Höhere Mathematik

Jil Zerndt, Lucien Perret
January 2025

Übersicht der Implementierungen

Basics

matrix_vector_basics.py Grundlegende Matrix- und Vektoroperationen
- `matrix_multiply(A, B)` - Multipliziert zwei Matrizen
- `matrix_subtract(A, B)` - Subtrahiert zwei Matrizen
- `matrix_add(A, B)` - Addiert zwei Matrizen
- `matrix_transpose(A)` - Transponiert eine Matrix
- `norm_vector(v, p=2)` - Berechnet p-Norm eines Vektors
- `norm_matrix(A, p=2)` - Berechnet Matrixnorm
- `normalize_vector(v)` - Normalisiert einen Vektor
- `is_symmetric(A)` - Prüft ob Matrix symmetrisch ist
- `condition_number(A, p=2)` - Berechnet Konditionszahl
- `plot_matrix_heatmap(A, title)` - Visualisiert Matrix als Heatmap

rechnerarithmetik.py Funktionen für Computerarithmetik und Fehleranalyse
- `is_close(a, b)` - Prüft Gleichheit mit Toleranz
- `get_machine_epsilon()` - Berechnet Maschinengenauigkeit
- `relative_error(true, approx)` - Berechnet relativen Fehler
- `absolute_error(true, approx)` - Berechnet absoluten Fehler
- `analyze_float_representation(x)` - Analysiert Gleitkommadarstellung
- `plot_rounding_errors(f, x_range)` - Visualisiert Rundungsfehler

plot_functions.py Plotting-Funktionen für numerische Methoden
- `plot_function_1d(f, x_range)` - Plot einer 1D-Funktion
- `plot_convergence(errors)` - Visualisiert Konvergenzverhalten
- `plot_function_2d(f, x_range, y_range)` - 3D-Plot einer 2D-Funktion
- `plot_contour(f, x_range, y_range)` - Erstellt Konturplot

NSP (Nullstellenprobleme)

newton.py Newton-Verfahren und Varianten
- `newton_method(f, df, x0)` - Standardverfahren
- `simplified_newton(f, df, x0)` - Vereinfachtes Newton-Verfahren
- `damped_newton(f, df, x0)` - Gedämpftes Newton-Verfahren
- `analyze_convergence_order(x_hist, x_star)` - Konvergenzanalyse
- `visualize_newton_steps(f, df, x0)` - Visualisierung der Iterationen
- `plot_basins_of_attraction(f, df)` - Visualisierung der Einzugsbereiche

secant.py Sekantenverfahren und Varianten
- `secant_method(f, x0, x1)` - Standardverfahren
- `regula_falsi(f, a, b)` - Regula-Falsi-Methode
- `modified_secant(f, x0)` - Modifiziertes Sekantenverfahren
- `visualize_secant_steps(f, x0, x1)` - Visualisierung
- `analyze_convergence(x_hist, x_star)` - Konvergenzanalyse

fixpunktiteration.py Fixpunktiteration und Konvergenzanalyse
- `fixed_point_iteration(g, x0)` - Fixpunktiteration
- `estimate_lipschitz_constant(g, x)` - Schätzt Lipschitz-Konstante
- `verify_banach_conditions(g, a, b)` - Prüft Banach-Bedingungen
- `fixed_point_iteration_with_error(g, x0)` - Mit Fehlerabschätzung
- `analyze_convergence(g, x0, x_star)` - Konvergenzanalyse
- `plot_iteration_process(g, x0)` - Visualisierung

nsp_fehlerabschatzung.py Fehlerabschätzung für Nullstellenverfahren
- `verify_bracket(f, x, epsilon)` - Prüft Einschließung
- `estimate_error_bound(f, df, x)` - Schätzt Fehlerschranke
- `compute_residual(f, x)` - Berechnet Residuum
- `verify_convergence_conditions(f, df, x)` - Prüft Konvergenzbedingungen
- `estimate_working_precision(f, x)` - Schätzt Arbeitsgenauigkeit
- `track_error_convergence(f, df, x0)` - Verfolgt Fehlerentwicklung

LGS (Lineare Gleichungssysteme)

gauss.py Gauß-Eliminationsverfahren
- `forward_elimination(A, b)` - Vorwärtselimination
- `back_substitution(U, y)` - Rückwärtssubstitution
- `gauss_elimination(A, b)` - Komplettes Gauß-Verfahren
- `compute_residual(A, x, b)` - Berechnet Residuum
- `estimate_error(A, x, b)` - Fehlerabschätzung
- `determine_matrix_rank(A)` - Bestimmt Matrixrang
- `analyze_pivoting_strategies()` - Vergleicht Pivotisierungsstrategien

lr_comp.py LR-Zerlegung (LU-Zerlegung)
- `lr_decomposition(A)` - LR-Zerlegung
- `forward_substitution(L, b)` - Vorwärtssubstitution
- `back_substitution(R, y)` - Rückwärtssubstitution
- `solve_lr(L, R, b)` - Löst System mit LR-Zerlegung
- `compute_determinant(R)` - Berechnet Determinante
- `invert_matrix(A)` - Berechnet Inverse

qr_comp.py QR-Zerlegung und Anwendungen
- `householder_vector(x)` - Berechnet Householder-Vektor
- `apply_householder(A, v, beta)` - Wendet Householder-Transformation an
- `qr_decomposition(A)` - QR-Zerlegung
- `solve_qr(A, b)` - Löst System mit QR-Zerlegung
- `least_squares_qr(A, b)` - Löst Ausgleichsproblem

iterative.py Iterative Lösungsverfahren
- `decompose_matrix(A)` - Zerlegt Matrix (D, L, U)
- `jacobi_iteration(A, b, x0)` - Jacobi-Verfahren
- `gauss_seidel_iteration(A, b, x0)` - Gauß-Seidel-Verfahren
- `sor_iteration(A, b, x0, omega)` - SOR-Verfahren
- `estimate_spectral_radius(A)` - Schätzt Spektralradius
- `check_convergence_conditions(A)` - Prüft Konvergenzbedingungen

pivotisierung.py Pivotisierungsstrategien
- `find_pivot_element(A, k, n)` - Findet Pivotelement
- `swap_rows(A, b, i, j)` - Tauscht Zeilen
- `swap_columns(A, i, j)` - Tauscht Spalten
- `PivotingManager` - Klasse zur Verwaltung der Pivotisierung
  - `set_scaling(A)` - Setzt Skalierungsfaktoren
  - `find_pivot(A, k)` - Findet Pivotelement
  - `apply_pivot(A, b, k)` - Wendet Pivotisierung an
  - `get_permutations()` - Liefert Permutationen

konvergenzcheck.py Konvergenzüberprüfung für iterative Methoden
- `ConvergenceCriterion` - Enumeration der Konvergenzkriterien
- `ConvergenceChecker` - Prüft Konvergenzkriterien
- `ConvergenceMonitor` - Überwacht Konvergenzverhalten
- `run_iterative_method()` - Führt iterative Methode aus

EW_EV (Eigenwerte & Eigenvektoren)

ew_matrix.py Matrixoperationen für Eigenwertberechnung
- `identity_matrix(n)` - Erstellt Einheitsmatrix
- `scalar_matrix(n, scalar)` - Erstellt Skalarmatrix
- `is_symmetric(A)` - Prüft Symmetrie
- `is_positive_definite(A)` - Prüft positive Definitheit
- `determinant(A)` - Berechnet Determinante
- `trace(A)` - Berechnet Spur
- `characteristic_polynomial(A, lambda)` - Charakteristisches Polynom
- `gerschgorin_circles(A)` - Berechnet Gerschgorin-Kreise

complex_numbers.py Komplexe Zahlen für Eigenwertberechnungen
- `Complex` - Klasse für komplexe Zahlen
  - Grundoperationen (+, -, *, /)
  - `conjugate()` - Konjugiert komplexe Zahl
  - `abs()` - Berechnet Betrag
  - `arg()` - Berechnet Argument
  - `to_polar()` - Konvertiert in Polarform
- `roots_of_unity(n)` - Berechnet Einheitswurzeln
- `solve_quadratic(a, b, c)` - Löst quadratische Gleichung
- `matrix_eigenvalues(A)` - Schätzt Eigenwerte

iterations.py Iterative Methoden für Eigenwertberechnung
- `power_iteration_with_deflation()` - Potenzmethode mit Deflation
- `inverse_iteration_with_shifts()` - Inverse Iteration
- `qr_algorithm_basic()` - QR-Algorithmus (Basis)
- `qr_algorithm_with_shifts()` - QR mit Wilkinson-Shifts
- `simultaneous_iteration()` - Simultane Iteration