minieigen Documentation

Release 0.53

Václav Šmilauer

Jan 21, 2018

Contents

1 Overview 1

2 Examples 3

3 Naming conventions 5

4 Limitations 7

5 Links 9

6 Documentation 11

CHAPTER 1

Overview

Todo: Something concise here.

CHAPTER 2

Examples

Todo: Some examples of what can be done with `minieigen`.

CHAPTER 3

Naming conventions

- Classes are suffixed with number indicating size where it makes sense (it does not make sense for `minieigen.Quaternion`):
  - `minieigen.Vector3` is a 3-vector (column vector);
  - `minieigen.Matrix3` is a 3×3 matrix;
  - `minieigen.AlignedBox3` is aligned box in 3d;
  - `X` indicates dynamic-sized types, such as `minieigen.VectorX` or `minieigen.MatrixX`.
- Scalar (element) type is suffixed at the end:
  - nothing is suffixed for floats (`minieigen.Matrix3`);
  - `i` indicates integers (`minieigen.Matrix3i`);
  - `c` indicates complex numbers (`minieigen.Matrix3c`).
- Methods are named as follows:
  - static methods are upper-case (as in c++), e.g. `minieigen.Matrix3.Random`;
    - nullary static methods are exposed as properties, if they return a constant (e.g. `minieigen.Matrix3.Identity`); if they don't, they are exposed as methods (`minieigen.Matrix3.Random`); the idea is that the necessity to call the method (`Matrix3.Random()`) singifies that there is some computation going on, whereas constants behave like immutable singletons.
  - non-static methods are lower-case (as in c++), e.g. `minieigen.Matrix3.inverse`.
- Return types:
  - methods modifying the instance in-place return `None` (e.g. `minieigen.Vector3.normalize`); some methods in c++ (e.g. Quaternion::setFromTwoVectors) both modify the instance *and* return the reference to it, which we don't want to do in Python (`minieigen.Quaternion.setFromTwoVectors`);
  - methods returning another object (e.g. `minieigen.Vector3.normalized`) do not modify the instance;

- methods returning (non-const) references return by value in python

CHAPTER 4

Limitations

- Type conversions (e.g. float to complex) are not supported.
- Methods returning references in c++ return values in Python (so e.g. `Matrix3().diagonal()[2]=0` would zero the last diagonal element in c++ but not in Python).
- Many methods are not wrapped, though they are fairly easy to add.
- Conversion from 1-column `MatrixX` to `VectorX` is not automatic in places where the algebra requires it.
- Alignment of matrices is not supported (therefore Eigen cannot vectorize the code well); it might be a performance issue in some cases; c++ code interfacing with minieigen (in a way that c++ values can be *set* from Python) **must** compile with `EIGEN_DONT_ALIGN`, otherwise there might be crashes at runtime when vector instructions receive unaligned data. It seems that alignment is difficult to do with boost::python.
- Proper automatic tests are missing.

CHAPTER 5

Links

- http://eigen.tuxfamily.org (Eigen itself)
- http://www.launchpad.net/minieigen (upstream repository, bug reports, answers)
- https://pypi.python.org/pypi/minieigen (Python package index page, used by `easy_install`)
- packages:
  - Debian
  - Ubuntu: distribution, PPA

CHAPTER 6

Documentation

- genindex
- search