PDE Constrained Optimization

In the broad context, the physics based machine learning can be formulated as a PDE-constrained optimization problem. The PDE-constrained optimization problem aims at finding the optimal design variables such that the objective function is minimized and the constraints--usually described by PDEs--are satisfied. PDE-contrained optimization has a large variety of applications, such as optimal control and inverse problem. The topic is at the intersection of numerical PDE discretization, mathematical optimization, software design, and physics modeling.

Mathematically, a PDE-constrained optimization can be formulated as

$$\begin{aligned} \min_{y\in\mathcal{Y}, u\in\mathcal{U}} \ & J(y,u) & \\ \text{s.t. } & F(y,u) = 0 & \text{the governing PDEs} \\ & c(y,u) = 0 & \text{additional equality constraints} \\ & h(y,u) \geq 0 & \text{additional inequality constraints} \end{aligned}$$

Here J is the objective function, y is the **design variable**, u is the **state variable**.

In this section, we will focus on the PDE-constrained optimization with only the governing PDE constraints, and we consider a discretize-then-optimize and gradient-based optimization approach. Specifically, the objective function and the PDEs are first discretized numerically, leading to a constrained optimization problem with a finite dimensional optimization variable. We use gradient-based optimization because it provides fast convergence and an efficient way to integrate optimization and simulation. However, this approach requires insight into the simulator and can be quite all-consuming to obtain the gradients.

Example

We consider an exemplary PDE-constrained optimization problem: assume we want to have a specific temperature distribution on a metal bar $\bar{u}(x)$ by imposing a heat source $y(x)$ on the bar

$$\begin{aligned} \min_{y\in\mathcal{Y}, u\in\mathcal{U}} \ & L(y,u) = \frac{1}{2}\int_0^1 \|u(x) - \bar{u}(x)\|^2 dx + \frac{\rho}{2}\int_0^1 y(x)^2 dx \\ \text{s.t. } & f(y,u) = 0 \end{aligned}$$

Here $f(y,u) = 0$ is the static heat equation with the boundary conditions.

$$c(x)u_{xx}(x) = y(x), \quad x\in(0,1), \quad u(0) = u_0, u(1) = u_1$$

where $c(x)$ is the diffusivity coefficient, u_0 and u_1 are fixed boundaries.

After discretization, the optimization problem becomes

$$\begin{aligned}\min_{y,u}\ & J(y,u) = \frac{1}{2}\|u-\bar u\|_2 + \frac{\rho}{2}\|y\|^2 \\ \text{s.t.}\ & F(y,u) = Ku - y = 0\end{aligned} \tag{1}$$

where K is the stiffness matrix, taking into account of the boundary conditions, u, $\bar u$, and y are vectors.

In what follows, we basically apply common optimization method to the constrained optimization problem Equation 1.

Method 1: Penalty Method

The simplest method for solving the unconstrained optimization problem Equation 1 is via the penalty method. Specifically, instead of solving the original constrained optimization problem, we solve

$$\min_{y,u} \frac{1}{2}\|u-\bar u\|_2 + \frac{\rho}{2}\|y\|^2 + \lambda\|f(y,u)\|_2^2$$

where λ is the penalty parameter.

The penalty method is **conceptually simple** and is also **easy-to-implement**. Additionally, it does not require solving the PDE constraint $f(y,u)=0$ and thus the comptuational cost for each iteration can be small. However, avoid solving the PDE constraint is also a disadvantage since it means the penalty method **does not eventually enforce the physical constraint**. The solution from the penalty method only converges to the the true solution when $\lambda\to\infty$, which is **not computationally feasible**. Additionally, despite less cost per iteration, the total number of iterations can be huge when the PDE constraint is "stiff". To gain some intuition, consider the following problem

$$\begin{aligned}\min_u\ & 0 \\ \text{s.t.}\ & Ku - y = 0\end{aligned}$$

The optimal value is $u = K^{-1}y$ and the cost by solving the linear system is usually propertional to $\mathrm{cond}(K)$. However, if we were to solve the problem using the penalty method

$$\min_u \|Ku-y\|_2^2$$

The condition number for solving the least square problem (e.g., by solving the normal equation $K^TKu = K^Ty$) is usually propertional to $\mathrm{cond}(K)^2$. When the problem is stiff, i.e., $\mathrm{cond}(K)$ is large, the penaty formulation can be much more **ill-conditioned** than the original problem.

Method 2: Primal and Primal-Dual Method

A classical theory regarding constrained optimization is the Karush-Kuhn-Tucker (KKT) Theorm. It states a necessary and sufficient condition for a value to be optimal under certain assumptions. To formulate the KKT condition, consider the **Lagrangian function**

$$L(u, y, \lambda) = \frac{1}{2}\|u - \bar u\|_2 + \frac{\rho}{2}\|y\|^2 + \lambda^T(Ku - y)$$

where λ is the **adjoint variable**. The corresponding KKT condition is

$$\begin{aligned}\frac{\partial L}{\partial u} &= u - \bar u + K^T\lambda = 0\\ \frac{\partial L}{\partial y} &= \rho y - \lambda = 0\\ \frac{\partial L}{\partial \lambda} &= Ku - y = 0\end{aligned}$$

The **primal-dual method** solves for (u, y, λ) simultaneously from the linear system

$$\begin{bmatrix} 1 & 0 & K^T\\ 0 & \rho & -1\\ K & -1 & 0\end{bmatrix}\begin{bmatrix} u\\ y\\ \lambda\end{bmatrix} = \begin{bmatrix}\bar u\\ 0\\ 0\end{bmatrix}$$

In contrast, the **dual method** eliminates the state variables u and y and solves for λ

$$(1 + \rho KK^T)\lambda = \rho K\bar u$$

The primal-dual and dual method have some advantages. For example, we reduce finding the minimum of a constrained optimization method to solving a nonlinear equation, where certain tools are available. For the primal-dual method, the limitation is that the system of equations can be very large and difficult to solve. The dual method requires analytical derivation and is not always obvious in practice, especially for nonlinear problems.

Method 3: Primal Method

The primal method reduces the constrained optimization problem to an unconstrained optimization problem by "solving" the numerical PDE first. In the previous example, we have $u(y) = K^{-1}y$ and therefore we have

$$\min_y \frac{1}{2}\|K^{-1}y - \bar u\|_2 + \frac{\rho}{2}\|y\|^2$$

The advantage is three-folds

- Dimension reduction. The optimization variables are reduced from (u, y) to the design variables y only.
- Enforced physical constraints. The physical constraints are enforced numerically.
- Unconstrained optimization. The reduced problem is a constrained optimization problem and many off-the-shelf optimizers (gradient descent, BFGS, CG, etc.) are available.

However, to compute the gradients, the primal method requires deep insights into the numerical solver, which may be highly nonlinear and implicit. The usual automatic differentiation (AD) framework are in general not applicable to this type of operators (nodes in the computational graph) and we need special algorithms.

Link to Adjoint-State Method

The adjoint state method is a standard method for computing the gradients of the objective function with respect to design variables in PDE-constrained optimization. Consider

$$\begin{aligned}\min_{y,u}\ & J(y,u)\\ \text{s.t. }\ & F(y,u) = 0\end{aligned}$$

Let u be the state variable and y be the design variable. Assume we solve for $u = u(y)$ from the PDE constraint, then we have the reduced objective function

$$\hat J(y) = J(y, u(y))$$

To conduct gradient-based optimization, we need to compute the gradient

$$\frac{d\hat J(y)}{dy} = \nabla_y J(y, u(y)) + \nabla_u J(y, u(y))\frac{du(y)}{dy} \tag{2}$$

To compute the $\frac{du(y)}{dy}$ we have

$$F(y,u(y))=0 \Rightarrow \nabla_y F + \nabla_u F \frac{du}{dy} = 0 \tag{3}$$

Therefore we can use Equations 2 and 3 to evaluate the gradient of the objective function

$$\frac{d\hat{J}(y)}{dy} = \nabla_y J(y,u(y)) - \nabla_u J(y,u(y))(\nabla_u F(y,u(y)))^{-1}\nabla_y F(y,u(y)) \tag{4}$$

Link to the Lagrange Function

There is a nice interpretation of Equations 2 and 3 using the Lagrange function. Consider the langrange function

$$L(y,u,\lambda) = J(y,u) + \lambda^T F(y,u) \tag{5}$$

The KKT condition says

$$\begin{aligned}\frac{\partial L}{\partial \lambda} &= F(y,u) = 0\\ \frac{\partial L}{\partial u} &= \nabla_u J + \lambda^T \nabla_u F(y,u) = 0\\ \frac{\partial L}{\partial y} &= \nabla_y J + \lambda^T \nabla_y F(y,u) = 0\end{aligned}$$

Now we relax the third equation: given a fixed y (not necessarily optimal), we can solve for (u,λ) from the first two equations. We plug the solutions into the third equation and obtain (note u satisfies $F(y,u)=0$)

$$\frac{\partial L}{\partial y} = \nabla_y J(y,u) - \nabla_u J(y,u)(\nabla_u F(y,u))^{-1}\nabla_y F(y,u)$$

which is the same expression as Equation 4. When y is optimal, this expression is equal to zero, i.e., all the KKT conditions are satisfied. The gradient of the unconstrained optimization problem is also zero. Both the primal system and the primal-dual system confirm the optimality. This relation also explains why we call the method above as "adjoint" state method. In summary, the adjoint-state method involves a three-step process

- **Step 1.** Create the Lagrangian Equation 5.
- **Step 2**. Conduct forward computation and solve for u from

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

- **Step 3.** Compute the adjoint variable λ from

$$\nabla_u J + \lambda^T \nabla_u F(y,u) = 0$$

- **Step 4.** Compute the sensitivity

$$\frac{\partial \hat{J}}{\partial y} = \nabla_y J + \lambda^T \nabla_y F(y,u)$$

Link to Automatic Differentiation

Computational Graph

The adjoint-state method is also closely related to the reverse-mode automatic differentiation. Consider a concrete PDE-constrained optimization problem

$$\begin{aligned}\min_{\mathbf{u}_1,\theta} J &= f_4(\mathbf{u}_1,\mathbf{u}_2,\mathbf{u}_3,\mathbf{u}_4),\\ \text{s.t. } \mathbf{u}_2 &= f_1(\mathbf{u}_1,\theta),\\ \mathbf{u}_3 &= f_2(\mathbf{u}_2,\theta),\\ \mathbf{u}_4 &= f_3(\mathbf{u}_3,\theta).\end{aligned}$$

where f_1, f_2, f_3 are PDE constraints, f_4 is the loss function, $\mathbf{u}_1$ is the initial condition, and θ is the model parameter.

Adjoint-State Method

The Lagrangian function is

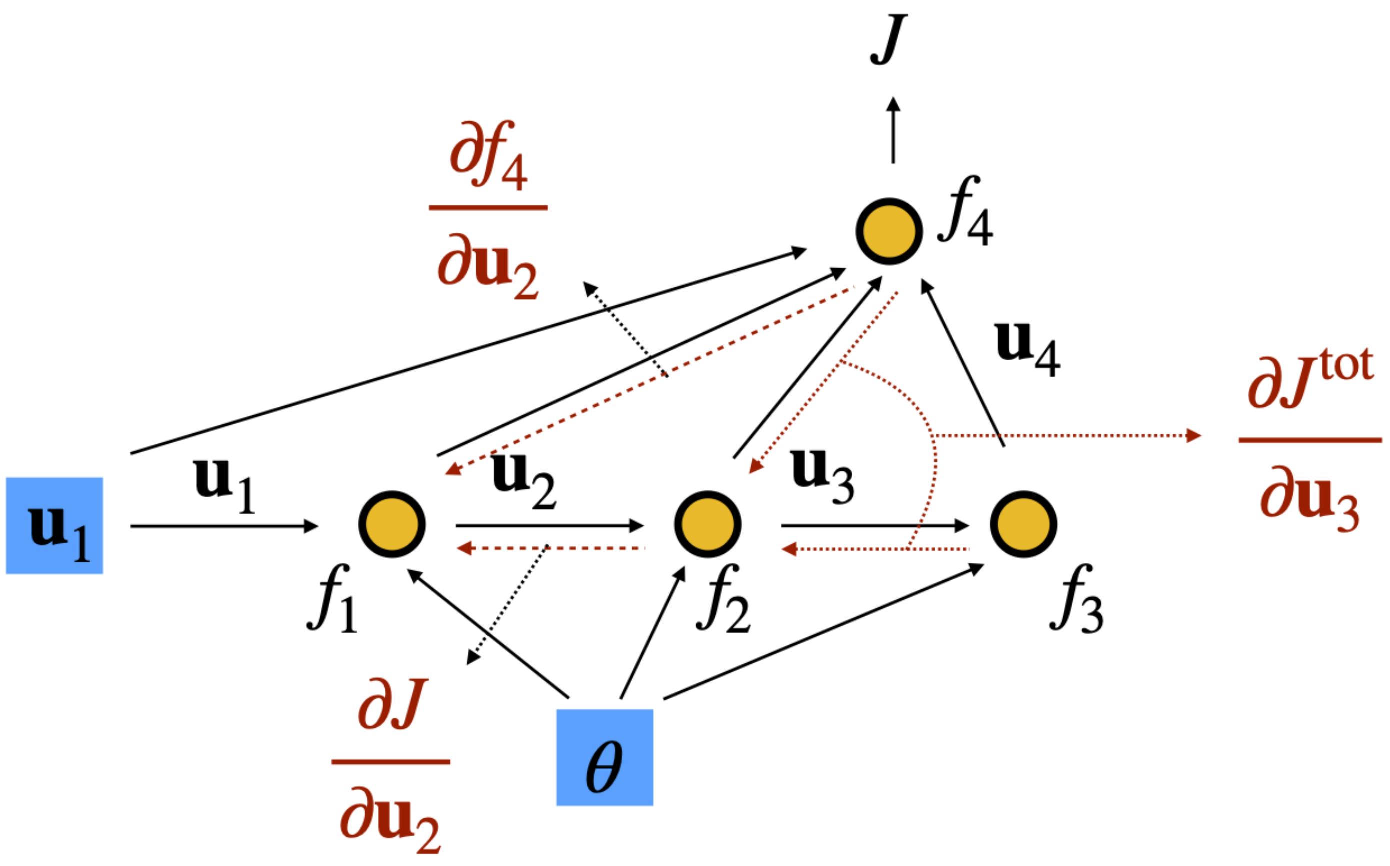
$$\mathcal{L} = f_4(\mathbf{u}_1,\mathbf{u}_2,\mathbf{u}_3,\mathbf{u}_4) + \lambda_2^T(f_1(\mathbf{u}_1,\theta)-\mathbf{u}_2) + \lambda_3^T(f_2(\mathbf{u}_2,\theta)-\mathbf{u}_3) + \lambda_4^T(f_3(\mathbf{u}_3,\theta)-\mathbf{u}_4$$

Upon conducting the foward computation we have all $\mathbf{u}_i$ available. To compute the adjoint variable λ_i, we have

$$\begin{aligned}\lambda_4^T &= \frac{\partial f_4}{\partial \mathbf{u}_4}\\ \lambda_3^T &= \frac{\partial f_4}{\partial \mathbf{u}_3} + \lambda_4^T\frac{\partial f_3}{\partial \mathbf{u}_3}\\ \lambda_2^T &= \frac{\partial f_4}{\partial \mathbf{u}_2} + \lambda_3^T\frac{\partial f_2}{\partial \mathbf{u}_2}\end{aligned}$$

The gradient of the objective function in the constrained optimization problem is given by

$$\frac{\partial \mathcal{L}}{\partial \theta} = \lambda_2^T \frac{\partial f_1}{\partial \theta} + \lambda_3^T \frac{\partial f_2}{\partial \theta} + \lambda_4^T \frac{\partial f_3}{\partial \theta}$$



J
f_4
$\frac{\partial f_4}{\partial \mathbf{u}_2}$
$\mathbf{u}_4$
$\frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_3}$
$\mathbf{u}_1$
$\mathbf{u}_1$
$\mathbf{u}_2$
$\mathbf{u}_3$
f_1
f_2
f_3
$\frac{\partial J}{\partial \mathbf{u}_2}$
θ

Automatic Differentiation

Now let's see how the computation is linked to automatic differentiation. As explained in the previous tutorials, when we implement the automatic differentiation operator, we need to backpropagate the "top" gradients to its upstreams in the computational graph. Consider the operator f_2, we need to implement two operators

$$\begin{aligned}\text{Forward: } & \mathbf{u}_3 = f_2(\mathbf{u}_2, \theta)\\ \text{Backward: } & \frac{\partial J}{\partial \mathbf{u}_2}, \frac{\partial J}{\partial \theta} = b_2\left(\frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_3}, \mathbf{u}_2, \theta\right)\end{aligned}$$

Here $\frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_3}$ is the "total" gradient $\mathbf{u}_3$ received from the downstream in the computational graph.

Relation between AD and Adjoint-State Method

The backward operator is implemented using the chain rule

$$\frac{\partial J}{\partial \mathbf{u}_2} = \frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_3}\frac{\partial f_2}{\partial \mathbf{u}_2} \qquad \frac{\partial J}{\partial \theta} = \frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_3}\frac{\partial f_2}{\partial \theta}$$

The total gradient $\mathbf{u}_2$ received is

$$\frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_2} = \frac{\partial f_4}{\partial \mathbf{u}_2} + \frac{\partial J}{\partial \mathbf{u}_2} = \frac{\partial f_4}{\partial \mathbf{u}_2} + \frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_3}\frac{\partial f_2}{\partial \mathbf{u}_2}$$

The dual constraint in the KKT condition $\lambda_2^T = \frac{\partial f_4}{\partial \mathbf{u}_2} + \lambda_3^T \frac{\partial f_2}{\partial \mathbf{u}_2}$

Now we see the important relation

$$\boxed{\lambda_i^T = \frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_i}}$$

That means, in general, **the reverse-mode AD is back-propagating the Lagrange multiplier (adjoint variables)**.

Dicussion and Physics based Machine Learning

However, although the link between AD and adjoint state methods enables us to use AD tools for PDE-constrained optimization, many standard numerical schemes, such as implicit ones, involve an iterative process (e.g., Newton-Raphson) in nature. AD is usually designed for explicit operators. To this end, we can borrow the idea from the adjoint-state methods and enhance the current AD framework to differentiate through iterative solvers or implicit schemes. This is known as **physics constrained learning**. For more details, see the paper here.

Another ongoing research is the combination of neural networks and physical modeling. One idea is to model the unknown relations in the physical system using neural networks. Those includes

- Koopman operator in dynamical systems
- Constitutive relations in solid mechanics.
- Turbulent closure relations in fluid mechanics.
- ......

In the context of PDE-constrained optimization, there is no essential difference between learning a neural network and finding the optimal physical parameters, except that the design variables become the weights and biases of neural networks. However, the neural networks raise some questions, for example: does one optimization technique preferable than the others? How to stabilize the numerical solvers when neural networks are present? How to add physical constraints to the neural network? How to scale the algorithm? How is the well-posedness and conditioning of the optimization problem? How much data do we need? How to stabilize the training (e.g., regularization, projected gradients)? Indeed, the application of neural networks in the physics machine learning leave more problems than what have been answered here.

Other Optimization Techniques

Besides the formulation and optimization techniques introduced here, there are many other topics, which we have not covered here, on PDE-constrained optimization. It is worthwhile mentioning that we consider the optimize-then-discretize approach. The alternative approach, discretize-then-optimize, derives the optimal condition (KKT condition) on the continuous level and then discretize the dual PDE. In this formulation, we can use the same discretization method for both the primal and dual system, and therefore we may preserve some essential physical properties. However, the gradients derived in this way may deviate from the true gradients of the constrained optimization problem.

Another noteworthy ongoing research is to formulate the optimization as an action functional. Just like every PDE can be viewed as a minimization of an energy function, a PDE-constrained optimization problem can also be formulated as a problem of minimizing a functional. These discussions are beyond the scope of the tutorial.

Summary

PDE-constrained optimization has a wide variety of applications. Specifically, formulating the physics based machine learning as a PDE-constrained optimization problem lends us a rich toolbox for optimization, discretization, and algorithm design. The combination of neural networks and physics modeling poses a lot of opportunities as well as challenges for solving long standing problems. The gradient based optimization with automatic differentiation has the potential to consolidate the techniques in a single framework.