An Introduction to Gradient Computation by the Discrete Adjoint Method

Austen C. Duffy

A Technical Report

Florida State University, Summer 2009

Abstract

This report is intended to provide guidance in the technical aspects of computing the gradient to a cost function using the discrete adjoint method. Adjoint methods are widely used in many areas including optimal control theory, design optimization, variational data assimilation and sensitivity analysis. While numerous works in these areas discuss the methodology behind adjoint based gradient computation, details of this topic are frequently ommited leaving the actual implementation of the method as a bit of a mystery to those new to the subject. In this report we will give detailed explanations of how to compute the gradient of a cost function subject to PDE constraints by way of the discrete adjoint method. This is accomplished by providing simple examples from data assimilation to illustrate its use. In addition, we provide algorithms for computing the required Jacobian matrices on the fly, eliminating the need to store them as well as the need to explicitly derive the discrete adjoint problem.

Chapter 1

The Adjoint Method

1.1 Introduction

Adjoint methods are popular in many fields including shape optimization [7, 9, 13], optimal control theory [8, 11], uncertainty or sensitivity analysis [1], and data assimilation [3–6]. In gradient based optimization, adjoint methods are widely used for the gradient computation when the problem at hand posseses a large number of design variables. These methods can be broken down into two categories, continuous and discrete. In the continuous version, the adjoint equations are found analyticaly from the governing equation, and are then discretized. This is in contrast to the discrete version where the governing equation is first discretized, and the discrete adjoint equations immediately follow. There have been numerous studies comparing the two versions (e.g. [15]) with none providing a clear winner, in essence leaving the choice as a user preference.

In [15] an extensive study of discrete and continous adjoint methods is reported. Some findings there include that the continuous adjoint equations are not unique and in fact they can vary due to choices in the derivation which could affect the optimization procedure, that the continuous adjoint may destroy symmetry properties for optimality conditions required in some optimization procedures, and that continuous adjoints may have degraded convergence properties over the discrete method, or may not converge at all. They also found that continuous adjoints can provide more flexibility and note that the solution provided by the discrete version is no more accurate than the discretization used to solve the state equation. The discrete adjoint will be used here due to the more systematic approach that can be taken to problem solving, however, a brief description of the continuous method is provided below.

1.2 Continuous Adjoint Method

In the continuous adjoint method, a constrained optimization problem is transformed into an unconstrained type by forming a Lagrangian, with the Lagrange multipliers posing as the adjoint variables. The adjoint problem can be determined through either a variational formulation or by finding shape derivatives. In the variational formulation, one could represent the Lagrangian, as done in [14], by a sum of integrals of the general form

$$L = \sum_i \int G_i(u, \phi)$$

and compute the variation in the Lagrangian, L, subject to a variation in the state variable u by

$$\delta L = \sum_i \int \left.\frac{\partial G_i}{\partial \phi}\right|_u \delta\phi + \sum_i \int \left.\frac{\partial G_i}{\partial u}\right|_\phi \delta u$$

The computation of δu is undesireable and so the Lagrange multipliers are defined in order to eliminate the term, i.e. the adjoint problem is determined so that it satisfies

$$\sum_i \int \left.\frac{\partial G_i}{\partial u}\right|_\phi \delta u = 0$$

After solving the adjoint problem, the adjoint solutions are then used to compute the gradient

$$\frac{dL}{d\phi} = \sum_i \int \left.\frac{\partial G_i}{\partial \phi}\right|_u$$

which is used for the optimization. The procedure for finding shape derivatives is similar and is covered later in the discussion of level set based shape and topology optimization.

1.3 Discrete Adjoint Method

In the discrete adjoint method, we essentially solve a set of governing equations forward and then solve the adjoint problem backwards in time in order to acquire the adjoint variables. In the case of linear governing equations, only the final solution data from the forward solve is required, however, in the case of nonlinear governing equations the solution data must be stored at every time

step which can become restrictive for large problems. We believe that this cost is not enough to detract from a method which can be systematically computed due to a lack of need for problem by problem analysis as is required in the continuous case. The treatment of boundary conditions is also an issue with the continuous method, and as [10] notes, it is much easier to obtain these conditions for the adjoint solver in the discrete version. An interesting solution method for the discrete adjoint problem is given in [16] where a Monte Carlo method is used to solve the adjoint problem forward in time for an explicit discretization of a time dependent problem. This allows the adjoint solution to be solved at the same time as the original problem without the need for storing the solution values at each time step.

Consider the general minimization problem

$$\begin{aligned} min \quad & L(u,\phi) = J(u,\phi) \\ subject\ to \quad & N(u,\phi) = 0 \end{aligned}$$

Where J is the cost function, N is the governing equation, u is the state variable and ϕ is the design variable. In order to use a gradient based method (like steepest descent) to perform the minimization, one will need to compute

$$\frac{dL}{d\phi} = \frac{\partial J}{\partial u}\frac{du}{d\phi} + \frac{\partial J}{\partial \phi} \tag{1.1}$$

The $\frac{du}{d\phi}$ term can be determined by looking at the derivative of the governing equation.

$$\frac{dN}{d\phi} = \frac{\partial N}{\partial u}\frac{du}{d\phi} + \frac{\partial N}{\partial \phi} = 0$$

$$\frac{\partial N}{\partial u}\frac{du}{d\phi} = -\frac{\partial N}{\partial \phi}$$

$$\frac{du}{d\phi} = \left(\frac{\partial N}{\partial u}\right)^{-1}\left(-\frac{\partial N}{\partial \phi}\right) \tag{1.2}$$

So from 1.1 and 1.2 we have

$$\frac{dL}{d\phi} = \frac{\partial J}{\partial \phi} - \frac{\partial J}{\partial u}\left(\frac{\partial N}{\partial u}\right)^{-1}\left(\frac{\partial N}{\partial \phi}\right)$$

Now, we will instead solve the adjoint problem by finding λ where

$$\lambda = \left[\frac{\partial J}{\partial u}\left(\frac{\partial N}{\partial u}\right)^{-1}\right]^T$$

and the problem of finding the gradient breaks down to

$$Solve \qquad \left(\frac{\partial N}{\partial u}\right)^T \lambda = \left(\frac{\partial J}{\partial u}\right)^T \tag{1.3}$$

$$Compute \qquad \frac{dL}{d\phi} = \frac{\partial J}{\partial \phi} - \lambda^T \frac{\partial N}{\partial \phi} \tag{1.4}$$

This abstract formulation can make the actual implementation seem like a bit of a mystery, so to clarify things two examples are now given.

1.3.1 Linear Example Problem

Suppose we wish to solve the one dimensional advection equation on $x \in$ [-1,1] where we do not know the initial condition, but do know the solution data at time T. We can recover the initial condition by treating this as a minimization problem.

$$min \quad J(u,\phi) = \int_{-1}^{1} (u(x,T) - u_d(x,T))^2 dx$$

$$\begin{aligned} subject\ to \quad N(u,\phi) &= u_t + a u_x = 0 \\ u(x,0) &= g(x) \\ u(-1,t) &= c \end{aligned}$$

In this case, the design variable ϕ is the initial condition $u(x,0)$, u_d is the observed solution data and c is a constant. We will assume a is positive and use a simple forward in time, backward in space discretization of the governing equation

$$N_i^k = \frac{u_i^k - u_i^{k-1}}{\Delta t} + a\left(\frac{u_i^{k-1} - u_{i-1}^{k-1}}{\Delta x}\right) = 0 \tag{1.5}$$

and approximate J at the final time step m by

$$J = \Delta x \sum_{i=0}^{n} \left(u_i^m - u_{di}\right)^2$$

Now, in terms of the discretization we have

$$N = [N_{-1}^0, N_0^0, ...N_n^0, N_{-1}^1, ...N_n^m]^T$$

$$u = [u_{-1}^0, u_0^0, ...u_n^0, u_{-1}^1, ...u_n^m]^T$$

$$\lambda = [\lambda_{-1}^0, \lambda_0^0, ...\lambda_n^0, \lambda_{-1}^1, ...\lambda_n^m]^T$$

$$G = [G_{-1}^0, G_0^0, ...G_n^0, G_{-1}^1, ...G_n^m]$$

where the -1 subscript represnts the left 'ghost' boundary point needed, and G represents $\frac{\partial J}{\partial u}$. To solve equation 1.3 we find

$$\frac{\partial N}{\partial u} = \begin{pmatrix} \frac{\partial N_{-1}^0}{\partial u_{-1}^0} & \frac{\partial N_{-1}^0}{\partial u_0^0} & .. & \cdots & \frac{\partial N_{-1}^0}{\partial u_n^m} \\ \frac{\partial N_0^0}{\partial u_{-1}^0} & \frac{\partial N_0^0}{\partial u_0^0} & & & . \\ . & & . & & . \\ . & & & . & . \\ \frac{\partial N_n^m}{\partial u_{-1}^0} & . & . & . & \frac{\partial N_n^m}{\partial u_n^m} \end{pmatrix}$$

And using the initial condition $N_i^0 = u_i^0$ along with the embedded boundary condition of

$$N_{-1}^k = u_{-1}^k - c = 0$$

This becomes

$$\frac{\partial N}{\partial u} = \begin{pmatrix} I & 0 & 0 & \cdot & \cdot & 0 \\ A1^1 & A2^1 & 0 & \cdot & \cdot & 0 \\ 0 & A1^2 & A2^2 & & & \cdot \\ \cdot & & \cdot & \cdot & & \cdot \\ \cdot & & & \cdot & \cdot & 0 \\ 0 & \cdot & & & A1^m & A2^m \end{pmatrix}$$

Where $A1^k$ and $A2^k$ are $(n+1) \times (n+1)$ block matrices (with superscripts denoting the time level) given by

$$A1^k = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ -\frac{a}{\Delta x} & -\frac{1}{\Delta t} + \frac{a}{\Delta x} & 0 & & \cdot \\ 0 & -\frac{a}{\Delta x} & -\frac{1}{\Delta t} + \frac{a}{\Delta x} & & \cdot \\ \cdot & \cdot & \cdot & & \\ \cdot & & \cdot & \cdot & \\ \cdot & & & \cdot & \cdot \\ 0 & \cdots & & -\frac{a}{\Delta x} & -\frac{1}{\Delta t} + \frac{a}{\Delta x} \end{pmatrix}$$

$$A2^k = \begin{pmatrix} 1 & \cdots & & 0 \\ 0 & \frac{1}{\Delta t} & & \cdot \\ \cdot & & \frac{1}{\Delta t} & \cdot \\ \cdot & & & \cdot \\ 0 & \cdots & & \frac{1}{\Delta t} \end{pmatrix}$$

For this problem, solving the adjoint turns out to require block solves of sparse structrured matrices allowing simple direct solvers to be used.

Once the adjoint variable has been found, the gradient can be found as noted in the previous section by computing 1.4, and the initial condition is updated according to the minimization method used. It should be noted that this problem,

Algorithm 1 Discrete Adjoint Solution, 1-D Linear Advection

$A2\lambda^m = G^m$
for i=m-1 to 1 **do**
  $A2\lambda^i = G^i - A1^T\lambda^{i+1}$
end for
$\lambda^0 = G^0 - A1^T\lambda^1$

like most inverse problems, is numerically ill-posed. As the effects of dissipation from the discretization become too great (e.g. increasing the spatial step size), much different oscillating solutions are allowed (and hence we have numerical ill-posedness) which can smooth out to match the data at time t when advected forward. This problem can be remedied by the addition of another term to the cost function, for example the term $(\nabla u(x,0))^T(\nabla u(x,0))$ penalizes the cost function when large jumps in the final solution (the recovered IC) are present, thus eliminating the presence of oscillations. A term used by the data assimilation and optimal control communities, known as the background or regularization term respectively, can be given by $(u(x,0) - u_B(x,0))^T(u(x,0) - u_B(x,0))$ where u_B is the 'background', i.e. the previous initial guess. Both of these terms can also benefit from a scaling parameter β.

1.3.2 Nonlinear Example Problem

For the nonlinear example we will solve essentially the same problem, but this time subject to the classical Burgers equation, which possesses many of the features of more complex fluid dynamics models.

$$min \quad J(u,\phi) = \int_{-1}^{1} (u(x,t) - u_d(x,t))^2 dx$$

$$\begin{aligned} subject\ to \quad N(u,\phi) &= u_t + uu_x - \mu u_{xx} = 0 \\ u(x,0) &= g(x) \\ u(-1,t) &= c \\ u_x(1,t) &= 0 \end{aligned}$$

Here, we will use a Lax-Freidrichs discretization for N_i^k

$$\frac{u_i^k - \frac{1}{2}(u_{i+1}^{k-1} + u_{i-1}^{k-1})}{\Delta t} + \frac{(u_{i+1}^{k-1})^2 - (u_{i-1}^{k-1})^2}{4\Delta x} - \mu\left(\frac{u_{i+1}^{k-1} - 2u_i^{k-1} + u_{i-1}^{k-1}}{\Delta x^2}\right) = 0 \quad (1.6)$$

The boundary conditions are discretized as

$$N_{-1}^k = u_{-1}^k - c = 0$$
$$N_{n+1}^k = \frac{u_{n+1}^k - u_n^k}{\Delta x} = 0$$

For this example, the observational data is produced by solving the problem forward using the above discretization along with $c = 0.5$ and $g(x) = 0.5 + 0.5e^{-10 \cdot x^2}$ and so there is no observational error present, i.e. observational data is exact since it is generated from the same discretization used in our model with the known solution (the IC $g(x)$) we are testing against. For the solution of the inverse problem, poor initial data is chosen so as to demonstrate the ability of the adjoint method, and so we take $g(x) = 0.5$.

Chapter 2

A Matrix Free Representation of the Discrete Adjoint Problem

From the previous example, one could imagine that as the number of dimensions increase and the governing equations (along with their discretizations) become more sophisticated, the resulting derivative matrix becomes very large, and (despite its sparseness) storage becomes very undesireable. In the case of realistic computational fluid dynamics problems, the use of traditional sparse solvers may even be out of the question. A few methods have been proposed to combat this issue, for example [16] uses a Monte Carlo method to compute the adjoint, as previously mentioned, without storing the Jacobian or the previous solution states, but here we will look to take advantage of the inherent structure of the Jacobian. For this reason, we want to seek a matrix free representation that will allow us to compute the gradient. It can easily be seen that the matrix $\frac{\partial N}{\partial u}$ will always take on a desireable block lower triangular shape, and in fact it can always (at worst) take the form

$$\frac{\partial N}{\partial u} = \begin{pmatrix} I & 0 & 0 & \cdot & \cdot & 0 \\ J_0^1 & J_1^1 & 0 & \cdot & \cdot & 0 \\ J_0^2 & J_1^2 & J_2^2 & & & \cdot \\ \cdot & & \cdot & \cdot & & \cdot \\ \cdot & & & \cdot & \cdot & 0 \\ J_0^m & \cdot & \cdot & & J_{m-1}^m & J_m^m \end{pmatrix}$$

Where $J_i^k = \frac{\partial N^k}{\partial u^i}$. It is important to note that this notation extends to all dimensions, to be specific, in 1-D

$$J_i^k = \begin{pmatrix} \frac{\partial N_{-1}^k}{\partial u_{-1}^i} & \frac{\partial N_{-1}^k}{\partial u_0^i} & .. & \dots & . & \frac{\partial N_{-1}^k}{\partial u_n^i} \\ \frac{\partial N_0^k}{\partial u_{-1}^i} & \frac{\partial N_0^k}{\partial u_0^i} & & & . & \\ . & & . & & & . \\ . & & & \frac{\partial N_j^k}{\partial u_j^i} & & \\ . & & & & . & . \\ \frac{\partial N_n^k}{\partial u_{-1}^i} & . & . & . & . & \frac{\partial N_n^k}{\partial u_n^i} \end{pmatrix}$$

Then in 2-D, each entry of the sub-Jacobian would become a sub-sub-Jacobian matrix

$$\frac{\partial N_j^k}{\partial u_j^i} = \begin{pmatrix} \frac{\partial N_{j,-1}^k}{\partial u_{j,-1}^i} & \frac{\partial N_{j,-1}^k}{\partial u_{j,0}^i} & .. & \dots & . & \frac{\partial N_{j,-1}^k}{\partial u_{j,n}^i} \\ \frac{\partial N_{j,0}^k}{\partial u_{j,-1}^i} & \frac{\partial N_{j,0}^k}{\partial u_{j,0}^i} & & & . & \\ . & & . & & & . \\ . & & & \frac{\partial N_{j,p}^k}{\partial u_{j,p}^i} & & \\ . & & & & . & . \\ \frac{\partial N_{j,n}^k}{\partial u_{j,-1}^i} & . & . & . & . & \frac{\partial N_{j,n}^k}{\partial u_{j,n}^i} \end{pmatrix}$$

And so on. A useful consequence of this setup is that J_i^i can always be made to be at least diagonal 'almost everywhere', if not the identity matrix. This is dependent on the choice of boundary conditions, with any non diagonal elements occuring in either the first or last row. It is also likely that this will be a banded lower triangular block matrix rather than a full lower triangular matrix depending on the time discretization, and the number of time steps. For example, the problem in the previous section resulted in a single lower band while the same problem utilizing a leap frog scheme would have two lower bands. Now, in order to solve 1.3 for λ^k we have a true back solve since the diagonal blocks are themselves diagonal matrices. So here, the solve looks just like a more generalized version of that found in the example problem.

Algorithm 2 Discrete Adjoint Solution, General Explicit Method

$J_m^m \lambda^m = G^m$
for i=m-1 to 1 **do**
  $J_i^i \lambda^i = G^i - \sum_{k=i+1}^m {J_i^k}^T \lambda^k$
end for
$\lambda^0 = G^0 - \sum_{k=1}^m {J_0^k}^T \lambda^k$

At this stage, the setup seems (and is) rather nice for simple problems, as the user can determine the individual sub-Jacobian matrices needed to solve the

problem. This can, however, quickly become much more complicated once the governing equations and their associated discretizations become more complex, e.g. higher dimensions, systems of equations, higher order methods, etc. So now, we would like to find an easier way to compute the sub-Jacobians. For this we will define the entries of J_i^k in terms of the approximate Fréchet derivative so that

$$J_i^k e_j = \frac{N^k(u_j^i + \epsilon e_j) - N^k(u_j^i - \epsilon e_j)}{2\epsilon}$$

where e_j is the j^{th} column of the identity matrix. So for example in 1-D, the (j,p) entry of J_i^k is $\frac{\partial N_j^k}{\partial u_p^i}$ and we would have

$$\frac{\partial N_j^k}{\partial u_p^i} = \frac{N_j^k(u_p^i + \epsilon) - N_j^k(u_p^i - \epsilon)}{2\epsilon}$$

Example: To compute ${J_i^k}^T$ for the discretization 1.5 we could say,

Algorithm 3 Sub-Jacobian Computation

for p=1 to n **do**
  if i=k or i=k+1 **then**
    set $temp1(u) = u$
    set $temp2(u) = u$
    $temp1(u_p^i) = u_p^i + \epsilon$
    $temp2(u_p^i) = u_p^i - \epsilon$
    for j=1 to n **do**
      $N1 = \frac{temp1(u_j^{k+1}) - temp1(u_j^k)}{\Delta t} + a\left(\frac{temp1(u_j^k) - temp1(u_{j-1}^k)}{\Delta x}\right)$
      $N2 = \frac{temp2(u_j^{k+1}) - temp2(u_j^k)}{\Delta t} + a\left(\frac{temp2(u_j^k) - temp2(u_{j-1}^k)}{\Delta x}\right)$
      ${J_i^k}^T(p,j) = \frac{\partial N_j^k}{\partial u_p^i} = \frac{N1 - N2}{2\epsilon}$
    end for
  else
    ${J_i^k}^T(p,j) = \frac{\partial N_j^k}{\partial u_p^i} = 0$
  end if
end for

And so we now have a way to compute the Jacobians required in Algorithm 2 that avoids the error prone and tedious task of explicitly determining them.

This allows for an automated procedure capable of solving the adjoint problem for even the most difficult of discretization methods. Note that for linear problems, this matrix could be computed once and stored if desired, while for the nonlinear case one needs to utilize the solution data from the forward solve of the governing equation, and so must be recomputed at every step. Note that unlike automatic differentiation (AD), this method does not require any specific problem by problem coding, and only relies on the solution data itself. Since AD will not be used here, we will not describe it but will note that it is a popular method due to its high accuracy, and good treatments of this can be found in [9], [7] and [2].

One should take note that in most methods, a vast majority of the J_i^k matrices will be 0. Indeed, as previously stated it is dependent upon the time discretization. In general, an n^{th} order explicit time discretization that utilizes data from only n previous time steps will result in

$$J_i^k = 0 \qquad if \quad |i-k| > n$$

We should also note that in the case of Dirichelet boundary conditions, $J_i^i = I$ and thus will not need to be computed. Also note that in a parallel setting, since we only need the matrix vector products $J\lambda$ in the computation, we do not need to store J, only the resultant vector. So J can be computed, multiplied by λ and be immediately discarded, hence reducing storage and communication costs.

The last bit of information needed is how to choose ϵ. In [12] a similar method is used in the Jacobian-free Newton Krylov method, and they state the simplest choice for epsilon as (in the translated context of this paper)

$$\epsilon = \frac{1}{N} \sum_{c=1}^{N} \beta u_c$$

Where N is the total number of grid points and β is a constant with value within a few orders of magnitude of machine epsilon.

Chapter 3

Numerical Results and Observations

Presented in this chapter are some numerical findings based on the solution of both a linear and nonlinear test problem formulated using the continuous and discrete versions of the adjoint method. For the linear case, a 1-D advection problem identical to that found in the example of this chapter is solved, while for the nonlinear case, a similar 1-D problem is posed utilizing Burgers equation in place of the advection equation. Comparisons for the two formulations of each problem have been made, and we have found that the solutions utilizing the stored matrix and the computed matrix free versions of the flow Jacobian are nearly identical. We have also found that for these test cases, the use of the penalty term $(\nabla u)^T(\nabla u)$ and the traditional background, or regularization term in the cost function give essentially the same result, producing solution errors that are consistently within the same order of magnitude. Some results for various cases are now presented, the steepest descent method is used for the minimization in each.

3.1 Linear Advection Problem

For the linear advection example, we demonstrate a recovery of the initial conditions in a 1-D data assimilation problem whose solution possesses a steep gradient, and starting with a poor initial guess. The problem is

$$min \quad J(u,\phi) = \int_{-1}^{1} (u(x,T) - u_d(x,T))^2 dx + \beta(\nabla u(x,0))^T (\nabla u(x,0))$$

$$subject\ to \quad N(u,\phi) = u_t + a u_x = 0$$

The data, u_d, is generated by advecting a step function IC forward using the same discretization as for the problem solution. The discretization method is again the forward Euler method, the exact IC we wish to recover is

$$g(x) = \begin{cases} 1.0 & -0.5 < x < 0 \\ 0.5 & Otherwise \end{cases}$$

and we start with an initial guess of $g(x) = 0$. The plots below (figures 3.1 - 3.4) show the actual data, and the solution u after convergence. The plots to the left are at time t=0, and show the exact IC we wish to recover (solid line), and the IC recovered using the adjoint (dash dot line). The plots on the right are at time t=1.0, and show the advected solutions from these starting IC's. These plots all have a value of $\beta = 1.0$ for the cost function.

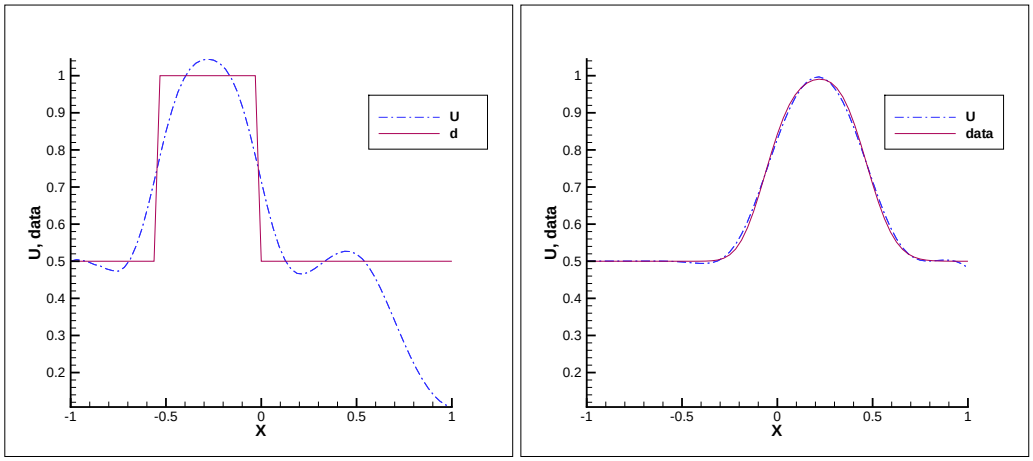

Figure 3.1: Linear IC Recovery. Left: Recovered IC for 64 grid points (U) versus exact IC (data). Right: Comparison of the advected solutions

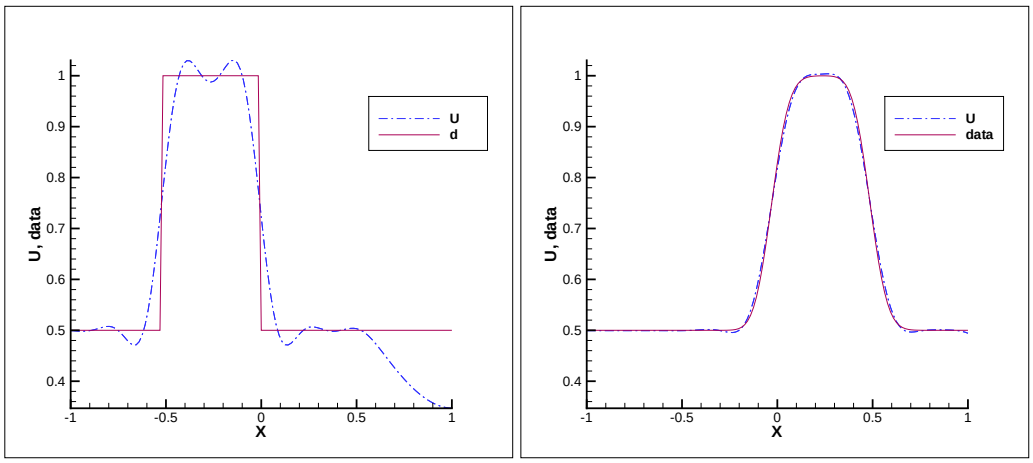

Figure 3.2: Linear IC Recovery. Left: Recovered IC for 128 grid points (U) versus exact IC (data). Right: Comparison of the advected solutions

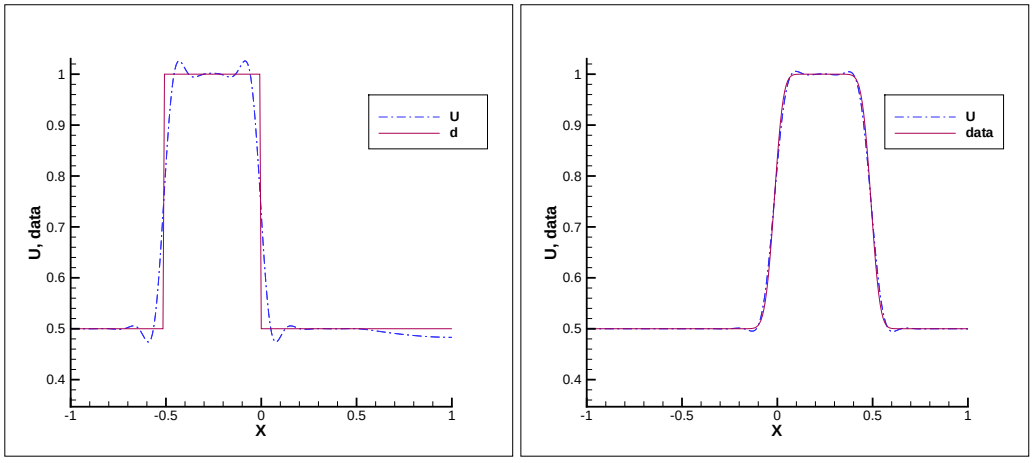

Figure 3.3: Linear IC Recovery. Left: Recovered IC for 256 grid points (U) versus exact IC (data). Right: Comparison of the advected solutions

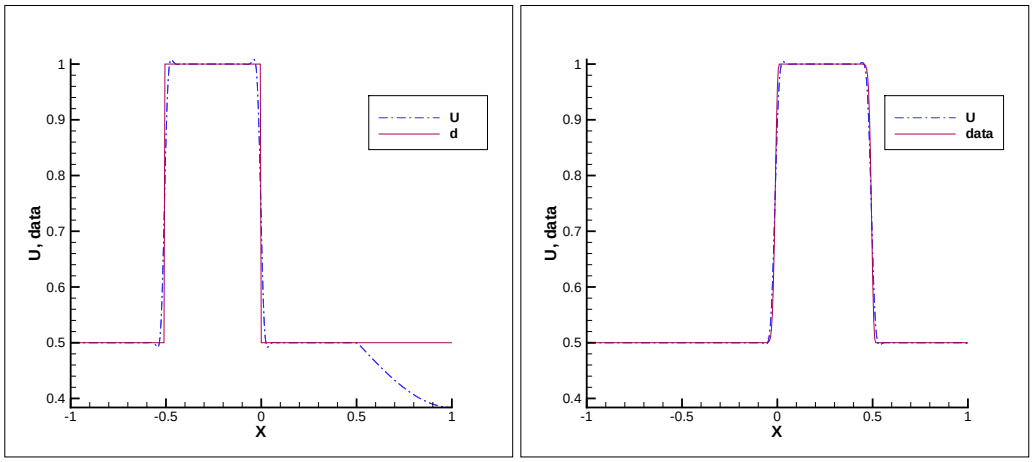

Figure 3.4: Linear IC Recovery. Left: Recovered IC for 400 grid points (U) versus exact IC (data). Right: Comparison of the advected solutions

3.2 Nonlinear Burgers Equation Problem

For the Burgers problem, we will again use a single set of final end time observations and a poor initial guess of $g(x) = 0.5$, with the fixed parameters N=96, dt=0.005, beta = 1.0. To demonstrate the effects of nonlinear features on the solution, we have solved this problem for various end times. The first figure 3.5 shows that the IC is recovered to within a pretty close range of the exact IC used, but only after a very short time. As time progresses, however, one can see that the recovered condition becomes progressively worse to the point of showing little resemblance as in figure 3.9. To remedy this, additional time observations must be provided. Note that increasing the number of observations in time directly leads to improved solution quality for nonlinear problems, as seen in figures 3.5 - 3.7. This is important, as it is known that increasing the number of observational data points for a single time does not have a major impact [5], but here we see that increasing the number of time observations indeed does.

3.3 Discussion of Numerical Results

We should note that the numerical experiments carried out here were under conditions far from ideal. Usually, a good starting estimate along with more available data (i.e. data collected at multiple time intervals) is used for the problem solution in data assimilation problems, but here we have purposely used poor initial guesses along with limited data. It can be seen that for linear problems, the adjoint method works very well and is able to recover a difficult

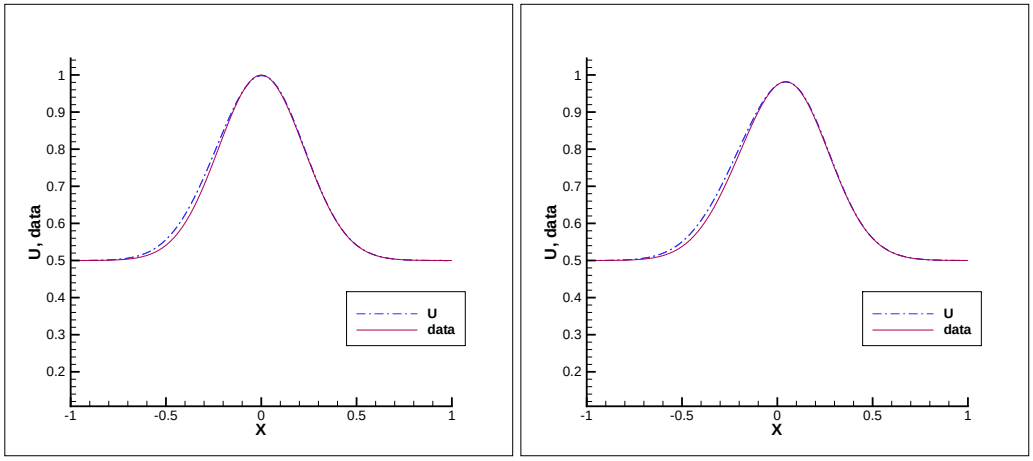

Figure 3.5: Nonlinear Burgers' equation IC Recovery. Left: Recovered IC (U) versus exact IC (data). Right: Comparison of the final solutions, t=0.05

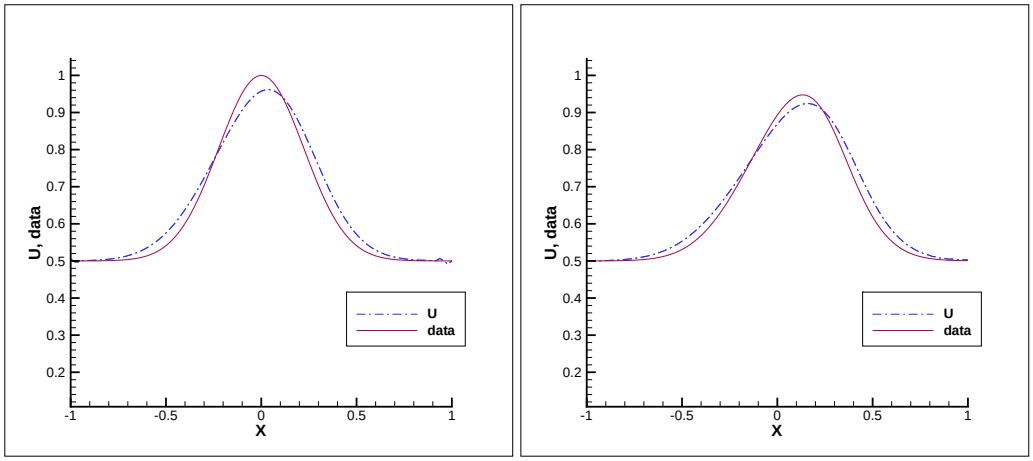

Figure 3.6: Nonlinear Burgers' equation IC Recovery. Left: Recovered IC (U) versus exact IC (data). Right: Comparison of the final solutions, t=0.15

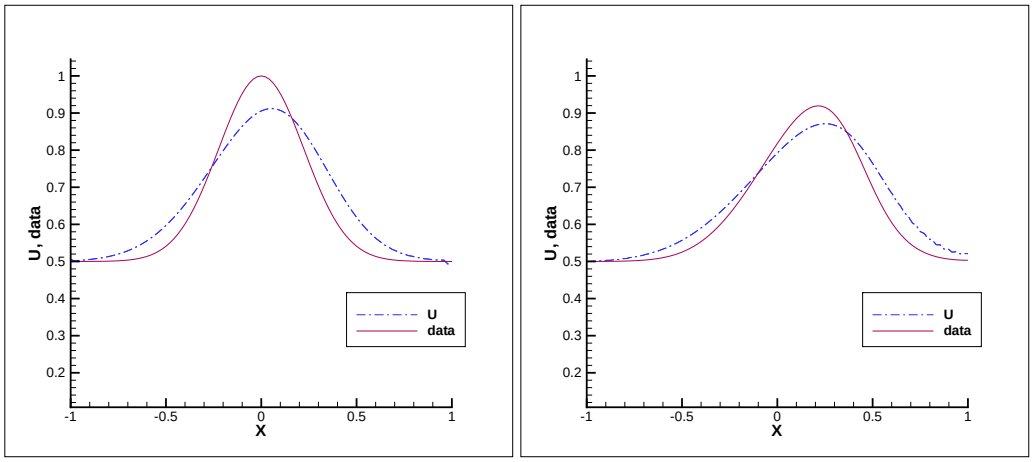

Figure 3.7: Nonlinear Burgers' equation IC Recovery. Left: Recovered IC (U) versus exact IC (data). Right: Comparison of the final solutions, t=0.25

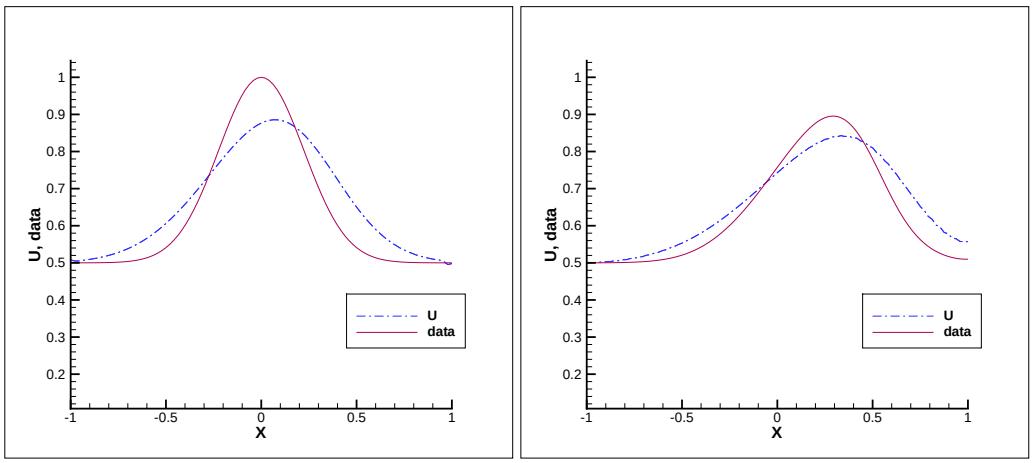

Figure 3.8: Nonlinear Burgers' equation IC Recovery. Left: Recovered IC (U) versus exact IC (data). Right: Comparison of the final solutions, t=0.35

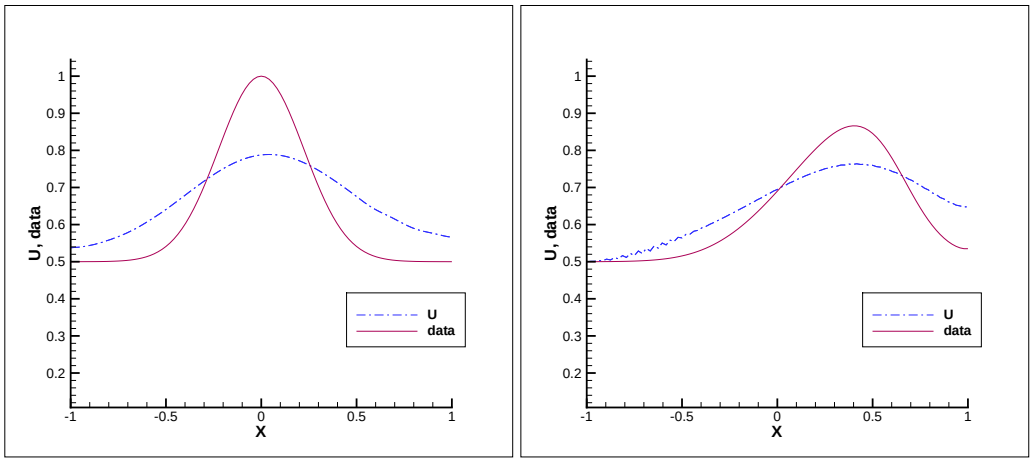

Figure 3.9: Nonlinear Burgers' equation IC Recovery. Left: Recovered IC (U) versus exact IC (data). Right: Comparison of the final solutions, t=0.5

initial condition possessing steep gradients with ease using only a single set of observed solutions, and that as the computational grid is resolved the error tends towards zero up the point that the CFL condition causes the adjoint solution to become unstable.

The nonlinear Burgers problem is a different story, as poor initial guesses result in poor solutions. This is likely due to the nonlinear features, and in this scenario the use of data for multiple times should be used to correct this problem. The figures provided (3.11, 3.12, 3.13) demonstrate how increasing the number of observations leads to improved solutions for nonlinear problems. In addition to the findings above, it appears that while the addition of a penalty or background term is necessary for the linear problem, it is unecessary for the nonlinear case, and in fact degrades the solution performance. The early results demonstrating this are shown in table 3.3, and it is clear that the best solutions are obtained using $\beta = 0$ and $N = 80$ grid points. Beyond $N = 112$ grid points, the solution fails to improve from the initial guess which is consistent with all of our numerical findings as the CFL number becomes too large.

Some conclusions that can be made at this point are indeed the fact that in the presence of a background term, the increase in the number of time observations directly leads to a decrease in the resulting ssolution error as can be seen in the figures 3.11 - 3.13 and table 3.3. If on the other hand the penalty or background term is neglected, we see that increasing the number of observations should decrease the error in the final solution, but this does not, however, necessarily mean that this will cause a decrease in the recovered IC as seen in table 3.3. This is exactly what we saw in the case of the linear problem when the penalty or background term was set to 0, as bad recovered IC's could

smooth out to match the final observation data. While completely neglecting the background term in this simple case may provide us with the best solution, this example shows us that it is likely not a good idea for more complex problems, and what we can learn from this is that more complex nonlinear problems should indeed include a background term along with as many time observations as is possible in order to produce the most accurate solution.

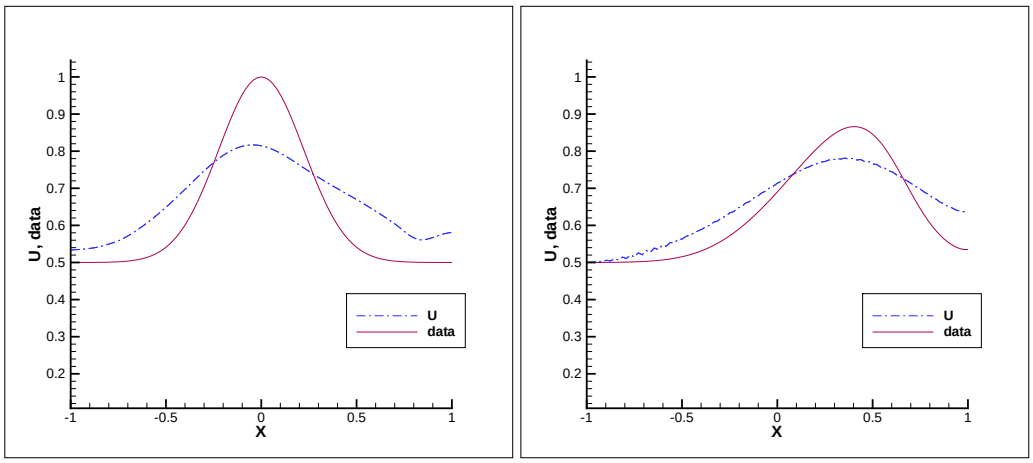

Figure 3.10: Nonlinear Burgers' equation IC Recovery, 2 observations at t=0.4 and t=0.5. Left: Recovered IC (U) versus exact IC (data). Right: Comparison of the final solutions, t=0.5

Table 3.1: Results for various runs of the 1-D nonlinear data assimilation problem for IC recovery with Burgers' equation for t=0.5 and $\Delta t = 0.005$. Obs = number of observations N = number of grid points, β = penalty constant. Errors are calculated using the L^2 norm.

Obs	N	β	Error in Recovered IC	Error in Final Solution
1	64	1.0	0.0288	0.00996
		0.5	0.0288	0.00996
		0.1	0.0283	0.00970
		0.0	0.00846	0.00202
	80	1.0	0.0224	0.0106
		0.5	0.0224	0.0106
		0.1	0.0176	0.00784
		0.0	0.00711	0.00271
	96	1.0	0.0218	0.0125
		0.5	0.0183	0.0104
		0.1	0.0157	0.00896
		0.0	0.0115	0.00691
	112	1.0	0.0174	0.0112
		0.5	0.0148	0.00951
		0.1	0.0129	0.00827
		0.0	0.0113	0.00747
2	64	1.0	0.0277	0.00946
		0.5	0.0276	0.00944
		0.1	0.0213	0.00695
		0.0	0.00636	0.00138
	80	1.0	0.0207	0.00967
		0.5	0.0208	0.00966
		0.1	0.0110	0.00289
		0.0	0.00579	0.00166
	96	1.0	0.0200	0.0112
		0.5	0.0152	0.00851
		0.1	0.0113	0.00674
		0.0	0.0101	0.00649
	112	1.0	0.0141	0.00895
		0.5	0.0127	0.00813
		0.1	0.0114	0.00750
		0.0	0.0111	0.00703

Table 3.2: Results for various runs of the 1-D linear advection data assimilation problem for IC recovery with Burgers' equation for t=0.5, $\Delta t = 0.005$, $N = 80$ and $\beta = 0.0$.

Obs	Error in Recovered IC	Error in Final Solution
1	0.00711	0.00271
2	0.00579	0.00166
5	0.00597	0.00162
10	0.00636	0.00155
20	0.00622	0.00122

Table 3.3: Results for various runs of the 1-D linear advection data assimilation problem for IC recovery with Burgers' equation for t=0.5, $\Delta t = 0.005$, $N = 80$ and $\beta = 0.5$.

Obs	Error in Recovered IC	Error in Final Solution
1	0.0224	0.0106
2	0.0208	0.00966
5	0.0180	0.00812
10	0.0113	0.00386
20	0.00991	0.00289

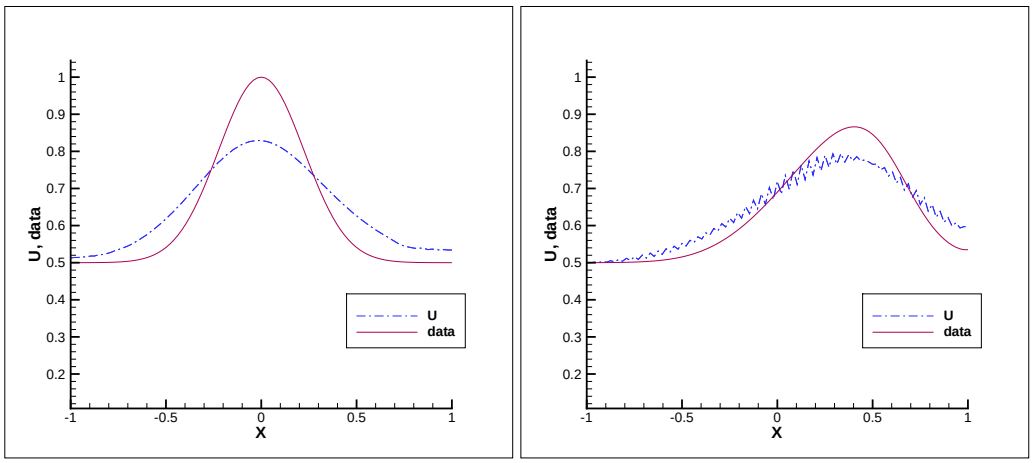

Figure 3.11: Nonlinear Burgers' equation IC Recovery, 5 observations between t=0.4 and t=0.5. Left: Recovered IC (U) versus exact IC (data). Right: Comparison of the final solutions, t=0.5

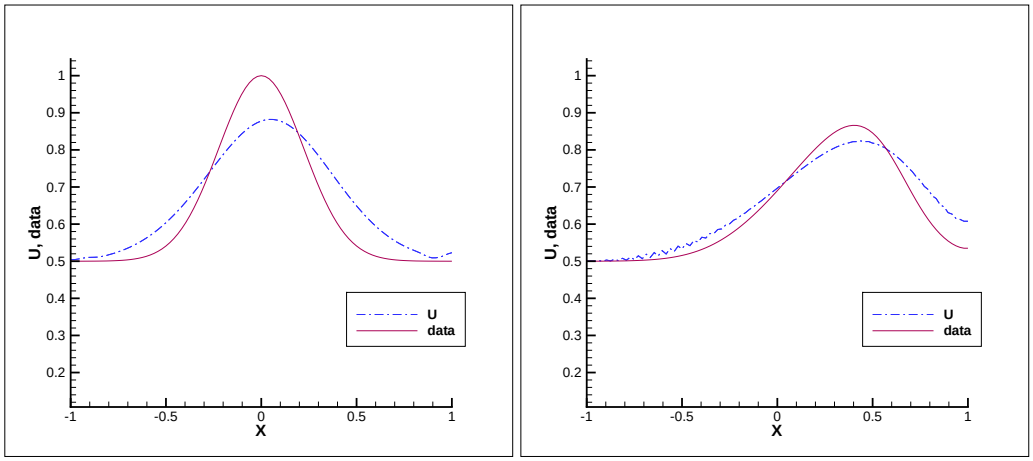

Figure 3.12: Nonlinear Burgers' equation IC Recovery, 10 observations between t=0.4 and t=0.5. Left: Recovered IC (U) versus exact IC (data). Right: Comparison of the final solutions, t=0.5

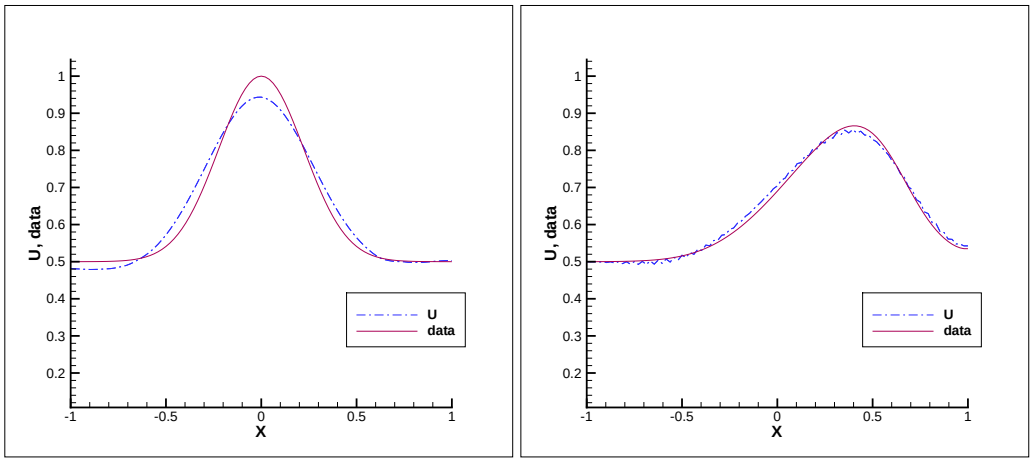

Figure 3.13: Nonlinear Burgers' equation IC Recovery, 20 observations between t=0.4 and t=0.5. Left: Recovered IC (U) versus exact IC (data). Right: Comparison of the final solutions, t=0.5

Bibliography

[1] Leon M. Arriola and James M. Hyman. Being sensitive to uncertainty. *Computing in Science and Engineering*, pages 10–20, 2007.

[2] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, editors. *The Sourcebook of Parallel Computing*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier, November 2002.

[3] F. Fang, C.C. Pain, M.D. Piggott, G.J. Gorman, and A.J.H Goddard. An adaptive mesh adjoint data assimilation method applied to free surface flows. *International Journal for Numerical Methods in Fluids*, 47:995–1001, 2005.

[4] F. Fang, M.D. Piggott, C.C. Pain, G.J. Gorman, and A.J.H Goddard. An adaptive mesh adjoint data assimilation method. *Ocean Modelling*, 15:39–55, 2006.

[5] David Furbish, M.Y. Hussaini, F.-X. Le Dimet, and Pierre Ngnepieba. On discretization error and its control in varational data assimilation. *Tellus A*, 60:979–991.

[6] Anne K. Griffith and N.K. Nichols. Data assimilation using optimal control theory. Numerical analysis report, The University of Reading, October 1994.

[7] J. Haslinger and R.A.E. Mäkinen. *Introduction to Shape Optimization: Theory, Apprximation and Computation*. Advances in Design and Control. SIAM, Philadelphia, PA, USA, 2003.

[8] Ronald D. Joslin, Max D. Gunzburger, Roy A. Nicolaides, Gordon Erlebacher, and M. Yousuff Hussaini. Self-contained automated methodology for optimal flow control. *AIAA Journal*, 35(5).

[9] Emmanuel Laporte and Patrick Le Tallec. *Numerical Methods in Sensitivity Analysis and Shape Optimization*. Modeling and Simulation in Science, Engineering and Technology. Birkhäuser, 2003.

[10] Charles A. Mader, Joaquim R. R. A. Martins, Juan J. Alonso, and Edwin Van Der Weide. Adjoint: An approach for the rapid developement of discrete adjoint solvers. *AIAA Journal*, 46(4):863–873, 2008.

[11] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. *ACM Transactions on Graphics*, 23:449–446, 2004.

[12] Robert Nourgaliev, Samet Kadioglu, Vincent Mousseau, and Dana Knoll. Marker re-distancing/level set (mrd/ls) method for high-fidelity implicit interface tracking. *submitted to SIAM Journal of Scientific Computing*, 2008.

[13] Oliver Pironneau. *Optimal Shape Design for Elliptic Systems*. Springer-Verlag, New York, NY, USA, 1984.

[14] Saad A. Ragab. Shape optimization of surface ships in potential flow using an adjoint formulation. *AIAA Journal*, 42(2):296–304, 2004.

[15] B. G. van Bloemen Waanders, R. A. Bartlett, S. S. Collis, E. R. Keiter, C. C. Ober, T. M. Smith, V. Akcelik, O. Ghattas, J. C. Hill, M. Berggren, M. Heinkenschloss, and L. C. Wilcox. Sensitivity technologies for large scale simulation. Technical Report SAND2004-6574, Sandia National Laboratories, January 2005.

[16] Qiqi Wang, David Gleich, Amin Saberi, Nasrollah Etemadi, and Parviz Moin. A monte carlo method for solving unsteady adjoint equations. *Journal of Computational Physics*, 227:6184–6205, 2008.