

FFLAS-FFPACK

Generated by Doxygen 1.9.1

1 FFLAS-FFPACK Documentation.	1
1.1 Introduction	1
1.2 Goals	1
1.3 Design	1
1.4 Using FFLAS-FFPACK.	1
1.5 Contributing to fflas-ffpack, getting assistance.	1
2 Configuring and Installing FFLAS-FFPACK	3
3 Copying and Licence	5
4 Tutorial	7
5 Architecture of the library.	9
6 Bug List	11
7 Bibliography	13
8 Todo List	15
9 Module Index	17
9.1 Modules	17
10 Namespace Index	19
10.1 Namespace List	19
11 Data Structure Index	21
11.1 Data Structures	21
12 File Index	23
12.1 File List	23
13 Module Documentation	25
13.1 CHECKER	25
13.2 FFLAS	25
13.3 Matrix Multiplication Algorithms	25
13.3.1 Detailed Description	25
13.4 SIMD wrapper	26
13.5 FFLAS-FFPACK	26
13.5.1 Detailed Description	26
13.6 FFPACK	26
13.7 FFLAS-FFPACK fields	26
13.7.1 Detailed Description	27
13.8 RNS	27
13.9 Interfaces	27

14 Namespace Documentation	29
14.1 FFLAS::FieldCategories Namespace Reference	29
14.1.1 Detailed Description	29
14.2 FFLAS::ModeCategories Namespace Reference	29
14.2.1 Detailed Description	30
14.3 FFLAS::ParSeqHelper Namespace Reference	30
14.3.1 Detailed Description	30
14.4 FFLAS::StructureHelper Namespace Reference	30
14.4.1 Detailed Description	30
14.5 FFPACK Namespace Reference	30
14.5.1 Detailed Description	37
14.5.2 Function Documentation	37
14.5.2.1 applyP()	38
14.5.2.2 MonotonicApplyP()	38
14.5.2.3 fgetrs() [1/2]	39
14.5.2.4 fgetrs() [2/2]	40
14.5.2.5 fgesv() [1/2]	41
14.5.2.6 fgesv() [2/2]	42
14.5.2.7 ftrtri()	42
14.5.2.8 ftrtrm()	43
14.5.2.9 ftrstr()	43
14.5.2.10 ftrssyr2k()	45
14.5.2.11 fsytrf()	46
14.5.2.12 fsytrf_nonunit()	46
14.5.2.13 PLUQ()	47
14.5.2.14 LUdivine() [1/2]	47
14.5.2.15 ColumnEchelonForm()	48
14.5.2.16 RowEchelonForm()	49
14.5.2.17 ReducedColumnEchelonForm()	50
14.5.2.18 ReducedRowEchelonForm()	50
14.5.2.19 Invert() [1/2]	51
14.5.2.20 Invert() [2/2]	51
14.5.2.21 Invert2()	52
14.5.2.22 CharPoly() [1/3]	53
14.5.2.23 CharPoly() [2/3]	53
14.5.2.24 CharPoly() [3/3]	54
14.5.2.25 MinPoly() [1/2]	54
14.5.2.26 MinPoly() [2/2]	55
14.5.2.27 MatVecMinPoly()	55
14.5.2.28 Rank()	56
14.5.2.29 IsSingular()	56
14.5.2.30 Det()	57

14.5.2.31 Solve()	57
14.5.2.32 RandomNullSpaceVector() [1/2]	58
14.5.2.33 NullSpaceBasis()	59
14.5.2.34 RowRankProfile()	59
14.5.2.35 ColumnRankProfile()	60
14.5.2.36 RankProfileFromLU()	60
14.5.2.37 LeadingSubmatrixRankProfiles()	61
14.5.2.38 RowRankProfileSubmatrixIndices()	62
14.5.2.39 ColRankProfileSubmatrixIndices()	62
14.5.2.40 RowRankProfileSubmatrix()	63
14.5.2.41 ColRankProfileSubmatrix()	64
14.5.2.42 getTriangular() [1/2]	64
14.5.2.43 getTriangular() [2/2]	65
14.5.2.44 getEchelonForm() [1/2]	66
14.5.2.45 getEchelonForm() [2/2]	66
14.5.2.46 getEchelonTransform()	67
14.5.2.47 getReducedEchelonForm() [1/2]	68
14.5.2.48 getReducedEchelonForm() [2/2]	69
14.5.2.49 getReducedEchelonTransform()	69
14.5.2.50 LQUPtoInverseOfFullRankMinor()	70
14.5.2.51 RandomNullSpaceVector() [2/2]	71
14.5.2.52 buildMatrix()	71
14.5.2.53 fsytrf_UP_RPM()	72
14.5.2.54 LUdivine() [2/2]	72
14.5.2.55 composePermutationsLLL()	72
14.5.2.56 composePermutationsLLM()	73
14.5.2.57 composePermutationsMLM()	73
14.5.2.58 NonZeroRandomMatrix() [1/2]	73
14.5.2.59 NonZeroRandomMatrix() [2/2]	74
14.5.2.60 RandomMatrix() [1/2]	75
14.5.2.61 RandomMatrix() [2/2]	75
14.5.2.62 RandomTriangularMatrix() [1/2]	76
14.5.2.63 RandomTriangularMatrix() [2/2]	76
14.5.2.64 RandomSymmetricMatrix()	78
14.5.2.65 RandomMatrixWithRank() [1/2]	78
14.5.2.66 RandomMatrixWithRank() [2/2]	79
14.5.2.67 RandomIndexSubset()	80
14.5.2.68 RandomPermutation()	80
14.5.2.69 RandomRankProfileMatrix()	80
14.5.2.70 RandomSymmetricRankProfileMatrix()	81
14.5.2.71 RandomMatrixWithRankandRPM() [1/2]	81
14.5.2.72 RandomMatrixWithRankandRPM() [2/2]	82

14.5.2.73 RandomSymmetricMatrixWithRankandRPM() [1/2]	82
14.5.2.74 RandomSymmetricMatrixWithRankandRPM() [2/2]	83
14.5.2.75 RandomMatrixWithRankandRandomRPM() [1/2]	84
14.5.2.76 RandomMatrixWithRankandRandomRPM() [2/2]	84
14.5.2.77 RandomSymmetricMatrixWithRankandRandomRPM() [1/2]	85
14.5.2.78 RandomSymmetricMatrixWithRankandRandomRPM() [2/2]	85
14.5.2.79 RandomMatrixWithDet() [1/2]	87
14.5.2.80 RandomMatrixWithDet() [2/2]	87
15 Data Structure Documentation	91
15.1 ArbitraryPrecIntTag Struct Reference	91
15.1.1 Detailed Description	91
15.2 ConvertTo< T > Struct Template Reference	91
15.2.1 Detailed Description	91
15.3 DefaultBoundedTag Struct Reference	92
15.3.1 Detailed Description	92
15.4 DefaultTag Struct Reference	92
15.4.1 Detailed Description	92
15.5 DelayedTag Struct Reference	92
15.5.1 Detailed Description	92
15.6 ElementTraits< Element > Struct Template Reference	93
15.6.1 Detailed Description	93
15.7 Failure Class Reference	93
15.7.1 Detailed Description	93
15.8 FieldTraits< Field > Struct Template Reference	93
15.8.1 Detailed Description	94
15.9 FixedPrecIntTag Struct Reference	94
15.9.1 Detailed Description	94
15.10 ftrsmLeftUpperNoTransNonUnit< Element > Class Template Reference	94
15.10.1 Detailed Description	94
15.11 GenericTag Struct Reference	95
15.11.1 Detailed Description	95
15.12 GenericTag Struct Reference	95
15.12.1 Detailed Description	95
15.13 LazyTag Struct Reference	95
15.13.1 Detailed Description	96
15.14 MachineFloatTag Struct Reference	96
15.14.1 Detailed Description	96
15.15 MachineIntTag Struct Reference	96
15.15.1 Detailed Description	96
15.16 MMHelper< Field, AlgoTrait, ModeCategories::DefaultTag, ParSeqTrait > Struct Template Reference	96
15.16.1 Detailed Description	97

15.17 ModeTraits< Field > Struct Template Reference	97
15.17.1 Detailed Description	97
15.18 ModularTag Struct Reference	97
15.18.1 Detailed Description	97
15.19 RNSElementTag Struct Reference	98
15.19.1 Detailed Description	98
15.20 TRSMHelper< ReclterTrait, ParSeqTrait > Struct Template Reference	98
15.20.1 Detailed Description	98
15.21 UnparametricTag Struct Reference	98
15.21.1 Detailed Description	98
16 File Documentation	99
16.1 debug.h File Reference	99
16.1.1 Detailed Description	99
16.2 fflas-ffpack-config.h File Reference	99
16.2.1 Detailed Description	100
16.3 fflas-ffpack.h File Reference	100
16.3.1 Detailed Description	100
16.4 fflas.h File Reference	100
16.4.1 Detailed Description	101
16.4.2 Macro Definition Documentation	101
16.4.2.1 DOUBLE_TO_FLOAT_CROSSOVER	101
16.5 fflas_ftsm_mp.inl File Reference	102
16.5.1 Detailed Description	102
16.6 fflas_lvl1.C File Reference	102
16.6.1 Detailed Description	102
16.7 fflas_lvl2.C File Reference	102
16.7.1 Detailed Description	103
16.8 fflas_lvl3.C File Reference	103
16.8.1 Detailed Description	103
16.9 fflas_sparse.C File Reference	103
16.9.1 Detailed Description	103
16.10 fflas_sparse.h File Reference	104
16.11 fflas_sparse.inl File Reference	104
16.12 ffpack.C File Reference	104
16.12.1 Detailed Description	104
16.13 ffpack.h File Reference	105
16.13.1 Detailed Description	110
16.13.2 Function Documentation	110
16.13.2.1 GaussJordan()	111
16.13.2.2 RandomKrylovPrecond()	111
16.14 fgemm_classical_mp.inl File Reference	112

16.14.1 Detailed Description	112
16.15 field-traits.h File Reference	112
16.15.1 Detailed Description	113
16.16 read_sparse.h File Reference	113
16.17 rns-double-elt.h File Reference	113
16.17.1 Detailed Description	113
16.18 rns-double.h File Reference	114
16.18.1 Detailed Description	114
16.19 rns-integer-mod.h File Reference	114
16.19.1 Detailed Description	114
16.20 rns-integer.h File Reference	115
16.20.1 Detailed Description	115
16.21 rns.h File Reference	115
16.22 schedule_bini.inl File Reference	115
16.22.1 Detailed Description	115
Index	117

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 specificities 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

2.3.0

Chapter 2

Configuring and Installing FFLAS-FFPACK

FFLAS-FFPACK is a header-only package.

Howver 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.

Chapter 3

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>

Chapter 4

Tutorial

no doc.

Chapter 5

Architecture of the library.

no doc.

Chapter 6

Bug List

Global `DOUBLE_TO_FLOAT_CROSSOVER`

to be benchmarked.

Global `FFPACK::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)

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 7

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=FpackSlabRecursive, 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::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 **ftsrsmLeftUpperNoTransNonUnit**< Element >

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

Chapter 8

Todo List

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 Ida, 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 Ida, 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 ??

File **debug.h**

we should put vector printing elsewhere.

Module **field**

biblio

Module **MMalgos**

biblio

Module **simd**

biblio

Chapter 9

Module Index

9.1 Modules

Here is a list of all modules:

CHECKER	25
Matrix Multiplication Algorithms	25
SIMD wrapper	26
FFLAS-FFPACK	26
FFLAS	25
Interfaces	27
FFPACK	26
FFLAS-FFPACK fields	26
RNS	27

Chapter 10

Namespace Index

10.1 Namespace List

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

FFLAS::FieldCategories	
Traits and categories will need to be placed in a proper file later	29
FFLAS::ModeCategories	
Specifies the mode of action for an algorithm w.r.t	29
FFLAS::ParSeqHelper	
ParSeqHelper for both fgemm and ftrsm	30
FFLAS::StructureHelper	
StructureHelper for ftrsm	30
FFPACK	
Finite Field PACK Set of elimination based routines for dense linear algebra	30

Chapter 11

Data Structure Index

11.1 Data Structures

Here are the data structures with brief descriptions:

ArbitraryPrecIntTag	Arbitrary precision integers: GMP	91
ConvertTo< T >	Force conversion to appropriate element type of ElementCategory T	91
DefaultBoundedTag	Use standard field operations, but keeps track of bounds on input and output	92
DefaultTag	No specific mode of action: use standard field operations	92
DelayedTag	Performs field operations with delayed mod reductions. Ensures result is reduced	92
ElementTraits< Element >	ElementTraits	93
Failure	A precondition failed	93
FieldTraits< Field >	FieldTrait	93
FixedPrecIntTag	Fixed precision integers above machine precision: Givaro::reclnt	94
ftrsmLeftUpperNoTransNonUnit< Element >	Computes the maximal size for delaying the modular reduction in a triangular system resolution	94
GenericTag	Default is generic	95
GenericTag	Generic ring	95
LazyTag	Performs field operations with delayed mod only when necessary. Result may not be reduced	95
MachineFloatTag	Float or double	96
MachineIntTag	Short, int, long, long long, and unsigned variants	96
MMHelper< Field, AlgoTrait, ModeCategories::DefaultTag, ParSeqTrait >	FGEMM Helper for Default and ConvertTo modes of operation	96
ModeTraits< Field >	ModeTraits	97
ModularTag	This is a modular field like e.g. <code>Modular<T></code> or <code>ModularBalanced<T></code>	97

RNSElementTag	
Representation in a Residue Number System	98
TRSMHelper< ReclterTrait, ParSeqTrait >	
TRSM Helper	98
UnparametricTag	
If the field uses a representation with infix operators	98

Chapter 12

File Index

12.1 File List

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

debug.h	Various utilities for debugging	99
fflas-ffpack-config.h	Defaults for optimised values	99
fflas-ffpack.h	Includes FFLAS and FFPACK	100
fflas.h	Finite Field Linear Algebra Subroutines	100
fflas_ftsm_mp.inl	Triangular system with matrix right hand side over multiprecision domain (either over \mathbb{Z} or over $\mathbb{Z}/p\mathbb{Z}$)	102
fflas_lv1.C	C functions calls for level 1 FFLAS in fflas-c.h	102
fflas_lv2.C	C functions calls for level 2 FFLAS in fflas-c.h	102
fflas_lv3.C	C functions calls for level 3 FFLAS in fflas-c.h	103
fflas_sparse.C	C functions calls for level 1.5 and 2.5 FFLAS in fflas-c.h	103
fflas_sparse.h		104
fflas_sparse.inl		104
ffpack.C	C functions calls for FFPACK in ffpack-c.h	104
ffpack.h	Set of elimination based routines for dense linear algebra	105
fgemm_classical_mp.inl	Matrix multiplication with multiprecision input (either over \mathbb{Z} or over $\mathbb{Z}/p\mathbb{Z}$)	112
field-traits.h	Field Traits	112
read_sparse.h		113
rns-double-elt.h	Rns elt structure with double support	113
rns-double.h	Rns structure with double support	114
rns-integer-mod.h	Representation of $\mathbb{Z}/p\mathbb{Z}$ using RNS representation (note: fixed precision)	114

rns-integer.h	
Representation of \mathbb{Z} using RNS representation (note: fixed precision)	115
rns.h	115
schedule_bini.inl	
Bini implementation	115

Chapter 13

Module Documentation

13.1 CHECKER

Class CHECKER provides functions to verify computations in FFLAS and [FFPACK](#).

Class CHECKER provides functions to verify computations in FFLAS and [FFPACK](#).

13.2 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.

13.3 Matrix Multiplication Algorithms

Matrix Multiplication (level 3) algorithms.

Files

- file [schedule_bini.inl](#)
Bini implementation.

13.3.1 Detailed Description

Matrix Multiplication (level 3) algorithms.

[Todo](#) biblio

13.4 SIMD wrapper

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

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

Todo biblio

13.5 FFLAS-FFPACK

the FFLAS [FFPACK](#) library

Modules

- [FFLAS](#)
The C-style wrapper of BLAS for finite field linear algebra.
- [Interfaces](#)
Intefaces for FFLAS-FFPACK.

13.5.1 Detailed Description

the FFLAS [FFPACK](#) library

C++ header library for fast exact dense linear algebra

See also

[FFLAS](#)
[FFPACK](#)

13.6 FFPACK

Class [FFPACK](#) provides functions using fflas much as Lapack uses BLAS.

Class [FFPACK](#) provides functions using fflas much as Lapack uses BLAS.

13.7 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](#)

13.7.1 Detailed Description

fields in the FFLAS-FFPACK library

Unparametric/Random elements

[Todo](#) biblio

13.8 RNS

just include them all

just include them all

13.9 Interfaces

Intefaces for FFLAS-FFPACK.

Intefaces for FFLAS-FFPACK.

C interface in folder

See also

libs

Chapter 14

Namespace Documentation

14.1 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.

14.1.1 Detailed Description

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

14.2 FFLAS::ModeCategories Namespace Reference

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

Data Structures

- 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](#)
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.

14.2.1 Detailed Description

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

its field

14.3 FFLAS::ParSeqHelper Namespace Reference

[ParSeqHelper](#) for both fgemm and ftrsm.

14.3.1 Detailed Description

[ParSeqHelper](#) for both fgemm and ftrsm.

[ParSeqHelper](#) for both fgemm and ftrsm

14.4 FFLAS::StructureHelper Namespace Reference

[StructureHelper](#) for ftrsm.

14.4.1 Detailed Description

[StructureHelper](#) for ftrsm.

14.5 FFPACK Namespace Reference

Finite Field PACK Set of elimination based routines for dense linear algebra.

Data Structures

- class [Failure](#)
A precondition 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 $P1 \times \text{Diag}(I_R, P2)$ 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 $AX = B$ or $XA = B$.
- 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 [ftrtm](#) (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::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 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=FpackSlabRecursive, 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=FpackSlabRecursive)

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=FpackSlabRecursive)

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=FpackSlabRecursive)

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 LuTag=FpackSlabRecursive)

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 Ida, 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 Ida, 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 Ida, const FFPACK_CHARPOLY_TAG CharpTag=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 Ida)`
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 Ida, 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::ConstElement_ptr A, const size_t Ida, 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 $(v, Av, \dots, A^N v)$.
- `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 Ida)`
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 Ida)`
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 Ida, 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 Ida, 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 Ida, typename Field::Element_ptr X, const size_t incX)`
Solve $LX = B$ or $XL = B$ 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 Ida, typename Field::Element_ptr &NS, size_t &Idn, 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 $r \times r$ 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 $r \times r$ 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 $A=L\backslash U$ 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 $A=L\backslash U$ 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 $A=L\backslash U$ of rank R obtained by RowEchelonForm or ColumnEchelonForm.
- `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 $A=L\backslash U$ 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 $A=L\backslash U$ 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 $A=L\backslash U$ 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 $A=L\backslash U$ 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 $A=L\backslash U$ 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 >
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 $LX = B$ or $XL = B$ in place.
- 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 &F, const size_t N, typename Field::Element_ptr A, const size_t lda, typename Field::Element_ptr Din, const size_t incDin, 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 $P1 \times \text{Diag}(I_R, P2)$ 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 $P1 \times \text{Diag}(I_R, P2)$ 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 \times 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 $\{0, \dots, N - 1\}$ 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 $M \times N$: 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 $N \times N$: 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 $m \times n$ 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 $m \times n$ matrix with random entries and rank r .
- `template<class Field , class Randlter >`
`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, Randlter &G)
Random Symmetric Matrix with prescribed rank and rank profile matrix Creates an $n \times n$ 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 $n \times n$ symmetric matrix with random entries and rank r .
- `template<class Field , class Randlter >`
`Field::Element_ptr RandomMatrixWithRankandRandomRPM` (const Field &F, size_t M, size_t N, size_t R, typename Field::Element_ptr A, size_t lda, Randlter &G)
Random Matrix with prescribed rank, with random rank profile matrix Creates an $m \times n$ 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 $m \times n$ matrix with random entries, rank r and with a rank profile matrix chosen uniformly at random.
- `template<class Field , class Randlter >`
`Field::Element_ptr RandomSymmetricMatrixWithRankandRandomRPM` (const Field &F, size_t N, size_t R, typename Field::Element_ptr A, size_t lda, Randlter &G)
Random Symmetric Matrix with prescribed rank, with random rank profile matrix Creates an $n \times n$ 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 $n \times n$ 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 Randlter >`
`Field::Element_ptr RandomMatrixWithDet` (const Field &F, size_t n, const typename Field::Element d, typename Field::Element_ptr A, size_t lda, Randlter &G)
Random Matrix with prescribed det.

14.5.1 Detailed Description

Finite Field **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.

14.5.2 Function Documentation

14.5.2.1 applyP()

```
void FFPACK::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 $P1 \times \text{Diag}(I_R, P2)$ where $P1$ is a LAPACK and $P2$ a LAPACK permutation and store the result in $P1$ as a LAPACK permutation.

Parameters

<i>in, out</i>	<i>P1</i>	a LAPACK permutation of size N
	<i>P2</i>	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

<i>F</i>	base field
<i>Side</i>	decides if rows (FflasLeft) or columns (FflasRight) are permuted
<i>Trans</i>	decides if the matrix is seen as columns (FflasTrans) or rows (FflasNoTrans)
<i>M</i>	size of the elements to permute
<i>ibeg</i>	first index to consider in P
<i>iend</i>	last index to consider in P
<i>A</i>	input matrix
<i>lda</i>	leading dimension of A
<i>P</i>	permutation in LAPACK format
<i>psh</i>	(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 $P=[2,2,2]$

14.5.2.2 MonotonicApplyP()

```
void FFPACK::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.

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 representation (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

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

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

14.5.2.3 fgetrs() [1/2]

```

void FFPACK::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

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

14.5.2.4 fgetrs() [2/2]

```
Field::Element_ptr FFPACK::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 $AX = B$ or $XA = B$.

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

<i>F</i>	base field
<i>Side</i>	Determine wheter the resolution is left (FflasLeft) or right (FflasRight) looking.
<i>M</i>	row dimension of A
<i>N</i>	col dimension of A

Parameters

<i>NRHS</i>	number of columns (if Side = FFLAS::FflasLeft) or row (if Side = FFLAS::FflasRight) of the matrices X and B
<i>R</i>	rank of A
<i>A</i>	input matrix
<i>lda</i>	leading dimension of A
<i>P</i>	row permutation of the PLUQ decomposition of A
<i>Q</i>	column permutation of the PLUQ decomposition of A
<i>X</i>	solution matrix
<i>ldx</i>	leading dimension of X
<i>B</i>	Right/Left hand side matrix.
<i>ldb</i>	leading dimension of B
<i>info</i>	Success of the computation: 0 if successful, >0 if system is inconsistent

14.5.2.5 fgesv() [1/2]

```

size_t FFPACK::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

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

14.5.2.6 fgesv() [2/2]

```
size_t FFPACK::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

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

14.5.2.7 ftrtri()

```
void FFPACK::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

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

14.5.2.8 ftrtrm()

```
void FFPACK::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

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

14.5.2.9 ftrstr()

```
void FFPACK::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 )
```

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

Parameters

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

14.5.2.10 ftrssyr2k()

```
void FFPACK::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.

C is overwritten by B.

Parameters

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

14.5.2.11 fsytrf()

```
bool FFPACK::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

	<i>F</i>	The computation domain
	<i>UpLo</i>	Determine wheter to store the upper (FflasUpper) or lower (FflasLower) triangular factor
	<i>N</i>	order of the matrix A
<i>in, out</i>	<i>A</i>	input matrix
	<i>lda</i>	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.

14.5.2.12 fsytrf_nonunit()

```
bool FFPACK::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

	<i>F</i>	The computation domain
	<i>UpLo</i>	Determine wheter to store the upper (FflasUpper) or lower (FflasLower) triangular factor
	<i>N</i>	order of the matrix A
<i>in, out</i>	<i>A</i>	input matrix
<i>in, out</i>	<i>D</i>	
	<i>lda</i>	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 : $A = L \times D_{inv} \times L^T$ 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.

14.5.2.13 PLUQ()

```
size_t 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 )
```

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

14.5.2.14 LUdivine() [1/2]

```
size_t 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 )

```

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

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

14.5.2.15 ColumnEchelonForm()

```

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 $A = [M \setminus V]$ such that $AU = C$ is a column echelon decomposition of A , with $U = P^T [V]$ and $C = M + Q [I_r] [0 \text{ } I_{n-r}] [0]$ If `LuTag == FfpackTileRecursive` then $A = [N \setminus V]$ such that the same holds with $M = Q N$

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

Parameters

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

14.5.2.16 RowEchelonForm()

```
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 $A = [L \setminus M]$ such that $XA = R$ is a row echelon decomposition of A , with $X = [L \ 0] P$ and $R = M + [I_r \ 0] Q^T [I_{n-r}]$ If `LuTag == FfpackTileRecursive` then $A = [L \setminus N]$ such that the same holds with $M = N Q^T Q^T = Q^T$ If `transform=false`, the matrix L is not computed. See also `test-rowechelon` for an example of use

Parameters

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

14.5.2.17 ReducedColumnEchelonForm()

```
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 $R = Q [I_r] [0\ I_{n-r}] [M\ 0] Q^T = Q^T$ 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

14.5.2.18 ReducedRowEchelonForm()

```
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 $XA = R$ is a reduced row echelon $[V2\ 0]$ decomposition of A , where $X = [V1\ 0] P$ and $R = [I_r\ M] Q^T [V2\ I_{n-r}] [0] Q^T = Q^T$ If transform=false, the matrix X is not computed and the matrix $A = R$

Parameters

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

14.5.2.19 Invert() [1/2]

```
Field::Element_ptr FFPACK::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

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

Returns

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

14.5.2.20 Invert() [2/2]

```
Field::Element_ptr FFPACK::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 <code>NULL</code> and <code>nullity = 0</code>). It is untouched otherwise.
	ldx	leading dimension of X
	$nullity$	dimension of the kernel of A

Returns

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

14.5.2.21 Invert2()

```
Field::Element_ptr 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 )
```

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

An $2n^3f$ 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 "psychological" factorisation LU.
	lda	leading dimension of A Generated by Doxygen
out	X	this is the inverse of A if A is invertible (non <code>NULL</code> and <code>nullity = 0</code>). It is untouched otherwise.
	ldx	leading dimension of X

Returns

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

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

14.5.2.22 CharPoly() [1/3]

```
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

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

14.5.2.23 CharPoly() [2/3]

```
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

	<i>R</i>	the polynomial ring of charp (contains the base field)
--	----------	--

Parameters

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

14.5.2.24 CharPoly() [3/3]

```
PolRing::Element& FFPACK::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

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

14.5.2.25 MinPoly() [1/2]

```
Polynomial& FFPACK::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*.

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

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

14.5.2.26 MinPoly() [2/2]

```
Polynomial& FFPACK::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 .

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

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

14.5.2.27 MatVecMinPoly()

```
Polynomial& FFPACK::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 )
```

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

Parameters

	F	the base field
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

14.5.2.28 Rank()

```
size_t FFPACK::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

14.5.2.29 IsSingular()

```
bool FFPACK::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 $M \neq N$, 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

14.5.2.30 Det()

```
Field::Element& FFPACK::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.

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

14.5.2.31 Solve()

```
Field::Element_ptr FFPACK::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.

@oaram F base field @oaram M matrix order

Parameters

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

14.5.2.32 RandomNullSpaceVector() [1/2]

```

* void FFPACK::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 $LX = B$ or $XL = B$ in place.

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

Parameters

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

14.5.2.33 NullSpaceBasis()

```
size_t FFPACK::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

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

14.5.2.34 RowRankProfile()

```
size_t FFPACK::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.

Parameters

	<i>F</i>	base field
	<i>M</i>	number of rows
	<i>N</i>	number of columns

Parameters

in	<i>A</i>	input matrix of dimension M x N
	<i>lda</i>	leading dimension of A
out	<i>rkprofile</i>	return the rank profile as an array of row indexes, of dimension r=rank(A)
	<i>LuTag</i>	chooses the elimination algorithm. SlabRecursive for LUdivine, TileRecursive for PLUQ

A is modified rkprofile is allocated during the computation.

Returns

R

14.5.2.35 ColumnRankProfile()

```
size_t FFPACK::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.

Parameters

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

A is modified rkprofile is allocated during the computation.

Returns

R

14.5.2.36 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

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

14.5.2.37 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

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

Returns

the rank of the $LS_m \times LS_n$ 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.

14.5.2.38 RowRankProfileSubmatrixIndices()

```
size_t FFPACK::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 $r \times r$ 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

14.5.2.39 ColRankProfileSubmatrixIndices()

```
size_t FFPACK::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 $r \times r$ 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

14.5.2.40 RowRankProfileSubmatrix()

```

size_t FFPACK::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 $r \times r$ 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 $M \times N$
	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

14.5.2.41 ColRankProfileSubmatrix()

```
size_t FFPACK::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
	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

14.5.2.42 getTriangular() [1/2]

```
void 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 )

```

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

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

Parameters

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

Todo just one triangular fzero+fassign ?

14.5.2.43 getTriangular() [2/2]

```

void 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 )

```

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

Parameters

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

Todo just one triangular fzero+fassign ?

14.5.2.44 getEchelonForm() [1/2]

```
void FFPACK::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 $A=LU$ 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

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

14.5.2.45 getEchelonForm() [2/2]

```
void FFPACK::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 $A=L\backslash U$ 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

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

14.5.2.46 getEchelonTransform()

```

void FFPACK::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 $A=L\backslash U$ of rank R obtained by Row↔ EchelonForm or ColumnEchelonForm.

If Uplo == FflasLower: T is $N \times N$ (already allocated) such that $A T = C$ is a transformation of A in Column echelon form Else T is $M \times M$ (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)

14.5.2.47 `getReducedEchelonForm()` [1/2]

```
void FFPACK::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 $A=L\backslash U$ 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

Parameters

	<i>ldt</i>	leading dimension of T
	<i>LuTag</i>	which factorized form (CUP/PLE if FfpackSlabRecursive, PLUQ if FfpackTileRecursive)
	<i>OnlyNonZeroVectors</i>	decides whether the last zero rows/columns should be ignored

14.5.2.48 getReducedEchelonForm() [2/2]

```
void FFPACK::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 $A=L\backslash U$ 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.

Parameters

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

14.5.2.49 getReducedEchelonTransform()

```
void FFPACK::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 $A=LU$ of rank R obtained by Row↔EchelonForm or ColumnEchelonForm.

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

Parameters

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

14.5.2.50 LQUPToInverseOfFullRankMinor()

```

Field::Element_ptr FFPACK::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 $Q^t.A.P^t$ has an invertible leading principal $r \times r$ 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

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

14.5.2.51 RandomNullSpaceVector() [2/2]

```
void FFPACK::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 $LX = B$ or $XL = B$ in place.

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

Parameters

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

14.5.2.52 buildMatrix()

```
Field::Element_ptr FFPACK::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 :

14.5.2.53 fsytrf_UP_RPM()

```

size_t FFPACK::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 $\leftarrow \begin{bmatrix} I \\ P1 \end{bmatrix} \begin{bmatrix} I_{(N1+R2)} \\ P2^T \end{bmatrix} \begin{bmatrix} P3^T \\ Q2^T \end{bmatrix}$

Changing $\begin{bmatrix} U1 & V1 & E1 & E21 & E22 \end{bmatrix}$ into $\begin{bmatrix} U1 & E11 & E12 & V1 & E* & E* \end{bmatrix} \begin{bmatrix} 0 & L2 & U2 & V21 & V22 \end{bmatrix} \begin{bmatrix} U4 & V41 & 0 & V42 & V43 \end{bmatrix} \begin{bmatrix} 0 & M2 & 0 & 0 \end{bmatrix} \begin{bmatrix} U3 & 0 & 0 & V3 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & H1 & H21 & H22 \end{bmatrix} \begin{bmatrix} 0 & U3 & V3 \end{bmatrix} \begin{bmatrix} 0 & 0 \end{bmatrix}$ where U4 is the 2R2 x 2R2 matrix formed by interleaving U2, L2^T and H1

14.5.2.54 LUdivine() [2/2]

```

size_t 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 ) [inline]

```

Todo std::swap ?

14.5.2.55 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	<i>P1</i>	a LAPACK permutation of size N
	<i>P2</i>	a LAPACK permutation of size N-R

14.5.2.56 composePermutationsLLM()

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

Computes $P1 \times \text{Diag}(I_R, P2)$ where $P1$ is a LAPACK and $P2$ a LAPACK permutation and store the result in MathP as a MathPermutation format.

Parameters

out		
-----	--	--

14.5.2.57 composePermutationsMLM()

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

Computes $\text{MathP1} \times \text{Diag}(I_R, P2)$ where MathP1 is a MathPermutation and $P2$ a LAPACK permutation and store the result in MathP1 as a MathPermutation format.

Parameters

in, out	<i>MathP1</i>	a MathPermutation of size N
	<i>P2</i>	a LAPACK permutation of size N-R

14.5.2.58 NonZeroRandomMatrix() [1/2]

```
Field::Element_ptr FFPACK::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 $m \times n$ 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 $m \times lda$ field elements)
	lda	leading dimension of A
	G	a random iterator

Returns

A .

14.5.2.59 NonZeroRandomMatrix() [2/2]

```

Field::Element_ptr FFPACK::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 $m \times n$ 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 $m \times lda$ field elements)
	lda	leading dimension of A

Returns

A .

14.5.2.60 RandomMatrix() [1/2]

```
Field::Element_ptr FFPACK::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 $m \times n$ 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 $m \times lda$ field elements)
	lda	leading dimension of A
	G	a random iterator

Returns

A .

14.5.2.61 RandomMatrix() [2/2]

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

Random Matrix.

Creates a $m \times n$ 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 $m \times lda$ field elements)
	lda	leading dimension of A

Returns

A.

14.5.2.62 RandomTriangularMatrix() [1/2]

```
Field::Element_ptr FFPACK::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 $m \times n$ triangular matrix with random entries. The `UpLo` parameter defines whether it is upper or lower triangular.

Parameters

	<i>F</i>	field
	<i>m</i>	number of rows in A
	<i>n</i>	number of cols in A
	<i>UpLo</i>	whether A is upper or lower triangular
out	<i>A</i>	the matrix (preallocated to at least $m \times lda$ field elements)
	<i>lda</i>	leading dimension of A
	<i>G</i>	a random iterator

Returns

A.

14.5.2.63 RandomTriangularMatrix() [2/2]

```
Field::Element_ptr FFPACK::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 $m \times n$ 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 $m \times lda$ field elements)
	lda	leading dimension of A

Returns

A .

14.5.2.64 RandomSymmetricMatrix()

```
Field::Element_ptr FFPACK::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 $m \times n$ 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 $n \times lda$ field elements)
	lda	leading dimension of A
	G	a random iterator

Returns

A .

14.5.2.65 RandomMatrixWithRank() [1/2]

```
Field::Element_ptr FFPACK::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 $m \times n$ matrix with random entries and rank r .

Parameters

<i>F</i>	field
<i>m</i>	number of rows in A
<i>n</i>	number of cols in A
<i>r</i>	rank of the matrix to build
<i>A</i>	the matrix (preallocated to at least $m \times lda$ field elements)
<i>lda</i>	leading dimension of A
<i>G</i>	a random iterator

Returns

A.

14.5.2.66 RandomMatrixWithRank() [2/2]

```

Field::Element_ptr FFPACK::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 $m \times n$ matrix with random entries and rank r .

Parameters

	<i>F</i>	field
	<i>m</i>	number of rows in A
	<i>n</i>	number of cols in A
	<i>r</i>	rank of the matrix to build
out	<i>A</i>	the matrix (preallocated to at least $m \times lda$ field elements)
	<i>lda</i>	leading dimension of A

Returns

A.

14.5.2.67 RandomIndexSubset()

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

Pick uniformly at random a sequence of R distinct elements from the set $\{0, \dots, N - 1\}$ 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)

14.5.2.68 RandomPermutation()

```
size_t* FFPACK::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)

14.5.2.69 RandomRankProfileMatrix()

```
void FFPACK::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 $M \times N$: a matrix with only R non-zeros equal to one, in a random rook placement.

Parameters

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

14.5.2.70 RandomSymmetricRankProfileMatrix()

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

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

Parameters

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

14.5.2.71 RandomMatrixWithRankandRPM() [1/2]

```
Field::Element_ptr FFPACK::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 $m \times n$ 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 $m \times lda$ field elements)

Parameters

<i>lda</i>	leading dimension of A
<i>RRP</i>	the R dimensional array with row positions of the rank profile matrix' pivots
<i>CRP</i>	the R dimensional array with column positions of the rank profile matrix' pivots
<i>G</i>	a random iterator

Returns

A.

14.5.2.72 RandomMatrixWithRankandRPM() [2/2]

```
Field::Element_ptr FFPACK::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 $m \times n$ matrix with random entries and rank r .

Parameters

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

Returns

A.

14.5.2.73 RandomSymmetricMatrixWithRankandRPM() [1/2]

```
Field::Element_ptr FFPACK::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 $n \times n$ symmetric matrix with random entries and rank r .

Parameters

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

Returns

A.

14.5.2.74 RandomSymmetricMatrixWithRankandRPM() [2/2]

```

Field::Element_ptr FFPACK::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 $n \times n$ symmetric matrix with random entries and rank r .

Parameters

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

Returns

A .

14.5.2.75 RandomMatrixWithRankandRandomRPM() [1/2]

```
Field::Element_ptr FFPACK::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 $m \times n$ 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 $m \times lda$ field elements)
lda	leading dimension of A
G	a random iterator

Returns

A .

14.5.2.76 RandomMatrixWithRankandRandomRPM() [2/2]

```
Field::Element_ptr FFPACK::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 $m \times n$ matrix with random entries, rank r and with a rank profile matrix chosen uniformly at random.

Parameters

F	field
-----	-------

Parameters

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

Returns

A.

14.5.2.77 RandomSymmetricMatrixWithRankandRandomRPM() [1/2]

```
Field::Element_ptr FFPACK::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

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

Returns

A.

14.5.2.78 RandomSymmetricMatrixWithRankandRandomRPM() [2/2]

```
Field::Element_ptr FFPACK::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 $n \times n$ matrix with random entries, rank r and with a rank profile matrix chosen uniformly at random.

Parameters

<i>F</i>	field
<i>n</i>	order of A
<i>r</i>	rank of A
<i>A</i>	the matrix (preallocated to at least $n \times lda$ field elements)
<i>lda</i>	leading dimension of A

Returns

A .

14.5.2.79 RandomMatrixWithDet() [1/2]

```
Field::Element_ptr FFPACK::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 $m \times n$ matrix with random entries and rank r .

Parameters

<i>F</i>	field
<i>d</i>	the prescribed value for the determinant of A
<i>n</i>	number of cols in A
<i>A</i>	the matrix to be generated (preallocated to at least $n \times lda$ field elements)
<i>lda</i>	leading dimension of A

Returns

A .

14.5.2.80 RandomMatrixWithDet() [2/2]

```
Field::Element_ptr FFPACK::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 $m \times n$ 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 $n \times lda$ field elements)
lda	leading dimension of A

Returns

A .

Chapter 15

Data Structure Documentation

15.1 ArbitraryPrecIntTag Struct Reference

Arbitrary precision integers: GMP.

```
#include <field-traits.h>
```

15.1.1 Detailed Description

Arbitrary precision integers: GMP.

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

- [field-traits.h](#)

15.2 ConvertTo< T > Struct Template Reference

Force conversion to appropriate element type of ElementCategory T.

```
#include <field-traits.h>
```

15.2.1 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<RecInt<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](#)

15.3 DefaultBoundedTag Struct Reference

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

```
#include <field-traits.h>
```

15.3.1 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](#)

15.4 DefaultTag Struct Reference

No specific mode of action: use standard field operations.

```
#include <field-traits.h>
```

15.4.1 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](#)

15.5 DelayedTag Struct Reference

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

```
#include <field-traits.h>
```

15.5.1 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](#)

15.6 ElementTraits< Element > Struct Template Reference

[ElementTraits.](#)

```
#include <field-traits.h>
```

15.6.1 Detailed Description

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

[ElementTraits.](#)

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

- [field-traits.h](#)

15.7 Failure Class Reference

A precondition failed.

```
#include <debug.h>
```

15.7.1 Detailed Description

A precondition 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](#)

15.8 FieldTraits< Field > Struct Template Reference

FieldTrait.

```
#include <field-traits.h>
```

15.8.1 Detailed Description

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

FieldTrait.

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

- [field-traits.h](#)

15.9 FixedPrecIntTag Struct Reference

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

```
#include <field-traits.h>
```

15.9.1 Detailed Description

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

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

- [field-traits.h](#)

15.10 ftrsmLeftUpperNoTransNonUnit< Element > Class Template Reference

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

15.10.1 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 \mathbb{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`

15.11 GenericTag Struct Reference

default is generic

```
#include <field-traits.h>
```

15.11.1 Detailed Description

default is generic

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

- [field-traits.h](#)

15.12 GenericTag Struct Reference

generic ring.

```
#include <field-traits.h>
```

15.12.1 Detailed Description

generic ring.

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

- [field-traits.h](#)

15.13 LazyTag Struct Reference

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

```
#include <field-traits.h>
```

15.13.1 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](#)

15.14 MachineFloatTag Struct Reference

float or double

```
#include <field-traits.h>
```

15.14.1 Detailed Description

float or double

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

- [field-traits.h](#)

15.15 MachineIntTag Struct Reference

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

```
#include <field-traits.h>
```

15.15.1 Detailed Description

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

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

- [field-traits.h](#)

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

FGEMM Helper for Default and ConvertTo modes of operation.

15.16.1 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](#)

15.17 ModeTraits< Field > Struct Template Reference

[ModeTraits](#).

```
#include <field-traits.h>
```

15.17.1 Detailed Description

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

[ModeTraits](#).

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

- [field-traits.h](#)

15.18 ModularTag Struct Reference

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

```
#include <field-traits.h>
```

15.18.1 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](#)

15.19 RNSElementTag Struct Reference

Representation in a Residue Number System.

```
#include <field-traits.h>
```

15.19.1 Detailed Description

Representation in a Residue Number System.

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

- [field-traits.h](#)

15.20 TRSMHelper< RecIterTrait, ParSeqTrait > Struct Template Reference

TRSM Helper.

15.20.1 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](#)

15.21 UnparametricTag Struct Reference

If the field uses a representation with infix operators.

```
#include <field-traits.h>
```

15.21.1 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 16

File Documentation

16.1 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 precondition failed.

Namespaces

- [FFPACK](#)
*Finite Field **PACK** Set of elimination based routines for dense linear algebra.*

16.1.1 Detailed Description

Various utilities for debugging.

Todo we should put vector printing elsewhere.

16.2 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"
```

16.2.1 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.

16.3 fflas-ffpack.h File Reference

Includes FFLAS and [FFPACK](#).

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

16.3.1 Detailed Description

Includes FFLAS and [FFPACK](#).

16.4 fflas.h File Reference

Finite Field Linear Algebra Subroutines

```
#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_fsyr2k.inl"
```

```
#include "fflas_ftrsm.inl"
#include "fflas_pfttrsm.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.

16.4.1 Detailed Description

Finite Field Linear Algebra Subroutines

Author

Clément Pernet.

16.4.2 Macro Definition Documentation

16.4.2.1 `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.

16.5 fflas_ftdsm_mp.inl File Reference

triangular system with matrix right hand side over multiprecision domain (either over \mathbb{Z} or over $\mathbb{Z}/p\mathbb{Z}$)

```
#include <cmath>
#include <givaro/modular-integer.h>
#include <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"
```

16.5.1 Detailed Description

triangular system with matrix right hand side over multiprecision domain (either over \mathbb{Z} or over $\mathbb{Z}/p\mathbb{Z}$)

16.6 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"
```

16.6.1 Detailed Description

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

Author

Brice Boyer

See also

fflas/fflas_level1.inl

16.7 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"
```

16.7.1 Detailed Description

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

Author

Brice Boyer

See also

[fflas/fflas_level2.inl](#)

16.8 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"
```

16.8.1 Detailed Description

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

Author

Brice Boyer

See also

[fflas/fflas_level3.inl](#)

16.9 fflas_sparse.C File Reference

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

16.9.1 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](#)

16.10 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"
```

16.11 fflas_sparse.inl File Reference

16.12 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"
```

16.12.1 Detailed Description

C functions calls for [FFPACK](#) in ffpack-c.h.

Author

Brice Boyer

See also

[ffpack/ffpack.h](#)

16.13 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.inl"
```

Namespaces

- [FFPACK](#)

*Finite Field **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 $P1 \times Diag(I_R, P2)$ 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 **fgets** (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 **fgets** (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 $AX = B$ or $XA = B$.
- 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 **fttrri** (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 **fttrrm** (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 **ftstr** (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)
Solve a triangular system with a triangular right hand side of the same shape.
- template<class Field >
void **ftssyr2k** (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=FpackSlabRecursive, 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=FpackSlabRecursive)
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=FpackSlabRecursive)
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=FpackSlabRecursive)
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 LuTag=FpackSlabRecursive)
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::Randlter &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 Ida, typename PolRing::Domain_t::Randlter &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 Ida, const FFPACK_CHARPOLY_TAG CharpTag=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 > &completedFactors, const size_t N, typename PolRing::Domain_t::Element_ptr A, const size_t Ida, size_t &Nb, typename PolRing::Domain_t::Element_ptr &B, size_t &ldb, typename PolRing::Domain_t::Randlter &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 Ida)

Compute the minimal polynomial of the matrix A.

- template<class Field , class Polynomial , class Randlter >
Polynomial & **MinPoly** (const Field &F, Polynomial &minP, const size_t N, typename Field::ConstElement_ptr A, const size_t Ida, Randlter &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::ConstElement_ptr A, const size_t Ida, 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 $(v, Av, \dots, A^N v)$.

- 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 Ida)

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 Ida)

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 Ida, 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 Ida, 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 Ida, typename Field::Element_ptr X, const size_t incX)

Solve $LX = B$ or $XL = B$ 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 Ida, 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=FpackSlabRecursive)

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=FpackSlabRecursive)

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=FpackSlabRecursive)

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 $r \times r$ 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 $r \times r$ 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 $A=L\backslash U$ 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 $A=L\backslash U$ 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=FpackSlabRecursive)

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

- 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=FpackSlabRecursive)

Cleans up a compact storage $A=L\backslash U$ 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 $A=L\backslash U$ 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 $A=L\backslash U$ 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 $A=L\backslash U$ 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 $A=L\backslash U$ 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 >`
`Field::Element_ptr LQUptolInverseOfFullRankMinor (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)`
[LQUptolInverseOfFullRankMinor](#).

16.13.1 Detailed Description

Set of elimination based routines for dense linear algebra.

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

16.13.2 Function Documentation

16.13.2.1 GaussJordan()

```
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

where the transformation matrix is stored at the pivot column position

16.13.2.2 RandomKrylovPrecond()

```
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 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 don't assing $K2 \text{ c} \times \text{noc} \times N$ but only $\text{mas}(\text{c}, \text{noc}) \times N$ and store each one after the other

16.14 fgemm_classical_mp.inl File Reference

matrix multiplication with multiprecision input (either over \mathbb{Z} or over $\mathbb{Z}/p\mathbb{Z}$)

```
#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"
```

16.14.1 Detailed Description

matrix multiplication with multiprecision input (either over \mathbb{Z} or over $\mathbb{Z}/p\mathbb{Z}$)

16.15 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::reclnt.
- 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

- [FFPACK](#)
*Finite **Field** **PACK** Set of elimination based routines for dense linear algebra.*
- [FFLAS::FieldCategories](#)
Traits and categories will need to be placed in a proper file later.
- [FFLAS::ModeCategories](#)
Specifies the mode of action for an algorithm w.r.t.

16.15.1 Detailed Description

Field Traits.

16.16 read_sparse.h File Reference

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

16.17 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

- [FFPACK](#)
*Finite **Field** **PACK** Set of elimination based routines for dense linear algebra.*

16.17.1 Detailed Description

rns elt structure with double support

16.18 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

- [FFPACK](#)

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

16.18.1 Detailed Description

rns structure with double support

16.19 rns-integer-mod.h File Reference

representation of $\mathbb{Z}/p\mathbb{Z}$ 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

- [FFPACK](#)

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

16.19.1 Detailed Description

representation of $\mathbb{Z}/p\mathbb{Z}$ using RNS representation (note: fixed precision)

16.20 rns-integer.h File Reference

representation of \mathbb{Z} using RNS representation (note: fixed precision)

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

Namespaces

- [FFPACK](#)

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

16.20.1 Detailed Description

representation of \mathbb{Z} using RNS representation (note: fixed precision)

16.21 rns.h File Reference

Namespaces

- [FFPACK](#)

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

16.22 schedule_bini.inl File Reference

Bini implementation.

16.22.1 Detailed Description

Bini implementation.

Index

applyP
 FFPACK, 37
ArbitraryPrecIntTag, 91

buildMatrix
 FFPACK, 71

CharPoly
 FFPACK, 53, 54
CHECKER, 25
ColRankProfileSubmatrix
 FFPACK, 64
ColRankProfileSubmatrixIndices
 FFPACK, 62
ColumnEchelonForm
 FFPACK, 48
ColumnRankProfile
 FFPACK, 60
composePermutationsLLL
 FFPACK, 72
composePermutationsLLM
 FFPACK, 73
composePermutationsMLM
 FFPACK, 73
ConvertTo< T >, 91

debug.h, 99
DefaultBoundedTag, 92
DefaultTag, 92
DelayedTag, 92
Det
 FFPACK, 57
DOUBLE_TO_FLOAT_CROSSOVER
 fflas.h, 101

ElementTraits< Element >, 93

Failure, 93
FFLAS, 25
FFLAS-FFPACK, 26
FFLAS-FFPACK fields, 26
fflas-ffpack-config.h, 99
fflas-ffpack.h, 100
fflas.h, 100
 DOUBLE_TO_FLOAT_CROSSOVER, 101
FFLAS::FieldCategories, 29
FFLAS::ModeCategories, 29
FFLAS::ParSeqHelper, 30
FFLAS::StructureHelper, 30
fflas_ftsm_mp.inl, 102
fflas_lvl1.C, 102
fflas_lvl2.C, 102
fflas_lvl3.C, 103
fflas_sparse.C, 103
fflas_sparse.h, 104
fflas_sparse.inl, 104
FFPACK, 26, 30
 applyP, 37
 buildMatrix, 71
 CharPoly, 53, 54
 ColRankProfileSubmatrix, 64
 ColRankProfileSubmatrixIndices, 62
 ColumnEchelonForm, 48
 ColumnRankProfile, 60
 composePermutationsLLL, 72
 composePermutationsLLM, 73
 composePermutationsMLM, 73
 Det, 57
 fgesv, 41
 fgetrs, 39, 40
 fsytrf, 45
 fsytrf_nonunit, 46
 fsytrf_UP_RPM, 72
 ftrssyr2k, 45
 ftrstr, 43
 ftrtri, 42
 ftrtrm, 43
 getEchelonForm, 66
 getEchelonTransform, 67
 getReducedEchelonForm, 68, 69
 getReducedEchelonTransform, 69
 getTriangular, 64, 65
 Invert, 51
 Invert2, 52
 IsSingular, 56
 LeadingSubmatrixRankProfiles, 61
 LQUPtoInverseOfFullRankMinor, 70
 LUdivine, 47, 72
 MatVecMinPoly, 55
 MinPoly, 54, 55
 MonotonicApplyP, 38
 NonZeroRandomMatrix, 73, 74
 NullSpaceBasis, 59
 PLUQ, 47
 RandomIndexSubset, 80
 RandomMatrix, 74, 75
 RandomMatrixWithDet, 87
 RandomMatrixWithRank, 78, 79
 RandomMatrixWithRankandRandomRPM, 84
 RandomMatrixWithRankandRPM, 81, 82

- RandomNullSpaceVector, [58](#), [71](#)
- RandomPermutation, [80](#)
- RandomRankProfileMatrix, [80](#)
- RandomSymmetricMatrix, [78](#)
- RandomSymmetricMatrixWithRankandRandomRPM, [85](#)
- RandomSymmetricMatrixWithRankandRPM, [82](#), [83](#)
- RandomSymmetricRankProfileMatrix, [81](#)
- RandomTriangularMatrix, [76](#)
- Rank, [56](#)
- RankProfileFromLU, [60](#)
- ReducedColumnEchelonForm, [50](#)
- ReducedRowEchelonForm, [50](#)
- RowEchelonForm, [49](#)
- RowRankProfile, [59](#)
- RowRankProfileSubmatrix, [63](#)
- RowRankProfileSubmatrixIndices, [62](#)
- Solve, [57](#)
- ffpack.C, [104](#)
- ffpack.h, [105](#)
 - GaussJordan, [110](#)
 - RandomKrylovPrecond, [111](#)
- fgemm_classical_mp.inl, [112](#)
- fgesv
 - FFPACK, [41](#)
- fgetrs
 - FFPACK, [39](#), [40](#)
- field-traits.h, [112](#)
- FieldTraits< Field >, [93](#)
- FixedPreIntTag, [94](#)
- fsytrf
 - FFPACK, [45](#)
- fsytrf_nonunit
 - FFPACK, [46](#)
- fsytrf_UP_RPM
 - FFPACK, [72](#)
- ftsmLeftUpperNoTransNonUnit< Element >, [94](#)
- ftssyr2k
 - FFPACK, [45](#)
- ftstr
 - FFPACK, [43](#)
- fttrti
 - FFPACK, [42](#)
- fttrtm
 - FFPACK, [43](#)
- GaussJordan
 - ffpack.h, [110](#)
- GenericTag, [95](#)
- getEchelonForm
 - FFPACK, [66](#)
- getEchelonTransform
 - FFPACK, [67](#)
- getReducedEchelonForm
 - FFPACK, [68](#), [69](#)
- getReducedEchelonTransform
 - FFPACK, [69](#)
- getTriangular
 - FFPACK, [64](#), [65](#)
- Interfaces, [27](#)
- Invert
 - FFPACK, [51](#)
- Invert2
 - FFPACK, [52](#)
- IsSingular
 - FFPACK, [56](#)
- LazyTag, [95](#)
- LeadingSubmatrixRankProfiles
 - FFPACK, [61](#)
- LQUPtoInverseOfFullRankMinor
 - FFPACK, [70](#)
- LUdivine
 - FFPACK, [47](#), [72](#)
- MachineFloatTag, [96](#)
- MachineIntTag, [96](#)
- Matrix Multiplication Algorithms, [25](#)
- MatVecMinPoly
 - FFPACK, [55](#)
- MinPoly
 - FFPACK, [54](#), [55](#)
- MMHelper< Field, AlgoTrait, ModeCategories::DefaultTag, ParSeqTrait >, [96](#)
- ModeTraits< Field >, [97](#)
- ModularTag, [97](#)
- MonotonicApplyP
 - FFPACK, [38](#)
- NonZeroRandomMatrix
 - FFPACK, [73](#), [74](#)
- NullSpaceBasis
 - FFPACK, [59](#)
- PLUQ
 - FFPACK, [47](#)
- RandomIndexSubset
 - FFPACK, [80](#)
- RandomKrylovPrecond
 - ffpack.h, [111](#)
- RandomMatrix
 - FFPACK, [74](#), [75](#)
- RandomMatrixWithDet
 - FFPACK, [87](#)
- RandomMatrixWithRank
 - FFPACK, [78](#), [79](#)
- RandomMatrixWithRankandRandomRPM
 - FFPACK, [84](#)
- RandomMatrixWithRankandRPM
 - FFPACK, [81](#), [82](#)
- RandomNullSpaceVector
 - FFPACK, [58](#), [71](#)
- RandomPermutation
 - FFPACK, [80](#)
- RandomRankProfileMatrix
 - FFPACK, [80](#)

- RandomSymmetricMatrix
 - FFPACK, [78](#)
- RandomSymmetricMatrixWithRankandRandomRPM
 - FFPACK, [85](#)
- RandomSymmetricMatrixWithRankandRPM
 - FFPACK, [82](#), [83](#)
- RandomSymmetricRankProfileMatrix
 - FFPACK, [81](#)
- RandomTriangularMatrix
 - FFPACK, [76](#)
- Rank
 - FFPACK, [56](#)
- RankProfileFromLU
 - FFPACK, [60](#)
- read_sparse.h, [113](#)
- ReducedColumnEchelonForm
 - FFPACK, [50](#)
- ReducedRowEchelonForm
 - FFPACK, [50](#)
- RNS, [27](#)
- rns-double-elt.h, [113](#)
- rns-double.h, [114](#)
- rns-integer-mod.h, [114](#)
- rns-integer.h, [115](#)
- rns.h, [115](#)
- RNSElementTag, [98](#)
- RowEchelonForm
 - FFPACK, [49](#)
- RowRankProfile
 - FFPACK, [59](#)
- RowRankProfileSubmatrix
 - FFPACK, [63](#)
- RowRankProfileSubmatrixIndices
 - FFPACK, [62](#)
- schedule_bini.inl, [115](#)
- SIMD wrapper, [26](#)
- Solve
 - FFPACK, [57](#)
- TRSMHelper< ReclterTrait, ParSeqTrait >, [98](#)
- UnparametricTag, [98](#)