Computational and Variational Methods for Inverse Problems - Homework 1 Solution

Mohammad Afzal Shadab (ms82697)

mashadab@utexas.edu

Due Monday, September 20, 2021

Problem 1

Considering the inverse problem for the given 1D heat equation (parabolic PDE)

$$\begin{cases} \frac{\partial u}{\partial t} - k\frac{\partial^2 u}{\partial x^2} = 0 & 0 < x < L, 0 < t \leq T \\ u(x,0) = m(x) & 0 < x < L \\ u(0,t) = u(L,t) = 0 & 0 < t \leq T \end{cases} \tag{0.1}$$

We are inverting for the initial condition (temperature field) $u(x,0) = m(x)$ given full observation of the solution (final temperature field) at some time T. Thus the parameter-to-observation (P2O) map is

$$\mathcal{F}(m) := u(x,T) \tag{0.2}$$

The problem is discretized using finite differences: first order implicit Euler in time and second order central difference in space with time step $\Delta t = T/n_t$ and mesh size $h = L/n_x$ respectively. The resulting discrete parameter-to-observable map $\mathbf{F}$ takes the form

$$\mathbf{F} = (\mathbf{I} + \Delta t\mathbf{K})^{-n_t} \tag{0.3}$$

where the stiffness matrix $\mathbf{K}$ is

$$\mathbf{K} = \frac{k}{h^2}\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 2 \end{bmatrix} \tag{0.4}$$

(a) Eigenvalues λ_i and eigenfunctions $v_i(x)$ of the continuous operator $\mathcal{F}$

To solve the PDE (0.1), we can use *separation of variables.* Considering the temperature field solution take the following form

$$u(x,t) = X(x)T(t) \tag{0.5}$$

$$\Rightarrow \frac{\partial u}{\partial t} = X(x)\frac{\partial T(t)}{\partial t} \tag{0.6}$$

$$\Rightarrow \frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 X(x)}{\partial x^2}T(t) \tag{0.7}$$

Plugging them in (0.1) and dividing by $u(x,t)$ from (0.5) gives two independent ODEs,

$$\frac{T'(t)}{kT(t)} = \frac{X''(x)}{X(x)} = -C \text{ (a constant)}, C > 0 \tag{0.8}$$

since LHS is a function of t and RHS of x, they should be equal to a constant. By solving the ODEs (0.8) separately, we get

$$T(t) = A^* \exp(-Ckt) \tag{0.9}$$

$$X(x) = A^\dagger \sin(\sqrt{C}x) + B^\dagger \cos(\sqrt{C}x) \tag{0.10}$$

where $A^*, A^\dagger$ and $B^\dagger$ are the constants of integration. Using the boundary conditions (0.1) for $u(x = 0, t) = u(x = L, t) = 0$, we get

$$B^\dagger = 0, \quad \sqrt{C}L = i\pi, \quad i = 1, 2, ... \tag{0.11}$$

where i is the number of mode. Therefore,

$$T(t) = A_i^* \exp\left(-\left(\frac{\pi i}{L}\right)^2 kt\right) \tag{0.12}$$

$$X(x) = A_i^\dagger \sin\left(\frac{\pi i}{L}\right) \tag{0.13}$$

Finally, the general solution is

$$u(x,t) = \sum_{i=1}^{\infty} A_i \sin\left(\pi i \frac{x}{L}\right) \exp\left(-\left(\frac{\pi i}{L}\right)^2 kt\right) \tag{0.14}$$

where $A_i = A_i^* \times A_i^\dagger$. Now, using the initial condition (0.1) with the full solution,

$$u(x, 0) = m(x) \qquad 0 < x < L \tag{0.15}$$

$$\Rightarrow \sum_{i=1}^{\infty} A_i \sin\left(\pi i \frac{x}{L}\right) = m(x) \quad \text{for each mode } i \tag{0.16}$$

$$\Rightarrow A_i \sin\left(\pi i \frac{x}{L}\right) = m(x) \tag{0.17}$$

For the initial condition we note that,

$$u(x, 0) = m(x) = A_i \sin\left(\pi i \frac{x}{L}\right) = v_i(x) = \sqrt{\frac{2}{L}} \sin\left(\pi i \frac{x}{L}\right) \tag{0.18}$$

Then the particular solution becomes

$$u(x,t) = \sum_{i=1}^{\infty} \sqrt{\frac{2}{L}} \sin\left(\pi i \frac{x}{L}\right) \exp\left(-\left(\frac{\pi i}{L}\right)^2 kt\right) \tag{0.19}$$

So, the P2O map (0.2) can be reconsidered for each mode i

$$\mathcal{F}v_i(x) = u_i(x, T) \tag{0.20}$$

$$= \sqrt{\frac{2}{L}} \sin\left(\pi i \frac{x}{L}\right) \exp\left(-\left(\frac{\pi i}{L}\right)^2 kT\right) \tag{0.21}$$

$$= \exp\left(-\left(\frac{\pi i}{L}\right)^2 kT\right) v_i(x) \tag{0.22}$$

$$\mathcal{F}v_i(x) = \lambda_i v_i(x) \tag{0.23}$$

Therefore this suggests that the eigenfunction $v_i(x)$ and eigenvalues $\lambda(T)$ of the P2O operator $\mathcal{F}$ are respectively

$$v_i(x) = \sqrt{\frac{2}{L}} \sin\left(\pi i \frac{x}{L}\right) \tag{0.24}$$

and

$$\lambda_i(x) = \exp\left(-\left(\frac{i\pi}{L}\right)^2 kT\right) \tag{0.25}$$

(b) Eigenvalues $\lambda_{D,i}$ and eigenfunctions $\mathbf{v}_{D,i}(x)$ of the discrete P2O operator F

Given, the eigenvalues of **K**

$$\mu_i = k\frac{4}{h^2}\sin^2\left(\frac{\pi i}{2n_x}\right), \quad i = 1, 2, ..., n_x - 1 \tag{0.26}$$

and the corresponding eigenvector $\mathbf{u}_i$ is given by

$$[\mathbf{u}_i]_j = \sqrt{\frac{2}{L}}\sin\left(\pi i\frac{jh}{L}\right) \tag{0.27}$$

As K is symmetric positive definite, its the eigen value decomposition is,

$$\mathbf{K} = \mathbf{U}\ \ \mathbf{MU}\ \ \mathbf{U}^T \tag{0.28}$$

where $\mathbf{MU} = \text{diag}[\mu_1, \mu_2, \ldots, \mu_{n_x-1}]$ and $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{n_x-1}]$. Therefore,

$$\begin{aligned}\mathbf{I} + \Delta t\mathbf{K} &= \mathbf{I} + \Delta t\mathbf{U}\ \ \mathbf{MU}\ \ \mathbf{U}^T && (0.29)\\ &= \mathbf{U}\ \ \mathbf{U}^T + \mathbf{U}\ \ \Delta t\mathbf{MU}\ \ \mathbf{U}^T && (0.30)\\ &= \mathbf{U}\ \ (\mathbf{I} + \Delta t\mathbf{MU}))\ \ \mathbf{U}^T && (0.31)\\ & && (0.32)\end{aligned}$$

Therefore,

$$(\mathbf{I} + \Delta t\mathbf{K})^{-n_t} = \mathbf{U}\ \ (\mathbf{I} + \Delta t\mathbf{MU}))^{-n_t}\ \ \mathbf{U}^T \tag{0.33}$$

So, the eigenvalues $\lambda_{D,i}$ and eigenvectors $\mathbf{v}_{D,i}$ of the discrete P2O operator **F** are respectively

$$\lambda_{D,i} = (1 + \Delta t\mu_i)^{-n_t} \tag{0.34}$$

$$[\mathbf{v}_{D,i}]_j = \sqrt{\frac{2}{L}}\sin\left(\pi i\frac{jh}{L}\right) \tag{0.35}$$

where $i, j = 1, 2, \ldots n_x - 1$.

(c) Decay of eigen values for continuous P2O operator $\mathcal{F}$

For $T = 1$, $L = 1$, $n_x = 100$, and $n_t = 100$ and $k = [10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1]$ the results are shown in Figure 1. It can be observed that although all eigen values decay with the number of mode i, the decay is much rapid for the case of higher diffusivity k. The reason being that the information is lost much rapidly due to higher diffusivity. For less value of k, the loss of information is much less for high-wave number modes $i > 50$.

(d) Decay of eigen values for discrete P2O operator F with different spatio-temporal discretization

For $(n_x, n_t) = (20, 20), (40, 40), (80, 80)$ and $(160, 160)$ the results are shown in Figure 2. If we increase the resolution, the analytical (continuous) curve is traversed even further. The spectrums are truncated to the largest mode $i = n_x - 1$ having the smallest eigen value for the corresponding choice of (n_x, n_t). The discrete eigenvalues are almost always higher than the analytical (continuous) eigenvalues. This effect is much prevalent for coarse discretization. At large i, the discrete eigenvalues deviate significantly. The discretization produces a regularization effect.

Problem 2

Considering the inverse problem for the following Poisson's equation (elliptic PDE) for axial displacement $u(x)$

$$\begin{cases}-k\frac{\partial^2 u}{\partial x^2} = m(x), & 0 < x < L\\ u(0) = u(L) = 0\end{cases} \tag{0.36}$$

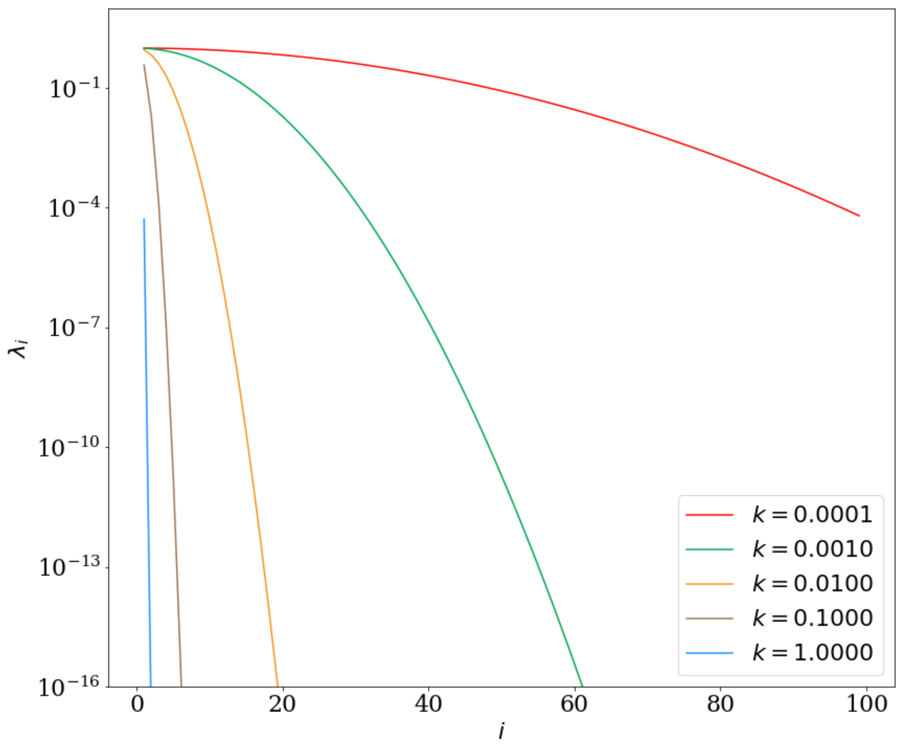

Figure 1: Eigenvalues of the continuous parameter-to-observable map, plotted as a function of i for $k = [0.0001, 0.001, 0.01, 0.1, 1.0]$

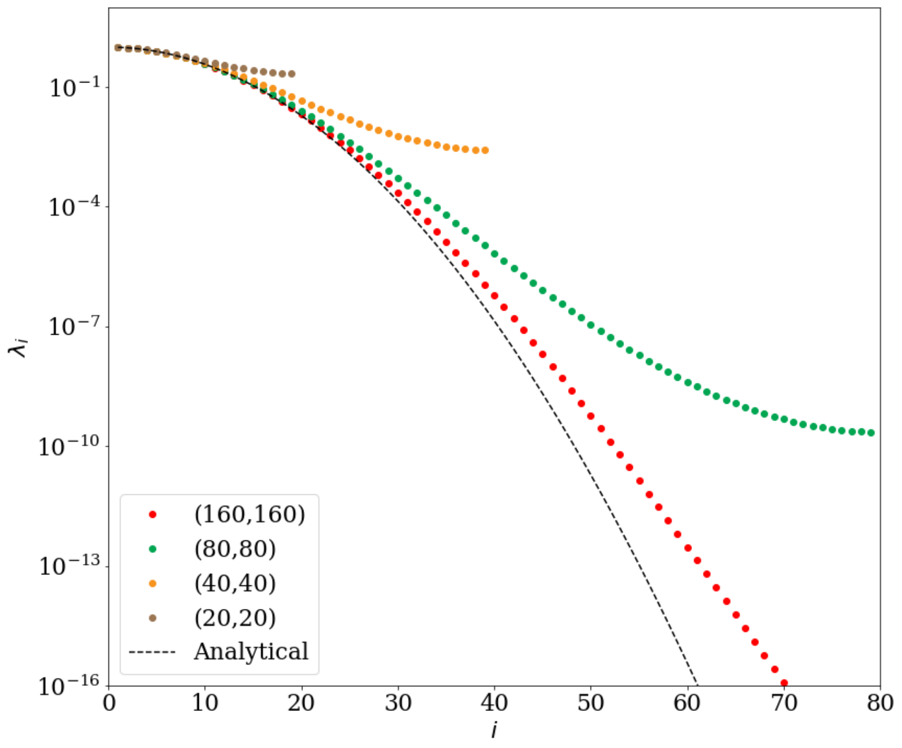

Figure 2: Eigenvalues of the discrete parameter-to-observable map, plotted as a function of i for discretization levels $(n_x, n_t) = (20, 20), (40, 40), (80, 80), (160, 160)$. Eigenvalues of the continuous operator are also plotted for comparison.

Now, the inversion parameter is the distributed forcing term $m(x)$ from the observations of axial displacement $u(x)$. The parameter-to-observable map is then for a given $m(x)$

$$\mathcal{F}(m) := u(x) \tag{0.37}$$

The equation (0.36) can be rewritten as

$$\begin{cases} \mathcal{L}u = m(x), & 0 < x < L \\ u(0) = u(L) = 0 \end{cases} \tag{0.38}$$

where $\mathcal{L}$ is the $-k\frac{\partial^2}{\partial x^2}$ operator. From (0.38)

$$u(x) = \mathcal{L}^{-1}(m(x)) \tag{0.39}$$

$$\Rightarrow \mathcal{F}(m(x)) = u(x) \tag{0.40}$$

where the operator $\mathcal{L}^{-1} \equiv \mathcal{F}$.

(a) Eigenvalues λ_i and eigenfunctions $v_i(x)$ of the continuous operator $\mathcal{F}$

Let's find the eigenvalues of the operator $\mathcal{L}$ using the hint for eigen functions,

$$v_i(x) = \sqrt{\frac{2}{L}} \sin\left(i\pi\frac{x}{L}\right), \quad i = 1, 2, \ldots \tag{0.41}$$

$$\mathcal{L}v_i(x) = -k\frac{\partial^2 v_i}{\partial x^2} = -k\left(\frac{i\pi}{L}\right)^2 \sqrt{\frac{2}{L}} \sin\left(i\pi\frac{x}{L}\right) = -k\left(\frac{i\pi}{L}\right)^2 v_i(x) \tag{0.42}$$

$$= \mu_i v_i(x) \tag{0.43}$$

$$\Rightarrow \mu_i = k\left(\frac{i\pi}{L}\right)^2 \tag{0.44}$$

Now, the eigenfunctions for the parameter-to-observable operator $\mathcal{L}^{-1} \equiv \mathcal{F}$ will be the same. And the eigen values will be the inverse of (0.44),

$$\lambda_i = \frac{1}{\mu_i} = \frac{1}{k}\left(\frac{L}{i\pi}\right)^2 \tag{0.45}$$

(b) Eigenvalues of the discrete P2O operator F

Again, the PDE is discretized using second order central differencing in space. This yields the system of equations

$$\mathbf{K}\mathbf{u} = \mathbf{m} \tag{0.46}$$

with $\mathbf{K}$ defined earlier in (0.4). The we have

$$\mathbf{u} = \mathbf{F}^{-1}\mathbf{m} = \mathbf{F}\mathbf{m} \tag{0.47}$$

such that $\mathbf{F} = \mathbf{K}^{-1}$. This time, the operator $\mathbf{F}$ has the same eigenvectors $\mathbf{u}_i$ with eigenvalues λ_i that are inverses of the eigenvalues of $\mathbf{K}$, i.e., μ_i given in (0.26) and (0.27) respectively.

$$\lambda_i = \frac{1}{\mu_i} = \frac{h^2}{4k\sin^2\left(\frac{\pi i}{2n_x}\right)}$$

$$[\mathbf{u}_i]_j = \sqrt{\frac{2}{L}} \sin\left(\pi i\frac{jh}{l}\right)$$

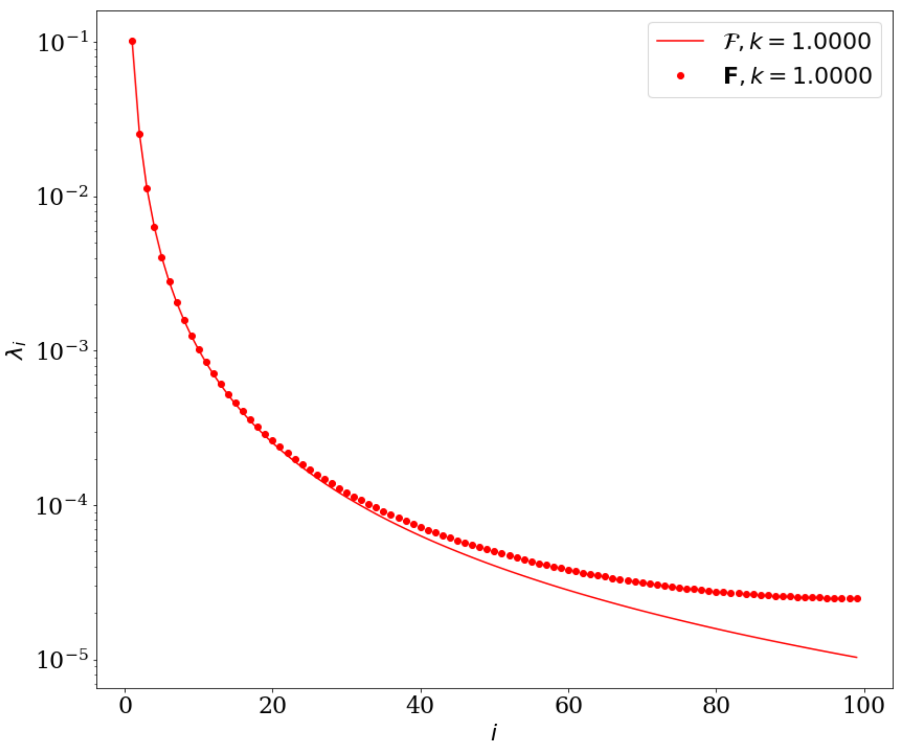

Figure 3: Variation of the eigenvalues of discrete $\mathbf{F}$ and continuous $\mathcal{F}$ parameter-to-observable operators as a function of mode number i.

where $i, j = 1, 2, \dots n_x - 1$.

Plot 3 shows the eigenvalue spectrum as a function of modes i with $k = L = 1$ and $n_x = 100$ for the discrete operator. It can be observed that the eigenvalues decay rapidly with the number of modes i. Fither the eigenvalues mathc upto $i = 40$, where the deviation starts to become significant. Since the eigenvalue decay is i^{-2} for the Poisson problem compared with $\exp(i^{-2})$ of the heat equation, the former is still a less ill-posed problem than the latter.

c) Inverse problem's direct solution without regularization

We try to solve the inverse problem directly without regularization, from (0.47)

$$\mathbf{m} = \mathbf{F}^{-1}\mathbf{d} \tag{0.48}$$

The true body force is given as

$$m_{true} = \max(0, 1 - |1 - 4x| + 100x^{10}(1 - x)^2) \tag{0.49}$$

Synthetic data is generated by adding i.i.d normally distibuted noise $\boldsymbol{\eta} \sim \mathcal{N}(0, \sigma^2\mathbf{I})$ with standard deviation $\sigma = 10^{-4}$. The resulting noisy observation of the displacement field is given as

$$\mathbf{d} = \mathbf{F}\mathbf{m} + \boldsymbol{\eta} \tag{0.50}$$

First, we solved the inverse problem for variation in the standard deviation $\sigma = [10^{-6}, 10^{-4}, 10^{-2}]$. The stifness $k = 1$ and the number of cells are kept fixed at $n_x = 80$. Figure 4 shows the synthetic observations formed after adding the Gaussian noise. Figure 5 shows the inverted distributed load field $m(x)$ inside the domain. Lastly, figure 6 illustrated the predicted displacements for the inverted $m(x)$. It can be easily observed that increasing the noise in the synthetic data increases the noise amplitude in the inverted $m(x)$. However, the predicted displacement field (observable) and noisy observed data overlap each other indicating that the noise in prediction of $m(x)$ is due to overfitting to the noise. An

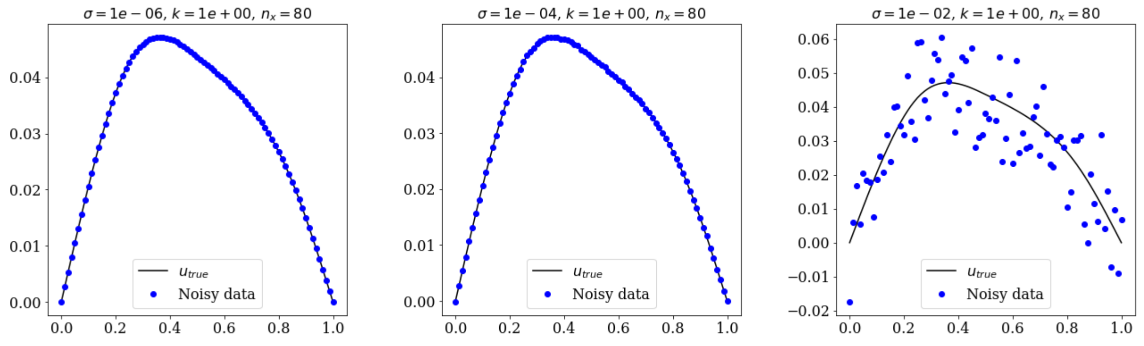

Figure 4: Synthetic observations for $k = 1$, $n_x = 80$ and $\sigma = [10^{-6}, 10^{-4}, 10^{-2}]$.

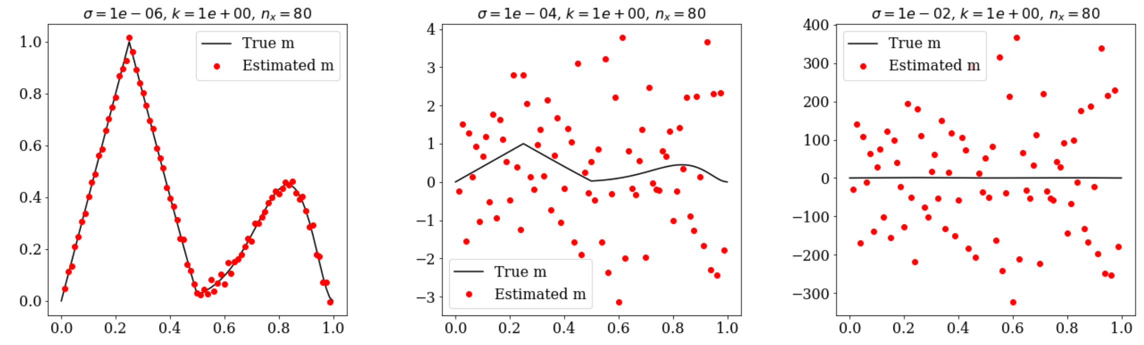

Figure 5: Results of inversion for $k = 1$, $n_x = 80$ and $\sigma = [10^{-6}, 10^{-4}, 10^{-2}]$.

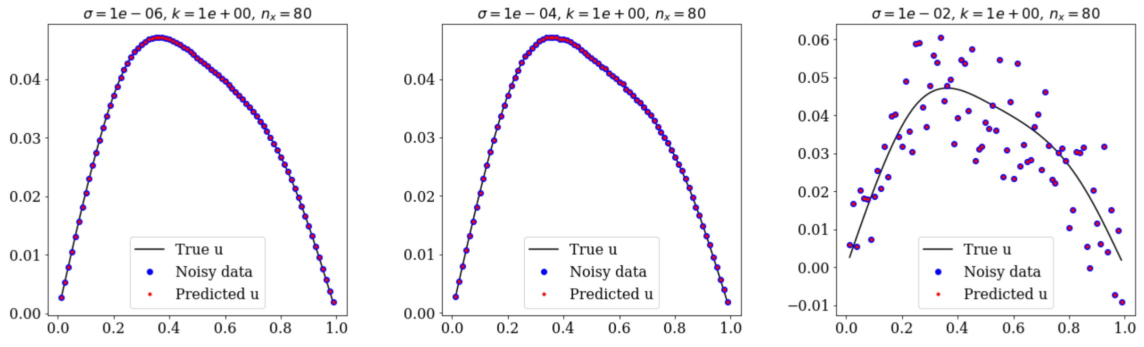

Figure 6: Predicted displacement field for the inverted $m(x)$ for $k = 1$, $n_x = 80$ and $\sigma = [10^{-6}, 10^{-4}, 10^{-2}]$.

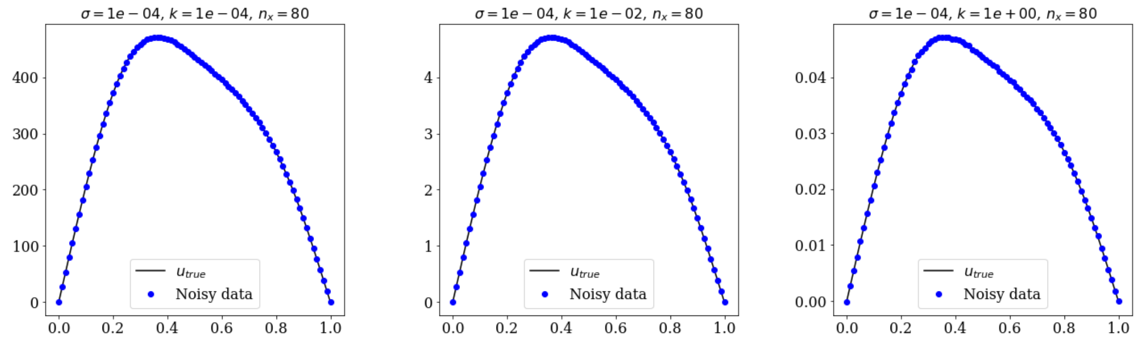

Figure 7: Synthetic observations for $\sigma = 10^{-4}$, $n_x = 80$ and $k = [10^{-4}, 10^{-2}, 1]$.

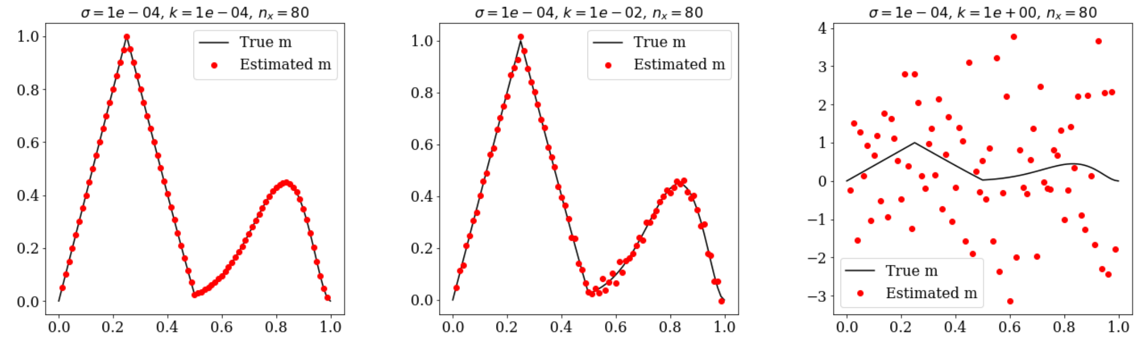

Figure 8: Results of inversion for $\sigma = 10^{-4}$, $n_x = 80$ and $k = [10^{-4}, 10^{-2}, 1]$.

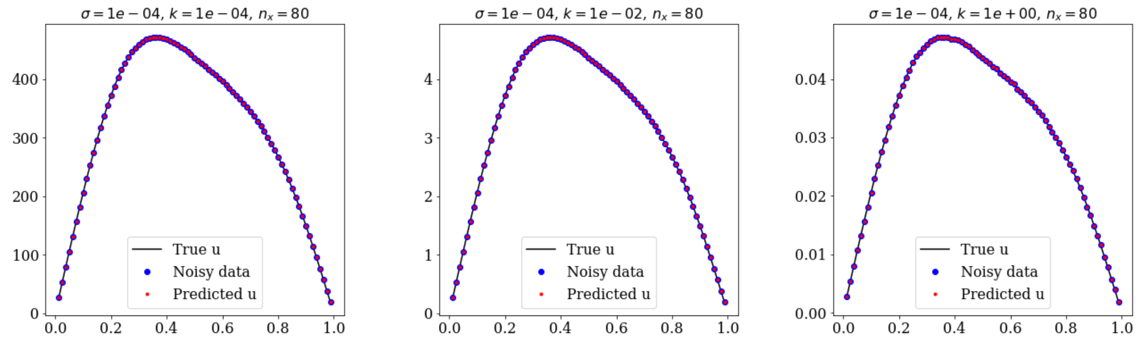

Figure 9: Predicted displacement field for the inverted $m(x)$ for $\sigma = 10^{-4}$, $n_x = 80$ and $k = [10^{-4}, 10^{-2}, 1]$.

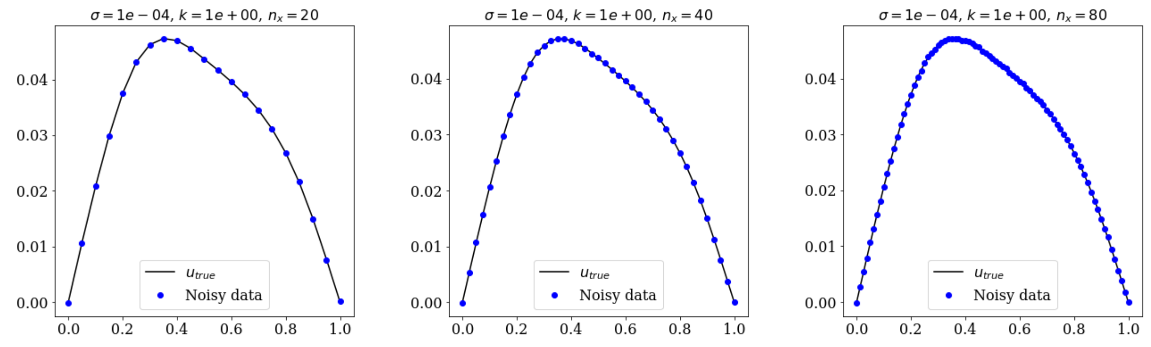

Figure 10: Synthetic observations for $\sigma = 10^{-4}$, $k = 1$ and $n_x = [20, 40, 80]$.

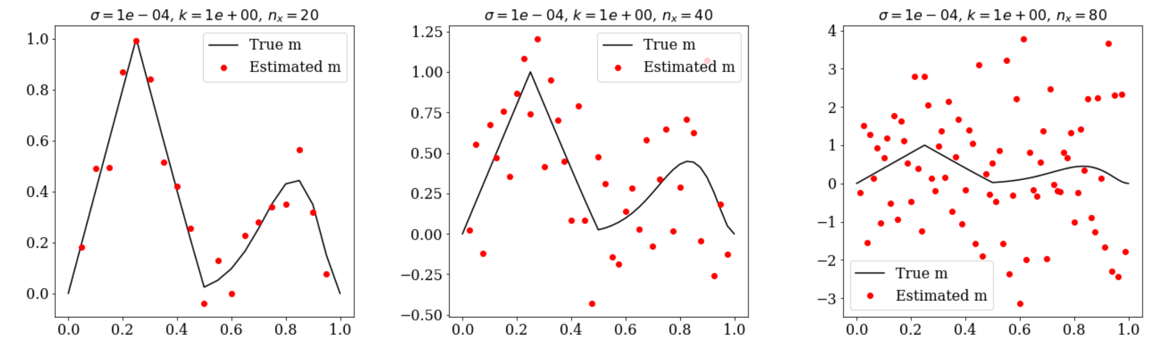

Figure 11: Results of inversion for $\sigma = 10^{-4}$, $k = 1$ and $n_x = [20, 40, 80]$.

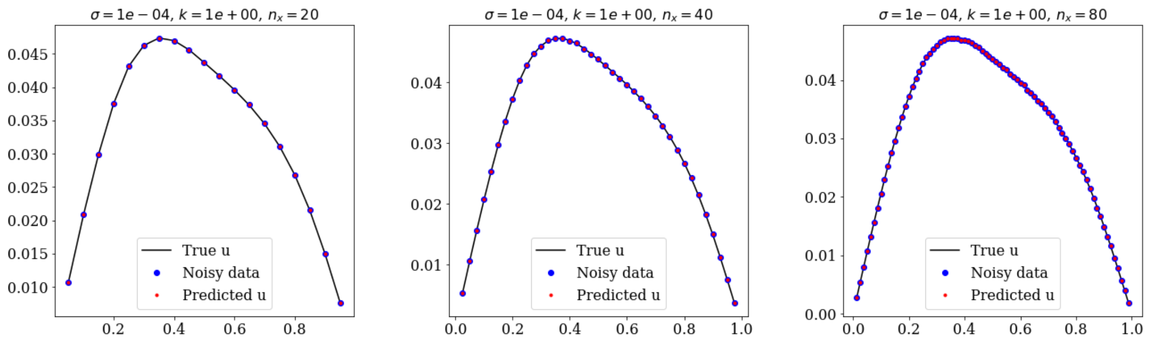

Figure 12: Predicted displacement field for the inverted $m(x)$ for $\sigma = 10^{-4}$, $k = 1$ and $n_x = [20, 40, 80]$.

improvement in accuracy for reducing the noise down to $\sigma = 10^{-6}$ illustrates the ill-posedness of the inverse problem when solved without regularization.

Next, we consider the effect of stiffness $k = [10^{-4}, 10^{-2}, 1]$. The standard deviation of the noise in the true data is $\sigma = 10^{-4}$ and the number of cells are kept fixed at $n_x = 80$. Figure 7 shows the synthetic observations formed after adding the Gaussian noise. Figure 8 shows the inverted distributed load field $m(x)$ inside the domain. Lastly, figure 9 illustrates the predicted displacements for the inverted $m(x)$. It can be easily observed that increasing the stiffness k for the similarly noisy synthetic data increases the noise amplitude in the inverted $m(x)$, or introduces instabilities. The reason being as stiffness k increases, the displacements are generally smaller, which decreases the signal-to-noise ratio. Second reason is that the eigenvalues of the discrete P2O operator $\mathbf{F}$ decrease with increasing k, thus decreases the lower bound on the eigenvalues for $n_x = 80$. The predicted displacement field (observable) and noisy observed data overlap each other, indicating the over fitting to the noise.

Lastly, we consider the effect of discretization $n_x = [20, 40, 80]$. The standard deviation of the noise in the true data is $\sigma = 10^{-4}$ and the stiffness $k = 1$. Figure 10 shows the synthetic observations formed after adding the Gaussian noise. Figure 11 shows the inverted distributed load field $m(x)$ inside the domain. Lastly, figure 12 illustrates the predicted displacements for the inverted $m(x)$. It can be easily observed that despite the data points are sparse, the inverted results are the most accurate for $n_x = 20$. This indicated the regularization due to low-resolution discretization as the eigenvalues upto $n_x - 1$ are kept. So, there is less matching of the eigenvectors of higher modes with those of noise.

(d) Solving inverse problem using Tikhanov regularization

Solving the inverse problem with Tikhonov regularization as an optimization problem

$$\min_{m} \|\mathbf{F}\mathbf{m} - \mathbf{d}\|_2^2 + \alpha\|\mathbf{m}\|_2^2 \tag{0.51}$$

which has solution

$$\mathbf{m} = (\mathbf{F}^T\mathbf{F} + \alpha\mathbf{I})^{-1}(\mathbf{F}^T\mathbf{d}) \tag{0.52}$$

We fix length $L = 1$, stiffness $k = 1$, discretization $n_x = 200$ and noise with $\sigma = 10^{-4}$. Varying the regularization parameter $\alpha = [10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$. Figure 13 shows the regularized inverted solutions $m(x)$ along with the true solution m_{true}. A visual inspection shows the optimal value of α to be in between 10^{-6} and 10^{-5}. Decreasing α causes instability due to overfitting the noise whereas the regularization smoothens out the solution to zero value.

(e) Optimal value of Tikhanov regularization parameter α using L-curve criterion

Plotting $\|\mathbf{F}\mathbf{m}_\alpha - \mathbf{d}\|$ against $\|\mathbf{m}\|$ for the L-curve as shown in Figure 14. It is not clear where the optimal α lies but the curve starts to bend steeply near $\alpha = 10^{-6} - 10^{-7}$.

(f) Optimal value of Tikhanov regularization parameter α using Morozov's criterion

For the Morozov criterion, with an estimate of the noise $\delta = \|\boldsymbol{\eta}\|$, the data misfit $\|\mathbf{F}\mathbf{m} - \mathbf{d}\|$ is plotted against against the regularization parameter α and find largest α such that $\|\mathbf{F}\mathbf{m}_\alpha - \mathbf{d}\| \leq \delta$. The results are shown in Figure 15. The value of $\alpha = 10^{-6}$ is the value where the two curves cross and hence is the optimal value. The noise value δ is calculated using $\|(\mathbf{d} - \mathbf{u}_{true})\|_2$ which should be bounded by $\sqrt{n_x}\sigma = \sqrt{200} \times 10^{-4} = 1.414 \times 10^{-3}$.

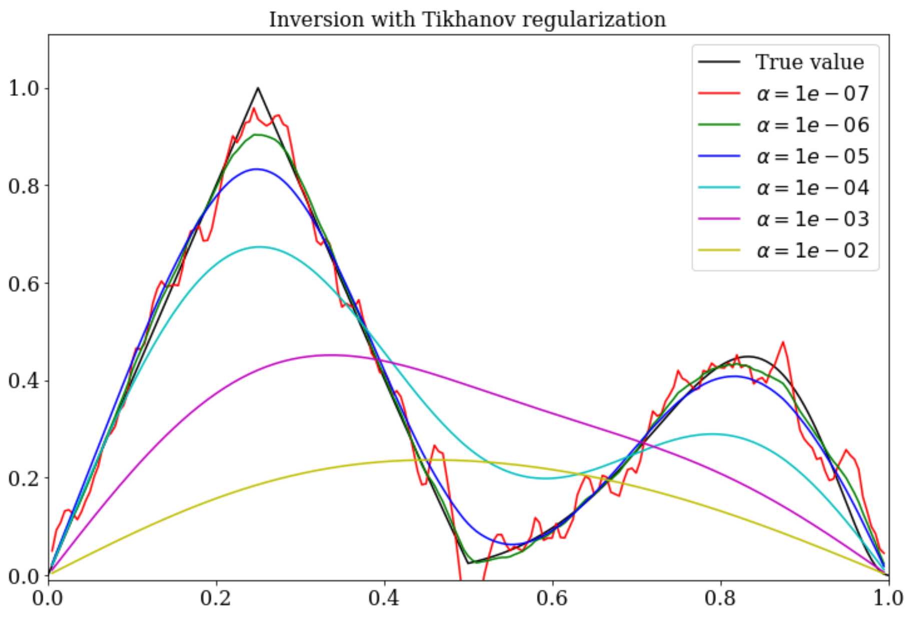

Figure 13: Regularized solutions m_α for $\alpha = [10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$.

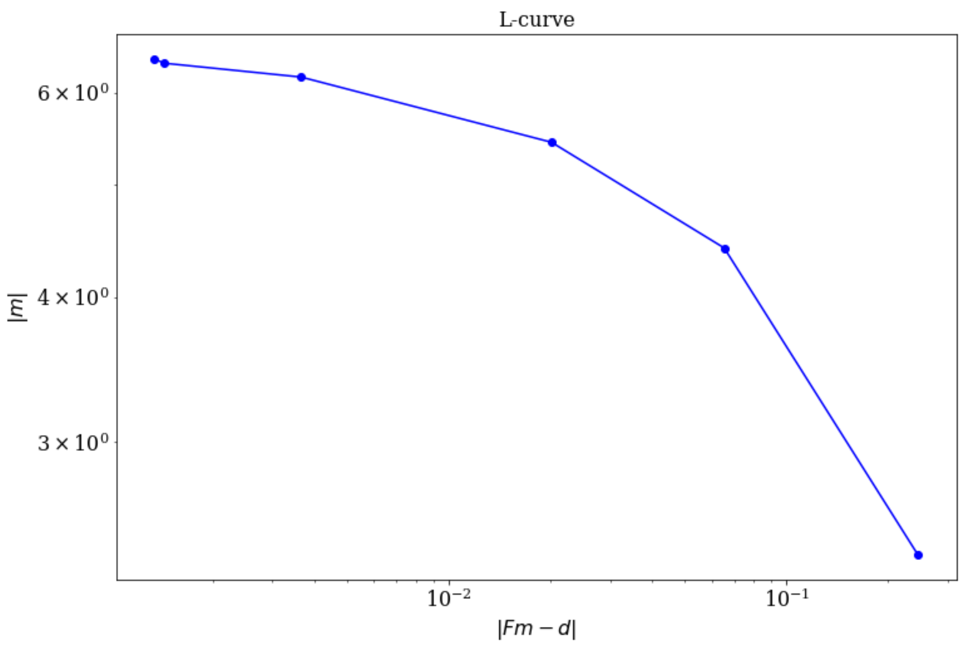

Figure 14: L-curve for $\alpha = [10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$ from left to right.

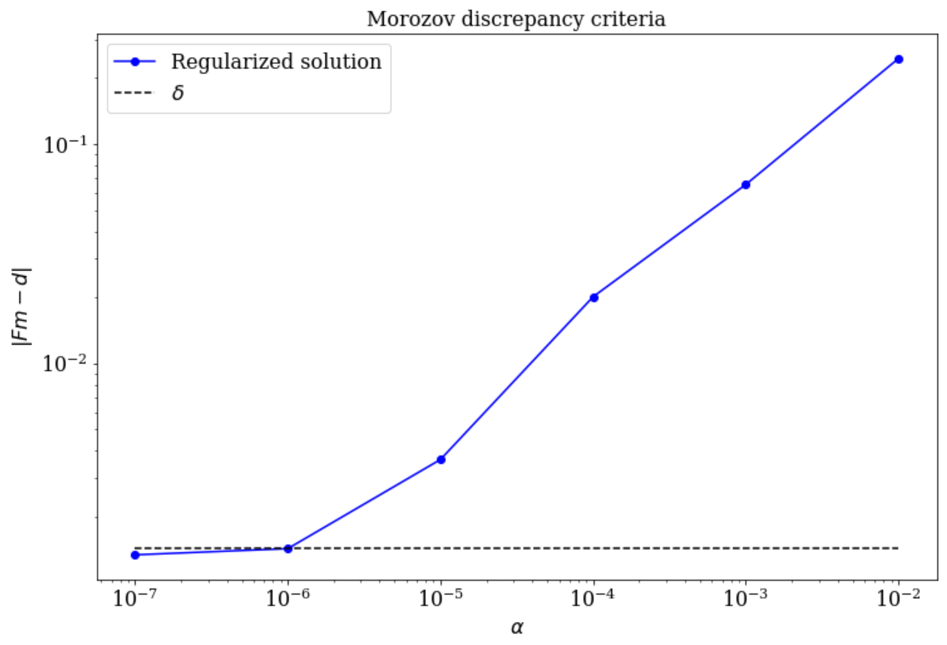

Figure 15: Data misfit for $\alpha = [10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$ for Morozov discrepancy criteria with noise level δ indicated.

(g) L$_2$ norm of the error in reconstruction $\|\mathbf{m}_{true} - \mathbf{m}_\alpha\|$ and optimal values of α

Figure 16 shows the error in reconstruction with an error minimum at $\alpha = 10^{-6}$. From the summary table 1, the optimal value of α is predicted accurately by Morozov's criterion. Eyeball and L-curve criteria also performed fairly well.

Table 1: Optimal α values using different criteria.

Criteria	Optimal α
Eyeball Norm	$10^{-6} - 10^{-5}$
L-curve	$10^{-7} - 10^{-6}$
Morozov	10^{-6}
Error in reconstruction	10^{-6}

Problem 3

Consider the 1D wave equation (hyperbolic PDE) with the forward problem given $m(x)$ solving

$$\begin{cases} \frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0 & 0 < x < L, 0 < t \leq T \\ u(x,0) = m(x) & 0 < x < L \\ \frac{\partial u}{\partial t}(x,0) = 0 & 0 < x < L \\ u(0,t) = u(L,t) = 0 & 0 < t \leq T \end{cases} \tag{0.53}$$

for the displacement field $u(x,T)$. The stability of inverse problem for the initial condition $m(x)$ through observing the cable displacement $u(x,T)$. Therefore the parameter-to-observable (P2O) map is $\mathcal{F}(m) := u(x,T)$.

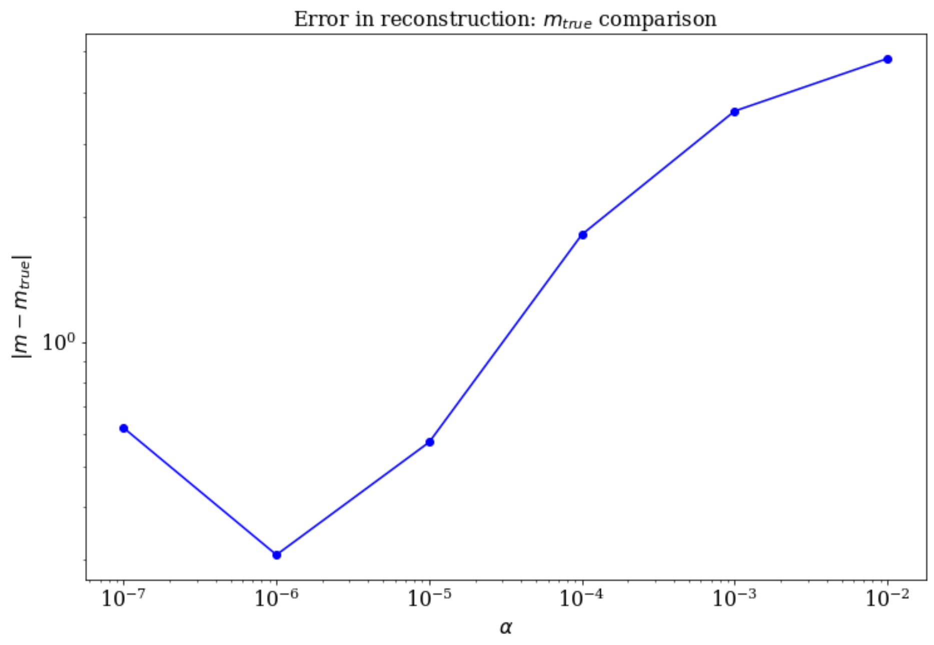

Figure 16: L_2 norm between the reconstruction and true value of m for $\alpha = [10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$.

(a) Eigenvalues and eigenfunctions of the continuous P2O operator $\mathcal{F}$

Since this is the case of wave propagation, the characteristics travels with the wave speed c. Again using the separation of variables with homogeneous Dirichlet BC and zero initial velocity, we get

$$u(x,t) = \sum_{i=1}^{\infty} A_i \sin\left(i\pi\frac{x}{L}\right)\cos\left(i\pi\frac{ct}{L}\right) \tag{0.54}$$

Similar to the previous problems considering the initial displacement to be $m(x) = \sqrt{\frac{2}{L}}\sin\left(i\pi\frac{x}{L}\right)$, we get the particular solution for the mode i,

$$u(x,T) = \sqrt{\frac{2}{L}}\sin\left(i\pi\frac{x}{L}\right)\cos\left(i\pi\frac{cT}{L}\right) \tag{0.55}$$

Therefore,

$$u(x,T) = \cos\left(i\pi\frac{cT}{L}\right)m(x) = \mathcal{F}m(x) \tag{0.56}$$

So, the eigenvalues and eigenfunctions of the P2O operator $\mathcal{F}$ are

$$\lambda_i = \cos\left(i\pi\frac{cT}{L}\right) \tag{0.57}$$

$$v_i = \sqrt{\frac{2}{L}}\sin\left(i\pi\frac{x}{L}\right) \tag{0.58}$$

The ill-posedness might enter from the eigenvalue λ_i becoming zero. For that purpose, choosing $c = L/T$ results in $\lambda_i = (-1)^i$. This selection will result in eigenvalues of equal magnitude and will recover the initial condition with **no** instability.

(b) Dependence on ability to reconstruct on final time $T = L/2c$

In this case

$$\lambda_i = \cos\left(i\frac{\pi c T}{L}\right) = \cos\left(i\frac{\pi}{2}\right) \tag{0.59}$$

$$= \begin{cases} 0, & i = 1, 3, 5, \dots \\ -1, & i = 2, 6, 10, \dots \\ 1 & i = 4, 8, 12, \dots \end{cases} \tag{0.60}$$

The zero eigenvalues cause ill-posedness for odd modes. This can be ameliorated by using the velocity measurements, as it is a sinusoid having non-zero values at odd modes when the displacement is zero.

(c) In real world, the ability to reconstruct

The real data contains instrument and observation errors. The instruments have their least counts whereas the observations are made upto some finite resolution (in space and time). Moreover, sometimes the models are not enough to capture the complete physics of this process such as one-dimensionality, thermal expansion, fatigue, and creep. There can also be irreversibilities due to air resistance and friction.

Code for 1(c-d):

```
#Decay of Eigenvalues in 1D heat diffusion
#Mohammad Afzal Shadab
#Date modified: 09/04/21

import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm          #color map
plt.rcParams.update({'font.family': "Serif"})
plt.rcParams.update({'font.size': 22})


eigen_cont = lambda k,T,L,i: np.exp(-k*T*(np.pi/L)**2*i**2) #eigen value of continuous
                                                            operator
eigen_disc = lambda dt,k,h,i,nx,nt: (1 + dt*k*4/h**2*(np.sin(np.pi*i/(2*nx)))**2)**(-nt)
                                                             #eigen value of discrete operator

#Colors
brown  = [181/255 , 101/255, 29/255]
red    = [255/255 ,0/255 ,0/255 ]
blue   = [ 30/255 ,144/255 , 255/255 ]
green  = [  0/255 , 166/255 ,  81/255]
orange = [247/255 , 148/255 ,  30/255]
purple = [102/255 ,  45/255 , 145/255]
brown  = [155/255 ,  118/255 ,  83/255]
tan    = [199/255 , 178/255 , 153/255]
gray   = [100/255 , 100/255 , 100/255]

color_array = [red,green,orange,brown,blue,purple]


#PART (c)
#Parameters
T = 1 #Final time
L = 1 #Length of the domain [0,1]
nx= 100#Number of cellls in x
nt= 100#Number of cells in time

i = np.linspace(1,nx - 1,nx - 1) #Modes
h = L/nx #Cell sixe
dt= T/nt #Time step
k_array = np.logspace(-4,0,5)


plt.figure(figsize=(12,10))
for count, k in enumerate(k_array):
    plot = plt.semilogy(i,eigen_cont(k,T,L,i),c=color_array[count],label=r'$k=%0.4f$'%k)

plt.legend(loc='best')
plt.ylim([1e-16,10])
plt.xlabel(r'$i$')
plt.ylabel(r'$\lambda_i$')

plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
plt.savefig(f"1c_discvscont_evals.png")


#PART (d)
#Parameters
T = 0.1 #Final time
L = 1 #Length of the domain [0,1]
k = 0.01#Diffusion coefficient

nxnt = [160,80,40,20]

plt.figure(figsize=(15,10))
for count, nx in enumerate(nxnt):
    i = np.linspace(1,nx- 1,nx - 1) #Modes
    nt = nx
    h = L/nx #Cell sixe
    dt= T/nt #Time step
    plot = plt.semilogy(i,eigen_disc(dt,k,h,i,nx,nt),'ro',c=color_array[count],label=f'(
                                                                {nx},{nt})')
```

```
i = np.linspace(1,160 - 1,160 - 1) #Modes

plot = plt.semilogy(i,eigen_cont(k,T,L,i),'k--',label=f'Analytical')

plt.legend(loc='lower left')
plt.xlabel(r'$i$')
plt.xlim([0,80])
plt.ylim([1e-16,10])
plt.ylabel(r'$\lambda_i$')

plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
plt.savefig(f"1d_discvscont_eval_discretization.png")
```

Code for 2(b):

```
#Decay of Eigenvalues in 1D Poisson equation
#Mohammad Afzal Shadab
#Date modified: 09/04/21


import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm         #color map
plt.rcParams.update({'font.family': "Serif"})
plt.rcParams.update({'font.size': 22})


eigen_cont = lambda k,L,i: 1/(k*(i*np.pi/L)**2) #eigen value of continuous operator
eigen_disc = lambda k,h,i,nx: 1/(k*4/h**2*(np.sin(np.pi*i/(2*nx)))**2) #eigen value of
                                             discrete operator

#Colors
brown  = [181/255 , 101/255, 29/255]
red    = [255/255 ,0/255 ,0/255 ]
blue   = [ 30/255 ,144/255 , 255/255 ]
green  = [  0/255 , 166/255 ,  81/255]
orange = [247/255 , 148/255 ,  30/255]
purple = [102/255 ,  45/255 , 145/255]
brown  = [155/255 ,  118/255 ,  83/255]
tan    = [199/255 , 178/255 , 153/255]
gray   = [100/255 , 100/255 , 100/255]

color_array = [red,green,orange,brown,blue,purple]


#PART (b)
#Parameters
L = 1 #Length of the domain [0,1]
nx= 100#Number of cellls in x

i = np.linspace(1,nx - 1,nx - 1) #Modes
h = L/nx #Cell sixe
k_array = [1]


plt.figure(figsize=(15,10))
for count, k in enumerate(k_array):
    plot = plt.semilogy(i,eigen_cont(k,L,i),c=color_array[count],label=r'$\mathcal{F}, k
                                                =%0.4f$'%k)
    plot = plt.semilogy(i,eigen_disc(k,h,i,nx),'ro',c=color_array[count],label=r'$ {\bf{
                                                F}}, k=%0.4f$'%k)

plt.legend(loc='best')
plt.xlabel(r'$i$')
plt.ylabel(r'$\lambda_i$')

plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
plt.savefig(f"2b_discvscont_evals.png")
```

Code for 2(c-g):

```python
#Decay of Eigenvalues in 1D Poisson problem
#Mohammad Afzal Shadab
#Date modified: 08/13/21

import numpy as np
import matplotlib.pyplot as plt
import scipy.sparse as sp
import scipy.sparse.linalg as la
plt.rcParams.update({'font.family': "Serif"})
plt.rcParams.update({'font.size': 22})

SMALL_FONT = 16
MEDIUM_FONT = 20
LARGE_FONT = 24
FIG_SIZE = (12,8)
FIG_SIZE_SMALL = (6,6)
PATH = ''

plt.rc('font', size=SMALL_FONT)
plt.rc('axes', titlesize=SMALL_FONT)
plt.rc('axes', labelsize=SMALL_FONT)
plt.rc('xtick', labelsize=SMALL_FONT)
plt.rc('ytick', labelsize=SMALL_FONT)
plt.rc('legend', fontsize=SMALL_FONT)
plt.rc('figure', titlesize=MEDIUM_FONT)

###############################################################################

def mTrueFun(x):
    """ plot returns the true m evaluated at x
    where m is 1D function """
    return np.maximum(0, 1 - np.abs(1 - 4*x)) + 100 * x**10 * (1-x)**2

def assembleMatrix(k, h, n_dof):
    """ assembles the stiffness matrix K based on central differences """

    diagonals = np.zeros((3, n_dof))
    diagonals[0,:] = -1.0
    diagonals[1,:] = 2.0
    diagonals[2,:] = -1.0

    K = k/(h**2) * sp.spdiags(diagonals, [-1, 0, 1], n_dof, n_dof)
    K = K.tocsr()
    return K

def generateData(u_true, sigma):
    """ generates data by adding to the true solution gaussian white noise
    with standard dev = sigma """

    return u_true + np.random.randn(u_true.shape[0])*sigma

def addEndPoints(arr, arr_end=[0,0]):
    """ adds endpoints to the array """
    arr_new = np.zeros(arr.shape[0]+2)
    arr_new[1:-1] = arr
    arr_new[0] = arr_end[0]
    arr_new[-1] = arr_end[1]
    return arr_new

def solveFwd(K,f):
    """ solves the forward problem given formed K and f=rhs
    i.e. Ku=f"""

    return la.spsolve(K,f)


def applyInvF(K, d):
    """ apply the inverse of the P2O operator to obtain inversion results
    noting that F = K-1 """

    return K.dot(d)

def assembleF(K, n_dof):
    """ assembles the matrix of parameter to observable map
    by solving the forward problem for the basis"""

    F = np.zeros((n_dof, n_dof))
    m_i = np.zeros(n_dof) # basis vectors

    for i in range(n_dof):
        m_i[i] = 1.0
        F[:,i] = solveFwd(K, m_i)
        m_i[i] = 0

    return F


def solveTikhonov(d, F, alpha):
    """ assuming R = Id """

    H = np.dot(F.transpose(), F) + alpha * np.identity(F.shape[1])
    rhs = np.dot(F.transpose(), d)

    return np.linalg.solve(H, rhs)


##############################################################################

if __name__ == '__main__':
    n_array = [10, 20, 40, 80, 160]
    k_array = [1, 0.01, 0.0001]
    noises = [1e-2, 1e-4, 1e-6]
    for noise_sigma in noises:
        for k in k_array:
            for nx in n_array:
                np.random.seed(100)
                #k = 0.01
                L = 1
                #nx = 80 # number of elements
                h = L/nx

                x = np.linspace(0, L, nx+1)
                m_true = mTrueFun(x)

                K = assembleMatrix(k,h,nx-1)
                F = assembleF(K, nx-1)

                x_interior = x[1:-1]
                f_true = m_true[1:-1] # interior nodes of the force

                # solving the system for true m
                u_true = solveFwd(K, f_true)

                # converting to plotting by adding endpoints
                u_plot = np.zeros(x.shape)
                u_plot[1:-1] = u_true

                # generate data
                d_plot = generateData(u_plot, noise_sigma)
                d = d_plot[1:-1]

                plt.figure(figsize=FIG_SIZE)
                plt.plot(x, m_true, 'k')
                plt.title("$m_{true}$")

                plt.savefig(PATH + "figures/p2_part_c_true_m.png")

                plt.figure(figsize=FIG_SIZE_SMALL)
                plt.plot(x, u_plot, '-k')
                plt.plot(x, d_plot, 'ob')
                plt.title("$\sigma = %.0e$, $k = %.0e$, $n_x = %d$" %(noise_sigma, k, nx
                                                                    ))
                plt.legend(["$u_{true}$", "Noisy data"])

                plt.savefig(PATH + "figures/p2_part_c_data_k=%.0e_n=%d_noise=%.0e.png" %
                                                                    (k, nx, noise_sigma))
                #
                                                            ##############################################

                # c. now solve the inverse problem naively with inversion of the
                                                            operator K

                m_naive = applyInvF(K, d)

                plt.figure(figsize=FIG_SIZE_SMALL)
                plt.plot(x, m_true, '-k')
                plt.plot(x_interior, m_naive, 'or')
                plt.legend(["True m", "Estimated m"])
                plt.title("$\sigma = %.0e$, $k = %.0e$, $n_x = %d$" %(noise_sigma, k, nx
                                                            ))

                plt.savefig(PATH + "figures/p2_partC_mOpt_k=%.0e_n=%d_noise=%.0e.png" %(
                                                            k, nx, noise_sigma))

                # prediction

                u_naive = la.spsolve(K, m_naive)
                plt.figure(figsize=FIG_SIZE_SMALL)
                plt.plot(x_interior, u_true, '-k')
                plt.plot(x_interior, d, 'ob')
                plt.plot(x_interior, u_naive, 'or', markersize=3)

                plt.legend(["True u", "Noisy data", "Predicted u"])
                plt.title("$\sigma = %.0e$, $k = %.0e$, $n_x = %d$" %(noise_sigma, k, nx
                                                            ))

                plt.savefig(PATH + "figures/p2_partC_uPred_k=%.0e_n=%d_noise=%.0e.png" %
                                                            (k, nx, noise_sigma))

#######################################


#(d) Solving inverse problem with Tikhanov regularization

FIG_SIZE = (12,8)
np.random.seed(100)

k = 1
L = 1
nx = 200 # number of elements
h = L/nx

x = np.linspace(0, L, nx+1)
m_true = mTrueFun(x)

K = assembleMatrix(k,h,nx-1)
F = assembleF(K, nx-1)

x_interior = x[1:-1]
f_true = m_true[1:-1] # interior nodes of the force

# solving the system for true m
u_true = solveFwd(K, f_true)

# converting to plotting by adding endpoints
u_plot = np.zeros(x.shape)
u_plot[1:-1] = u_true

# generate data
noise_sigma = 1e-4
d_plot = generateData(u_plot, noise_sigma)
d = d_plot[1:-1]

plt.figure()
plt.plot(x, m_true, 'k')
plt.title("$m_{true}$")
plt.figure()
plt.plot(x, u_plot, '-k')
plt.plot(x, d_plot, 'ob')
plt.title("Synthetic observations")
plt.legend(["$u_{true}$", "Noisy data"])

plt.show()

##############################################################################

alphas = np.logspace(-7,-2,6)
colors = ['r', 'g', 'b', 'c', 'm', 'y']

legend_reg = ["$\\alpha = %.0e$" %(alpha) for alpha in alphas]
legend = ["True value"] + legend_reg

plt.figure(figsize=FIG_SIZE)
plt.plot(x, m_true, '-k')

for (alpha, c) in zip(alphas, colors):
    m_reg = solveTikhonov(d, F, alpha)
    plt.plot(x_interior, m_reg, color = c)
plt.legend(legend, loc="best")
plt.title("Inversion with Tikhanov regularization")
plt.axis([0, 1, -0.01, 1.11])
plt.savefig(PATH + "figures/p2_part_d_regularization_%.0e.png" %(alphas[0]))
plt.show()

##############################################################################

# (e) L-curve criterion

delta = np.linalg.norm(d - u_true)

norm_m = np.zeros(alphas.shape)
norm_residual = np.zeros(alphas.shape)
norm_diff_true = np.zeros(alphas.shape)


for ii in range(alphas.size):
    m_reg = solveTikhonov(d, F, alphas[ii])
    norm_m[ii] = np.linalg.norm(m_reg)

    u_pred = solveFwd(K, m_reg)
    norm_residual[ii] = np.linalg.norm(u_pred - d)

    norm_diff_true[ii] = np.linalg.norm(m_reg - f_true)

plt.figure(figsize=FIG_SIZE)
plt.loglog(norm_residual, norm_m, '-ob')
plt.xlabel("$\|Fm-d\|$")
plt.ylabel("$\|m\|$")
plt.title("L-curve")
plt.savefig(PATH + "figures/p2_part_e_lCurve_%.0e.png" %(alphas[0]))

delta_plot = np.ones(alphas.shape) * delta

# (f) Morozov discrepancy criterion

plt.figure(figsize=FIG_SIZE)
plt.loglog(alphas, norm_residual, '-ob')
plt.loglog(alphas, delta_plot, '--k')
plt.xlabel("$\\alpha$")
plt.ylabel("$\|Fm-d\|$")
plt.legend(["Regularized solution", "$\delta$"])
plt.title("Morozov discrepancy criteria")
plt.savefig(PATH + "figures/p2_part_f_morozov_%.0e.png" %(alphas[0]))
plt.show()

# (g) Error in reconstruction

plt.figure(figsize=FIG_SIZE)
plt.loglog(alphas, norm_diff_true, '-ob')
plt.xlabel("$\\alpha$")
plt.ylabel("$\|m-m_{true}\|$")
plt.title("Error in reconstruction: $m_{true}$ comparison")
plt.savefig(PATH + "figures/p2_part_g_bestAlpha_%.0e.png" %(alphas[0]))
plt.show()

##############################################################################
```