TensorFlow Meets PyTorch: Using PyTorch to Create TensorFlow Custom Operators

Why do we need `PyTorch`?

In another tutorial, we talked about four types of forward simulation operators. There is no general way to code nonlinear implicit operators in `TensorFlow` and therefore we resort to custom operators. However, this brings another problem: we need to implement the backward operator, i.e.,

$$\frac{\partial J}{\partial x} = -\frac{\partial J}{\partial y} F_y^{-1} F_x$$

This requires us to solve a linear system where the coefficient matrix is the Jacobian F_y. Also, we need to compute the matrix vector production where the matrix is the Jacobian F_x. The problem also arises when one tries to implement custom operators for explicit nonlinear operator, where the backward operator is

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial y} F_x(x)$$

Step-by-Step Instruction

In this tutorial, we show how to implement the backward operator without deriving the Jacobian by leveraging the `PyTorch` `Aten` library. Again, for concreteness, we assume that the nonlinear implicit operator is

$$F(x, y) = x^2 - y^3/(1+x)$$

The operator can be written in the explicit form for verification

$$y = (x^2(1+x))^{\frac{1}{3}}$$

- **Step 1: Computing** $g = \frac{\partial J}{\partial y} F_y^{-1}$

This step requires solving a linear system

$$F_y u = \frac{\partial J}{\partial y}$$

To solve the linear system, we can apply an iterative solver such as GMRES. This requires us to be able to do matrix vector production. This can be cleverly done in `PyTorch` by letting y be a `Variable`, x, u be non-trainable tensors, and differentiate $F(x, y) \cdot u$ with respect to y, and we obtained $F_y u$ directly.

- **Step 2: Computing gF_x**

Similar to Step 1, this can be done by treating x as `Variable` while y, u as non-trainable tensors. Differentiation with respect to x gives the desired results.

For a full working script, see here.