Bachelor Thesis

Building Blocks for Finite Element computations in IPPL

Lukas Bühler

Tuesday 30th January, 2024

Outline

1. Introduction
2. Framework & Implementation
3. Results & Conclusion

Introduction

Currently, IPPL supports electrostatic PIC simulations.

Development of a full electromagnetic (EM) solver is ongoing.

The Finite Element Method (FEM) is one of the numerical methods used in EM solvers.

Goal: Implement the building blocks for the FEM in IPPL.

EM Solver Schemes

Common in EM solvers: Finite Difference Time Domain (FDTD) scheme. (2nd-order accuracy)

- Space discretization with Finite Differences.
- Time discretization with Finite Differences.

EM Solver Schemes

Common in EM solvers: Finite Difference Time Domain (FDTD) scheme. (2nd-order accuracy)

- Space discretization with Finite Differences.
- Time discretization with Finite Differences.

Finite Element Time Domain (FETD)

- Space discretization using FEM.
- Time discretization with other schemes, Runge-Kutta methods.

Advantages of FEM vs. Finite Differences

Advantages of using FEM compared to Finite Differences

- More complex geometries

Advantages of FEM vs. Finite Differences

Advantages of using FEM compared to Finite Differences

- More complex geometries
- Higher-order elements: Using higher-order basis functions (p-refinement)
  Improve accuracy without affecting runtime, scalability and memory footprint.

Advantages of FEM vs. Finite Differences

Advantages of using FEM compared to Finite Differences

- More complex geometries
- Higher-order elements: Using higher-order basis functions (p-refinement)
  Improve accuracy without affecting runtime, scalability and memory footprint.
- Still possible to do mesh refinement (h-refinement).

Advantages of FEM vs. Finite Differences

Advantages of using FEM compared to Finite Differences

- More complex geometries
- Higher-order elements: Using higher-order basis functions (p-refinement)
  Improve accuracy without affecting runtime, scalability and memory footprint.
- Still possible to do mesh refinement (h-refinement).
- ... or both (hp-refinement).

Advantages of FEM vs. Finite Differences

Advantages of using FEM compared to Finite Differences

- More complex geometries
- Higher-order elements: Using higher-order basis functions (p-refinement)
  Improve accuracy without affecting runtime, scalability and memory footprint.
- Still possible to do mesh refinement (h-refinement).
- ... or both (hp-refinement).
- Even smaller memory footprint with a matrix-free assembly algorithm.
  $\Rightarrow$ Better performance on GPUs.

Finite Element Method (FEM)

The Finite Element Method is used to solve PDEs.

Steps:

1. Discretization (Meshing) of the domain with elements.
2. Approximating the solution of the PDE on the elements.
3. Assembling the approximated solutions of the elements in a LSE.

PAUL SCHERRER INSTITUT

Finite Element Method (FEM)

The Finite Element Method is used to solve PDEs.

Steps:

1. Discretization (Meshing) of the domain with elements.
2. Approximating the solution of the PDE on the elements.
3. Assembling the approximated solutions of the elements in a LSE.

$$\mathbf{A}\boldsymbol{\mu} = \boldsymbol{\varphi} \tag{1}$$

The LSE then needs to be solved.

FEM produces sparse problems $\Rightarrow$ Use a sparse solver.

Finite Element Method (FEM)

The Finite Element Method is used to solve PDEs.

Steps:

1. Discretization (Meshing) of the domain with elements.
2. Approximating the solution of the PDE on the elements.
3. Assembling the approximated solutions of the elements in a LSE.

$$\mathbf{A}\boldsymbol{\mu} = \boldsymbol{\varphi} \tag{1}$$

The LSE then needs to be solved.

FEM produces sparse problems $\Rightarrow$ Use a sparse solver.

We use the Conjugate Gradient (CG) method to iteratively solve the LSE. This allows us to use a matrix-free assembly method.

Motivation for Matrix-free Method

Ljungkvist, 2017: Matrix-free finite element algorithms have many benefits on modern manycore processors and GPUs compared to sparse matrix-vector products. [1]

Settgast et. al, 2023: The matrix-free approach, in the context of the CG method, compares favorably even for low-order FEM. [2]

Conjugate Gradient (CG) Method

The CG method is an iterative method to approximate the solution of a LSE.

$\boldsymbol{x} \leftarrow$ initial guess, (usually $\mathbf{0}$)
$\boldsymbol{b} \leftarrow \boldsymbol{\varphi}$
$\boldsymbol{p} \leftarrow \mathbf{A}\boldsymbol{x}$
$\boldsymbol{r} \leftarrow \boldsymbol{b} - \boldsymbol{p}$
while $\underline{\|\boldsymbol{r}\|_2 < \epsilon}$ **do**

$$\boldsymbol{z} \leftarrow \mathbf{A}\boldsymbol{p}$$

$$\alpha \leftarrow \frac{\boldsymbol{r}^\top \boldsymbol{r}}{\boldsymbol{p}^\top \boldsymbol{z}}$$

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha \boldsymbol{p}$$

$$\boldsymbol{r}_{\text{old}} \leftarrow \boldsymbol{r}$$

$$\boldsymbol{r} \leftarrow \boldsymbol{r} - \alpha \boldsymbol{z}$$

$$\beta \leftarrow \frac{\boldsymbol{r}^\top \boldsymbol{r}}{\boldsymbol{r}_{\text{old}}^\top \boldsymbol{r}_{\text{old}}}$$

$$\boldsymbol{p} \leftarrow \boldsymbol{r} + \beta \boldsymbol{p}$$

end

Conjugate Gradient (CG) Method

The CG method is an iterative method to approximate the solution of a LSE.

$\boldsymbol{x} \leftarrow$ initial guess, (usually $\mathbf{0}$)
$\boldsymbol{b} \leftarrow \boldsymbol{\varphi}$
$\boldsymbol{p} \leftarrow \mathbf{A}\boldsymbol{x}$ `// Stiffness matrix used here`
$\boldsymbol{r} \leftarrow \boldsymbol{b} - \boldsymbol{p}$
while $\underline{\|\boldsymbol{r}\|_2 < \epsilon}$ **do**

$\quad \boldsymbol{z} \leftarrow \mathbf{A}\boldsymbol{p}$ `// Stiffness matrix used here`

$$\alpha \leftarrow \frac{\boldsymbol{r}^\top \boldsymbol{r}}{\boldsymbol{p}^\top \boldsymbol{z}}$$

$\quad \boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha \boldsymbol{p}$
$\quad \boldsymbol{r}_{\text{old}} \leftarrow \boldsymbol{r}$
$\quad \boldsymbol{r} \leftarrow \boldsymbol{r} - \alpha \boldsymbol{z}$

$$\beta \leftarrow \frac{\boldsymbol{r}^\top \boldsymbol{r}}{\boldsymbol{r}_{\text{old}}^\top \boldsymbol{r}_{\text{old}}}$$

$\quad \boldsymbol{p} \leftarrow \boldsymbol{r} + \beta \boldsymbol{p}$
end

Outline

FEMPoissonSolver

The `FEMPoissonSolver` is a proof of concept FEM solver.

It solves the Poisson equation for a given right-hand side function.

$$\begin{aligned} -\Delta u &= f \qquad u \in \Omega, \\ u &= 0 \qquad u \in \partial\Omega. \end{aligned} \tag{2}$$

Currently uses:

- Homogeneous Dirichlet boundary conditions
- IPPL uniform meshes
- 1st-order Lagrangian finite elements
- Gauss-Jacobi quadrature

PAUL SCHERRER INSTITUT

Solver and Assembly

What `FEMPoissonSolver` does:

1. Precompute the transformations, the quadrature weights and quadrature nodes
2. Define the "eval" lambda function (for the Poisson equation)
3. Use the CG solver with the matrix-free assembly function (`evaluateAx`) to solve the problem

Solver and Assembly

What `FEMPoissonSolver` does:

1. Precompute the transformations, the quadrature weights and quadrature nodes
2. Define the "eval" lambda function (for the Poisson equation)
3. Use the CG solver with the matrix-free assembly function (`evaluateAx`) to solve the problem

$$\boldsymbol{z} = \mathbf{A}\boldsymbol{x}$$

Solver and Assembly

What `FEMPoissonSolver` does:

1. Precompute the transformations, the quadrature weights and quadrature nodes
2. Define the "eval" lambda function (for the Poisson equation)
3. Use the CG solver with the matrix-free assembly function (`evaluateAx`) to solve the problem

$$\boldsymbol{z} = \texttt{evaluateAx}(\texttt{eval}, \boldsymbol{x})$$

PAUL SCHERRER INSTITUT

Solver and Assembly

What `FEMPoissonSolver` does:

1. Precompute the transformations, the quadrature weights and quadrature nodes
2. Define the "eval" lambda function (for the Poisson equation)
3. Use the CG solver with the matrix-free assembly function (`evaluateAx`) to solve the problem

$$\boldsymbol{z} = \texttt{evaluateAx}(\texttt{eval}, \boldsymbol{x})$$

For the Poisson equation:

$$\texttt{eval}(i,j,k) := (\mathbf{D\Phi}_K(\hat{\boldsymbol{q}}_k))^{-\top} \nabla \hat{b}^j(\hat{\boldsymbol{q}}_k) \cdot (\mathbf{D\Phi}(\hat{\boldsymbol{q}}_k))^{-\top} \nabla \hat{b}^i(\hat{\boldsymbol{q}}_k) \, |\det \mathbf{D\Phi}_K(\hat{\boldsymbol{q}}_k)|$$

Assembly Prerequisites

Strong form:

$$\begin{aligned} -\Delta u &= f \qquad u \in \Omega, \\ u &= 0 \qquad u \in \partial\Omega. \end{aligned} \tag{2}$$

Assembly Prerequisites

Strong form:

$$\begin{aligned} -\Delta u &= f \qquad u \in \Omega, \\ u &= 0 \qquad u \in \partial\Omega. \end{aligned} \tag{2}$$

Weak form (variational formulation):

$$\int_\Omega \nabla u \cdot \nabla v \; d\boldsymbol{x} = \int_\Omega f \, v \; d\boldsymbol{x}. \tag{3}$$

Assembly Prerequisites

Strong form:

$$\begin{aligned} -\Delta u &= f \qquad u \in \Omega, \\ u &= 0 \qquad u \in \partial\Omega. \end{aligned} \tag{2}$$

Weak form (variational formulation):

$$\underbrace{\int_\Omega \nabla u \cdot \nabla v \; d\boldsymbol{x}}_{=:\, \mathrm{a}(u,v)} = \underbrace{\int_\Omega f\, v \; d\boldsymbol{x}}_{=:\, \ell(v)} . \tag{3}$$

Assembly Prerequisites

Strong form:

$$\begin{aligned} -\Delta u &= f \qquad u \in \Omega, \\ u &= 0 \qquad u \in \partial\Omega. \end{aligned} \tag{2}$$

Weak form (variational formulation):

$$\underbrace{\int_\Omega \nabla u \cdot \nabla v \; d\boldsymbol{x}}_{=:\, \mathrm{a}(u,v)} = \underbrace{\int_\Omega f\, v \; d\boldsymbol{x}}_{=:\, \ell(v)} . \tag{3}$$

(Element) stiffness matrix and load vector:

$$\mathbf{A} = \left[\mathrm{a}(b_h^J, b_h^I)\right]_{I,J=1}^{N}, \qquad \boldsymbol{\varphi} = \left[\ell(b_h^I)\right]_{I=1}^{N}, \tag{4}$$

$$\mathbf{A}_K = \left[\mathrm{a}(b_K^j, b_K^i)\right]_{i,j=1}^{M}, \qquad \boldsymbol{\varphi}_K = \left[\ell(b_K^i)\right]_{i=1}^{M}, \tag{5}$$

with $b_K^i := b_{h|K}^i$, for element K,
N number of global basis functions, M number of local shape functions.

Matrix-free Assembly Derivation (`evaluateAx`)

Input: $\boldsymbol{x}$

$\boldsymbol{z} \leftarrow \mathbf{0}$

for Element K in Mesh **do**

  `// 1. Compute the Element matrix` $\mathbf{A}_K$

  $\mathsf{DOFs}_K \leftarrow \{4, 5, 8, 7\}$, $\mathsf{DOFs}_{\hat{K}} \leftarrow \{0, 1, 2, 3\}$

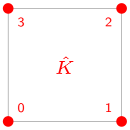
  ...

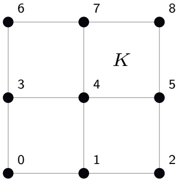
  `// 2. Compute` $\boldsymbol{z} = \mathbf{A}\boldsymbol{x}$ `contribution with` $\mathbf{A}_K$

  ...

end

return $\boldsymbol{z}$





Matrix-free Assembly Derivation (`evaluateAx`)

Input: $\boldsymbol{x}$

$\boldsymbol{z} \leftarrow \mathbf{0}$

for Element K in Mesh **do**

  `// 1. Compute the Element matrix` $\mathbf{A}_K$

  $\mathsf{DOFs}_K \leftarrow \{4, 5, 8, 7\}$, $\mathsf{DOFs}_{\hat{K}} \leftarrow \{0, 1, 2, 3\}$

  for $i, j \in \mathsf{DOFs}_{\hat{K}}$ **do**

    $I = \mathsf{DOFs}_K[i],\ J = \mathsf{DOFs}_K[j]$

    $(\mathbf{A}_K)_{i,j} = \int_K \nabla b_K^J(\boldsymbol{x}) \cdot \nabla b_K^I(\boldsymbol{x})\ d\boldsymbol{x}$

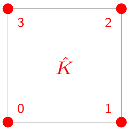
  end

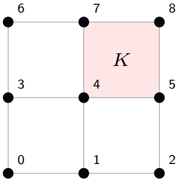
  `// 2. Compute` $\boldsymbol{z} = \mathbf{A}\boldsymbol{x}$ `contribution with` $\mathbf{A}_K$

  ...

end

return $\boldsymbol{z}$





Matrix-free Assembly Derivation (`evaluateAx`)

Input: $\boldsymbol{x}$

$\boldsymbol{z} \leftarrow \mathbf{0}$

for Element K in Mesh **do**

  `// 1. Compute the Element matrix` $\mathbf{A}_K$

  DOFs$_K \leftarrow \{4, 5, 8, 7\}$, DOFs$_{\hat{K}} \leftarrow \{0, 1, 2, 3\}$

  for $i, j \in$ DOFs$_{\hat{K}}$ **do**

    $I =$ DOFs$_K[i]$, $J =$ DOFs$_K[j]$

    $$(\mathbf{A}_K)_{i,j} = \int_{\hat{K}} \mathbf{\Phi}_K^* \nabla b_K^J \cdot \mathbf{\Phi}_K^* \nabla b_K^I \, |\det \mathbf{D}\mathbf{\Phi}_K| \, d\hat{\boldsymbol{x}}$$

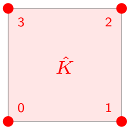
  end

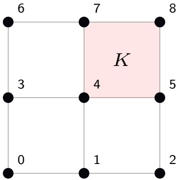
  `// 2. Compute` $\boldsymbol{z} = \mathbf{A}\boldsymbol{x}$ `contribution with` $\mathbf{A}_K$

  ...

end

return $\boldsymbol{z}$





Matrix-free Assembly Derivation (`evaluateAx`)

Input: $\boldsymbol{x}$

$\boldsymbol{z} \leftarrow \mathbf{0}$

for Element K in Mesh **do**

    `// 1. Compute the Element matrix` $\mathbf{A}_K$

    DOFs$_K \leftarrow \{4, 5, 8, 7\}$, DOFs$_{\hat{K}} \leftarrow \{0, 1, 2, 3\}$

    for $i, j \in$ DOFs$_{\hat{K}}$ **do**

$$(\mathbf{A}_K)_{i,j} = \int_{\hat{K}} (\mathbf{D}\mathbf{\Phi}_K)^{-\top} \nabla \hat{b}^j \cdot (\mathbf{D}\mathbf{\Phi}_K)^{-\top} \nabla \hat{b}^i |\det \mathbf{D}\mathbf{\Phi}_K| \, d\hat{\boldsymbol{x}}$$

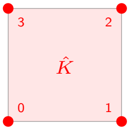
    end

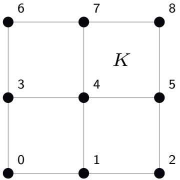
    `// 2. Compute` $\boldsymbol{z} = \mathbf{A}\boldsymbol{x}$ `contribution with` $\mathbf{A}_K$

    ...

end

return $\boldsymbol{z}$





Matrix-free Assembly Derivation (`evaluateAx`)

Input: $\boldsymbol{x}$

$\boldsymbol{z} \leftarrow \mathbf{0}$

for Element K in Mesh **do**

// 1. Compute the Element matrix $\mathbf{A}_K$

DOFs$_K \leftarrow \{4, 5, 8, 7\}$, DOFs$_{\hat{K}} \leftarrow \{0, 1, 2, 3\}$

for $i, j \in$ DOFs$_{\hat{K}}$ **do**

$$(\mathbf{A}_K)_{i,j} \approx \sum_{k}^{N_{\text{Int}}} \hat{\omega}_k (\mathbf{D}\mathbf{\Phi}_K(\hat{\boldsymbol{q}}_k))^{-\top} \nabla \hat{b}^j(\hat{\boldsymbol{q}}_k) \cdot (\mathbf{D}\mathbf{\Phi}(\hat{\boldsymbol{q}}_k))^{-\top} \nabla \hat{b}^i(\hat{\boldsymbol{q}}_k) |\det \mathbf{D}\mathbf{\Phi}_K(\hat{\boldsymbol{q}}_k)|$$

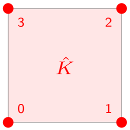
end

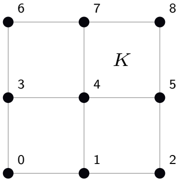
// 2. Compute $\boldsymbol{z} = \mathbf{A}\boldsymbol{x}$ contribution with $\mathbf{A}_K$

...

end

return $\boldsymbol{z}$





Matrix-free Assembly Derivation (`evaluateAx`)

Input: $\boldsymbol{x}$

$\boldsymbol{z} \leftarrow \mathbf{0}$

for Element K in Mesh **do**

  // 1. Compute the Element matrix $\mathbf{A}_K$

  $\text{DOFs}_K \leftarrow \{4, 5, 8, 7\}$, $\text{DOFs}_{\hat{K}} \leftarrow \{0, 1, 2, 3\}$

  for $i, j \in \text{DOFs}_{\hat{K}}$ **do**

$$(\mathbf{A}_K)_{i,j} \approx \sum_{k}^{N_{\text{Int}}} \hat{\omega}_k (\mathbf{D}\boldsymbol{\Phi}_K(\hat{\boldsymbol{q}}_k))^{-\top} \nabla \hat{b}^j(\hat{\boldsymbol{q}}_k) \cdot (\mathbf{D}\boldsymbol{\Phi}(\hat{\boldsymbol{q}}_k))^{-\top} \nabla \hat{b}^i(\hat{\boldsymbol{q}}_k) |\det \mathbf{D}\boldsymbol{\Phi}_K(\hat{\boldsymbol{q}}_k)|$$

  end

  // 2. Compute $\boldsymbol{z} = \mathbf{A}\boldsymbol{x}$ contribution with $\mathbf{A}_K$

  for $i, j \in \text{DOFs}_{\hat{K}}$ **do**

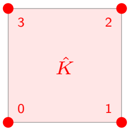
    $I = \text{DOFs}_K[i]$, $J = \text{DOFs}_K[j]$

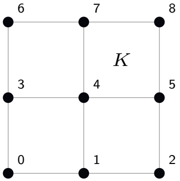
    $\boldsymbol{z}_I \leftarrow \boldsymbol{z}_I + (\mathbf{A}_K)_{i,j} \cdot \boldsymbol{x}_J$

  end

end

return $\boldsymbol{z}$





Matrix-free Assembly Algorithm (Poisson equation)

Input: $\boldsymbol{x}$, eval(i, j, k)
$\boldsymbol{z} \leftarrow \mathbf{0}$ // Resulting vector to return
for Element K in Mesh **do**
  localDOFs $\leftarrow$ getLocalDOFsForElement(K)
  globalDOFs $\leftarrow$ getGlobalDOFsForElement(K)
  // 1. Compute the Element matrix $\mathbf{A}_K$
  for $i \in$ localDOFs **do**
    for $j \in$ localDOFs **do**
      $$(\mathbf{A}_K)_{i,j} \approx \sum_{k}^{N_{\mathrm{Int}}} \hat{\omega}_k \underbrace{(\mathbf{D\Phi}_K(\hat{\boldsymbol{q}}_k))^{-\top} \nabla \hat{b}^j(\hat{\boldsymbol{q}}_k) \cdot (\mathbf{D\Phi}(\hat{\boldsymbol{q}}_k))^{-\top} \nabla \hat{b}^i(\hat{\boldsymbol{q}}_k) \, |\det \mathbf{D\Phi}_K(\hat{\boldsymbol{q}}_k)|}_{=:\mathtt{eval}(i,j,k)}$$
    end
  end
  // 2. Compute $\boldsymbol{z} = \mathbf{A}\boldsymbol{x}$ contribution with $\mathbf{A}_K$
  for $i \in$ localDOFs **do**
    $I =$ globalDOFs$[i]$
    for $j \in$ localDOFs **do**
      $J =$ globalDOFs$[j]$
      $\boldsymbol{z}_I \leftarrow \boldsymbol{z}_I + (\mathbf{A}_K)_{i,j} \cdot \boldsymbol{x}_J$
    end
  end
end
return $\boldsymbol{z}$

PAUL SCHERRER INSTITUT PSI

Software Architecture

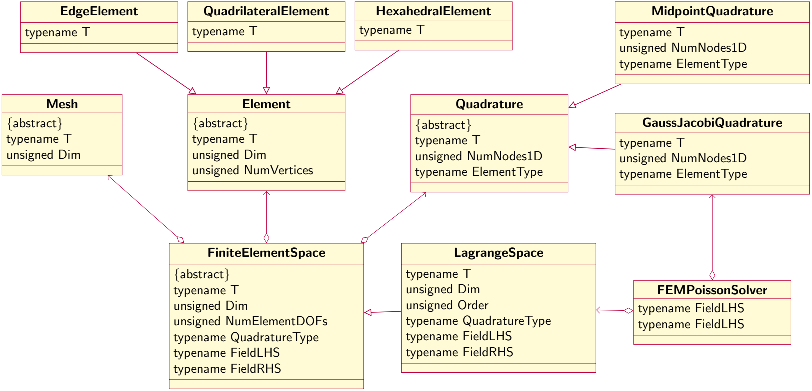


Figure: Software architecture of the FEM framework, showing the classes with their template arguments.

Outline

Sinusoidal Problem

Problem:

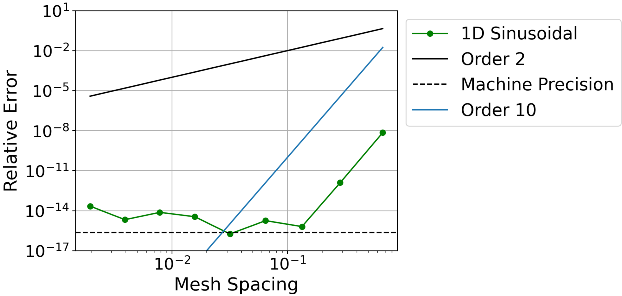
$$\begin{aligned} -\Delta u &= \pi^2 \sin(\pi x), \quad && x \in [-1, 1], \\ u(x) &= 0, && x \in \{-1, 1\}. \end{aligned} \tag{6}$$

Exact solution:

$$u(x) = \sin(\pi x). \tag{7}$$

Mesh spacing: $h = \frac{2}{n-1}$.

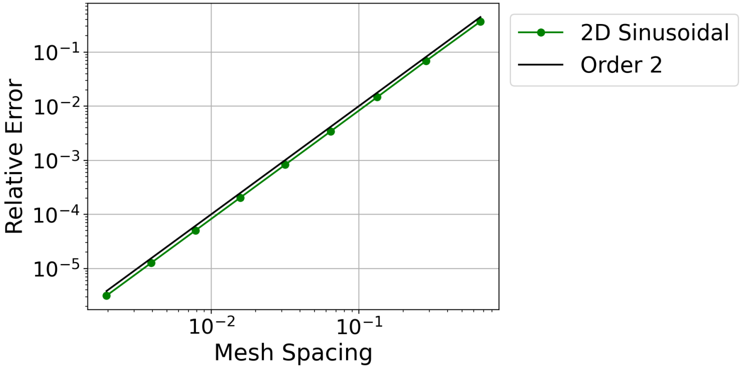
Results



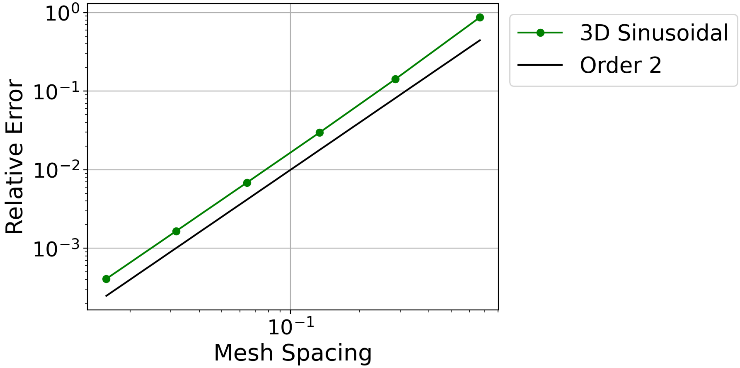
Expected convergence is order 2.

The trapezoidal quadrature rule converges rapidly when applied to analytic functions on periodic intervals. [3]

Results



Results



10^0
10^{-1}
10^{-2}
10^{-3}
Relative Error
10^{-1}
Mesh Spacing
3D Sinusoidal
Order 2

Conclusion

Goal: Implement the building blocks for FEM with the possibility for p-refinement.

Conclusion

Goal: Implement the building blocks for FEM with the possibility for p-refinement.

Achieved:

Conclusion

Goal: Implement the building blocks for FEM with the possibility for p-refinement.

Achieved:

- Software Architecture Design

Conclusion

Goal: Implement the building blocks for FEM with the possibility for p-refinement.

Achieved:

- Software Architecture Design
- Building blocks implemented

PAUL SCHERRER INSTITUT

Conclusion

Goal: Implement the building blocks for FEM with the possibility for p-refinement.

Achieved:

- Software Architecture Design
- Building blocks implemented
    - Mesh-Element-Mapping

Conclusion

Goal: Implement the building blocks for FEM with the possibility for p-refinement.

Achieved:

- Software Architecture Design
- Building blocks implemented
  - Mesh-Element-Mapping
  - Element classes with dimension-independent affine transformations

Conclusion

Goal: Implement the building blocks for FEM with the possibility for p-refinement.

Achieved:

- Software Architecture Design
- Building blocks implemented
  - Mesh-Element-Mapping
  - Element classes with dimension-independent affine transformations
  - 1st-order Lagrangian finite elements (dimension independent, extensible to higher-order)

Conclusion

Goal: Implement the building blocks for FEM with the possibility for p-refinement.

Achieved:

- Software Architecture Design
- Building blocks implemented
  - Mesh-Element-Mapping
  - Element classes with dimension-independent affine transformations
  - 1st-order Lagrangian finite elements (dimension independent, extensible to higher-order)
  - Quadrature base class and tensor-product quadrature

Conclusion

Goal: Implement the building blocks for FEM with the possibility for p-refinement.

Achieved:

- Software Architecture Design
- Building blocks implemented
  - Mesh-Element-Mapping
  - Element classes with dimension-independent affine transformations
  - 1st-order Lagrangian finite elements (dimension independent, extensible to higher-order)
  - Quadrature base class and tensor-product quadrature
  - Gauss-Jacobi Quadrature with an iterative algorithm to compute the roots of the Jacobi-Polynomial

Conclusion

Goal: Implement the building blocks for FEM with the possibility for p-refinement.

Achieved:

- Software Architecture Design
- Building blocks implemented
  - Mesh-Element-Mapping
  - Element classes with dimension-independent affine transformations
  - 1st-order Lagrangian finite elements (dimension independent, extensible to higher-order)
  - Quadrature base class and tensor-product quadrature
  - Gauss-Jacobi Quadrature with an iterative algorithm to compute the roots of the Jacobi-Polynomial
  - Matrix-free Assembly Algorithm, with local evaluations, interfacing with the CG algorithm

PAUL SCHERRER INSTITUT

Conclusion

Goal: Implement the building blocks for FEM with the possibility for p-refinement.

Achieved:

- Software Architecture Design
- Building blocks implemented
  - Mesh-Element-Mapping
  - Element classes with dimension-independent affine transformations
  - 1st-order Lagrangian finite elements (dimension independent, extensible to higher-order)
  - Quadrature base class and tensor-product quadrature
  - Gauss-Jacobi Quadrature with an iterative algorithm to compute the roots of the Jacobi-Polynomial
  - Matrix-free Assembly Algorithm, with local evaluations, interfacing with the CG algorithm
- Proof of concept solver implemented (`FEMPoissonSolver`)

Conclusion

Goal: Implement the building blocks for FEM with the possibility for p-refinement.

Achieved:

- Software Architecture Design
- Building blocks implemented
  - Mesh-Element-Mapping
  - Element classes with dimension-independent affine transformations
  - 1st-order Lagrangian finite elements (dimension independent, extensible to higher-order)
  - Quadrature base class and tensor-product quadrature
  - Gauss-Jacobi Quadrature with an iterative algorithm to compute the roots of the Jacobi-Polynomial
  - Matrix-free Assembly Algorithm, with local evaluations, interfacing with the CG algorithm
- Proof of concept solver implemented (`FEMPoissonSolver`)

$\Rightarrow$ A working beginning of a FEM framework in IPPL

Future Work

- Higher-order Lagrangian finite elements (p-refinement)

Future Work

- Higher-order Lagrangian finite elements (p-refinement)
- Adding support for more boundary conditions
  - non-homogeneous Dirichlet boundary conditions
  - (periodic boundary conditions)

Future Work

- Higher-order Lagrangian finite elements (p-refinement)
- Adding support for more boundary conditions
  - non-homogeneous Dirichlet boundary conditions
  - (periodic boundary conditions)
- Parallelization, GPU support, scaling studies

Future Work

- Higher-order Lagrangian finite elements (p-refinement)
- Adding support for more boundary conditions
  - non-homogeneous Dirichlet boundary conditions
  - (periodic boundary conditions)
- Parallelization, GPU support, scaling studies
- Adding more elements and finite element spaces
  - Nédélec
  - Raviart-Thomas

Bibliography

[1] K. Ljungkvist, "Matrix-Free Finite-Element Computations On Graphics Processors with Adaptively Refined Unstructured Meshes," in 25th High Performance Computing Symposium (HPC 2017). Virginia Beach, VA, USA: Society for Modeling and Simulation International (SCS), 2017. [Online]. Available: http://dl.acm.org/citation.cfm?id=3108097

[2] R. R. Settgast, Y. Dudouit, N. Castelletto, W. R. Tobin, B. C. Corbett, and S. Klevtsov, "Performant low-order matrix-free finite element kernels on GPU architectures," Aug. 2023, arXiv:2308.09839 [cs, math]. [Online]. Available: http://arxiv.org/abs/2308.09839

[3] L. N. Trefethen and J. A. C. Weideman, "The Exponentially Convergent Trapezoidal Rule," SIAM Review, vol. 56, no. 3, pp. 385–458, Jan. 2014. [Online]. Available: http://epubs.siam.org/doi/10.1137/130932132