xtensor-blas

Wolf Vollprecht, Johan Mabille and Sylvain Corlay

Dec 17, 2019

INSTALLATION

xtensor bindings for BLAS and LAPACK

`xtensor-blas` is an extension to the xtensor library, offering bindings to BLAS and LAPACK libraries through cxxblas and cxxlapack from the FLENS project.

`xtensor-blas` currently provides non-broadcasting `dot`, `norm` (1- and 2-norm for vectors), `inverse`, `solve`, `eig`, `cross`, `det`, `slogdet`, `matrix_rank`, `inv`, `cholesky`, `qr`, `svd` in the `xt::linalg` namespace (check the corresponding `xlinalg.hpp` header for the function signatures). The functions, and signatures, are trying to be 1-to-1 equivalent to NumPy. Low-level functions to interface with BLAS or LAPACK with xtensor containers are also offered in the blas and lapack namespace.

`xtensor` and `xtensor-blas` require a modern C++ compiler supporting C++14. The following C++ compilers are supported:

- On Windows platforms, Visual C++ 2015 Update 2, or more recent
- On Unix platforms, gcc 4.9 or a recent version of Clang

CHAPTER

ONE

LICENSING

We use a shared copyright model that enables all contributors to maintain the copyright on their contributions.

This software is licensed under the BSD-3-Clause license. See the LICENSE file for details.

1.1 Installation

Although `xtensor-blas` is a header-only library, we provide standardized means to install it, with package managers or with cmake.

Besides the xtendor headers, all these methods place the *cmake* project configuration file in the right location so that third-party projects can use cmake's find_package to locate xtensor headers.

See also:

Read the *Performance and Linking* chapter on how to link against BLAS and improve performance

1.1.1 Using the conda package

A package for xtensor-blas is available on the conda package manager.

```
conda install -c conda-forge xtensor-blas
```

1.1.2 From source with cmake

You can install `xtensor-blas` from source with cmake. Note that you need to have a BLAS installation available (e.g. OpenBLAS, MKL . . .). On Unix platforms, from the source directory:

```
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=/path/to/prefix ..
make install
```

On Windows platforms, from the source directory:

```
mkdir build
cd build
cmake -G "NMake Makefiles" -DCMAKE_INSTALL_PREFIX=/path/to/prefix ..
nmake
nmake install
```

1.2 Performance and Linking

For optimal performance, the target program **has** to be linked against a BLAS implementation. The BLAS implementation that we install by default with conda is `OpenBLAS`, but other options, such as `MKL` are available on conda, too. If xtensor-blas was not installed from conda, the user has to manually verify that a BLAS and LAPACK implementation is available. If you want to fallback to a slower, more generic BLAS implementation, you can use the compile time define `-DXTENSOR_USE_FLENS_BLAS`.

In order to link against `OpenBLAS` from CMake, the following lines have to be added to the `CMakeLists.txt` file.

```
add_definitions(-DHAVE_CBLAS=1)

if (WIN32)
    find_package(OpenBLAS REQUIRED)
    set(BLAS_LIBRARIES ${CMAKE_INSTALL_PREFIX}${OpenBLAS_LIBRARIES})
else()
    find_package(BLAS REQUIRED)
    find_package(LAPACK REQUIRED)
endif()

message(STATUS "BLAS VENDOR:    " ${BLA_VENDOR})
message(STATUS "BLAS LIBRARIES: " ${BLAS_LIBRARIES})

target_link_libraries(your_target_name ${BLAS_LIBRARIES} ${LAPACK_LIBRARIES})
```

If CMake is not used, the flags can be passed manually to e.g. `g++`:

```
g++ test.cpp -o test -lblas -llapack -DHAVE_CBLAS=1
```

1.3 Usage

To use xtensor-blas functions, the `xlinalg.hpp` header has to be included. In the `xt::linalg` namespace, many of NumPy's `np.linalg` functions are implemented. We make an effort to keep the interfaces very similar.

For example, calculating a determinant:

```
#include "xtensor-blas/xlinalg.hpp"

int main()
{
    xt::xarray<double> a = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    auto d = xt::linalg::det(a);
    std::cout << d << std::endl;  // 6.661338e-16
}
```

We can also try to compute the same determinant using the `slogdet` function, which is more robust against under- or overflows by summing up the logarithm. The slogdet function in NumPy returns a tuple of (sign, val). In C++, we emulate the behaviour by returning a `std::tuple`, which can be unpacked using `std::get<N>(tuple)`.

```
xt::xarray<double> a = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
auto d = xt::linalg::slogdet(a);
std::cout << std::get<0>(d) << ", " << std::get<1>(d) << std::endl;  // 1, -34.9450...
```

Returning tuples is used throughout the xlinalg package.

1.3.1 Using xblas and xlapack directly

It is not necessarily recommended to use `xblas` or `xlapack` directly, but it's possible and can improve performance in certain cases. Some things have to be taken into consideration: For one thing, the result container needs to be allocated and passed into the function beforehand. And for the LAPACK functions, all arguments have to be in column_major order. Furthermore it's required that the xexpressions are evaluated and are stored in contiguous memory. All of this is taken care of when using `xlinalg`.

1.4 Reference

Defined in `xtensor-blas/xlinalg.hpp`

The functions here are closely modeled after NumPy's linalg package.

1.4.1 Matrix, vector and tensor products

template<class **T**, class **O**>
auto *xt*`::linalg::`**dot**(**const** xexpression<*T*> &*xt*, **const** xexpression<*O*> &*xo*)

Non-broadcasting dot function.

In the case of two 1D vectors, computes the vector dot product. In the case of complex vectors, computes the dot product without conjugating the first argument. If *t* or *o* is a 2D matrix, computes the matrix-times-vector product. If both *t* and *o* ar 2D matrices, computes the matrix-product.

Return resulting array

Parameters

- `t`: input array
- `o`: input array

template<class **T**, class **O**>
auto `xt::linalg::`**vdot**(**const** xexpression<*T*> &*a*, **const** xexpression<*O*> &*b*)

Computes the dot product for two vectors.

Behaves different from dot in the case of complex vectors. If vectors are complex, vdot conjugates the first argument *t*. Note: Unlike NumPy, xtensor-blas currently doesn't flatten the input arguments.

Return resulting array

Parameters

- `t`: input vector (1D)

- `o`: input vector (1D)

template<class **T**, class **O**>
auto `xt::linalg::outer`(**const** xexpression<*T*> &*a*, **const** xexpression<*O*> &*b*)
Compute the outer product of two vectors.

Return resulting array

Parameters

- `t`: input vector (1D)
- `o`: input vector (1D)

template<class **E**>
auto `xt::linalg::matrix_power`(**const** xexpression<*E*> &*A*, long *n*)
Calculate matrix power A**n.

Return resulting array

Parameters

- `mat`: The matrix
- `n`: The exponent

template<class **T**, class **E**>
auto `xt::linalg::kron`(**const** xexpression<*T*> &*a*, **const** xexpression<*E*> &*b*)
Calculate the Kronecker product between two 2D xexpressions.

template<class **T**, class **O**>
auto `xt::linalg::tensordot`(**const** xexpression<*T*> &*xa*, **const** xexpression<*O*> &*xb*, std::size_t *naxes* = 2)
Compute tensor dot product along specified axes for arrays.

Compute the sum of products along the last *naxes* axes of a and first *naxes* axes of b.

Return resulting array

Parameters

- `xa`: input array
- `xb`: input array
- `naxes`: the number of axes to sum over

template<class **T**, class **O**>
auto `xt::linalg::tensordot`(**const** xexpression<*T*> &*xa*, **const** xexpression<*O*> &*xb*, **const** std::vector<std::size_t> &*ax_a*, **const** std::vector<std::size_t> &*ax_b*)
Compute tensor dot product along specified axes for arrays.

Compute the sum of products along the axes *ax_a* for a and *ax_b* for b

Return resulting array

Parameters

- `xa`: input array
- `xb`: input array
- `ax_a`: axes to sum over for *a*

- `ax_b`: axes to sum over for *b*

1.4.2 Decompositions

template<class **T**>
auto `xt::linalg::`**`cholesky`**`(`**`const`** xexpression<*T*> &*A*)
Compute the Cholesky decomposition of *A*.

Return the decomposed matrix

enum `xt::linalg::`**`qrmode`**
Select the mode for the qr decomposition `K = min(M, K)`

Values:

`reduced`
return Q, R with dimensions (M, K), (K, N) (default)

`complete`
return Q, R with dimensions (M, M), (M, N)

`r`
return empty Q and R with dimensions (0, 0), (K, N)

`raw`
return H, Tau with dimensions (N, M), (K, 1)

template<class **T**>
auto `xt::linalg::`**`qr`**`(`**`const`** xexpression<*T*> &*A*, *qrmode* *mode* = *qrmode*::reduced)
Compute the QR decomposition of *A*.

Return std::tuple with Q and R

Parameters

- `t`: The matrix to calculate Q and R for

template<class **T**>
auto `xt::linalg::`**`svd`**`(`**`const`** xexpression<*T*> &*A*, bool *full_matrices* = true, bool *compute_uv* = true)
Compute the SVD decomposition of *A*.

Return tuple containing S, V, and D

1.4.3 Matrix eigenvalues

template<class **E**, std::enable_if_t<!xtl::is_complex<**`typename`** *E*::value_type>::value> * = nullptr>
auto `xt::linalg::`**`eig`**`(`**`const`** xexpression<*E*> &*A*)
Compute the eigenvalues and right eigenvectors of a square array.

Return (eigenvalues, eigenvectors) tuple. The first element corresponds to the eigenvalues, each repeated according to its multiplicity. The eigenvalues are not necessarily ordered. The second (1) element are the normalized (unit "length") eigenvectors, such that the column v[:, i] corresponds to the eigenvalue w[i].

Parameters

- `Matrix`: for which the eigenvalues and right eigenvectors will be computed

template<class **E**, std::enable_if_t<!xtl::is_complex<**typename** *E*::value_type>::value> * = nullptr>
auto `xt::linalg::eigvals`(**const** xexpression<*E*> &*A*)
Compute the eigenvalues of a square xexpression.

Return xtensor containing the eigenvalues.

Parameters

- `A`: Matrix for which the eigenvalues are computed

Warning: doxygenfunction: Unable to resolve multiple matches for function "xt::linalg::eigh" with arguments () in doxygen xml output for project "xtensor-blas" from directory: ../xml. Potential matches:

```
- template<class E, std::enable_if_t<!xtl::is_complex<typename E::value_type>
↪::value> * = nullptr>
  auto xt::linalg::eigh(const xexpression<E>&, char)
- template<class E, std::enable_if_t<!xtl::is_complex<typename E::value_type>
↪::value> * = nullptr>
  auto xt::linalg::eigh(const xexpression<E>&, const xexpression<E>&, const char)
```

template<class **E**, std::enable_if_t<!xtl::is_complex<**typename** *E*::value_type>::value> * = nullptr>
auto `xt::linalg::eigvalsh`(**const** xexpression<*E*> &*A*, char *UPLO* = 'L')
Compute the eigenvalues of a Hermitian or real symmetric matrix xexpression.

Return xtensor containing the eigenvalues.

Parameters

- `Matrix`: for which the eigenvalues are computed

1.4.4 Norms and other numbers

enum `xt::linalg::normorder`
Selects special norm orders.

Values:

frob
Frobenius norm.

nuc
Nuclear norm.

inf
Positive infinity norm.

neg_inf
Negative infinity norm.

template<class **E**>
auto `xt::linalg::norm`(**const** xexpression<*E*> &*vec*, int *ord*)
Calculate norm of vector, or matrix.

Return scalar result

Parameters

- `vec`: input vector

- `ord`: order of norm. This can be any integer for a vector or [-2,-1,1,2] for a matrix.

Template Parameters

- `type`: of xexpression

template<class **E**>
auto `xt::linalg::norm`(**const** xexpression<*E*> &*vec*, *normorder* *ord*)
Calculate matrix or vector norm using normorder.

Return norm value

Parameters

- `vec`: The matrix or vector to take the norm of
- `ord`: normorder (frob, nuc, inf, neg_inf)

template<class **E**>
E::value_type `xt::linalg::norm`(**const** xexpression<*E*> &*vec*)
Calculate default norm (2-norm for vector, Frobenius norm for matrix)

Return norm

Parameters

- `vec`: Input vector or matrix

template<class **T**>
auto `xt::linalg::det`(**const** xexpression<*T*> &*A*)
Compute the determinant by utilizing LU factorization.

Return determinant of the *A*

Parameters

- `A`: matrix for which determinant is to be computed

template<class **T**, std::enable_if_t<xtl::is_complex<**typename** *T*::value_type>::value, int> = 0>
auto `xt::linalg::slogdet`(**const** xexpression<*T*> &*A*)
Compute the sign and (natural) logarithm of the determinant of an xexpression.

If an array has a very small or very large determinant, then a call to det may overflow or underflow. This routine is more robust against such issues, because it computes the logarithm of the determinant rather than the determinant itself.

Return tuple containing (sign, determinant)

Parameters

- `A`: matrix for which determinant is to be computed

template<class **T**>
int `xt::linalg::matrix_rank`(**const** xexpression<*T*> &*m*, double *tol* = -1.0)
Calculate the matrix rank of m.

If tol == -1, the tolerance is automatically computed.

Parameters

- `m`: matrix for which rank is calculated

- `tol`: tolerance for finding rank

template<class **T**>
auto `xt::linalg::`**`trace`**(**const** xexpression<*T*> &*M*, int *offset* = 0, int *axis1* = 0, int *axis2* = 1)
Compute the trace of a xexpression.

1.4.5 Solving equations and inverting matrices

template<class **E1**, class **E2**>
auto `xt::linalg::`**`solve`**(**const** xexpression<*E1*> &*A*, **const** xexpression<*E2*> &*b*)
Solve a linear matrix equation, or system of linear scalar equations.

Computes the "exact" solution, x, of the well-determined, i.e., full rank, linear matrix equation ax = b.

Return Solution to the system a x = b. Returned shape is identical to b.

Parameters

- `a`: Coefficient matrix
- `b`: Ordinate or "dependent variable" values.

template<class **T**, class **E**>
auto `xt::linalg::`**`lstsq`**(**const** xexpression<*T*> &*A*, **const** xexpression<*E*> &*b*, double *rcond* = -1.0)
Calculate the least-squares solution to a linear matrix equation.

Return tuple containing (x, residuals, rank, s) where: *x* is the least squares solution. Note that the solution is always returned as a 2D matrix where the columns are the solutions (even for a 1D *b*). *s* Sums of residuals; squared Euclidean 2-norm for each column in b - a*x. If the rank of *A* is < N or M <= N, this is an empty xtensor. *rank* the rank of *A* *s* singular values of *A*

Parameters

- `A`: coefficient matrix
- `b`: Ordinate, or dependent variable values. If b is two-dimensional, the least-squares solution is calculated for each of the K columns of b.
- `rcond`: Cut-off ratio for small singular values of *A*. For the purposes of rank determination, singular values are treated as zero if they are smaller than rcond times the largest singular value of a.

template<class **E1**>
auto `xt::linalg::`**`inv`**(**const** xexpression<*E1*> &*A*)
Compute the (multiplicative) inverse of a matrix.

Return (Multiplicative) inverse of the matrix a.

Parameters

- `A`: xexpression to be inverted

template<class **T**>
auto `xt::linalg::`**`pinv`**(**const** xexpression<*T*> &*A*, double *rcond* = 1e-15)
Calculate Moore-Rose pseudo inverse using LAPACK SVD.

1.4.6 Other

template<class **E1**, class **E2**>
auto xt::linalg::**cross**(**const** xexpression<*E1*> &*a*, **const** xexpression<*E2*> &*b*)

Non-broadcasting cross product between two vectors.

Calculate cross product between two 1D vectors with 2- or 3 entries. If only two entries are available, the third entry is assumed to be 0.

Return resulting array

Parameters

- a: input vector
- b: input vector

1.5 Build and configuration

1.5.1 Build

xtensor-blas build supports the following options:

- BLA_VENDOR: selects the BLAS installation

1.6 Releasing xtensor-blas

1.6.1 Releasing a new version

From the master branch of xtensor-blas

- Make sure that you are in sync with the master branch of the upstream remote.
- In file xblas_config.hpp.in, set the macros for XTENSOR_BLAS_VERSION_MAJOR, XTENSOR_BLAS_VERSION_MINOR and XTENSOR_BLAS_VERSION_PATCH to the desired values.
- Add dependency information in the README.md
- Stage the changes (git add), commit the changes (git commit) and add a tag of the form Major.minor.patch. It is important to not add any other content to the tag name.
- Push the new commit and tag to the main repository. (git push, and git push --tags)

1.6.2 Updating the conda-forge recipe

xtensor-blas has been packaged for the conda package manager. Once the new tag has been pushed on GitHub, edit the conda-forge recipe for xtensor-blas in the following fashion:

- Update the version number to the new Major.minor.patch.
- Set the build number to 0.
- Update the hash of the source tarball.
- Check for the versions of the dependencies.
- Optionally, rerender the conda-forge feedstock.

INDEX

X