Fast and smooth simulation of space-time problems

Day 1

Department of Computer Science
AGH University of Science and Technology, Kraków, Poland
home.agh.edu.pl/paszynsk

Department of Computer Science
AGH University, Kraków, Poland

Outline

- **Isogeometric finite element method**
- **Alternating Directions Implicit (ADI) method**
- **Isogeometric L2 projections**
- **Explicit dynamics**
- **Example 1: Heat transfer**
- Installation of IGA-ADS solver
- Parallel distributed memory explicit dynamics
- Parallel shared memory explicit dynamics
- Example 2: Non-linear flow in heterogenous media
- Implicit dynamics
- Example 3: Implicit heat transfer
- Example 4: Linear elasticity
- Example 5: Pollution problem
- Labs with implict dynamics

Software

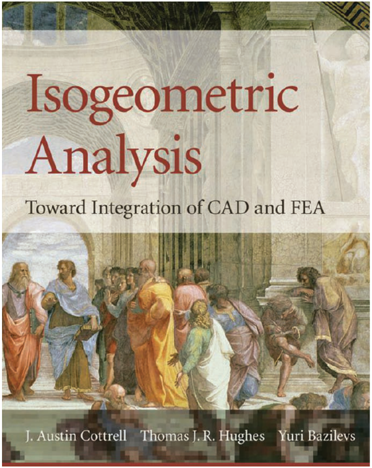
Program Title: IGA-ADS
Code: `git clone https://github.com/marcinlos/iga-ads`
Licensing provisions: MIT license (MIT)
Programming language: C++
Nature of problem: Solving non-stationary problems in 1D, 2D and 3D
Solution method: Alternating direction solver with isogeometric finite element method

If you use this software in your work, please cite

Marcin Łoś, Maciej Woźniak, Maciej Paszyński, Andrew Lenharth, Keshav Pingali *IGA-ADS : Isogeometric Analysis FEM using ADS solver*, **Computer & Physics Communications** 217 (2017) 99-116 (available on researchgate.org)

Isogeometric finite element method

J.A. Cottrel, T.J.R. Hughes, Y. Bazilevs, *Isogeometric Analysis. Toward Integration of CAD and FEA*, Wiley, (2009).



Original recursive definition of B-spline basis functions

- $$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \le \xi < \xi_{i+1}, \\ 0 & \text{otherwise} \end{cases}$$
- $$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi)$$

Figure: Recursive formulae for B-spline basis functions

How to remember this formulae graphically

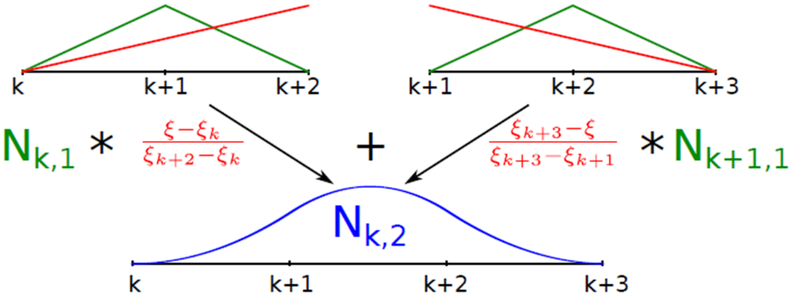


Figure: Practical implementation of the recursive formulae for B-spline basis functions

How these B-spline basis functions look like

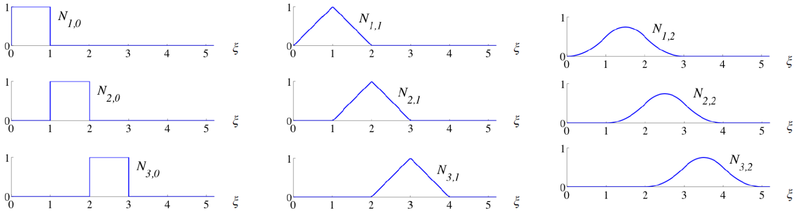


Figure: Basis functions of order 0,1,2 for uniform knot vector $\{0,1,2,3,4,5\}$

Representation of B-splines by knot vectors

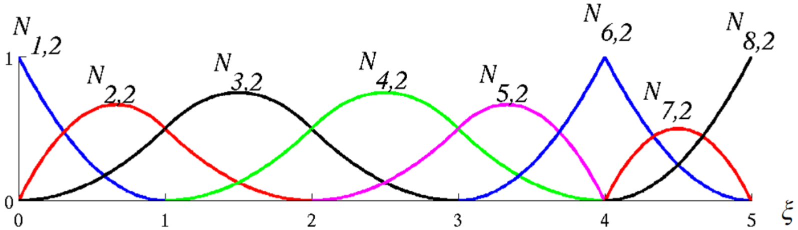
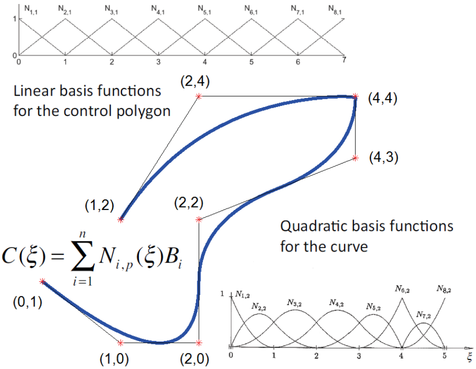


Figure: B-spline basis functions represented by knot vector $\{0,0,0,1,2,3,4,4,5,5,5\}$

Isogeometric finite element method



Quadratic B-spline basis functions represented by knot vector $\{0,0,0,1,2,3,4,4,5,5,5\}$

B-spline curve:

$N_{1,2}$*(0,1)+$N_{2,2}$*(1,0)+$N_{3,2}$*(2,0)+$N_{4,2}$*(2,2)+$N_{5,2}$*(4,3)+ $N_{6,2}$*(4,4)+$N_{7,2}$*(2,4)+$N_{8,2}$*(1,2)

The Alternating Direction Implicit (ADI) method

G. Birkhoff, R.S. Varga, D. Young, *Alternating direction implicit methods*, **Advanced Computing** (1962)

$$\frac{du}{dt} - L_x u - L_y u = f$$

$$\frac{du}{dt} - \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} - \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2} = f$$

$$\frac{u_{i,j}^{t+0.5} - u_{i,j}^{t}}{dt} - \frac{u_{i-1,j}^{t+0.5} - 2u_{i,j}^{t+0.5} + u_{i+1,j}^{t+0.5}}{h^2} = \frac{u_{i,j-1}^{t} - 2u_{i,j}^{t} + u_{i,j+1}^{t}}{h^2} + f_{i,j}^{t}$$

$$\frac{u_{i,j}^{t+1} - u_{i,j}^{t+0.5}}{dt} - \frac{u_{i,j-1}^{t+1} - 2u_{i,j}^{t+1} + u_{i,j+1}^{t+1}}{h^2} = \frac{u_{i-1,j}^{t+0.5} - 2u_{i,j}^{t+0.5} + u_{i+1,j}^{t+0.5}}{h^2} + f_{i,j}^{t+0.5}$$

The Alternating Direction Implicit (ADI) method
G. Birkhoff, R.S. Varga, D. Young, *Alternating direction implicit methods*, **Advanced Computing** (1962)

$$u_{i-1,j}^{t+0.5}[-\frac{2dt}{h^2}] + u_{i,j}^{t+0.5}[1+\frac{2dt}{h^2}] + u_{i+1,j}^{t+0.5}[-\frac{2dt}{h^2}] = dt\frac{u_{i,j-1}^{t} - 2u_{i,j}^{t} + u_{i,j+1}^{t}}{h^2} + dtf_{i,j}^{t}$$

for $i = 1, ..., N_x$, $j = 1, ..., N_y$.

$$u_{i,j-1}^{t}[-\frac{2dt}{h^2}] + u_{i,j}^{t}[1+\frac{2dt}{h^2}] + u_{i,j+1}^{t}[-\frac{2dt}{h^2}] = dt\frac{u_{i-1,j}^{t+0.5} - 2u_{i,j}^{t+0.5} + u_{i-1,j}^{t+0.5}}{h^2} + dtf_{i,j}^{t+0.5}$$

for $j = 1, ..., N_y$, $i = 1, ..., N_x$.

The Alternating Direction Implicit (ADI) method
G. Birkhoff, R.S. Varga, D. Young, *Alternating direction implicit methods*, **Advanced Computing** (1962)

$$u_{i-1,j}^{t+0.5}[-2dt] + u_{i,j}^{t+0.5}[h^2 + 2dt] + u_{i+1,j}^{t+0.5}[-2dt] = dtu_{i,j-1}^{t} - 2u_{i,j}^{t} + u_{i,j+1}^{t} + h^2 dtf_{i,j}^{t}$$

for $i = 1, ..., N_x$, $j = 1, ..., N_y$.

$$u_{i,j-1}^{t}[-2dt] + u_{i,j}^{t}[h^2 + 2dt] + u_{i,j+1}^{t}[-2dt] = u_{i-1,j}^{t+0.5} - 2u_{i,j}^{t+0.5} + u_{i-1,j}^{t+0.5} + h^2 dtf_{i,j}^{t+0.5}$$

for $j = 1, ..., N_y$, $i = 1, ..., N_x$.

The Alternating Direction Implicit (ADI) method

G. Birkhoff, R.S. Varga, D. Young, *Alternating direction implicit methods*, **Advanced Computing** (1962)

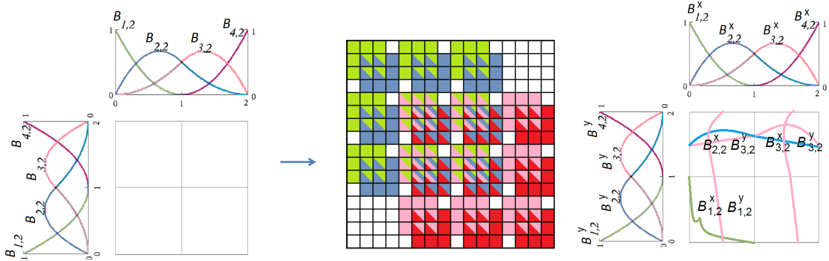
$$
\begin{bmatrix}
h^2+2dt & -2dt & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
-2dt & h^2+2dt & -2dt & 0 & \cdots & \cdots & \cdots & 0 \\
0 & -2dt & h^2+2dt & -2dt & 0 & \cdots & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\
0 & \cdots & \cdots & 0 & -2dt & h^2+2dt & -2dt & \\
0 & \cdots & \cdots & \cdots & 0 & -2dt & h^2+2dt &
\end{bmatrix}
\begin{bmatrix}
u_{1,1}^{t+0.5} \\ u_{1,2}^{t+0.5} \\ u_{1,3}^{t+0.5} \\ \vdots \\ u_{N_x,N_y-1}^{t+0.5} \\ u_{N_x,N_y}^{t+0.5}
\end{bmatrix}
$$

$$=$$

$$
\begin{bmatrix}
-2u_{1,1}^t + u_{1,2}^t + h^2 dtf_{1,1}^t \\
u_{1,1}^t - 2u_{1,2}^t + u_{1,3}^t + h^2 dtf_{i,j}^t \\
\vdots \\
u_{N_x,N_y-2}^t - 2u_{N_x,N_y-1}^t + u_{N_x,N_y}^t + h^2 dtf_{N_x,N_y-1}^t \\
u_{N_x,N_y-1}^t - 2u_{N_x,N_y}^t + h^2 dtf_{N_x,N_y}^t
\end{bmatrix}
$$

Isogeometric L2 projections

Longfei Gao, *Kronecker Products on Preconditioning*, PhD. Thesis, KAUST (supervised by Victor Calo), 2013.



Isogeometric basis functions:

- 1D B-splines basis $B_1(x), \ldots, B_n(x)$
- higher dimensions: tensor product basis $B_{i_1 \cdots i_d}(x_1, \ldots, x_d) \equiv B^{x_1}_{i_1}(x_1) \cdots B^{x_d}_{i_d}(x_d)$

Isogeometric L2 projections

Longfei Gao, *Kronecker Products on Preconditioning*, PhD. Thesis, KAUST (supervised by Victor Calo), 2013.

Gram matrix of B-spline basis on 2D domain $\Omega = \Omega_x \times \Omega_y$:

$$\mathcal{M}_{ijkl} = (B_{ij}, B_{kl})_{L^2} = \int_\Omega B_{ij} B_{kl} \, d\Omega$$

$$= \int_\Omega B^x_i(x) B^y_j(y) B^x_k(x) B^y_l(y) \, d\Omega$$

$$= \int_\Omega (B_i B_k)(x) \, (B_j B_l)(y) \, d\Omega$$

$$= \left(\int_{\Omega_x} B_i B_k \, dx \right) \left(\int_{\Omega_y} B_j B_l \, dy \right)$$

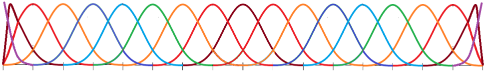
$$= \mathcal{M}^x_{ik} \mathcal{M}^y_{jl}$$

$$\mathcal{M} = \mathcal{M}^x \otimes \mathcal{M}^y \quad \text{(Kronecker product)}$$

Isogeometric L2 projections

Isogeometric L2 projections

Longfei Gao, *Kronecker Products on Preconditioning*, PhD. Thesis, KAUST (supervised by Victor Calo), 2013.



B-spline basis functions have **local support** (over $p+1$ elements)
$\mathcal{M}^x$, $\mathcal{M}^y$, ...– banded structure
$\mathcal{M}^x_{ij} = 0 \iff |i-j| > 2p+1$
Exemplary basis functions and matrix for cubics

$$\begin{bmatrix} (B_1,B_1)_{L^2} & (B_1,B_2)_{L^2} & (B_1,B_3)_{L^2} & (B_1,B_4)_{L^2} & 0 & 0 & \cdots & 0 \\ (B_2,B_1)_{L^2} & (B_2,B_2)_{L^2} & (B_2,B_3)_{L^2} & (B_2,B_4)_{L^2} & (B_2,B_5)_{L^2} & 0 & \cdots & 0 \\ (B_3,B_1)_{L^2} & (B_3,B_2)_{L^2} & (B_3,B_3)_{L^2} & (B_3,B_4)_{L^2} & (B_3,B_5)_{L^2} & (B_3,B_6)_{L^2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & (B_n,B_{n-3})_{L^2} & (B_n,B_{n-2})_{L^2} & (B_n,B_{n-1})_{L^2} & (B_n,B_n)_{L^2} & \end{bmatrix}$$

Isogeometric L2 projections

Isogeometric L2 projections

Two steps – solving systems with **A** and **B** in different *directions*

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & 0 \\ A_{21} & A_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} y_{11} & y_{21} & \cdots & y_{m1} \\ y_{12} & y_{22} & \cdots & y_{m1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{1n} & y_{2n} & \cdots & y_{mn} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{21} & \cdots & b_{m1} \\ b_{12} & b_{22} & \cdots & b_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{1n} & b_{2n} & \cdots & b_{mn} \end{bmatrix}$$

$$\begin{bmatrix} B_{11} & B_{12} & \cdots & 0 \\ B_{21} & B_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B_{mm} \end{bmatrix} \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ x_{21} & \cdots & x_{2n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{bmatrix}$$

Two 1D problems with multiple RHS, linear cost $O(N)$

- $n \times n$ with m right hand sides $\rightarrow O(n * m) = O(N)$
- $m \times m$ with n right hand sides $\rightarrow O(m * n) = O(N)$

Derivation of Spatial Direction Splitting

Idea exploit Kronecker product structure of $\mathcal{M} = \mathcal{M}^x \otimes \mathcal{M}^y$

Generally, consider

$$\mathbf{M}\mathbf{x} = \mathbf{b}$$

with $\mathbf{M} = \mathbf{A} \otimes \mathbf{B}$, where $\mathbf{A}$ is $n \times n$, $\mathbf{B}$ is $m \times m$

Definition of Kronecker (tensor) product:

$$\mathbf{M} = \mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} \mathbf{A}\,B_{11} & \mathbf{A}\,B_{12} & \cdots & \mathbf{A}\,B_{1m} \\ \mathbf{A}\,B_{21} & \mathbf{A}\,B_{22} & \cdots & \mathbf{A}\,B_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}\,B_{m1} & \mathbf{A}\,B_{m2} & \cdots & \mathbf{A}\,B_{mm} \end{bmatrix}$$

Derivation of Spatial Direction Splitting

RHS and solution are partitioned into m blocks of size n each

$$\mathbf{x}_i = (x_{i1}, \dots, x_{in})^\mathsf{T}$$
$$\mathbf{b}_i = (b_{i1}, \dots, b_{in})^\mathsf{T}$$

We can rewrite the system as a block matrix equation:

$$\begin{cases} \mathbf{A}B_{11}\mathbf{x}_1 + \mathbf{A}B_{12}\mathbf{x}_2 + \cdots + \mathbf{A}B_{1m}\mathbf{x}_m = \mathbf{b}_1 \\ \mathbf{A}B_{21}\mathbf{x}_1 + \mathbf{A}B_{22}\mathbf{x}_2 + \cdots + \mathbf{A}B_{2m}\mathbf{x}_m = \mathbf{b}_2 \\ \quad\vdots \qquad\qquad \vdots \qquad \vdots \qquad\qquad \vdots \\ \mathbf{A}B_{m1}\mathbf{x}_1 + \mathbf{A}B_{m2}\mathbf{x}_2 + \cdots + \mathbf{A}B_{mm}\mathbf{x}_m = \mathbf{b}_m \end{cases}$$

Derivation of Spatial Direction Splitting

Factor out $\mathbf{A}$:

$$\begin{cases} \mathbf{A}\big(B_{11}\mathbf{x}_1 + B_{12}\mathbf{x}_2 + \cdots + B_{1m}\mathbf{x}_m\big) = \mathbf{b}_1 \\ \mathbf{A}\big(B_{21}\mathbf{x}_1 + B_{22}\mathbf{x}_2 + \cdots + B_{2m}\mathbf{x}_m\big) = \mathbf{b}_2 \\ \quad\vdots \qquad\qquad \vdots \qquad \vdots \qquad\qquad \vdots \\ \mathbf{A}\big(B_{m1}\mathbf{x}_1 + B_{m2}\mathbf{x}_2 + \cdots + B_{mm}\mathbf{x}_m\big) = \mathbf{b}_m \end{cases}$$

Wy multiply by $\mathbf{A}^{-1}$ and define $\mathbf{y}^i = \mathbf{A}^{-1}\mathbf{b}^i$
(we have one 1D problem here $\mathbf{A}\ \mathbf{y}^i = \mathbf{b}^i$ with multiple RHS)

$$\begin{cases} B_{11}\mathbf{x}_1 + B_{12}\mathbf{x}_2 + \cdots + B_{1m}\mathbf{x}_m = \mathbf{y}_1 \\ B_{21}\mathbf{x}_1 + B_{22}\mathbf{x}_2 + \cdots + B_{2m}\mathbf{x}_m = \mathbf{y}_2 \\ \quad\vdots \qquad\qquad \vdots \qquad \vdots \qquad\qquad \vdots \\ B_{m1}\mathbf{x}_1 + B_{m2}\mathbf{x}_2 + \cdots + B_{mm}\mathbf{x}_m = \mathbf{y}_m \end{cases}$$

Derivation of Spatial Direction Splitting

Consider each component of $\mathbf{x}_i$ and $\mathbf{y}_i$ $\Rightarrow$ family of linear systems

$$\begin{cases} B_{11}x^{1i} + B_{12}x^{2i} + \cdots + B_{1m}x^{mi} = y_{1i} \\ B_{21}x^{1i} + B_{22}x^{2i} + \cdots + B_{2m}x^{mi} = y_{2i} \\ \quad\vdots \qquad\qquad \vdots \qquad \vdots \qquad\qquad \vdots \\ B_{m1}x^{1i} + B_{m2}x^{2i} + \cdots + B_{mm}x^{mi} = y_{mi} \end{cases}$$

for each $i = 1, \ldots, n$

$\Rightarrow$ linear systems with matrix $\mathbf{B}$ (We have another 1D problem here with multiple RHS $\mathbf{B}\ \mathbf{x}^i = \mathbf{y}^i$)

Explicit dynamics

Applications to time-dependent problems (Fortran sequential)
M. Łoś, M. Woźniak, M. Paszyński, L. Dalcin, V.M. Calo, Dynamics with Matrices Possessing Kronecker Product Structure, **Procedia Computer Science** 51 (2015) 286-295

In general: non-stationary problem of the form

$$\partial_t u - \mathcal{L}(u) = f(x, t)$$

with some initial state u_0 and boundary conditions

$\mathcal{L}$ – well-posed linear spatial partial differential operator

Discretization:

- spatial discretization: isogeometric FEM

  Basis functions: tensor product B-splines
  $u(x, y) \approx \sum_{i,j} u_{i,j} B^x_{i,p}(x) B^y_{j,p}(y)$

Explicit dynamics

Applications to time-dependent problems (Fortran sequential)
M. Łoś, M. Woźniak, M. Paszyński, L. Dalcin, V.M. Calo, Dynamics with Matrices Possessing Kronecker Product Structure, **Procedia Computer Science** 51 (2015) 286-295

- spatial discretization: isogeometric FEM

  Basis functions: tensor product B-splines
  $u(x,y) \approx \sum_{i,j} u_{k,l} B^x_{i,p}(x) B^y_{j,p}(y)$

- time discretization with explicit method
  $\frac{u_{t+1}-u_t}{dt} = Lu_t + f_t \rightarrow u_{t+1} = u_t + dtLu_t$

- implies isogeometric L2 projections in every time step
  $(u_{t+1}, v)_{L2} = (u_t + dtLu_t, v)_{L2}$

Explicit dynamics

Applications to time-dependent problems (Fortran sequential)
M. Łoś, M. Woźniak, M. Paszyński, L. Dalcin, V.M. Calo, Dynamics with Matrices Possessing Kronecker Product Structure, **Procedia Computer Science** 51 (2015) 286-295

- implies isogeometric L2 projections in every time step
  $(u_{t+1}, v)_{L2} = (u_t + dtLu_t, v)_{L2}$
  $u_{t+1} \approx \sum_{i,j} u_{t+1}^{i,j} B^x_{i,p}(x) B^y_{j,p}(y)$, $v \leftarrow B^x_{k,p}(x) B^y_{l,p}(y)$
  $u_t \approx \sum_{i,j} u_t^{i,j} B^x_{i,p}(x) B^y_{j,p}(y))$
- so the system looks like
  $\sum_{i,j} u_{t+1}^{i,j} (B^x_{i,p}(x) B^y_{j,p}(y), B^x_{k,p}(x) B^y_{l,p}(y))_{L2} =$
  $\sum_{i,j} u_t^{i,j} (B^x_{i,p}(x) B^y_{j,p}(y)) +$
  $dt \sum_{i,j} u_t^{i,j} L(B^x_{i,p}(x) B^y_{j,p}(y)), v)_{L2} \quad \forall k, l$

Applications to time-dependent problems (Fortran sequential)

M. Łoś, M. Woźniak, M. Paszyński, L. Dalcin, V.M. Calo, Dynamics with Matrices Possessing Kronecker Product Structure, **Procedia Computer Science** 51 (2015) 286-295

- sequence of isogeometric L2 projections

$$\sum_{i,j} u^{i,j}_{t+1}(B^x_{i,p}(x)B^y_{j,p}(y), B^x_{k,p}(x)B^y_{l,p}(y))_{L2} =$$
$$\sum_{i,j} u^{i,j}_{t}(B^x_{i,p}(x)B^y_{j,p}(y)) + dt \sum_{i,j} u^{i,j}_{t} L(B^x_{i,p}(x)B^y_{j,p}(y)), B^x_{k,p}B^y_{l,p})_{L2} \quad \forall k,l$$

$$\begin{bmatrix} (B^x_{1,p}B^y_{1,p}, B^x_{1,p}B^y_{1,p})_{L2} & (B^x_{1,p}B^y_{1,p}, B^x_{2,p}B^y_{1,p})_{L2} & \cdots & (B^x_{1,p}B^y_{1,p}, B^x_{N_x,p}B^y_{N_y,p})_{L2} \\ (B^x_{2,p}B^y_{1,p}, B^x_{1,p}B^y_{1,p})_{L2} & (B^x_{2,p}B^y_{1,p}, B^x_{2,p}B^y_{1,p})_{L2} & \cdots & (B^x_{2,p}B^y_{1,p}, B^x_{N_x,p}B^y_{N_y,p})_{L2} \\ \vdots & \vdots & \vdots & \vdots \\ (B^x_{N_x,p}B^y_{N_y,p}, B^x_{1,p}B^y_{1,p})_{L2} & (B^x_{N_x,p}B^y_{N_y,p}, B^x_{2,p}B^y_{1,p})_{L2} & \cdots & (B^x_{N_x,p}B^y_{N_y,p}, B^x_{N_x,p}B^y_{N_y,p})_{L2} \end{bmatrix} \begin{bmatrix} u^{1,1}_{t+1} \\ u^{2,1}_{t+1} \\ \vdots \\ u^{N_x,N_y}_{t+1} \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i,j} u^{i,j}_{t}(B^x_{i,p}(x)B^y_{j,p}(y)) + dt \sum_{i,j} u^{i,j}_{t} L(B^x_{i,p}(x)B^y_{j,p}(y)), B^x_{1,p}B^y_{1,p})_{L2} \\ \sum_{i,j} u^{i,j}_{t}(B^x_{i,p}(x)B^y_{j,p}(y)) + dt \sum_{i,j} u^{i,j}_{t} L(B^x_{i,p}(x)B^y_{j,p}(y)), B^x_{2,p}B^y_{1,p})_{L2} \\ \vdots \\ \sum_{i,j} u^{i,j}_{t}(B^x_{i,p}(x)B^y_{j,p}(y)) + dt \sum_{i,j} u^{i,j}_{t} L(B^x_{i,p}(x)B^y_{j,p}(y)), B^x_{N_x,p}B^y_{N_y,p})_{L2} \end{bmatrix}$$

Example 1: Heat transfer equation

We seek the temperature scalar field $u\colon \Omega \to \mathbb{R}$ such as:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \Delta u + f(\mathbf{x}) & \text{on } \Omega \times [0, T] \\ \nabla u \cdot \hat{\mathbf{n}} = 0 & \text{on } \partial\Omega \times [0, T] \\ u(\mathbf{x}, 0) = u_0(\mathbf{x}) & \text{on } \Omega \end{cases} \tag{1}$$

where $\Omega = [0, 1]^2$,
$\hat{\mathbf{n}}$ is a normal vector of the domain boundary,
T is a length of the time interval for the simulation,
and u_0 is an initial state.
$f = 0$ (no heat source)

Example 1: Heat transfer equation

The corresponding weak formulation is obtained by multiplying (1) by test function $w \in H^1(\Omega)$, integrating by parts over Ω, and imposing the boundary conditions.

Find $u \in \mathcal{C}^1([0,T], H^1(\Omega))$ such that for each $t \in [0,T]$

$$(\frac{\partial u}{\partial t}, w)_{L^2} = -(\nabla u, \nabla w)_{L^2} + (f, w)_{L^2} \qquad \forall w \in H^1(\Omega) \tag{2}$$

where $(\cdot,\cdot)_{L^2}$ stands for the $L^2(\Omega)$ scalar product
We utilize Euler time integration scheme

$$(u_{t+1}, w)_{L^2} = (u_t, w)_{L_2} - dt * \nabla u_t, \nabla w)_{L^2} \qquad \forall w \in H^1(\Omega) \tag{3}$$

Example 1: Heat transfer equation

Click in the middle

Code for Example 1 (Heat transfer equation)

"problems/heat/heat_3d.cpp"

```
#include "problems/heat/heat_3d.hpp"
using namespace ads;
using namespace ads::problems;
```

pilot for the simulation

```
int main() {
```

quadratic B-splnes, 12 elements along axis

```
  dim_config dim{ 2, 12 };
```

5000 time steps, time step size 10^{-7}

```
  timesteps_config steps{ 5000, 1e-7 };
```

we will need to compute first derivatives during the computations

```
  int ders = 1;
```

some auxiliary objects for configuration and simulation

```
config_3d c{dim, dim, dim, steps, ders};
heat_3d sim{c};
```

run the simulation

```
sim.run();
}
```

Code for Example 1 (Heat transfer equation)

"problems/heat/heat_3d.hpp"

```
#include "ads/simulation.hpp"
using namespace ads;
using namespace problems;
class heat_3d :  public simulation_3d {
...
```

implementation of the initial state

```
double init_state(double x, double y, double z)
```

executed once before the simulation starts

```
void before() override
```

executed before every simulation step

```
void before_step() override
```

implementation of the simulation step

```
void step() override
```

executed after every simulation step

```
void after_step() override
```

implementation of generation of RHS

```
void compute_rhs() override
```

executed once after the simulation ends

```
void after() override
```

Code for Example 1 (Heat transfer equation)

"problems/heat/heat_3d.hpp"

this functions is called from *before* at the beginning of the simulation

the function returns the value of $u_0 = u(x, y, z)|_{t=0}$)

computed at point (x, y, z)

```
double init_state(double x, double y, double z) {
    double dx = x - 0.5;
    double dy = y - 0.5;
    double dz = z - 0.5;
    double r2 = std::min(8*(dx*dx+dy*dy+dz*dz),1.0);
    return (r2 - 1) * (r2 - 1) * (r2 + 1) * (r2 + 1);
  };
```

Code for Example 1 (Heat transfer equation)

"problems/heat/heat_3d.hpp"

this function is called once before the simulation starts

```
void before() override {
```

performs LU factorization of three 1D systems, representing B-splines along x, y and z axes

```
    prepare_matrices();
```

pointer to *init_state* function

```
    auto init = [this](double x, double y, double z)
     { return init_state(x, y, z); };
```

preparation of the initial state

```
    projection(u, init);
```

forward and backward substitutions with multiple RHS

```
    solve(u);
    }
```

Code for Example 1 (Heat transfer equation)

"problems/heat/heat_3d.hpp"

this function is called before every time step

```
void before_step(int /*iter*/, double /*t*/) override
{
    using std::swap;
```

swap u_t and u_{t-1}

```
    swap(u, u_prev);
}
```

this function implements every time step

```
void step(int /*iter*/, double /*t*/) override {
```

generate new RHS using u_prev

```
    compute_rhs();
```

forward and backward substitutions with multiple RHS

```
    solve(u);
}
```

Example 1: Heat transfer equation

$$(u_{t+1}, w)_{L^2} = (u_t, w)_{L_2} - dt * (\nabla u_t, \nabla w)_{L^2} \qquad \forall w \in H^1(\Omega) \quad (4)$$

value of test function a over element e at Gauss point q

```
value_type v = eval_basis(e, q, a);
```

value of u_t at Gauss point

```
value_type u = eval_fun(u_prev, e, q);
```

computations of double gradient

```
double gradient = u.dx*v.dx+u.dy*v.dy+u.dz*v.dz;
```

$RHS = u_t - dt \nabla u_t \cdot \nabla v$

```
double val = u.val*v.val - steps.dt * gradient;
```

scale by Jacobian and weight

```
rhs(a[0],a[1],a[2])+=val*w*J;
```

Code for Example 1 (Heat transfer equation)

```
void compute_rhs() {
auto& rhs = u; zero(rhs);
for (auto e :  elements()) {  loop through elements
  double J = jacobian(e); compute Jacobian
  for (auto q:quad_points()){ loop through Gauss points
   double w = weigth(q);  Gauss weight
   for (auto a :  dofs_on_element(e)){loop through dofs
```

value of basis function q over element e at Gauss point q

```
    value_type v = eval_basis(e, q, a);
```

value of u_t at Gauss point

this also computes derivatives and stored at *.dx

```
    value_type u = eval_fun(u_prev, e, q);
```

computations of double gradient

```
    double gradient = u.dx*v.dx+u.dy*v.dy+u.dz*v.dz;
```

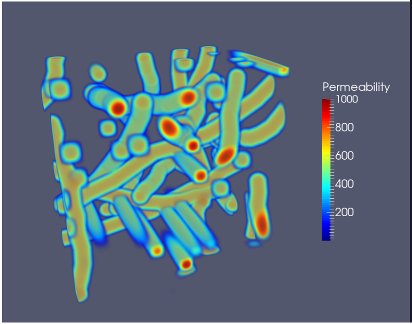
$RHS = u_t - dt\nabla u \cdot \nabla v$

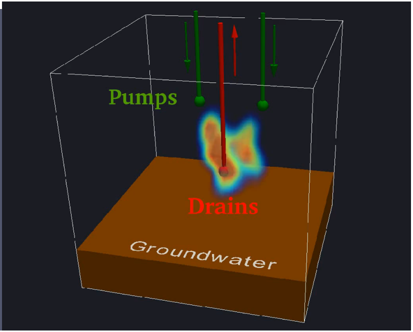
```
    double val = u.val*v.val - steps.dt * gradient;
```

scale by Jacobian and weight

```
    rhs(a[0],a[1],a[2])+=val*w*J;
```

Hydraulic fracturing - oil/gas extraction technique consisting in high-pressure fluid injection into the deposit





Example 2: Non-linear flow in heterogenous media

Hydraulic fracturing - oil/gas extraction technique consisting in high-pressure fluid injection into the deposit
Spatial domain $= \Omega = [0,1]^3$

$$\begin{cases} \dfrac{\partial u}{\partial t} - \nabla \cdot (\kappa(\mathbf{x}, u)\, \nabla u) = h(\mathbf{x}, t) & \text{in } \Omega \times [0, T] \\ \nabla u \cdot \hat{n} = 0 & \text{on } \partial \Omega \times [0, T] \\ u(x, 0) = u_0 & \text{in } \Omega \end{cases}$$

- u – pressure
- zero Neumann boundary conditions
- initial state u_0
- κ – permeability
- h – **forcing** (induced by extraction method)

M. Alotaibi, V.M. Calo, Y. Efendiev, J. Galvis, M. Ghommem, *Global-Local Nonlinear Model Reduction for Flows in Heterogeneous Porous Media* **arXiv:1407.0782 [math.NA]**

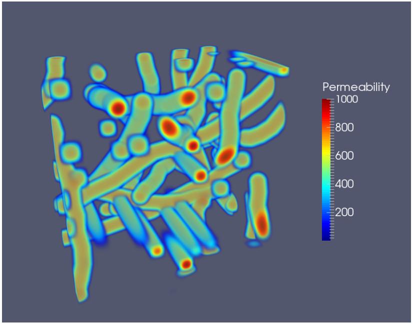
Example 2: Non-linear flow in heterogenous media

$$\kappa(\mathbf{x}, u) = K_q(x)\, b(u)$$

$$b(u) = e^{\mu u}$$

$$\mu = 10$$

$K_q(\mathbf{x})$ – property of the terrain (example below)



Example 2: Non-linear flow in heterogenous media

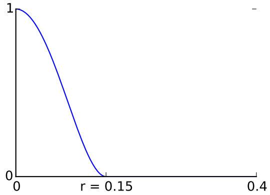
Extraction process modeled by **pumps** and **sinks**

- pump/sink has a location $\mathbf{x} \in \Omega$
- pumps locally increase the pressure u
- sinks locally decrease u (the higher, the faster)

$$h(x,t) = \sum_{p \in P} \phi\left(\|x_p - x\|\right) - \sum_{s \in S} u(x,t)\phi\left(\|x_s - x\|\right)$$

- P, S – sets of pump and sinks
- x_p, x_s – location of pump p/sink s
- ϕ – cut-off function ($r = 0.15$)

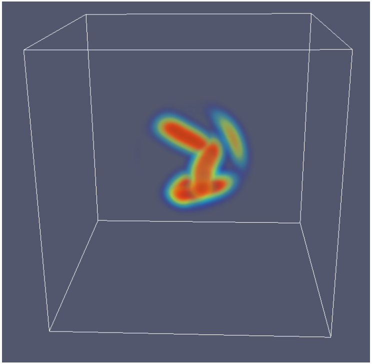
$$\phi(t) = \begin{cases} \left(\frac{t}{r} - 1\right)^2 \left(\frac{t}{r} + 1\right)^2 & \text{for } t \leq r \\ 0 & \text{for } t > r \end{cases}$$



Example 2: Non-linear flow in heterogenous media

Initial state is derived from the permeability of the material K_q

$$\tilde{K}_q(\mathbf{x}) = (K_q(\mathbf{x}) - 1)/(1000 - 1)$$
$$u_0(\mathbf{x}) = 0.1\,\tilde{K}_q(\mathbf{x})\,\theta_{0.2,0.3}(\|\mathbf{x} - \mathbf{c}\|)$$
$$\mathbf{c} = (0.5, 0.5, 0.5)$$



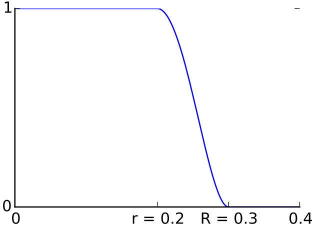


Figure: $\theta_{r,R}$

Example 2: Non-linear flow in heterogenous media

We utilize Euler time integration scheme

$$(u_{t+1}, w)_{L^2} = (u_t - dt * K_q(x)\, e^{10 * u_t}, u_t) + (\nabla u_t + h(u_t), \nabla w)_{L^2} \quad \forall w \in H^1 ($$

where $K_q(x, t)$ does not change with time, and it is given by the permeability map,
$h(x, t)$ are pumps and sinks

$$h(x, t) = \sum_{p \in P} \phi\left(\|x_p - x\|\right) - \sum_{s \in S} u(x, t)\phi\left(\|x_s - x\|\right)$$

Example 2: Non-linear flow in heterogenous media

Click in the middle