「計算機実験 I」レポート課題 (2023/06/12 更新)

言語の指定がない課題については、Python や Julia などでプログラムを作成してもよい。ただし、その場合でも収束の様子などの解析はきちんと行うこと

課題は順次追加・修正の可能性あり

[数値誤差・アルゴリズム基礎]

1-1) $f(x) = \sin x$ について、$x = 0.3\pi$ における $f'(x)$ の値を数値微分により計算するプログラムを作成せよ。数値微分の刻みを $h = 1, 1/2, 1/4, 1/8, \cdots$ と減少させていった時、誤差がどのように振る舞うか図示せよ。h の最小値は 10^{-16} 程度まで計算すること。一次近似 (前進差分) と二次近似 (中心差分) における誤差の振る舞いの違いを調べよ。同様に、$x = 0.3\pi$ における $f''(x)$ の値を数値微分により計算し、その誤差を評価せよ。特に数値微分の刻み幅に対する誤差の振る舞いについての理論的な性質を論じ、それと実際の実験結果の整合性を検証せよ[*1]。一致しない場合はその理由を議論せよ

1-2) $f(x) = \tanh x + 0.2x + 0.3 = 0$ の解を Newton 法により求めよ。反復にしたがって、値がどのように真値 ($x = -0.2544612950513368\cdots$) に近づいていくか図示せよ。また、初期値による収束の違いを調べ、その理由について考察せよ

1-3) 上と同様の計算を $f(x) = x^5 - 7x^4 + 16x^3 - 8x^2 - 16x + 16 = 0$ に対して行ってみよ。残差 $|x_{t+1} - x_t|$ と誤差 $f(x_t)$ はどのように異なると考えられるか？ 単根と多重根の収束の速度の違いを調査・考察せよ

1-4) C (あるいは C++) 言語を用いてフィボナッチ数列を $n = 95$ まで正しく計算するプログラムを作成せよ。($a_{95} = 31\,940\,434\,634\,990\,099\,905$ となるはずである)

1-5) 非線形連立方程式

$$f(x, y) = x^2 + y^2 - 1 = 0$$
$$g(x, y) = x^2(2 + x) - y^2(2 - x) = 0$$

を多次元のニュートン法を用いて解け。ニュートン法の初期値 (x_0, y_0) によって収束先がどのように変わるか図示せよ。2 次元平面上の各点を、その点を初期点として解いた際の収束先で色分けするとどのような図形になるか？ (この課題に限り、公式を使って 2×2 ヤコビ行列の逆行列を計算してよい)

1-6) 代数方程式の解をすべてもとめる方法について調べよ。Durand-Kerner-Aberth 法を用いて、6 次方程式

$$z^6 - 12z^5 + 63z^4 - 190z^3 + 358z^2 - 400z + 200 = 0$$

の全ての解を求めるプログラムを作成せよ。また、解の収束の様子を観察し、解ごとの収束性を論じよ (解のうちいくつかは複素数となることに注意)

[常微分方程式]

2-1) 空気による摩擦のあるバネの問題を考える。壁にバネが繋がれ、バネの先には質量 m の物体が繋がっている。床との摩擦は考えないものとする。バネの伸びる方向に x 座標を取り、自然長の位

[*1] 誤差と h の関係を両対数プロットにすると見やすい

置を原点とすると、物体の運動方程式は以下のように与えられる。

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -kx - \kappa\frac{\mathrm{d}x}{\mathrm{d}t}$$

ここで、k はバネ定数、κ は摩擦の比例定数とする。Euler 法を使い $x(t)$ を $t = 30$ まで計算せよ。その際、刻み幅 h の大きさを変化させ、解の変わる様子を確認せよ。ただし、$k = 2$、$\kappa = 0.2$、$m = 1$、初期条件は $x(0) = 10$、$x'(0) = 0$ とする。さらに、3 次の Runge-Kutta 法、4 次の Runge-Kutta 法を用いて同様の計算を行い、精度の向上の様子を調べよ

2-2) 古典調和振動子 $H = \frac{1}{2}(p^2 + q^2)$ をオイラー法、4 次の Runge-Kutta 法により解き、横軸を p、縦軸を q とした 2 次元位相空間上の軌道をプロットせよ。また、全エネルギー時間変化の様子を観察せよ。次に、逆オイラー法、リープ・フロッグ法を用いて計算を行い、同様のプロットを行った上で、全エネルギーのゆらぎの刻み幅 h 依存性を調べよ

2-3) Numerov 法とシューティング法を用いて、一次元井戸型ポテンシャル中の粒子のシュレディンガー方程式の固有エネルギーと固有関数の組をいくつか求めよ

2-4) ローレンツ方程式は、カオス的な振る舞いを示す非線形方程式の代表例である

$$\begin{aligned}\dot{x} &= -\sigma x + \sigma y \\ \dot{y} &= -xz + rx - y \\ \dot{z} &= xy - bz\end{aligned}$$

パラメータを $\sigma = 10$, $b = 8/3$, $r = 28$、初期条件を $(x, y, z) = (0, 1, 0)$ として、Runge-Kutta 法を用いてローレンツ方程式を数値的に解き、軌道を 3 次元プロットせよ。また、わずかにずらした初期値 (例: $y = 1 \to 1 + \epsilon$) を考え、もとの軌道からのずれがどのように拡大していくかプロットせよ

2-5) 方程式によっては、刻み幅を小さくしても、なかなか精度が上がらないものがある。一つの例として、“硬い方程式” が知られている。“硬い方程式” とは何か、これを精度良く解くためにはどうすれば良いか調べよ[*2]。また、具体的な問題について計算を行ってみよ

[連立一次方程式]

3-1) `lu_decomp.c` を参考にして、LU 分解を用いて行列の行列式を計算するプログラムを作成せよ。$n \times n$ の Vandermonde 行列 ($v_{ij} = x_j^{i-1}$) ($x_1 \cdots x_n$ は実数) の行列式を計算し、厳密な値 $\displaystyle\prod_{1 \le i < j \le n} (x_j - x_i)$ と比較せよ。ピボット選択で行を入れ替えると、行列式の符号が反転することに注意せよ。なお、この課題では、LU 分解 (LAPACK の `dgetrf` 関数、Python の `scipy.linalg.lu` 関数など) 以外のライブラリ関数は使ってはならない

3-2) LU 分解を用いて Dirichlet 型の境界条件のもとでの二次元 Laplace 方程式の解を求めるプログラムを作成せよ。適当な境界条件 [例えば $u(0, y) = \sin(2\pi y)$, $u(1, y) = \sin(\pi y)$, $u(x, 0) = u(x, 1) = 0$] や電荷分布を仮定して解を計算し、Gnuplot の `splot` コマンド (あるいは Matplotlib の `plot_surface` など) を用いて解をプロットせよ。また、メッシュ数を増やすと、解の形や計算時間がどのように変化するか調べよ (計算時間の測り方については、ハンドブック 1.1.7 節参照)

3-3) 8 行 8 列のランダムな実対称行列、複素エルミート行列、実直交行列、複素ユニタリー行列を生成し、さらに、実際に生成された行列がそれらの性質を満たしているかチェックするプログラムを作成・実行せよ。なお、この課題では、QR 分解や特異値分解などのライブラリ関数は使ってはならない

[*2] 例えば、三井斌友「常微分方程式の数値解法」(岩波オンデマンド) などが参考になる

3-4) 非線形連立方程式

$$f(x,y) = x^2 + y^2 - 1 = 0$$
$$g(x,y) = x^2(2+x) - y^2(2-x) = 0$$

を多次元のニュートン法 (講義 1 スライド p.20) を用いて解け。ヤコビ行列の逆行列をかける代わりに、LU 分解を用いて線形連立方程式を解くこと

3-5) Laplace 方程式の境界値問題を Gauss-Seidel 法、SOR 法で解くプログラムを作成し、計算結果や計算速度を LU 分解・Jacobi 法と比較せよ。また、収束までの回数を Jacobi 法と比較せよ。特に SOR 法の場合、パラメータ ω の選び方により、どのように収束回数が変化するか観察し、最適な ω の値について考察せよ

3-6) 行列・行列積の計算を行うサンプルプログラム `multiply.c` と、BLAS ライブラリの `dgemm` 関数を使ってそれと等価な計算を行う `multiply_dgemm.c` の速度を比較せよ[*3]。行列サイズによっては 100 倍以上もの性能差が出ことがある。BLAS で使われている最適化手法について調べてみよ

[*3] ai では、OS 付属の BLAS、LAPACK ではなく、Intel 製の MKL (Math Kernel Library) に含まれる BLAS や LAPACK を利用するのがよい。MKL を使うには、GNU C コンパイラ (`cc`, `gcc`) の代わりに Intel C コンパイラ (`icx` あるいは `icc`) を使い、`-llapack -lblas` の代わりに`-mkl` を指定してリンクする。例: `icx -O3 multiply_dgemm.c -mkl`