Documentation for GitHub Repository CSC431-Final

The code in this repository was created by Kimberly Barchenger and Jennifer Shaw based off of the code shared in DePaul University course CSC 431 by Massimo DiPierro. This implementation is written in Javascript with test functions in an html file; the original code was presented in Python. This code is distributed under the BSD license.

The code is designed to implement various numerical algorithms, including functions from linear algebra, solvers for non-linear equations, and one-dimensional optimizations. What follows is a listing of the included functions.

Github link: https://github.com/kbarchenger/CSC431-Final

Linear Algebra

- Matrix: creates a matrix object, which is essentially an array of arrays.
  - Parameter: data, holds the data for the matrix
  - Parameter: rows, holds the number of rows in the matrix
  - Parameter: cols, holds the number of columns in the matrix
- print_matrix(m): prints the matrix m to the console one row at a time
- identity_matrix(rows): creates an identity matrix, of size rows
- diagonal_matrix(d): creates a matrix filled with 0's, except the diagonal filled based on d
- matrix_from_list(l): creates a matrix from the list of list l
- Matrix prototype functions:
  - neg(): negates each value in this matrix
  - add(B): adds B to this matrix, for B either a matrix or number
  - sub(B): subtracts B from this matrix, for B either a matrix or number
  - mult(B): multiplies this matrix by B, for B either a matrix or number
  - div(B): divides this matrix by B, for B either a matrix or number
  - inverse(): returns the inverse of this matrix (uses Gauss-Jordan elimination)
  - swap_rows(i, j): swaps the rows i and j in this matrix
  - transpose(): transposes this matrix (swaps rows with columns)
  - is_almost_symmetric(): determines if the matrix is almost symmetric
  - is_almost_zero(): determines if the matrix is almost zero
- norm(A, p): finds the p-norm of A, where A is an array or matrix
  (Note: for matrices, only the 1-norm is implemented)
- condition_number(f, x): finds the condition number of matrix f or of function f at point x
- exp(x): computes the exponential of x for x a matrix or a number
- Cholesky(A): runs the Cholesky decomposition on matrix A
- is_positive_definite(A): determines if the matrix A is positive definite and returns a Boolean
- Markovitz(mu, A, r_free): assesses the Markovitz risk/return for given mu (arithmetic return matrix), A (covariance matrix), and r_free risk-free return
- fit_least_squares(points, f): computes the fitting function for the given list of points that matches the form given in f (for a quadratic fitting function , f = [1, x, x^2])

Solvers for non-linear equations

- D(f, x): returns the first derivative of f(x) at x
- DD(f,x): returns the second derivative of f(x) at x
- solve_fixed_point(f, x): uses the fixed point method to find a zero of f(x) near x
- solve_bisection(f, a, b): uses the bisection method to find a zero of f(x) between a and b
- solve_newton(f, x): uses the Newton method to find a zero of f(x) near x
- solve_secant(f, x): uses the secant method to find a zero of f(x) near x
- solve_newton_stabilized(f, a, b): uses the Newton stabilized method to find a zero of f(x) between a and b

One-dimensional optimization

- optimize_bisection(f, a, b): uses the bisection method to find a minimum or maximum of f(x) between a and b
- optimize_newton(f, x): uses the Newton method to find a minimum or maximum of f(x) near x
- optimize_secant(f, x): uses the secant method to find a minimum or maximum of f(x) near x
- optimize_newton_stabilized(f, a, b): uses the Newton stabilized method to find a minimum or maximum of f(x) between a and b
- optimize_golden_search(f, a, b): uses the Golden-section search method to find a minimum or maximum of f(x) between a and b

Multi-variable functions

- partial(f, i): computes the partial of f with respect to x_i
- gradient(f, x): computes the gradient of f at point x
- hessian(f, x): computes the Hessian of f at point x
- jacobian(f, x): computes the Jacobian of f at point x
- solve_newton_multi(f,x): uses the Newton method to find the root of a multi-dimensional function f(x), near point x
- optimize_newton_multi(f, x): uses the Newton method to find a minimum or maximum of f(x), a multi-dimensional function, near point x