ZHAW · HM1 & HM2

Python-Skripte · Zusammenfassung

Repository: github.com/ShinoDev110/ZHAW-IT-HM1-HM2-Python

Inhalt: Eigenständige Python-Skripte zu allen Übungsthemen der Vorlesungen Höhere Mathematik 1 und 2. Die Skripte decken sowohl die Aufgaben aus dem Skript als auch alle verwendeten Übungsblätter ab. Aufgeteilt in zwei Hauptordner (`HM1/` und `HM2/`), jeweils in Kapitel-Ordner gegliedert.

Aufbau jedes Skripts: TOPIC-Header → **PART 1 Inputs** → **PART 2 Methodenwahl** → **PART 3 Implementation** → **PART 4 Aufruf**. Konvention siehe `AGENTS.md` im Repo.

HM1 — Höhere Mathematik 1

Maschinenzahlen

- **compute_min_max_machine_number** — kleinste/grösste positive Maschinenzahl für M(B, n, e_min, e_max)
- **count_machine_numbers** — Anzahl darstellbarer Maschinenzahlen
- **compute_machine_precision** — Maschinengenauigkeit eps = (B/2)·B^(−n)
- **round_to_machine_number** — Dezimalwert → IEEE-Bitmuster |Sign|Exp|Mantisse|
- **sum_series_with_rounding** — Σ 1/i² auf-/absteigend → Reihenfolge-Effekt bei Rundung
- **approximate_e_with_powers_of_ten** — (1 + 1/10ⁿ)^(10ⁿ) vs. math.e — Konvergenzabbruch
- **test_precision_loss_at_1_plus_eps** — ab welchem n ist 1 + 10^(−n) nicht mehr von 1 unterscheidbar

Funktionen — Basis & Nullstellen

- **differentiate_and_integrate_symbolic** — symbolische Ableitung & Stammfunktion + Auswertung
- **simplify_function** — symbolische Vereinfachung via sympy
- **compute_condition_number_of_function** — κ_f(x) analytisch, numerisch oder per Sweep
- **evaluate_polynomial_with_horner** — Horner-Schema für p(x), p'(x), P(x)
- **bisection / fixed_point / newton / simplified_newton / secant** — Nullstellenverfahren mit Toleranz- oder Iterationsstopp
- **classify_attractive_repulsive_fixed_point** — Klassifikation via |F'(x̄)|
- **estimate_iterations_a_priori** — Banach-Schranke aus Lipschitz α und erstem Schritt
- **estimate_convergence_order** — p und C aus Iterationsverlauf schätzen
- **verify_root_with_sign_change** — harte Schranke |x* − x̂| ≤ r per Vorzeichenwechsel

Funktionen — Visualisierung

- **plot_function_generic** — linear / log / semilog Plotter für sympy-Strings
- **plot_fixed_point_iteration** — F(x) − x mit attraktiven/repulsiven Fixpunkten
- **plot_newton_iteration / plot_secant_iteration** — Iterationspfad visualisieren
- **plot_polynomial_instability_demo** — Auslöschungsproblem (x−2)⁷ entwickelt vs. kompakt

Matrizen — Basisoperationen

- **invert_matrix** — A^{-1} via numpy.linalg.inv
- **check_orthogonal_matrix** — Test $A \cdot A^T \approx I$
- **check_diagonally_dominant** — Zeilen-/Spaltensummen-Kriterium
- **compute_matrix_norm_and_condition** — $\|A\|_1$, $\|A\|_2$, $\|A\|_\infty$ und cond
- **compute_vector_norm_and_condition** — $\|v\|_1$, $\|v\|_2$, $\|v\|_\infty$

Matrizen — Zerlegungen & Lineare Systeme

- **decompose_with_lr / plr / qr / qr_householder** — LR, PLR (mit Pivot), QR (numpy & eigene Householder)
- **solve_with_gauss / lr / plr / cramer** — Lösung von Ax = b mit verschiedenen Verfahren
- **estimate_arithmetic_complexity** — Faustregeln (2/3)n³ und 2n²

Matrizen — Iterative Verfahren & Fehler

- **solve_with_jacobi / gauss_seidel** — iterative Verfahren mit a-priori / a-posteriori Stopp
- **estimate_error_general / right_side** — Fehlerschranken für Störungen in A und/oder b

Interpolation, Eigenwerte & Komplexe Dynamik

- **interpolate_with_cubic_polynomial** — kubisches Polynom durch 4 Punkte via Vandermonde
- **compute_complete_eigen_analysis** — vollständige Analyse mit algebraischer & geometrischer Vielfachheit
- **eigenvalues_only / algebraic_multiplicity / characteristic_polynomial** — Einzelaspekte der Eigenwertanalyse
- **compute_spectral_radius / spectrum** — ρ(B) für Jacobi/Gauss-Seidel, volles Spektrum
- **build_diagonalization_matrix_t** — Diagonalisierung: T aus Eigenvektoren
- **qr_iteration / power_method** — QR-Iteration (mit Rayleigh-Shift), Von-Mises-Verfahren
- **plot_mandelbrot_set** — Mandelbrot-Menge als Iterationen-bis-Escape

HM2 — Höhere Mathematik 2

Nichtlineare Systeme — Newton-Verfahren

- **solve_nonlinear_system_newton** — Standard-Newton-Iteration
- **solve_…_simplified_newton / frozen_jacobian_newton** — vereinfachte und Frozen-Jacobian-Varianten
- **compare_newton_methods** — Vergleich Standard / vereinfacht / frozen
- **solve_newton_with_iteration_norms** — Residuum- und Schrittnorm pro Iteration
- **solve_all_newton_solutions** — mehrere Startvektoren → alle Lösungen sammeln
- **solve_3d_nonlinear_system_damped_newton** — gedämpftes Newton mit Norm-Logging
- **fit_soil_pressure_model_damped_newton** — Bodendruck-Modell-Fit + Bisektion für min. Scheibenradius

Nichtlineare Systeme — Linearisierung & Plots

- **calculate_jacobian_matrix** — symbolische Jacobi-Matrix + Auswertung
- **linearize_multivariable_function** — Linearisierung am gewählten Punkt
- **compute_partial_derivatives_and_linearize** — partielle Ableitungen 1. Ordnung + Linearisierung
- **plot_implicit_equations** — implizite Gleichungen mit sympy.plot_implicit
- **plot_nonlinear_system_contours** — Nulllinien eines 2D-Systems
- **plot_wave_equation_wireframe** — 3D-Wireframe wellenartiger Funktionen
- **visualize_2d_scalar_function** — Surface / Wireframe / Contour einer 2-Variablen-Funktion
- **plot_projectile_range_and_ideal_gas** — Wurfweite & ideales Gas als 3D-Plots

Interpolation & Ausgleichsrechnung

- **interpolate_value_with_lagrange** — Lagrange-Polynom (eigene Implementation)
- **interpolate_with_polyfit_polyval** — numpy.polyfit/polyval, optional zentriert
- **compare_lagrange_vs_polyfit** — Overlay-Vergleich
- **interpolate_with_natural_cubic_spline** — natürlicher kubischer Spline (Kap. 6.2.3)
- **interpolate_with_scipy_cubic_spline** — scipy.interpolate.CubicSpline mit Randbedingungen
- **compare_cubic_spline_methods** — eigener Spline vs. scipy vs. hochgradiges Polynom
- **fit_polynomial_with_normal_equations** — Polynom-LS via Normalgleichungen / QR / polyfit
- **fit_multivariate_linear_regression** — multivariate lineare Regression mit Intercept
- **fit_data_via_log_linearization** — exponentielle Modelle via Logarithmus linearisieren
- **fit_with_gauss_newton / damped_gauss_newton** — Gauss-Newton ungedämpft und gedämpft (δ/2^p)
- **compare_gauss_newton_methods** — gedämpft vs. ungedämpft, mehrere Startvektoren
- **fit_with_scipy_optimize_fmin** — direkte Minimierung mit Nelder-Mead

Numerische Integration

- **integrate_with_summed_rectangle / trapezoidal / simpson_rule** — summierte Newton-Cotes-Formeln (äquidistant)
- **integrate_with_trapezoidal_rule_non_equidistant** — Trapezregel für tabellierte Daten
- **integrate_with_gauss_formulas** — Gauss-Quadratur mit 1, 2 oder 3 Stützstellen
- **integrate_with_romberg_extrapolation** — vollständiges Romberg-Schema, T(0, m)
- **estimate_required_step_size** — max. h bzw. min. n aus Fehlerschranken (Satz 7.1)
- **compare_quadrature_methods** — alle Verfahren vs. scipy.integrate.quad

Gewöhnliche Differentialgleichungen

- **plot_direction_field** — Richtungsfeld y'(x) = f(x, y) per meshgrid + quiver
- **solve_ode_with_euler / midpoint / modified_euler** — Einschrittverfahren (Ordnung 1 bzw. 2)
- **solve_ode_with_classical_runge_kutta** — klassisches RK4 (Ordnung 4)
- **solve_ode_with_custom_runge_kutta** — generisches s-stufiges RK aus Butcher-Tableau
- **compare_ode_single_step_methods** — Euler / Mittelpunkt / Heun / RK4 + globaler Fehler
- **reduce_higher_order_ode_to_system** — k-te Ordnung → System 1. Ordnung via sympy
- **solve_ode_system_with_midpoint** — vektorwertiges System mit Mittelpunktverfahren
- **solve_rocket_ascent_via_integration** — v(t), h(t) aus a(t) per kumulierter Trapezregel

⚠ **Wichtig:** Korrektheit der Skripte ist nicht garantiert — Resultate vor Verwendung immer selbst überprüfen. *Pull Requests, Bug-Reports und Ergänzungen sind willkommen.*