A community software approach to BIE-PDE coupling

Dan Fortunato

Flatiron Institute

Introduction
Community software

- Many pieces of community software exist for the high-order accurate solution of BIEs and PDEs.

Introduction
Community software

- Many pieces of community software exist for the high-order accurate solution of BIEs and PDEs.

BIE	PDE
chunkie **fmm3dbie** **bempp** **pyBIE2D**	**chebfun** **Dedalus** **ApproxFun** **surfacefun**
Askham, Rachh, Hoskins, O'Neil, Vico, Betcke, Kailasa, Stein, F., …	*Trefethen, Hale, Townsend, Burns, Vasil, Lecoanet, Olver, F., …*

Introduction
Community software

- Many pieces of community software exist for the high-order accurate solution of BIEs and PDEs.

BIE	PDE
chunkie **fmm3dbie** **bempp** **pyBIE2D**	**chebfun** **Dedalus** **ApproxFun** **surfacefun**
Askham, Rachh, Hoskins, O'Neil, Vico, Betcke, Kailasa, Stein, F., …	*Trefethen, Hale, Townsend, Burns, Vasil, Lecoanet, Olver, F., …*

- Some have reached a mature state with nice abstractions provided in a high-level language, allowing us to write code that reflects the mathematics.

Introduction
Community software

- Many pieces of community software exist for the high-order accurate solution of BIEs and PDEs.

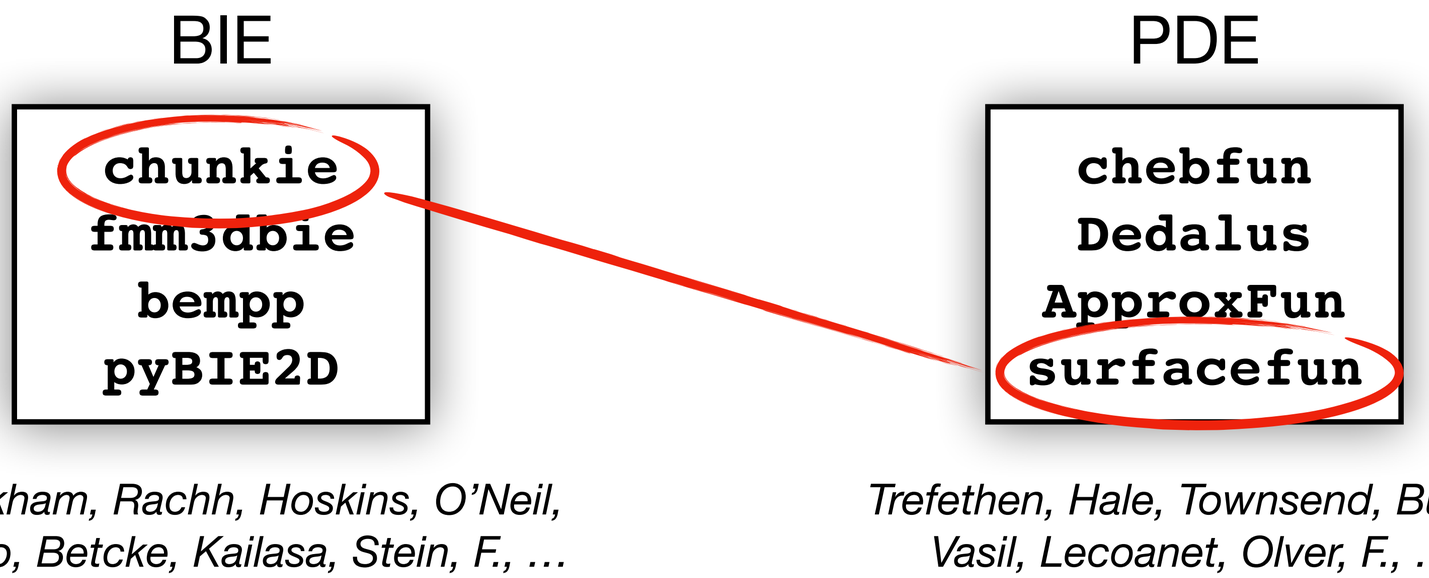
BIE	PDE
chunkie **fmm3dbie** **bempp** **pyBIE2D**	**chebfun** **Dedalus** **ApproxFun** **surfacefun**
Askham, Rachh, Hoskins, O'Neil, Vico, Betcke, Kailasa, Stein, F., …	*Trefethen, Hale, Townsend, Burns, Vasil, Lecoanet, Olver, F., …*

- Some have reached a mature state with nice abstractions provided in a high-level language, allowing us to write code that reflects the mathematics.

- This talk is a practical demonstration of how two pieces of community software can be "glued" together (in MATLAB).

Introduction
Community software

- Many pieces of community software exist for the high-order accurate solution of BIEs and PDEs.



- Some have reached a mature state with nice abstractions provided in a high-level language, allowing us to write code that reflects the mathematics.

- This talk is a practical demonstration of how two pieces of community software can be "glued" together (in MATLAB).

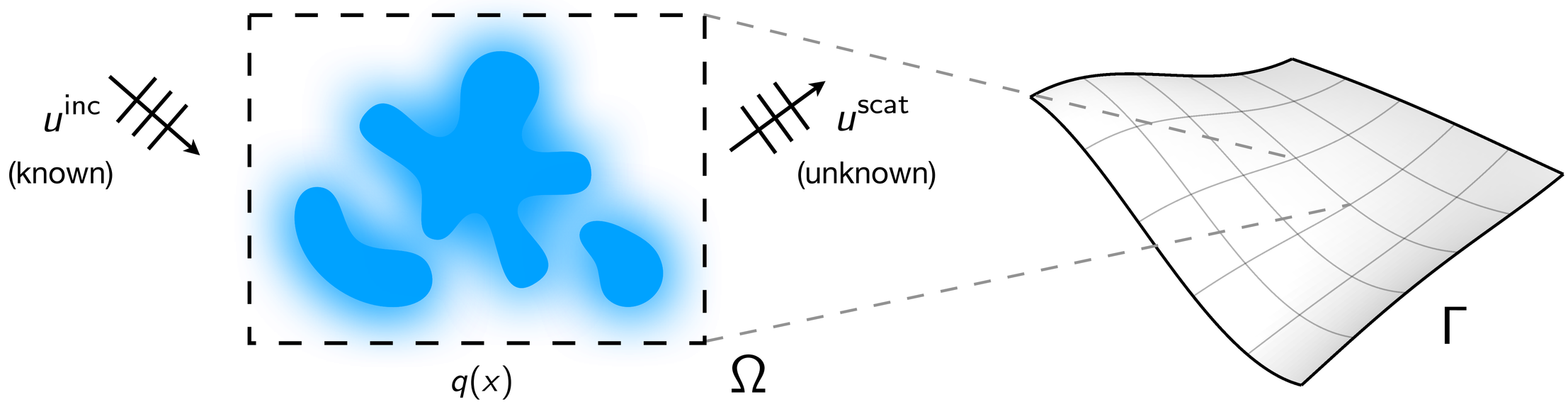
A model problem
Scattering on a surface

Consider wave scattering in a variable medium on a manifold Γ, modeled by the surface Helmholtz equation

$$\Delta_\Gamma u(x) + k^2(1 - q(x))u(x) = 0$$

with the total field $u = u^{\text{inc}} + u^{\text{scat}}$ and the Sommerfeld radiation condition

$$\lim_{r\to\infty} \sqrt{r}\left(\frac{\partial u^{\text{scat}}}{\partial r} - iku^{\text{scat}}\right) = 0$$



[Gillman, Barnett, Martinsson, 2015]
[Borges, Gillman, Greengard, 2017]

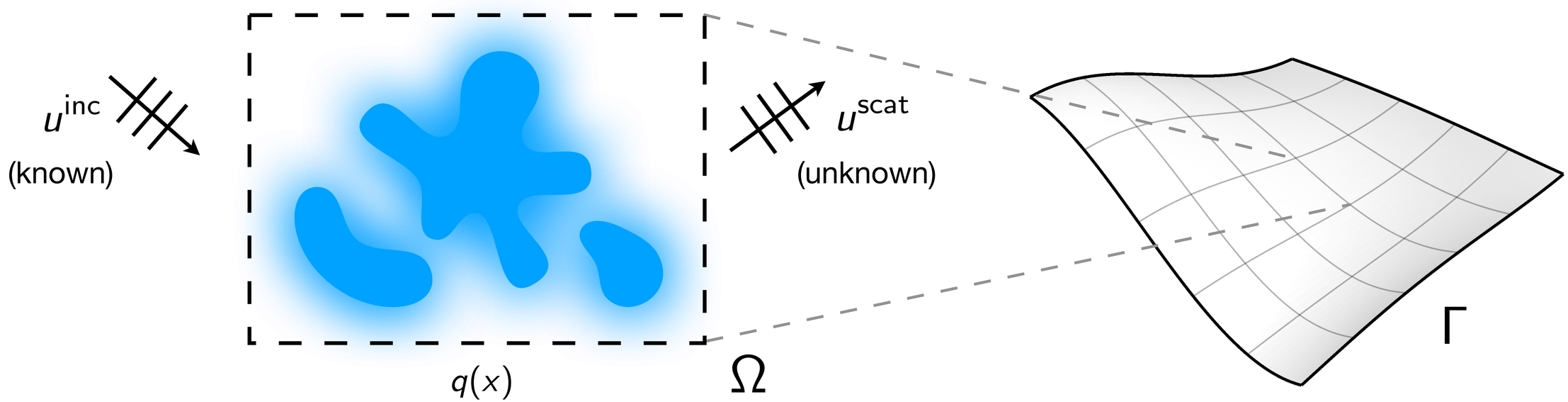
A model problem
Scattering on a surface

Consider wave scattering in a variable medium on a manifold Γ, modeled by the surface Helmholtz equation

$$\Delta_\Gamma u(x) + k^2(1 - q(x))u(x) = 0$$

with the total field $u = u^{\text{inc}} + u^{\text{scat}}$ and the Sommerfeld radiation condition

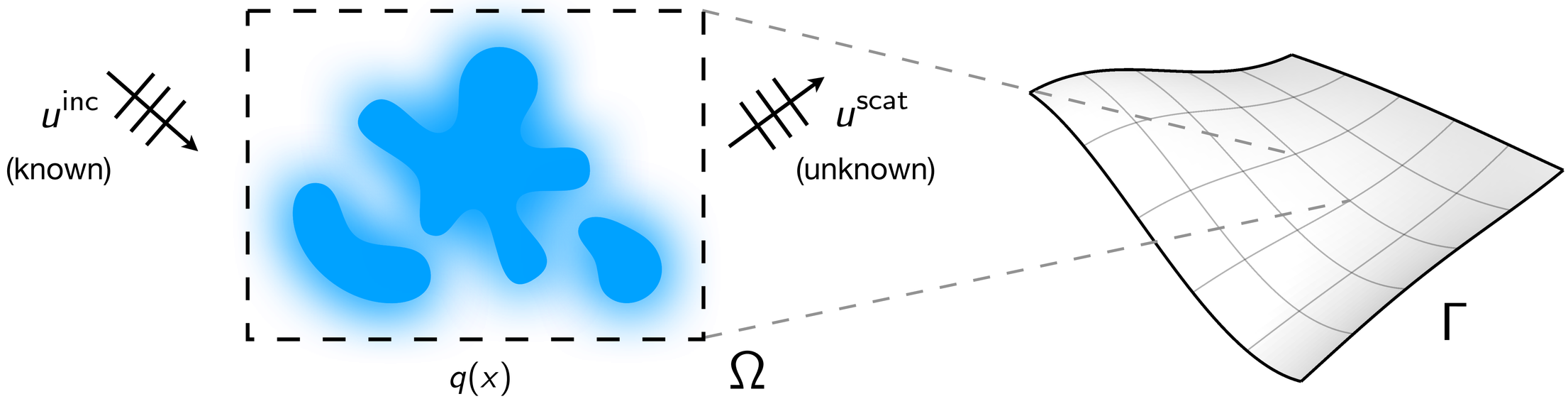
$$\lim_{r\to\infty} \sqrt{r}\left(\frac{\partial u^{\text{scat}}}{\partial r} - iku^{\text{scat}}\right) = 0$$



[Gillman, Barnett, Martinsson, 2015]
[Borges, Gillman, Greengard, 2017]

A model problem
Scattering on a surface

For compactly supported $q(x)$ and Γ, we can decompose this into two problems:



[Gillman, Barnett, Martinsson, 2015]
[Borges, Gillman, Greengard, 2017]

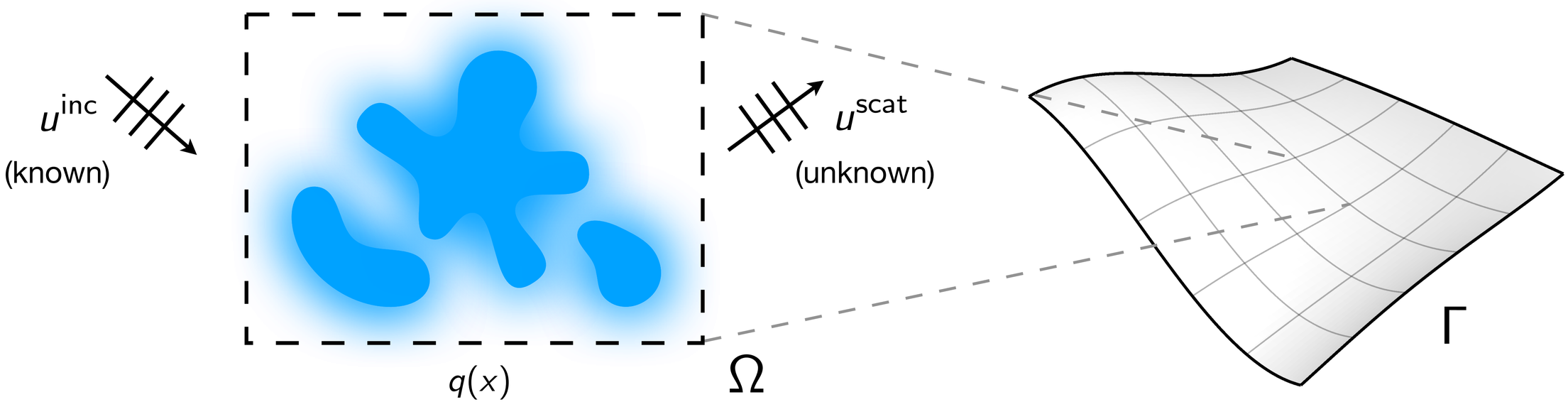
A model problem
Scattering on a surface

For compactly supported $q(x)$ and Γ, we can decompose this into two problems:

Interior problem

$$\begin{aligned} \Delta_\Gamma u(x) + k^2(1 - q(x))u(x) &= 0 && \text{in } \Omega \\ u(x) &= g(x) && \text{on } \partial\Omega \end{aligned}$$

curved, variable coefficient



[Gillman, Barnett, Martinsson, 2015]
[Borges, Gillman, Greengard, 2017]

A model problem
Scattering on a surface

For compactly supported $q(x)$ and Γ, we can decompose this into two problems:

Interior problem

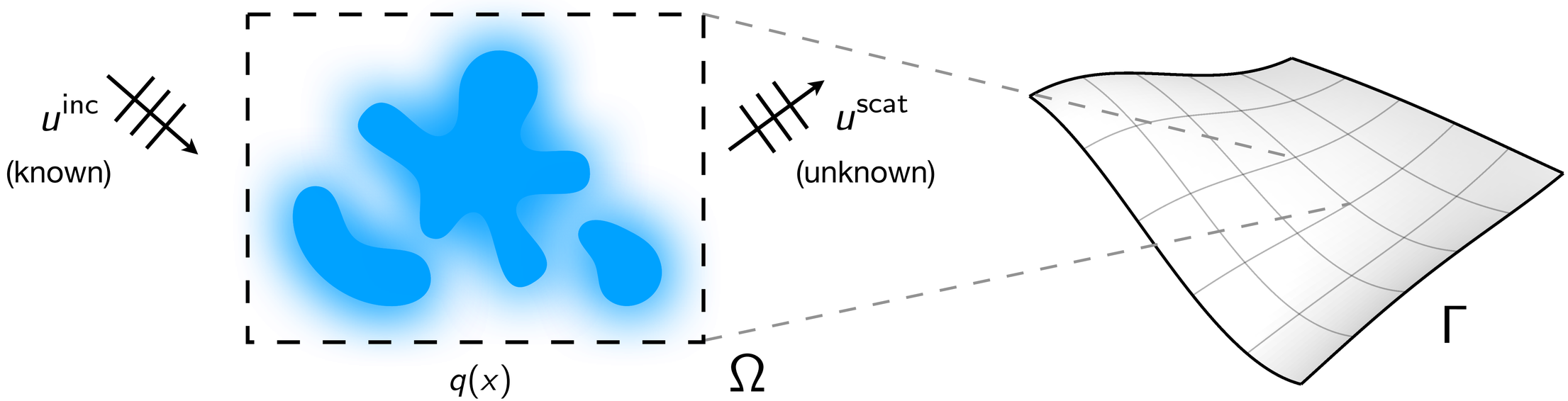
$$\begin{aligned} \Delta_\Gamma u(x) + k^2(1 - q(x))u(x) &= 0 && \text{in } \Omega \\ u(x) &= g(x) && \text{on } \partial\Omega \end{aligned}$$

curved, variable coefficient

Exterior problem

$$\begin{aligned} \Delta u^{\text{scat}}(x) + k^2 u^{\text{scat}}(x) &= 0 && \text{in } \mathbb{R}^2 \setminus \Omega \\ u^{\text{scat}}(x) &= h(x) && \text{on } \partial\Omega \\ \frac{\partial u^{\text{scat}}}{\partial r} - iku^{\text{scat}} &= o(r^{-1/2}) && \text{as } r = \|x\| \to \infty \end{aligned}$$

flat, constant coefficient



[Gillman, Barnett, Martinsson, 2015]
[Borges, Gillman, Greengard, 2017]

A model problem
Scattering on a surface

For compactly supported $q(x)$ and Γ, we can decompose this into two problems:

Interior problem

$$\begin{aligned} \Delta_\Gamma u(x) + k^2(1-q(x))u(x) &= 0 && \text{in } \Omega \\ u(x) &= g(x) && \text{on } \partial\Omega \end{aligned}$$

curved, variable coefficient

Exterior problem

$$\begin{aligned} \Delta u^{\text{scat}}(x) + k^2 u^{\text{scat}}(x) &= 0 && \text{in } \mathbb{R}^2 \setminus \Omega \\ u^{\text{scat}}(x) &= h(x) && \text{on } \partial\Omega \\ \frac{\partial u^{\text{scat}}}{\partial r} - iku^{\text{scat}} &= o(r^{-1/2}) && \text{as } r = \|x\| \to \infty \end{aligned}$$

flat, constant coefficient

1. *What data $g(x), h(x)$ will give us the solution to the original problem?*
2. *How to solve the interior problem?* (Hint: PDE)
3. *How to solve the exterior problem?* (Hint: BIE)

[Gillman, Barnett, Martinsson, 2015]
[Borges, Gillman, Greengard, 2017]

Coupling the interior and exterior
Dirichlet-to-Neumann operators

For a second-order elliptic PDE, imposing continuity of $u(x)$ and its normal derivative across Ω will recover the original solution.

[Gillman, Barnett, Martinsson, 2015]
[Borges, Gillman, Greengard, 2017]

Coupling the interior and exterior
Dirichlet-to-Neumann operators

For a second-order elliptic PDE, imposing continuity of $u(x)$ and its normal derivative across Ω will recover the original solution.

Define the ***Dirichlet-to-Neumann operators*** as the unique operators satisfying

$$T_{\text{int}}\, g = \left.\frac{\partial u}{\partial n}\right|_{\partial\Omega}, \quad T_{\text{ext}}\, h = \left.\frac{\partial u^{\text{scat}}}{\partial n}\right|_{\partial\Omega} \qquad \forall g, h \in H^1(\partial\Omega)$$

[Gillman, Barnett, Martinsson, 2015]
[Borges, Gillman, Greengard, 2017]

Coupling the interior and exterior
Dirichlet-to-Neumann operators

For a second-order elliptic PDE, imposing continuity of $u(x)$ and its normal derivative across Ω will recover the original solution.

Define the ***Dirichlet-to-Neumann operators*** as the unique operators satisfying

$$T_{\text{int}}\, g = \left.\frac{\partial u}{\partial n}\right|_{\partial\Omega}, \quad T_{\text{ext}}\, h = \left.\frac{\partial u^{\text{scat}}}{\partial n}\right|_{\partial\Omega} \qquad \forall g, h \in H^1(\partial\Omega)$$

Then the continuity conditions imply

$$\left(T_{\text{int}} - T_{\text{ext}}\right) u^{\text{scat}}|_{\partial\Omega} = \underbrace{\left.\frac{\partial u^{\text{inc}}}{\partial n}\right|_{\partial\Omega} - T_{\text{int}}\, u^{\text{inc}}|_{\partial\Omega}}_{\text{known}}$$

[Gillman, Barnett, Martinsson, 2015]
[Borges, Gillman, Greengard, 2017]

Coupling the interior and exterior
Dirichlet-to-Neumann operators

For a second-order elliptic PDE, imposing continuity of $u(x)$ and its normal derivative across Ω will recover the original solution.

Define the ***Dirichlet-to-Neumann operators*** as the unique operators satisfying

$$T_{\text{int}}\, g = \left.\frac{\partial u}{\partial n}\right|_{\partial\Omega}, \quad T_{\text{ext}}\, h = \left.\frac{\partial u^{\text{scat}}}{\partial n}\right|_{\partial\Omega} \qquad \forall g, h \in H^1(\partial\Omega)$$

Then the continuity conditions imply

$$(T_{\text{int}} - T_{\text{ext}})\, u^{\text{scat}}|_{\partial\Omega} = \underbrace{\left.\frac{\partial u^{\text{inc}}}{\partial n}\right|_{\partial\Omega} - T_{\text{int}}\, u^{\text{inc}}|_{\partial\Omega}}_{\text{known}}$$

Solving this linear system will tell us the scattered field on the boundary.

[Gillman, Barnett, Martinsson, 2015]
[Borges, Gillman, Greengard, 2017]

Exterior problem

Boundary integral formulation and `chunkie`

$$\begin{aligned} \Delta u^{\text{scat}}(x) + k^2 u^{\text{scat}}(x) &= 0 && \text{in } \mathbb{R}^2 \setminus \Omega \\ u^{\text{scat}}(x) &= h(x) && \text{on } \partial\Omega \\ \frac{\partial u^{\text{scat}}}{\partial r} - iku^{\text{scat}} &= o(r^{-1/2}) && \text{as } r = \|x\| \to \infty \end{aligned}$$

Exterior problem
Boundary integral formulation and `chunkie`

$$\begin{aligned} \Delta u^{\text{scat}}(x) + k^2 u^{\text{scat}}(x) &= 0 && \text{in } \mathbb{R}^2 \setminus \Omega \\ u^{\text{scat}}(x) &= h(x) && \text{on } \partial\Omega \\ \frac{\partial u^{\text{scat}}}{\partial r} - iku^{\text{scat}} &= o(r^{-1/2}) && \text{as } r = \|x\| \to \infty \end{aligned}$$

As we have a constant-coefficient problem in flat space, a BIE-based discretization is a good choice.

Exterior problem
Boundary integral formulation and `chunkie`

$$\begin{aligned} \Delta u^{\text{scat}}(x) + k^2 u^{\text{scat}}(x) &= 0 && \text{in } \mathbb{R}^2 \setminus \Omega \\ u^{\text{scat}}(x) &= h(x) && \text{on } \partial\Omega \\ \frac{\partial u^{\text{scat}}}{\partial r} - iku^{\text{scat}} &= o(r^{-1/2}) && \text{as } r = \|x\| \to \infty \end{aligned}$$

As we have a constant-coefficient problem in flat space, a BIE-based discretization is a good choice.

Green's representation formula: $\frac{1}{2}u^{\text{scat}}(x) = \mathcal{D}[u^{\text{scat}}|_{\partial\Omega}](x) - \mathcal{S}[\frac{\partial u^{\text{scat}}}{\partial n}|_{\partial\Omega}](x) \qquad \forall x \in \mathbb{R}^2 \setminus \Omega$

Helmholtz Green's function

$$\mathcal{S}[\sigma](x) = \int_{\partial\Omega} G(x,y)\sigma(y)ds(y) \qquad \textit{"Single layer potential"}$$

$$\mathcal{D}[\sigma](x) = \int_{\partial\Omega} \frac{\partial G(x,y)}{\partial n_y}\sigma(y)ds(y) \qquad \textit{"Double layer potential"}$$

Exterior problem
Boundary integral formulation and `chunkie`

$$\begin{aligned} \Delta u^{\text{scat}}(x) + k^2 u^{\text{scat}}(x) &= 0 && \text{in } \mathbb{R}^2 \setminus \Omega \\ u^{\text{scat}}(x) &= h(x) && \text{on } \partial\Omega \\ \frac{\partial u^{\text{scat}}}{\partial r} - iku^{\text{scat}} &= o(r^{-1/2}) && \text{as } r = \|x\| \to \infty \end{aligned}$$

As we have a constant-coefficient problem in flat space, a BIE-based discretization is a good choice.

Green's representation formula: $\frac{1}{2}u^{\text{scat}}(x) = \mathcal{D}[u^{\text{scat}}|_{\partial\Omega}](x) - \mathcal{S}[\frac{\partial u^{\text{scat}}}{\partial n}|_{\partial\Omega}](x) \qquad \forall x \in \mathbb{R}^2 \setminus \Omega$

Helmholtz Green's function

$$\mathcal{S}[\sigma](x) = \int_{\partial\Omega} G(x,y)\sigma(y)ds(y) \qquad \textit{"Single layer potential"}$$

$$\mathcal{D}[\sigma](x) = \int_{\partial\Omega} \frac{\partial G(x,y)}{\partial n_y}\sigma(y)ds(y) \qquad \textit{"Double layer potential"}$$

Hence, $\frac{\partial u^{\text{scat}}}{\partial n}\Big|_{\partial\Omega} = \mathcal{S}^{-1}\left(\mathcal{D} - \frac{1}{2}I\right) u^{\text{scat}}|_{\partial\Omega}$ and so: $T_{\text{ext}} = \mathcal{S}^{-1}(\mathcal{D} - \frac{1}{2}I)$

Exterior problem
Boundary integral formulation and `chunkie`

`chunkie` provides convenient abstractions for constructing layer potentials and discretizing boundary integral equations.

https://chunkie.readthedocs.io/

Exterior problem
Boundary integral formulation and `chunkie`

`chunkie` provides convenient abstractions for constructing layer potentials and discretizing boundary integral equations.

- Define the Helmholtz single- and double-layer kernels:

```
Skern = kernel('helmholtz', 's', k);
Dkern = kernel('helmholtz', 'd', k);
```

https://chunkie.readthedocs.io/

Exterior problem
Boundary integral formulation and `chunkie`

`chunkie` provides convenient abstractions for constructing layer potentials and discretizing boundary integral equations.

- Define the Helmholtz single- and double-layer kernels:

```
Skern = kernel('helmholtz', 's', k);
Dkern = kernel('helmholtz', 'd', k);
```

- Construct matrices which discretize the layer potentials on the boundary:

```
S = chunkermat(Omega, Skern);
D = chunkermat(Omega, Dkern);
I = eye(Omega.npt);
```

https://chunkie.readthedocs.io/

Exterior problem
Boundary integral formulation and `chunkie`

`chunkie` provides convenient abstractions for constructing layer potentials and discretizing boundary integral equations.

- Define the Helmholtz single- and double-layer kernels:

```
Skern = kernel('helmholtz', 's', k);
Dkern = kernel('helmholtz', 'd', k);
```

- Construct matrices which discretize the layer potentials on the boundary:

```
S = chunkermat(Omega, Skern);
D = chunkermat(Omega, Dkern);
I = eye(Omega.npt);
```

- The exterior DtN map is then given by linear algebra:

```
DtN_ext = S \ (D - I/2);
```

https://chunkie.readthedocs.io/

Interior problem
Domain decomposition and `surfacefun`

$$\begin{aligned} \Delta_\Gamma u(x) + k^2(1 - q(x))u(x) &= 0 \qquad \text{in } \Omega \\ u(x) &= g(x) \quad \text{on } \partial\Omega \end{aligned}$$

Interior problem
Domain decomposition and `surfacefun`

$$\begin{aligned} \Delta_\Gamma u(x) + k^2(1 - q(x))u(x) &= 0 \qquad \text{in } \Omega \\ u(x) &= g(x) \quad \text{on } \partial\Omega \end{aligned}$$

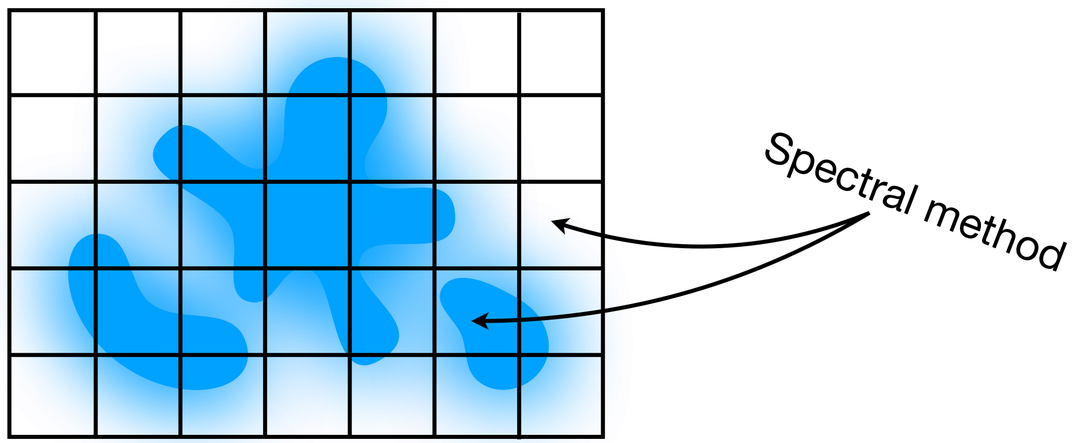
As we have a variable-coefficient problem on a surface, a PDE-based discretization is a good choice.

Interior problem
Domain decomposition and `surfacefun`

$$\begin{aligned} \Delta_\Gamma u(x) + k^2(1 - q(x))u(x) &= 0 \qquad \text{in } \Omega \\ u(x) &= g(x) \quad \text{on } \partial\Omega \end{aligned}$$

As we have a variable-coefficient problem on a surface, a PDE-based discretization is a good choice.

We will use a multidomain spectral method for discretization.

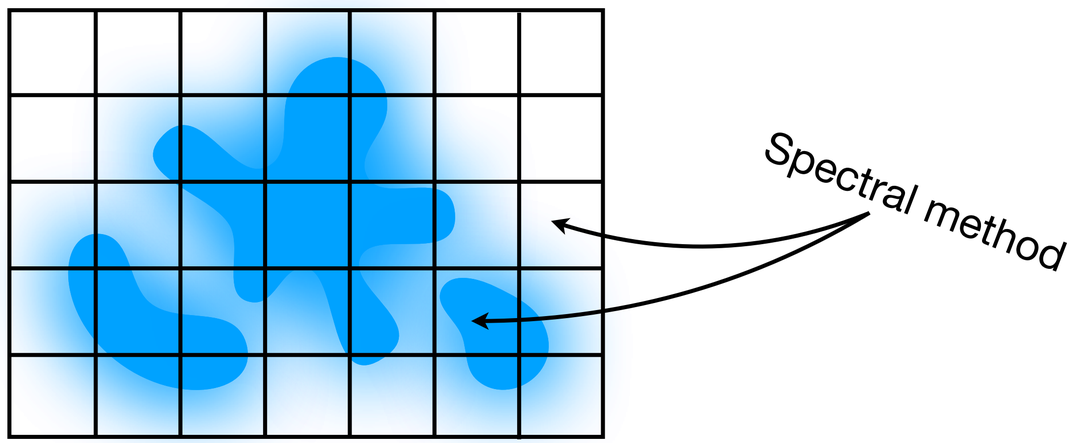


Interior problem
Domain decomposition and `surfacefun`

$$\begin{aligned} \Delta_\Gamma u(x) + k^2(1 - q(x))u(x) &= 0 \qquad \text{in } \Omega \\ u(x) &= g(x) \quad \text{on } \partial\Omega \end{aligned}$$

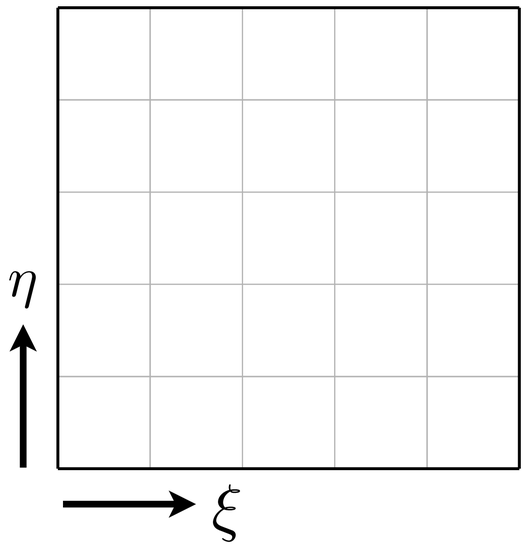
As we have a variable-coefficient problem on a surface, a PDE-based discretization is a good choice.

We will use a multidomain spectral method for discretization.

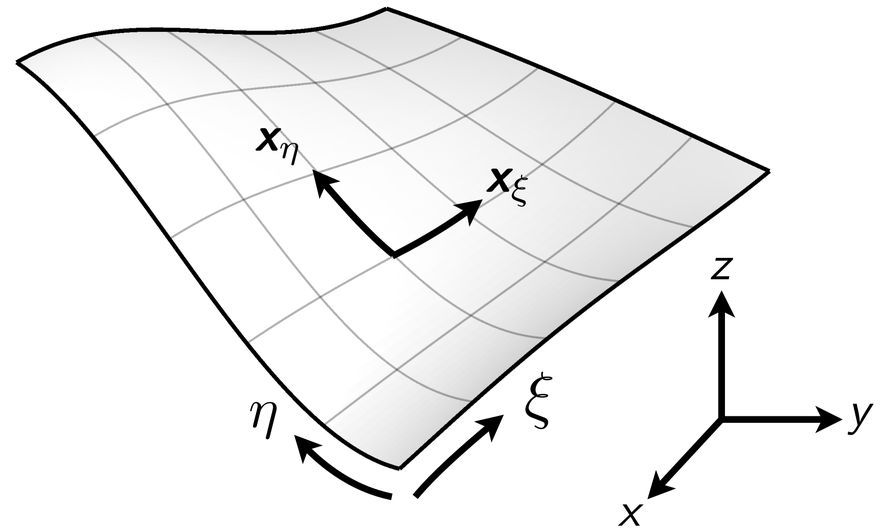


`surfacefun` provides abstractions for computing with functions on surfaces, along with a fast direct solver for surface PDEs using domain decomposition.

Interior problem
Differential operators



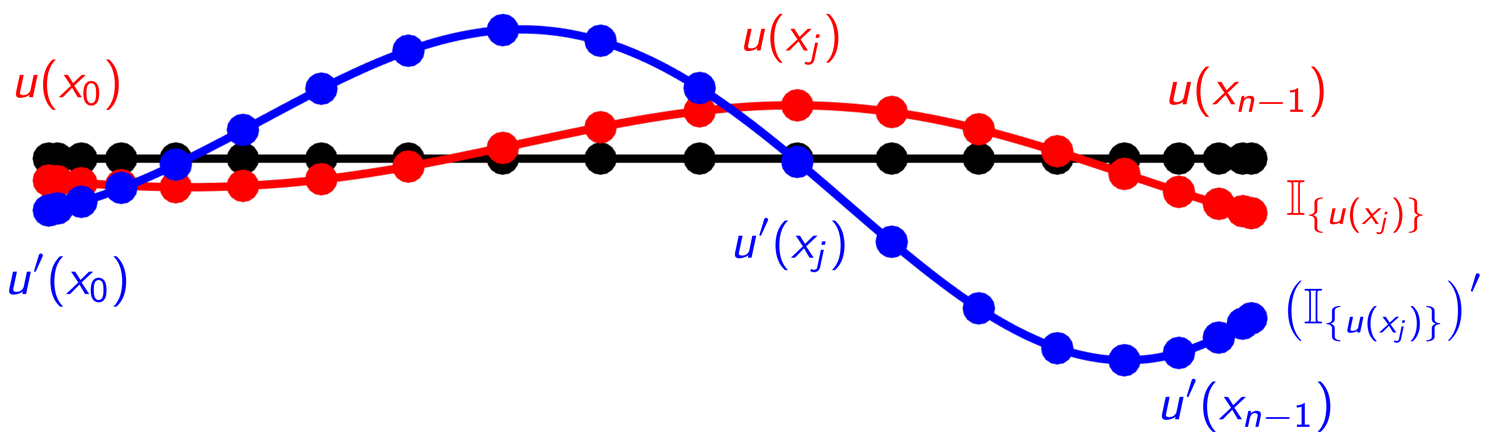
$$\boldsymbol{x}(\xi,\eta) = \begin{bmatrix} x(\xi,\eta) \\ y(\xi,\eta) \\ z(\xi,\eta) \end{bmatrix} : \mathbb{R}^2 \to \mathbb{R}^3$$



- Metric tensor $g = \begin{bmatrix} \boldsymbol{x}_\xi \cdot \boldsymbol{x}_\xi & \boldsymbol{x}_\xi \cdot \boldsymbol{x}_\eta \\ \boldsymbol{x}_\eta \cdot \boldsymbol{x}_\xi & \boldsymbol{x}_\eta \cdot \boldsymbol{x}_\eta \end{bmatrix}$ encodes how lengths and angles change along surface.
- Surface gradient: $\nabla_\Gamma u = [\boldsymbol{x}_\xi \;\; \boldsymbol{x}_\eta] \; g^{-1} \nabla_{\xi\eta} u$ $\qquad \partial_x^\Gamma = \boldsymbol{e}_x \cdot \nabla_\Gamma$
- Surface divergence: $\nabla_\Gamma \cdot \boldsymbol{u} = \dfrac{1}{\sqrt{\det g}} \nabla_{\xi\eta} \cdot \left(\sqrt{\det g}\, \boldsymbol{u} \right)$ $\qquad \partial_y^\Gamma = \boldsymbol{e}_y \cdot \nabla_\Gamma$
- Laplace–Beltrami: $\Delta_\Gamma u = \nabla_\Gamma \cdot \nabla_\Gamma u = \dfrac{1}{\sqrt{\det g}} \nabla_{\xi\eta} \cdot \left(\sqrt{\det g}\, g^{-1} \nabla_{\xi\eta} u \right)$ $\qquad \partial_z^\Gamma = \boldsymbol{e}_z \cdot \nabla_\Gamma$

Interior problem
Spectral collocation

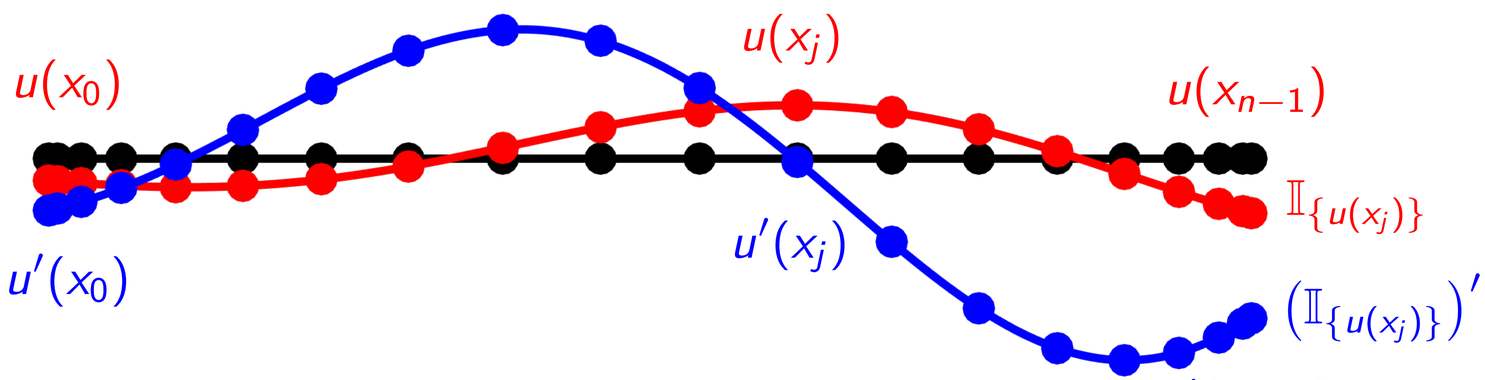
- Function values stored at Chebyshev nodes.
- Derivatives and metric information (e.g. Jacobian) computed via spectral differentiation.



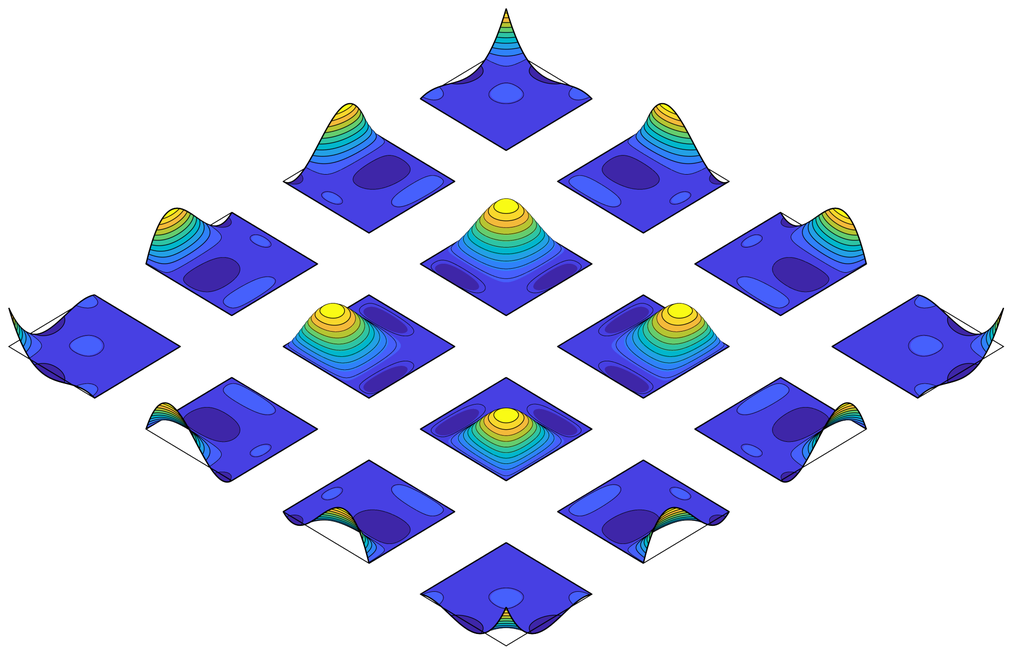
$$D\,u_j \mapsto u_j'$$

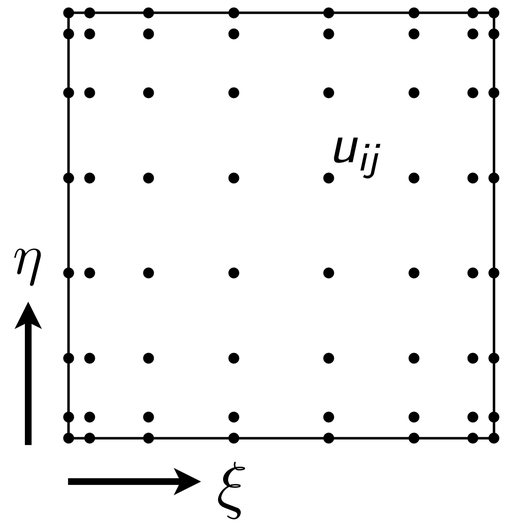
Interior problem
Spectral collocation

- Function values stored at Chebyshev nodes.
- Derivatives and metric information (e.g. Jacobian) computed via spectral differentiation.



$$D\,u_j \mapsto u_j'$$

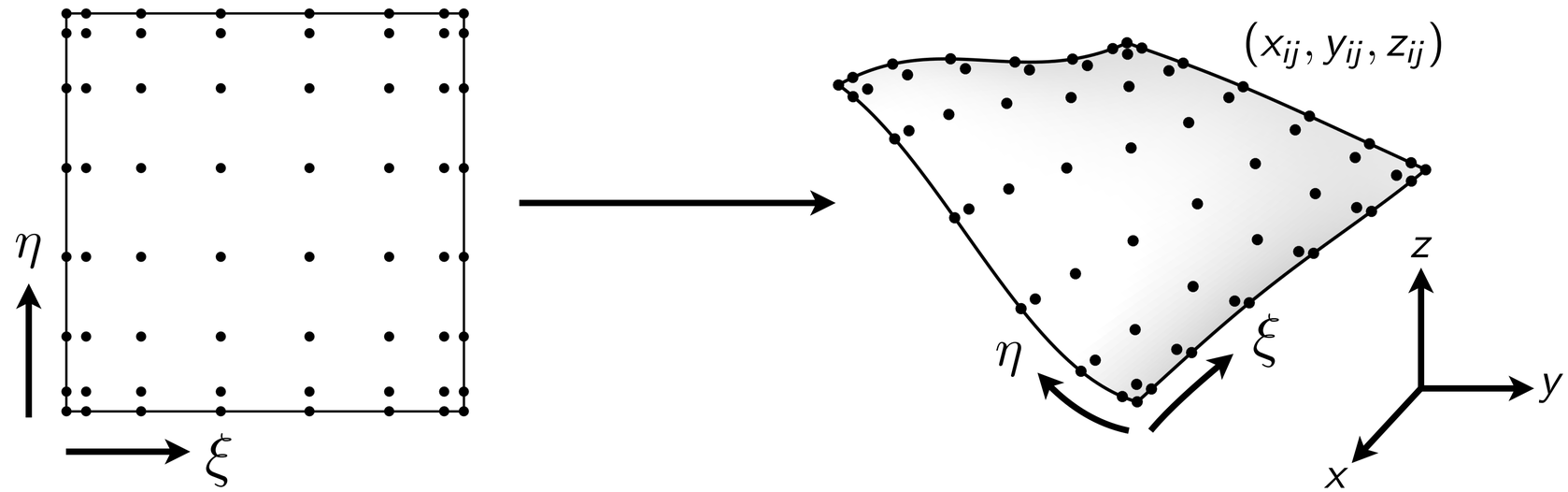




$$(D \otimes I)\,u_{ij} \mapsto \partial u_{ij}/\partial \xi$$
$$(I \otimes D)\,u_{ij} \mapsto \partial u_{ij}/\partial \eta$$

Interior problem
Spectral collocation

- PDE is discretized through spectral differentiation and pointwise multiplication.

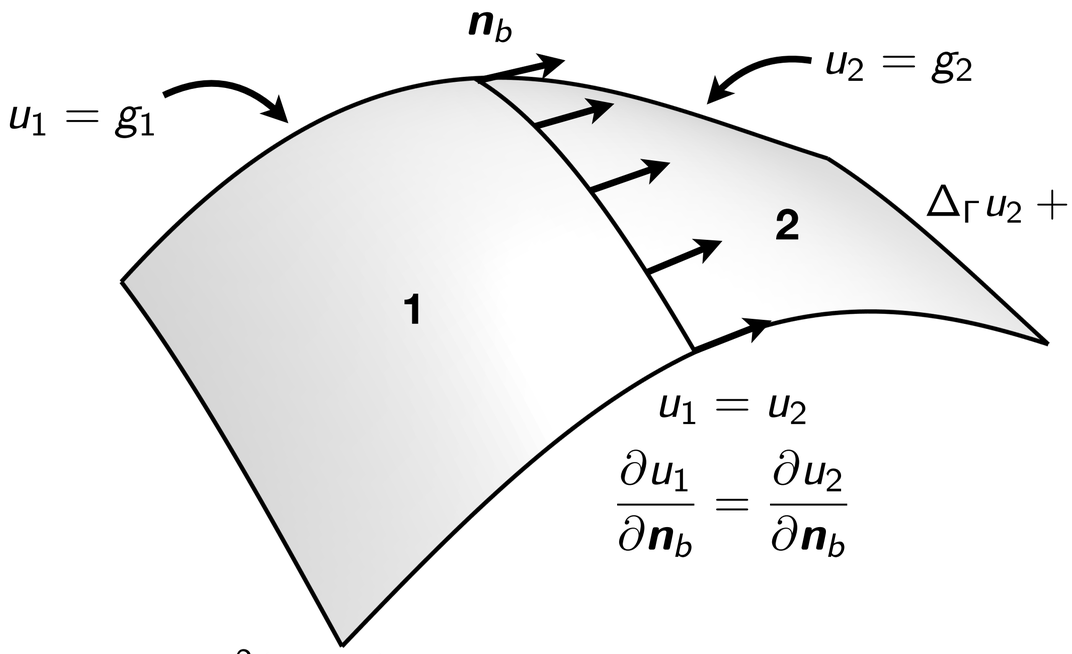


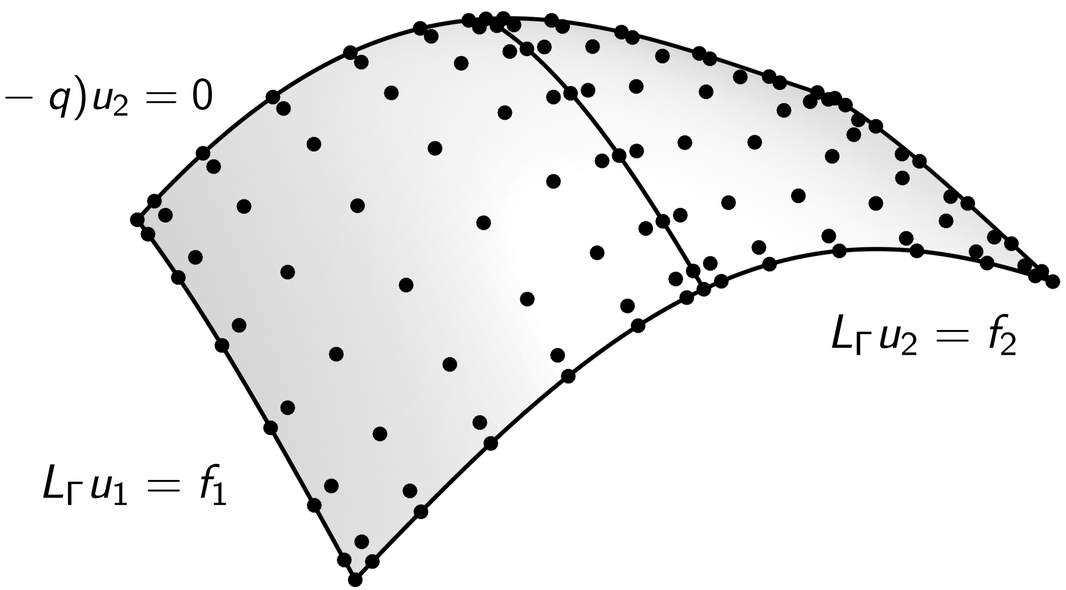
- For example, the discrete tangential x-derivative operator is:

$$D_x = \begin{bmatrix} \ddots & & \\ & (\xi_x)_{ij} & \\ & & \ddots \end{bmatrix} (D \otimes I) \; + \; \begin{bmatrix} \ddots & & \\ & (\eta_x)_{ij} & \\ & & \ddots \end{bmatrix} (I \otimes D)$$

- In general, the PDE results in a $(p+1)^2 \times (p+1)^2$ linear system, $L_\Gamma u = f$, which we can invert directly.

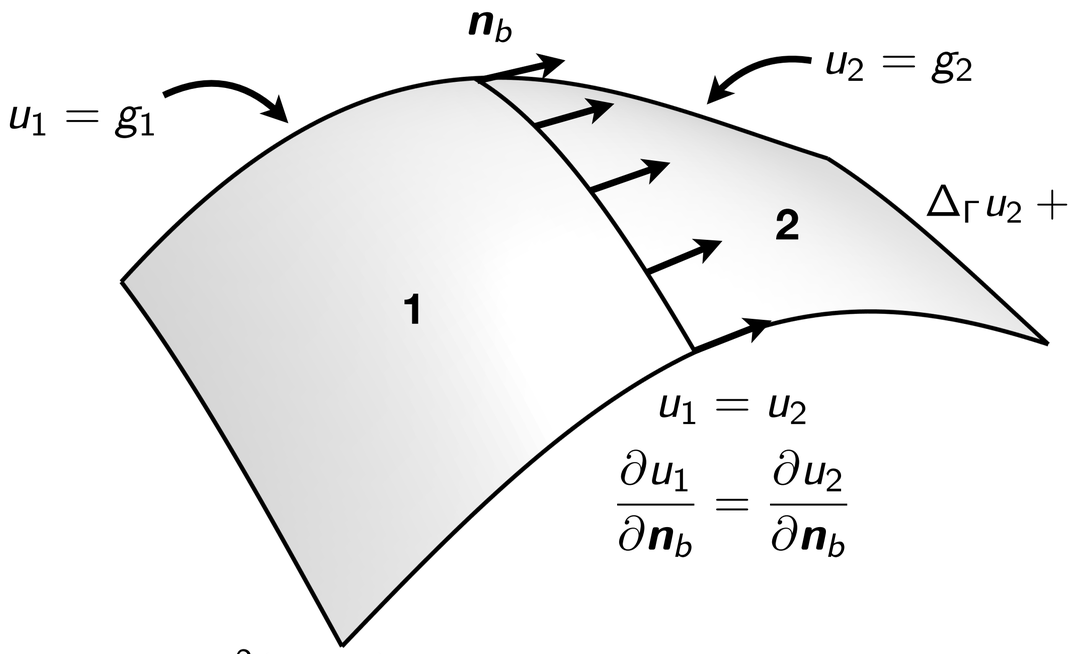
Interior problem
Domain decomposition

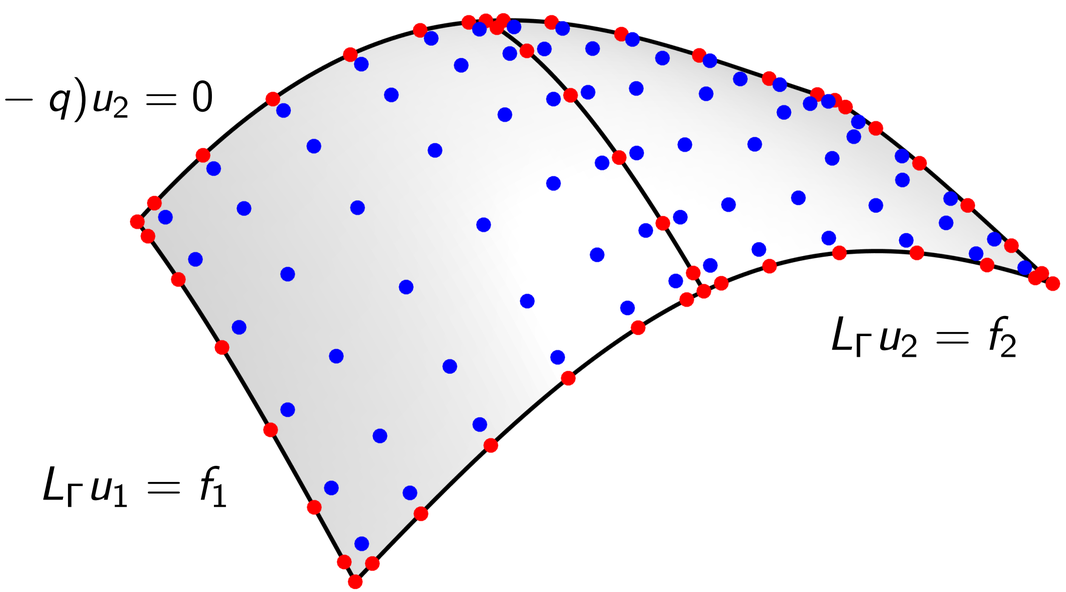




[F., 2022]

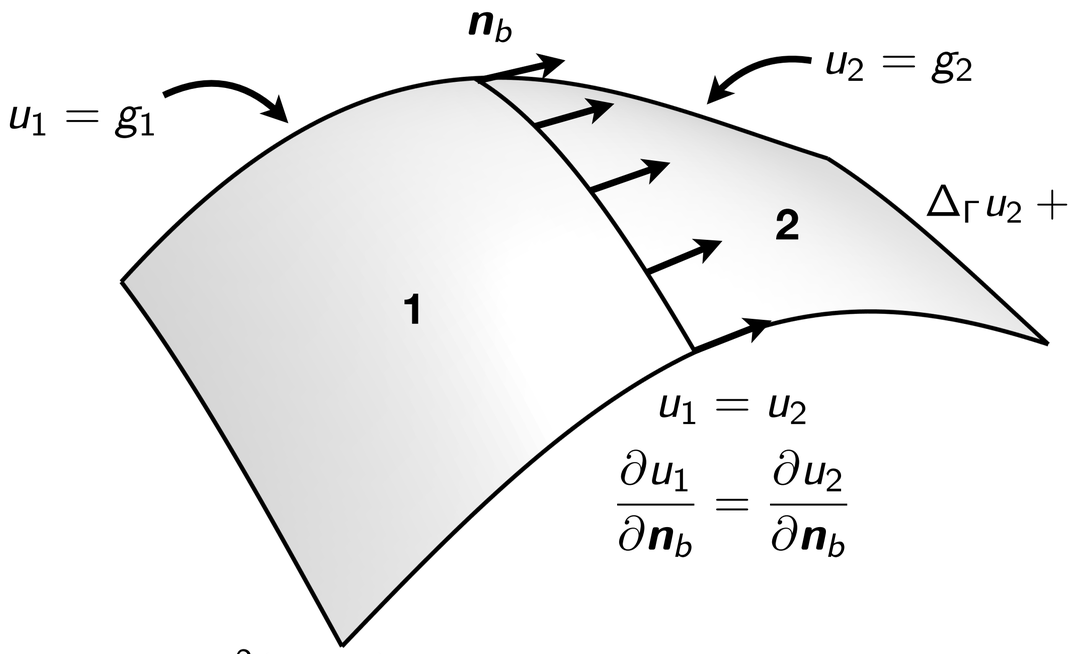
Interior problem
Domain decomposition

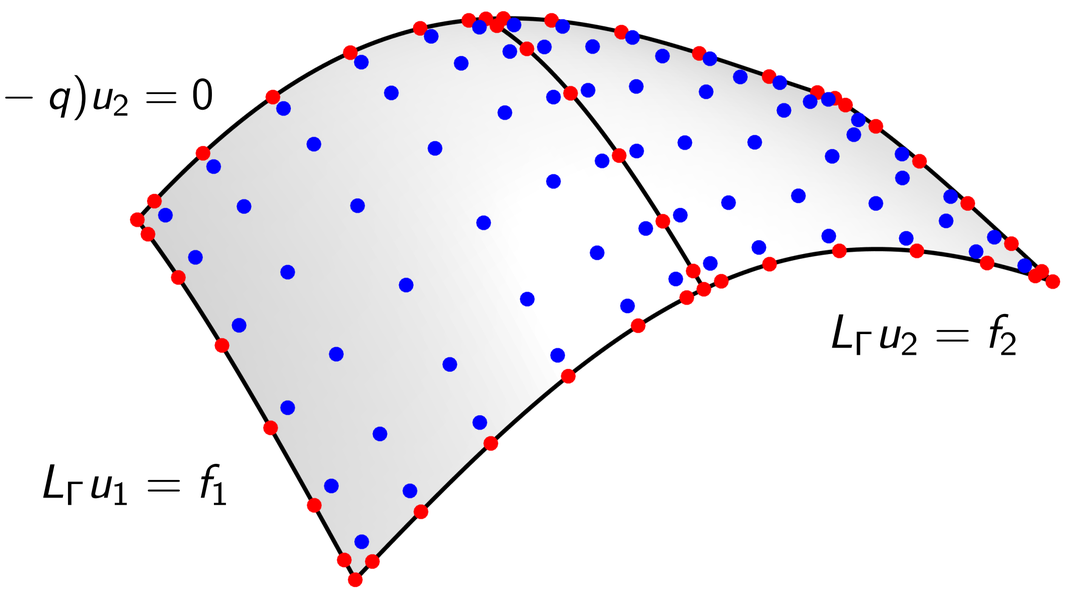




[F., 2022]

Interior problem
Domain decomposition



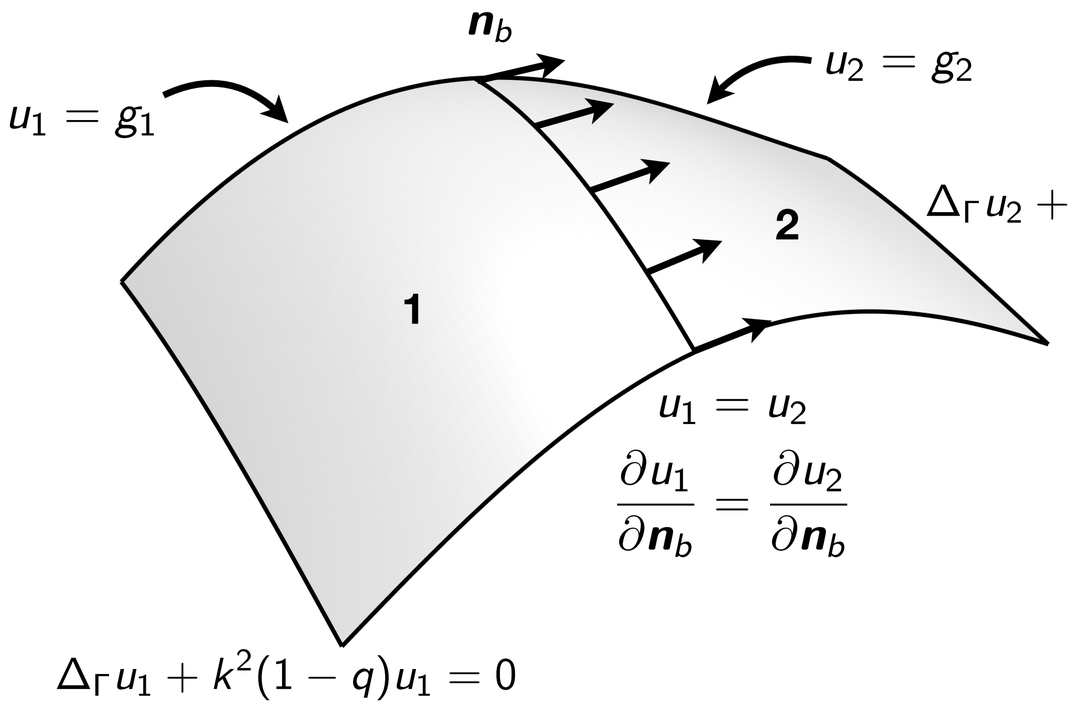


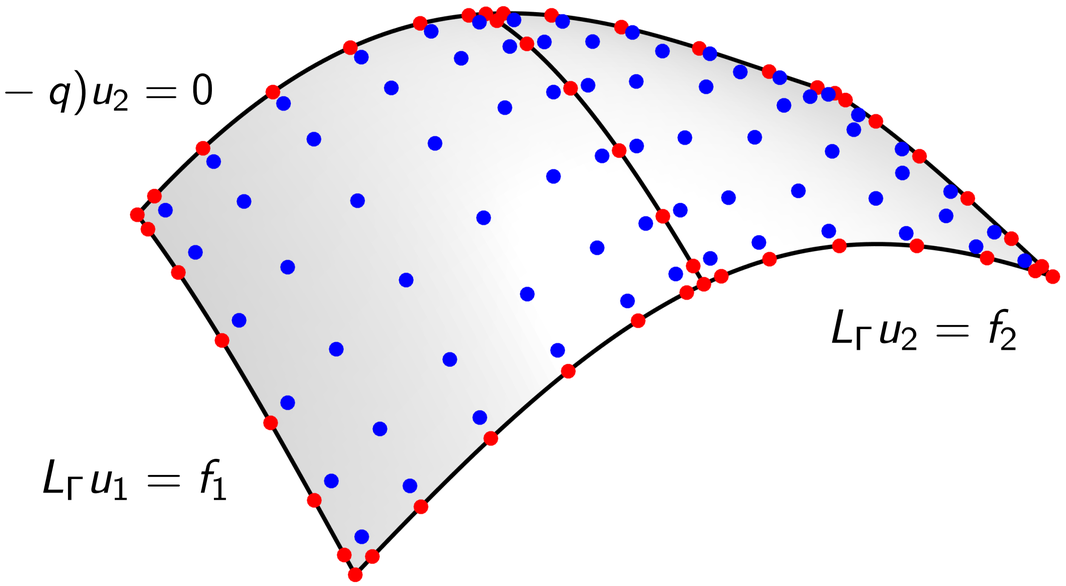
- Know how to do local solves on each element: “Solution operator”

$$S_1 \begin{bmatrix} g_1 \\ u_{\text{glue}} \end{bmatrix} \mapsto u_1 \qquad S_2 \begin{bmatrix} g_2 \\ u_{\text{glue}} \end{bmatrix} \mapsto u_2$$

[F., 2022]

Interior problem
Domain decomposition

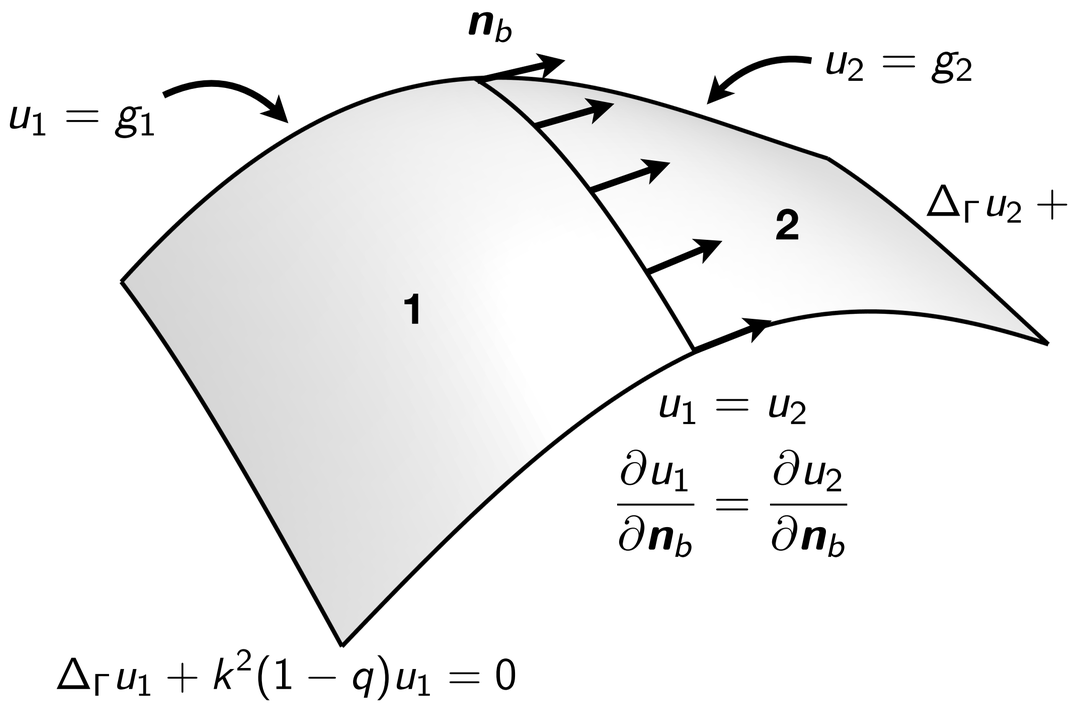


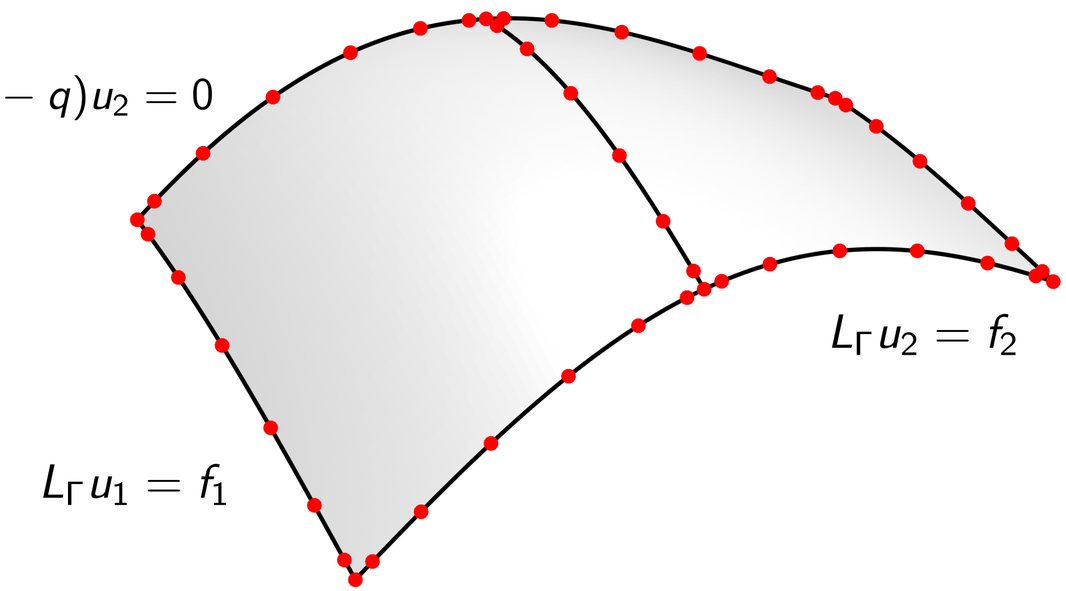


- Know how to do local solves on each element: “Solution operator” $S_1 \begin{bmatrix} g_1 \\ u_{\text{glue}} \end{bmatrix} \mapsto u_1$ $S_2 \begin{bmatrix} g_2 \\ u_{\text{glue}} \end{bmatrix} \mapsto u_2$
- Know how information flows out of each element: “Dirichlet-to-Neumann map” $DtN_1 \begin{bmatrix} g_1 \\ u_{\text{glue}} \end{bmatrix} \mapsto \frac{\partial u_1}{\partial n_b}$ $DtN_2 \begin{bmatrix} g_2 \\ u_{\text{glue}} \end{bmatrix} \mapsto \frac{\partial u_2}{\partial n_b}$

[F., 2022]

Interior problem
Domain decomposition





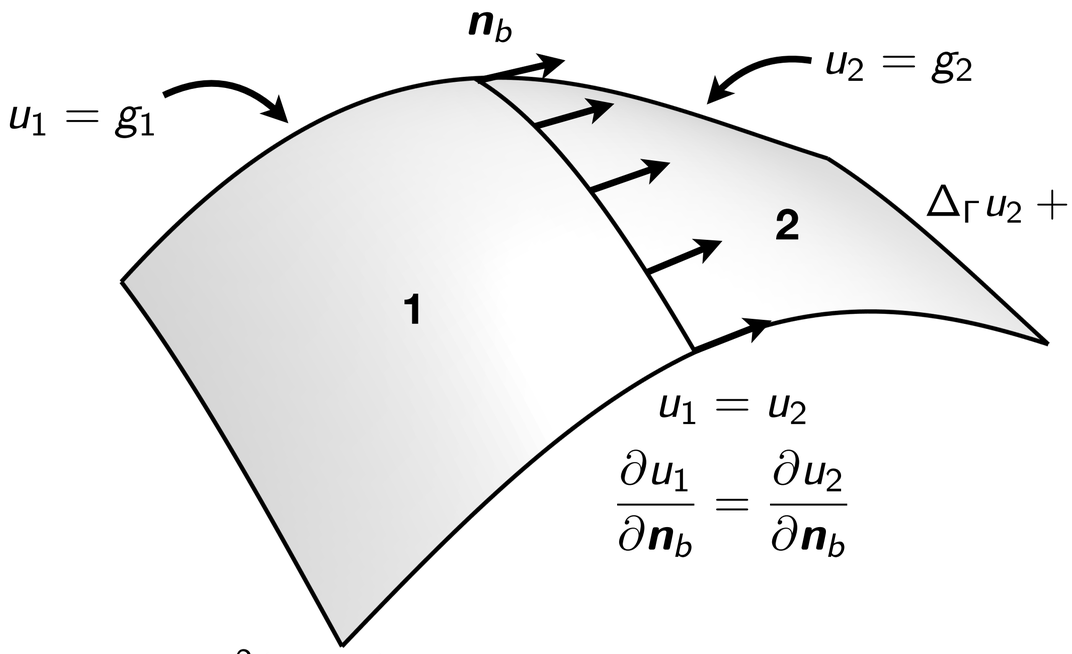
- Know how to do local solves on each element: “Solution operator”
- Know how information flows out of each element: “Dirichlet-to-Neumann map”

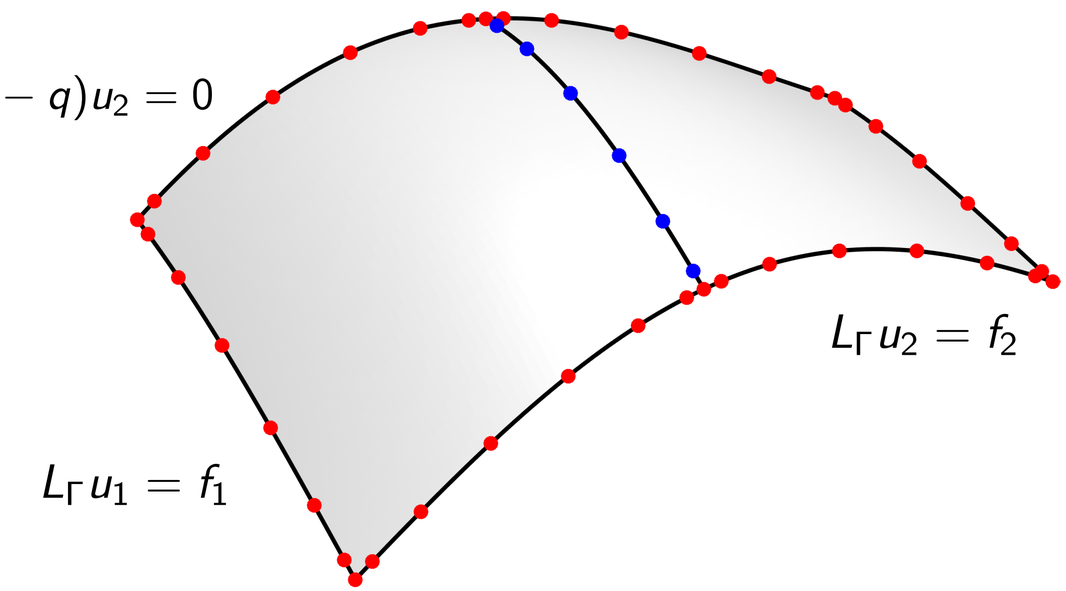
$$S_1 \begin{bmatrix} g_1 \\ u_{\text{glue}} \end{bmatrix} \mapsto u_1 \qquad S_2 \begin{bmatrix} g_2 \\ u_{\text{glue}} \end{bmatrix} \mapsto u_2$$

$$DtN_1 \begin{bmatrix} g_1 \\ u_{\text{glue}} \end{bmatrix} \mapsto \frac{\partial u_1}{\partial n_b} \qquad DtN_2 \begin{bmatrix} g_2 \\ u_{\text{glue}} \end{bmatrix} \mapsto \frac{\partial u_2}{\partial n_b}$$

[F., 2022]

Interior problem
Domain decomposition





- Know how to do local solves on each element: “Solution operator”
- Know how information flows out of each element: “Dirichlet-to-Neumann map”
- Take Schur complement to eliminate interior degrees of freedom:

$$S_1 \begin{bmatrix} g_1 \\ u_{\text{glue}} \end{bmatrix} \mapsto u_1 \qquad S_2 \begin{bmatrix} g_2 \\ u_{\text{glue}} \end{bmatrix} \mapsto u_2$$

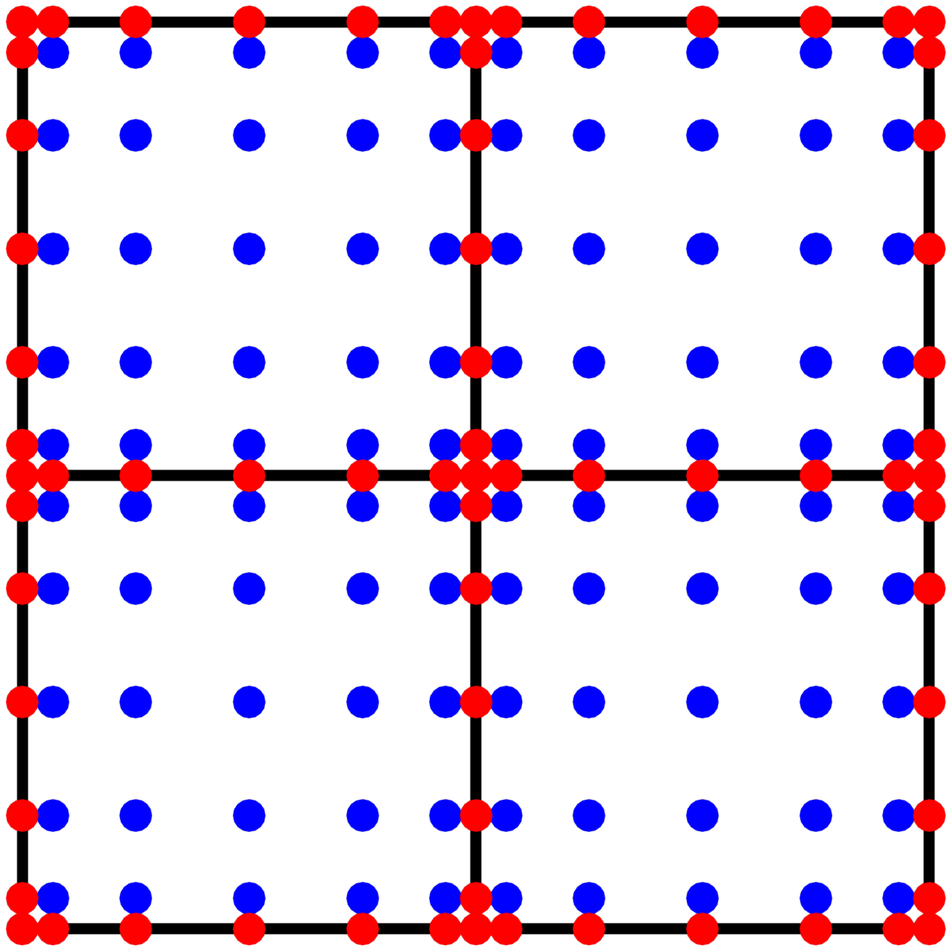
$$DtN_1 \begin{bmatrix} g_1 \\ u_{\text{glue}} \end{bmatrix} \mapsto \frac{\partial u_1}{\partial n_b} \qquad DtN_2 \begin{bmatrix} g_2 \\ u_{\text{glue}} \end{bmatrix} \mapsto \frac{\partial u_2}{\partial n_b}$$

$$S_{\text{glue}} = -\left(DtN_1^{\text{glue}} + DtN_2^{\text{glue}}\right)^{-1} \begin{bmatrix} DtN_1^{\text{glue},1} & DtN_2^{\text{glue},2} \end{bmatrix} \qquad S_{\text{glue}} \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} = u_{\text{glue}}$$

[F., 2022]

A fast direct solver on surfaces
Hierarchical Poincaré–Steklov method

Key idea: Recursively glue elements together in a hierarchy (cf. nested dissection)



[F., 2024]

[Martinsson, 2013]

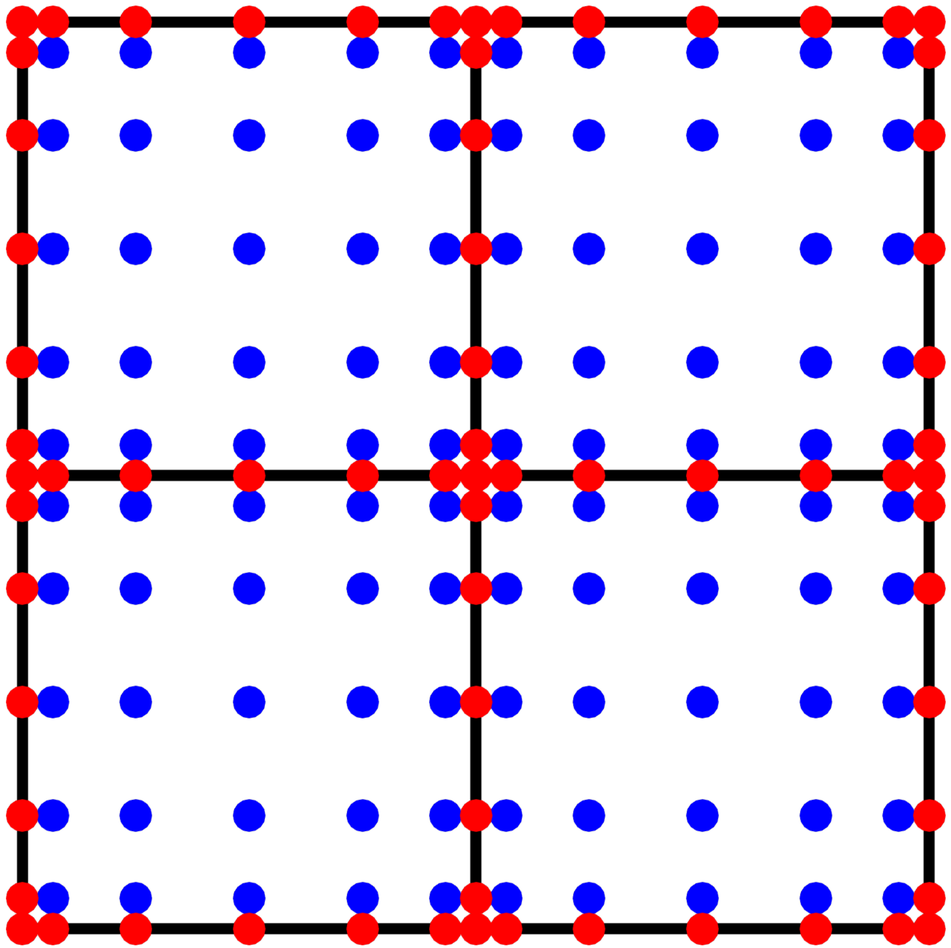
[Gillman & Martinsson, 2014]

A fast direct solver on surfaces

Hierarchical Poincaré–Steklov method

<u>Key idea</u>: Recursively glue elements together in a hierarchy (cf. nested dissection)

Build element operators $\mathcal{O}(Np^6)$



[F., 2024]

[Martinsson, 2013]

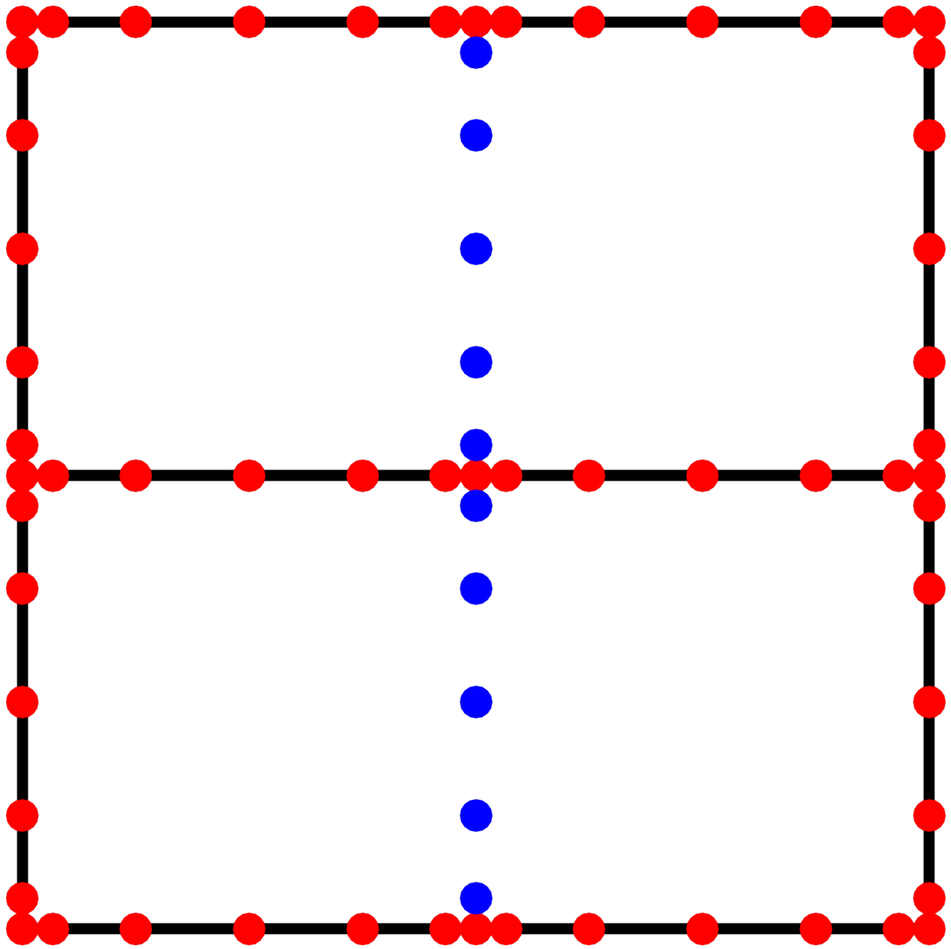
[Gillman & Martinsson, 2014]

A fast direct solver on surfaces

Hierarchical Poincaré–Steklov method

Key idea: Recursively glue elements together in a hierarchy (cf. nested dissection)

Merge operators $\mathcal{O}(N^{3/2})$



[F., 2024]

[Martinsson, 2013]

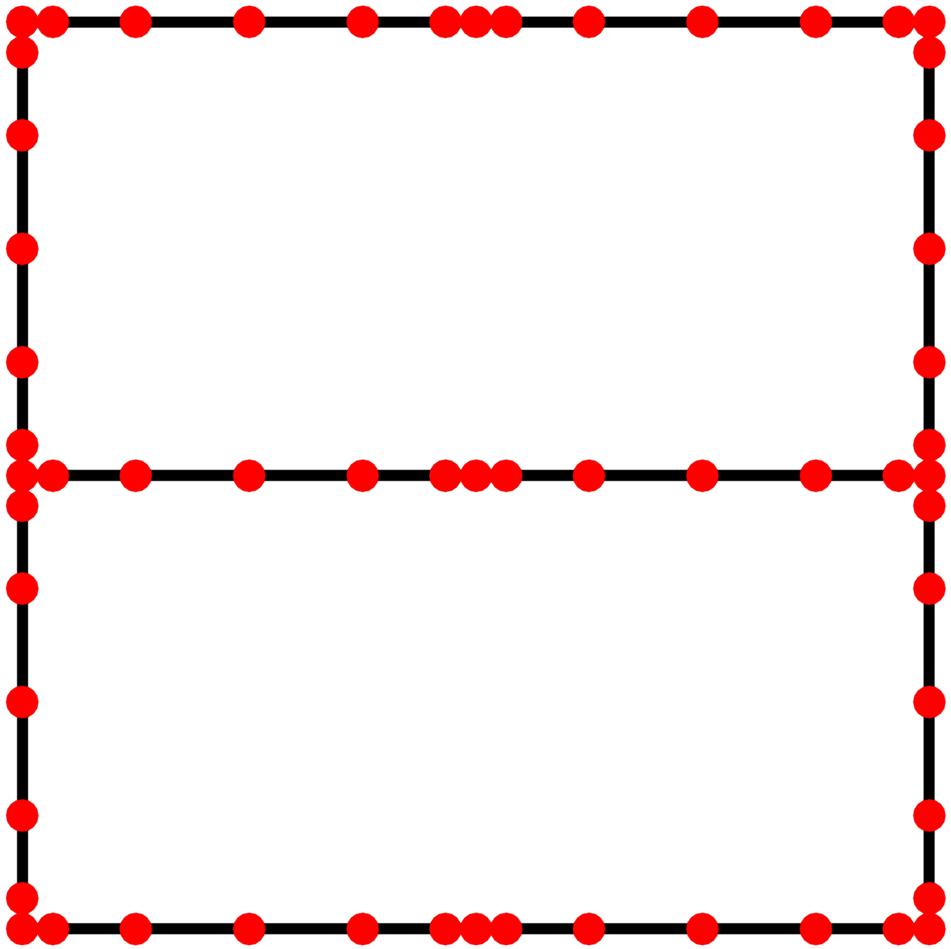
[Gillman & Martinsson, 2014]

A fast direct solver on surfaces

Hierarchical Poincaré–Steklov method

Key idea: Recursively glue elements together in a hierarchy (cf. nested dissection)

Merge operators $\mathcal{O}(N^{3/2})$



[F., 2024]

[Martinsson, 2013]

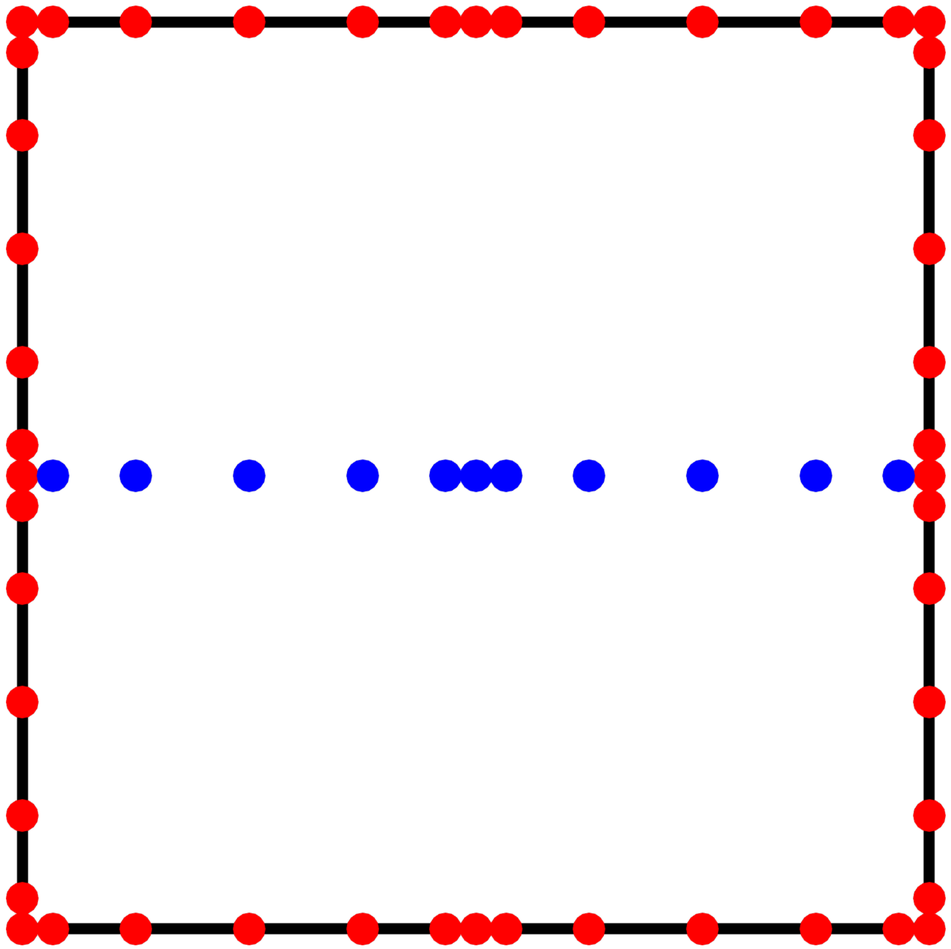
[Gillman & Martinsson, 2014]

A fast direct solver on surfaces

Hierarchical Poincaré–Steklov method

Key idea: Recursively glue elements together in a hierarchy (cf. nested dissection)

Merge operators $\mathcal{O}(N^{3/2})$



[F., 2024]

[Martinsson, 2013]

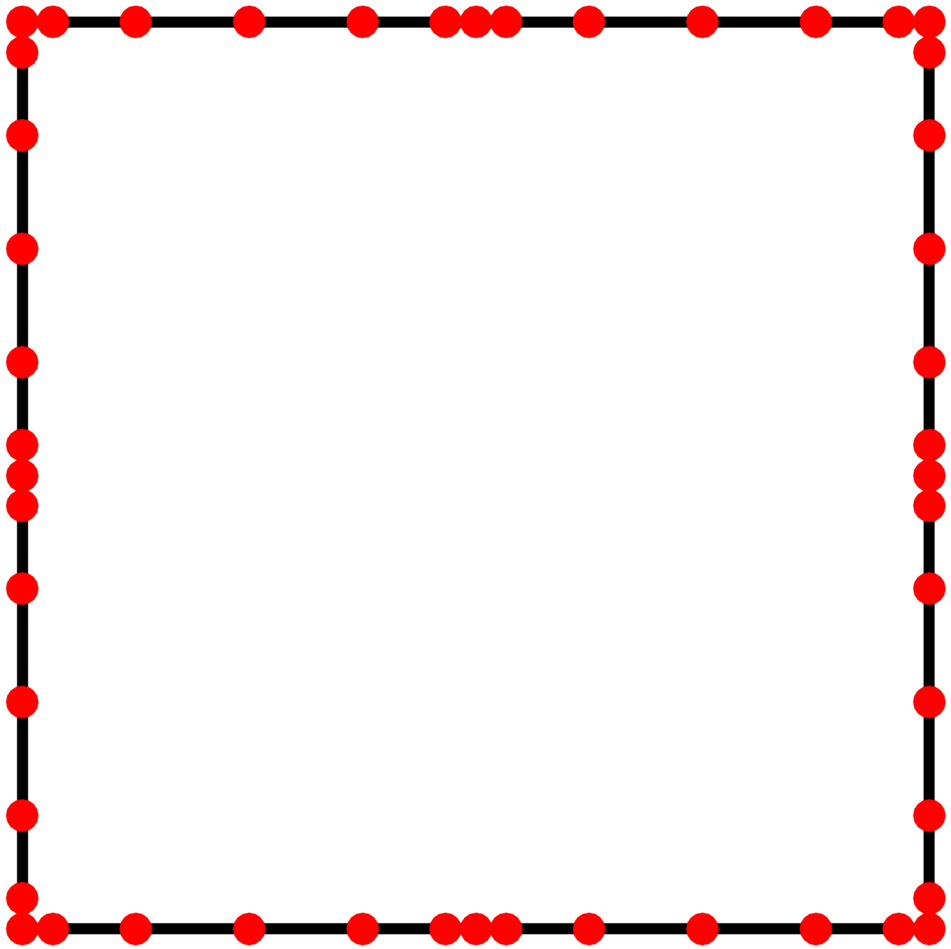
[Gillman & Martinsson, 2014]

A fast direct solver on surfaces

Hierarchical Poincaré–Steklov method

Key idea: Recursively glue elements together in a hierarchy (cf. nested dissection)

Merge operators $\mathcal{O}(N^{3/2})$



[F., 2024]

[Martinsson, 2013]

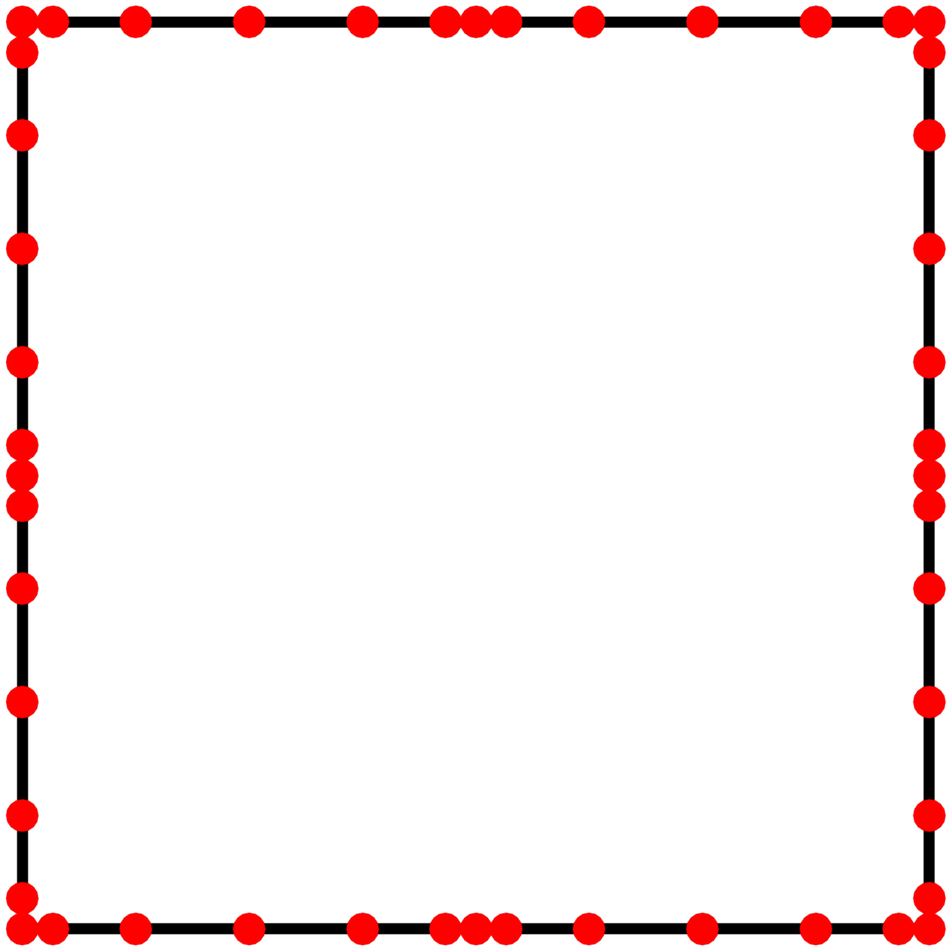
[Gillman & Martinsson, 2014]

A fast direct solver on surfaces

Hierarchical Poincaré–Steklov method

Key idea: Recursively glue elements together in a hierarchy (cf. nested dissection)

Apply solution operators $\mathcal{O}(N \log N)$



[F., 2024]

[Martinsson, 2013]

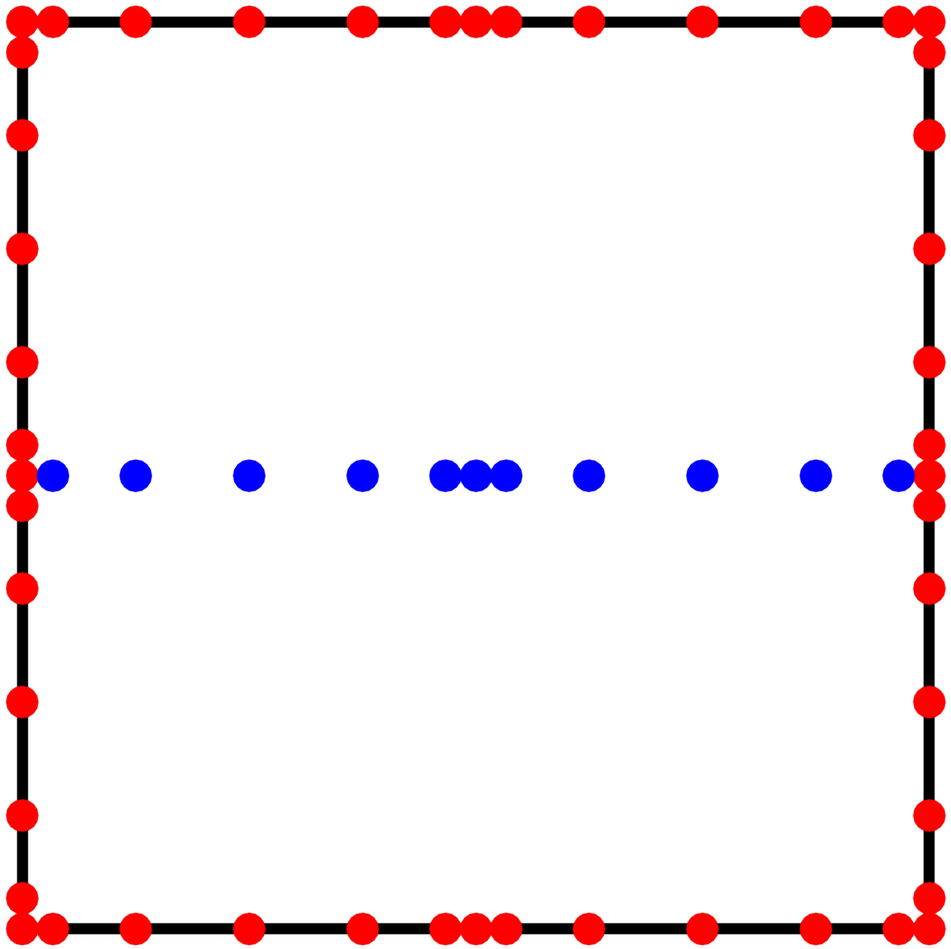
[Gillman & Martinsson, 2014]

A fast direct solver on surfaces

Hierarchical Poincaré–Steklov method

Key idea: Recursively glue elements together in a hierarchy (cf. nested dissection)

Apply solution operators $\mathcal{O}(N \log N)$



[F., 2024]

[Martinsson, 2013]

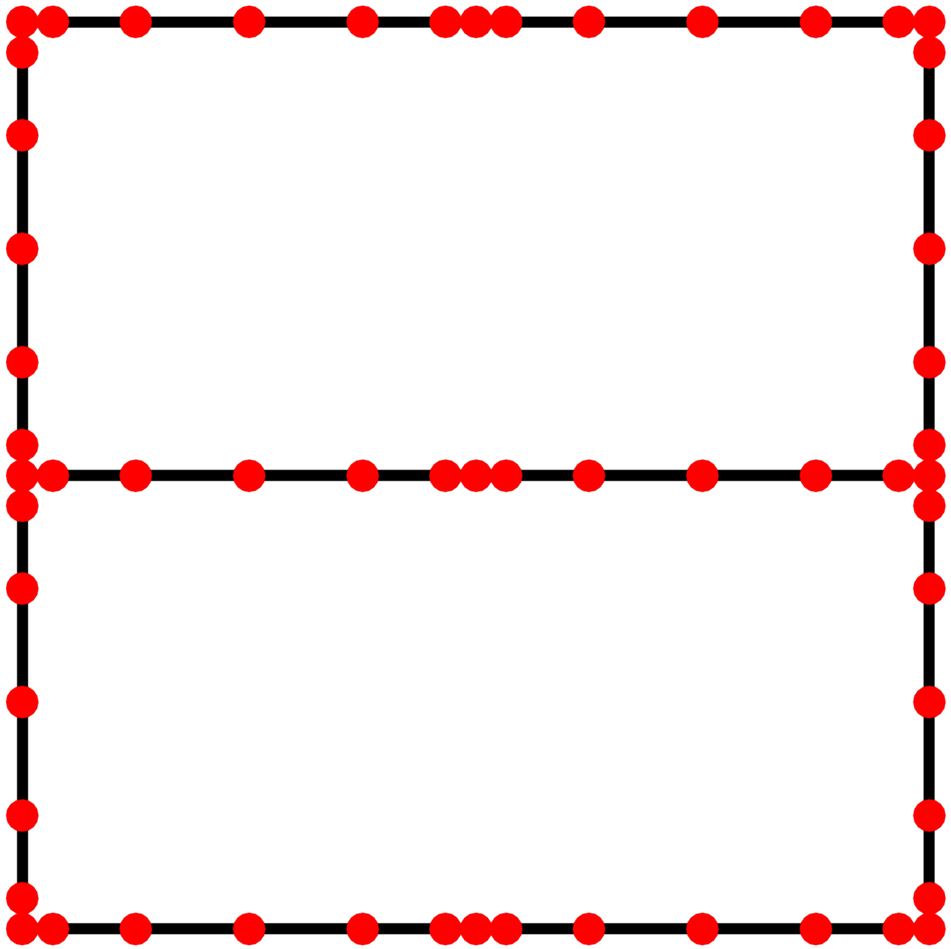
[Gillman & Martinsson, 2014]

A fast direct solver on surfaces

Hierarchical Poincaré–Steklov method

Key idea: Recursively glue elements together in a hierarchy (cf. nested dissection)

Apply solution operators $\mathcal{O}(N \log N)$



[F., 2024]

[Martinsson, 2013]

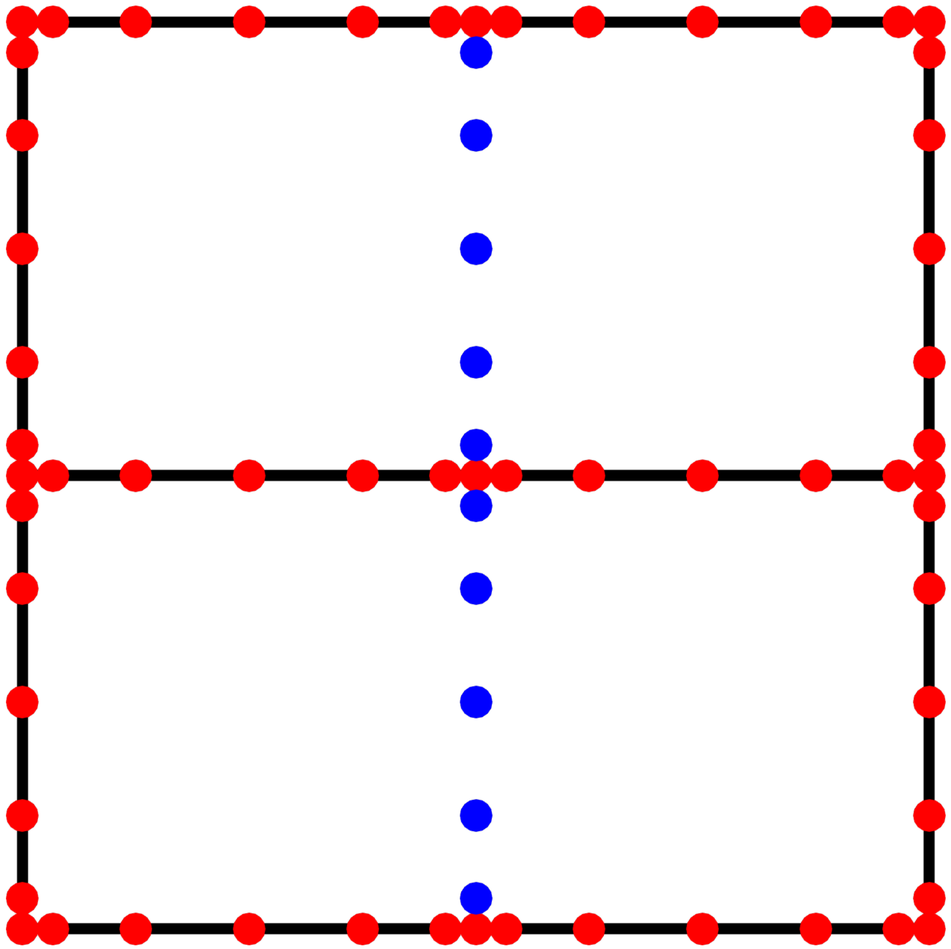
[Gillman & Martinsson, 2014]

A fast direct solver on surfaces

Hierarchical Poincaré–Steklov method

Key idea: Recursively glue elements together in a hierarchy (cf. nested dissection)

Apply solution operators $\mathcal{O}(N \log N)$



[F., 2024]

[Martinsson, 2013]

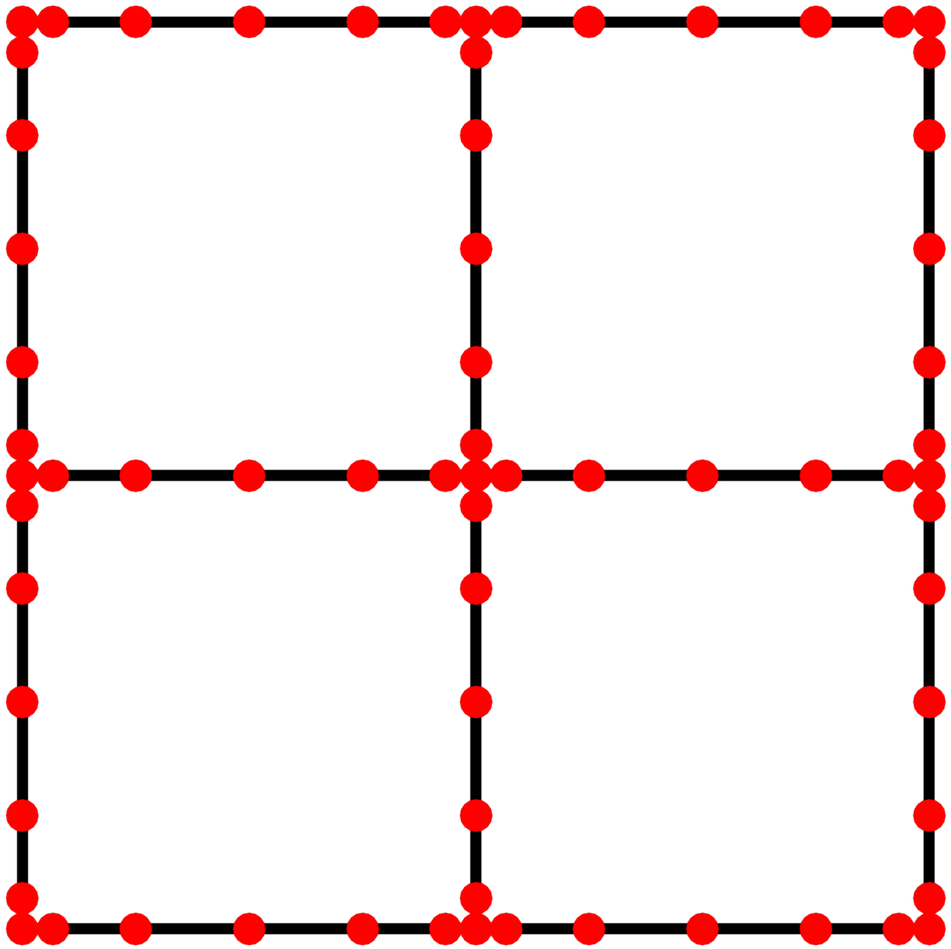
[Gillman & Martinsson, 2014]

A fast direct solver on surfaces

Hierarchical Poincaré–Steklov method

Key idea: Recursively glue elements together in a hierarchy (cf. nested dissection)

Apply solution operators $\mathcal{O}(N \log N)$



[F., 2024]

[Martinsson, 2013]

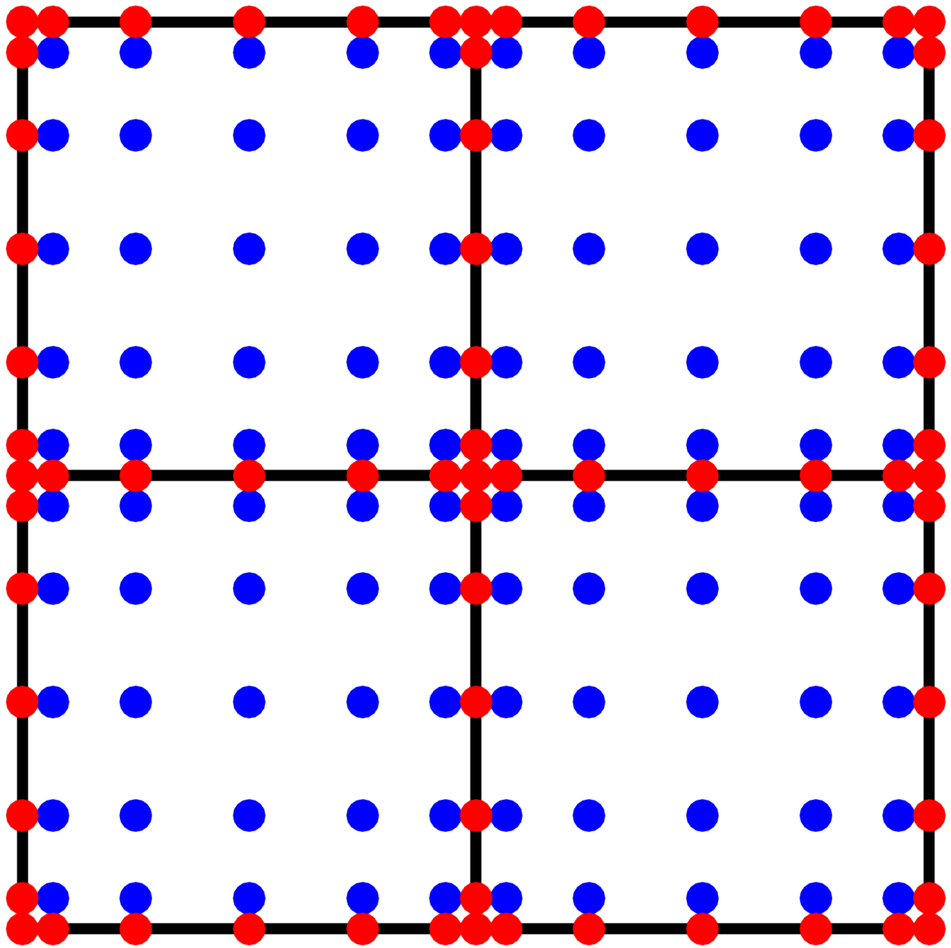
[Gillman & Martinsson, 2014]

A fast direct solver on surfaces

Hierarchical Poincaré–Steklov method

Key idea: Recursively glue elements together in a hierarchy (cf. nested dissection)

Apply solution operators $\mathcal{O}(N \log N)$



[F., 2024]

[Martinsson, 2013]

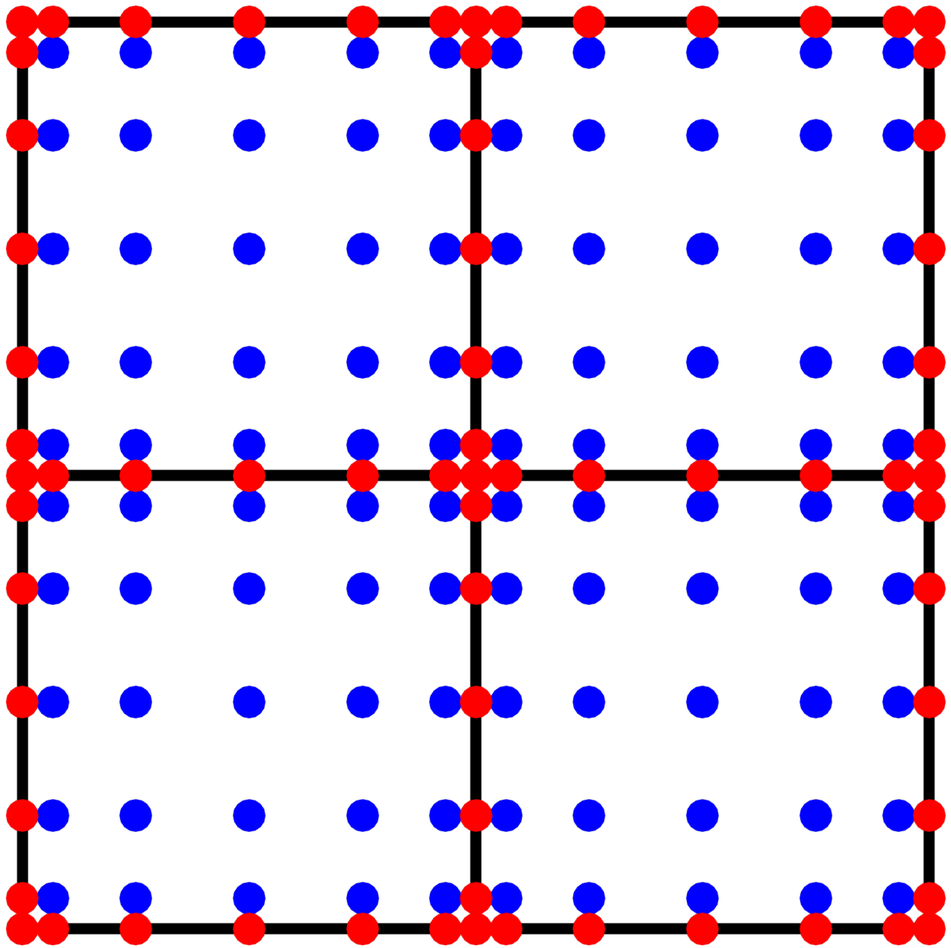
[Gillman & Martinsson, 2014]

A fast direct solver on surfaces

Hierarchical Poincaré–Steklov method

Key idea: Recursively glue elements together in a hierarchy (cf. nested dissection)

Apply solution operators $\mathcal{O}(N \log N)$



$$\underbrace{N^{3/2}}_{\text{Factorization}} + \underbrace{N \log N}_{\text{Solve}}$$

Factorization results in a hierarchy of solution operators stored in memory, so repeated solves are fast.

[F., 2024]

[Martinsson, 2013]

[Gillman & Martinsson, 2014]

Interior problem
Domain decomposition and `surfacefun`

`surfacefun` provides abstractions for computing with functions on surfaces, along with a fast direct solver for surface PDEs.

https://surfacefun.readthedocs.io/

Interior problem
Domain decomposition and `surfacefun`

`surfacefun` provides abstractions for computing with functions on surfaces, along with a fast direct solver for surface PDEs.

- Define the PDE and geometry

```
pdo = []; pdo.lap = 1; pdo.c = @(x,y,z) k^2*(1-q(x,y,z));
dom = surfacemesh(x, y, z);
```

https://surfacefun.readthedocs.io/

Interior problem
Domain decomposition and `surfacefun`

`surfacefun` provides abstractions for computing with functions on surfaces, along with a fast direct solver for surface PDEs.

- Define the PDE and geometry

```
pdo = []; pdo.lap = 1; pdo.c = @(x,y,z) k^2*(1-q(x,y,z));
dom = surfacemesh(x, y, z);
```

- Construct the fast direct solver:

```
L = surfaceop(dom, pdo, method='ItI');
```

In practice, we use “impedance-to-impedance” maps instead of Dirichlet-to-Neumann maps to avoid artificial resonances.

https://surfacefun.readthedocs.io/

Interior problem
Domain decomposition and `surfacefun`

`surfacefun` provides abstractions for computing with functions on surfaces, along with a fast direct solver for surface PDEs.

- Define the PDE and geometry

```
pdo = []; pdo.lap = 1; pdo.c = @(x,y,z) k^2*(1-q(x,y,z));
dom = surfacemesh(x, y, z);
```

- Construct the fast direct solver:

```
L = surfaceop(dom, pdo, method='ItI');
```

- The interior DtN map is then computed:

```
DtN_int = L.DtN();
```

In practice, we use “impedance-to-impedance” maps instead of Dirichlet-to-Neumann maps to avoid artificial resonances.

https://surfacefun.readthedocs.io/

Putting it all together

`chunkie` and `surfacefun`

Now that we know Dirichlet-to-Neumann operators for the interior and exterior, we can solve the coupled problem.

Demo

Conclusion

Why couple these codes together?

Conclusion
Why couple these codes together?

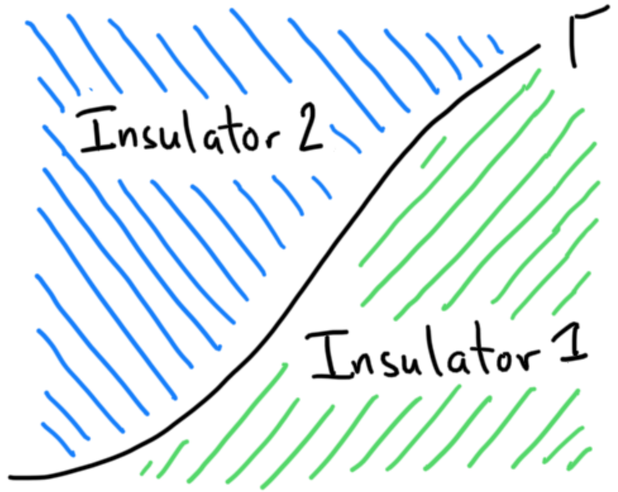
Coupling a surface PDE solver to an integral-equation-based solver is surprisingly useful.

Conclusion
Why couple these codes together?

Coupling a surface PDE solver to an integral-equation-based solver is surprisingly useful.

- The inverse surface Helmholtzian serves as an efficient preconditioner when simulating 3D topological insulators using integral equations.

  (Ongoing work with Manas Rachh, Jeremy Hoskins, & Adrianna Gillman)

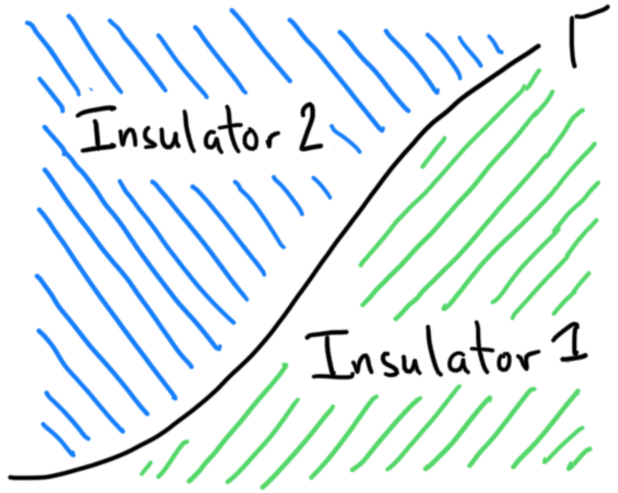


Conclusion
Why couple these codes together?

Coupling a surface PDE solver to an integral-equation-based solver is surprisingly useful.

- The inverse surface Helmholtzian serves as an efficient preconditioner when simulating 3D topological insulators using integral equations.

  (Ongoing work with Manas Rachh, Jeremy Hoskins, & Adrianna Gillman)



- The inverse surface Laplacian plays in important role in generalized Debye representations for Maxwell's equations.

  (Ongoing work with Mike O'Neil, Francisco Rilloraza, & Manas Rachh)

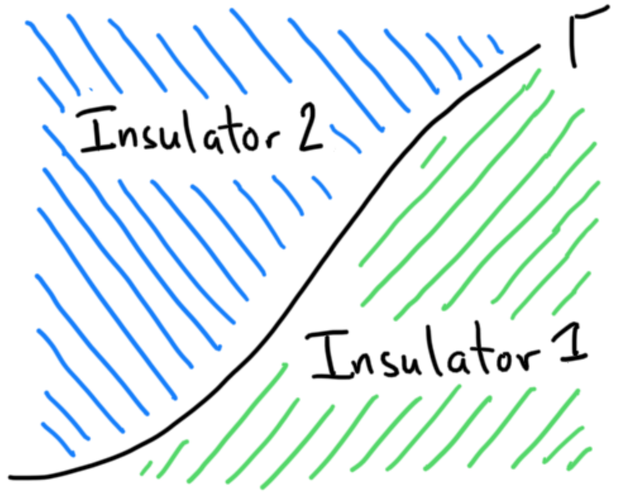
$$S_k[n \times \nabla_\Gamma \Delta_\Gamma^{-1} \sigma]$$

Conclusion
Why couple these codes together?

Coupling a surface PDE solver to an integral-equation-based solver is surprisingly useful.

- The inverse surface Helmholtzian serves as an efficient preconditioner when simulating 3D topological insulators using integral equations.

  (Ongoing work with Manas Rachh, Jeremy Hoskins, & Adrianna Gillman)



- The inverse surface Laplacian plays in important role in generalized Debye representations for Maxwell's equations.

  (Ongoing work with Mike O'Neil, Francisco Rilloraza, & Manas Rachh)

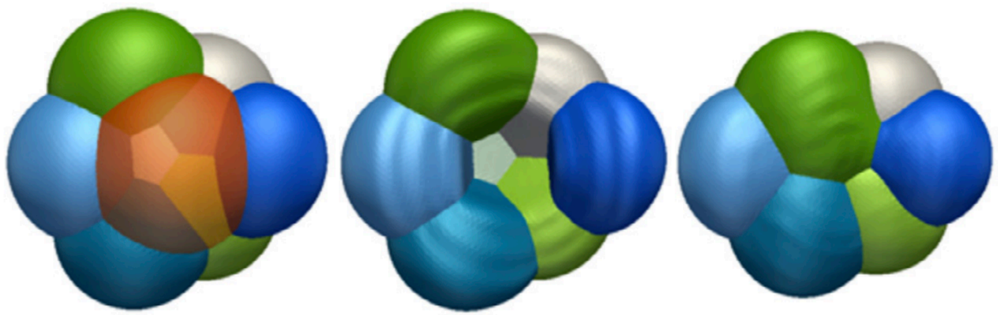
$$S_k[n \times \nabla_\Gamma \Delta_\Gamma^{-1} \sigma]$$

https://surfacefun.readthedocs.io/
https://github.com/fastalgorithms/chunkie

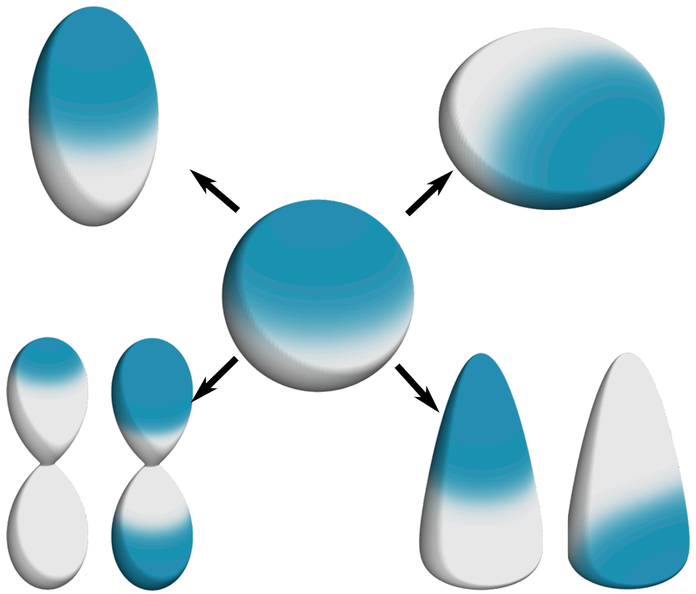
Introduction
Surface PDEs

Surface-bound phenomena arise in many applications.

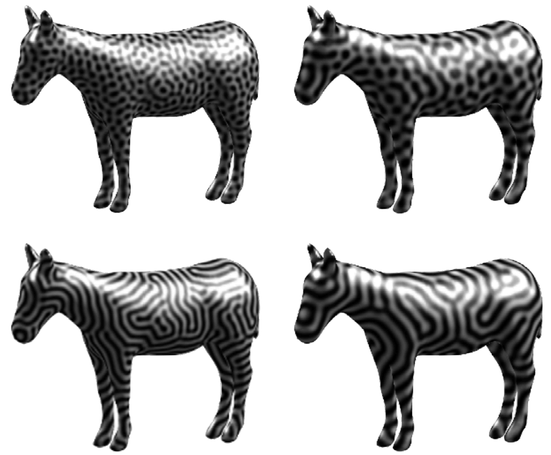
Thin-film hydrodynamics [Saye, 2016]



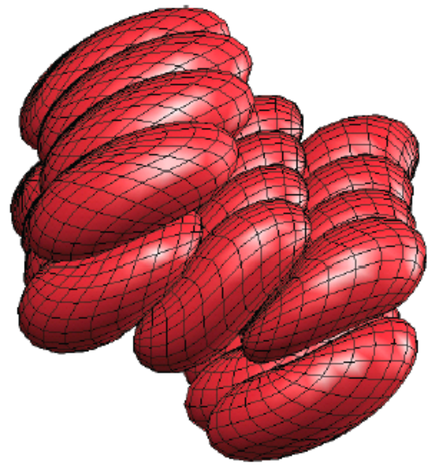
Cell polarization [Miller, F., Muratov, Greengard, & Shvartsman, 2022]



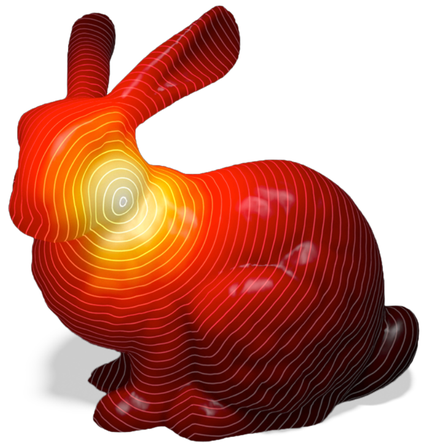
Pattern formation [Jeong, 2017]



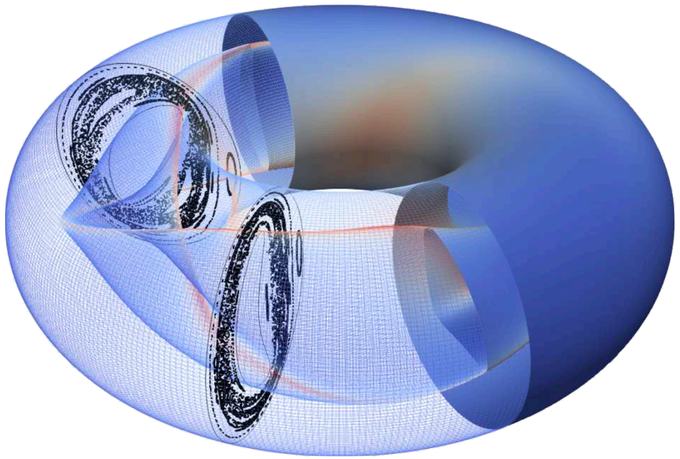
Vesicle flows [Veerapaneni et al., 2011]



Geodesic distance [Crane et al., 2017]



Stellarator design [Malhotra et al., 2019]



Introduction
Surface PDEs

Surface PDEs describe the dynamics of such phenomena.

Steady-state problem

$$\mathcal{L}_\Gamma u(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Gamma$$

- Laplace–Beltrami
- convection–diffusion
- steady Stokes

Time-dependent problem

$$\frac{\partial u}{\partial t} = \underbrace{\mathcal{L}_\Gamma u}_{\text{Linear}} + \underbrace{\mathcal{N}(u)}_{\text{Nonlinear}} \text{ on } \Gamma$$

- reaction–diffusion
- heat
- Navier–Stokes

Implicit time discretization:

$$(I - \Delta t \mathcal{L}_\Gamma)\, u^{k+1} = u^k + \Delta t \mathcal{N}(u^k)$$

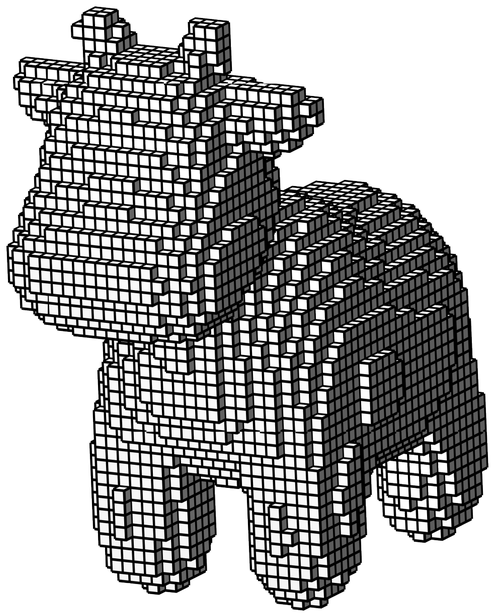
Model surface PDE: $\nabla_\Gamma \cdot (\boldsymbol{A}(\boldsymbol{x}) \nabla_\Gamma u(\boldsymbol{x})) + \nabla_\Gamma \cdot (\boldsymbol{b}(\boldsymbol{x}) u(\boldsymbol{x})) + c(\boldsymbol{x}) u(\boldsymbol{x}) = f(\boldsymbol{x})$

(+ BCs if surface is not closed)

Surface representation

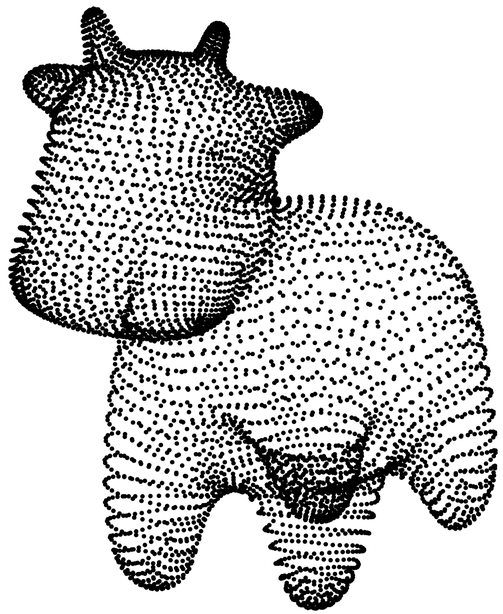
Many ways to represent a surface

Embedded grid



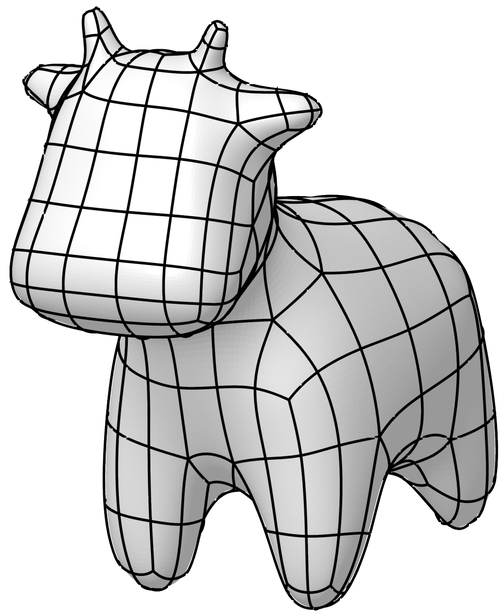
- Closest point methods
  (Macdonald, Greer, Ruuth, …)
- Level set methods
  (Sethian, Osher, …)

Unstructured point cloud



- Radial basis functions
  (Fornberg, Piret, Wendland, Wright, …)

Surface mesh

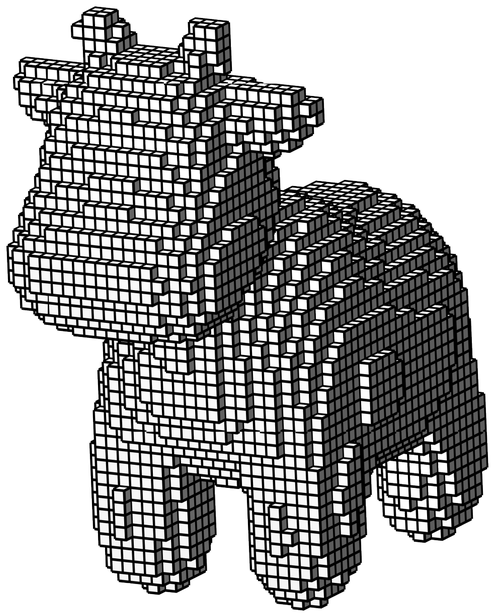


- Finite element methods
  (Dziuk, Elliott, …)
- Integral equation methods
  (O'Neil, Goodwill, Rachh, …)

Surface representation

Many ways to represent a surface

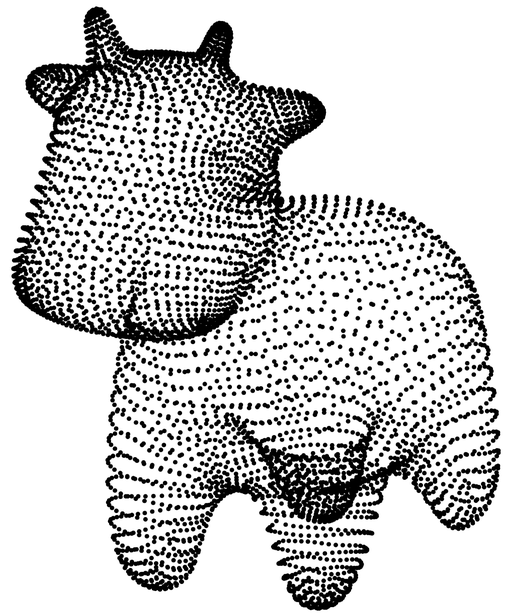
Embedded grid



- Closest point methods
  (Macdonald, Greer, Ruuth, …)
- Level set methods
  (Sethian, Osher, …)

high order → more volume DoFs

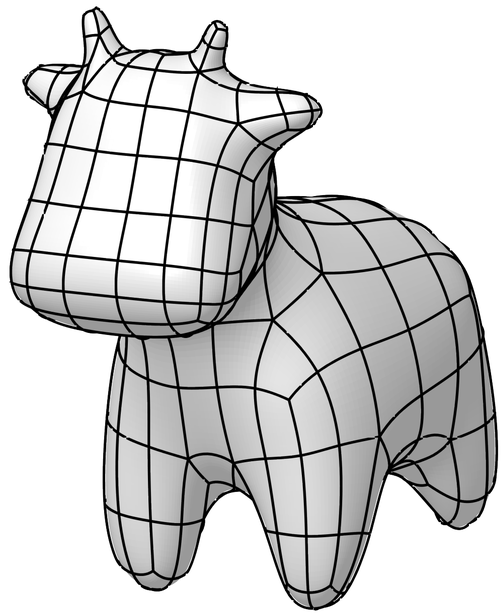
Unstructured point cloud



- Radial basis functions
  (Fornberg, Piret, Wendland, Wright, …)

high order → ill-conditioned

Surface mesh



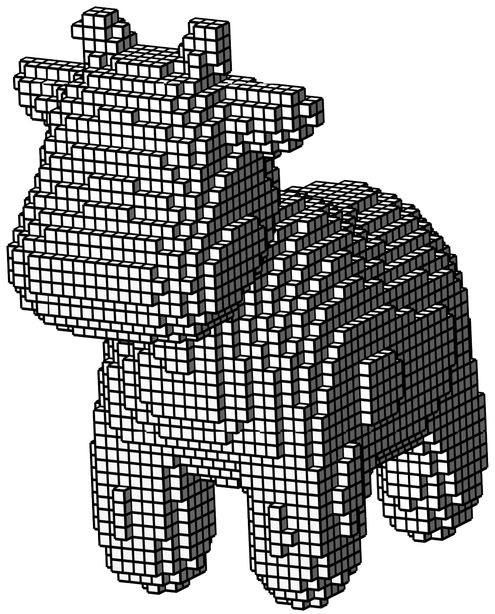
- Finite element methods
  (Dziuk, Elliott, …)
- Integral equation methods
  (O'Neil, Goodwill, Rachh, …)

high order → dense fill-in

Surface representation

Many ways to represent a surface

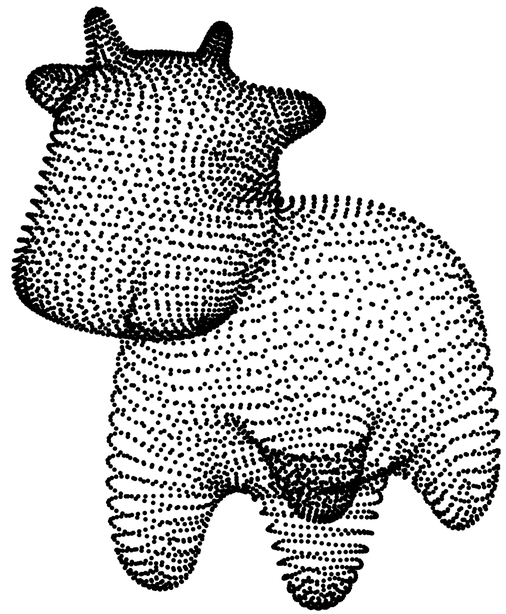
Embedded grid



- Closest point methods
  (Macdonald, Greer, Ruuth, …)
- Level set methods
  (Sethian, Osher, …)

high order → more volume DoFs

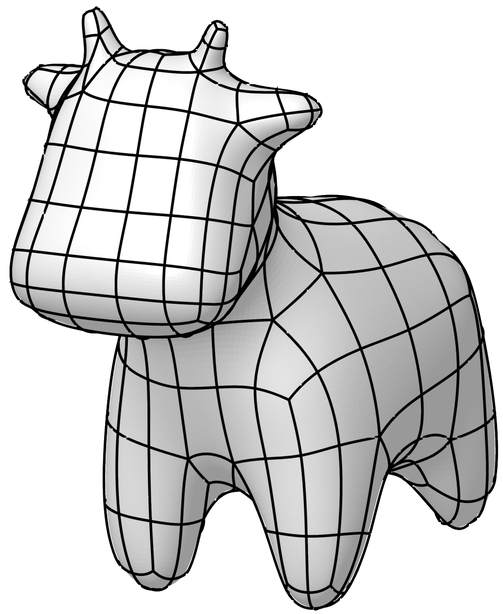
Unstructured point cloud



- Radial basis functions
  (Fornberg, Piret, Wendland, Wright, …)

high order → ill-conditioned

Surface mesh



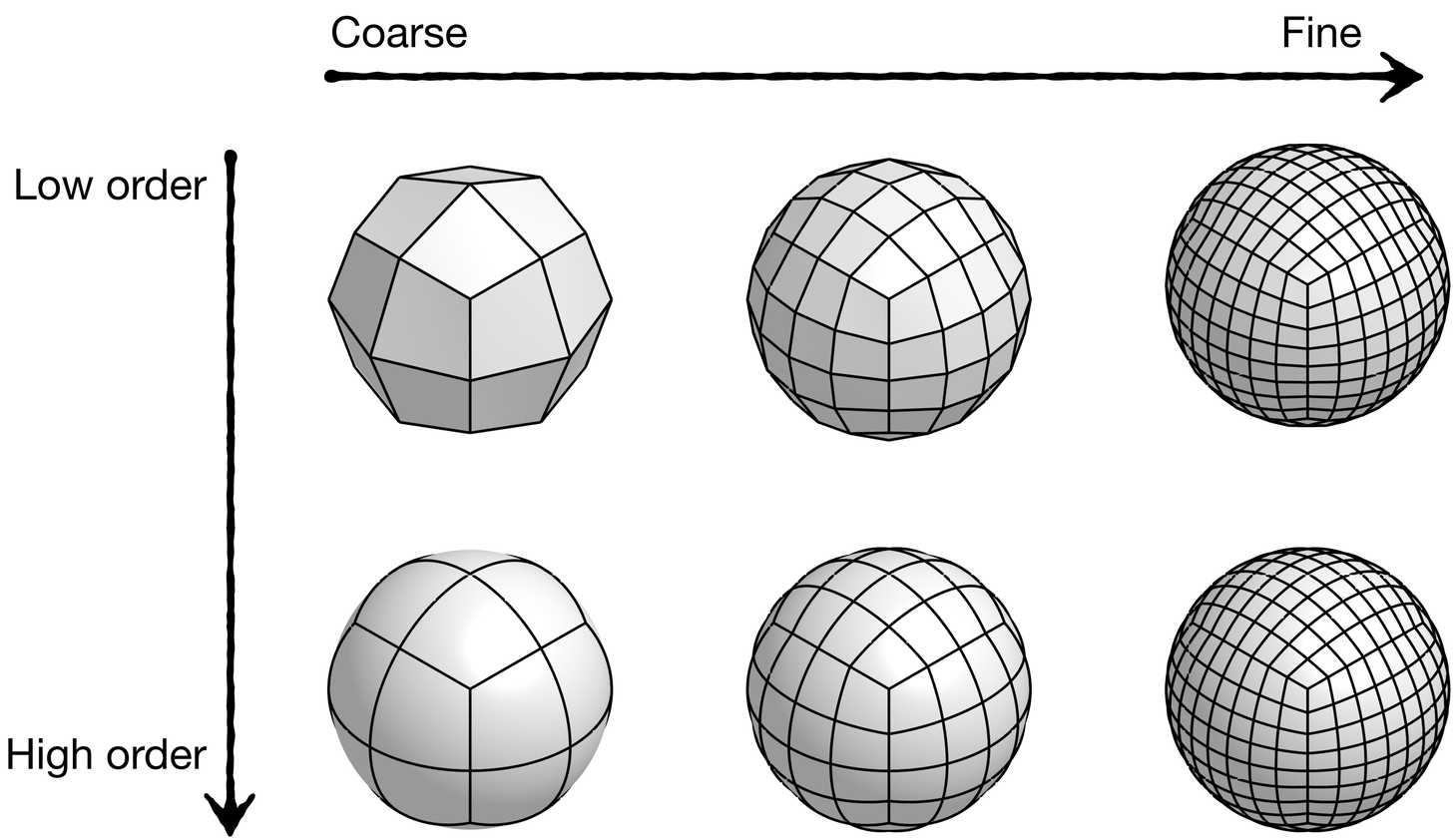
- Finite element methods
  (Dziuk, Elliott, …)
- Integral equation methods
  (O'Neil, Goodwill, Rachh, …)

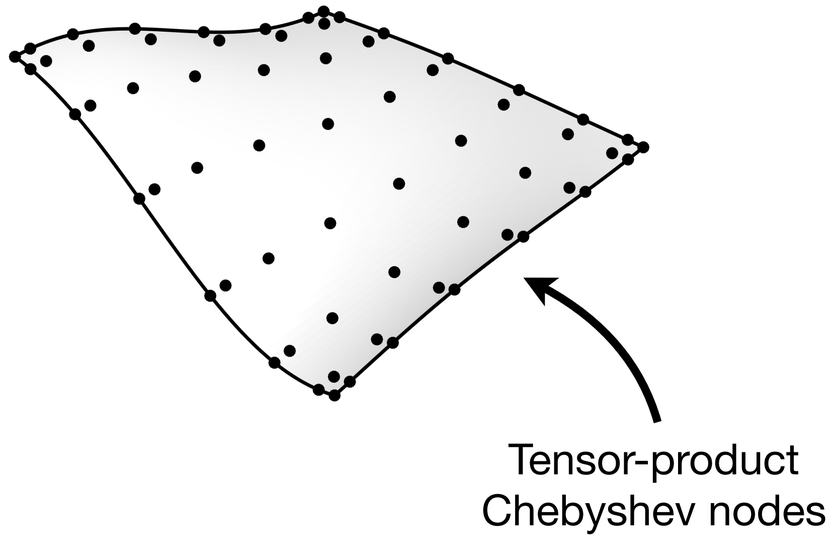
high order → dense fill-in

Developing efficient, high-order accurate solvers on surfaces is challenging!

Surface representation
Low-order vs. high-order

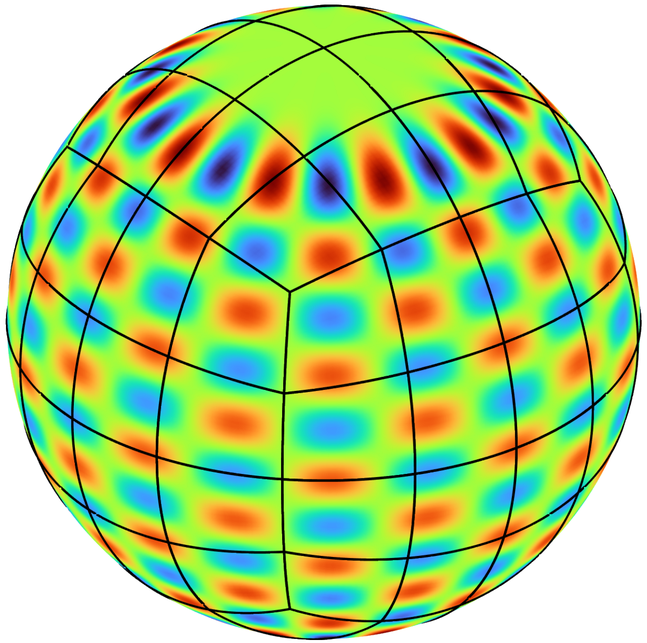
- Meshes are a good choice for CAD-compatibility. We use high-order quadrilateral patches.
- High-order elements allow faster convergence to solution.
- Coordinate maps of a patch are discretized via tabulation at Chebyshev nodes.





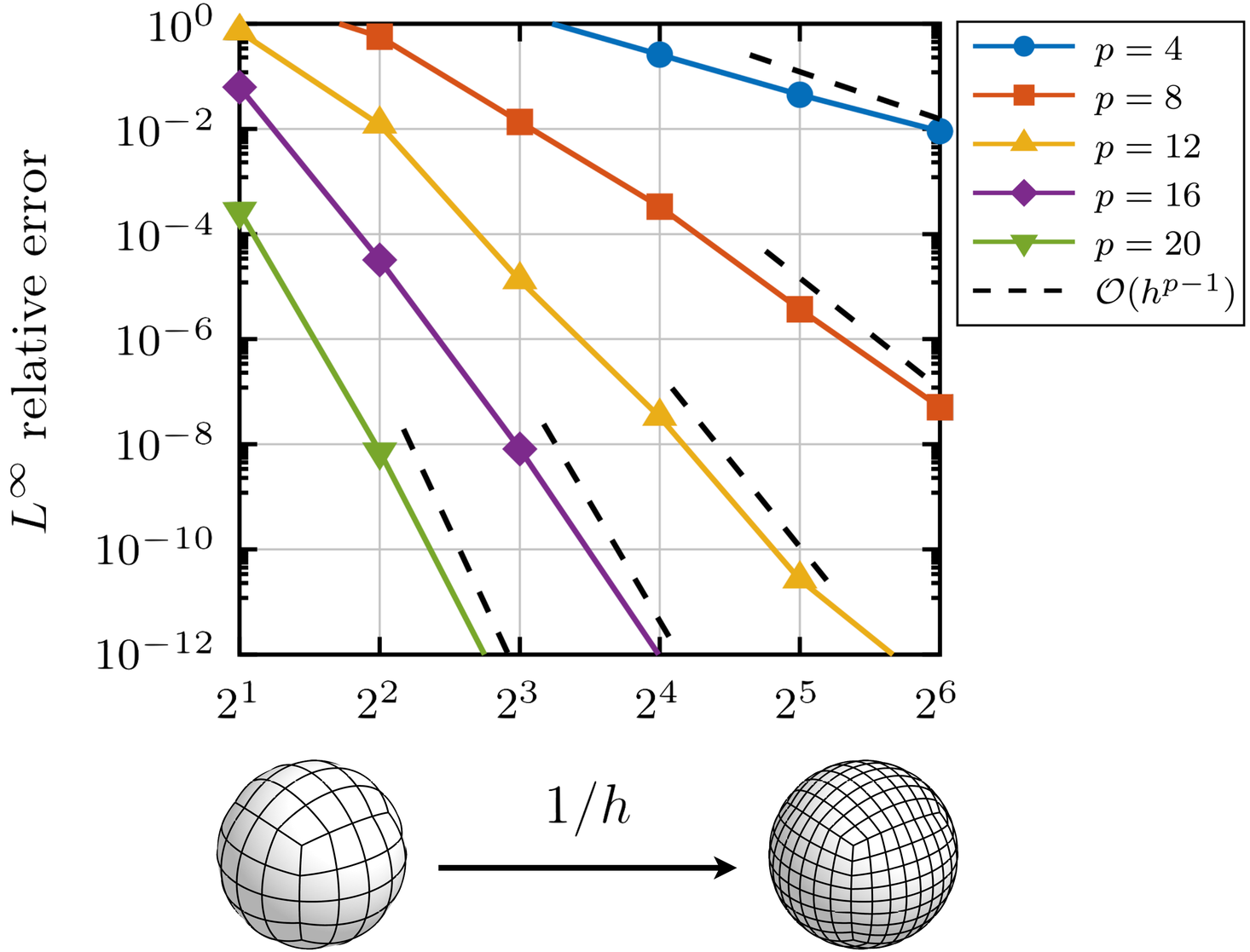
Examples
Laplace–Beltrami and convergence

$\Delta_\Gamma u = f, \quad \Gamma = \text{sphere}$

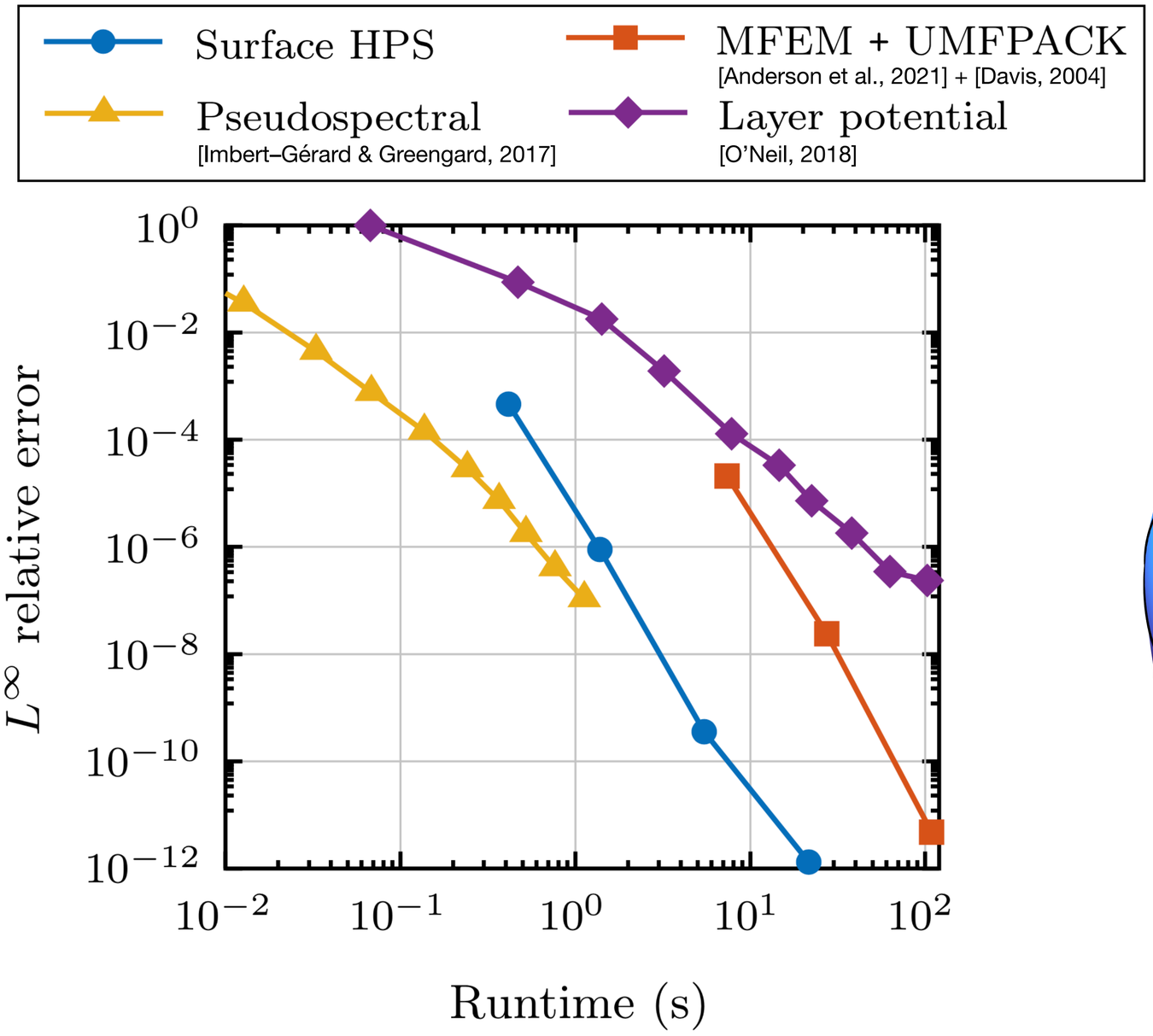


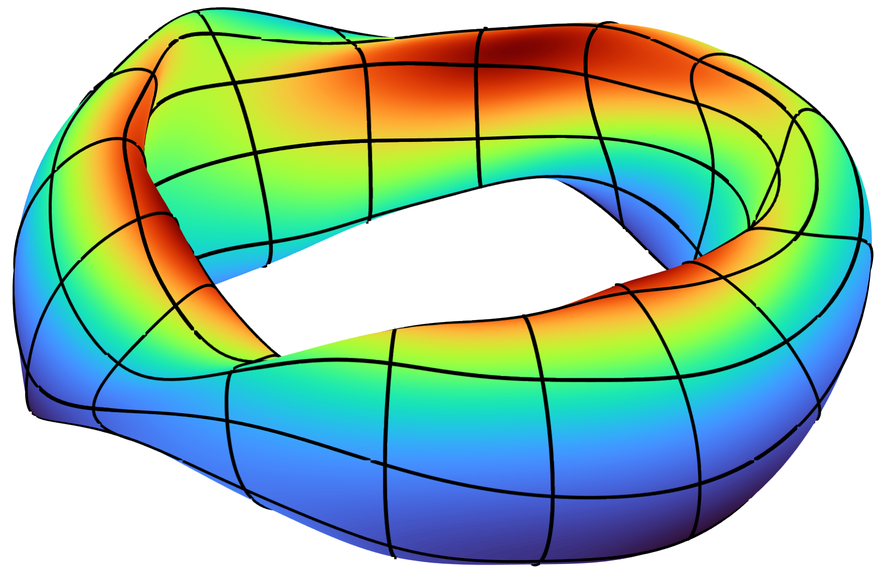
$u(\boldsymbol{x}) = \text{spherical harmonic, } Y_\ell^m(\boldsymbol{x})$

$f(\boldsymbol{x}) = -\ell(\ell+1)Y_\ell^m(\boldsymbol{x})$



Examples
Laplace–Beltrami: *“accuracy vs. effort”*

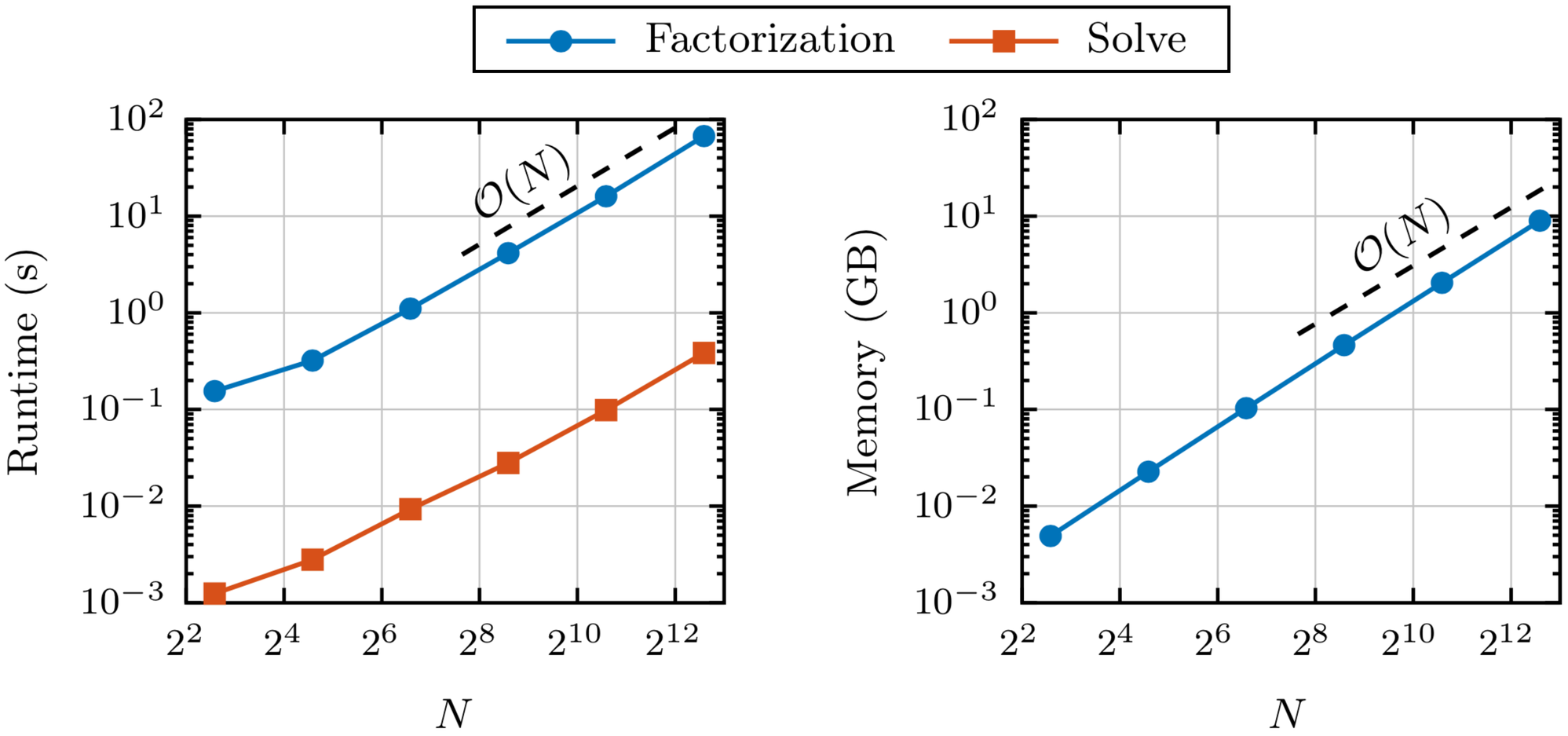




16th order
8-core Intel i9 Macbook Pro
64GB RAM

Examples
Performance

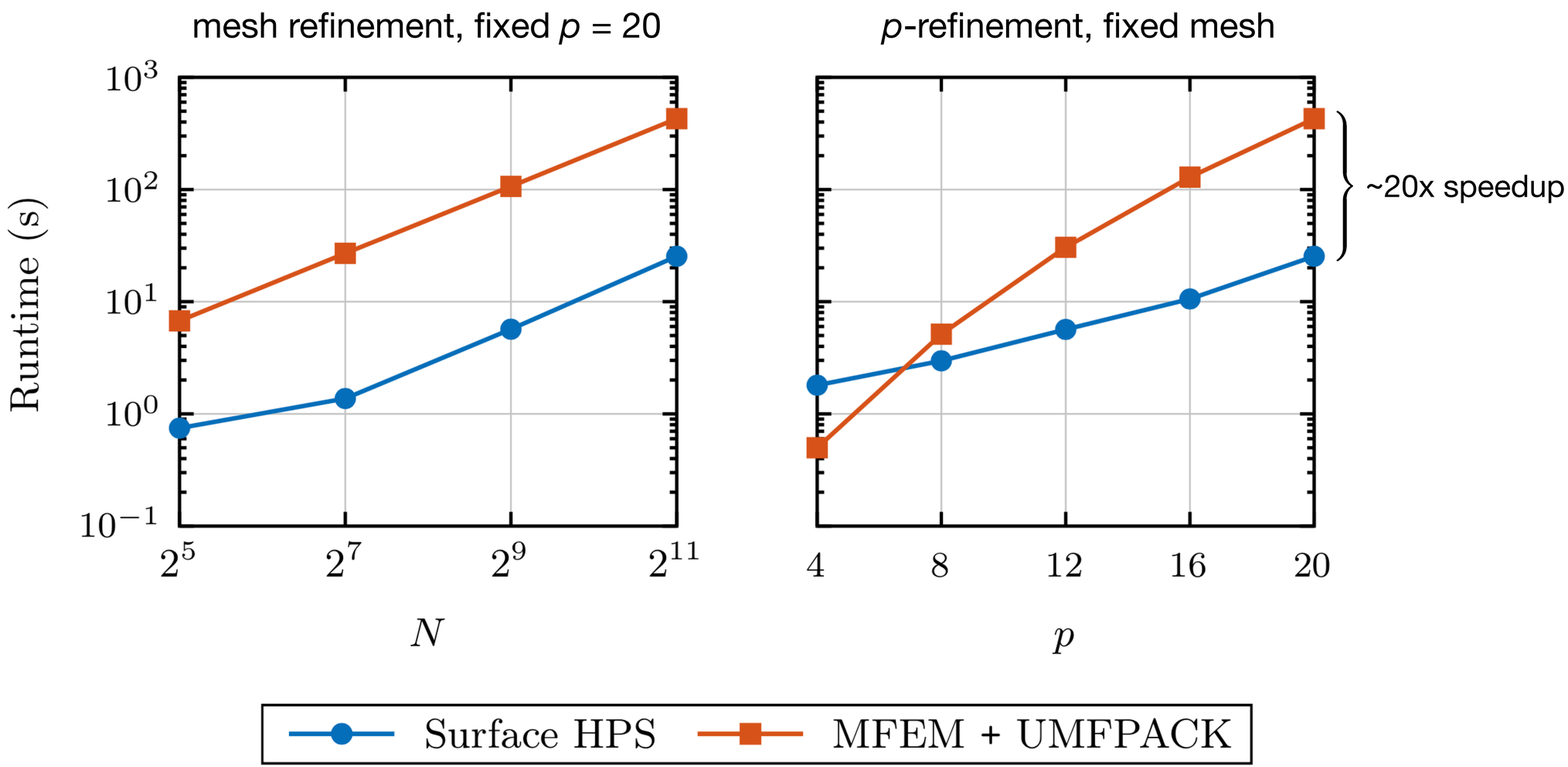
Computational complexity: $\mathcal{O}(N^{3/2})$ factorization, but $\mathcal{O}(N)$ is observed pre-asymptotically.



Examples
Performance

`github.com/danfortunato/surface-hps`

Significant speedups at high order when compared to FEM with a sparse direct solver.



Examples
Reaction–diffusion systems

- Reaction and diffusion timescales are often orders of magnitude different.

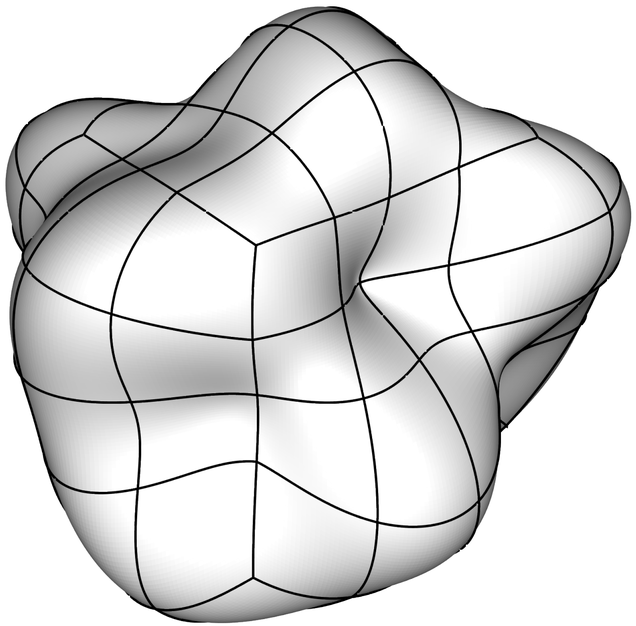
$$\frac{\partial u}{\partial t} = \underbrace{\mathcal{L}_\Gamma u}_{\text{Diffusion}} + \underbrace{\mathcal{N}(u)}_{\text{Reaction}} \quad \text{on } \Gamma$$

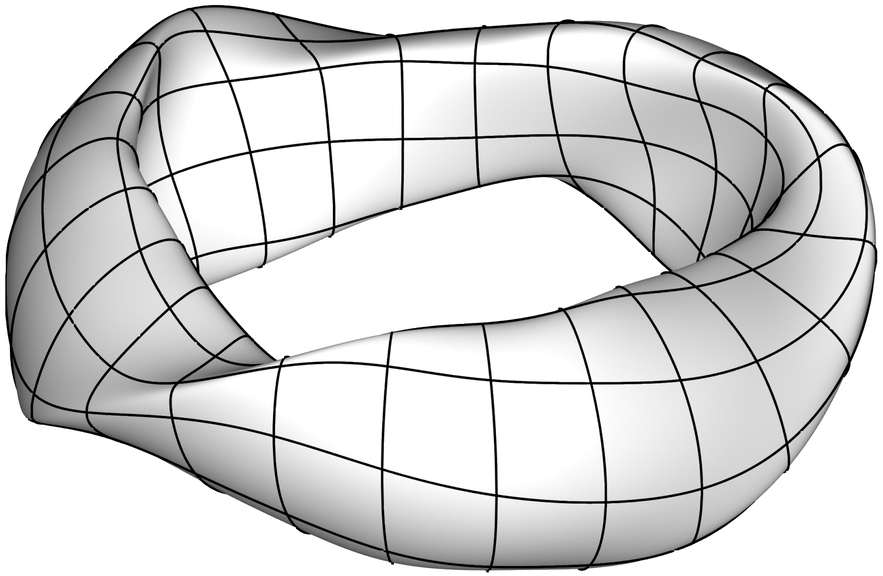
- Implicit time-stepping can alleviate stability issues (e.g., backward Euler or IMEX-BDF4)

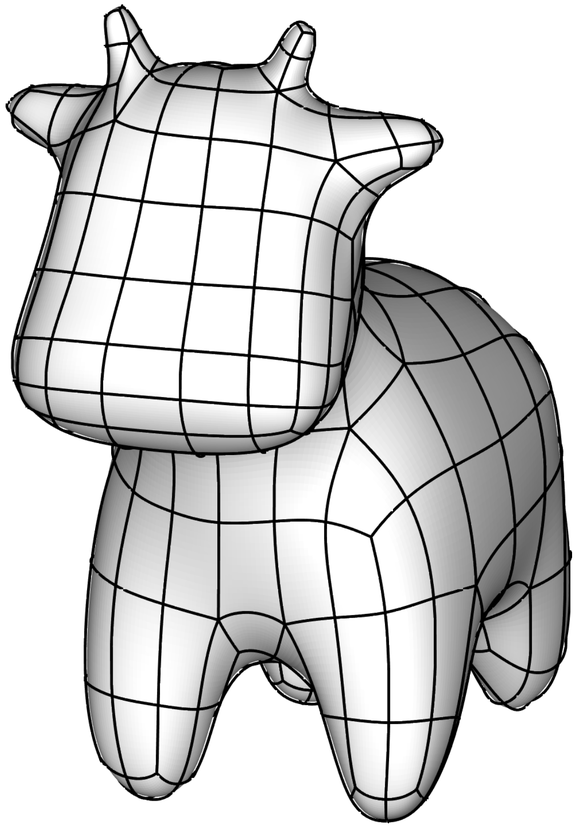
$$\frac{\partial u}{\partial t} = \mathcal{L}_\Gamma u + \mathcal{N}(u) \xrightarrow{\text{(e.g. backward Euler)}} u^{k+1} = \underbrace{(I - \Delta t \mathcal{L}_\Gamma)^{-1}}_{\substack{\text{Stored in RAM,} \\ \text{very fast apply}}} \left(u^k + \Delta t \mathcal{N}(u^k)\right)$$

- If geometry, time step, and parameters do not change with time, we can precompute a solver once and reuse it at every step.

Examples
Reaction–diffusion systems

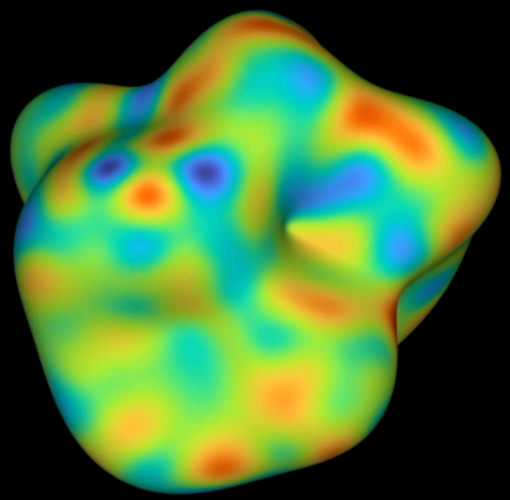
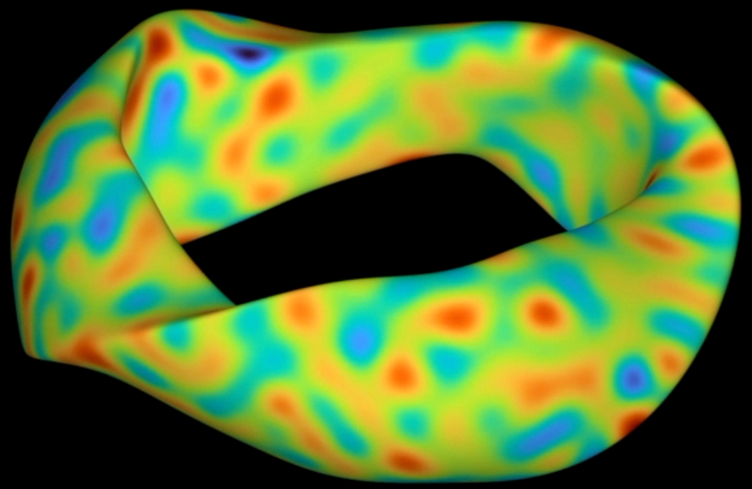
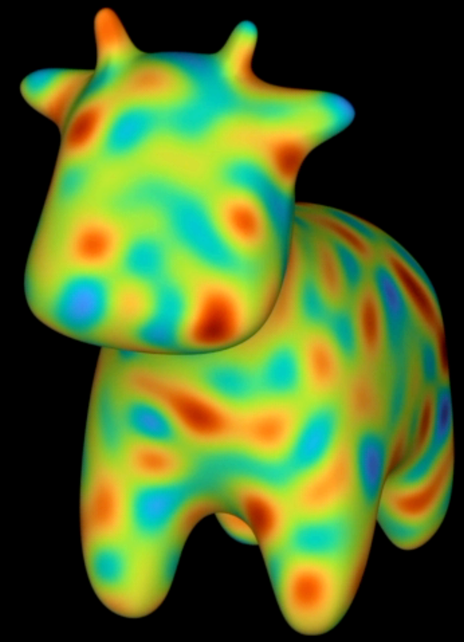


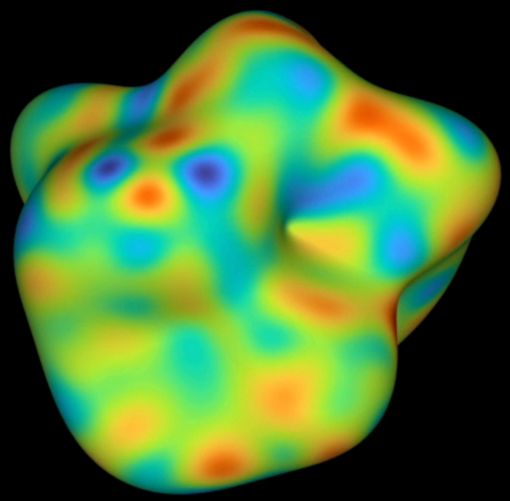
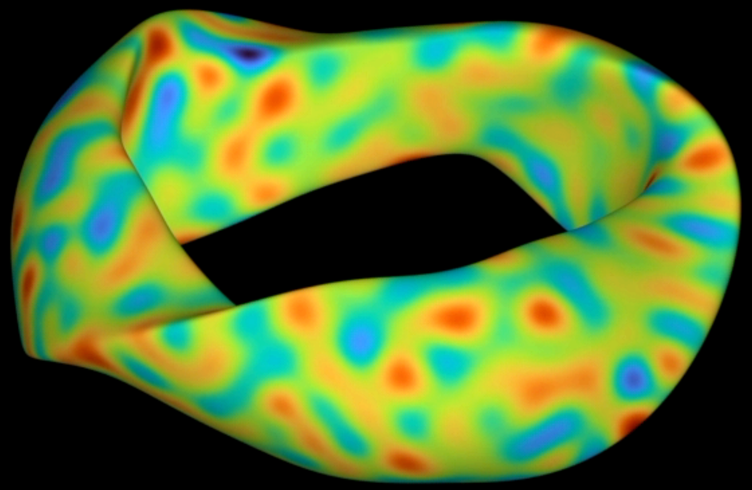
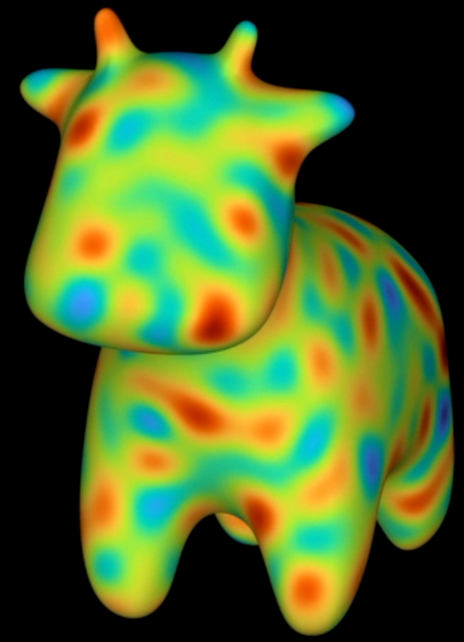




16th order
8-core Intel i9 Macbook Pro
64GB RAM

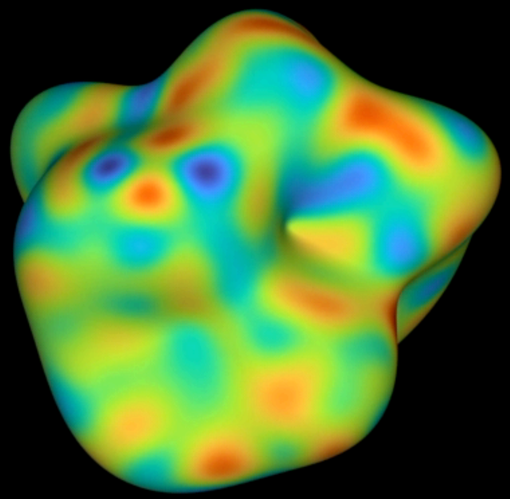
$$\frac{\partial u}{\partial t} = \delta_u^2 \Delta_\Gamma u + \alpha u(1 - \tau_1 v^2) + v(1 - \tau_2 u)$$
$$\frac{\partial v}{\partial t} = \delta_v^2 \Delta_\Gamma v + \beta v(1 + \alpha \tau_1 uv/\beta) + u(\gamma + \tau_2 v)$$

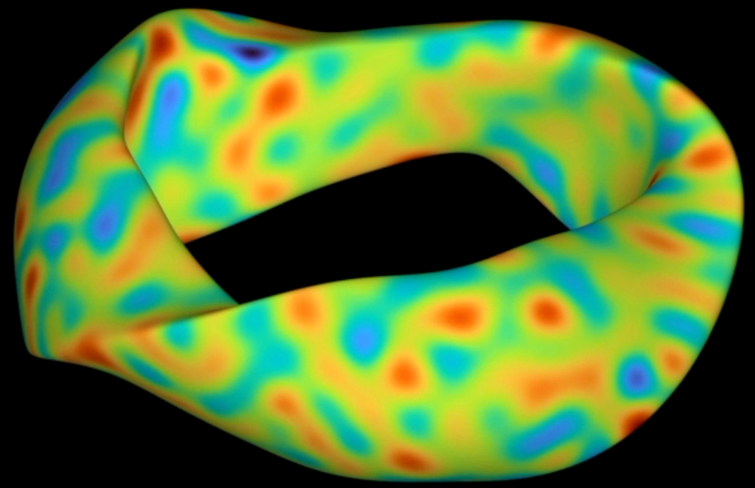
$$\frac{\partial u}{\partial t} = \delta^2 \Delta_\Gamma u + u - (1 + ci)u|u|^2$$

$$\frac{\partial u}{\partial t} = \delta_u^2 \Delta_\Gamma u + \alpha u(1 - \tau_1 v^2) + v(1 - \tau_2 u)$$
$$\frac{\partial v}{\partial t} = \delta_v^2 \Delta_\Gamma v + \beta v(1 + \alpha\tau_1 uv/\beta) + u(\gamma + \tau_2 v)$$



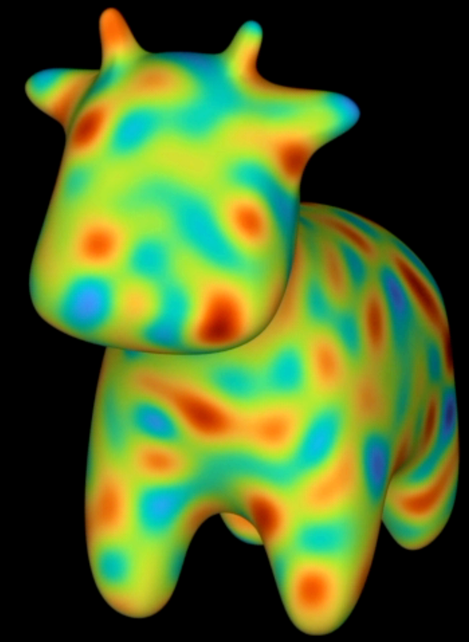
25k unknowns

~20 fps



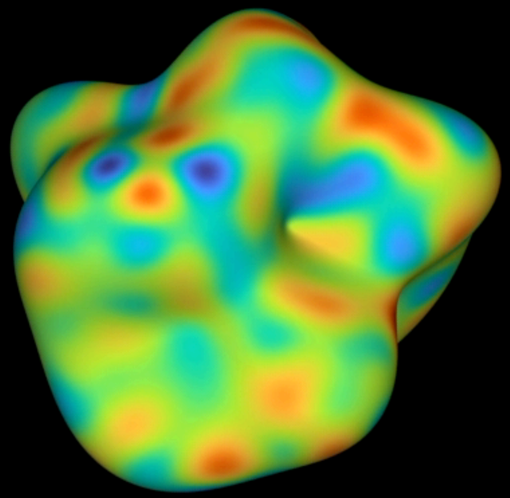
50k unknowns

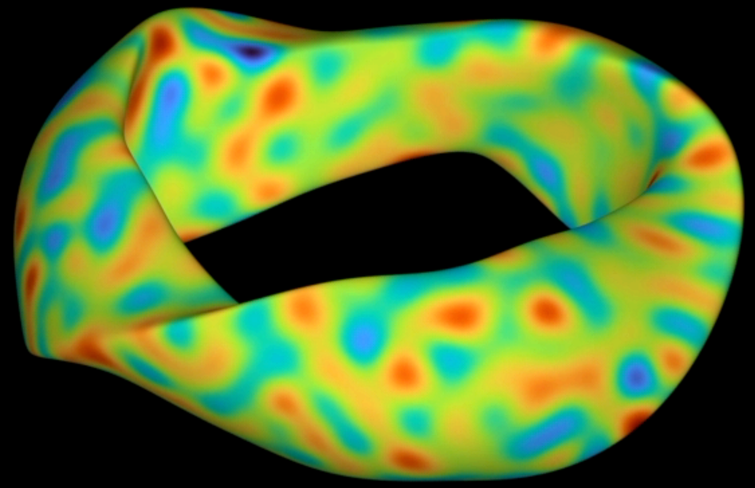
~13 fps

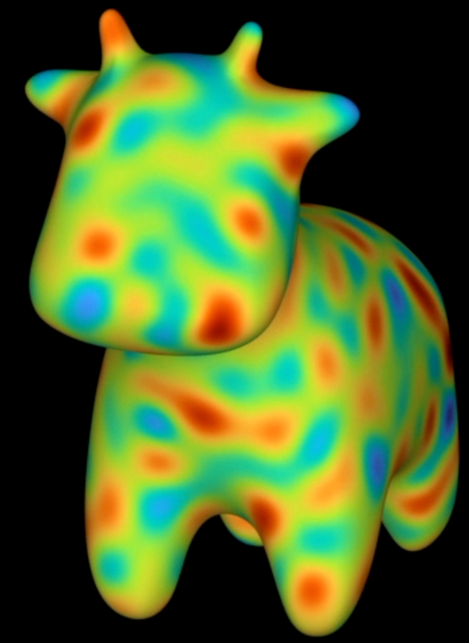


135k unknowns

~4 fps



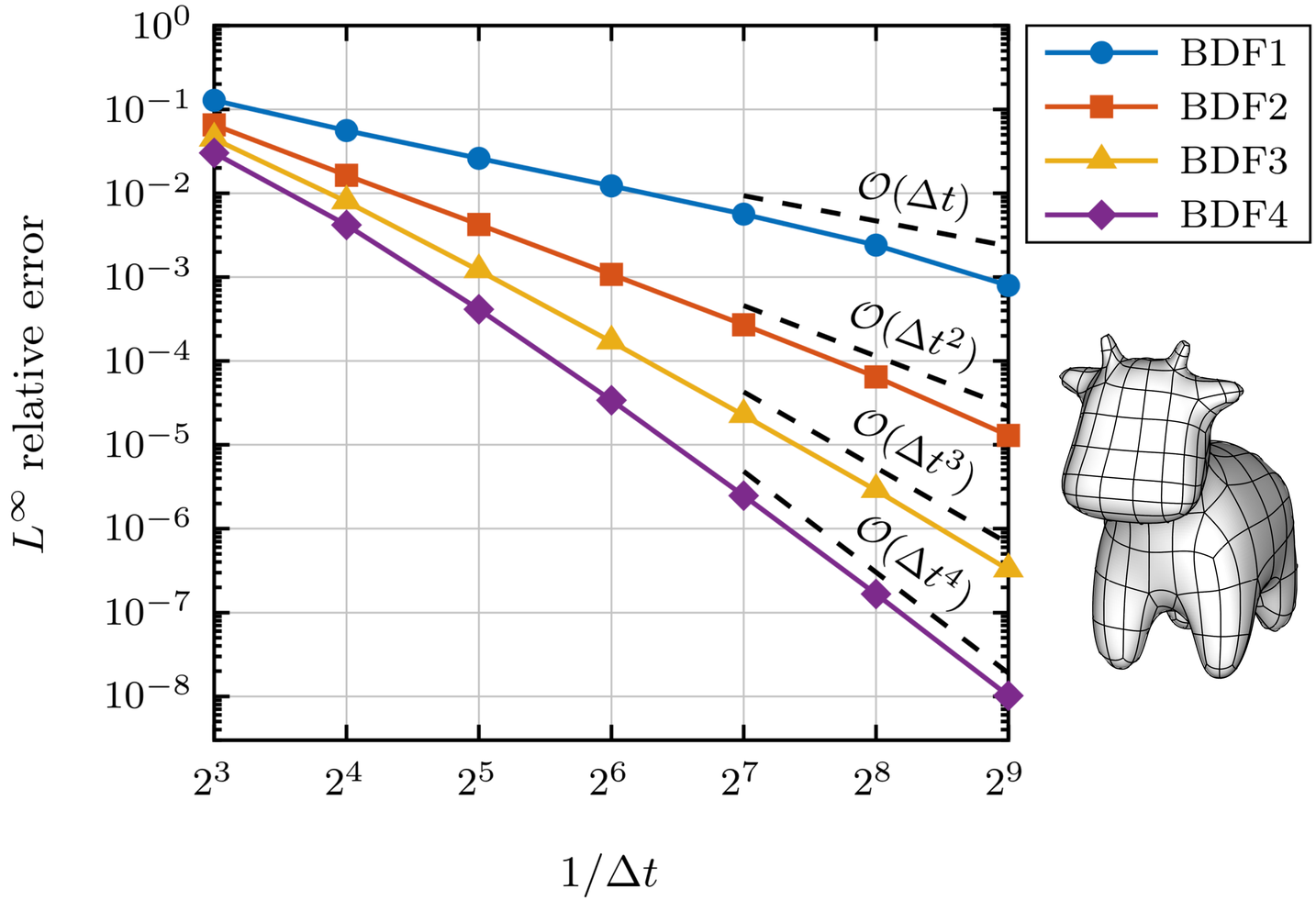




$$\frac{\partial u}{\partial t} = \delta^2 \Delta_\Gamma u + u - (1 + ci)u|u|^2$$

Examples
Reaction–diffusion systems

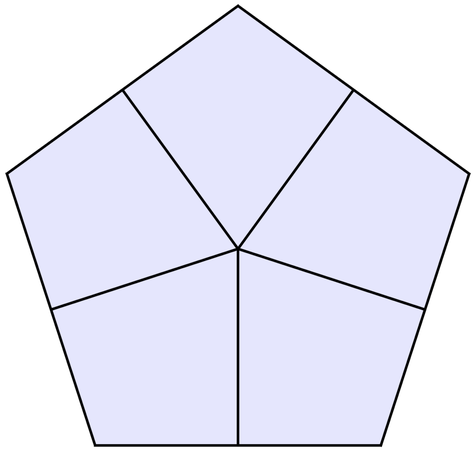
IMEX-BDF methods can achieve high-order accuracy in time.



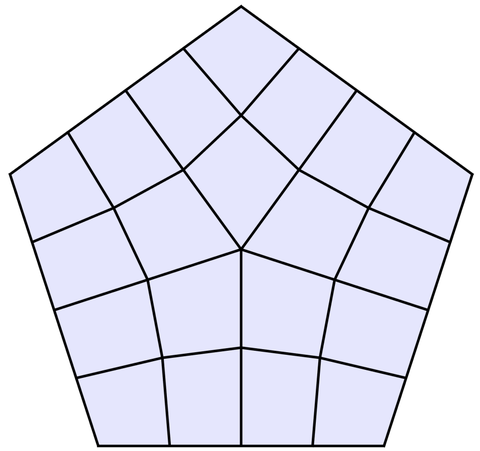
Software

ultraSEM.org

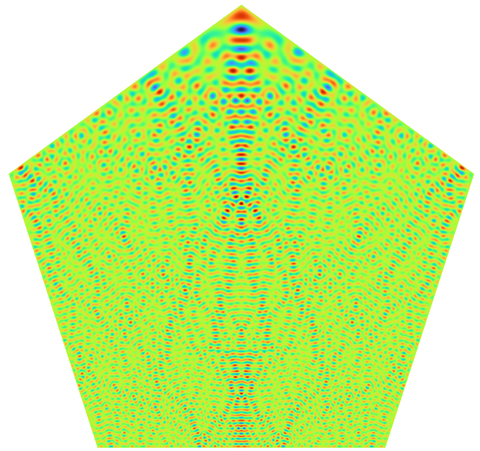
$$\nabla^2 u + 50000(1-y)u = -1 \quad \text{in pentagon}$$
$$u = 0 \quad \text{on boundary}$$



```
dom = ultraSEM.polygon(5);
plot(dom)
```



```
dom = refine(dom);
plot(dom)
```



```
p = 60; rhs = -1; bc = 0;
pdo = {1, 0, @(x,y) 50000*(1-y)};
S = ultraSEM(dom, pdo, rhs, p);
u = S \ bc;
plot(u)
```

[F., Hale, & Townsend, 2021]

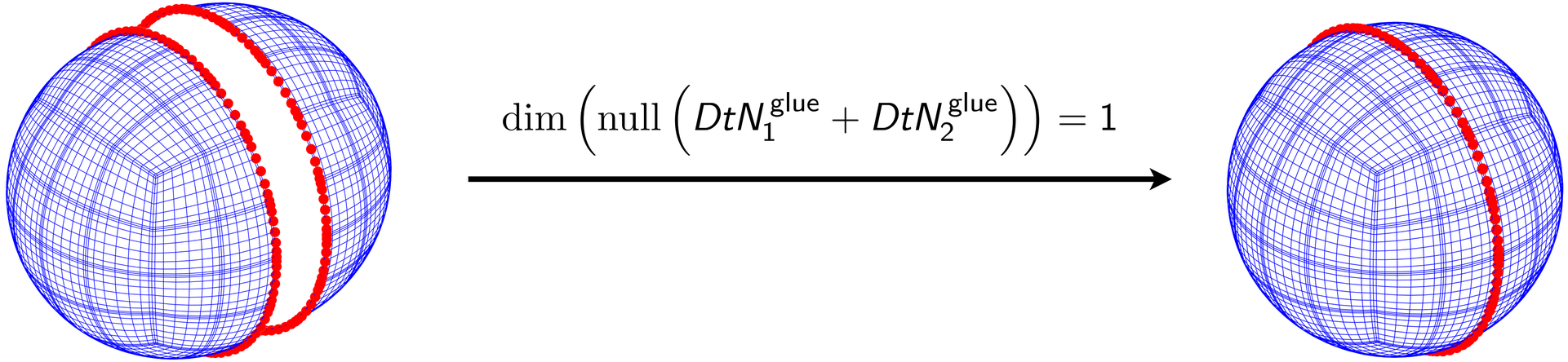
A fast direct solver on surfaces
Laplace–Beltrami and rank deficiency

$$\Delta_\Gamma u = f$$

- The Laplace–Beltrami problem on a closed surface is rank-one deficient, but is uniquely solvable under the mean-zero conditions:

$$\int_\Gamma u = 0 \qquad \text{and} \qquad \int_\Gamma f = 0$$

- In HPS, this rank deficiency is only seen in the final gluing:



- We add the mean-zero constraint to fix the rank deficiency at the top level:

$$\dim\left(\text{null}\left(DtN_1^{\text{glue}} + DtN_2^{\text{glue}} + \mathbf{q}\mathbf{q}^T\right)\right) = 0$$