Vorlesungsnotizen zu Numerik für Informatiker

Nicolas Gres

2025-01-01

Inhaltsverzeichnis

Einführung

1 Arithmetik

1.1 Gleitkommazahlen

Wir betrachten für eine gegebene Basis $B \geq 2$, einen minimalen Exponent E^- und Längen M und E die endliche Menge der normalisierten Gleitpunktzahlen FL.

$$\mathrm{FL} := \{\pm B^e \underbrace{\sum_{l=1}^{M} a_l B^{-l}}_{=m} \mid e = E^- + \sum_{k=0}^{E-1} c_k B^k,\ a_l, c_k \in \{0, \dots, B-1\},\ a_1 \neq 0\} \cup \{0\}$$

1.2 Auslöschung

```
N = 2**10

def exp(x):
    """
    Compute the exponential function using Taylor series expansion.
    """
    return np.sum([x**n / math.factorial(n) for n in range(N)], axis=0)

x = 10

z_bad = exp(-x)
z_good = 1 / exp(x)

r = np.exp(-x) # reference

np.abs(z_bad - r) / r, np.abs(z_good - r) / r
```

```
(np.float64(6.529424994681785e-09), np.float64(1.4925713791816933e-16))
```

Quadratische Gleichung

Anstatt $x_2 = p - \sqrt{p^2 - q}$, verwenden wir

$$x_2 = p - \sqrt{p^2 - q} \cdot \frac{p + \sqrt{p^2 + q}}{p + \sqrt{p^2 + q}} = \frac{q}{p + \sqrt{p^2 - q}} = \frac{q}{x_1}$$

(Satz von Vieta) um die Auslöschung zwischen p und $\sqrt{p^2 - q}$ zu vermeiden.

```
p = 1e10
q = 1e2

print(np.roots([1, -2*p, q])) # reference

x1 = p + math.sqrt(p**2 - q)

x2_bad = p - math.sqrt(p**2 - q)
x2_good = q / x1

x2_bad, x2_good
```

```
[2.e+10 5.e-09]
```

```
(0.0, 5e-09)
```

2 Direkte Lösungsverfahren für lineare Gleichungen

2.1 LR-Zerlegung

(en. LU-Decomposition)

https://www.youtube.com/watch?v=BFYFkn-eOQk

```
matrix = np.array([[2.0, 1.0],
[1.0, 4.0]])
rhs = np.array([1.0, 2.0])
solution = solve_with_lu(matrix, rhs)
print("solution",solution)
test = rhs - np.dot(matrix,solution)
print("test ",test)
```

```
solution [0.5 0. ]
test  [0.  1.5]
```

```
def lu_decomposition(matrix):
    n = matrix.shape[0]
    lower = np.zeros(shape=matrix.shape)
    upper = np.zeros(shape=matrix.shape)
    for j in range(n):
        lower[j][j] = 1.0
        for i in range(j + 1):
            first_sum = sum(upper[k][j] * lower[i][k] for k in range(i))
            upper[i][j] = matrix[i][j] - first_sum
        for i in range(j, n):
            second_sum = sum(upper[k][j] * lower[i][k] for k in range(j))
            lower[i][j] = (matrix[i][j] - second_sum) / upper[j][j]
    return lower, upper

def forward_sub(lower, rhs):
    n = lower.shape[0]
    solution = np.zeros(n)
    for i in range(n):
        solution[i] = rhs[i]
        for j in range(i):
            solution[i] = solution[i] - (lower[i, j] * solution[j])
            solution[i] = solution[i] / lower[i, i]
    return solution

def backward_sub(upper, rhs):
    n = upper.shape[0]
    solution = np.zeros(n)
    for i in range(n - 1, -1, -1):
        tmp = rhs[i]
        for j in range(i + 1, n):
            tmp -= upper[i, j] * solution[j]
            solution[i] = tmp / upper[i, i]
    return solution

def solve_with_lu(matrix, rhs):
    lower, upper = lu_decomposition(matrix)
    y = forward_sub(lower, rhs)
    return backward_sub(upper, y)
```

Abbildung 2.1

Appendix

Hinweis

Für alle reellen $q \neq 1$ und für alle $n \in \mathbb{N}_0$ ist:

$$\sum_{k=0}^{n} q^k = \frac{1 - q^{n+1}}{1 - q}$$

$$\sum_{k=0}^{\infty} r^k = \frac{1}{1 - r}$$

Referenzen