CSE393P : Assignment01

Problem 1

1a: compute eigenvalues, eigenfunctions of P20 map

Arushi R. Sadam
Dr. Omar Ghattas

CSE 393P, Computational & Variational Methods for Inverse Problems
12 February 2025

Assignment 1
Ill-posedness and Regularization

1. $$\begin{cases} \frac{\partial u}{\partial t} - K\frac{\partial^2 u}{\partial x^2} = 0 & 0<x<L \quad 0<t\leq T \\ u(x,0) = m(x) & 0<x<L \\ u(0,t) = u(L,t) = 0 & 0<t\leq T \end{cases}$$

$\mathcal{F}(m) := u(x,T)$, $Fm = d$, $F = (I + \Delta t\, K)^{-n_t}$
where n_t is the number of time steps.

$F \in \mathbb{R}^{(n_x-1)\times(n_x-1)}$ where $h = \frac{L}{n_x}$ is the spatial mesh size

a) Confirm the eigenvalues λ_i & eigenvectors $v_i(x)$ of the continuous operator $\mathcal{F}$ are:

$$\lambda_i = e^{-kT\left(\frac{\pi i}{L}\right)^2} \qquad v_i(x) = \sqrt{\frac{2}{L}}\sin\left(i\pi\frac{x}{L}\right), \; i = 1,2,\ldots$$

Solution:
To solve the PDE, we use seperation of variables. Consider solution $u(x,T)$ to be expressable as:

$$u(x,T) = X(x)\cdot T(t)$$ (or some linear combination of $X(x)T(t)$ elements)

Then:

$$\frac{\partial u}{\partial t} = X(x)\frac{\partial T(t)}{\partial t}; \quad \frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 X(x)}{\partial x^2}T(t)$$

Plugging the partials into the PDE:

$$X(x)\frac{\partial T(t)}{\partial t} - K\frac{\partial^2 X(x)}{\partial x^2}T(t) = 0$$

Rewriting the equation to seperate time-dependent & spatially dependent terms:

$$\frac{T'(t)}{kT(t)} = \frac{X''(x)}{X(x)} = -C$$, where C must be a constant (function of t = function of x only when both functions are constants)

Now, we solve the temporal ODE:

$$\frac{T'(t)}{kT(t)} = -C; \quad T'(t) = -CkT(t)$$

the only solution $T(t)$ is exponential:

$T(t) = De^{-Ckt}$, where D is a constant.

For the spatial ODE:

$$\frac{X''(x)}{X(x)} = -C; \quad X''(x) + CX(x) = 0$$

with the characterestic equation:

$r^2 + C = 0$

$r = \pm i\sqrt{C}$ ⇒ imaginary roots so the characteristic equation indicate solutions of form:

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

where A, B are constants.

For the boundary conditions, for $u(0,t) = u(L,t) = 0$ $\forall t \in (0,T)$, per $u(x,t) = X(x)T(t)$, either $T(t) = 0$ $\forall t \in (0,T]$ or $X(0) = X(L) = 0$.

Since $T(t)=0 \ \forall \ t \in (0,T]$ would lead to the trivial solution $u(x,t)=0 \ \forall \ (0,T]$, we will consider the non-trivial case $X(0)=X(L)=0$ as our boundary conditions:

for $X(0)=0$:

$X(0) = A\sin(\sqrt{c}\cdot 0) + B\cos(\sqrt{c}\cdot 0) = 0 \Rightarrow B=0$

for $X(L)=0$

$X(L) = A\sin(\sqrt{c}\,L) + B\cos(\sqrt{c}\,L) = 0$

$\Rightarrow A\sin(\sqrt{c}\,L) = 0.$

if we consider $A=0$, we again are left with the trivial solution $X(x)=0$ or $u(x,t)=0$.

So, we consider $A \neq 0$:

$\sin(\sqrt{c}L)=0; \quad \sqrt{c}L = i\pi, \ i=1,2,3\ldots$

[Aside: $n \neq 0$, since $\sqrt{c}L=0 \Rightarrow c=0$; $X(x)=A\sin(0x)=0 \ \forall \ x,t$]

$\sqrt{c}L = i\pi; \quad c = \frac{i^2\pi^2}{L^2}, \ i=1,2,3\ldots$

So, $X(x) = A\sin\left(\frac{i\pi}{L}x\right), \ i=1,2,3\ldots$

$u_n(x,t) = X(x)T(t) = AD\,e^{-\left(\frac{i\pi}{L}\right)^2 kt}\sin\left(\frac{i\pi}{L}x\right); \ i=1,2,3\ldots$

↑ one specific type of solution / building block

We redefine constant product AD as B.

$u(x,t) = \sum_i B_i\, e^{-\left(\frac{\pi i}{L}\right)^2 kt}\sin\left(\frac{i\pi}{L}x\right)$, where B_i is the specific constant B for each mode of the solution.

Now, we can substitute the initial conditions: 04

$u(x,0) = m(x)$

$$\Rightarrow \sum_i B_i \underbrace{e^{-CK\cdot 0}}_{\text{constant}} \underbrace{\sin\left(\frac{i\pi x}{L}\right)}_{\text{eigenfunctions (w/ some normalization)}} = m(x)$$

we see the eigenfunctions $v_i(x) = G_i \sin\left(\frac{i\pi x}{L}\right)$, where G_i is some normalization constant.

for the eigenfunctions, we know:

$$\int_0^L v_n v_m \, dx = \begin{cases} 0, & n \neq m \\ 1, & n = m \end{cases}$$

since the eigenfunctions v_n & v_m must be orthogonal.

To solve for the constants G_i, $i = 1, 2, \ldots$:

$$\int_0^L v_i v \, dx = \int_0^L G_i^2 \sin^2\left(\frac{i\pi}{L}x\right) dx = 1$$

$$G_i^2 \int_0^L \sin^2\left(\frac{i\pi x}{L}\right) dx = G_i^2 \int_0^L \frac{1-\cos\left(2\frac{i\pi}{L}x\right)}{2} dx$$

$$= G_i^2 \cdot \frac{1}{2} \int_0^L 1 - \cos\left(\frac{2i\pi}{L}x\right) dx$$

$$= \frac{G_i^2}{2}\left[x - \sin\left(\frac{2i\pi}{L}x\right)\cdot\frac{L}{2i\pi}\right]_0^L$$

$$= \frac{G_i^2}{2}\left[L\right]$$

$\frac{G_i^2}{2} L = 1$; $G_i = \pm\sqrt{\frac{2}{L}}$ by convention we choose the positive normalization (prevents the flipping of the eigenfunction sign) constant: $G_i = \sqrt{\frac{2}{L}}$.

Thus, $v_i = \sqrt{\frac{2}{L}} \sin\left(\frac{i\pi x}{L}\right)$ are the eigenfunctions ($i = 1, 2, 3 \ldots$)

The general solution is:

$$u(x,T) = \sum_i \underbrace{c_i}_{\text{constant}} e^{-k\left(\frac{i\pi}{L}\right)^2 T} \underbrace{\sqrt{\frac{2}{L}} \sin\left(\frac{i\pi x}{L}\right)}_{\text{eigenfunctions}}$$

$$c_i = \int_0^L u(x,0)\, v_i(x)\, dx = \int_0^L m(x)\, v_i(x)\, dx \quad \left.\begin{array}{l}\\ \\ \end{array}\right\} \text{based on initial conditions}$$

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

For each mode: $\mathcal{F}(v_i) = u_i(x,T) = e^{-k\left(\frac{i\pi}{L}\right)^2 T} \sqrt{\frac{2}{L}} \sin\left(\frac{i\pi x}{L}\right)$

$\mathcal{F}(v_i) = e^{-k\left(\frac{i\pi}{L}\right)^2 T}\, v_i(x) = \lambda_i v_i(x)$

so, the eigenvalues are: $\lambda_i = e^{-k\left(\frac{i\pi}{L}\right)^2 T}$, $i = 1, 2, 3 \ldots$

Or: the eigenvalue, eigenfunction pair is:

$$(\lambda_i, v_i) = \left(e^{-k\left(\frac{i\pi}{L}\right)^2 T}, \sqrt{\frac{2}{L}} \sin\left(\frac{i\pi x}{L}\right)\right); \; i = 1, 2, 3 \ldots$$

1b:derive eigenvalues and eigenvectors of discrete P2O map

b) Determine the eigenvalues & eigenvectors of the discrete P2O map $\underset{\sim}{F}$. 06

Given:

eigenvalues of $\underset{\sim}{K}$, $\mu_i = K\frac{4}{h^2}\sin^2\left(\frac{\pi i}{2n_x}\right)$; $i = 1,2 \dots n_x - 1$

eigenvectors of $\underset{\sim}{K}$, $[\vec{u}_i]_j = \sqrt{\frac{2}{L}}\sin\left(\pi i \frac{jh}{L}\right)$; $i = 1,2 \dots n_x - 1$ (at index j)

To discretize, we use a centered finite difference for the spatial 2nd derivative & a backward euler method for the time discretization:

$h = \frac{L}{n_x}$ where h is the spatial element size, n_x is the # of subintervals.

$\underset{\sim}{u}(t) = \begin{bmatrix} u_1(t) \\ \vdots \\ u_n(t) \end{bmatrix}$, $n = n_x - 1$ (where $\underset{\sim}{u}$ is the heat solution at each discretized points)

$\underset{\sim}{K} = \frac{K}{h^2}\begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & \ddots & \ddots & \ddots \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}$

$\Delta t = \frac{T}{n_t}$, where n_t is the number of time steps

$\frac{\underset{\sim}{u}^{i+1} - \underset{\sim}{u}^{i}}{\Delta t} + \underset{\sim}{K}\,\underset{\sim}{u}^{i+1} = 0$ (discretized PDE $\frac{\partial}{\partial t}\underset{\sim}{u}(t) + \underset{\sim}{K}\underset{\sim}{u}(t) = 0$)

Solving for $\underset{\sim}{u}^{i+1}$:

$$\underset{\sim}{u}^{i+1} = (\underset{\sim}{I} + \Delta t\, \underset{\sim}{K})^{-1}\, \underset{\sim}{u}^{i}$$

or:

$$\underset{\sim}{u}^{i} = (\underset{\sim}{I} + \Delta t\, \underset{\sim}{K}^{-1})^{-i}\, \underset{\sim}{m} \quad (u^0 = m)$$

$$\mathcal{F}(m) \cong \underset{\sim}{F}\, \underset{\sim}{m} = (\underset{\sim}{I} + \Delta t\, \underset{\sim}{K}^{-1})^{-n_t}\, \underset{\sim}{m}$$

after discretization

Thus, $\underset{\sim}{F} = (\underset{\sim}{I} + \Delta t\, \underset{\sim}{K}^{-1})^{-n_t}$.

Solution:

For u_i to be a eigenvector of K:

$\underset{\sim}{K}\, \vec{u}_i = \mu_i \vec{u}_i$ or, with an eigenvalue decomposition:

$K = V \Lambda V^{-1}$ where V is a matrix of eigenvectors & Λ is a diagonal matrix of eigenvalues:

$V^{-1} = V^T$

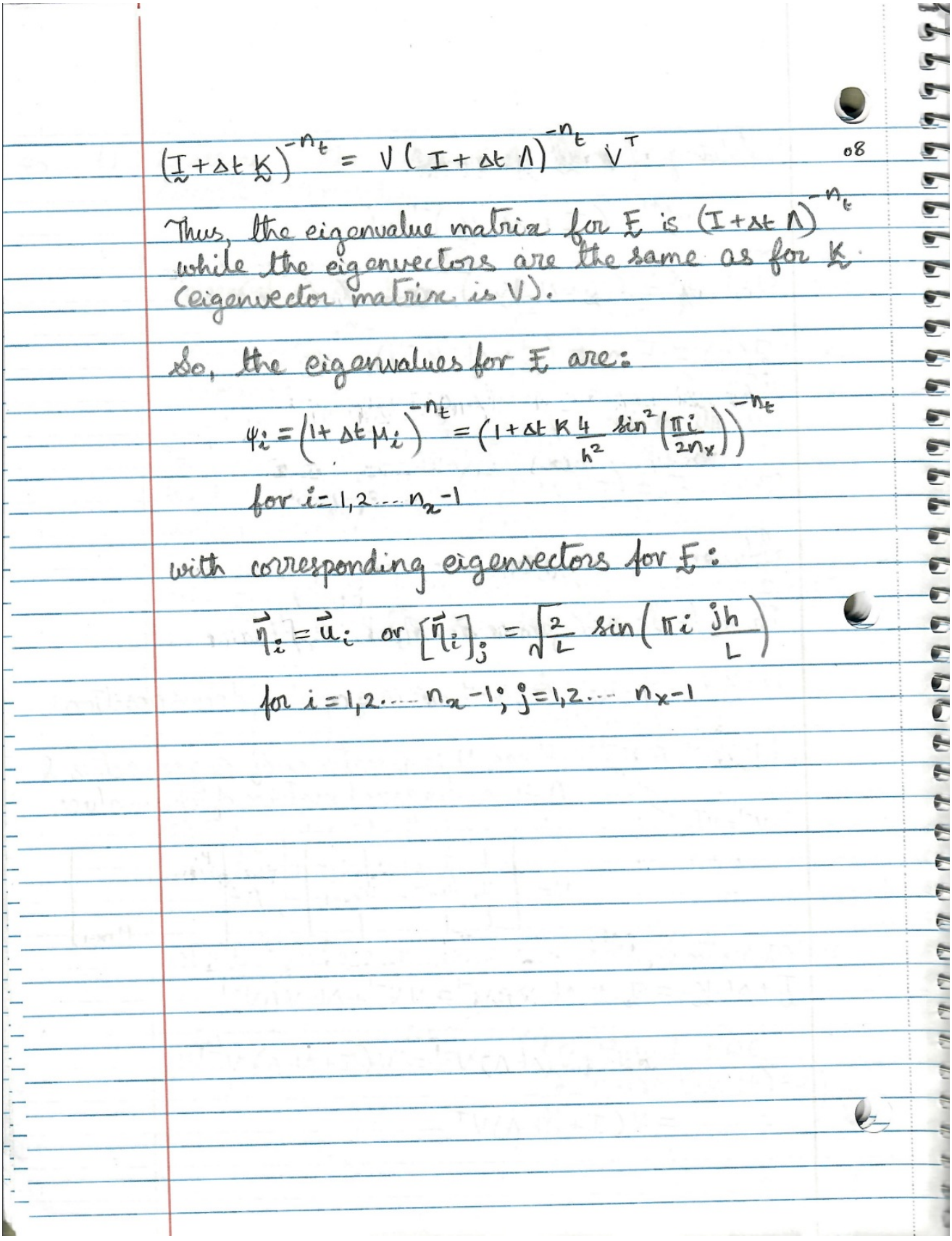
$$V = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_{nx-1} \\ | & | & & | \end{bmatrix}, \quad \Lambda = \begin{bmatrix} \mu_1 & & & \\ & \mu_2 & & \\ & & \ddots & \\ & & & \mu_{nx-1} \end{bmatrix}$$

We notice:

$$\underset{\sim}{I} + \Delta t\, \underset{\sim}{K} = \underset{\sim}{I} + \Delta t\, V \Lambda V^{-1} = V V^{-1} + \Delta t\, V \Lambda V^{-1}$$

$$= V V^{-1} + V(\Delta t \Lambda) V^{-1} = V(I + \Delta t \Lambda) V^{-1}$$

$$= V(I + \Delta t \Lambda) V^T$$

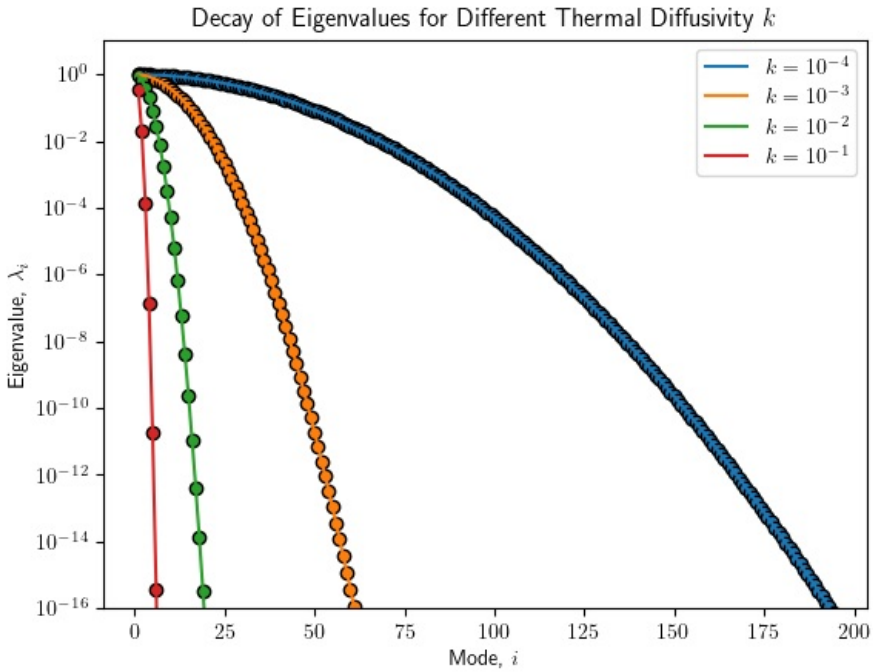


1c: decay of continuous P2O map eigenvalues

In [21]:
```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib as mpl
mpl.rcParams['text.usetex'] = True
```

In [22]:
```
# Constants
T = 1
L = 1

def eigenvalues(k):
    eigvals = []
    i = 1
    eigval = 1  # Arbitrary initial value to start the loop
```

```
    while eigval >= 10**(-16):
        eigval = np.exp(-k * T * (i * np.pi / L)**2)
        eigvals.append(eigval)  # Append to list
        i += 1  # Increment index
    return eigvals

# Loop through k values and plot eigenvalues
for k in np.logspace(-4, -1, 4):  # Added range for logspace
    eigvals = eigenvalues(k)
    plt.plot(range(1, len(eigvals) + 1), eigvals, "-", label=rf"$k=10^{{{int(np.log10(k))}}}$")
    plt.scatter(range(1, len(eigvals) + 1), eigvals, marker = "o", edgecolors="black")


# Configure plot
plt.yscale('log')  # Log scale for y-axis
plt.ylim(10**(-16))
plt.xlabel(r"Mode, $i$")
plt.ylabel(r"Eigenvalue, $\lambda_{i}$")
plt.title("Decay of Eigenvalues for Different Thermal Diffusivity $k$")
plt.legend()
plt.show()
```



When using $T = 1$ and $L = 1$ for the heat diffusion problem, the eigenvalues of the continuous operator F decay exponentially. We know the eigenvalues, $\lambda_i = e^{-kT(i\pi/L)^2}$, decay exponentially with respect to the squared mode, i^2 (decay is proportional to the negative quadratic term, which grows rapidly); this is shown by the quadratic relation between the eigenvalues and the modes on a logarithmic scale.

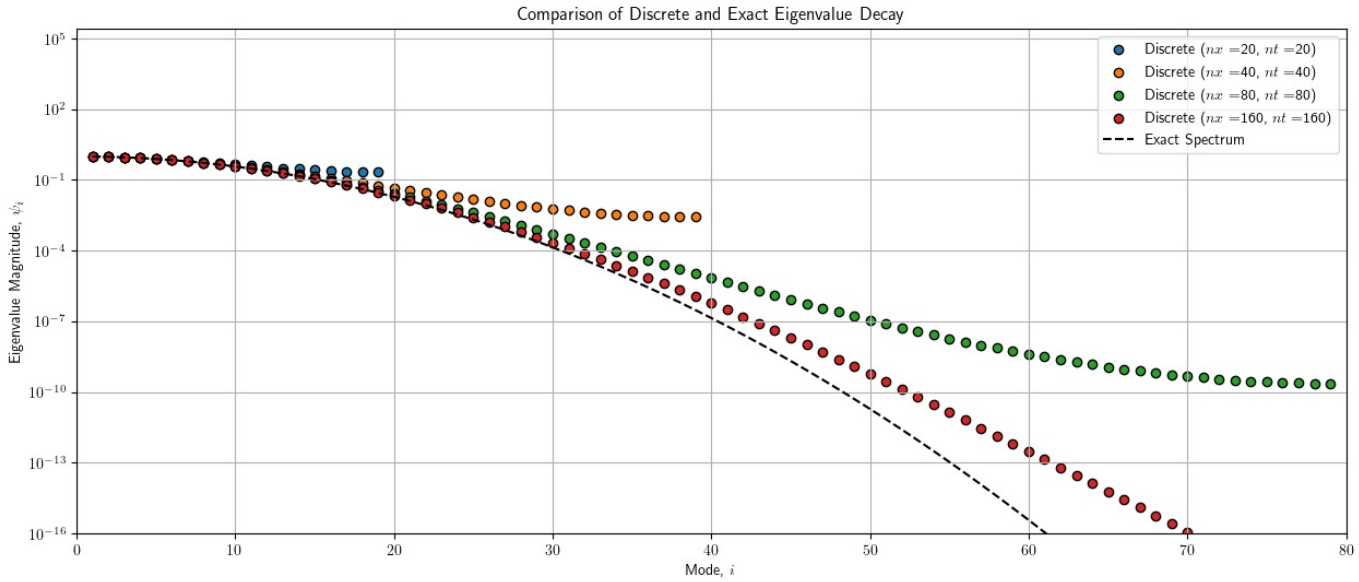
For higher thermal diffusivities, the eigenvalues decay more quickly (with respect to the modes). With higher thermal diffusivities, the initial heat distribution $m(x)$ is rapidly dispersed; when measurement/observation data is collected at the same final time T, more information about the initial heat distribution has been lost for large thermal diffusivities (data for the heat distribution $d(x, T) \approx 0$), leading to less accurate inverse problem solutions (recovering $m(x)$). The continuous operator F becomes more ill-posed for larger k as the eigenvalues approach machine precision faster.

For small thermal diffusivities, the rate of eigenvalue decay is lower for higher modes; at time T, more modes are recoverable (more eigenvalues of higher frequency) meaning more detail/information of the initial heat distribution is preserved.

1d: decay of discretized P2O map eigenvalues

In [23]:
```
import numpy as np
import matplotlib.pyplot as plt


# Constants
L = 1       # Length of the domain
k = 0.01    # Diffusion coefficient
T = 0.1     # Final time

def exact_eigenvalues(nx):
    """Compute exact eigenvalues."""
    return [np.exp(-k * T * (i * np.pi / L)**2) for i in range(1, nx)]

def discrete_eigenvalues(nx, nt):
```

```
    """Compute discrete eigenvalues."""
    h = L / nx
    delta_t = T / nt
    mu = lambda i: k * 4 / h**2 * np.sin(np.pi * i / (2 * nx))**2
    return [(1 + delta_t * mu(i))**-nt for i in range(1, nx)]

# Discretization pairs (nx, nt)
discretizations = [(20, 20), (40, 40), (80, 80), (160, 160)]

# Plot
plt.figure(figsize=(15, 6))
for nx, nt in discretizations:
    exact = exact_eigenvalues(nx)
    discrete = discrete_eigenvalues(nx, nt)
    plt.scatter(range(1, nx), discrete, label=f"Discrete ($nx=${nx}, $nt=${nt})", marker='o', edgecolors="black
    if nx == 160:  # Plot exact for the finest resolution
        plt.plot(range(1, nx), exact, label="Exact Spectrum", linestyle="--", color="black")

# Configure plot
plt.yscale('log')
plt.ylim(10**(-16))
plt.xlim(0,80)
plt.xlabel("Mode, $i$")
plt.ylabel("Eigenvalue Magnitude, $\psi_i$")
plt.title("Comparison of Discrete and Exact Eigenvalue Decay")
plt.legend()
plt.grid()
plt.show()
```



For higher resolution (finer temporal and spatial discretization), the discrete operator's eigenvalues, ψ_i, match the continuous operator's eigenvalues, λ_i, more accurately, especially for higher modes.

For the plotted discretizations above, the discretized operator's eigenvalues are always larger than the true/continuous operator's eigenvalues. Additionally, the true eigenvalues decay more rapidly than the discretized eigenvalues. With these two facts, we can conclude that discretization of the operator results in regularization of the problem. Coarse discretizations provide more regularization than finer discretizations; for 20 spatial/temporal elements, the recoverable eigenvalues are on the same scale of magnitude, while, for 160 spatial/temporal elements, the recoverable eigenvalues drop more than 16 orders of magnitude. Coarse discretizations elimate higher modes (and smaller eigenvalues), and slightly increase the magnitude for the remaining low mode eigenvalues; combined, these two effects make the inverse problem less ill-posed (acting as a regularizer) for coarser discretizations (but also lead to less accurate forward problem solutions).

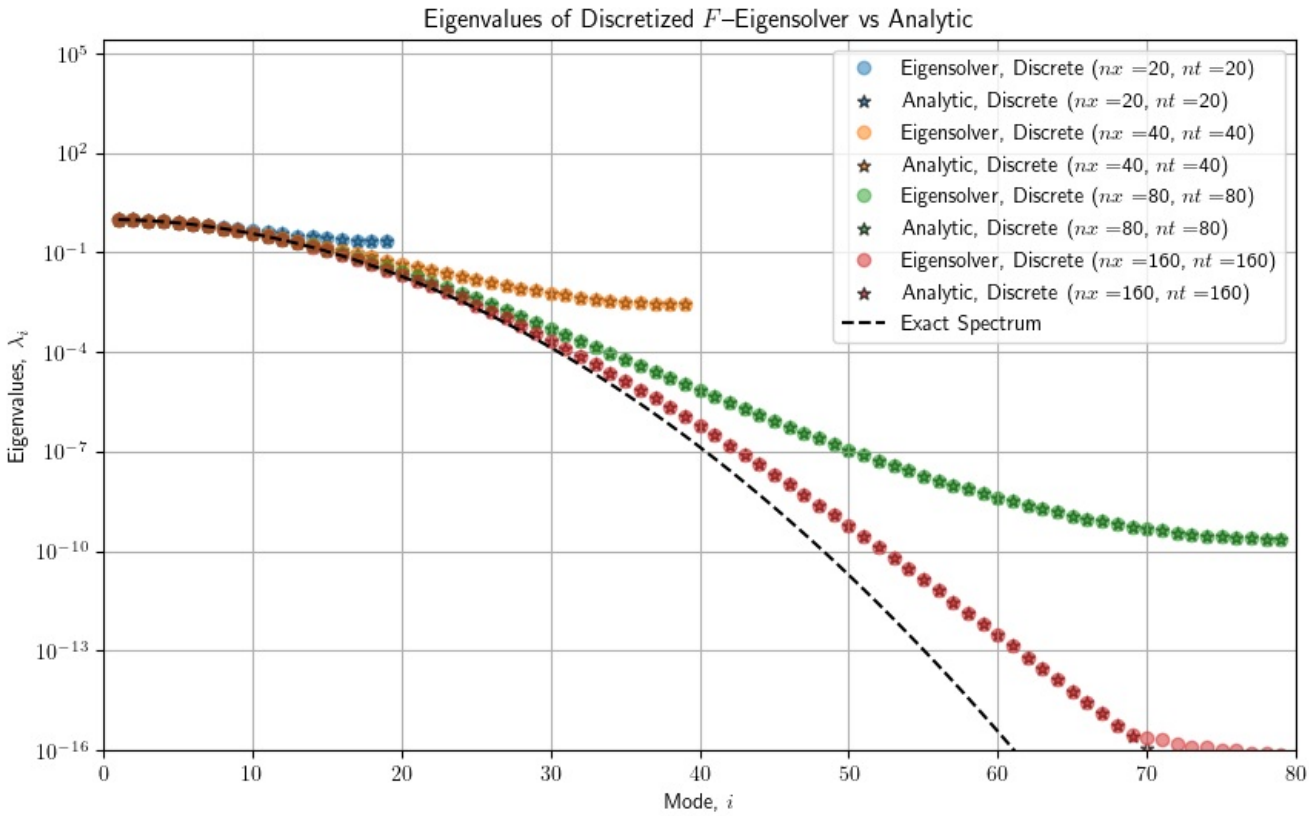
To recheck the discretized operator's eigenvalues, we can also plot the numerically evaluated eigenvalues (via the eigensolver written in the provided Notebook01_Ill_posedness.ipynb):

In [24]:
```
import scipy.sparse as sp
import scipy.sparse.linalg as la
def assembleMatrix(k, h, dt, n):
    """
    The function assembles the matrix (I + \delta_t K)
    Here:
    - k is the thermal diffusivity coefficient
    - h is the spacial discretization size
    - dt is the temporal discretization size
    - n is the number of subintervals
    """
    # To assemble matrix K, first create 3 diagonals
    # and then call scipy function `spdiags`,
```

```
    # which creates a sparse matrix given the diagonals
    diagonals = np.zeros((3, n))   # 3 diagonals
    diagonals[0,:] = -1.0/h**2
    diagonals[1,:] =  2.0/h**2
    diagonals[2,:] = -1.0/h**2
    K = k*sp.spdiags(diagonals, [-1,0,1], n,n)

    # M holds an identity matrix,
    # which is assembled similarly to K,
    # but by specifying only main diagonal of ones
    M = sp.spdiags(np.ones(n), 0, n,n)

    return M + dt*K

def solveFwd(m, k, h, dt, n, nt):
    """
    The function computes the temperature at the final time, u(T),
    for a given initial condition m.
    Here:
    - m is the initial condition
    - k is the thermal diffusivity coefficient
    - h is the spacial discretization size
    - dt is the temporal discretization size
    - n is the number of subintervals
    - nt is the number of time steps
    """


    # Assemble the matrix (note, that it is
    # constant, so we can use it for all time steps)
    A = assembleMatrix(k, h, dt, n)

    # u_old holds the most recent known value of u
    u_old = m.copy()

    # solve the system for nt time steps
    for i in np.arange(nt):
        # scipy function `spsolve` solves
        # sparse linear system
        u = la.spsolve(A, u_old)
        u_old[:] = u

    return u
def computeEigendecomposition(k, n, nt):
    h = L/float(nx)
    dt = T/float(nt)
    ## Compute F as a dense matrix
    F = np.zeros((n,n))
    m_i = np.zeros(n)

    for i in np.arange(n):
        m_i[i] = 1.0
        F[:,i] = solveFwd(m_i, k, h, dt, n, nt)
        m_i[i] = 0.0

    ## solve the eigenvalue problem
    lmbda, U = np.linalg.eigh(F)
    ## sort eigenpairs in decreasing order
    lmbda[:] = lmbda[::-1]
    lmbda[lmbda < 0.] = 0.0
    U[:] = U[:,::-1]

    return lmbda, U


plt.figure(figsize=(10, 6))
for nx, nt in discretizations:
    lmbda, U = computeEigendecomposition(k, nx-1, nt)
    # plt.figure(figsize=(20,10))
    plt.semilogy(range(1, len(lmbda)+1),lmbda, 'o', label=f"Eigensolver, Discrete ($nx=${nx}, $nt=${nt})", alpha
    discrete = discrete_eigenvalues(nx, nt)
    plt.scatter(range(1, nx), discrete, label=f"Analytic, Discrete ($nx=${nx}, $nt=${nt})",marker='*', edgecolo
    if nx == 160:  # Plot exact for the finest resolution
        exact = exact_eigenvalues(nx)
        plt.plot(range(1, nx), exact, label="Exact Spectrum", linestyle="--", color="black")
plt.legend()
plt.xlabel(r"Mode, $i$")
plt.ylabel(r"Eigenvalues, $\lambda_{i}$")
plt.title("Eigenvalues of Discretized $F$--Eigensolver vs Analytic")
plt.yscale('log')
plt.ylim(10**(-16))
plt.xlim(0,80)
plt.grid()
```

```
plt.show()
```

```
/tmp/ipykernel_29516/719482091.py:53: SparseEfficiencyWarning: spsolve requires A be CSC or CSR matrix format
  u = la.spsolve(A, u_old)
```



Per the above plot, we see that the analytic eigenvalues and the eigensolver-derived eigenvalues match, generally. Eigensolver-derived eigenvalues with magnitude near machine precision--10^{-16}--do not match with the analytic eigenvalues exactly, which is as expected; near machine precision, error is acceptable in the eigensolvers. With the above plot, we can confirm that the analytic/derived eigenvalues are accurate.

Problem 2

2a: determine eigenvalues, eigenfunctions for continuous P2O map

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

a) Determine the eigenvalues & eigenfunctions for the continuous operator $\mathcal{F}(m) := u(x)$.

Solution:
First, we consider the fownrd problem:

$$-K\frac{\partial^2 u}{\partial x^2} = m(x) \Rightarrow -\frac{\partial^2 u}{\partial x^2} = \frac{m(x)}{K}$$

for any eigenfunction v_i, $\mathcal{F}(v_i) = \lambda_i v_i$ where λ_i is the corresponding eigenvalue.
Then, $u_i(x) = \lambda_i v_i(x)$ so $-K\frac{\partial^2(\lambda_i v_i(x))}{\partial x^2} = v_i(x)$

$$-K\lambda_i\frac{\partial^2 v_i(x)}{\partial x^2} = v_i(x) \; ; \; \frac{\partial^2 v_i(x)}{\partial x^2} \cdot K\lambda_i + v_i(x) = 0$$

$$\Rightarrow \frac{\partial^2 v_i(x)}{\partial x^2} + \frac{1}{K\lambda_i} v_i(x) = 0$$

characteristic equation:

$$r^2 + \frac{1}{K\lambda_i} = 0 \quad \text{let } \mu_i = \frac{1}{K\lambda_i} \; ; \; r = \pm i\sqrt{\mu_i}$$

$$v_i(x) = C_1 e^{i\sqrt{\mu_i}} + C_2 e^{-i\sqrt{\mu_i}} = A\cos(\sqrt{\mu_i}x) + B\sin(\sqrt{\mu_i}x)$$

where A, B are constants

with the Boundary Conditions:

$v_i(0) = A = 0$.
$v_i(L) = B\sin(\sqrt{\mu_i}L) = 0$

For non-trivial eigenfunctions, $B \neq 0$:

$$\sin(\sqrt{\mu_i} L) = 0 \; ; \; \sqrt{\mu_i} L = i\pi \; , \; i = 1, 2 \ldots$$

(if $i=0$, we would have $v_i = B \sin(0) = 0$, which would again be the trivial case)

$$\sqrt{\frac{1}{K\lambda_i}} L = i\pi \; ; \; \mu_i = \left(\frac{i\pi}{L}\right)^2 = \frac{1}{K\lambda_i} \; ; \; \lambda_i = \frac{L^2}{i^2\pi^2 K} \; , \; i = 1, 2 \ldots$$

$$v_i = B \sin\left(\frac{i\pi}{L} x\right) \; ; \; i = 1, 2 \ldots$$

for orthonormal eigenfunctions:

$$\int_0^L v_i \, v_j \, dx = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

i, j are arbitrary eigenfunction indices.

$$\Rightarrow \int_0^L B^2 \sin^2\left(\frac{i\pi}{L} x\right) dx = 1$$

$$\frac{B^2}{2} \int_0^L 1 - \cos\left(\frac{2i\pi}{L} x\right) dx = 1$$

$$\frac{B^2}{2} \left[x - \frac{L}{2i\pi} \sin\left(\frac{2i\pi}{L} x\right) \right]_0^L = 1$$

$$\frac{B^2}{2} L = 1 \; ; \; B = \pm\sqrt{\frac{2}{L}} \Rightarrow \text{we choose } B > 0$$

$$B = \sqrt{\frac{2}{L}}$$

Thus, the eigenvalue, eigenfunction pair is: 11

$$(\lambda_i, v_i) = \left(\frac{L^2}{i^2 \pi^2 K}, \sqrt{\frac{2}{L}} \cdot \sin\left(\frac{i\pi}{L} x\right) \right)$$

for $i = 1, 2 \ldots$

2b: comparision of eigenvalues of discrete vs continuous P2O operator

b) Discretize the scaled Laplacian $-K u_{xx}$ with central differences s.t. $\underset{\sim}{K}$ is defined as in problem 1. Given the eigenvalues of $\underset{\sim}{K}$, find the eigenvalues of the discrete p2o map $\underset{\sim}{F}$. Compare eigenvalues of $\underset{\sim}{F}$, ψ_i $(i = 1, 2 \ldots n_x - 1)$ with the eigenvalues of $\mathcal{F}$, λ_i.

Solution

We have, after discretization of the PDE:

$\underset{\sim}{K}\, \underset{\sim}{u} = \underset{\sim}{m}$ or, $\underset{\sim}{F}(\underset{\sim}{m}) = \underset{\sim}{u}$

↗ linear operator, so:

$\underset{\sim}{F}\, \underset{\sim}{m} = \underset{\sim}{u}$; $\underset{\sim}{F}^{-1}\, \underset{\sim}{u} = \underset{\sim}{m}$

$\underset{\sim}{K} = \underset{\sim}{F}^{-1}$ or $\underset{\sim}{F} = \underset{\sim}{K}^{-1}$

We are given $\underset{\sim}{K} = V \Lambda V^{-1}$ where $V = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_{n_x-1} \\ | & | & & | \end{bmatrix}$

& $\Lambda = \begin{bmatrix} \mu_1 & & & \\ & \mu_2 & & \\ & & \ddots & \\ & & & \mu_{n_x-1} \end{bmatrix}$ (u_i represents each eigenvector of $\underset{\sim}{K}$ & μ_i represents the corresponding eigenvalue).

Also, $[u_i]_j = \sqrt{\frac{2}{L}} \sin\left(\pi i \frac{jh}{L}\right)$, $i = 1, 2 \ldots n_x - 1$, $j = 1, 2 \ldots n_x - 1$

and $\mu_i = K\frac{4}{h^2}\sin^2\left(\frac{\pi i}{2n_x}\right)$, $i = 1, 2 \ldots n_x - 1$

$$\underset{\sim}{F} = \underset{\sim}{K}^{-1} = \left(V \Lambda V^{-1}\right)^{-1} = V \Lambda^{-1} V^{-1}$$

Given Λ is diagonal, $\Lambda^{-1} = \begin{bmatrix} 1/\mu_1 & & & \\ & 1/\mu_2 & & \\ & & \ddots & \\ & & & 1/\mu_{n_x-1} \end{bmatrix} = M$

Thus, $\underset{\sim}{F} = V M V^{-1}$ is the eigendecomposition of $\underset{\sim}{F}$. We notice the eigenvector matrix V is the same for both $\underset{\sim}{F}$ & $\underset{\sim}{K}$. So, $\underset{\sim}{F}$ & $\underset{\sim}{K}$ share the same eigenvectors. M is the eigenvalue matrix; the eigenvalues of $\underset{\sim}{K}$ are the inverse of the eigenvalues of $\underset{\sim}{F}$.

The eigenvalues of $\underset{\sim}{F}$, ψ_i are:

$$\psi_i = \frac{1}{\mu_i} = \frac{h^2}{4K\sin^2\left(\frac{\pi i}{2n_x}\right)}, \quad i = 1, 2 \ldots n_x - 1$$

The eigenvectors of $\underset{\sim}{F}$, η_i have j^{th} components:

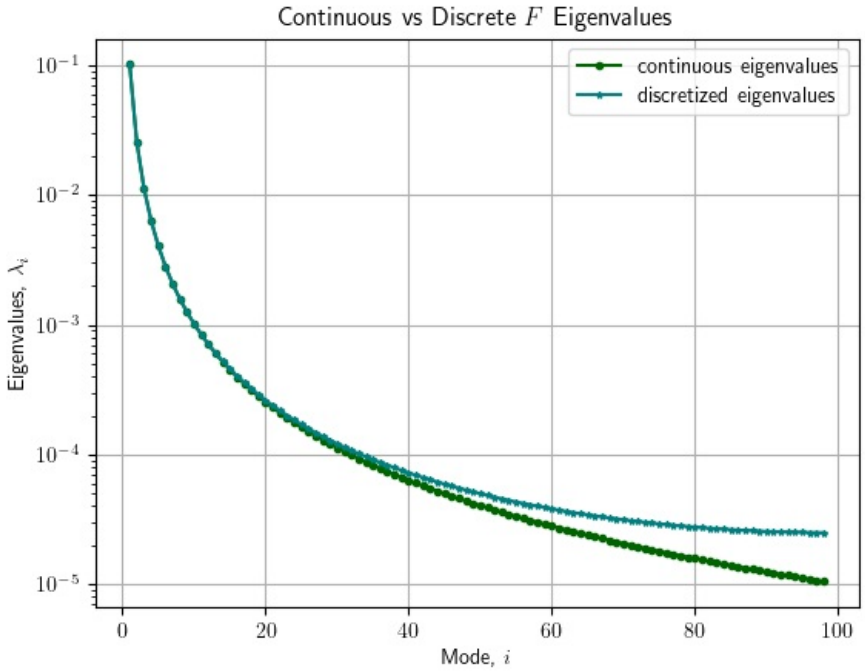
$$[\eta_i]_j = [\mu_i]_j = \sqrt{\frac{2}{L}}\sin\left(\pi i \frac{jh}{L}\right)$$

$i = 1, 2 \ldots n_x - 1$; $j = 1, 2 \ldots n_x - 1$

In [25]:

```
k = 1
l = 1
nx = 100
h = l / nx
L = l
i = np.arange(1, nx-1)
discrete = h**2 / (4 * k * (np.sin(np.pi * i / (2 * nx)))**2)
continuous = L**2 / (i**2 * np.pi**2 * k)
plt.title(r"Continuous vs Discrete $F$ Eigenvalues")
plt.plot(i,continuous, label = "continuous eigenvalues", color = "darkgreen", marker = "o", markersize = 3)
plt.plot(i,discrete, label = "discretized eigenvalues",color = "teal", marker = "*", markersize = 3)
plt.yscale('log')
plt.xlabel(r"Mode, $i$")
plt.ylabel(r"Eigenvalues, $\lambda_{i}$")
plt.legend()
plt.grid()
```

```
plt.show()
```



For small modes (large eigenvalues), the continuous and discrete F eigenvalues match. As the mode increases, however, and the eigenvalues decrease, the discrete F eigenvalues gain more error from the continuous F eigenvalues. As further explained/shown in problem 1d, discretization acts similarly to a regularizer, boosting/increasing small eigenvalues.

2c: poisson inverse problem without regularization

2c,d): Code / algorithm explanation / notes

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

We choose a central finite difference discretization:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1})}{h^2} = \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}$$

where $h = L/n_x$

$$\frac{-K}{h^2}(u_{i-1} - 2u_i + u_{i+1}) = m_i$$

Reduced System ($u_0 = u_{N_x} = 0$):

$$\frac{-K}{h^2}\begin{bmatrix} 2 & -1 & 0 & 0 & & 0 \\ -1 & 2 & -1 & 0 & & 0 \\ 0 & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & & -1 & \\ & & & -1 & 2 & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n_x-1} \end{bmatrix} = \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_{n_x-1} \end{bmatrix}$$

$\underset{\sim}{A}$ constructed by "assemblePoissonMatrix" function

"solvePoisson" solves $\underset{\sim}{A}\vec{u} = \vec{m}$ for $\vec{u}$ (forward pass for model)

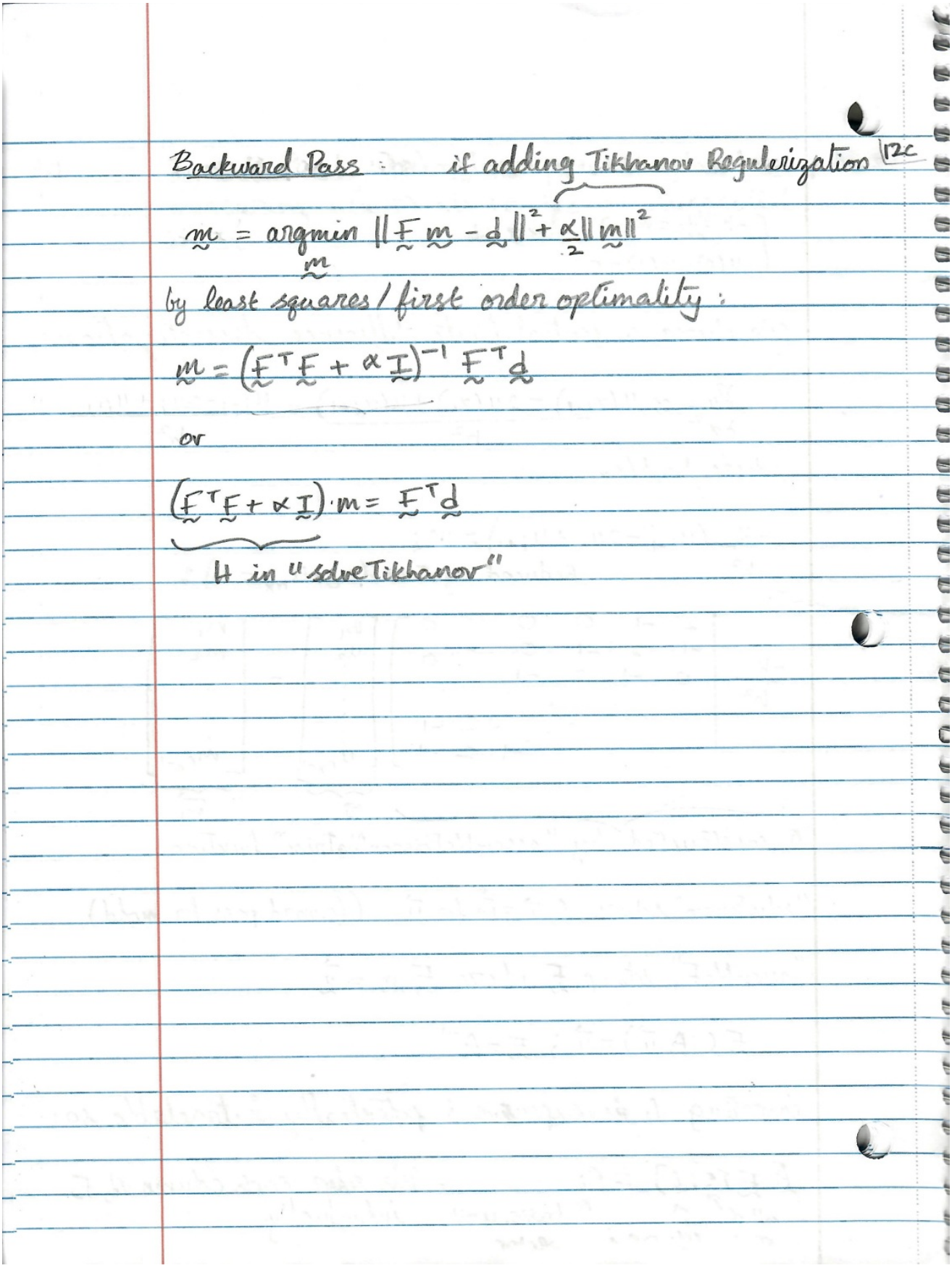
"assembleF" sets up $\underset{\sim}{F}$ where $\underset{\sim}{F}\,\vec{m} = \vec{u}$

$$\underset{\sim}{F}(\underset{\sim}{A}\vec{u}) = \vec{u}\,; \quad \underset{\sim}{F} = \underset{\sim}{A}^{-1}$$

inverting $\underset{\sim}{A}$ is expensive & potentially intractable so:

$\underset{\sim}{A}\,\underset{\sim}{F}[:, i] = e_i$. We solve each column of $\underset{\sim}{F}$ individually.

(: = all of rows; i = column i; e_i = basis, unit vector)



In [26]:
```
import scipy.sparse as sp
import scipy.sparse.linalg as la

# SMALL_SIZE = 20
# MEDIUM_SIZE = 30
# BIGGER_SIZE = 40
# plt.rc('font', size=SMALL_SIZE)
# plt.rc('axes', titlesize=SMALL_SIZE)
# plt.rc('axes', labelsize=SMALL_SIZE)
# plt.rc('xtick', labelsize=SMALL_SIZE)
# plt.rc('ytick', labelsize=SMALL_SIZE)
# plt.rc('legend', fontsize=SMALL_SIZE)
# plt.rc('figure', titlesize=BIGGER_SIZE)

def plot(f, style, **kwargs):
```

```
    x = np.linspace(0., L, nx+1)
    f_plot = np.zeros_like(x)
    f_plot[1:-1] = f
    plt.plot(x,f_plot, style, **kwargs)

def assemblePoissonMatrix(h, n, k):
    diagonals = np.zeros((3, n))   # 3 diagonals for Laplacian
    diagonals[0,:] = -1.0/h**2
    diagonals[1,:] =  2.0/h**2
    diagonals[2,:] = -1.0/h**2
    A = sp.spdiags(diagonals, [-1,0,1], n, n)
    return A * k

def solvePoisson(f, h, n, k):
    A = assemblePoissonMatrix(h, n, k)
    u = la.spsolve(A, f)
    return u

def plot_datagen(m_source, m_inverted, u_true, u_forward, d,noise,nx,k):
    # Plot results
    plt.figure(figsize=(20,10))
    plt.subplot(1,3,1)
    plot(m_source, "-.", color = "indigo", label = r'true $m(x)$')
    plot(m_inverted, "*", color = "cornflowerblue", label = r'inverted $m(x)$' )
    plt.legend(loc = "upper left")
    plt.title(r"Source term $m(x)$")
    plt.xlabel(r"$x$, Location along bar")
    plt.ylabel(r"$\|m(x)\|$")

    plt.subplot(1,3,2)
    plot(u_true, ".-", color = "indigo", label = r"$u(x)$")
    plot(d, "*", markersize = 7, color = "cornflowerblue", label = r"$d(x)$, data")
    plt.legend(loc = "upper left")
    plt.title(r"Solution $u(x)$ vs Data $d(x)$")
    plt.xlabel(r"$x$, Location along bar")
    plt.ylabel(r"$\|u(x)\|$")

    plt.subplot(1,3,3)
    plot(u_true, ".-", color = "indigo", label = r'$u(x)$')
    plot(u_forward, "*", color = "mediumblue", label = r'$u(x)$ forward prediction')
    plot(d, "o",color = "mediumaquamarine", markersize = 2, label = r"$d(x)$, data")
    plt.legend(loc = "upper left")
    plt.title(r"Solution $u(x)$, $d(x)$, and forward prediction")
    plt.xlabel(r"$x$, Location along bar")
    plt.ylabel(r"$\|u(x)\|$")

    plt.suptitle(rf"$\sigma = {noise:.1e},\ n_x = {nx},\ k = {k}$")
    plt.show()
```

In [27]:

```
import matplotlib.cm as cm
def assembleF(k, h, n):
    F = np.zeros((n,n))
    m_i = np.zeros(n)

    for i in np.arange(n):
        m_i[i] = 1.0
        F[:,i] = solvePoisson(m_i, h, n, k)
        m_i[i] = 0.0
    return F

def solveTikhonov(d, F, alpha):
    H = np.dot( F.transpose(), F) + alpha*np.identity(F.shape[1])
    rhs = np.dot( F.transpose(), d)
    return np.linalg.solve(H, rhs)



# Physical parameters for Poisson problem
m_true = lambda x: np.maximum( np.zeros_like(x), 1. - np.abs(1. - 4.*x)) + 100.*np.power(x,10)*np.power(1.-x,2)

# Discretization (default)
L = 1.0
k = 1.0
nx = 100
noise = 10**(-4)
h = L / float(nx)
x = np.linspace(0.+h, L-h, nx-1)

def calcu_m(noise_std_dev, nx, k):

    h = L / float(nx)
    x = np.linspace(0.+h, L-h, nx-1)
    F = assembleF(k, h, nx-1)
```

```
    m_source = m_true(x)
    u_true = solvePoisson(m_source, h, nx-1, k)
    d = u_true + noise_std_dev*np.random.randn(u_true.shape[0])
    m_inverted = solveTikhonov(d, F, alpha=0) # no regularization, naive inversion
    u_forward =  solvePoisson(m_inverted, h, nx-1, k)
    return m_source, m_inverted, u_true, u_forward, d
```

The first set of graphs considers the variation in data (standard deviation of noise addition is σ). Next, we consider the effects of varying rod elastic modulus k on the inverse problem. Finally, we we consider the effects of discretization (n_x) on inversion.

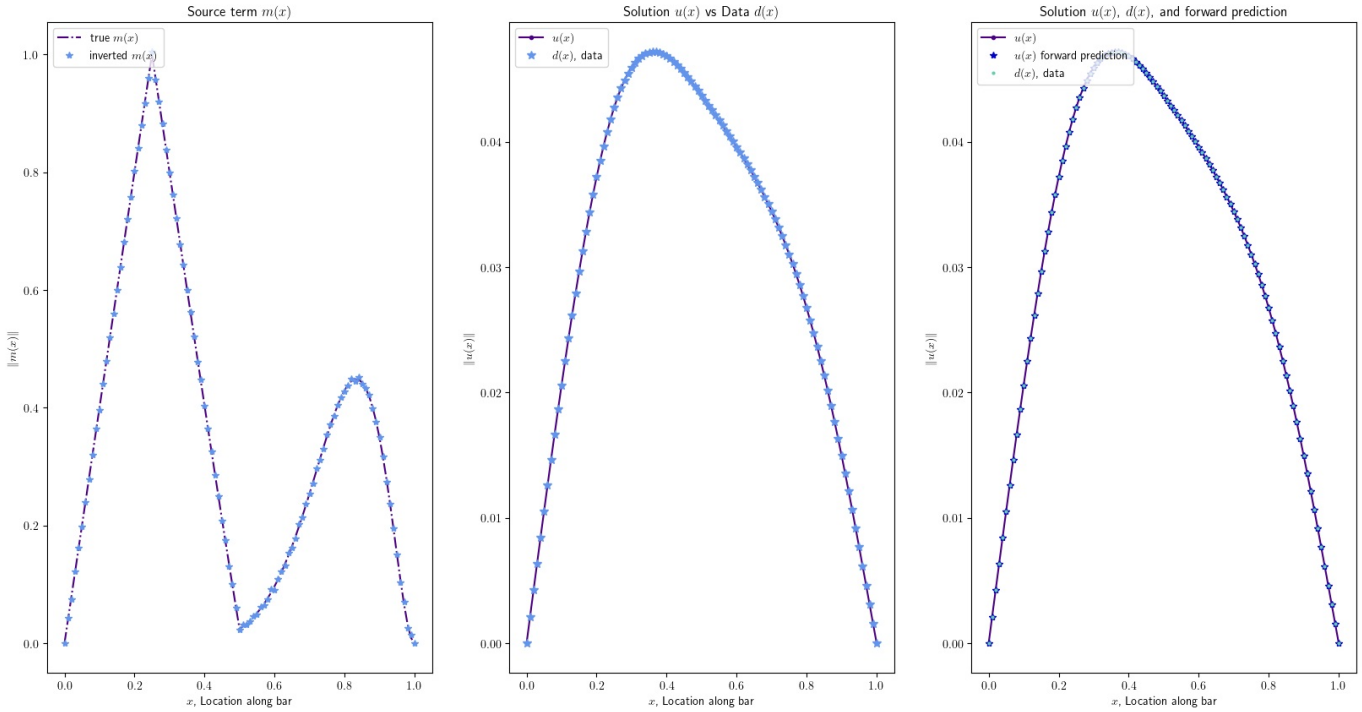
Column 1 shows the source term $m(x)$ and the inverted source term (solution of the inverse problem). Column 2 shows the known displacement (forward prediction of true $m(x)$) $u(x)$ and the data $d(x)$ formed by adding error to $u(x)$. Column 3 shows the difference between the true displacement $u(x)$, the forward prediction of the inverted source term, and the data used for inversion $d(x)$.

variation in σ

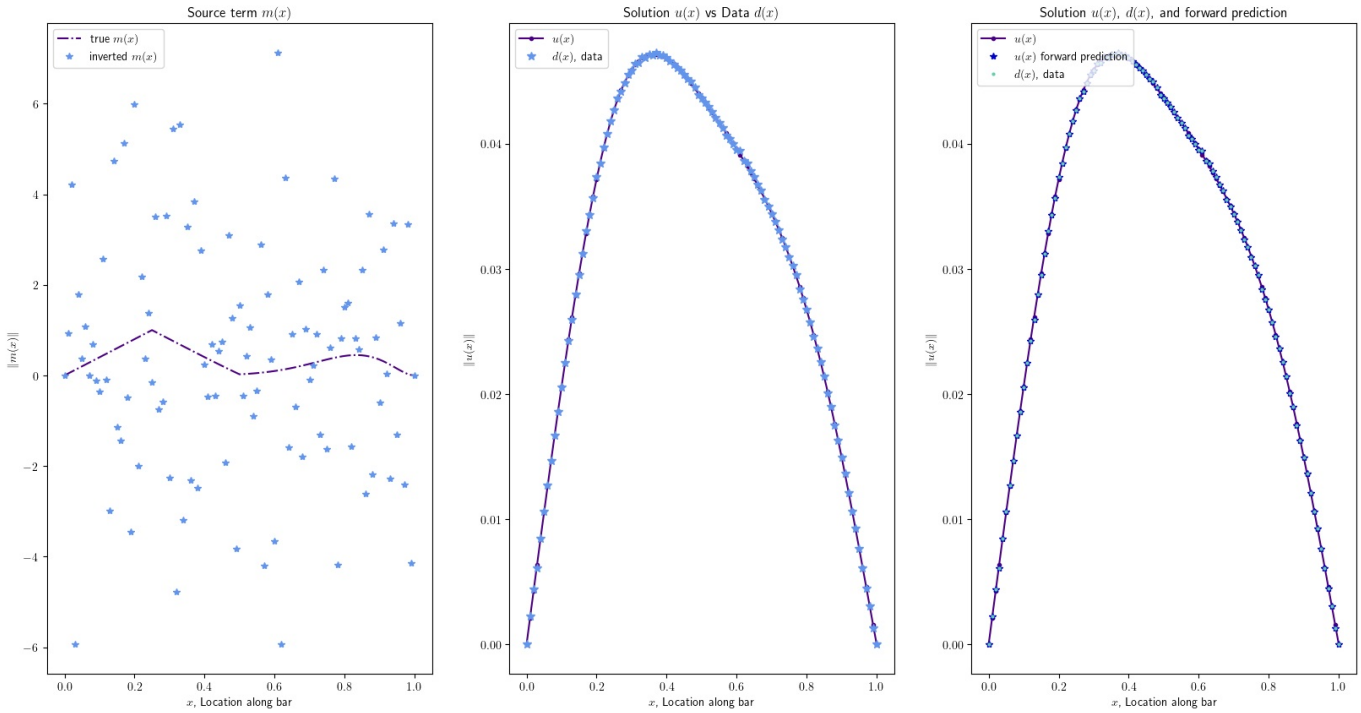
In [28]:
```
noises = [10**(-7), 10**(-4), 10**(-2)]
for noise in noises:
    plot_datagen(*calcu_m(noise,nx,k),noise,nx,k)
```

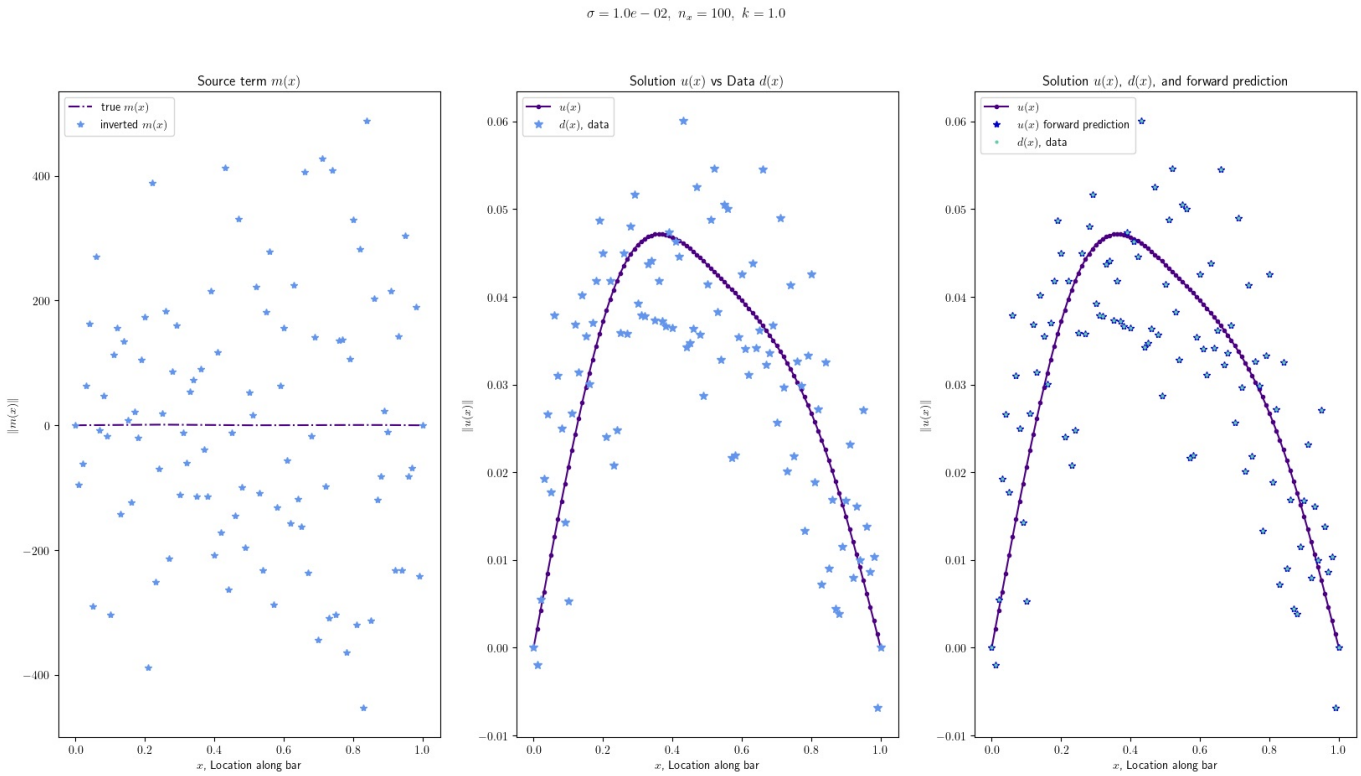
```
/tmp/ipykernel_29516/1978269560.py:31: SparseEfficiencyWarning: spsolve requires A be CSC or CSR matrix format
  u = la.spsolve(A, f)
```

$\sigma = 1.0e-07,\ n_x = 100,\ k = 1.0$



$\sigma = 1.0e-04,\ n_x = 100,\ k = 1.0$





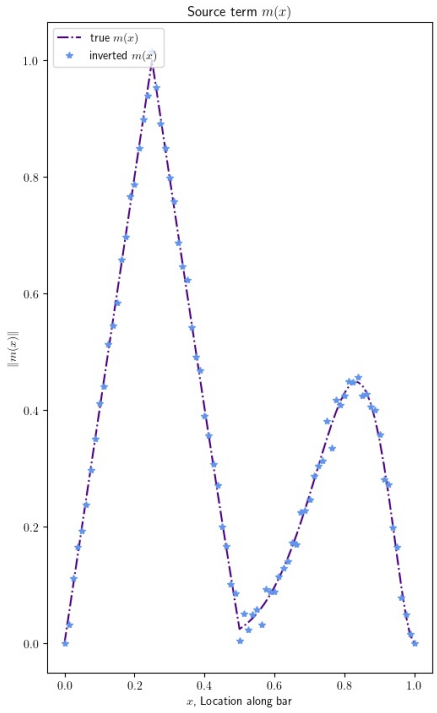
As expected, increases in the noise of data $d(x)$ cause increases in the error of inverted $m(x)$. We notice that for large σ, the inverted $m(x)$ is quite unstable (large magnitudes $\|m(x)\|$). However, despite the large noise, the predicted displacement (based on inverted $m(x)$) matches the data field $d(x)$ quite well, suggesting that the error in $m(x)$ is due to the data error (σ) (model overfits to error) rather than inherent model/physics/computational error.

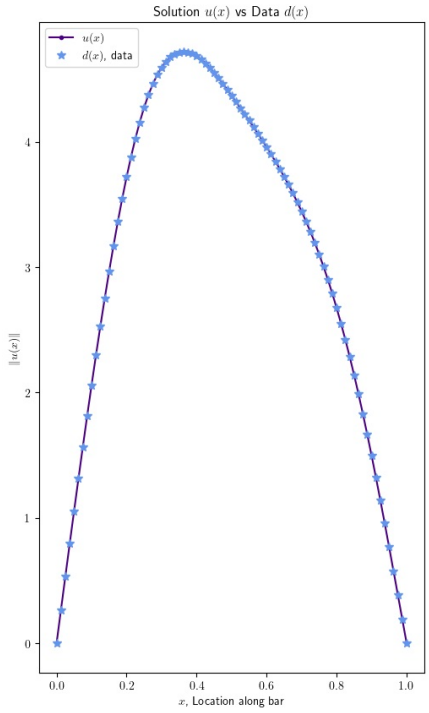
variation in k

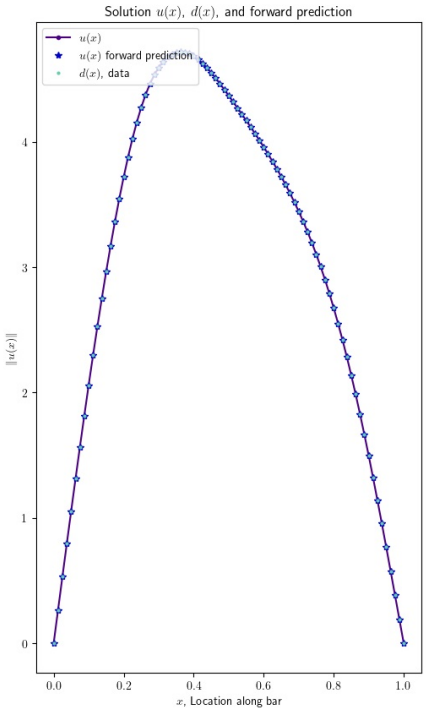
In [29]:
```
noise = 10**(-4) # reset to default
nx = 80
ks = [10**(-2), 1, 10**(2)]
for k in ks:
    plot_datagen(*calcu_m(noise,nx,k),noise,nx,k)
```

```
/tmp/ipykernel_29516/1978269560.py:31: SparseEfficiencyWarning: spsolve requires A be CSC or CSR matrix format
  u = la.spsolve(A, f)
```

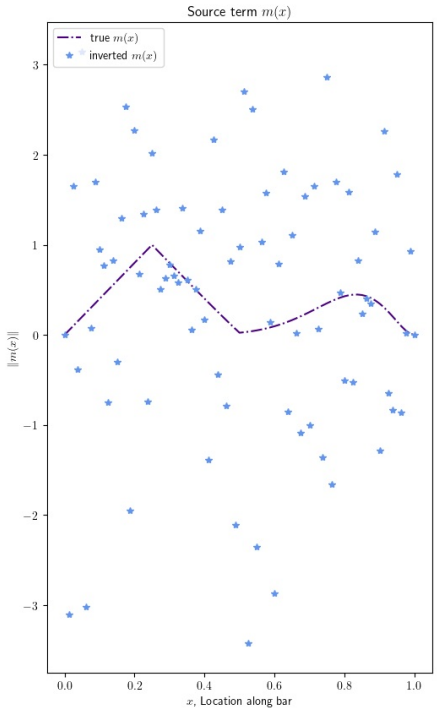
$\sigma = 1.0e-04,\ n_x = 80,\ k = 0.01$

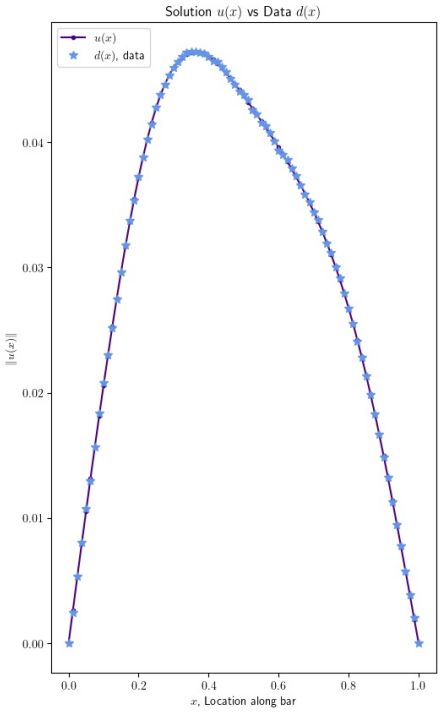


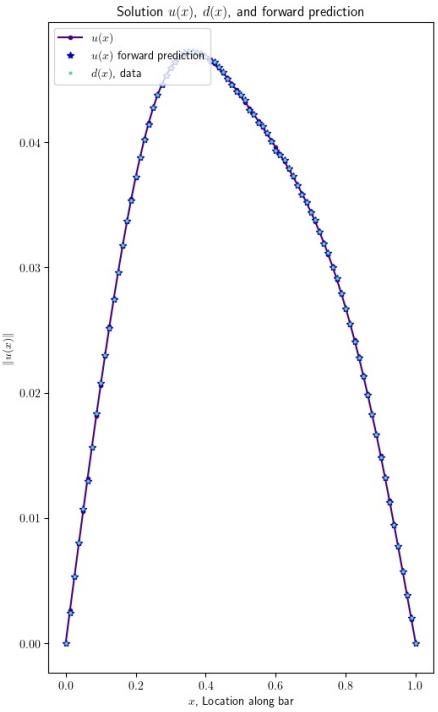


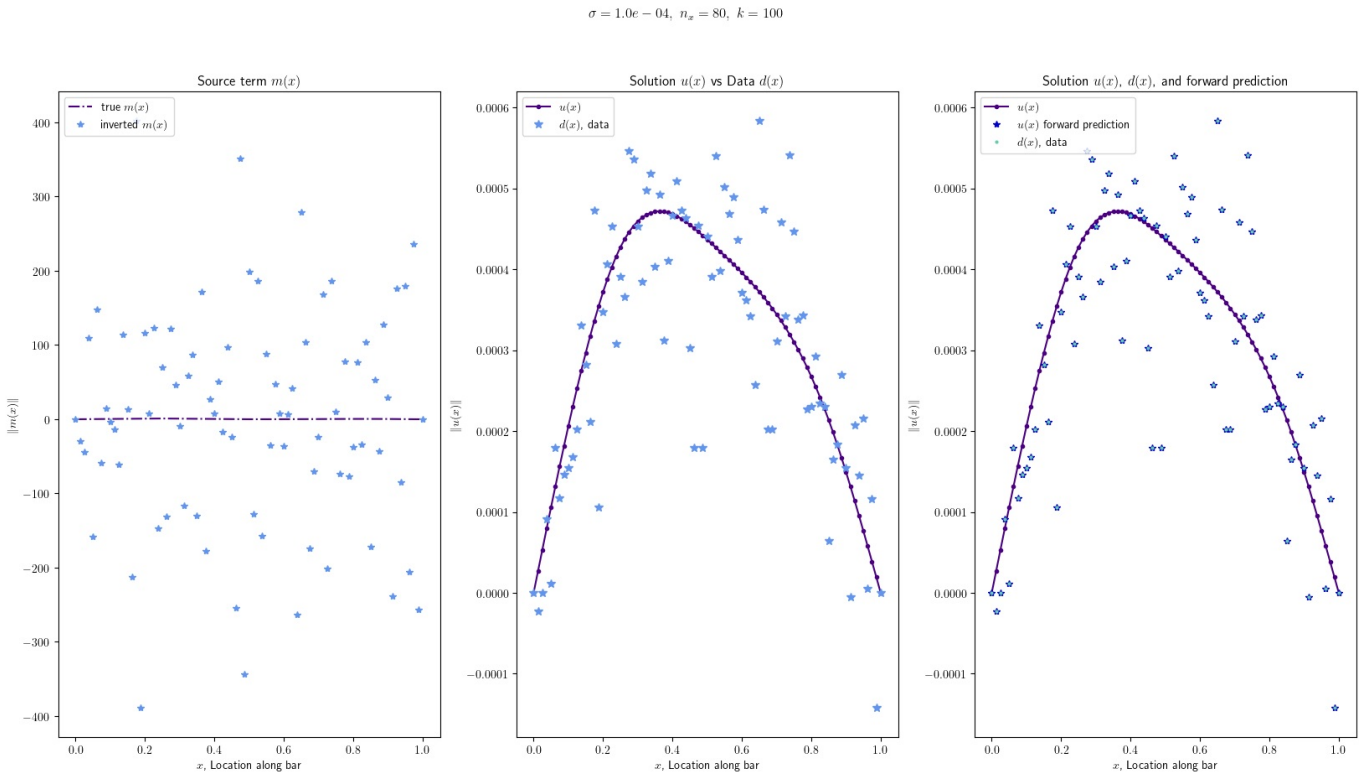


$\sigma = 1.0e-04,\ n_x = 80,\ k = 1$









Increased stiffness k under non-zero standard deviation of noise σ leads to increased noise in the inverted $m(x)$. Notice the relative noise of the data is also increased; increased stiffness leads to overall reduced magnitude of displacement ($\|u(x)\|$) but σ is constant. For small measurements of displacement, any error in measurement becomes highly obvious (signal to noise ratio is reduced). With more faulty/erraneous data, inversion for $m(x)$ becomes unstable. Additionally, since the eigenvalues of the P2O map, F, are proportional to k^{-1} , for increases in k, the eigenvalues are reduced, further contributing to the instability of of the inverse problem.

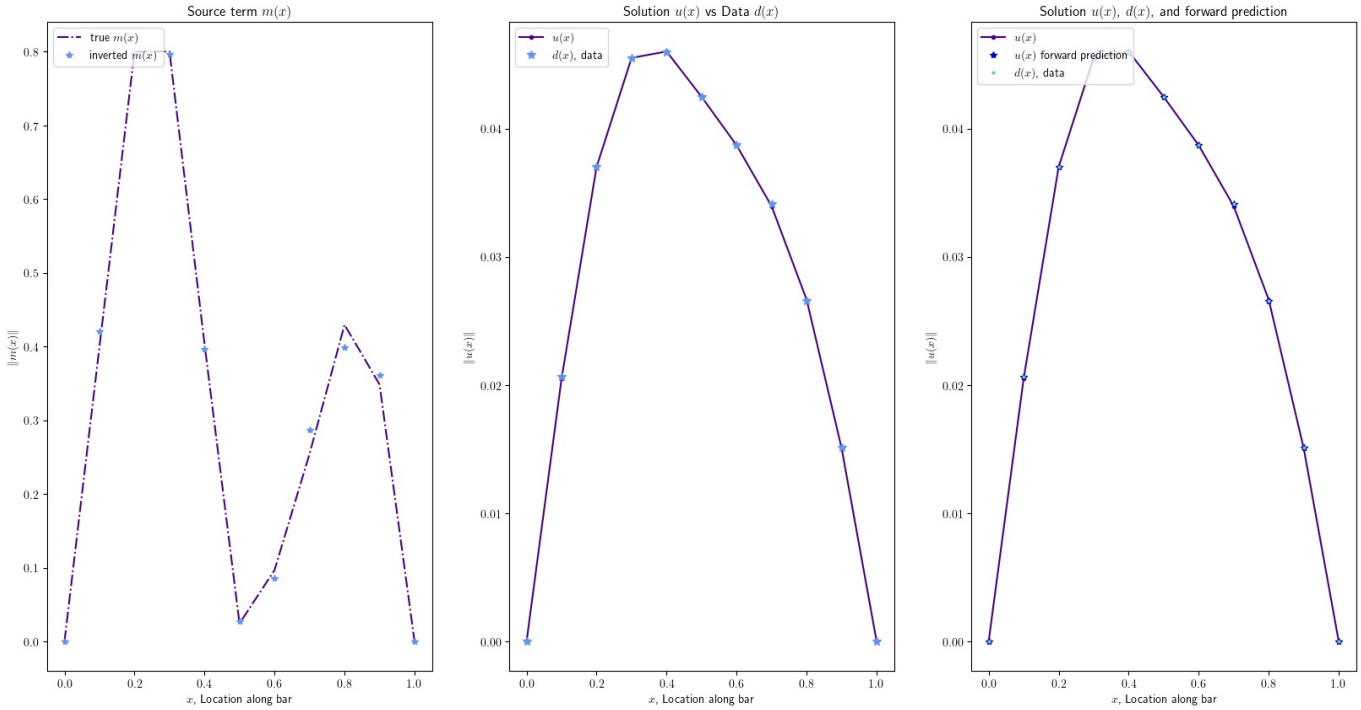
The predicted displacements and the noisy observed data still match fairly well, suggesting that the model is again overfitting to the noise of the data, as in the above example.

variation in discretization n_x

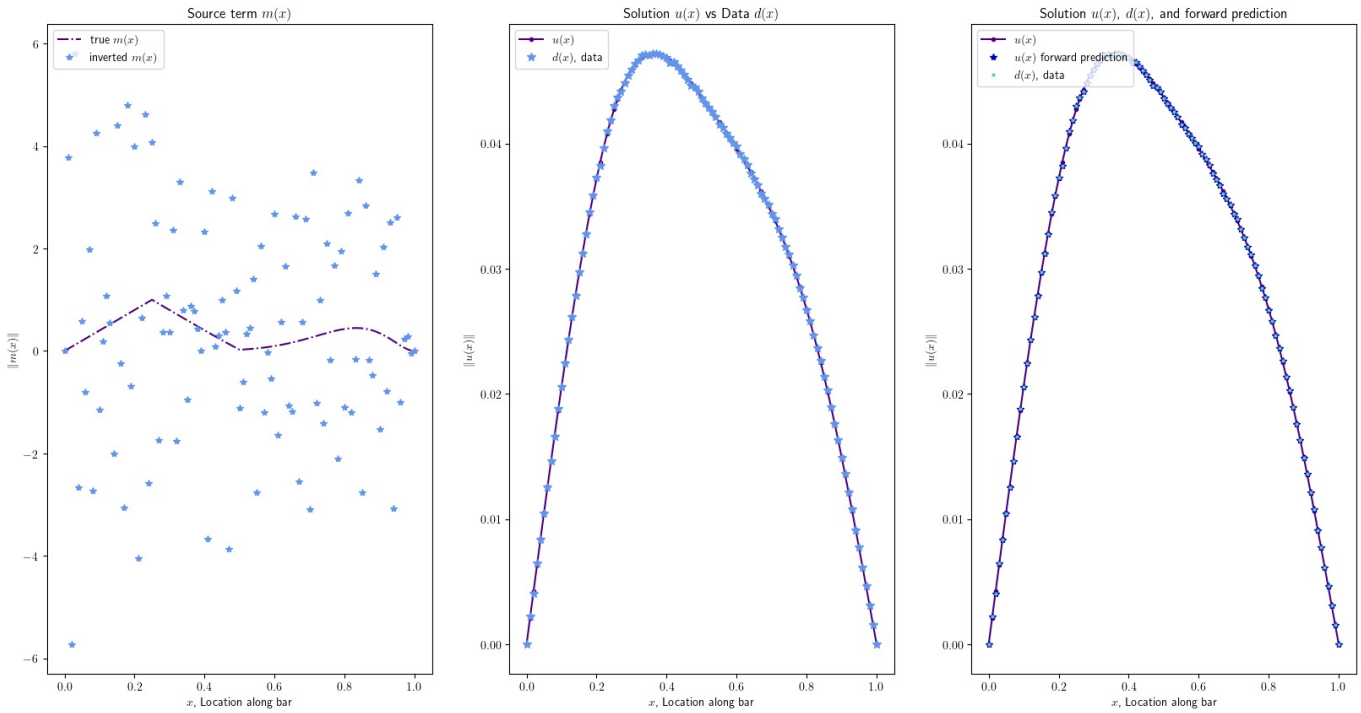
In [30]:
```
k = 1 # reset to default
nxs = [10, 100, 500]
for nx in nxs:
    plot_datagen(*calcu_m(noise,nx,k),noise,nx,k)
```

```
/tmp/ipykernel_29516/1978269560.py:31: SparseEfficiencyWarning: spsolve requires A be CSC or CSR matrix format
  u = la.spsolve(A, f)
```

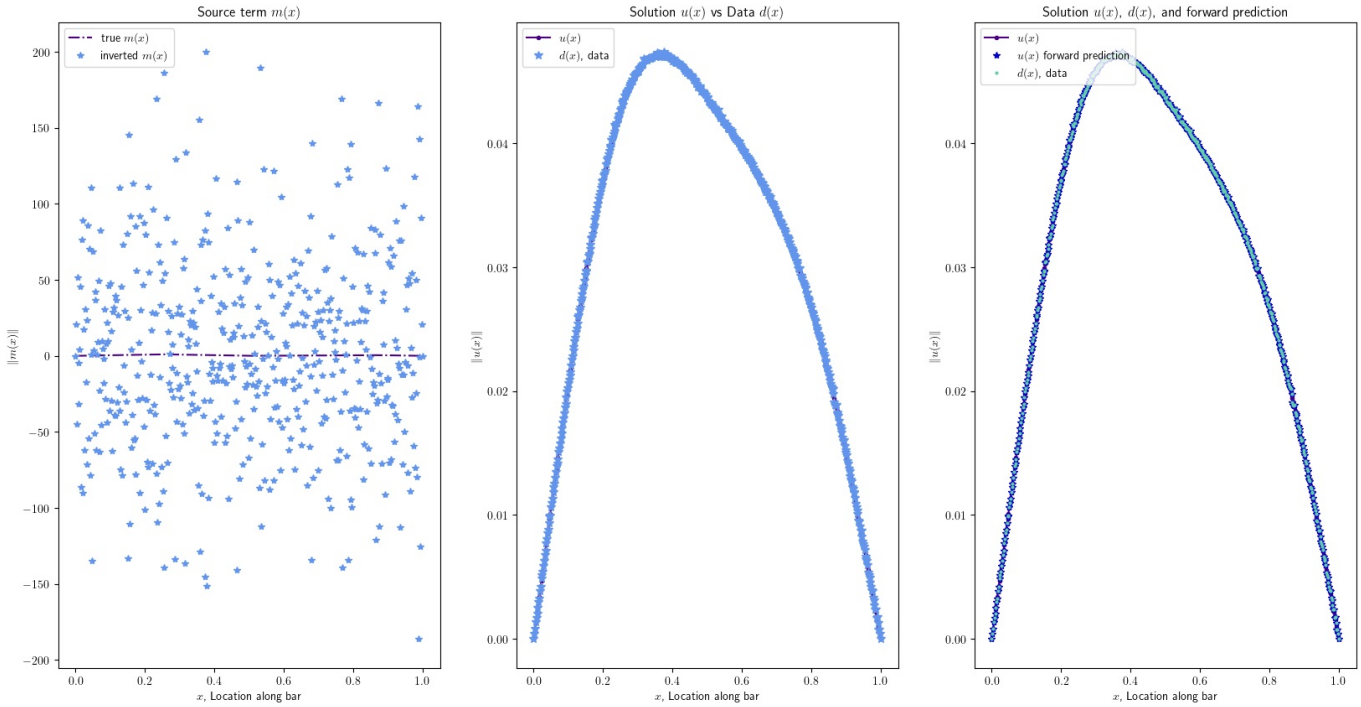
$\sigma = 1.0e-04,\ n_x = 10,\ k = 1$



$\sigma = 1.0e-04,\ n_x = 100,\ k = 1$



$\sigma = 1.0e-04,\ n_x = 500,\ k = 1$



Inversion for $m(x)$ is most accurate at coarse discretizations ($n_x = 10$ is more accurate than $n_x = 100, 500$). As suggested in problem 1, coarse discretization acts similar to a regularizer.

Notice that the discrete eigenvalues decay as the mode increases from 1 to $n_x - 1 : \lambda_i \propto \frac{1}{\sin^2\left(\frac{\pi i}{2n_x}\right)}$ for $i = 1, 2, \ldots . n_x - 1$.

For small n_x, the smallest eigenvalue ($i = n_x - 1$) is proportional to $\frac{1}{\sin^2\left(\frac{\pi i}{2n_x}\right)}$ which is relatively large (>1), but for larger n_x, the smallest eigenvalue ($i = n_x - 1$) is proportional to $\frac{1}{\sin^2\left(\frac{\pi i}{2n_x}\right)}$ which is much smaller and approaches 1 (although 1 is not necessarily a small eigenvalue, consider that the eigenvalue is also proportional to h^2). Inversion error is caused by small eigenvalues (error is proportional to inverse of the eigenvalue), so coarse discretizations with large eigenvalues correspond to less ill-posedness. Additionally, for coarser meshes, there is reduced alignment between the high mode eigenvectors (not stored/considered) and the error associated with gaussian noise (white noise has high modes).

For fine discretizations, the condition number of the discretized Poisson matrix (P2O map) also increases, meaning that the inverse problem becomes more ill-conditioned. For larger systems, round-off errors can also increase (especially when dealing with small eigenvalues), ultimately leading to instable $m(x)$ inversion.

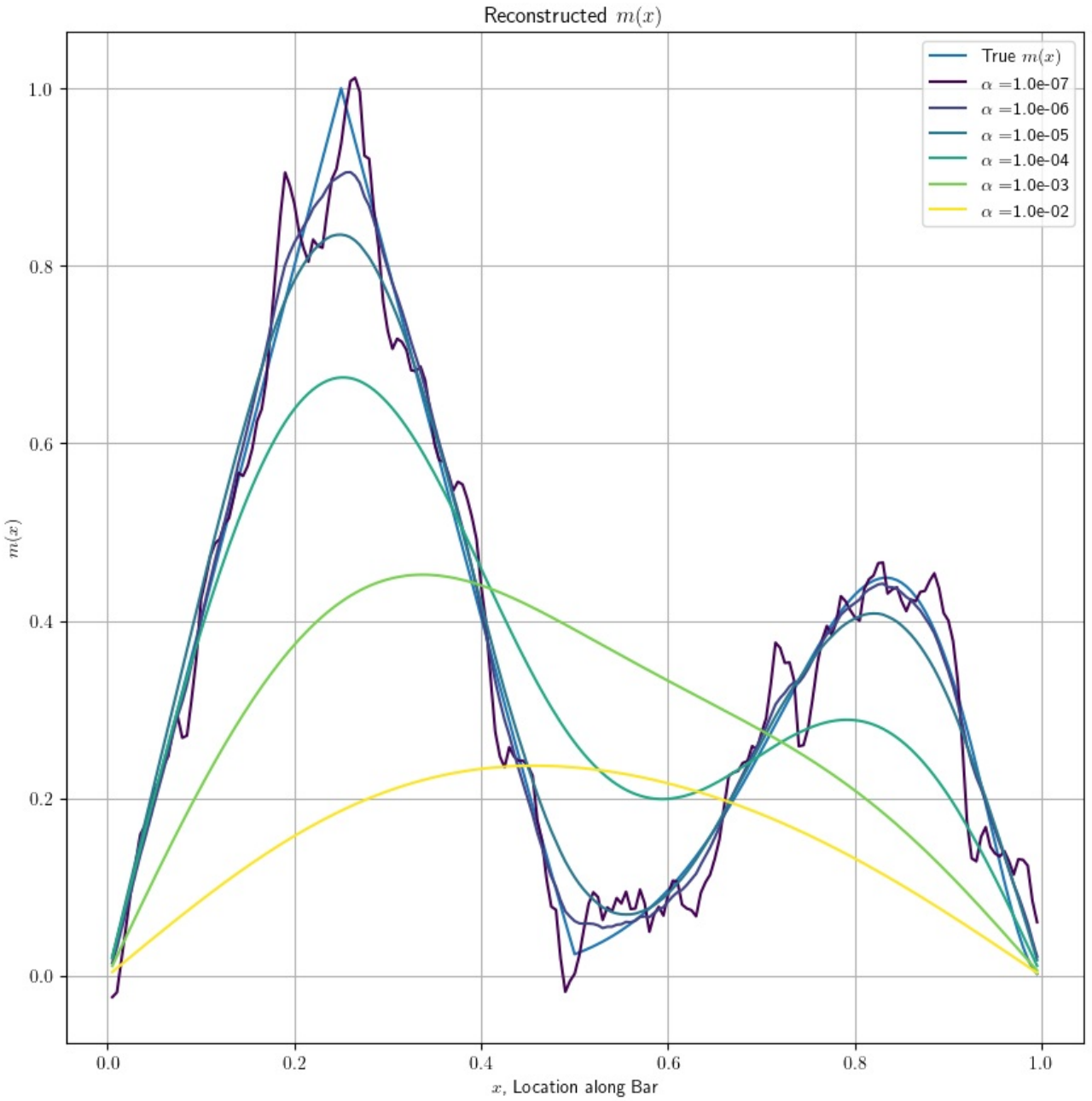
Note that the predicted displacements (per inverted $m(x)$) and the noisy observed data still match fairly well for both the coarse and fine discretizations. This suggests while the inverse problem might be highly ill-conditioned, the forward problem is well-posed.

2d: Tikhanov regularized solution ("eyeball" norm)

In [35]:
```
# reset all to standard parameters + nx = 200
k = 1
noise = 10**(-4)
nx = 200

h = L / float(nx)
x = np.linspace(0.+h, L-h, nx-1)
F = assembleF(k, h, nx-1)
u_true = solvePoisson(m_true(x), h, nx-1, k)
d = u_true + noise*np.random.randn(u_true.shape[0])

# Plotting
plt.figure(figsize=(10, 10))
cmap = plt.colormaps.get_cmap('viridis')
plt.plot(x, m_true(x), "-", label= r"True $m(x)$")
```

```
alphas = np.logspace(-7,-2,6)
for i, alpha in enumerate(alphas):
    m_alpha = solveTikhonov(d, F, alpha)
    plt.plot(x, m_alpha, "-", color=cmap(i / (len(alphas) - 1)), label= r'$\alpha = $' + f"{alpha:.1e}")


plt.legend(loc = "upper right")
plt.title(r"Reconstructed $m(x)$")
plt.ylabel(r"$m(x)$")
plt.xlabel(r"$x$, Location along Bar")
plt.grid()
plt.show()
```

```
/tmp/ipykernel_29516/1978269560.py:31: SparseEfficiencyWarning: spsolve requires A be CSC or CSR matrix format
  u = la.spsolve(A, f)
```



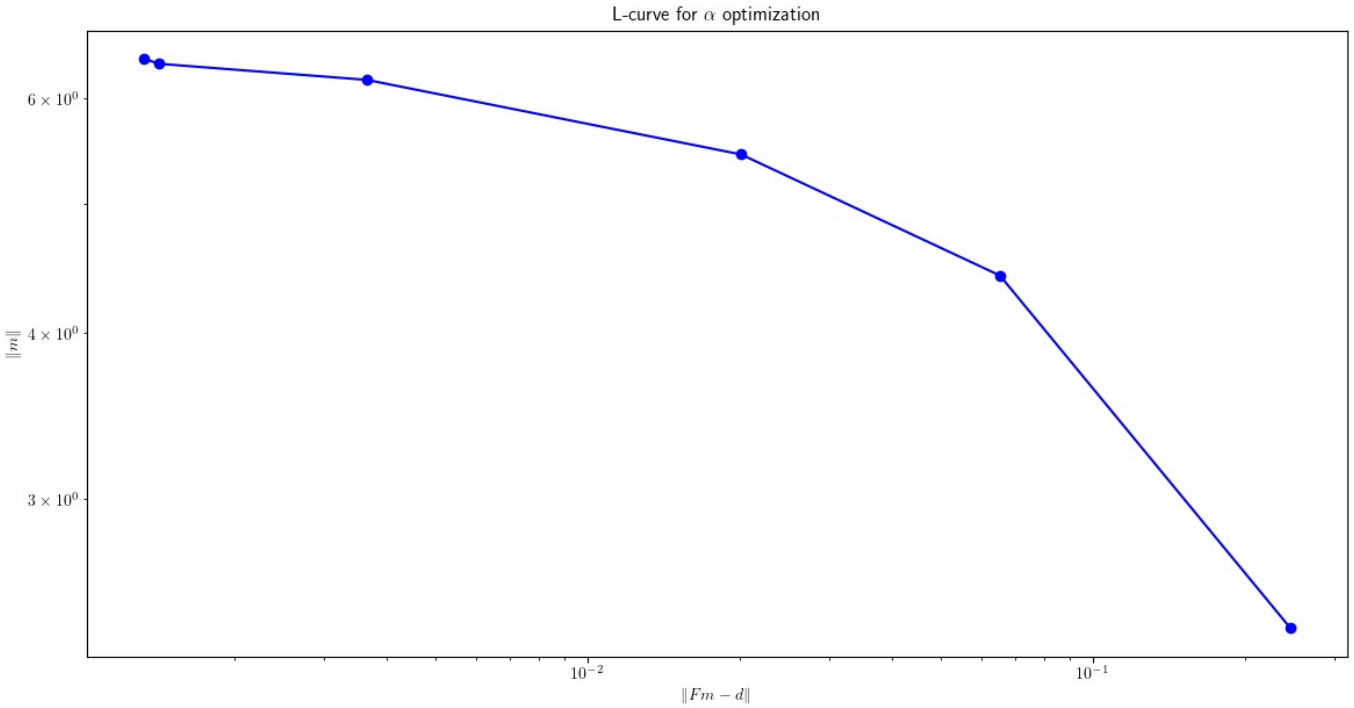
Visually, we expect the ideal regularization α to be between 10^{-6} and 10^{-7}, possibly closer to 10^{-6}.

2e: L-curve for α optimization

In [39]:
```
norm_m = [] #norm of parameter
norm_r = [] #norm of misfit (residual)
plt.figure(figsize=(14,7))
for alpha in alphas:
    m_alpha = solveTikhonov(d, F, alpha)
    norm_m.append( np.sqrt( np.dot(m_alpha,m_alpha) ) )
    u_alpha = solvePoisson(m_alpha, h, nx-1, k)
    norm_r.append( np.sqrt( np.dot(d-u_alpha,d-u_alpha) ) )

plt.loglog(norm_r, norm_m, "-ob")
```

```
plt.xlabel(r"$\|Fm - d\|$")
plt.ylabel(r"$\| m \|$")
plt.title(r"L-curve for $\alpha$ optimization")
plt.show()
```

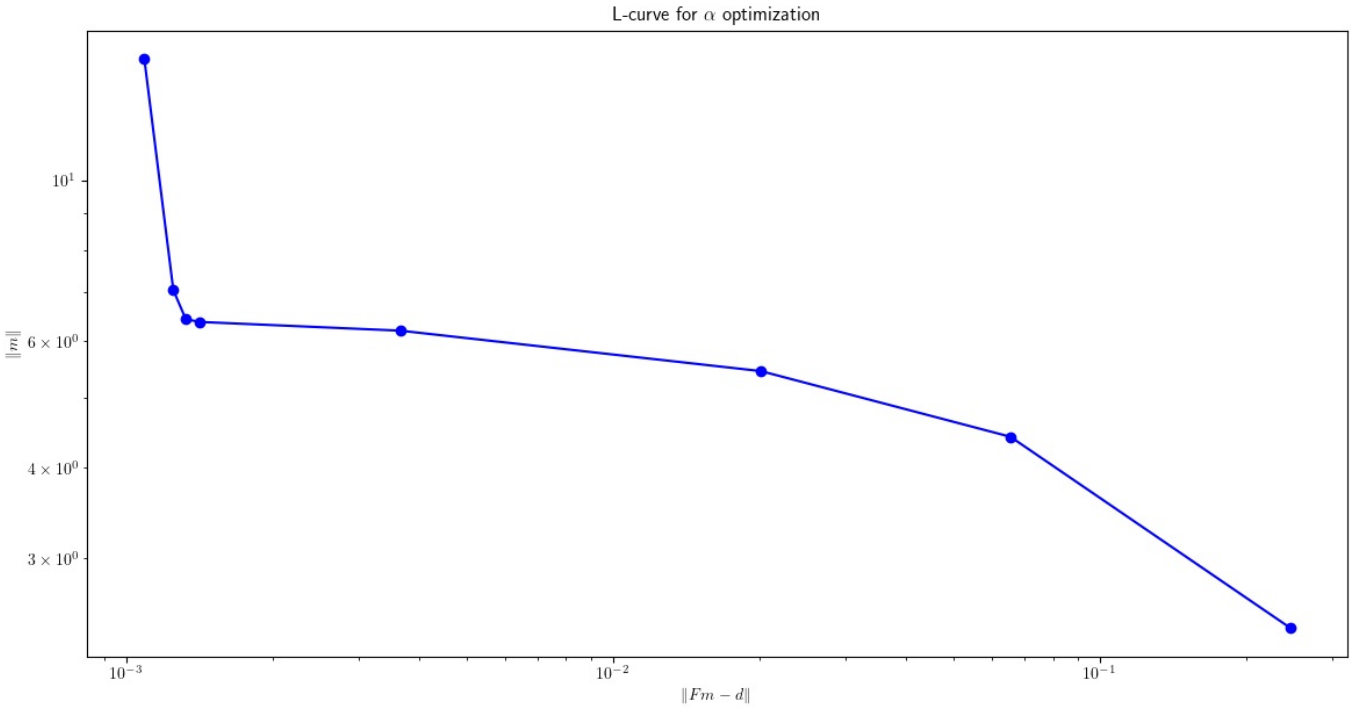
```
/tmp/ipykernel_29516/1978269560.py:31: SparseEfficiencyWarning: spsolve requires A be CSC or CSR matrix format
  u = la.spsolve(A, f)
```



Per the L-curve, $\alpha \approx 10^{-7}$ is the most optimal. It is difficult to see the "elbow" in the L-curve for the above graph, so we plot a few more regularization points to confirm.

In [41]:
```
norm_m = [] #norm of parameter
norm_r = [] #norm of misfit (residual)
plt.figure(figsize=(14,7))
for alpha in np.logspace(-9,-2,8):
    m_alpha = solveTikhonov(d, F, alpha)
    norm_m.append( np.sqrt( np.dot(m_alpha,m_alpha) ) )
    u_alpha = solvePoisson(m_alpha, h, nx-1, k)
    norm_r.append( np.sqrt( np.dot(d-u_alpha,d-u_alpha) ) )

plt.loglog(norm_r, norm_m, "-ob")
plt.xlabel(r"$\|Fm - d\|$")
plt.ylabel(r"$\| m \|$")
plt.title(r"L-curve for $\alpha$ optimization")
plt.show()
```

```
/tmp/ipykernel_29516/1978269560.py:31: SparseEfficiencyWarning: spsolve requires A be CSC or CSR matrix format
  u = la.spsolve(A, f)
```

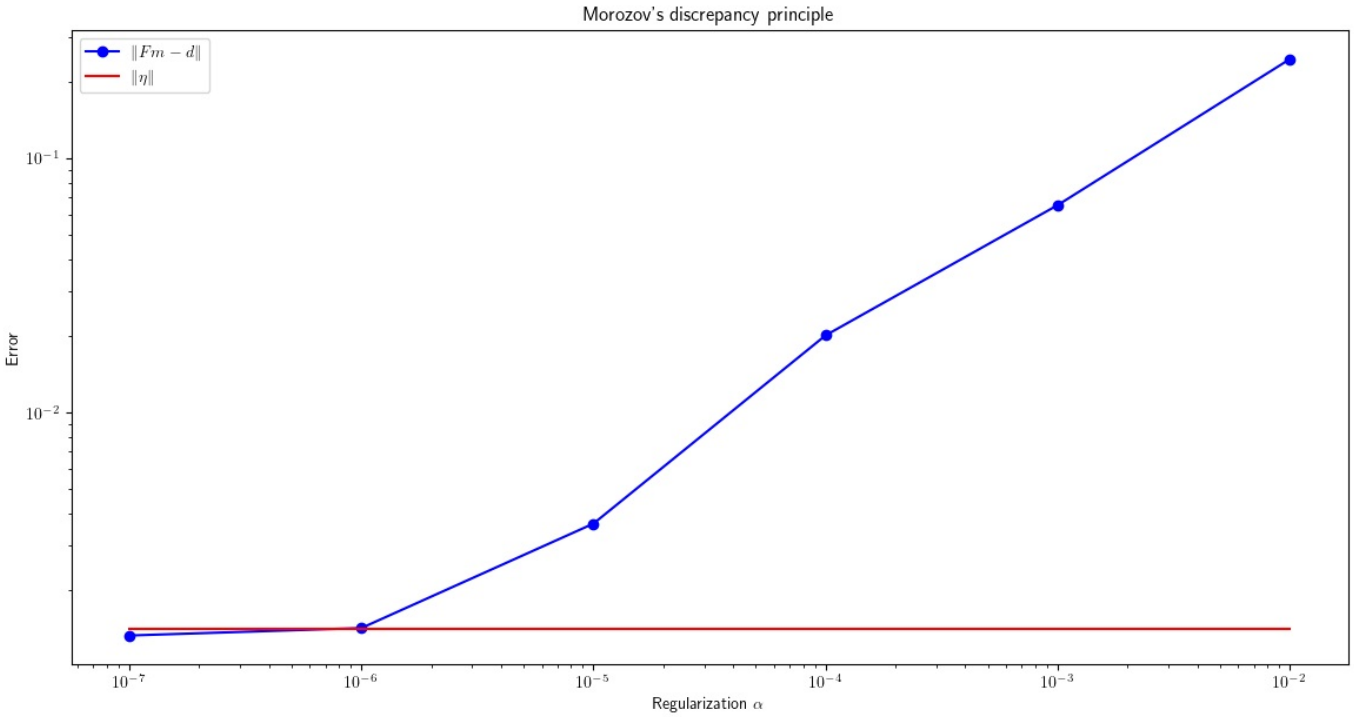


This confirms that, per the L-curve, $\alpha \approx 10^{-7}$ is the most optimal.

2f: Morozov discrepency for α optimization

In [42]:
```
# alphas = [1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3]
norm_r = [] #norm of misfit (residual)
plt.figure(figsize=(14,7))
for alpha in alphas:
    m_alpha = solveTikhonov(d, F, alpha)
    u_alpha = solvePoisson(m_alpha, h, nx-1, k)
    norm_r.append( np.sqrt( np.dot(d-u_alpha,d-u_alpha) ) )

plt.loglog(alphas, norm_r, "-ob", label=r"$\|Fm - d\|$")
plt.loglog(alphas, [noise*np.sqrt(nx-1)]*len(alphas), "-r", label=r"$\|\eta \|$")
plt.xlabel(r"Regularization $\alpha$")
plt.ylabel("Error")
plt.legend()
plt.title("Morozov's discrepancy principle")
plt.show()
```

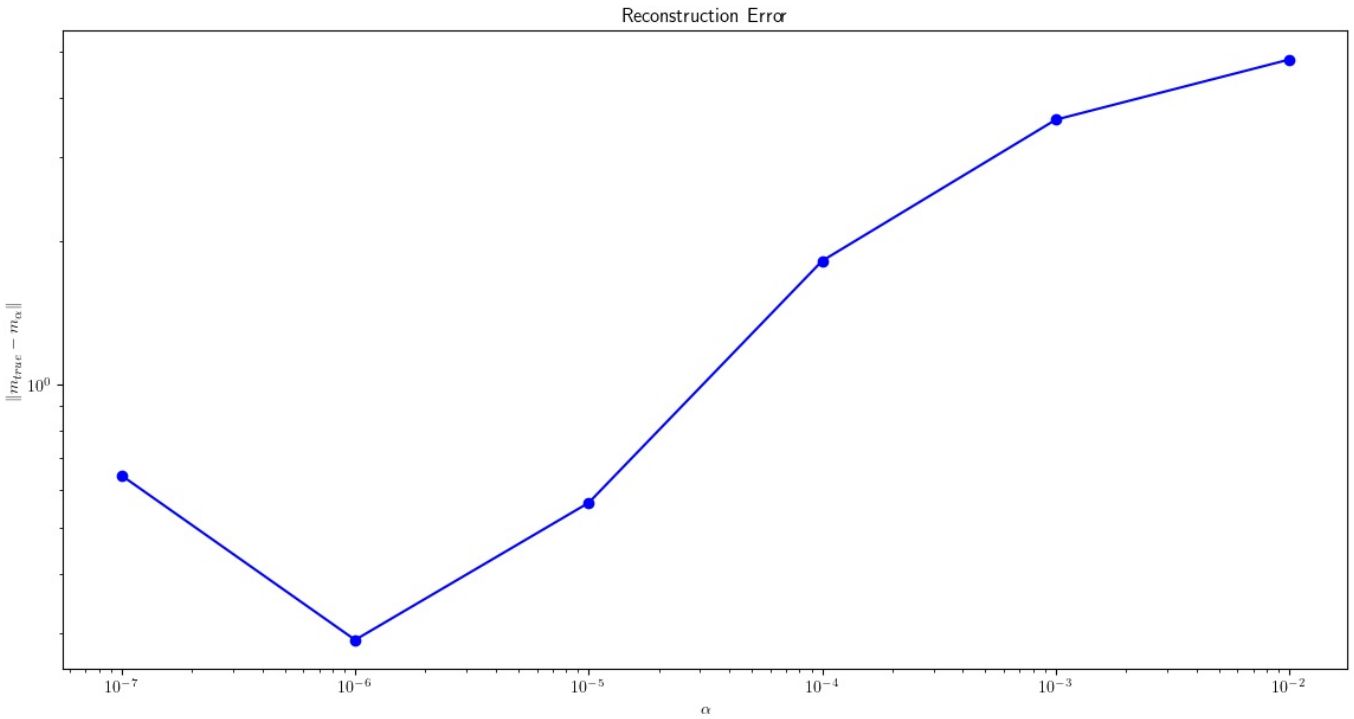
```
/tmp/ipykernel_29516/1978269560.py:31: SparseEfficiencyWarning: spsolve requires A be CSC or CSR matrix format
  u = la.spsolve(A, f)
```



Per the Morozov discrepancy, α is between 10^{-7} and 10^{-6} (closer to 10^{-6}). We say $\alpha \approx 10^{-6}$ is the most optimal.

2g: L2 Norm reconstruction to decide α

In [43]:
```
err_m = [] #norm of misfit (residual)
true_m = m_true(x)
plt.figure(figsize=(14,7))
for alpha in alphas:
    m_alpha = solveTikhonov(d, F, alpha)
    err_m.append( np.sqrt( np.dot(true_m-m_alpha,true_m-m_alpha) ) )

plt.loglog(alphas, err_m, "-ob")
plt.xlabel(r"$\alpha$")
plt.ylabel(r"$\|m_{true} - m_{\alpha}\|$")
plt.title("Reconstruction Error")
plt.show()
```



Per the L2 norm error in the reconstruction, $\|m_{true} - m_{\alpha}\|$, $\alpha \approx 10^{-6}$ is the most optimal.

The regularizers are summarized as: $\alpha_{eyeball} \approx 10^{-6}$, $\alpha_{L-curve} \approx 10^{-7}$, $\alpha_{Morozov} \approx 10^{-6}$, and $\alpha_{L2} \approx 10^{-6}$.

The most optimal value is $\alpha_{Morozov}$ as it considers the data's error when choosing the regularization magnitude. Any model error $\|Fm - d\|$ within the data error $\|\eta\|$ should be allowable (no need to reduce regularization). Unlike the "eyeball approximation" or L-curve method, both of which are heurestical, the Morozov method provides the most intuitive α which considers all information (specifically, the error of the data) and has explicit statistical justification. Since we know $\|\eta\|$, the Morozov method is most appropriate.

All methods--eyeball, L-curve, Morozov, and L2 reconstruction--are mostly in accordance ($\alpha \approx 10^{-6}$). The L-curve method is the only method that suggests a slightly different ($\alpha \approx 10^{-7}$), but this can be easily explained since: 1) only few choices are plotting so the curvature could very easily be maximized at 10^{-6} rather than at 10^{-7} and 2) the log-log scale makes curvature approximations difficult to observe, especially with limited points on the L-curve. Note that the L-curve method balances parameter (m) stability versus model fidelty while the Morozov method is more concerned with the inherent data error ($\|\eta\|$) and the model fidelty (indirectly seeks to stabilize m by choosing maximum regularization). Note that the L-curve method balances parameter (m) stability versus model fidelty while the Morozov method is more concerned with the inherent data error $\|\eta\|$ and the model fidelty (indirectly seeks to stabilize m by choosing maximum regularization).

Problem 3

3a: eigenvalues, eigenfunctions of continuous P2O map

3. $$\begin{cases} \frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0 \; ; 0 < x < L, \; 0 < t \leq T \\ u(x,0) = m(x) \quad ; 0 < x < L \\ \frac{\partial u}{\partial t}(x,0) = 0 \quad ; 0 < x < L \\ u(0,t) = u(L,t) = 0 \; ; 0 < t \leq T \end{cases}$$

(c is a constant, wave propagation speed)

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

a) Derive expressions for the eigenvalues λ_i & the eigenfunctions $v_i(x)$ of the continuous operator $\mathcal{F}$.

Solution:

As in problem 1, we implement a seperation-of-variables solution:

$$u(x,T) = X(x)\,T(t)$$ ← or some linear combination of $X(x) \cdot T(t)$.

$$\frac{\partial^2 u}{\partial t^2} = X(x)\,T''(t) \; ; \; \frac{\partial^2 u}{\partial x^2} = X''(x)\,T(t)$$

Substituting back into the PDE:

$$X(x)\,T''(t) - c^2 X''(x)\,T(t) = 0$$

$$\Rightarrow \underbrace{\frac{X(x)}{c^2 X''(x)}}_{\text{function of } x} = \underbrace{\frac{T(t)}{T''(t)}}_{\text{function of } t} = \underbrace{-1/\mu}_{\text{constant}}$$

We now have to ODEs:

1) $\frac{c^2 X''(x)}{X(x)} = -\mu$

2) $\frac{T''(t)}{T(t)} = -\mu$

1): Spatial $X(x)$ solution: $c^2 X''(x) + \mu X(x) = 0$

characteristic equation:

$c^2 r^2 + \mu = 0$

$r = \pm\sqrt{\frac{\mu}{c^2}}\, i$

$X(x) = c_1 e^{-\sqrt{\frac{\mu}{c^2}}\, i\, x} + c_2 e^{\sqrt{\frac{\mu}{c^2}}\, i\, x}$

$= A\cos\left(\sqrt{\frac{\mu}{c^2}}\, x\right) + B\sin\left(\sqrt{\frac{\mu}{c^2}}\, x\right)$

where A, B are constants.

Now, for the Boundary Conditions:

$u(0,t) = u(L,t) = 0 \quad \forall\, t \in (0,T]$

if $u(0,t) = X(0)\,T(t) = 0 \quad \forall\, t \in (0,T]$,

$= X(L)\,T(t)$

then either:

1. $X(0) = X(L) = 0$ or:
2. $T(t) = 0 \quad \forall\, t \in (0,T]$.

The 2nd case would lead to the trivial solution $u(x,t) = X(x) \cdot 0 = 0$.

So, we consider the Boundary Conditions:

$$X(0) = X(L) = 0$$

Substituting into the general $X(x)$ equation:

$$X(0) = A\cos(0) + B\sin(0) = 0 \; ; \; A = 0$$

($\sin(0) = 0$)

$$X(L) = B\sin\left(\sqrt{\frac{\mu}{c^2}}\, L\right) = 0$$

if $B = 0$, then we have the trivial solution $X(x) = 0 \Rightarrow u(x,t) = 0$.

So, $B \neq 0$; $\sin\left(\sqrt{\frac{\mu}{c^2}}\, L\right) = 0$; $\sqrt{\frac{\mu}{c^2}}\, L = i\pi$, $i = 1, 2, 3 \ldots$

($i \neq 0$; since, if $i = 0$, $\mu = 0$ & $X(x) = 0 \Rightarrow u(x,t) = 0$)

$$\sqrt{\frac{\mu}{c^2}} = \frac{i\pi}{L} \Rightarrow \mu = \left(\frac{ci\pi}{L}\right)^2 \quad i = 1, 2, 3 \ldots$$

$$X(x) = B\sin\left(\frac{i\pi x}{L}\right) ; \; i = 1, 2 \ldots$$

Now for the temporal ODE:

$$2): \; T''(t) + \mu T(t) = 0$$

characteristic equation:

$$r^2 + \mu = 0$$

$$r = \pm\sqrt{\mu}\, i$$

$$T(t) = P\cos\left(\sqrt{\mu}\, t\right) + Q\sin\left(\sqrt{\mu}\, t\right)$$

For Boundary conditions:

$$\frac{\partial u}{\partial t}(x,0)=0$$

$$\frac{\partial u}{\partial T}(x,t)= X(x)\,T'(t) \Rightarrow X(x)\cdot T'(0)=0.$$

since $X(x)=0$ leads to Trivial $u(x,t)=0$,

$T'(0)=0$:

$$T'(t)=-P\sqrt{\mu}\sin(\sqrt{\mu}\,t)+Q\sqrt{\mu}\cos(\sqrt{\mu}\,t)$$

$$T'(0)=Q\sqrt{\mu}\cos(0)=0;\quad Q\sqrt{\mu}=0.$$

we know $\mu=\left(\frac{c\,i\pi}{L}\right)^2$ for $i=1,2\ldots$; $Q=0$.

So, $T(t)=P\cos\left(\frac{c\,i\pi}{L}t\right)$, $i=1,2\cdots$

$$u(x,t)=\sum_i X(x)T(t)\cdot B_i$$

(B_i: constant)

$$=\sum_i B\sin\left(\frac{i\pi}{L}x\right)P\cos\left(\frac{c\,i\pi}{L}t\right)\cdot B_i$$

$$u(x,0)=\sum B\sin\left(\frac{i\pi}{L}x\right)P\,B_i$$

$$\mathcal{F}(m(x))=u(x,T)=\sum_i B\sin\left(\frac{i\pi x}{L}\right)P\cos\left(\frac{c\,i\pi}{L}T\right)B_i$$

So, the eigenvalues are $\lambda_i=\cos\left(\frac{c\,i\pi}{L}T\right)$.

$(i=1,2\ldots)$

The eigenfunctions are $v_i=\frac{B_i}{w}\sin\left(\frac{i\pi}{L}x\right)$

(w: normalization constant)

$(i=1,2\ldots)$

To determine B_i, recall for to eigenfunctions v_m, v_n to be orthonormal:

$$\int_0^L v_m(x) v_n(x) = \begin{cases} 0 \; ; \; m \neq n \\ 1 \; ; \; m = n \end{cases}$$

$$\Rightarrow \int_0^L (B_i)^2 \sin^2\left(\frac{i\pi}{L} x\right) dx = 1$$

$$\frac{B_i^2}{2} \int_0^L 1 - \cos\left(\frac{2i\pi}{L} x\right) dx = 1$$

$$\frac{B_i^2}{2} \left[x - \sin\left(\frac{2i\pi}{L} x\right) \frac{L}{2i\pi}\right]_0^L = 1$$

$$\frac{B_i^2}{2} L = 1; \; B_i = \sqrt{\frac{2}{L}}$$

Thus, the eigenvalue, eigenfunction pair is:

$$(\lambda_i, v_i) = \left(\cos\left(\frac{ci\pi}{L} T\right), \sqrt{\frac{2}{L}} \sin\left(\frac{i\pi}{L} x\right)\right)$$

where c is the wave propogation speed & $i = 1, 2 \ldots$.

We derive that the eigenvalue, eigenfunction pair is: $(\lambda_i, v_i) = (\cos(\frac{cT\pi}{L} i), \sqrt{\frac{2}{L}} \sin(\frac{\pi}{L} ix))$, for $i = 1, 2, \ldots$

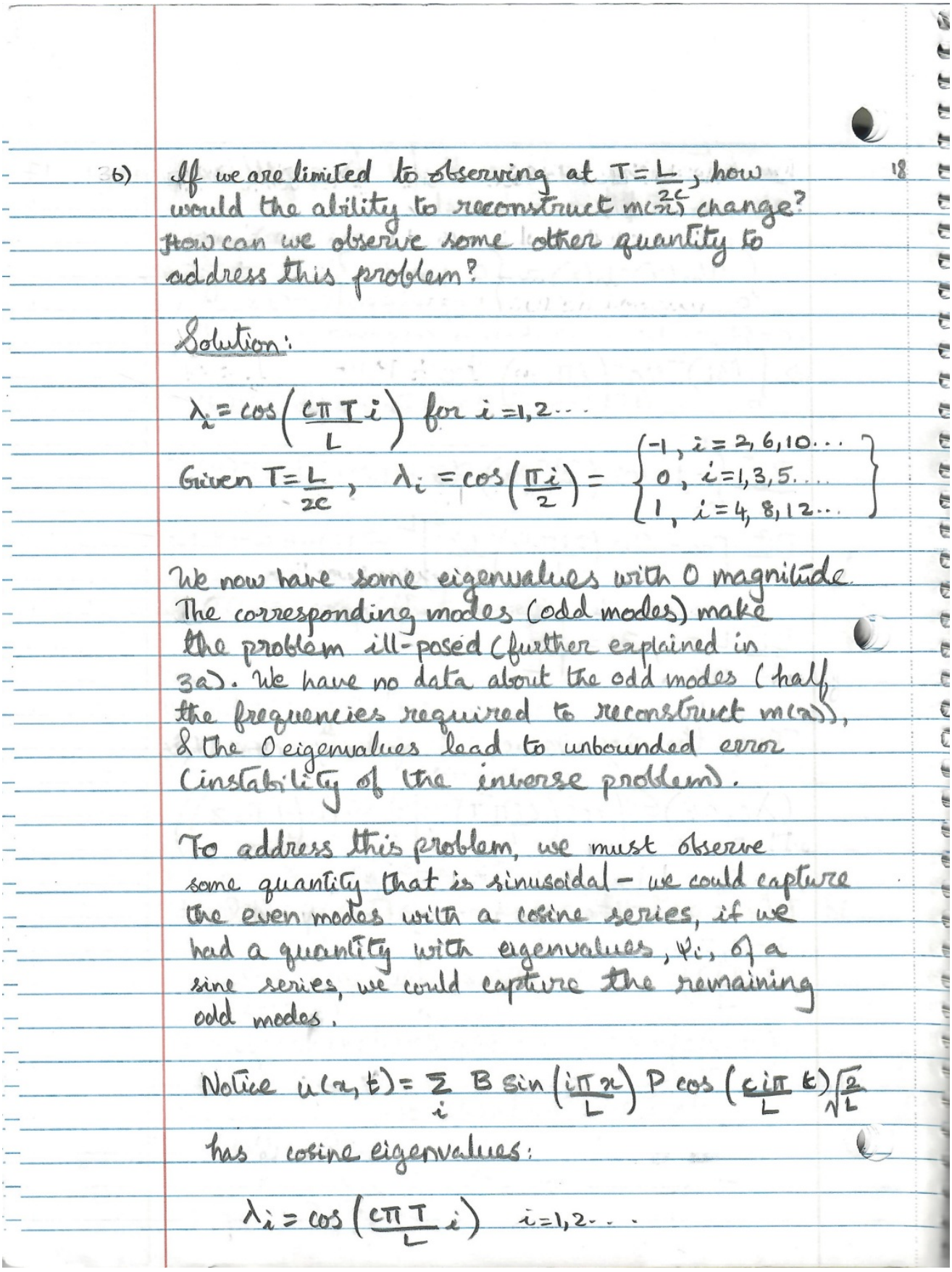
Issues with ill-posedness and stability arise when the eigenvalues λ_i approach or equal 0. As proved in lecture, the inverse problem error is defined as $m_{true} - m = \sum_{i=1}^{n} \frac{1}{\lambda_i} (v_i^T \eta) v_i$, where η is the error in the observable data and λ_i, v_i are the eigenvalue, eigenvector pair. Small eigenvalues, λ_i, contribute inversely to the error (lead to large errors) and lead to instability.

We notice that the cosine function may achieve 0 for different non-zero final times, T, or wave speeds, c. So, the problem is ill-posed. We desire $\frac{cT\pi}{L} i \neq n\frac{\pi}{2}$ for $n = 1, 3, 5 \ldots$, any positive integer mode $i = 1, 2, 3 \ldots$, and any final time T or length L.

This can be achieved by setting the observation time: $T = L/c$. Then, $\lambda_i = \cos(i\pi) = (-1)^i \neq 0 \forall i \in Z^+$; with this setting, the eigenvalues

are strictly non-zero, are all on the same order of magnitude (in fact the same magnitude exactly), and do not approach small numbers near machine precision (which leads to large magnification of any data or model noise)! We should now be able to recover the initial condition $m(x)$ perfectly with no instability whatsoever.

3b: ability to reconstruct $m(x)$ for different observation times



6) If we are limited to observing at $T=\frac{L}{2c}$, how would the ability to reconstruct $m(x)$ change? How can we observe some other quantity to address this problem? 18

Solution:

$\lambda_i = \cos\left(\frac{c\pi T i}{L}\right)$ for $i = 1, 2 \ldots$

Given $T=\frac{L}{2c}$, $\lambda_i = \cos\left(\frac{\pi i}{2}\right) = \begin{cases} -1, & i = 2, 6, 10 \ldots \\ 0, & i = 1, 3, 5 \ldots \\ 1, & i = 4, 8, 12 \ldots \end{cases}$

We now have some eigenvalues with 0 magnitude. The corresponding modes (odd modes) make the problem ill-posed (further explained in 3a). We have no data about the odd modes (half the frequencies required to reconstruct $m(x)$), & the 0 eigenvalues lead to unbounded error (instability of the inverse problem).

To address this problem, we must observe some quantity that is sinusoidal – we could capture the even modes with a cosine series, if we had a quantity with eigenvalues, ψ_i, of a sine series, we could capture the remaining odd modes.

Notice $u(x,t) = \sum_i B \sin\left(\frac{i\pi x}{L}\right) P \cos\left(\frac{c i \pi}{L} t\right) \sqrt{\frac{2}{L}}$

has cosine eigenvalues:

$\lambda_i = \cos\left(\frac{c\pi T}{L} i\right)$ $i = 1, 2 \ldots$

We know, then that $\frac{\partial u(x,t)}{\partial t}$ will have sine eigenvalues, which is exactly what we are looking for! With $\frac{\partial u(x,t)}{\partial t}$, we can capture data from odd modes.

$$\frac{\partial u(x,t)}{\partial t} = \sum_i -B \sin\left(\frac{i\pi x}{L}\right) P \sqrt{\frac{2}{L}} \cdot \frac{L}{ci\pi} \sin\left(\frac{ci\pi}{L} t\right)$$

$$\frac{\partial u}{\partial t}(x,T) = \sum_i -BP \frac{L}{ci\pi} \underbrace{\sqrt{\frac{2}{L}} \sin\left(\frac{i\pi x}{L}\right)}_{\text{eigenfunction}} \sin\left(\frac{ci\pi}{L} T\right)$$

The eigenvalue, eigenfunction pair for $\frac{\partial u}{\partial t}(x,T)$ is:

$$(\psi_i, \eta_i) = \left(\frac{1}{ci\pi} \sin\left(\frac{ci\pi}{L} T\right), \sqrt{\frac{2}{L}} \sin\left(\frac{i\pi x}{L}\right)\right)$$

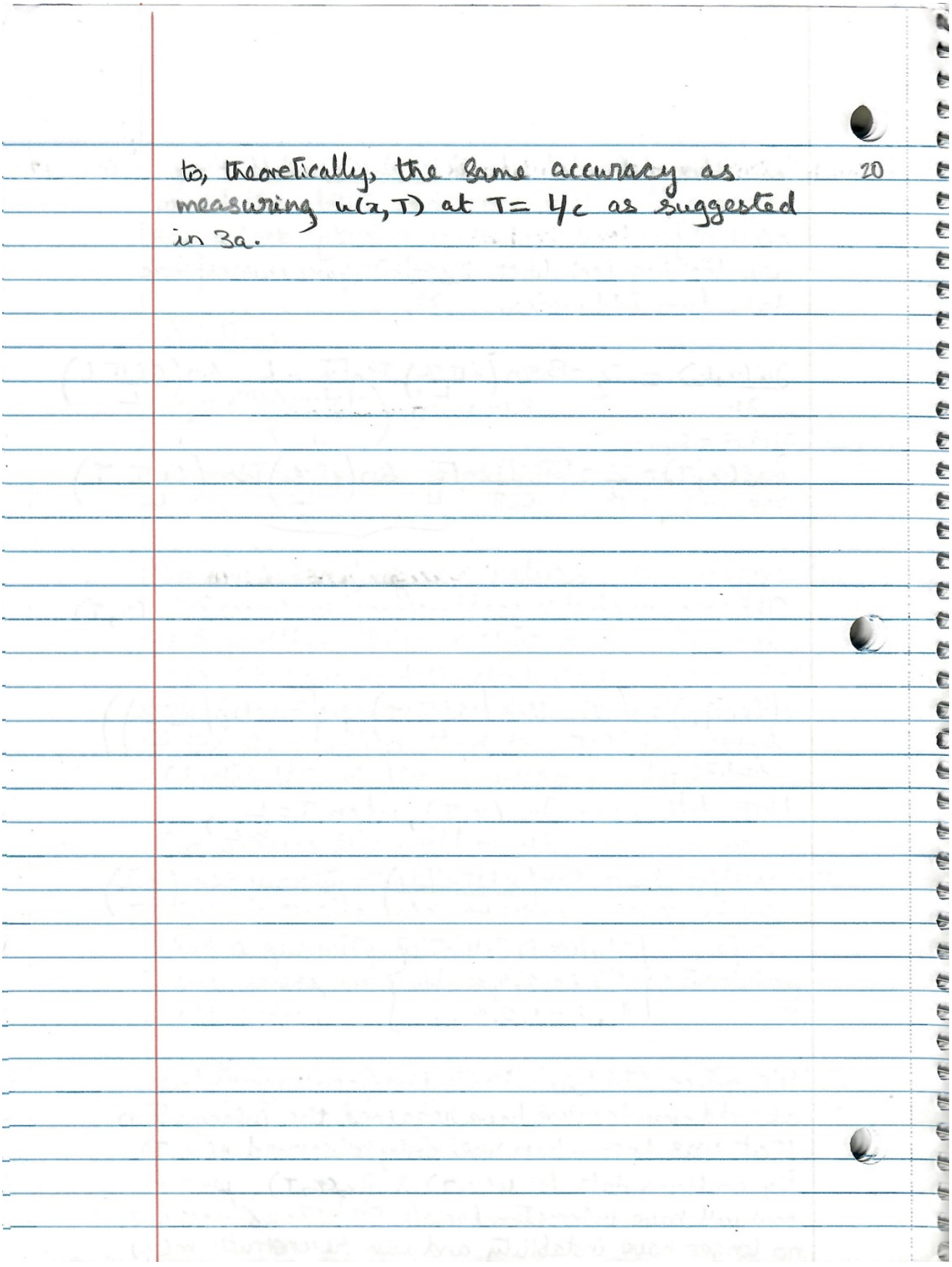
for: $i = 1, 2 \ldots$

With data from $\frac{\partial u}{\partial t}(x,T)$, when $T = \frac{L}{2c}$,

$$\psi_i = \frac{1}{ci\pi} \sin\left(\frac{ci\pi}{L} \frac{L}{2c}\right) = \frac{1}{ci\pi} \sin\left(\frac{i\pi}{2}\right)$$

$$= \begin{Bmatrix} -1, & i = 3, 7, 11 \ldots \\ 0, & i = 2, 4, 6 \ldots \\ 1, & i = 1, 5, 9 \ldots \end{Bmatrix}$$

We notice that we have non-zero eigenvalues at odd-modes; we have regained the information that was lost when we only observed $u(x,T)$. By combining data for $u(x,T)$ & $\frac{\partial u(x,T)}{\partial t}$, we now will have information for all the modes; we no longer have instability and can reconstruct $m(x)$



3c: constraints on reconstructing $m(x)$

Some sources of error that might limit our ability to reconstruct $m(x)$ include:

1. Instrumental error, where the instruments can only capture data to a certain degree of accuracy (e.g. 10^{-3} error for a gauge) can cause ill-posedness in the inverse problem. Given most gauges have random noise--white noise with high modes, small eigenvalues--this can dramatically increase error (which is directly proportional to the inverse of the eigenvalue magnitude). Additionally, $d(x, T)$ is observed at a finite number of discrete points along the vable/wire, rather than continuosly over $[0, L]$.

2. Limited spatial resolution can make the problem underdetermined and ill-posed. Insufficient sampling (and/or poorly distributed sampling) causes higher modes of m(x) to not be caputured accurately; to recognize/reproduce a high mode wave, a large number of data points is required to identify all the crests/troughes of the wave and intermediate curves (especially in comparision to a lower

mode wave).

3. Observation time choices can also affect reconstruction ability. Certain observation times (e.g. $T = \frac{L}{2c}$) cause particular eigenfunctions indistinguishable (further described in 3b) and certain modes to be entirely lost. Without additional measurements, reconstruction of $m(x)$ would be highly innacurate with $\approx$ half the modes missing from $m(x)$ (if limited to observing only $u(x, T)$ at $T = \frac{L}{2c}$).
4. Model error: our model may not account for the entire physics/laws governing the problem. For example, there are specific material or environmental properties like creep, friction, thermal inconsistencies (uneven heating dissipates energy non-uniformly) or air resistance that introduce sources of error unaccounted for in a simple wave model. These sources for error are irreversible and cause excessive damping on some or all modes of data; with data unrepresentative of the initial conditions, the models cannot account for unobserved modes, and thus, will be unable to fully reconstruct $m(x)$.