Math 564 - Project 05

Sajjad Uddin Mahmud

Dec 12, 2025

1 Results

A single optimization was performed using the ℓ_2 norm and the baseline initial guess. The objective value was reduced from 4.31×10^{-3} to 1.06×10^{-3}, corresponding to a 75.5% reduction.

Table 1.1: Optimization Performance

Metric	Value
Number of parameters	32
Initial objective value	4.31×10^{-3}
Final objective value	1.06×10^{-3}
Iterations	200
Function evaluations	363
Converged	No
Wall-clock time	~ 5300 s

1.1 Convergence History

Table 1.2 reports the evolution of the best and worst objective values during the Nelder–Mead iterations, together with the cumulative number of function evaluations. The results show a steady reduction of the objective value with diminishing improvement in later iterations.

The final objective value achieved was 1.06×10^{-3} after 200 iterations and 363 function evaluations, corresponding to a 75.5% reduction relative to the initial guess.

Table 1.2: Nelder–Mead convergence history (selected iterations)

Iteration	f_{best}	f_{worst}	Function calls
0	4.3105e-03	5.3341e-02	33
10	4.3105e-03	1.3972e-02	52
20	4.3105e-03	1.0040e-02	66
30	1.8363e-03	5.2380e-03	116
40	1.8363e-03	3.7465e-03	130
50	1.8363e-03	3.3428e-03	142
60	1.8363e-03	3.2626e-03	152
70	1.7867e-03	3.1155e-03	164
80	1.7867e-03	2.7553e-03	177
90	1.5632e-03	2.5106e-03	191
100	1.5632e-03	2.3021e-03	204
110	1.5214e-03	2.1562e-03	215
120	1.2544e-03	2.0078e-03	227
130	1.2544e-03	1.8080e-03	240
140	1.1956e-03	1.4225e-03	288
150	1.1709e-03	1.3472e-03	301
160	1.0962e-03	1.3045e-03	314
170	1.0813e-03	1.2645e-03	327
180	1.0709e-03	1.2258e-03	338
190	1.0709e-03	1.1956e-03	351

2 Discussion

The Nelder–Mead algorithm successfully reduced the objective without requiring derivatives. Although convergence was not achieved within the iteration limit, the objective decreased, indicating stable progress.

3 Repository

All code and iteration logs for this project are available in a public GitHub repository:

`https://github.com/sajjad30148/WSU_Math564_Fall2025`

The repository includes the code, and result folders containing iterations from each run.

4 Code

Listing 1: project01_main.py

```

import numpy as np
import time
import TF # Import the ToroidalPeriods calculator

# ==================== TASK 1: NELDER-MEAD OPTIMIZER ====================

def nelder_mead(f, x0, max_iter=200, tol=1e-6, alpha=1.0, gamma=2.0,
    rho=0.5, sigma=0.5, verbose=False):
    """
    Nelder-Mead optimization algorithm for derivative-free optimization.

    Parameters:
    -----------
    f : callable
        Objective function to minimize
    x0 : array-like
        Initial guess (will be used to create initial simplex)
    max_iter : int
        Maximum number of iterations (default: 200)
    tol : float
        Tolerance for convergence (default: 1e-6)
    alpha : float
        Reflection coefficient (default: 1.0)
    gamma : float
        Expansion coefficient (default: 2.0)
    rho : float
        Contraction coefficient (default: 0.5)
    sigma : float
        Shrink coefficient (default: 0.5)
    verbose : bool
        Print iteration details

    Returns:
    --------
    x_best : ndarray
        Optimal parameter vector
    f_best : float
        Optimal function value
    history : dict
        Optimization history
```

```
    """

    n = len(x0)

    print(f"Initializing simplex with {n+1} vertices...")

    # Create initial simplex (n+1 vertices in n-dimensional space)
    simplex = np.zeros((n+1, n))
    simplex[0] = x0

    # Create remaining vertices by perturbing each dimension
    for i in range(n):
        vertex = np.copy(x0)
        if vertex[i] != 0:
            vertex[i] *= 1.05 # 5% perturbation
        else:
            vertex[i] = 0.00025 # small perturbation for zero values
        simplex[i+1] = vertex

    # Evaluate function at all vertices
    print(f"Evaluating initial simplex ({n+1} function calls)...")
    f_values = []
    for i, vertex in enumerate(simplex):
        print(f" Evaluating vertex {i+1}/{n+1}...", end='\r')
        f_values.append(f(vertex))
    f_values = np.array(f_values)
    print(f" Completed {n+1}/{n+1} vertices! ")



    history = {
        'iterations': [],
        'f_best': [],
        'f_calls': n + 1,
        'converged': False
    }

    print(f"Initial best value: {np.min(f_values):.8f}")
    print("Starting optimization...\n")

    for iteration in range(max_iter):
        # Sort simplex by function values
        order = np.argsort(f_values)
        simplex = simplex[order]
        f_values = f_values[order]
```

```
        # Store best value
        history['iterations'].append(iteration)
        history['f_best'].append(f_values[0])

        # Print progress every 10 iterations
        if verbose and iteration % 10 == 0:
            print(f"Iter {iteration:3d}: f_best={f_values[0]:.10f}, f_worst={f_values[-1]:.10f}, f_calls={history['f_calls']}")

        # Check convergence: standard deviation of function values
        f_std = np.std(f_values)
        if f_std < tol:
            if verbose:
                print(f"\n*** CONVERGED at iteration {iteration} with f_std = {f_std:.2e} ***")
            history['converged'] = True
            break

        # Calculate centroid of best n points (excluding worst)
        centroid = np.mean(simplex[:-1], axis=0)

        # REFLECTION
        x_r = centroid + alpha * (centroid - simplex[-1])
        f_r = f(x_r)
        history['f_calls'] += 1

        if f_values[0] <= f_r < f_values[-2]:
            # Accept reflection
            simplex[-1] = x_r
            f_values[-1] = f_r

        elif f_r < f_values[0]:
            # EXPANSION - try to go further
            x_e = centroid + gamma * (x_r - centroid)
            f_e = f(x_e)
            history['f_calls'] += 1

            if f_e < f_r:
                simplex[-1] = x_e
                f_values[-1] = f_e
            else:
                simplex[-1] = x_r
                f_values[-1] = f_r

```

```python
        else:
            # CONTRACTION
            if f_r < f_values[-1]:
                # Outside contraction
                x_c = centroid + rho * (x_r - centroid)
            else:
                # Inside contraction
                x_c = centroid - rho * (simplex[-1] - centroid)

            f_c = f(x_c)
            history['f_calls'] += 1

            if f_c < min(f_r, f_values[-1]):
                simplex[-1] = x_c
                f_values[-1] = f_c
            else:
                # SHRINK - contract all points toward best point
                for i in range(1, n+1):
                    simplex[i] = simplex[0] + sigma * (simplex[i] - simplex[0])
                    f_values[i] = f(simplex[i])
                    history['f_calls'] += 1

    # Return best solution
    best_idx = np.argmin(f_values)
    return simplex[best_idx], f_values[best_idx], history


# ==================== TASK 2: OBJECTIVE FUNCTIONS ====================

def objective_L2_norm(x):
    """Objective function using L2 (Euclidean) norm - standard least squares"""
    Tc, Te = TF.ToroidalPeriods(np.array(x))
    if np.isnan(Tc[0]):
        return np.inf
    return np.linalg.norm(Tc - Te) / np.linalg.norm(Te)

def objective_L1_norm(x):
    """Objective function using L1 (Manhattan) norm - robust to outliers"""
    Tc, Te = TF.ToroidalPeriods(np.array(x))
    if np.isnan(Tc[0]):
        return np.inf
    return np.linalg.norm(Tc - Te, ord=1) / np.linalg.norm(Te, ord=1)
```

```python

def objective_Linf_norm(x):
    """Objective function using L-infinity norm - minimizes maximum error"""
    Tc, Te = TF.ToroidalPeriods(np.array(x))
    if np.isnan(Tc[0]):
        return np.inf
    return np.linalg.norm(Tc - Te, ord=np.inf) / np.linalg.norm(Te, ord=np.inf)


# ==================== TASK 3: SOLVE OPTIMIZATION PROBLEM ====================

def solve_single_optimization(norm_name='L2', initial_guess='baseline', max_iter=200):
    """
    Solve a single optimization problem.

    Parameters:
    -----------
    norm_name : str
        'L2', 'L1', or 'Linf'
    initial_guess : str
        'baseline', 'plus10', or 'minus10'
    max_iter : int
        Maximum iterations (default: 200)
    """

    # Initial decision variable vector (from project description)
    x0_base = np.array([0.6, 2.6, -3.6, 7.0, -7.0, 11.2, -1.6, 5.0,
                        -3.0, 5.6, -6.4, 8.0, 5.6, -1.0, -4.4, 8.8,
                        -18.6, 22.2, -4.8, 10.0, 0.8, -2.0, -17.2, 22.4,
                        -9.2, 17.2, -14.0, 11.4, 1.0, -2.2, 1.4, 6.4])

    # Select initial guess
    if initial_guess == 'baseline':
        x0 = x0_base
    elif initial_guess == 'plus10':
        x0 = x0_base * 1.1
    elif initial_guess == 'minus10':
        x0 = x0_base * 0.9
    else:
        raise ValueError("initial_guess must be 'baseline', 'plus10', or 'minus10'")
```

```python

    # Select objective function
    if norm_name == 'L2':
        obj_func = objective_L2_norm
    elif norm_name == 'L1':
        obj_func = objective_L1_norm
    elif norm_name == 'Linf':
        obj_func = objective_Linf_norm
    else:
        raise ValueError("norm_name must be 'L2', 'L1', or 'Linf'")

    print("="*70)
    print(f"OPTIMIZATION: {norm_name} norm with {initial_guess} initial
        guess")
    print("="*70)

    start_time = time.time()
    x_opt, f_opt, history = nelder_mead(
        obj_func,
        x0,
        max_iter=max_iter,
        tol=1e-7,
        verbose=True
    )
    elapsed_time = time.time() - start_time

    print(f"\n{'='*70}")
    print("RESULTS:")
    print(f"{'='*70}")
    print(f"Optimal objective value: {f_opt:.12f}")
    print(f"Function calls: {history['f_calls']}")
    print(f"Iterations: {len(history['iterations'])}")
    print(f"Converged: {history['converged']}")
    print(f"Time elapsed: {elapsed_time:.2f} seconds")
    if len(history['f_best']) > 0:
        print(f"Initial f(x0): {history['f_best'][0]:.12f}")
        print(f"Improvement: {(1 - f_opt/history['f_best'][0])*100:.4f}%")
    print(f"{'='*70}\n")

    return x_opt, f_opt, history


def solve_all_optimizations(max_iter=200):
    """Run all 9 optimization combinations"""

```

```
    norms = ['L2', 'L1', 'Linf']
    guesses = ['baseline', 'plus10', 'minus10']

    results = {}

    print("\n" + "="*70)
    print("EARTH MODEL OPTIMIZATION - PROJECT 5 (ALL COMBINATIONS)")
    print("="*70 + "\n")

    total = len(norms) * len(guesses)
    count = 0

    for norm in norms:
        for guess in guesses:
            count += 1
            print(f"\n>>> Running optimization {count}/{total} <<<\n")

            x_opt, f_opt, history = solve_single_optimization(norm, guess,
                max_iter)

            key = f"{guess}_{norm}"
            results[key] = {
                'x_optimal': x_opt,
                'f_optimal': f_opt,
                'history': history,
                'norm': norm,
                'initial_guess': guess
            }

    # Print summary
    print("\n" + "="*70)
    print("SUMMARY OF ALL RESULTS")
    print("="*70)
    for key, res in results.items():
        print(f"{key:20s}: f_opt={res['f_optimal']:.10f},
            calls={res['history']['f_calls']},
            conv={res['history']['converged']}")

    return results


# ==================== MAIN EXECUTION ====================

if __name__ == "__main__":
    print("\nProject 5 - Earth Model Optimization (Optimized Version)")
```

```python
    print("="*70)
    print("="*70 + "\n")

    # OPTION 1: Run a single optimization (faster for testing)
    print("Running SINGLE optimization (L2 norm, baseline guess)...")


    x_opt, f_opt, history = solve_single_optimization('L2', 'baseline',
        max_iter=200)

    # Save result
    with open('optimal_parameters.txt', 'w') as f:
        f.write("Optimal Earth Model Parameters\n")
        f.write("="*70 + "\n")
        f.write(f"Objective value (L2 norm): {f_opt:.12f}\n")
        f.write(f"Function calls: {history['f_calls']}\n")
        f.write(f"Converged: {history['converged']}\n\n")
        f.write("Parameters:\n")
        for i, val in enumerate(x_opt):
            f.write(f" x[{i:2d}] = {val:12.8f}\n")

    print("\nResults saved to 'optimal_parameters.txt'")

    # OPTION 2: Uncomment below to run all 9 optimizations
    # results = solve_all_optimizations(max_iter=200)
```