The Journal of Open Source Software

tmech a C++ library for the numerical study of the physics of continuous materials using higher-oder tensors

Peter Lenz[1¶]

1 Chair of Engineering Mechanics, Paderborn University, Germany ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

The objective of the tmech framework is to propose high-level semantics, inspired by different linear algebra packages, allowing fast software prototyping in a low-level compiled language numerical study of the physics of continuous materials using higher-oder tensors.

The tmech framework is provided as an open source software under the BSD-3-Clause License, compiled and validated with clang and gcc

Statement of need

For the numerical study of the physics of continuous materials using higher-oder tensors. Partial Differential Equations (PDEs) describing natural phenomena are modelled using tensors of different order. Two commonly studied problems are heat transfer, which include temperature and heat flux (rank-0 and rank-1 tensor, respectively), and continuum mechanics, which include stress and strain (rank-2 tensors) and the so-called tangent stiffness (rank-4 tensor).

Overview and features

Non-indexed lower case light face Latin letters (e.g. f and h) are used for scalars

Tensor operations involving overloaded C++ operator functions

- Addition of tensors of same rank and dimension
- Subtraction of tensors of same rank and dimension
- Scalar update of tensors

General inner product

Hence if one of the original tensors is of rank-m and the other is of rank-n, the inner product will be of rank-(m + n − 2). Controlled by template parameters SeqLHS and SeqRHS. Bases contained in sequence SeqLHS are contracted with bases contained in sequence SeqLHS. tmech provides some wrapper functions for the most common inner products $a : B$, $A : b$, $A : B$ and $A :: B$, see documentation for more details.

```
using SeqL = tmech::sequence<3,4>;
using SeqR = tmech::sequence<1,2>;
//Double contraction of two 4th order tensors
tmech::tensor<double, 3, 4> A, B, C;
C = tmech::inner_product<SeqL, SeqR>(A,B);
//Double contraction of a 4th and a 2th order tensor
```

The Journal of Open Source Software

```cpp
tmech::tensor<double, 3, 2> a, c;
c = tmech::inner_product<SeqL, SeqR>(C,a);
```

General outer product

On multiplying each component of a tensor of rank r by each component of a tensor of rank k, both of dimension m, a tensor of rank $(r+k)$ with mr+k components is obtained The outer product of a tensor of type (m, n) by a tensor of type (p, q) results in a tensor of type (m + p, n + q) the outer product of two tensors their outer product is a tensor. The outer product of tensors is also referred to as their tensor product, ontrolled by template parameters SeqLHS and SeqRHS. Bases contained in sequence SeqLHS are used for ordered element acces in A. Bases contained in SeqRHS are used for ordered element acces in B expression. tmech provides some wrapper functions for the most common outer products $\otimes$, $\underline{\otimes}$ and $\underline{\otimes}$, see documentation for more details. An example code snippet, where two second-order tensor are multiplied to form a fourth-oder tensor is given as

```cpp
using SeqL = tmech::sequence<1,2>;
using SeqR = tmech::sequence<3,4>;
...
tmech::tensor<double, 3, 2> a, b;
tmech::tensor<double, 3, 4> C;
C = tmech::outer_product<SeqL, SeqR>(a,b);
```

where bases 1,2 of the new tensor C are given by a and bases 3,4 are given by b.

General basis rearrangement

Controlled by template parameter Sequence, which contains the new order of bases. tmech provides some wrapper functions for the most common basis rearrangement like transposition of a second-order tensor and a major-transposition of a fourth-order tensor. An example code snippet

```cpp
//Basis 1,2,3,4 is swaped to 3,4,1,2.
tmech::tensor<double, 3, 4> A, B;
A.randn();
B = tmech::basis_change<tmech::sequence<3,4,1,2»(A);
B = tmech::basis_change<tmech::sequence<3,4,1,2»(A+2*A);
```

Decompositions

Eigendecomposition

Eigendecomposition of a positive semi-definite symmetric second-order tensor

$$\boldsymbol{Y} = \sum_{i=1}^{m} \lambda_i \boldsymbol{e}_i \otimes \boldsymbol{e}_i = \sum_{i=1}^{m} \lambda_i \boldsymbol{E}_i, \tag{1}$$

where m is the number of non repeated eigenvalues, λ_i are the corresponding eigenvalues, $\boldsymbol{e}_i$ are eigenvectors and $\boldsymbol{E}_i$ are eigenbasen.

```cpp
tmech::tensor<double, 3, 2> A, A_inv;
A = tmech::sym(tmech::randn<double,3,2>());
auto A_eig = tmech::eigen_decomposition(A);
const auto[eigenvalues, eigenbasis]{A_eig.decompose_eigenbasis()};
const auto idx{A_eig.non_repeated_eigenvalues_index()};
for(auto idx : A_eig.non_repeated_eigenvalues_index()){
    A_inv += (1.0/eigenvalues[idx])*eigenbasis[idx];
```

```
}
std::cout«std::boolalpha«tmech::almost_equal(tmech::inv(A), A_inv, 5e-7)«std::endl;
```

Polar decomposition

Polar decomposition of a positive semi-definite symmetric second-order tensor

$$\boldsymbol{F} = \boldsymbol{R}\boldsymbol{U} = \boldsymbol{V}\boldsymbol{R} \tag{2}$$

where $\boldsymbol{R}$ is an orthogonal tensor also knwon as the rotation tensor, $\boldsymbol{U}$ and $\boldsymbol{V}$ are symmetric tensors called the right and the left stretch tensor, respectively. This function provides two different methods to determine $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{R}$. The first method uses spectral decomposition

$$\boldsymbol{U} = \sqrt{\boldsymbol{F}^T\boldsymbol{F}}, \quad \boldsymbol{R} = \boldsymbol{F}\boldsymbol{U}^{-1}, \quad \boldsymbol{V} = \boldsymbol{R}\boldsymbol{U}\boldsymbol{R}^T, \tag{3}$$

```
//use the spectral decomposition
tmech::tensor<double, 3, 2> F;
F.randn();
auto F_polar = tmech::polar_decomposition(F);
F = F_polar.R()*F_polar.U();
F = F_polar.V()*F_polar.R();
```

The second one is based on a Newton iteration

$$\boldsymbol{R}_{k+1} = \frac{1}{2}\left(\boldsymbol{R}_k + \boldsymbol{R}_k^{-T}\right), \quad \text{with} \quad \boldsymbol{R}_0 = \boldsymbol{F} \tag{4}$$

```
//use the newton iteration
tmech::tensor<double, 3, 2> F;
F.randn();
auto F_polar = tmech::polar_decomposition(F, // tensor to decompose
                                          true, //use newton iteration
                                          5e-7, // tolerance
                                          5 // number of max steps
                                          );
F = F_polar.R()*F_polar.U();
F = F_polar.V()*F_polar.R();
```

The following derivatives are also important

$$\frac{\partial \boldsymbol{U}}{\partial \boldsymbol{F}}, \quad \frac{\partial \boldsymbol{V}}{\partial \boldsymbol{F}}, \quad \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{F}} \tag{5}$$

explicit results are given here

```
//use the spectral decomposition
tmech::tensor<double, 3, 2> F;
F.randn();
auto F_polar = tmech::polar_decomposition(F);
auto dR = F_polar.R().derivative();
auto dU = F_polar.U().derivative();
auto dV = F_polar.V().derivative();
```

Nummerical differentiation

Numerical differentiation based on central difference scheme $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$

Non-symmetric tensor functions

```
tmech::tensor<double,3,2> X;
X.randn();
auto func = [&](auto const& F){return 1.5*(tmech::trace(tmech::trans(F)*F) - 3); };
auto dFunc = tmech::num_diff_central(func, X);
auto dFunc_ = [&](auto const& F){return tmech::num_diff_central(func, F);};
auto ddFunc = tmech::num_diff_central<tmech::sequence<1,2,3,4»(func, X);
```

Symmetric tensor functions

```
using Sym2x2 = std::tuple<tmech::sequence<1,2>,tmech::sequence<2,1»;
//symmetry of the result
using Sym4x4 = std::tuple<
               tmech::sequence<1,2,3,4>,
               tmech::sequence<2,1,3,4>,
               tmech::sequence<1,2,4,3>,
               tmech::sequence<2,1,4,3»;
tmech::tensor<double,3,2> X;
X = tmech::sym(tmech::rand<double,3,2>());
//first deriv
auto func = [&](auto const& F){return 1.5*(tmech::trace(tmech::trans(F)*F) - 3);};
auto dfunc_res = tmech::num_diff_sym_central<Sym2x2>(func, X);
//second deriv
auto dfunc = [&](auto const& F){return tmech::num_diff_sym_central<Sym2x2>(func, F);};
auto ddfunc_res = tmech::num_diff_sym_central<Sym2x2, Sym4x4>(dfunc, X);
```

Compile-time differentiation

As an illustrative example of using tmech, we here give the mathematical formulation of an energy potential and stress in large deformation continuum mechanics and its implementation as a function in Julia. For a deformation gradient F = I + grad u, where u is the displacement from the reference to the current configuration, the right Cauchy-Green deformation tensor is defined by C=FTF. The Second Piola-Kirchoff stress tensor S is derived from the Helmholtz free energy psi by the relation

$$\boldsymbol{S} = 2\frac{\partial \psi}{\partial \boldsymbol{C}} \tag{6}$$

The strain energy density function for an incompressible Mooney–Rivlin material is

$$\psi = C_{10}(I_1 - 3) + C_{01}(I_2 - 3), \quad \text{where} \quad I_1 = \text{trace}\boldsymbol{C}, \quad I_2 = \frac{1}{2}(I_1^2 - \text{trace}(\boldsymbol{C}^2)) \tag{7}$$

where is the isochoric part

User material in Abaqus

Abaqus is a commerical software and often used in research (Abaqus, 2014)

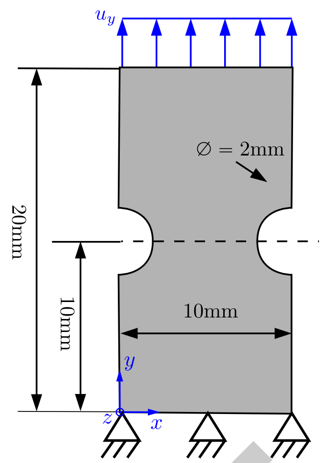
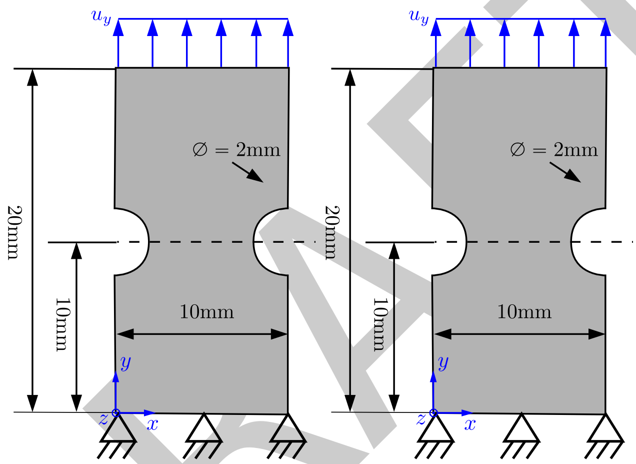
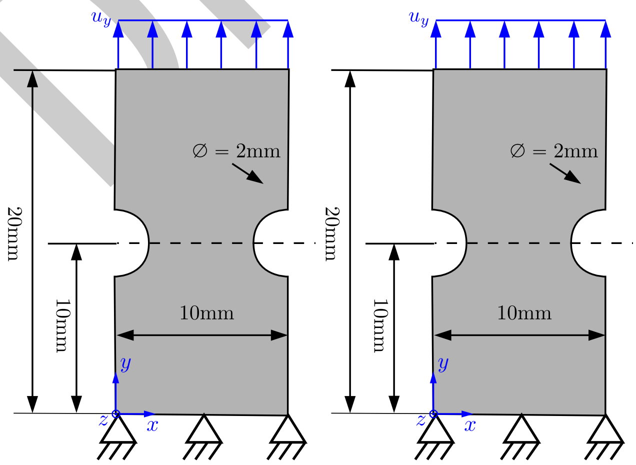


Figure 1: Caption for example figure.



Caption: Look at all my baby seals!



Look at my tiny horses!

$$\boldsymbol{\sigma} = \lambda \mathrm{trace} \boldsymbol{\varepsilon} \boldsymbol{I} + 2\mu\boldsymbol{\varepsilon}, \qquad \mathbb{C} = \lambda \mathbb{II} + 2\mu\mathbb{I}^s \tag{8}$$

```
#include <tmech/tmech.h>
```

```
using tensor2 = tmech::tensor<double,3,2>;
using tensor4 = tmech::tensor<double,3,4>;

extern ”C” void umat_(double *stress, double *statev, double *ddsdde, double *sse,
        double *spd, double *scd, double *rpl, double *ddsddt, double *drplde,
        double *drpldt, double *stran, double *dstran, double *time, double *dtime,
        double *temp, double *dtemp, double *predef, double *dpred, char *cmname,
        int *ndi, int *nshr, int *ntens, int *nstatv, double *props, int *nprops,
        double *coords, double *drot, double *pnewdt, double *celent, double *dfgrd0,
        double *dfgrd1, int *noel, int *npt, int *layer, int *kspt,
        int *kstep, int *kinc, short cmname_len){
    //parameters
    const double E     = props[0]; //210000
    const double nue   = props[1]; //0.3

    //Lamé parameters
    const double mu     = E/(2*(1+nue));
    const double lambda = E*nue/((1+nue)*(1-2*nue));


    //second order tensor
    const tmech::eye<double,3,2> I;
    //fourth order symmetric identity tensor
    const auto IIsym = 0.5*(tmech::otimesu(I,I) + tmech::otimesl(I,I));
    //fourth order identity tensor
    const auto II = tmech::otimes(I,I);

    //voigt to tensor currently abq_std only Dim==3 supported
    tmech::adaptor<double,3,2, tmech::abq_std<3,true» strain(stran);
    tmech::adaptor<double,3,2, tmech::abq_std<3,true» dstrain(dstran);
    tmech::adaptor<double,3,2, tmech::abq_std<3» sig(stress);
    tmech::adaptor<double,3,4, tmech::abq_std<3» C(ddsdde);

    //updated strain
    const tensor2 eps = strain + dstrain;
    //trail stress
    const tensor2 sig_tr = lambda*tmech::trace(eps)*I + 2*mu*(eps);
    //elasticity tensor
    C = lambda*II + 2*mu*IIsym;
}
```

: Abaqus example linear elasticity.

Compile to obtaine the . With the following commands for a Linux based operating system the object file is generated

```
gcc -c -fPIC -O3 -march=native -std=c++17 -I/path_to_tmech -o  object_file_output.o inpu
```

The compile flags -03 and -march=native are used to enable optimization and auto vectorization of the code, respectively. After the object file is generated, a Abaqus job can be submitted by the following commands

```
abaqus job=input_file_name user=object_file_output
```

where input_file_name is the name of the .inp input file and object_file_output is the name of the generated object file.

A more complex example of J2-plasticity is given in the ‘tmech’ github repositories

References

Abaqus. (2014). *Abaqus/standard user's manual, version 6.14*. Dassault Systèmes Simulia Corporation.