An introduction to Mesh generation

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 14, 2012

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

Numerical solution of PDEs

To construct a numerical method for solving PDEs we need to consider

- How to represent the solution $u(x,t)$ by an approximate solution $u_h(x,t)$?
- In which sense will the approximate solution $u_h(x,t)$ satisfy the PDE?

The two choices separate and define the properties of different numerical methods...

The choice of how to represent the solution is intimately connected with a need for meshes

Numerical solution of PDEs

From approximation theory (see Chapter 4) we have the general result

Theorem 4.8. Assume that $u \in H^p(D^k)$, $p > 1/2$, and that u_h represents a piecewise polynomial interpolation of order N. Then

$$||u - u_h||_{\Omega,q,h} \leq C \frac{h^{\sigma-q}}{N^{p-2q-1/2}} |u|_{\Omega,\sigma,h}$$

for $0 \leq q \leq \sigma$, and $\sigma = \min(N+1, p)$.

- Upper bound for interpolation error depends on element size $h = \max_k h^k$ and polynomial order N of local expansions (implicitly element sizes, shapes and local node distributions)
- Regularity of the solution

Numerical solution of PDEs

For the general case of a hyperbolic system (see Chapter 4)

$$\frac{\partial \mathbf{u}}{\partial t} + \mathcal{A}\frac{\partial \mathbf{u}}{\partial x} = 0$$

we have obtained Courant-Friedrichs-Levy (CFL) conditions of the form for explicit schemes

$$\Delta t \leq C \frac{1}{\max(|\lambda(\mathcal{A})|)} \min_{k,i} \frac{h^k}{2}(\Delta_i r)$$

which shows

- Discrete stability is governed by the mesh size h^k and grid spacing in standard element $\Delta_i r$ (node distribution), and
- The discrete operator $\mathcal{A}$ is dependent on the scaling of extreme eigenvalues with h^k and so is the conditioning.

This highlights that optimal meshes are problem dependent

Numerical solution of PDEs

Recall, results obtained by solving the simple advection equation on a periodic domain

$$\partial_t u - 2\pi \partial_x u = 0, \quad x \in [0, 2\pi], \quad u(x,0) = \sin(lx), \quad l = \frac{2\pi}{\lambda}$$

Errors at final time $T = \pi$.

N\ K	2	4	8	16	32	64	Convergence rate
1	-	4.0E-01	9.1E-02	2.3E-02	5.7E-03	1.4E-03	2.0
2	2.0E-01	4.3E-02	6.3E-03	8.0E-04	1.0E-04	1.3E-05	3.0
4	3.3E-03	3.1E-04	9.9E-06	3.2E-07	1.0E-08	3.3E-10	5.0
8	2.1E-07	2.5E-09	4.8E-12	2.2E-13	5.0E-13	6.6E-13	$\cong$ 9.0

Final time (T)	π	10π	100π	1000π	2000π
(N,K)=(2,4)	4.3E-02	7.8E-02	5.6E-01	>1	>1
(N,K)=(4,2)	3.3E-03	4.4E-03	2.8E-02	2.6E-01	4.8E-01
(N,K)=(4,4)	3.1E-04	3.3E-04	3.4E-04	7.7E-04	1.4E-03

Error is seen to behave as

$$||u - u_h||_{\Omega,h} \leq C(T)h^{N+1} \cong (c_1 + c_2 T)h^{N+1}$$

Optimal meshes are dependent on accuracy requirements

Examples: error behavior

Cost measured in terms of CPU time showed,
Time$\cong C(T)K(N+1)^2$.

N\K	2	4	8	16	32	64
1	1.00	2.19	3.50	8.13	**19.6**	54.3
2	2.00	3.75	**7.31**	15.3	38.4	110.
4	**4.88**	8.94	20.0	45.0	115.	327.
8	15.1	32.0	68.3	163.	665.	1271.
16	57.8	121.	279.	664.	1958.	5256.

N\K	2	4	8	16	32	64	Convergence rate
1	-	4.0E-01	9.1E-02	2.3E-02	**5.7E-03**	1.4E-03	2.0
2	2.0E-01	4.3E-02	**6.3E-03**	8.0E-04	1.0E-04	1.3E-05	3.0
4	**3.3E-03**	3.1E-04	9.9E-06	3.2E-07	1.0E-08	3.3E-10	5.0
8	2.1E-07	2.5E-09	4.8E-12	2.2E-13	5.0E-13	6.6E-13	$\cong$ 9.0

Optimal meshes achieves balance between accuracy and cost

When do we need a mesh?

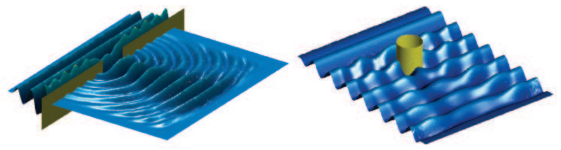
A mesh is needed when

- We want to solve a given problem on a computer using a method which requires a discrete representation of the domain

Two main problems to consider

- Numerical method:
  Which numerical method to employ for defining a suitable solution procedure for the mathematical problem.
- Mesh generation:
  How to represent the domain of interest for use in our solution procedure.

When do we need a mesh?



Thus, it is convenient if

- We can independently consider the *problem solution procedure* and *mesh generation* as two distinct problems.
- Proto-type PDE solvers using a component-based setup which provides the basis for this abstraction.

Domain of interest

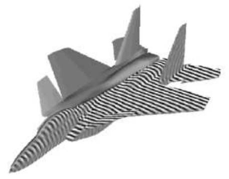


Figure: F-15. From `www.useme.org`

In DG-FEM we have chosen to represent the problem domain Ω by a partitioning of the domain into a union of K nonoverlapping local elements D^k, $k = 1, ..., K$ such that

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

Thus, the local representation of our solution u_h^k is intimately connected with the representation in terms of modal/nodal basis functions on the elements of the mesh.

Terminology

Mesh terminology:

- Structured mesh
  Nearly all nodes have the same number of neighbors (interior vs. boundary nodes).
- Unstructured mesh
  Non-obvious number of neighbors for each node in mesh.
- Conformal mesh
  Nodes, sides and faces of neighboring elements are perfectly matched.
- Hanging nodes
  Nodes, which are not perfectly matched with a neighboring element node.

Unstructured Mesh Generation

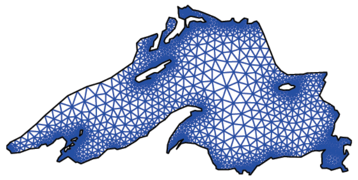
Motivation for the use of unstructured meshes

- To represent complex geometric features of solution domain.
- To straightforwardly adapt the mesh to features of the solution via refinement/de-refinement.

For the generation of unstructured meshes it is standard to follow

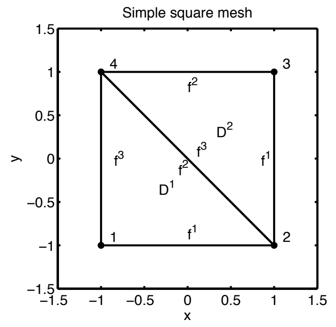
- In 2D: Triangulation (fx. DistMesh in Matlab), Quadrilaterals (few free options and no immediate options in Matlab?).
- Simple automatic Tet-to-Quad conversion an option.
- Fix mesh.
- Post processing to assess and possibly improve initial mesh quality.

What defines a mesh? I



- A mesh is completely defined in terms of a set of (unique) vertices and defined connections among these.
  - coordinate tables, VX and VY (unique vertices)
  - mesh element table, EToV (triangulation/quadrilaterals/etc.)
- In addition it is customary to define types of boundaries for specifying boundary conditions where needed.
  - a boundary type table, BCType (element face types)

Mesh data

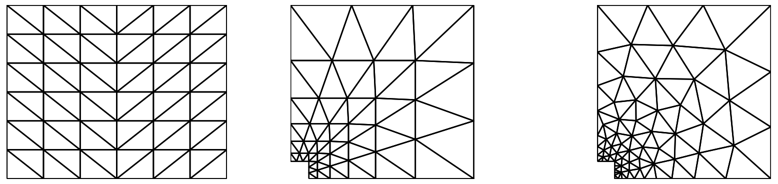


From our favorite mesh generator we obtain

- Basic 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];
```

What defines a mesh? I



- Very simple meshes can be created manually by hand.
- Automatic mesh generation is generally faster and more efficient
  - Some user input for accurately describing the geometry and desired (initial) mesh resolution may be required.
- Note: Mesh data can be stored for reuse several times - not necessary to generate every time!

Mesh generators available

- Lots of standard open source or commercial mesh generation tools available!
  - Test and pick you own favorite!
  - Disadvantage: may require a translation script to be created for use with your own solver (e.g. in Matlab).
- Important properties of mesh generators
  - Grid quality (e.g. aspect ratio and element angles)
  - Efficiency
  - Features for handling BCs, adaptivity, etc.
- An example of a free software distribution package for generating unstructured triangular meshes is DistMesh for Matlab.

Unstructured Mesh Generation in Matlab with DistMesh

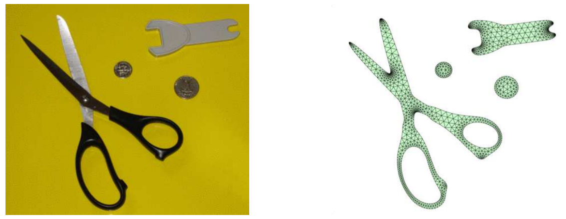


Figure: Figures are from
`http://persson.berkeley.edu/distmesh/gallery_images.html`.

- Persson, P.-O. and Strang, G. 2004 A simple mesh generator in Matlab. SIAM Review. Download scripts at: `http://persson.berkeley.edu/distmesh/`
- A simple algorithm that combines a physical principle of force equilibrium in a truss structure with a mathematical representation of the geometry using signed distance functions.
- Can generate meshes in 1D, 2D and 3D with few lines of code.

Introduction to DistMesh for Matlab

- Algorithm (Conceptual):
  1. Define a domain using signed distance functions.
  2. Distribute a set of nodes interior to the domain.
  3. Move interior nodes to obtain force equilibrium.
  4. Apply terminate criterion when all nodes are (nearly) fixed in space.
- Post-processing steps (Preparation):
  (Note: not done by DistMesh)
  5. Validate final output!
  6. Reorder element vertices to be defined counter-clockwise (standard convention).
  7. Setup boundary table.
  8. Store mesh for reuse.

Introduction to DistMesh for Matlab

Definition: A signed distance function, $d(x)$

$$d(x) = \begin{cases} < 0 & , x \in \Omega \quad \text{(interior)} \\ 0 & , x \in \partial\Omega \quad \text{(boundary)} \\ > 0 & , x \notin \Omega \quad \text{(exterior)} \end{cases}$$

Define metric using an appropriate norm, e.g. the Euclidian metric.

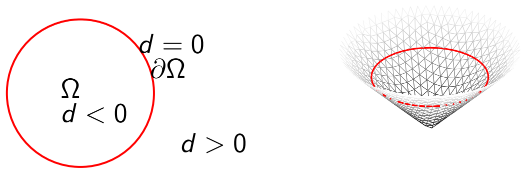
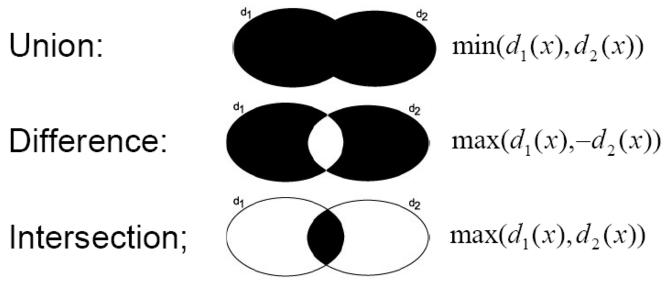


Figure: Example of a signed distance function for a circle.

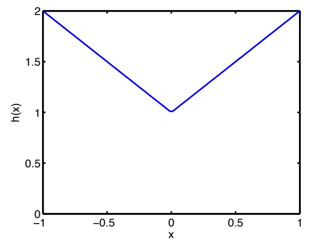
Introduction to DistMesh for Matlab

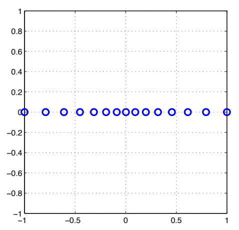
Combine and create geometries defined by distance functions using the Union, difference and intersection operations of sets



Introduction to DistMesh for Matlab

Example 1. Create a nonuniform mesh in 1D with local refinement near center.





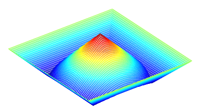
Using DistMesh (in Matlab) only 3 lines of code needed:

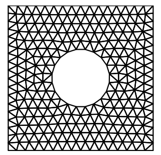
```
>> d=inline('sqrt(sum(p.^2,2))-1','p');
>> h=inline('sqrt(sum(p.^2,2))+1','p');
>> [p,t]=distmeshnd(d,h,0.1,[-1;1],[]);
```

Weight function *h* measures distance from origo and adds a unit to the measure.

Introduction to DistMesh for Matlab

Example 1. Create a uniform mesh for a square with hole.



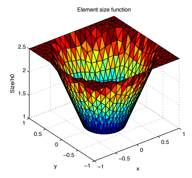


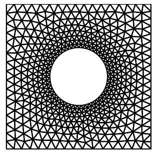
Using DistMesh (in Matlab) only 3 lines of code needed:

```
>> fd=inline('ddiff(drectangle(p,-1,1,-1,1),dcircle(p,0,0,0.4))','p');
>> pfix = [-1,-1;-1,1;1,-1;1,1];
>> [p,t] = distmesh2d(fd,@huniform,0.125,[-1,-1;1,1],pfix);
```

Introduction to DistMesh for Matlab

Example 2. A refined mesh for a square with hole.





Using DistMesh (in Matlab) only 4 lines of code needed:

```
>> fd = inline('ddiff(drectangle(p,-1,1,-1,1),dcircle(p,0,0,0.4))','p')
>> pfix = [-1,-1;-1,1;1,-1;1,1];
>> fh = inline(['min( sqrt( p(:,1).^2 + p(:,2).^2 ) , 1 )'],'p');
>> [p,t] = distmesh2d(fd,fh,0.125/2.5,[-1,-1;1,1],pfix);
```

- Size function `fh` defines *relative* sizes of elements (`fh` constant result in a uniform mesh distribution)
- The *initial* characteristic size of the elements is `h0`.
- In final triangulation, the characteristic size of the smallest elements will be approx. `h0`.

Introduction to DistMesh for Matlab

DistMesh output in two tables;

p	Unique vertice coordinates
t	Element to Vertice table (random element orientations by DistMesh)

From these tables we can determine, e.g.

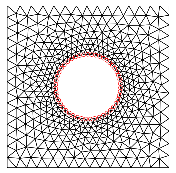
```
>> K=size(t,1);       % Number of elements
>> Nv=size(p,1);      % Number of vertices in mesh
>> Nfaces=size(t,2);  % Number of faces/element
>> VX = p(:,1);       % Vertice x-coordinates
>> VY = p(:,2);       % Vertice y-coordinates
>> EToV = t;          % Element to Vertice table
```

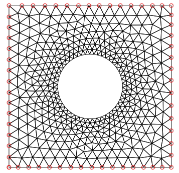
To ensure same element node orientations use DistMesh function

```
>> [p,t]=fixmesh(p,t); % remove duplicate nodes and orientate
```

Introduction to DistMesh for Matlab

Example 3. Selecting boundary nodes for a square with hole.


(a) Inner boundary nodes

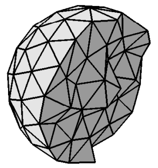

(b) Outer boundary nodes

Nodes can be selected using distance functions; $|d| = 0$ or $|d| <$tol.

```
>> fdInner    = inline(dcircle(p,0,0,0.4),p);
>> nodesInner = find(abs(fdInner([p]))<1e-3);
>> fdOuter    = inline(drectangle(p,-1,1,-1,1),p);
>> nodesOuter = find(abs(fdOuter([p]))<1e-3);
>> nodesB     = find(abs(fd([p]))<1e-3);
```

Introduction to DistMesh for Matlab

Example 4. Uniform mesh for a unit ball (3D).



```
>> fh = @huniform;
>> fd=inline('sqrt(sum(p.^2,2))-1','p'); % ball
>> Bbox = [-1 -1 -1; 1 1 1]; % cube
>> Fix  = [-1 -1 -1; 1 -1 -1; 1 1 -1; -1 1 -1;...
           -1 -1  1; 1 -1  1; 1 1  1; -1 1  1];
>> [Vert,EToV]=distmeshnd(fd,fh,h0,Bbox, Fix);
```

Visualization

MATLAB commands for visualization:

```
% 2-D Triangular plot (also works for quadrilaterals!)
>> triplot(t,p(:,1),p(:,2),'k')

% 3-D Visualization of solution
>> trimesh(t,p(:,1),p(:,2),u)

% 3-D Visualization of solution
>> trisurf(t,p(:,1),p(:,2),u)

% 3-D Visualization of part of solution
>> trisurf(t(idxlist,:),p(:,1),p(:,2),u)

% Visualization of node connections in matrix
>> gplot(A,p)
```

Computing geometric information

We seek to determine outward pointing normal vectors for an element edge of a straight-sided polygon.

Assume that the order of element vertices is counter-clockwise, then for an boundary edge defined from (x_1, y_1) to (x_2, y_2) we find

$$\Delta x = x_2 - x_1, \quad \Delta y = y_2 - y_1$$

and thus a tangential vector becomes

$$\mathbf{t} = (t_1, t_2)^T = (\Delta x, \Delta y)^T$$

which should be orthogonal to the normal vector. Hence an outward pointing normalized vector is given as

$$\mathbf{n} = (n_1, n_2)^T = (t_2, -t_1)^T / \sqrt{t_1^2 + t_2^2}$$

Normal vectors useful for imposing boundary conditions.

Local vertice ordering

To make sure that the vertices are ordered in an counter-clockwise fashion, the following metric can be used

$$D = \begin{pmatrix} x^1 - x^3 \\ y^1 - y^3 \end{pmatrix} \cdot \begin{pmatrix} y^2 - y^3 \\ -(x^2 - x^3) \end{pmatrix} = \hat{\mathbf{t}}_{31} \cdot \hat{\mathbf{n}}_{32}$$

If $D < 0$ then ordering is clockwise and if $D > 0$ counter-clockwise.

```
function [EToV] = Reorder(EToV,VX,VY)
% Purpose: Reorder elements to ensure counter clockwise orientation
x1 = VX(EToV(:,1)); y1 = VY(EToV(:,1));
x2 = VX(EToV(:,2)); y2 = VY(EToV(:,2));
x3 = VX(EToV(:,3)); y3 = VY(EToV(:,3));
D = (x1-x3).*(y2-y3)-(x2-x3).*(y1-y3);
i = find(D<0);
EToV(i,:) = EToV(i,[1 3 2]); % reorder
```

Creating special index maps I

For imposing boundary conditions or extracting information from the solution it can be useful to create special index maps.

Having already created a mesh, create a new boundary table of size $K \times N_{\text{faces}}$ for all element faces

```
>> BCType = int8(not(EToE)); % initialization
```

This table can then be used to store information about different types of boundaries, e.g. Inflow/Outflow, West/East, etc.

Creating special index maps I

To create different index maps for imposing special types of boundary conditions, e.g. Dirichlet and Neumann BC's on all or selected boundaries

```
% for selecting all outer boundaries
>> x1 = -1; x2 = 1; y1 = -1; y2 = 1;
>> fd = @(p) drectangle(p,x1,x2,y1,y2);
>> BCType = CorrectBCTable_v2(EToV,VX,VY,BCType,fd,AllBoundaries);

% for selecting south and west boundaries
>> fd = @(p) drectangle(p,-1,2,-1,2);
>> BCType = CorrectBCTable_v2(EToV,VX,VY,BCType,fd,Dirichlet);

% select north and east boundaries
>> fd = @(p) drectangle(p,-2,1,-2,1);
>> BCType = CorrectBCTable_v2(EToV,VX,VY,BCType,fd,Neumann);
```

Note: new version v2 of CorrectBCTable in ServiceRoutintes/

Creating special index maps I

Then, using the **BCType** table we can create our special index maps

```
% face maps
>> mapB   = ConstructMap(BCType,AllBoundaries);
>> mapD   = ConstructMap(BCType,Dirichlet);
>> mapN   = ConstructMap(BCType,Neumann);

% volume maps
>> vmapB  = vmapM(mapB);
>> vmapN  = vmapM(mapN);
>> vmapD  = vmapM(mapD);
```

Remember to validate the created index maps

For example

```
% visual check of boundary volume map
>> triplot(EToV,VX,VY,'k');
>> hold on;
>> plot(x(vmapB),y(vmapB),'ro');
% or via precomputed face coordinate arrays
>> plot(Fx(mapB),Fx(mapB),'b*');
```

Creating special index maps

In problems with periodic boundaries the standard indexmaps can be modified systematically in the following way

1. Create and modify a BCType table to hold information about boundary types
   - `ServiceRoutines/CorrectBCTable_v2`
2. For simple opposing boundaries that are conforming create a distance function for generating a sorted list of face center distances
3. From the sorted list, create indexmaps for each boundary
   - **ConstructPeriodicMap**
4. Modify volume-to-face index map vmapP to account for periodicity.
5. Validate implementation!

Let's consider a square mesh $\Omega_h([-1,1]^2)$...

Creating special index maps

Step 1: Create and modify a BCType table

```
BCType = int16(not(EToE));

fdW = @(p) drectangle(p,-1,2,-2,2);
fdE = @(p) drectangle(p,-2,1,-2,2);

BCcodeW = 1; BCcodeE = 2;

BCType = CorrectBCTable_v2(EToV,VX,VY,BCType,fdW,BCcodeW);
BCType = CorrectBCTable_v2(EToV,VX,VY,BCType,fdE,BCcodeE);

fdS = @(p) drectangle(p,-2,2,-1,2);
fdN = @(p) drectangle(p,-2,2,-2,1);

BCcodeS = 3; BCcodeN = 4;

BCType = CorrectBCTable_v2(EToV,VX,VY,BCType,fdS,BCcodeS);
BCType = CorrectBCTable_v2(EToV,VX,VY,BCType,fdN,BCcodeN);
```

Creating special index maps

Step 2: Create a distance function useful for sorting opposing face centers

```
pv = [-1 1; 1 -1;];
% dsegment is a signed distance function for a line
% provided in DistMesh package
fd = @(p) dsegment(p,pv); % line segment from (-1,1) to (1,-1)
```

Step 3: Create indexmaps for each periodic boundary

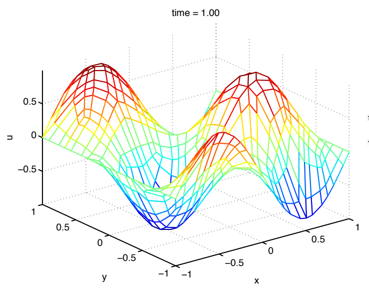
```
[mapW,mapE] = ConstructPeriodicMap(EToV,VX,VY,BCType,BCcodeW,BCcodeE,fd);
[mapS,mapN] = ConstructPeriodicMap(EToV,VX,VY,BCType,BCcodeS,BCcodeN,fd);
```

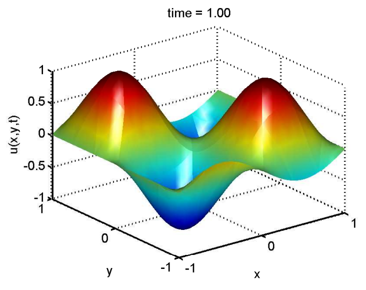
Step 4: Modify exterior vmapP to be periodic with vmapM

```
vmapP(mapW) = vmapM(mapW);
vmapP(mapE) = vmapM(mapE);
vmapP(mapS) = vmapM(mapS);
vmapP(mapN) = vmapM(mapN);
```

Creating special index maps

Step 5: Validation!





- Periodic initial condition $u_h(x,y,0)$
- Constant advection speed vector arbitrary $\mathbf{c} = (c_x, c_y)^T$
- Upwind flux gives as expected ideal convergence $\mathcal{O}(h^{N+1})$

Creating special index maps

```
function [rhsu] = AdvecRHS2DupwindPeriodic(u, timelocal, cx, cy, alpha)

% function [rhsu] = AdvecRHS2D(u, timelocal, a, alpha)
% Purpose  : Evaluate RHS flux in 2D advection equation
%            using upwinding

Globals2D;

% Define flux differences at faces
df = zeros(Nfp*Nfaces,K);

% phase speed in normal directions
cn = cx*nx(:) + cy*ny(:);

% upwinding according to characteristics
ustar = 0.5*(cn+abs(cn)).*u(vmapM) + 0.5*(cn-abs(cn)).*u(vmapP);
df(:) = cn.*u(vmapM) - ustar;

% local derivatives of fields
[ux,uy] = Grad2D(u);

% compute right hand sides of the PDE's
rhsu  = -(cx.*ux + cy.*uy) + LIFT*(Fscale.*df);
return
```

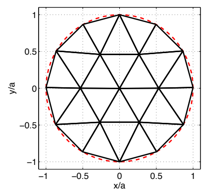
What defines a "good" mesh?

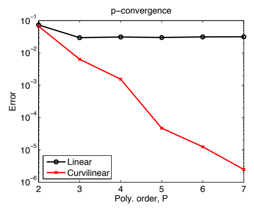
To define a "good" mesh (cf. Shewshuk ref.) we are usually concerned about

- accuracy in representation of the (usually) unknown solution(!)
- minimal cost in solution process for a given numerical accuracy requirement, recall for DG-FEM

$$\mathrm{CPU} \propto C(T)K(N+1)^2, \quad ||u-u_h||_{2,\Omega_h} \propto \mathcal{O}(h^P)$$

- approximating the right geometry of the problem(!)





Three general rules for "good" meshes

There are three general rules of thumb dictated by error analysis;

- very large and small element angles should be avoided
  - this suggest that equilateral triangles are optimal
- elements should be placed most densely in regions where the solution of the problem and its derivatives are expected to vary rapidly,
- high accuracy requires a fine mesh or many nodes per element (the latter conditions yields high accuracy, however, only if the solution is sufficiently smooth).

As a user, it is always a good idea to visualize the mesh and to check if these criteria are met.

- To improve mesh quality, it can be beneficial to apply some mesh smoothing procedure
  Fx. use **smoothmesh.m** from Mesh2D v23, Matlab Central Exchange.

A simple measure of mesh quality

The mesh quality measure described by Persson & Strang (2004) is adopted in the following.

A common mesh quality measure is the following ratio

$$q = 2\frac{r_{in}}{r_{out}}$$

where r_{in} is the radius of the largest inscribed circle and r_{out} is the smallest circumscribed circle.

- Equilateral triangles has $q = 1$
- Degenerate triangles has $q = 0$ (bad elements)
- "Good triangles" we define as having $q > 0.5$ (rule of thumb)

On Mesh quality metrics

As proposed by P. M. Knupp (2007) [1]

> *Mesh quality concerns the characteristics of a mesh that permit a particular numerical PDE simulation to be efficiently performed, with fidelity to the underlying physics, and with the accuracy required for the problem.*

In short: Mesh quality cannot be defined solely in terms of element shapes. Problem characteristics and requirements needs to be of concern in the assessment.

[1] P. M. Knupp (2007) Remarks on Mesh Quality. 45th AIAA Aerospace Sciences Meeting and Exhibit, 7-10 January, 2007. Reno, NV.

Laplacian smoothing

To improve the mesh quality, we can apply a simple Laplacian smoothing procedure

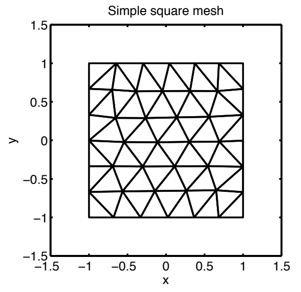
$$\mathbf{x}_i^{[k+1]} = \frac{1}{N_{i,connect}} \sum_{j=1}^{N_{i,connect}} \alpha_i \mathbf{x}_j^{[k]}, \quad \forall i : \mathbf{x}_i \notin \partial\Omega_h$$

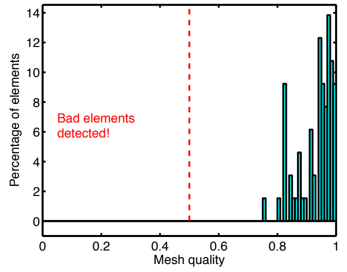
with α_j weight factors and $N_{i,connect}$ the number of nodes connected to the i'th node dictated by the mesh structure.

There are a few pitfalls

- Mesh tangling can occur near reentrant corners and needs special treatment.
- Local mesh adaption (anisotropic mesh density) can be reduced in the process.

Mesh quality

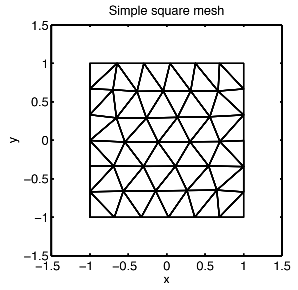


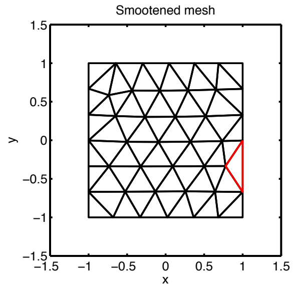


```
fd    = inline('drectangle(p,-1,1,-1,1)','p','param');
fh    = @huniform;
h0    = 0.35;
Bbox  = [-1 -1; 1 1];
pfix  = [-1 -1; 1 -1; 1 1; -1 1];
param = [];

% Call distmesh
[Vert,EToV]=distmesh2d(fd,fh,h0,Bbox,pfix,param);
[q] = MeshQuality(EToV,VX,VY);
```

Mesh quality



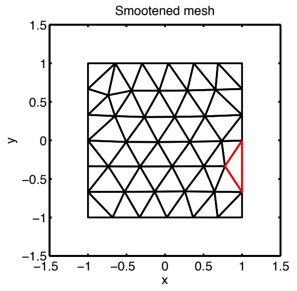


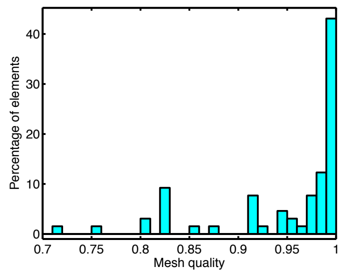
Remark: Degenerated triangle fixed by Laplacian smoothing.

```
% Call distmesh
[Vert,EToV]=distmesh2d(fd,fh,h0,Bbox,pfix,param);

% Call Mesh2d v23 function
maxit = 100; tol = 1e-10;
[p,EToV] = smoothmesh([VX' VY'],EToV,maxit,tol);
VX = p(:,1)'; VY = p(:,2)';
```

Mesh quality





Mapping a straight-sided quadrilateral

In practice, a map between a straight-sided quadrilateral D and a reference domain I

$$I = \{\mathbf{r} = (r, s) | (r, s) \in [-1, 1]^2\}$$

is useful. D is defined in terms of four vertices $\{\mathbf{v}^j\}_{j=1}^4$ defined by coordinates pairs $\{\mathbf{x}j\}_{j=1}^4$ in physical space and typically ordered anti-clockw

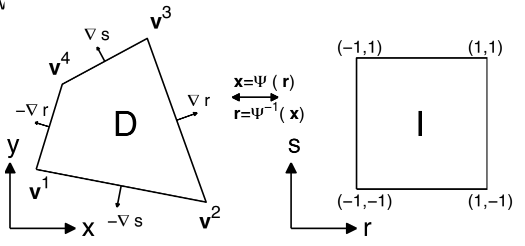


Figure: Notation for the mapping between two straight-sided quadrilaterals.

Mapping a straight-sided quadrilateral

An explicit linear mapping Ψ which takes a general quadrilateral, $\mathbf{x} \in D$, to the straight-sided reference quadrilateral I can be defined as

$$\begin{aligned}\boldsymbol{\Psi}(r, s) =& \frac{1}{4}\left[\mathbf{x}^1(1-r)(1-s) + \mathbf{x}^2(r+1)(1-s)\right.\\ &\left.+\mathbf{x}^3(r+1)(s+1) + \mathbf{x}^4(1-r)(s+1)\right]\end{aligned}$$

Thus given the four corner points (vertices) of a straight-sided quadrilateral in physical space we can map any point in the computational domain to the corresponding physical coordinate. The mapping can be expressed compactly as

$$\mathbf{x} = \lambda^1\mathbf{v}^1 + \lambda^2\mathbf{v}^2 + \lambda^3\mathbf{v}^3 + \lambda^4\mathbf{v}^4 = \Psi(\mathbf{r}), \quad \mathbf{v}^i = \begin{pmatrix} xi \\ y^i \end{pmatrix}$$

where the generalized barycentric coordinates λ^i are defined as

$$\lambda^1 = \frac{(1-r)(1-s)}{4}, \quad \lambda^2 = \frac{(r+1)(1-s)}{4}, \quad \lambda^3 = \frac{(r+1)(s+1)}{4}, \quad \lambda^4 = \frac{(1-r)(s+1)}{4}$$

Transformation of Equations under Mappings

The introduction of coordinate mappings between physical and computational spaces implies transformation of differential equations as well.

The effect of a mapping can be determined using the chain rule. Given a map $\mathbf{x} = \boldsymbol{\Psi}(\mathbf{r})$ derivatives of a function $u(x, y)$ transform according to

$$\begin{aligned}\frac{\partial u}{\partial x} &= \frac{\partial u}{\partial r}\frac{\partial r}{\partial x} + \frac{\partial u}{\partial s}\frac{\partial s}{\partial x}\\ \frac{\partial u}{\partial y} &= \frac{\partial u}{\partial r}\frac{\partial r}{\partial y} + \frac{\partial u}{\partial s}\frac{\partial s}{\partial y}\end{aligned}$$

which can be expressed in matrix-vector form as

$$\begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix} = \begin{bmatrix} r_x & s_x \\ r_y & s_y \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial r} \\ \frac{\partial u}{\partial s} \end{bmatrix}$$

having introduced a compact notation for the coordinate derivatives.

Metric coefficients of mapping in 2D

The constant metrics of the linear affine mapping can be found by the bijective property

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

From which we can deduce the following metric 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 straight-sided quadrilaterals connected through $\Psi(\mathbf{r})$

$$\begin{aligned} \mathbf{x}_r &= \tfrac{s-1}{4}\mathbf{v}^1 + \tfrac{1-s}{4}\mathbf{v}^2 + \tfrac{s+1}{4}\mathbf{v}^3 - \tfrac{s+1}{4}\mathbf{v}^4, \\ \mathbf{x}_s &= \tfrac{r-1}{4}\mathbf{v}^1 - \tfrac{r+1}{4}\mathbf{v}^2 + \tfrac{r+1}{4}\mathbf{v}^3 - \tfrac{1-r}{4}\mathbf{v}^4 \end{aligned}$$

with the Jacobian J expressed in terms of these coefficients.
It is possible to show that $J > 0$ for quadrilaterals with vertices ordered counter-clockwise.

Metric coefficients of mapping in 2D

Boundary normal vectors are often needed for evaluating boundary contributions and conditions. These can be determined by making use of the directional transformation obtained via the 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.
The normal vectors which are either parallel or orthogonal to the coordinate lines in the reference domain are

$$\begin{array}{lll} \mathbf{n}_1 = \begin{pmatrix} 0 \\ -1 \end{pmatrix} & \Rightarrow & \hat{\mathbf{n}}_1 = -\frac{\nabla s}{||\nabla s||} \\ \mathbf{n}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} & \Rightarrow & \hat{\mathbf{n}}_2 = \frac{\nabla r}{||\nabla r||} \\ \mathbf{n}_3 = \begin{pmatrix} -1 \\ 0 \end{pmatrix} & \Rightarrow & \hat{\mathbf{n}}_3 = -\frac{\nabla r}{||\nabla r||} \\ \mathbf{n}_4 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} & \Rightarrow & \hat{\mathbf{n}}_4 = \frac{\nabla s}{||\nabla s||} \end{array}$$

Local approximations in 2D on quads

On quadrilaterals we can construct the reference basis as a tensor product between 1D basis functions

$$\psi_m(\mathbf{r}) = \tilde{P}_i^{(0,0)}(r)\tilde{P}_j^{(0,0)}(s)$$

where $\tilde{P}_n^{(\alpha,\beta)}$ is the n'th order normalized Jacobi polynomial.

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

For the interpolations on the quadrilaterals to be well-behaved we use the optimal lobatto-gauss node positions identified in 1D. These node distributions are also used in gaussian cubature rules.

Construction of discrete operators is similar to earlier approaches. See Startup2DQuad.m

Discrete operations in 2D

The approximation of metric terms can be computed for each of the collocation points through the discrete operations

$$\begin{array}{ll} \mathbf{x}_r = \mathcal{D}_r\mathbf{x}, & \mathbf{y}_s = \mathcal{D}_s\mathbf{y} \\ \mathbf{y}_r = \mathcal{D}_r\mathbf{y}, & \mathbf{y}_s = \mathcal{D}_s\mathbf{y} \\ \multicolumn{2}{c}{\mathbf{J} = \mathbf{x}_r \boxtimes \mathbf{y}_r - \mathbf{x}_s \boxtimes \mathbf{y}_r} \end{array}$$

The differentiation matrices for use in reference domain are

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

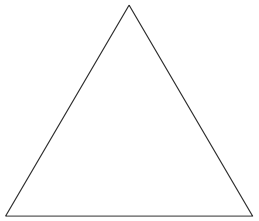
With these, we are in position to determine the discrete directional mappings derived from the chain rule

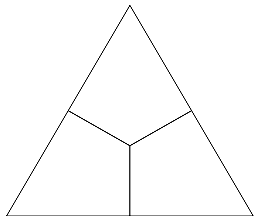
$$\begin{aligned} \mathcal{D}_x &= \mathrm{diag}(\mathbf{r}_x)\mathcal{D}_r + \mathrm{diag}(\mathbf{s}_x)\mathcal{D}_s \\ \mathcal{D}_y &= \mathrm{diag}(\mathbf{r}_y)\mathcal{D}_r + \mathrm{diag}(\mathbf{s}_y)\mathcal{D}_s \end{aligned}$$

Thus, when mappings are employed we can substitute the physical spatial derivative operations with these in numerical schemes.

Unstructured Mesh Conversion (conforming elements)

It is possible to efficiently convert triangles to quadrilaterals in a mesh using a single iteration of the Catmull-Clark subdivision algorithm. Work complexity is $\mathcal{O}(K)$ with K number of elements. Each triangle in the mesh is subdivided to consist of three quadrilaterals according to the following subdivision template

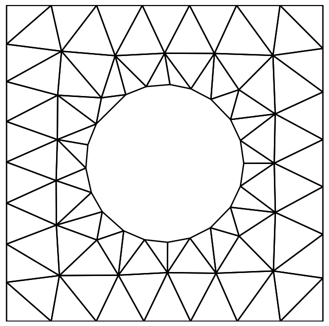

(a) Before subdivision

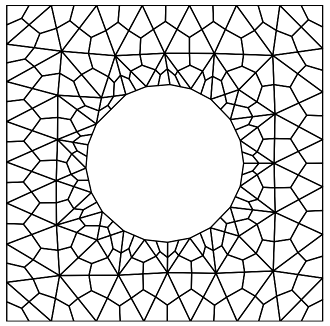

(b) After subdivision

For use with PDE solvers, it is standard to use anti-clockwise vertice orderings in `EToV`.

Unstructured Mesh Generation in Matlab

```
>> fd=inline('ddiff(drectangle(p,-1,1,-1,1),dcircle(p,0,0,0.5))','p');
>> fh=inline('min(4*sqrt(sum(p.^2,2))-1,2)','p'); % Mesh grading
>> [p,EToV]=distmesh2d(fd,fh,0.14,[-1,-1;1,1],[-1,-1;-1,1;1,-1;1,1]);
>> [p,EToV]=fixmesh(p,EToV); % Remove duplicate nodes, reorder vertices
>> VX = p(:,1); VY = p(:,2);
>> [Nv, VX, VY, K, EToV] = CatmullClarkSubdivision(VX,VY,EToV);
```


(c) Before subdivision (Triangles)

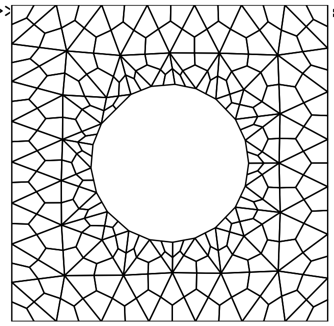

(d) After subdivision (Quads)

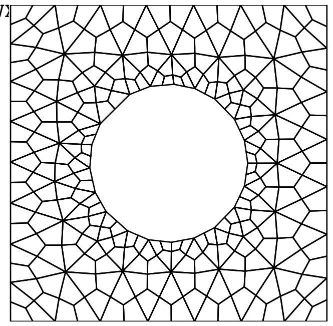
Enhancing mesh quality by Laplace smoothening

In a post-processing step node locations can be adjusted while maintaining element connectivitites.

$$\mathbf{x}_i = \frac{1}{N_{\text{connect}}} \sum_{i=1}^{N_{\text{connect}}} \mathbf{x}_{c(i)}$$

```
>>                                      sh(EToV,V
```


(e) Before smoothening


(f) After smoothening

Enhancing mesh quality by removing high valence nodes

Node valence is defined as the number of edges attached to a node.

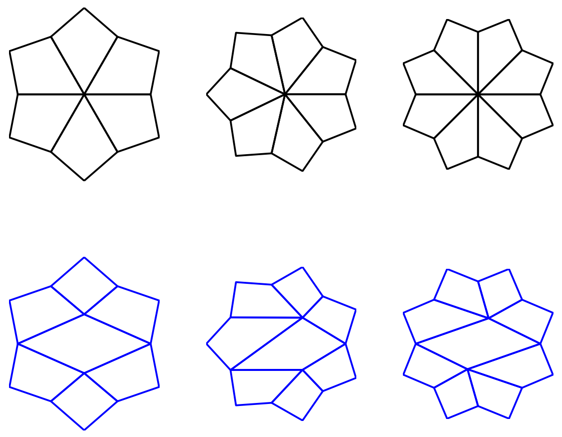
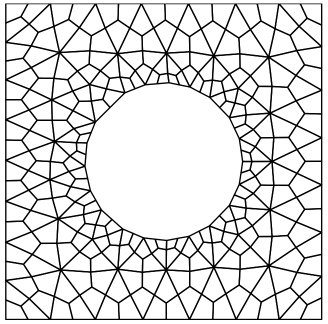
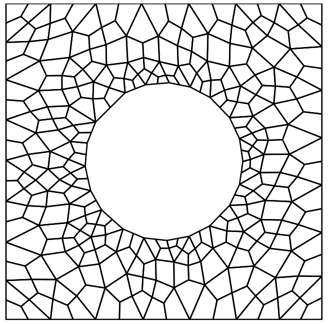


Figure: Templates for removing high valence nodes in quad meshes by inserting an element. Top row: Meshes with one high valence node. Bottom row: Meshes with high valence node removed.

Enhancing mesh quality by removing high valence nodes

```
>> for q = 1:2
      for Valence = [8:-1:6]
          [EToV,VX,VY,K,Nv] = RemovalHighValenceNodes(EToV,VX,VY,Valence);
          [VX,VY] = MeshSmoothingQuadMesh(EToV,VX,VY,20);
      end
   end
```


(a) Before node removal


(b) After node removal and smoothening

Mesh quality

To try and improve initial mesh quality a number of mesh post-processing and cleanup techniques are often used

- Mesh cleanup or improvement techniques (node/edge swapping, point removal/insertion, etc.)
- Mesh smoothing (move vertices to improve mesh quality)
- Mesh coarsening and improvement (doublet removal, edge rotation, diagonal collapse, etc.)
- Feature preserving techniques (no change of boundary shapes, aligned interior mesh interfaces, etc.)

To support assessment of the quality of meshes it is useful to introduce some metrics for characterizing mesh properties.
Remark: In fact, the quality of any mesh depends on the true solution of the physical problem.

Mesh quality metrics

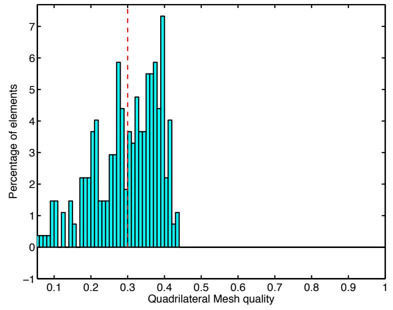
For assessment of mesh quality and automatic detection of bad elements (not always visible!) it is useful to introduce mesh quality metrics.
Simple metric. A simple metric can be based on the internal angles θ_i measured at the vertices at the inside of the quadrilateral, $i = 1, 2, 3, 4$, of the quadrilateral in the form

$$q = \prod_{i=1}^{4} \left(1 - \left|\frac{\frac{\pi}{2} - \theta_i}{\frac{\pi}{2}}\right|\right), \quad 0 \leq q \leq 1$$

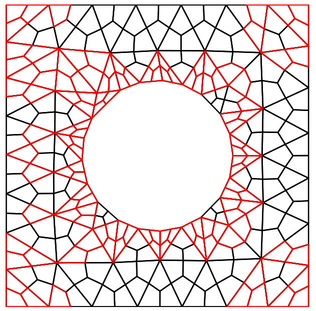
This quality indicator is not taking into account the aspect ratio, which may also be important. At the extremities, the metric can detect rectangles ($q = 1$) and quadrilaterals which have been distorted to triangles ($q = 0$).

Mesh quality metrics

```
>> [q] = MeshQualityQuads(EToV,VX,VY);
```



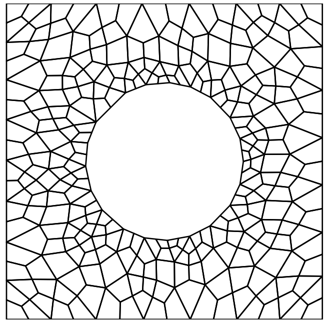
(c) Metric statistics.

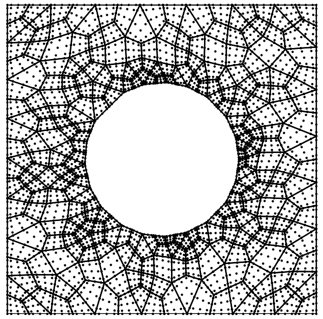

(d) Quad mesh just after Tet→Quad conversion. Bad elements detected in red.

Remark: Threshold for automatically detecting good and bad elements is somewhat arbitrary as it is based entirely on element shape without any consideration of the solution.

Mesh for discretization

The elements of a final mesh can be equipped with nodes through transfinite mappings based on tensorproducts of grid points as illustrated.

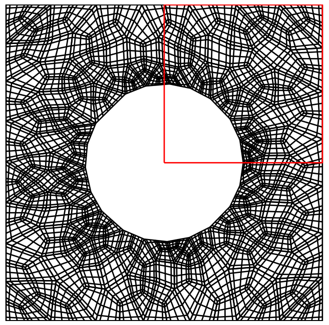

(e) Before transfinite interpolation of nodes.

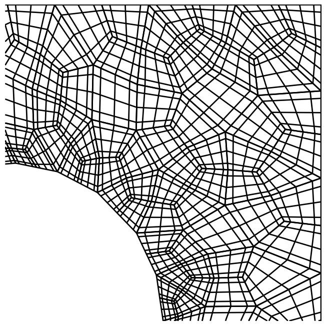

(f) After transfinite interpolation of nodes ($N = 4$ and LGL).

High-order basis functions can alleviate the "pain" in mesh generation..."

Mesh for discretization

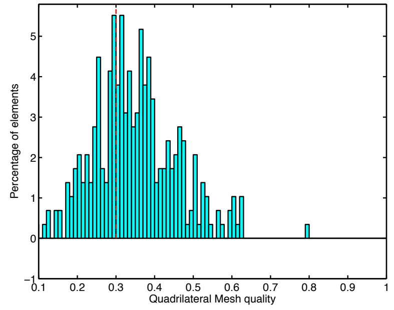
Grid lines are local to the elements as illustrated.


(g) Grid lines of final mesh

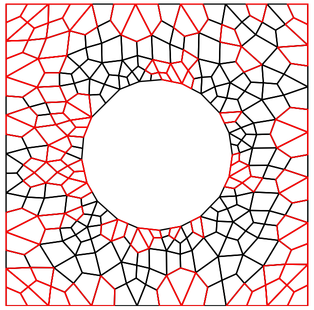

(h) Corner zoom.

Mesh quality metrics

```
>> [q] = MeshQualityQuads(EToV,VX,VY);
```



(i) Metric statistics.


(j) Quad mesh after Tet→Quad conversion, node removal and smoothening. Bad elements detected in red.

Some improvements, however, we need to play more tricks to improve further...

Open source mesh generation software

Unstructured mesh generation software

- DistMesh (`http://www-math.mit.edu/~persson/mesh/`)
- Triangle (`http://www.cs.cmu.edu/~quake/triangle.html`)
- Mesh2D (`http://www.mathworks.com/matlabcentral/`)
- Gmsh (`http://www.geuz.org/gmsh/`)

Note: list is not exhaustive.

Do you have a favorite?

References

Mesh generation software

- Owen, J. S. (1999) A Survey of unstructured mesh generation technology.
  A Meshing Software Survey (from 1999?) can be found at `http://www.andrew.cmu.edu/user/sowen/softsurv.html`.

On the use of Mesh Quality Metrics

- Field, D. A. (2000) Qualitative Measures for Initial Meshes. International Journal for Numerical Methods in Engineering 47:887906.
- Shewchuk, J. R. What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. Preprint.
- Knupp, P. M. (2007) Remarks on Mesh Quality.

Putting the pieces together in a 2D code

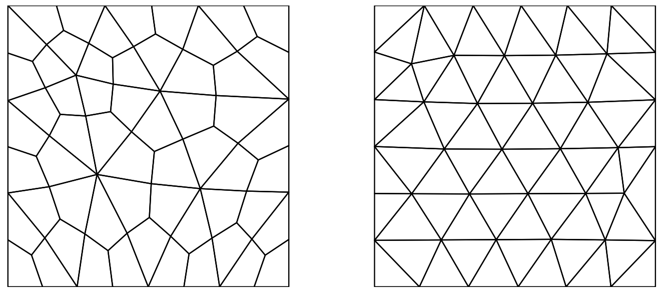
Consider the linear advection equation in 2D

$$\partial_t u + c_x \partial_x u + c_y \partial_y u = 0, \quad (x, y) \in [-1, 1]^2$$

with IC and BC conditions defined using exact solution

$$u(x, y, t) = \sin(\pi(x - c_x t)) \sin(\pi(y - c_y t))$$

Decomposition of domain in terms of quadrilaterals or triangles can be done as



Putting the pieces together in a 2D code

$$\frac{du_h^k}{dt} = -(c_x \mathcal{D}_x^k + c_y \mathcal{D}_y^k) u_h^k + (\mathcal{M}^k)^{-1} \oint_{\partial D^k} \hat{n} \cdot (f^k - f^*) h(\mathbf{x}) d\mathbf{x}^k$$

$$f^*(u^-, u^+) = \kappa \mathbf{c} u^- + (1 - \kappa) \mathbf{c} u^+, \quad \kappa = \tfrac{1}{2}\left(1 + \frac{\hat{\mathbf{n}} \cdot \mathbf{c}}{|\hat{\mathbf{n}} \cdot \mathbf{c}|}\right)$$

```
function [rhsu] = AdvecRHS2Dupwind(u, timelocal, cx, cy)
% Purpose  : Evaluate RHS flux in 2D advection equation by upwinding
Globals2D;
% Define flux differences at faces
df = zeros(Nfp*Nfaces,K);
cn = cx*nx(:) + cy*ny(:); % normalized phase speed in normal direction
kappa = 0.5*(1+cn./abs(cn)); % if kappa=1 then outflow => pick uM, if kappa=0 then inflow  => pick uP
ustar = (kappa .* u(vmapM) + (1-kappa) .* u(vmapP));
df(:) = u(vmapM) - ustar;

% Impose boundary conditions at inflow
Lx = 2; Ly = 2; uexact = sin(2*pi/Lx*(Fx-cx*timelocal)).*sin(2*pi/Ly*(Fy-cy*timelocal));
uin = uexact(mapD);
uM  = u(vmapD);
df(mapD) = uM - (kappa(mapD) .* uM(:) + (1-kappa(mapD)) .* uin(:));
df(:) = cn.*df(:);

% compute right hand sides of the PDE's
[ux,uy] = Grad2D(u);
rhsu  = -(cx*ux + cy*uy) + LIFT*(Fscale.*df);
return
```

2D Advection equation

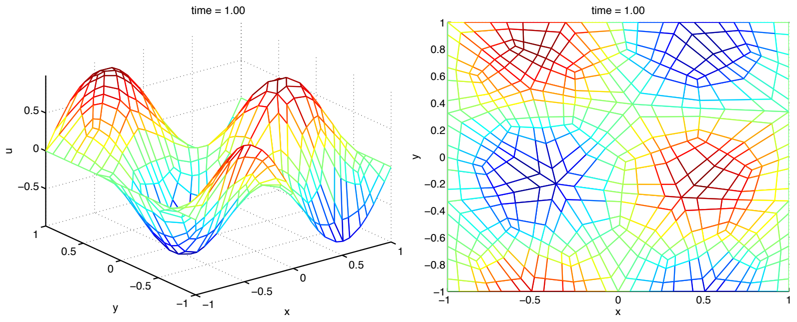


Figure: a) Snapshot of Computed solution using quadrilateral mesh. b) Top view of computed solution. No. elements $K = 57$ and poly. order $N = 3$. $(c_x, c_y) = (1, 0.3)$.

2D Advection equation

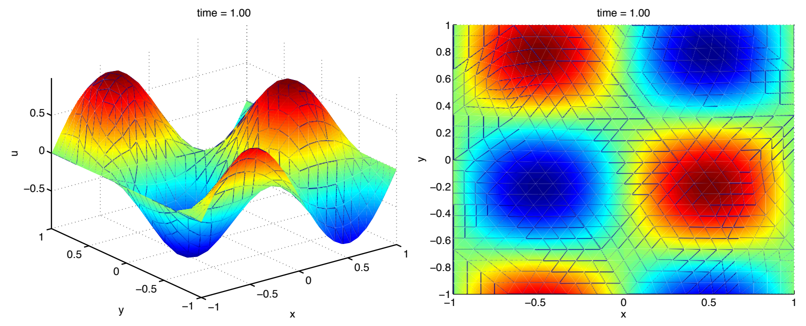


Figure: a) Snapshot of Computed solution using triangle mesh. b) Top view of computed solution. No. elements $K = 65$ and poly. order $N = 3$. $(c_x, c_y) = (1, 0.3)$.

Remark: Same solver used for both quad and tri meshes.

Rotating Hill Problem (2D)

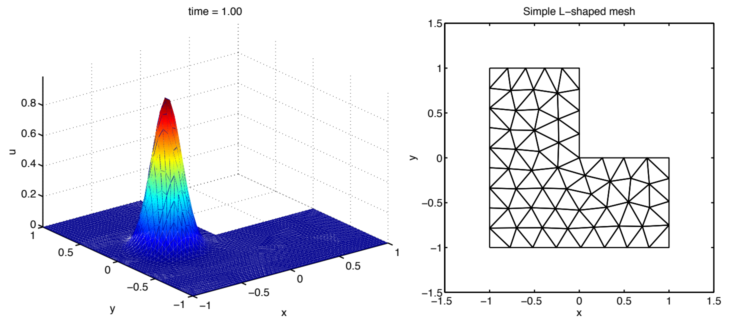


Figure: a) Snapshot of Computed solution using triangle mesh. b) L-shaped domain.