FEniCS Course

Lecture 8: From sensitivities to optimisation

Contributors
Simon Funke

What is PDE-constrained optimisation?

Optimisation problems where at least one constrained is a partial differential equation

Applications

- Data assimilation.
  Example: Weather modelling.
- Shape and topology optimisation.
  Example: Optimal shape of an aerfoil.
- Parameter estimation.
- Optimal control.
- ...

What is PDE-constrained optimisation?

Optimisation problems where at least one constrained is a partial differential equation

Applications

- Data assimilation.
  Example: Weather modelling.
- Shape and topology optimisation.
  Example: Optimal shape of an aerfoil.
- Parameter estimation.
- Optimal control.
- ...

Hello World of PDE-constrained optimisation!

We will solve the optimal control of the Poisson equation:

$$\min_{u,m} \frac{1}{2} \int_\Omega \|u - u_d\|^2 \, \mathrm{d}x + \frac{\alpha}{2} \int_\Omega \|m\|^2 \, \mathrm{d}x$$

subject to

$$\begin{aligned} -\Delta u &= m \quad && \text{in } \Omega \\ u &= u_0 && \text{on } \partial\Omega \end{aligned}$$

- This problem can be physically interpreted as: Find the heating/cooling term m for which u best approximates the desired heat distribution u_d.
- The second term in the objective functional, known as Thikhonov regularisation, ensures existence and uniqueness for $\alpha > 0$.

Hello World of PDE-constrained optimisation!

We will solve the optimal control of the Poisson equation:

$$\min_{u,m} \frac{1}{2} \int_\Omega \|u - u_d\|^2 \,\mathrm{d}x + \frac{\alpha}{2} \int_\Omega \|m\|^2 \,\mathrm{d}x$$

subject to

$$\begin{aligned} -\Delta u &= m \quad && \text{in } \Omega \\ u &= u_0 && \text{on } \partial\Omega \end{aligned}$$

- This problem can be physically interpreted as: Find the heating/cooling term m for which u best approximates the desired heat distribution u_d.
- The second term in the objective functional, known as Thikhonov regularisation, ensures existence and uniqueness for $\alpha > 0$.

The canconical abstract form

$$\min_{u,m} J(u,m)$$

$$\text{subject to:}$$

$$F(u,m) = 0,$$

with

- the objective functional J.
- the parameter m.
- the PDE operator F with solution u, parametrised by m.

The reduced problem

$$\min_m \tilde{J}(m) = J(u(m), m)$$

with

- the reduced functional $\tilde{J}$.
- the parameter m.

How do we solve this problem?

- Gradient descent.
- Newton method.
- Quasi-Newton methods.

The reduced problem

$$\min_m \tilde{J}(m) = J(u(m), m)$$

with

- the reduced functional $\tilde{J}$.
- the parameter m.

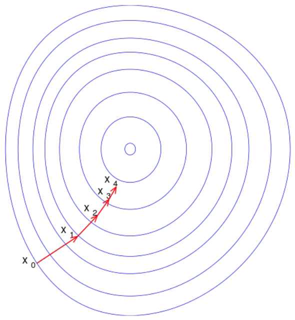
How do we solve this problem?

- Gradient descent.
- Newton method.
- Quasi-Newton methods.

Gradient descent

Algorithm

1. Choose initial parameter value m^0 and $\gamma > 0$.
2. For $i = 0, 1, \ldots :$
   - $m^{i+1} = m^i - \gamma \nabla J(m^i)$



Features

\+ Easy to implement.

– Slow convergence.

Newton method

Optimisation problem: $\min_m \tilde{J}(m)$.

Optimality condition:

$$\nabla \tilde{J}(m) = 0. \quad (1)$$

Newton method applied to (1):

1. Choose initial parameter value m^0.
2. For $i = 0, 1, \ldots$:
   - $H(J)\delta m = -\nabla J(m^i)$, where H denotes the Hessian.
   - $m^{i+1} = m^i + \delta m$

Features

+ Fast (locally quadratic) convergence.
– Requires iteratively solving a linear system with the Hessian, which might require many Hessian action computations.
– Hessian might not be positive definite, resulting in an update δm which is not a descent direction.

Newton method

Optimisation problem: $\min_m \tilde{J}(m)$.
Optimality condition:

$$\nabla \tilde{J}(m) = 0. \tag{1}$$

Newton method applied to (1):

1. Choose initial parameter value m^0.
2. For $i = 0, 1, \ldots$:
   - $H(J)\delta m = -\nabla J(m^i)$, where H denotes the Hessian.
   - $m^{i+1} = m^i + \delta m$

Features

+ Fast (locally quadratic) convergence.
− Requires iteratively solving a linear system with the Hessian, which might require many Hessian action computations.
− Hessian might not be positive definite, resulting in an update δm which is not a descent direction.

Newton method

Optimisation problem: $\min_m \tilde{J}(m)$.
Optimality condition:

$$\nabla \tilde{J}(m) = 0. \tag{1}$$

Newton method applied to (1):

1. Choose initial parameter value m^0.
2. For $i = 0, 1, \ldots$:
   - $H(J)\delta m = -\nabla J(m^i)$, where H denotes the Hessian.
   - $m^{i+1} = m^i + \delta m$

Features

+ Fast (locally quadratic) convergence.
− Requires iteratively solving a linear system with the Hessian, which might require many Hessian action computations.
− Hessian might not be positive definite, resulting in an update δm which is not a descent direction.

Newton method

Optimisation problem: $\min_m \tilde{J}(m)$.
Optimality condition:

$$\nabla \tilde{J}(m) = 0. \tag{1}$$

Newton method applied to (1):

1. Choose initial parameter value m^0.
2. For $i = 0, 1, \ldots$:
   - $H(J)\delta m = -\nabla J(m^i)$, where H denotes the Hessian.
   - $m^{i+1} = m^i + \delta m$

Features

\+ Fast (locally quadratic) convergence.

– Requires iteratively solving a linear system with the Hessian, which might require many Hessian action computations.

– Hessian might not be positive definite, resulting in an update δm which is not a descent direction.

Quasi-Newton methods

Like Newton method, but use approximate, low-rank Hessian approximation using gradient information only. A common approximation method is *BFGS*.

Features

+ Robust: Hessian approximation is always positive definite.
+ Cheap: No Hessian computation required, only gradient computations.
- Only superlinear convergence rate.

Quasi-Newton methods

Like Newton method, but use approximate, low-rank Hessian approximation using gradient information only. A common approximation method is *BFGS*.

Features

- \+ Robust: Hessian approximation is always positive definite.
- \+ Cheap: No Hessian computation required, only gradient computations.
- – Only superlinear convergence rate.

Solving the optimal Poisson problem

```
from dolfin import *
from dolfin_adjoint import *

# Solve Poisson problem
# ...

J = Functional(inner(s, s)*dx)
m = SteadyParameter(f)

rf = ReducedFunctional(J, m)
m_opt = minimize(rf, method="L-BFGS-B", tol=1e-2)
```

Tipps

- You can call **print_optimization_methods()** to list all available methods.
- Use **maximize** if you want to solve a maximisation problem.

Solving the optimal Poisson problem

```
from dolfin import *
from dolfin_adjoint import *

# Solve Poisson problem
# ...

J = Functional(inner(s, s)*dx)
m = SteadyParameter(f)

rf = ReducedFunctional(J, m)
m_opt = minimize(rf, method="L-BFGS-B", tol=1e-2)
```

Tipps

- You can call **print_optimization_methods()** to list all available methods.
- Use **maximize** if you want to solve a maximisation problem.

Bound constraints

Sometimes it is usefull to specify lower and upper bounds for parameters:

$$l_b \leq m \leq u_b. \tag{2}$$

Example:

```
lb = interpolate(0, V)
ub = interpolate(Expression("x[0]"), V)
m_opt = minimize(rf, method="L-BFGS-B",
    bounds=[lb, ub])
```

Note: Not all optimisation algorithms support bound constraints.

Inequality constraints

Sometimes it is usefull to specify (in-)equality constraints on the parameters:

$$g(m) \leq 0. \tag{3}$$

You can do that by overloading the **InequalityConstraint** class.
For more information visit the *Example* section on `dolfin-adjoint.org`.

The FEniCS challenge!

1. Solve the "Hello world" PDE-constrained optimisation problem on the unit square with $u_d(x,y) = \sin(\pi x)\sin(\pi y)$, homogenous boundary conditions and $\alpha = 10^{-6}$.
2. Compute the difference between optimised heat profile and u_d before and after the optimisation.
3. Use the optimisation algorithms SLSQP, Newton-CG and L-BFGS-B and compare them.
4. What happens if you increase α?