C++/Eigen snippets

Schüttler, Janik

Lambda functions

```cpp
auto func = [&globalVar] (int x) {return globalVar + x; };
```

Eval lamda functions

```cpp
x.unaryExpr(func);
```

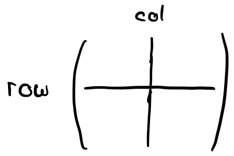
Iterate $2^0, 2^1, 2^2, \dots$

```cpp
for ( int i = 0, i <= N, ++i)   int n = 1 << i;
```

Matrix dimensions

A is m×n

```cpp
m = A.rows()
n = A.cols()
```



Solve $Ax = b$

use

```cpp
FullPivLU<MatrixXd> B = A.fullPivLu();
PartialPivLU<MatrixXd> B = A.partialPivLu();   // if A is invertible
ColPivHouseholder<MatrixXd> B = A.colPivHouseholderQr();
// Full..                          A.fullPiv...
TriangularView<MatrixXd, Upper> B = A.triangularView<Upper>();
//                     Lower                          Lower
typedef Eigen::SparseLU<Eigen::SparseMatrix<double>> solver_t;
solver_t solver;
solver.compute(A);
VectorXd b = B.solve(b);
```

C++ vectors

```cpp
std::vector<int> v;
std::sort (v.begin(), v.end());
v.size();
v.push_back(42);
int m = v[0];
v.pop_back();
```

(needs #include <algorithms>)

```cpp
Map<VectorXd> myNewEigenVector (v.data(), v.size());
vector<double> v2 (mat.data(), mat.data() + mat.rows() * mat.cols());

for (auto it=v.begin(); it != v.end(); ++it) {cout << *it << endl;}
for (auto t: v) { cout<< t << endl; }
```

Triplets

```cpp
Eigen::Triplet<double> t(3, 4, 42);
//                  t.row() t.col() t.value()
```

Sparse matrices

```cpp
SparseMatrix<double> A(n,n);
std::vector<Triplet<double>> triplets;
triplets.reserve(n);
triplets.push_back(Triplet<double>(1,2,42))
for (auto& triplet : triplets ) {
  triplet = Triplet<double>(triplet.row(), triplet.col(), triplet.value() + 1);
}
A.setFromTriplets(triplets.begin(), triplets.end());
A.makeCompressed();
```

C++ constants and functions

```cpp
pi:  M_PI   (#include <cmath>)
sin(), exp(), cos(), abs()
```

Max coefficient + index

```cpp
int index;     double max = v.maxCoeff(&index);
```

Linspaced Vector

For (0,1,2,...,N): (N+1, 0, N)

```cpp
VectorXd x = VectorXd::LinSpaced(len, from, to)
ArrayXd x = ArrayXd::LinSpaced(len, from, to)
```

Eps

(not :: denorm_min())

```cpp
double eps = std::numeric_limits<double>::epsilon();
```

Pretty print

```cpp
std::cout << std::scientific << std::setprecision(3) << std::setw(10)
  << out << std::endl;     #include <iomanip>
```

Structs

```cpp
struct data_t {
  data_t(const fes_t& fespace, vector_t& u0,
    numeric_t t): fes(fespace), u0(u0), tau(t) {}
  coord_t v(const coord_t& x) const {
    return (coord_t() << 1, 2+x(0)).finished();
  }
  const fes_t& fes;
  vector_t& u0;
  numeric_t tau;
} data (fes, u0, 0.1);
data.u0 = update(data.u0);
```

Code snippets

Schüttler, Janik

Compute norm

- ```
  template<typename FESPACE_T, typename VECTOR_T>
  static double H1norm(FESPACE &fespace, VECTOR_T &mu){
      GalerkinMatrixAssembler<AnalyticStiffnessLocalAssembler> Assembler;
      Eigen::SparseMatrix<double> A = Assembler.assembleMatrix(fespace,
                                                                fespace, 1.0);
      return sqrt(mu.transpose() * A * mu);
  }
  ```
- ```
  template<typename FESPACE_T, typename VECTOR_T>
  static double L1norm(FESPACE &fespace, VECTOR_T &mu){
      GalerkinMatrixAssembler<AnalyticMassLocalAssembler> Assembler;
      Eigen::SparseMatrix<double> A = Assembler.assembleMatrix(fespace,
                                                                fespace, 1.0);
      return sqrt(mu.transpose() * A * mu);
  }
  ```

Looping over entities

- Loop over all boundary edges
  ```
  using itsct_t = eth::grid::Intersection<betl2::volume2dGrid::
                                          hybrid::GridTraits>;
  std::vector<const itsct_t*> boundary_inters;
  for (const auto & el : gridView.template entities<0>()) {
      for (const auto& inters : gridView.intersections(el)) {
          if( inters.boundary())  boundary_inters.push(&inters);
      }
  }
  for (const auto& inters : boundary_inters) {
      const auto & el = inters->inside();  // master element
      const auto & geom = inters->geometry();
  }
  ```
- Basic Loops
  ```
  for (auto const element& : fespace) {
      for (auto dof : fespace.indices(element)) {
          auto localIndex = dof.local();
          auto globalIndex = dof.global();
      }
  }
  for (auto el_it = fespace.begin(); el_it != fespace.end(); ++el_it) {
      for (auto dof_it = fespace.begin(*el_it); dof_it != fespace.end(*el_it);
           ++dof_it) {
          auto localIndex = fespace.localIndex(dof_it, *el_it);
          auto globalIndex = fespace.globalIndex(dof_it);
      }
  }
  for (const auto element& : gridView.template entities<0>()) { }
  ```

Schüttler, Janik

BETL

Geometry object (3.7.42, p 364)

- Constant dimFrom
- Constant dimTo
- Type gridTraits_t
- Vector type globalCoord_t
- Vector type localCoord_t
- Integer type size_type
- method size_type mapCorners()
- method globalCoord_t mapCorner(int i)
- method gridTraits_t::ctype_t volume()
- method globalCoord_t center()

Geometric entity object (3.7.28, p 251)

- refElType() POINT, SEGMENT, TRIA, QUAD, TETRA, ...
- geometry()
- int countSubEntities<codim>()
- EntityPtr subEntity<codim>(int locidx)

Index set indexSetRef_t set(gv.indexSet())

- index_t index(const Entity &)
- template <CODIM> index_t subIndex(const Entity<GRID_TRAITS,0> &element, size_type locidx)

Intersection object (3.7.47, p 267)

- bool boundary()
- bool neighbor()
- geometry()
- inside()
- outside()
- indexInInside()
- indexInOutside()

FESpace (3.7.72, p 279)

- begin(), end()
- begin(e), end(e)
- dofsOnElement()
- globalIndex(dIter)
- localIndex(dIter, e)
- filter<CODIM>(e, intersectionIndex)
- filterAll(e, intersectionIndex)
- filterIndices(e, intersectionIndex)
- indices(e, intersectionIndex)
- indices(e)
- numDofs()
- numElements()

NPDE

GalerkinMatrixAssembler, LoadVectorAssembler
AnalyticStiffnessLocalAssembler, AnalyticMassLocalAssembler,
LocalVectorAssembler

Quadrature

template<enum eth::base::RefElType RET, eth::base::signed_t NUM_POINTS> class Quadrature

- getNumPoints()
- getRefEl()
- getPoints()
- getWeights()
- getScale()
- getRefDim()

Example Code

Initialization

```
using grid_t = volume2dGrid::hybrid::Grid;
using eth::grid::GridViewTypes view = eth::grid::GridViewTypes::LeafView;
using gridView_t = typename eth::grid::GridView<grid_t::
                   gridTraits_t::template viewTraits_t<view>>;

using gridTraits_t = gridView_t::gridTraits_t;
using gridCreator_t = GridCreator<grid_t, view>;
using gridFactory_t = gridCreator_t::gridFactory_t;
using itsct_t = eth::grid::Intersection<betl2::volume2dGrid::hybrid::GridTraits>;

const std::string basename = "./hex" + std::to_string(6);
betl2::input::gmsh::Input input(basename);
const gridFactory_t gridFactory = gridCreator_t()(input);
const gridView_t gridView = gridFactory.getView();

using febasis_t = fe::FEBasis<fe::Linear, fe::FEBasisType::Lagrange>;
using dofHandler_t = fe::DofHandler<febasis_t, fe::FESContinuity::Continuous
                                   gridFactory_t>;
dofHandler_t dh;
dh.distributeDofs(gridFactory);
auto mu = solveImpedanceBVP(dh.fespace(), gridView, boundary_iter);
```

Solve

```
template <typename LINSPACE, typename INTERS>
Eigen::VectorXd solveImpedanceBVP(const LINSPACE& fes, const
    gridView_t& gv, const INTERS& boundary_inters) {
  const auto f = [](const coords_t& x){ return std::cos(x.norm()); };
  typedef NPDE::LocalVectorAssembler RhsAssembler_t;
  typedef NPDE::LoadVectorAssembler<RhsAssembler_t> linearForm_t;
  linearForm_t linearForm;
  const Eigen::VectorXd & rhs = linearForm.assembleRhs(fes, f);

  typedef NPDE::AnalyticStiffnessLocalAssembler LocalMatAssembler_t;
  typedef NPDE::GalerkinMatrixAssembler<LocalMatAssembler_t> BilinearForm_t;
  BilinearForm_t bilinearForm;
  auto A_trips = bilinearForm.assembleTripletMatrix(fes, fes, gv);

  const auto gamma = [](const coord_t& x) { return 1.0; }
  typedef NPDE::LaplRobinLocalMatrixAssembler LocalMatAssembler2_t;
  typedef NPDE::IntersectionGalMatAsse<LocalMatAssembler2_t>
                   BilinearForm2_t;
  BilinearForm2_t bilinearForm2;
  auto A2_trips = bilinearForm2.assembleTripletMatrix(fes, fes,
                                   gamma, boundary_inters);

  A_trips.insert(A_trips.end(), A2_trips.begin(), A2_trips.end());
  Eigen::SparseMatrix<numeric_t> A(fes.numDofs(), fes.numDofs());
  A.setFromTriplets(A_trips.begin(), A_trips.end());

  typedef Eigen::SparseLU<Eigen::SparseMatrix<numeric_t>> solver_t;
  solver_t solver;
  solver.compute(A);
  return solver.solve(rhs);
}
```

Schüttler, Janik

Formulas

First Green $$\int_\Omega j \nabla v \, dx = -\int_\Omega \operatorname{div} j \, v \, dx + \int_{\partial\Omega} j \cdot n \, v \, dS$$

Gauss $$\int_\Omega \operatorname{div} j \, dx = \int_{\partial\Omega} j \cdot n \, dS \quad \forall j \in \left(C^1_{pw}(\bar\Omega)\right)^d$$

Fourier's law $$j(x) = -\kappa(x) \nabla u(x)$$
$$j(x) = -\kappa(x) \nabla u(x) + v(x) \rho u(x)$$

Energy conservation $$\int_{\partial V} j \cdot n \, dS = \int_V f \, dx$$

Trapezoidal rules $$\int_a^b f \, dx \approx \frac{b-a}{2}\left(f(a) + f(b)\right)$$
$$\int_a^b f \, dx \approx \frac{1}{2} \sum_{l=1}^{n} h_l \left(f(x_{l-1}) + f(x_l)\right)$$
$$\int_K f \, dx \approx \frac{|K|}{3}\left(f(a^1) + f(a^2) + f(a^3)\right)$$

Composite Midpoint $$\int_a^b f \, dx \approx \sum_{l=1}^{n} h_l f\left(\frac{x_l + x_{l-1}}{2}\right)$$

Energies
kinetic $$\frac{1}{2} m(\dot u, \dot u) = \frac{1}{2} \dot\mu^T M \mu^T$$
elastic $$\frac{1}{2} a(\dot u, \dot u) = \frac{1}{2} \dot\mu^T A \mu^T$$

Taylor $$f(x+h) = f(x) + Df(x) h + O(h^2) \quad h \to 0$$
$$\begin{aligned} F(x+\delta x, y+\delta y) &= F(x,y) + \partial_x F(x,y)\delta x + \\ &+ \partial_y F(x,y)\delta y + \tfrac{1}{2}\partial_x^2 F(x,y)\delta x^2 + \partial_x \partial_y F(x,y)\delta x \delta y \\ &+ \tfrac{1}{2} \delta y^T \partial_y^2 F(x,y) \delta y + O(|\delta x|^3 + \|\delta y\|^3) \end{aligned}$$

Inequalities
$$\left|\int u v \, dx\right|^2 \le \int |u|^2 dx \cdot \int |v|^2 dx \qquad (CS)$$
$$|\langle u, v \rangle| \le \|u\| \, \|v\|$$
$$\|u_0\|_{L^2(\Omega)} \le \operatorname{diam}(\Omega) \|\nabla u_0\|_{L^2(\Omega)} \qquad (PF1)$$
$$\|u_*\|_{L^2(\Omega)} \le C \operatorname{diam}(\Omega) \|\nabla u_*\|_{L^2(\Omega)} \qquad (PF2)$$
$$\|u\|^2_{L^2(\partial\Omega)} \le C \|u\|_{L^2(\Omega)} \|u\|_{H^1(\Omega)} \qquad \left(\begin{array}{c}\text{multip.}\\ \text{trace}\end{array}\right)$$

Explicit Euler $$\Psi^{t, t+\tau} u = u + \tau f(t, u)$$
Implicit Euler $$\Psi^{t, t+\tau} u = u + \tau f(t, \Psi^{t, t+\tau} u)$$
Implicit Midpoint $$\Psi^{t, t+\tau} u = k, \quad k = u + \tau f\left(t + \tfrac{1}{2}\tau, \frac{k+u}{2}\right)$$
Runge Kutta $$k_i = f\left(t + c_i \tau, u + \sum_{j=1}^{s} a_{ij} k_j\right)$$
$$\Psi^{t, t+\tau} u = u + \tau \sum_{i=1}^{s} b_i k_i \qquad \begin{array}{c|c} c & A \\ \hline & b^T \end{array}$$

Rewriting 2nd order ODEs
$$\ddot\omega = g(t, \omega) \iff \dot u = \begin{pmatrix} \dot\omega \\ \dot v \end{pmatrix} = \begin{pmatrix} v \\ g(t, \omega) \end{pmatrix} = f(t, u)$$

Convection Diffusion equations

2nd order, diffusive		1st order, convective			
$-\operatorname{div}(\kappa \nabla u)$	$+$	$\operatorname{div}(\rho V u)$	$=$	f	
$-\kappa \Delta u$	$+$	$\rho V \nabla u$	$=$	f	$(\operatorname{div} v = 0)$
$-\varepsilon \Delta u$	$+$	$V \nabla u$	$=$	f	$(\operatorname{div} v = 0)$

$$\partial_t(\rho u) + \operatorname{div}(-\kappa \nabla u + \rho V u) = f$$
$$\dot u - \varepsilon \Delta u + V \nabla u = f \qquad (\operatorname{div} v = 0)$$