rows of A.

Parameters

	F	base field
	M	number of rows
	N	number of columns
in	*A*	input matrix of dimension M x N
	lda	leading dimension of A
out	*X*	the output matrix
out	*R*	list of indices

A is not modified X is allocated during the computation.

Returns

R

10.5.2.41 ColRankProfileSubmatrix()

```
template<class Field >
size_t ColRankProfileSubmatrix (
            const Field & F,
            const size_t M,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            typename Field::Element_ptr & X,
            size_t & R )
```

Compute the $r \times r$ submatrix X of A, by picking the row rank profile rows of A.

Parameters

	F	base field
	M	number of rows

Parameters

	N	number of columns
in	*A*	input matrix of dimension M x N
	lda	leading dimension of A
out	*X*	the output matrix
out	*R*	list of indices

A is not modified X is allocated during the computation.

Returns

R

10.5.2.42 getTriangular() [1/2]

```
template<class Field >
void getTriangular (
            const Field & F,
            const FFLAS::FFLAS_UPLO Uplo,
            const FFLAS::FFLAS_DIAG diag,
            const size_t M,
            const size_t N,
            const size_t R,
            typename Field::ConstElement_ptr A,
            const size_t lda,
            typename Field::Element_ptr T,
            const size_t ldt,
            const bool OnlyNonZeroVectors = false )  [inline]
```

Extracts a triangular matrix from a compact storage A=L\U of rank R.

if OnlyNonZeroVectors is false, then T and A have the same dimensions Otherwise, T is R x N if UpLo = FflasUpper, else T is M x R

Parameters

	F	base field
	UpLo	selects if the upper (FflasUpper) or lower (FflasLower) triangular matrix is returned
	diag	selects if the triangular matrix unit-diagonal (FflasUnit/NoUnit)
	M	row dimension of T
	N	column dimension of T
	R	rank of the triangular matrix (how many rows/columns need to be copied)
in	*A*	input matrix
	lda	leading dimension of A
out	*T*	output matrix
	ldt	leading dimension of T
	OnlyNonZeroVectors	decides whether the last zero rows/columns should be ignored

Todo just one triangular fzero+fassign ?

Todo just one triangular fzero+fassign ?

10.5.2.43 getTriangular() [2/2]

```
template<class Field >
void getTriangular (
             const Field & F,
             const FFLAS::FFLAS_UPLO Uplo,
             const FFLAS::FFLAS_DIAG diag,
             const size_t M,
             const size_t N,
             const size_t R,
             typename Field::Element_ptr A,
             const size_t lda )  [inline]
```

Cleans up a compact storage A=L\U to reveal a triangular matrix of rank R.

Parameters

	F	base field
	UpLo	selects if the upper (FflasUpper) or lower (FflasLower) triangular matrix is revealed
	diag	selects if the triangular matrix unit-diagonal (FflasUnit/NoUnit)
	M	row dimension of A
	N	column dimension of A
	R	rank of the triangular matrix
in,out	*A*	input/output matrix
	lda	leading dimension of A

Todo just one triangular fzero+fassign ?

Todo just one triangular fzero+fassign ?

10.5.2.44 getEchelonForm() [1/2]

```
template<class Field >
void getEchelonForm (
             const Field & F,
             const FFLAS::FFLAS_UPLO Uplo,
             const FFLAS::FFLAS_DIAG diag,
             const size_t M,
             const size_t N,
             const size_t R,
             const size_t * P,
             typename Field::ConstElement_ptr A,
             const size_t lda,
             typename Field::Element_ptr T,
```

```
            const size_t ldt,
            const bool OnlyNonZeroVectors = false,
            const FFPACK_LU_TAG LuTag = FfpackSlabRecursive )  [inline]
```

Extracts a matrix in echelon form from a compact storage A=L\U of rank R obtained by RowEchelonForm or ColumnEchelonForm.

Either L or U is in Echelon form (depending on Uplo) The echelon structure is defined by the first R values of the array P. row and column dimension of T are greater or equal to that of A

Parameters

	F	base field
	UpLo	selects if the upper (FflasUpper) or lower (FflasLower) triangular matrix is returned
	diag	selects if the echelon matrix has unit pivots (FflasUnit/NoUnit)
	M	row dimension of T
	N	column dimension of T
	R	rank of the triangular matrix (how many rows/columns need to be copied)
	P	positions of the R pivots
in	*A*	input matrix
	lda	leading dimension of A
out	*T*	output matrix
	ldt	leading dimension of T
	OnlyNonZeroVectors	decides whether the last zero rows/columns should be ignored
	LuTag	which factorized form (CUP/PLE if FfpackSlabRecursive, PLUQ if FfpackTileRecursive)

10.5.2.45 getEchelonForm() [2/2]

```
template<class Field >
void getEchelonForm (
            const Field & F,
            const FFLAS::FFLAS_UPLO Uplo,
            const FFLAS::FFLAS_DIAG diag,
            const size_t M,
            const size_t N,
            const size_t R,
            const size_t * P,
            typename Field::Element_ptr A,
            const size_t lda,
            const FFPACK_LU_TAG LuTag = FfpackSlabRecursive )  [inline]
```

Cleans up a compact storage A=L\U obtained by RowEchelonForm or ColumnEchelonForm to reveal an echelon form of rank R.

Either L or U is in Echelon form (depending on Uplo) The echelon structure is defined by the first R values of the array P.

Parameters

	F	base field
	UpLo	selects if the upper (FflasUpper) or lower (FflasLower) triangular matrix is returned

Parameters

	diag	selects if the echelon matrix has unit pivots (FflasUnit/NoUnit)
	M	row dimension of A
	N	column dimension of A
	R	rank of the triangular matrix (how many rows/columns need to be copied)
	P	positions of the R pivots
in,out	*A*	input/output matrix
	lda	leading dimension of A
	LuTag	which factorized form (CUP/PLE if FfpackSlabRecursive, PLUQ if FfpackTileRecursive)

10.5.2.46 getEchelonTransform()

```
template<class Field >
void getEchelonTransform (
            const Field & F,
            const FFLAS::FFLAS_UPLO Uplo,
            const FFLAS::FFLAS_DIAG diag,
            const size_t M,
            const size_t N,
            const size_t R,
            const size_t * P,
            const size_t * Q,
            typename Field::ConstElement_ptr A,
            const size_t lda,
            typename Field::Element_ptr T,
            const size_t ldt,
            const FFPACK_LU_TAG LuTag = FfpackSlabRecursive )  [inline]
```

Extracts a transformation matrix to echelon form from a compact storage A=L\U of rank R obtained by Row↩ EchelonForm or ColumnEchelonForm.

If Uplo == FflasLower: T is N x N (already allocated) such that A T = C is a transformation of A in Column echelon form Else T is M x M (already allocated) such that T A = E is a transformation of A in Row Echelon form

Parameters

	F	base field
	UpLo	Lower (FflasLower) means Transformation to Column Echelon Form, Upper (FflasUpper), to Row Echelon Form
	diag	selects if the echelon matrix has unit pivots (FflasUnit/NoUnit)
	M	row dimension of A
	N	column dimension of A
	R	rank of the triangular matrix
	P	permutation matrix
in	*A*	input matrix
	lda	leading dimension of A
out	*T*	output matrix
	ldt	leading dimension of T
	LuTag	which factorized form (CUP/PLE if FfpackSlabRecursive, PLUQ if FfpackTileRecursive)

10.5.2.47 getReducedEchelonForm() [1/2]

```
template<class Field >
void getReducedEchelonForm (
             const Field & F,
             const FFLAS::FFLAS_UPLO Uplo,
             const size_t M,
             const size_t N,
             const size_t R,
             const size_t * P,
             typename Field::ConstElement_ptr A,
             const size_t lda,
             typename Field::Element_ptr T,
             const size_t ldt,
             const bool OnlyNonZeroVectors = false,
             const FFPACK_LU_TAG LuTag = FfpackSlabRecursive )  [inline]
```

Extracts a matrix in echelon form from a compact storage A=L\U of rank R obtained by ReducedRowEchelonForm or ReducedColumnEchelonForm with transform = true.

Either L or U is in Echelon form (depending on Uplo) The echelon structure is defined by the first R values of the array P. row and column dimension of T are greater or equal to that of A

Parameters

	F	base field
	UpLo	selects if the upper (FflasUpper) or lower (FflasLower) triangular matrix is returned
	diag	selects if the echelon matrix has unit pivots (FflasUnit/NoUnit)
	M	row dimension of T
	N	column dimension of T
	R	rank of the triangular matrix (how many rows/columns need to be copied)
	P	positions of the R pivots
in	*A*	input matrix
	lda	leading dimension of A
	ldt	leading dimension of T
	LuTag	which factorized form (CUP/PLE if FfpackSlabRecursive, PLUQ if FfpackTileRecursive)
	OnlyNonZeroVectors	decides whether the last zero rows/columns should be ignored

10.5.2.48 getReducedEchelonForm() [2/2]

```
template<class Field >
void getReducedEchelonForm (
             const Field & F,
             const FFLAS::FFLAS_UPLO Uplo,
             const size_t M,
             const size_t N,
             const size_t R,
             const size_t * P,
             typename Field::Element_ptr A,
             const size_t lda,
             const FFPACK_LU_TAG LuTag = FfpackSlabRecursive )  [inline]
```

Cleans up a compact storage A=L\U of rank R obtained by ReducedRowEchelonForm or ReducedColumn↩ EchelonForm with transform = true.

Either L or U is in Echelon form (depending on Uplo) The echelon structure is defined by the first R values of the array P.

Parameters

	F	base field
	UpLo	selects if the upper (FflasUpper) or lower (FflasLower) triangular matrix is returned
	diag	selects if the echelon matrix has unit pivots (FflasUnit/NoUnit)
	M	row dimension of A
	N	column dimension of A
	R	rank of the triangular matrix (how many rows/columns need to be copied)
	P	positions of the R pivots
in,out	*A*	input/output matrix
	lda	leading dimension of A
	LuTag	which factorized form (CUP/PLE if FfpackSlabRecursive, PLUQ if FfpackTileRecursive)

10.5.2.49 getReducedEchelonTransform()

```
template<class Field >
void getReducedEchelonTransform (
             const Field & F,
             const FFLAS::FFLAS_UPLO Uplo,
             const size_t M,
             const size_t N,
             const size_t R,
             const size_t * P,
             const size_t * Q,
             typename Field::ConstElement_ptr A,
             const size_t lda,
             typename Field::Element_ptr T,
             const size_t ldt,
             const FFPACK_LU_TAG LuTag = FfpackSlabRecursive )  [inline]
```

Extracts a transformation matrix to echelon form from a compact storage A=L\U of rank R obtained by Row↩ EchelonForm or ColumnEchelonForm.

If Uplo == FflasLower: T is N x N (already allocated) such that A T = C is a transformation of A in Column echelon form Else T is M x M (already allocated) such that T A = E is a transformation of A in Row Echelon form

Parameters

	F	base field
	UpLo	selects Col (FflasLower) or Row (FflasUpper) Echelon Form
	diag	selects if the echelon matrix has unit pivots (FflasUnit/NoUnit)
	M	row dimension of A
	N	column dimension of A
	R	rank of the triangular matrix
	P	permutation matrix
in	*A*	input matrix

Parameters

	lda	leading dimension of A
out	*T*	output matrix
	ldt	leading dimension of T
	LuTag	which factorized form (CUP/PLE if FfpackSlabRecursive, PLUQ if FfpackTileRecursive)

10.5.2.50 LTBruhatGen()

```
template<class Field >
size_t LTBruhatGen (
             const Field & Fi,
             const FFLAS::FFLAS_DIAG diag,
             const size_t N,
             typename Field::Element_ptr A,
             const size_t lda,
             size_t * P,
             size_t * Q )  [inline]
```

LTBruhatGen Suppose A is Left Triangular Matrix This procedure computes the Bruhat Representation of A and return the rank of A.

Parameters

Fi	base Field
diag	
N	size of A
A	the matrix we search the Bruhat representation
lda	the leading dimension of A
P	a permutation matrix
Q	a permutation matrix

10.5.2.51 getLTBruhatGen() [1/2]

```
template<class Field >
void getLTBruhatGen (
             const Field & Fi,
             const size_t N,
             const size_t r,
             const size_t * P,
             const size_t * Q,
             typename Field::Element_ptr R,
             const size_t ldr )  [inline]
```

GetLTBruhatGen This procedure Computes the Rank Revealing Matrix based on the Bruhta representation of a Matrix.

Parameters

Fi	base Field
N	size of the matrix

Parameters

r	the rank of the matrix
P	a permutation matrix
Q	a permutation matrix
R	the matrix that will contain the rank revealing matrix
ldr	the leading fimension of R

10.5.2.52 getLTBruhatGen() [2/2]

```
template<class Field >
void getLTBruhatGen (
             const Field & Fi,
             const FFLAS::FFLAS_UPLO Uplo,
             const FFLAS::FFLAS_DIAG diag,
             const size_t N,
             const size_t r,
             const size_t * P,
             const size_t * Q,
             typename Field::ConstElement_ptr A,
             const size_t lda,
             typename Field::Element_ptr T,
             const size_t ldt )  [inline]
```

GetLTBruhatGen This procedure computes the matrix L or U f the Bruhat Representation Suppose that A is the bruhat representation of a matrix.

Parameters

Fi	base Field
Uplo	choose if the procedure return L or U
diag	
N	size of A
r	rank of A
P	permutaion matrix
Q	permutation matrix
A	a bruhat representation
lda	leading dimension of A
T	matrix that will contains L or U
ldt	leading dimension of T

10.5.2.53 LTQSorder()

```
size_t LTQSorder (
             const size_t N,
             const size_t r,
             const size_t * P,
             const size_t * Q )  [inline]
```

LTQSorder This procedure computes the order of quasiseparability of a matrix.

Parameters

N	size of the matrix
r	rank of the matrix
P	permutation matrix
Q	permutation matrix

10.5.2.54 CompressToBlockBiDiagonal()

```
template<class Field >
size_t CompressToBlockBiDiagonal (
            const Field & Fi,
            const FFLAS::FFLAS_UPLO Uplo,
            size_t N,
            size_t s,
            size_t r,
            const size_t * P,
            const size_t * Q,
            typename Field::Element_ptr A,
            size_t lda,
            typename Field::Element_ptr X,
            size_t ldx,
            size_t * K,
            size_t * M,
            size_t * T )  [inline]
```

CompressToBlockBiDiagonal This procedure compress a compact representation of a row echelon form or column echelon form.

Parameters

Fi	base Field
Uplo	chosse if the procedure is based on row or column
N	size of the matrix
s	order of qausiseparability
r	rank
P	permutation matrix
Q	permutation matrix
A	the matrix to compact
lda	leading dimension of A
X	matrix that will stock the representation
ldx	leading dimension of X
K	stock the position of the blocks in A
M	permutation matrix
T	stock the operation done in the procedure

10.5.2.55 ExpandBlockBiDiagonalToBruhat()

```
template<class Field >
void ExpandBlockBiDiagonalToBruhat (
```

```
            const Field & Fi,
            const FFLAS::FFLAS_UPLO Uplo,
            size_t N,
            size_t s,
            size_t r,
            typename Field::Element_ptr A,
            size_t lda,
            typename Field::Element_ptr X,
            size_t ldx,
            size_t NbBlocks,
            size_t * K,
            size_t * M,
            size_t * T )  [inline]
```

ExpandBlockBiDiagonal This procedure expand a compact representation of a row echelon form or column echelon form.

Parameters

Fi	base Field
Uplo	chosse if the procedure is based on row or column
N	size of the matrix
s	order of qausiseparability
r	rank
A	the matrix that will sotck the expanded representation
lda	leading dimension of A
X	matrix to expand
ldx	leading dimension of X
K	stock the position of the blocks in A
M	permutation matrix
T	stock the operation done in the procedure

10.5.2.56 Bruhat2EchelonPermutation()

```
void Bruhat2EchelonPermutation (
            size_t N,
            size_t R,
            const size_t * P,
            const size_t * Q,
            size_t * M )  [inline]
```

Bruhat2EchelonPermutation (N,R,P,Q) Compute M such that LM or MU is in echelon form where L or U are factors of the Bruhat Rpresentation.

Parameters

in	*N*	size of the matrix
in	*R*	rank
in	*P*	permutation Matrix
in	*Q*	permutation Matrix
out	*M*	output permutation matrix

10.5.2.57 LQUPtoInverseOfFullRankMinor()

```
template<class Field >
Field::Element_ptr LQUPtoInverseOfFullRankMinor (
             const Field & F,
             const size_t rank,
             typename Field::Element_ptr A_factors,
             const size_t lda,
             const size_t * QtPointer,
             typename Field::Element_ptr X,
             const size_t ldx )
```

LQUPtoInverseOfFullRankMinor.

Suppose A has been factorized as L.Q.U.P, with rank r. Then Qt.A.Pt has an invertible leading principal r x r submatrix This procedure efficiently computes the inverse of this minor and puts it into X.

Note

It changes the lower entries of A_factors in the process (NB: unless A was nonsingular and square)

Parameters

F	base field
rank	rank of the matrix.
A_factors	matrix containing the L and U entries of the factorization
lda	leading dimension of A
QtPointer	theLQUP->getQ()->getPointer() (note: getQ returns Qt!)
X	desired location for output
ldx	leading dimension of X

10.5.2.58 RandomNullSpaceVector() [2/2]

```
template<class Field >
void RandomNullSpaceVector (
             const Field & F,
             const FFLAS::FFLAS_SIDE Side,
             const size_t M,
             const size_t N,
             typename Field::Element_ptr A,
             const size_t lda,
             typename Field::Element_ptr X,
             const size_t incX )
```

Solve L X = B or X L = B in place.

L is M∗M if Side == FFLAS::FflasLeft and N∗N if Side == FFLAS::FflasRight, B is M∗N. Only the R non trivial column of L are stored in the M∗R matrix L Requirement : so that L could be expanded in-place Computes a vector of the Left/Right nullspace of the matrix A.

Parameters

	F	The computation domain

Parameters

	Side	decides whether it computes the left (FflasLeft) or right (FflasRight) nullspace
	M	number of rows
	N	number of columns
in,out	*A*	input matrix of dimension M x N, A is modified to its LU version
	lda	leading dimension of A
out	*X*	output vector
	incX	increment of X

10.5.2.59 productBruhatxTS()

```
template<class Field >
void productBruhatxTS (
            const Field & Fi,
            size_t N,
            size_t s,
            size_t r,
            size_t t,
            const size_t * P,
            const size_t * Q,
            typename Field::ConstElement_ptr Xu,
            size_t ldu,
            size_t NbBlocksU,
            const size_t * Ku,
            const size_t * Tu,
            const size_t * MU,
            typename Field::ConstElement_ptr Xl,
            size_t ldl,
            size_t NbBlocksL,
            const size_t * Kl,
            const size_t * Tl,
            const size_t * ML,
            typename Field::Element_ptr B,
            size_t ldb,
            const typename Field::Element beta,
            typename Field::Element_ptr D,
            size_t ldd )  [inline]
```

Compute the product of a left-triangular quasi-separable matrix A, represented by a compact Bruhat generator, with a dense rectangular matrix B: $C \leftarrow A \times B + betaC$.

Parameters

	F	the base field
	N	the order of A
	s	the order of quasiseparability of A
	r	the number of pivots in the left-triangular par of the rank profile matrix of A
	t	the number of columns of B
	P	the row indices of the pivots of A
	Q	the column indices of the pivots of A
	Xu	the compact storage of U: Du blocks in the first s rows, Su blocks in the last s rows
	ldxu	the leading dimension of Xu

Parameters

	NbBlocksU	the number of diagonal blocks in the compact storage of U
	Ku	the list of starting column positions for each block of the storage of U
	Tu	the folding matrix for the compact storage of U: $Du + TuSu$ is in row echelon form
	Mu	a permutation matrix such that $Mu(Du + TuSu)$ is the U factor of the Bruhat generator
	Xl	the compact storage of L: Dl blocks in the first s columns, Sl blocks in the last s columns
	ldxl	the leading dimension of `Xl`
	NbBlocksL	the number of diagonal blocks in the compact storage of L
	Kl	the list of starting row positions for each block of the storage of L
	Tl	the folding matrix for the compact storage of L: $Dl + SlTl$ is in column echelon form
	Ml	a permutation matrix such that $(Dl + TlSl)Ml$ is the L factor of the Bruhat generator
	B	an $N \times t$ dense matrix
	ldb	leading dimension of `B`
	beta	scaling constant
`in,out`	*C*	output matrix
	ldc	leading dimension of `C`

Bibliography Pernet C. and Storjohann A. *Time and space efficient generators for quasiseparable matrices* , JSC (85), 2018, doi:10.1016/j.jsc.2017.07.010

10.5.2.60 buildMatrix()

```
template<class Field >
Field::Element_ptr buildMatrix (
             const Field & F,
             typename Field::ConstElement_ptr E,
             typename Field::ConstElement_ptr C,
             const size_t lda,
             const size_t * B,
             const size_t * T,
             const size_t me,
             const size_t mc,
             const size_t lambda,
             const size_t mu )
```

Bug is this :

10.5.2.61 fsytrf_UP_RPM()

```
template<class Field >
size_t fsytrf_UP_RPM (
             const Field & Fi,
             const size_t N,
```

```
            typename Field::Element_ptr A,
            const size_t lda,
            typename Field::Element_ptr Dinv,
            const size_t incDinv,
            size_t * P,
            size_t BCThreshold )  [inline]
```

MathP <-[[I] x P1 |] [I_(N1+R2)] [P2^T] |] x [P3^T] [----------—|--—] [| Q2^T]

Changing [U1 V1 | E1 E21 E22] into [U1 E11 E12 V1 E∗ E∗] [0 | L2 \ U2 V21 V22] [U4 V41 0 V42 V43] [0 | M2 0 0] [U3 0 0 V3] [-—-—|---------——] [0 0 0] [0 | H1 H21 H22] [0 | U3 V3] [0 | 0] where U4 is the 2R2 x 2R2 matrix formed by interleaving U2, L2^T and H1

10.5.2.62 LUdivine() [2/2]

```
template<class Field >
size_t LUdivine (
            const Field & F,
            const FFLAS::FFLAS_DIAG Diag,
            const FFLAS::FFLAS_TRANSPOSE trans,
            const size_t M,
            const size_t N,
            typename Field::Element_ptr A,
            const size_t lda,
            size_t * P,
            size_t * Q,
            const FFPACK::FFPACK_LU_TAG LuTag,
            const size_t cutoff )  [inline]
```

Todo std::swap ?

10.5.2.63 composePermutationsLLL()

```
void composePermutationsLLL (
            size_t * P1,
            const size_t * P2,
            const size_t R,
            const size_t N )  [inline]
```

Computes P1 x Diag (I_R, P2) where P1 is a LAPACK and P2 a LAPACK permutation and store the result in P1 as a LAPACK permutation.

Parameters

in,out	*P1*	a LAPACK permutation of size N
	P2	a LAPACK permutation of size N-R

10.5.2.64 composePermutationsLLM()

```
void composePermutationsLLM (
```

```
            size_t * MathP,
            const size_t * P1,
            const size_t * P2,
            const size_t R,
            const size_t N )  [inline]
```

Computes P1 x Diag (I_R, P2) where P1 is a LAPACK and P2 a LAPACK permutation and store the result in MathP as a MathPermutation format.

Parameters

out		

a MathPermutation of size N

Parameters

P1	a LAPACK permutation of size N
P2	a LAPACK permutation of size N-R

10.5.2.65 composePermutationsMLM()

```
void composePermutationsMLM (
            size_t * MathP1,
            const size_t * P2,
            const size_t R,
            const size_t N )  [inline]
```

Computes MathP1 x Diag (I_R, P2) where MathP1 is a MathPermutation and P2 a LAPACK permutation and store the result in MathP1 as a MathPermutation format.

Parameters

in,out	*MathP1*	a MathPermutation of size N
	P2	a LAPACK permutation of size N-R

10.5.2.66 NonZeroRandomMatrix() [1/2]

```
template<class Field , class RandIter >
Field::Element_ptr NonZeroRandomMatrix (
            const Field & F,
            size_t m,
            size_t n,
            typename Field::Element_ptr A,
            size_t lda,
            RandIter & G )  [inline]
```

Random non-zero Matrix.

Creates a m x n matrix with random entries, and at least one of them is non zero.

Parameters

	F	field
	m	number of rows in A
	n	number of cols in A
out	*A*	the matrix (preallocated to at least m x lda field elements)
	lda	leading dimension of A
	G	a random iterator

Returns

A.

10.5.2.67 NonZeroRandomMatrix() [2/2]

```
template<class Field , class RandIter >
Field::Element_ptr NonZeroRandomMatrix (
            const Field & F,
            size_t m,
            size_t n,
            typename Field::Element_ptr A,
            size_t lda )  [inline]
```

Random non-zero Matrix.

Creates a m x n matrix with random entries, and at least one of them is non zero.

Parameters

	F	field
	m	number of rows in A
	n	number of cols in A
out	*A*	the matrix (preallocated to at least m x lda field elements)
	lda	leading dimension of A

Returns

A.

10.5.2.68 RandomMatrix() [1/2]

```
template<class Field , class RandIter >
Field::Element_ptr RandomMatrix (
            const Field & F,
            size_t m,
            size_t n,
            typename Field::Element_ptr A,
            size_t lda,
            RandIter & G )  [inline]
```

Random Matrix.

Creates a m x n matrix with random entries.

Parameters

	F	field
	m	number of rows in A
	n	number of cols in A
out	*A*	the matrix (preallocated to at least m x lda field elements)
	lda	leading dimension of A
	G	a random iterator

Returns

A.

10.5.2.69 RandomMatrix() [2/2]

```
template<class Field >
Field::Element_ptr RandomMatrix (
            const Field & F,
            size_t m,
            size_t n,
            typename Field::Element_ptr A,
            size_t lda )  [inline]
```

Random Matrix.

Creates a m x n matrix with random entries.

Parameters

	F	field
	m	number of rows in A
	n	number of cols in A
out	*A*	the matrix (preallocated to at least m x lda field elements)
	lda	leading dimension of A

Returns

A.

10.5.2.70 RandomTriangularMatrix() [1/2]

```
template<class Field , class RandIter >
Field::Element_ptr RandomTriangularMatrix (
            const Field & F,
            size_t m,
            size_t n,
            const FFLAS::FFLAS_UPLO UpLo,
            const FFLAS::FFLAS_DIAG Diag,
            bool nonsingular,
```

```
            typename Field::Element_ptr A,
            size_t lda,
            RandIter & G )  [inline]
```

Random Triangular Matrix.

Creates a m x n triangular matrix with random entries. The UpLo parameter defines wether it is upper or lower triangular.

Parameters

	F	field
	m	number of rows in A
	n	number of cols in A
	UpLo	whether A is upper or lower triangular
out	*A*	the matrix (preallocated to at least m x lda field elements)
	lda	leading dimension of A
	G	a random iterator

Returns

A.

10.5.2.71 RandomTriangularMatrix() [2/2]

```
template<class Field >
Field::Element_ptr RandomTriangularMatrix (
            const Field & F,
            size_t m,
            size_t n,
            const FFLAS::FFLAS_UPLO UpLo,
            const FFLAS::FFLAS_DIAG Diag,
            bool nonsingular,
            typename Field::Element_ptr A,
            size_t lda )  [inline]
```

Random Triangular Matrix.

Creates a m x n triangular matrix with random entries. The UpLo parameter defines wether it is upper or lower triangular.

Parameters

	F	field
	m	number of rows in A
	n	number of cols in A
	UpLo	whether A is upper or lower triangular
out	*A*	the matrix (preallocated to at least m x lda field elements)
	lda	leading dimension of A

Returns

A.

10.5.2.72 RandomSymmetricMatrix()

```
template<class Field , class RandIter >
Field::Element_ptr RandomSymmetricMatrix (
            const Field & F,
            size_t n,
            bool nonsingular,
            typename Field::Element_ptr A,
            size_t lda,
            RandIter & G )  [inline]
```

Random Symmetric Matrix.

Creates a m x n triangular matrix with random entries. The UpLo parameter defines wether it is upper or lower triangular.

Parameters

	F	field
	n	order of A
out	*A*	the matrix (preallocated to at least n x lda field elements)
	lda	leading dimension of A
	G	a random iterator

Returns

A.

10.5.2.73 RandomMatrixWithRank() [1/2]

```
template<class Field , class RandIter >
Field::Element_ptr RandomMatrixWithRank (
            const Field & F,
            size_t m,
            size_t n,
            size_t r,
            typename Field::Element_ptr A,
            size_t lda,
            RandIter & G )  [inline]
```

Random Matrix with prescribed rank.

Creates an m x n matrix with random entries and rank r.

Parameters

F	field
m	number of rows in A

Parameters

n	number of cols in A
r	rank of the matrix to build
A	the matrix (preallocated to at least m x lda field elements)
lda	leading dimension of A
G	a random iterator

Returns

A.

10.5.2.74 RandomMatrixWithRank() [2/2]

```
template<class Field >
Field::Element_ptr RandomMatrixWithRank (
            const Field & F,
            size_t m,
            size_t n,
            size_t r,
            typename Field::Element_ptr A,
            size_t lda )  [inline]
```

Random Matrix with prescribed rank.

Creates an m x n matrix with random entries and rank r.

Parameters

	F	field
	m	number of rows in A
	n	number of cols in A
	r	rank of the matrix to build
out	*A*	the matrix (preallocated to at least m x lda field elements)
	lda	leading dimension of A

Returns

A.

10.5.2.75 RandomIndexSubset()

```
size_t * RandomIndexSubset (
            size_t N,
            size_t R,
            size_t * P )  [inline]
```

Pick uniformly at random a sequence of R distinct elements from the set $\{0, \ldots, N-1\}$ using Knuth's shuffle.

Parameters

	N	the cardinality of the sampling set
	R	the number of elements to sample
out	*P*	the output sequence (pre-allocated to at least R indices)

10.5.2.76 RandomPermutation()

```
size_t * RandomPermutation (
            size_t N,
            size_t * P )  [inline]
```

Pick uniformly at random a permutation of size N stored in LAPACK format using Knuth's shuffle.

Parameters

	N	the length of the permutation
out	*P*	the output permutation (pre-allocated to at least N indices)

10.5.2.77 RandomRankProfileMatrix()

```
void RandomRankProfileMatrix (
            size_t M,
            size_t N,
            size_t R,
            size_t * rows,
            size_t * cols )  [inline]
```

Pick uniformly at random an R-subpermutation of dimension M x N : a matrix with only R non-zeros equal to one, in a random rook placement.

Parameters

	M	row dimension
	N	column dimension
out	*rows*	the row position of each non zero element (pre-allocated)
out	*cols*	the column position of each non zero element (pre-allocated)

10.5.2.78 RandomSymmetricRankProfileMatrix()

```
void RandomSymmetricRankProfileMatrix (
            size_t N,
            size_t R,
            size_t * rows,
            size_t * cols )  [inline]
```

Pick uniformly at random a symmetric R-subpermutation of dimension N x N : a symmetric matrix with only R non-zeros, all equal to one, in a random rook placement.

Parameters

	N	matrix order
out	*rows*	the row position of each non zero element (pre-allocated)
out	*cols*	the column position of each non zero element (pre-allocated)

10.5.2.79 RandomMatrixWithRankandRPM() [1/2]

```
template<class Field , class RandIter >
Field::Element_ptr RandomMatrixWithRankandRPM (
            const Field & F,
            size_t M,
            size_t N,
            size_t R,
            typename Field::Element_ptr A,
            size_t lda,
            const size_t * RRP,
            const size_t * CRP,
            RandIter & G )  [inline]
```

Random Matrix with prescribed rank and rank profile matrix Creates an m x n matrix with random entries and rank r.

Parameters

F	field
m	number of rows in A
n	number of cols in A
r	rank of the matrix to build
A	the matrix (preallocated to at least m x lda field elements)
lda	leading dimension of A
RRP	the R dimensional array with row positions of the rank profile matrix' pivots
CRP	the R dimensional array with column positions of the rank profile matrix' pivots
G	a random iterator

Returns

A.

10.5.2.80 RandomMatrixWithRankandRPM() [2/2]

```
template<class Field >
Field::Element_ptr RandomMatrixWithRankandRPM (
            const Field & F,
            size_t M,
            size_t N,
            size_t R,
            typename Field::Element_ptr A,
            size_t lda,
```

```
            const size_t * RRP,
            const size_t * CRP )  [inline]
```

Random Matrix with prescribed rank and rank profile matrix Creates an m x n matrix with random entries and rank r.

Parameters

F	field
m	number of rows in A
n	number of cols in A
r	rank of the matrix to build
A	the matrix (preallocated to at least m x lda field elements)
lda	leading dimension of A
RRP	the R dimensional array with row positions of the rank profile matrix' pivots
CRP	the R dimensional array with column positions of the rank profile matrix' pivots

Returns

A.

10.5.2.81 RandomSymmetricMatrixWithRankandRPM() [1/2]

```
template<class Field , class RandIter >
Field::Element_ptr RandomSymmetricMatrixWithRankandRPM (
            const Field & F,
            size_t N,
            size_t R,
            typename Field::Element_ptr A,
            size_t lda,
            const size_t * RRP,
            const size_t * CRP,
            RandIter & G )  [inline]
```

Random Symmetric Matrix with prescribed rank and rank profile matrix Creates an n x n symmetric matrix with random entries and rank r.

Parameters

F	field
n	order of A
r	rank of A
A	the matrix (preallocated to at least n x lda field elements)
lda	leading dimension of A
RRP	the R dimensional array with row positions of the rank profile matrix' pivots
CRP	the R dimensional array with column positions of the rank profile matrix' pivots
G	a random iterator

Returns

A.

10.5.2.82 RandomSymmetricMatrixWithRankandRPM() [2/2]

```
template<class Field >
Field::Element_ptr RandomSymmetricMatrixWithRankandRPM (
            const Field & F,
            size_t M,
            size_t N,
            size_t R,
            typename Field::Element_ptr A,
            size_t lda,
            const size_t * RRP,
            const size_t * CRP )  [inline]
```

Random Symmetric Matrix with prescribed rank and rank profile matrix Creates an n x n symmetric matrix with random entries and rank r.

Parameters

F	field
n	order of A
r	rank of A
A	the matrix (preallocated to at least n x lda field elements)
lda	leading dimension of A
RRP	the R dimensional array with row positions of the rank profile matrix' pivots
CRP	the R dimensional array with column positions of the rank profile matrix' pivots

Returns

A.

10.5.2.83 RandomMatrixWithRankandRandomRPM() [1/2]

```
template<class Field , class RandIter >
Field::Element_ptr RandomMatrixWithRankandRandomRPM (
            const Field & F,
            size_t M,
            size_t N,
            size_t R,
            typename Field::Element_ptr A,
            size_t lda,
            RandIter & G )  [inline]
```

Random Matrix with prescribed rank, with random rank profile matrix Creates an m x n matrix with random entries, rank r and with a rank profile matrix chosen uniformly at random.

Parameters

F	field
m	number of rows in A
n	number of cols in A
r	rank of the matrix to build
A	the matrix (preallocated to at least m x lda field elements)
lda	leading dimension of A
G	a random iterator

Returns

A.

10.5.2.84 RandomMatrixWithRankandRandomRPM() [2/2]

```
template<class Field >
Field::Element_ptr RandomMatrixWithRankandRandomRPM (
             const Field & F,
             size_t M,
             size_t N,
             size_t R,
             typename Field::Element_ptr A,
             size_t lda )  [inline]
```

Random Matrix with prescribed rank, with random rank profile matrix Creates an m x n matrix with random entries, rank r and with a rank profile matrix chosen uniformly at random.

Parameters

F	field
m	number of rows in A
n	number of cols in A
r	rank of the matrix to build
A	the matrix (preallocated to at least m x lda field elements)
lda	leading dimension of A

Returns

A.

10.5.2.85 RandomSymmetricMatrixWithRankandRandomRPM() [1/2]

```
template<class Field , class RandIter >
Field::Element_ptr RandomSymmetricMatrixWithRankandRandomRPM (
             const Field & F,
             size_t N,
             size_t R,
             typename Field::Element_ptr A,
             size_t lda,
             RandIter & G )  [inline]
```

Random Symmetric Matrix with prescribed rank, with random rank profile matrix Creates an n x n matrix with random entries, rank r and with a rank profile matrix chosen uniformly at random.

Parameters

F	field
n	order of A
r	rank of A
A	the matrix (preallocated to at least n x lda field elements)
lda	leading dimension of A
G	a random iterator

Returns

A.

10.5.2.86 RandomSymmetricMatrixWithRankandRandomRPM() [2/2]

```
template<class Field >
Field::Element_ptr RandomSymmetricMatrixWithRankandRandomRPM (
            const Field & F,
            size_t N,
            size_t R,
            typename Field::Element_ptr A,
            size_t lda )  [inline]
```

Random Symmetric Matrix with prescribed rank, with random rank profile matrix Creates an n x n matrix with random entries, rank r and with a rank profile matrix chosen uniformly at random.

Parameters

F	field
n	order of A
r	rank of A
A	the matrix (preallocated to at least n x lda field elements)
lda	leading dimension of A

Returns

A.

10.5.2.87 RandomMatrixWithDet() [1/2]

```
template<class Field >
Field::Element_ptr RandomMatrixWithDet (
            const Field & F,
            size_t n,
            const typename Field::Element d,
            typename Field::Element_ptr A,
            size_t lda )  [inline]
```

Random Matrix with prescribed det.

Creates a m x n matrix with random entries and rank r.

Parameters

F	field
d	the prescribed value for the determinant of A
n	number of cols in A
A	the matrix to be generated (preallocated to at least n x lda field elements)
lda	leading dimension of A

Returns

A.

10.5.2.88 RandomMatrixWithDet() [2/2]

```
template<class Field , class RandIter >
Field::Element_ptr RandomMatrixWithDet (
            const Field & F,
            size_t n,
            const typename Field::Element d,
            typename Field::Element_ptr A,
            size_t lda,
            RandIter & G )  [inline]
```

Random Matrix with prescribed det.

Creates a m x n matrix with random entries and rank r.

Parameters

F	field
d	the prescribed value for the determinant of A
n	number of cols in A
A	the matrix to be generated (preallocated to at least n x lda field elements)
lda	leading dimension of A

Returns

A.

Chapter 11

Data Structure Documentation

11.1 ArbitraryPrecIntTag Struct Reference

Arbitrary precision integers: GMP.

`#include <field-traits.h>`

11.1.1 Detailed Description

Arbitrary precision integers: GMP.

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

- field-traits.h

11.2 ConvertTo< T > Struct Template Reference

Force conversion to appropriate element type of ElementCategory T.

`#include <field-traits.h>`

11.2.1 Detailed Description

template<class T>
struct FFLAS::ModeCategories::ConvertTo< T >

Force conversion to appropriate element type of ElementCategory T.

e.g.

- ConvertTo<ElementCategories::MachineFloatTag> tries conversion of Modular<int> to Modular<double>
- ConvertTo<ElementCategories::FixedPrecIntTag> tries conversion of Modular<Integer> to Modular<Rec↩ Int<K> >
- ConvertTo<ElementCategories::ArbitraryPrecIntTag> tries conversion of Modular<Integer> to RNSInteger

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

- field-traits.h

11.3 DefaultBoundedTag Struct Reference

Use standard field operations, but keeps track of bounds on input and output.

`#include <field-traits.h>`

11.3.1 Detailed Description

Use standard field operations, but keeps track of bounds on input and output.

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

- field-traits.h

11.4 DefaultTag Struct Reference

No specific mode of action: use standard field operations.

`#include <field-traits.h>`

11.4.1 Detailed Description

No specific mode of action: use standard field operations.

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

- field-traits.h

11.5 DelayedTag Struct Reference

Performs field operations with delayed mod reductions. Ensures result is reduced.

`#include <field-traits.h>`

11.5.1 Detailed Description

Performs field operations with delayed mod reductions. Ensures result is reduced.

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

- field-traits.h

11.6 ElementTraits< Element > Struct Template Reference

ElementTraits.

`#include <field-traits.h>`

11.6.1 Detailed Description

template<class Element>
struct FFLAS::ElementTraits< Element >

ElementTraits.

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

- field-traits.h

11.7 Failure Class Reference

A precondtion failed.

`#include <debug.h>`

11.7.1 Detailed Description

A precondtion failed.

The `throw` mechanism is usually used here as in
```
if (!check)
failure()(__func__,__LINE__,"this check just failed");
```

The parameters of the constructor help debugging.

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

- debug.h

11.8 FieldTraits< Field > Struct Template Reference

FieldTrait.

`#include <field-traits.h>`

11.8.1 Detailed Description

template<class Field>
struct FFLAS::FieldTraits< Field >

FieldTrait.

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

- field-traits.h

11.9 FixedPrecIntTag Struct Reference

Fixed precision integers above machine precision: Givaro::recInt.

`#include <field-traits.h>`

11.9.1 Detailed Description

Fixed precision integers above machine precision: Givaro::recInt.

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

- field-traits.h

11.10 ftrsmLeftUpperNoTransNonUnit< Element > Class Template Reference

Computes the maximal size for delaying the modular reduction in a triangular system resolution.

11.10.1 Detailed Description

template<class Element>
class FFLAS::Protected::ftrsmLeftUpperNoTransNonUnit< Element >

Computes the maximal size for delaying the modular reduction in a triangular system resolution.

Compute the maximal dimension k, such that a unit diagonal triangular system of dimension k can be solved over Z without overflow of the underlying floating point representation.

Bibliography

• Dumas, Giorgi, Pernet 06, arXiv:cs/0601133.

Parameters

F	Finite Field/Ring of the computation

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

- fflas_level3.inl

11.11 GenericTag Struct Reference

default is generic

`#include <field-traits.h>`

11.11.1 Detailed Description

default is generic

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

- field-traits.h

11.12 GenericTag Struct Reference

generic ring.

`#include <field-traits.h>`

11.12.1 Detailed Description

generic ring.

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

- field-traits.h

11.13 LazyTag Struct Reference

Performs field operations with delayed mod only when necessary. Result may not be reduced.

`#include <field-traits.h>`

11.13.1 Detailed Description

Performs field operations with delayed mod only when necessary. Result may not be reduced.

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

- field-traits.h

11.14 MachineFloatTag Struct Reference

float or double

`#include <field-traits.h>`

11.14.1 Detailed Description

float or double

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

- field-traits.h

11.15 MachineIntTag Struct Reference

short, int, long, long long, and unsigned variants

`#include <field-traits.h>`

11.15.1 Detailed Description

short, int, long, long long, and unsigned variants

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

- field-traits.h

11.16 MMHelper< Field, AlgoTrait, ModeCategories::DefaultTag, ParSeqTrait > Struct Template Reference

FGEMM Helper for Default and ConvertTo modes of operation.

11.16.1 Detailed Description

template<class Field, typename AlgoTrait, typename ParSeqTrait>
struct FFLAS::MMHelper< Field, AlgoTrait, ModeCategories::DefaultTag, ParSeqTrait >

FGEMM Helper for Default and ConvertTo modes of operation.

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

- fflas_helpers.inl

11.17 ModeTraits< Field > Struct Template Reference

ModeTraits.

`#include <field-traits.h>`

11.17.1 Detailed Description

template<class Field>
struct FFLAS::ModeTraits< Field >

ModeTraits.

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

- field-traits.h

11.18 ModularTag Struct Reference

This is a modular field like e.g. `Modular<T>` or `ModularBalanced<T>`

`#include <field-traits.h>`

11.18.1 Detailed Description

This is a modular field like e.g. `Modular<T>` or `ModularBalanced<T>`

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

- field-traits.h

11.19 RNSElementTag Struct Reference

Representation in a Residue Number System.

`#include <field-traits.h>`

11.19.1 Detailed Description

Representation in a Residue Number System.

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

- field-traits.h

11.20 TRSMHelper< RecIterTrait, ParSeqTrait > Struct Template Reference

TRSM Helper.

11.20.1 Detailed Description

template<typename RecIterTrait = StructureHelper::Recursive, typename ParSeqTrait = ParSeqHelper::↩ Sequential>
struct FFLAS::TRSMHelper< RecIterTrait, ParSeqTrait >

TRSM Helper.

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

- fflas_helpers.inl

11.21 UnparametricTag Struct Reference

If the field uses a representation with infix operators.

`#include <field-traits.h>`

11.21.1 Detailed Description

If the field uses a representation with infix operators.

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

- field-traits.h

Chapter 12

File Documentation

12.1 fflas-ffpack-config.h File Reference

Defaults for optimised values.

```
#include "fflas-ffpack/config.h"
#include "fflas-ffpack/fflas-ffpack-thresholds.h"
#include "fflas-ffpack/fflas-ffpack-default-thresholds.h"
#include "givaro/givconfig.h"
```

12.1.1 Detailed Description

Defaults for optimised values.

While `fflas-ffpack-optimise.h` is created by `configure` script, (either left blank or filled by optimiser), this file produces the defaults for the optimised values. If `fflas-ffpack-optimise.h` is not empty, then its values preceeds the defaults here.

12.2 fflas-ffpack.h File Reference

Includes FFLAS and FFPACK.

```
#include "fflas-ffpack/fflas-ffpack-config.h"
#include "fflas/fflas.h"
```

12.2.1 Detailed Description

Includes FFLAS and FFPACK.

12.3 fflas.h File Reference

Finite **F**ield **L**inear **A**lgebra **S**ubroutines

```
#include "fflas-ffpack/fflas-ffpack-config.h"
#include "fflas-ffpack/config.h"
#include "fflas-ffpack/config-blas.h"
#include <cmath>
#include <cstring>
#include <float.h>
#include <algorithm>
#include "fflas_enum.h"
#include "fflas-ffpack/utils/fflas_memory.h"
#include "fflas-ffpack/paladin/parallel.h"
#include "fflas_level1.inl"
#include "fflas_level2.inl"
#include "fflas_level3.inl"
#include "fflas-ffpack/checkers/checkers_fflas.h"
#include "fflas_freduce.h"
#include "fflas_fadd.h"
#include "fflas_fscal.h"
#include "fflas_fassign.h"
#include "fflas_fgemm.inl"
#include "fflas_pfgemm.inl"
#include "fflas_fgemv.inl"
#include "fflas-ffpack/paladin/pfgemv.inl"
#include "fflas_freivalds.inl"
#include "fflas_fger.inl"
#include "fflas_fsyrk.inl"
#include "fflas_fsyrk_strassen.inl"
#include "fflas_fsyr2k.inl"
#include "fflas_ftrsm.inl"
#include "fflas_pftrsm.inl"
#include "fflas_ftrmm.inl"
#include "fflas_ftrsv.inl"
#include "fflas_faxpy.inl"
#include "fflas_fdot.inl"
#include "fflas-ffpack/field/rns.h"
#include "fflas_fscal_mp.inl"
#include "fflas_freduce_mp.inl"
#include "fflas-ffpack/fflas/fflas_fger_mp.inl"
#include "fflas_fgemm/fgemm_classical_mp.inl"
#include "fflas_ftrsm_mp.inl"
#include "fflas_fgemv_mp.inl"
#include "fflas-ffpack/field/rns.inl"
#include "fflas-ffpack/paladin/fflas_plevel1.h"
#include "fflas_sparse.h"
#include "fflas-ffpack/checkers/checkers_fflas.inl"
```

Macros

- #define DOUBLE_TO_FLOAT_CROSSOVER 800

  Thresholds determining which floating point representation to use, depending on the cardinality of the finite field.

12.3.1 Detailed Description

Finite **F**ield **L**inear **A**lgebra **S**ubroutines

Author

Clément Pernet.

12.3.2 Macro Definition Documentation

12.3.2.1 DOUBLE_TO_FLOAT_CROSSOVER

```
#define DOUBLE_TO_FLOAT_CROSSOVER 800
```

Thresholds determining which floating point representation to use, depending on the cardinality of the finite field.

This is only used when the element representation is not a floating point type.

Bug to be benchmarked.

12.4 fgemm_classical_mp.inl File Reference

matrix multiplication with multiprecision input (either over Z or over Z/pZ)

```
#include <givaro/modular-integer.h>
#include <givaro/zring.h>
#include "fflas-ffpack/field/rns-double.h"
#include "fflas-ffpack/field/rns-integer.h"
#include "fflas-ffpack/field/rns-integer-mod.h"
#include "fflas-ffpack/field/field-traits.h"
#include "fflas-ffpack/fflas/fflas_helpers.inl"
#include "fflas-ffpack/fflas/fflas_bounds.inl"
```

12.4.1 Detailed Description

matrix multiplication with multiprecision input (either over Z or over Z/pZ)

12.5 schedule_bini.inl File Reference

Bini implementation.

12.5.1 Detailed Description

Bini implementation.

12.6 fflas_ftrsm_mp.inl File Reference

triangular system with matrix right hand side over multiprecision domain (either over Z or over Z/pZ)

```
#include <cmath>
#include <givaro/modular-integer.h>
#include <givaro/givinteger.h>
#include "fflas-ffpack/fflas/fflas_bounds.inl"
#include "fflas-ffpack/fflas/fflas_level3.inl"
#include "fflas-ffpack/field/rns-integer-mod.h"
#include "fflas-ffpack/field/rns-integer.h"
```

12.6.1 Detailed Description

triangular system with matrix right hand side over multiprecision domain (either over Z or over Z/pZ)

12.7 fflas_sparse.h File Reference

```
#include "fflas-ffpack/fflas-ffpack-config.h"
#include "fflas-ffpack/config.h"
#include "fflas-ffpack/config-blas.h"
#include "fflas-ffpack/paladin/parallel.h"
#include <recint/recint.h>
#include <givaro/udl.h>
#include "fflas-ffpack/fflas/fflas.h"
#include "fflas-ffpack/field/field-traits.h"
#include "fflas-ffpack/fflas/fflas_bounds.inl"
#include "fflas-ffpack/utils/fflas_memory.h"
#include <type_traits>
#include <vector>
#include <iostream>
#include "fflas-ffpack/fflas/fflas_sparse/sparse_matrix_traits.h"
#include "fflas-ffpack/fflas/fflas_sparse/utils.h"
#include "fflas-ffpack/fflas/fflas_sparse/csr.h"
#include "fflas-ffpack/fflas/fflas_sparse/coo.h"
#include "fflas-ffpack/fflas/fflas_sparse/ell.h"
#include "fflas-ffpack/fflas/fflas_sparse/sell.h"
#include "fflas-ffpack/fflas/fflas_sparse/csr_hyb.h"
#include "fflas-ffpack/fflas/fflas_sparse/ell_simd.h"
#include "fflas-ffpack/fflas/fflas_sparse/hyb_zo.h"
#include "fflas-ffpack/fflas/fflas_sparse.inl"
#include "fflas-ffpack/fflas/fflas_sparse/read_sparse.h"
```

12.8 fflas_sparse.inl File Reference

12.9 read_sparse.h File Reference

```
#include "fflas-ffpack/fflas-ffpack-config.h"
#include <fstream>
```

```
#include <string>
#include <cstdlib>
#include <iterator>
```

12.10 fflas_transpose.h File Reference

transpose the storage of the matrix (switch between row and col major mode)

```
#include "fflas-ffpack/utils/debug.h"
#include "fflas-ffpack/fflas/fflas.h"
#include "fflas-ffpack/utils/fflas_memory.h"
#include "fflas-ffpack/fflas/fflas_simd.h"
```

12.10.1 Detailed Description

transpose the storage of the matrix (switch between row and col major mode)

12.11 ffpack.h File Reference

Set of elimination based routines for dense linear algebra.

```
#include "givaro/givpoly1.h"
#include <fflas-ffpack/fflas-ffpack-config.h>
#include "fflas-ffpack/fflas/fflas.h"
#include "fflas-ffpack/fflas/fflas_helpers.inl"
#include <list>
#include <vector>
#include <iostream>
#include <algorithm>
#include "fflas-ffpack/checkers/checkers_ffpack.h"
#include "ffpack_fgesv.inl"
#include "ffpack_fgetrs.inl"
#include "fflas-ffpack/checkers/checkers_ffpack.inl"
#include "ffpack_pluq.inl"
#include "ffpack_pluq_mp.inl"
#include "ffpack_ppluq.inl"
#include "ffpack_ludivine.inl"
#include "ffpack_ludivine_mp.inl"
#include "ffpack_echelonforms.inl"
#include "ffpack_fsytrf.inl"
#include "ffpack_invert.inl"
#include "ffpack_ftrtr.inl"
#include "ffpack_ftrstr.inl"
#include "ffpack_ftrssyr2k.inl"
#include "ffpack_charpoly_kglu.inl"
#include "ffpack_charpoly_kgfast.inl"
#include "ffpack_charpoly_kgfastgeneralized.inl"
#include "ffpack_charpoly_danilevski.inl"
#include "ffpack_charpoly.inl"
```

```
#include "ffpack_frobenius.inl"
#include "ffpack_minpoly.inl"
#include "ffpack_krylovelim.inl"
#include "ffpack_permutation.inl"
#include "ffpack_rankprofiles.inl"
#include "ffpack_det_mp.inl"
#include "ffpack_bruhatgen.inl"
#include "ffpack.inl"
```

Namespaces

- namespace FFPACK

  ***F**inite **F**ield **PACK** Set of elimination based routines for dense linear algebra.*

Functions

- void **LAPACKPerm2MathPerm** (size_t *MathP, const size_t *LapackP, const size_t N)

  Conversion of a permutation from LAPACK format to Math format.
- void **MathPerm2LAPACKPerm** (size_t *LapackP, const size_t *MathP, const size_t N)

  Conversion of a permutation from Maths format to LAPACK format.
- template<class Field >
  void applyP (const Field &F, const FFLAS::FFLAS_SIDE Side, const FFLAS::FFLAS_TRANSPOSE Trans, const size_t M, const size_t ibeg, const size_t iend, typename Field::Element_ptr A, const size_t lda, const size_t *P)

  Computes P1 x Diag (I_R, P2) where P1 is a LAPACK and P2 a LAPACK permutation and store the result in P1 as a LAPACK permutation.
- template<class Field >
  void MonotonicApplyP (const Field &F, const FFLAS::FFLAS_SIDE Side, const FFLAS::FFLAS_↩ TRANSPOSE Trans, const size_t M, const size_t ibeg, const size_t iend, typename Field::Element_ptr A, const size_t lda, const size_t *P, const size_t R)

  Apply a R-monotonically increasing permutation P, to the matrix A.
- template<class Field >
  void fgetrs (const Field &F, const FFLAS::FFLAS_SIDE Side, const size_t M, const size_t N, const size_t R, typename Field::Element_ptr A, const size_t lda, const size_t *P, const size_t *Q, typename Field::Element↩ _ptr B, const size_t ldb, int *info)

  Solve the system $AX = B$ or $XA = B$.
- template<class Field >
  Field::Element_ptr fgetrs (const Field &F, const FFLAS::FFLAS_SIDE Side, const size_t M, const size↩ _t N, const size_t NRHS, const size_t R, typename Field::Element_ptr A, const size_t lda, const size_t *P, const size_t *Q, typename Field::Element_ptr X, const size_t ldx, typename Field::ConstElement_ptr B, const size_t ldb, int *info)

  Solve the system A X = B or X A = B.
- template<class Field >
  size_t fgesv (const Field &F, const FFLAS::FFLAS_SIDE Side, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr B, const size_t ldb, int *info)

  Square system solver.
- template<class Field >
  size_t fgesv (const Field &F, const FFLAS::FFLAS_SIDE Side, const size_t M, const size_t N, const size_t NRHS, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr X, const size_t ldx, typename Field::ConstElement_ptr B, const size_t ldb, int *info)

  Rectangular system solver.

- template<class Field >
  void ftrtri (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG Diag, const size_t N, typename Field::Element_ptr A, const size_t lda, const size_t threshold=__FFLASFFPACK_FTRTRI_↩ THRESHOLD)

  Compute the inverse of a triangular matrix.
- template<class Field >
  void ftrtrm (const Field &F, const FFLAS::FFLAS_SIDE side, const FFLAS::FFLAS_DIAG diag, const size_t N, typename Field::Element_ptr A, const size_t lda)

  Compute the product of two triangular matrices of opposite shape.
- template<class Field >
  void ftrstr (const Field &F, const FFLAS::FFLAS_SIDE side, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diagA, const FFLAS::FFLAS_DIAG diagB, const size_t N, typename Field::Const↩ Element_ptr A, const size_t lda, typename Field::Element_ptr B, const size_t ldb, const size_t threshold=64)

  Solve a triangular system with a triangular right hand side of the same shape.
- template<class Field >
  void ftrssyr2k (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diagA, const size_t N, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr B, const size↩ _t ldb, const size_t threshold=64)

  Solve a triangular system in a symmetric sum: find B upper/lower triangular such that A^ T B + B^ T A = C where C is symmetric.
- template<class Field >
  bool fsytrf (const Field &F, const FFLAS::FFLAS_UPLO UpLo, const size_t N, typename Field::Element_ptr A, const size_t lda, const size_t threshold=__FFLASFFPACK_FSYTRF_THRESHOLD)

  Triangular factorization of symmetric matrices.
- template<class Field >
  bool fsytrf_nonunit (const Field &F, const FFLAS::FFLAS_UPLO UpLo, const size_t N, typename Field::↩ Element_ptr A, const size_t lda, typename Field::Element_ptr D, const size_t incD, const size_t threshold=↩ __FFLASFFPACK_FSYTRF_THRESHOLD)

  Triangular factorization of symmetric matrices.
- template<class Field >
  size_t PLUQ (const Field &F, const FFLAS::FFLAS_DIAG Diag, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t ∗P, size_t ∗Q)

  Compute a PLUQ factorization of the given matrix.
- template<class Field >
  size_t LUdivine (const Field &F, const FFLAS::FFLAS_DIAG Diag, const FFLAS::FFLAS_TRANSPOSE trans, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t ∗P, size_t ∗Qt, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive, const size_t cutoff=__FFLASFFPACK_LUDIVINE↩ _THRESHOLD)

  Compute the CUP or PLE factorization of the given matrix.
- template<class Field >
  size_t ColumnEchelonForm (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t ∗P, size_t ∗Qt, bool transform=false, const FFPACK_LU_TAG LuTag=FfpackSlab↩ Recursive)

  Compute the Column Echelon form of the input matrix in-place.
- template<class Field >
  size_t RowEchelonForm (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t ∗P, size_t ∗Qt, const bool transform=false, const FFPACK_LU_TAG LuTag=Ffpack↩ SlabRecursive)

  Compute the Row Echelon form of the input matrix in-place.
- template<class Field >
  size_t ReducedColumnEchelonForm (const Field &F, const size_t M, const size_t N, typename Field::↩ Element_ptr A, const size_t lda, size_t ∗P, size_t ∗Qt, const bool transform=false, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Compute the Reduced Column Echelon form of the input matrix in-place.

- template<class Field >
  size_t ReducedRowEchelonForm (const Field &F, const size_t M, const size_t N, typename Field::Element↩
  _ptr A, const size_t lda, size_t *P, size_t *Qt, const bool transform=false, const FFPACK_LU_TAG Lu↩
  Tag=FfpackSlabRecursive)

  Compute the Reduced Row Echelon form of the input matrix in-place.

- template<class Field >
  size_t GaussJordan (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, const size_t colbeg, const size_t rowbeg, const size_t colsize, size_t *P, size_t *Q, const FFPACK::FFPACK_LU_TAG LuTag)

  Gauss-Jordan algorithm computing the Reduced Row echelon form and its transform matrix.

- template<class Field >
  Field::Element_ptr Invert (const Field &F, const size_t M, typename Field::Element_ptr A, const size_t lda, int &nullity)

  Invert the given matrix in place or computes its nullity if it is singular.

- template<class Field >
  Field::Element_ptr Invert (const Field &F, const size_t M, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr X, const size_t ldx, int &nullity)

  Invert the given matrix or computes its nullity if it is singular.

- template<class Field >
  Field::Element_ptr Invert2 (const Field &F, const size_t M, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr X, const size_t ldx, int &nullity)

  Invert the given matrix or computes its nullity if it is singular.

- template<class PolRing >
  std::list< typename PolRing::Element > & CharPoly (const PolRing &R, std::list< typename PolRing::↩
  Element > &charp, const size_t N, typename PolRing::Domain_t::Element_ptr A, const size_t lda, typename PolRing::Domain_t::RandIter &G, const FFPACK_CHARPOLY_TAG CharpTag=FfpackAuto, const size_↩
  t degree=__FFLASFFPACK_ARITHPROG_THRESHOLD)

  Compute the characteristic polynomial of the matrix A.

- template<class PolRing >
  PolRing::Element & CharPoly (const PolRing &R, typename PolRing::Element &charp, const size_t N, typename PolRing::Domain_t::Element_ptr A, const size_t lda, typename PolRing::Domain_t::RandIter &G, const FFPACK_CHARPOLY_TAG CharpTag=FfpackAuto, const size_t degree=__FFLASFFPACK_ARITHPROG↩
  _THRESHOLD)

  Compute the characteristic polynomial of the matrix A.

- template<class PolRing >
  PolRing::Element & CharPoly (const PolRing &R, typename PolRing::Element &charp, const size_t N, typename PolRing::Domain_t::Element_ptr A, const size_t lda, const FFPACK_CHARPOLY_TAG Charp↩
  Tag=FfpackAuto, const size_t degree=__FFLASFFPACK_ARITHPROG_THRESHOLD)

  Compute the characteristic polynomial of the matrix A.

- template<class PolRing >
  void RandomKrylovPrecond (const PolRing &PR, std::list< typename PolRing::Element > &completed↩
  Factors, const size_t N, typename PolRing::Domain_t::Element_ptr A, const size_t lda, size_t &Nb, typename PolRing::Domain_t::Element_ptr &B, size_t &ldb, typename PolRing::Domain_t::RandIter &g, const size_↩
  t degree=__FFLASFFPACK_ARITHPROG_THRESHOLD)

- template<class Field , class Polynomial >
  Polynomial & MinPoly (const Field &F, Polynomial &minP, const size_t N, typename Field::ConstElement_ptr A, const size_t lda)

  Compute the minimal polynomial of the matrix A.

- template<class Field , class Polynomial , class RandIter >
  Polynomial & MinPoly (const Field &F, Polynomial &minP, const size_t N, typename Field::ConstElement_ptr A, const size_t lda, RandIter &G)

  Compute the minimal polynomial of the matrix A.

- template<class Field , class Polynomial >
  Polynomial & MatVecMinPoly (const Field &F, Polynomial &minP, const size_t N, typename Field::Const↩
  Element_ptr A, const size_t lda, typename Field::ConstElement_ptr v, const size_t incv)

Compute the minimal polynomial of the matrix A and a vector v, namely the first linear dependency relation in the Krylov basis $(v, Av, ..., A^N v)$.

- template<class Field >
  size_t Rank (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda)

  Computes the rank of the given matrix using a PLUQ factorization.
- template<class Field >
  bool IsSingular (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda)

  Returns true if the given matrix is singular.
- template<class Field >
  Field::Element & Det (const Field &F, typename Field::Element &det, const size_t N, typename Field::↩ Element_ptr A, const size_t lda, size_t ∗P=NULL, size_t ∗Q=NULL)

  Returns the determinant of the given square matrix.
- template<class Field >
  Field::Element_ptr Solve (const Field &F, const size_t M, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr x, const int incx, typename Field::ConstElement_ptr b, const int incb)

  Solves a linear system AX = b using PLUQ factorization.
- template<class Field >
  ∗void RandomNullSpaceVector (const Field &F, const FFLAS::FFLAS_SIDE Side, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr X, const size_t incX)

  Solve L X = B or X L = B in place.
- template<class Field >
  size_t NullSpaceBasis (const Field &F, const FFLAS::FFLAS_SIDE Side, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr &NS, size_t &ldn, size_↩ t &NSdim)

  Computes a basis of the Left/Right nullspace of the matrix A.
- template<class Field >
  size_t RowRankProfile (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t ∗&rkprofile, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Computes the row rank profile of A.
- template<class Field >
  size_t ColumnRankProfile (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, size_t ∗&rkprofile, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Computes the column rank profile of A.
- void RankProfileFromLU (const size_t ∗P, const size_t N, const size_t R, size_t ∗rkprofile, const FFPACK_↩ LU_TAG LuTag)

  Recovers the column/row rank profile from the permutation of an LU decomposition.
- size_t LeadingSubmatrixRankProfiles (const size_t M, const size_t N, const size_t R, const size_t LSm, const size_t LSn, const size_t ∗P, const size_t ∗Q, size_t ∗RRP, size_t ∗CRP)

  Recovers the row and column rank profiles of any leading submatrix from the PLUQ decomposition.
- template<class Field >
  size_t RowRankProfileSubmatrixIndices (const Field &F, const size_t M, const size_t N, typename Field::↩ Element_ptr A, const size_t lda, size_t ∗&rowindices, size_t ∗&colindices, size_t &R)

  RowRankProfileSubmatrixIndices.
- template<class Field >
  size_t ColRankProfileSubmatrixIndices (const Field &F, const size_t M, const size_t N, typename Field::↩ Element_ptr A, const size_t lda, size_t ∗&rowindices, size_t ∗&colindices, size_t &R)

  Computes the indices of the submatrix r∗r X of A whose columns correspond to the column rank profile of A.
- template<class Field >
  size_t RowRankProfileSubmatrix (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr &X, size_t &R)

  Computes the r∗r submatrix X of A, by picking the row rank profile rows of A.
- template<class Field >
  size_t ColRankProfileSubmatrix (const Field &F, const size_t M, const size_t N, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr &X, size_t &R)

Compute the $r \times r$ submatrix X of A, by picking the row rank profile rows of A.

- template<class Field >
  void getTriangular (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diag, const size_t M, const size_t N, const size_t R, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr T, const size_t ldt, const bool OnlyNonZeroVectors=false)

  Extracts a triangular matrix from a compact storage A=L\U of rank R.

- template<class Field >
  void getTriangular (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diag, const size_t M, const size_t N, const size_t R, typename Field::Element_ptr A, const size_t lda)

  Cleans up a compact storage A=L\U to reveal a triangular matrix of rank R.

- template<class Field >
  void getEchelonForm (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diag, const size_t M, const size_t N, const size_t R, const size_t *P, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr T, const size_t ldt, const bool OnlyNonZeroVectors=false, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Extracts a matrix in echelon form from a compact storage A=L\U of rank R obtained by RowEchelonForm or Column↩ EchelonForm.

- template<class Field >
  void getEchelonForm (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diag, const size_t M, const size_t N, const size_t R, const size_t *P, typename Field::Element_ptr A, const size_t lda, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Cleans up a compact storage A=L\U obtained by RowEchelonForm or ColumnEchelonForm to reveal an echelon form of rank R.

- template<class Field >
  void getEchelonTransform (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diag, const size_t M, const size_t N, const size_t R, const size_t *P, const size_t *Q, typename Field::↩ ConstElement_ptr A, const size_t lda, typename Field::Element_ptr T, const size_t ldt, const FFPACK_LU↩ _TAG LuTag=FfpackSlabRecursive)

  Extracts a transformation matrix to echelon form from a compact storage A=L\U of rank R obtained by RowEchelon↩ Form or ColumnEchelonForm.

- template<class Field >
  void getReducedEchelonForm (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const size_t M, const size_t N, const size_t R, const size_t *P, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr T, const size_t ldt, const bool OnlyNonZeroVectors=false, const FFPACK_LU_TAG Lu↩ Tag=FfpackSlabRecursive)

  Extracts a matrix in echelon form from a compact storage A=L\U of rank R obtained by ReducedRowEchelonForm or ReducedColumnEchelonForm with transform = true.

- template<class Field >
  void getReducedEchelonForm (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const size_t M, const size_t N, const size_t R, const size_t *P, typename Field::Element_ptr A, const size_t lda, const FFPACK_↩ LU_TAG LuTag=FfpackSlabRecursive)

  Cleans up a compact storage A=L\U of rank R obtained by ReducedRowEchelonForm or ReducedColumnEchelon↩ Form with transform = true.

- template<class Field >
  void getReducedEchelonTransform (const Field &F, const FFLAS::FFLAS_UPLO Uplo, const size_t M, const size_t N, const size_t R, const size_t *P, const size_t *Q, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr T, const size_t ldt, const FFPACK_LU_TAG LuTag=FfpackSlabRecursive)

  Extracts a transformation matrix to echelon form from a compact storage A=L\U of rank R obtained by RowEchelon↩ Form or ColumnEchelonForm.

- void **PLUQtoEchelonPermutation** (const size_t N, const size_t R, const size_t *P, size_t *outPerm)

  Auxiliary routine: determines the permutation that changes a PLUQ decomposition into a echelon form revealing PLUQ decomposition.

- template<class Field >
  size_t LTBruhatGen (const Field &Fi, const FFLAS::FFLAS_DIAG diag, const size_t N, typename Field::↩ Element_ptr A, const size_t lda, size_t *P, size_t *Q)

LTBruhatGen Suppose A is Left Triangular Matrix This procedure computes the Bruhat Representation of A and return the rank of A.

- template<class Field >
  void getLTBruhatGen (const Field &Fi, const size_t N, const size_t r, const size_t *P, const size_t *Q, typename Field::Element_ptr R, const size_t ldr)

  GetLTBruhatGen This procedure Computes the Rank Revealing Matrix based on the Bruhta representation of a Matrix.
- template<class Field >
  void getLTBruhatGen (const Field &Fi, const FFLAS::FFLAS_UPLO Uplo, const FFLAS::FFLAS_DIAG diag, const size_t N, const size_t r, const size_t *P, const size_t *Q, typename Field::ConstElement_ptr A, const size_t lda, typename Field::Element_ptr T, const size_t ldt)

  GetLTBruhatGen This procedure computes the matrix L or U f the Bruhat Representation Suppose that A is the bruhat representation of a matrix.
- size_t LTQSorder (const size_t N, const size_t r, const size_t *P, const size_t *Q)

  LTQSorder This procedure computes the order of quasiseparability of a matrix.
- template<class Field >
  size_t CompressToBlockBiDiagonal (const Field &Fi, const FFLAS::FFLAS_UPLO Uplo, size_t N, size_t s, size_t r, const size_t *P, const size_t *Q, typename Field::Element_ptr A, size_t lda, typename Field::↩ Element_ptr X, size_t ldx, size_t *K, size_t *M, size_t *T)

  CompressToBlockBiDiagonal This procedure compress a compact representation of a row echelon form or column echelon form.
- template<class Field >
  void ExpandBlockBiDiagonalToBruhat (const Field &Fi, const FFLAS::FFLAS_UPLO Uplo, size_t N, size↩ _t s, size_t r, typename Field::Element_ptr A, size_t lda, typename Field::Element_ptr X, size_t ldx, size_t NbBlocks, size_t *K, size_t *M, size_t *T)

  ExpandBlockBiDiagonal This procedure expand a compact representation of a row echelon form or column echelon form.
- void Bruhat2EchelonPermutation (size_t N, size_t R, const size_t *P, const size_t *Q, size_t *M)

  Bruhat2EchelonPermutation (N,R,P,Q) Compute M such that LM or MU is in echelon form where L or U are factors of the Bruhat Rpresentation.
- template<class Field >
  void **productBruhatxTS** (const Field &Fi, size_t N, size_t s, size_t r, const size_t *P, const size_t *Q, const typename Field::Element_ptr Xu, size_t ldu, size_t NbBlocksU, size_t *Ku, size_t *Tu, size_t *MU, const typename Field::Element_ptr Xl, size_t ldl, size_t NbBlocksL, size_t *Kl, size_t *Tl, size_t *ML, typename Field::Element_ptr B, size_t t, size_t ldb, typename Field::Element_ptr C, size_t ldc)

  productBruhatxTS Comput the product between the CRE compact representation of a matrix A and B a tall matrix
- template<class Field >
  Field::Element_ptr LQUPtoInverseOfFullRankMinor (const Field &F, const size_t rank, typename Field::↩ Element_ptr A_factors, const size_t lda, const size_t *QtPointer, typename Field::Element_ptr X, const size↩ _t ldx)

  LQUPtoInverseOfFullRankMinor.

12.11.1 Detailed Description

Set of elimination based routines for dense linear algebra.

Matrices are supposed over finite prime field of characteristic less than 2^26.

12.11.2 Function Documentation

12.11.2.1 GaussJordan()

```
template<class Field >
size_t GaussJordan (
             const Field & F,
             const size_t M,
             const size_t N,
             typename Field::Element_ptr A,
             const size_t lda,
             const size_t colbeg,
             const size_t rowbeg,
             const size_t colsize,
             size_t * P,
             size_t * Q,
             const FFPACK::FFPACK_LU_TAG LuTag )  [inline]
```

Gauss-Jordan algorithm computing the Reduced Row echelon form and its transform matrix.

Bibliography

- Algorithm 2.8 of A. Storjohann Thesis 2000,
- Algorithm 11 of Jeannerod C-P., Pernet, C. and Storjohann, A. *Rank-profile revealing Gaussian elimination and the CUP matrix decomposition* , J. of Symbolic Comp., 2013

Parameters

	M	row dimension of A
	N	column dimension of A
in,out	*A*	an m x n matrix
	lda	leading dimension of A
	P	row permutation
	Q	column permutation
	LuTag	set the base case to a Tile (FfpackGaussJordanTile) or Slab (FfpackGaussJordanSlab) recursive RedEchelon

| I | A11 | A12 | | |—|—|—|—| | | I | * | A22 | | | |0 | 0| A22 | | |—|—|—|—| | | 0 | A32 | | |—|—|—|—|
where the transformation matrix is stored at the pivot column position

12.11.2.2 RandomKrylovPrecond()

```
template<class PolRing >
void RandomKrylovPrecond (
             const PolRing & PR,
             std::list< typename PolRing::Element > & completedFactors,
             const size_t N,
             typename PolRing::Domain_t::Element_ptr A,
             const size_t lda,
             size_t & Nb,
             typename PolRing::Domain_t::Element_ptr & B,
             size_t & ldb,
             typename PolRing::Domain_t::RandIter & g,
             const size_t degree = __FFLASFFPACK_ARITHPROG_THRESHOLD )  [inline]
```

Todo swap to save space ??

Todo

Todo don't assing K2 c∗noc x N but only mas (c,noc) x N and store each one after the other

Todo swap to save space ??

Todo

Todo don't assing K2 c∗noc x N but only mas (c,noc) x N and store each one after the other

12.12 field-traits.h File Reference

Field Traits.

```
#include <type_traits>
#include "fflas-ffpack/field/rns-double-elt.h"
#include "recint/rmint.h"
#include "givaro/modular-general.h"
#include "givaro/zring.h"
```

Data Structures

- struct GenericTag

  generic ring.
- struct ModularTag

  This is a modular field like e.g. `Modular<T>` *or* `ModularBalanced<T>`
- struct UnparametricTag

  If the field uses a representation with infix operators.
- struct DefaultTag

  No specific mode of action: use standard field operations.
- struct DefaultBoundedTag

  Use standard field operations, but keeps track of bounds on input and output.
- struct ConvertTo< T >

  Force conversion to appropriate element type of ElementCategory T.
- struct DelayedTag

  Performs field operations with delayed mod reductions. Ensures result is reduced.
- struct LazyTag

  Performs field operations with delayed mod only when necessary. Result may not be reduced.
- struct GenericTag

  default is generic
- struct MachineFloatTag

  float or double
- struct MachineIntTag

  short, int, long, long long, and unsigned variants
- struct FixedPrecIntTag

  Fixed precision integers above machine precision: Givaro::recInt.
- struct ArbitraryPrecIntTag

  Arbitrary precision integers: GMP.
- struct RNSElementTag

  Representation in a Residue Number System.
- struct ElementTraits< Element >

  ElementTraits.

- struct ModeTraits< Field >

  ModeTraits.
- struct FieldTraits< Field >

  FieldTrait.

Namespaces

- namespace FFPACK

  ***F**inite **F**ield **PACK** Set of elimination based routines for dense linear algebra.*
- namespace FFLAS::FieldCategories

  Traits and categories will need to be placed in a proper file later.
- namespace FFLAS::ModeCategories

  Specifies the mode of action for an algorithm w.r.t.

12.12.1 Detailed Description

Field Traits.

12.13 rns-double-elt.h File Reference

rns elt structure with double support

```
#include "fflas-ffpack/utils/fflas_memory.h"
#include "fflas-ffpack/utils/cast.h"
```

Namespaces

- namespace FFPACK

  ***F**inite **F**ield **PACK** Set of elimination based routines for dense linear algebra.*

12.13.1 Detailed Description

rns elt structure with double support

12.14 rns-double.h File Reference

rns structure with double support

```
#include <iterator>
#include <vector>
#include <givaro/modular-floating.h>
#include <givaro/givinteger.h>
#include <givaro/givintprime.h>
#include "givaro/modular-extended.h"
#include <recint/ruint.h>
#include "fflas-ffpack/config-blas.h"
#include "fflas-ffpack/utils/fflas_memory.h"
#include "fflas-ffpack/utils/align-allocator.h"
#include "fflas-ffpack/field/rns-double-elt.h"
#include "rns-double.inl"
#include "rns-double-recint.inl"
```

Namespaces

- namespace FFPACK

  ***F**inite **F**ield **PACK** Set of elimination based routines for dense linear algebra.*

12.14.1 Detailed Description

rns structure with double support

12.15 rns-integer-mod.h File Reference

representation of Z/pZ using RNS representation (note: fixed precision)

```
#include <vector>
#include <cmath>
#include <recint/recint.h>
#include <givaro/modular-integer.h>
#include <givaro/givinteger.h>
#include <givaro/udl.h>
#include "givaro/modular-extended.h"
#include "fflas-ffpack/field/rns-double.h"
#include "fflas-ffpack/field/rns-integer.h"
#include "fflas-ffpack/fflas/fflas_level1.inl"
#include "fflas-ffpack/fflas/fflas_level2.inl"
#include "fflas-ffpack/fflas/fflas_level3.inl"
#include "fflas-ffpack/fflas/fflas_enum.h"
#include "fflas-ffpack/fflas/fflas_fscal_mp.inl"
```

Namespaces

- namespace FFPACK

  ***F**inite **F**ield **PACK** Set of elimination based routines for dense linear algebra.*

12.15.1 Detailed Description

representation of Z/pZ using RNS representation (note: fixed precision)

12.16 rns-integer.h File Reference

representation of Z using RNS representation (note: fixed precision)

```
#include <givaro/givinteger.h>
#include "fflas-ffpack/field/rns-double.h"
```

Namespaces

- namespace FFPACK

  ***F**inite **F**ield **PACK** Set of elimination based routines for dense linear algebra.*

12.16.1 Detailed Description

representation of Z using RNS representation (note: fixed precision)

12.17 rns.h File Reference

Namespaces

- namespace FFPACK

  ***F**inite **F**ield **PACK** Set of elimination based routines for dense linear algebra.*

12.18 fflas_lvl1.C File Reference

C functions calls for level 1 FFLAS in fflas-c.h.

```
#include "fflas-ffpack/interfaces/libs/fflas_c.h"
#include "fflas-ffpack/fflas/fflas.h"
#include "givaro//modular-balanced.h"
#include "givaro//modular.h"
```

12.18.1 Detailed Description

C functions calls for level 1 FFLAS in fflas-c.h.

Author

Brice Boyer

See also

fflas/fflas_level1.inl

12.19 fflas_lvl2.C File Reference

C functions calls for level 2 FFLAS in fflas-c.h.

```
#include "fflas-ffpack/interfaces/libs/fflas_c.h"
#include "fflas-ffpack/fflas/fflas.h"
#include "givaro//modular-balanced.h"
#include "givaro//modular.h"
```

12.19.1 Detailed Description

C functions calls for level 2 FFLAS in fflas-c.h.

Author

Brice Boyer

See also

fflas/fflas_level2.inl

12.20 fflas_lvl3.C File Reference

C functions calls for level 3 FFLAS in fflas-c.h.

```
#include "fflas-ffpack/interfaces/libs/fflas_c.h"
#include "fflas-ffpack/fflas/fflas.h"
#include "givaro//modular-balanced.h"
#include "givaro//modular.h"
```

12.20.1 Detailed Description

C functions calls for level 3 FFLAS in fflas-c.h.

Author

Brice Boyer

See also

fflas/fflas_level3.inl

12.21 fflas_sparse.C File Reference

C functions calls for level 1.5 and 2.5 FFLAS in fflas-c.h.

12.21.1 Detailed Description

C functions calls for level 1.5 and 2.5 FFLAS in fflas-c.h.

Author

Brice Boyer

See also

fflas/fflas_sparse.h

12.22 ffpack.C File Reference

C functions calls for FFPACK in ffpack-c.h.

```
#include "fflas-ffpack/interfaces/libs/ffpack_c.h"
#include "fflas-ffpack/fflas/fflas.h"
#include "fflas-ffpack/ffpack/ffpack.h"
#include "givaro//modular-balanced.h"
#include "givaro//modular.h"
```

12.22.1 Detailed Description

C functions calls for FFPACK in ffpack-c.h.

Author

Brice Boyer

See also

ffpack/ffpack.h

12.23 debug.h File Reference

Various utilities for debugging.

```
#include <fflas-ffpack/fflas-ffpack-config.h>
#include <iostream>
#include <sstream>
#include <cmath>
#include <stdexcept>
```

Data Structures

- class Failure

  A precondtion failed.

Namespaces

- namespace FFPACK

  ***F**inite **F**ield **PACK** Set of elimination based routines for dense linear algebra.*

12.23.1 Detailed Description

Various utilities for debugging.

Todo we should put vector printing elsewhere.

Index