FINAL PROJECT REPORT

Comparison of Solution Methods for a Strongly Nonlinear Inverse Problem

Victor Agaba, December 5, 2023

CONTENTS

1.0 Summary

I implemented a directed search algorithm based on analytical derivatives for focal mechanism inversion. It is compared with previously used systematic searches on the basis of solution quality and computational efficiency.

2.0 Introduction

My current research is on focal mechanism inversion using data collected by a single seismometer (Sita and van der Lee 2022). The goal is to quantify the quality of a mechanism predicted by an underlying mathematical model using statistical and computational methods. Because it is a strongly nonlinear inverse problem, we have been using systematic searches (grid search and random search) to perform simulations for model assessment.

The goal of this project is to implement a directed search algorithm for the problem in the hope of finding a reliable solution with much better computational efficiency, and then assess the corresponding tradeoff in solution quality. With this method implemented, we can also get a clearer picture of the model space and make meaningful progress towards a comprehensive approach for the inversion.

3.0 Forward Problem

According to (Sita and van der Lee 2022), a focal mechanism can be defined by its strike $\psi \in (0°, 360°)$, dip $\delta \in (0°, 90°)$ and rake $\lambda \in (-180°, 180°)$.

When a quake event is recorded by a seismometer, we can also get information about the ray path's azimuth $\phi \in (0°, 360°)$ and the P- and S-wave take-off angles i and j respectively associated with the event in relation to the seismic station.

If the focal mechanism (ψ, δ, λ) and P- and S-wave velocities at source depth (α, β) are known, then the relative amplitudes A^P, A^{SV} and A^{SH} from the seismometer can be modeled as a function of the azimuth and take-off angles as follows:

$$A^P(\phi, i) \sim \frac{1}{\alpha^3}(s_R(3\ cos^2(i) - 1) - q_R\, sin(2i) - p_R\, sin^2(i))$$

$$A^{SV}(\phi, j) \sim \frac{1}{\beta^3}(1.5s_R\, sin(2j) + q_R\, cos(2j) + 0.5p_R\, sin(2j))$$

$$A^{SH}(\phi, j) \sim \frac{1}{\beta^3}(q_L\, cos(j) + p_L\, sin(j))$$

where

$$s_R = 0.5\, sin(\lambda)\, sin(2\delta)$$

$$q_R = sin(\lambda)\, cos(2\delta)\, sin(\psi - \phi) + cos(\lambda)\, cos(\delta)\, cos(\psi - \phi)$$

$$p_R = cos(\lambda)\, sin(\delta)\, sin\big(2(\psi - \phi)\big) - 0.5\, sin(\lambda)\, sin(2\delta)\, cos\big(2(\psi - \phi)\big)$$

$$q_L = -\, cos(\lambda)\, cos(\delta)\, sin(\psi - \phi) + sin(\lambda)\, cos(2\delta)\, cos(\psi - \phi)$$

$$p_L = 0.5\, sin(\lambda)\, sin(2\delta)\, sin\big(2(\psi - \phi)\big) + cos(\lambda)\, sin(\delta)\, cos\big(2(\psi - \phi)\big)$$

Re-labeling variables, we can put it into the general form for an inverse problem:

$$\boldsymbol{m} = (\psi, \delta, \lambda)^\top$$

$$\boldsymbol{d} = (A^P, A^{SV}, A^{SH})^\top$$

$$\boldsymbol{h} = (s_R, q_R, p_R, q_L, p_L)^\top$$

$$\boldsymbol{C} = \begin{bmatrix} \frac{1}{\alpha^3}(3\ cos^2(i) - 1) & \frac{-1}{\alpha^3} sin(2i) & \frac{-1}{\alpha^3} sin^2(i) & 0 & 0 \\ \frac{1.5}{\beta^3} sin(2j) & \frac{1}{\beta^3} cos(2j) & \frac{0.5}{\beta^3} sin(2j) & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\beta^3} cos(j) & \frac{1}{\beta^3} sin(j) \end{bmatrix} \qquad \text{Eq1}$$

Let $\boldsymbol{d}$ represent the observed amplitudes and $\boldsymbol{g}(\boldsymbol{m})$ represent the predicted ones from inversion. The inverse problem is then expressed as

$$\boldsymbol{d}\ \sim \boldsymbol{g}(\boldsymbol{m}) = \boldsymbol{C}\ \cdot \boldsymbol{h}(\boldsymbol{m})$$

where $\boldsymbol{h}$ is a multivalued nonlinear function of the model parameters, and $\boldsymbol{C}$ is a linear transformation of $\boldsymbol{h}$. The dot is notation for multiplication, not a dot product.

4.0 Classification

To characterize the inverse problem, notice that for any 3-dimensional input in focal-mechanism space we have a 3-dimensional output in amplitude space. Under normal circumstances, this would be classified as an exact-determined problem. However, it is actually mixed-determined problem because:

1) We are interested in the relative and not absolute amplitudes, so we get more information than we need from one seismogram. This would make it overdetermined.
2) We have one seismometer, which is not enough to pinpoint the location of the event's hypocenter from which values of ϕ, i and j are derived.

5.0 Inversion

We invert by searching through model space to find parameters that most closely predict the observed relative amplitudes. Our misfit function is defined as the angle between predicted and observed amplitude vectors in 3D space:

$$\zeta = \arccos\left(\frac{\boldsymbol{d}^{\top}\boldsymbol{g}(\mathrm{m})}{\|\boldsymbol{d}\|\|\boldsymbol{g}(m)\|}\right)$$

This is minimized when the cosine similarity S_C between $\boldsymbol{d}$ and $g(\boldsymbol{m})$ is maximized. For notational convenience, let $g = g(m)$. Then we have the following relationship:

$$S_C(d, g) = \frac{\boldsymbol{d}^{\top}\boldsymbol{g}}{\|\boldsymbol{d}\|\|\boldsymbol{g}\|} = \frac{\widehat{\boldsymbol{d}}^{\top}\boldsymbol{g}}{\|\boldsymbol{g}\|}$$

where $\hat{d}$ is the direction of d. Note that in vector form we have the following relationships:

$$\nabla_{\boldsymbol{g}}\left(\widehat{\boldsymbol{d}}^{\top}\boldsymbol{g}\right) = \widehat{\boldsymbol{d}}$$

$$\nabla_{\boldsymbol{g}}(\|\boldsymbol{g}\|) = \nabla_{\boldsymbol{g}}\left(\sqrt{\boldsymbol{g}^{\top}\boldsymbol{I}\boldsymbol{g}}\right) = \frac{2\boldsymbol{I}\boldsymbol{g}}{2\sqrt{\boldsymbol{g}^{\top}\boldsymbol{I}\boldsymbol{g}}} = \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|} = \widehat{\boldsymbol{g}}$$

$$\frac{\partial}{\partial g_j} S_C(\boldsymbol{d}, \boldsymbol{g}) = \frac{\|\boldsymbol{g}\|\hat{d}_j - \widehat{\boldsymbol{d}}^{\top}\boldsymbol{g} \cdot \hat{g}_j}{\|\boldsymbol{g}\|^2}$$

$$\Rightarrow \nabla_{\boldsymbol{g}}\left(S_C(\boldsymbol{d}, \boldsymbol{g})\right) = \frac{1}{\|\boldsymbol{g}\|}\left(\widehat{\boldsymbol{d}} - S_C(\boldsymbol{d}, \boldsymbol{g}) \cdot \widehat{\boldsymbol{g}}\right)$$

At a local stationary point, we have $\widehat{\boldsymbol{d}} = S_C(\boldsymbol{d}, \boldsymbol{g}) \cdot \widehat{\boldsymbol{g}}$ which is only possible when the observed relative amplitudes are parallel in the abstract space to their predictions, but not necessarily equal in magnitude. Local minima correspond to $\widehat{\boldsymbol{d}} = -\widehat{\boldsymbol{g}}$ such that $S_C(\boldsymbol{d}, \boldsymbol{g}) = -1$, and local maxima correspond to $\widehat{\boldsymbol{d}} = \widehat{\boldsymbol{g}}$ such that $S_C(\boldsymbol{d}, \boldsymbol{g}) = 1$.

For gradient descent to apply, let the modified misfit function be the negative cosine similarity between predictions and observations.

$$E(\boldsymbol{m}) = -\,S_C\big(\boldsymbol{d}, \boldsymbol{g}(\boldsymbol{m})\big)$$

$$\nabla_{\boldsymbol{g}} E = \frac{1}{\|\boldsymbol{g}\|}\big(S_C(\boldsymbol{d}, \boldsymbol{g}) \cdot \widehat{\boldsymbol{g}} - \widehat{\boldsymbol{d}}\big)$$

The gradient with respect to model parameters can be computed from the corresponding Jacobians. Define the Jacobian $\boldsymbol{J}$ of $\boldsymbol{h}(\boldsymbol{m})$ as follows:

$$a = sin(\lambda)\,, b = cos(2\delta)\,, c = cos(\lambda)\,, d = sin(2\delta)$$
$$e = cos(\psi - \phi)\,, f = cos(\delta)\,, g = sin(\psi - \phi)\,, h = sin(\delta)$$
$$p = cos\big(2(\psi - \phi)\big)\,, q = sin\big(2(\psi - \phi)\big)$$

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial h_1}{\partial m_1} & \cdots & \frac{\partial h_1}{\partial m_3} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_5}{\partial m_1} & \cdots & \frac{\partial h_5}{\partial m_3} \end{bmatrix} = \begin{bmatrix} 0 & ab & 0.5cd \\ abe - cfg & -2adg - che & cbg - afe \\ 2chp + adq & cfq - abp & -ahq - 0.5cdp \\ -cfe - abg & chg - 2ade & afg + cbe \\ adp - 2chq & abq + cfp & 0.5cdq - ahp \end{bmatrix}$$

Note also that the matrix $\boldsymbol{C}$ of coefficients in Eq1 is the Jacobian of $\boldsymbol{g}(\boldsymbol{h})$. The overall gradient is thus given by

$$\nabla_{\boldsymbol{m}} E = \boldsymbol{J}^{\top} \cdot \boldsymbol{C}^{\top} \cdot \nabla_{\boldsymbol{g}} E$$

A directed search can now be performed by iteratively updating the model parameters in the direction of negative gradient:

$$\boldsymbol{m}^{(k+1)} \leftarrow \boldsymbol{m}^{(k)} - \eta \cdot \nabla_{\boldsymbol{m}} E\big(\boldsymbol{m}^{(k)}\big)$$

6.0 Computation

I used object-oriented programming in Python to set up the inverse problem as a class called SeismicModel (appendix 1). This approach is suitable because the computing the gradient involves keeping track of the function's inner components, akin to backpropagation used in neural networks. Conceptually, the object is built to store a calculation graph:

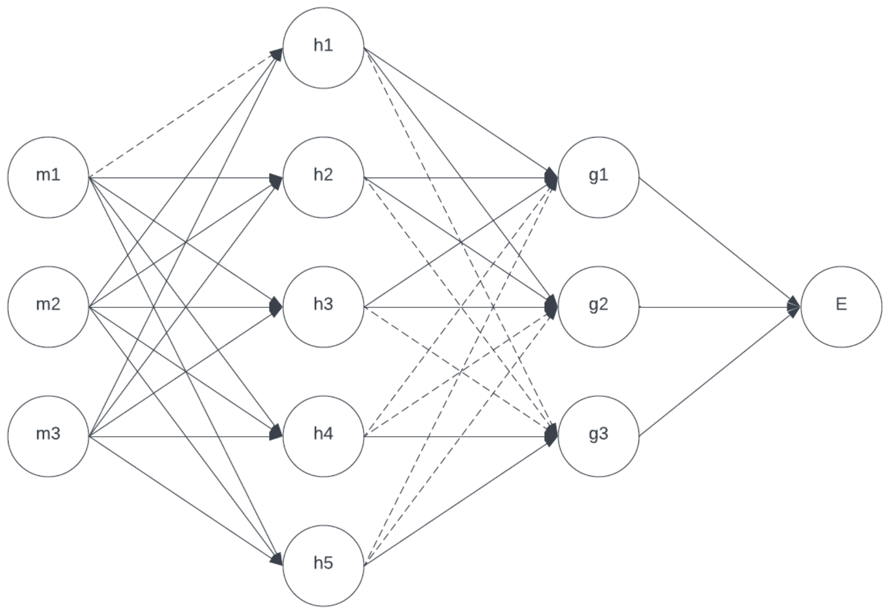


Figure 1: Calculation graph for forward problem

The SiesmicModel class stores the $\boldsymbol{m}$, $\boldsymbol{g}$ and E layers to keep track of iterates in model space, prediction in data space and the loss function. It also stores Jacobians $\boldsymbol{J}$, $\boldsymbol{C}$ and $\nabla_{\boldsymbol{g}}E$, which can be visualized as the set of edges from one layer to the next. Dashed edges represent positions where the corresponding Jacobian is 0.

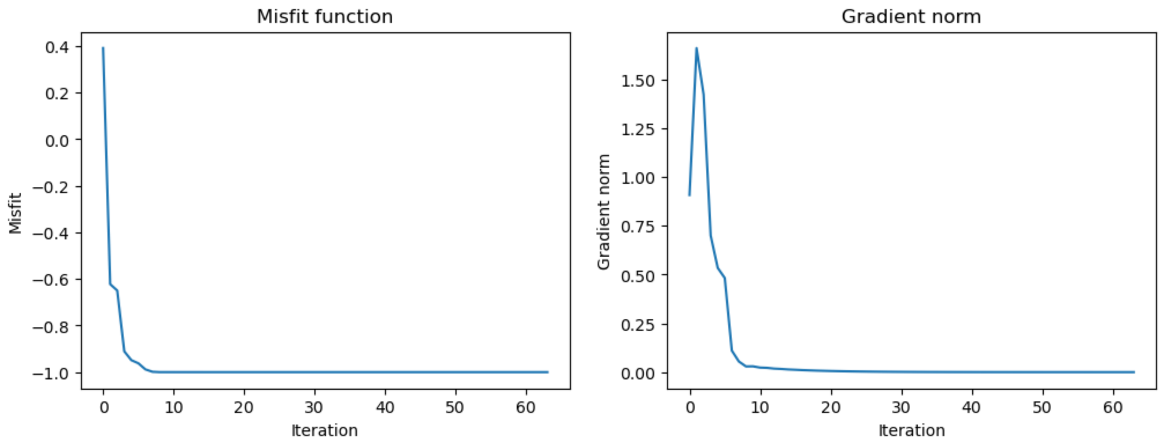
Standard and hierarchical grid search, which are used as benchmarks for comparison, are implemented as functions looping through model space. The difference between the two is that hierarchical grid search starts off with larger steps and zooms recursively on the best solution until the desired precision is reached.

For comparison, I simulated a source mechanism and inverted with the same convergence threshold (1e-5) for each method, keeping track of the final solution and duration.

7.0 Results

7.1 Solution for gradient descent

The algorithm converges to a solution very quickly. Printed information at each iteration (appendix 2) shows progress towards an optimal solution. Figures 2-4 also show how it behaved from a starting mechanism $\boldsymbol{m} = \mathbf{0}$:



Figures 2 and 3: Progress of misfit function and gradient norm

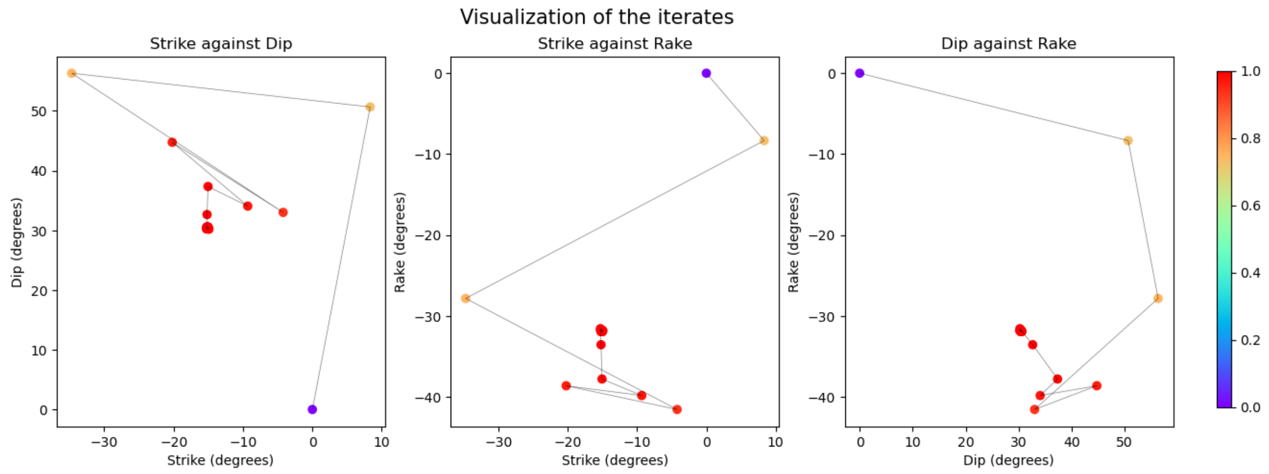


Figure 4: Progress of the iterates from source to solution, color-coded by how much the misfit has changed since the first iteration

7.2 Algorithm comparisons

Forward simulation:

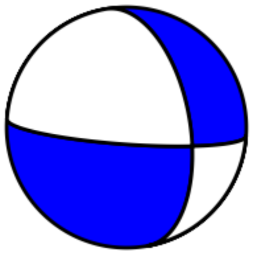
Mechanism (ψ, δ, λ)	[349.8°, 42.4°, -18.9°]
Quake depth	15 km
Epicentral distance	10°
Azimuth (ϕ)	200°
Beachball (visual)	

Table 1: Parameters for simulating source mechanism

Inversions:

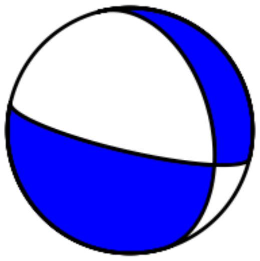
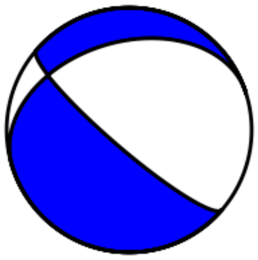
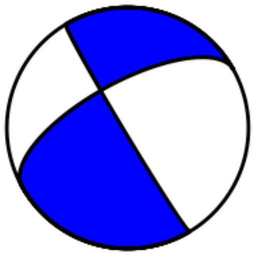
Method	Gradient descent + line search	Standard grid search	Hierarchical grid search
Solution	[-15.1°, 30.5°, -31.9°]	[130.7°, 76.0°, -68.8°]	[149.0°, 86.9° -34.2°]
1 – cos similarity	7.47×10^{-12}	5.67×10^{-6}	2.07×10^{-6}
Num iterations	63	453,960	108,000
Duration (s)	1.70×10^{-2}	6.72×10^{1}	3.20×10^{0}
Beachball (solution visual)			

Table 2: Summary metrics for different solution methods

8.0 Analysis

8.1 Method assessment

Gradient descent converges to a solution as expected, and we see that it is indeed similar to the simulated source mechanism when visualized. This is our immediate evidence that the method works: recovering the original mechanism after inversion. A few other diagnostics we can use are:

- The trend of the gradient norm: If a local minimum of a differentiable function exists, the computed gradient approaches $\mathbf{0}$ because gradient descent is globally convergent (Nocedal and Wright 2006). The trend in figure 3 tells us that the solution is at least a local minimum.
- The trend of the misfit function: By definition of the (modified) misfit function, a global minimum exists at $E(\boldsymbol{m}) = -1$ since cosine similarity has a range of $[-1,1]$. The trend in figure 2 tells us that the solution is a global minimum.

These diagnostics show also that the solution inverted for is good. However, the caveat evident in figure 4 is that for a starting point of $\boldsymbol{m} = \mathbf{0}$, the iterates get out of the strike's domain: $\psi \in (0°, 360°)$. Analytically this makes no difference because angles are periodic, so the solution is equivalent to one from inside the domain. But it would have to be accounted for computationally depending on the compatibility of the algorithm's implementation with broader infrastructure.

Regarding computational efficiency, gradient descent far surpasses both standard and hierarchical grid search, both in duration and number of iterations as seen in table 2. It also has a much better final precision even though each method had the same convergence threshold, because unlike the other two, it is able to improve precision by orders of magnitude in one step.

8.2 Implications for research

Moving forward with my research, I hope to use a hybrid approach when performing inversions with both real and simulated data. I expect the following benefits:

- Faster computations to minimize the duration of the simulation phase, which is where I currently am.
- A more intuitive feel for the solution space by assessing characteristics of solutions found using directed search. This could also show us if disjoint solution ranges exist for the same observed amplitude.
- Propagation of data uncertainties, whose approach may depend on the gradient. This is still an avenue for exploration since the misfit function is different.

9.0 Conclusion

The highlight of this project has been an exploration of gradient-based directed search methods for focal mechanism inversion in a strongly nonlinear, mixed-determined setting. We see that gradient descent with backtracking line search is not only a possible alternative to previously used grid search, but also a much more computationally efficient one.

The report gives an account of the mathematical and computational process undertaken to explore this new inversion method, and suggests potential avenues for pushing the current research forward accordingly.

10.0 References

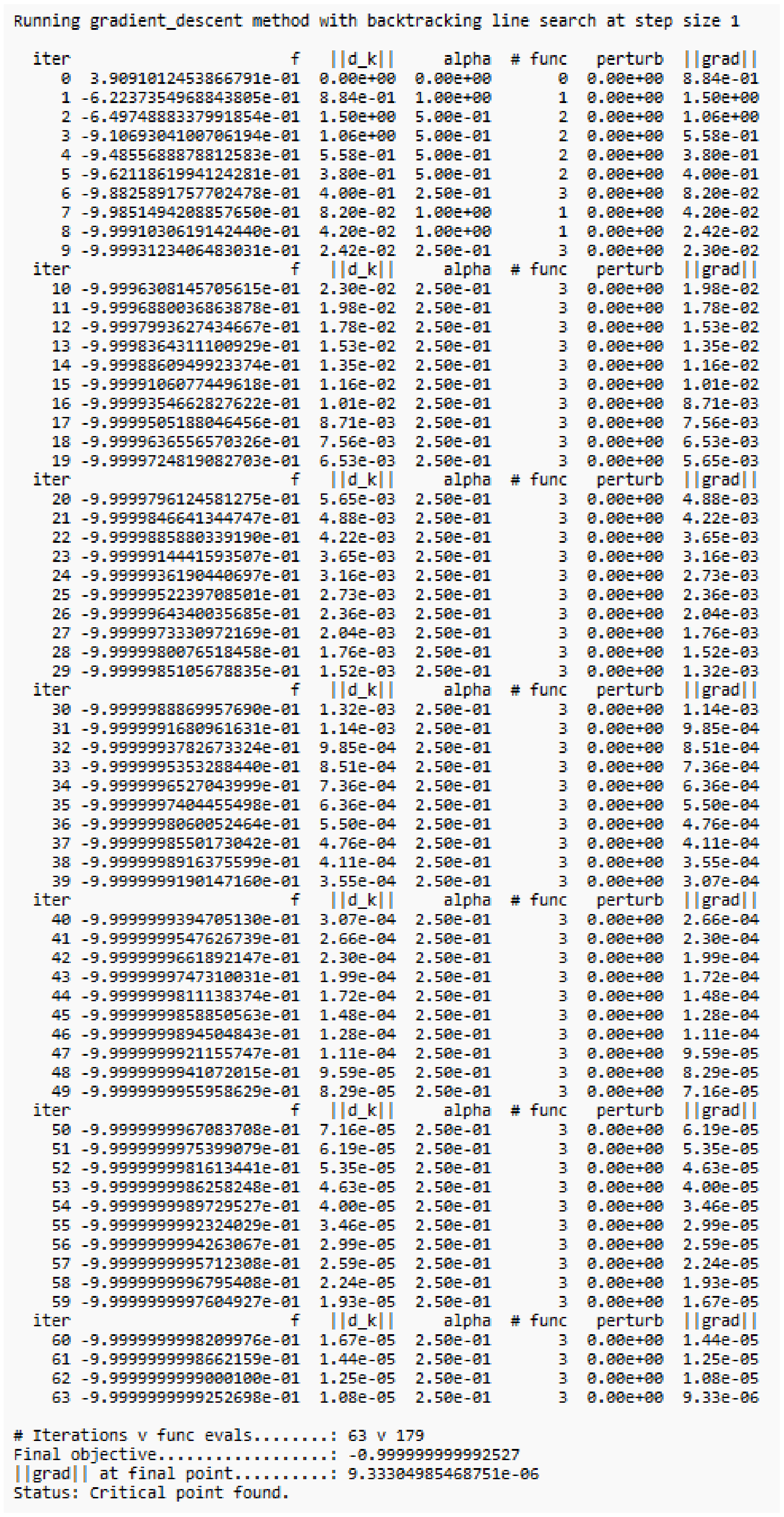
- Sita, Madelyn, and Suzan van der Lee. 2022. "Potential Volcano-Tectonic Origins and Faulting Mechanisms of Three Low-Frequency Marsquakes Detected by a Single InSight Seismometer." *Journal of Geophysical Research: Planets* 127 (10). https://doi.org/10.1029/2022je007309.
- Nocedal, Jorge, and Stephen J. Wright. 2006. Numerical Optimization, 2nd ed. Springer.

APPENDIX

Appendix 1: Source code for object-oriented implementation of seismic model

```
class SeismicModel(InvProblem):
    '''
    This class will compute the misfit function and its gradient
    for a given model.
    '''

    def __init__(self, phi, i, j, alpha, beta, d):
        '''
        Initialize with the parts of the model that don't change.
        Uses the same notation as in the project description.
        ALL ANGLES ARE IN RADIANS!!!
        '''
        # initialize misfits and grad norms for tracking
        self.misfits = []
        self.grad_norms = []
        self.iterates = []

        # initialize constant Jacobian matrix
        factors = array([alpha, beta, beta])**3
        self.C = array([[(3*cos(i)**2 - 1), -sin(2*i), -sin(i)**2, 0, 0],
                        [1.5*sin(2*j), cos(2*j), 0.5*sin(2*j), 0, 0],
                        [0, 0, 0, cos(j), sin(j)]])

        # scale each row of C by the corresponding factor
        self.C /= factors[:,newaxis]

        # also store J and nabla_E for later use
        self.J = zeros((5,3))
        self.nabla_E = zeros(3)

        # store other inputs for later use
        self.phi = phi
        self.d = d

        # store current value of m, g and E
        self.m = None
        self.g = None
        self.E = None

    def value(self, m):
        '''
        Compute the value of the misfit function at m.
        This works a lot like Rpattern from the summer project.
        '''
        self.m = m
        psi, delta, lamb = m

        # from Maddy's paper
        sR = .5*sin(lamb)*sin(2*delta)
        qR = sin(lamb)*cos(2*delta)*sin(psi-self.phi) + \
            cos(lamb)*cos(delta)*cos(psi-self.phi)
        pR = cos(lamb)*sin(delta)*sin(2*(psi-self.phi)) - \
            .5*sin(lamb)*sin(2*delta)*cos(2*(psi-self.phi))
        qL = -cos(lamb)*cos(delta)*sin(psi-self.phi) + \
            sin(lamb)*cos(2*delta)*cos(psi-self.phi)
        pL = .5*sin(lamb)*sin(2*delta)*sin(2*(psi-self.phi)) + \
            cos(lamb)*sin(delta)*cos(2*(psi-self.phi))

        # compute g from vector h
        h = array([sR, qR, pR, qL, pL])
        self.g = self.C @ h

        # compute the misfit function: negative cosine similarity
        self.E = -dot(self.d, self.g)/(linalg.norm(self.d)*linalg.norm(self.g))

        return self.E

    def gradient(self, m):
        '''
        Compute the gradient of the misfit function at m.
        '''
        psi, delta, lamb = m

        # update Jacobian J
        a, b, c, d = sin(lamb), cos(2*delta), cos(lamb), sin(2*delta)
        e, f, g, h = cos(psi-self.phi), cos(delta), sin(psi-self.phi), sin(delta)
        p, q = cos(2*(psi-self.phi)), sin(2*(psi-self.phi))
        self.J = array([[0, a*b, .5*c*d],
                  [a*b*e - c*f*g, -2*a*d*g - c*h*e, c*b*g - a*f*e],
                  [2*c*h*p + a*d*q, c*f*q - a*b*p, -a*h*q - .5*c*d*p],
                  [-c*f*e - a*b*g, -c*h*g - 2*a*d*e, a*f*g + c*b*e],
                  [a*d*p - 2*c*h*q, a*b*q + c*f*p, .5*c*d*q - a*h*p]])

        # update nabla_E
        cossim = -self.E
        ghat = self.g/linalg.norm(self.g)
        dhat = self.d/linalg.norm(self.d)
        self.nabla_E = (1/linalg.norm(self.g))*(cossim*ghat - dhat)

        # compute the gradient and store its norm
        grad = self.J.T @ self.C.T @ self.nabla_E
        self.grad_norms.append(linalg.norm(grad))

        # store the current misfit and iterate during gradient step
        self.misfits.append(self.E)
        self.iterates.append(self.m)

        return grad
```

Appendix 2: Iteration progress for gradient descent



```
Running gradient_descent method with backtracking line search at step size 1

  iter                       f    ||d_k||      alpha  # func    perturb    ||grad||
     0  3.9091012453866791e-01  0.00e+00  0.00e+00       0  0.00e+00  8.84e-01
     1 -6.2237354968843805e-01  8.84e-01  1.00e+00       1  0.00e+00  1.50e+00
     2 -6.4974888337991854e-01  1.50e+00  5.00e-01       2  0.00e+00  1.06e+00
     3 -9.1069304100706194e-01  1.06e+00  5.00e-01       2  0.00e+00  5.58e-01
     4 -9.4855688878812583e-01  5.58e-01  5.00e-01       2  0.00e+00  3.80e-01
     5 -9.6211861994124281e-01  3.80e-01  5.00e-01       2  0.00e+00  4.00e-01
     6 -9.8825891757702478e-01  4.00e-01  2.50e-01       3  0.00e+00  8.20e-02
     7 -9.9851494208857650e-01  8.20e-02  1.00e+00       1  0.00e+00  4.20e-02
     8 -9.9991030619142440e-01  4.20e-02  1.00e+00       1  0.00e+00  2.42e-02
     9 -9.9993123406483031e-01  2.42e-02  2.50e-01       3  0.00e+00  2.30e-02
  iter                       f    ||d_k||      alpha  # func    perturb    ||grad||
    10 -9.9996308145705615e-01  2.30e-02  2.50e-01       3  0.00e+00  1.98e-02
    11 -9.9996880036863878e-01  1.98e-02  2.50e-01       3  0.00e+00  1.78e-02
    12 -9.9997993627434667e-01  1.78e-02  2.50e-01       3  0.00e+00  1.53e-02
    13 -9.9998364311100929e-01  1.53e-02  2.50e-01       3  0.00e+00  1.35e-02
    14 -9.9998860949923374e-01  1.35e-02  2.50e-01       3  0.00e+00  1.16e-02
    15 -9.9999106077449618e-01  1.16e-02  2.50e-01       3  0.00e+00  1.01e-02
    16 -9.9999354662827622e-01  1.01e-02  2.50e-01       3  0.00e+00  8.71e-03
    17 -9.9999505188046456e-01  8.71e-03  2.50e-01       3  0.00e+00  7.56e-03
    18 -9.9999636556570326e-01  7.56e-03  2.50e-01       3  0.00e+00  6.53e-03
    19 -9.9999724819082703e-01  6.53e-03  2.50e-01       3  0.00e+00  5.65e-03
  iter                       f    ||d_k||      alpha  # func    perturb    ||grad||
    20 -9.9999796124581275e-01  5.65e-03  2.50e-01       3  0.00e+00  4.88e-03
    21 -9.9999846641344747e-01  4.88e-03  2.50e-01       3  0.00e+00  4.22e-03
    22 -9.9999885880339190e-01  4.22e-03  2.50e-01       3  0.00e+00  3.65e-03
    23 -9.9999914441593507e-01  3.65e-03  2.50e-01       3  0.00e+00  3.16e-03
    24 -9.9999936190440697e-01  3.16e-03  2.50e-01       3  0.00e+00  2.73e-03
    25 -9.9999952239708501e-01  2.73e-03  2.50e-01       3  0.00e+00  2.36e-03
    26 -9.9999964340035685e-01  2.36e-03  2.50e-01       3  0.00e+00  2.04e-03
    27 -9.9999973330972169e-01  2.04e-03  2.50e-01       3  0.00e+00  1.76e-03
    28 -9.9999980076518458e-01  1.76e-03  2.50e-01       3  0.00e+00  1.52e-03
    29 -9.9999985105678835e-01  1.52e-03  2.50e-01       3  0.00e+00  1.32e-03
  iter                       f    ||d_k||      alpha  # func    perturb    ||grad||
    30 -9.9999988869957690e-01  1.32e-03  2.50e-01       3  0.00e+00  1.14e-03
    31 -9.9999991680961631e-01  1.14e-03  2.50e-01       3  0.00e+00  9.85e-04
    32 -9.9999993782673324e-01  9.85e-04  2.50e-01       3  0.00e+00  8.51e-04
    33 -9.9999995353288440e-01  8.51e-04  2.50e-01       3  0.00e+00  7.36e-04
    34 -9.9999996527043999e-01  7.36e-04  2.50e-01       3  0.00e+00  6.36e-04
    35 -9.9999997404455498e-01  6.36e-04  2.50e-01       3  0.00e+00  5.50e-04
    36 -9.9999998060052464e-01  5.50e-04  2.50e-01       3  0.00e+00  4.76e-04
    37 -9.9999998550173042e-01  4.76e-04  2.50e-01       3  0.00e+00  4.11e-04
    38 -9.9999998916375599e-01  4.11e-04  2.50e-01       3  0.00e+00  3.55e-04
    39 -9.9999999190147160e-01  3.55e-04  2.50e-01       3  0.00e+00  3.07e-04
  iter                       f    ||d_k||      alpha  # func    perturb    ||grad||
    40 -9.9999999394705130e-01  3.07e-04  2.50e-01       3  0.00e+00  2.66e-04
    41 -9.9999999547626739e-01  2.66e-04  2.50e-01       3  0.00e+00  2.30e-04
    42 -9.9999999661892147e-01  2.30e-04  2.50e-01       3  0.00e+00  1.99e-04
    43 -9.9999999747310031e-01  1.99e-04  2.50e-01       3  0.00e+00  1.72e-04
    44 -9.9999999811138374e-01  1.72e-04  2.50e-01       3  0.00e+00  1.48e-04
    45 -9.9999999858850563e-01  1.48e-04  2.50e-01       3  0.00e+00  1.28e-04
    46 -9.9999999894504843e-01  1.28e-04  2.50e-01       3  0.00e+00  1.11e-04
    47 -9.9999999921155747e-01  1.11e-04  2.50e-01       3  0.00e+00  9.59e-05
    48 -9.9999999941072015e-01  9.59e-05  2.50e-01       3  0.00e+00  8.29e-05
    49 -9.9999999955958629e-01  8.29e-05  2.50e-01       3  0.00e+00  7.16e-05
  iter                       f    ||d_k||      alpha  # func    perturb    ||grad||
    50 -9.9999999967083708e-01  7.16e-05  2.50e-01       3  0.00e+00  6.19e-05
    51 -9.9999999975399079e-01  6.19e-05  2.50e-01       3  0.00e+00  5.35e-05
    52 -9.9999999981613441e-01  5.35e-05  2.50e-01       3  0.00e+00  4.63e-05
    53 -9.9999999986258248e-01  4.63e-05  2.50e-01       3  0.00e+00  4.00e-05
    54 -9.9999999989729527e-01  4.00e-05  2.50e-01       3  0.00e+00  3.46e-05
    55 -9.9999999992324029e-01  3.46e-05  2.50e-01       3  0.00e+00  2.99e-05
    56 -9.9999999994263067e-01  2.99e-05  2.50e-01       3  0.00e+00  2.59e-05
    57 -9.9999999995712308e-01  2.59e-05  2.50e-01       3  0.00e+00  2.24e-05
    58 -9.9999999996795408e-01  2.24e-05  2.50e-01       3  0.00e+00  1.93e-05
    59 -9.9999999997604927e-01  1.93e-05  2.50e-01       3  0.00e+00  1.67e-05
  iter                       f    ||d_k||      alpha  # func    perturb    ||grad||
    60 -9.9999999998209976e-01  1.67e-05  2.50e-01       3  0.00e+00  1.44e-05
    61 -9.9999999998662159e-01  1.44e-05  2.50e-01       3  0.00e+00  1.25e-05
    62 -9.9999999999000100e-01  1.25e-05  2.50e-01       3  0.00e+00  1.08e-05
    63 -9.9999999999252698e-01  1.08e-05  2.50e-01       3  0.00e+00  9.33e-06

# Iterations v func evals........: 63 v 179
Final objective..................: -0.999999999992527
||grad|| at final point..........: 9.33304985468751e-06
Status: Critical point found.
```