fastscapelib Documentation

Release 0.1.3

Benoit Bovy

Jan 25, 2021

CONTENTS

A C++ library of efficient algorithms for processing topographic data and landscape evolution modeling.

This library currently has Python bindings and is easily extensible to other languages.

CHAPTER

ONE

DOCUMENTATION INDEX

1.1 Install Fastscapelib

This library is header only and uses C++14 standards. It depends on xtensor (>0.18).

1.1.1 Install the C++ library

Using conda

Fastscapelib is available as a conda package in the conda-forge channel.

```
$ conda install fastscapelib -c conda-forge
```

From source using cmake

You need to first install the dependencies. You can install xtensor, e.g., using conda:

```
$ conda install xtensor -c conda-forge
```

Run the commands below from the source directory to install the fastscapelib's header files using cmake (on Unix platforms):

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

Where `/path/to/prefix` is the path where the header files will be installed.

On Windows platforms:

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

1.1.2 Install the Python library

Using conda

Fastscapelib's Python bindings is available as a separate conda package, still in the conda-forge channel.

```
$ conda install fastscapelib-python -c conda-forge
```

From source using pip

Fastscapelib's Python bindings requires Python (2.7.x or 3.4+), numpy, pybind11 and xtensor-python, which are also available through conda:

```
$ conda install python numpy pybind11 xtensor-python -c conda-forge
```

The `fastscapelib` Python package can then be installed locally using `pip` in editable mode:

```
$ cd python
$ pip install -e .
```

1.2 Release Notes

1.2.1 v0.2.0 (Unreleased)

- Add a cmake option that allows downloading/using xtensor development version (GH37).

1.2.2 v0.1.3 (5 November 2018)

- Update to xtensor latest version (0.18) (GH35).

1.2.3 v0.1.2 (23 August 2018)

- More robust and cleaner version management (GH32).
- Support older versions of apple-clang (GH30, GH33).

1.2.4 v0.1.1 (22 August 2018)

- Fix version (GH29).

1.2.5 v0.1.0 (21 August 2018)

Initial release.

1.3 Citing fastscapelib

If you want to use fastscapelib in a scientific publication, we would appreciate a citation. We provide DOIs for specific versions via Zenodo. Click on the badge below for more information on how to cite the latest version of fastscapelib.

1.4 C++ API Reference

Fastscapelib heavily relies on xtensor for handling n-dimensional arrays. Almost all arrays defined in the C++ interface have the `xtensor_t` type, which refers to any xtensor-compatible array such as `xt::xtensor`, `xt::xarray`, `xt::pytensor` or `xt::pyarray` (this list is extensible).

In fact, `xtensor_t` is just an alias of xtensor's class `xt::xexpression`. It has a unique template parameter which refers to the derived array type like one of those above. By convention, the parameter name is chosen using the first character(s) of the corresponding argument in function signatures.

The fastscapelib library is organized into different topics listed here below. The namespace `fastscapelib` is used for the public API. You can either include all functions in your project at once:

```
#include "fastscapelib/fastscapelib.hpp"
```

or include just a subset of the functions (by topic), e.g.,

```
#include "fastscapelib/flow_routing.hpp"
```

1.4.1 Sinks (depressions)

Functions used for depression filling or pit resolving.

Defined in `fastscapelib/sinks.hpp`.

Sink filling

template<class **G**, class **E**>
void `fastscapelib::fill_sinks_flat`(*G* &*grid*, xtensor_t<*E*> &*elevation*)

Fill all pits and remove all digital dams from a digital elevation model (rectangular grid).

Elevation values may be updated so that no depression (sinks or local minima) remains.

The algorithm is based on priority queues and is detailed in Barnes (2014). It fills sinks with flat areas.

Parameters

- `grid`: Grid object
- `elevation`: Elevation values at grid nodes

template<class **G**, class **E**>

void `fastscapelib::fill_sinks_sloped`(*G* &*grid*, xtensor_t<*E*> &*elevation*)

Fill all pits and remove all digital dams from a digital elevation model (rectangular grid).

Elevation values may be updated so that no depression (sinks or local minima) remains.

The algorithm is based on priority queues and is detailed in Barnes (2014). This variant fills sinks with nearly flat areas (i.e. elevation is increased by small amount) so that there is no drainage singularities.

See *fill_sinks_flat*

Parameters

- `grid`: Grid object
- `elevation`: Elevation values at grid nodes

1.4.2 Flow routing

Functions used to route (water) flow on a topographic surface and compute flow path-related features or structures.

Defined in `fastscapelib/flow_routing.hpp`.

Flow topology

template<class **R**, class **D**, class **E**, class **A**>
void `fastscapelib::compute_receivers_d8`(xtensor_t<*R*> &*receivers*, xtensor_t<*D*> &*dist2receivers*, **const** xtensor_t<*E*> &*elevation*, **const** xtensor_t<*A*> &*active_nodes*, double *dx*, double *dy*)

Compute flow receivers on a rectangular grid using D8 single flow routing method.

Each node on the grid is assigned one receiver among its 8 direct neighboring nodes according to the steepest slope (O'Callaghan and Mark, 1984).

When no downslope neighbor exist (i.e., pit nodes or grid boundaries), the assigned receiver is the node itself. When two or more neighbors have the same slope, the chosen neighbor is the first one considered by the algorithm.

This function also computes the planimetric distance between the node and its receiver, which equals to the grid spacing in x or y or to the distance between two diagonal neighbor nodes or 0.

Parameters

- `receivers: [intent=out, shape=(nnodes)]` Index of flow receiver at grid node.
- `dist2receivers: [intent=out, shape=(nnodes)]` Distance to receiver at grid node.
- `elevation: [intent=in, shape=(nrows, ncols)]` Topographic elevation at grid node
- `active_nodes: [intent=in, shape=(nrows, ncols)]` Boolean array for boundaries
- `dx: [intent=in]` Grid spacing in x
- `dy: [intent=in]` Grid spacing in y

template<class **N**, class **D**, class **R**>
void `fastscapelib::compute_donors`(xtensor_t<*N*> &*ndonors*, xtensor_t<*D*> &*donors*, **const** xtensor_t<*R*> &*receivers*)

Compute flow donors for each grid/mesh node.

Flow donors are retrieved by simply inverting flow receivers.

Parameters

- `ndonors: : [intent=out, shape=(nnodes)]` Number of flow donors at grid node.
- `donors: : [intent=out, shape=(nnodes, :)]` Indexes of flow donors at grid node.
- `receivers: : [intent=in, shape=(nnodes)]` Index of flow receiver at grid node.

Flow tree sorting

template<class **S**, class **N**, class **D**, class **R**>
void `fastscapelib::compute_stack`(xtensor_t<*S*> &*stack*, **const** xtensor_t<*N*> &*ndonors*, **const** xtensor_t<*D*> &*donors*, **const** xtensor_t<*R*> &*receivers*)

Compute a stack of grid/mesh nodes to be used for flow tree traversal.

The stack is calculated recursively from outlets (or sinks) to sources, using Braun and Willet's (2013) algorithm.

Parameters

- `stack: : [intent=out, shape=(nnodes)]` Stack position at grid node.
- `ndonors: : [intent=in, shape=(nnodes)]` Number of flow donors at grid node.
- `donors: : [intent=in, shape=(nnodes, :)]` Indexes of flow donors at grid node.
- `receivers: : [intent=in, shape=(nnodes)]` Index of flow receiver at grid node.

Drainage area, basins, outlets & pits

template<class **B**, class **O**, class **S**, class **R**>
index_t `fastscapelib::compute_basins`(xtensor_t<*B*> &*basins*, xtensor_t<*O*> &*outlets_or_pits*, **const** xtensor_t<*S*> &*stack*, **const** xtensor_t<*R*> &*receivers*)

Assign an id (integer) to each node of the grid/mesh that corresponds to the catchment to which it belongs.

A catchment (or drainage basin) is defined by an ensemble of adjacent nodes through which all flow converges towards a common, single node (outlet or pit).

The algorithm performs a single traversal of the flow tree (in the stack order) and increments the catchment id each time an outlet or a pit is found (i.e., when the index of the flow receiver equals the index of the node itself).

This functions also computes the grid/mesh node indexes of catchment outlets (or pits) and returns the total number of catchments found inside the domain.

Return Total number of drainage basins (`1 <= nbasins <= nnodes`).

Parameters

- `basins: [intent=out, shape=(nnodes)]` Basin id at grid node.
- `outlets_or_pits: : [intent=out, shape=(nnodes)]` Grid node index of the outlet (or pit) for basin id=0,1,. . . ,nbasins-1.
- `stack: :[intent=in, shape=(nnodes)]` Stack position at grid node.
- `receivers: : [intent=in, shape=(nnodes)]` Index of flow receiver at grid node.

template<class **P**, class **O**, class **A**>
index_t `fastscapelib::find_pits`(xtensor_t<*P*> &*pits*, **const** xtensor_t<*O*> &*outlets_or_pits*, **const** xtensor_t<*A*> &*active_nodes*, index_t *nbasins*)

Find grid/mesh nodes that are pits.

Return Total number of pits found (`0 <= npits <= nbasins`).

Parameters

- `pits`: `[intent=out, shape=(nnodes)]` Grid node index of the pit for pits in [0,1,...,npits-1].
- `outlets_or_pits`: : `[intent=in, shape=(nnodes)]` Grid node index of the outlet (or pit) for basin id=0,1,...,nbasins-1.
- `active_nodes`: : `[intent=in, shape=(nrows, ncols)]` Boolean array for boundaries
- `nbasins`: : `[intent=in]` Total number of drainage basins (`1 <= nbasins <= nnodes`).

template<class **D**, class **C**, class **S**, class **R**>
void `fastscapelib::compute_drainage_area`(xtensor_t<*D*> &*drainage_area*, **const** xtensor_t<*C*> &*cell_area*, **const** xtensor_t<*S*> &*stack*, **const** xtensor_t<*R*> &*receivers*)

Compute drainage area for each node on a generic grid/mesh.

Parameters

- `drainage_area`: : `[intent=out, shape=(nrows, ncols)||(nnodes)]` Drainage area at grid node.
- `cell_area`: : `[intent=in, shape=(nrows, ncols)||(nnodes)||()]` Grid/mesh cell area at grid node (also accepts a scalar).
- `stack`: :`[intent=in, shape=(nnodes)]` Stack position at grid node.
- `receivers`: : `[intent=in, shape=(nnodes)]` Index of flow receiver at grid node.

template<class **D**, class **S**, class **R**>
void `fastscapelib::compute_drainage_area`(xtensor_t<*D*> &*drainage_area*, **const** xtensor_t<*S*> &*stack*, **const** xtensor_t<*R*> &*receivers*, double *dx*, double *dy*)

Compute drainage area for each node on a 2-d rectangular grid.

Parameters

- `drainage_area`: : `[intent=inout, shape=(nrows, ncols)]` Drainage area at grid node.
- `stack`: :`[intent=in, shape=(nnodes)]` Stack position at grid node.
- `receivers`: : `[intent=in, shape=(nnodes)]` Index of flow receiver at grid node.
- `dx`: : `[intent=in]` Grid spacing in x.
- `dy`: : `[intent=in]` Grid spacing in y.

1.4.3 Bedrock channel

Functions used to drive the evolution of bedrock channels.

Defined in `fastscapelib/bedrock_channel.hpp`.

template<class **Er**, class **El**, class **S**, class **R**, class **Di**, class **Dr**>
index_t `fastscapelib::erode_stream_power`(xtensor_t<*Er*> &*erosion*, **const** xtensor_t<*El*> &*elevation*, **const** xtensor_t<*S*> &*stack*, **const** xtensor_t<*R*> &*receivers*, **const** xtensor_t<*Di*> &*dist2receivers*, **const** xtensor_t<*Dr*> &*drainage_area*, double *k_coef*, double *m_exp*, double *n_exp*, double *dt*, double *tolerance*)

Compute bedrock channel erosion during a single time step using the Stream Power Law.

It numerically solves the Stream Power Law [dh/dt = K A^m (dh/dx)^n] using an implicit finite difference scheme 1st order in space and time. The method is detailed in Braun and Willet's (2013) and has been slightly reformulated/optimized.

As it requires some input information on the flow tree topology (e.g., flow receivers and stack order) and geometry (e.g., distance to receivers), this generic function is grid/mesh agnostic and can be applied in both 1-d (river profile) and 2-d (river network) cases.

The implicit scheme ensure numerical stability but doesn't totally prevent erosion from lowering the elevation of a node below that of its receiver. If this occurs, erosion will be limited so that the node will be lowered (nearly) down to the level of its receiver. In general it is better to adjust input values (e.g., `dt`) so that such arbitrary limitation doesn't occur. The value returned by this function allows to detect and track the number of these occurrences.

Return Total number of nodes for which erosion has been arbitrarily limited to ensure consistency.

Parameters

- `erosion: : [intent=out, shape=(nrows, ncols)||(nnodes)]` Erosion at grid node.
- `elevation: : [intent=in, shape=(nrows, ncols)||(nnodes)]` Elevation at grid node.
- `stack: :[intent=in, shape=(nnodes)]` Stack position at grid node.
- `receivers: : [intent=in, shape=(nnodes)]` Index of flow receiver at grid node.
- `dist2receivers: : [intent=out, shape=(nnodes)]` Distance to receiver at grid node.
- `drainage_area: : [intent=out, shape=(nrows, ncols)||(nnodes)]` Drainage area at grid node.
- `k_coef: : [intent=in]` Stream Power Law coefficient.
- `m_exp: : [intent=in]` Stream Power Law drainage area exponent.
- `n_exp: : [intent=in]` Stream Power Law slope exponent.
- `dt: : [intent=in]` Time step duration.
- `tolerance: : [intent=in]` Tolerance used for Newton's iterations (`n_exp` != 1).

template<class **Er**, class **El**, class **S**, class **R**, class **Di**, class **Dr**, class **K**>

index_t fastscapelib::**erode_stream_power**(xtensor_t<*Er*> &*erosion*, **const** xtensor_t<*El*> &*elevation*, **const** xtensor_t<*S*> &*stack*, **const** xtensor_t<*R*> &*receivers*, **const** xtensor_t<*Di*> &*dist2receivers*, **const** xtensor_t<*Dr*> &*drainage_area*, **const** xtensor_t<*K*> &*k_coef*, double *m_exp*, double *n_exp*, double *dt*, double *tolerance*)

Compute bedrock channel erosion during a single time step using the Stream Power Law.

This version accepts a spatially variable stream-power law coefficient.

Return Total number of nodes for which erosion has been arbitrarily limited to ensure consistency.

Parameters

- `erosion: : [intent=out, shape=(nrows, ncols)||(nnodes)]` Erosion at grid node.
- `elevation: : [intent=in, shape=(nrows, ncols)||(nnodes)]` Elevation at grid node.
- `stack: :[intent=in, shape=(nnodes)]` Stack position at grid node.
- `receivers: : [intent=in, shape=(nnodes)]` Index of flow receiver at grid node.
- `dist2receivers: : [intent=out, shape=(nnodes)]` Distance to receiver at grid node.
- `drainage_area: : [intent=out, shape=(nrows, ncols)||(nnodes)]` Drainage area at grid node.
- `k_coef: : [intent=in, shape=(nrows, ncols)||(nnodes)]` Stream Power Law coefficient.
- `m_exp: : [intent=in]` Stream Power Law drainage area exponent.
- `n_exp: : [intent=in]` Stream Power Law slope exponent.
- `dt: : [intent=in]` Time step duration.
- `tolerance: : [intent=in]` Tolerance used for Newton's iterations (`n_exp` != 1).

1.4.4 Hillslope

Functions used to drive the evolution of hillslopes.

Defined in `fastscapelib/hillslope.hpp`.

template<class **Er**, class **El**>
void fastscapelib::**erode_linear_diffusion**(xtensor_t<*Er*> &*erosion*, **const** xtensor_t<*El*> &*elevation*, double *k_coef*, double *dt*, double *dx*, double *dy*)

Compute hillslope erosion by linear diffusion on a 2-d regular grid using finite differences with an Alternating Direction Implicit (ADI) scheme.

This numerical scheme is implicit and unconditionally stable. It is second order in time and space (its accuracy still depends on the values chosen for step duration, grid resolution and diffusivity).

This implementation assumes fixed (Dirichlet) boundary conditions on the four sides of the grid.

Parameters

- `erosion: : [intent=out, shape=(nrows, ncols)]` Erosion at grid node.
- `elevation: : [intent=in, shape=(nrows, ncols)]` Elevation at grid node.

- k_coef: : [intent=in] Diffusion coefficient.
- dt: : [intent=in] Time step duration.
- dx: : [intent=in] Grid spacing in x
- dy: : [intent=in] Grid spacing in y

template<class **Er**, class **El**, class **K**>
void fastscapelib**::erode_linear_diffusion**(xtensor_t<*Er*> &*erosion*, **const** xtensor_t<*El*> &*elevation*, **const** xtensor_t<*K*> &*k_coef*, double *dt*, double *dx*, double *dy*)

Compute hillslope erosion by linear diffusion on a 2-d regular grid using finite differences with an Alternating Direction Implicit (ADI) scheme.

This version accepts a spatially variable diffusion coefficient.

Parameters

- erosion: : [intent=out, shape=(nrows, ncols)] Erosion at grid node.
- elevation: : [intent=in, shape=(nrows, ncols)] Elevation at grid node.
- k_coef: : [intent=in, shape=(nrows, ncols)] Diffusion coefficient.
- dt: : [intent=in] Time step duration.
- dx: : [intent=in] Grid spacing in x
- dy: : [intent=in] Grid spacing in y

1.5 Python API Reference

Fastscapelib Python bindings are built with pybind11 and xtensor-python. Although there is not yet comprehensive documentation for the Python API, it closely matches the C++ API (see *C++ API Reference*). Python functions accept numpy arrays instead of xtensor arrays.

1.6 Build and Configuration

1.6.1 Build options

fastscapelib build supports the following options (all disabled by default). See below for more explanations.

- BUILD_TESTS: enables the run_test target.
- BUILD_BENCHMARK: enables the run_benchmark target.
- DOWNLOAD_GTEST: downloads google-test and builds it locally instead of using a binary installation.
- GTEST_SRC_DIR: indicates where to find the google-test sources instead of downloading them.
- BUILD_PYTHON_MODULE: enables building fastscapelib as a Python extension.
- DOWNLOAD_XTENSOR: downloads xtensor development version (master branch on github) and uses it to build fastscapelib (useful for testing - might be needed for building fastscapelib development version).

1.6.2 Build and run tests

Fastscapelib has a test suite based on google-test. The enabled `BUILD_TESTS` adds the target `run_tests` which builds and runs the whole test suite, e.g.,

```
$ mkdir build
$ cd build
$ cmake -DBUILD_TESTS=ON ..
$ make run_tests
```

Google-test must have been already installed, e.g., using conda:

```
$ conda install gtest -c conda-forge
```

Alternatively, google-test may be downloaded automatically by enabling `DOWNLOAD_GTEST`, or a custom install path may be given by setting `GTEST_SRC_DIR`. Note that enabling `DOWNLOAD_GTEST` or setting `GTEST_SRC_DIR` enables `BUILD_TESTS`.

1.6.3 Build and run benchmarks

Fastscapelib has also a benchmark suite based on google-benchmark. Building and running benchmarks is similar to building and running tests (the `BUILD_BENCHMARK` option and `run_benchmark` target are used instead). Note that `BUILD_BENCHMARK` automatically downloads google-benchmark.

1.6.4 Build the Python extension

Fastscapelib has Python bindings, which can be built by enabling `BUILD_PYTHON_MODULE`, e.g.,

```
$ mkdir build
$ cd build
$ cmake -DBUILD_PYTHON_MODULE=ON ..
$ make _fastscapelib_py
```

Note that the created Python library is not intended to be directly used within a regular Python installation. The preferred way to build and install fastscapelib locally is to use pip:

```
$ cd python
$ pip install -e .
```

CHAPTER

TWO

ACKNOWLEDGMENT

This project is supported by the Earth Surface Process Modelling group of the GFZ Helmholtz Centre Potsdam.

CHAPTER

THREE

CITING FASTSCAPELIB

If you use fastscapelib in a scientific publication, we would certainly appreciate a citation (see *Citing fastscapelib* section).

INDEX

F