Lab10a_Regularization_AM

December 16, 2023

1 Regularized least squares - bounds and smoothness

Computational Methods for Geoscience - EPS 400/522

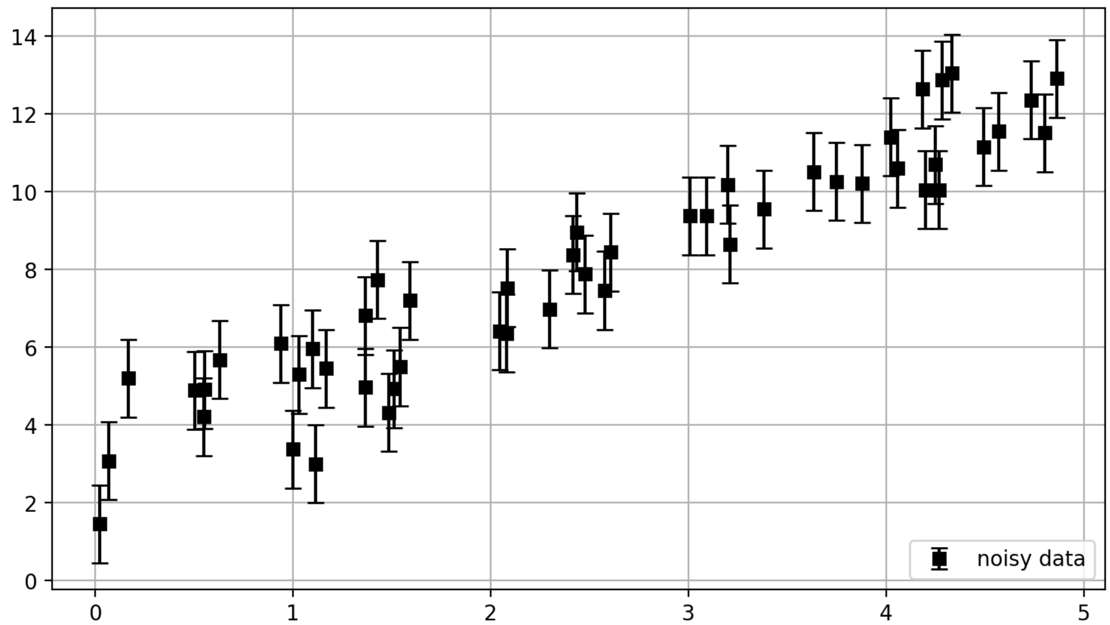
Instructor: Eric Lindsey

[ ]:
```
# some useful imports and settings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.optimize
import scipy.stats

import fault_model

%config InlineBackend.figure_format = 'retina' # better looking figures on␣
 ↪high-resolution screens
# automatically reload modules when running, otherwise jupyter does not notice␣
 ↪if our functions have changed
%load_ext autoreload
%autoreload 2
```

1.1 Example - Line fitting

[ ]:
```
# create some test data that follows a line

# True parameter values
a_true = 2
b_true = 3
sigma = 1

# Size of dataset
size = 50

# Predictor variable ("independent variable", or in geophysics, "coordinates")
x = 5*np.random.rand(size)
```

```
# Simulate outcome (dependent variable)
y = a_true*x + b_true + np.random.normal(size=size) * sigma

# simulate y errors
yerr = sigma*np.ones(size)

# plot the simulated data
plt.figure(figsize=(9,5))
plt.errorbar(x,y,yerr=sigma,fmt='ks',label='noisy data',capsize=4)
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



Since this is a 2-parameter, linear model, it's great for testing that our code is working, since we can benchmark it against the least squares solution.

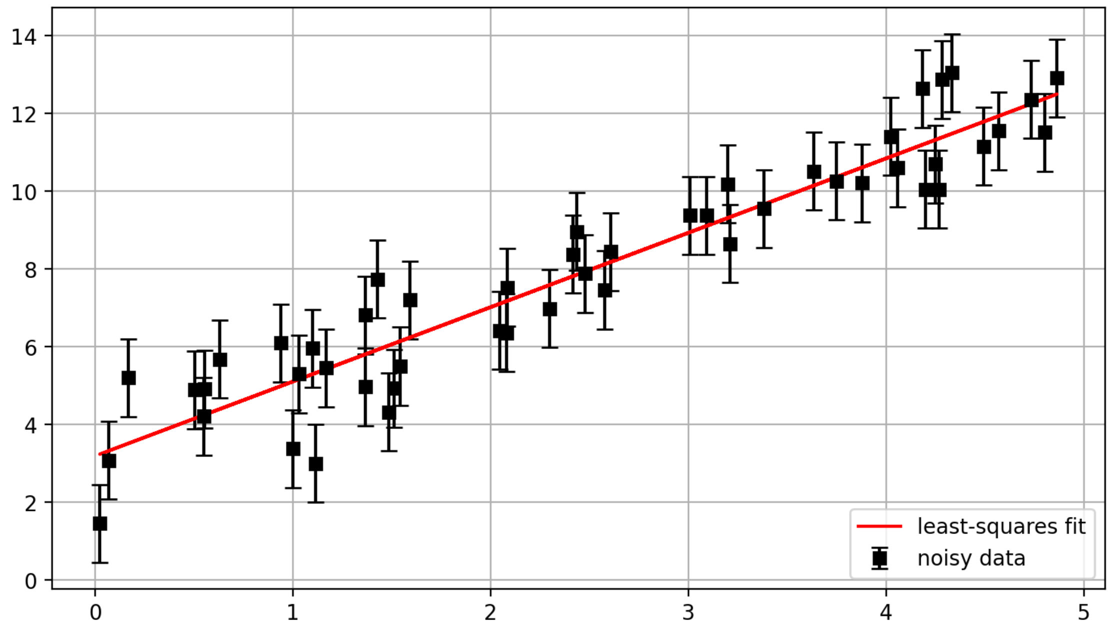
Benchmarking is always a good idea - don't try out your code on new data until you know for sure it works correctly!

1.1.1 Solution 1. Least-squares analysis

[ ]:
```
# scipy.stats package for linear regression
slope, intercept, r_value, p_value, std_err = scipy.stats.linregress(x,y)
print(slope,intercept)
# plot the best-fit line
lsq_model = slope*x + intercept
```

```
plt.figure(figsize=(9,5))
plt.errorbar(x,y,yerr=sigma,fmt='ks',label='noisy data',capsize=4)
plt.plot(x,lsq_model, '-r', label='least-squares fit')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```

```
1.9124647031877202 3.192380769225288
```



[ ]:
```
# Another method: use Scipy least squares - we need to define our "G" matrix to␣
↪use this
G = np.column_stack( (x, 0*x+1) )
m = scipy.optimize.lsq_linear(G,y)

# the model output contains lots of info:
print(m)

# the model parameters are in "m.x"
print("\nSolution:")
print(m.x)

# compare to the values from linregress - should be equal to floating point␣
↪error
print("\nDifference from scipy.stats.linregress:")
print(m.x[0]-slope, m.x[1]-intercept)
```

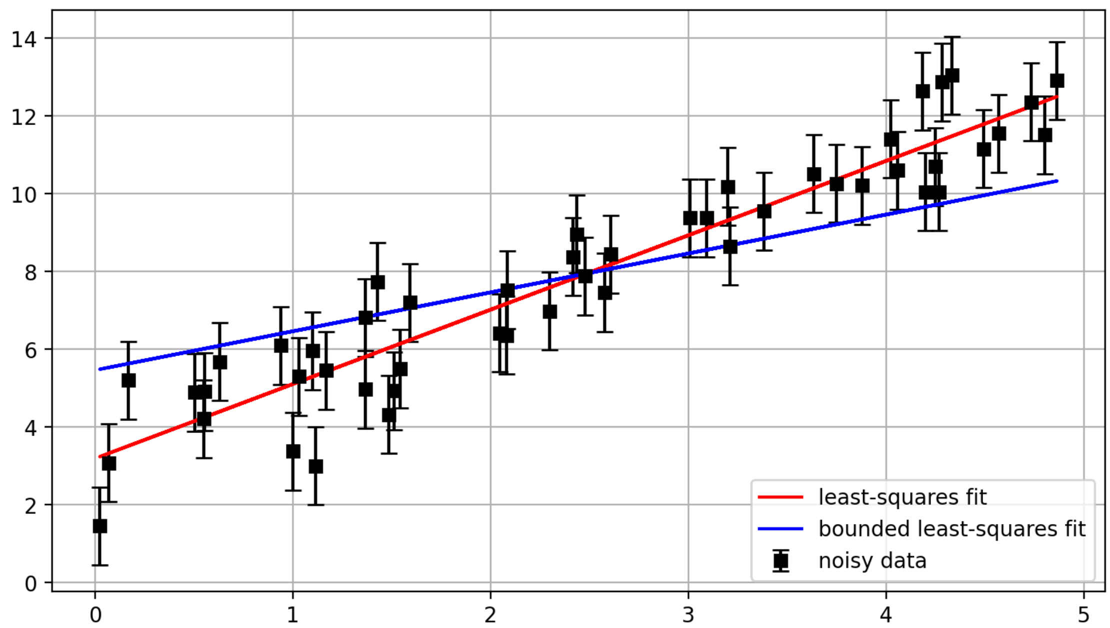
```
   active_mask: array([0., 0.])
          cost: 23.468360640330243
           fun: array([ 0.607992  , -1.57141179,  1.17038625,  0.10058767,
0.38129473,
        0.04834456, -0.36935661,  0.66336628,  0.34970763,  0.11644903,
        0.24102634, -0.42457783, -0.74242318,  0.68095597, -0.11121179,
       -1.2881909 ,  0.86509574,  0.6317594 ,  0.8325306 ,  1.77532471,
       -0.52022599,  0.68918337, -1.10290934, -0.13587397, -0.41253662,
       -0.66790128,  0.03111652,  1.7266158 , -0.87255575,  0.63872821,
        0.80524527,  1.15061085, -1.44873242,  0.39671494, -1.68587893,
       -0.5659314 ,  1.71127571, -0.96337549, -0.02343958, -1.8107373 ,
       -1.48878356, -0.27050677, -1.0125774 , -0.65865188, -1.10358437,
       -0.27145021,  0.61982062,  2.32830546,  1.30434252, -0.34395582])
       message: 'The unconstrained solution is optimal.'
           nit: 0
    optimality: 3.66064971773087e-13
        status: 3
       success: True
 unbounded_sol: (array([1.9124647 , 3.19238077]), array([46.93672128]), 2,
array([21.34023397,  3.43680834]))
             x: array([1.9124647 , 3.19238077])

Solution:
[1.9124647  3.19238077]

Difference from scipy.stats.linregress:
1.5543122344752192e-15 -1.7763568394002505e-15
```

1.2 Bounded least squares

[ ]:
```python
# place bounds that the slope is between 0 and 1, and the intercept is between␣
 ↪0 and 10.
m = scipy.optimize.lsq_linear(G,y,bounds=([0,0],[1,10]))

# the model parameters are in "m.x"
print("Solution:")
print(m.x)

# plot this new solution and the original one

bounded_lsq_model = m.x[0]*x + m.x[1]

plt.figure(figsize=(9,5))
plt.errorbar(x,y,yerr=sigma,fmt='ks',label='noisy data',capsize=4)
plt.plot(x,lsq_model, '-r', label='least-squares fit')
plt.plot(x,bounded_lsq_model, '-b', label='bounded least-squares fit')
plt.legend(loc='lower right')
```

```
plt.grid()
plt.show()

# note, printing out m reveals that it also knows the unbounded solution:
print(m)
```

```
Solution:
[1.         5.46306305]
```




```
   active_mask: array([1, 0])
          cost: 68.25424228056761
           fun: array([ 0.78001566, -3.25426184, -0.39052128, -1.0509764 ,
-1.51798742,
        0.05828982, -1.41555165,  0.58034643, -1.08279197, -0.70112445,
        2.44917342, -0.90078355,  1.06818805,  0.0212455 , -2.1620495 ,
        0.40788095, -1.24772348, -1.19695747,  1.85632262,  4.02371524,
       -1.92213262,  1.09315061, -1.05715462,  1.19310265, -2.58130769,
        0.59812442,  1.7990889 ,  3.08299967, -1.52215451,  1.50275786,
        1.17827564,  2.04173108, -2.99579345, -0.87179862,  0.42903147,
       -0.4994023 ,  2.62522001, -0.14676828,  1.17895883, -0.84518583,
       -3.12539875, -0.37937272,  0.01105195,  1.10658889,  0.30931968,
       -0.82422524, -0.98796102,  3.58072487, -0.31847494,  0.02255539])
       message: 'The first-order optimality measure is less than `tol`.'
           nit: 8
    optimality: 1.813324269407483e-14
        status: 1
       success: True
```

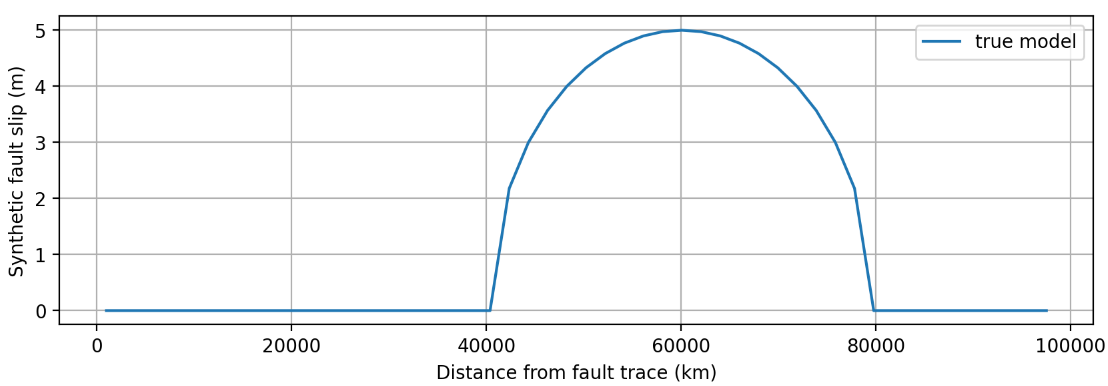
```
 unbounded_sol: (array([1.9124647 , 3.19238077]), array([46.93672128]), 2,
array([21.34023397,  3.43680834]))
             x: array([1.        , 5.46306305])
```

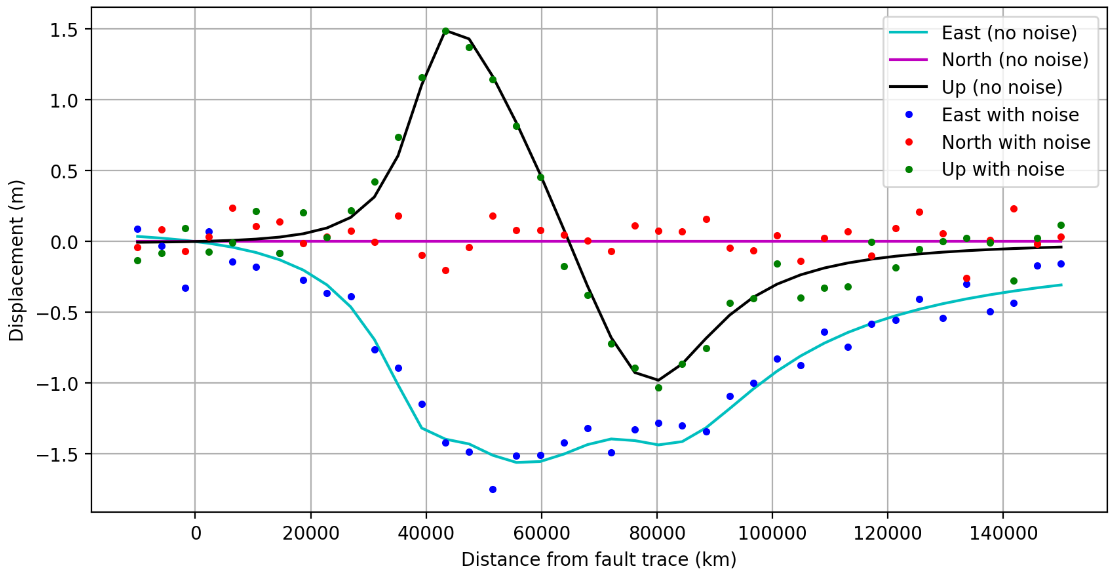
1.3 Setting up a 2D fault model

[ ]:
```python
# create an example 2D fault model and create a synthetic earthquake

x0 = 0
y0 = -5e8 # half of the along-strike width so the long fault is centered at␣
 ↪zero in the y direction
z0 = 0
strike = 0 # relative to north
dip = 10
L = 1e9 # very long, to simulate a 2D fault
W = 100e3 # 100 km width down-dip
Npatches = 50 # how many fault patches down-dip

# create an example 2D fault model
# create the model
Fmod=fault_model.FaultModel()
Fmod.
 ↪create_planar_model_cartesian(latcorner=y0,loncorner=x0,depthcorner=z0,strike=strike,dip=di

# get the patch-center coordinates
patchx = Fmod.lonc
patchy = Fmod.latc
patchz = Fmod.depth

# this is our synthetic fault slip
dip_slip = np.zeros(Npatches) # positive means thrust
for i in range(Npatches):
    # assign slip in an elliptical pattern along-dip
    slipmag = 5
    slipcenter = 30
    slipwidth = 10
    dip_slip[i] = slipmag * np.sqrt(max(0, 1-(i-slipcenter)**2/slipwidth**2))

plt.figure(figsize=(10,3))
plt.plot(patchx,dip_slip,label='true model')
plt.xlabel('Distance from fault trace (km)')
plt.ylabel('Synthetic fault slip (m)')
plt.legend()
plt.grid()
plt.show()

# define some GPS stations
```

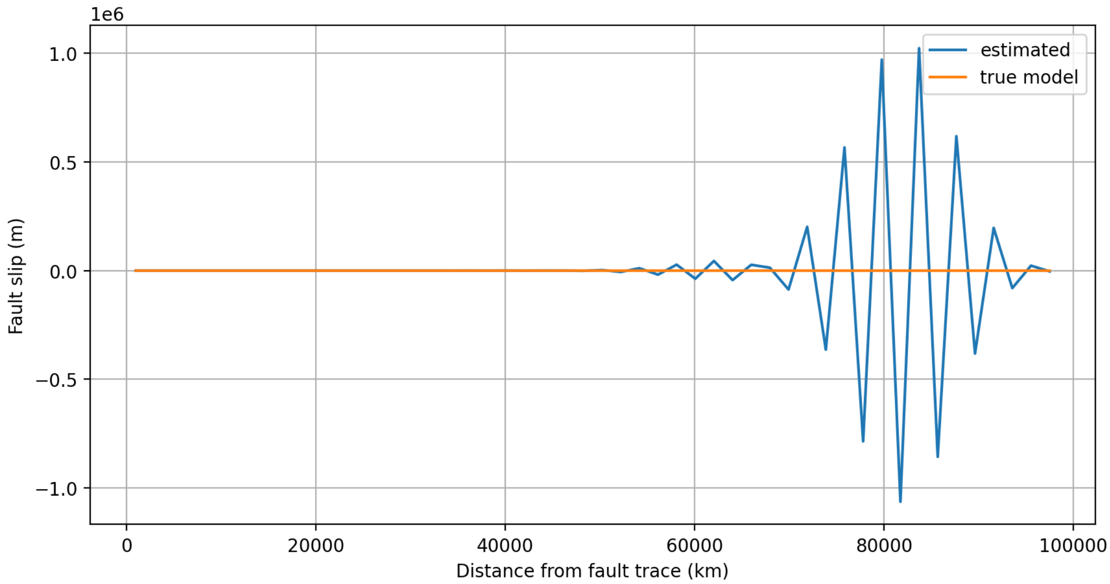
```
gpsx = np.linspace(-10e3, 150e3, 40)
gpsy = 0*gpsx

# get the Green's functions ("G" matrix) for our fault and particular GPS site␣
 ↪locations
G=Fmod.get_greens(gpsy,gpsx,coords='cartesian')

# keep only the dip components
G=G[:,1::2]


# predict GPS displacements with d=G*m
predicted_displacements = np.matmul(G,dip_slip)

# get the E, N, U components
pred_E = predicted_displacements[0::3]
pred_N = predicted_displacements[1::3]
pred_U = predicted_displacements[2::3]

noiseamp=0.1
noisydata = predicted_displacements + noiseamp * np.random.
 ↪randn(len(predicted_displacements))

# get the E, N, U components
data_E = noisydata[0::3]
data_N = noisydata[1::3]
data_U = noisydata[2::3]

plt.figure(figsize=(10,5))
plt.plot(gpsx,pred_E,'-c',label='East (no noise)')
plt.plot(gpsx,pred_N,'-m',label='North (no noise)')
plt.plot(gpsx,pred_U,'-k',label='Up (no noise)')
plt.plot(gpsx,data_E,'b.',label='East with noise')
plt.plot(gpsx,data_N,'r.',label='North with noise')
plt.plot(gpsx,data_U,'g.',label='Up with noise')

plt.xlabel('Distance from fault trace (km)')
plt.ylabel('Displacement (m)')
plt.legend(loc='upper right')
plt.grid()
plt.show()


# Fit the model in the simplest way
mfit = scipy.optimize.lsq_linear(G,noisydata)
```

```python
# plot the synthetic model slip
plt.figure(figsize=(10,5))
plt.plot(patchx,mfit.x,label='estimated')
plt.plot(patchx,dip_slip,label='true model')
plt.xlabel('Distance from fault trace (km)')
plt.ylabel('Fault slip (m)')
plt.grid()
plt.legend()
plt.show()

# plot the fit to the data
data_fit = np.matmul(G,mfit.x)

plt.figure(figsize=(10,5))
plt.plot(gpsx,data_E,'b.',label='East')
plt.plot(gpsx,data_N,'r.',label='North')
plt.plot(gpsx,data_U,'g.',label='Up')
plt.plot(gpsx,data_fit[0::3],'-c',label='East fit')
plt.plot(gpsx,data_fit[1::3],'-m',label='North fit')
plt.plot(gpsx,data_fit[2::3],'-k',label='Up fit')
plt.xlabel('Distance from fault trace (km)')
plt.ylabel('Displacement (m)')
plt.legend(loc='upper right')
plt.grid()
plt.show()
```

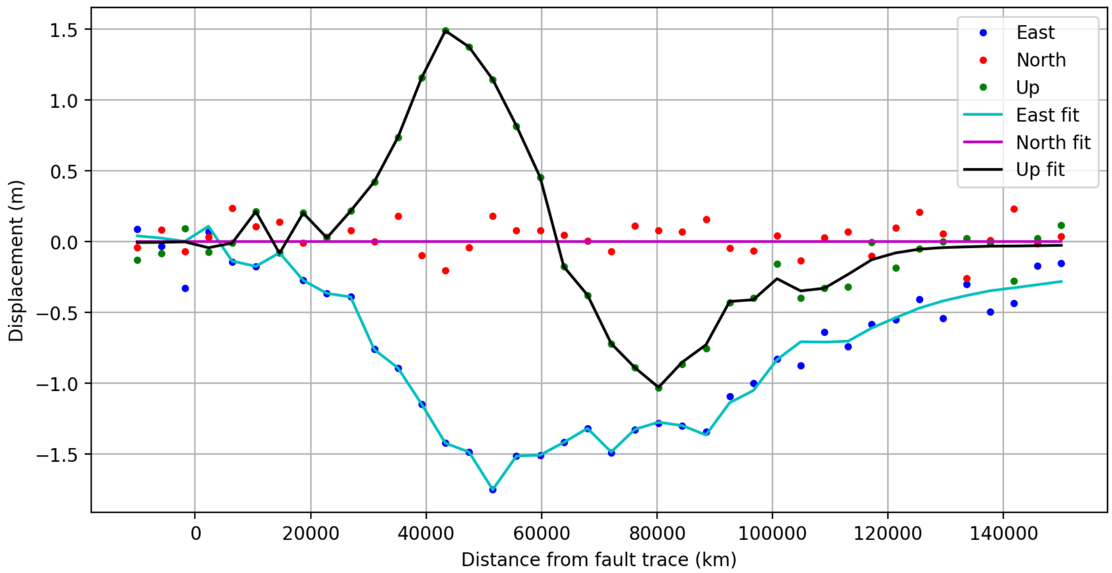




East (no noise)
North (no noise)
Up (no noise)
East with noise
North with noise
Up with noise
Displacement (m)
Distance from fault trace (km)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
20000
40000
60000
80000
100000
120000
140000

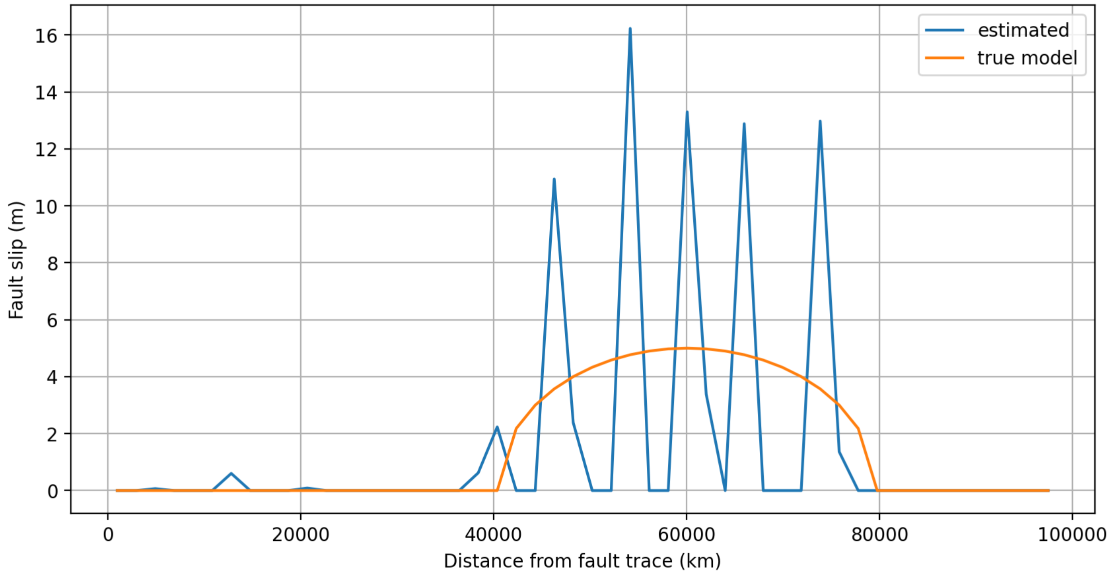


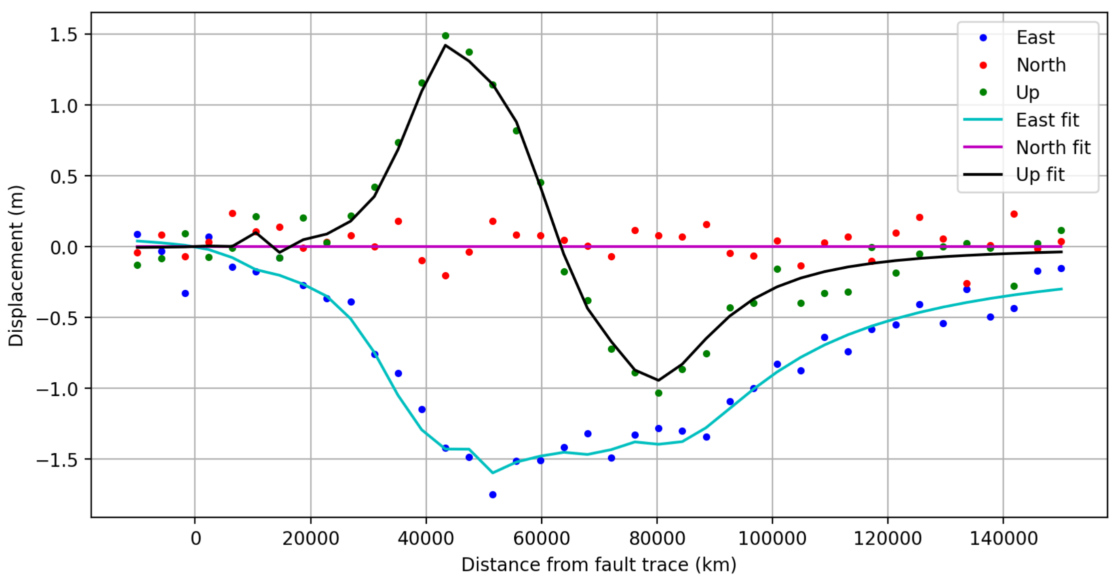
1e6
estimated
true model
Fault slip (m)
Distance from fault trace (km)
1.0
0.5
0.0
−0.5
−1.0
0
20000
40000
60000
80000
100000



[ ]:
```
# the above fits the data perfectly, but the model makes no sense... Let's try
 ↪adding bounds:

model_min = np.zeros(Npatches)
model_max = 100*np.ones(Npatches)

mfit_bounds = scipy.optimize.lsq_linear(G,noisydata,bounds =
 ↪(model_min,model_max))

# plot the synthetic model slip
plt.figure(figsize=(10,5))
plt.plot(patchx,mfit_bounds.x,label='estimated')
plt.plot(patchx,dip_slip,label='true model')
plt.xlabel('Distance from fault trace (km)')
plt.ylabel('Fault slip (m)')
plt.grid()
plt.legend()
plt.show()

# plot the fit to the data
data_fit = np.matmul(G,mfit_bounds.x)

plt.figure(figsize=(10,5))
plt.plot(gpsx,data_E,'b.',label='East')
plt.plot(gpsx,data_N,'r.',label='North')
plt.plot(gpsx,data_U,'g.',label='Up')
plt.plot(gpsx,data_fit[0::3],'-c',label='East fit')
plt.plot(gpsx,data_fit[1::3],'-m',label='North fit')
```

```python
plt.plot(gpsx,data_fit[2::3],'-k',label='Up fit')
plt.xlabel('Distance from fault trace (km)')
plt.ylabel('Displacement (m)')
plt.legend(loc='upper right')
plt.grid()
plt.show()
```





1.4 Smoothed least squares

[ ]:
```
# To add smoothing, we need to define a Laplacian matrix that, when multiplied␣
 ↪by m,
# computes the smoothness (or roughness) of a model. This is usually just the␣
 ↪2nd order
# finite difference operator (https://en.wikipedia.org/wiki/Finite_difference).

# For one dimension, where each model parameter is next to its neighbor,
# the Laplacian can be represented as:

# L = [ -1   2  -1   0  ...       0 ]
#     [ -1   2  -1   0  ...       0 ]
#     [  0  -1   2  -1  0  ...    0 ]
#     [  0   0  -1   2  -1  0 ... 0 ]
#       ...
#     [ 0  ...     0  -1   2  -1   0 ]
#     [ 0    ...       0  -1   2  -1 ]
#     [ 0    ...       0  -1   2  -1 ]

# note that the first and last rows are repeated, this is not a mistake,
# it's just how the definition works for the edge cases.

# here is a function that implements this:
import numpy as np

def create_1d_laplacian(size):
    """
    Create a Laplacian matrix of size n x n for one-dimensional data.

    Parameters:
    n (int): The size of the matrix (number of parameters in the model).

    Returns:
    numpy.ndarray: The Laplacian matrix.
    """
    # Initialize an n x n matrix with zeros
    L = np.zeros((size, size))

    # Fill the diagonal with 2s
    np.fill_diagonal(L, 2)

    # Fill the off-diagonals with -1s
    np.fill_diagonal(L[1:], -1)
    np.fill_diagonal(L[:, 1:], -1)

    # Adjust the first and last rows
```

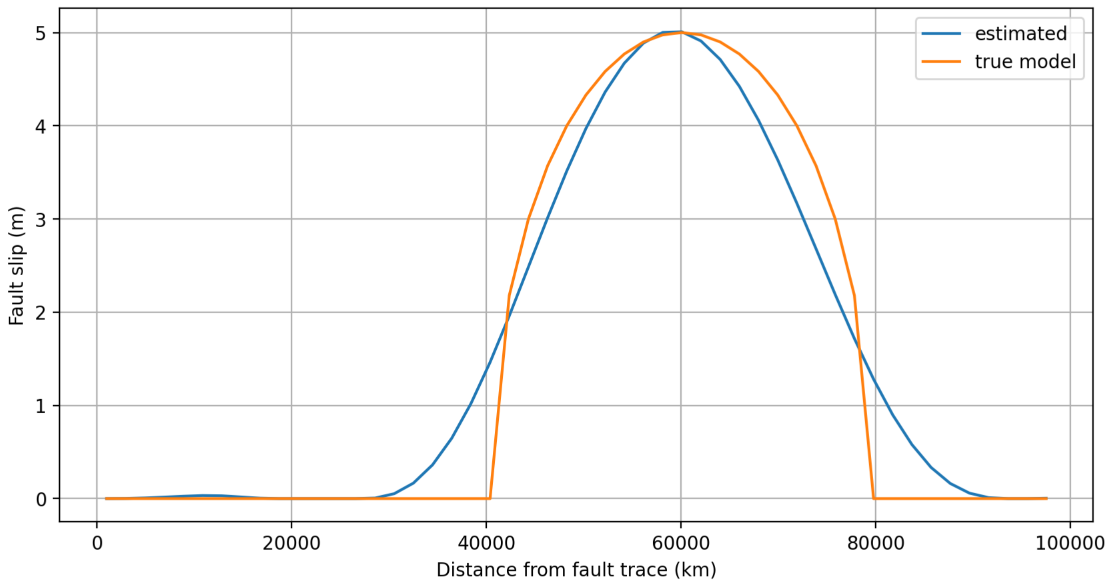
```
    L[0, 0:3] = [-1, 2, -1]
    L[-1, -4:-1] = [-1, 2, -1]

    return L

# Example of creating a Laplacian matrix for n = 8
n = 8
L = create_1d_laplacian(n)
print(L)
```

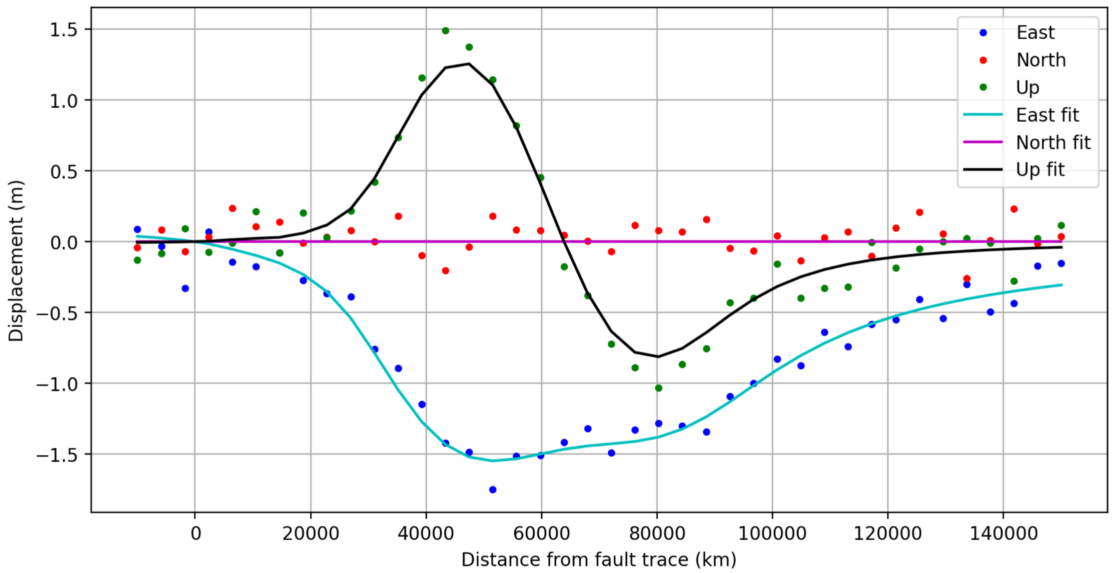
```
[[-1.  2. -1.  0.  0.  0.  0.  0.]
 [-1.  2. -1.  0.  0.  0.  0.  0.]
 [ 0. -1.  2. -1.  0.  0.  0.  0.]
 [ 0.  0. -1.  2. -1.  0.  0.  0.]
 [ 0.  0.  0. -1.  2. -1.  0.  0.]
 [ 0.  0.  0.  0. -1.  2. -1.  0.]
 [ 0.  0.  0.  0.  0. -1.  2. -1.]
 [ 0.  0.  0.  0. -1.  2. -1.  2.]]
```

[ ]:
```
# The bounded model is much closer than before, but we could still do better if␣
↪the model was smoother. Let's add smoothing:
L = create_1d_laplacian(Npatches)

# smoothing factor
lam = 1.8

print('G shape:', np.shape(G))
print('L shape:', np.shape(L))
G_augmented = np.row_stack( (G, lam*L) )
print('G_aug shape:', np.shape(G_augmented))

# we also need to augment the data vector with zeros
print('Data shape:', np.shape(noisydata))
d_augmented = np.concatenate( (noisydata,np.zeros(Npatches)), axis=None)
print('Data_aug shape:', np.shape(d_augmented))

mfit_smooth = scipy.optimize.lsq_linear(G_augmented,d_augmented,bounds =␣
↪(model_min,model_max))

# plot the synthetic model slip
plt.figure(figsize=(10,5))
plt.plot(patchx,mfit_smooth.x,label='estimated')
plt.plot(patchx,dip_slip,label='true model')
plt.xlabel('Distance from fault trace (km)')
plt.ylabel('Fault slip (m)')
plt.grid()
plt.legend()
```

```python
plt.show()

# plot the fit to the data
data_fit = np.matmul(G,mfit_smooth.x)

plt.figure(figsize=(10,5))
plt.plot(gpsx,data_E,'b.',label='East')
plt.plot(gpsx,data_N,'r.',label='North')
plt.plot(gpsx,data_U,'g.',label='Up')
plt.plot(gpsx,data_fit[0::3],'-c',label='East fit')
plt.plot(gpsx,data_fit[1::3],'-m',label='North fit')
plt.plot(gpsx,data_fit[2::3],'-k',label='Up fit')
plt.xlabel('Distance from fault trace (km)')
plt.ylabel('Displacement (m)')
plt.legend(loc='upper right')
plt.grid()
plt.show()
```

```
G shape: (120, 50)
L shape: (50, 50)
G_aug shape: (170, 50)
Data shape: (120,)
Data_aug shape: (170,)
```





1.5 Your turn: finding the optimal model

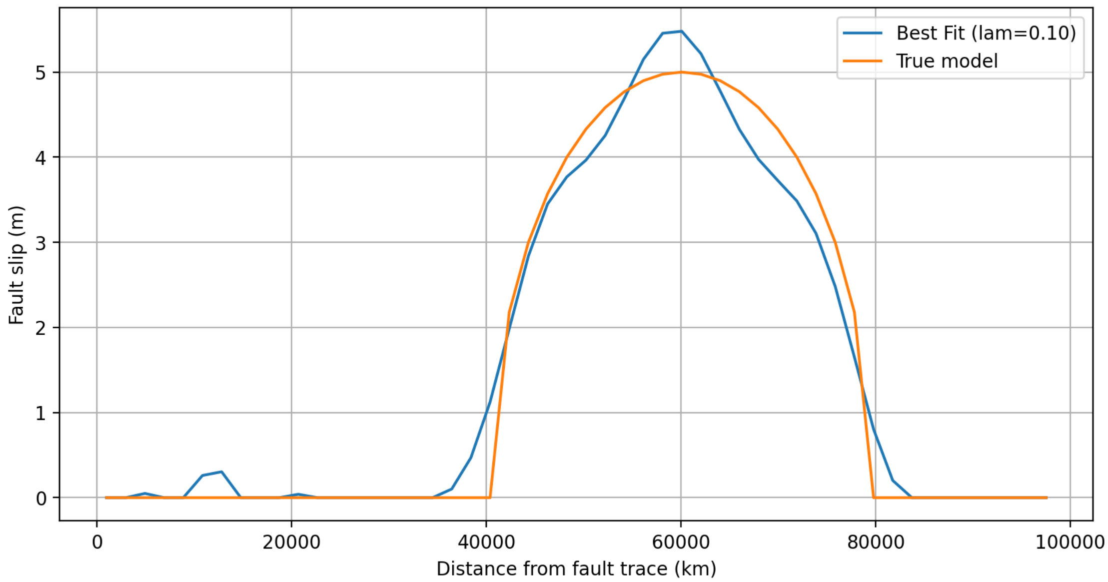
Try computing the Chi-squared misfit between the noisy data and the model predictions. You can use 'noiseamp' as the uncertainty.

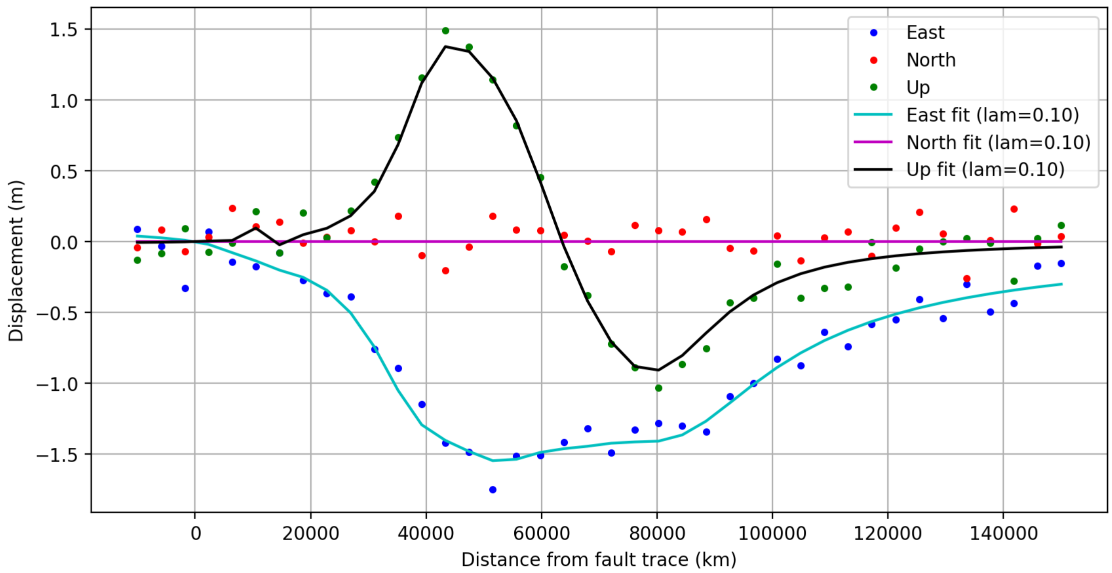
Then, run the model several times with different smoothing, and plot the misfit as a function of the smoothing value, and choose the 'best' one.

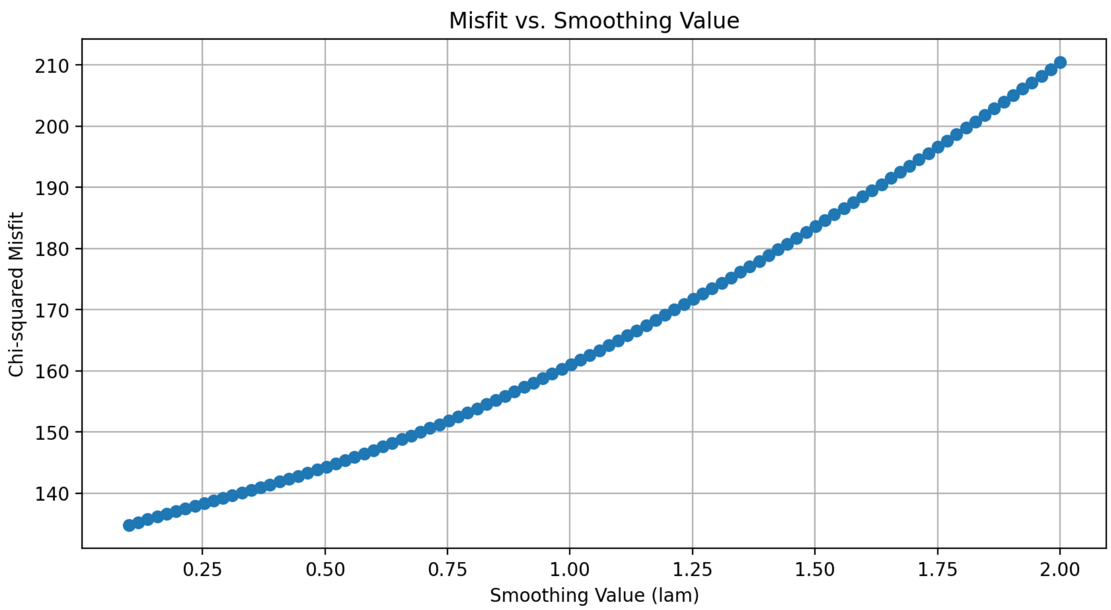
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize

# Define a function to compute Chi-squared misfit
def compute_misfit(G, model, data, noiseamp):
    residual = G.dot(model) - data
    chi_squared = np.sum((residual / noiseamp) ** 2)
    return chi_squared

# Set up a range of smoothing values
num_smooth_values = 100
smooth_values = np.linspace(0.1, 2.0, num_smooth_values)

# Arrays to store misfit values and models
misfit_values = []
models = []

# Arrays to store fitted displacements for plotting
fitted_displacements = []
```

```
# Iterate over different smoothing values
for lam in smooth_values:
    # Augment the Design Matrix G
    G_augmented = np.row_stack((G, lam * L))

    # Augment the Data Vector
    d_augmented = np.concatenate((noisydata, np.zeros(Npatches)), axis=None)

    # Perform Smoothed Linear Least Squares Optimization
    mfit_smooth = scipy.optimize.lsq_linear(G_augmented, d_augmented,␣
 ↪bounds=(model_min, model_max))

    # Compute the Chi-squared misfit
    misfit = compute_misfit(G_augmented, mfit_smooth.x, d_augmented, noiseamp)

    # Store misfit and model
    misfit_values.append(misfit)
    models.append(mfit_smooth.x)

    # Compute the fitted displacements for plotting
    data_fit = np.matmul(G, mfit_smooth.x)
    fitted_displacements.append(data_fit)

# Find the index of the minimum misfit value
best_index = np.argmin(misfit_values)
best_lam = smooth_values[best_index]
best_model = models[best_index]
best_fitted_displacements = fitted_displacements[best_index]

# Plot the best-fit model
plt.figure(figsize=(10, 5))
plt.plot(patchx, best_model, label='Best Fit (lam={:.2f})'.format(best_lam))
plt.plot(patchx, dip_slip, label='True model')
plt.xlabel('Distance from fault trace (km)')
plt.ylabel('Fault slip (m)')
plt.grid()
plt.legend()
plt.show()

# Plot the fit to the data
plt.figure(figsize=(10, 5))
plt.plot(gpsx, data_E, 'b.', label='East')
plt.plot(gpsx, data_N, 'r.', label='North')
plt.plot(gpsx, data_U, 'g.', label='Up')
plt.plot(gpsx, best_fitted_displacements[0::3], '-c', label='East fit (lam={:.
 ↪2f})'.format(best_lam))
```

```python
plt.plot(gpsx, best_fitted_displacements[1::3], '-m', label='North fit (lam={:.
↪2f})'.format(best_lam))
plt.plot(gpsx, best_fitted_displacements[2::3], '-k', label='Up fit (lam={:.
↪2f})'.format(best_lam))
plt.xlabel('Distance from fault trace (km)')
plt.ylabel('Displacement (m)')
plt.legend(loc='upper right')
plt.grid()
plt.show()

# Plot misfit as a function of smoothing value
plt.figure(figsize=(10, 5))
plt.plot(smooth_values, misfit_values, marker='o')
plt.xlabel('Smoothing Value (lam)')
plt.ylabel('Chi-squared Misfit')
plt.title('Misfit vs. Smoothing Value')
plt.grid()
plt.show()
```







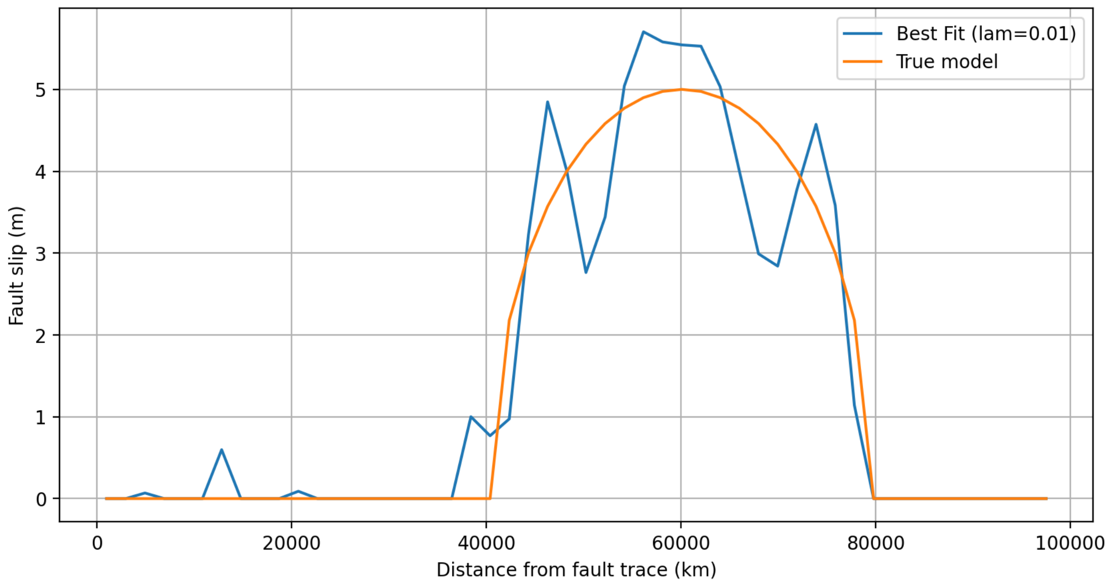
1.5.1 Optimal Lam value: 0.10

The above code finds that the optimal smoothing value is 0.10. This results in a low Chi-Squared misfit, or an optimal smoothing value without overfitting.

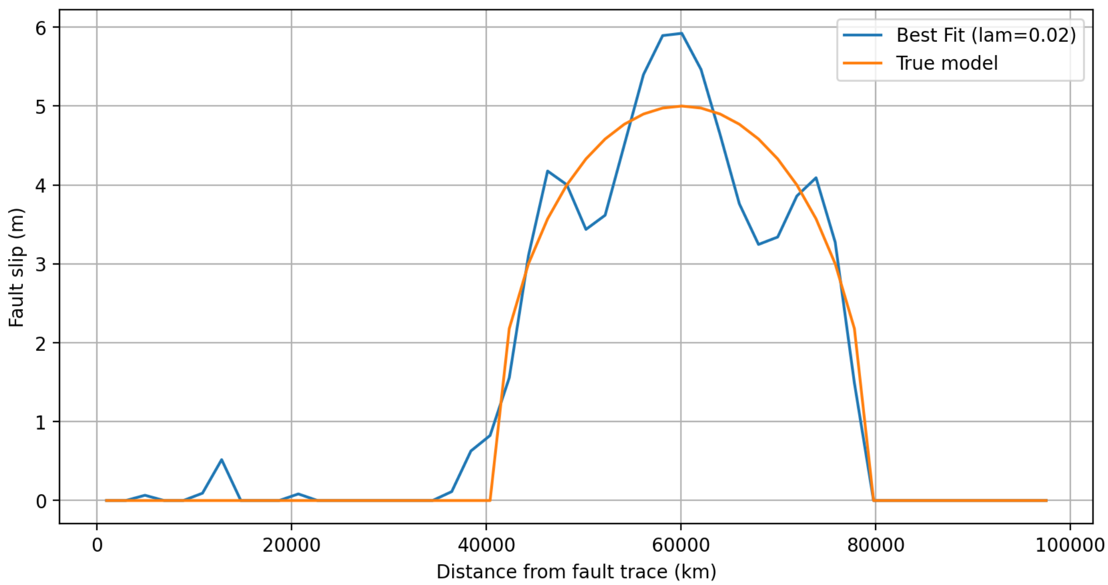
The following code explores more smoothing (lam) values…

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize

# Function to compute Chi-squared misfit
def compute_misfit(G, model, data, noiseamp):
    residual = G.dot(model) - data
    chi_squared = np.sum((residual / noiseamp) ** 2)
    return chi_squared

# Set up a range of smoothing values
num_smooth_values = 10
smooth_values = np.linspace(0.1, 2.0, num_smooth_values)

# Arrays to store misfit values and models
misfit_values = []
models = []

# Arrays to store fitted displacements for plotting
fitted_displacements = []

# Iterate over different smoothing values
for lam in smooth_values:
    # Augment the Design Matrix G
    G_augmented = np.row_stack((G, lam * L))

    # Augment the Data Vector
    d_augmented = np.concatenate((noisydata, np.zeros(Npatches)), axis=None)

    # Perform Smoothed Linear Least Squares Optimization
    mfit_smooth = scipy.optimize.lsq_linear(G_augmented, d_augmented,␣
↪bounds=(model_min, model_max))

    # Compute the Chi-squared misfit
    misfit = compute_misfit(G_augmented, mfit_smooth.x, d_augmented, noiseamp)

    # Store misfit and model
    misfit_values.append(misfit)
    models.append(mfit_smooth.x)

    # Compute the fitted displacements for plotting
    data_fit = np.matmul(G, mfit_smooth.x)
    fitted_displacements.append(data_fit)

# Find the index of the minimum misfit value
best_index = np.argmin(misfit_values)
best_lam = smooth_values[best_index]
```

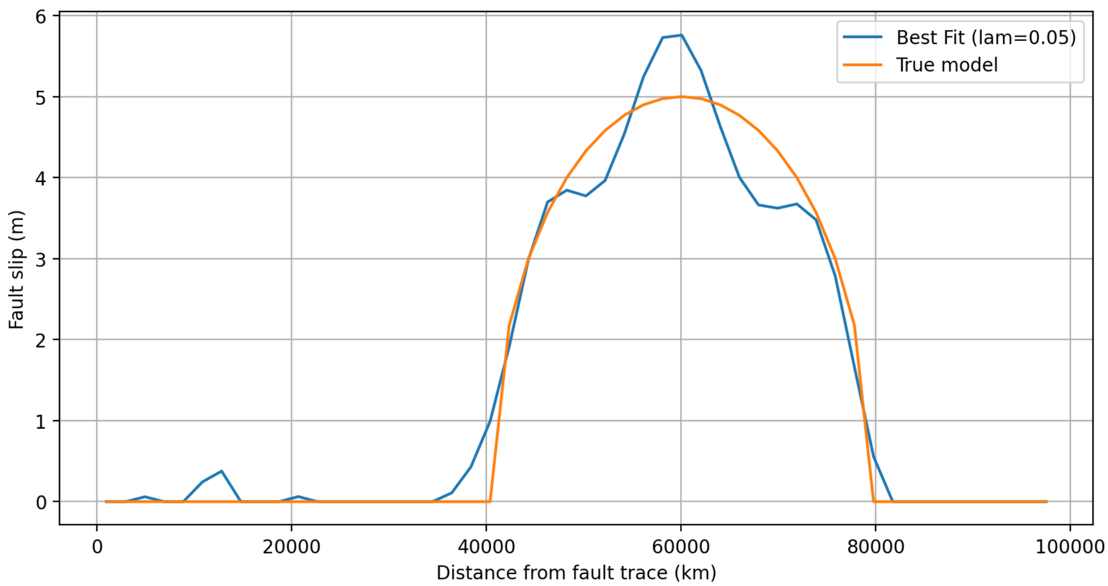
```python
best_model = models[best_index]
best_fitted_displacements = fitted_displacements[best_index]

# Input loop for manual testing of smoothing factors
while True:
    try:
        # Get the smoothing factor from the user
        lam = float(input("Enter smoothing factor (0 to exit): "))

        # Check if the user wants to exit
        if lam == 0:
            break

        # Augment the Design Matrix G
        G_augmented = np.row_stack((G, lam * L))

        # Augment the Data Vector
        d_augmented = np.concatenate((noisydata, np.zeros(Npatches)), axis=None)

        # Perform Smoothed Linear Least Squares Optimization
        mfit_smooth = scipy.optimize.lsq_linear(G_augmented, d_augmented,␣
↪bounds=(model_min, model_max))

        # Compute the Chi-squared misfit
        misfit = compute_misfit(G_augmented, mfit_smooth.x, d_augmented,␣
↪noiseamp)

        # Plot the best-fit model
        plt.figure(figsize=(10, 5))
        plt.plot(patchx, mfit_smooth.x, label='Best Fit (lam={:.2f})'.
↪format(lam))
        plt.plot(patchx, dip_slip, label='True model')
        plt.xlabel('Distance from fault trace (km)')
        plt.ylabel('Fault slip (m)')
        plt.grid()
        plt.legend()
        plt.show()

        print(f"Chi-squared misfit for lam={lam:.2f}: {misfit:.4f}")

    except ValueError:
        print("Invalid input. Please enter a valid smoothing factor or 0 to␣
↪exit.")
```



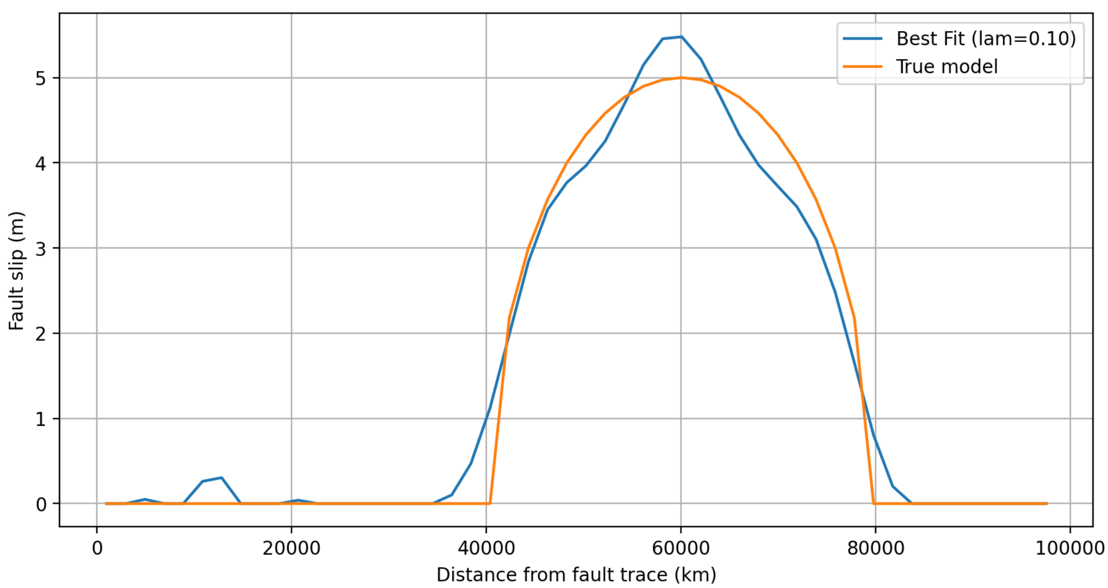

```
Chi-squared misfit for lam=0.01: 131.5933
```



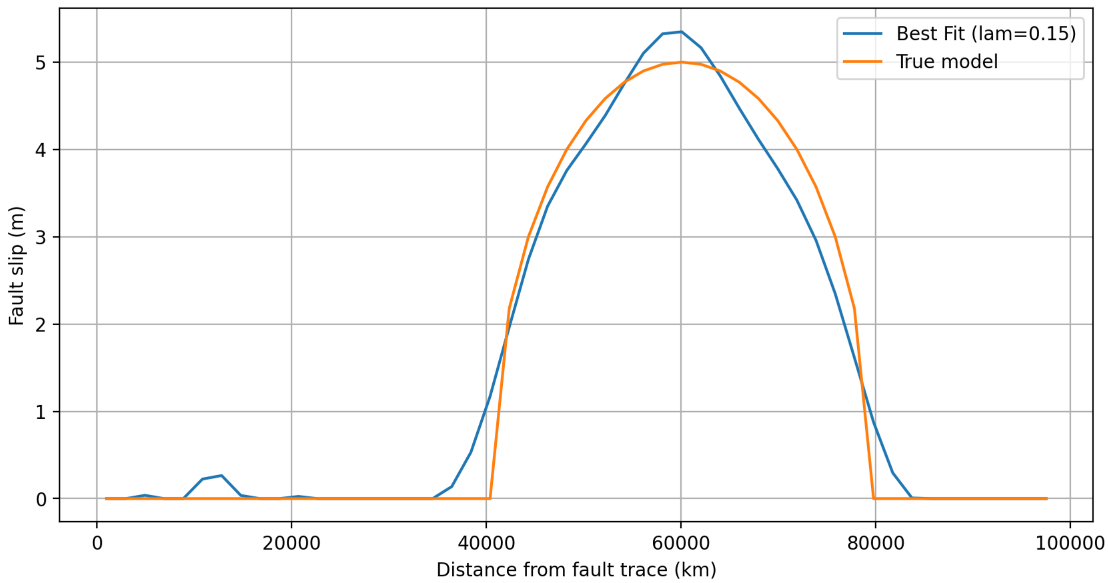

```
Chi-squared misfit for lam=0.02: 132.1108
```



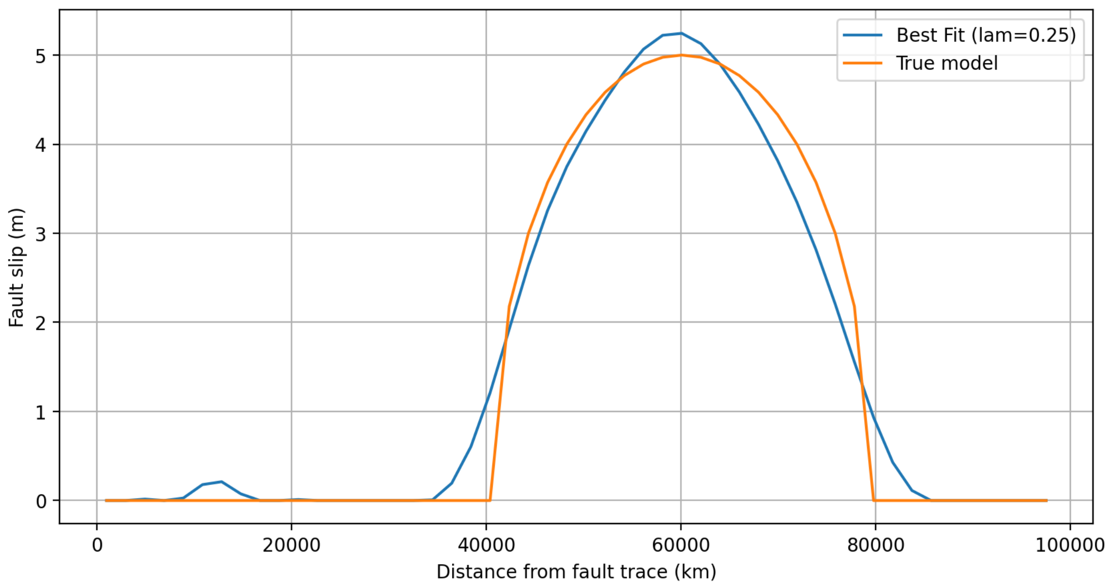
Chi-squared misfit for lam=0.05: 133.2256



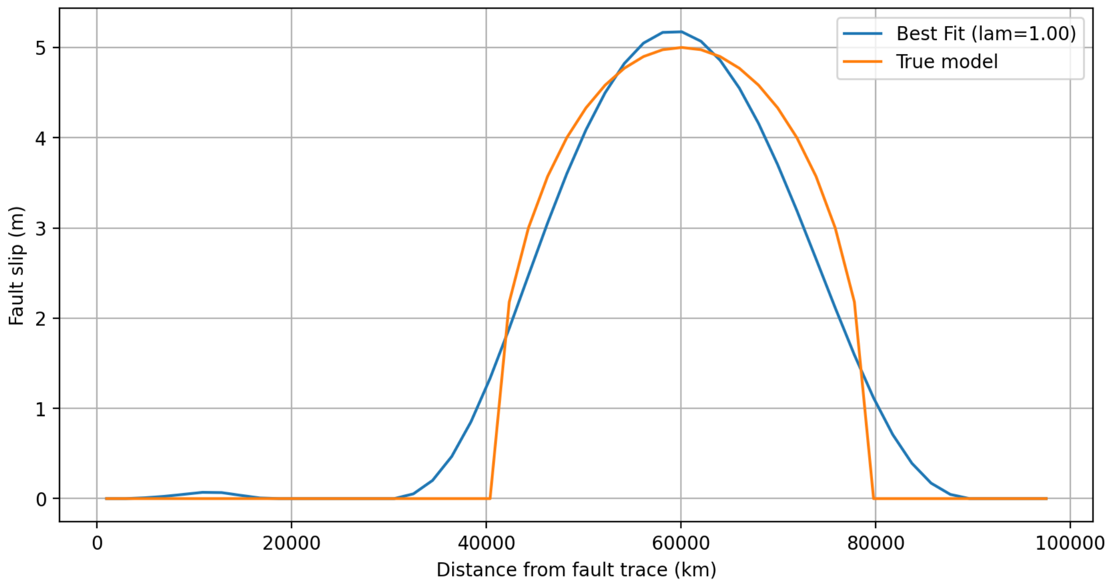
Chi-squared misfit for lam=0.10: 134.7600



Chi-squared misfit for lam=0.15: 136.0615



Chi-squared misfit for lam=0.25: 138.3035




```
Chi-squared misfit for lam=1.00: 160.9734
```