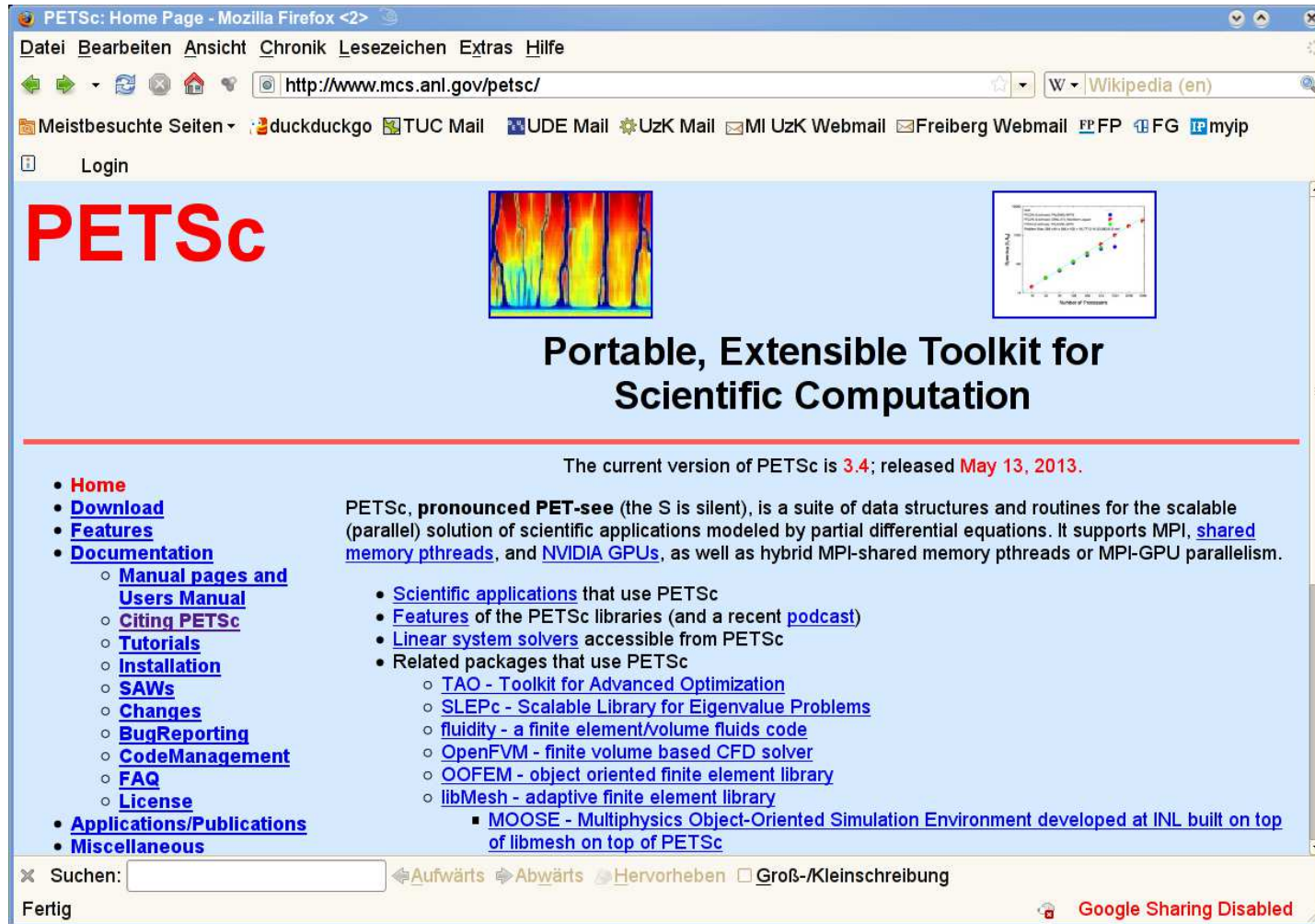


Software Packages for Scientific Computing: PETSc



PETSc: Home Page - Mozilla Firefox <2>

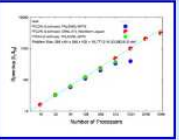
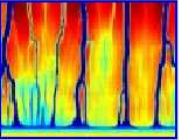
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

http://www.mcs.anl.gov/petsc/

Meistbesuchte Seiten duckduckgo TUC Mail UDE Mail UzK Mail MI UzK Webmail Freiberg Webmail FP FG myip

Login

PETSc



Portable, Extensible Toolkit for Scientific Computation

The current version of PETSc is 3.4; released May 13, 2013.

PETSc, pronounced **PET-see** (the S is silent), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, [shared memory pthreads](#), and [NVIDIA GPUs](#), as well as hybrid MPI-shared memory pthreads or MPI-GPU parallelism.

- [Home](#)
- [Download](#)
- [Features](#)
- [Documentation](#)
 - [Manual pages and Users Manual](#)
 - [Citing PETSc](#)
 - [Tutorials](#)
 - [Installation](#)
 - [SAWs](#)
 - [Changes](#)
 - [BugReporting](#)
 - [CodeManagement](#)
 - [FAQ](#)
 - [License](#)
- [Applications/Publications](#)
- [Miscellaneous](#)

- [Scientific applications](#) that use PETSc
- [Features](#) of the PETSc libraries (and a recent [podcast](#))
- [Linear system solvers](#) accessible from PETSc
- Related packages that use PETSc
 - [TAO - Toolkit for Advanced Optimization](#)
 - [SLEPc - Scalable Library for Eigenvalue Problems](#)
 - [fluidity - a finite element/volume fluids code](#)
 - [OpenFVM - finite volume based CFD solver](#)
 - [OOFEM - object oriented finite element library](#)
 - [libMesh - adaptive finite element library](#)
 - [MOOSE - Multiphysics Object-Oriented Simulation Environment developed at INL built on top of libmesh on top of PETSc](#)

Suchen: Aufwärts Abwärts Hervorheben Groß-/Kleinschreibung

Fertig

Google Sharing Disabled

By Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes Barry F. Smith, and Hong Zhang at Argonne National Laboratory, USA

MPIAJ-Matrices in PETSc

$$\left(\begin{array}{ccc|ccc|cc} 1 & 2 & 0 & 0 & 3 & 0 & 0 & 4 \\ 0 & 5 & 6 & 7 & 0 & 0 & 8 & 0 \\ 9 & 0 & 10 & 11 & 0 & 0 & 12 & 0 \\ \hline 13 & 0 & 14 & 15 & 16 & 17 & 0 & 0 \\ 0 & 18 & 0 & 19 & 20 & 21 & 0 & 0 \\ 0 & 0 & 0 & 22 & 23 & 0 & 24 & 0 \\ \hline 25 & 26 & 27 & 0 & 0 & 28 & 29 & 0 \\ 30 & 0 & 0 & 31 & 32 & 33 & 0 & 34 \end{array} \right)$$

The “diagonal” submatrix, d , on the first process is given by

$$\begin{pmatrix} 1 & 2 & 0 \\ 0 & 5 & 6 \\ 9 & 0 & 10 \end{pmatrix},$$

while the “off-diagonal” submatrix, o , matrix is given by

$$\begin{pmatrix} 0 & 3 & 0 & 0 & 4 \\ 7 & 0 & 0 & 8 & 0 \\ 11 & 0 & 0 & 12 & 0 \end{pmatrix}.$$

MPI-Matrices are distributed row-wise.

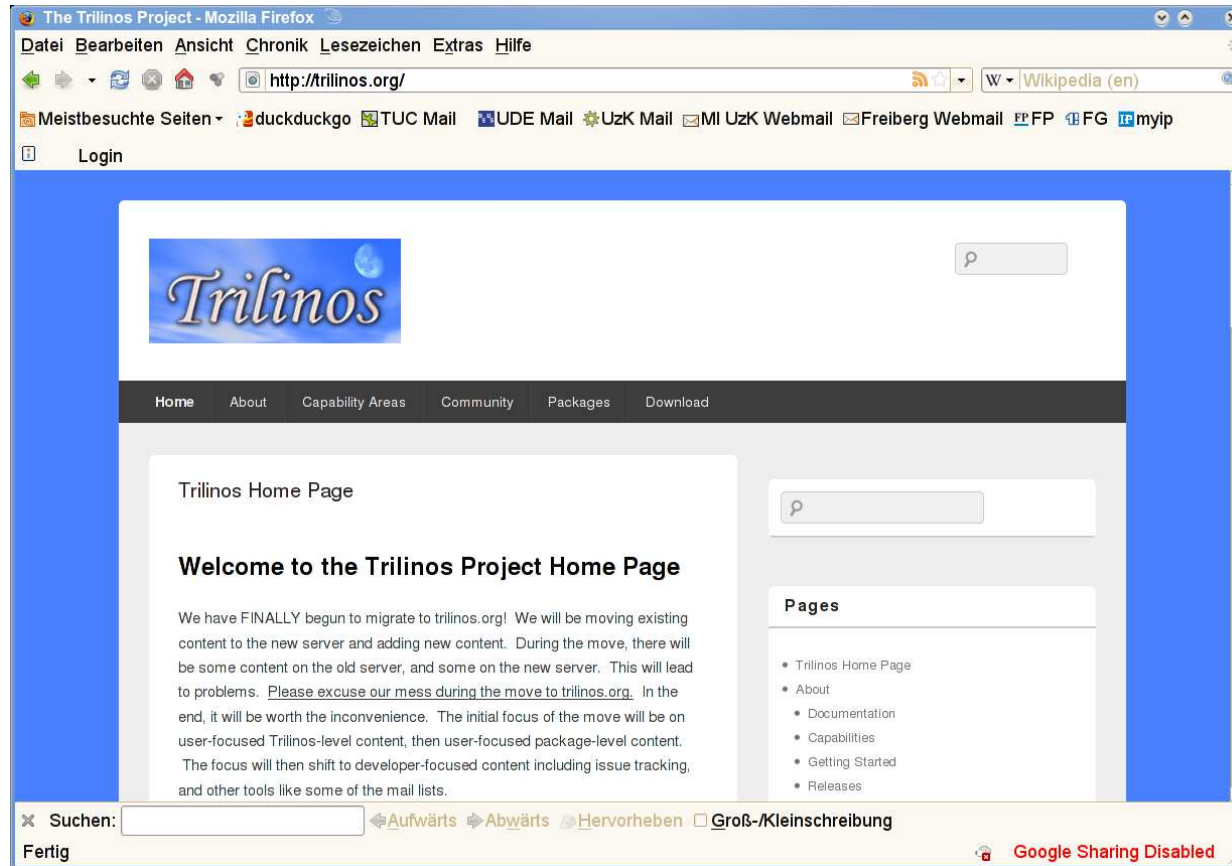
Diagonal part and off-diagonal part is stored seperately.

MPIAJ-Matrices in PETSc

Function Name	Operation
<code>MatAXPY(Mat Y, PetscScalar a, Mat X, MatStructure);</code>	$Y = Y + a * X$
<code>MatMult(Mat A, Vec x, Vec y);</code>	$y = A * x$
<code>MatMultAdd(Mat A, Vec x, Vec y, Vec z);</code>	$z = y + A * x$
<code>MatMultTranspose(Mat A, Vec x, Vec y);</code>	$y = A^T * x$
<code>MatMultTransposeAdd(Mat A, Vec x, Vec y, Vec z);</code>	$z = y + A^T * x$
<code>MatNorm(Mat A, NormType type, double *r);</code>	$r = \ A\ _{type}$
<code>MatDiagonalScale(Mat A, Vec l, Vec r);</code>	$A = \text{diag}(l) * A * \text{diag}(r)$
<code>MatScale(Mat A, PetscScalar a);</code>	$A = a * A$
<code>MatConvert(Mat A, MatType type, Mat *B);</code>	$B = A$
<code>MatCopy(Mat A, Mat B, MatStructure);</code>	$B = A$
<code>MatGetDiagonal(Mat A, Vec x);</code>	$x = \text{diag}(A)$
<code>MatTranspose(Mat A, MatReuse, Mat* B);</code>	$B = A^T$
<code>MatZeroEntries(Mat A);</code>	$A = 0$
<code>MatShift(Mat Y, PetscScalar a);</code>	$Y = Y + a * I$

Matrix operations in PETSc.

Software Packages for Scientific Computing: Trilinos



Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring and Alan Williams, and Kendall S. Stanley from Sandia National Laboratories, USA

<http://trilinos.sandia.gov>

Software Packages for Scientific Computing: Trilinos

- Portable, Extensible Toolkit for Scientific Computation PETSc
- Epetra matrices and vectors from the Epetra Linear Algebra package that is part of Trilinos

<http://trilinos.sandia.gov/packages/epetra/>

Row-wise linear decomposition

$$\left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array} \right] \left[\begin{array}{c} x_1 \\ x_2 \\ \hline x_3 \\ x_4 \end{array} \right]$$

Here, the multiplication

$$\left[\begin{array}{cc} a_{11} & a_{12} \\ a_{21} & a_{22} \end{array} \right] \left[\begin{array}{c} x_1 \\ x_2 \end{array} \right]$$

can be performed locally on processor 0. For

$$\left[\begin{array}{cc} a_{13} & a_{14} \\ a_{23} & a_{24} \end{array} \right] \left[\begin{array}{c} x_3 \\ x_4 \end{array} \right]$$

the vector has to be transferred (by MPI) from processor 1 to processor 0.

Trilinos Epetra Vectors

Epetra has usual sequential vectors

`Epetra_SerialDenseVector`

local to each processor.

The parallel vector implementation is given by

`Epetra_Vector`

which can be distributed among all processors.

Trilinos Epetra Vectors

Entries of Epetra-vectors can be distributed uniquely to the processors or in an overlapping way, i.e., $x^T = (x_1, x_2, x_3, x_4, x_5)$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \hline x_4 \\ x_5 \end{bmatrix}$$

Elements are stored uniquely;

The mapping from local to global can be arbitrary

$$\begin{bmatrix} x_1 \\ x_2 \\ \hline x_3 & x_3 \\ \hline & x_4 \\ & x_5 \end{bmatrix}$$

Elements are duplicated;

communication such as adding duplicate entries, duplicating entries is done by Exporters/Importers

Distribution is defined by a map.

Trilinos Epetra Vector

Declaring a Vector:

```
// Comm is MPI_COMM_WORLD
int NumElements;
// ...

// Construct a linear Map with NumElements and index base of 0
Epetra_Map Map(NumElements, 0, Comm);

// Create vectors x and b
Epetra_Vector x(Map); // allocates space and sets all the elements to zero
Epetra_Vector b(Map);
```

An Epetra vector has to be constructed using a Map defining the parallel distribution.

Transfer from a Vector defined using one map to a Vector defined using a different map is done using `Epetra_Import` and `Epetra_Export` classes.

Trilinos Epetra Vector

Filling Data into a Vector:

```
// Construction can be performed by:  
Epetra_Vector y(x); // Use Copy Constructor  
  
double LocalValues[]={1.0,2.0,3.1}  
  
Epetra_Vector xa(Copy,Map,LocalValues); // Copy from local array LocalValues  
  
Epetra_Vector xb(View,Map,LocalValues); // Use local array LocalValues  
  
xa[0] = 2.0;
```

Trilinos Epetra FEVector

```
Epetra_FEVector vv(Map); // derived type
```

```
// Can do
```

```
//int SumIntoGlobalValues (int numIndices, const int *indices, const double *values,  
                           int format=Epetra_FECrsMatrix::COLUMN_MAJOR)
```

```
// instead of only
```

```
//int SumIntoGlobalValues (int NumEntries, double *Values, int *Indices)
```

Trilinos Epetra Maps

```
int NumGlobalElements;  
int NumMyElements;  
int MyGlobalElements [i]; // global indexing vector
```

Three ways to define a map:

a) take global number, base index and comm

```
Epetra_Map Map(NumGlobalElements,0,Comm);
```

b) take local number,

```
Epetra_Map Map(-1,NumMyElements,0,Comm);
```

c) a more general case.

```
Epetra_Map Map(-1,MyElements,MyGlobalElements,0,Comm);
```

Different maps may coexist even with same elements, element numbers are only labels!?

Trilinos Epetra Maps: Example (didasco, ex3.cpp)

User-defined arbitrary distribution of elements: Each process both the number of local elements, and the global indexing of each local element.

```
#include "Epetra_Map.h"
MyPID = Comm.MyPID();
switch( MyPID ) {
case 0:  MyElements = 2;
        MyGlobalElements = new int[MyElements];
        MyGlobalElements[0] = 0;
        MyGlobalElements[1] = 4;
        break;
case 1:  MyElements = 3;
        MyGlobalElements = new int[MyElements];
        MyGlobalElements[0] = 1;
        MyGlobalElements[1] = 2;
        MyGlobalElements[2] = 3;
        break;
}
Epetra_Map Map(-1,MyElements,MyGlobalElements,0,Comm);
```

Epetra_Map (int NumGlobalElements, int NumMyElements, int *MyGlobalElements,
int IndexBase, const Epetra_Comm &Comm)

NumGlobalElements=-1 \Rightarrow NumGlobalElements will be computed automatically.

Usage of Maps: Example (didasco, ex3.cpp)

```
// Querying a Map for the number of local and global elements
```

```
int NumGlobalElements = Map.NumGlobalElements();
```

```
int NumMyElements = Map.NumMyElements();
```

```
// Obtaining global IDs of the local elements as an array
```

```
int * MyGlobalElements = Map.MyGlobalElements();
```

Importers and Exporters

Importers and Exporters are used to perform communication operations for the distributed data.

Exporters let the user specify how to combine distributed data that has the same global index.

For example, one may replace old data with new data or sum them together. Such operations are also called Scatter (=Import) and Gather (=Export) operations.

Usage of Exporters: Example (didasco, ex3.cpp)

Gather operation requires an Exporter

```
Epetra_Vector x(Map);          // x constructed using Map
Epetra_Vector y(TargetMap);    // y constructed using TargetMap
//...
Epetra_Export Exporter(Map,TargetMap);

y.Export(x,Exporter,Add);
cout << y;
```

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \hline x_3 + x_3 \\ \hline x_4 \\ x_5 \end{bmatrix}$$

The inverse (scatter) operation uses an Importer

Import = from a uniquely owned to a possibly not uniquely owned Map.

Export = from a possibly not uniquely owned to a uniquely owned Map

Epetra Matrices

An Epetra matrix one Map to determine the row distribution and one Map to define the column distribution.

Here, the row distribution is unique whereas the column distribution has an overlap.

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \color{red}{a_{13}} & & \\ a_{21} & a_{22} & \color{red}{a_{23}} & & \\ a_{31} & a_{32} & \color{red}{a_{33}} & & \\ \hline & & \color{red}{a_{43}} & a_{44} & a_{45} \\ & & \color{red}{a_{53}} & a_{54} & a_{55} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \color{red}{y_3} \\ y_4 \\ y_5 \end{bmatrix}$$

If y uses the column Map then multiplications can now be performed locally:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \color{red}{a_{13}} \\ a_{21} & a_{22} & \color{red}{a_{23}} \\ a_{31} & a_{32} & \color{red}{a_{33}} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \color{red}{y_3} \end{bmatrix}$$

$$\begin{bmatrix} x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} \color{red}{a_{43}} & a_{44} & a_{45} \\ \color{red}{a_{53}} & a_{54} & a_{55} \end{bmatrix} \begin{bmatrix} \color{red}{y_3} \\ y_4 \\ y_5 \end{bmatrix}$$

Epetra Matrices

More precisely four Maps have to be provided to an Epetra Matrix to use it in Matrix-Vector multiplications, i.e. the Maps of the source and the target Vectors have to be provided.

```
// Assemble Matrix A
for( int i=0 ; i<NumElementsA ; ++i ) {
    double one = 2.0;
    int indices = MyGlobalElementsA[i];
    A.InsertGlobalValues(MyGlobalElementsA[i], 1, &one, &indices );
}
```

```
Epetra_Vector x(SourceVectorMap);
Epetra_Vector y(DestinationVectorMap);
```

```
A.FillComplete(SourceVectorMap, DestinationVectorMap);
```

```
A.Multiply(false, x, y);
```

Library will perform all necessary communication during the multiply.

Epetra Matrices

In Epetra it is also possible that the complete matrix A is replicated among the processors, i.e., the row Map of all processors contains all indices and the column Map as well.

In a multiplication with a vector x the matrix will still (usually) work as expected, i.e., will yield a correct result of a matrix-vector multiplication.

An example where this is not the case is the following: Let the destination y vector also be replicated among all p processors. Then $A * x$ will result in $p * A * x$ because an Export-Add followed by an Import will be performed to generate the replicated entries of the result y .

Epetra Matrices: Exporters/Importers

Similarly to the use with Vectors Exporters and Importers can also be used with Matrices.

Warnung: Using Export for scatter operations (instead of Import) may lead to unexpected results:

Import will replicate entries of the matrix if the Map is overlapping. Multiplication with a vector will result in $\#Multiplicity \times \text{value}$.

Export will NOT replicate the values, i.e. the value will be set on local processor, all others will be set to 0. As a result in a matrix-vector multiplication the expected result will be obtained.

In duplated values in matrices or vectors the Matrix-vector multiplication will assume that the local values are correct and compute with these.

Amesos Solver Wrapper

The Trilinos Amesos is a wrapper to Umfpack, MUMPS, Pardiso, DSCPACK, (SuperLU, SuperLUdist, KLU, Lapack). Expects an `Epetra_Row_Matrix` (pure virtual).

Uses multivectors for X and r.h.s. B For the sequential packages data will be send to processor 0, solved and broadcast to all.

```
Epetra_LinearProblem Problem(&A, &X, &B);  
  
Amesos_BaseSolver *Solver;  
Amesos Factory;  
Solver=Factory.Create("Amesos_Klu",Problem);
```

BLAS and LAPACK Libraries

- BLAS and LAPACK are single processor, single core routines.
- BLAS (Basic Linear Algebra Subprograms) date back to 1976.

Idea:

- Hardware vendors should provide optimized implementations of vector operations, especially for vector processors.
- BLAS implements vector functions, i.e., vectors operations.
- LAPACK implements more complex operations building upon BLAS routines.

Implementations of BLAS and LAPACK Libraries

Vendor provided:

AMD	ACML (AMD Core Math Library)
Apple	Accelerate framework
Compaq	CXML
Cray	libsci
HP	MLIB
IBM	ESSL (Engineering and Scientific Subroutine Library)
Intel	MKL (Math Kernel Library)
NEC	PDLIB/SX
SGI	SCSL (Scientific Computing Software Library)
SUN	Sun Performance Library

Other:

ATLAS	Autotuning library
Goto Blas	Optimized for different processors (development discontinued)

BLAS Routines Naming Convention

- Prefix shows data type (s: single precision; d: double precision; c: complex), e.g.

`saxpy(n,alpha,x,incx, y, incy)`

`daxpy(n,alpha,x,incx, y, incy)`

`caxpy(n,alpha,x,incx, y, incy)`

- Suffix if there are versions (u: unconjugated complex dot product; c: conjugated complex dot product).

`cdotc(n,x,incx,y,incy)`

`cdotu(n,x,incx,y,incy)`

`incx, incy` are the increments for the vectors – usually 1.

Manpages

Manpages for BLAS and LAPACK are installed on many Linux system:

NAME

SAXPY - BLAS level one axpy subroutine

SYNOPSIS

```
SUBROUTINE SAXPY      ( n, alpha, x, incx, y, incy )  
    INTEGER            n, incx, incy  
    REAL               alpha, x, y
```

DESCRIPTION

SAXPY adds a scalar multiple of a real vector to another real vector. SAXPY computes a constant alpha times a vector x plus a vector y. The result overwrites the initial values of vector y.

This routine performs the following vector operation:

$$y \leftarrow \alpha * x + y$$

incx and incy specify the increment between two consecutive elements of respectively vector x and y.

ARGUMENTS

n	INTEGER. (input)	Number of elements in the vectors. If $n \leq 0$, these routines return without any computation.
alpha	REAL. (input)	If $\alpha = 0$ this routine returns without any computation.

x REAL. (input)
 Array of dimension $(n-1) * |incx| + 1$. Contains the vector to
 be scaled before summation.

incx INTEGER. (input) Increment between elements of x.
 If $incx = 0$, the results will be unpredictable.

y REAL. (input and output)
 array of dimension $(n-1) * |incy| + 1$.
 Before calling the routine, y contains the vector to be summed.
 After the routine ends, y contains the result of the summation.

incy INTEGER. (input) Increment between elements of y.
 If $incy = 0$, the results will be unpredictable.

NOTES

This routine is Level 1 Basic Linear Algebra Subprograms (Level 1 BLAS).

When working backward ($incx < 0$ or $incy < 0$), each routine starts at the end of the vector and moves backward, as follows:

$x(1-incx * (n-1)), x(1-incx * (n-2)), \dots, x(1)$
 $y(1-incy * (n-1)), y(1-incy * (n-2)), \dots, y(1)$

RETURN VALUES

When $n \leq 0$, real $\alpha = 0.$, this routine returns immediately with no change in its arguments.

BLAS Level 1 Routines (blasqr.pdf)

Generate plane rotation

Generate modified plane rotation

Apply plane rotation

Apply modified plane rotation

$$x \leftrightarrow y$$

$$x \leftarrow \alpha x$$

$$y \leftarrow x$$

$$y \leftarrow \alpha x + y$$

$$dot \leftarrow x^T y$$

$$dot \leftarrow x^T y$$

$$dot \leftarrow x^H y$$

$$dot \leftarrow \alpha + x^T y$$

$$nrm2 \leftarrow \|x\|_2$$

$$asum \leftarrow \|re(x)\|_1 + \|im(x)\|_1$$

$$\begin{aligned} amax &\leftarrow 1^{st} k \ni |re(x_k)| + |im(x_k)| \\ &= \max(|re(x_i)| + |im(x_i)|) \end{aligned}$$

BLAS Level 1 Routines

Level 1 BLAS

	dim	scalar	vector	vector	scalars	5-element array
SUBROUTINE xROTG (A, B, C, S)	
SUBROUTINE xROTMG(D1, D2, A, B,	PARAM)
SUBROUTINE xROT (N,			X, INCX, Y, INCY,		C, S)	
SUBROUTINE xROTM (N,			X, INCX, Y, INCY,			PARAM)
SUBROUTINE xSWAP (N,			X, INCX, Y, INCY)			
SUBROUTINE xSCAL (N,	ALPHA,		X, INCX)			
SUBROUTINE xCOPY (N,			X, INCX, Y, INCY)			
SUBROUTINE xAXPY (N,	ALPHA,		X, INCX, Y, INCY)			
FUNCTION xDOT (N,			X, INCX, Y, INCY)			
FUNCTION xDOTU (N,			X, INCX, Y, INCY)			
FUNCTION xDOTC (N,			X, INCX, Y, INCY)			
FUNCTION xxDOT (N,			X, INCX, Y, INCY)			
FUNCTION xNRM2 (N,			X, INCX)			
FUNCTION xASUM (N,			X, INCX)			
FUNCTION IxAMAX(N,			X, INCX)			

BLAS Level 2 Routines

$$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$$

$$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$$

$$y \leftarrow \alpha Ax + \beta y$$

$$y \leftarrow \alpha Ax + \beta y$$

$$y \leftarrow \alpha Ax + \beta y$$

$$y \leftarrow \alpha Ax + \beta y$$

$$y \leftarrow \alpha Ax + \beta y$$

$$y \leftarrow \alpha Ax + \beta y$$

$$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$$

$$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$$

$$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$$

$$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$$

$$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$$

$$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$$

$$A \leftarrow \alpha xy^T + A, A - m \times n$$

$$A \leftarrow \alpha xy^T + A, A - m \times n$$

$$A \leftarrow \alpha xy^H + A, A - m \times n$$

$$A \leftarrow \alpha xx^H + A$$

$$A \leftarrow \alpha xx^H + A$$

$$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$$

$$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$$

$$A \leftarrow \alpha xx^T + A$$

$$A \leftarrow \alpha xx^T + A$$

$$A \leftarrow \alpha xy^T + \alpha yx^T + A$$

$$A \leftarrow \alpha xy^T + \alpha yx^T + A$$

BLAS Level 2 Routines

Level 2 BLAS

options	dim	b-width	scalar	matrix	vector	scalar	vector
xGEMV (TRANS,	M, N,		ALPHA, A, LDA, X, INCX, BETA, Y, INCY)				
xGBMV (TRANS,	M, N, KL, KU,		ALPHA, A, LDA, X, INCX, BETA, Y, INCY)				
xHEMV (UPLO,	N,		ALPHA, A, LDA, X, INCX, BETA, Y, INCY)				
xHBMV (UPLO,	N, K,		ALPHA, A, LDA, X, INCX, BETA, Y, INCY)				
xHPMV (UPLO,	N,		ALPHA, AP, X, INCX, BETA, Y, INCY)				
xSYMV (UPLO,	N,		ALPHA, A, LDA, X, INCX, BETA, Y, INCY)				
xSBMV (UPLO,	N, K,		ALPHA, A, LDA, X, INCX, BETA, Y, INCY)				
xSPMV (UPLO,	N,		ALPHA, AP, X, INCX, BETA, Y, INCY)				
xTRMV (UPLO, TRANS, DIAG,	N,		A, LDA, X, INCX)				
xTBMV (UPLO, TRANS, DIAG,	N, K,		A, LDA, X, INCX)				
xTPMV (UPLO, TRANS, DIAG,	N,		AP, X, INCX)				
xTRSV (UPLO, TRANS, DIAG,	N,		A, LDA, X, INCX)				
xTBSV (UPLO, TRANS, DIAG,	N, K,		A, LDA, X, INCX)				
xTPSV (UPLO, TRANS, DIAG,	N,		AP, X, INCX)				
options	dim	scalar	vector	vector	matrix		
xGER (M, N,	ALPHA, X, INCX, Y, INCY, A, LDA)					
xGERU (M, N,	ALPHA, X, INCX, Y, INCY, A, LDA)					
xGERC (M, N,	ALPHA, X, INCX, Y, INCY, A, LDA)					
xHER (UPLO,	N,	ALPHA, X, INCX,		A, LDA)			
xHPR (UPLO,	N,	ALPHA, X, INCX,		AP)			
xHER2 (UPLO,	N,	ALPHA, X, INCX, Y, INCY, A, LDA)					
xHPR2 (UPLO,	N,	ALPHA, X, INCX, Y, INCY, AP)					
xSYR (UPLO,	N,	ALPHA, X, INCX,		A, LDA)			
xSPR (UPLO,	N,	ALPHA, X, INCX,		AP)			
xSYR2 (UPLO,	N,	ALPHA, X, INCX, Y, INCY, A, LDA)					
xSPR2 (UPLO,	N,	ALPHA, X, INCX, Y, INCY, AP)					

BLAS Level 3 Routines

$$C \leftarrow \alpha op(A)op(B) + \beta C, op(X) = X, X^T, X^H, C - m \times n$$

$$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^T$$

$$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^H$$

$$C \leftarrow \alpha AA^T + \beta C, C \leftarrow \alpha A^T A + \beta C, C - n \times n$$

$$C \leftarrow \alpha AA^H + \beta C, C \leftarrow \alpha A^H A + \beta C, C - n \times n$$

$$C \leftarrow \alpha AB^T + \bar{\alpha} BA^T + \beta C, C \leftarrow \alpha A^T B + \bar{\alpha} B^T A + \beta C, C - n \times n$$

$$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C, C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C, C - n \times n$$

$$B \leftarrow \alpha op(A)B, B \leftarrow \alpha Bop(A), op(A) = A, A^T, A^H, B - m \times n$$

$$B \leftarrow \alpha op(A^{-1})B, B \leftarrow \alpha Bop(A^{-1}), op(A) = A, A^T, A^H, B - m \times n$$

BLAS Level 3 Routines

Level 3 BLAS

	options		dim	scalar	matrix	matrix	scalar	matrix
xGEMM (TRANSA, TRANSB,	M, N, K,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)
xSYMM (SIDE, UPLO,		M, N,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)
xHEMM (SIDE, UPLO,		M, N,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)
xSYRK (UPLO, TRANS,		N, K,	ALPHA,	A, LDA,		BETA,	C, LDC)
xHERK (UPLO, TRANS,		N, K,	ALPHA,	A, LDA,		BETA,	C, LDC)
xSYR2K(UPLO, TRANS,		N, K,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)
xHER2K(UPLO, TRANS,		N, K,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)
xTRMM (SIDE, UPLO, TRANSA,	DIAG, M, N,		ALPHA,	A, LDA,	B, LDB)		
xTRSM (SIDE, UPLO, TRANSA,	DIAG, M, N,		ALPHA,	A, LDA,	B, LDB)		

Memory references and floating point operations

	Read	Write	Flops	Flops/memory access
ddot	$2n$	1	$2n$	1
daxpy	$2n$	n	$2n$	$2/3$
dgemv	$n^2 + n$	n	$2n^2$	2
dger	$n^2 + 2n$	n^2	$2n^2$	1
dgemm	$2n^2$	n^2	$2n^3$	$2n/3$

LAPACK Routines

Dense Linear Algebra Routines for **solving** problems. Naming is pmmaaa,

- p Prefix
 - ★ S: single precision
 - ★ D: Double Precision
 - ★ C: Complex
- mm Matrix type (28 types)
 - ★ GE: General Matrix
 - ★ GB: Band Matrix
 - ★ SY: Symmetric Matrix
 - ★ SP: Symmetric Packed Storage
 - ★ ...
- aaa Algorithm
 - ★ SV: Solve linear system
 - ★ EVD: Compute Eigenvalues
 - ★ SVD: Compute Singular Values
 - ★ LSY: Solve Least Squares Problem
 - ★ ...

LAPACK Routines

Routines for

- Solving (dense) linear equations
 - ★ General: DGESV, SGESV
 - ★ Banded: DGBSV, SGBSV
 - ★ Tridiagonal: DGTSV, SGTSV
- Solving (dense) symmetric/hermitian positive definite (general DPOSV, band DGPSV, tridiagonal DGBSV)
- Solving (dense) symmetric/hermitian indefinite
 - ★ General: DSYSV, SSYSV
 - ★ Banded: DSPSV), SSPSV)
- Solving Least Squares Problems
- Solving Eigenvalue und Singular Value problems.
- Solving Generalized Eigenvalue und Singular Value problems.