Compressed Row Storage

Thursday, August 27, 2020 4:06 PM

Compressed Row Storage

$$\begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 4 & 5 & 6 & 0 \\ 0 & 7 & 0 & 0 \end{bmatrix}$$

values = [3, 4, 5, 6, 7]

columns = [1, 1, 2, 3, 2]

row_start = [0, 1, 1, 4, 5]

(look up row i directly)

what column each entry is in

need one extra value, so skip next value in

```
for row = 0 : n-1
    for idx = row_start[row], ..., row_start[row+1]-1
        A(row, columns[idx]) = values[idx]
    otherwise A(i,j) = 0
    end
end
```

Sparse Matrix-Vector product:

Compute $y = A \cdot x$

$$y_i = \sum_j a_{ij} x_j \quad , \quad i = 0, 1, \ldots, n-1$$

```
for i = 0, ..., n-1
    y[i] = 0
    for idx = row_start[i], ..., row_start[i+1]-1
        y[i] = y[i] + values[idx] * x[column[idx]]
```

"random" access, indirect addressing

```
    for idx = row_start : ...
        y[i] = y[i] + values[idx] * x[column[idx]]
    end
end
```

sequentially accessed O(n)

FEM Review

Tuesday, September 1, 2020 3:51 PM

Finite Element Method

$V_h = \langle \phi_1, \dots, \phi_n \rangle$, how to define basis?

$u_h(x) = \sum_i u_i \phi_i(x)$

Ciarlet's definition of a Finite Element

(Ω, P, Σ)

1.) $\Omega \subseteq \mathbb{R}^d$ domain: connected, compact, non-empty, boundary $\partial\Omega$ piecewise linear or Lipschitz

2.) $P \subseteq V(\Omega) = \{v : \Omega \to \mathbb{R}\}$ vector space of functions

3.) Σ set of n linear forms $\sigma_1, \dots, \sigma_n$, called degrees of freedom

$\sigma_i : P \to \mathbb{R}$ (linear)

They form a basis of all $\mathcal{L}(P; \mathbb{R})$

example: $\sigma_i(v) = v(x_i)$, evaluate v at x_i

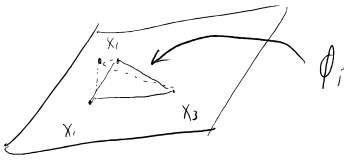
x_i support points $\{x_1, \dots, x_n\}$, $x_i \in \Omega$

$\Rightarrow$ nodal FE space

$\Rightarrow$ There exists a basis $\{\varphi_1, \dots, \varphi_n\}$ of P

with $\sigma_i(\varphi_j) = \delta_{ij}$

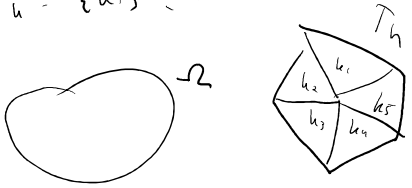
example:



more detail

1.) Subdivide $\Omega \subseteq \mathbb{R}^d$ into mesh

$T_h = \{K_i\}$, K_i cell



T_h is conforming iff $(\forall i \neq j)$

$\bar{K}_i \cap \bar{K}_j$ is $\emptyset$ or a vertex

$\Omega_h = \bigcup \bar{k}_i \approx \Omega$

k_i : simplicies (triangles or tetrahedral)

or k_i : quadrilaterals

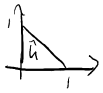
$\mathcal{T}_h$ is conforming

$\bar{k}_i \cap \bar{k}_j$ is $\emptyset$ or a vertex or a face (like in 3D)

Later: allow non-conforming meshes w/ hanging nodes

2.) Reference Cell $\hat{k}$

$\hat{k} = [0,1]^d$ for quads 

$\hat{k} = \{\sum x_i \leq 1\}$ simplices 
$x_i \geq 0$

3.) Mapping from reference cell

$F_{k_i} : \hat{k} \to k_i$

4.) Define local basis on $\hat{k}$ with

local support points $\hat{x}_i \in \hat{k}_i$

basis $\{\hat{\phi}_i\}$, $\hat{\phi}_i : \hat{k} \to \mathbb{R}$

$Q_1 = \text{span}\{1, x, y, xy\}$ 

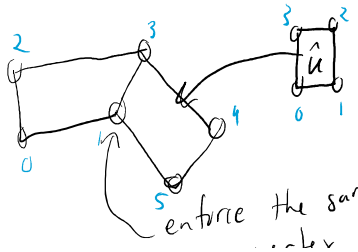
$\mathbb{P} = \text{span}\{1, x, y\}$ 

5.) Number degree of freedom to create

Global basis $\phi_1, \ldots, \phi_n$ tensor-product degree k, FEM space

$Q_k(\mathcal{T}_h) = \{v \in C(\mathcal{T}_h), v|_{k_i} \circ F_{k_i} \in Q_k\}$

on each cell use Q_k from $\hat{k}$, make continuous



local dim: 4

$4 \cdot 2 = 8$ local shape functions

global: dim 6

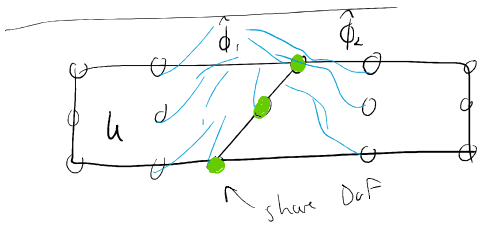
0 global: dim 6

5 enforce the same value in vertex

$l_1: [0,1,2,3] \to [1,5,3,4]$

local DoFs global DoFs

$l_2: [0,1,2,3] \to [0,2,3,1]$



$u_h = \sum u_i \phi_i \in V_h$

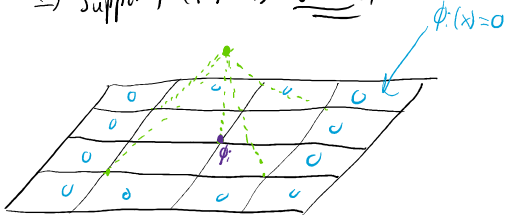
ϕ_i need to be continuous

$\Rightarrow$ combine the local $\hat{\phi}_i$ to be continuous

$\Rightarrow \phi_i(x)|_K = \begin{cases} \hat{\phi}_j, & \exists j : D_K(j) = i \\ 0 \end{cases}$

$\Rightarrow \phi_i(x) \in C(\Omega_h)$, $\forall i \Rightarrow V_h \subseteq C(\Omega_h)$

$\Rightarrow$ support (ϕ_i) is small (one layer of cells!)

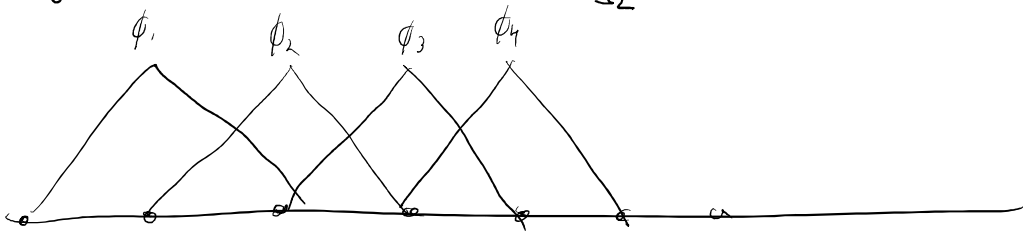


$\Rightarrow$ Tensor-product polynomials on each cell,
glued together continuously
basis: local support

More FEM review

Thursday, September 3, 2020 4:37 PM

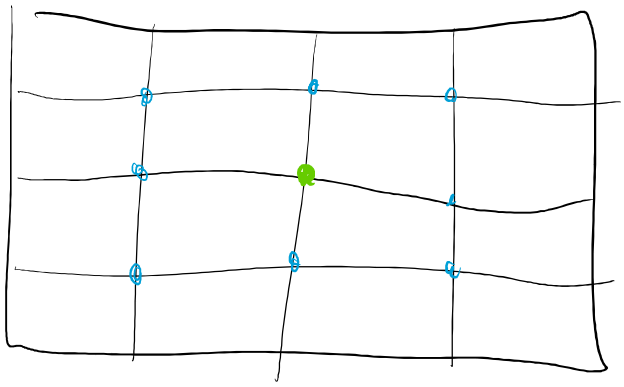
$$A_{ij} = (\nabla\phi_i, \nabla\phi_j) = \int_\Omega \nabla\phi_i(x) \cdot \nabla\phi_j(x)\, dx$$



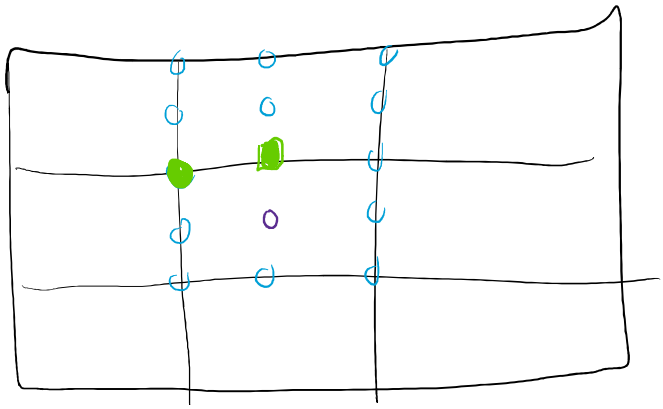
support ϕ_i

support $\nabla\phi_i$

$\Rightarrow$ A is sparse



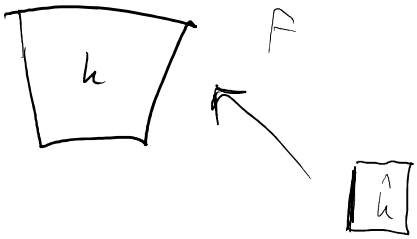
Q1 regular: 9 entries / row



Q_2:

- 25 entries
- 9 entries
- 15 entries

$$A_{ij} = \int_{\Omega_h} \nabla\phi_i(x) \cdot \nabla\phi_j(x)\, dx$$



$$= \sum_{K \in \mathcal{T}_h} \int_K \nabla\phi_i(x) \cdot \nabla\phi_j(x)\, dx$$

$$= \sum_K \int_{\hat{K}} \left(\nabla\phi_i\right)(F(\hat{x})) \cdot \left(\nabla\phi_j\right)(F(\hat{x}))\, \left|\det(J(\hat{x}))\right|\, d\hat{x}$$

Define $\hat{\phi}_i = \phi_i(F(\hat{x}))$

$$\nabla\hat{\phi}_i(x) = \nabla\left[\phi_i\left(F(\hat{x})\right)\right]$$

$$= (\nabla\phi_i)(F(\hat{x})) \cdot \underbrace{\nabla F(\hat{x})}_{= J(\hat{x})}$$

$$= \sum_K \int_{\hat{K}} J^{-1}(\hat{x})\, \nabla\hat{\phi}_i(\hat{x}) \cdot J^{-1}(\hat{x})\, \nabla\phi_j(\hat{x})\, \left|\det(J(\hat{x}))\right|\, d\hat{x}$$

quad rule

$$\approx \sum_K \sum_{q=1}^{n_q} J^{-1}(x_q)\, \nabla\hat{\phi}_i(x_q) \cdot J^{-1}(x_q)\, \nabla\phi_j(x_q)\, \left|\det(J(x_q))\right|\, w_q$$

shape_grad(i,q) · shape_grad(j,q) · $= J_x w(q)$

See step-3

Naive way to assemble:

for $k \in T_h$

for $q = 1 : n_q$

$A_{ij} \mathrel{+}= \text{shape\_grad}(i,q)$

$O(N^3)$ bad

better

for $k \in T_h$ ← local indices!

$M = 0$

for $\hat{i} = 1 : n\_local\_dofs$

for $\hat{j} = 1 : n\_local\_dofs$

for $q : n\text{-}1$

$M(\hat{i},\hat{j}) = \text{shape\_grad}(\hat{i},q)$

$A_{ij} \mathrel{+}= M(\hat{i},\hat{j})$ where $i = \alpha(\hat{i})$ ← dof indices local → global

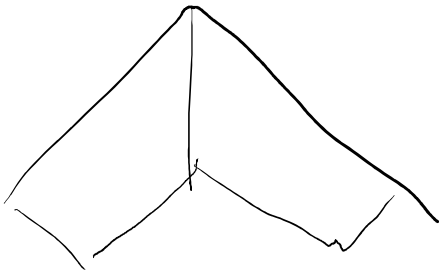
$j = \alpha(\hat{j})$

copy local to global

$\Rightarrow O(N)$:)

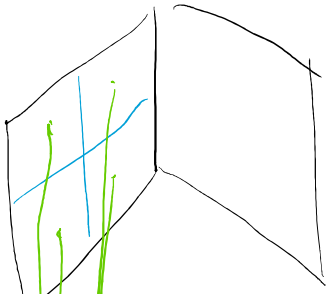
Mesh refinement

Thursday, September 10, 2020 3:38 PM



lvl 0

refine global



lvl 1

n_cells : 6

n_active_cells : 5

n_cells on lvl 0 : 2

n_cells on lvl 1 : 4

quad tree

A cell without children is called

active

Computation is done on active cells

Quadrature

Thursday, September 10, 2020 4:07 PM

$$\int_u f(x)\,dx \approx \sum_q f(x_q)\, w_q$$

(eval pts: x_q; weights: w_q)

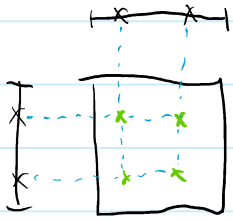
⇒ Gauss most common with n points

⇒ Exact for polynomials of degree $2n-1$ (optimal for fixed n)

in 2D or 3D?

Tensor product quadrature:

$$G_n(f,[0,1]) = \sum_{q=1} f(x_q)\, w_q$$



$$G_n(f,[0,1]^2) = \sum_{p,q} f\left(\begin{bmatrix} x_p \\ x_q \end{bmatrix}\right) w_p\, w_q$$

⇒ for any quadrature, any degree (very difficult on triangles)

What order do I need?

degree? $\rightarrow \int_\Omega \nabla\phi_i \cdot \nabla\phi_j\, dx$

($\nabla\phi_i$: fe degree)

$\nabla\phi_i$ still is at most fe degree (due to J^{-1} transform)

$\nabla\phi_i \cdot \nabla\phi_j$ is 2·fe degree

So $2n-1$ = fe degree $\Rightarrow n =$ fe. degree $+\frac{1}{2}$

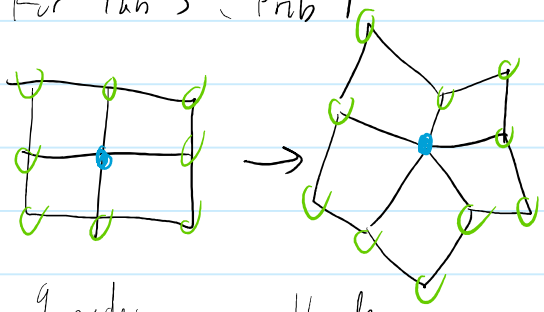
↑ fe degree +1

Ex.) Q2 in 3D

Ex.) Q_2 in 3D

$3 \cdot 3 \cdot 3 = 27$

For lab 3, Prob 4



9 nodes 11 nodes

Prob 6.) $A_{ij} = (\nabla\phi_i, \nabla\phi_j)$ are sparsity pattern symmetric?

$A_{ij} \neq 0$ iff ϕ_i and ϕ_j have support on some cell k.

⇒ For a scalar PDE, the SP is symmetric

5.) for each cell

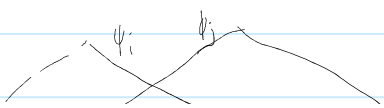
for each i, j

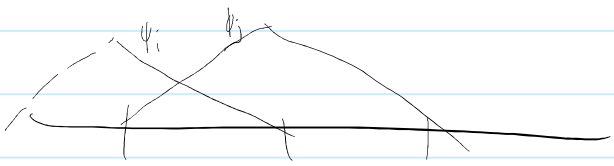
d_s_p.add(global_i, global_j)

SparsityPattern :: iterator it = s_p.begin(42)

gives iterator

```
while (it != s_p.end(42))
 {
   std::cout << it->column() << std::endl;
   it++
 }
```





(ϕ_i, ϕ_j) support is
(ϕ_j, ϕ_i) symmetric

For Lab

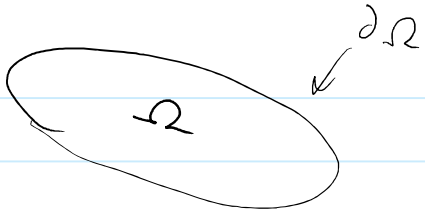
To get Gauss points

Point2D d_ref = fe_values.quadrature_point(q_index)

const

Boundary Conditions

Thursday, September 17, 2020 3:59 PM



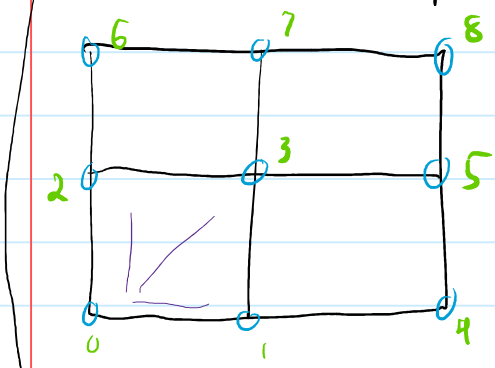
Example: $-\Delta u = f$

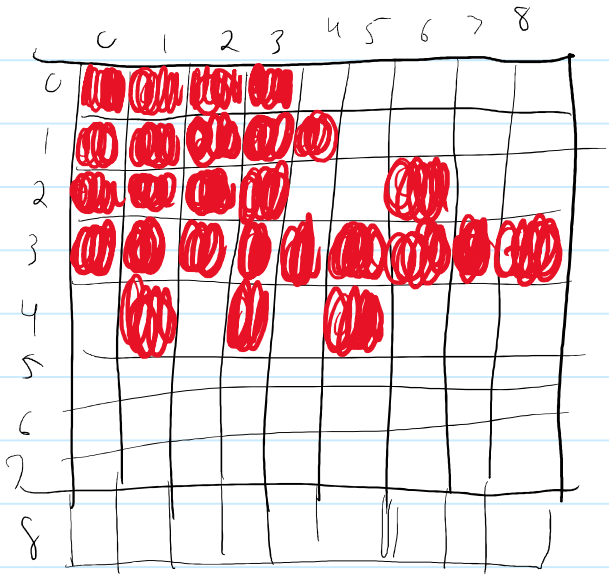
$u|_{\partial\Omega} = 0$

↑ required for unique solutions

$V = H_0^1(\Omega)$

$V_h \subseteq V$, $V_h = \text{span}\{\phi_i\}$





$\rightarrow u = \sum u_i \phi_i$

$\Rightarrow u_0 = 0, u_1 = 0, \ldots$

option A: do not include $\phi_0, \phi_1, \ldots$

$\Rightarrow$ a cell can have fewer shape functions

$\Rightarrow$ Not a good solution

option B: 1.) include ϕ_j on the boundary

2.) Fix the boundary values by enforcing $u_0 = 0, u_1 = 0, \ldots$

3.) Remove these lines from linear system

$\Rightarrow$ not $u = \begin{bmatrix} u_0 \\ \vdots \\ u_8 \end{bmatrix}$, but just for leftover unknowns

not ideal: $\dim V_h \neq \text{size}(u)$

option C: Like B but we store 0 entries

how to enforce $u_j = 0$? $(j \in \partial\Omega)$

⇒ replace row j by $A_{jj} = 1$, $F_j = 0$

⇒ matrix is no longer symmetric even if PDE is symmetric

optional: eliminate column j using row j

- advantage: A SPD

Cost: how many columns? $\sim \sqrt{N}$

cost for one column?

if S.P. is symmetric, only check row

check row j ⇒ $O(1)$ ok ✓

if not: $O(N)$ ☹

⇒ only do this if you require symmetry

But: MatrixTools::apply

Better: local elimination (step-6)

$$\begin{bmatrix} x & x & x & \cdots \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \downarrow & & & & \end{bmatrix} \quad \Big| \; F$$

until now: $u|_{\partial\Omega} = 0$, $H_0^1(\Omega)$

How to extend to $u|_{\partial\Omega} = g$?

(non-homogeneous Dirichlet b.c.)

$u_i = g(x_i)$

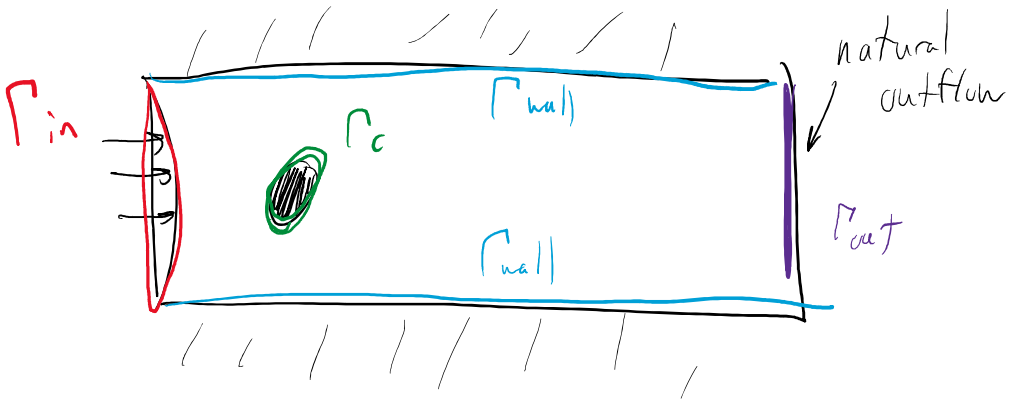
↖ support point of ϕ_i

↳ done in interpolate_boundary_values()

Boundary conditions 2

Tuesday, September 22, 2020 3:35 PM

Example:

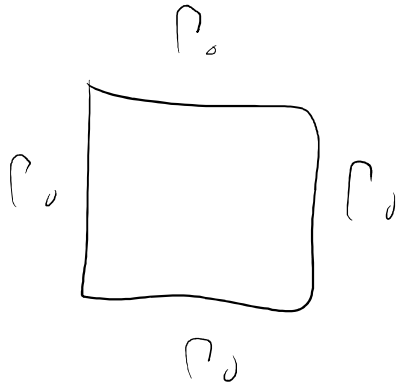


$$\partial\Omega = \Gamma_{in} \cup \Gamma_{wall} \cup \Gamma_{out} \cup \Gamma_c$$

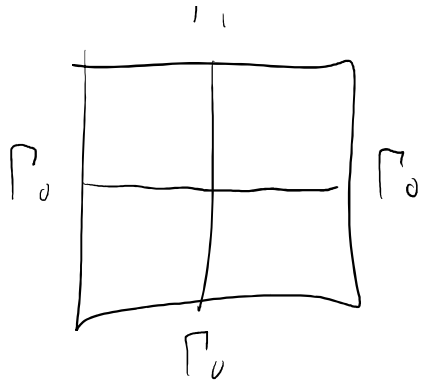
Γ_0 Γ_1 Γ_2 Γ_3

↑ boundary id

default hyper-cube: all faces have id 0



$u|_{\Gamma_1} = 0$



$u|_{\Gamma_0} = 0$

$u|_{\Gamma_1} = ?$

↑ do nothing

b.c. $\nabla u \cdot n = 0$

On Lab 4: Prob 6

The discontinuous point privritizes the boundary that is last defined, in the example the boundary is changed after, so that "gets" the point.

Natural Boundary Conditions

$-\Delta u = f \quad , \quad u|_{\partial\Omega} = g$

$$\int_\Omega v(-\Delta u) = \int_\Omega v f$$

$$\int_\Omega \nabla v \cdot \nabla u - \int_{\partial\Omega} v(\nabla u \cdot n) = \int_\Omega vf$$

$$\| \quad 0 \text{ if } v \in H_0^1$$

$\Rightarrow (\nabla v, \nabla u) = (v, f) \Rightarrow$ step-3

if "do nothing" on Γ_1, then I have $v \neq 0$ on Γ_1 and this integral doesn't vanish.

if I have solution u_h of weak form

$$\Rightarrow \int_\Omega v(-\Delta u) + \int_{\partial\Omega} v(\nabla u \cdot n) = \int_\Omega vf$$

Test w/ $v = 0$ on $\partial\Omega$

$$-\Delta u = f$$

test w/ $v = 1$ on $\partial\Omega$, $v = 0$ interior

$\Rightarrow \nabla u \cdot n = 0$ on $\partial\Omega$

$\Rightarrow$ that is why "do nothing" means $\boxed{\nabla u \cdot n = 0}$

Manufactured solutions

Thursday, September 24, 2020 3:34 PM

Goal: Validation of a FE code

Example: $-\Delta u = f$, $u|_{\partial\Omega} = g$

steps.) 1: pick any function u as your solution
2: Compute RHS f for given u
3: Solve numerically $-\Delta u_h = f$, $u_h|_{\partial\Omega} = u$
4: Compute errors $\|u - u_h\|_1$, $\|u - u_h\|_{L^2}$
$|\text{mean}(u_h) - \text{mean}(u)|$, ...
goal: verify theory

Note: Pick something where you can pick $f = -\Delta u$
- what about $u \in V_h$? bad idea, small error
- Careful w/ polynomials b/c polynomial space Q_n
- Better use trig functions
- Choose difficult enough u s.t. $u - u_h$ is not close to Emach
otherwise
  - Quadrature error
  - Floating point arithmetic
  - linear solve tolerance
- u smooth enough for theory to hold
- step 4 contains approximation error

$$\text{mean}(u_h) = \frac{1}{|\Omega|}\int_\Omega u_h\,dx \approx \sum_k \sum_q u_h(x_q)\,w_q$$

$$\|u - u_h\|_{L^2}^2 = \int_\Omega (u - u_h)^2$$

$$\text{integrate\_difference} \approx \sum_k \sum_q \underbrace{(u(x_q) - u_h(x_q))^2}\, w_q$$

$$\text{integrate\_difference} \approx \sum_{\text{cells}} \sum_{q} \underbrace{(u(x_q) - u_h(x_q))^2}_{\text{exact, but quadrature introduces error}} w_q$$

note: good idea to use higher degree quadrature

Manufactured solutions pt. 2

Tuesday, September 29, 2020 4:18 PM

ex.) $-\Delta u = f$

$(\nabla v, \nabla u) = (v, f)$ weak form

$a(u,v) = (\nabla u, \nabla v)$ bilinear form

$f(v) = (f, v)$ linear form

Find $u \in V : a(u,v) = f(v)$ (*)

Lax Milgram: $\exists$ unique $u \in V$ s.t. (*) holds

Assume: $V_h \subseteq V$, $V_h = Q_k$ or $V_h = P_k$ uniform w/ mesh size h

discrete weak form: $a(u_h, v_h) = f(v_h) \quad \forall v_h \in V_h$

Cea's Lemma: $\|u - u_h\|_{H^1} \leq C \inf_{v_h \in V_h} \|u - v_h\|_1$

(Assume $u \in H(\Omega)^{k+1}$)

$\leq C \|u - \Pi u\|_1$ ← interpolate

$\leq C \cdot h^k |u|_{k+1}$

Note: FE_Q(k) $\Rightarrow \|u - u_h\|_1 \leq C \cdot h^k$

What about $\|u - u_h\|_{L^2}$?

"Nitsche trick", Aubin-Nitsche Lemma

needs regularity of the domain, $u \in H^{k+1}(\Omega)$

$\Rightarrow \|u - u_h\|_{L^2} \leq C \cdot h \cdot |u - u_h|_1$

How to compute $\|u - u_h\|_{L^2}^2$

$= \int_\Omega (u - u_h)(u - u_h) = \sum_T \int_K (u - u_h)(u - u_h)$

integrate-difference $\approx \sum_T \sum_q (u - u_h)(u - u_h) w_q$

error-per-cell

$\Rightarrow \|u - u_h\|_{L^2} = \sqrt{\sum_T \text{error\_per\_cell}(T)}$

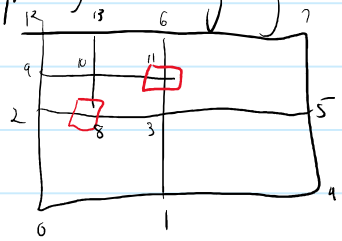
compute_global_error

Why adaptivity?

Thursday, October 1, 2020 3:40 PM

- Error is not distributed equally, especially with non-smooth solution
- Represents features in geometry

Example) hanging nodes



$u_8 = \frac{1}{2}(u_2 + u_3)$

$u_{11} = \frac{1}{2}(u_3 + u_6)$

$\Rightarrow u_h \in H_1(\Omega)$

apply these constraints to

make_hanging_node_constraint

$2u_2 - 10u_8 + 3u_3 = 0$

add a line

constraint_add_line(8)

$u_8 = -\frac{2}{10}u_2 - \frac{3}{10}u_3$

constraint_add_entry(8, 2, $-\frac{1}{5}$)

" (8, 3, $-\frac{3}{10}$)

Need to write the constraint

$a_i = \sum a_{ij} u_j + b$

$j \in [0, N]$, N DoFs

b: Dirichlet B.C.

$j \in [0, N]$, N DoFs

$i \in I$ (constraint)

most $a_{ij} = 0$

- $O(1)$ constrained entries

Logically: Sparse matrix C_M

RHS C_R

$$C_M \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \end{bmatrix} = C_R$$

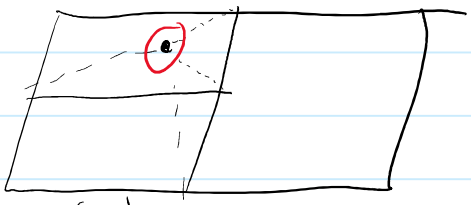
$$V_h = \{ u = \sum u_i \phi_i \;,\; C_M \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \end{bmatrix} = C_R \}$$

Affine Constraints <double>

$$x_i = \sum_j a_{ij} x_j + c_i$$

used for

1.) hanging nodes



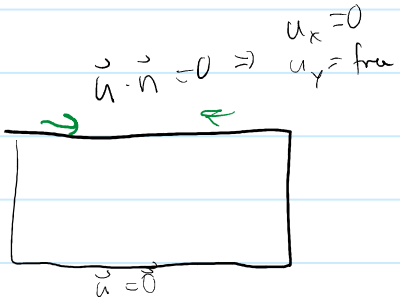
2.) boundary conditions: $u_i = c_i = g(x_i)$

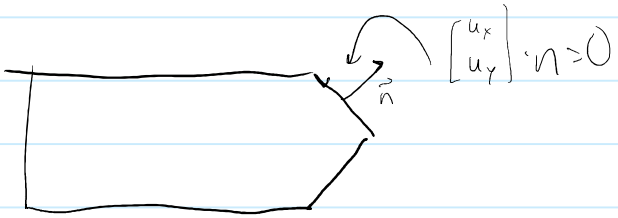
3.) other boundary conditions:

for example Stokes

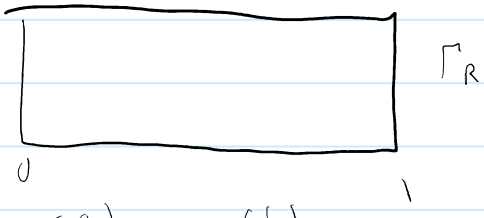
velocity $\vec{u}$

$\vec{u} \cdot \vec{n} = 0$ "free slip"





periodic B.C.:



$$u\begin{pmatrix} 0 \\ y \end{pmatrix} = u\begin{pmatrix} 1 \\ y \end{pmatrix}$$

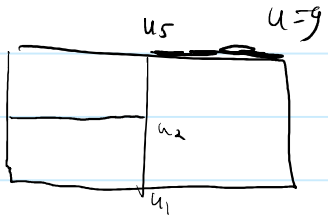
$$u\begin{pmatrix}0\\y\end{pmatrix} = u\begin{pmatrix}1\\y\end{pmatrix}$$

4.) other things:
mean value constraint
for example
$-\Delta u = f$, $\int_\Omega u = 0$
b.c.: $\nabla u \cdot n = 0$

Step-6:

Constraint.close();
simplifies representation
resolves chains

$u_2 = \frac{1}{2}u_1 + \frac{1}{2}u_5$
$u_5 = 42$



$\Rightarrow u_2 = \frac{1}{2}u_1 + 21$

distribute_local_to_global():
copies local matrix/RHS to global one
and applies constraints

distribute(): ?

Chapter 2: parallel computing

Tuesday, October 6, 2020 4:27 PM

example: 1 workstation
2 cpus
16 cores
2 processing units (hyper threading)
⇒ 64 processing units
⇒ concurrently

important: parallel computing doesn't happen automatically!

/proc/cpuinfo

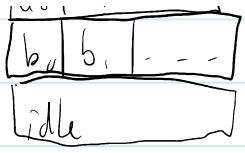
Linux command to see specs on cpu

past: task switching (even on one processing unit), gives illusion of switching

ex.) task a
task b
1 pu: a_0 a_1 b_0 a_2 a_3 ...
→ time
task switching

2 pu:
a_0 a_1 a_2 ...
b_0 b_1 b_2 ...

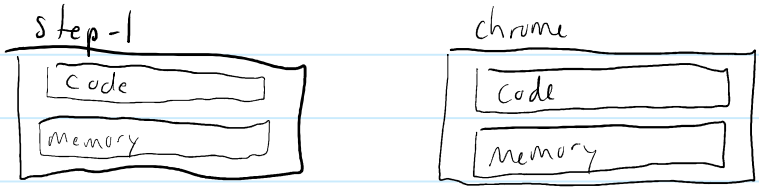
3 pu:
a_0 a_1 ...
b_0 b_1 ...



pu: processing units
→ hyperthreading / SMT
OS thinks you have x2 cores
→ about 0-20% extra performance over 1 task per core

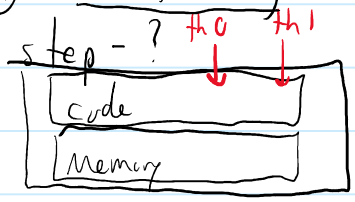
Processes vs Threads

Processes:



- all processes have separate memory
- isolated
- scheduled by OS to run on PU

(A) Multithreading



- create thread (ask OS)
⇒ 2 concurrent threads of execution
- memory is shared (!)
- low overhead
- arbitrary number of threads
- data is accessed concurrently?
  - data races/corruption possible
  - invalid data

- data races/corruption possible
- invalid data

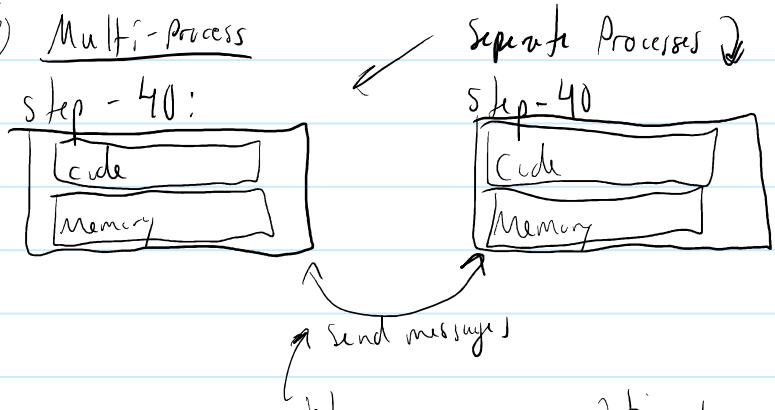
⇒ explicit synchronization needed

03_threads_race:

problem: concurrent reading and writing in one memory location

solution: explicit synchronization

⑧ Multi-Process



inter-process communication by OS

Library: Message Passing Interface (MPI)

↳ same machine (OS function)

or over network (super computer)

⇒ - memory is separate

- explicit communication

Ex.) 8 Cu (laptop, 4 cores x2 HT)

a.) 1 process, 8 threads ./step-7

b.) 8 processes, 1 thread each mpirun -n8

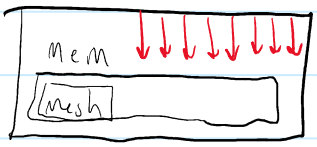
(hybrid parallel)

4 processes, 2 threads each

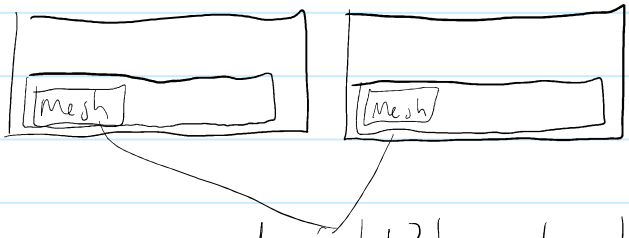
2 processes, 4 threads

⇒ complicated, probably not useful

a.)



b.)



x 8

duplicated?! maybe only store part of the mesh.

Concurrency example:

fixed work, N workers

ideal situation: N times faster than 1 worker

but: - don't split up work evenly
- communication
- limited resource
- sequential part of the code

(fraction)

Let $0 \leq s \leq 1$ sequential part of the work

$p = 1 - s$ parallel part/fraction

$T(1)$ time w/ 1 worker

$T(N) = s \cdot T(1) + p \frac{T(1)}{N}$ Amdahl's Law

Speed up: $\frac{T(1)}{T(N)}$

max? $\xrightarrow{N \to \infty} \frac{1}{s}$

80% parallel $\Rightarrow$ max. speed up is 5

realizy: - There is no $N = \infty$
- Can you split up work evenly for large N? (limits N)
- communication (often takes more time) does not

(limits N)

- communication (often takes more time) does not scale, gets slower with large N

$$T(N) = sT(1) + p\left(\frac{T(1)}{N}\right) + aN + bN^2 + c\log(N)$$

↑ reduction (think binary tree)

Threads in C++:

1.) Passing objects by reference makes (unnecessary?) copies

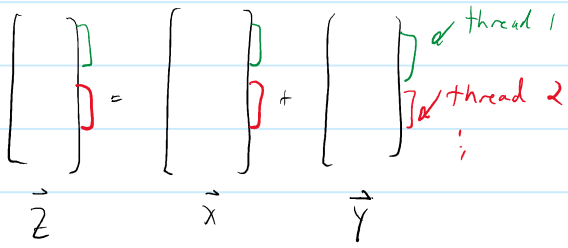
⇒ avoid this by doing

```
std::thread (func, std::ref(x));
```

note: x needs to be valid until thread ends

2.) std::thread::hardware_concurrency()

⇒ int = no. of PU



3.) Synchronization

necessary if a.) 2+ writers
b.) 1 writer, 1 reader
to the same memory address

A.) Locking using mutex

times recorded in test code

int (incorrect)	.04 s
atomic int	.25 s
mutex	1.1 s

---

atomic<T> x

x.compare_exchange_weak(T & expected, T desired)

returns bool

if x currently contains "expected" } atomic
1.) replace x by "desired"
2.) returns true //success

else:
1.) puts current value into expected
2.) return false

Multi threading

create threads, split up work
⇒ concurrent computation

danger: synchronization
⇒ lock-based (mutex, ...)
or lock-free (atomic operations, exchange or retry)

Numerical linear algebra library:
- vectors, sparse matrices
- operations ($x+y$, $A \cdot x$, $\|x\|$, $x \cdot y$, $\alpha \cdot A$)
- Solvers: CG, GMRES, ...
- concurrent implementation of operations

⇒ the user does not need to know that things happen concurrently (includes CG)

Note:
- deal.ii does this (vector<double>::norm())
- only create threads if it's worth doing

- only create threads if it's worth doing
  (overhead to create threads
  vs
  time to run concurrently)

  issue: how do you know how many threads to create?

  what if n things are already running?

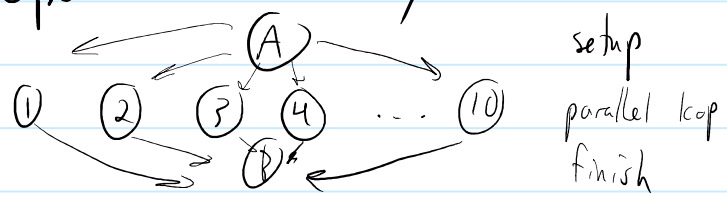
⇒ maybe "tasks" is a better abstraction than threads?

⇒ tasking frameworks
  (create 100-1000s of tasks, let the runtime schedule them)
  - "Intel TBB"
  - "taskflow"

Taskflow

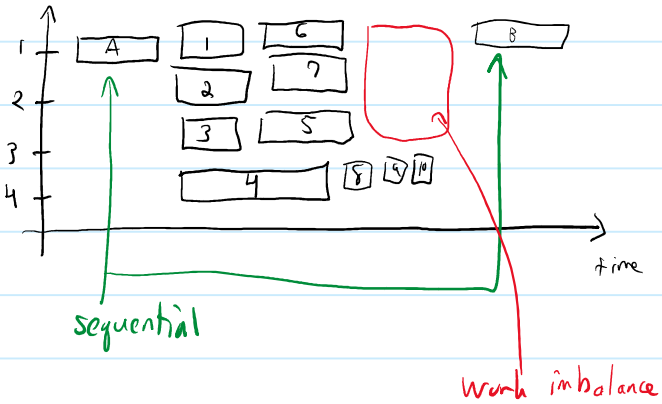
Thursday, October 22, 2020 3:36 PM

Open source library



Dependency A → B means A needs to finish before B starts

Maybe schedule like this:



Tasking:

- Do not worry about # of threads
- Remove sequential steps if no dependency exists
  ↳ better performance!

MPI and linear algebra

Thursday, October 22, 2020 4:01 PM

1.) MPI_Recieve, _send

2.) MPI_Barrier(..) - wait until everybody gets to this line

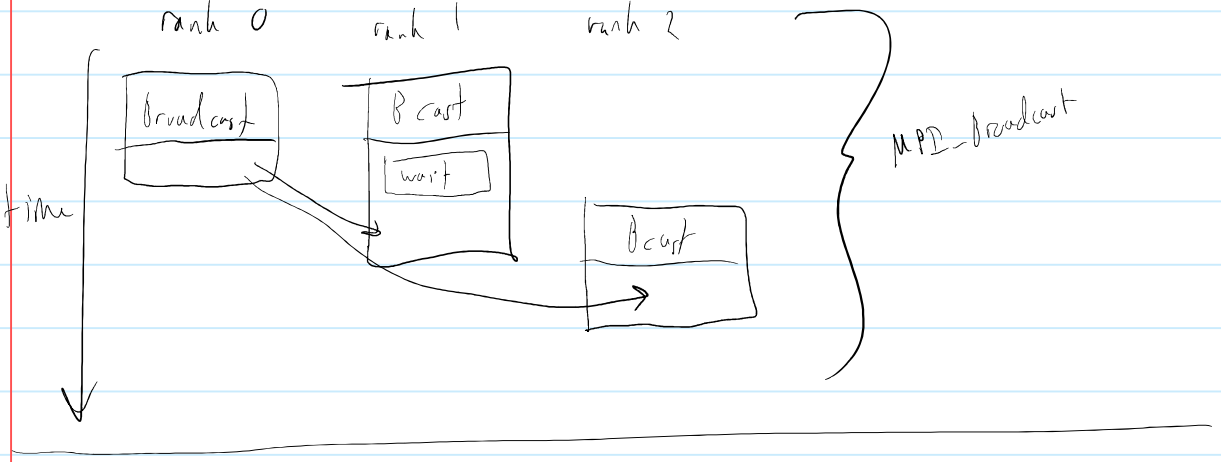
(rarely necessary)

use to sync. text output

3.) MPI_Probe()

- checks for a waiting message

4.) Collective operations (>2 participants)



Linear Algebra

ingredients: - Sparse Matrix, Vector

- operations: Ax, x+y, x·y, ||x||, αx
- algorithms: GMRES, CG

Multithreading: Break up work and run on different threads

MPI: Split up the data!

⇒ afterwards: aka split the work

why? - faster

- larger problem

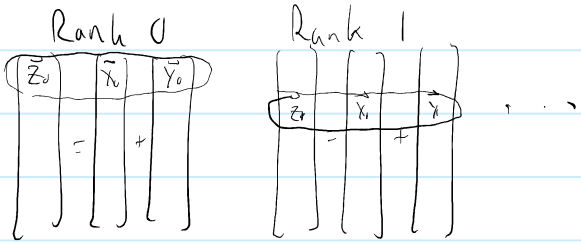
⇒ afterwards: aka split the work — larger problem

Vector $\vec{x}$ = #0 / #1 / #2 / #3 $\begin{bmatrix} x_0 \\ \vdots \\ x_{N-1} \end{bmatrix} \in \mathbb{R}^N$

each rank stores a subset of all entries

each entry has a single owner

$\vec{z} = \vec{x} + \vec{y}$



⇒ no communication

Assume rank m owns $I_m \subseteq [0, N-1)$ — Index set

for example $[0, N-1) = \underbrace{I_0}_{0-10} \cup \underbrace{I_1}_{11-21} \cup I_2 \cup I_3 \cdots$

$\|x\| = \sqrt{\sum_i x_i^2}$

1.) local part = $\sum_{i \in I_m} x_i^2$

2.) MPI_Reduce (... sum ...)

3.) sqrt(value) and return

$\vec{x} \cdot \vec{y}$ works the same

$\vec{y} = A\vec{x}$

$$\begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix}\begin{bmatrix} \vec{y}_0 \\ \vec{y}_1 \\ \vec{y}_2 \\ \vec{y}_3 \end{bmatrix} = \begin{bmatrix} A_{00} & A_{10} & A_{20} & 0 \\ 0 & A_{11} & 0 & 0 \\ 0 & 0 & A_{22} & 0 \\ 0 & 0 & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} \vec{x}_0 \\ \vec{x}_1 \\ \vec{x}_2 \\ \vec{x}_3 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \#0 \end{matrix}$$

$\vec{y}$ $\quad$ A $\quad$ $\vec{x}$

⇒ store matrix blocked by row in the

same way

$$\vec{y}_3 = \underbrace{A_{32}}_{\text{local}} \vec{x}_2 + \underbrace{A_{33} \cdot \vec{x}_3}_{\text{local}}$$

Import $\vec{x}_2$ from rank 2

⇒ do not transfer matrix entries, instead, import entries from $\vec{x}$ as needed

do not import all of $\vec{x}$ (too expensive)

as a pre-process (just with sparsity pattern)

compute $\{j \mid A_{ij} \neq 0, i \in I_m\}$

indices I need to import

1.) Find owner m of each index j ($j \in I_{m_j}$)

2.) For each rank listed in 1.), send them the list of indices I will need

3.) Recieve list of indices from certain ranks

Implementation of $A\vec{x}$

1.) Send out parts of $\vec{x}$ to others (based on setup above)

optimization: MPI_Isend()

2.) Compute local part:

$\vec{y}_m = A_{m,m} \cdot \vec{x}_m$

3.) For each message from rank s:

grab $\vec{x}_j$, compute

$\vec{y}_m \mathrel{+}= A_{s,m} \vec{x}_s$

4.) wait for all sends to complete

Note: send early, recieve as late as possible

(do as much computation as possible)

⇒ hides the latency (time to send messages)

⇒ better parallel scaling!

⇒ better parallel scaling!
in contrast: $\vec{x} \cdot \vec{y}$ has to wait.

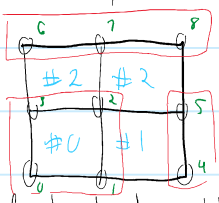
Preconditioner:
This is difficult & problem dependent.
Only multigrid (potentially) gives you $O(N)$ time

Given a preconditioner, solving $Ax=b$ in $O(\frac{N}{p})$ time (N = # of DoFs, p = # of ranks) is possible

With these operations ($Ax, xy, \ldots$) GMRES/CG is equivalent to a sequential algorithm

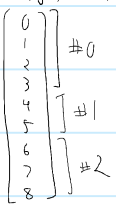
---

missing: how to compute A & b?
⇒ parallel F.E. assembly
⇒ rank n does not know the whole mesh
⇒ can only compute part of A.



9 DoFs (Q1)

1.) distribute the mesh (1 owner per cell)



2.) assign DoFs to ranks:
if internal to a rank, assign to rank
if on interface: decide somehow
(deal.II assign to lower rank)

(deal.II assign to lower rank)

3.) Sort/number the unknowns so that they are consecutive by rank.

4.) assemble <u>own</u> cells

$A_{i,j}$ where $i \notin \mathbb{I}_m$ (non-local entries)

$\Rightarrow$ send this to owner of i

(matrix.compress())

5.) <u>done</u>

Running on palmetto

Thursday, November 5, 2020 3:46 PM

Timo has a page on how to do this

- Don't compute on the login node
- login node might be different than compute nodes
  I know this already
- Pay attention to large I/O
  ⇒ 1.) be careful (not wasteful!)
  2.) special file system for large files
  (/scratch1/USERNAME)

Vector-values problems

Thursday, November 5, 2020 4:20 PM

step 8: elasticity

20: mixed Laplace

22: Stokes

instead of $u: \Omega \to \mathbb{R}$

we solve for $u: \Omega \to \mathbb{R}^n$

Stokes equation

Find $\vec{u}, p$ s.t. $-\Delta \vec{u} + \nabla p = \vec{f}$

$\nabla \cdot \vec{u} = 0$

$\vec{u}(x) \in \mathbb{R}^d$, $\vec{p}(x) \in \mathbb{R}$

$\Delta = \nabla \cdot \nabla$

$\vec{\Delta} = \begin{bmatrix} \Delta \\ \Delta \end{bmatrix}$ $(d=2)$

$$-\int \vec{v} (\Delta \vec{u}) + \int \vec{v} (\nabla p) = \int \vec{v} \vec{f} \quad \forall v \in V$$

$$\int q \, \nabla \cdot u = 0 \quad \forall q \in Q$$

$$\Rightarrow (\nabla \vec{v}, \nabla \vec{u}) - (\nabla \cdot \vec{v}, p) = (\vec{v}, \vec{f}) \quad \forall v \in V$$

$$(q, \nabla \cdot u) = 0$$

How do I get $Ax = b$?

$\Rightarrow$ number all unknowns as $\{\vec{\phi}_i\}$

$\vec{\phi}_i$ is a vector-valued basis function

$U = \begin{bmatrix} u_x \\ u_y \\ p \end{bmatrix} \in X = \{f: \Omega \to \mathbb{R}^{d+1}\}$

$$\int_\Omega \Phi^T \left[\begin{array}{c|c} -\vec{\Delta} & \nabla \\ \hline \nabla \cdot & 0 \end{array}\right] \begin{bmatrix} u_x \\ u_y \\ p \end{bmatrix} = \int_\Omega \Phi^T \cdot \vec{f}$$

$$\begin{bmatrix} \phi_x^u \\ \vdots \end{bmatrix} \quad \begin{bmatrix} \vec{\phi}^u \end{bmatrix} \quad \checkmark$$

Ω

$$\forall \underline{\Phi} = \begin{bmatrix} \phi_x^u \\ \phi_y^u \\ \phi_p \end{bmatrix} = \begin{bmatrix} \vec{\phi}^u \\ \phi_p \end{bmatrix} \in X$$

$$\left(\vec{\phi}^u, -\Delta\vec{u} + \nabla p\right)$$
$$+\left(\phi^p, \nabla\cdot\vec{u}\right) = \left(\vec{\phi}^u, \vec{f}\right)$$

$$\Rightarrow \left(\nabla\vec{\phi}^u, \nabla\vec{u}\right) - \left(\nabla\cdot\vec{\phi}^u, p\right) + \left(\vec{\phi}^p, \nabla\cdot\vec{u}\right) = \left(\vec{\phi}^u, \vec{f}\right)$$

1 equation

$$X_h = \text{span}\left\{\underline{\Phi}_i : \Omega \to \mathbb{R}^{d+1}\right\}$$

$$\sum \underbrace{\left[\left(\nabla\phi_i^u, \nabla\phi_j^u\right) + \left(\nabla\cdot\phi_i^u, \phi_j^p\right) + \left(\phi_i^p, \nabla\cdot\phi_j^u\right)\right]}_{=A_{ij}} \cdot u_j = \left(\phi_i^u, f\right)$$

$$\Rightarrow A\,U = F$$

Specify X_h?

Specify ϕ_i^u

Continuous space X: $X = V \times Q$

LBB stability / inf-sup condition

$\Rightarrow$ V (velocity) needs to be "big enough" compared to Q

$$\inf_{q\in Q}\sup_{v\in V}\frac{(\nabla\cdot v, q)}{\|v\|_1\|q\|_0} \geq c > 0$$

$V = H_0^1(\Omega)$, $Q = L^2(\Omega)$ ✓

discrete: more work (not always true!)

$V_h = Q_{k+1}^d$, $Q_h = Q_k$

(Taylor-Hood elements)

There are other options

$X_h = \text{span}\{\underline{\Phi}_i\} = V_h \times Q_h$

$X_h = \text{span}\{\underline{\Phi}_i\} = V_h \times Q_h$

$= Q_{k+1}^d \times Q_k$

(in 2D) $= Q_{k+1} \times Q_{k+1} \times Q_k$

$Q_2 = \text{span}\{1, x, y, x^2, y^2, xy, \ldots, x^2y^2\}$

on reference cell

$Q_2 : \Omega \to \mathbb{R}$

on ref. cell:

$$X_h = \text{span}\left\{ \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}x\\0\\0\end{bmatrix}, \begin{bmatrix}y\\0\\0\end{bmatrix}, \ldots, \begin{bmatrix}x^2y^2\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\x\\0\end{bmatrix}, \ldots, \begin{bmatrix}0\\x^2y^2\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\0\\x\end{bmatrix}, \ldots, \begin{bmatrix}0\\0\\x^2y^2\end{bmatrix} \right\}$$

$\Rightarrow$ extend shape functions to have $d+1$ components (all but one are nonzero)

primitive FE space

$X_h = \underset{\text{vel.}}{Q_{k+1}^d} \times \underset{\text{pres.}}{Q_k}$: FESystem(FE_Q<dim>(k+1),

$d+1$ components

dim, // dim copies

FE_Q<dim>(k)

1) ↑ dim of ref cell ↑ degree

FESystem can be an arbitrary combination of finite Elements, including FESystem.

$\Rightarrow$ FE w/ m components

access to components / shape functions?

FE Values :: shape_grad(i,q) $\nabla\phi_i(x_q)$

shape_value(i,q) $\phi_i(x_q)$

↑ return the gradient / value for the only nonzero component

$\Rightarrow$ not useful

instead shape_grad_component(i,q,m)

$\nabla(\underline{\Phi}_i)_m(x_q) \in \mathbb{R}^d$

shape_value_component(i,q,m)

$\nabla (\underline{\Phi}_i)_m (x_q) \in \mathbb{R}^d$

shape_value_component (i, q, m)

$(\underline{\Phi}_i)_m (x_q) \in \mathbb{R}$

---

even better: "extractors"

stokes 2D

div ϕ_i^u $\rightarrow$ shape_grad_component $(i, q, 0)$ [0] + " " $(i, q, 1)$ [1]

$\frac{\partial}{\partial x} \phi_i^{ux} + \frac{\partial}{\partial y} \phi_i^{uy}$ $\Rightarrow$ annoying, dim-dep

div only makes sense if you have d components forming a vector in d dimensions

$\underline{\Phi} = \begin{bmatrix} \vec{\phi}^u \\ \vec{\phi}^p \end{bmatrix}$

FEValues [vector extractor] :: divergence $(i, q) \in \mathbb{R}$

:: gradient $(i, q) \in \mathbb{R}^{d \times d}$

:: value $(i, q) \in \mathbb{R}^d$

FEValues [scalar extractor] :: value $(i, q) \in \mathbb{R}$

:: gradient $(i, q) \in \mathbb{R}^d$

→ step-22

Note: For vector valued problems:

use extractors

or shape_value/grad if you

use component_index

Timo talked about dof numbering for the vector u and p. Can change that with things, DoFRenumbering

Multigrid methods and step 22

Tuesday, November 10, 2020 3:46 PM

Goal: solve FEM problem in $O(\frac{N}{p})$ time

N = # DoFs

p = processors

All ingredients need to be $O(\frac{N}{p})$

- numbering of DoFs ✓
- sparsity patterns ✓
- assembly ✓
- mat-vecs ✓
- it. solvers (✓)
- preconditioners !
  - ↳ multigrid

$-2\,\mathrm{div}\,\varepsilon(u) + \nabla p = f$ symmetrized formulation

is equivalent to

$-\Delta u + \nabla p = f$

if:

- viscosity is constant (here $\equiv 1$)
- regularity $u \in [C^2]^d$
- $\mathrm{div}\, u = 0$

⇒ is this a good assumption in discrete setting?

Block systems and numbering

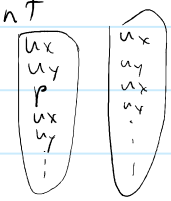
Tuesday, November 17, 2020 3:35 PM

By default, distribute_dofs() distributes the unknowns by:

1.) cell by cell

2.) through objects: vertices, faces, cells

3.) Component

ex.) 

either is ok, sometimes we want to control this however

DoFRenumbering: permutation $[0, N) \to [0, N)$

component-wise:

for each component c, specify an index or block

block_component[c]

The reordering first enumerates all DoFs with block_component[c] = 0, then 1, then 2, ---

$$\begin{bmatrix} u_x \\ u_y \\ p \end{bmatrix} \to \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$ velocity first, then all pressures

$$\begin{bmatrix} u_x \\ u_y \\ p \end{bmatrix} \to \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$ x vel, y vel, pressure

Note: Assembly does not need to know/change based on numbering

⇒ only for solves

⇒ only for solves

Note: Combine ordering:

1.) Cuthill-McKee } step-22

2.) Component-wise

Many other options

For Block Linear Systems

arbitrary grouping into blocks

⇒ independent of components

⇒ nxn array of SparseMatrix/Sparsity Pattern

⇒ Block

Debug and release mode

Tuesday, November 17, 2020 4:24 PM

Switch using
"make debug"
"make release"

debug mode

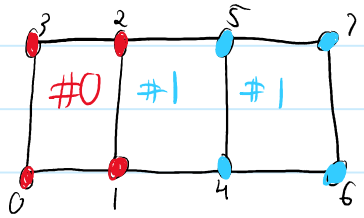
⊕ can debug using gdb and other tools
- turns off all optimizations (⇒ slow)
  good for debugging
- deal.II has many consistency checks in the library
  to help you find bugs
  "defensive programming"
  ⇒ only in debug mode

⇒ release mode 2-10x faster, not helpful
(use only for large computations)

Distributed/ghosted vectors

Thursday, November 19, 2020 3:50 PM

Index Sets: owned, active, relevant



	owned	active	relevant
#0:	0,1,2,3	0,1,2,3	0-5
#1:	4,5,6,7	1,3 | 4,5,6,7	0-7

#1: 0 1 2 3 4 5 6 7 [?]

#0 [?]

"distributed vectors"

reinit(owned_set, communicator)

step 40

"ghosted vectors", active or relevant

(or anything else)

#1 0 1 2 3 4 5 6 7 [? ?] ghost indices; [ ] owned

reinit(owned, ghost, communicator)

assignment:

solution = distributed_solution;

ghosted ← imports values ← distr.

ghosted vectors: dangerous to allow modifications

(out of date information)

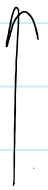
⇒ ghosted vectors are read-only

Create them via assignment from distr. vector

distributed vectors	ghosted vectors
- assembly	- output (at least "active")
- solvers	- read values during assembly?
- modified	old timestep, nonlinear iteration,...
(constraints,...)	- Computing errors (at least

(constraints, ...)

- Computing errors (at least active for UG: relevant)
- error estimates (at least relevant)



Viewpoint: "linear algebra"

Viewpoint: "PDE"

Time dependent problems

Thursday, November 19, 2020 4:16 PM

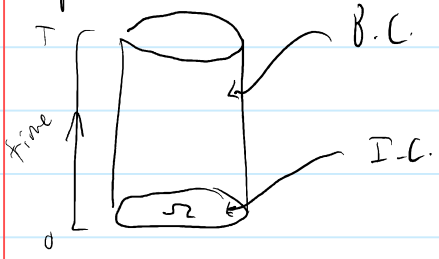
ex.) heat equation: $\frac{d}{dt}u - \nabla \cdot \kappa \nabla u = f$, $\forall x \in \Omega$, $\forall t \in (0,T)$

$u: \Omega \times [0,T] \to \mathbb{R}$

$u(x,0) = u_0(x)$, $\forall x \in \Omega$

$u(x,t) = u_\beta$, $\forall x \in \partial\Omega$, $t \in (0,T)$

Space-time cylinder



ex.) Navier-Stokes Equations

$\frac{d}{dt}u - \nu \Delta u + u \cdot \nabla u + \nabla p = f$ + I.C., B.C.

$\nabla \cdot u = 0$

$u: \Omega \times [0,T] \to \mathbb{R}^d$

$p: \Omega \times [0,T] \to \mathbb{R}$

ex.) Wave Equation

$\frac{d^2}{dt^2}u - \Delta u = f$ (step-23)

need u & $\frac{du}{dt}$ at $t=0$ for I.C.

Spaces? $u(t): \Omega \to \mathbb{R}$ → appropriate Sobolev spaces

regularity in time?

$C^1([0,T];V)$, $\|\cdot\| = \sup_t \left(\|u(t)\|_V + \left\| \frac{d}{dt}u(t) \right\|_V \right)$

$L^p([0,T];V)$, $\|\cdot\| = \left(\int_0^T \|u(t)\|_V^p \right)^{1/p}$

Theory:

- existence & uniqueness is difficult and problem dependent
- parabolic problems: easy like Lax-Milgram (heat eqn)

$$(u_t, v) + a(t; u, v) = (f(t), v)$$

a bilinear form + bounded + coercive

$\Rightarrow \exists!$ solution

Computing?

a.) Method of Lines (discretize space then time)

1.) discretize in space

(weak form $\rightarrow$ Galerkin $\rightarrow Ax = b$)

but your $u_h = \sum \phi_i u_i$ still time-dependent

$$u_h(t) = \sum \phi_i u_i(t)$$ ← coefficients are functions of time

$$\frac{d}{dt}\begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} + A \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} = b$$

$\Rightarrow$ system of ODEs

$\Rightarrow$ ODE solver

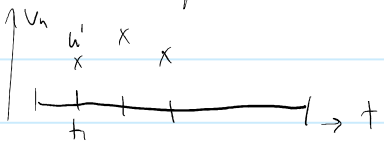
problems

- large DoFs difficult
- cannot change the mesh
- difficult if A (PDE) depends on t

b.) Rothe's method (disc. time, then space)

$0 = t_0 < t_1 < \dots < t_N = T$

$u(t_n) \approx u_h^n \in V_h$



$\frac{d}{dt} u = f(t, u)$

explicit Euler: $\frac{d}{dt} u = \frac{u^{n+1} - u^n}{\Delta t}$

explicit Euler: $\frac{d}{dt}u = \frac{u^{n+1}-u^n}{\Delta t}$

$$\frac{u^{n+1}-u^n}{\Delta t} = f(t_n, u^n)$$

implicit Euler (backward)

$$\frac{u^{n+1}-u^n}{\Delta t} = f(t_{n+1}, u^{n+1})$$

BDF (implicit)

RK (either)

:

when to use explicit vs implicit?

stability (conditional)

explicit methods are (most of the time) only conditionally stable $\Rightarrow \Delta t < X(h, PDE, \dots)$

generally: implicit methods are unconditionally stable

explicit methods:

$$\frac{1}{\Delta t}u^{n+1} = \underbrace{f(t^n, u^n) + f(\dots) + c_1 u^n + c_2 u^{n-1} \dots}_{\text{known}}$$

$\Rightarrow$ no PDE solve necessary!

often: weak form

$$\frac{1}{\Delta t}(v, u^{n+1}) = (v, f(t^n, u^n)) + \dots$$

↑ mass matrix $\Rightarrow$ invert/solve for mass matrix

BDF2: 2nd order, implicit (popular, default choice)

$$\frac{3}{2\Delta t}\left(u^{n+1} - 2u^n + \frac{1}{2}u^{n-1}\right) = f(t^{n+1}, u^{n+1})$$

note: initial solve requires 2 old solutions!

how to get u^1?

start up method (use implicit Euler for u^1)

startup method (use implicit Euler for u')

$\Rightarrow$ could destroy convergence

$\Rightarrow$ stationary PDE solve for u^{n+1}

example: heat equation + BDF2

$$\frac{3}{2\Delta t} u^{n+1} - \nabla \cdot \kappa \nabla u^{n+1} = f + \frac{2}{\Delta t} u^n - \frac{1}{2\Delta t} u^{n-1}$$

$\Rightarrow$ weak form $\Rightarrow$ assemble & solve for u^{n+1}

Nonlinear, time-dependent problems

Tuesday, December 1, 2020 4:19 PM

NSE:

$$\frac{d}{dt}u + \nu \Delta u + u \cdot \nabla u + \nabla p = f$$
$$\nabla \cdot u = 0$$

a) explicit in time

⇒ at most a single mass matrix solve

b.) implicit in time:

$$c\,u^{n+1} - \nu \Delta u^{n+1} + \underbrace{u^{n+1} \cdot \nabla u^{n+1}}_{\text{nonlinear}} + \nabla p^{n+1} = f$$
$$\nabla \cdot u^{n+1} = 0$$

options:

(B1) $u^{n+1} \cdot \nabla u^{n+1} \approx u^n \cdot \nabla u^n$ explicit nonlinear term

⇒ PDE does not change over time

⇒ reuse matrix/preconditioner

stability? no longer unconditional?

better: extrapolation or explicit method to approximate $u \cdot \nabla u$.

IMEX methods

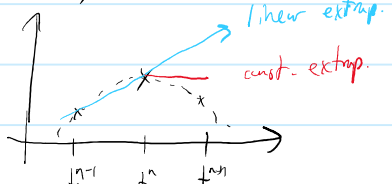
(B2) $u^{n+1} \cdot \nabla u^{n+1} \approx u^* \cdot \nabla u^{n+1}$ (linearized, implicit)

u^* can be u^n ⇒ error $\sim O(\Delta t)$

or extrapolation:

$u^* = 2u^n - u^{n-1}$

(to be paired with 2nd order method)



$$u^n + \Delta t \frac{u^n - u^{n-1}}{\Delta t} = 2u^n - u^{n-1}$$

(B3) keep $u^{n+1} \cdot \nabla u^{n+1}$

⇒ Picard/Newton iteration @ each step

how expensive?

Boundary Conditions

Thursday, December 3, 2020 3:58 PM

- Boundary conditions are required for us to think about
- Required for unique solvability, $-\Delta u = f$ is only solvable up to a constant.

Dirichlet, $u|_{\partial\Omega} = g$ → constraints

Neumann $\nabla u \cdot n = g$

→ 1.) integration by parts

2.) replace $\nabla u \cdot n$ by g

3.) move $\int_\Gamma v \cdot g \, ds$ to RHS (boundary integral)

if $g \neq 0$: assembly (g) FEFaceValues

$g = 0$: $\int_\Gamma v \cdot g = 0$ → "do nothing"

Robin: $c_1 u + c_2 \nabla u \cdot \hat{n} = g$

$\Rightarrow \nabla u \cdot n = \frac{1}{c_2}(g - c_1 u)$ → assemble

no-normal flux: $\vec{u} \cdot n = 0$

"free slip"

compute_no_normal_flux_constraints

no-tangential flow: $\vec{u} \times n = \vec{0}$

→ can be combined

Linear Solvers

Thursday, December 3, 2020 4:15 PM

- small problems / starting out / not parallel:
  ⇒ use direct solver, solverDirectUMFPack step-29
  (Matlab backslash)

why?
- hard to beat (for small/medium problems)
- no tolerance to pick (no accuracy problems)
- no convergence issues
- not invertible? → solver will fail

limits:
- 3d (even medium sized)
- 100k + DoFs
- parallelization beyond a single workstation

- Alternative: Krylov solvers
  ↳ difficult topic, often requires PDE knowledge

- Basic guideline:
  SPD → CG
  S → MINRES
  else → GMRES

- Preconditioning: want
  - 10-50 iterations
  - h independent (hard)
  - parameter indep. (hard)

O(n)

- approach
  1.) decompose PDE on blocks (≈ components)

1.) decompose PDE on blocks (≈ components)

2.) block factorization

3.) for each block: by type of PDE term

Laplace → AMG / GMG

mass → ILU