Inverse Problems and Model Reduction

A brief summary of Learning physics-based models from data: perspectives from inverse problems and model reduction

What is an inverse problem?

We're trying to infer the parameters of some PDE system from observations of its outcomes. Formally we can let the paramerters be a vector $\mathbf{m} \in \mathbb{R}^p$ and the data be a vector $\mathbf{d} \in \mathbb{R}^n$. Then we're trying to solve the following optimization problem or a similar one

$$\arg\min_{\boldsymbol{m}} \Phi(\boldsymbol{m}) = \frac{1}{2}||\boldsymbol{f}(\boldsymbol{m}) - \boldsymbol{d}||^2 + \frac{\beta}{2}||\boldsymbol{m}||^2_{\mathbf{H}^{\text{reg}}} \tag{1}$$

Where $\boldsymbol{f}(\boldsymbol{m})$ is the forward solution to the PDE with parameter $\boldsymbol{m}$, $\boldsymbol{d}$ is the observed outcome, and $\mathbf{H}^{\text{reg}}$ is a regularization norm that prescribes some nice properties of $\boldsymbol{m}$, maybe smoothness, maybe something else desireable. The first term is just how well the forward solution with the parameter we found matches the data, and the second term is a regularization penalty.

Generally inverse problems are ill-posed because there are often many different parameters that will give the same output to the PDE. An intuitive example is 1D heat distribution. If our data is a flat heat profile, there are infinitely many initial heat distributions that will result in a flat heat distribution eventually.
Regularization lets us refine to try to pick which of those solutions we want our solver to prefer.

Solving with Newton-CG

We start with a guess $\boldsymbol{m}_0$. Ultimately we want to make the following update to $\boldsymbol{m_0}$, so that our new $\boldsymbol{m}$ reduces $\boldsymbol{\Phi(m)}$.

$$\boldsymbol{m}_{k+1} = \boldsymbol{m}_k + \alpha_k \boldsymbol{p}_k \tag{2}$$

But we first need to find the right search direction $\boldsymbol{p}_k$ that will affect $\boldsymbol{m}_k$ in the right way. We do this by solving the system [1]

$$\mathbf{H}_k \boldsymbol{p}_k = -\boldsymbol{g}_k \tag{3}$$

[1] Technically we want to solve this system inaccurately to not overfit and there are a few other details I'm skipping in this summary, but this is the gist.

Where $\boldsymbol{g}_k$ is the gradient of $\Phi(\boldsymbol{m}_k)$ and $\mathbf{H}_k$ is the Hessian of $\Phi(\boldsymbol{m}_k)$. This is difficult because $\mathbf{H}_k$ is very large and dense, and for a lot of problems it is completely intractable to formulate it explicitly. Instead we use the conjugate gradient method to get away with only needing to compute the Hessians effect on an arbitrary vector, which we can compute with a forward/adjoint pair solve, rather than the full Hessian.

What is the expensive part?

The main expensive parts of this are solving the $\mathbf{H}_k \boldsymbol{p}_k = -\boldsymbol{g}_k$ system with CG, which requires multiple Hessian-vector products, and performing the linetrace to find an appropriate step size α_k. Each Hessian-vector product and each evaluation of $\Phi(\boldsymbol{m})$ during the linetrace requires solving the forward PDE.

Table 6.1. Performance and scalability of inexact Newton-CG for Pine Island Glacier inverse problem. The columns report the number of state variable unknowns (#sdof), basal friction parameter unknowns (#pdof), Newton iterations (#N), total and average (per Newton iteration) number of CG iterations (#CG, avgCG) and total number of linear(ized) Stokes solves (from forward, adjoint and incremental forward and adjoint problems) (#Stokes). The iterations are terminated when the norm of the gradient is decreased by a factor of 10^5. The superlinear choice of the forcing term is made, *i.e.* $\eta_k = \|\mathbf{g}_k\|^{0.5}$.

#sdof	#pdof	#N	#CG	avgCG	#Stokes
95 796	10 371	42	2718	65	7031
233 834	25 295	39	2342	60	6440
848 850	91 787	39	2577	66	6856
3 372 707	364 649	39	2211	57	6193
22 570 303	1 456 225	40	1923	48	5376

Above is one of the sample problems given in the paper. The "#Stokes" column is the number of forward solves required to solve the inverse problem for the different scales of this problem they tried. I don't yet have a sense of the scale of the problems we're trying to solve, but needing on the order of thousands of forward solves seems like a reasonable assumption of this method.

Deciding which inverse problems we're interested in.

The "generic" inverse problem

Given

$$c_i(\boldsymbol{x}, T_f), \phi(\boldsymbol{x}, T_f) \tag{4}$$

infer

$$c_i(\boldsymbol{x}, 0), \phi(\boldsymbol{x}, 0), D_i, \eta, ... \tag{5}$$

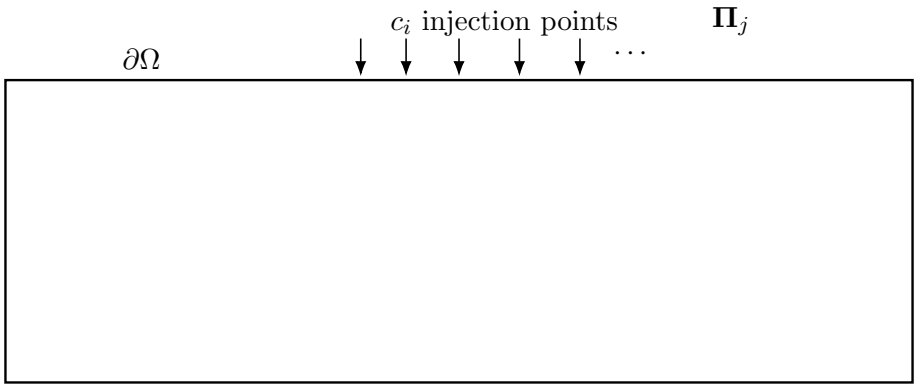
This is probably very hard and (almost) definitely very ill-conditioned. Could be possible, I think I could try to begin to explore a version of this using a simpler poisson forward solver, maybe the partial one in the github.

I'd like to know a bit more about the applications. If the goal is to create a desired $c_i(\boldsymbol{x}, T_f), \phi(\boldsymbol{x}, T_f)$, and we have some mechanisms for controlling the system, then an inverse problem with the control mechanism as the constraints might have a better shot and not being quite so ill-conditioned.
The control mechanism would be application specific, so I'd like to learn more about the applications, but an example one might be

$$c_i(\mathbf{x}, 0) = 0, \qquad \mathbf{x} \in \Omega, \tag{6}$$

$$\left.\frac{\partial c_i}{\partial t}\right|_{\mathbf{x}=\mathbf{\Pi}_j} = f(t), \qquad j = 1, 2, \ldots, \tag{7}$$

$$f_j(t) = \begin{cases} r_j, & t < T_{\text{inj}}^{(j)}, \\ 0, & \text{otherwise.} \end{cases} \qquad 0 < T_{\text{inj}} \leq T_f \tag{8}$$



This is a setup where you start with initially no c_i and then you inject c_i at a set of points $\mathbf{\Pi}_j$ on the boundary for some time $T_{\text{inj}}^{(j)}$. The inverse problem is then to find the injection rates r_j and injection times $T_{\text{inj}}^{(j)}$ such that at final time T_f the concentration and potential match some desired profile. This could be useful as it might allow you achieve an arbitrary desired concentration profile by controlling the injection rate and time. This would likely be much less ill-conditioned as the control mechanism limits the space of possible solutions.
Depending on the application, a setup like this would definitely make the problem much more tractable.

According to https://pmc.ncbi.nlm.nih.gov/articles/PMC3122111/ which was linked in the github, "there are very limited experimental data of diffusion coefficients available for biomolecular systems, such as ion channels, nanopores, and microchannels." So perhaps creating a good computational setup to infer D_i from experimental data would be a useful inverse problem to solve.

The OED (Optimal Experimental Design) problem

The paper also has a section about Optimal Experiemental design, ie. can we infer how best to collect data? Where should we put the sensors in our experiment etc. They note that "the OED problem is still several orders of magnitude more expensive than the inverse problem, which in turn is several orders of magnitude more expensive than the forward problem," and "OED for large-scale inverse problems governed by PDE forward models with large numbers of design variables remained out of reach. In the last several years, a number of advances have made the solution of such OED problems tractable, at least for simpler PDE systems," this doesn't seem promising for our non-simple PDE forward model.

Non-linear model reduction

This will take a lot more looking into to see if it might be worthwhile to use.
I'm not sure how this would work with a weak formulation based forward method. To my understanding this involves spacially descritizing the PDE, ie going from

$$\frac{\partial u}{\partial t} = \mathcal{F}(u) \tag{9}$$

to

$$\frac{d\boldsymbol{u}}{dt} = \boldsymbol{F}(\boldsymbol{u}) \tag{10}$$

by letting each row of $\boldsymbol{u}$ be the value of u at a spacial gridpoint. We now have a system of ODEs which we can reduce by using a kind of spectral analysis to figure out a reduced order basis for this new system.
This is something I would need to spend a lot more time looking into to see if it would be worthwhile to use. We might lose the advantages of the weak formulation by doing this spacial discretization.

In the immidiate future I plan to explore using firedrake to create a Newton-CG solver that works with a simpler poisson system. I can't seem to find any existing implementations of

Newton-CG with the firedrake framework.