Angewandte Mathematik
Modellierung und Numerik
Daniel Tenbrinck, Tim Roith

Friedrich-Alexander-Universität
Naturwissenschaftliche Fakultät

Einführung in die Numerik

Programmieraufgaben Blatt 05, Abgabe: 13.01.2023, 12:00
Wintersemester 2022/2023

— AUFGABE 1: Horner-Schema; 1 + 2 + 2 -Punkte —

In dieser Aufgabe beschäftigen wir uns mit der effizienten Auswertung von Polynomen $P \in \Pi_n$ vom Grad n.

(i) Implementieren Sie eine Funktion `simple_polyval(x, p)` welche ein Polynom

$$P(\mathtt{x}) = \mathtt{p}_n \mathtt{x}^n + \mathtt{p}_{n-1} \mathtt{x}^{n-1} + \cdots + \mathtt{p}_1 \mathtt{x} + \mathtt{p}_0 \tag{1}$$

mit Koeffizienten `p` am Punkt `x` in der Monom-Basis auswertet. D.h. der Aufwand für eine Auswertung soll hierbei durch

- n Additionen und
- $\sum_{i=0}^{n} i = 1/2\ (n+1)\ n$ Multiplikationen (jeweils für die Potenzen),

gegeben sein. Beachten Sie, dass per Konventionen der Vektor `p=(p_n...,p_0)` die Ordnung „Koeffizienten zur höchsten Potenz zuerst“ hat.

(ii) Das Horner-Schema ist eine Methode zur effizienten Auswertung von Polynomen. Man benutzt die zu Gleichung (1) algebraische äquivalente Darstellung

$$P(\mathtt{x}) = ((((\mathtt{p}_n \mathtt{x} + \mathtt{p}_{n-1})\ \mathtt{x} + \mathtt{p}_{n-2})\ \mathtt{x} + \dots)\mathtt{x} + \mathtt{p}_1)\mathtt{x} + \mathtt{p}_0$$

und wertet die geklammerten Ausdrücke sukzessive aus. Implementieren Sie eine Funktion `Horner_polyval(x, a)` welche ein Polynom mit Koeffizienten `p` am Punkten `x` mit dem Horner-Schema nach auswertet. Der Aufwand soll hierbei durch

- n Additionen und
- n Multiplikationen,

gegeben sein.

(iii) Vergleichen Sie die Laufzeiten zwischen den Funktionen `simple_polyval`, `Horner_polyval` und der Funktion `numpy.polyval`.

— AUFGABE 2: Polynom-Modelle; 10 + 10 + 10 + 5 -Punkte —

In dieser Aufgabe beschäftigen wir uns mit drei verschiedenen Varianten zur Polynominterpolation und bewerten sie anhand der Effizienz

- des Polynom-Fittings (d.h. Finden des interpolierenden Polynoms),

- der Polynom-Auswertung,
- und der Änderungen bei Hinzunahme von neuen Stützstellen.

(i) Schreiben Sie eine Klasse `Vandermonde_model` welche das Fitting-Polynom in der Monom-Basis, d.h.

$$P(x) = p_n x^n + p_{n-1} x^{n-1} + \cdots + p_1 x + p_0$$

abspeichert. Implementieren Sie dazu die folgenden Funktionen.

- `fit(self, x, y)`: zu gegeben Daten `x=(x_0,...,x_n)`, `y=(y_0,...,y_n)` sollen die Koeffizienten `p=(p_n,..., p_0)` mithilfe der Vandermonde-Matrix bestimmt und in `self.p` abgespeichert werden, s.d. die Interpolationsbedingung $P(\mathtt{x}_i) = \mathtt{y}_i$ für $i = 0, \dots, n$ gilt.
- `__call__(self, x)`: diese Funktion soll das Polynom am Punkt `x` auswerten. Verwenden Sie hierzu das Horner-Schema oder `numpy.polyval`.
- `add_points(self, x, y)`: Diese Funktion soll ein Polynom mit schon vorhanden Koeffizienten `p=(p_n,..., p_0)` zusätzlich auf die neuen Stützstellen `x,y` fitten. Die schon vorhandene Interpolationsbedingung soll hierbei **nicht** zerstört werden.

(ii) Schreiben Sie eine Klasse `Lagrange_model` welche das Fitting-Polynom mithilfe des Lagrange Ansatzes

$$P(x) = \sum_{k=0}^{n} y_k L_k(x)$$

abspeichert. Um die Auswertung in dieser Form effizienter zu machen, benutzen wir die baryzentrische Form. D.h. wir schreiben

$$L_k(x) = \prod_{i \neq k} \frac{x - x_i}{x_k - x_i} = \underbrace{\prod_{i=0}^{n} (x - x_i)}_{=:l(x)} \; \frac{1}{x - x_k} \; \underbrace{\prod_{i \neq k} (x_k - x_i)^{-1}}_{=:w_k}$$

wobei $l(\cdot)$ unabhängig von k ist und w_k das sogenannte baryzentrische Gewicht unabhängig von x ist. Somit sehen wir, dass

$$P(x) = l(x) \; \sum_{k=0}^{n} y_k \; \frac{w_k}{x - x_k}$$

gilt. Weiterhin gilt, da $\sum_{k=0}^n L_k = 1$

$$P(x) = P(x)/1 = P(x) \left/ \sum_{k=0}^{n} L_k \right. = \left(l(x) \sum_{k=0}^{n} y_k \frac{w_k}{x - x_k} \right) \middle/ \left(l(x) \sum_{k=0}^{n} \frac{w_k}{x - x_k} \right)$$
$$= \left(\sum_{k=0}^{n} y_k \frac{w_k}{x - x_k} \right) \middle/ \left(\sum_{k=0}^{n} \frac{w_k}{x - x_k} \right)$$

und somit muss $l(x)$ nicht explizit ausgewertet werden.

- `fit(self, x, y)`: Gegeben Stützstellen `x,y` soll erneut das Polynom gefittet werden. Hier sollen die baryzentrischen Gewichte bestimmt und abgespeichert werden.
- `__call__(self, x)`: diese Funktion soll das Lagrange-Polynom am Punkt `x` in baryzentrischer Form auswerten.
- `add_points(self, x, y)`: Analog zur `add_points` Funktion von `Vandermonde_model`.

(iii) Schreiben Sie eine Klasse `Newton_model` welche das Fitting-Polynom mithilfe des Newton Ansatzes

$$P(x) = \sum_{k=0}^{n} f_{0,\ldots,k} N_k(x)$$

abspeichert.

- `fit(self, x, y)`: Gegeben Stützstellen `x,y` soll erneut das Polynom gefittet werden. Benutzen Sie hierfür das Schema der dividierten Differenzen.
- `__call__(self, x)`: diese Funktion soll das Newton-Polynom am Punkt `x` auswerten. Benutzen Sie hierbei die Idee des Horner-Schemas, s.d. die Auswertung mit linearem, **nicht** mit quadratischem Aufwand ausgeführt wird.
- `add_points(self, x, y)`: Analog zur `add_points` Funktion von `Vandermonde_model`.

(iv) Vergleichen die jeweils die Zeit, welche die drei Modelle für

- Polynom-Fitting an gegebenen Stützstellen,
- Auswertung an beliebigen Punkten,
- Re-Fitting mit zusätzlichen Stützstellen,

benötigen.

— AUFGABE 3: Kubische Splines in 2d; 15 + 5 -Punkte —

In dieser Aufgabe beschäftigen wir uns mit kubischen Spline-Kurven

$$\begin{aligned} S_\Delta : [0,1] &\to \mathbb{R}^2, \\ t &\mapsto S_\Delta(t). \end{aligned}$$

Eine Spline-Kurve hat nun also zwei Komponenten

$$S_\Delta(t) = (S_\Delta^{(1)}(t), S_\Delta^{(2)}(t))$$

wobei wir die Bedingungen auf den eindimensionalen Fall zurückführen können. Für gegebene Stützstellen

$$(t_0, (y_0^{(1)}, y_0^{(2)})), \quad \ldots, \quad (t_n, (y_n^{(1)}, y_n^{(2)}))$$

soll nun für $i = 0, 1$ gelten, dass $S_\Delta^{(i)}$ das kubische Spline zu den Stützstellen $(t_0, y_0^{(i)}), \ldots, (t_n, , y_n^{(i)})$ ist. Wir benutzten im folgenden natürliche Randbedingungen.

(i) Implementieren Sie eine Klasse `cubic_spline_2D` welche eine kubische Spline-Kurve repräsentiert, mit den folgenden Methoden.

- `fit(self,t,y)`: Diese Funktion soll die Koeffizienten des Splines bestimmen und in `self.p` abspeichern, s.d. Sie die Spline-Kurve zu den Stützstellen `t,y` erhalten. Sie können hierbei davon ausgehen, dass `t` aufsteigend sortiert ist.
- `__call__(self,t)`: Diese Funktion soll die Spline-Kurve am Punkt `t` auswerten.

(ii) Testen Sie ihre Funktion anhand der gegeben Daten `HNY.txt` welche Sie mit `numpy.loadtxt` einlesen können. Die Daten speichern nur Koordinaten $y_0, \ldots, y_n$, überlegen Sie sich in wie fern die Werte $t_0, \ldots, t_n$ relevant sind, bzw. wie Sie sie hier wählen können.