

# Numerische Mathematik

Max Mustermann

## 1. Quadratur

### Newton-Cotes Quadratur

Mit äquidistanten Stützstellen: Wir lösen  $\int_a^b f(x)dx = \sum_{i=1}^n w_i f(x_i)$  mit Stützstellen  $f(x_i)$  und Gewichten  $w_i$ . Zudem lohnt es sich mit

```
1 np.linspace(a,b,N)
```

Zwischenpunkte einzubauen, sodass die Quadratur auf Teilintervallen erfolgen kann.

VERFAHREN	ORDNUNG	FORMEL
Mittelpunktsregel	2	$f(\frac{a+b}{2})(b-a)$
Trapezregel	2	$\frac{f(a)+f(b)}{2}(b-a)$
Simpsonregel	4	$\frac{b-a}{6}(f(a) + 4f(\frac{a+b}{2}) + f(b))$

→ **Adaptive Quadratur** Nutze zwei Verfahren verschiedener Ordnung. Falls  $||I_1 - I_2|| < tol$  apzeptiere höhere Ordnung, ansonsten Teile Intervall auf und und beginne auf Teilintervallen von vorne.

### Monte-Carlo Quadratur

In chaotischen Situationen kann diese probabilistische Methode mit garantierter Konvergenz  $\mathcal{O}(n^{-\frac{1}{2}})$  eingesetzt werden dabei gilt:

$$s_i = a + (b - a)t_i$$

wobei  $t_i \in [0, 1]$  Vektoren sind welche zufällig erzeugt wurden.

$$I = \int_a^b z(s)ds = |b - a|\frac{1}{N} \sum_{i=1}^N z(s_i)$$

**Vertrauensintervall:**

$$\widetilde{\sigma}_N = \sqrt{\frac{\frac{1}{N} \sum_{i=1}^N z(t_i)^2 - (\frac{1}{N} \sum_{i=1}^N z(t_i))^2}{N - 1}} = \frac{\sigma_N}{\sqrt{N}}$$

In 68.3% der Fälle liegt der Wert unseres Integrals in

$$[I_N - \widetilde{\sigma}_N, I_N + \widetilde{\sigma}_N]$$

### Gauss-Legendre Quadratur

Wir wollen mit  $n$  Gewichten und Stützstellen Integrale der Ordnung  $2n$  genau berechnen. Mithilfe des GOLUB WELSCH ALGORITHMUS (Skript Seite 167) können wir die Gewichte berechnen. Die Gewichte können wir wie folgt von  $[-1, 1] \rightarrow [a, b]$  umskalieren:

$$\hat{x}_i = \frac{b-a}{2}(x_i + 1) + a \ \& \ \hat{w}_i = \frac{b-a}{w_i}$$

### Mehrdimensionale Quadratur

```
1 def trapezoid2d(f, a, b, Nx, c, d, Ny):
2     """Approximiert das Doppelintegral.
3     """
4     # Definiere F(y) = int_a^b f(x,y) dx
5     F = lambda y: integrate(lambda x: f(x, y), a, b,
6                             Nx)
7     #integriere wie gewohnt.
8     return integrate(F, c, d, Ny)
```

## 2. Ausgleichsrechnung

### Lineare Ausgleichsrechnung

Wir haben hier Probleme bei welchen ein Gleichungssystem  $Ax = b$  überbestimmt ist. In  $A$  haben wir die Punkte an welchen wir Messwerte sammeln, in  $x$  die Variabeln und in  $b$  die Resultate der Messung.

#### Beispiel

Seien  $(2, \sqrt{6})$ ,  $(0, -\sqrt{6})$ ,  $(-1, 2\sqrt{6})$  Datenpunkte, dann können wir beim Modell  $f(x) = \beta_1 x + \beta_2$  Folgendes Ausgleichsproblem kann nun ausgestellt werden:

$$\begin{pmatrix} 2 & 0 \\ 0 & 1 \\ -2 & 1 \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} \sqrt{6} \\ -\sqrt{6} \\ 2\sqrt{6} \end{pmatrix}$$

Wir setzen dabei alle  $\beta_2 = 1$  da es ja genau 1 mal im Modell vorkommt.

### Lösen des Problems

#### 1. Normalengleichung

```
1 B = np.dot(A.T,A) # calc B=A'*A & y=A'*t
2 y = np.dot(A.T,b)
```

```
3 x_ng = np.linalg.solve(B,y) # Löse LGS: Bx=y
```

#### 2. QR-Zerlegung

```
1 Q,R_bar = np.linalg.qr(A)
2 # Bestimme quadratische Matrix R und Vektor c
3 m,n = np.shape(A)
4 R = R_bar[:n,:n]
5 b_tmp = np.dot(Q.T,b)
6 c = b_tmp[:n];
7 x_qr = np.linalg.solve(R,c) # Löse GLS: Rx=Q'b
```

#### 3. (SVD) Singulärwertzerlegung

```
1 U,s,V = np.linalg.svd(A)
2 S = np.zeros_like(A)
3 S[:2, :2] = np.diag(s)
4 invS = np.linalg.pinv(S)
5 x_svd = np.dot(np.dot(np.dot(V,invS),U.T),b)
```

## 3. Polynominterpolation

1. **Monombasis** Wir stellen folgende Bedingungen an die Parameter:

$$\begin{matrix} c_0x_0^0 + c_1x_0^1 \dots c_{m-1}x_0^{m-1} \\ \vdots \\ c_0x_{m-1}^0 + c_1x_{m-1}^1 \dots c_{m-1}x_{m-1}^{m-1} \end{matrix}$$

Dabei werden wir wie folgt vorgehen:

```
1 x = np.array([x_0, ..., x_m-1])
2 y = np.array([y_0, ..., y_m-1])
3 c = np.polyfit(x,y,m) # m ist der Grad
4 coeffs = coeffs[:-1]#da c in Ordnung ..+Ax^2+Bx+C
5 x_new = np.linspace(a,b,c)
6 y_fit = np.polyval(coeffs, x_new)
7 plt.plot(...)
```

Achtung → dieses Verfahren muss bei neuem Stützpunkt komplett neu gerechnet werden.

2. **Lagrange Basis** Wir machen den Ansatz

$$P(x) = \sum_{j=0}^{m-1} y_j L_j(x)$$

. Dabei gilt  $L_j(x) = \prod_{i=0, i \neq j}^{m-1} \frac{x-x_i}{x_j-x_i}$

3. **Newton Basis** Sei die Newton Basis gegeben durch:

$$\begin{aligned} N_0(x) &= 1 \\ N_1(x) &= (x - x_0) \\ N_2(x) &= (x - x_0)(x - x_1) \\ &\vdots \\ N_n(x) &= \prod_{i=0}^{n-1} (x - x_i) \end{aligned}$$

Das Interpolationspolynom ist gegeben durch:

$$P(x) = \sum_{j=0}^{m-1} \alpha_j N_j(x)$$

Dies Lösen wir durch das aufstellen folgender Gleichung  $P(x_i) = \sum_{j=0}^{m-1} \alpha_j N_j(x_i) \stackrel{!}{=} y_i$  Wir können dabei folgendes Gleichungssystem herleiten, welches wir lösen können:

$$\begin{pmatrix} N_0(x_0) & \dots & N_{m-1}(x_0) \\ \vdots & \ddots & \vdots \\ N_0(x_{m-1}) & \dots & N_{m-1}(x_{m-1}) \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_{m-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_{m-1} \end{pmatrix}$$

Diese Gleichungen sind aber ineffizient zu lösen, deshalb benutzt man sogenannte "Dividierte Differenzen". Siehe dafür Seite 114 im Skript.