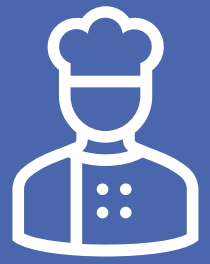
A RECIPE FOR ADAPTIVE HP-FEMS

Adam Blakey



Blakey Bakery Co.

Today's Menu

- **Appetiser**
  - Introductory analysis

- **A quick snack**
  - Interactive example

- **Sorbet**
  - C++ implementation
  - Numerical experiments

- **Main course**
  - hp-adaptive FEMs in action

- **Pudding**
  - Future work

The Appetiser

Why FEMs?

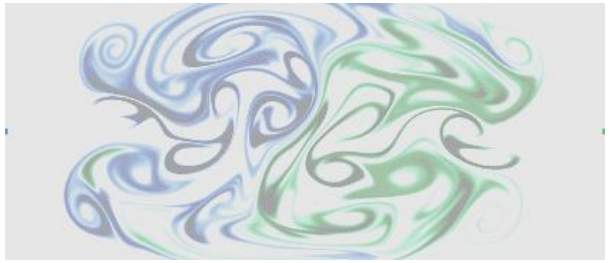
Differential equations are vital to understanding our universe:

- Fluid flow
- Weather prediction
- Space flight

Few have analytical solutions:

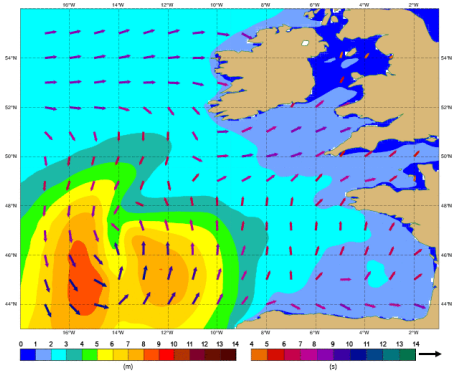
- Finite differences
- Finite volumes
- Finite element methods

[Karl Sims]



[NASA]

[ECMWF]



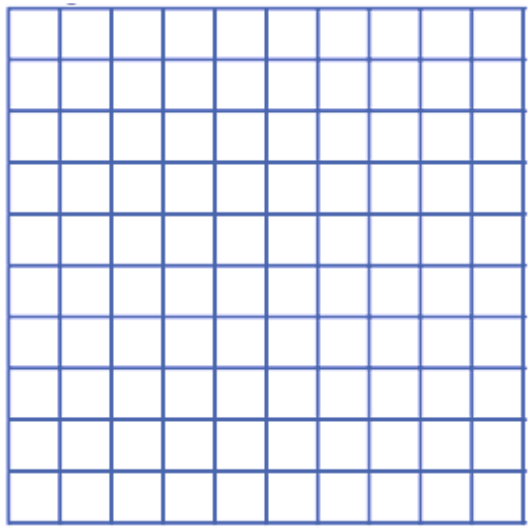
The Appetiser

FEM Properties

We generally choose FEMs over other methods thanks to their:

- Geometric flexibility
- Adaptivity and error control
- Optimal (high) orders of convergence

Regular mesh



Irregular mesh [Hans Petter Langtangen]

The Appetiser

PDEs

Infinite-Dimensional PDEs

Loosely speaking, all real PDE problems can be written as:

$$\text{Find } u \in V \text{ s.t. } \mathcal{L}u(x) = f(x),$$

where u lives in a suitable space, and $\mathcal{L}$ is some real differential operator.

Finite-Dimensional PDEs

To solve on a compute, we must somehow convert our problem to a finite dimensional representation:

$$\text{Find } u_h \in V_h \text{ s.t. } \Pi_h(\mathcal{L}u_h) = \Pi_h(f),$$

where $\Pi_h : V \to V_h$.

The Appetiser

Model Problem

Strong form

Find u s.t.

$$\begin{aligned} -\epsilon \Delta u + cu &= f, && \text{in } \Omega \\ u &= g_D, && \text{on } \partial\Omega. \end{aligned}$$

Weak form

Find $u \in V$ s.t.

$$\epsilon \int_\Omega \nabla u \cdot \nabla v + \int_\Omega cuv = \int_\Omega fv, \qquad \forall v \in V.$$

Galerkin

Find $u_h \in V_h$ s.t.

$$\epsilon \int_\Omega \nabla u_h \cdot \nabla v_h + \int_\Omega cu_h v_h = \int_\Omega f v_h, \qquad \forall v_h \in V_h.$$

The Appetiser

CG-FEM

Taking the scheme element-wise, we have: For each $\kappa \in \mathcal{T}_h$, find $u_h \in V_h$ s.t.

$$\epsilon \int_\kappa \nabla u_h \cdot \nabla v_h + \int_\kappa c u_h v_h = \int_\kappa f v_h, \qquad \forall v_h \in V_h ,$$

where $\mathcal{T}_h$ is the discretised version of Ω, and some boundary conditions are somehow applied to u_h.

The Appetiser

DG-FEM

Taking the scheme element-wise, we have: For each $\kappa \in \mathcal{T}_h$, find $u_h \in V_h$ s.t.

$$\epsilon \int_\kappa \nabla u_h \cdot \nabla v_h + \int_\kappa c u_h v_h + \int_{\partial\kappa} -\{\{\epsilon \nabla u_h\}\}[\![v_h]\!] - \{\{\epsilon \nabla v_h\}\}[\![u_h]\!] + \alpha [\![u_h]\!][\![v_h]\!] = \int_\kappa f v_h, \qquad \forall v_h \in V_h ,$$

where $\mathcal{T}_h$ is the discretised version of Ω, and ~~some boundary conditions are somehow applied to u_h~~.

We've chosen here the flux functions to be

$$\hat{u} = \begin{cases} \{\{u_h\}\} & \text{on } F \in \mathcal{T}_h^I \\ g_D & \text{on } F \in \mathcal{T}_h^B \end{cases} \qquad \hat{\sigma} = \begin{cases} \{\{\nabla u_h\}\} - \alpha [\![u_h]\!] & \text{on } F \in \mathcal{T}_h^I \\ \nabla u_h - \alpha (u_h - g_D)\boldsymbol{n} & \text{on } F \in \mathcal{T}_h^B \end{cases}$$

Specifically speaking, we will use the Symmetric Interior Penalty scheme.

The Appetiser

DG-FEM

$$\int_\Omega \sigma_h \cdot \tau \, dx = -\int_\Omega u_h \nabla_h \cdot \tau \, dx + \sum_{K \in \mathcal{T}_h} \int_{\partial K} \widehat{u}_K \, n_K \cdot \tau \, ds \quad \forall \tau \in \Sigma_h,$$

$$\int_\Omega \sigma_h \cdot \nabla_h v \, dx = \int_\Omega f \, v \, dx + \sum_{K \in \mathcal{T}_h} \int_{\partial K} \widehat{\sigma}_K \cdot n_K \, v \, ds \quad \forall v \in V_h,$$

TABLE 3.1
Some DG methods and their numerical fluxes.

Method	$\widehat{u}_K$	$\widehat{\sigma}_K$
Bassi–Rebay [10]	$\{u_h\}$	$\{\sigma_h\}$
Brezzi et al. [22]	$\{u_h\}$	$\{\sigma_h\} - \alpha_r([\![u_h]\!])$
LDG [41]	$\{u_h\} - \beta \cdot [\![u_h]\!]$	$\{\sigma_h\} + \beta [\![\sigma_h]\!] - \alpha_j([\![u_h]\!])$
IP [50]	$\{u_h\}$	$\{\nabla_h u_h\} - \alpha_j([\![u_h]\!])$
Bassi et al. [13]	$\{u_h\}$	$\{\nabla_h u_h\} - \alpha_r([\![u_h]\!])$
Baumann–Oden [15]	$\{u_h\} + n_K \cdot [\![u_h]\!]$	$\{\nabla_h u_h\}$
NIPG [64]	$\{u_h\} + n_K \cdot [\![u_h]\!]$	$\{\nabla_h u_h\} - \alpha_j([\![u_h]\!])$
Babuška–Zlámal [7]	$(u_h\|_K)\|_{\partial K}$	$-\alpha_j([\![u_h]\!])$
Brezzi et al. [23]	$(u_h\|_K)\|_{\partial K}$	$-\alpha_r([\![u_h]\!])$

[Arnold et al, 2001]

The Appetiser

Construction of V_h in 1D

We take our discretised finite element space to be

$$V_h \coloneqq \{u \in H_0^1(0,1) \ : \ u\Big|_{(x_{i-1},x_i)} \in \mathcal{P}_{p_i}(x_{i-1},x_i), \forall i \}$$

where (x_{i-1}, x_i) are the ith element's coordinates, and p_i is the ith element's maximum polynomial degree.

Abstract Recipe: A General FEM

1. Split your mesh
2. Take V_h as above
3. Solve for all combinations of basis functions

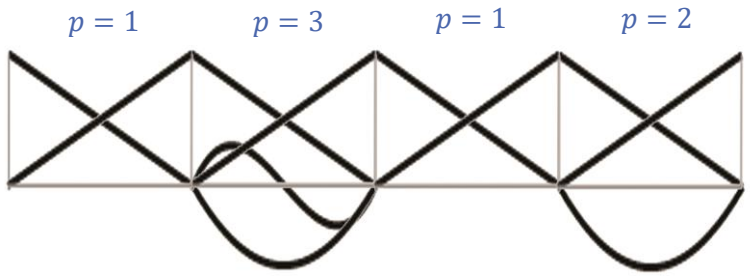
The Appetiser

Basis Functions

As suggested in [Pavel Šolin et. al, 2003], we propose to use Lobatto shape functions:

On **reference elements**:

$$l_0(x) = \frac{1-x}{2}, l_1(x) = \frac{1+x}{2},$$
$$l_k(x) = \frac{1}{\sqrt{2/(2k-1)}} \int_{-1}^{x} L_{k-1}(\xi),\ 2 \le k.$$



Lobatto shape functions

[Pavel Šolin et. al, 2003]

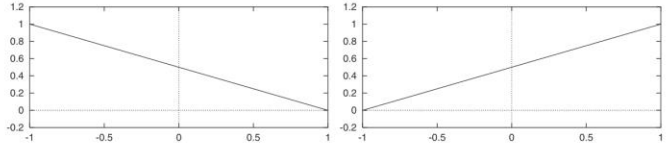


FIGURE 1.14: Lobatto shape functions l_0, l_1.

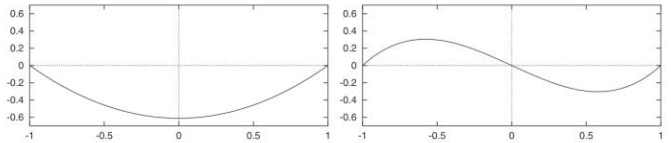


FIGURE 1.15: Lobatto shape functions l_2, l_3.

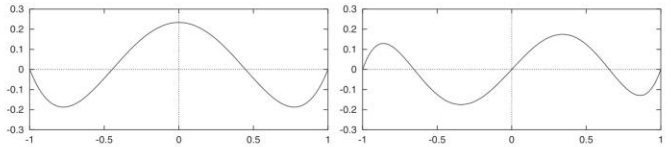


FIGURE 1.16: Lobatto shape functions l_4, l_5.

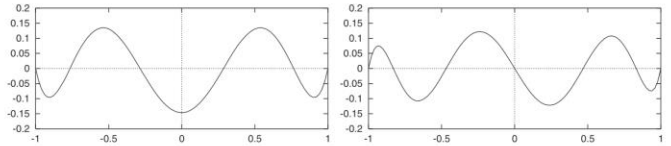


FIGURE 1.17: Lobatto shape functions l_6, l_7.

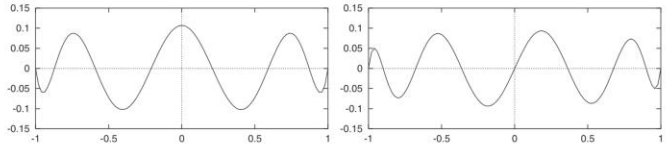


FIGURE 1.18: Lobatto shape functions l_8, l_9.

The Appetiser

Linear PDE CG-FEM Procedure

Ingredients:

- Elements in a mesh
- Choice of basis functions
- Linear solver

Recipe:

1. Create storage for A and L
2. Loop over all elements
   a. Loop over all combinations of basis functions on this element
      i. Calculate $a(u, v)$ and $l(v)$ and add contributions to A and L
3. Impose boundary conditions
4. Solve $Au = L$
5. Output $\sum_i u_i^h \phi_i(x)$

The Appetiser

Linear PDE DG-FEM Procedure

Ingredients:

- Elements in a mesh
- **Faces between elements**
- **Penalisation parameter**
- Choice of basis functions
- Linear solver

Recipe:

1. Create storage for A and L
2. Loop over all elements
   a. Loop over all combinations of basis functions on this element
      i. Calculate $a(u, v)$ and $l(v)$ and add contributions to A and L
      ii. **Calculate face contributions**
3. ~~Impose boundary conditions~~
4. Solve $Au = L$
5. Output $\sum_i u_i^h \phi_i(x)$

Specifically speaking, we will use the Symmetric Interior Penalty scheme.

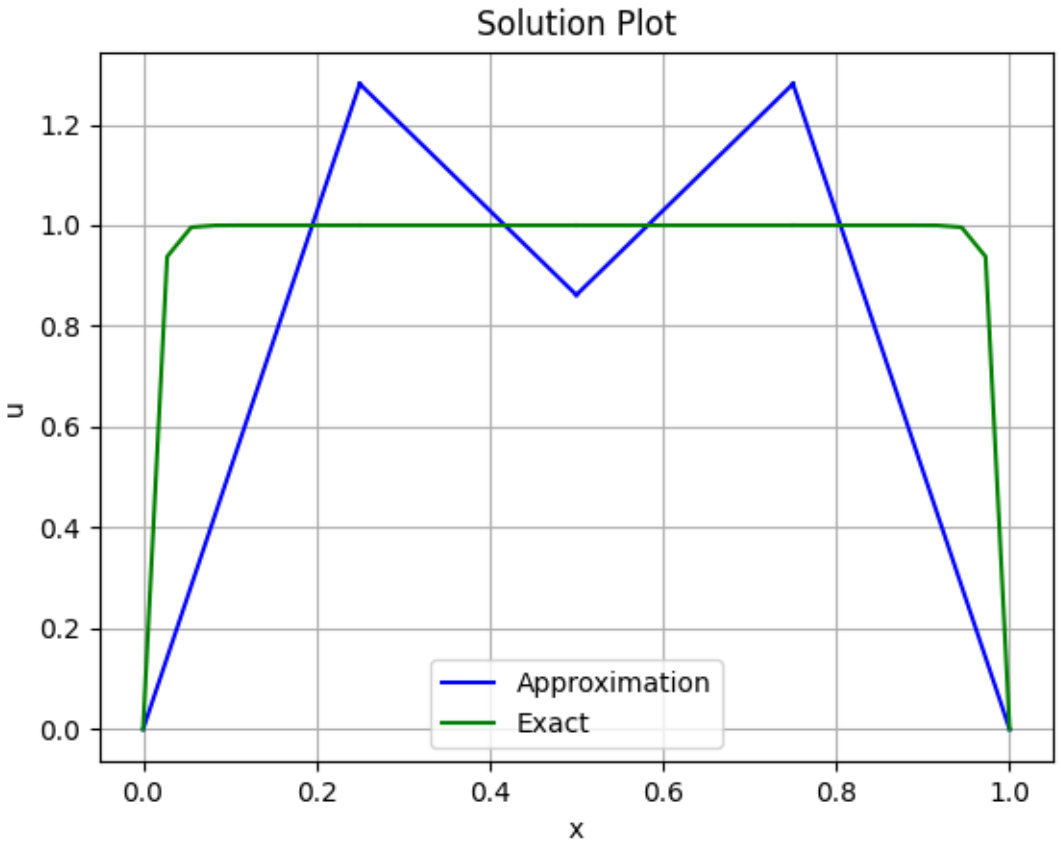
The Appetiser

Some Motivating Examples

Take the boundary layer problem:

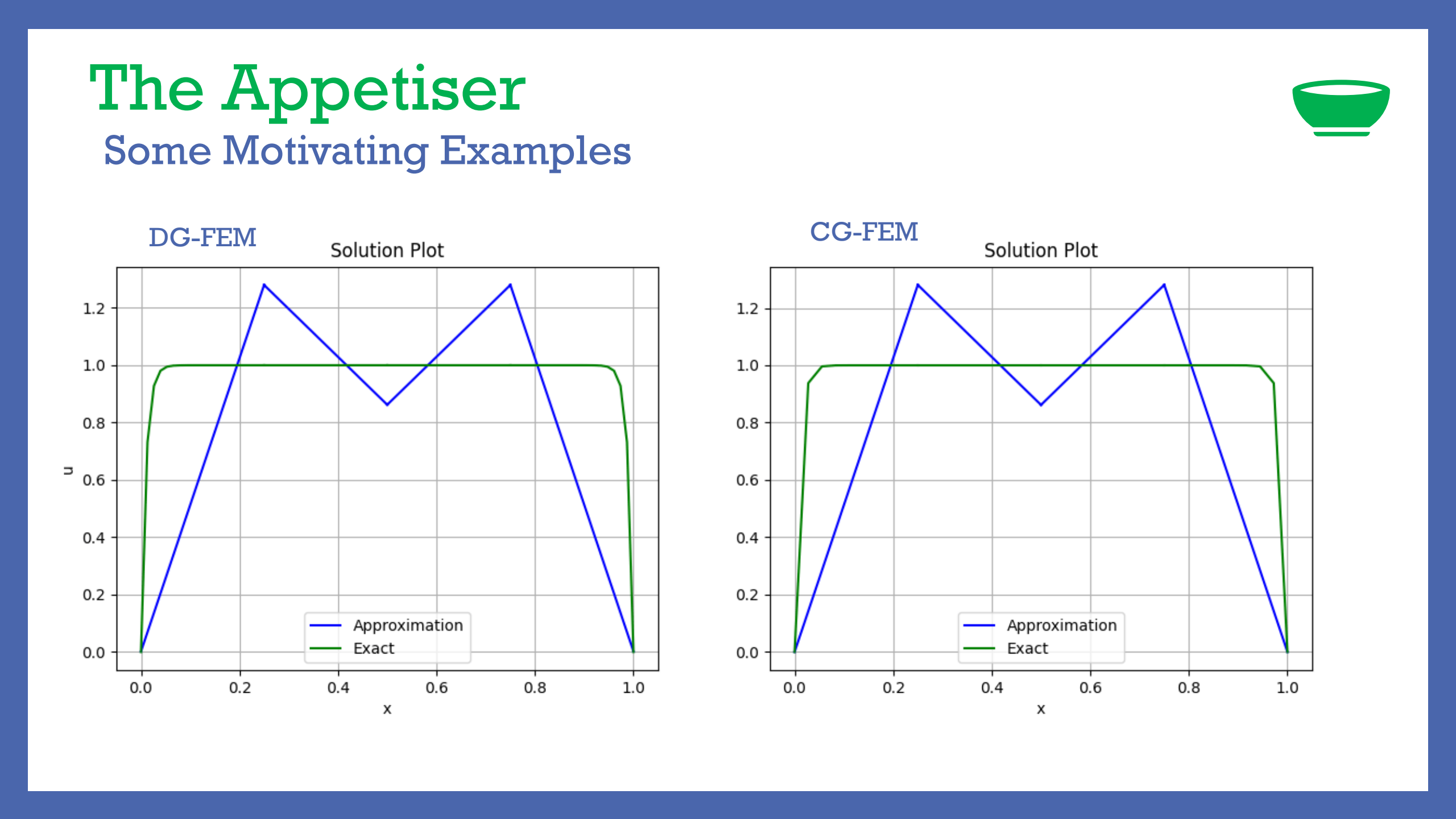
$$-0.0001u'' + u = 1,$$
$$u(-1) = 0,$$
$$u(1) = 0.$$

With **4 to 64 linear** elements, we have…



The Appetiser

Some Motivating Examples



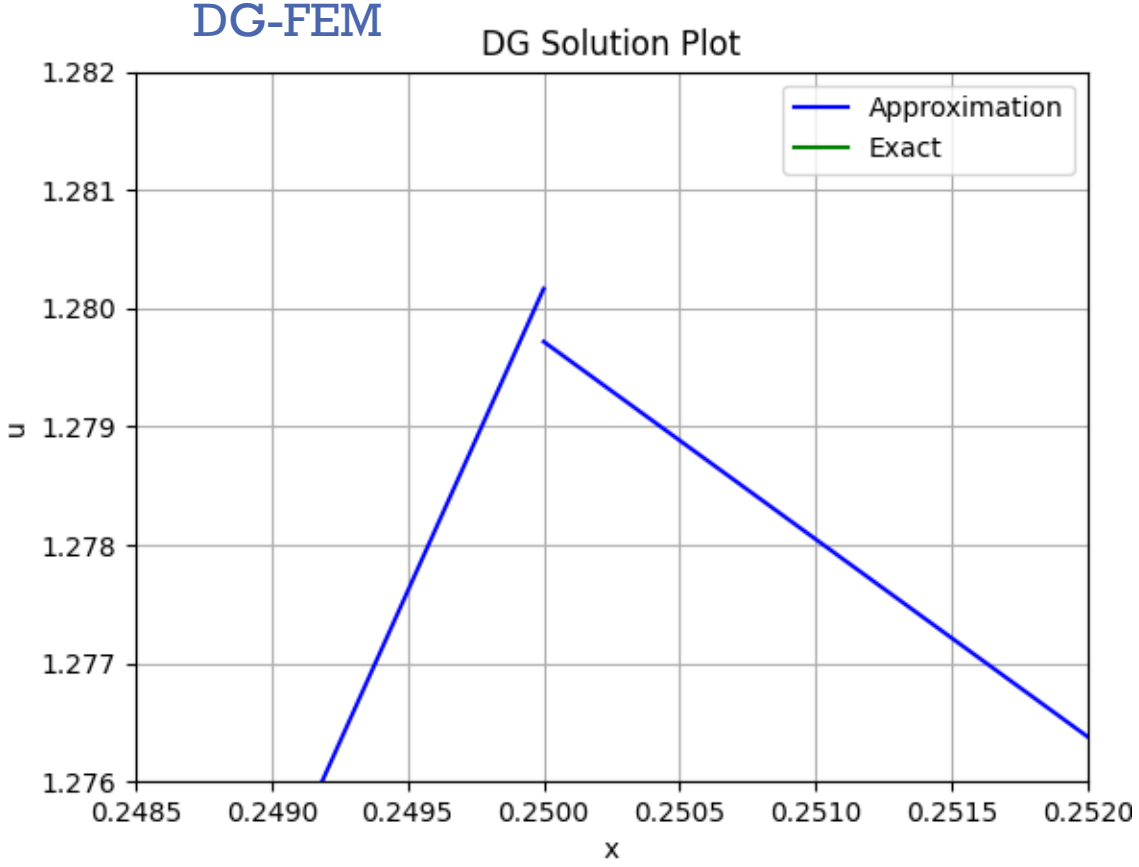
DG-FEM

CG-FEM

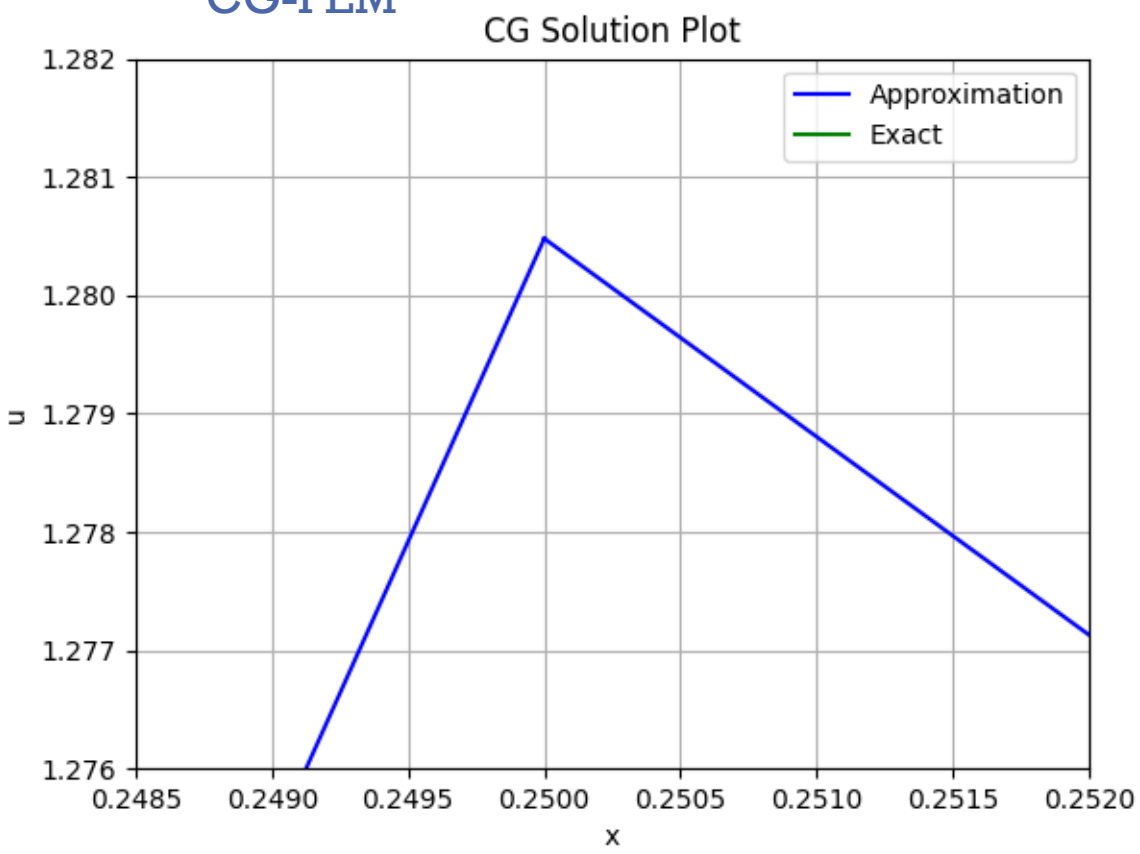
The Appetiser

Some Motivating Examples

DG-FEM



CG-FEM



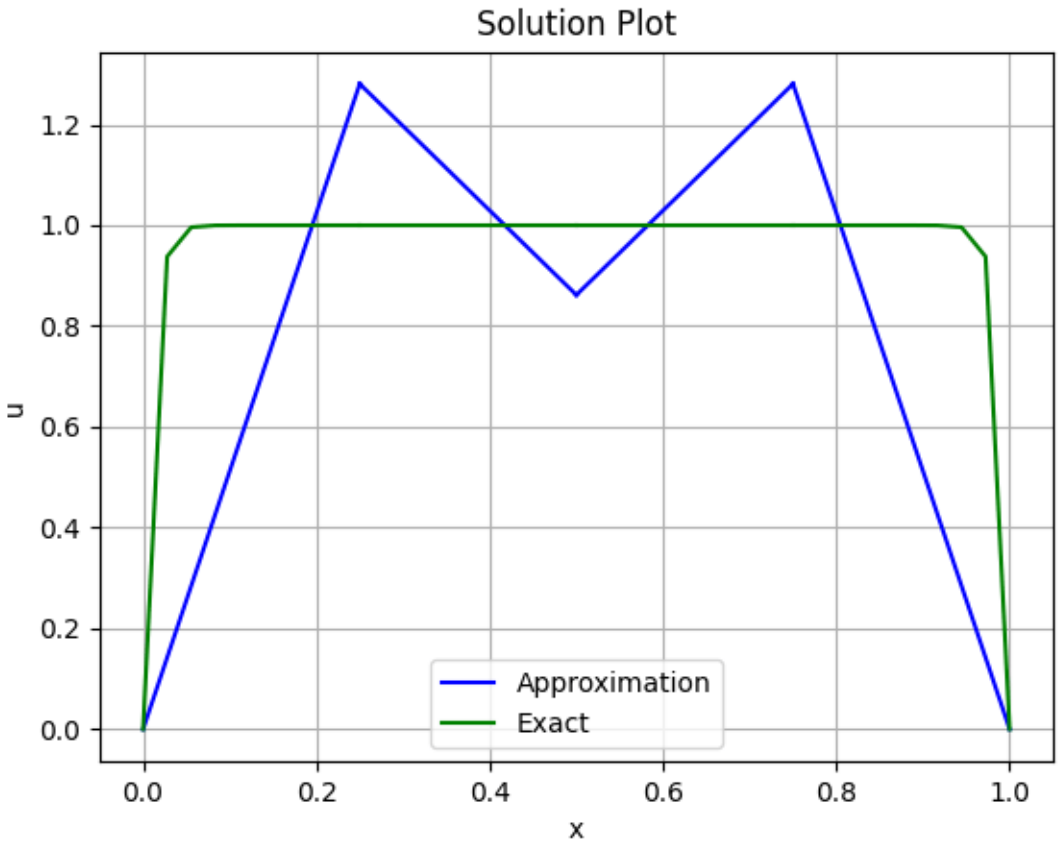
The Appetiser

Some Motivating Examples

Take the boundary layer problem:

$$-0.0001u'' + u = 1,$$
$$u(-1) = 0,$$
$$u(1) = 0.$$

With **4 higher-order** elements, we have…



The Appetiser

A Priori Error Analysis for CG

For a given problem, we can relatively easily construct an **a priori** error bound of the form:

$$\|u - u_h\|_E \leq \mathcal{E}(u, h, p).$$

We introduce a new error norm called the **energy norm**:

$$\|u\|_E := a(u, u)$$
$$\equiv \int \epsilon \nabla u \cdot \nabla u + \int cuv.$$

By [Schwab, 1998], we have:

$$\mathcal{E}(u, h, p) = C \frac{h^{\min(p,k)}}{p^k} \|u\|_{H^k(0,1)}.$$

For DG, we refer to [Castillo et al, 2007]

The Appetiser

A Posteriori Error Analysis for CG

For a given problem, we can relatively easily construct an **a posteriori** error bound of the form:

$$\|u - u_h\|_E \leq \mathcal{E}(u_h, h, p).$$

Following from an interpolation result from [Schwab, 1998], we obtain:

$$\mathcal{E}(u_h, h, p) = \epsilon \sqrt{\sum_{i=1}^{N} \eta_{\kappa_i}^2(u_h, h, p)}\,,$$

where $\eta_{\kappa_i}^2 := \frac{1}{p_i(p_i+1)} \frac{1}{\epsilon} \left\| w_i^{\frac{1}{2}} R(u_h) \right\|_{L^2(x_{i-1}, x_i)}^2$.

For our problem:

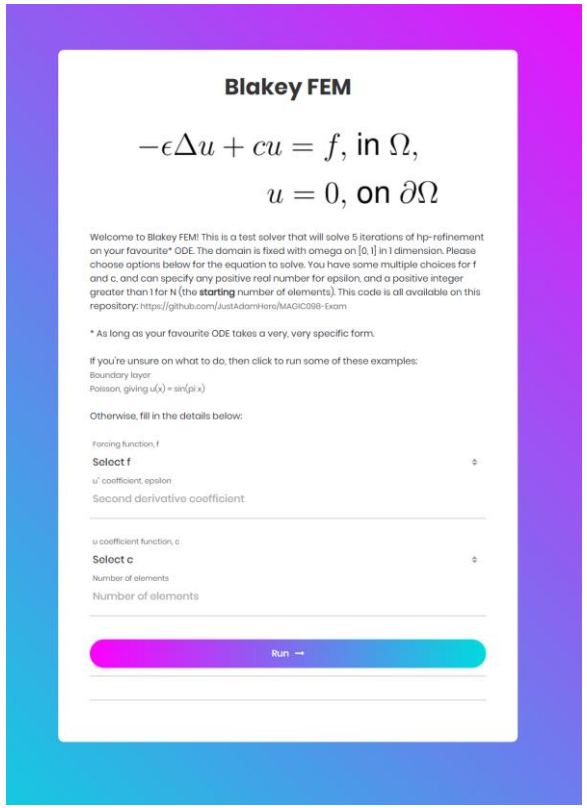
- $w_i := (x_i - x)(x - x_{i-1})$
- $R(u_h) := f + \epsilon u_h'' - c u_h$

For DG, we refer to [Houston et al, 2007]

A Quick Snack

A Simple Interactive Example

Go to fem.blakey.family to run the code for yourself!



https://fem.blakey.family

Sorbet

C++ Implementation

Programming Language

Various FEM software packages already exist in a number of languages. We decided to use C++ due to its:

- Object-oriented structures
- Advanced memory management
- Portability

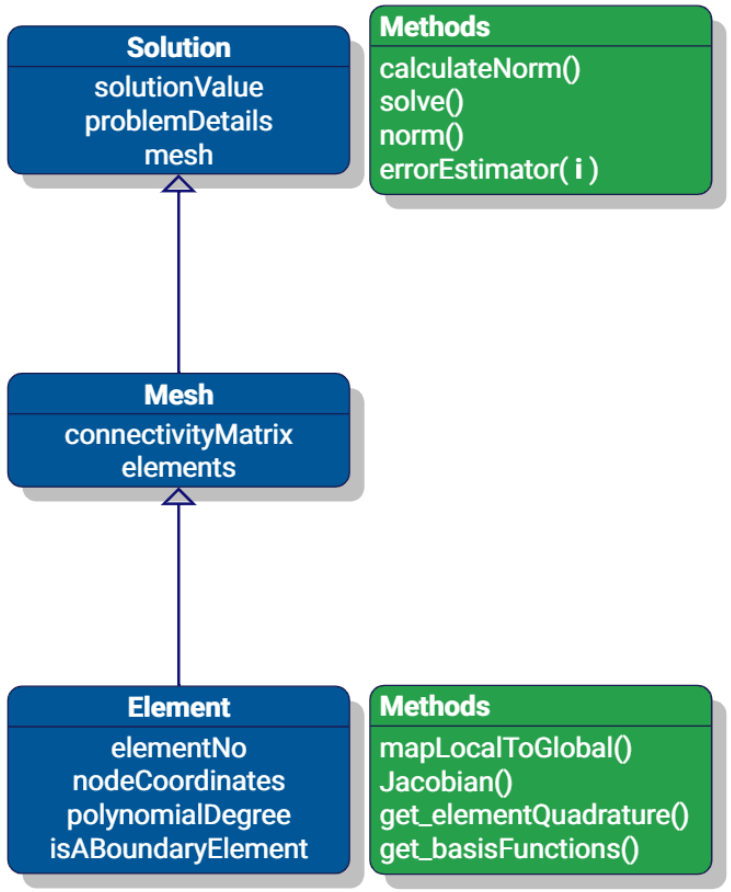
Programming Structure

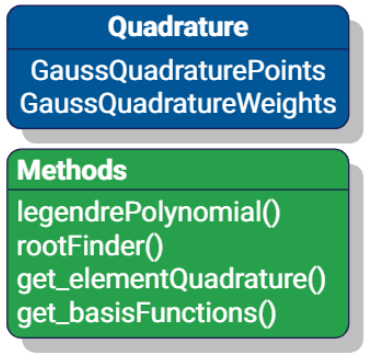
To make use of the object-oriented paradigm, we split our problem into human-understandable parts:

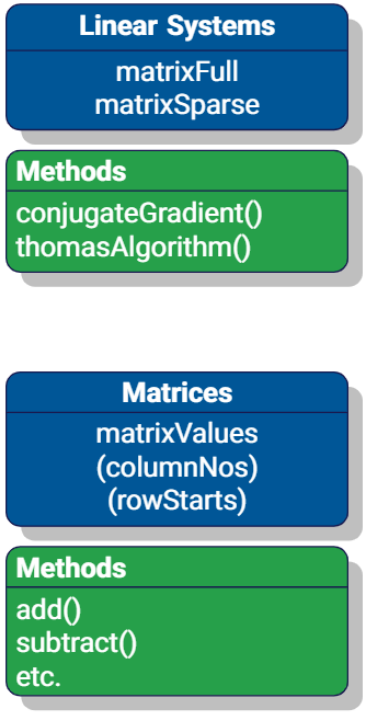
- Mesh
  - Elements
- Solution
  - CG solution
  - DG solution
- Linear systems
  - Matrix storage
  - Various linear solvers

Sorbet

C++ Implementation





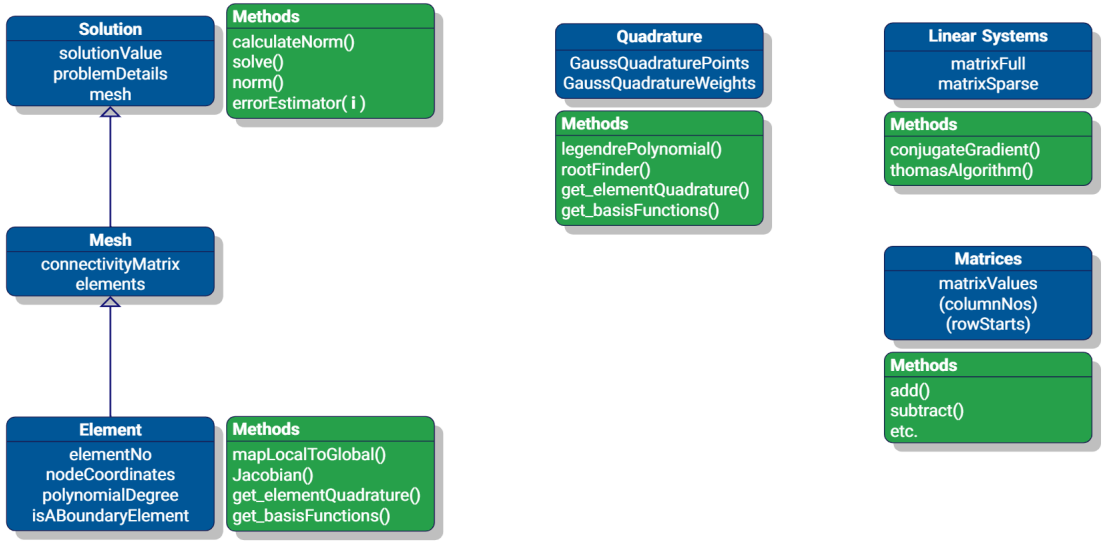


Sorbet

C++ Implementation

Primary C++ Ingredients

- Linear (and nonlinear) solvers
- Quadrature
  - Mappings to reference elements
- Efficient matrix storage
- FEM solvers
- ★ Refinement procedures
  - Error estimation



Sorbet

Numerical Experiments

Let's solve a simple ODE for which we know the exact solution, u, in order to study the convergence rates. Find $u \in V_h$ s.t.

$$\begin{aligned} -u_h'' &= \pi^2 \sin(\pi x)\,, \\ u_h(-1) &= 0, \\ u_h(1) &= 0, \end{aligned}$$

which exactly gives $u(x) = \sin(\pi x)$.

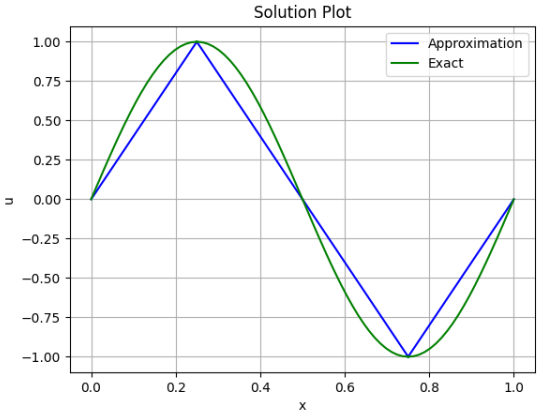
Error Measurement

There's no one way to measure the error of our approximate solution. We will focus on measuring in the following norms:

- L^2 norm
- H^1 norm
- Energy norm

Sorbet

Numerical Experiments

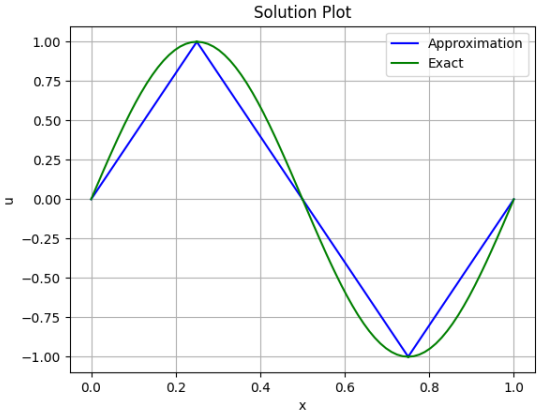


Our exact solution is $u(x) = \sin(\pi x)$. For h-refinement we get:

# DoFs	$\|u - u_h\|_{L^2(\Omega)}$	$\|u - u_h\|_{H^1(\Omega)}$	$\|u - u_h\|_{E(\Omega)}$	$\mathcal{E}(u_h, h, p)$
5	0.15101	1.93952	1.93364	2.01462
9	0.039292	0.997789	0.997015	1.00731
17	0.00992138	0.502461	0.502363	0.503656
33	0.00248653	0.251679	0.251666	0.251828
65	0.00062202	0.125895	0.125894	0.125914
129	0.000155529	0.0629547	0.0629545	0.062957
257	0.0000388838	0.0314782	0.0314782	0.0314785
513	0.00000972104	0.0157392	0.0157392	0.0157393

Sorbet

Numerical Experiments



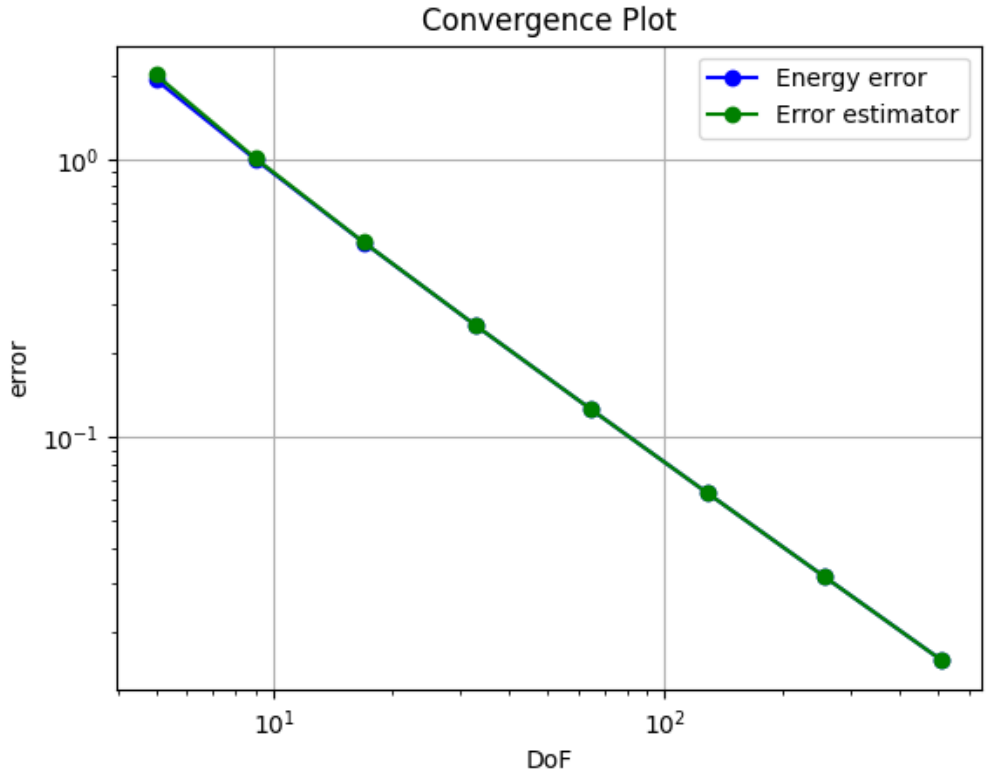
Our exact solution is $u(x) = \sin(\pi x)$. For h-refinement we get:

DoF ratio	$\|u - u_h\|_{L^2(\Omega)}$	$\|u - u_h\|_{H^1(\Omega)}$	$\|u - u_h\|_{E(\Omega)}$	$\mathcal{E}(u_h, h, p)$
-	-	-	-	-
5/9	3.84328	1.94382	1.93943	2
9/17	3.96033	1.9858	1.98465	2
17/33	3.99005	1.99644	1.99615	2
33/65	3.99751	1.99911	1.99904	2
65/129	3.99938	1.99978	1.99976	2
129/257	3.99984	1.99994	1.99994	2
257/513	3.99996	1.99999	1.99998	2
	~4	~2	~2	~2

Sorbet

Numerical Experiments

Our exact solution is $u(x) = \sin(\pi x)$. For h-refinement we get:



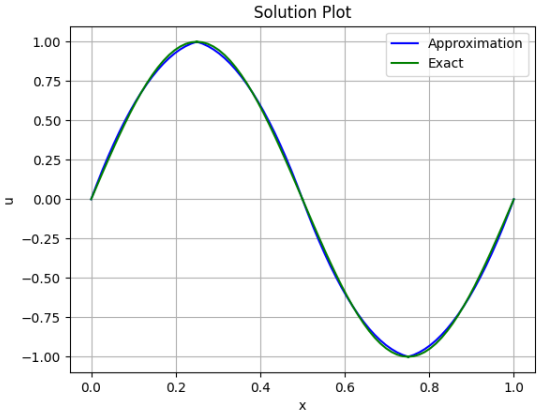
Efficiency index

$$\Theta := \frac{\mathcal{E}(u_h, h, p)}{\|u - u_h\|_E}$$

DoF	Θ
5	1.04187956
9	1.01032582
17	1.00257384
33	1.0006371
65	1.00015886
129	1.00003971
257	1.00000953

Sorbet

Numerical Experiments

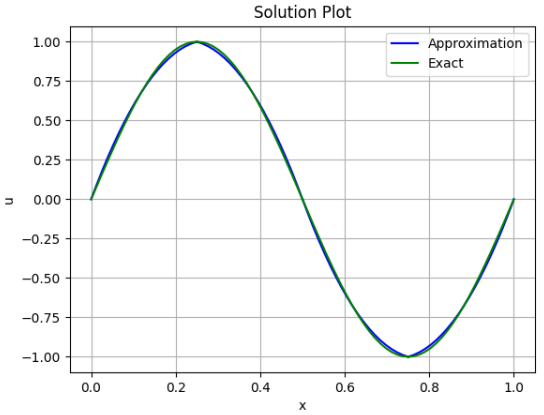


Our exact solution is $u(x) = \sin(\pi x)$. For p-refinement we get:

# DoFs	$\|u - u_h\|_{L^2(\Omega)}$	$\|u - u_h\|_{H^1(\Omega)}$	$\|u - u_h\|_{E(\Omega)}$	$\mathcal{E}(u_h, h, p)$
5	0.15101	1.93952	1.93364	2.01462
9	0.0151992	0.394667	0.394374	0.397937
13	0.00138912	0.0526826	0.0526643	0.05284
17	0.00010558	0.00523556	0.00523449	0.0052428
21	6.78332e-06	0.000414636	0.00041458	0.000414944
25	3.76084e-07	2.73023e-05	2.72997e-05	2.73143e-05
29	1.8312e-08	1.53869e-06	1.53858e-06	1.53912e-06
33	7.94325e-10	7.58027e-08	7.57986e-08	7.58169e-08

Sorbet

Numerical Experiments



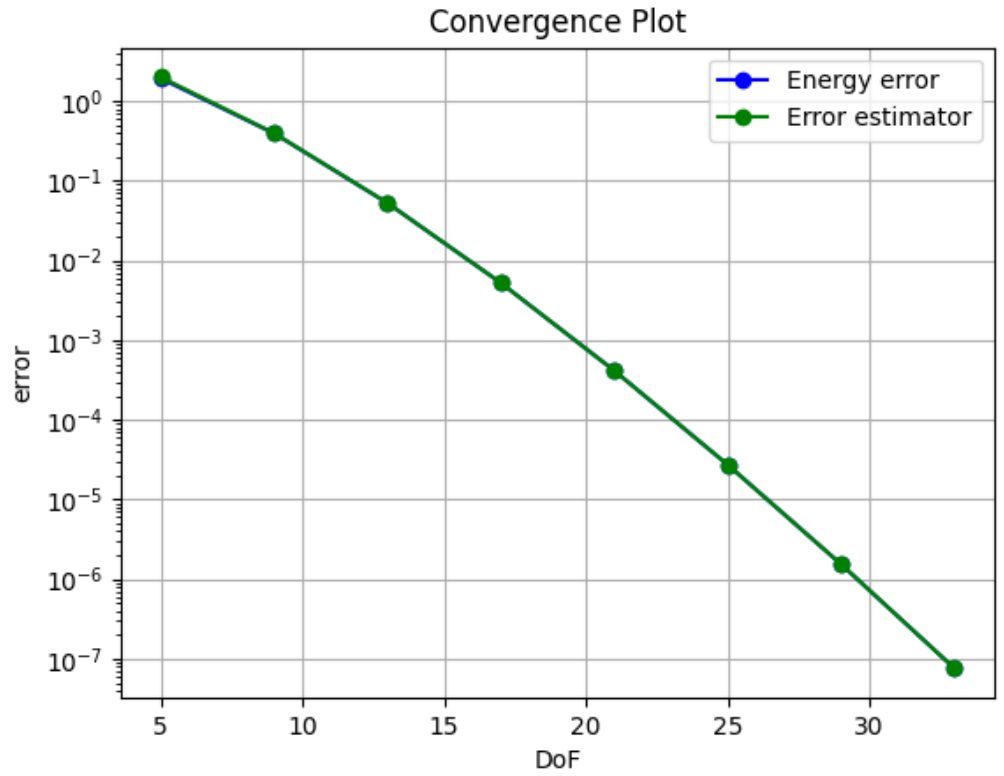
Our exact solution is $u(x) = \sin(\pi x)$. For p-refinement we get:

DoF ratio	$\|u - u_h\|_{L^2(\Omega)}$	$\|u - u_h\|_{H^1(\Omega)}$	$\|u - u_h\|_{E(\Omega)}$	$\mathcal{E}(u_h, h, p)$
-	-	-	-	-
5/9	9.93536	4.91433	4.90305	5.06268
9/13	10.9416	7.4914	7.48844	7.53098
13/17	13.157	10.0625	10.061	10.0786
17/21	15.5647	12.6269	12.626	12.635
21/25	18.0367	15.1869	15.1863	15.1915
25/29	20.5376	17.7438	17.7434	17.7467
29/33	23.0536	20.2986	20.2983	20.3005
	~exp	~exp	~exp	~exp

Sorbet

Numerical Experiments

Our exact solution is $u(x) = \sin(\pi x)$. For p-refinement we get:



Efficiency index

$$\Theta := \frac{\mathcal{E}(u_h, h, p)}{\|u - u_h\|_E}$$

DoF	Θ
5	1.04187956
9	1.00903457
13	1.00333623
17	1.00158755
21	1.000878
25	1.0005348
29	1.00035097

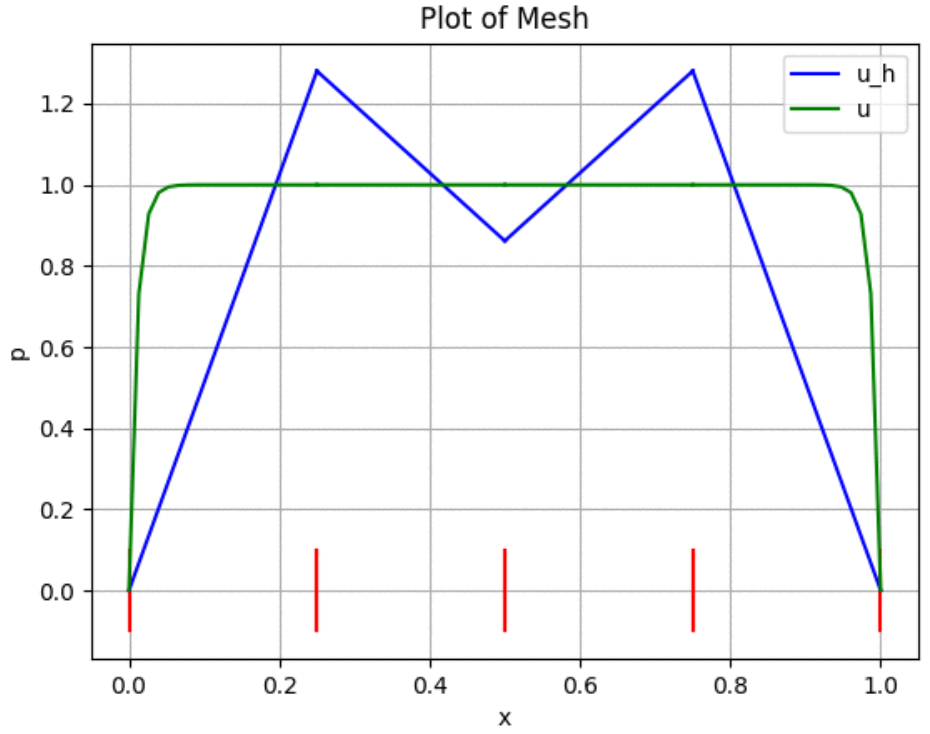
Main Course

hp-FEMs in Action

We introduce our next recipe, for automatic adaptive h-, and p-refinement.

Recipe:

1. Solve
2. Estimate all η_{κ_i}
3. Mark large η_{κ_i}
4. Refine marked



Main Course

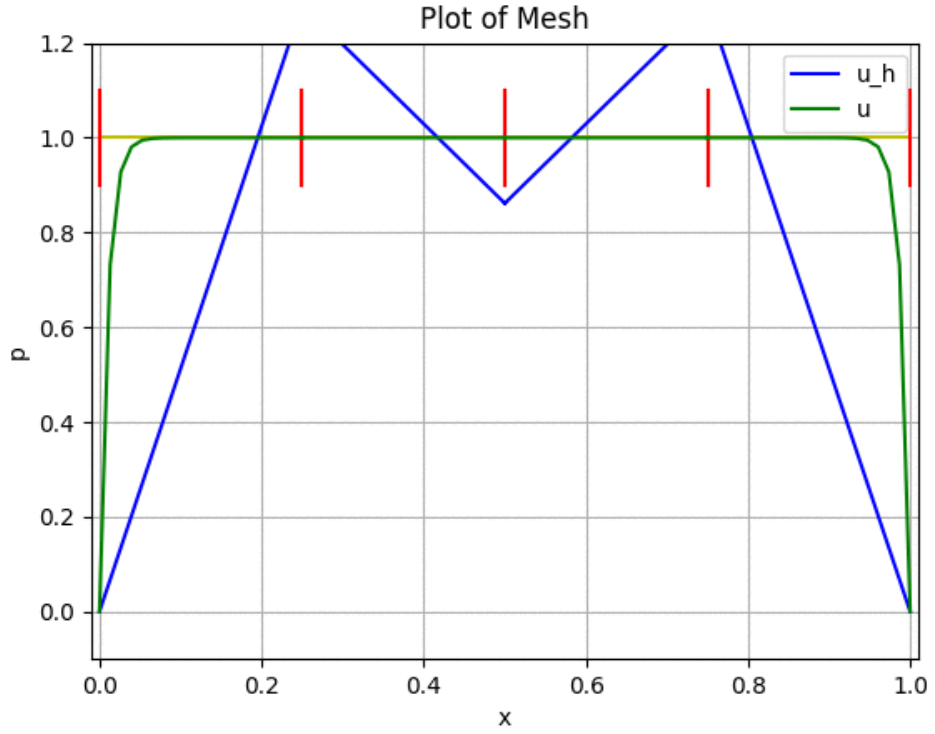
hp-FEMs in Action

We introduce our next recipe, for automatic adaptive h-, and p-refinement.

Recipe:

1. Solve
2. Estimate all η_{κ_i}
3. Mark large η_{κ_i}
4. Refine marked

What about p-refinement?



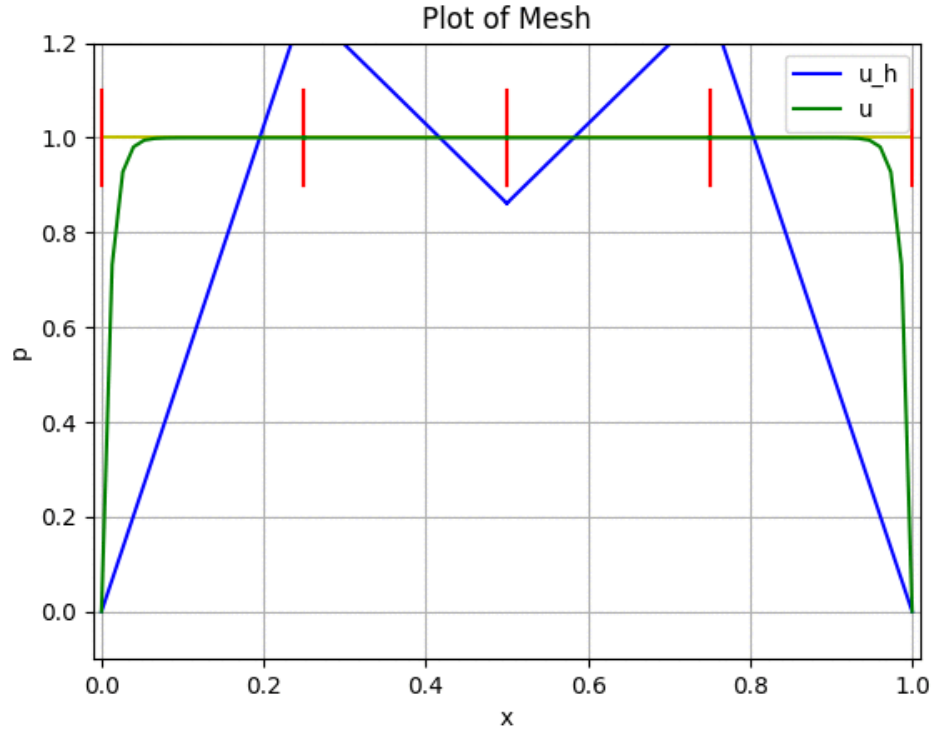
Main Course

hp-FEMs in Action

We introduce our next recipe, for automatic adaptive hp-refinement.

Modified Recipe:

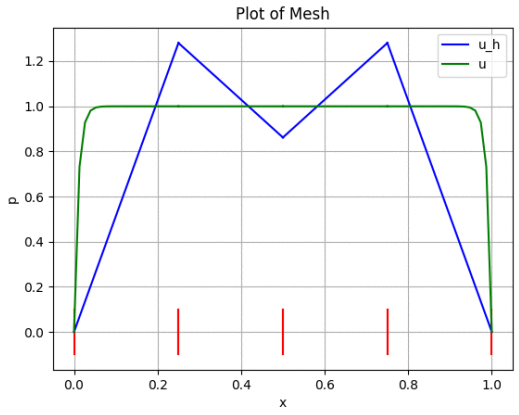
1. Solve
2. Estimate all η_{κ_i}
3. Mark large η_{κ_i}
4. Decide refinement on h or p
5. Refine marked

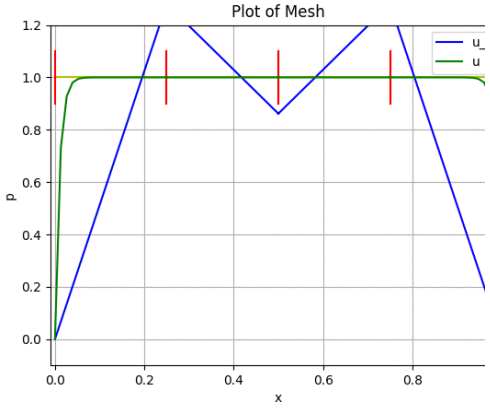


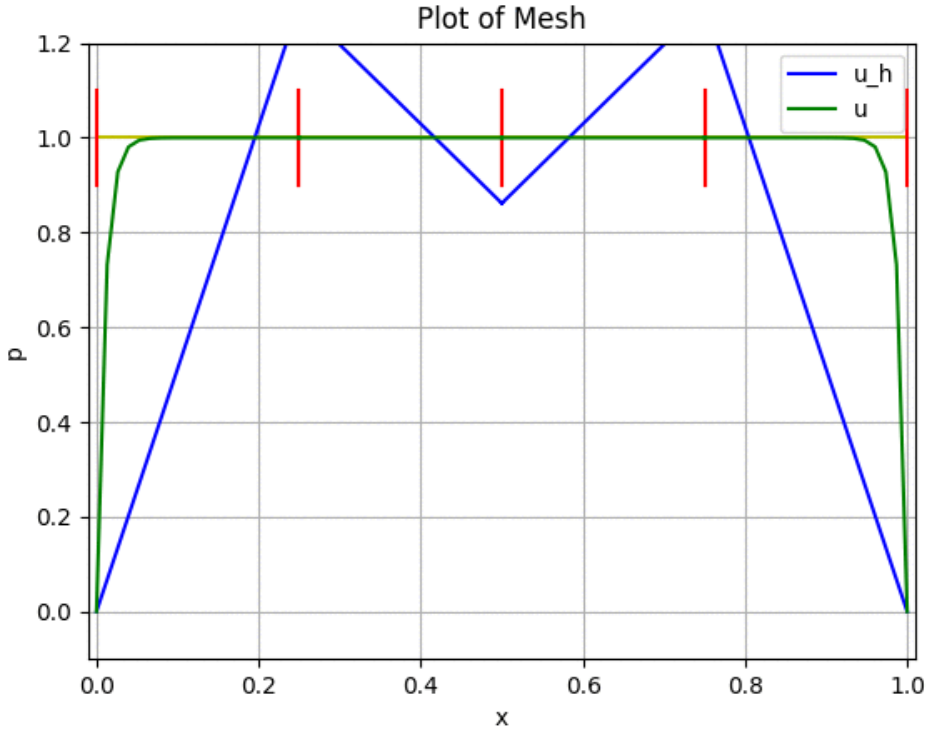
Main Course

hp-FEMs in Action

We introduce our next recipe, for automatic adaptive *hp*-refinement.



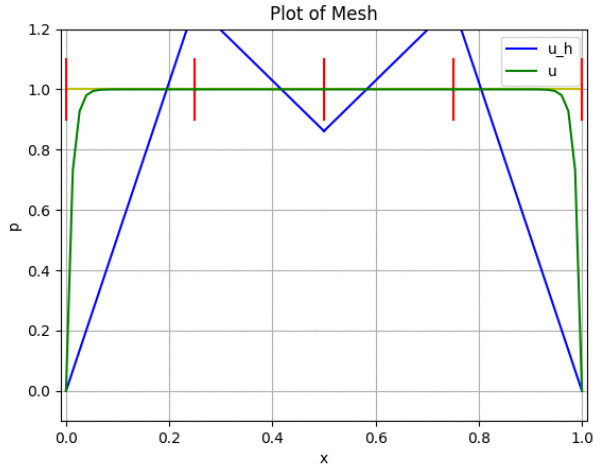


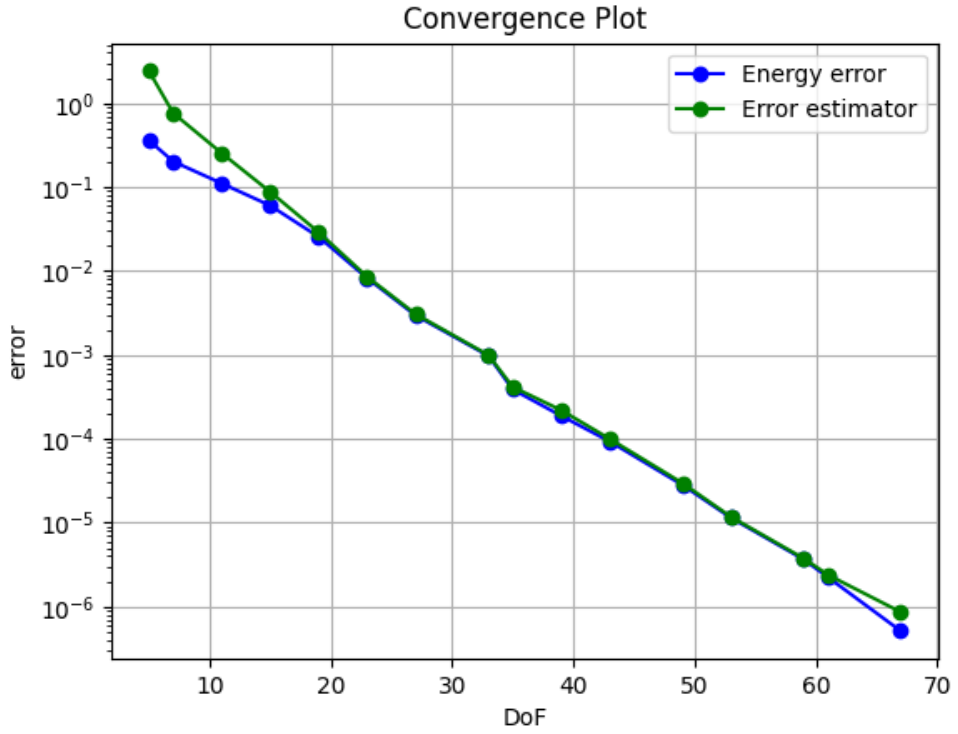


Main Course

hp-FEMs in Action

We introduce our next recipe, for automatic adaptive *hp*-refinement.





Main Course

hp-FEMs in Action

How do we choose to mark large η_{κ_i}?

- ★ Maximum strategy
  - Pick $\gamma \in (0,1)$
  - Mark when $\eta_{\kappa_i} > \gamma \max \eta_{\kappa_i}$
  - Refine all marked $\hat{\eta}_{\kappa_i}$
- Dörfler strategy
  - Pick $\gamma \in (0,1)$
  - Sort elements by largest indicator
  - Continue until marked elements satisfy $\sum_i \hat{\eta}_{\kappa_i} > \gamma \sum_i \eta_{\kappa_i}$
  - Refine all marked $\hat{\eta}_{\kappa_i}$

Main Course

hp-FEMs in Action

How do we choose whether to h- or p-refine?

In 1D, we can derive the following "smoothness indicator" [Wihler, 2011]:

$$\mathcal{F}_\kappa(u_h) := \begin{cases} \|u_h\|^2_{\infty(\kappa)} \left[\coth(1)\left(h_\kappa^{-1}\|u_h\|^2_{L^2(\kappa)} + h_\kappa |u_h|^2_{H^1(\kappa)}\right)\right]^{-1} & u_h \not\equiv 0, \\ 1 & u_h \equiv 0. \end{cases}$$

- We expect smooth functions to have $\mathcal{F}_\kappa(u_h) \approx 1$ and non-smooth functions to have $\mathcal{F}_\kappa(u_h) \approx 0$.
- We choose to refine h when $\mathcal{F}_\kappa(u_h) \leq 0.5$, and refine p otherwise.

Main Course

hp-FEMs in Action

Why was all of that OOP necessary?

Example Boundary Layer Simulation

```
int main()
{
    // Sets up problem.
    Mesh*            myMesh     = new Mesh(4, 1);
    Solution_linear* mySolution = new Solution_linear(myMesh, one, 1e-4, one);

    // Solves the problem, and then outputs solution and mesh to files.
    mySolution->Solve(1e-15);
    mySolution->output_solution(exact);
    mySolution->output_mesh();

    delete mySolution;
    delete myMesh;

    return 0;
}
```

hp-Adaptive Boundary Layer Simulation

```
int main()
{
    // Sets up problem.
    Mesh*     myMesh     = new Mesh(4);
    Solution_linear* mySolution = new Solution_linear(myMesh, one, 1e-4, one);

    // Adaptivity variables.
    Mesh*            myNewMesh;
    Solution_linear* myNewSolution_linear;
    Solution*        myNewSolution = myNewSolution_linear;

    // Performs the refinement with the correct type of adaptivity.
    refinement::refinement(myMesh, &myNewMesh, mySolution, &myNewSolution, 1e-15,
         1e-10, 16, true, true, true, exact, exact_);

    // Solves the new problem, and then outputs solution and mesh to files.
    myNewSolution->output_solution(exact);
    myNewSolution->Solve(1e-15);
    myNewSolution->output_mesh();

    delete mySolution;
    delete myMesh;

    return 0;
}
```

Pudding

hp-FEMs in Action

Summary

- CG-FEM and DG-FEM in 1D
- Key implementation aspects
- A priori and a posteriori error bounds
- h- and p-refinement
- A posteriori error bounds
  - → adaptive h- and p-refinement
- Smoothness indicator
  - → adaptive hp-refinement

Outlook

- Convection-dominated PDEs
- Generalisation to 2D and 3D
- Time dependant problems
- Placentas
  - Complicated geometries
  - Fluid-structure interaction