Rodin: Lightweight and modern C++17 shape, density and topology optimization framework

Carlos BRITO PACHECO[1]

[1] carlos.brito-pacheco@etu.univ-grenoble-alpes.fr

October 24, 2022

LABORATOIRE JEAN KUNTZMANN
MATHÉMATIQUES APPLIQUÉES - INFORMATIQUE

Université Grenoble Alpes

Table of Contents

1. Introduction
   - Weak formulation specification
   - Integration with MMG
   - Integration with ISCD tools
2. Geometrical shape optimization
   - Mathematical framework
   - Applied example
   - Implementation
   - Results
   - Conclusion
3. Appendix

1 Introduction
- Weak formulation specification
- Integration with MMG
- Integration with ISCD tools

2 Geometrical shape optimization
- Mathematical framework
- Applied example
- Implementation
- Results
- Conclusion

3 Appendix

What is a shape optimization problem? *A shape optimization problem is an optimization problem where the optimization variable is the shape of the structure itself.*[1]

General references. (Not a comprehensive list.) Allaire et al. (2021); Allaire and Schoenauer (2007); Allaire et al. (2002); Delfour and Zolésio (2011); Henrot and Pierre (2018); Bendsoe and Sigmund (2003)

Selected sample applications. Dapogny et al. (2017, 2020); Amstutz et al. (2022); Feppon et al. (2020)

What is Rodin?

- It is a numerical C++17 library which facilitates the implementation of shape and topology optimization algorithms.
- It is a wrapper around various numerical libraries and utilities under one single API.
- Wrapped libraries include:
  - MFEM - `https://mfem.org/`
  - MMG[2] - `https://www.mmgtools.org/`
  - ISCD Tools - `https://iscd.sorbonne-universite.fr/`
- **Above all, it is a tool for the rapid prototyping of shape optimization algorithms.**

Where can I find Rodin?

- `https://cbritopacheco.github.io/rodin/`
- `https://github.com/cbritopacheco/rodin/`

[1] Allaire and Schoenauer (2007)

[2] Dapogny et al. (2014)

What is a shape optimization problem? *A shape optimization problem is an optimization problem where the optimization variable is the shape of the structure itself.*[1]

General references. (Not a comprehensive list.) Allaire et al. (2021); Allaire and Schoenauer (2007); Allaire et al. (2002); Delfour and Zolésio (2011); Henrot and Pierre (2018); Bendsoe and Sigmund (2003)

Selected sample applications. Dapogny et al. (2017, 2020); Amstutz et al. (2022); Feppon et al. (2020)

What is Rodin?

- It is a numerical C++17 library which facilitates the implementation of shape and topology optimization algorithms.
- It is a wrapper around various numerical libraries and utilities under one single API.
- Wrapped libraries include:
  - MFEM - `https://mfem.org/`
  - MMG[2] - `https://www.mmgtools.org/`
  - ISCD Tools - `https://iscd.sorbonne-universite.fr/`
- **Above all, it is a tool for the rapid prototyping of shape optimization algorithms.**

Where can I find Rodin?

- `https://cbritopacheco.github.io/rodin/`
- `https://github.com/cbritopacheco/rodin/`

[1] Allaire and Schoenauer (2007)

[2] Dapogny et al. (2016)

What is a shape optimization problem? *A shape optimization problem is an optimization problem where the optimization variable is the shape of the structure itself.*[1]

General references. (Not a comprehensive list.) Allaire et al. (2021); Allaire and Schoenauer (2007); Allaire et al. (2002); Delfour and Zolésio (2011); Henrot and Pierre (2018); Bendsoe and Sigmund (2003)

Selected sample applications. Dapogny et al. (2017, 2020); Amstutz et al. (2022); Feppon et al. (2020)

What is Rodin?

- It is a numerical C++17 library which facilitates the implementation of shape and topology optimization algorithms.
- It is a wrapper around various numerical libraries and utilities under one single API.
- Wrapped libraries include:
  - MFEM - https://mfem.org/
  - MMG[2] - https://www.mmgtools.org/
  - ISCD Tools - https://iscd.sorbonne-universite.fr/
- **Above all, it is a tool for the rapid prototyping of shape optimization algorithms.**

Where can I find Rodin?

- https://cbritopacheco.github.io/rodin/
- https://github.com/cbritopacheco/rodin/

[1] Allaire and Schoenauer (2007)

[2] Dapogny et al. (2016)

Introduction

What is a shape optimization problem? *A shape optimization problem is an optimization problem where the optimization variable is the shape of the structure itself.*[1]

General references. (Not a comprehensive list.) Allaire et al. (2021); Allaire and Schoenauer (2007); Allaire et al. (2002); Delfour and Zolésio (2011); Henrot and Pierre (2018); Bendsoe and Sigmund (2003)

Selected sample applications. Dapogny et al. (2017, 2020); Amstutz et al. (2022); Feppon et al. (2020)

What is Rodin?

- It is a numerical C++17 library which facilitates the implementation of shape and topology optimization algorithms.
- It is a wrapper around various numerical libraries and utilities under one single API.
- Wrapped libraries include:
  - MFEM - `https://mfem.org/`
  - MMG[2] - `https://www.mmgtools.org/`
  - ISCD Tools - `https://iscd.sorbonne-universite.fr/`
- **Above all, it is a tool for the rapid prototyping of shape optimization algorithms.**

Where can I find Rodin?

- `https://cbritopacheco.github.io/rodin/`
- `https://github.com/cbritopacheco/rodin/`

---

[1] Allaire and Schoenauer (2007)

[2] Dapogny et al. (2014)

Introduction

What is a shape optimization problem? *A shape optimization problem is an optimization problem where the optimization variable is the shape of the structure itself.*[1]

General references. (Not a comprehensive list.) Allaire et al. (2021); Allaire and Schoenauer (2007); Allaire et al. (2002); Delfour and Zolésio (2011); Henrot and Pierre (2018); Bendsoe and Sigmund (2003)

Selected sample applications. Dapogny et al. (2017, 2020); Amstutz et al. (2022); Feppon et al. (2020)

What is Rodin?

- It is a numerical C++17 library which facilitates the implementation of shape and topology optimization algorithms.
- It is a wrapper around various numerical libraries and utilities under one single API.
- Wrapped libraries include:
  - MFEM - `https://mfem.org/`
  - MMG[2] - `https://www.mmgtools.org/`
  - ISCD Tools - `https://iscd.sorbonne-universite.fr/`
- **Above all, it is a tool for the rapid prototyping of shape optimization algorithms.**

Where can I find Rodin?

- `https://cbritopacheco.github.io/rodin/`
- `https://github.com/cbritopacheco/rodin/`

[1] Allaire and Schoenauer (2007)

[2] Dapogny et al. (2014)

What is a shape optimization problem? *A shape optimization problem is an optimization problem where the optimization variable is the shape of the structure itself.*[1]

General references. (Not a comprehensive list.) Allaire et al. (2021); Allaire and Schoenauer (2007); Allaire et al. (2002); Delfour and Zolésio (2011); Henrot and Pierre (2018); Bendsoe and Sigmund (2003)

Selected sample applications. Dapogny et al. (2017, 2020); Amstutz et al. (2022); Feppon et al. (2020)

What is Rodin?

- It is a numerical C++17 library which facilitates the implementation of shape and topology optimization algorithms.
- It is a wrapper around various numerical libraries and utilities under one single API.
- Wrapped libraries include:
  - MFEM - `https://mfem.org/`
  - MMG[2] - `https://www.mmgtools.org/`
  - ISCD Tools - `https://iscd.sorbonne-universite.fr/`
- **Above all, it is a tool for the rapid prototyping of shape optimization algorithms.**

Where can I find Rodin?

- `https://cbritopacheco.github.io/rodin/`
- `https://github.com/cbritopacheco/rodin/`

[1] Allaire and Schoenauer (2007)

[2] Dapogny et al. (2014)

What is a shape optimization problem? *A shape optimization problem is an optimization problem where the optimization variable is the shape of the structure itself.*[1]

General references. (Not a comprehensive list.) Allaire et al. (2021); Allaire and Schoenauer (2007); Allaire et al. (2002); Delfour and Zolésio (2011); Henrot and Pierre (2018); Bendsoe and Sigmund (2003)

Selected sample applications. Dapogny et al. (2017, 2020); Amstutz et al. (2022); Feppon et al. (2020)

What is Rodin?

- It is a numerical C++17 library which facilitates the implementation of shape and topology optimization algorithms.
- It is a wrapper around various numerical libraries and utilities under one single API.
- Wrapped libraries include:
  - MFEM - `https://mfem.org/`
  - MMG[2] - `https://www.mmgtools.org/`
  - ISCD Tools - `https://iscd.sorbonne-universite.fr/`
- **Above all, it is a tool for the rapid prototyping of shape optimization algorithms.**

Where can I find Rodin?

- `https://cbritopacheco.github.io/rodin/`
- `https://github.com/cbritopacheco/rodin/`

[1] Allaire and Schoenauer (2007)

[2] Dapogny et al. (2014)

What is a shape optimization problem? *A shape optimization problem is an optimization problem where the optimization variable is the shape of the structure itself.*[1]

General references. (Not a comprehensive list.) Allaire et al. (2021); Allaire and Schoenauer (2007); Allaire et al. (2002); Delfour and Zolésio (2011); Henrot and Pierre (2018); Bendsoe and Sigmund (2003)

Selected sample applications. Dapogny et al. (2017, 2020); Amstutz et al. (2022); Feppon et al. (2020)

What is Rodin?

- It is a numerical C++17 library which facilitates the implementation of shape and topology optimization algorithms.
- It is a wrapper around various numerical libraries and utilities under one single API.
- Wrapped libraries include:
  - MFEM - `https://mfem.org/`
  - MMG[2] - `https://www.mmgtools.org/`
  - ISCD Tools - `https://iscd.sorbonne-universite.fr/`
- **Above all, it is a tool for the rapid prototyping of shape optimization algorithms.**

Where can I find Rodin?

- `https://cbritopacheco.github.io/rodin/`
- `https://github.com/cbritopacheco/rodin/`

[1] Allaire and Schoenauer (2007)
[2] Dapogny et al. (2014)

Introduction

What is a shape optimization problem? *A shape optimization problem is an optimization problem where the optimization variable is the shape of the structure itself.*[1]

General references. (Not a comprehensive list.) Allaire et al. (2021); Allaire and Schoenauer (2007); Allaire et al. (2002); Delfour and Zolésio (2011); Henrot and Pierre (2018); Bendsoe and Sigmund (2003)

Selected sample applications. Dapogny et al. (2017, 2020); Amstutz et al. (2022); Feppon et al. (2020)

What is Rodin?

- It is a numerical C++17 library which facilitates the implementation of shape and topology optimization algorithms.
- It is a wrapper around various numerical libraries and utilities under one single API.
- Wrapped libraries include:
  - MFEM - `https://mfem.org/`
  - MMG[2] - `https://www.mmgtools.org/`
  - ISCD Tools - `https://iscd.sorbonne-universite.fr/`
- **Above all, it is a tool for the rapid prototyping of shape optimization algorithms.**

Where can I find Rodin?

- `https://cbritopacheco.github.io/rodin/`
- `https://github.com/cbritopacheco/rodin/`

[1] Allaire and Schoenauer (2007)

[2] Dapogny et al. (2014)

What is a shape optimization problem? *A shape optimization problem is an optimization problem where the optimization variable is the shape of the structure itself.*[1]

General references. (Not a comprehensive list.) Allaire et al. (2021); Allaire and Schoenauer (2007); Allaire et al. (2002); Delfour and Zolésio (2011); Henrot and Pierre (2018); Bendsoe and Sigmund (2003)

Selected sample applications. Dapogny et al. (2017, 2020); Amstutz et al. (2022); Feppon et al. (2020)

What is Rodin?

- It is a numerical C++17 library which facilitates the implementation of shape and topology optimization algorithms.
- It is a wrapper around various numerical libraries and utilities under one single API.
- Wrapped libraries include:
  - MFEM - `https://mfem.org/`
  - MMG[2] - `https://www.mmgtools.org/`
  - ISCD Tools - `https://iscd.sorbonne-universite.fr/`
- **Above all, it is a tool for the rapid prototyping of shape optimization algorithms.**

Where can I find Rodin?

- `https://cbritopacheco.github.io/rodin/`
- `https://github.com/cbritopacheco/rodin/`

[1] Allaire and Schoenauer (2007)

[2] Dapogny et al. (2014)

Introduction

What is a shape optimization problem? *A shape optimization problem is an optimization problem where the optimization variable is the shape of the structure itself.*[1]

General references. (Not a comprehensive list.) Allaire et al. (2021); Allaire and Schoenauer (2007); Allaire et al. (2002); Delfour and Zolésio (2011); Henrot and Pierre (2018); Bendsoe and Sigmund (2003)

Selected sample applications. Dapogny et al. (2017, 2020); Amstutz et al. (2022); Feppon et al. (2020)

What is Rodin?

- It is a numerical C++17 library which facilitates the implementation of shape and topology optimization algorithms.
- It is a wrapper around various numerical libraries and utilities under one single API.
- Wrapped libraries include:
  - MFEM - `https://mfem.org/`
  - MMG[2] - `https://www.mmgtools.org/`
  - ISCD Tools - `https://iscd.sorbonne-universite.fr/`
- **Above all, it is a tool for the rapid prototyping of shape optimization algorithms.**

Where can I find Rodin?

- `https://cbritopacheco.github.io/rodin/`
- `https://github.com/cbritopacheco/rodin/`

[1] Allaire and Schoenauer (2007)

[2] Dapogny et al. (2014)

What is a shape optimization problem? *A shape optimization problem is an optimization problem where the optimization variable is the shape of the structure itself.*[1]

General references. (Not a comprehensive list.) Allaire et al. (2021); Allaire and Schoenauer (2007); Allaire et al. (2002); Delfour and Zolésio (2011); Henrot and Pierre (2018); Bendsoe and Sigmund (2003)

Selected sample applications. Dapogny et al. (2017, 2020); Amstutz et al. (2022); Feppon et al. (2020)

What is Rodin?

- It is a numerical C++17 library which facilitates the implementation of shape and topology optimization algorithms.
- It is a wrapper around various numerical libraries and utilities under one single API.
- Wrapped libraries include:
  - MFEM - `https://mfem.org/`
  - MMG[2] - `https://www.mmgtools.org/`
  - ISCD Tools - `https://iscd.sorbonne-universite.fr/`
- **Above all, it is a tool for the rapid prototyping of shape optimization algorithms.**

Where can I find Rodin?

- `https://cbritopacheco.github.io/rodin/`
- `https://github.com/cbritopacheco/rodin/`

[1] Allaire and Schoenauer (2007)

[2] Dapogny et al. (2014)

What is a shape optimization problem? *A shape optimization problem is an optimization problem where the optimization variable is the shape of the structure itself.*[1]

General references. (Not a comprehensive list.) Allaire et al. (2021); Allaire and Schoenauer (2007); Allaire et al. (2002); Delfour and Zolésio (2011); Henrot and Pierre (2018); Bendsoe and Sigmund (2003)

Selected sample applications. Dapogny et al. (2017, 2020); Amstutz et al. (2022); Feppon et al. (2020)

What is Rodin?

- It is a numerical C++17 library which facilitates the implementation of shape and topology optimization algorithms.
- It is a wrapper around various numerical libraries and utilities under one single API.
- Wrapped libraries include:
  - MFEM - `https://mfem.org/`
  - MMG[2] - `https://www.mmgtools.org/`
  - ISCD Tools - `https://iscd.sorbonne-universite.fr/`
- **Above all, it is a tool for the rapid prototyping of shape optimization algorithms.**

Where can I find Rodin?

- `https://cbritopacheco.github.io/rodin/`
- `https://github.com/cbritopacheco/rodin/`

[1] Allaire and Schoenauer (2007)
[2] Dapogny et al. (2014)

What does Rodin do?

- Allows for easy specification and resolution of variational problems.
- Provides uniform class interfaces and transparent conversions between different types of meshes and grid functions.
- Enforces semantic rules related to weak formulations (e.g. differentiability).
- Adds functionalities of its own on top of wrapped classes.

How does Rodin do it?

- **MFEM** is used for solving and assembling the linear systems related to the weak formulations.
- **MMG** is a meshing and remeshing tool with a suite of features to operate on meshes (most notably improving quality of meshes).
- **ISCD** is a collection of tools which include **advection** of a level set function, **distancing** of a domain, etc.

What does Rodin do?

- Allows for easy specification and resolution of variational problems.
- Provides uniform class interfaces and transparent conversions between different types of meshes and grid functions.
- Enforces semantic rules related to weak formulations (e.g. differentiability).
- Adds functionalities of its own on top of wrapped classes.

How does Rodin do it?

- **MFEM** is used for solving and assembling the linear systems related to the weak formulations.
- **MMG** is a meshing and remeshing tool with a suite of features to operate on meshes (most notably improving quality of meshes).
- **ISCD** is a collection of tools which include **advection** of a level set function, **distancing** of a domain, etc.

What does Rodin do?

- Allows for easy specification and resolution of variational problems.
- Provides uniform class interfaces and transparent conversions between different types of meshes and grid functions.
- Enforces semantic rules related to weak formulations (e.g. differentiability).
- Adds functionalities of its own on top of wrapped classes.

How does Rodin do it?

- **MFEM** is used for solving and assembling the linear systems related to the weak formulations.
- **MMG** is a meshing and remeshing tool with a suite of features to operate on meshes (most notably improving quality of meshes).
- **ISCD** is a collection of tools which include **advection** of a level set function, **distancing** of a domain, etc.

What does Rodin do?

- Allows for easy specification and resolution of variational problems.
- Provides uniform class interfaces and transparent conversions between different types of meshes and grid functions.
- Enforces semantic rules related to weak formulations (e.g. differentiability).
- Adds functionalities of its own on top of wrapped classes.

How does Rodin do it?

- MFEM is used for solving and assembling the linear systems related to the weak formulations.
- MMG is a meshing and remeshing tool with a suite of features to operate on meshes (most notably improving quality of meshes).
- ISCD is a collection of tools which include **advection** of a level set function, **distancing** of a domain, etc.

What does Rodin do?

- Allows for easy specification and resolution of variational problems.
- Provides uniform class interfaces and transparent conversions between different types of meshes and grid functions.
- Enforces semantic rules related to weak formulations (e.g. differentiability).
- Adds functionalities of its own on top of wrapped classes.

How does Rodin do it?

- **MFEM** is used for solving and assembling the linear systems related to the weak formulations.
- **MMG** is a meshing and remeshing tool with a suite of features to operate on meshes (most notably improving quality of meshes).
- **ISCD** is a collection of tools which include **advection** of a level set function, **distancing** of a domain, etc.

What does Rodin do?

- Allows for easy specification and resolution of variational problems.
- Provides uniform class interfaces and transparent conversions between different types of meshes and grid functions.
- Enforces semantic rules related to weak formulations (e.g. differentiability).
- Adds functionalities of its own on top of wrapped classes.

How does Rodin do it?

- **MFEM** is used for solving and assembling the linear systems related to the weak formulations.
- **MMG** is a meshing and remeshing tool with a suite of features to operate on meshes (most notably improving quality of meshes).
- **ISCD** is a collection of tools which include **advection** of a level set function, **distancing** of a domain, etc.

What does Rodin do?

- Allows for easy specification and resolution of variational problems.
- Provides uniform class interfaces and transparent conversions between different types of meshes and grid functions.
- Enforces semantic rules related to weak formulations (e.g. differentiability).
- Adds functionalities of its own on top of wrapped classes.

How does Rodin do it?

- **MFEM** is used for solving and assembling the linear systems related to the weak formulations.
- **MMG** is a meshing and remeshing tool with a suite of features to operate on meshes (most notably improving quality of meshes).
- **ISCD** is a collection of tools which include **advection** of a level set function, **distancing** of a domain, etc.

What does Rodin do?

- Allows for easy specification and resolution of variational problems.
- Provides uniform class interfaces and transparent conversions between different types of meshes and grid functions.
- Enforces semantic rules related to weak formulations (e.g. differentiability).
- Adds functionalities of its own on top of wrapped classes.

How does Rodin do it?

- **MFEM** is used for solving and assembling the linear systems related to the weak formulations.
- **MMG** is a meshing and remeshing tool with a suite of features to operate on meshes (most notably improving quality of meshes).
- **ISCD** is a collection of tools which include **advection** of a level set function, **distancing** of a domain, etc.

1 Introduction
- Weak formulation specification
- Integration with MMG
- Integration with ISCD tools

2 Geometrical shape optimization
- Mathematical framework
- Applied example
- Implementation
- Results
- Conclusion

3 Appendix

Weak formulation specification

Poisson equation

Let $\Omega \subset \mathbb{R}^n$ be a bounded Lipschitz domain with boundary $\Gamma := \partial\Omega$.

$$\begin{cases} -\nabla \cdot (\lambda \nabla u) = f & \text{in } \Omega \\ u = 0 & \text{on } \Gamma \end{cases} .$$

The associated weak formulation reads:

Find $u \in H^1_\Gamma(\Omega)$ such that

$$\forall v \in H^1_\Gamma(\Omega), \quad \int_\Omega \lambda \nabla u \cdot \nabla v \ dx - \int_\Omega f v \ dx = 0 ,$$

where

$$H^1_\Gamma(\Omega) := \{ u \in H^1(\Omega) : u = 0 \text{ on } \Gamma \} .$$

In Rodin, the problem specification is written as follows:

```
Problem poisson(u, v);
poisson = Integral(lambda * Grad(u), Grad(v))
        - Integral(f, v)
        + DirichletBC(f, ScalarFunction(0.0)).on(Gamma);
```

Weak formulation specification

Poisson equation

Let $\Omega \subset \mathbb{R}^n$ be a bounded Lipschitz domain with boundary $\Gamma := \partial\Omega$.

$$\begin{cases} -\nabla \cdot (\lambda \nabla u) = f & \text{in } \Omega \\ u = 0 & \text{on } \Gamma \end{cases} .$$

The associated weak formulation reads:

Find $u \in H^1_\Gamma(\Omega)$ such that

$$\forall v \in H^1_\Gamma(\Omega), \quad \int_\Omega \lambda \nabla u \cdot \nabla v \; dx - \int_\Omega f v \; dx = 0 \; ,$$

where

$$H^1_\Gamma(\Omega) := \{ u \in H^1(\Omega) : u = 0 \text{ on } \Gamma \} \; .$$

In Rodin, the problem specification is written as follows:

```
Problem poisson(u, v);
poisson = Integral(lambda * Grad(u), Grad(v))
        - Integral(f, v)
        + DirichletBC(f, ScalarFunction(0.0)).on(Gamma);
```

Poisson equation

Let $\Omega \subset \mathbb{R}^n$ be a bounded Lipschitz domain with boundary $\Gamma := \partial\Omega$.

$$\begin{cases} -\nabla \cdot (\lambda \nabla u) = f & \text{in } \Omega \\ u = 0 & \text{on } \Gamma \end{cases} .$$

The associated weak formulation reads:

Find $u \in H^1_\Gamma(\Omega)$ such that

$$\forall v \in H^1_\Gamma(\Omega), \quad \int_\Omega \lambda \nabla u \cdot \nabla v \, dx - \int_\Omega fv \, dx = 0 \ ,$$

where

$$H^1_\Gamma(\Omega) := \{u \in H^1(\Omega) : u = 0 \text{ on } \Gamma\} \ .$$

In Rodin, the problem specification is written as follows:

```
Problem poisson(u, v);
poisson = Integral(lambda * Grad(u), Grad(v))
        - Integral(f, v)
        + DirichletBC(f, ScalarFunction(0.0)).on(Gamma);
```

Full code

```
#include <Rodin/Mesh.h>
#include <Rodin/Solver.h>
#include <Rodin/Variational.h>
using namespace Rodin;
using namespace Rodin::Variational;
int main(int, char**)
{
  int Gamma = 1;
  Mesh Omega;
  Omega.load("poisson-example.mesh");
  auto lambda = ScalarFunction(1.0);
  auto f = ScalarFunction(1.0);
  H1 Vh(Omega);
  TrialFunction u(Vh);
  TestFunction  v(Vh);
  Problem poisson(u, v);
  poisson = Integral(lambda * Grad(u), Grad(v))
          - Integral(f, v)
          + DirichletBC(u, ScalarFunction(0.0)).on(Gamma);
  Solver::CG().solve(poisson);
  u.getGridFunction().save("u.gf");
  Omega.save("Omega.mesh");
  return 0;
}
```

Full code

```cpp
#include <Rodin/Mesh.h>
#include <Rodin/Solver.h>
#include <Rodin/Variational.h>
using namespace Rodin;
using namespace Rodin::Variational;
int main(int, char**)
{
  int Gamma = 1;
  Mesh Omega;
  Omega.load("poisson-example.mesh");
  auto lambda = ScalarFunction(1.0);
  auto f = ScalarFunction(1.0);
  H1 Vh(Omega);
  TrialFunction u(Vh);
  TestFunction  v(Vh);
  Problem poisson(u, v);
  poisson = Integral(lambda * Grad(u), Grad(v))
          - Integral(f, v)
          + DirichletBC(u, ScalarFunction(0.0)).on(Gamma);
  Solver::CG().solve(poisson);
  u.getGridFunction().save("u.gf");
  Omega.save("Omega.mesh");
  return 0;
}
```

Full code

```cpp
#include <Rodin/Mesh.h>
#include <Rodin/Solver.h>
#include <Rodin/Variational.h>

using namespace Rodin;
using namespace Rodin::Variational;

int main(int, char**)
{
  int Gamma = 1;
  Mesh Omega;
  Omega.load("poisson-example.mesh");

  auto lambda = ScalarFunction(1.0);
  auto f = ScalarFunction(1.0);

  H1 Vh(Omega);
  TrialFunction u(Vh);
  TestFunction  v(Vh);

  Problem poisson(u, v);
  poisson = Integral(lambda * Grad(u), Grad(v))
          - Integral(f, v)
          + DirichletBC(u, ScalarFunction(0.0)).on(Gamma);

  Solver::CG().solve(poisson);

  u.getGridFunction().save("u.gf");
  Omega.save("Omega.mesh");
  return 0;
}
```

Full code

```
#include <Rodin/Mesh.h>
#include <Rodin/Solver.h>
#include <Rodin/Variational.h>
using namespace Rodin;
using namespace Rodin::Variational;
int main(int, char**)
{
  int Gamma = 1;
  Mesh Omega;
  Omega.load("poisson-example.mesh");
  auto lambda = ScalarFunction(1.0);
  auto f = ScalarFunction(1.0);
  H1 Vh(Omega);
  TrialFunction u(Vh);
  TestFunction  v(Vh);
  Problem poisson(u, v);
  poisson = Integral(lambda * Grad(u), Grad(v))
          - Integral(f, v)
          + DirichletBC(u, ScalarFunction(0.0)).on(Gamma);
  Solver::CG().solve(poisson);
  u.getGridFunction().save("u.gf");
  Omega.save("Omega.mesh");
  return 0;
}
```

```cpp
#include <Rodin/Mesh.h>
#include <Rodin/Solver.h>
#include <Rodin/Variational.h>
using namespace Rodin;
using namespace Rodin::Variational;
int main(int, char**)
{
  int Gamma = 1;
  Mesh Omega;
  Omega.load("poisson-example.mesh");
  auto lambda = ScalarFunction(1.0);
  auto f = ScalarFunction(1.0);
  H1 Vh(Omega);
  TrialFunction u(Vh);
  TestFunction  v(Vh);
  Problem poisson(u, v);
  poisson = Integral(lambda * Grad(u), Grad(v))
          - Integral(f, v)
          + DirichletBC(u, ScalarFunction(0.0)).on(Gamma);
  Solver::CG().solve(poisson);
  u.getGridFunction().save("u.gf");
  Omega.save("Omega.mesh");
  return 0;
}
```

```cpp
#include <Rodin/Mesh.h>
#include <Rodin/Solver.h>
#include <Rodin/Variational.h>

using namespace Rodin;
using namespace Rodin::Variational;

int main(int, char**)
{
  int Gamma = 1;
  Mesh Omega;
  Omega.load("poisson-example.mesh");

  auto lambda = ScalarFunction(1.0);
  auto f = ScalarFunction(1.0);

  H1 Vh(Omega);
  TrialFunction u(Vh);
  TestFunction  v(Vh);

  Problem poisson(u, v);
  poisson = Integral(lambda * Grad(u), Grad(v))
          - Integral(f, v)
          + DirichletBC(u, ScalarFunction(0.0)).on(Gamma);

  Solver::CG().solve(poisson);

  u.getGridFunction().save("u.gf");
  Omega.save("Omega.mesh");
  return 0;
}
```

Full code

```cpp
#include <Rodin/Mesh.h>
#include <Rodin/Solver.h>
#include <Rodin/Variational.h>

using namespace Rodin;
using namespace Rodin::Variational;

int main(int, char**)
{
  int Gamma = 1;
  Mesh Omega;
  Omega.load("poisson-example.mesh");

  auto lambda = ScalarFunction(1.0);
  auto f = ScalarFunction(1.0);

  H1 Vh(Omega);
  TrialFunction u(Vh);
  TestFunction  v(Vh);

  Problem poisson(u, v);
  poisson = Integral(lambda * Grad(u), Grad(v))
          - Integral(f, v)
          + DirichletBC(u, ScalarFunction(0.0)).on(Gamma);

  Solver::CG().solve(poisson);

  u.getGridFunction().save("u.gf");
  Omega.save("Omega.mesh");
  return 0;
}
```

Full code

```cpp
#include <Rodin/Mesh.h>
#include <Rodin/Solver.h>
#include <Rodin/Variational.h>

using namespace Rodin;
using namespace Rodin::Variational;

int main(int, char**)
{
  int Gamma = 1;
  Mesh Omega;
  Omega.load("poisson-example.mesh");

  auto lambda = ScalarFunction(1.0);
  auto f = ScalarFunction(1.0);

  H1 Vh(Omega);
  TrialFunction u(Vh);
  TestFunction  v(Vh);

  Problem poisson(u, v);
  poisson = Integral(lambda * Grad(u), Grad(v))
          - Integral(f, v)
          + DirichletBC(u, ScalarFunction(0.0)).on(Gamma);

  Solver::CG().solve(poisson);

  u.getGridFunction().save("u.gf");
  Omega.save("Omega.mesh");
  return 0;
}
```

Visualize the result using `mfem`'s companion tool `glvis`.

```
glvis -m Omega.mesh -g u.gf
```


Figure: Solution of the Poisson equation on a star-shaped domain.

Visualize the result using `mfem`'s companion tool `glvis`.

```
glvis -m Omega.mesh -g u.gf
```

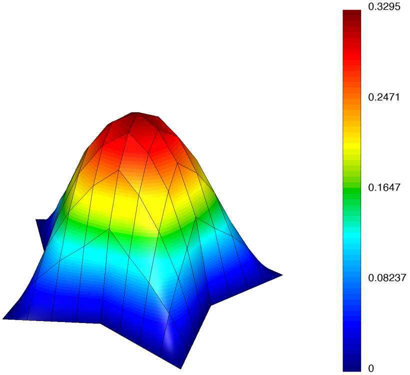


Figure: Solution of the Poisson equation on a star-shaped domain.

1. Introduction
   - Weak formulation specification
   - Integration with MMG
   - Integration with ISCD tools

2. Geometrical shape optimization
   - Mathematical framework
   - Applied example
   - Implementation
   - Results
   - Conclusion

3. Appendix

I/O.

```
Mesh Omega;
Omega.load(meshFile, IO::FileFormat::MEDIT);
```

Mesh optimization.

```
MMG::MeshOptimizer().setHMax(hmax) // maximal edge size
                    .setHMin(hmin) // minimal edge size
                    .setGradation(hgrad) // ratio between two edges
                    .setHausdorff(hausd) // curvature refinement
                    .optimize(Omega);
```

I/O.

```
Mesh Omega;
Omega.load(meshFile, IO::FileFormat::MEDIT);
```

Mesh optimization.

```
MMG::MeshOptimizer().setHMax(hmax) // maximal edge size
                    .setHMin(hmin) // minimal edge size
                    .setGradation(hgrad) // ratio between two edges
                    .setHausdorff(hausd) // curvature refinement
                    .optimize(Omega);
```

1 Introduction
- Weak formulation specification
- Integration with MMG
- Integration with ISCD tools

2 Geometrical shape optimization
- Mathematical framework
- Applied example
- Implementation
- Results
- Conclusion

3 Appendix

Signed distance function computation

Distancing. Implementation of Dapogny and Frey (2012).

```
int interior = 1;
H1 fes(mesh);
auto dist = MMG::Distancer(fes).setInteriorDomain(interior)
                                .distance(Omega);
```

$$\phi(x) = \begin{cases} d(x, \partial\Omega) & \text{if } x \in D \backslash \overline{\Omega} \\ 0 & \text{if } x \in \partial\Omega \\ -d(x, \partial\Omega) & \text{if } x \in \Omega \end{cases}.$$

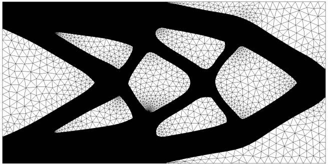
(a) Shape represented by its attributes.

(b) Shape represented by its distance function.

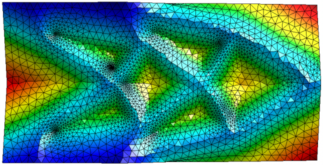
Distancing. Implementation of Dapogny and Frey (2012).

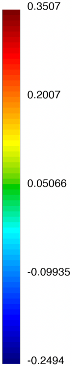
```
int interior = 1;
H1 fes(mesh);
auto dist = MMG::Distancer(fes).setInteriorDomain(interior)
                              .distance(Omega);
```

$$\phi(x) = \begin{cases} d(x, \partial\Omega) & \text{if } x \in D \setminus \overline{\Omega} \\ 0 & \text{if } x \in \partial\Omega \\ -d(x, \partial\Omega) & \text{if } x \in \Omega \end{cases} .$$



(a) Shape represented by its attributes.





(b) Shape represented by its distance function.

Advection.

```
H1 vfes(Omega, Omega.getSpaceDimension());
GridFunction displacement(vfes);
MMG::Advect(dist, displacement).step(dt);
```

Advection-like equation of a level-set function:

$$\frac{\partial \phi}{\partial t} + V(t, x) \cdot \nabla \phi = 0 \quad (t, x) \in [0, T] \times D \ ,$$

where $T > 0$ and $V(t, x)$ is a time-dependent vector field defined on D.

All of the previous tools are commonplace used in shape optimization algorithms.

Advection.

```
H1 vfes(Omega, Omega.getSpaceDimension());
GridFunction displacement(vfes);
MMG::Advect(dist, displacement).step(dt);
```

Advection-like equation of a level-set function:

$$\frac{\partial \phi}{\partial t} + V(t, x) \cdot \nabla \phi = 0 \quad (t, x) \in [0, T] \times D \ ,$$

where $T > 0$ and $V(t, x)$ is a time-dependent vector field defined on D.

All of the previous tools are commonplace used in shape optimization algorithms.

1 Introduction
- Weak formulation specification
- Integration with MMG
- Integration with ISCD tools

2 Geometrical shape optimization
- Mathematical framework
- Applied example
- Implementation
- Results
- Conclusion

3 Appendix

Deformation of a domain Ω. For $\vec{\theta} \in W^{1,\infty}(\mathbb{R}^d, \mathbb{R}^d)$, define:

$$x \mapsto T(x) := (\mathrm{Id} + \vec{\theta})(x)$$
$$\Omega_{\vec{\theta}} := T(\Omega)$$

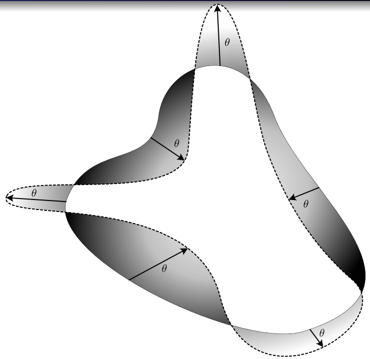


Figure: Depiction of boundary variation by a vector field $\vec{\theta}$.

Definition (Shape differentiability)

The mapping $\Omega \mapsto J(\Omega)$ is said to be **shape differentiable** at Ω if there exists a continuous linear mapping $L : W^{1,\infty}(\mathbb{R}^d; \mathbb{R}^d) \to \mathbb{R}$ such that

$$J(\Omega_{\vec{\theta}}) = J(\Omega) + L(\vec{\theta}) + o(\vec{\theta}), \quad \text{with} \quad \lim_{w \to 0} \frac{|o(\theta)|}{||\vec{\theta}||_{W^{1,\infty}(\mathbb{R}^d; \mathbb{R}^d)}} = 0$$

We call $J'(\Omega) := L$, the **shape derivative** of J at Ω.

Deformation of a domain Ω. For $\vec{\theta} \in W^{1,\infty}(\mathbb{R}^d, \mathbb{R}^d)$, define:

$$x \mapsto T(x) := (\mathrm{Id} + \vec{\theta})(x)$$
$$\Omega_{\vec{\theta}} := T(\Omega)$$

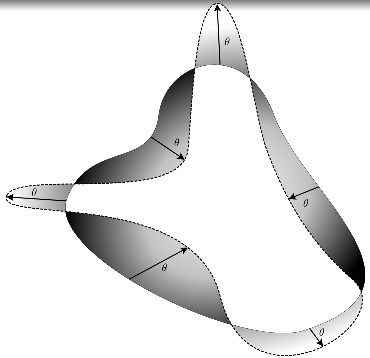


Figure: Depiction of boundary variation by a vector field $\vec{\theta}$.

Definition (Shape differentiability)

The mapping $\Omega \mapsto J(\Omega)$ is said to be **shape differentiable** at Ω if there exists a continuous linear mapping $L : W^{1,\infty}(\mathbb{R}^d;\mathbb{R}^d) \to \mathbb{R}$ such that

$$J(\Omega_{\vec{\theta}}) = J(\Omega) + L(\vec{\theta}) + o(\vec{\theta}), \quad \text{with} \quad \lim_{w \to 0} \frac{|o(\theta)|}{||\vec{\theta}||_{W^{1,\infty}(\mathbb{R}^d;\mathbb{R}^d)}} = 0$$

We call $J'(\Omega) := L$, the **shape derivative** of J at Ω.

1 Introduction
- Weak formulation specification
- Integration with MMG
- Integration with ISCD tools

2 Geometrical shape optimization
- Mathematical framework
- Applied example
- Implementation
- Results
- Conclusion

3 Appendix

We consider the mimization of the *compliance* (energy dissipated inside the structure) and the amount of material used:

$$\min_{\Omega \subset D} J(\Omega) = \underbrace{\int_\Omega Ae(\vec{u}) : e(\vec{u})\ dx}_{\text{Compliance}} + \ell \underbrace{\int_\Omega dx}_{\mathrm{Vol}(\Omega)}$$

where D is a "hold-all" domain, $\ell > 0$ is a volume penalization parameter and $\vec{u}$ is the weak solution to:

$$(1) \quad \begin{cases} -\nabla \cdot (Ae(\vec{u})) = 0 & \text{in } \Omega \\ \vec{u} = 0 & \text{on } \Gamma_D \\ Ae(\vec{u})\vec{n} = f & \text{on } \Gamma_N \\ Ae(\vec{u})\vec{n} = 0 & \text{on } \Gamma \end{cases},$$

with

$$Ae(\vec{u}) = 2\mu e(\vec{u}) + \mathrm{tr}(e(\vec{u}))I, \quad e(\vec{u}) = \frac{\nabla \vec{u} + \nabla \vec{u}^T}{2},$$

where I is the identity matrix and λ, μ are the Lamé parameters and $\vec{n}$ is the normal vector to $\partial\Omega$.

Theorem

The shape derivative $J'(\Omega)(\theta)$ of $J(\Omega)$ in the direction $\vec{\theta} \in W^{1,\infty}(\mathbb{R}^n, \mathbb{R}^n)$ is given by:

$$J'(\Omega)(\vec{\theta}) = -\int_\Gamma (Ae(\vec{u}) : e(\vec{u}) - \ell)\ \theta \cdot \vec{n}\ dx$$

where $\vec{u}$ is solution to (1).

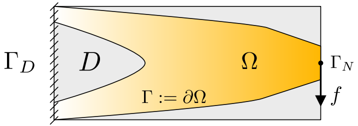
The benchmark cantilever test case

We consider the mimization of the *compliance* (energy dissipated inside the structure) and the amount of material used:

$$\min_{\Omega \subset D} J(\Omega) = \underbrace{\int_\Omega Ae(\vec{u}) : e(\vec{u})\ dx}_{\text{Compliance}} + \ell \underbrace{\int_\Omega dx}_{\mathrm{Vol}(\Omega)}$$

where D is a "hold-all" domain, $\ell > 0$ is a volume penalization parameter and $\vec{u}$ is the weak solution to:

$$(1) \qquad \begin{cases} -\nabla \cdot (Ae(\vec{u})) = 0 & \text{in } \Omega \\ \vec{u} = 0 & \text{on } \Gamma_D \\ Ae(\vec{u})\vec{n} = f & \text{on } \Gamma_N \\ Ae(\vec{u})\vec{n} = 0 & \text{on } \Gamma \end{cases},$$



with

$$Ae(\vec{u}) = 2\mu e(\vec{u}) + \mathrm{tr}(e(\vec{u}))I\ , \quad e(\vec{u}) = \frac{\nabla \vec{u} + \nabla \vec{u}^T}{2}\ ,$$

where I is the identity matrix and λ, μ are the Lamé parameters and $\vec{n}$ is the normal vector to $\partial\Omega$.

Theorem

The shape derivative $J'(\Omega)(\theta)$ of $J(\Omega)$ in the direction $\vec{\theta} \in W^{1,\infty}(\mathbb{R}^n, \mathbb{R}^n)$ is given by:

$$J'(\Omega)(\vec{\theta}) = -\int_\Gamma (Ae(\vec{u}) : e(\vec{u}) - \ell)\ \theta \cdot \vec{n}\ dx$$

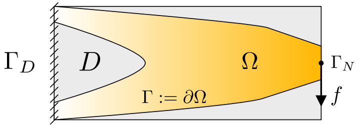
where $\vec{u}$ is solution to (1).

We consider the mimization of the *compliance* (energy dissipated inside the structure) and the amount of material used:

$$\min_{\Omega \subset D} J(\Omega) = \underbrace{\int_\Omega Ae(\vec{u}) : e(\vec{u}) \, dx}_{\text{Compliance}} + \ell \underbrace{\int_\Omega dx}_{\text{Vol}(\Omega)}$$

where D is a "hold-all" domain, $\ell > 0$ is a volume penalization parameter and $\vec{u}$ is the weak solution to:

$$(1) \quad \begin{cases} -\nabla \cdot (Ae(\vec{u})) = 0 & \text{in } \Omega \\ \vec{u} = 0 & \text{on } \Gamma_D \\ Ae(\vec{u})\vec{n} = f & \text{on } \Gamma_N \\ Ae(\vec{u})\vec{n} = 0 & \text{on } \Gamma \end{cases},$$



with

$$Ae(\vec{u}) = 2\mu e(\vec{u}) + \mathrm{tr}(e(\vec{u}))I \,, \quad e(\vec{u}) = \frac{\nabla \vec{u} + \nabla \vec{u}^T}{2} \,,$$

where I is the identity matrix and λ, μ are the Lamé parameters and $\vec{n}$ is the normal vector to $\partial\Omega$.

Theorem

The shape derivative $J'(\Omega)(\theta)$ of $J(\Omega)$ in the direction $\vec{\theta} \in W^{1,\infty}(\mathbb{R}^n, \mathbb{R}^n)$ is given by:

$$J'(\Omega)(\vec{\theta}) = -\int_\Gamma (Ae(\vec{u}) : e(\vec{u}) - \ell) \, \theta \cdot \vec{n} \, dx$$

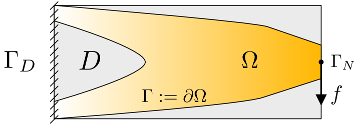
where $\vec{u}$ is solution to (1).

We consider the mimization of the *compliance* (energy dissipated inside the structure) and the amount of material used:

$$\min_{\Omega \subset D} J(\Omega) = \underbrace{\int_\Omega Ae(\vec{u}) : e(\vec{u})\, dx}_{\text{Compliance}} + \ell \underbrace{\int_\Omega dx}_{\text{Vol}(\Omega)}$$

where D is a "hold-all" domain, $\ell > 0$ is a volume penalization parameter and $\vec{u}$ is the weak solution to:

$$(1) \qquad \begin{cases} -\nabla \cdot (Ae(\vec{u})) = 0 & \text{in } \Omega \\ \vec{u} = 0 & \text{on } \Gamma_D \\ Ae(\vec{u})\vec{n} = f & \text{on } \Gamma_N \\ Ae(\vec{u})\vec{n} = 0 & \text{on } \Gamma \end{cases},$$



with

$$Ae(\vec{u}) = 2\mu e(\vec{u}) + \mathrm{tr}(e(\vec{u}))I \,, \quad e(\vec{u}) = \frac{\nabla \vec{u} + \nabla \vec{u}^T}{2} \,,$$

where I is the identity matrix and λ, μ are the Lamé parameters and $\vec{n}$ is the normal vector to $\partial\Omega$.

Theorem

The shape derivative $J'(\Omega)(\theta)$ of $J(\Omega)$ in the direction $\vec{\theta} \in W^{1,\infty}(\mathbb{R}^n, \mathbb{R}^n)$ is given by:

$$J'(\Omega)(\vec{\theta}) = -\int_\Gamma (Ae(\vec{u}) : e(\vec{u}) - \ell)\ \theta \cdot \vec{n}\, dx$$

where $\vec{u}$ is solution to (1).

We refer to the two very comprehensive books Osher and Fedkiw (2005), and Sethian (1999).

Representation of Ω via a level-set function.

$$\forall x \in D, \quad \begin{cases} \phi(x) < 0 & \text{if } \Omega \\ \phi(x) = 0 & \text{if } \partial\Omega \\ \phi(x) > 0 & \text{if } D \setminus \overline{\Omega} \end{cases},$$

Motion of Ω is performed via the advection of a level-set.

$$\frac{\partial \phi}{\partial t} + \theta(x) \cdot \nabla \phi(x) = 0 \quad (t, x) \in [0, T] \times D$$

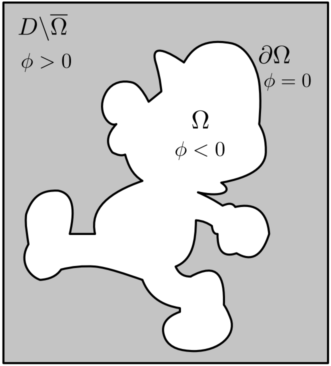
We refer to the two very comprehensive books Osher and Fedkiw (2005), and Sethian (1999).

Representation of Ω via a level-set function.

$$\forall x \in D, \quad \begin{cases} \phi(x) < 0 & \text{if } \Omega \\ \phi(x) = 0 & \text{if } \partial\Omega \\ \phi(x) > 0 & \text{if } D\backslash\overline{\Omega} \end{cases},$$

Motion of Ω is performed via the advection of a level-set.

$$\frac{\partial \phi}{\partial t} + \theta(x) \cdot \nabla \phi(x) = 0 \quad (t, x) \in [0, T] \times D$$



Gradient descent algorithm

Algorithm Shape optimization of a linearly elastic structure.

for $i \leftarrow$ `MAX_IT` **do**

Find $\vec{u}$ solution to:

$$\begin{cases} -\nabla \cdot (Ae(\vec{u})) = 0 & \text{in } \Omega \\ \vec{u} = 0 & \text{on } \Gamma_D \\ Ae(\vec{u})\vec{n} = f & \text{on } \Gamma_N \\ Ae(\vec{u})\vec{n} = 0 & \text{on } \Gamma \end{cases}$$

Compute the shape gradient $\vec{\theta} \in W^{1,\infty}(\mathbb{R}^n, \mathbb{R}^n)$ from:

$$J'(\Omega)(\vec{\theta}) = -\int_\Gamma (Ae(\vec{u}) : e(\vec{u}) - \ell)\ \theta \cdot \vec{n}\ dx$$

Compute the signed distance function ϕ to Ω.

$$\phi(x) = \begin{cases} d(x, \partial\Omega) & \text{if } x \in D \backslash \overline{\Omega} \\ 0 & \text{if } x \in \partial\Omega \\ -d(x, \partial\Omega) & \text{if } x \in \Omega \end{cases}$$

(Gradient descent step) Advect the signed distance function with the shape gradient $\vec{\theta}$ we just computed.

$$\frac{\partial \phi}{\partial t}(x) + \theta(x) \cdot \nabla \phi(x) = 0 \quad (t, x) \in [0, T] \times D$$

Recover the implicit domain.

$$\Omega \leftarrow \{x \in D : \phi(x) < 0\}$$

end for

1 Introduction
- Weak formulation specification
- Integration with MMG
- Integration with ISCD tools

2 Geometrical shape optimization
- Mathematical framework
- Applied example
- Implementation
- Results
- Conclusion

3 Appendix

Header inclusion and parameter definition

We include the modules and define the optimization parameters.

```
#include <Rodin/Mesh.h>
#include <Rodin/Solver.h>
#include <Rodin/Variational.h>
#include <RodinExternal/MMG.h>

using namespace Rodin;
using namespace Rodin::External;
using namespace Rodin::Variational;
static const int interior = 1, // Omega attribute
                 exterior = 2; // Exterior attribute

// Define boundary attributes
static const int Gamma0 = 1, GammaD = 2, GammaN = 3, Gamma = 4;

// Lamé coefficients
static const double mu = 0.3846;
static const double lambda = 0.5769;

// Optimization parameters
static const size_t maxIt = 250; // max optimization iterations
static const double eps = 1e-6; // epsilon
static const double hmax = 0.05; // maximal edge length
static const double ell = 1.0; // volume penalization
static const double alpha = 4 * hmax * hmax; // regularization parameter
```

We include the modules and define the optimization parameters.

```cpp
#include <Rodin/Mesh.h>
#include <Rodin/Solver.h>
#include <Rodin/Variational.h>
#include <RodinExternal/MMG.h>

using namespace Rodin;
using namespace Rodin::External;
using namespace Rodin::Variational;
static const int interior = 1, // Omega attribute
                 exterior = 2; // Exterior attribute

// Define boundary attributes
static const int Gamma0 = 1, GammaD = 2, GammaN = 3, Gamma = 4;

// Lamé coefficients
static const double mu = 0.3846;
static const double lambda = 0.5769;

// Optimization parameters
static const size_t maxIt = 250; // max optimization iterations
static const double eps = 1e-6; // epsilon
static const double hmax = 0.05; // maximal edge length
static const double ell = 1.0; // volume penalization
static const double alpha = 4 * hmax * hmax; // regularization parameter
```

Now we load the mesh, specify our linear solver and define the pull force.

```
int main(int, char**)
{
  // Load mesh
  Mesh D;
  D.load("levelset-cantilever2d-example.mesh");

  // UMFPack
  auto solver = Solver::UMFPack();

  // Pull force
  auto f = VectorFunction{0.0, -1.0};

  // Shape optimization loop
  for (size_t i = 0; i < maxIt; i++)
  {
    // ...
  }
}
```

State equation resolution

We compute the displacement field $\vec{u}$.

Find $\vec{u} \in {H^1_{\Gamma_D}(\Omega)}^d$ such that:

$$\forall \vec{v} \in {H^1_{\Gamma_D}(\Omega)}^d, \quad \int_\Omega Ae(\vec{u}) : e(\vec{v}) \, dx = \int_{\Gamma_N} \vec{f} \cdot \vec{v} \, ds \; .$$

Expanding out the terms:

$$\forall \vec{v} \in {H^1_{\Gamma_D}(\Omega)}^d, \quad \int_\Omega \lambda(\nabla \cdot \vec{u})(\nabla \cdot \vec{v}) + \frac{1}{2}\mu(\nabla \vec{u} + \nabla \vec{u}^T) : (\nabla \vec{v} + \nabla \vec{v}^T) \, dx = \int_{\Gamma_N} \vec{f} \cdot \vec{v} \, dx \; .$$

```
SubMesh Omega = D.trim(exterior);
int d = D.getSpaceDimension();
H1 Vh(Omega, d);
TrialFunction u(Vh);
TestFunction  v(Vh);
Problem elasticity(u, v);
elasticity =
    Integral(lambda * Div(u), Div(v))
  + Integral(
      0.5 * mu * (Jacobian(u) + Jacobian(u).T()), (Jacobian(v) + Jacobian(v).T()))
  - BoundaryIntegral(f, v).over(GammaN)
  + DirichletBC(u, VectorFunction{0, 0}).on(GammaD);
solver.solve(elasticity);
```

State equation resolution

We compute the displacement field $\vec{u}$.

Find $\vec{u} \in H^1_{\Gamma_D}(\Omega)^d$ such that:

$$\forall \vec{v} \in H^1_{\Gamma_D}(\Omega)^d, \quad \int_\Omega Ae(\vec{u}) : e(\vec{v}) \ dx = \int_{\Gamma_N} \vec{f} \cdot \vec{v} \ ds \ .$$

Expanding out the terms:

$$\forall \vec{v} \in H^1_{\Gamma_D}(\Omega)^d, \quad \int_\Omega \lambda(\nabla \cdot \vec{u})(\nabla \cdot \vec{v}) + \frac{1}{2}\mu(\nabla \vec{u} + \nabla \vec{u}^T) : (\nabla \vec{v} + \nabla \vec{v}^T) \ dx = \int_{\Gamma_N} \vec{f} \cdot \vec{v} \ dx \ .$$

```
SubMesh Omega = D.trim(exterior);
int d = D.getSpaceDimension();
H1 Vh(Omega, d);
TrialFunction u(Vh);
TestFunction  v(Vh);
Problem elasticity(u, v);
elasticity =
    Integral(lambda * Div(u), Div(v))
  + Integral(
      0.5 * mu * (Jacobian(u) + Jacobian(u).T()), (Jacobian(v) + Jacobian(v).T()))
  - BoundaryIntegral(f, v).over(GammaN)
  + DirichletBC(u, VectorFunction{0, 0}).on(GammaD);
solver.solve(elasticity);
```

State equation resolution

We compute the displacement field $\vec{u}$.

Find $\vec{u} \in H^1_{\Gamma_D}(\Omega)^d$ such that:

$$\forall \vec{v} \in H^1_{\Gamma_D}(\Omega)^d, \quad \int_\Omega Ae(\vec{u}) : e(\vec{v}) \; dx = \int_{\Gamma_N} \vec{f} \cdot \vec{v} \; ds \; .$$

Expanding out the terms:

$$\forall \vec{v} \in H^1_{\Gamma_D}(\Omega)^d, \quad \int_\Omega \lambda(\nabla \cdot \vec{u})(\nabla \cdot \vec{v}) + \frac{1}{2}\mu(\nabla \vec{u} + \nabla \vec{u}^T) : (\nabla \vec{v} + \nabla \vec{v}^T) \; dx = \int_{\Gamma_N} \vec{f} \cdot \vec{v} \; dx \; .$$

```
SubMesh Omega = D.trim(exterior);
int d = D.getSpaceDimension();
H1 Vh(Omega, d);
TrialFunction u(Vh);
TestFunction  v(Vh);
Problem elasticity(u, v);
elasticity =
    Integral(lambda * Div(u), Div(v))
  + Integral(
      0.5 * mu * (Jacobian(u) + Jacobian(u).T()), (Jacobian(v) + Jacobian(v).T()))
  - BoundaryIntegral(f, v).over(GammaN)
  + DirichletBC(u, VectorFunction{0, 0}).on(GammaD);
solver.solve(elasticity);
```

Computation of a regularized descent direction

The Hilbertian extension-regularization procedure allows to infer a **regularized shape gradient** from $J'(\Omega)(\vec{\theta})$ which is defined on all of D.

Find $\vec{\theta} \in H^1_{\Gamma_N}(D)^d$ such that:

$$\forall \vec{w} \in H^1_{\Gamma_N}(D)^d, \quad \underbrace{\alpha \int_D \nabla\vec{\theta} : \nabla\vec{w} + \vec{\theta}\cdot\vec{w}\, dx}_{\text{Inner product on } H^1(D)^d} - \underbrace{\int_\Gamma (Ae(\vec{u}) : e(\vec{w}) - \ell)\, \vec{\theta}\cdot\vec{n}\, dx}_{J'(\Omega)(\vec{\theta})} = 0.$$

```
H1 VhExt(D, d);
auto e = 0.5 * (Jacobian(u) + Jacobian(u).T());
auto Ae = 2.0 * mu * e + lambda * Trace(e) * IdentityMatrix(d);
auto n = Normal(d);

TrialFunction theta(VhExt);
TestFunction  w(VhExt);
Problem hilbert(theta, w);
hilbert = Integral(alpha * Jacobian(theta), Jacobian(w))
        + Integral(theta, w)
        - BoundaryIntegral(Dot(Ae, e) - ell, Dot(n, w)).over(Gamma)
        + DirichletBC(theta, VectorFunction{0, 0}).on(GammaN);
solver.solve(hilbert);
```

Computation of a regularized descent direction

The Hilbertian extension-regularization procedure allows to infer a **regularized shape gradient** from $J'(\Omega)(\vec{\theta})$ which is defined on all of D.

Find $\vec{\theta} \in H^1_{\Gamma_N}(D)^d$ such that:

$$\forall \vec{w} \in H^1_{\Gamma_N}(D)^d, \quad \underbrace{\alpha \int_D \nabla\vec{\theta} : \nabla\vec{w} + \vec{\theta} \cdot \vec{w} \, dx}_{\text{Inner product on } H^1(D)^d} - \underbrace{\int_\Gamma (Ae(\vec{u}) : e(\vec{w}) - \ell) \, \vec{\theta} \cdot \vec{n} \, dx}_{J'(\Omega)(\vec{\theta})} = 0.$$

```
H1 VhExt(D, d);
auto e = 0.5 * (Jacobian(u) + Jacobian(u).T());
auto Ae = 2.0 * mu * e + lambda * Trace(e) * IdentityMatrix(d);
auto n = Normal(d);

TrialFunction theta(VhExt);
TestFunction  w(VhExt);
Problem hilbert(theta, w);
hilbert = Integral(alpha * Jacobian(theta), Jacobian(w))
        + Integral(theta, w)
        - BoundaryIntegral(Dot(Ae, e) - ell, Dot(n, w)).over(Gamma)
        + DirichletBC(theta, VectorFunction{0, 0}).on(GammaN);
solver.solve(hilbert);
```

Computation of a regularized descent direction

The Hilbertian extension-regularization procedure allows to infer a **regularized shape gradient** from $J'(\Omega)(\vec{\theta})$ which is defined on all of D.

Find $\vec{\theta} \in H^1_{\Gamma_N}(D)^d$ such that:

$$\forall \vec{w} \in H^1_{\Gamma_N}(D)^d, \quad \underbrace{\alpha \int_D \nabla\vec{\theta} : \nabla\vec{w} + \vec{\theta} \cdot \vec{w} \, dx}_{\text{Inner product on } H^1(D)^d} - \underbrace{\int_\Gamma (Ae(\vec{u}) : e(\vec{w}) - \ell) \, \vec{\theta} \cdot \vec{n} \, dx}_{J'(\Omega)(\vec{\theta})} = 0.$$

```
H1 VhExt(D, d);
auto e = 0.5 * (Jacobian(u) + Jacobian(u).T());
auto Ae = 2.0 * mu * e + lambda * Trace(e) * IdentityMatrix(d);
auto n = Normal(d);

TrialFunction theta(VhExt);
TestFunction  w(VhExt);
Problem hilbert(theta, w);
hilbert = Integral(alpha * Jacobian(theta), Jacobian(w))
        + Integral(theta, w)
        - BoundaryIntegral(Dot(Ae, e) - ell, Dot(n, w)).over(Gamma)
        + DirichletBC(theta, VectorFunction{0, 0}).on(GammaN);
solver.solve(hilbert);
```

Now we distance the domain, advect the distance function and recover the implicit domain.

$$\phi(x) = \begin{cases} d(x, \partial\Omega) & \text{if } x \in D \backslash \overline{\Omega} \\ 0 & \text{if } x \in \partial\Omega \\ -d(x, \partial\Omega) & \text{if } x \in \Omega \end{cases}, \qquad \frac{\partial \phi}{\partial t} + \vec{\theta} \cdot \nabla \phi = 0$$

```
H1 Dh(D);
auto dist = MMG::Distancer(Dh).setInteriorDomain(interior).distance(D);
MMG::Advect(dist, g.getGridFunction()).step(0.001);
D = MMG::ImplicitDomainMesher().setBoundaryReference(Gamma).discretize(dist);
```

Distancing, advection and recovering the implicit domain

Now we distance the domain, advect the distance function and recover the implicit domain.

$$\phi(x) = \begin{cases} d(x, \partial\Omega) & \text{if } x \in D \setminus \overline{\Omega} \\ 0 & \text{if } x \in \partial\Omega \\ -d(x, \partial\Omega) & \text{if } x \in \Omega \end{cases}, \qquad \frac{\partial \phi}{\partial t} + \vec{\theta} \cdot \nabla \phi = 0$$

```
H1 Dh(D);
auto dist = MMG::Distancer(Dh).setInteriorDomain(interior).distance(D);
MMG::Advect(dist, g.getGridFunction()).step(0.001);
D = MMG::ImplicitDomainMesher().setBoundaryReference(Gamma).discretize(dist);
```

Distancing, advection and recovering the implicit domain

Now we distance the domain, advect the distance function and recover the implicit domain.

$$\phi(x) = \begin{cases} d(x, \partial\Omega) & \text{if } x \in D \setminus \overline{\Omega} \\ 0 & \text{if } x \in \partial\Omega \\ -d(x, \partial\Omega) & \text{if } x \in \Omega \end{cases}, \qquad \frac{\partial \phi}{\partial t} + \vec{\theta} \cdot \nabla \phi = 0$$

```
H1 Dh(D);
auto dist = MMG::Distancer(Dh).setInteriorDomain(interior).distance(D);
MMG::Advect(dist, g.getGridFunction()).step(0.001);
D = MMG::ImplicitDomainMesher().setBoundaryReference(Gamma).discretize(dist);
```

Distancing, advection and recovering the implicit domain

Now we distance the domain, advect the distance function and recover the implicit domain.

$$\phi(x) = \begin{cases} d(x, \partial\Omega) & \text{if } x \in D \backslash \overline{\Omega} \\ 0 & \text{if } x \in \partial\Omega \\ -d(x, \partial\Omega) & \text{if } x \in \Omega \end{cases}, \qquad \frac{\partial \phi}{\partial t} + \vec{\theta} \cdot \nabla \phi = 0$$

```
H1 Dh(D);
auto dist = MMG::Distancer(Dh).setInteriorDomain(interior).distance(D);
MMG::Advect(dist, g.getGridFunction()).step(0.001);
D = MMG::ImplicitDomainMesher().setBoundaryReference(Gamma).discretize(dist);
```

1 Introduction
- Weak formulation specification
- Integration with MMG
- Integration with ISCD tools

2 Geometrical shape optimization
- Mathematical framework
- Applied example
- Implementation
- Results
- Conclusion

3 Appendix

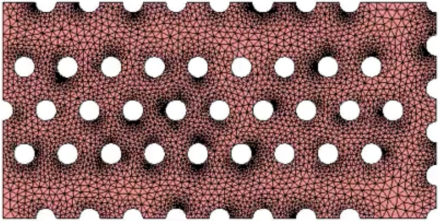


Figure: Optimization process of a cantilever in 2D.

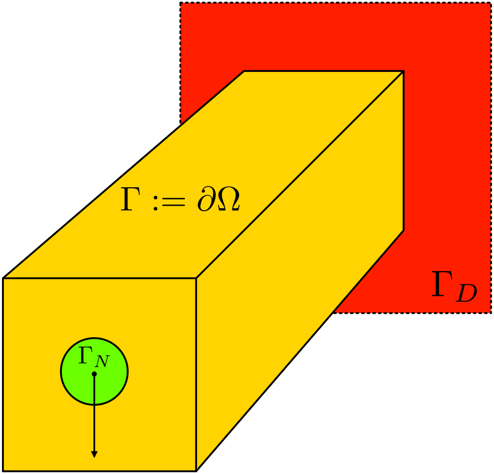


Figure: Configuration in 3D.

Cantilever in 3D

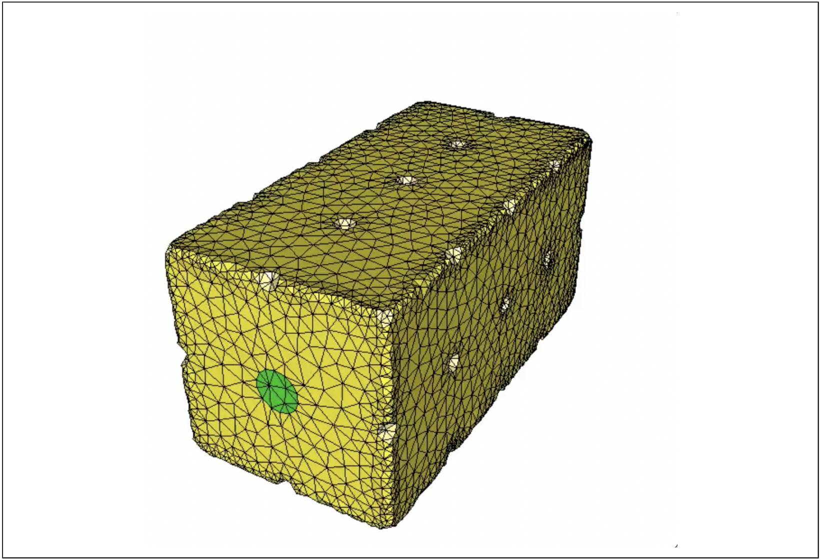


Figure: Front view of the optimization process of the cantilever in 3D.

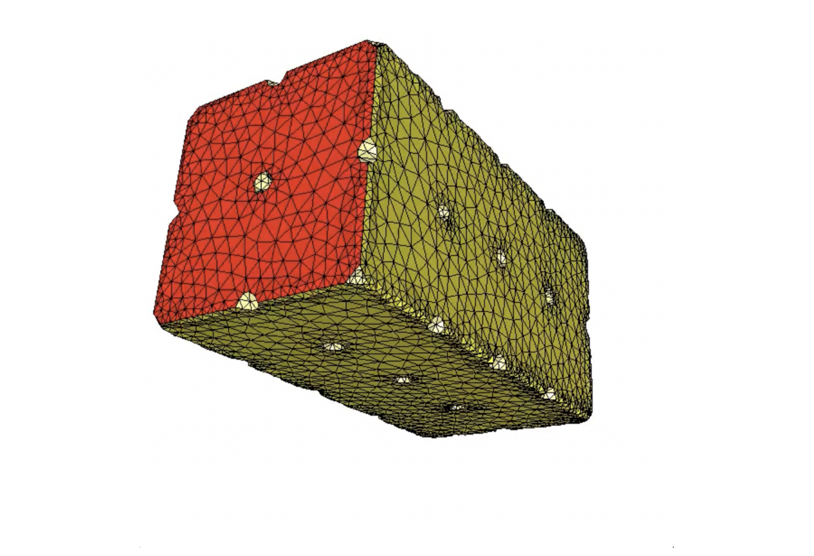


Figure: Back view of the optimization process of the cantilever in 3D.

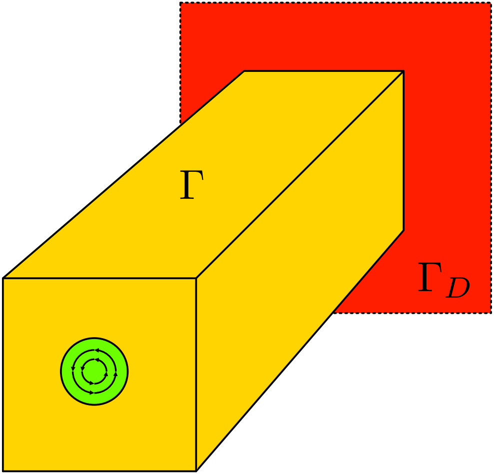


Figure: Configuration in 3D, now with a rotational force on Γ_N.

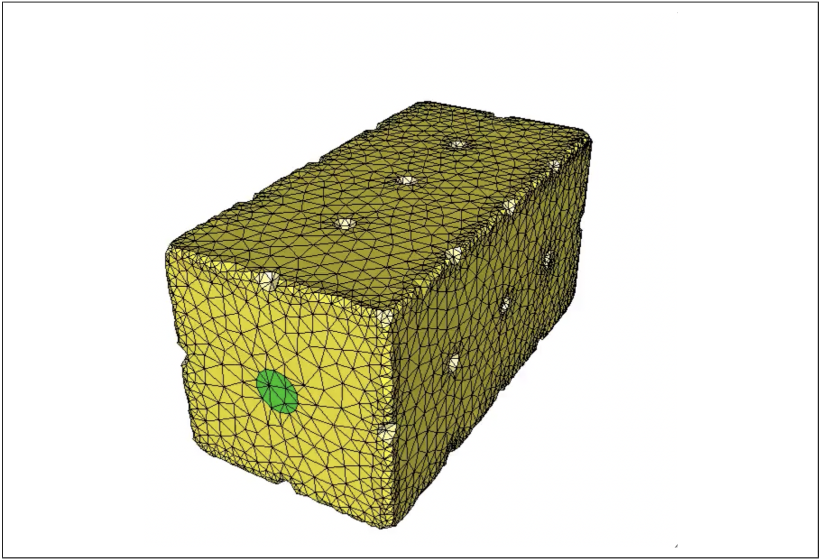


Figure: Back view of the optimization process, now with a rotational force.

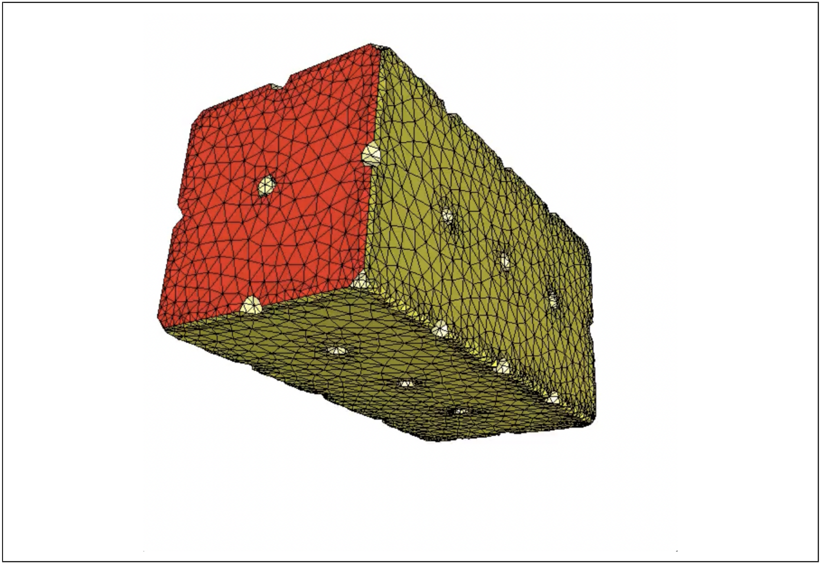


Figure: Back view of the optimization process, now with a rotational force.

1. Introduction
   - Weak formulation specification
   - Integration with MMG
   - Integration with ISCD tools
2. Geometrical shape optimization
   - Mathematical framework
   - Applied example
   - Implementation
   - Results
   - Conclusion
3. Appendix

All examples can be found here:
`https://github.com/cbritopacheco/rodin/blob/master/examples/`

Similar software which support automated assembly from syntactic expressions (not a comprehensive list):

- **Rheolef**. Saramito (2020).
  `https://membres-ljk.imag.fr/Pierre.Saramito/rheolef/html/index.html`
- **FreeFem++**. Hecht et al. (2005).
  `https://freefem.org/`
- **FEniCS**. Alnæs et al. (2015).
  `https://fenicsproject.org/`
- **Feel++**. Prud'Homme et al. (2012).
  `http://www.feelpp.org/`

All examples can be found here:
`https://github.com/cbritopacheco/rodin/blob/master/examples/`

Similar software which support automated assembly from syntactic expressions (not a comprehensive list):

- **Rheolef**. Saramito (2020).
  `https://membres-ljk.imag.fr/Pierre.Saramito/rheolef/html/index.html`
- **FreeFem++**. Hecht et al. (2005).
  `https://freefem.org/`
- **FEniCS**. Alnæs et al. (2015).
  `https://fenicsproject.org/`
- **Feel++**. Prud'Homme et al. (2012).
  `http://www.feelpp.org/`

Thank you for your attention. Please do not hesitate to ask any questions.

Definition

Let $J'(\Omega)(\theta)$ be a the **shape derivative** of a shape functional J. Let $H \subset W^{1,\infty}(\mathbb{R}^d, \mathbb{R}^d)$ be a Hilbert space, with inner product $\langle \cdot, \cdot \rangle : H \times H \to \mathbb{R}$. The **shape gradient in H** is defined as the unique element $g \in H$ such that:

$$\forall w \in H, \quad \langle g, w \rangle = J'(\Omega)(w) \, .$$

- Note $-g$ is a descent direction since for $\tau > 0$ small enough:

$$\begin{aligned} J(\Omega_{\tau g}) &= J(\Omega) - \tau J'(\Omega)(g) + o(\tau) \\ &= J(\Omega) - \tau \underbrace{\langle g, g \rangle}_{>0} + o(\tau) \\ &< J(\Omega) \, . \end{aligned}$$

- The choice of H will determine the regularity of the gradient g !

Definition

Let $J'(\Omega)(\theta)$ be a the **shape derivative** of a shape functional J. Let $H \subset W^{1,\infty}(\mathbb{R}^d, \mathbb{R}^d)$ be a Hilbert space, with inner product $\langle \cdot, \cdot \rangle : H \times H \to \mathbb{R}$. The **shape gradient in H** is defined as the unique element $g \in H$ such that:

$$\forall w \in H, \quad \langle g, w \rangle = J'(\Omega)(w) \ .$$

- Note $-g$ is a descent direction since for $\tau > 0$ small enough:

$$\begin{aligned} J(\Omega_{\tau g}) &= J(\Omega) - \tau J'(\Omega)(g) + o(\tau) \\ &= J(\Omega) - \tau \underbrace{\langle g, g \rangle}_{>0} + o(\tau) \\ &< J(\Omega) \ . \end{aligned}$$

- The choice of H will determine the regularity of the gradient g !

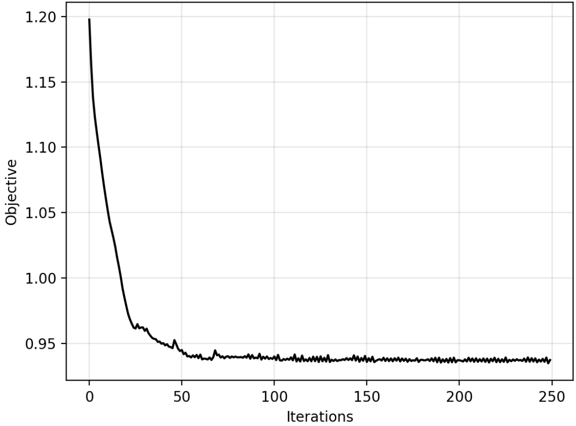


Figure: Evolution of the objective when minimizing the 2D cantilever.

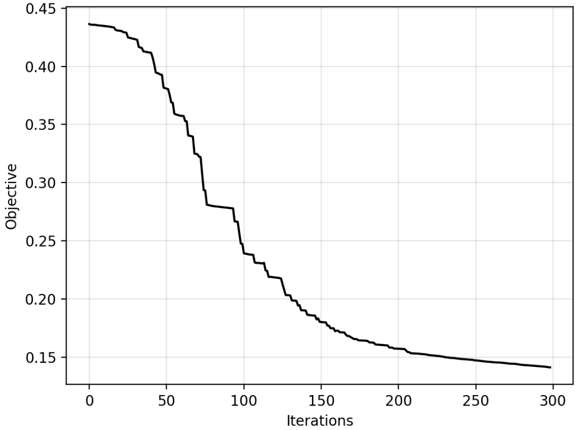


Figure: Evolution of the objective when minimizing the 3D cantilever.

These are shortcomings or missing features that will be implemented in due time or are already in the course of being implemented.

- No support for mixed bilinear forms.
- No support for systems of equations (e.g. Navier-Stokes).
- No support for discontinuous Galerkin methods.
- No `L2`, `HDiv`, `HCurl` finite element spaces as of yet.
- No parallel objects as of yet.

Rodin provides public access to the underlying MFEM objects.

```
// Mesh
Mesh mesh;
mfem::Mesh& meshHandle = mesh.getHandle();

// FiniteElementSpace
H1 fes(mesh);
mfem::FiniteElementSpace& fesHandle = fes.getHandle();

// GridFunction
GridFunction u(fes);
mfem::GridFunction& uHandle = u.getHandle();
```

Assembly

Rodin supports general assembly of bilinear forms (and also linear forms).

```
void getElementMatrix(const Bilinear::Assembly::Common& as) const
{
   auto& trial = m_prod.getLHS();
   auto& test = m_prod.getRHS();

   as.mat.SetSize(
    test.getDOFs(as.trial, as.trans), trial.getDOFs(as.test, as.trans));
   as.mat = 0.0;

   const int order = getIntegrationOrder(as);
   const mfem::IntegrationRule* ir =
      &mfem::IntRules.Get(as.trans.GetGeometryType(), order);
   ShapeComputator shapeCompute;
   for (int i = 0; i < ir->GetNPoints(); i++)
   {
      const mfem::IntegrationPoint &ip = ir->IntPoint(i);
      as.trans.SetIntPoint(&ip);
      mfem::Add(as.mat,
            m_prod.getElementMatrix(
              as.trial, as.test, as.trans, ip, shapeCompute),
            as.trans.Weight() * ip.weight,
            as.mat);
   }
}
```

```
template <class Integrand>
class Integral;

template <>
class Integral<
  Dot<ShapeFunctionBase<TrialSpace>, ShapeFunctionBase<TestSpace>>>
   : public BilinearFormIntegratorBase
{ /* Code to assemble bilinear form */ };
```

For example, the following formal expression:

$$\int_\Omega (f \nabla u) \cdot \nabla v \; dx \; ,$$

can be identified in code via:

```
template <class FES>
class Integral<
  Dot<
    Mult<FunctionBase, Grad<ShapeFunction<FES, TrialSpace>>>,
    Grad<ShapeFunction<FES, TestSpace>>>>
  : public Integral<
      Dot<ShapeFunctionBase<TrialSpace>, ShapeFunctionBase<TestSpace>>>
{ /* Will use mfem::DifussionIntegrator with device assembly */ };
```

At the end, this enables us to optimize the assembly of any expression and utilize features such as MFEM's device assembly.

```
template <class Integrand>
class Integral;

template <>
class Integral<
  Dot<ShapeFunctionBase<TrialSpace>, ShapeFunctionBase<TestSpace>>>
   : public BilinearFormIntegratorBase
{ /* Code to assemble bilinear form */ };
```

For example, the following formal expression:

$$\int_{\Omega} (f \nabla u) \cdot \nabla v \; dx \ ,$$

can be identified in code via:

```
template <class FES>
class Integral<
  Dot<
    Mult<FunctionBase, Grad<ShapeFunction<FES, TrialSpace>>>,
    Grad<ShapeFunction<FES, TestSpace>>>>
  : public Integral<
      Dot<ShapeFunctionBase<TrialSpace>, ShapeFunctionBase<TestSpace>>>
{ /* Will use mfem::DifussionIntegrator with device assembly */ };
```

At the end, this enables us to optimize the assembly of any expression and utilize features such as MFEM's device assembly.

Allaire, G., Dapogny, C., and Jouve, F. (2021). Shape and topology optimization. In *Handbook of Numerical Analysis*, volume 22, pages 1–132. Elsevier.

Allaire, G., Jouve, F., and Toader, A.-M. (2002). A level-set method for shape optimization. *Comptes Rendus Mathematique*, 334(12):1125–1130.

Allaire, G. and Schoenauer, M. (2007). *Conception optimale de structures*, volume 58. Springer.

Alnæs, M., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M. E., and Wells, G. N. (2015). The fenics project version 1.5. *Archive of Numerical Software*, 3(100).

Amstutz, S., Dapogny, C., and Ferrer, A. (2022). A consistent approximation of the total perimeter functional for topology optimization algorithms. *ESAIM: Control, Optimisation and Calculus of Variations*, 28:18.

Bendsoe, M. P. and Sigmund, O. (2003). *Topology optimization: theory, methods, and applications*. Springer Science & Business Media.

Dapogny, C., Dobrzynski, C., and Frey, P. (2014). Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems. *Journal of computational physics*, 262:358–378.

Dapogny, C., Faure, A., Michailidis, G., Allaire, G., Couvelas, A., and Estevez, R. (2017). Geometric constraints for shape and topology optimization in architectural design. *Computational Mechanics*, 59(6):933–965.

Dapogny, C. and Frey, P. (2012). Computation of the signed distance function to a discrete contour on adapted triangulation. *Calcolo*, 49(3):193–219.

Dapogny, C., Lebbe, N., and Oudet, E. (2020). Optimization of the shape of regions supporting boundary conditions. *Numerische Mathematik*, 146(1):51–104.

Delfour, M. C. and Zolésio, J.-P. (2011). *Shapes and geometries: metrics, analysis, differential calculus, and optimization*. SIAM.

Feppon, F., Allaire, G., Dapogny, C., and Jolivet, P. (2020). Topology optimization of thermal fluid–structure systems using body-fitted meshes and parallel computing. *Journal of Computational Physics*, 417:109574.

Hecht, F., Pironneau, O., Le Hyaric, A., and Ohtsuka, K. (2005). Freefem++ manual.

Henrot, A. and Pierre, M. (2018). Shape variation and optimization. a geometrical analysis.

Osher, S. and Fedkiw, R. P. (2005). *Level set methods and dynamic implicit surfaces*, volume 1. Springer New York.

Prud'Homme, C., Chabannes, V., Doyeux, V., Ismail, M., Samake, A., and Pena, G. (2012). Feel++: A computational framework for galerkin methods and advanced numerical methods. In *ESAIM: Proceedings*, volume 38, pages 429–455. EDP Sciences.

Saramito, P. (2020). Efficient c++ finite element computing with rheolef.

Sethian, J. A. (1999). *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, volume 3. Cambridge university press.