Otto-von-Guericke-Universität Magdeburg
Institut für Analysis und Numerik
apl. Prof. F. Schieweck, M.Sc. L. Lautsch

21.05.2024

Programmieraufgaben zur Lehrveranstaltung

"Numerik Computer-Übung für SDA und LA" (SS 2024)

Aufgabe 1 (Newton-Verfahren)

Bei einem chemischen Prozeß kann die Konzentration $c(t)$ einer gefährlichen Schadstoffsubstanz im Reaktor zur Zeit t durch die Formel

$$c(t) = e^{-\alpha t} + \frac{t^2}{t^2 + \beta}$$

beschrieben werden, wobei man für die Parameter die Werte $\alpha = 2$ und $\beta = 3$ annimmt.

a) Mit Hilfe der `Matplotlib.pyplot`-Funktionen `subplot` und `plot` erzeuge man in einer Grafik untereinander die Funktionsgraphen (mit eingezeichneten Gitterlinien) zur Schadstoffkonzentration $c(t)$ und zur zeitlichen Änderungsrate $f(t) := c'(t)$ über dem Zeitintervall $t \in [0, 10]$.

b) Man bestimme eine Näherung für den Zeitpunkt $t^* \in [0, 10]$, an dem die Schadstoffkonzentration $c(t)$ minimal wird, d.h. dass gilt

$$c(t^*) = \min_{t \in [0,10]} c(t).$$

Hierzu löse man die Extremalgleichung $f(t) = c'(t) = 0$ näherungsweise bis auf Maschinengenauigkeit durch Anwendung hinreichend vieler Schritte des klassischen Newton-Verfahrens mit dem Startwert $t_0 = 0$.

c) Man benutze die in b) gefundene numerische Lösung für t^* als Referenzlösung `tref`, mit deren Hilfe nun der Fehler der einzelnen Näherungswerte t_n aus dem obigen Newton-Verfahren ermittelt werden kann. Unter Nutzung des Befehls `print` mit formatierter Ausgabe erzeuge man eine tabellarische Ausgabe, in der pro Zeile ausgegeben werden: der Index n der Iterierten, der Wert der Newton-Iterierten t_n in 16-stelliger Genauigkeit und die Werte $f(t_n)$, $c(t_n)$ sowie $E_n = |t_n - \texttt{tref}|$ jeweils mit 3-stelliger Ausgabe im e-Format.

Aufgabe 2 (Sekanten-Verfahren zur Eigenwert-Berechnung)

Gegeben sei die symmetrische und positiv definite Matrix $A = (a_{i,j}) \in \mathbb{R}^{N \times N}$ durch

$$a_{i,j} := \begin{cases} 2, & \text{falls} \quad j = i, \\ -1, & \text{falls} \quad j = i-1 \quad \text{oder} \quad j = i+1, \\ 0, & \text{sonst.} \end{cases}$$

Der kleinste Eigenwert λ^* von A kann durch die Formel

$$\lambda^* = 2\left(1 - \cos(\frac{\pi}{N+1})\right)$$

charakterisiert werden oder aber als kleinste Nullstelle x^* des charakteristischen Polynoms $f : \mathbb{R} \to \mathbb{R}$ mit

$$f(x) := \det(A - xE),$$

wobei $E \in \mathbb{R}^{N\times N}$ die Einheitsmatrix bezeichnet.

a) In einem Python-Programm erzeuge man für einen einzugebenden Wert von N die Matrix A. Ferner programmiere man die Funktion f als eigene Python-Funktion mit Hilfe der vorhandenen Funktionen `det` und `eye` zur Berechnung der Determinante bzw. der Einheitsmatrix E.

b) Man programmiere das Sekanten-Verfahren zur Berechnung eines Näherungswertes x_n für x^*. Dabei verwende man die Startwerte $x_0 = 0$ und $x_1 = 0.5x^*$. Die Iteration soll nach Berechnung von x_n abgebrochen werden, falls gilt $f(x_n) \leq 10^{-15}$ oder $|x_n - x_{n-1}| < 10^{-15}$.

c) Unter Nutzung des Befehls `print` mit formatierter Ausgabe erzeuge man eine tabellarische Ausgabe, in der pro Zeile ausgegeben werden: der Index n der Iterierten, der Wert der Iterierten x_n in Gleitpunkt-Darstellung mit Mantissenlänge 15 und die Werte $f(x_n)$ sowie $E_n = |x_n - x^*|$ jeweils mit 3-stelliger Ausgabe im e-Format.

Aufgabe 3 (Kondition einer Matrix und Fehlerverhalten)

Gegeben sei die symmetrische Hilbert-Matrix

$$A = (a_{i,j}) \in \mathbb{R}^{N\times N} \qquad \text{mit} \qquad a_{i,j} := \frac{1}{i+j-1}.$$

Für größere Werte von N ist sie ein Beispiel für eine extrem schlecht konditionierte Matrix.

a) In einem Python-Programm erzeuge man eine Tabelle, in der für alle Werte $N \in \{2, 3, \ldots, 10\}$ pro Zeile ausgegeben werden: die Dimension N der Matrix, die Determinante von A, der kleinste und größte Eigenwert von A, sowie die Konditionszahlen von A bezüglich der Maximumnorm und und der Spektralnorm (jeweils mit 3-stelliger Ausgabe im e-Format).

b) Für jeden Wert $N \in \{2, 4, 6, 8, 10\}$ führe man das folgende numerische Experiment durch. Man definiere den Vektor $b \in \mathbb{R}^N$ durch $b_i := \sum_{j=1}^{N} a_{i,j}$ für $i = 1, \ldots, N$. Dann ist die exakte Lösung des Gleichungssystems $Ax = b$ der Vektor $x^* = (1, 1, \ldots, 1)^T \in \mathbb{R}^N$. Mit Hilfe einer geeigneten Python-Funktion berechne man die numerische Lösung des Gleichungssystems $Ax = b$. Danach ermittle man den absoluten und relativen Fehler

$e_{abs} = \|x - x^*\|_\infty$ und $e_{rel} = \|x - x^*\|_\infty / \|x^*\|_\infty$ sowie die oberen Fehlerschranken für e_{abs} und e_{rel} aus den theoretischen Abschätzungen

$$e_{abs} \leq \|A^{-1}\|_\infty \cdot \|r\|_\infty \qquad \text{und} \qquad e_{rel} \leq \mathrm{cond}_\infty(A) \frac{\|r\|_\infty}{\|b\|_\infty},$$

wobei $r := A\tilde{x} - b$ das Residuum von $\tilde{x}$ bezeichnet. In einer Tabelle gebe man für jeden Wert von N pro Zeile aus: den Wert von N, den absoluten Fehler e_{abs} und die zugehörige obere Fehlerschranke sowie den relativen Fehler e_{rel} mit seiner oberen Fehlerschranke. Wie ist der Einfluß der Konditionszahl auf die relative Genauigkeit der numerischen Lösung ?

Zur Lösung der Aufgabe informiere man sich über die Funktionen `eig`, `max`, `min`, `cond` und `norm`.

Aufgabe 4 (Richardson-Verfahren, Parameterwahl)

Zur iterativen Lösung des Gleichungssystems $Ax = b$ mit

$$A = \begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix} \qquad \text{und} \qquad b = \begin{pmatrix} 1 \\ 5 \end{pmatrix}$$

soll das Richardson-Verfahren mit der Iterationsvorschrift

$$x^{(k+1)} = x^{(k)} + \tau\big(b - Ax^{(k)}\big)$$

angewendet und hinsichtlich der Konvergenzgeschwindigkeit für die verschiedenen Parameter-Werte $\tau \in \{1/9,\, 2/9,\, 3/9,\, 4/9\}$ verglichen werden. Der Startvektor soll dabei stets der Nullvektor sein.

a) In einem Python-Programm erzeuge man zu jedem τ-Wert eine Tabelle, in der für alle Werte k des Iterationszählers mit $1 \leq k \leq 30$ pro Zeile ausgegeben werden soll: der Wert von k, der Fehler E_k der Näherung $x^{(k)}$ in der Eulidischen Norm sowie der Fehlerdämpfungskoeffizient q_k definiert als

$$q_k := \frac{\|x^{(k)} - x^*\|_2}{\|x^{(k-1)} - x^*\|_2}.$$

Dabei speichere man zu jedem τ-Wert die Fehler E_k in geeigneten Vektoren zum Zwecke einer späteren grafischen Darstellung.

b) Man berechne zu jedem τ-Wert jeweils den Mittelwert aller q_k mit $k = 1, \ldots, 30$ und ermittle denjenigen τ-Wert mit der höchsten Konvergenzgeschwindigkeit.

c) Man erzeuge eine Grafik, in der zu jedem τ-Wert eine Kurve aus den zugehörigen Fehlerwerten E_k, $k = 1, \ldots, 30$, mit logarithmischer Achsenskalierung dargestellt wird. Die Kurven sollen dabei mit einer Legende erklärt werden.

Aufgabe 5 (Gauß-Seidel- und SOR-Verfahren)

Gegeben sei die symmetrische und positiv definite Matrix $A = (a_{i,j}) \in \mathbb{R}^{N\times N}$ durch

$$a_{i,j} := \begin{cases} 2, & \text{falls} \quad j = i, \\ -1, & \text{falls} \quad j = i-1 \quad \text{oder} \quad j = i+1, \\ 0, & \text{sonst.} \end{cases}$$

a) In einem Python-Programm erzeuge man die Matrix A für den Wert $N = 20$. Ferner erzeuge man durch Anwendung der Funktion `numpy.random.rand` einen zufälligen exakten Lösungsvektor $x^* \in \mathbb{R}^N$ und danach den dazu passenden rechte-Seite-Vektor $b = Ax^*$.

b) Man programmiere jeweils N Schritte des Gauß-Seidel-Verfahrens sowie des SOR-Verfahrens mit $\omega = 1.75$. Dabei speichere man in geeigneten Vektoren (z.B. auf `e_gs(k)` und `e_sor(k)`) den Fehler des Näherungsvektors $x^{(k)}$ zur exakten Lösung x^* gemessen in der Euklidischen Norm für alle $k = 1, \ldots, N$. Als Startvektor $x^{(0)}$ verwende man stets den Nullvektor.

c) Unter Nutzung des Befehls `print` mit formatierter Ausgabe erzeuge man eine tabellarische Ausgabe, in der pro Zeile $k = 1, \ldots, N$ ausgegeben werden: der Index k und die Fehlernormen des Näherungsvektors $x^{(k)}$ jeweils für das Gauß-Seidel-Verfahren sowie für das SOR-Verfahren mit 3-stelliger Ausgabe im e-Format.

d) Mit Hilfe der Funktionen `semilogy` erzeuge man ein Diagramm, in dem für jedes der betrachteten Verfahren eine Kurve erzeugt wird, in der zum Iterationszähler $k = 1, \ldots, N$ auf der x-Achse der Logarithmus zur Basis 10 der Fehlernorm des entsprechenden Näherungsvektors $x^{(k)}$ auf der y-Achse dargestellt wird. Man sorge dafür, dass die Kurven verschiedene Farben haben, und erzeuge mit Hilfe der Funktion `legend` eine Erklärung der Kurven.

Aufgabe 6 (Bisektionsverfahren)

Unter Verwendung des Bisektionsverfahrens bestimme man unter allen Schnittpunkten des Graphen der Funktion $\varphi(x) = \sin(6\pi x)$ mit der Geraden $g(x) = \frac{1}{2}(1 + x)$ im Intervall $x \in [0, 1]$ denjenigen, der die größte x-Koordinate hat. Hierzu stelle man zunächst beide Funktionen mit Hilfe der Funktion `plot` aus dem Grafik-Paket `Matplotlib.pyplot` in einem Koordinatensystem graphisch dar und lasse dann den Nutzer die Intervallgrenzen eines geeigneten Startintervalls $[a_0, b_0]$ für das Bisektionsverfahren eingeben. Mit diesem Startintervall wende man das Bisektionsverfahren auf die Gleichung

$$f(x) := \varphi(x) - g(x) = 0, \qquad x \in [a_0, b_0]$$

zur Bestimmung der x-Koordinate x^* des Schnittpunktes an. Man gebe in einer Tabelle für $k = 0, 1, 2, \ldots$ jeweils zeilenweise an: den Index k, die Näherung x_k, die Intervallgrenzen

a_k, b_k und die zugehörige theoretische obere Schranke δ_k für den Fehler $|x_k - x^*|$. Das Bisektionsverfahren soll abgebrochen werden, sobald der absolute Fehler der Näherung x_k mit Sicherheit kleiner als 10^{-5} ist.

Aufgabe 7 (Newton- und Sekantenverfahren)

Mit Hilfe des Newton-Verfahrens ermittle man mit größtmöglicher Genauigkeit (Maschinengenauigkeit) die Nullstelle x^* der Funktion

$$f(x) = 1 - x^2 + 2x(1 - x^2 + x)\ln x$$

im Intervall $(0, 1)$. Danach benutze man die erhaltene Näherung als exakten Wert für die Nullstelle x^*, um damit absolute Fehler von Näherungswerten berechnen zu können. In einer erneuten Rechnung vergleiche man das Fehlerverhalten des Newton- und Sekanten-Verfahrens. Hierzu gebe man in einer Tabelle zu wachsendem Iterationszähler n in zwei Spalten die Näherungen $x_n^{(N)}$ aus dem Newton-Verfahren mit den zugehörigen absoluten Fehlern an und in zwei weiteren Spalten die Näherungen $x_n^{(S)}$ aus dem Sekanten-Verfahren mit den entsprechenden Fehlern. Für jedes Verfahren gebe man soviele Näherungen an, bis der absolute Fehler zum anfangs berechneten Wert x^* kleiner als 10^{-14} ist. Beide Verfahren sollen denselben Startwert $x_0^{(N)} = x_0^{(S)}$ haben, welcher geeignet zu wählen ist. Der zweite Anfangswert $x_1^{(S)}$ des Sekanten-Verfahrens soll gleich der ersten Näherung $x_1^{(N)}$ des Newton-Verfahrens gewählt werden.

Aufgabe 8 (Jacobi- und Gauß-Seidel-Verfahren)

Man löse das Gleichungssystem

$$\begin{pmatrix} 7 & 1 & -1 & 2 \\ 1 & 8 & 0 & -2 \\ -1 & 0 & 4 & -1 \\ 2 & -2 & -1 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 3 \\ -5 \\ 4 \\ -3 \end{pmatrix}$$

iterativ mit Hilfe des Jacobi-Verfahrens und des Gauß-Seidel-Verfahrens. Als Startvektor verwende man jeweils $x^{(0)} = 0$. Zu jedem Iterationsverfahren erzeuge man eine Tabelle mit folgenden Angaben. Zu wachsender Nummer des Iterationszählers n soll in einer Spalte die Euklidische Norm des Fehlers der Näherungen $x^{(n)}$ zur exakten Lösung $x^* = (1, -1, 1, -1)^T$ angegeben werden und in zwei weiteren Spalten die entsprechenden a-priori und a-posteriori Fehlerschranken, die auf der zugeordneten Norm der Iterationsmatrix basieren. Für die Berechnung der Iterationsmatrix und ihrer Norm verwende man vorhandene Funktionen aus geeigneten Python-Paketen. Die Iteration soll beendet werden, wenn sowohl die a-priori als auch die a-posteriori Fehlerschranke kleiner als 10^{-5} sind.

Aufgabe 9 (Newton-Verfahren für nichtlineare Gleichungssysteme)

Mit Hilfe des Newton-Verfahrens bestimme man Näherungslösungen des Gleichungssystems

$$\begin{aligned} x + y + z &= 3 \\ x^2 + y^2 + z^2 &= 5 \\ e^x + xy - xz &= 1. \end{aligned}$$

Für jede der beiden zu approximierenden exakten Lösungen $(x^*, y^*, z^*) = (0, 1, 2)$ und $(x^*, y^*, z^*) = (0, 2, 1)$ wähle man geeignete Startvektoren (x_0, y_0, z_0). Zur Lösung der in den Newton-Schritten enstehenden linearen Gleichungssysteme verwende man geeignete Funktionen aus Python-Paketen. In einer Tabelle gebe man zu wachsendem Iterationszähler n die Komponenten der Näherung (x_n, y_n, z_n) an, ferner die Norm des entsprechenden Defektvektors sowie die Norm des Fehlervektors zur exakten Lösung (x^*, y^*, z^*). Als Norm ist dabei die Euklidische Norm zu wählen. Das Newton-Verfahren soll abgebrochen werden, wenn die Norm des Fehlers kleiner als 10^{-12} ist. Man vergleiche die Konvergenzeigenschaften der Iterationen für die beiden exakten Lösungen.

Aufgabe 10 (SOR-Verfahren, Parameter-Optimierung)

Man löse das Gleichungssystem

$$\begin{pmatrix} 7 & 3 & -1 & 2 \\ 3 & 8 & 1 & -4 \\ -1 & 1 & 4 & -1 \\ 2 & -4 & -1 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ -3 \\ 1 \end{pmatrix}$$

iterativ mit Hilfe des SOR-Verfahrens:

$$x_i^{(k+1)} = (1 - \omega) x_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right),$$

wobei n die Dimension des Gleichungssystems, $i = 1, \ldots, n$ und $k = 0, 1, 2, \ldots$. Die erste Summe auf der rechten Seite ist hierbei als Null definiert, falls $i = 1$, und die zweite Summe ist Null, falls $i = n$. Man untersuche wie folgt die Konvergenzgeschwindigkeit für die verschiedenen Relaxationsparameter $\omega_m = m \cdot 0.1$ mit $m = 1, \ldots, 19$. Als Startvektor verwende man jeweils $x^{(0)} = 0$. Die Iteration soll beendet werden, sobald die Euklidische Norm des Defektvektors kleiner als 10^{-15} ist. Man erzeuge eine Tabelle, in der zu jedem Parameterwert ω_m folgende Daten zeilenweise ausgegeben werden: die Anzahl der Iterationen bis zum Erreichen der Abbruchbedingung, die Euklidische Norm des zugehörigen Defektvektors sowie des Fehlervektors zwischen der letzten Iterierten und dem exakten Lösungsvektor. In einer Grafik stelle man zu jedem Parameterwert ω_m die Anzahl der Iterationen bis zum Erreichen der Abbruchbedingung dar und lese daraus den besten Parameterwert ab.

Aufgabe 11 (Vergleich von Jacobi- und SOR-Verfahren)

Gegeben sei die symmetrische und positiv definite Matrix $A = (a_{i,j}) \in \mathbb{R}^{N\times N}$ durch

$$a_{i,j} := \begin{cases} 2, & \text{falls} \quad j = i, \\ -1, & \text{falls} \quad j = i-1 \quad \text{oder} \quad j = i+1, \\ 0, & \text{sonst.} \end{cases}$$

(a) In einem Python-Programm erzeuge man die Matrix A für den Wert $N = 20$. Ferner erzeuge man durch Anwendung der Funktion `numpy.random.rand` einen zufälligen exakten Lösungsvektor $x^* \in \mathbb{R}^N$ und danach den dazu passenden rechte-Seite-Vektor $b = Ax^*$.

(b) Man programmiere jeweils N Schritte des SOR- und Jacobi-Verfahrens. Beim SOR-Verfahren verwende man die Relaxationsparameter $\omega \in \{1, 1.3\}$. Für jedes der drei Verfahren speichere man in geeigneten Vektoren (z.B. auf `e_SOR_1_3(k)`) den Fehler des Näherungsvektors $x^{(k)}$ zur exakten Lösung x^* gemessen in der Euklidischen Norm für alle $k = 1, \ldots, N$. Als Startvektor $x^{(0)}$ verwende man stets den Nullvektor.

(c) Unter Nutzung des Befehls `print` mit formatierter Ausgabe erzeuge man eine tabellarische Ausgabe, in der pro Zeile $k = 1, \ldots, N$ ausgegeben werden: der Index k und die Fehlernormen des Näherungsvektors $x^{(k)}$ jeweils für das SOR(1)-, SOR(1.3)- und Jacobi-Verfahren mit 3-stelliger Ausgabe im e-Format.

(d) Mit Hilfe der Funktion `semilogy` erzeuge man ein Diagramm, in dem für jedes der betrachteten Verfahren eine Kurve erzeugt wird, in der zum Iterationszähler $k = 1, \ldots, N$ auf der x-Achse der Logarithmus zur Basis 10 der Fehlernorm des entsprechenden Näherungsvektors $x^{(k)}$ auf der y-Achse dargestellt wird. Man sorge dafür, dass die Kurven verschiedene Farben haben, und erzeuge mit Hilfe der Funktion `legend` eine Erklärung der Kurven.

Aufgabe 12 (Vergleich verschiedener numerischer Integrationsformeln)

Man berechne das Integral

$$I = \int_0^\pi e^x \sin(x)\, dx$$

exakt über die Bestimmung der Stammfunktion sowie näherungsweise mit Hilfe der Trapez-Regel, der Simpson-Regel und der 3/8-Regel, welche jeweils in der summierten Version anzuwenden sind. Dabei soll der Integrationsbereich äquidistant in N gleichgroße Teilintervalle zerlegt werden, d.h.

$$\begin{aligned} T_N &= \textstyle\sum_{j=1}^N \frac{h}{2}\big\{f(x_{j-1}) + f(x_j)\big\} && \text{(Trapez-Regel)} \\ S_N &= \textstyle\sum_{j=1}^N \frac{h}{6}\big\{f(x_{j-1}) + 4f(x_{j-1}+\frac{h}{2}) + f(x_j)\big\} && \text{(Simpson-Regel)} \\ D_N &= \textstyle\sum_{j=1}^N \frac{h}{8}\big\{f(x_{j-1}) + 3f(x_{j-1}+\frac{h}{3}) + 3f(x_{j-1}+\frac{2h}{3}) + f(x_j)\big\} && \text{(3/8-Regel)} \end{aligned}$$

wobei $h := \frac{\pi}{N}$ und $x_j := jh$ für $j = 0, 1, \ldots, N$. Man bestimme die entsprechenden Näherungswerte jeweils für $N = 10, 20, 30, \ldots, 100$. Mit Hilfe der Funktion `plot` aus dem entsprechenden Python-Paket erzeuge man eine Grafik, in der für jedes der drei Verfahren in einer Kurve der Logarithmus des Fehlers über dem Logarithmus der Schrittweite $h = \pi/N$ aufgetragen wird. Welche Ordnung des Fehlers in Bezug auf die Schrittweite h läßt sich dabei für jedes einzelne Verfahren ablesen? In einer weiteren Grafik trage man für jedes Verfahren in einer Kurve die benötigte Rechenzeit über dem Logarithmus des Fehlers auf.

Aufgabe 13 (Laufzeiten in numpy)

Die Effizienz von `numpy`-Funktionen soll für das Beispiel der Funktion `numpy.dot` anhand von Laufzeitvergleichen experimentell untersucht werden. Man schreibe ein Python-Programm, in dem experimentell die Laufzeit der Funktion `numpy.dot` untersucht wird und mit der einer eigenen Funktion zur Matrix-Muliplikation verglichen wird. Dafür gehe man wie folgt vor. Als erstes programmiere man eine Funktion `MyMatrixMultiplication`, die das Produkt $C = (c_{i,j})$ von zwei Matrizen $A = (a_{i,j}) \in \mathbb{R}^{k \times k}$ und $B = (b_{i,j}) \in \mathbb{R}^{k \times k}$ zurückgibt entsprechend der Definition

$$c_{i,j} = \sum_{m=1}^{k} a_{i,m} b_{m,j}.$$

Mit der Funktion `numpy.random.rand` erstelle man für $k = 20, 40, \ldots, 1000$ jeweils zwei zufällige $k \times k$-Matrizen A und B, die miteinander multipliziert werden einerseits mit `np.dot` und andererseits mit der eigenen Funktion `MyMatrixMultiplication`, wobei jeweils die Laufzeiten gemessen werden. Man wiederhole für jedes k die Rechnung drei mal und speichere am Ende zu jedem k jeweils die entsprechende durchschnittliche Laufzeit. Danach gebe man in einer Grafik in Abhängigkeit von k auf der x-Achse zu beiden Varianten `np.dot` und `MyMatrixMultiplication` jeweils die durchschnittliche Laufzeit in Sekunden auf der y-Achse an. Zur Einordnung des Wachstumsverhaltens der Laufzeiten stelle man auch die Kurven k^2 und k^3 in der Grafik dar. Welche Komplexität (asymptotisches Wachstumsverhalten) hat zu jeder der beiden Varianten die experimentelle Laufzeit für große bzw. kleine k? Wie lässt sich dieses Verhalten erklären?