Numeric / Math / Linear Algebra

Numerically Robust Variance and weighted sum

- John D. Cook's Accurately computing running variance
- https://kunalbandekar.blogspot.com/2011/03/algorithms-for-calculating-variance.html
- https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance
    - Welford algorithm
    - West algorithm
- https://en.wikipedia.org/wiki/Squared_deviations_from_the_mean
- https://en.wikipedia.org/wiki/Kahan_summation_algorithm
    - https://www.ddj.com/floating-point-summation/184403224
    - https://rosettacode.org/wiki/Kahan_summation
    - https://stackoverflow.com/questions/57301596/optimal-implementation-of-iterative-kahan-summation

C/C++ Libraries

- C: Intel(R) Math Kernel Library (MKL)
    - https://en.wikipedia.org/wiki/Math_Kernel_Library
    - https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html
    - https://www.intel.com/content/www/us/en/develop/documentation/onemkl-linux-developer-guide/top.html
- C: Intel(R) Integrated Performance Primitives (IPP)
    - https://en.wikipedia.org/wiki/Integrated_Performance_Primitives
    - https://www.intel.com/content/www/us/en/developer/tools/oneapi/ipp.html
    - https://www.intel.com/content/www/us/en/develop/documentation/ipp-dev-reference/top.html
- C: Vector Optimized Library of Kernels (libVOLK)
    - https://www.libvolk.org/
    - currently GPLv3 license
    - future version 3 will be LGPL, see https://www.libvolk.org/help-us-relicense-volk-under-lgpl.html
- C++ Eigen
    - https://eigen.tuxfamily.org/
    - https://eigen.tuxfamily.org/dox/index.html
    - pre-allocated arrays can be mapped to Eigen's Matrix/Vector types
        - https://eigen.tuxfamily.org/dox/classEigen_1_1Map.html
        - see https://eigen.tuxfamily.org/dox/group__matrixtypedefs.html for the native types
    - matrix self transpose/adjoint multiply optimization, e.g. for least squares
        - https://stackoverflow.com/questions/39606224/does-eigen-have-self-transpose-multiply-optimization-like-h-transposeh
        - https://stackoverflow.com/questions/42712307/efficient-matrix-transpose-matrix-multiplication-in-eigen
- Boost Interval Arithmetic
    - https://www.boost.org/doc/libs/1_75_0/libs/numeric/interval/doc/interval.htm
- C++ Armadillo

    - https://arma.sourceforge.net/
- Fastor
    - https://github.com/romeric/Fastor
- libxsmm
    - https://github.com/libxsmm/libxsmm
- NumPy
    - NumPy is python, but there are promising c++ libraries:
    - https://github.com/dpilger26/NumCpp see documentation
    - https://github.com/xtensor-stack/xtensor see documentation
        - optionally utilizes https://github.com/xtensor-stack/xsimd / https://xsimd.readthedocs.io/en/latest/
- other libraries:
    - https://en.wikipedia.org/wiki/Comparison_of_linear_algebra_libraries
    - https://stackoverflow.com/questions/1380371/what-are-the-most-widely-used-c-vector-matrix-math-linear-algebra-libraries-a

IEEE-754 Float Numbers

- https://en.wikipedia.org/wiki/IEEE_754
    - existence of denormalized numbers (aka subnormal numbers) should be known
    - existence of signed zeros might be interesting
    - existence of following non-numbers (Not-a-Number: NaN) should be known
        - quiet NaN (qNaN)
        - signaling NaN (sNaN)
        - positive and negative infinity (+/- inf)
    - rounding modes can be controlled
        - nearest, towards 0, towards + or -inf
        - be aware, that rotating a 2D-point or complex coordinate in a loop will go towards zero, due to error propagation with default rounding mode '*round towards zero*' !
    - exceptions can be handled, by setting up float-traps/signal handler

Performance Issues with Denormals and NANs

Calculation with denormals or non-numbers slows down performance - even when not signalled.
it might be interesting to abort a calculation, e.g. a matrix/vector multiplication, with first occurence of NaN - or with one of the other conditions .. Unfortunately, SIMD instruction sets have issues on exception trapping and NAN propagation. See Agner Fog's article at https://www.agner.org/optimize/#nan_propagation

Special options like DAZ (Denormals-Are-Zero) and FTZ (Flush-To-Zero) can be used, if the application won't care about very small denomalized numbers, see https://en.wikipedia.org/wiki/Subnormal_number

IPP library does also provide helper functions:

- https://www.intel.com/content/www/us/en/docs/ipp/developer-reference/2021-9/setdenormarezeros.html
- https://www.intel.com/content/www/us/en/docs/ipp/developer-reference/2021-9/setflushtozero.html

Fast cache-efficient matrix transposition

The topic is explained at wikipedia - with a specialized article for the *in-place* operation. Matrix transposition can also be utilized in the field of image processing. (De-)Interleaving is a different wording for the same operation, e.g. multi-channel audio data. See https://stackoverflow.com/questions/7780279/de-interleave-an-array-in-place

Transposing a matrix the *simple* way will produce many cache misses. That is, why special algorithms, like Cache-oblivious ones, are beneficial. There are numerous scientific papers on this topic, e.g. Cache-efficient matrix transposition. But the problem is also discussed on stackoverflow - happily with some code snippets.

In general, one should consider following aspects:

- rectangular or square matrix
- transpose only from/to rectangular regions of bigger matrices or images
- in-place or out-of-place operation
- combination with conjugation

Here some libraries, which should be quite performance efficient, providing transpose functions:

- Intel IPP: ippiTranspose_*()
- Intel MKL: out-of-place mkl_*omatcopy()
  - unfortunately no smaller data types than float
  - Intel MKL: in-place mkl_*imatcopy()
- Eigen: .transpose()
  - returns new created/transposed matrix
  - pre-allocated arrays can be mapped to Eigen's Matrix/Vector types
    - https://eigen.tuxfamily.org/dox/classEigen_1_1Map.html
    - see https://eigen.tuxfamily.org/dox/group__matrixtypedefs.html for the native types
- Armadillo: trans() - also with slower in-place variant
  - returns new created/transposed matrix
  - advanced constructor with `copy_aux_mem = false` for using external data directly

github.com also produces many results, when searching for "*transpose*".

Pavel Zemtsov wrote a bunch of related articles at Experiments in program optimisation, backed with sources at https://github.com/pzemtsov/article-e1-cache and https://github.com/pzemtsov/article-E1-demux-C:

- De-multiplexing of E1 stream: converting to C
- Fast E1 de-multiplexing in C using SSE/AVX
- How to transpose a 16×16 byte matrix using SSE
- Bug story 2: Unrolling the 16×16 matrix transposition, or be careful with macros
- How cache affects demultiplexing of E1

other links:

- https://github.com/FFmpeg/FFmpeg/blob/master/libavfilter/opencl/transpose.cl
- https://github.com/FFmpeg/FFmpeg/blob/master/libavcodec/arm/neon.S
- https://www.cs.technion.ac.il/~itai/Courses/Cache/matrix-transposition.pdf

- https://dl.acm.org/doi/10.1145/3555353
- https://github.com/romeric/Fastor/blob/master/Fastor/backend/transpose/transpose_kernels.h
- https://groups.google.com/g/llvm-dev/c/P_CXf5hPro8/m/RI6Fm3bzAQAJ

there's also a new library: https://github.com/hayguen/libtranspose

Links

- Numerical Computation Guide
  - https://docs.oracle.com/cd/E19957-01/806-3568/ncgTOC.html
- Several Blog Entries of Bruce Dawson
  - https://randomascii.wordpress.com/category/floating-point/
- C99 and C++11 Floating-point environment reference
  - https://en.cppreference.com/w/c/numeric/fenv
  - https://en.cppreference.com/w/cpp/numeric/fenv