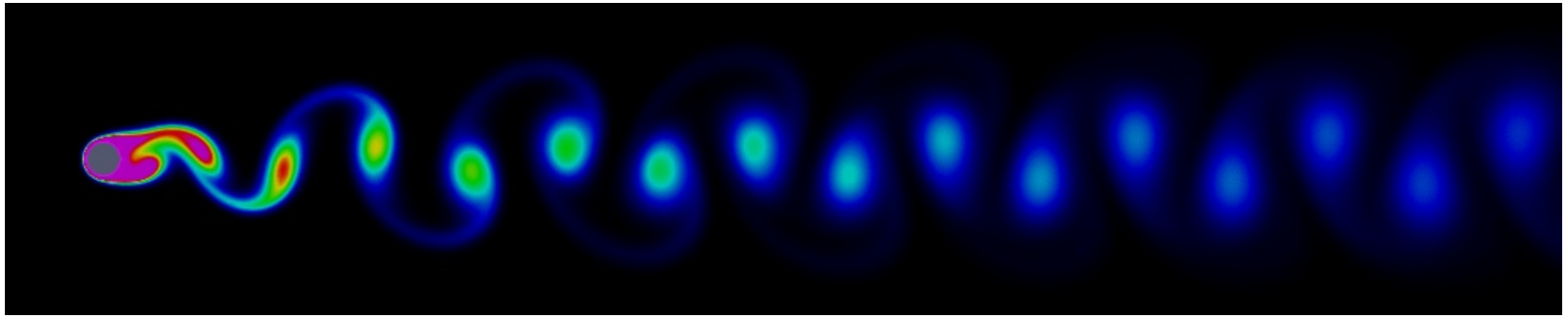


ffmatlib

Technical Information

Objective

- **ffmatlib** is a library to plot and post process **FreeFem++** simulation data in **Matlab** and **Octave**
- 2D type simulations as well as 3D type
- As of 20/05/2019 native support of
  - {P0,P1,P1b,P2} elements in 2D
  - {P0,P1,P2} elements in 3D
  - And vector valued elements as well
- License: GNU General Public License v3.0
- Download Source Code

Part I

Introduction

Usage

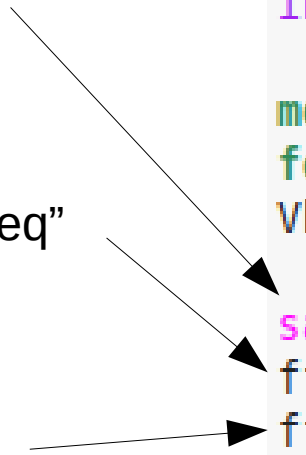
- In *.edp export the mesh and data by library functions:
  - include "ffmatlib.idp"
  - savemesh(), ffsaveVh(), ffsaveData()
- In Matlab / Octave read, plot and interpolate the data with the functions:
  - ffreadmesh(), ffreaddata(), ffpdeplot(), ffpdeplot3D(), ffinterpolate()

FreeFem++

export_mesh.msh
contains the “mesh” data

export_vh.txt contains the
“FE space sequence”, “Vhseq”
or “FE space connectivity”

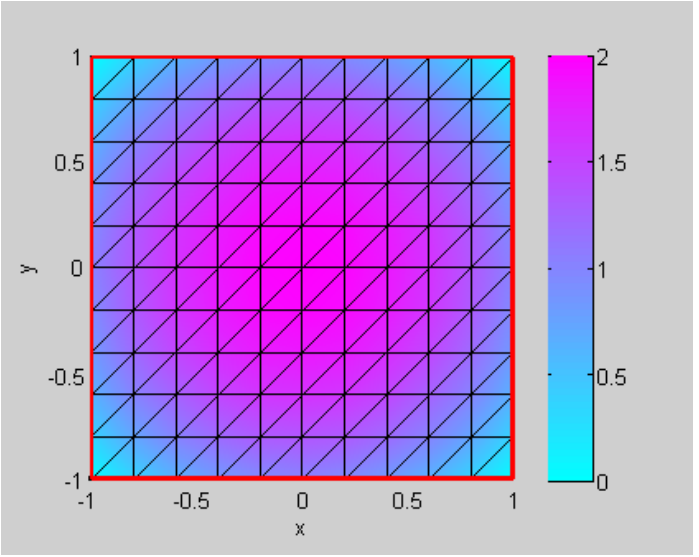
export_data.txt contains the
“PDE solution” or “data”



```
include "ffmatlib.idp"

mesh Th = square(10, 10, [2*x-1, 2*y-1]);
fespace Vh(Th, P1);
Vh u = 2-x*x-y*y;

savemesh(Th,"export_mesh.msh");
ffSaveVh(Th,Vh,"export_vh.txt");
ffSaveData(u,"export_data.txt");
```

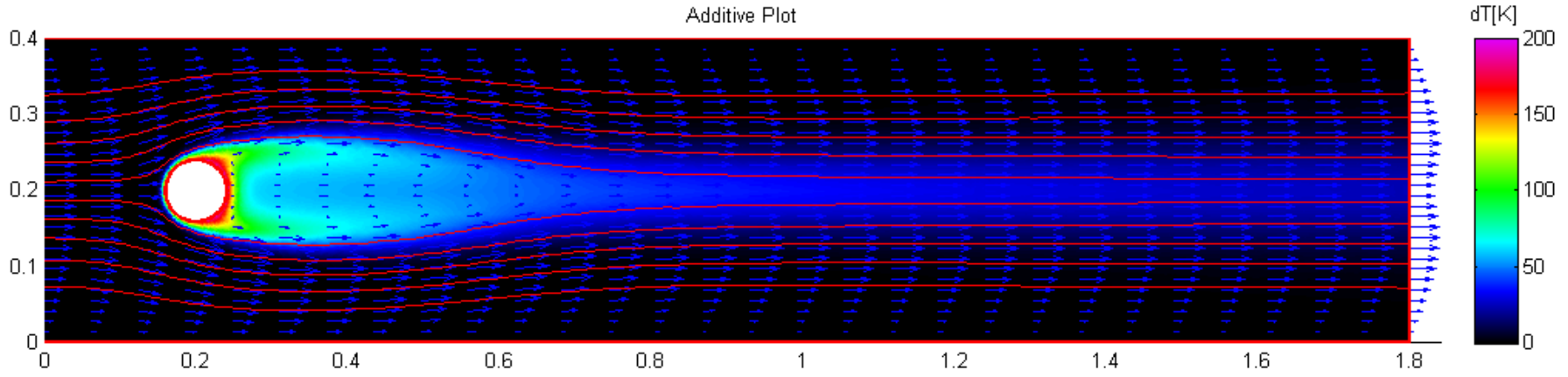
Matlab / Octave

```
addpath('ffmatlib');

[p,b,t] = ffreadmesh('export_mesh.msh');
[vh] = ffreaddata('export_vh.txt');
[u] = ffreaddata('export_data.txt');
ffpdeplot(p,b,t,'VhSeq',vh,'XYData',u,'Mesh','on','Boundary','on');
ylabel('y');
xlabel('x');
```



Customizing

- Meshplot, 2D Map Plot, 3D Surface Plot, Contour Plot, Vectorfield Plot (Quiver), Label Plot, Region Plot, Colormap adjustment, Crosssections and many more functions
- See the example folder and the documentation

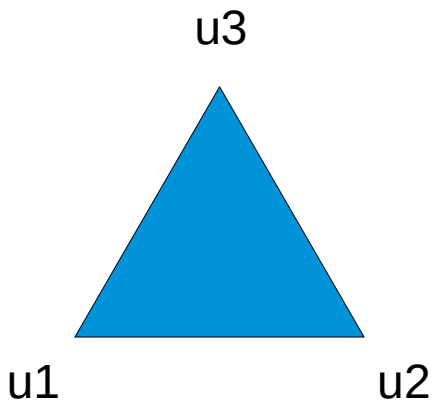


Part II

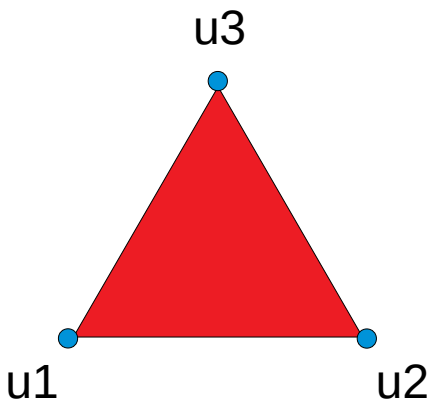
Expert information

Lagrangian Element Numbering - Convention

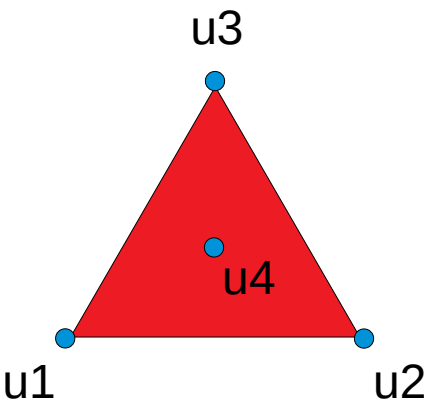
P0
nDoFK=1



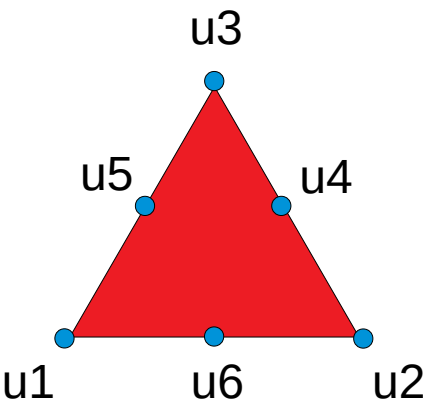
P1
nDoFK=3



P1b (P1+Bubble)
nDoFK=4

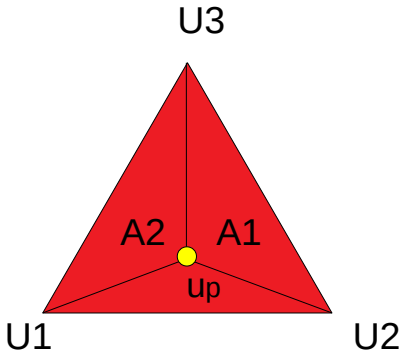


P2
nDoFK=6



Base Functions

- Barycentric coordinates: $w_1 = \frac{A_1(x,y)}{A_0}; w_2 = \frac{A_2(x,y)}{A_0}$
- Interpolation methods:



P1

$$N_1 = w_1$$

$$N_2 = w_2$$

$$N_3 = (1 - w_1 - w_2)$$

$$u_p = u_1 N_1 + u_2 N_2 + u_3 N_3$$

P1b

$$N_1 = w_1$$

$$N_2 = w_2$$

$$N_3 = (1 - w_1 - w_2)$$

$$u_p = u_1 N_1 + u_2 N_2 + u_3 N_3 + 9(3u_4 - (u_1 + u_2 + u_3))N_1 N_2 N_3$$

P2

$$N_1 = w_1(2w_1 - 1)$$

$$N_2 = w_2(2w_2 - 1)$$

$$N_3 = (1 - w_1 - w_2)(1 - 2w_1 - 2w_2)$$

$$N_4 = 4w_2(1 - w_1 - w_2)$$

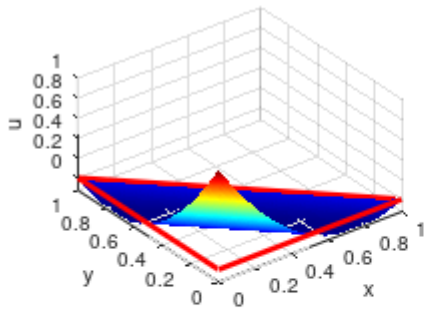
$$N_5 = 4w_1(1 - w_1 - w_2)$$

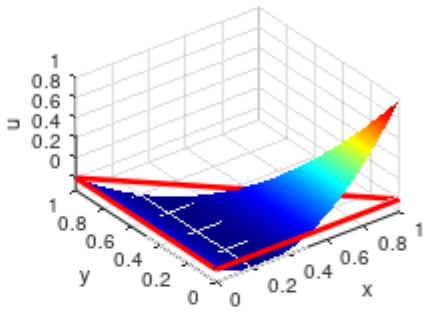
$$N_6 = 4w_1 w_2$$

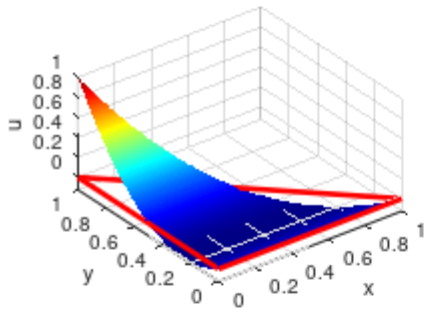
$$u_p = u_1 N_1 + u_2 N_2 + u_3 N_3 + u_4 N_4 + u_5 N_5 + u_6 N_6$$

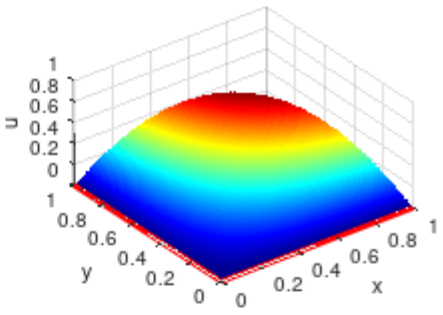
Base Functions

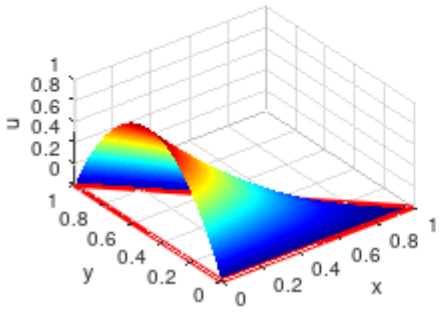
P2

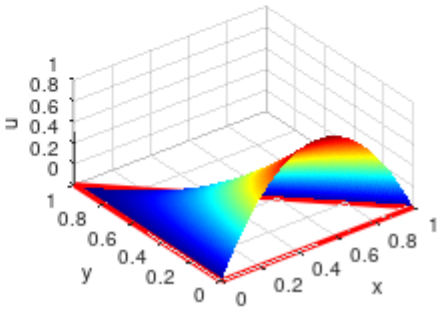




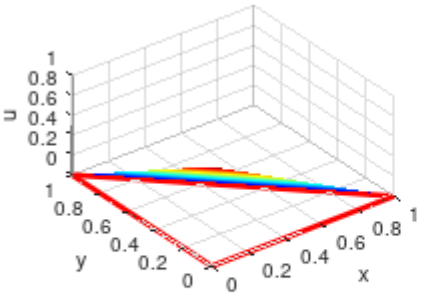


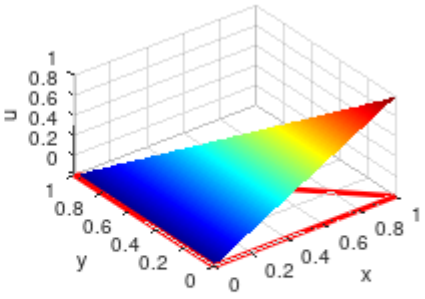


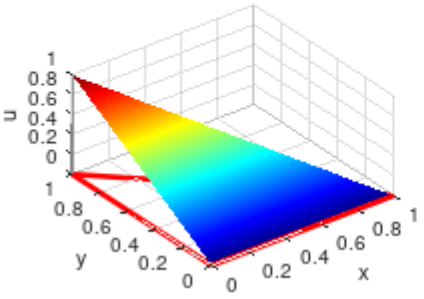


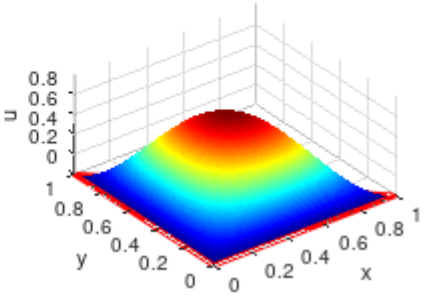


P1b









Interpolation

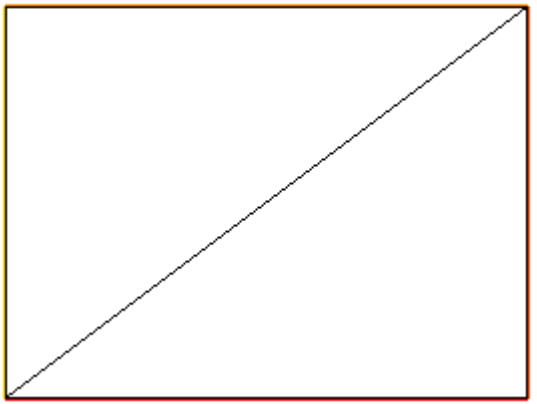
- Compute the function values w1, w2, ... over a mesh grid defined by the arguments x, y from a set of functions u1, u2, ... with values given on a triangular mesh tx, ty
- Located in fftri2grid.m, fftet2grid.m
- fftri2gridfast.c, fftet2gridfast.c are MEX - implementations (runtime improved)
- The high level wrapper functions are:
  - ffpdeplot() and ffpdeplot3D()
  - ffinterpolate() which is used to interpolate on cartesian or curved grids

Part III

Library code overview

Meshfile Example nt=2

```
mesh Th=square(1,1,[0.8*x,0.6*y]);
```



```
nb of Triangles = 2
Th[0][0]=0      x=0     y=0     label=4
Th[0][1]=1      x=0.8   y=0     label=2
Th[0][2]=3      x=0.8   y=0.6   label=3
Th[1][0]=0      x=0     y=0     label=4
Th[1][1]=3      x=0.8   y=0.6   label=3
Th[1][2]=2      x=0     y=0.6   label=4
```

```
4 2 4
0 0 4
0.8 0 2
0 0.6 4
0.8 0.6 3
1 2 4 0
1 4 3 0
1 2 1
2 4 2
4 3 3
3 1 4
```

1: Header (section sizes)

2: (x,y) coordinates of four points + label number

3: Connectivity for two triangles (nt=2): A set of 3 numbers per triangle indicating the position (index) where to find the nodal coordinates in the points list in section #2 + region number

4: Boundary (set of edges defined by two points) + label number

Recall: A mesh looks always the same regardless of the used Lagrangian Element (P1,P2 …)!!!

Mesh Conversion

- A mesh is read with [p,b,t]=ffreadmesh()
- Deconvolution into “triangle format” done in the file **prepare_mesh.m**:
  - Resolving the connectivity
  - Plug in the coordinates
- xmesh: Matrix (nt columns x 3 rows) containing the x-coordinates of the triangles
- ymesh: Matrix (nt columns x 3 rows) containing the y-coordinates of the triangles

```
xpts=points(1,:);
ypts=points(2,:);
xmesh=[xpts(triangles(1,:)); xpts(triangles(2,:)); xpts(triangles(3,:))];
ymesh=[ypts(triangles(1,:)); ypts(triangles(2,:)); ypts(triangles(3,:))];
```

```
debug> xpts
xpts =

   0.00   0.80   0.00   0.80

debug> ypts
ypts =

   0.00   0.00   0.60   0.60

debug> xmesh
xmesh =

   0.00   0.00
   0.80   0.80
   0.80   0.00

debug> ymesh
ymesh =

   0.00   0.00
   0.00   0.60
   0.60   0.60
```

PDE-Solution (data)

- For P1 the data is arranged in the same way as the nodal mesh coordinates are located in the mesh file (i.e. the mesh connectivity is equal to the “data connectivity” in u)
- For P1b and P2 the data (connectivity) is scrambled. However the information where to find the nDoFK-values belonging to a given triangle number can be found in the vhseq-file
- Note: The vhseq-concept is universal and works also for P1 data

```
fespace Vh(Th,P1);
Vh u=x+y;
```

u=[0;0.8;0.6;1.4];

```
fespace Vh(Th,P1b);
Vh u=x+y;
```

u=[0.8;0.733;0;1.4;0.6;0.666];

```
fespace Vh(Th,P2);
Vh u=x+y;
```

u=[0.8;0.4;1.1;0;0.7;1.4;0.6;0.3;1];

Vhseq

- Contains nt*nDoFK number of elements
- Contains the indices where to find the nDoFK-PDE data values stored in the u-data array for a given triangle-number
- Can be simply understood as a hash– or look up table. Hence named as “sequence” of the “FE space”
- Note: It would be possible to skip the vhseq-file and write the PDE-data in triangle format in the *.edp but this enlarges the data-file size and cause problems for vectored FE-spaces

```
fespace Vh(Th,P1);
Vh u=x+y;
```

vhseq=[0;1;3;0;3;2];

```
fespace Vh(Th,P1b);
Vh u=x+y;
```

vhseq=[2;0;3;1;2;3;4;5];

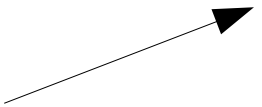
```
fespace Vh(Th,P2);
Vh u=x+y;
```

vhseq=[3;0;5;2;4;1;3;5;6;8;7;4];

PDE Data Decoding

- In **convert_pde_data.m** the simulation type (P0..P2) is determined via the length of vhseq which equals nDoFK*nt assuming the Lagrangian Elements have a unique nDoFK-value
- The PDE-data is converted into “triangle format”
- Result has size of (nt columns x nDoFK rows)
- Recall that (xmesh,ymesh) stores the nodal coordinates

```
switch (length(vhseq))
    case (3*nt)
        elementType='P1'; %nDoF=3 / element
        xydata=reshape(cCols(vhseq+1),3,nt);
    case (4*nt)
        elementType='P1b'; %nDoF=4 / element
        xydata=reshape(cCols(vhseq+1),4,nt);
    case (6*nt)
        elementType='P2'; %nDoF=6 / element
        xydata=reshape(cCols(vhseq+1),6,nt);
end
```

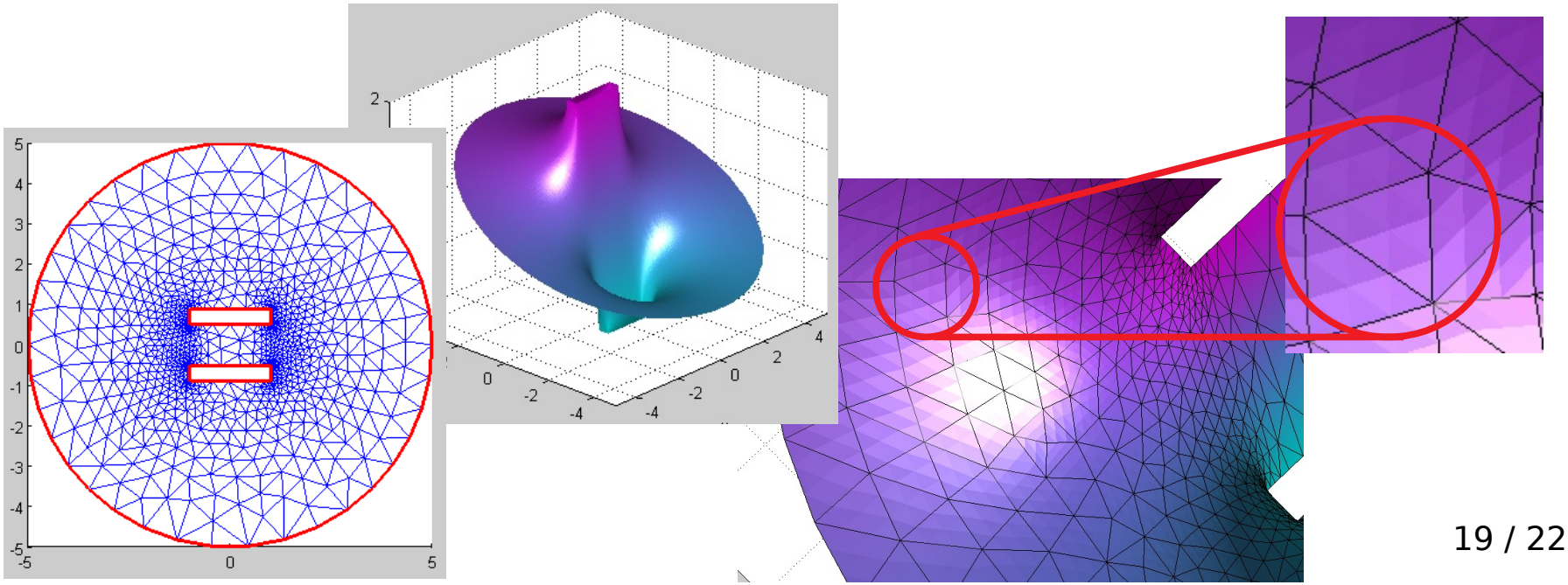


```
xydata =
{
  [1,1] =

    0.00   0.00
    0.80   1.40
    1.40   0.60
    1.10   1.00
    0.70   0.30
    0.40   0.70

}
```

2 triangles (2 columns) containing the P2-PDE-data

Plot Algorithmus

- Meshrefinement
  - 1:9 for P2 Elements
  - 1:3 for P1b Elements (note that FreeFem++ plots 1:1)
  - 1:1 for P1 Elements



Meshrefinement (I)

- Coordinates of the refined mesh are prepared in the file **prepare_coordinates.m**
- The refined mesh coordinates are stored in (xdata,ydata)
- 3D surf plots are critical if ‘Mesh’ is ‘on’ because of interlacing effects → It is necessary to prepare extra refined mesh coordinates for “z-style” mesh - plots (xdataz, ydataz)
- Example P2-Refinement (as per “Numbering Convention”):

  {1,16,15},{16,62,c},{62,2,24},{24,43,c},{43,3,35},{35,15,c},{15,1,16},
  {15,16,c},{62,24,c},{43,35,c} where c denotes the barycenter

```
debug> xdata
xdata =

   0.00   0.00   0.27   0.27   0.53   0.53   0.80   0.53   0.80   0.27   0.53   0.00   0.27   0.00   0.53   0.53   0.80   0.27
   0.27   0.27   0.53   0.53   0.80   0.80   0.80   0.27   0.80   0.00   0.27   0.00   0.27   0.27   0.80   0.53   0.53   0.00
   0.27   0.00   0.53   0.27   0.80   0.53   0.53   0.27   0.53   0.00   0.53   0.27   0.53   0.27   0.53   0.27   0.53   0.27

debug> ydata
ydata =

   0.00   0.00   0.00   0.20   0.00   0.40   0.20   0.60   0.40   0.60   0.40   0.40   0.20   0.20   0.00   0.40   0.40   0.60
   0.00   0.20   0.00   0.40   0.00   0.60   0.40   0.60   0.60   0.60   0.20   0.20   0.00   0.20   0.20   0.60   0.40   0.40
   0.20   0.20   0.20   0.40   0.20   0.60   0.20   0.40   0.40   0.40   0.20   0.40   0.20   0.40   0.20   0.40   0.20   0.40
```

Meshrefinement (II)

- The color data (cdata) linked to the refined mesh (xdata,ydata) is prepared in the file **prepare_data.m**
- If necessary an interpolation is performed in order to create new points (e.g. P2 interpolation by 2nd-order base functions)
- Returns the refined PDE data as “varargout”: The reason is that ffpdeplot() is able to plot not only scalar but also 2d vector fields
- Offtopic: ffinterpolate() is capable to interpolate real or complex vector fields of dimension up to R^n=99

```
cdata =

   0.00   0.00   0.27   0.47   0.53   0.93   1.00   1.13   1.20   0.87   0.93   0.40   0.47   0.20   0.53   0.93   1.20   0.87
   0.27   0.47   0.53   0.93   0.80   1.40   1.20   0.87   1.40   0.60   0.47   0.20   0.27   0.47   1.00   1.13   0.93   0.40
   0.47   0.20   0.73   0.67   1.00   1.13   0.73   0.67   0.93   0.40   0.73   0.67   0.73   0.67   0.73   0.67   0.73   0.67
```

ffpdeplot()

- The management of the mesh (xmesh,ymesh), the refined mesh (xdata,ydata) as well as (xdataz,ydataz) in order to create the correct plot is task of **ffpdeplot.m**
- Because of runtime reasons the patch command is invoked in different ways for P1, P1b and P2