Lecture - Implementation and numerical aspects of DG-FEM in 2D

Ph.D. Course:
An Introduction to DG-FEM
for solving partial differential equations

Allan P. Engsig-Karup
Scientific Computing Section
DTU Informatics
Technical University of Denmark

August 24, 2009

Course content

The following topics are covered in the course

1. Introduction & DG-FEM in one spatial dimension
2. Implementation and numerical aspects (1D)
3. Insight through theory
4. Nonlinear problems
5. Extensions to two spatial dimensions
6. Introduction to mesh generation
7. Higher-order operators
8. Problem with three spatial dimensions and other advanced topics

Requirements

What do we want?

- A flexible and generic framework useful for solving different problems
- Easy maintenance by a component-based setup
- Splitting of the mesh and solver problems for reusability
- Easy proto-type implementation of solvers for users

Domain of interest

We want to solve a given problem in a domain Ω which is approximated by the union of K nonoverlapping local elements D^k, $k = 1, ..., K$ such that

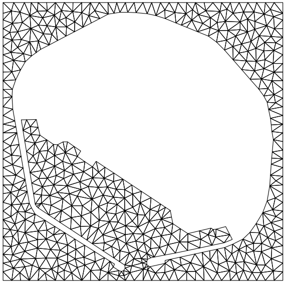
$$\Omega \cong \Omega_h = \bigcup_{k=1}^{K} D^k$$

Thus, we need to deal with implementation issues with respect to the local elements and how they are related.

The shape of the local elements can in principle be of any shape, however, in practice we mostly consider d-dimensional simplexes (e.g. triangles in two dimensions).

Sketch and notations for a two-dimensional domain

Consider a two-dimensional domain defined on $\mathbf{x} \in \Omega_h$

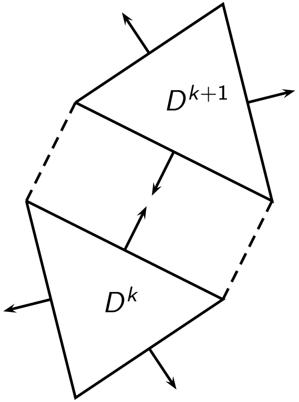

a)Unstructured grid.

b)Photo from a bird's view.
Source: Google Earth.

Figure: Water area surrounding an artificial island called *Middelgrunden* near the Copenhagen harbour in Denmark.

Sketch and notations for a two-dimensional domain

We choose to restrict ourselves to triangles in the following.



Local approximation in 2D

On each of the local elements, we choose to represent the solution locally using N_p grid points sufficient to form a complete multi-dimensional polynomial basis of order N as

$$\mathbf{x} \in D^k : u_h^k(\mathbf{x}, t) = \sum_{n=1}^{N_p} \hat{u}_n^k(t)\psi_n(\mathbf{x}) = \sum_{i=1}^{N_p} u_i^k(t) l_i^k(\mathbf{x})$$

using either a modal or nodal representation.

On the triangle, the relation between number of points N_p and polynomial order N for a complete basis[1] is

$$N_p = \frac{(N+1)(N+2)}{2}$$

[1]cf. Pascals triangle

Local approximation in 2D

For 2D discretizations we may require that our basis functions span a complete 2D polynomial space of order N

$$\mathcal{P}^N = SPAN\{x^\alpha y^\beta\}, \quad \alpha, \beta \geq 0, \alpha + \beta \leq N$$

which leads to a dimension of $\frac{1}{2}(N+1)(N+2)$ basis functions as deduced from Pascal's triangle

$$\begin{array}{ccccccccc}
 & & & & 1 & & & & \\
 & & & \mathrm{x} & & \mathrm{y} & & & \\
 & & x^2 & & \mathrm{xy} & & y^2 & & \\
 & x^3 & & x^2y & & xy^2 & & y^3 & \\
\ldots & & \ldots & & \ldots & & \ldots & & \ldots
\end{array}$$

Note: It is also possible to choice an incomplete basis, or an extended basis. However, any choice of polynomial approximation space will impact the accuracy and aliasing properties of the scheme.

Local approximation in 2D

Similar to the 1D setup we want to map the local solutions to a reference domain where we can define the local operations.

Introduce a linear mapping Ψ which takes a general straight-sided triangle, $\mathbf{x} \in D^k$, to the reference triangle defined as

$$I = \{\mathbf{r} = (r,s) | (r,s) \geq -1; r+s \leq 0\}$$

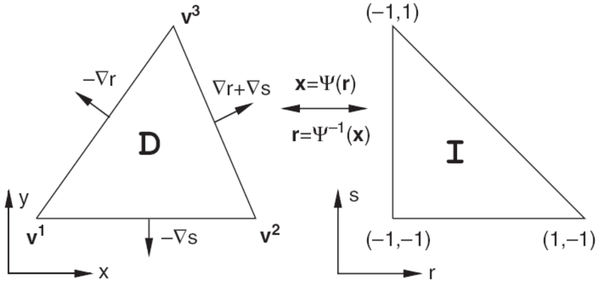


Fig. 6.1. Notation for the mapping between two simplices.

Local approximation in 2D

If D^k is spanned by three vertices v^i, $i = 1, 2, 3$ then

$$\mathbf{x} = \lambda^2 \mathbf{v}^1 + \lambda^3 \mathbf{v}^2 + \lambda^1 \mathbf{v}^3 = \Psi(\mathbf{r})$$

$$\begin{pmatrix} r \\ s \end{pmatrix} = \lambda^2 \begin{pmatrix} -1 \\ -1 \end{pmatrix} + \lambda^3 \begin{pmatrix} 1 \\ -1 \end{pmatrix} + \lambda^1 \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \Psi^{-1}(\mathbf{x})$$

where the barycentric coordinates are defined as

$$\lambda^1 = \frac{s+1}{2}, \quad \lambda^2 = -\frac{r+s}{2}, \quad \lambda^3 = \frac{r+1}{2}$$

The constant metrics of the affine mapping can be found

$$\frac{\partial \mathbf{x}}{\partial \mathbf{r}} \frac{\partial \mathbf{r}}{\partial \mathbf{x}} = \begin{bmatrix} x_r & x_s \\ y_r & y_s \end{bmatrix} \begin{bmatrix} r_x & r_y \\ s_x & s_y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Local approximation in 2D

From which we can deduce the following relationships

$$\begin{array}{lll} r_x = \frac{1}{J} y_s, & r_y = -\frac{1}{J} x_s, & J = x_r y_s - x_s y_r, \\ s_x = -\frac{1}{J} y_r, & s_y = \frac{1}{J} x_r & \end{array}$$

For any two straightsided triangles connected through $\Psi(\mathbf{r})$

$$\mathbf{x}_r = \frac{\mathbf{v}^2 - \mathbf{v}^1}{2}, \quad \mathbf{x}_s = \frac{\mathbf{v}^3 - \mathbf{v}^1}{2}, \quad \mathbf{v}^i = \begin{pmatrix} x^i \\ y^i \end{pmatrix}$$

Thus, for such cases the Jacobian can be expressed as

$$J = \frac{1}{4} \begin{pmatrix} x^2 - x^1 \\ y^2 - y^1 \end{pmatrix} \cdot \begin{pmatrix} y^3 - y^1 \\ -(x^3 - x^1) \end{pmatrix}$$

or more compactly

$$J = \tfrac{1}{4} \hat{\mathbf{t}}_{12} \cdot \hat{\mathbf{n}}_{13} = \tfrac{1}{4} |\hat{\mathbf{t}}_{12}| |\hat{\mathbf{n}}_{13}| \cos\varphi, \quad 0 < \varphi < \tfrac{\pi}{2}$$

Thus $J > 0$ for straighsided triangles with vertices ordered counter-clockwise.

Local approximation in 2D

Furthermore, to determine boundary normal vectors make use of the directional transformation via chain rule

$$\begin{bmatrix} \partial_x \\ \partial_y \end{bmatrix} = \begin{bmatrix} r_x & s_x \\ r_y & s_y \end{bmatrix} \begin{bmatrix} \partial_r \\ \partial_s \end{bmatrix} = \begin{bmatrix} \nabla r & \nabla s \end{bmatrix} \begin{bmatrix} \partial_r \\ \partial_s \end{bmatrix}$$

which is equivalent to the action of stretching and rotation operations on the directional vector.

Thus, the normal vectors in the reference domain can be transformed as follows

$$\begin{aligned} \mathbf{n}_1 &= \begin{pmatrix} 0 \\ -1 \end{pmatrix} \quad \Rightarrow \quad \hat{\mathbf{n}}_1 = -\frac{\nabla s}{||\nabla s||} \\ \mathbf{n}_2 &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \Rightarrow \quad \hat{\mathbf{n}}_2 = \frac{(\nabla r + \nabla s)}{||\nabla r + \nabla s||} \\ \mathbf{n}_3 &= \begin{pmatrix} -1 \\ 0 \end{pmatrix} \quad \Rightarrow \quad \hat{\mathbf{n}}_3 = -\frac{\nabla r}{||\nabla r||} \end{aligned}$$

Local approximation in 2D

For being able to setup our DG-FEM discretization in a generic way, we need procedures for

- computing polynomial expansions
- numerical evaluations of integrals and derivatives
- computing geometric factors

This involves

- identifying an orthonormal polynomial reference basis $\psi_n(\mathbf{r})$ defined on the triangle I
- identifying families of point distributions that leads to good behavior of the multi-dimensional interpolating polynomial defined on I

Local approximation in 2D I

We need to define a reference basis on the reference triangle

$$I = \{\mathbf{r} = (r, s) | (r, s) \geq -1; r + s \leq 0\}$$

To do this a collapsed coordinate system is introduced through the mapping

$$a = 2\frac{1+r}{1-s}, \quad b = s$$

such that the reference triangle basis can be defined on a reference quadrilateral

$$I_q = \{\mathbf{r} = (a, b) | -1 \leq (a, b) \leq 1\}$$

Local approximation in 2D II

which are suitable for using one-dimensional basis functions for the construction of a multi-dimensional basis. In the collapsed coordinate system the reference basis can be defined as

$$\psi_m(\mathbf{r}) = \sqrt{2} P_i^{(0,0)}(a) P_j^{(2i+1,0)}(b)(1-b)^i$$

where $P_n^{(\alpha,\beta)}(x)$ is the n'th order Jacobi polynomial.

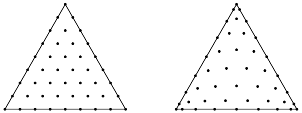
This makes it possible to exploit the orthogonal properties of the one-dimensional basis functions on the triangle.

```
>> P = Simplex2DP(a,b,i,j)
```

For the interpolations on the triangle to be well-behaved we need to identify good positions for the N_p points on I to avoid ill-conditioning issues.

Warp & Blend procedure

The nodes on the simplex can be determined using an optimized explicit Warp & Blend construction procedure [1], cf. **Nodes2D.m**.



Idea is to create for each edge a warp (deformation) function

$$w(r) = \sum_{i=1}^{N_p} (r_i^{LGL} - r_i^{EQUI}) l_i^{EQUI}(r), \quad r \in [-1, 1]$$

that combined with a suitable blending function b^j, $j = 1, 2, 3$, can deform equidistant nodes on the simplex. Then, the sum deformations

$$\mathbf{g}(\lambda^1, \lambda^2, \lambda^3) = \sum_{j=1}^{3} (1 + (\alpha\lambda^j)^2) b^j \mathbf{w}^j$$

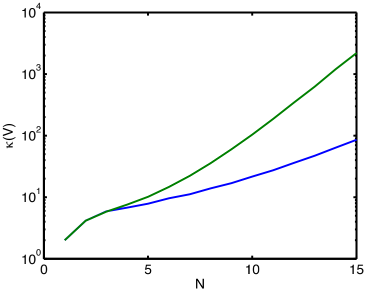
form a set of α-optimized nodes suitable for interpolation.

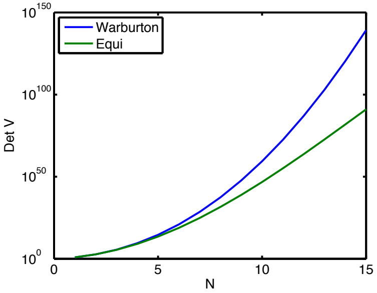
Local approximation in 2D

For stable and accurate computations we need to ensure that the generalized Vandermonde matrix $\mathcal{V}$ is well-conditioned. This implies minimizing the Lebesque constant

$$\Lambda = \max_{\mathbf{r}\in I} \sum_{i=1}^{N_p} |l_i(\mathbf{r})|$$

and maximizing Det $\mathcal{V}$.





Local approximation in 2D

N	Fekete	Warburton α_{opt}	Warburton $\alpha = 0$	Blyth Et. al.	Hesthaven	Equidistant
	Tabular	Explicit	Explicit	Explicit	Tabular	Explicit
1	1.00	1.00	1.00	1.00	1.00	1.00
2	1.67	1.67	1.67	1.67	1.67	1.67
3	2.11	2.11	2.11	2.11	2.11	2.27
4	2.66	2.66	2.66	2.66	2.59	3.47
5	3.12	3.12	3.14	3.14	3.19	5.45
6	4.17	3.70	3.82	3.87	4.07	8.75
7	4.91	4.27	4.55	4.65	4.78	14.35
8	5.90	4.96	5.69	5.92	5.85	24.01
9	6.80	5.74	7.02	7.38	6.88	40.92
10	7.75	6.67	9.16	9.82	8.46	70.89
11	7.89	7.90	11.83	12.90	10.14	124.53
12	8.03	9.36	16.06	17.78	12.63	221.41
13	9.21	11.47	21.17	24.12	-	397.70
14	9.72	13.97	30.33	34.12	-	720.70
15	9.97	17.65	42.48	49.51	-	1315.9

Table: Comparison of Lebesque constants Λ for some popular symmetric nodal distributions on the triangle with 1D optimal Gauss-Lobatto-Legendre distributions on edges.

Local approximation in 2D

The duality between using a modal or nodal interpolating polynomial representation is related through the choice of modal representation $\psi_n(\mathbf{r})$ and the distinct nodal interpolation points $\mathbf{r}_i \in I,\ i = 1, ..., N_p$.

We can express the local approximations as

$$u(\mathbf{r}_i) \cong u_h(\mathbf{r}_i) = \sum_{n=1}^{N_p} \hat{u}_n \psi_n(\mathbf{r}_i) = \sum_{n=1}^{N_p} u_n l_n(\mathbf{r}_i), \quad i = 1, ..., N_p$$

Thus, we find the relationship for the modal and nodal coefficients

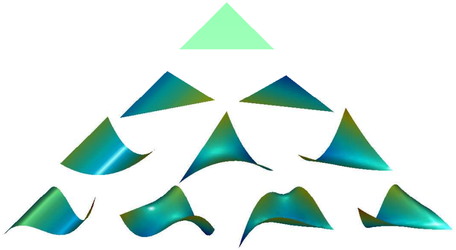
$$\mathbf{u} = \mathcal{V}\hat{\mathbf{u}}$$

where

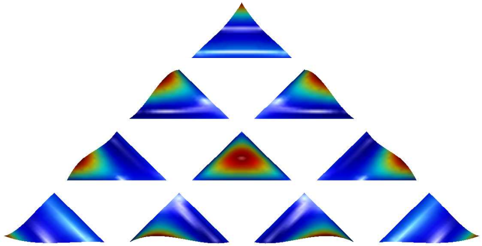
$$\mathcal{V}_{ij} = \psi_j(\mathbf{r}_i), \quad \hat{\mathbf{u}}_\mathbf{i} = \hat{u}_i, \quad \mathbf{u}_\mathbf{i} = u(\mathbf{r}_i)$$

Local approximation in 2D

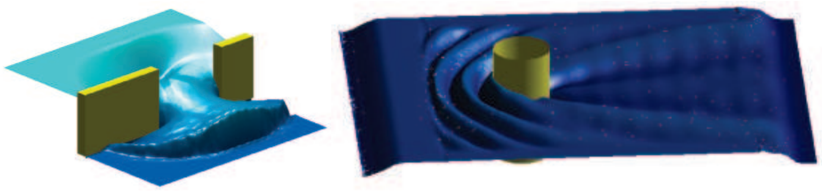
Modal basis:



Nodal basis:



Global approximation



The global solution $u(x,t)$ can then be approximated by the direct summations of the local elemental solutions

$$u_h(x,t) = \bigoplus_{k=1}^{k} u_h^k(x,t)$$

Recall, at the traces of adjacent elements there will be two distinct solutions. Thus, there is an ambiguity in terms of representing the solution.

How to satisfy the PDE

Consider the PDE for a general conservation law

$$\partial_t \mathbf{u} + \nabla f(\mathbf{u}) = 0, \quad \mathbf{x} \in \Omega$$

For the approximation of the unknown solution we make use of the expansion $u_h(x,t)$ and insert it into the PDE.

Doing this, result in an expression for the residual $\mathcal{R}_h(x,t)$. We require that the test functions are orthogonal to the residual function

$$\mathcal{R}_h(\mathbf{x},t) = \partial_t \mathbf{u}_h + \nabla f(\mathbf{u}_h)$$

in the Galerkin sense as

$$\int_{D^k} \mathcal{R}_h(\mathbf{x},t)\psi_n(\mathbf{x})d\mathbf{x} = 0, \quad 1 \le n \le N_p$$

on each of the $k = 1, ..., K$ elements.

Standard notation for elements

It is customary to refer to the interior information of the element by a superscript "-" and the exterior by a "+".

Furthermore, this notation is used to define the average operator

$$\{\{\mathbf{u}\}\} \equiv \frac{\mathbf{u}^- + \mathbf{u}^+}{2}$$

where u can be both a scalar and a vector.

Jumps can be defined along an outward point normal $\hat{\mathbf{n}}$ (to the element in question) as

$$[[u]] \equiv \hat{\mathbf{n}}^- u^- + \hat{\mathbf{n}}^+ u^+$$
$$[[\mathbf{u}]] \equiv \hat{\mathbf{n}}^- \cdot \mathbf{u}^- + \hat{\mathbf{n}}^+ \cdot \mathbf{u}^+$$

Local approximation

As we have seen, in a DG-FEM discretization we can apply a polynomial expansion of arbitrary order within each element.

To exploit this in a code we need procedures for

- computing polynomial expansions (already in place)
- numerical evaluation of integrals and derivatives
- setting up means for stabilization through filtering

Element-wise operations

For implementations local operators needs to be defined.

Consider the mass matrix $\mathcal{M}^k$ for the k'th straightsided element

$$\mathcal{M}^k_{ij} = \int_{D^k} l_i^k(\mathbf{x}) l_j^k(\mathbf{x}) d\mathbf{x} = \mathcal{J}^k \int_I l_i(\mathbf{r}) l_j(\mathbf{r}) d\mathbf{r} = \mathcal{J}^k \mathcal{M}_{ij}$$

with $\mathcal{M}$ the standard mass matrix constructed using an orthonormal basis as

$$\mathcal{M} = (\mathcal{V}\mathcal{V}^T)^{-1}$$

Consider the stiffness matrices $\boldsymbol{\nabla} S^k = (\mathcal{S}^k_x, \mathcal{S}^k_y)^T$ for the k'th element defined as

$$\mathcal{S}^k_{x,ij} = \int_{D^k} l_i^k(\mathbf{x}) \frac{d l_j^k(\mathbf{x})}{dx} d\mathbf{x} = \mathcal{S}_{x,ij}$$

$$\mathcal{S}^k_{y,ij} = \int_{D^k} l_i^k(\mathbf{x}) \frac{d l_j^k(\mathbf{x})}{dy} d\mathbf{x} = \mathcal{S}_{y,ij}$$

Element-wise operations

Reall, in 2D we have from the chain rule the directional mappings

$$\mathcal{D}_x = r_x \mathcal{D}_r + s_x \mathcal{D}_s$$
$$\mathcal{D}_y = r_y \mathcal{D}_r + s_y \mathcal{D}_s$$

where relationships for the metrics was derived earlier.

The differentiation matrices on the simplex is similar to 1D case

$$\mathcal{D}_r = \mathcal{V}^{-1}\mathcal{V}_r, \quad \mathcal{D}_s = \mathcal{V}^{-1}\mathcal{V}_s$$

where the gradients of the basis functions are needed

$$\begin{aligned} \mathcal{V}_{r,(i,j)} &= \left.\frac{\partial \psi_j}{\partial r}\right|_{\mathbf{r}_i} = a_r \left.\frac{\partial \psi_j}{\partial a}\right|_{\mathbf{a}_i}, & a_r &= \frac{2}{1-s} \\ \mathcal{V}_{s,(i,j)} &= \left.\frac{\partial \psi_j}{\partial s}\right|_{\mathbf{r}_i} = a_s \left.\frac{\partial \psi_j}{\partial a}\right|_{\mathbf{a}_i} + \left.\frac{\partial \psi_j}{\partial b}\right|_{\mathbf{a}_i}, & a_s &= -2\frac{(1+r)}{(1-s)^2} \end{aligned}$$

Then, the stiffness matrices can be determined as

$$\mathcal{S}_x = \mathcal{M}^{-1}\mathcal{D}_x, \quad \mathcal{S}_y = \mathcal{M}^{-1}\mathcal{D}_y, \quad \mathcal{S}_r = \mathcal{M}^{-1}\mathcal{D}_r, \quad \mathcal{S}_s = \mathcal{M}^{-1}\mathcal{D}_s$$

Element-wise operations

For stabilization purposes filtering can also be employed in 2D.

Filtering in 2D can be carried out by filtering the expansion coefficients $\hat{u}$ of the reference basis as

$$\mathcal{F}u(\mathbf{x}) = \sum_{i=0}^{N} \sum_{j=0}^{N-i} \sigma\left(\frac{i+j}{N}\right) \hat{u}_{ij}\phi_{ij}(\mathbf{x})$$

In practice, this is implemented through a filter matrix

$$\mathcal{F} = \mathcal{V}\Lambda\mathcal{V}^{-1}$$

where the diagonal spectral filter matrix is

$$\Lambda_{mm} = \sigma\left(\frac{i+j}{N}\right), \quad m = j + (N+1)i + 1 - \frac{i}{2}(i-1),$$

with $(i,j) \geq 0$, $i + j \leq N$. See **Filter2D.m**.

Element-wise operations

A useful filter is the exponential cut-off filter

$$\sigma(i, N_c, \alpha, s) = \begin{cases} 1 & , 0 \leq i < N_c \\ \exp\left(-\alpha \left(\frac{i-N_c}{N-N_c}\right)^s\right) & , N_c \leq i \leq N \end{cases}$$

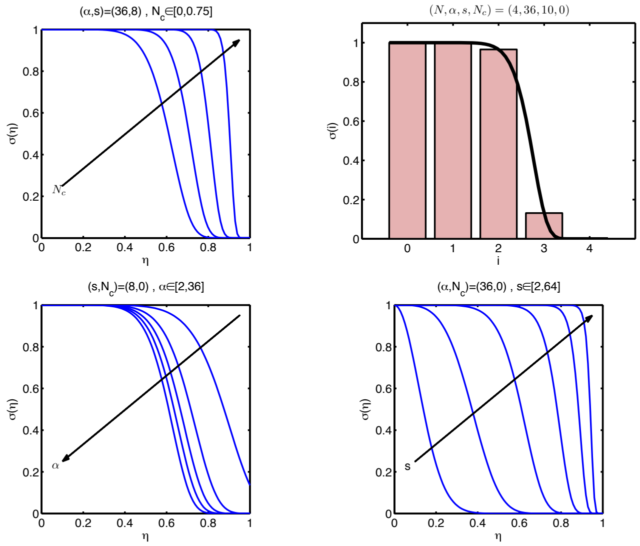
where i is the modal index, $0 \leq N_c \leq N$ is the cut-off frequency (order), α is a tunable parameter, and s is the filter order.

The filter function has the following asymptotic properties

$$\begin{aligned} \lim_{\alpha\to\infty} \sigma(i, N_c, \alpha, s) \to 0, &\quad s \text{ fixed}, N_c \leq i < N \\ \lim_{\alpha\to 0} \sigma(i, N_c, \alpha, s) \to 1, &\quad s \text{ fixed}, N_c \leq i < N \\ \lim_{s\to\infty} \sigma(i, N_c, \alpha, s) \to 1, &\quad \alpha \text{ fixed}, N_c \leq i < N \\ \sigma(i, N_c, \alpha, s) = \exp(-\alpha), &\quad i = N \end{aligned}$$

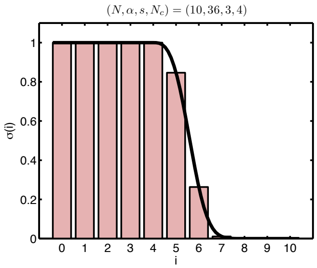
which gives good control in fine-tuning of a damping profile.

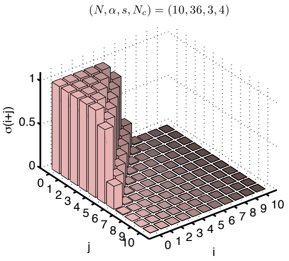
Element-wise operations

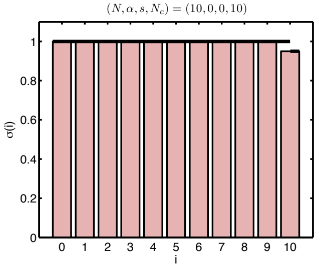


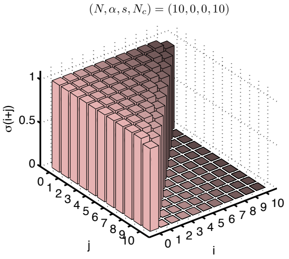
Exponential cut-off filter damping profiles for different parameters.

Element-wise operations









Examples of exponential cut-off filter damping profiles.

Element-wise operations

Integration can be done using the mass matrix (matrix based) or via numerical quadrature or cubature rules in respectively single or multi-dimensions.

- Integration via the mass matrix requires its construction
  - Integral is exact for polynomial orders of at most $2N$ with a polynomial basis of order N.
- Integration via quadrature/cubature rules requires use of specific nodes and weights
  - quadrature/cubature nodes can be reached via interpolation

Pitfalls that might affect the overall accuracy of the scheme

- Aliasing errors in integrand
- Insufficient order of accuracy of rule in question
- Integrand cannot be represented exactly by a polynomial

Element-wise operations

To approximate multi-dimensional integrals, we can use cubature rules which takes the general form

$$\int_I f(\mathbf{r})d\mathbf{r} \cong \sum_{i=1}^{N_c} f(\mathbf{r}_i^c)w_i^c$$

based on a set of N_c nodes, $\mathbf{r}_i^c$, and associated weights, w_i^c.

The order of accuracy of a cubature rule is determined by the maximum order polynomial that can be integrated exactly.

For the approximation of integrals, e.g. inner products, we need to interpolate the integrand to the set of cubature integration points to use the cubature rule.

See **Cubature2D.m**, which gives tabulated nodes and weights for cubature rules on the triangle useful for the exact integration of polynomials of a specified order N in the argument.

Element-wise operations

A summary of useful scripts for the 2D element operators in Matlab

Vandermonde2D	Compute $\mathcal{V}$
GradVandermonde2D	Compute gradients of modal basis $\mathcal{V}_r, \mathcal{V}_s$
xytors	Maps (x,y) to (r,s) coordinates in the triangles
Nodes2D	Compute (x,y) nodes in equilateral triangle
rstoab	Maps (r,s) to (a,b) "collapsed" coordinates
Simplex2DP	Orthonormal basis on the 2D simplex
GradSimplex2DP	Derivatives of orthonormal basis on the 2D simplex
Dmatrices2D	Compute $\mathcal{D}_r, \mathcal{D}_s$
Filter2D	Initialize 2D filter matrix
InterpMatrix2D	Compute local 2D elemental interpolation matrix

A summary of useful scripts for the 2D element operations in Matlab

Grad2D	Compute 2D gradient
Div2D	Compute 2D divergence of vector field
Curl2D	Compute 2D curl operation

Element-wise operations

Examples of element-wise operations (2D) in Matlab

- Compute spatial derivatives ∇u_h

```
>> [ux,uy] = Grad2D(u);
```

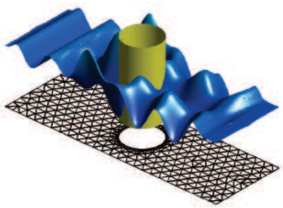
- Compute divergence of vector field $\nabla \cdot (u_h, v_h)^T$

```
>> [divu] = Div2D(u,v);
```

Prototyping and setup made simple!

Assembling the grid

Assembling the grid



We want to (semi-)automatically

1 Generate mesh for domain topology

2 Build local mesh data based on user input, e.g. x, y

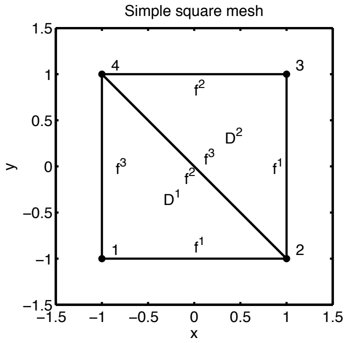
3 Geometrical data, i.e. normals and metrics

4 Build index maps for easy user setup of boundary conditions

Note: Some user input may be required for stage 1 and stage 4.

Bookkeeping is the main problem....

Assembling the grid

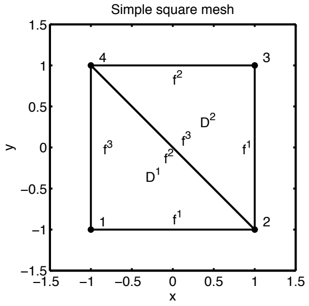


Now, we have (from some favorite mesh generator)

- Basic global mesh data tables, i.e. VX, VY and EToV

```
VX   = [-1 1 1 -1];
VY   = [-1 -1 1 1];
EToV = [1 2 4;
        2 3 4];
```

More mesh data tables



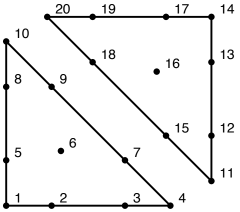
It is useful to define some additional mesh tables (**Connect2D.m**)

```
EToE = [1 2 1;
        2 2 1]; % Element-To-Element Connectivity Table
EToF = [1 3 3;
        1 2 2]; % Element-To-Face Connectivity Table
```

These tables are useful for the pre-processing of the standard solver, e.g. index maps and local operators.

General rule: create mesh data tables as needed if it makes the setup easier.

Local data



On the reference triangle element I the nodes $\mathbf{r} = (r, s)$ are found using the explicit Warp & Blend procedure

```
>> [x,y] = Nodes2D(N); [r,s] = xytors(x,y);
```

For all the local elements D^k, $k = 1, .., K$, the local physical coordinates $\mathbf{x}^k = (x^k, y^k)$ can be determined alltogether as

```
>> v1 = EToV(:,1); v2 = EToV(:,2); v3 = EToV(:,3);
>> L1 = 0.5*(s+1); L2 = -0.5*(r+s); L3 = 0.5*(r+1);
>> x = L2*VX(v1) + L3*VX(v2) + L1*VX(v3);
>> y = L2*VY(v1) + L3*VY(v2) + L1*VY(v3);
```

This defines the set and ordering of the global nodal numbers.

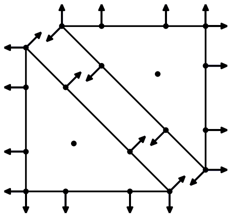
Metrics of the mapping

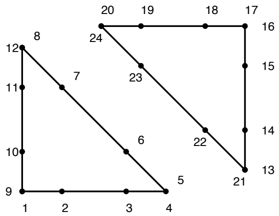
The metric of the mapping can be calculated by utilizing the local derivative operators $\mathcal{D}_r$ and $\mathcal{D}_s$ on the local coordinates (x^k, y^k)

```
function [rx,sx,ry,sy,J] = GeometricFactors2D(x,y,Dr,Ds)
% function [rx,sx,ry,sy,J] = GeometricFactors2D(x,y,Dr,Ds)
% Purpose  : Compute the metric elements for the
%            local mappings of the elements

% Calculate geometric factors
xr = Dr*x; xs = Ds*x; yr = Dr*y; ys = Ds*y; J = -xs.*yr + xr.*ys;
rx = ys./J; sx =-yr./J; ry =-xs./J; sy = xr./J;
return;
```

Note: Data for all elements are processed at once using matrix-matrix products.

Geometric factors





For the DG-FEM setup we need locally at every node on each face

- the outward pointing normal vector $\hat{\mathbf{n}}^i$, $i = 1, 2, 3$
- the surface Jacobians, J_s^i, and the Jacobian of mapping, J

This defines the set and ordering of the global face nodal numbers.

$$\hat{n}^1 = -\frac{\nabla s}{||\nabla s||} = -\frac{J}{J_s^1}\left(\begin{array}{c} s_x \\ s_y \end{array}\right) = \frac{1}{J_s^1}\left(\begin{array}{c} y_r \\ -x_r \end{array}\right)$$

$$||\hat{n}^1|| = \frac{1}{J_s^1}\sqrt{y_r^2 + x_r^2} = 1, \quad J_s^1 = \sqrt{x_r^2 + y_r^2}$$

Surface jacobians and normal vector components computed in **Normals2D**.

Index maps for imposing BCs I

For imposing boundary conditions on the local elements, we create special index maps for imposing different types of boundary conditions, cf. lecture on Mesh generation & Appendix B. Two types of maps will be useful.

Index maps collecting face nodes from volume nodes (sets of global nodal numbers)

vmapM	Vector of global nodal numbers at faces for interior values $u^- = u(\text{vmapM})$
vmapP	Vector of global nodal numbers at faces for exterior values $u^+ = u(\text{vmapP})$
vmapB	Vector of global nodal numbers at faces for boundary values $u^-(\partial\Omega_h) = u(\text{vmapB})$

Index maps for modifying collected face nodes (sets of global face nodal numbers), e.g.

mapM	Vector of indices to global face nodal numbers of interior face values
mapP	Vector of indices to global face nodal numbers of exterior face values
mapB	Vector of indices to global face nodal numbers of boundary face values

Index maps for imposing BCs II

Index maps of type **vmap** can be used for creating/modification of the face arrays of size **Nfaces** · **K** · **Nfp**.

Index maps of type **map** can be used for appropriate modification of the face arrays.

Some standard index maps are setup in **BuildMaps2D.m**.

Special index maps can be created using a **BCType** table storing information about the type of boundary for each face of every element in the mesh.

Note: **vmaps** are volume index maps and **maps** are face maps.

Putting the pieces together...

Putting the pieces together in a code

Consider the 2D linear advection equation

$$\partial_t u + c_x \partial_x u + c_y \partial_y u = 0, \quad \mathbf{x} \in \Omega$$

with IC conditions

$$u(\mathbf{x}, 0) = \sin\left(\frac{2\pi}{L_x} x\right) \sin\left(\frac{2\pi}{L_y} y\right)$$

The exact solution to this problem is given as

$$u(\mathbf{x}, t) = \sin\left(\frac{2\pi}{L_x}(x - c_x t)\right) \sin\left(\frac{2\pi}{L_y}(y - c_y t)\right), \quad \mathbf{x} \in \partial\Omega$$

which is used for specifying boundary conditions $\mathbf{x} \in \partial\Omega_h$ where $\hat{\mathbf{n}} \cdot \mathbf{c} < 0$ (incoming characteristics).

Putting the pieces together in a code

The DG-FEM method for the 2D linear advection equation on the k'th element is

$$\int_{D^k} l_i^k(\mathbf{x}) \partial_t u_h^k d\mathbf{x} + \int_{D^k} l_i^k(\mathbf{x}) \boldsymbol{\nabla} \cdot \mathbf{f}_h^k d\mathbf{x} = 0, \quad \mathbf{f}_h^k = \mathbf{c} u_h^k$$

where the local solution is approximated as

$$u_h^k(\mathbf{x}, t) = \sum_{i=1}^{N_p} u_h^k(\mathbf{x}_i^k, t) l_i^k(\mathbf{x})$$

By integration by parts twice and exchanging the numerical flux

$$\int_{D^k} l_i^k(\mathbf{x}) \partial_t u_h^k d\mathbf{x} + \int_{D^k} l_i^k(\mathbf{x}) \boldsymbol{\nabla} \cdot \mathbf{f}^k d\mathbf{x} = \oint_{\partial D^k} l_i^k \hat{\mathbf{n}} \cdot (\mathbf{f}_h^k - \mathbf{f}_h^*) d\mathbf{x}$$

where $i = 1, ..., N_p$.

Putting the pieces together in a code

By inserting the local approximation for the solution we can now obtain the local semidiscrete scheme

$$\mathcal{M}^k \frac{du_h^k}{dt} + (c_x \mathcal{S}_x + c_y \mathcal{S}_y) u_h^k = \oint_{\partial D^k} l_i^k \hat{\mathbf{n}} \cdot (\mathbf{f}_h^k - \mathbf{f}_h^*) d\mathbf{x}$$

Then, we need to pick a suitable numerical flux $\mathbf{f}_h^*$ for the problem, e.g. upwinding according to the characteristics

$$\mathbf{f}_h^{k,*}(u_h^{k,-}, u_h^{k,+}) = \begin{cases} \mathbf{c} u_h^{k,-}, & \mathbf{c} \cdot \hat{\mathbf{n}} \geq 0 \\ \mathbf{c} u_h^{k,+}, & \mathbf{c} \cdot \hat{\mathbf{n}} < 0 \end{cases}$$

which leads to a stable scheme.

To solve the system in time, apply a suitable ODE solver, e.g. a Runge-Kutta method.

Putting the pieces together in a code

To solve a semidiscrete problem of the form

$$\frac{du_h}{dt} = \mathcal{L}_h(u_h, t)$$

employ some appropriate ODE solver to deal with time, e.g. the low-storage explicit fourth order Runge-Kutta method (LSERK4)[2]

$$\mathbf{p}^{(0)} = \mathbf{u}^n$$

$$i \in [1, ..., 5] : \begin{cases} \mathbf{k}^{(i)} = a_i \mathbf{k}^{(i-1)} + \Delta t \mathcal{L}_h(\mathbf{p}^{(i-1)}, t^n + c_i \Delta t) \\ \mathbf{p}^{(i)} = \mathbf{p}^{(i-1)} + b_i \mathbf{k}^{(i)} \end{cases}$$

$$\mathbf{u}_h^{n+1} = \mathbf{p}^{(5)}$$

- For every element, the time step size Δt has to obey a CFL condition of the form

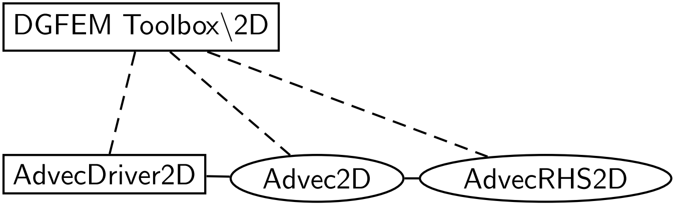
$$\Delta t \leq \frac{C}{a} \min_{k,i} \Delta x_i^k$$

[2] See book p. 64.

Putting the pieces together in a code

To build your own solver using the DGFEM codes

AdvecDriver2D	Matlab main function for solving the 2D advection equation.
Advec2D	Matlab function for time integration of the semidiscrete PDE.
AdvecRHS2D	Matlab function defining right hand side of semidiscrete PDE



- Programming effort comparable to 1D case

Element-wise operations

A summary of preprocessing scripts for 2D DG-FEM computations in Matlab

Globals2D	Define list of globals variables
Startup2D	Main script for pre-processing
MeshGenDistMesh2D	Generates a simple square grid using DistMesh
BuildMaps2D	Automatically create index maps from conn. and bc tables
BuildBCMaps2D	Construct special index maps for imposing BC's
Normals2D	Compute outward pointing normals at elements faces
Connect2D	Build global connectivity arrays for 2D grid
GeometricFactors2D	Compute metrics of local mappings
Lift2D	Compute surface integral term in DG formulation
dtscale2D	Compute characteristic inscribed circle diameter for grid
(Hrefine2D)	Apply non-conforming refinement to specific elements

Element-wise operations

How to use create a simple square initial mesh in Matlab

```
>> NN = 3;    % numbers of vertices on an edge, adjustable
>> X = linspace(-1,1,NN);
>> [VX,VY]= meshgrid(X,X);
>> VX = VX(:)';
>> VY = VY(:)';
>> EToV = delaunay(VX,VY);
>> triplot(EToV,VX,VY,'k')   % show mesh
>> K = size(EToV,1);
>> Nv = length(VX(:));
```

Note: more on mesh generation in the Lecture on mesh generation tomorrow.

Element-wise operations

How to use **hrefine.m** for *hp*-convergence tests

```
>> refineflag = 1:K;      % mark elements for refinement
>> Hrefine2D(refineflag); % modify EToV
```

Variables and mesh tables in global scope via **Globals2D.m**, thus only need to state which elements needs to be refined through the **refineflag** vector.

Note: **Hrefine2D.m** can be used for non-conforming refinement if the code are setup to exploit this.

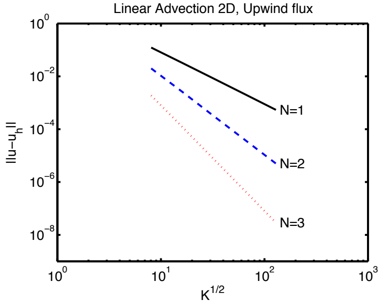
Putting the pieces together in a code I

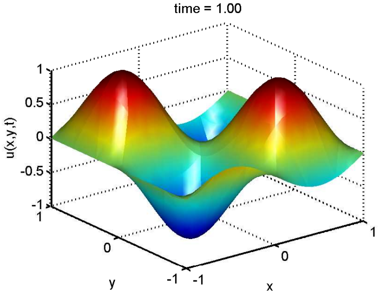
```
% Driver script for solving the 2D advection equation
Globals2D;

% Polynomial order used for approximation
N = 6;

% Create Mesh
NN = 6;
X = linspace(-1,1,NN);
[VX,VY]= meshgrid(X,X); VX = VX(:)'; VY = VY(:)';
EToV = delaunay(VX,VY);
K = size(EToV,1); Nv = length(VX(:));

% Initialize solver and construct grid and metric
StartUp2D;

% Set initial conditions
Lx = 2; Ly = 2; u = sin(2*pi/Lx*x).*sin(2*pi/Ly*y);
cx = 1; cy = 0.1; % advection speed vector

% Solve Problem
FinalTime = 1;
[u,time] = Advec2D(u, FinalTime, cx, cy);
```

Putting the pieces together in a code I

```
function [u,time] = Advec2D(u, FinalTime, cx, cy, alpha)
% function [u,time] = Advec2D(u, FinalTime, cx, cy)
% Purpose : Integrate 2D advection equation until FinalTime starting with
%           initial cocndition u
Globals2D;
time = 0;

% Runge-Kutta residual storage
resu = zeros(Np,K);

% compute time step size
rLGL = JacobiGQ(0,0,N); rmin = abs(rLGL(1)-rLGL(2));
dtscale = dtscale2D; dt = min(dtscale)*rmin*2/3

% outer time step loop
tstep = 0;
while (time<FinalTime)
  tstep= tstep+1;
  if(time+dt>FinalTime), dt = FinalTime-time; end

   for INTRK = 1:5
       timelocal = time + rk4c(INTRK)*dt;
       [rhsu] = AdvecRHS2Dupwind(u, timelocal, cx, cy);
       resu   = rk4a(INTRK)*resu + dt*rhsu;
       u = u+rk4b(INTRK)*resu;
   end
   % Increment time
   time = time+dt;
end
return
```

Putting the pieces together in a code I





- Upwind flux gives as expected ideal convergence $\mathcal{O}(h^{N+1})$
- Advection speed vector, $\mathbf{c} = (1.0, 0.1)$
- Dirichlet boundary conditions specified where characteristics are incoming
- **Hrefine2D.m** used to generate fine meshes from initial coarse mesh.

References I

T. Warburton.
An explicit construction for interpolation nodes on the simplex.
J. Engineering Math., 56(3):247–262, 2006.