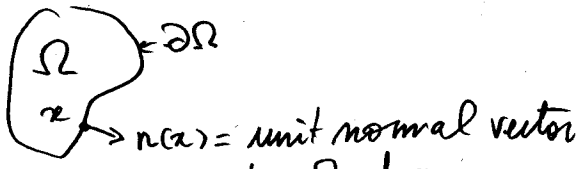
The variational problem in 2D

We consider the problem:

$$\begin{cases} -\Delta u = f & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases}$$



To find WF we multiply PDE on both sides by some test function

$$v \in V = \{ v \in H^1(\Omega) \mid v|_{\partial\Omega} = 0 \}$$

and integrate:

$$-\int_\Omega v \Delta u \, dx = \int_\Omega f v \, dx$$

Then we use <u>Green's identity</u> (the analogous of integration by parts in higher dimensions)

$$\int_\Omega -v \Delta u \, dx = \int_\Omega \nabla v \cdot \nabla u \, dx - \int_{\partial\Omega} v (\nabla u \cdot n) \, dS$$

But $v|_{\partial\Omega} = 0$, thus we get the WF:

> Find $u \in V$ s.t.
> $$a(u,v) = (f,v) \quad \forall v \in V.$$

where

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \, dx$$

$$(f,v) = \int_\Omega f v \, dx$$

<u>Note</u>: other boundary conditions can be considered. For example:

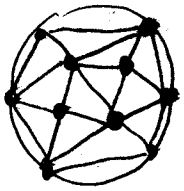
$$\begin{cases} -\Delta u = f & \text{in } \Omega \\ u = 0 & \text{on } \Gamma \subset \partial\Omega \\ n \cdot \nabla u = h & \text{on } \partial\Omega \setminus \Gamma \end{cases} \Rightarrow$$

Find $u \in V = \{ v \in H^1 \mid v|_\Gamma = 0 \}$

$$a(u,v) = (f,v) + \int_{\partial\Omega} v h \quad \forall v \in V.$$

Implementation of 2D triangular P1 elements

Triangulation:



Given the coordinates of points in 2d (or even n-dimensions) it is possible to find a triangulation that maximizes the minimal angle of the triangles
→ Delaunay triangulation.

Well known open source library: QHULL

There is an easy to use Matlab interface:

```
tri = delaunay(x, y);
```

x = vertex with x coord
y = ——— y ———

tri = #tri × 3 matrix

each row of t gives index (in x, y) of vertex of a triangle.

To further control quality of mesh (min angle) one can adjust position of nodes and add/remove nodes → many mesh generators out there:

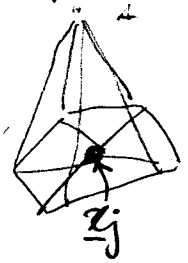
- triangle - Shewchuck
- distmesh - Strang and Persson (simple to use and understand)

If we are using P1 elements  dof are values at vertices.
Therefore the local to global map is given automatically by tri.

tri(e, j) is the global index of j-th vertex of triangle e

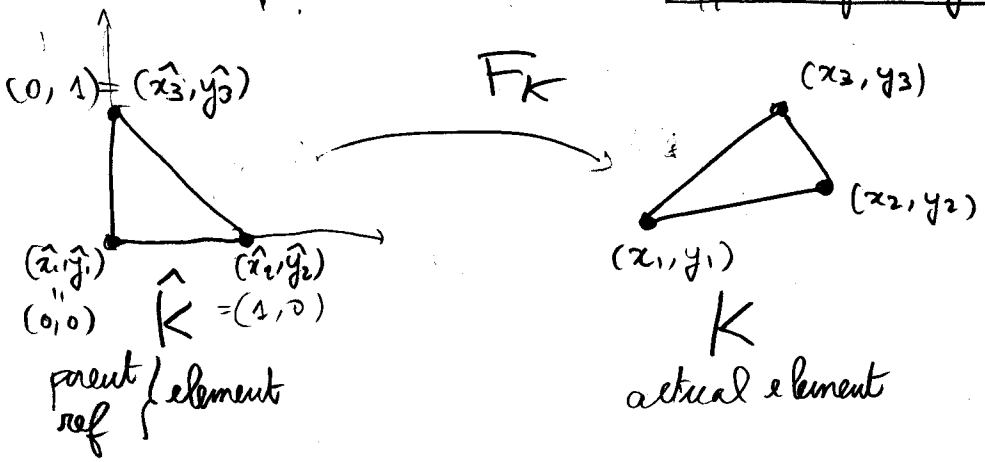
Basis functions of V_h:

$\phi_j(\underline{x})$



they are piecewise linear functions with value one at one node and zero at all other → pyramid shape.

P_1 triangular elements are an affine family of elements:



$$F_K(\hat{x},\hat{y}) = B\begin{bmatrix}\hat{x}\\ \hat{y}\end{bmatrix} + b = \text{invertible affine mapping}$$

$$= \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - x_1 & y_3 - y_1 \end{bmatrix} \begin{bmatrix}\hat{x}\\ \hat{y}\end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Can check that:

$$F_K(\hat{x}_i, \hat{y}_i) = \begin{pmatrix} x_i \\ y_i \end{pmatrix}, \quad i = 1,2,3$$

So the basis functions restricted to element K can be expressed by means of F_K in terms of basis functions at parent element.

$$\hat{\phi}_i(\hat{x}_j, \hat{y}_j) = \delta_{ij}$$

$$\Rightarrow \boxed{\begin{aligned} \hat{\phi}_1(\hat{x},\hat{y}) &= 1 - \hat{x} - \hat{y} \\ \hat{\phi}_2(\hat{x},\hat{y}) &= \hat{x} \\ \hat{\phi}_3(\hat{x},\hat{y}) &= \hat{y} \end{aligned}} \quad \text{(check)}$$

Local basis functions have then the expression:

$$\phi_i^K(x,y) = \hat{\phi}_i\left(F_K^{-1}(x,y)\right) = \left(\hat{\phi}_i \circ F_K^{-1}\right)(x,y)$$

ith local basis function at element K

Stiffness matrix:

As in 1D we assemble the (global) stiffness matrix by adding the contribution of each element:

for e = 1 : # of triangles,

$$A(tri(e,:), tri(e,:)) = A(tri(e,:), tri(e,:)) + A_{loc}^{K_e};$$

where $$(A_{loc}^{K_e})_{ij} = a_{K_e}(\phi_i^{K_e}, \phi_j^{K_e}) = \int_{K_e} \nabla\phi_i^{K_e}(x,y)\nabla\phi_j^{K_e}(x,y)\,dxdy$$

for $i,j = 1..3$ (idx of local dof).

↑ local bilin. form

[Here we assume we are solving Laplace's equation: $\begin{cases} -\Delta u = f \text{ in } \Omega \\ u = 0 \text{ on } \partial\Omega \end{cases}$]

The above loop works because:

assuming polygonal domain Ω

$$A_{ij} = a(\phi_i, \phi_j) = \int_\Omega \nabla\phi_i \cdot \nabla\phi_j = \sum_{K \text{ elem}} \int_K \nabla\phi_i \cdot \nabla\phi_j$$

$$= \sum_{K \text{ element}} a_K(\phi_i, \phi_j)$$

And if $v, w \in V_h$:

$$a_k(v,w) = a_k\left(\sum_{j=1}^{3} v_{tri(k,j)}\,\phi_j^k,\ \sum_{i=1}^{3} w_{tri(k,i)}\,\phi_i^k\right)$$

$$= \sum_{i,j=1}^{3} v_{tri(k,j)}\, w_{tri(k,i)}\, a_k(\phi_j^k, \phi_i^k)$$

$$= v_{tri(k,:)}^T \, K_{loc}\, w_{tri(k,:)}$$

Local stiffness matrix calculation

Idea: change of variables and chain rule.

Recall:

- change of variables (c.o.v)

$$\int_\Omega f(\underline{x})\,d\underline{x} \underset{\uparrow}{=} \int_{\hat{\Omega}=F(\Omega)} f(F^{-1}(\hat{\underline{x}}))\,\left|\det DF[F^{-1}(x)]\right|\,d\hat{\underline{x}}$$

$$F(\underline{x}) = \hat{\underline{x}}$$

- chain rule (for gradients)

$$\nabla(f\circ F)(x) = DF^T[x]\,(\nabla f)(F(x))$$

Thus:

$$(A_{loc}^k)_{ij} = \int_k \nabla\phi_i^k(x,y)\cdot\nabla\phi_j^k(x,y)\,dx\,dy$$

$$= \int_k \nabla(\hat{\phi}_i\circ F_k^{-1})(x,y)\cdot\nabla(\hat{\phi}_j\circ F_k^{-1})(x,y)\,dx\,dy$$

$$\overset{\text{chain rule}}{=} \int_k \left[DF_k^{-T}\,\nabla\hat{\phi}_i(F_k^{-1}(x,y))\right]^T DF_k^{-T}\nabla\hat{\phi}_j(F_k^{-1}(x,y))\,dx\,dy$$

$$\overset{\text{c.o.v.}}{=} \int_{\hat{k}} \nabla\hat{\phi}_i(\hat{x},\hat{y})^T\left(DF_k^{-1}DF_k^{-T}\right)\nabla\hat{\phi}_j(\hat{x},\hat{y})\cdot$$

$$\left|\det DF_k\right|\,d\hat{x}\,d\hat{y}$$

Now the gradients are constant on each element because we are using P1 polyn in each element.

Let $G = \left[\nabla\hat{\phi}_1 \quad \nabla\hat{\phi}_2 \quad \nabla\hat{\phi}_3\right] = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}$

Notice that since we have affine elements we have

$DF_K = B_K$ = some matrix that is constant on each element.

$$\Rightarrow \boxed{A^K_{loc} = \int_{\hat{K}} G^T \left(B_K^T B_K\right)^{-1} G \, |\det(B_K)| \, d\hat{x}\, d\hat{y}}$$

$$= \boxed{|\hat{K}| \, |\det(B_K)| \; G^T (B_K^T B_K)^{-1} G}$$

$|\hat{K}| = \frac{1}{2}$ = area of parent element.

We can assemble right hand side in a similar way:

Assume for simplicity that $f \in V_h$ (more general f can be considered by doing numerical integration)

Then we add contributions of each element:

for $e = 1 \ldots$ # of triangles

$\quad F(tri(e,:)) = F(tri(e,:)) + Floc^K$

where

$$(Floc)_j = \int_K f(x,y)\, \phi_j^K(x,y)\, dx\, dy$$

$$= \int_K \left(\sum_{i=1}^{3} f(x_{tri(k,i)}, y_{tri(k,i)})\, \phi_i^K(x,y) \right) \phi_j^K(x,y)\, dx\, dy$$

Thus:

$$F_{loc} = M_{loc}^{K} \left[f(x_{tri}(K,:), y_{tri}(K,:)) \right]$$

where $\left(M_{loc}^{K}\right)_{ij} = \int_K \phi_i^K(x,y)\, \phi_j^K(x,y)\, dx\, dy$

To evaluate these integrals we go back to coord. at the ref. elem.

$$\left(M_{loc}^{K}\right)_{ij} = \int_{\hat{K}} \hat{\phi}_i(\hat{x},\hat{y})\, \hat{\phi}_j(\hat{x},\hat{y})\, |\det DF_K|\, d\hat{x}\, d\hat{y}$$

$$= \frac{|\det DF_K|}{24} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

by direct calculations.

Note: f could also be given as a piecewise const. and function on the elements, in this case the construction of RHS is much more simple.

One key aspect of finite elements is that the matrices that are constructed are <u>sparse</u> meaning they only have a few non-zero elements. (Can you see <u>why</u>?)

So it is more convenient both for storage and when doing computations to store only the non-zero elements.

<u>Example</u>:

$$A = \begin{bmatrix} 1 & \cdot & \cdot & 3 & \cdot \\ \cdot & \cdot & 4 & \cdot & 2 \\ 5 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 6 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 7 \end{bmatrix}$$

Can be stored as three vectors:

I	J	VAL
1	1	1
1	4	3
2	3	4
2	5	2
3	1	5
4	3	6
5	5	7

Then matrix vector products (and other operations) can take advantage of sparsity:

compute $w = Av$ ← # of non zero elements of A

```
for p = 1 .. nnz(A)
  | w(I(p)) = w(I(p)) + VAL(p) * v(J(p))
```

Systems $Ax = b$ can be solved efficiently when A is sparse:

- <u>iterative methods</u> (CG can be used for elliptic problems) since applying A to a vector is cheap (see matvec code above)
- <u>direct sparse solvers</u>:
  - UMFPACK (what is behind matlabs Backslash '\')
  - Super LU
  - MUMPS