Solving Neural-network simulation systems in CRIKit with Pyadjoint

Grant Bruer, Tobin Isaac

CRIKit

The Constitutive Relations Inference Toolkit is a software framework that aims to simplify the use of novel constitutive relations (CRs) in physical systems described by partial differential equations (PDEs). It consists of four principal components:

CR: Contains the parameters for the constitutive relation.

Observer: Reduces the full system state to a set of observations.

Experiment: The experiment uses a given CR and Observer to run the forward problem and get observations. It expresses the PDE system using FEniCS or Firedrake and solves it using our Reduced System Solver.

Loss: Compares simulated observations from an Experiment to measured observations from the true system. By using the automatic differentiation tool Pyadjoint, we can take the gradient of the loss wrt the CR parameters in order to train the CR.

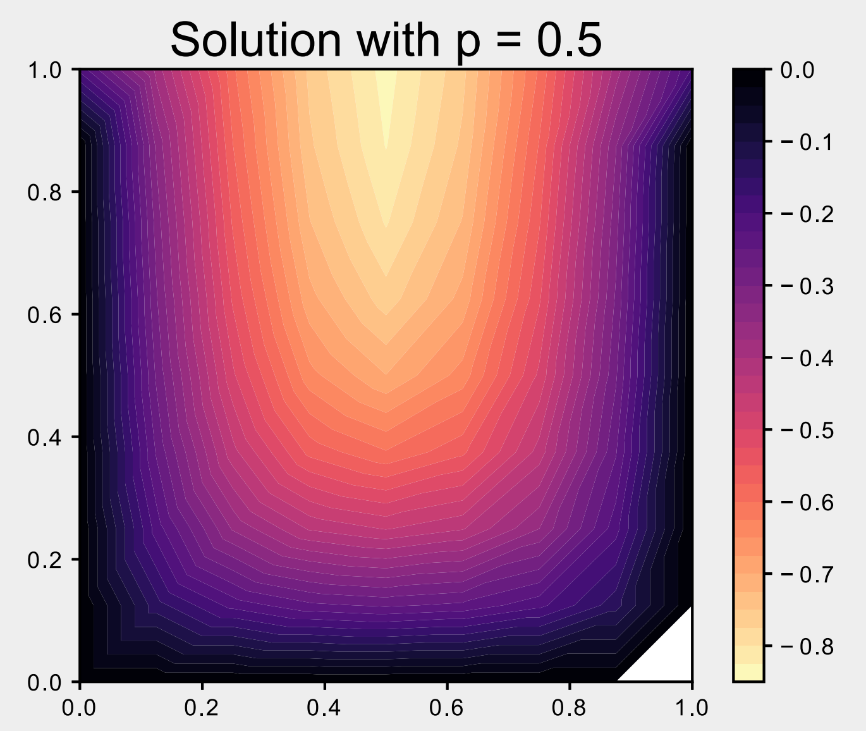
Test Problem

The *p*-Laplacian generalizes the Laplacian to a non-linear regime. Example applications include fluid flow and elastic deformation.

$$\nabla \cdot (|\nabla u|^p \nabla u) = -f$$

This can be generalized by letting σ be a parameterized CR with the corresponding PDE below. We test on a 2D mesh with homogeneous Neumann conditions on the top boundary and Dirichlet conditions on the other boundaries.

$$\sigma = \sigma(u, \nabla u; p)$$
$$\nabla \cdot \sigma = -f$$



Example CRs

Network CR

The network CR feeds the inputs through a neural network. We use a simple deep network implementation consisting of an input layer, a sequence of hidden layers, and an output layer. The layers are densely connected; i.e., each neuron in a layer is connected to every neuron in the next layer. The hidden layers have sigmoid activation and the output layer has linear activation. The number of hidden layers and the size of each layer can be arbitrarily chosen. In our tests, we use a network with three hidden layers of sizes 8, 6, and 5.

Plap CR

The plap CR directly calculates the *p*-Laplacian at each quadrature point. It's only parameter is *p*.

CR Tests

Training

Using Pyadjoint, we optimize the CR parameters to minimize the loss. We used two versions of the loss:

1. J_{full}: *u* is observed at every grid point.
2. J_{top}: *u* is observed along top surface.

$$J = \int (u_{obs} - u(\sigma))^2 dx$$

Results

Figure 1 shows the result of training. Both CRs learned to match the observations. However, while the plap CR is able to learn the correct *p* exactly, the network CR only matches the contours in the area of negative y-gradient.

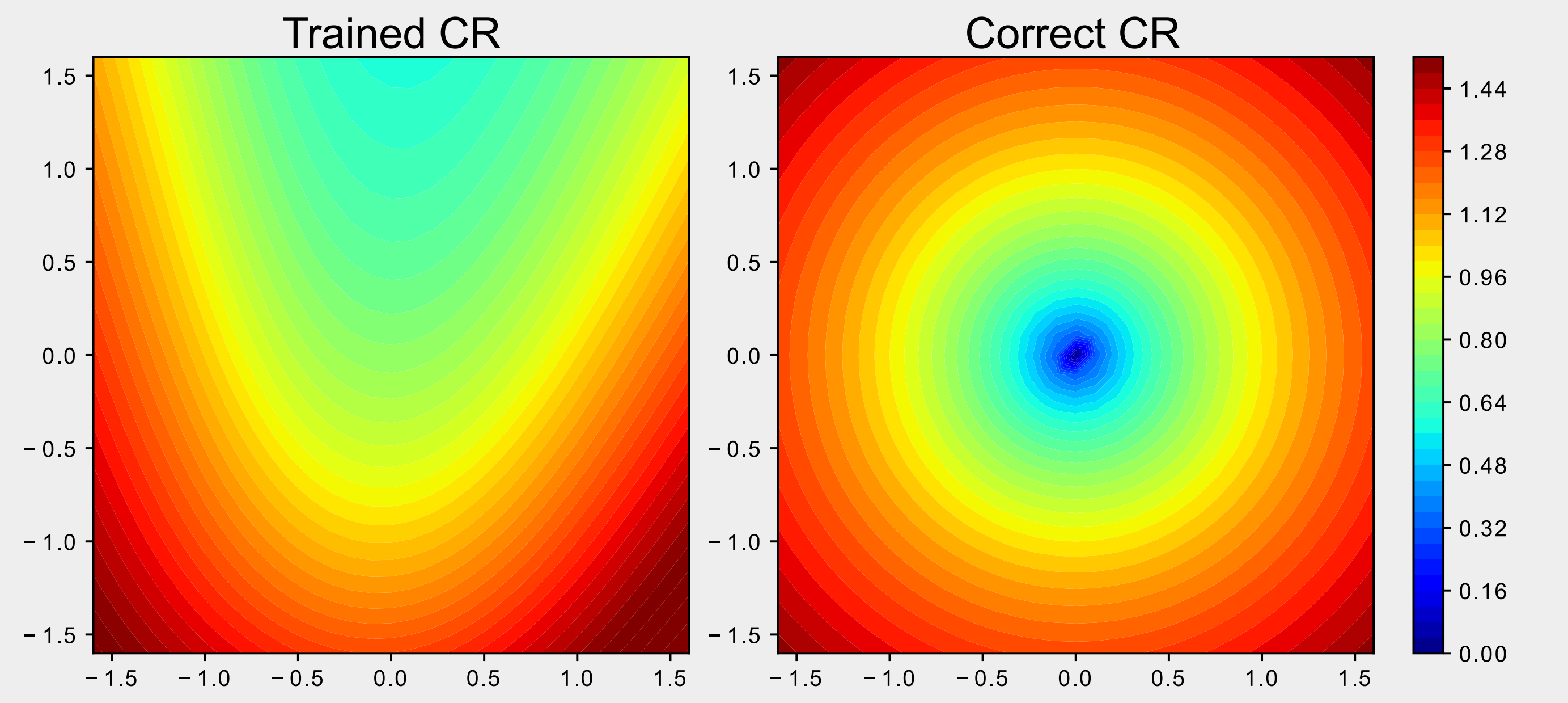


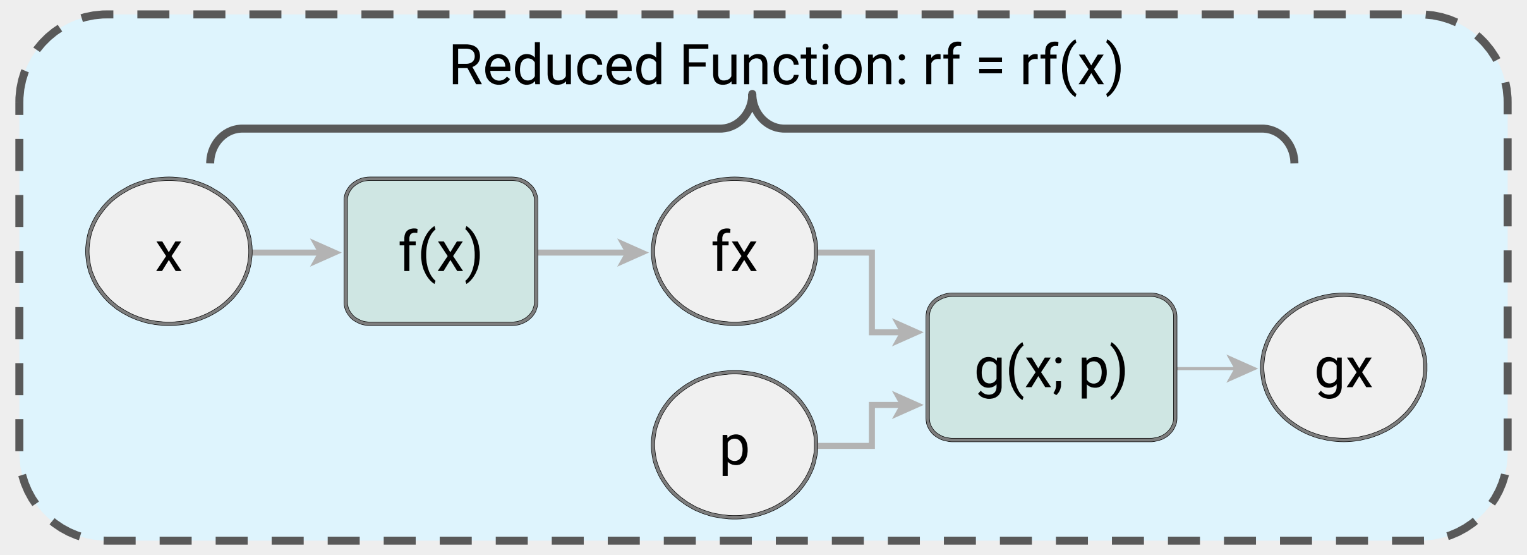
Figure 1: the network CR (left) matches the correct CR (right) in the area constrained by observations.

Pyadjoint Usage

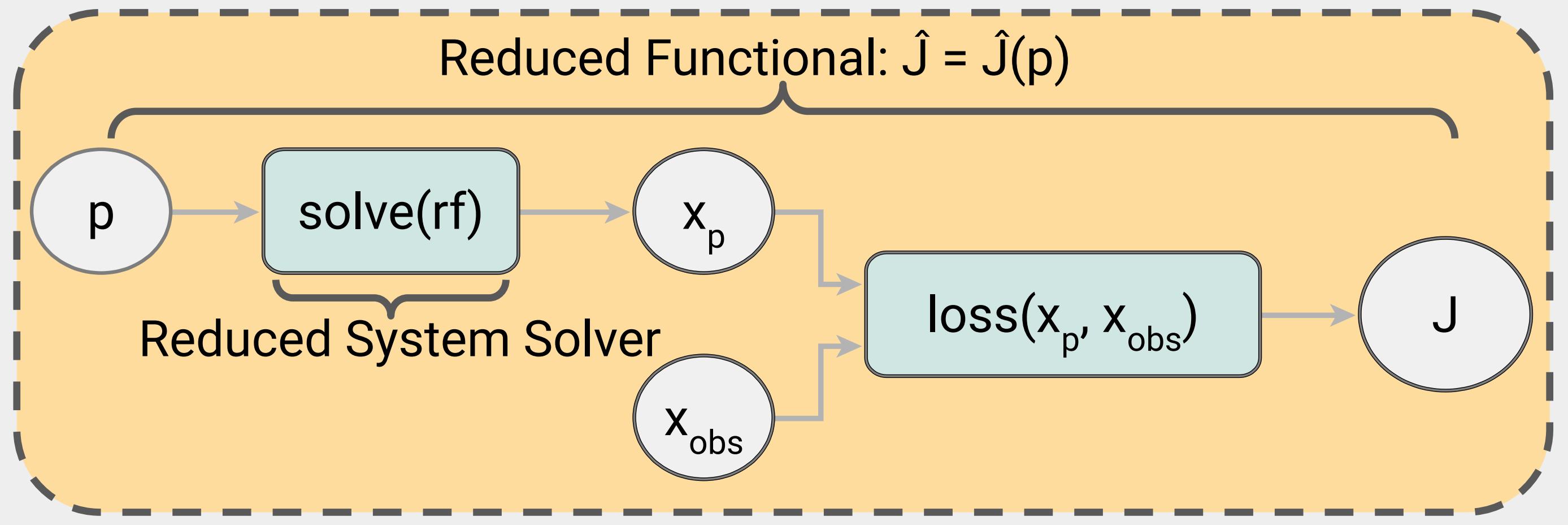
The Pyadjoint library overloads FEniCS and Firedrake functions in order to form a computation graph with blocks that each know how to redo their computation and how to compute their gradient and Hessian action. From this graph, derivatives of functionals can be calculated simply using the chain rule.

We worked on two new components to Pyadjoint:

Reduced Function: The reduced function records the calculation of the CR so that it can be recomputed with different inputs and so its Jacobian and adjoint can be calculated.



Reduced System Solver: The solver uses a Reduced Function to calculate the residual and Jacobian and uses SNES to drive the residual to zero. The solver also handles solving the adjoint system used to get the gradient $\frac{dJ}{dp}$. This takes the solution out of the control of the finite element framework, allowing more general systems to be solved.



Example Code

```
# 1: Create the reduced function for the residual.
with push_tape():
    fx = f(x)
    gx = g(fx, p)

    rf = ReducedFunction(gx, Control(x))

# 2: Get solution x_sol such that rf(x_sol) == 0.
solver = SNESSolver(rf, **solver_args)
x_sol = solver.solve(Control(x))

# 3: Get the observational loss.
J = loss(x_sol, x_obs)

# 4: Find the optimal p that minimizes the loss.
Jhat = ReducedFunctional(J, Control(p))
p_opt = minimize(Jhat)
```

Conclusion

We built a software framework to train CRs to match observational data. This work allows scientists to build complicated PDEs describing a system by using neural networks, FEniCS or Firedrake, and lots of observations. In future work, we will build known invariances into the system so that fewer observations will be required and symmetries will be exactly conserved. We also plan to integrate with common neural network software such as Pytorch and Keras, and optimizers such as Tao.

Georgia Tech | School of Computational Science and Engineering

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1835792. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.