Boise State University

BSU

Final Project

INVERSE METHODS FOR SHALLOW WATER EQUATIONS

Author
Brian KYANJO

Supervisor
Prof. Jodi Mead

May 6, 2022

Boise State University

Abstract

Estimating parameters for shallow water equation (SWE) problems is still a challenge as its difficult to approximate meaningful estimates without obtaining negative depths and dry fonts (depth, $h = 0$) as these would break the available numerical approximation codes. However, inverse methods present a promising approach in estimating these parameters by taking advantage of some inverse methods techniques like the Occams' inversion model. In this work, the Forestclaw code (Dcalhoun and Cbasterde) is employed to generate synthetic data, and several values of the initial guesses (m_0) are tried in the Occam inversion model to see how close the recovered estimates are to the true parameter estimates (m_{true}). Results show that this depends on how close m_0 is to m_{true}.

Contents

1 Introduction

1.1 Background

The shallow water equations (SWE) are a system of hyperbolic partial differential equations (PDEs) describing the flow below a pressure surface in a fluid. They have been frequently used to model several real-life problems i.e. the propagation of tsunamis waves in the ocean (Dias and Dutykh, 2007) and modeling of atmospheric turbulence. Deep knowledge is required to handle such events, therefore first, robust, and computationally efficient methods like inverse methods are required to solve the shallow water equations.

1.2 General Objectives

- To discover the best approach for choosing an initial guess (m_0) for the SWE problem while recovering true estimates (m_{true}).
- To verify if the confidence interval of the recovered estimates captures the true parameter estimates.
- To check the relationship between the recovered estimates.
- To investigate how the order of the roughening matrix impacts the χ^2, p-value, and L2-norm of the recovered estimates.

1.3 Literature

Inverse methods have been frequently used to handle shallow water problems; for instance, Voronina (2013) proposed a technique that reconstructed the initial tsunami waveform (ill-posed inverse problem) using inversion of the remote measurements of the water level data. This was done by regularizing the ill-posed inverse problem using least-squares inversion based on the truncated singular value decomposition technique. Results from this approach indicated instability control of the numerical solution. Furthermore, Gessese and Sellier (2012) derived explicit partial differential equations that directly reconstructed the channel bed topographic elevation (inverse problem). Several tests of the technique on various sets of artificial data-sets generated from solving the forward SWE problem depicted at most 3% level of accuracy in reconstructing the inverse problem.

2 Problem Formulation and Methods

2.1 Formulation of the Forestclaw code

2.1.1 Riemann solvers

Riemann solvers are numerical methods in which time-averaged fluxes of all conserved quantities are calculated to solve fundamental problems in conservation laws named Riemann problems. A Riemann problem can be defined as a specific initial value problem (Cauchy) of a partial differential equation (PDE) that consists of conservation equations (1) combined with piece-wise constant initial data which has a single discontinuity in the domain of interest as shown in equation (2) (George, 2011).

$$q_t + f(q)_x = 0 \tag{1}$$

$$q(x,0) = \begin{cases} q_L, & \text{if} \quad x \leq 0, \\ q_R, & \text{if} \quad x > 0, \end{cases} \tag{2}$$

where $f(q)_x \in \mathbb{R}^m$ is a vector of conserved quantities, q_R and q_L are two piece-wise constant states separated by a discontinuity.

2.1.2 Shallow Water Equations (SWE)

The SWE is a system of hyperbolic PDEs governing the flow below a pressure surface in a fluid. They arise from the Navier-Stokes equations. In one dimension, the SWE can be used to model a fluid in a channel of unit width, taking the vertical velocity negligible, and horizontal velocity roughly constant throughout any cross-section of the channel (George, 2008).

Consider small-amplitude waves in a one-dimensional fluid channel that is shallow relative to its wavelength. The conservation of momentum equation is written in terms of pressure, $p(x,t) = \frac{1}{2}gh^2$, and the height field $h(x,t)$ (m), which breaks down into system (3).

$$\begin{aligned} h_t + (hu)_x &= 0 \\ (hu)_t + \left(hu^2 + \frac{1}{2}gh^2\right)_x &= 0 \end{aligned} \tag{3}$$

where hu measures the flow rate of water past a point, ρ (kg/m^3) is the constant density of the in-compressible fluid, and $u(x,t)$ (m/s) is the horizontal velocity.

A very simple set of initial conditions is a single discontinuity at the middle of the channel. In this case, we set h and hu equal to constants on either side of the channel. This problem is a classic Riemann Problem, and for the SWE, has an exact solution. We assume the discontinuity is at $x = 0$. The variation of h and hu on either side of the discontinuity leads the waves in the Riemann problem to move at different speeds creating discontinuities (shocks) or changing regions (rarefactions) (LeVeque et al., 2002). At $x = 0$ and $t = 0$, the discontinuity is located between the left and right state, so the solution at the left $(q_l)$and right(q_r) states are given by:

$$q_l = [h_l,\, u_l,\, hu_l]^T \quad \text{and} \quad q_r = [h_r,\, u_r,\, hu_r]^T \tag{4}$$

As t increases, four distinct regions are created, separated by characteristics. The middle state called the intermediate state($q_m = [h_m,\, u_m,\, hu_m]^T$), is generated. The determination of this state characterizes the Riemann problem and how it connects to other states via waves in each respective characteristic family (Bale et al., 2003). This can only hold if the connection wave speeds satisfy the Lax entropy condition. The intermediate state is obtained by solving the Riemann problem using the exact or approximate method, however, in this project, we considered the exact solver, due to its computational accuracy, robustness, and ability to handle wet and dry states better than the numerical approach.

2.1.3 Exact Riemann Solver

The states can be separated by either *shocks* or *rarefactions*. General left and right states can be connected by a combination of the two (either two shocks, two rarefactions, or one of each). We describe how to determine if two states are connected by a shock. We refer the reader to LeVeque et al. (2002) other cases.

We can obtain an exact solution to the Riemann Problem for the SWE as follows. The shock speed, $s(t)$, from the shock wave as the solution emerges is determined from the Rankine-Hugoniot jump condition given by equation (5) which must be satisfied across any shock wave. If q_l and q_r are connected by a shock, the Rankine Hugoniot conditions will be satisfied (Mandli et al., 2016).

$$\begin{aligned} s_1(q_m - q_l) &= f(q_m) - f(q_l) \\ s_2(q_r - q_m) &= f(q_r) - f(q_m) \end{aligned} \tag{5}$$

Applying condition (5) to shallow water equations (3) creates a system of four equations (6) that must be satisfied simultaneously.

$$\begin{aligned} s_1(h_m - h_l) &= hu_m - hu_l \\ s_1(hu_m - hu_l) &= hu_m^2 - hu_l^2 + \frac{1}{2}g(h_m^2 - h_l^2) \\ s_2(h_r - h_m) &= hu_r - hu_m \\ s_1(hu_r - hu_m) &= hu_r^2 - hu_m^2 + \frac{1}{2}g(h_r^2 - h_m^2) \end{aligned} \tag{6}$$

Since the (h_l, u_l) and (h_r, u_r) are fixed, we find all states: (h_m, u_m) and their corresponding speeds: s_1 and s_2 that satisfy system (6). We have four equations and four unknowns, which gives a two parameter family of solutions: one-shock and two-shock. Using h_l and h_r as parameters, corresponding u_l, u_r, s_1,and s_2 are determined for each h_l and h_r.

Consider a general Riemann problem whose known solution consists of two shocks with initial data (4). This problem can be solved by finding the state q_m that can be connected to q_l by a *1-shock* and simultaneously connects to q_r by a *2-shock*. The point q_m lies on the curve (7) of points through point q_r that connects to q_r by a *2-shock* (Berger et al., 2011).

$$u_m = u_r + (h_m - h_r)\sqrt{\frac{g}{2}\left(\frac{1}{h_m} + \frac{1}{h_r}\right)} \tag{7}$$

Likewise, the state(q_m), must also lie on the Hugonoit locus (equation (8)) of the 1-shock wave passing through q_l

$$u_m = u_l - (h_m - h_l)\sqrt{\frac{g}{2}\left(\frac{1}{h_m} + \frac{1}{h_l}\right)} \tag{8}$$

Equations (7) and (8) form a system of two equations with two unknowns (h_m and u_m) that are equated since the left-hand sides of both equations are equal. A nonlinear equation that consists of only one unknown h_m is formed and solved using an iterative method such as the Newton method to obtain a desired intermediate state in the Riemann solution (LeVeque et al., 2011).

As an example, consider a SWE Riemann problem with $h_l = h_r = 1, u_l = 0.5$, and $u_r = -0.5$. These initial values are used by the Newton solver to solve equations (7) and (8) to produce $h_m = 1.554$. The shock speeds (s_l and s_r) in each region are different due to different wave characteristics, we use this concept to loop through all interfaces and determine h, u, and hu in each region as shown in the code 5.1 . At each interface the function *forestclaw*, is called and respective values of height (h), velocity (u) and momentum (hu) solutions are stored, forming the simulated data ($G(m)$) where $m = [q_l,\, q_m,\, q_r]$. According to Lax entropy conditions, a *1-shock* that physically connects q_l to q_m is obtained if $h_m > h_l$, and similarly a *2-shock* wave that physically connects q_m to q_r requires $h_m > h_r$ (LeVeque et al., 2011).

The synthetic data, d_s, is generated at each interface by adding noise (ϵ) to obtained simulated data ($G(m)$) as shown in equation (9).

$$d_s = G(m) + \epsilon, \qquad \text{for } \epsilon \sim N(0, \sigma^2) \tag{9}$$

2.2 Occam Inversion

The Occam model was implemented and used to recover the true parameter estimates (m_{true}) for the SWE problem using the following inputs;

- Synthetic data (d_s) given by equation (9)
- A regularization parameter, $\delta = \sigma\sqrt{N}$, where σ is the uncertainity in the simulated data ($G(m)$) and N is the size of the spatial domain.
- A different roughening matrix (L) for each simulation i.e, either zeroth-order Tikhonov regularization (L_0) or a finite difference approximation of a first (L_1) or second (L_2) derivative for higher-order regularization as shown in equation (10)

$$L_0 = \mathbf{I}, \quad L_1 = \begin{pmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \\ & & & & -1 \ 1 \end{pmatrix}, \quad \text{and} \quad L_2 = \begin{pmatrix} 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \ 1 \end{pmatrix} \tag{10}$$

where $\mathbf{I} \in \mathbb{R}^{3\times 3}$ is an identity matrix, $L_1 \in \mathbb{R}^{2\times 3}$ and $L_2 \in \mathbb{R}^{1\times 3}$.

- An initial solution ($m^{(0)}$) given by equation (11), where $q_l^{(0)}$ is the intial left state, $q_m^{(0)}$ is the value of $G(m)$ at $t = 0$ obtained from the Forestclaw code, and $q_r^{(0)}$ is the initial right state.

$$m^{(0)} = \begin{pmatrix} q_l^{(0)} \\ q_m^{(0)} \\ q_r^{(0)} \end{pmatrix}^T = \begin{pmatrix} h_l^{(0)} & u_l^{(0)} & hu_l^{(0)} \\ h_m^{(0)} & u_m^{(0)} & hu_m^{(0)} \\ h_r^{(0)} & u_r^{(0)} & hu_r^{(0)} \end{pmatrix} \tag{11}$$

- A nonlinear forward model, $G(m)$, and its Jacobian, J_{ij}, which is designed based on a centered difference approach as shown in equation (12).

$$J_{ij} \approx \frac{(G(m + h_s e_j)_i - G(m - h_s e_j))_i}{2h_s}, \tag{12}$$

where h_s is the step size, $e_j \in \mathbb{R}^3$ is a vector of ones, $j = 1, 2, 3$, and $i = 1, 2, ..., N$.

Since the number of equations are greater than the number of parameter estimates, then at each iterations, the model uses the discrepancy principle to search for a solution that minimizes $||Lm||_2$ subject to the constraint $||G(m) - d_s||_2 \leq \delta$ while updating the Jacobian ($J(m^{(k)})$) and the vector $\hat{d}(m^{(k)})$ given by equation (13) at every k^{th} iteration.

$$\hat{d}(m^{(k)}) = d_s - G(m^{(k)}) + J^{(k)} m^{(k)} \tag{13}$$

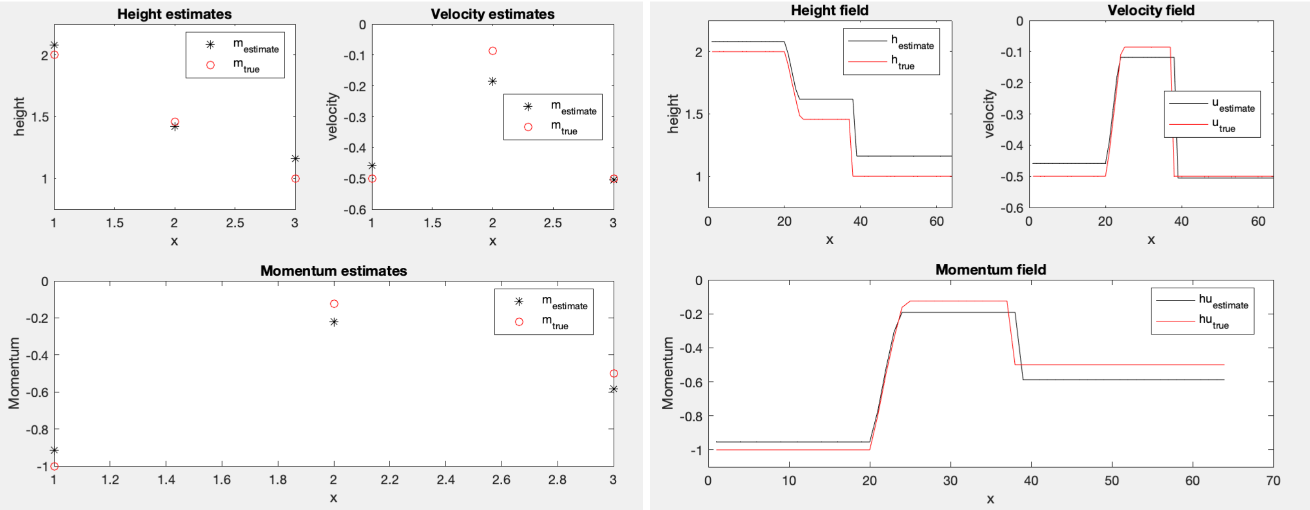
Then the updated modules corresponding to the mean of the regularization parameter values are computed using equation (14).

$$m^{k+1} = (J(m^{(k)})^T J(m^{(k)}) + \alpha^2 L^T L)^{-1} J(m^{(k)})^T \hat{d}(m^{(k)}) \tag{14}$$

where α is a regularization parameter. The procedure is repeated several times until when a specific value of m^{k+1} with the maximum value of α such that $\chi^2(m^{(k+1)}) \leq \delta^2$ is found, otherwise instead a value of α that minimizes $\chi^2(m^{(k+1)})$ is used to locate m^{k+1}.

3 Results and Discussion

In this section results obtained from various simulations are presented and discussed

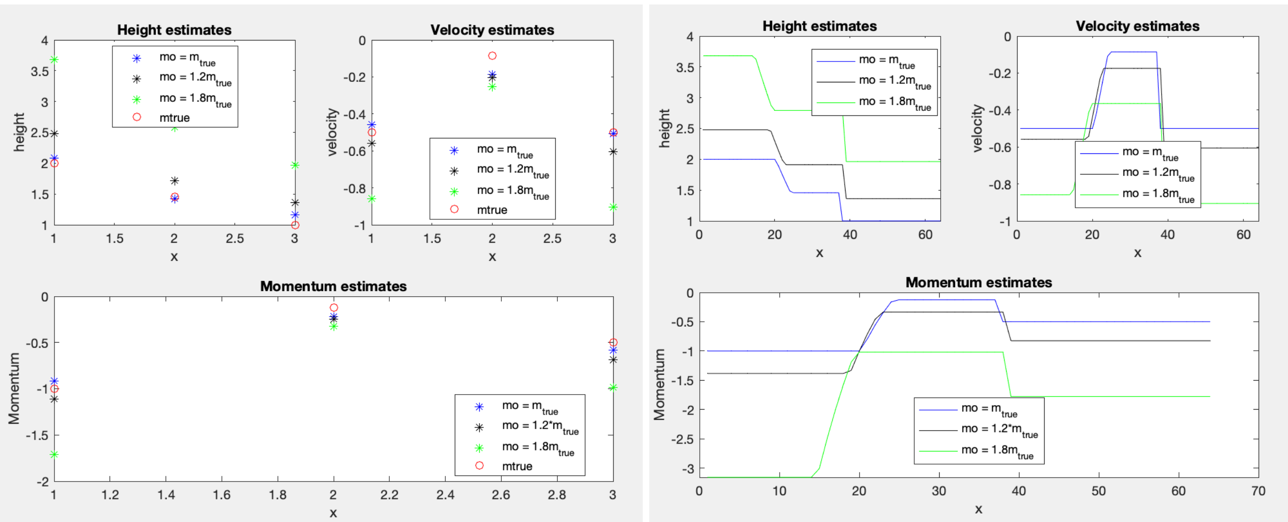


(a) (b)

Figure 1: (a) and (b) respectively show the parameter estimates and their corresponding data

Figure 1a shows the comparison between the recovered estimates and the true estimates obtained using $m_0 = m_{true}$ for height, velocity, and momentum. As seen on the graphs above, each parameter is recovered differently, for instance, the left estimates for height, velocity, and momentum are recovered close to the true estimates (red). The middle estimate for height and right estimate for the velocity almost coincide with the true estimates. Even though both the middle and right momentum and right height estimates are not recovered so closely to m_{true}, the intermediate velocity estimate is recovered kinder far from m_{true}.

Figure 1b depicts the comparison between the recovered fields and the true fields obtained using $m_0 = m_{true}$ for height, velocity, and momentum. These recovered fields depend on how the estimates in figure 1a are recovered, however, in all the fields the shapes of the true fields (red) are recovered even though they are out of phase in some regions.



(a) (b)

Figure 2: (a) and (b) respectively show the parameter estimates and their corresponding data based on the value of the initial estimates.

Figures 2a and 2b represents recovered estimates depending on the initial estimates (m_0) used. In both cases we can see, host best we can choose m_0 to recover an estimate close enough to m_{true}. According to the graphs and the p-values in table 1, the closer m_0 is close to m_{true}, the closer the recovered estimates are to m_{true}. For m_0, not close to m_{true} ($m_0 = 1.8m_{true}$), we see from both the figures and the last column table 1 that the estimates are not well recovered even though the field shapes is replicated. p-value = 0 in table 1 explains that the $m_0 = 1.8m_{true}$ used is not good data to be used hence we reject the null hypothesis.

	$m_0 = m_{true}$	$m_0 = 1.2m_{true}$	$m_0 = 1.8m_{true}$
h	0.69264	0.0099329	0
u	0.97701	0.90333	0.062386
hu	0.94026	0.10929	0

Table 1: P-values of different values of m_0

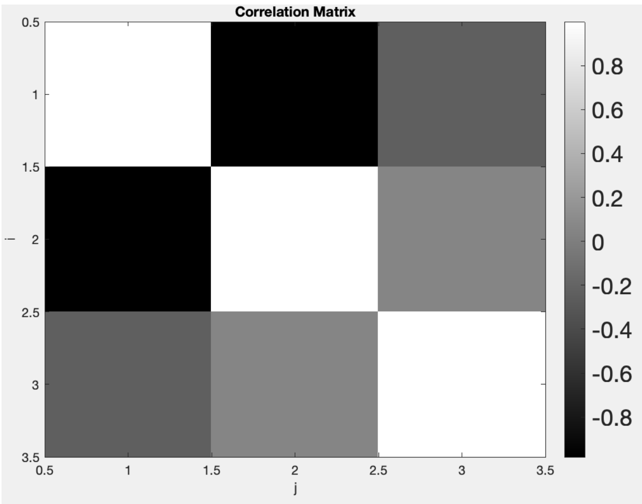


Figure 3: Correlation matrix

Figure. 3, displays the correlations between the recovered parameter estimates. The correlation matrix explains the correlation between the recovered estimates. Basing on figure 3, the entry for $\sigma_{h,u} \approx -0.95$ depicts that the recovered: h and u are highly negatively correlated and statistically dependant, for $\sigma_{h,hu} \approx -0.21$ shows that recovered estimates: h and hu are poorly negatively correlated, and for $\sigma_{u,hu} \approx -0.048$ represents that recovered estimates: u and hu are weakly negatively correlated.

$$\text{Covariance matrix} = \begin{pmatrix} 4.4814 & -4.4333 & -0.2542 \\ -4.4333 & 4.7621 & -0.0597 \\ -0.2542 & -0.0597 & 0.3228 \end{pmatrix} \tag{15}$$

According to equation (15), the product of the major diagonal elements is not equal to zero ($\sigma_{11}\sigma_{22}\sigma_{33} \neq 0$), which implies that calculated interval does not capture the relationship between the parameters: height (h), velocity (u), and momentum (hu).

$$m_{true} = \begin{pmatrix} 2.000 & -0.500 & -1.000 \\ 1.4571 & -0.0858 & -0.1250 \\ 1.000 & -0.500 & -0.500 \end{pmatrix} \tag{16}$$

$$C_h = \begin{pmatrix} -2.0528 & 6.2456 \\ -2.7290 & 5.5694 \\ -2.9935 & 5.3049 \end{pmatrix} \quad C_u = \begin{pmatrix} -4.7359 & 3.8184 \\ -4.4700 & 4.0844 \\ -4.7801 & -3.7742 \end{pmatrix} \quad C_{hu} = \begin{pmatrix} -2.0406 & 0.1867 \\ -1.3347 & 0.8927 \\ -1.6924 & 0.5349 \end{pmatrix} \tag{17}$$

C_h,C_u, and C_{hu} respectively represent the confidence interval for parameter: height (h), velocity (u), and momentum (hu). Comparing the first, second, and third columns of equation (16) with C_h,C_u, and C_{hu} in equation (3) show that the true parameters are captured with in the confidence interval.

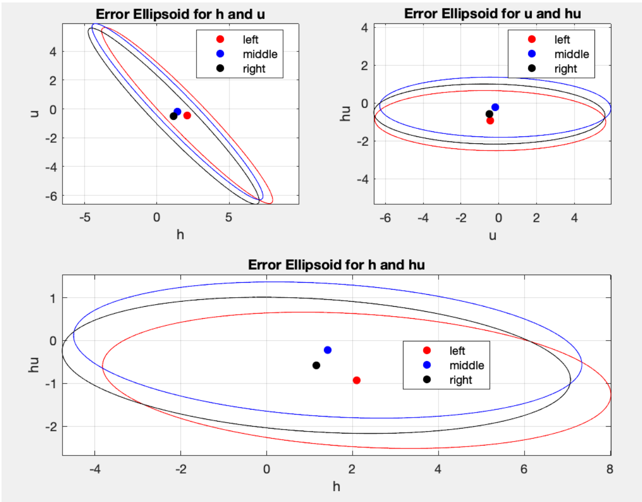


Figure 4: Linear error ellipsoids

Figure. 4, contains three linear error ellipsoids whose width narrows as they tend to the vertices and with a longer length, these depict the confidence regions between parameters: h and u, u and hu, and h and hu. The red, blue, and black dots represent the left, middle and right parameter estimates. Since these estimates lie at the center for each of the corresponding color regions, implies that the estimates lie within the confidence regions.

	L0			L1			L2		
	h	u	hu	h	u	hu	h	u	hu
χ2	1.4564	0.20341	0.39861	1.4564	0.20356	0.39916	1.4559	0.20311	0.39858
p-value	0.69237	0.97704	0.94053	069237	0.97701	0.94042	0.6925	0.97709	0.94054
L2-norm	0.1888			0.1884			0.1880		

Table 2: The χ^2, p-value, and the L2-norm for the three roughening matrices: zeroth, first, and second.

As seen in the table2, the order of the roughening matrix affects the χ^2, p-value, and the L2-norm. This is because of the order of accuracy of each matrix, for instance, L_2 is second-order accurate, L_1 is first-order accurate and L_0 is an identity which means it has no impact on the regularization term (Lm) as L_1 and L_2, which explains the variations in the χ^2, p-value, and the L2-norm for each recovered parameter estimates.

4 Conclusion

The estimated parameters and fields which are obtained by choosing $m_0 = m_{true}$ would have been recovered exactly as expected. However, this is not possible due to many factors i.e., the values of h_s used in the Jacobian matrix ((12)), order of the roughening matrix ((10)) used, final time, size of the spatial domain, the uncertainty in the simulated data (σ, the range in which the regularization parameter (α) is selected from, how the noise added to the simulated data is scaled, e.t.c.

Based on the results presented in the previous sector we also conclude that the confidence interval ((3)), the correlation matrix ((3)), and the covariance matrix ((15)) respectively captured the true parameter estimates, the correlation between the estimates, and did not capture the relationship between the parameters.

According to the table2, we can conclude that the order of the roughening matrix impacted the χ^2, p-value, and L2-norm of the recovered parameter estimates.

References

D. Bale, R. J. LeVeque, S. Mitran, and J. Rossmanith. A Wave Propagation Method for Conservation Laws and Balance Laws with Spatially Varying Flux Functions. 24(3): 955–978, 2003. doi: 10.1137/S106482750139738X. URL `http://dx.doi.org/10.1137/S106482750139738X`.

M. J. Berger, D. L. George, R. J. LeVeque, and K. T. Mandli. The GeoClaw software for depth-averaged flows with adaptive refinement. 34(9):1195–1206, September 2011. doi: 10.1016/j.advwatres.2011.02.016. arXiv:1008.0455v2 [physics.geo-ph].

F. Dias and D. Dutykh. Dynamics of tsunami waves. In *Extreme man-made and natural hazards in dynamics of structures*, pages 201–224. Springer, 2007.

D. L. George. Augmented Riemann solvers for the shallow water equations over variable topography with steady states and inundation. *J Comp Phys*, 227(6):3089 – 3113, 2008. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2007.10.027. URL `http://www.sciencedirect.com/science/article/pii/S0021999107004767`.

D. L. George. Adaptive finite volume methods with well-balanced Riemann solvers for modeling floods in rugged terrain: Application to the Malpasset dam-break flood (France, 1959). 66 (8):1000–1018, 2011. ISSN 1097-0363. doi: 10.1002/fld.2298. URL `http://dx.doi.org/10.1002/fld.2298`.

A. Gessese and M. Sellier. A direct solution approach to the inverse shallow-water problem. *Mathematical Problems in Engineering*, 2012, 2012.

R. J. LeVeque, D. L. George, and M. J. Berger. Tsunami modelling with adaptively refined finite volume methods. 20:211 – 289, May 2011. doi: 10.1017/S0962492911000043.

R. J. LeVeque et al. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

K. T. Mandli, A. J. Ahmadia, M. Berger, D. Calhoun, D. L. George, Y. Hadjimichael, D. I. Ketcheson, G. I. Lemoine, and R. J. LeVeque. Clawpack: Building an open source ecosystem for solving hyperbolic PDEs. 2(68), August 8 2016. doi: https://doi.org/10.7717/peerj-cs.68. URL `https://peerj.com/search/?q=mandli&t=&type=articles&subject=&topic=&uid=&sort=&journal=cs&discipline=&type=articles`.

T. Voronina. An inverse algorithm for reconstructing an initial tsunami waveform. *Tsunami: From Fundamentals to Damage Mitigation*, page 37, 2013.

5 Appendix

5.1 Forestclaw solver

```
%%======================================================================
% Boise State University
% Author: Brian KYANJO
% supervised by: Prof. Jodi Mead
% class: Inverse Methods
% Date: March 17th, 2022
% final project
%
% Description:
% -----------
% Uses the Initial Riemann problem to find an intemediate state (qm) which either
% the intial left or right state connects to it via any combination of shocks and
% rarefactions in the two families.
%
% For Riemann problems with an initial dry state on one side, the exact Riemann
% solution contains only a single rarefaction connecting the wet to dry state.
% The evolving wet dry interface is therefore simply one edge of the rarefaction.
% The propagation speed of this interface can be exactly determined using the
% Riemann invariants of the corresponding characteristics field.
%
% Input:
% -----
% x  - array of spacial points
% t  - array of temporal points
% mq - specifies the output (0 and 1 corresponds to h and hu respectively)
% ql - left initial state
% qr - right initial state
% Returns:
% h  - array of hieght field values
% hu - array of momentum field values
%%======================================================================

% Exact solver
function [Q,hs,us] = forestclaw(ql,qr,xi)
global g;
hl = ql(1); hr = qr(1);
ul = ql(2); ur = qr(2);

hs = Newton(hl,hr,ul,ur);  %calling the newton solver
us = ul - phi(hs,hl);

if xi <= us
```

```
if hs > hl
s = ul - sqrt(0.5*g*hs/hl*(hl+hs));
if xi <= s
h = hl;
hu = hl*ul;
else
h = hs;
hu = hs*us;
end
else
head = ul -sqrt(g*hl);
tail = us - sqrt(g*hs);
if xi <= head
h = hl;
hu = hl*ul;
elseif xi >= tail
h = hs;
hu = hs*us;
else
h = (((ul + 2*sqrt(g*hl) - xi)/3)^2)/g;
u = xi + sqrt(g*h);
hu = h*u;
end
end
else
if hs > hr
s = ur + sqrt(0.5*g*hs/hr*(hs+hr));
if xi <=s
h = hs;
hu = hs*us;
else
h = hr;
hu = hr*ur;
end
else
head = ur + sqrt(g*hr);
tail = us + sqrt(g*hs);
if xi >= head
h = hr;
hu = hr*ur;
elseif xi <= tail
h = hr;
hu = hr*ur;
else
h = (((xi-ur+2*sqrt(g*hr))/3)^2)/g;
u = xi - sqrt(g*h);
hu = h*u;
end
end
end
```

```
Q = [h;hu./h;hu];
end

% Newton solver
function [hs] = Newton(hl,hr,ul,ur)
global g;
hs = ((sqrt(hl) + sqrt(hr) - (ur-ul)/2/sqrt(g))^2)/4;
tol = 1e-12;
max_iter = 100;

for i=1:max_iter
gk = func(hs,hl,hr,ul,ur);
res = abs(gk);

if (res<tol)
break
else
continue
end

dg = dfun(hs,hl,hr,ul,ur);
dh = -gk/dg;
delta = 1;

for i=1:500
if (abs(func(hs+dh*delta,hl,hr,ul,ur)) >= res)
delta = 0.5*delta;
else
break
end
end
hs = hs + delta*dh;
end
end

% phi function
function [h] = phi(hs,hlr)
global g;
if (hs>hlr)
h = sqrt(0.5*g*(hs + hlr)/(hs*hlr))*(hs - hlr);
else
h = 2*sqrt(g)*(sqrt(hs) - sqrt(hlr));
end
end

% function f
function [f] = func(hs,hl,hr,ul,ur)
global g;
f = phi(hs,hl) + phi(hs,hr) + ur - ul;
end
```

```
% Jacobian of f
function [df] = dfun(hs,hl,hr,ul,ur)
global g;
eps = 1e-7;
df = (func(hs+eps,hl,hr,ul,ur) - func(hs-eps,hl,hr,ul,ur))/(2*eps);
end


function plot_ellipse(DELTA2,C,m,first,second)
n=5000;
%first - first parameter
%second - second parameter
%construct a vector of n equally-spaced angles from (0,2*pi)
theta=linspace(0,2*pi,n)';

%corresponding unit vector
xhat=[cos(theta),sin(theta)];
Cinv=inv(C);

%preallocate output array
r=zeros(n,2);
for i=1:n
%store each (x,y) pair on the confidence ellipse in the corresponding row of r
r(i,:)=sqrt(DELTA2/(xhat(i,:)*Cinv*xhat(i,:)'))*xhat(i,:);
end

% Plot the ellipse and set the axes.
for i = 1:3
a=['r' 'b' 'k'];
plot(m(i,first)+r(:,1), m(i,second)+r(:,2),'color',a(i)); hold on
plot(m(i,first),m(i,second),'.','color',a(i),'MarkerSize',17);grid on
axis equal
end
end
```

5.2 Model

```
%%===================================================================
% Author: Brian KYANJO
% supervised by: Prof. Jodi Mead
% class: Inverse Methods
% Date: March 17th, 2022
% final project
%  [Q] = model(m)
%
% INPUT
%   m - a guess at the model
%
% OUTPUT
```

```
%   Q   - a model matrix G(ql,qr,x/t) for the forward problem
%%==================================================================

function [Q] = model(m)
global N; global M; global h;
global t; global x;

for j= 2:M % time loop
for i=1:N % spartial loop
xi = x(i)/t(j);
[Q(:,i),hs,us] = forestclaw(m(1,:),m(3,:),xi);
end
end
Q = Q';
```

5.3 Jacobian

```
%%==================================================================
% Boise State University
% Author: Brian KYANJO
% supervised by: Prof. Jodi Mead
% class: Inverse Methods
% Date: March 17th, 2022
% final project
%  [J] = Jacobian(m)
%
% INPUT
%   m - a guess at the model
%
% OUTPUT
%   J   - its corresponding Jacobian
%%==================================================================

function [J] = Jacobian(m)
global N; global M; global h;
global t; global x;

for j= 2:M % time loop
J = [];
for i=1:N % spartial loop
xi = x(i)/t(j);

% Formulation of Jacobian Matrix
ej = ones(3,1);
[Qmin(:,i),hs,us] = forestclaw(m(1,:)-h*ej,m(3,:)-h*ej,xi);
[Qmax(:,i),hs,us] = forestclaw(m(1,:)+h*ej,m(3,:)+h*ej,xi);
J = [J (Qmax(:,i) - Qmin(:,i))./(2*h)]; % Jacobian
end
end
```

```
J = J';
```

5.4 Test script

```
%%====================================================================
% Boise State University
% Author: Brian KYANJO
% supervised by: Prof. Jodi Mead
% class: Inverse Methods
% Date: March 17th, 2022
% final project
% main_fuction script
%%=====================================================================
function main_function(mtrue,mo,h,L,x,ql,qr,M,t,mq)
global N; global sig;
hr = qr(1); hl = ql(1)
% Initial conditions
Q = zeros(3,N);
for i=1:N
if (x(i)<=0)
Q(:,i) = ql;
else
Q(:,i) = qr;
end
end

% writing a video
v = VideoWriter('dam.avi');
open(v);

for j= 2:M % time loop
J = [];
for i=1:N % spartial loop
xi = x(i)/t(j);
[Q(:,i),hs,us] = forestclaw(ql,qr,xi);
end

noise = sig*randn(size(Q(:,:)));      % genrate noise
d = Q(:,:) + noise;   d = d';         % add noise to the data

dh = d(:,1); du = d(:,2); dhu = d(:,3);
dnoise = [0.005*dh  0.005*du 0.005*dhu ];
d = Q(:,:) + dnoise';

d = d';                               % transposing data d
delta = sig*sqrt(N);                  % delta
fun = @(m) model(m);                  % model function handle
jac = @(m) Jacobian(m);               % Jacobian function handle
```

```
m = occam(fun, jac, L, d, mo, delta);    % calling the Occam model to  recover mtrue

% plotting estimates
figure(1)
if mq == 1
plot(m(:,mq),'k*'); hold on
plot(mtrue(:,mq),'ro'); hold off
legend('m_{estimate}','m_{true}',Location='best')
title('Height field');
ylim([hr-0.25,hl+0.25]);
ylabel('height');xlabel('x')
frame = getframe(gcf);
writeVideo(v,frame);
elseif mq == 2
plot(m(:,mq),'k*'); hold on
plot(mtrue(:,mq),'ro'); hold off
legend('m_{estimate}','m_{true}',Location='best')
title('velocity field')
ylabel('velocity');xlabel('x')
frame = getframe(gcf);
writeVideo(v,frame);
elseif mq == 3
plot(m(:,mq),'k*'); hold on
plot(mtrue(:,mq),'ro'); hold off
legend('m_{estimate}','m_{true}',Location='best')
title('momentum field')
ylabel('Momentum');xlabel('x')
frame = getframe(gcf);
writeVideo(v,frame);
else
subplot(2,2,1)
plot(m(:,1),'k*'); hold on
plot(mtrue(:,1),'ro');  hold off
legend('m_{estimate}','m_{true}',Location='best')
title('Height estimates')
ylim([hr-0.25,hl+0.25])
ylabel('height');xlabel('x')
subplot(2,2,2);
plot(m(:,2),'k*'); hold on
plot(mtrue(:,2),'ro'); hold off
legend('m_{estimate}','m_{true}',Location='best')
title('Velocity estimates')
ylabel('velocity');xlabel('x')
subplot(2,2,[3,4]);
plot(m(:,3),'k*'); hold on
plot(mtrue(:,3),'ro'); hold off
title('Momentum estimates')
ylabel('Momentum');xlabel('x')
legend('m_{estimate}','m_{true}',Location='best')
frame = getframe(gcf);
```

```
writeVideo(v,frame);
end

for i=1:N
xi = x(i)/t(j);
[Qest(:,i),hs,us] = forestclaw(m(1,:),m(3,:),xi);
end

% chi-square
chi_s = zeros(3,1); ad = d';
for k = 1:N
chi_s = chi_s + (Qest(:,k) - ad(:,k)).^2;
end

% pvalue
dof = N - 9; %degrees of freedom
p = 1 - chi2cdf(chi_s,dof);

% plotting data
figure(2)
if mq == 1
plot(Qest(mq,:),'k'); hold on
plot(Q(mq,:),'r'); hold off
legend('h_{estimate}','h_{true}',Location='best')
title('Height field');
ylim([hr-0.5,hl+0.5]);
ylabel('height');xlabel('x')
frame = getframe(gcf);
writeVideo(v,frame);
elseif mq == 2
plot(Qest(mq,:),'k'); hold on
plot(Q(mq,:),'r'); hold off
legend('u_{estimate}','u_{true}',Location='best')
title('velocity field')
ylabel('velocity');xlabel('x')
frame = getframe(gcf);
writeVideo(v,frame);
elseif mq == 3
plot(Qest(mq,:),'k'); hold on
plot(Q(mq,:),'r'); hold off
legend('hu_{estimate}','hu_{true}',Location='best')
title('momentum field')
ylabel('Momentum');xlabel('x')
frame = getframe(gcf);
writeVideo(v,frame);
else
subplot(2,2,1)
plot(Qest(1,:),'k'); hold on
plot(Q(1,:),'r'); hold off
legend('h_{estimate}','h_{true}',Location='best')
```

```
title('Height field')
ylim([hr-0.25,hl+0.25])
ylabel('height');xlabel('x')
subplot(2,2,2);
plot(Qest(2,:),'k'); hold on
plot(Q(2,:),'r'); hold off
legend('u_{estimate}','u_{true}',Location='best')
title('Velocity field')
ylabel('velocity');xlabel('x')
subplot(2,2,[3,4]);
plot(Qest(3,:),'k'); hold on
plot(Q(3,:),'r'); hold off
ylim([-1.1,0])
title('Momentum field')
ylabel('Momentum');xlabel('x')
legend('hu_{estimate}','hu_{true}',Location='best')
frame = getframe(gcf);
writeVideo(v,frame);

end


%Covariance Matrix
J = Jacobian(m);
C = inv(J'*J);

%confidence interval
za = 1.96; % 95% confidence interval

% first parameter
s1 = sqrt(C(1,1)); % standard deviation
s2 = sqrt(C(2,2));
s3 = sqrt(C(3,3));

% confidence intervals
c1 = m(:,1)- za*s1;
c2 = m(:,1) + za*s1;
c11 = m(:,2)- za*s2;
c22 = m(:,2) + za*s2;
c13 = m(:,3)- za*s3;
c33 = m(:,3) + za*s3;

%Correlation matrix
rho11 = C(1,1)/sqrt(C(1,1)*C(1,1));
rho22 = C(2,2)/sqrt(C(2,2)*C(2,2));
rho33 = C(3,3)/sqrt(C(3,3)*C(3,3));
rho12 = C(1,2)/sqrt(C(1,1)*C(2,2));
rho13 = C(1,3)/sqrt(C(1,1)*C(3,3));
rho23 = C(2,3)/sqrt(C(2,2)*C(3,3));
Correlation_matrix = [rho11 rho12 rho13;rho12 rho22 rho23;...
```

```
rho13 rho23 rho33];


end
close(v);

disp(['chi-square obs = [',num2str(chi_s'),']'])
disp(['pvalue = [',num2str(p'),']'])
L2norm = norm(mtrue - m,2)
mtrue
covariance_matrix = C
cofidence_interval_height = [c1 c2]
cofidence_interval_velocity = [c11 c22]
cofidence_interval_Momentum = [c13 c33]

Correlation_matrix = Correlation_matrix


%Linearised ellipsoid
Delta = chi2inv(0.95,3); %Delta2

figure(4)
subplot(2,2,1)
plot_ellipse(Delta,C(1:2,1:2),m,1,2);
grid on
title('Error Ellipsoid for h and u')
legend('','left','','middle','','right',Location='best')
xlabel('h'); ylabel('u')
subplot(2,2,2)
C1 = [C(2,2) C(2,3); C(3,2) C(3,3)];
plot_ellipse(Delta,C1,m,2,3);
grid on
title('Error Ellipsoid for u and hu')
legend('','left','','middle','','right',Location='best')
xlabel('u'); ylabel('hu')
subplot(2,2,[3,4]);
C2 = [C(1,1) C(1,3); C(3,1) C(3,3)];
plot_ellipse(Delta,C2,m,1,3);
grid on
title('Error Ellipsoid for h and hu')
legend('','left','','middle','','right',Location='best')
xlabel('h'); ylabel('hu')

Corr = corrcoef(C);

figure(7)
clf
colormap('gray')

imagesc(Corr)
```

```
set(colorbar,'Fontsize',18);
xlabel('j')
ylabel('i')
title('Correlation Matrix')
end
```

5.5 Main script

```
%%==================================================================
% Boise State University
% Author: Brian KYANJO
% supervised by: Prof. Jodi Mead
% class: Inverse Methods
% Date: March 17th, 2022
% final project
% main script
%%===================================================================

clc
clear
close all

global g; global N;
global sig; global h;
global t; global x;
global M;
warning('off','all')

% problem
hl = 2;                          % left depth
hr = 1;                          % right depth
ul = -0.5;                       % left velocity
ur = -0.5;                       % right velocity

% Spatial domain
ax = -5;
bx = 5;
ay = -2;
by = 4;
meqn = 2;                       % Number of equations in the system


g = 1;                          % Gravity

to = 0;                         % initial time
Tfinal = 1;                     % final time

ql = [hl; ul; hl*ul];           % left conservation variable
```

```
qr = [hr; ur; hr*ur];           % right consrvation variable
qm = [(hl+hr)/2 (ul+ur)/2 ...   % intermediate initial state
(hl*ul+hr*ur)/2];

N = 64;                         % Number of spartial steps

dx = (bx - ax)/N;               % spartial step size

cfl = 0.9;                      % cfl number

a = 1.5;                        % maximum velocity

dt_est = cfl*dx/a;

M = (floor(Tfinal/dt_est) + 1); % number of time steps

dt = Tfinal/(M);                % temporal step size

t = linspace(to,Tfinal,M);      % temporal domain

xe = linspace(ax,bx,N+1);       % edge locations
x = xe(1:end-1) + dx/2;         % Cell-center locations

mq = 4;                         % 1-height,
% 2-velocity,
% 3-momentum,
% 4-all fields
h = 0.9;

sig = 1e-3;                     % standard deviation

L0 = eye(3);          % generate roughening matrices
L1 = get_l_rough(3,1);
L2 = get_l_rough(3,2);


qll = [hl ul hl*ul];            % left conservation variable
qrr = [hr ur hr*ur];            % right conservation variable
xi = x(1)/t(2);                 % initial speed
[Q(:,1),hs,us] = forestclaw(ql,qr,xi); % initial conservation variable
qmm = [hs us hs*us];                   % intermediate state
%mo = [qll;qm;qrr];
mtrue = [qll; qmm; qrr];               % true parameters

mo = 1.5*mtrue;
main_function(mtrue,mo,h,L0,x,ql,qr,M,t,mq) %calling the main function script
```