Fast prototyping for polytopal numerical schemes

10 years of DiSk++ (almost)

Matteo Cicuttin

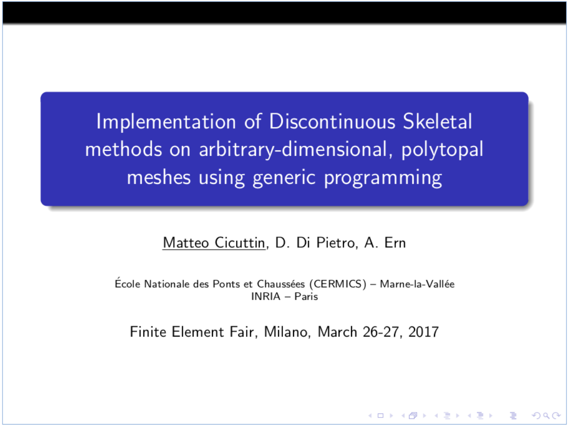
Politecnico di Torino

“Towards Polytopal meshes in Gmsh” workshop
January 26-27, 2026, Montpellier

Some history

DiSk++ is a **software** library to develop **polyhedral methods for PDEs**.

- Started in 2016 with focus on HHO.
- One of the first presentations (likely the first) of Disk++ at EFEF2017 in Milan.
- In these years we implemented many exciting features and we support various polyhedral methods.
- Built a small community of devels.
- Curious that we *did not* reach version 1.0 yet!



GMSH and me

GMSH is a central tool in my work and well integrated into my codes:

- **DiSk++**: a polyhedral library for PDEs (https://github.com/wareHHOuse/diskpp)
- **GMSH/DG**: a massively parallel, GPU-accelerated Discontinuous Galerkin code for conservation laws (https://gitlab.onelab.info/gmsh/dg)
- **FRICO**: a Method of Moments (=BEM) code for simulating antennas and scatterers (https://github.com/datafl4sh/frico) (brand new!)

DiSk++ can use GMSH prismatic elements and GMSH/METIS agglomeration as polytopal elements. Currently using agglomeration in a work about multilevel preconditioners with Tommaso Vanzan.

Discontinuous Skeletal methods

DiSk++ is specialized in Discontinuous and/or Skeletal methods, and is written in C++.

- Discontinuous: piecewise polynomial approximation that jumps on interfaces between elements
- Skeletal: unknowns of the global problem placed on the mesh skeleton

The main assets:

- Arbitrary polynomial order
- Arbitrary element shape
- Dimension-independent formulation
- Simple *hp*-refinement

The main players:

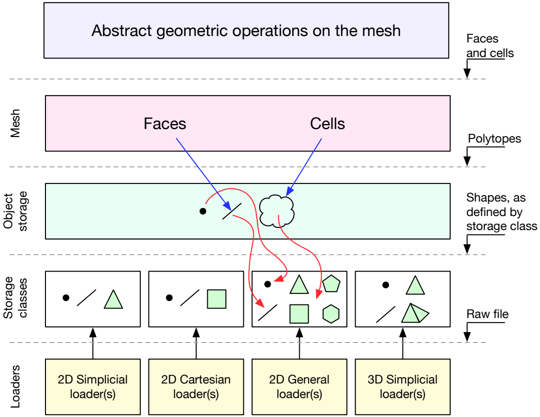
- (Hybrid) Finite Volumes (FV/HFV)
- (Hybrid) Discontinuous Galerkin (DG/HDG)
- Virtual Elements (VEM)
- Hybrid High-Order (HHO)
- MFD/HFV/WG and others...

DiSk++ goals and structure

For many polygonal methods, their mathematical treatment does not care about spatial dimension or specific mesh element shape.

You care only about mesh cells and mesh faces.

DiSk++ goal: provide in software the same level of abstraction that you have in the mathematical definition of the method.



Meshes

```
using mesh_type = disk::simplicial_mesh<T,2>;
mesh_type msh;
auto mesher = disk::make_simple_mesher(msh);
for (auto nr = 0; nr < num_refs; nr++)
    mesher.refine();
```

Automatic meshers for the unit square/cube

- Declare mesh object & construct mesher
- Refine by subdivision as you like
- Ideal for "academic experiments"

Full GMSH integration

- Import directly GMSH geometries
- Ideal for real-world computations

```
using mesh_type = disk::simplicial_mesh<T,3>;
mesh_type msh;
disk::gmsh_geometry_loader<mesh_type> loader;
loader.read_mesh(argv[1]); /* Read GMSH .geo */
loader.populate_mesh(msh);
```

For other shapes than `simplicial`, there are the `cartesian` and `generic` categories.

Other formats also supported: Netgen, FVCA5, FVCA6.

Iterating on mesh elements

Once you have a mesh loaded, you would like to iterate on its elements. In a dimension independent and element-shape independent way of course.

```
for (auto& cl : msh) { /* for each element */
  auto cmeas = measure(msh, cl);
  auto cbar = barycenter(msh, cl);
  auto fcs = faces(msh, cl); /* get faces of cl */
  for (auto& fc : fcs) { /* for each face of cl */
    auto fmeas = measure(msh, fc);
    auto fbar = barycenter(msh, fc);
  }
}
```

- measure() on cells will automatically give volume/area/length
- measure() on faces will automatically give area/length/1
- the same goes barycenter() and all the other geometrical functions of DiSk++

This code will work on any mesh: you don't need to care about dimension or element shape. At all.

Basis functions

DiSk++ employs the scaled monomials as basis functions. Appropriate rescaling [MC '25] allows to keep "under control" matrix condition numbers.

```
for (auto& cl : msh)
{
  using namespace disk::basis;
  auto phi = scaled_monomial_basis(msh, cl, degree);
  auto bar = barycenter(msh, cl);
  auto val_phi = phi(bar);
  auto gradphi = grad(phi);
  auto val_gradphi = gradphi(bar);
}
```

When you ask for a basis on an element, you get a *functor* (in the C++ sense)

- Functors are callable on points and return all the basis funcs evaluated at that point → exactly $\mathbb{P}_d^k(T)$
- You can apply differential operators to functors (WIP)

A remark about the scaled monomials

Let me take a little detour: scaled monomials are known to lead to ill-conditioned matrices, **but...**

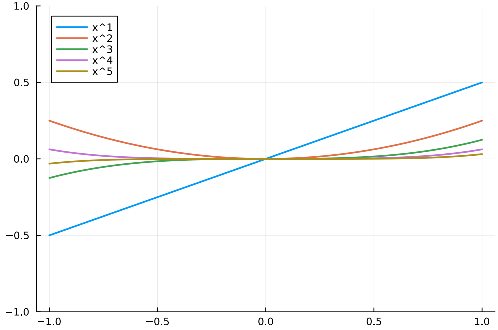
Definition (Scaled monomials usually found in literature)

Let T be a mesh element, x_i the i-th coordinate of a point in T, $x_{T,i}$ the i-th coordinate of the barycenter of T and h_T its diameter. In addition, let $\alpha \in \mathbb{N}^d$ be a multi-index with magnitude $|\alpha| := \sum_{1 \leqslant i \leqslant d} \alpha_i$. The usual definition of scaled monomials, for all $\alpha \in \mathbb{N}^d$, is

$$\mu_{T,\alpha}(\boldsymbol{x}) := \prod_{1 \leqslant i \leqslant d} \left(\frac{x_i - x_{T,i}}{h_T} \right)^{\alpha_i},$$

...this definition is **not OK**.

How the "usual" scaled monomials do look in 1D



If we plot the first few monomials, they *look ugly*...

- The "usual" definition is **incorrectly** scaled
- With this scaling the condition number of the matrices will be unnecessarly and horribly high...
- Sounds trivial? Bear with me, apparently everyone is using this basis...

The 1D mass matrix

Let $T = [a, b]$, $h_T = b - a$ and $x_T = (a + b)/2$. The mass matrix of T is then

$$M_{ij}^{(\eta)} = \int_a^b \left(\eta \frac{x - x_T}{h_T}\right)^i \left(\eta \frac{x - x_T}{h_T}\right)^j dx = \int_a^b \left(\eta \frac{x - x_T}{h_T}\right)^{i+j} dx.$$

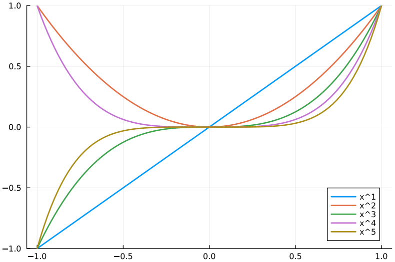
The 1D mass matrix

Let $T = [a, b]$, $h_T = b - a$ and $x_T = (a + b)/2$. The mass matrix of T is then

$$M_{ij}^{(\eta)} = \int_a^b \left(\eta \frac{x - x_T}{h_T}\right)^i \left(\eta \frac{x - x_T}{h_T}\right)^j dx = \int_a^b \left(\eta \frac{x - x_T}{h_T}\right)^{i+j} dx.$$

Computing the integral, the explicit entries of M are

$$M_{ij}^{(\eta)} = \left(\frac{\eta}{2}\right)^{i+j} \frac{h_T}{2} \frac{1 - (-1)^{i+j+1}}{i + j + 1}.$$



If $\eta \neq 2$ the smallest and the largest eigenvalue get pushed far apart very quickly! (Use Gershgorin cirles to see it).

The recipe for higher spatial dimensions

Of course the trick can be extended in arbitrary dimension.
Let's outline how:

- Find the principal axes of T via inertia matrix

$$\int_T (\boldsymbol{x}-\boldsymbol{x}_T)(\boldsymbol{x}-\boldsymbol{x}_T)^T \mathrm{d}\boldsymbol{x} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T \in \mathbb{R}^{d\times d}, \qquad \boldsymbol{x} \in T.$$

- Construct the bounding box aligned with those axes
- Compute the scaled monomials on each direction: let $\lambda_{\max} = \max(\boldsymbol{\Lambda})$ and $\mathbf{B} := 2h_T^{-1}\sqrt{\lambda_{\max}}\sqrt{\boldsymbol{\Lambda}^{-1}}\mathbf{Q}^T$

$$\bar{\mu}_{T,\alpha}(\boldsymbol{x}) := \prod_{i=1}^{d} (\mathbf{B}(\boldsymbol{x}-\boldsymbol{x}_T))_i^{\alpha_i}.$$

- Let the magic happen

The recipe for higher spatial dimensions

Of course the trick can be extended in arbitrary dimension. Let's outline how:

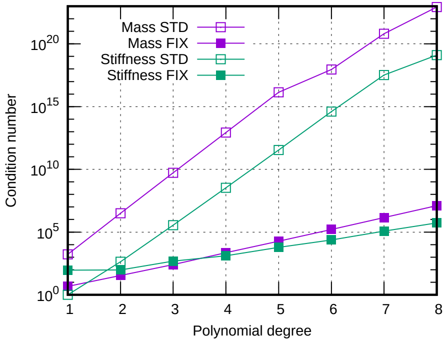
- Find the principal axes of T via inertia matrix

$$\int_T (\boldsymbol{x}-\boldsymbol{x}_T)(\boldsymbol{x}-\boldsymbol{x}_T)^T \mathrm{d}\boldsymbol{x} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T \in \mathbb{R}^{d\times d}, \qquad \boldsymbol{x} \in T.$$

- Construct the bounding box aligned with those axes
- Compute the scaled monomials on each direction: let $\lambda_{\max} = \max(\mathbf{\Lambda})$ and $\mathbf{B} := 2h_T^{-1}\sqrt{\lambda_{\max}}\sqrt{\mathbf{\Lambda}^{-1}}\mathbf{Q}^T$

$$\bar{\mu}_{T,\alpha}(\boldsymbol{x}) := \prod_{i=1}^{d}(\mathbf{B}(\boldsymbol{x}-\boldsymbol{x}_T))_i^{\alpha_i}.$$

- Let the magic happen



Results above on a stretched and rotated pentagon, more details in [MC '25].

Local linear and bilinear forms

```
auto f [](const point& p) {
  return sin(M_PI*p.x())*sin(M_PI*p.y());
};
auto phi = scaled_monomial_basis(msh, cl, degree);
auto RHS = integrate(msh, cl, f, phi);
```

Let's say that $(f, v_h)_T$ is the classical local RHS of the usual Laplacian model problem

← here is how you write it in DiSk++

The function `f` does not need to be hardcoded: DiSk++ can call external scripts written in Lua that allow you to define stuff without recompiling everything.

Let's now build the local stiffness matrix from the usual grad-grad $(\nabla u_h, \nabla v_h)_T$

here is how you write it in DiSk++ →

```
auto phi = scaled_monomial_basis(msh, cl, degree);

auto K = integrate(msh, cl, grad(phi) , grad(phi));
```

Symmetric Interior Penalty DG: recall

Let mesh $\mathcal{T}$, skeleton Γ. DG space: piecewise d-variate polynomials of degree k.

$$V_h = \{v_h \in L^2(\Omega) : v_h|_T \in \mathbb{P}_d^k(T), \forall T \in \mathcal{T}\}.$$

Symmetric Interior Penalty DG bilinear form for Laplacian [Georgoulis '11; Di Pietro, Ern '12]:

$$a_h^{sip}(v, w_h) = \sum_{T\in\mathcal{T}} \int_T \nabla_h v \cdot \nabla_h w_h - \sum_{F\in\Gamma} \int_F \{\nabla_h v\} \cdot \boldsymbol{n}_F [\![w_h]\!] - \sum_{F\in\Gamma} \int_F [\![v]\!] \{\nabla_h w_h\} \cdot \boldsymbol{n}_F + \sum_{F\in\Gamma} \int_F \frac{\eta}{h_F} [\![v]\!][\![w_h]\!]$$

Find $u_h \in V_h$ s.t.

$$a_h^{sip}(u_h, v_h) = \sum_{T\in\mathcal{T}} \int_T f v_h \qquad \text{for all } v_h \in V_h$$

Discontinuous Galerkin: the actual code

```
auto tbasis = disk::basis::scaled_monomial_basis(msh, tcl, degree);

matrix_type K = integrate(msh, tcl, grad(tbasis), grad(tbasis));
vector_type loc_rhs = integrate(msh, tcl, f, tbasis);

auto fcs = faces(msh, tcl);
for (auto& fc : fcs)
{
    auto n     = normal(msh, tcl, fc);
    auto eta_l = eta / diameter(msh, fc);

    auto nv = cvf.neighbour_via(msh, tcl, fc);
    if (nv) {
        matrix_type Att = matrix_type::Zero(tbasis.size(), tbasis.size());
        matrix_type Atn = matrix_type::Zero(tbasis.size(), tbasis.size());

        auto ncl = nv.value();
        auto nbasis = disk::basis::scaled_monomial_basis(msh, ncl, degree);
        assert(tbasis.size() == nbasis.size());

        Att += + eta_l * integrate(msh, fc, tbasis, tbasis);
        Att += - 0.5 * integrate(msh, fc, grad(tbasis).dot(n), tbasis);
        Att += - 0.5 * integrate(msh, fc, tbasis, grad(tbasis).dot(n));

        Atn += - eta_l * integrate(msh, fc, nbasis, tbasis);
        Atn += - 0.5 * integrate(msh, fc, grad(nbasis).dot(n), tbasis);
        Atn += + 0.5 * integrate(msh, fc, nbasis, grad(tbasis).dot(n));
```

For each element of the mesh:

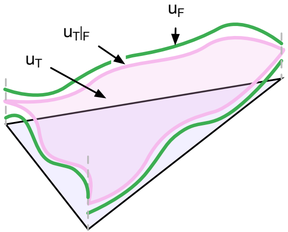
- build the grad-grad term and the source term

Then, iterate on the faces of the current element and build:

- the consistency term
- the symmetry term
- the penalization term
- (notice that $\{\cdot\}$s and $[\![\cdot]\!]$s were expanded)

HHO recall: the reconstruction operator

HHO attaches polynomials to both mesh cells and mesh faces. Then, a reconstruction operator reconstructs a higher-order polynomial in the cells. [Di Pietro, Ern, Lemaire '14; MC, Ern, Pignet '21]



- $u_T \in \mathbb{P}_d^{k'}(T)$: cell-based polynomial, $k' \in \{k-1, k, k+1\}$
- $u_{F_i} \in \mathbb{P}_{d-1}^{k}(F_i)$: face-based polynomials
- $u_{\partial T}$: $(u_{F_1}, \dots, u_{F_n})$
- $U_T^{k',k} := \mathbb{P}_d^{k'}(T) \times \mathbb{P}_{d-1}^{k}(\partial T)$, $\underline{u}_T := (\mathrm{u}_T, \mathrm{u}_{\partial T}) \in U_T^{k',k}$
- Local reconstruction operator $\mathrm{R} : U_T^{k',k} \to \mathbb{P}_d^{k+1}(T)$

The reconstruction is defined from an integration by parts formula, plus average fixing (not shown)

$$(\nabla \mathrm{R}(u_T, u_{\partial T}), \nabla v)_T := (u_T, \Delta v)_T + \sum_{F_i \in \partial T} (u_{F_i}, \nabla v \cdot \mathbf{n})_{F_i}$$

$$= (\nabla u_T, \nabla v)_T + \sum_{F_i \in \partial T} (u_{F_i} - u_T, \nabla v \cdot \mathbf{n})_{F_i}$$

HHO reconstruction: the actual code

```
dynamic_matrix<T> stiffness = integrate(msh, cl, grad(phiR), grad(phiR));
dynamic_matrix<T> lhs = stiffness.bottomRightCorner(szR-1, szR-1);
dynamic_matrix<T> rhs = dynamic_matrix<T>::Zero(rows, cols);
rhs.block(0,0,szR-1,szT) = stiffness.bottomLeftCorner(szR-1, szT);

size_t offset = szT;
for (const auto& fc : fcs)
{
    auto n = normal(msh, cl, fc);
    auto phiF = Space::face_basis(msh, fc, di.face);

    rhs.block(0,offset,szR-1,szF) +=
        integrate(msh, fc, phiF, grad(phiR).dot(n)).block(1,0,szR-1,szF);

    rhs.block(0,0,szR-1,szT) -=
        integrate(msh, fc, phiT, grad(phiR).dot(n)).block(1,0,szR-1,szT);

    offset += szF;
}

dynamic_matrix<T> R = lhs.ldlt().solve(rhs);
dynamic_matrix<T> A = rhs.transpose()*R;
return std::pair(R, A);
```

1st `integrate()`: $(\nabla R(u_T, u_{\partial T}), \nabla v)_T$

extract $(\nabla u_T, \nabla v)_T$

for loop: $\sum_{F_i \in \partial T}$

2nd `integrate()`: $(u_{F_i}, \nabla v \cdot \mathbf{n})_{F_i}$

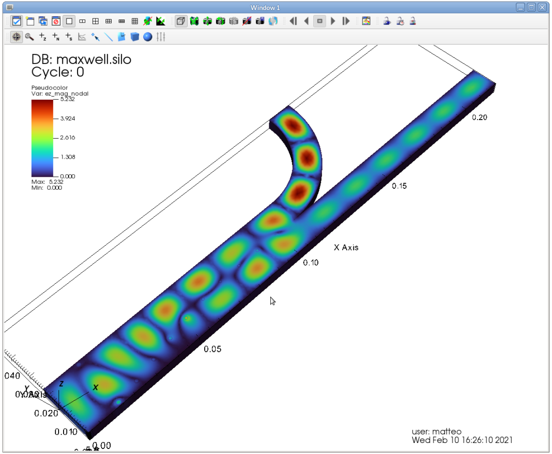
3rd `integrate()`: $-(u_T, \nabla v \cdot \mathbf{n})_{F_i}$

Data visualization

For historical reasons, DiSk++ does its data visualization via LLNL Silo/Visit. But if you are more comfortable with Paraview, it can read DiSk++ output.

Silo is deeply integrated in DiSk++ and exporting your data is a matter of lines

```
std::string filename = "mydb.silo";
disk::silo_database db;
db.create(filename);
db.add_mesh(msh, "mesh");
db.add_variable("mesh", "u", u_data,
    disk::zonal_variable_t);
```



Applications

DiSk++ is a library for polyhedral methods, but is also a collection of applications.

We have plenty of them:

- Poisson
- Linear elasticity
- Solid mechanics
- Fluid mechanics
- Electromagnetics
- Eigenvalues

Users can add new applications easily: the `./newapp.sh` script in the source tree creates an empty template application and adds it to the build system.

HHO for the indefinite Maxwell problem

Let angular frequency ω, materials μ, ϵ, unknown electric field $\mathbf{e} \in H_0(curl;\Omega)$, test function $\mathbf{v} \in H_0(curl;\Omega)$, source $\mathbf{f}$. Solve time-harmonic wave equation:

$$(\nabla \times \mathbf{e}, \nabla \times \mathbf{v})_\Omega - \omega^2 \mu\epsilon(\mathbf{e}, \mathbf{v})_\Omega = (\mathbf{f}, \mathbf{v})_\Omega, \quad \forall \mathbf{v} \in H_0(curl;\Omega).$$

Vector-valued local HHO spaces

$$\underline{\mathsf{U}}_T^k := \mathbb{P}_3^k(T)^3 \times \left\{ \bigtimes_{F \in \partial T} \mathbb{P}_2^k(F)^2 \right\}.$$

- $\mathbb{C}^3$-valued cell-based polynomials
- face tangent $\mathbb{C}^2$-valued face polys
- Global spaces and Dirichlet BCs as usual in HHO

Curl reconstruction $\mathcal{C} : \underline{\mathsf{U}}_T^k \to \mathbb{R}^3$ [Chave, Di Pietro, Lemaire]:

$$(\mathcal{C}(\underline{\mathsf{u}}_T), \mathbf{v})_T := (\mathsf{u}_T, \nabla \times \mathbf{v})_T + \sum_{F \in \partial T} (\mathsf{u}_F, \mathbf{v} \times \mathbf{n})_F, \ \forall \mathbf{v} \in \mathbb{P}_3^k(T)^3$$

Lerhenfeld-Schöberl stabilization. Let $\gamma_{t,F}(\mathbf{u}) := \mathbf{n} \times (\mathbf{u} \times \mathbf{n})$ and $\pi_\gamma^k = \pi_F^k \circ \gamma_{t,F}$:

$$s_T(\underline{\mathsf{u}}_T, \underline{\mathsf{v}}_T) := \sum_{F \in \partial T} \omega \sqrt{\frac{\epsilon}{\mu}} (\mathsf{u}_F - \pi_\gamma^k(\mathsf{u}_T), \mathsf{v}_F - \pi_\gamma^k(\mathsf{v}_T))_F,$$

HHO memory usage and floating point operations

Curl-curl is difficult for iterative solvers: usually direct + domain decomposition $\Longrightarrow$ memory efficiency is of primary importance. Resonator $[0,1]^3$, tetrahedral mesh, 3072 elements:

	HHO(k,k)		SIP-DG(k)[1]	
Degree	Memory	Mflops	Memory	Mflops
k=1	0.5 Gb	8.723	0.3 Gb	20.040
k=2	0.9 Gb	66.759	2.4 Gb	313.133
k=3	2.6 Gb	309.072	9.3 Gb	2.560.647

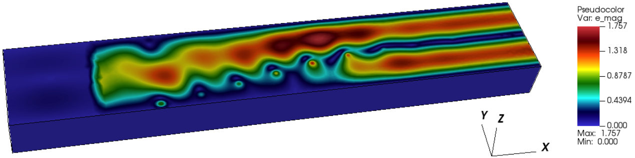
Computation 8.3x improvement, memory 3.5x improvement.

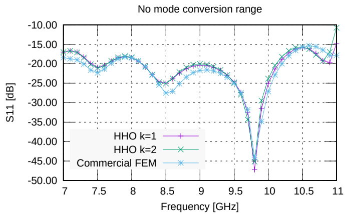
Mesh h	k	Error	Mflops	DOFs	Memory
0.103843	2	3.56e-5	4089984	571392	11.7 Gb
0.207712	3	1.38e-5	309072	115200	2.6 Gb
0.415631	4	1.98e-5	16287	20160	0.5 Gb
0.832917	6	1.24e-5	1265	4032	0.1 Gb

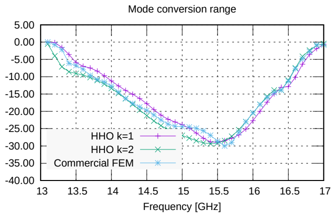
[1] SIP-DG for Maxwell: [Houston, Perugia, Schneebeli, Schötzau '05]

Study of a waveguide mode converter

A "polytopal" use of GMSH: due to the solution features, the mode converter was meshed with a layer of triangular prisms.







MC, Geuzaine - Numerical investigation of a 3D hybrid high-order method for the indefinite time-harmonic Maxwell problem - FINEL '24

Stabilization-free HHO

Model problem: for $f \in L^2(\Omega)$, find $u \in H_0^1(\Omega)$ s.t.

$$(\nabla u, \nabla v)_{L^2(\Omega)} = (f, v)_{L^2(\Omega)} \qquad \forall v \in H_0^1(\Omega)$$

Let R reconstruction operator discussed before. HHO discrete problem forms:

$$a_h(\underline{u}_h, \underline{v}_h) := \sum_{T \in \mathcal{T}} (\nabla R(\underline{u}_T), \nabla R(\underline{v}_T))_T + s_T(\underline{u}_T, \underline{v}_T),$$
$$l_h(\underline{v}_h) := \sum_{T \in \mathcal{T}} (f, \underline{v}_T)_T.$$

Direct correspondence only for grad-grad and RHS terms. Stabilization term is not physical and is there "only" to make the discrete formulation work.

Can we get rid of stabilization term?

Borio, Cascavita, MC, Marcon - Towards stabilization-free Hybrid High-Order methods for elliptic problems - JSC '25

Another route to stability: changing the reconstruction space

It is well known that by changing/enriching the reconstruction space methods like HHO [Abbas, Ern, Pignet '18], WG [Ye, Zhang '21] and VEM [Berrone, Borio, Marcon '21 & '24] can be made stable without adding a stabilization term.

Current HHO/WG variants w/o stabilization have one or more of the following limitations:

- only simplicial elements
- no consecutive collinear edges
- reduced convergence rate ($k + 1$ instead of $k + 2$ in L^2-norm)
- not optimal in terms of additional degrees of freedom required
- subtriangulations required

We propose an approach which is not subject to those limitations.

Another route to stability: changing the reconstruction space

It is well known that by changing/enriching the reconstruction space methods like HHO [Abbas, Ern, Pignet '18], WG [Ye, Zhang '21] and VEM [Berrone, Borio, Marcon '21 & '24] can be made stable without adding a stabilization term.

Current HHO/WG variants w/o stabilization have one or more of the following limitations:

- only simplicial elements
- no consecutive collinear edges
- reduced convergence rate ($k+1$ instead of $k+2$ in L^2-norm)
- not optimal in terms of additional degrees of freedom required
- subtriangulations required

We propose an approach which is not subject to those limitations.

We enrich the reconstruction space as follows: Let $\mathbb{H}_d^k(T)$ be the the space of **harmonic polynomials** of a given degree k defined on T. For an **increment** $\ell \geqslant 0$

$$\tilde{\mathbb{P}}_d^{k',\ell}(T) := \mathbb{P}_d^{k'+2}(T) \oplus \left(\mathbb{H}_d^{k'+2+\ell}(T) \setminus \mathbb{H}_d^{k'+2}(T)\right) = \{p \in \mathbb{P}^{k'+2+\ell}(T) \colon \Delta p \in \mathbb{P}^{k'}(T)\},$$

Stab-free reconstructions

If $k' = k+1$, (mixed-order high) we define $\hat{\mathrm{R}}^{k+1,k}_{\ell,T} : \underline{\mathrm{U}}^{k+1,k}_T \to \tilde{\mathbb{P}}^{k+1,\ell}_d(T)$ such that, $\forall \underline{\mathrm{v}}_T \in \underline{\mathrm{U}}^{k+1,k}_T$,

$$\begin{cases} (\nabla \hat{\mathrm{R}}^{k+1,k}_{\ell,T}(\underline{\mathrm{v}}_T), \nabla q)_T = -(\mathrm{v}_T, \Delta q)_T + (\mathrm{v}_{\partial T} - \hat{\mathcal{E}}^{k+1,k}_T(\underline{\mathrm{v}}_T), \mathbf{n}\cdot\nabla q)_{\partial T} \quad \forall q \in \tilde{\mathbb{P}}^{k+1,\ell}_d(T), \\ (\hat{\mathrm{R}}^{k+1,k}_{\ell,T}(\underline{\mathrm{v}}_T), 1)_T = (\mathrm{v}_T, 1)_T, \end{cases}$$

where

$$\hat{\mathcal{E}}^{k+1,k}_T(\underline{\mathrm{v}}_T) := \pi^k_{\partial T}\mathrm{v}_T - \mathrm{v}_T .$$

If $k' \in \{k-1, k\}$, (mixed-order low/equal order) we define $\check{\mathrm{R}}^{k',k}_{\ell,T} : \underline{\mathrm{U}}^{k',k}_T \to \tilde{\mathbb{P}}^{k',\ell}_d(T)$ such that, $\forall \underline{\mathrm{v}}_T \in \underline{\mathrm{U}}^{k',k}_T$,

$$\begin{cases} (\nabla \check{\mathrm{R}}^{k',k}_{\ell,T}(\underline{\mathrm{v}}_T), \nabla q)_T = -(\mathrm{v}_T, \Delta q)_T + (\mathrm{v}_{\partial T} - \check{\mathcal{E}}^{k',k}_T(\underline{\mathrm{v}}_T), \mathbf{n}\cdot\nabla q)_{\partial T}, \quad \forall q \in \tilde{\mathbb{P}}^{k',\ell}_d(T), \\ (\check{\mathrm{R}}^{k',k}_{\ell,T}(\underline{\mathrm{v}}_T), 1)_T = (\mathrm{v}_T, 1)_T, \end{cases}$$

where

$$\check{\mathcal{E}}^{k',k}_T(\underline{\mathrm{v}}_T) := \pi^k_{\partial T}\mathrm{R}^{k',k}_T(\underline{\mathrm{v}}_T) - \mathrm{R}^{k',k}_T(\underline{\mathrm{v}}_T) .$$

The rationale behind modified space and reconstruction

Let $v \in H^1(T)$ and $\underline{I}_T^{k',k}(v) := (\pi_T^{k'} v, \pi_{\partial T}^{k} v)$ be the HHO reduction operator. A key property (elliptic projection) of the reconstruction operator is that

$$\begin{aligned}(\nabla \mathrm{R}_T^{k',k}(\underline{I}_T^{k',k}(v)), \nabla q)_T &= -(\pi_T^{k'} v, \Delta q)_T + (\pi_{\partial T}^{k} v, \mathbf{n} \cdot \nabla q)_{\partial T} \\ &= -(v, \Delta q)_T + (v, \mathbf{n} \cdot \nabla q)_{\partial T} = (\nabla v, \nabla q)_T\end{aligned}, \qquad \forall q \in \mathbb{P}_d^{k+1}(T)$$

If we take higher-order polynomials as test functions, we can't remove the projectors anymore: the very core of HHO breaks down.

Fortunately, things are easily fixed:

- Take q such that $\Delta q \in \mathbb{P}_d^{k'}(T)$: this fixes the cell-based term and is the reason of the definition of our modified reconstruction space
- Add a correction to the face based term, to remove the error due to $\mathbf{n} \cdot \nabla q \notin \mathbb{P}_d^{k}(\partial T)$

Properties of the stab-free reconstructions

Let $\mathcal{R} \in \left\{\check{\mathrm{R}}^{k',k}_{\ell,T}, \hat{\mathrm{R}}^{k+1,k}_{\ell,T}\right\}$, $\mathcal{E} \in \left\{\check{\mathcal{E}}^{k',k}_{T}, \hat{\mathcal{E}}^{k+1,k}_{T}\right\}$ and $\mathsf{I} := \underline{\mathsf{I}}^{k',k}_{T}(v) := (\pi^{k'}_{T} v, \pi^{k}_{\partial T} v)$. Three properties are easily proved:

- **Orthogonality of corrections w.r.t** $\mathbb{P}^{k+1}_d$ For all $v \in H^1(T)$,

$$(\mathcal{E}(\mathsf{I}(v)), \mathbf{n} \cdot \nabla q)_{\partial T} = 0, \qquad \forall q \in \mathbb{P}^{k+1}_d(T)$$

- **Elliptic projection.** For all $v \in H^1(T)$, the modified reconstructions $\mathcal{R}$ are such that

$$(\nabla \mathcal{R}(\mathsf{I}(v)), \nabla q)_T = (\nabla v, \nabla q)_T, \qquad \forall q \in \mathbb{P}^{k+1}_d(T)$$
$$(\mathcal{R}(\mathsf{I}(v)), 1)_T = (v, 1)_T$$

- **Polynomial consistency.** For all $p \in \mathbb{P}^{k+1}_d(T)$, the modified reconstructions $\mathcal{R}$ are such that

$$\mathcal{R}(\mathsf{I}(p)) = p.$$

With the following properties we can prove (with a caveat) usual HHO high-order convergence rates.

Stability: what the ℓ?

The **stability** of the method is **connected with the choice of** ℓ **in** $\tilde{\mathbb{P}}_d^{k',\ell}(T)$

Conjecture

Let T be a polygon and $\underline{\mathrm{u}}_T \in \underline{\mathrm{U}}_T^{k',k}$. Then, there exists $\ell \in \mathbb{N}$ depending on T, k' and k, and satisfying $\dim \tilde{\mathbb{P}}_d^{k',\ell}(T) \geqslant \dim \underline{\mathrm{U}}_T^{k',k}$, such that

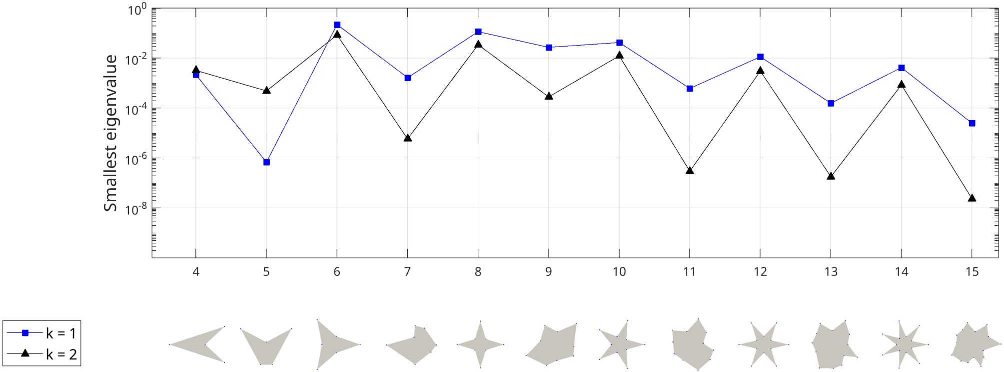
$$\text{if } k' \in \{k-1, k\}, \text{ then there exists } \alpha_T > 0 \text{ such that} \quad \left\| \nabla \check{\mathrm{R}}_{\ell,T}^{k',k}(\underline{\mathrm{u}}_T) \right\|_{\mathrm{L}^2(T)}^2 \geqslant \alpha_T \left| \underline{\mathrm{u}}_T \right|_{\underline{\mathrm{U}}_T^{k',k}}^2,$$

$$\text{if } k' = k+1, \text{ then there exists } \alpha_T > 0 \text{ such that} \quad \left\| \nabla \hat{\mathrm{R}}_{\ell,T}^{k+1,k}(\underline{\mathrm{u}}_T) \right\|_{\mathrm{L}^2(T)}^2 \geqslant \alpha_T \left| \underline{\mathrm{u}}_T \right|_{\underline{\mathrm{U}}_T^{k+1,k}}^2,$$

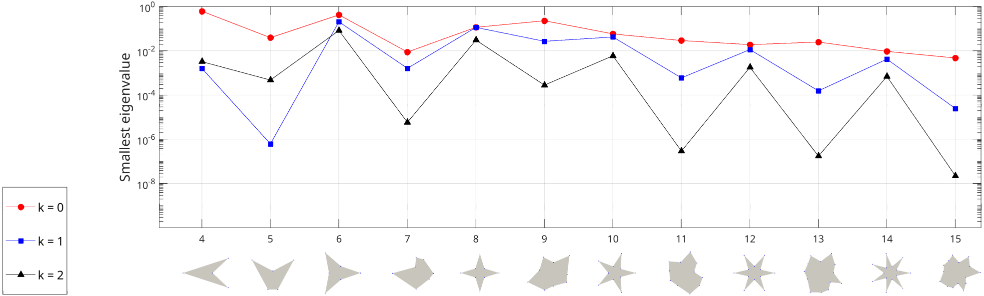
where $|\cdot|_{\underline{\mathrm{U}}_T^{k',k}}^2$ and $|\cdot|_{\underline{\mathrm{U}}_T^{k+1,k}}^2$ are the standard HHO seminorms.

- Numerical evidence hints stability, but did not manage to prove the conjecture yet :(
- Optimality criteria for ℓ: the smallest one such that $\dim \tilde{\mathbb{P}}_d^{k',\ell}(T) \geqslant \dim \underline{\mathrm{U}}_T^{k',k}$ (equality is +/- 1 in 2D, more complicated in 3D).

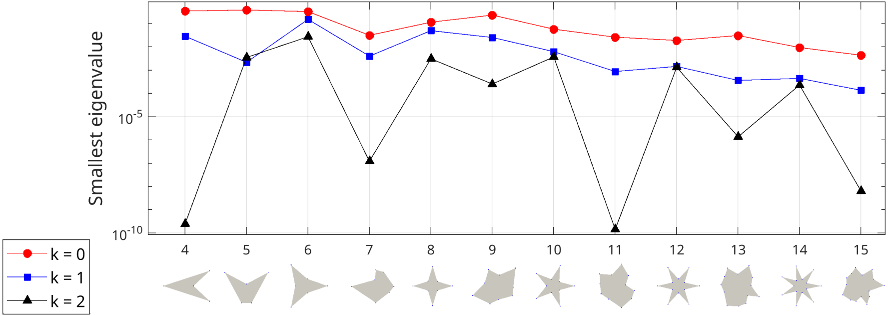
2D: Smallest eigenvalue for HHO($k-1, k$)



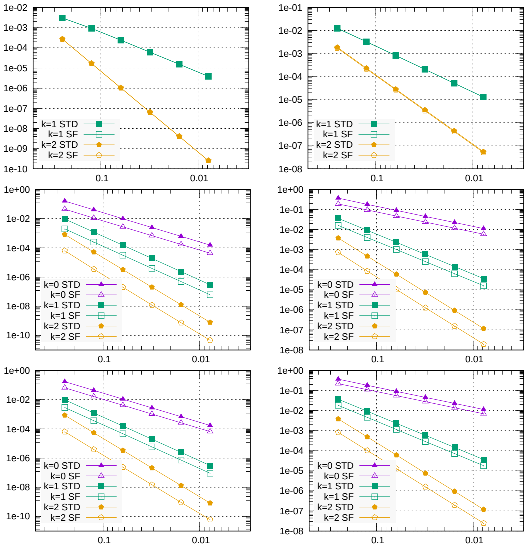
2D: Smallest eigenvalue for HHO(k, k)



2D: Smallest eigenvalue for HHO$(k+1, k)$

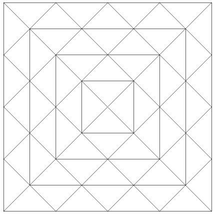


Convergence on simplicial meshes

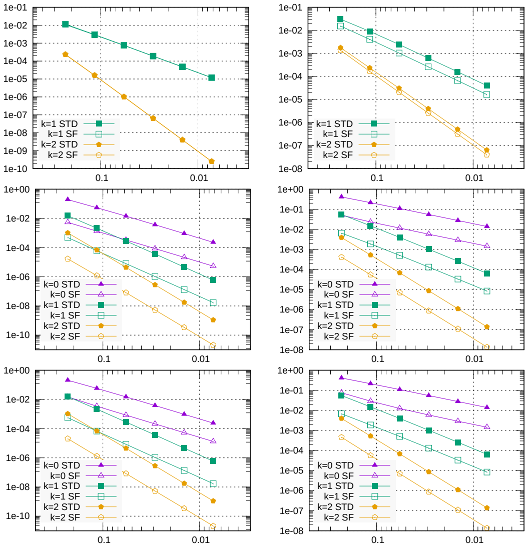


Test on a sequence of triangular meshes

- On the left column, L^2-norm error
- On the right column, energy norm error
- From top to bottom: $\{k-1, k\}$, $\{k, k\}$ and $\{k+1, k\}$ HHO variants
- Full dots is std. HHO, empty dots is our variant

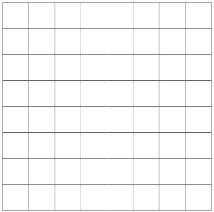


Convergence on cartesian meshes

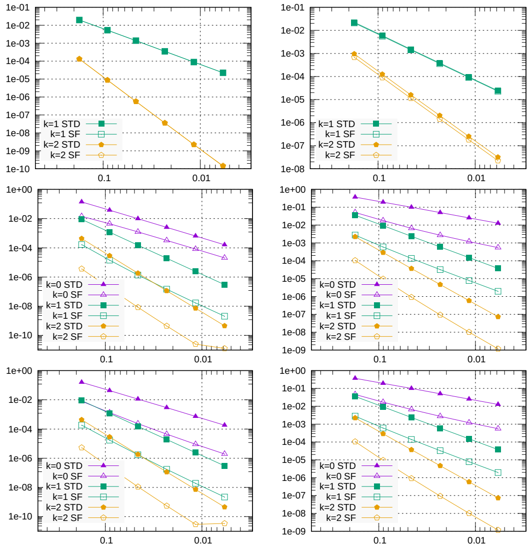


Test on a sequence of cartesian meshes

- On the left column, L^2-norm error
- On the right column, energy norm error
- From top to bottom: $\{k-1, k\}$, $\{k, k\}$ and $\{k+1, k\}$ HHO variants
- Full dots is std. HHO, empty dots is our variant

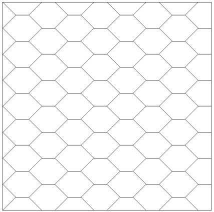


Convergence on hex-dominant meshes



Test on a sequence of hex-dominant meshes

- On the left column, L^2-norm error
- On the right column, energy norm error
- From top to bottom: $\{k-1, k\}$, $\{k, k\}$ and $\{k+1, k\}$ HHO variants
- Full dots is std. HHO, empty dots is our variant



Main DiSk++ assets

- Familiar user experience: write local forms in the `integrate()` style
- Flexible mesh handling simple meshers for unit domains, to make experimentation and convergence tests easy; full integration with GMSH for real world stuff
- Advanced visualization via LLNL Silo/VisIt: all types of mesh and related data can be exported using a simple interface
- Many numerical methods: HHO, DG, DGA, we are in the process of adding VEM
- Many applications: DiSk++ includes tens of applications in various domains
- Getting started is easy: with a single command (`./newapp.sh mynewappname`) you get an empty template application added to the source tree, and you can start coding straight away

What are we still missing?

Thank you for your attention!

matteo.cicuttin@polito.it

DiSk++ is brought to you by

- Karol Cascavita
- Matteo Cicuttin
- Nicolas Pignet

And with significant contributions from

- Omar Duran
- Romain Mottier

GitHub: https://github.com/wareHHOuse/