Iterative Lösung dünnbesetzter Gleichungssysteme

Stefan Lang

Interdisziplinäres Zentrum für Wissenschaftliches Rechnen
Universität Heidelberg
INF 368, Raum 532
D-69120 Heidelberg
phone: 06221/54-8264
email: Stefan.Lang@iwr.uni-heidelberg.de

WS 12/13

Themen

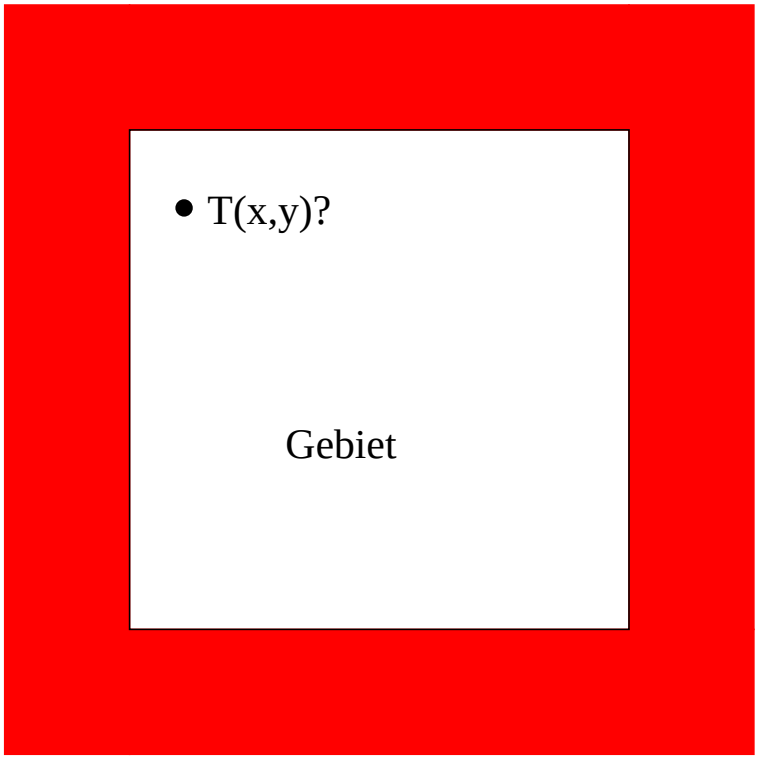
Iteratives Lösen dünnbesetzter linearer Gleichungssystemen

- Problemstellung
- Iterationsverfahren
- Parallelisierung
- Mehrgitterverfahren
- paralleles Mehrgitterverfahren

Problemstellung: Beispiel

Ein kontinuierliches Problem und seine Diskretisierung:

- Beispiel: Eine dünne, quadratische Metallplatte sei an allen Seiten eingespannt.
- Die zeitlich konstante Temperaturverteilung am Rand der Metallplatte sei bekannt.
- Welche Temperatur herrscht an jedem inneren Punkt der Metallplatte, wenn sich ein stationärer Zustand eingestellt hat?



Problemstellung: kontinuierlich

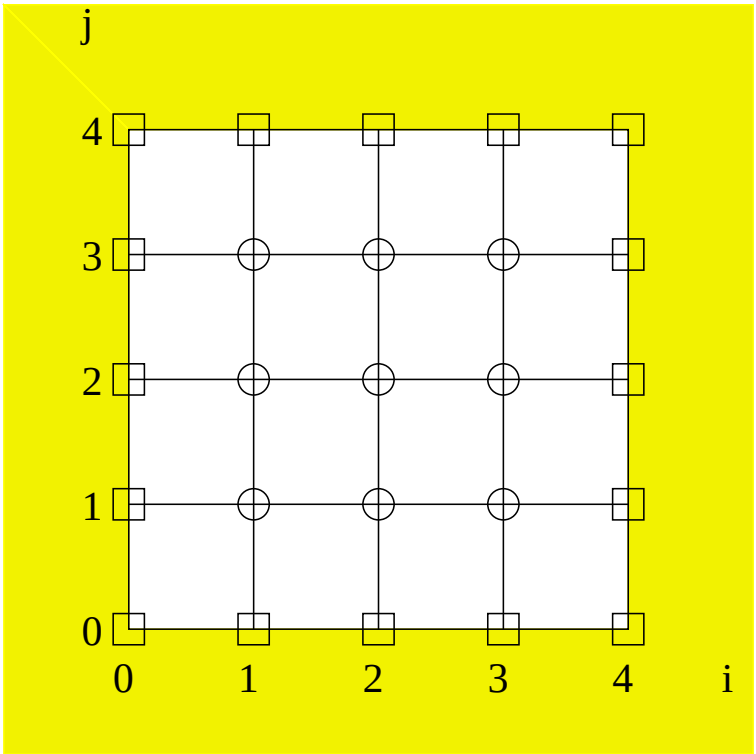
- Dieser Wärmeleitungsvorgang kann mit einem mathematischen Modell (näherungsweise) beschrieben werden.
- Die Metallplatte wird durch ein Gebiet $\Omega \subset \mathcal{R}^2$ beschrieben.
- Gesucht ist die Temperaturverteilung $T(x,y)$ für alle $(x,y) \in \Omega$.
- Die Temperatur $T(x,y)$ für (x,y) auf dem Rand $\partial\Omega$ sei bekannt.
- Ist die Metallplatte homogen (überall gleiche Wärmeleitfähigkeit), so wird die Temperatur im Inneren durch die partielle Differentialgleichung

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0\,, \quad T(x,y) = g(x,y) \text{ auf } \partial\Omega \tag{1}$$

beschrieben.

Problemstellung: diskret

- Im Rechner kann man die Temperatur nicht an jedem Punkt $(x, y) \in \Omega$ (überabzählbar viele) bestimmen, sondern nur an einigen ausgewählten.
- Dazu sei speziell $\Omega = [0, 1]^2$ gewählt.
- Mittels dem Parameter $h = 1/N$, für ein $N \in \mathbf{N}$, wählen wir speziell die Punkte $(x_i, y_j) = (ih, jh)$, für alle $0 \leq i, j \leq N$.
- Man bezeichnet diese Menge von Punkten auch als regelmässiges, äquidistantes Gitter



Die Punkte am Rand wurden dabei mit anderen Symbolen (Quadrate) gekennzeichnet als die im Inneren (Kreise).

Diskretisierung I

Wie bestimmt man nun die Temperatur T_{ij} am Punkt (x_i, y_j)?

- Eine gängige Methode, das Verfahren der „Finiten Differenzen“
- Idee: Die Temperatur am Punkt (x_i, y_j) durch die Werte an den vier Nachbarpunkten ausgedrückt wird:

$$T_{i,j} = \frac{T_{i-1,j} + T_{i+1,j} + T_{i,j-1} + T_{i,j+1}}{4} \tag{2}$$

$$\iff \quad T_{i-1,j} + T_{i+1,j} - 4T_{i,j} + T_{i,j-1} + T_{i,j+1} = 0 \tag{3}$$

für $1 \leq i, j \leq N - 1$.

- An der Form (3) erkennt man, dass alle $(N-1)^2$ Gleichungen für $1 \leq i, j \leq N - 1$ zusammen ein lineares Gleichungssystem ergeben:

$$AT = b \tag{4}$$

Diskretisierung II

- Dabei entspricht $G(A)$ genau dem oben gezeichneten Gitter, wenn man die Randpunkte (Quadrate) weglässt. Die rechte Seite b von (3) ist nicht etwa Null, sondern enthält die Temperaturwerte am Rand!
- Die so berechneten Temperaturwerte $T_{i,j}$ an den Punkten (x_i, y_j) sind *nicht* identisch mit der Lösung $T(x_i, y_i)$ der partiellen Differentialgleichung (1). Vielmehr gilt

$$|T_{i,j} - T(x_i, y_i)| \leq O(h^2) \tag{5}$$

- Diesen Fehler bezeichnet man als „Diskretisierungsfehler“. Eine Vergrößerung von N entspricht somit einer genaueren Temperaturberechnung.

Iterationsverfahren I

- Wir wollen nun das Gleichungssystem (4) „iterativ“ lösen. Dazu geben wir uns einen beliebigen Wert der Temperatur $T^0_{i,j}$ an jedem Punkt $1 \leq i,j \leq N-1$ vor (die Temperatur am Rand ist ja bekannt).
- Ausgehend von dieser Näherungslösung wollen wir nun eine verbesserte Lösung berechnen. Dazu benutzen wir die Vorschrift (2) und setzen

$$T^{n+1}_{i,j} = \frac{T^n_{i-1,j} + T^n_{i+1,j} + T^n_{i,j-1} + T^n_{i,j+1}}{4} \quad \text{für alle } 1 \leq i,j \leq N-1. \qquad (6)$$

- Offensichtlich können die verbesserten Werte $T^{n+1}_{i,j}$ für alle Indizes (i,j) gleichzeitig berechnet werden, da sie nur von den alten Werten $T^n_{i,j}$ abhängen.

Iterationsverfahren II

- Man kann tatsächlich zeigen, dass

$$\lim_{n\to\infty} T_{i,j}^n = T_{i,j} \tag{7}$$

  gilt.
- Den Fehler $|T_{i,j}^n - T_{i,j}|$ in der n-ten Näherungslösung bezeichnet man als „Iterationsfehler“.
- Wie groß ist nun dieser Iterationsfehler? Man benötigt ja ein Kriterium bis zu welchem n man rechnen muss.

Iterationsverfahren III

- Dazu betrachtet man, wie gut die Werte $T_{i,j}^n$ die Gleichung (3) erfüllen, d.h. wir setzen
$$E^n = \max_{1 \leq i,j \leq N-1} \left| T_{i-1,j}^n + T_{i+1,j}^n - 4T_{i,j}^n + T_{i,j-1}^n + T_{i,j+1}^n \right|$$
- Üblicherweise verwendet man diesen Fehler nur relativ, d.h. man iteriert so lange bis
$$E^n < \epsilon E^0$$
gilt, also der Anfangsfehler E^0 um den Reduktionsfaktor ϵ reduziert wurde.
- Dies führt uns zu dem sequentiellen Verfahren:

wähle N, ϵ;
wähle $T_{i,j}^0$;
$E^0 = \max_{1 \leq i,j \leq N-1} \left| T_{i-1,j}^0 + T_{i+1,j}^0 - 4T_{i,j}^0 + T_{i,j-1}^0 + T_{i,j+1}^0 \right|$;
$n = 0$;
while $(E^n \geq \epsilon E^0)$
{
  for $(1 \leq i,j \leq N-1)$
    $T_{i,j}^{n+1} = \frac{T_{i-1,j}^n + T_{i+1,j}^n + T_{i,j-1}^n + T_{i,j+1}^n}{4}$;
  $E^{n+1} = \max_{1 \leq i,j \leq N-1} \left| T_{i-1,j}^{n+1} + T_{i+1,j}^{n+1} - 4T_{i,j}^{n+1} + T_{i,j-1}^{n+1} + T_{i,j+1}^{n+1} \right|$;
  $n = n + 1$;
}

Iterationsverfahren IV

- Kompakter können wir dies alles mit Vektoren schreiben. Zu lösen ist das Gleichungssystem (4), also $AT = b$. Die Näherungswerte $T_{i,j}^n$ entsprechen jeweils Vektoren T^n.
- Formal berechnet sich T^{n+1} als

$$T^{n+1} = T^n + D^{-1}(b - AT^n)$$

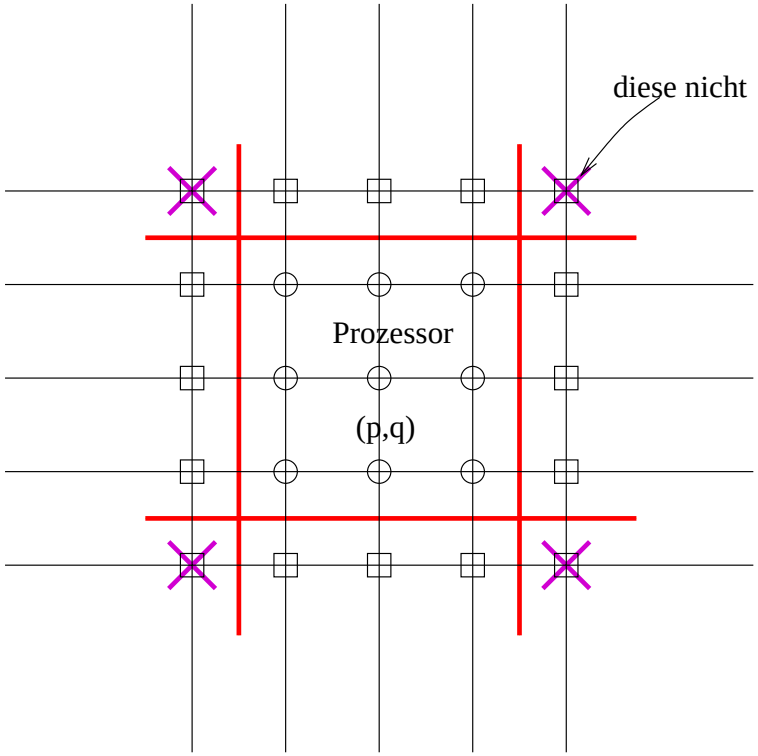
  mit der Diagonalmatrix $D = diag(A)$. Dies bezeichnet man als Jacobiverfahren.
- Der Fehler E^n ergibt sich aus

$$E^n = \|b - A \cdot T^n\|_\infty ,$$

  wobei $\|\cdot\|_\infty$ die Maximumnorm eines Vektors ist.

Parallelisierung I

- Der Algorithmus erlaubt wieder eine datenparallele Formulierung.
- Die $(N+1)^2$ Gitterpunkte werden dazu durch Partitionierung der Indexmenge $I = \{0, \ldots, N\}$ auf ein $\sqrt{P} \times \sqrt{P}$-Prozessorfeld aufgeteilt.
- Die Partitionierung geschieht dabei *blockweise*:



Parallelisierung II

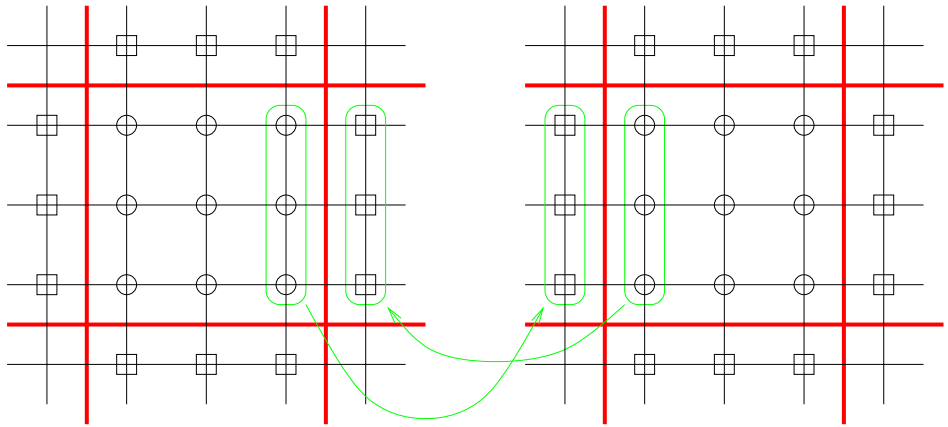
- Prozessor (p,q) berechnet somit die Werte $T_{i,j}^{n+1}$ mit $(i,j) \in \{start(p),\ldots,end(p)\} \times \{start(q),\ldots,end(q)\}$.
- Um dies zu tun, benötigt er jedoch auch Werte $T_{i,j}^{n}$ aus den Nachbarprozessoren mit $(i,j) \in \{start(p)-1,\ldots,end(p)+1\} \times \{start(q)-1,\ldots,end(q)+1\}$.
- Dies sind die Knoten, die in der Abbildung oben mit Quadraten gekennzeichnet sind!
- Jeder Prozessor speichert also zusätzlich zu den ihm zugewiesenen Gitterpunkten noch eine zusätzliche Schicht von Gitterpunkten.

Parallelisierung III

- Der parallele Algorithmus besteht somit aus folgenden Schritten:

  Startwerte $T_{i,j}^0$ sind allen Prozessoren bekannt.
  while $(E^n > \epsilon E^0)$
  {
  berechne $T_{i,j}^{n+1}$ für $(i,j) \in \{start(p), \ldots, end(p)\} \times \{start(q), \ldots, end(q)\}$;
  tausche Randwerte (Quadrate) mit Nachbarn aus;
  Berechne E^{n+1} auf $(i,j) \in \{start(p), \ldots, end(p)\} \times \{start(q), \ldots, end(q)\}$;
  Bilde globales Maximum;
  $n = n + 1$;
  }

- Im Austauschschritt tauschen zwei benachbarte Prozessoren Werte aus:



Parallelisierung IV

- Für diesen Austauschschritt verwendet man entweder asynchrone Kommunikation oder synchrone Kommunikation mit Färbung.
- Wir berechnen die Skalierbarkeit *einer* Iteration:

$$W = T_S(N) = N^2 t_{op} \quad \Longrightarrow N = \sqrt{\frac{W}{t_{op}}}$$

$$T_P(N,P) = \underbrace{\left(\frac{N}{\sqrt{P}}\right)^2 t_{op}}_{\text{Berechnung}} + \underbrace{\left(t_s + t_h + t_w \frac{N}{\sqrt{P}}\right) 4}_{\text{Randaustausch}} + \underbrace{(t_s + t_h + t_w)\,\mathrm{ld}\,P}_{\substack{\text{globale Komm.: Max.} \\ \text{für } E^n}}$$

$$T_P(W,P) = \frac{W}{P} + \frac{\sqrt{W}}{\sqrt{P}} \frac{4t_w}{\sqrt{t_{op}}} + (t_s + t_h + t_w)\,\mathrm{ld}\,P + 4(t_s + t_h)$$

$$T_O(W,P) = PT_P - W =$$

$$= \sqrt{W}\sqrt{P} \frac{4t_w}{\sqrt{t_{op}}} + P\,\mathrm{ld}\,P(t_s + t_h + t_w) + P4(t_s + t_h)$$

Parallelisierung V

- Asymptotisch erhalten wir die Isoeffizienzfunktion $W = O(P \operatorname{ld} P)$ aus dem zweiten Term, obwohl der erste Term für praktische Werte von N dominant sein wird. Der Algorithmus ist nahezu optimal skalierbar.
- Aufgrund der blockweisen Aufteilung hat man einen Oberfläche-zu-Volumen Effekt: $\frac{N}{\sqrt{P}} / \left(\frac{N}{\sqrt{P}}\right)^2 = \frac{\sqrt{P}}{N}$. In drei Raumdimensionen erhält man $\left(\frac{N}{P^{1/3}}\right)^2 / \left(\frac{N}{P^{1/3}}\right)^3 = \frac{P^{1/3}}{N}$.
- Für gleiches N und P ist also die Effizienz etwas schlechter als in zwei Dimensionen.

Mehrgitterverfahren I

- Fragen wir uns nach der Gesamteffizienz eines Verfahrens, so ist die Zahl der Operationen entscheidend.
- Dabei ist

$$T_S(N) = IT(N) \cdot T_{IT}(N)$$

- Wieviele Iterationen nun tatsächlich auszuführen sind, hängt neben N natürlich von dem benutzen Verfahren ab.
- Dazu erhält man die folgenden Klassifikationen:

$$\begin{aligned}
\text{Jacobi, Gauß-Seidel}: &\quad IT(N) = \mathcal{O}(N^2) \\
\text{SOR mit } \omega_{\text{opt}}: &\quad IT(N) = \mathcal{O}(N) \\
\text{konjugierte Gradienten (CG)}: &\quad IT(N) = \mathcal{O}(N) \\
\text{hierarchische Basis d=2}: &\quad IT(N) = \mathcal{O}(\log N) \\
\text{Mehrgitterverfahren}: &\quad IT(N) = \mathcal{O}(1)
\end{aligned}$$

- Die Zeit für eine Iteration $T_{IT}(N)$ ist dabei für alle Verfahren in $\mathcal{O}(N^d)$ mit vergleichbarer Konstante (liegt im Bereich von 1 bis 5).

Mehrgitterverfahren II

- Wir sehen also, daß z.B. das Mehrgitterverfahren sehr viel schneller ist, als das Jacobi-Verfahren.
- Auch die Parallelisierung des Jacobi-Verfahrens hilft da nicht, denn es gelten:

$$T_{P,\text{Jacobi}}(N,P) = \frac{\mathcal{O}(N^{d+2})}{P} \quad \text{und} \quad T_{S,\text{MG}}(N) = \mathcal{O}(N^d)$$

- Eine Verdopplung von N hat also eine Vervierfachung des Aufwandes des parallelisierten Jacobi- Verfahrens im Vergleich zum `sequentiellen` Mehrgitterverfahren zur Folge!
- Das führt uns auf ein grundsätzliches Gebot der parallelen Programmierung:

> Parallelisiere den besten sequentiellen Algorithmus, wenn irgend möglich!

Mehrgitterverfahren III

- Betrachten wir noch einmal die Diskretisierung der Laplacegleichung $\Delta T = 0$.
- Diese ergibt das lineare Gleichungssystem

$$Ax = b$$

- Dabei wird der Vektor b durch die Dirichlet-Randwerte bestimmt. Sei nun eine Näherung der Lösung durch x^i gegeben. Setze dazu den Iterationsfehler

$$e^i = x - x^i$$

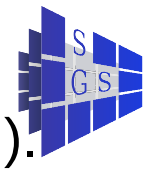
- Aufgrund der Linearität von A können wir folgern:

$$Ae^i = \underbrace{Ax}_{b} - Ax^i = b - Ax^i =: d^i$$

- Dabei nennen wir d^i den `Defekt` .
- Eine gute Näherung für e^i berechnet man durch das Lösen von

$$Mv^i = d^i \quad \text{also} \quad v^i = M^{-1}d^i$$

- Dabei sei M leichter zu lösen, als A (in $\mathcal{O}$(N) Schritten, wenn $x \in \mathbb{R}^N$).



Mehrgitterverfahren IV

- Für spezielle M bekommen wir bereits bekannte Iterationsverfahren :

$$\begin{aligned} M &= I && \rightarrow \text{Richardson} \\ M &= \text{diag}(A) && \rightarrow \text{Jacobi} \\ M &= L(A) && \rightarrow \text{Gauß-Seidel} \end{aligned}$$

- Wir erhalten lineare Iterationsverfahren der Form

$$x^{i+1} = x^i + \omega M^{-1}(b - Ax^i)$$

- Dabei stellt das $\omega \in [0, 1]$ einen Dämpfungsfaktor dar.
- Für den Fehler $e^{i+1} = x - x^{i+1}$ gilt:

$$e^{i+1} = (I - \omega M^{-1}A)e^i$$

- Dabei bezeichnen wir die Iterationsmatrix $I - \omega M^{-1}A$ mit S.
- Es liegt genau dann Konvergenz ($\lim_{i\rightarrow\infty} e^i = 0$) vor , wenn der betragsgrößte Eigenwert von S echt(!) kleiner als Eins ist.

Glättungseigenschaft I

- Ist die Matrix A symmetrisch und positiv definit, so hat sie nur reelle, positive Eigenwerte λ_k zu Eigenvektoren z_k.
- Die Richardson-Iteration

$$x^{i+1} = x^i + \omega(b - Ax^i)$$

führt wegen M = I auf die Fehlerfortpfanzung

$$e^{i+1} = (I - \omega A)e^i$$

- Nun setzen wir den Dämpfungsfaktor $\omega = \frac{1}{\lambda_{\max}}$ und betrachten $e^i = z_k (\forall k)$.
- Dann erhalten wir

$$e^{i+1} = \left(I - \frac{1}{\lambda_{\max}} A\right) z_k = z_k - \frac{\lambda_k}{\lambda_{\max}} z_k = \left(1 - \frac{\lambda_k}{\lambda_{\max}}\right) e^i$$

$$\left(1 - \frac{\lambda_k}{\lambda_{\max}}\right) = \begin{cases} 0 & \lambda_k = \lambda_{\max} \\ \approx 1 & \lambda_k \text{ klein } (\lambda_{\min}) \end{cases}$$

Glättungseigenschaft II

- Im Fall kleiner Eigenwerte haben wir also eine schlechte Dämpfung des Fehlers (sie liegt in der Größenordnung von 1 - $\mathcal{O}$ (h^2)).
- Dieses Verhalten ist für die Jacobi und Gauß-Seidel Iterationsverfahren qualitativ identisch.
- Zu kleinen Eigenwerten gehören aber langwellige Eigenfunktionen.
- Diese langwelligen Fehler werden also nur sehr schlecht gedämpft.
- Anschaulich betrachtet bieten die Iterationsverfahren nur eine lokale Glättung des Fehlers, auf dem sie arbeiten, denn sie ermitteln neue Iterationswerte ja nur aus Werten in einer lokalen Nachbarschaft.
- Schnelle Oszillationen können werden schnell herausgeglättet, während langwellige Fehler die lokalen Glättungen weitgehend unverändert überstehen.

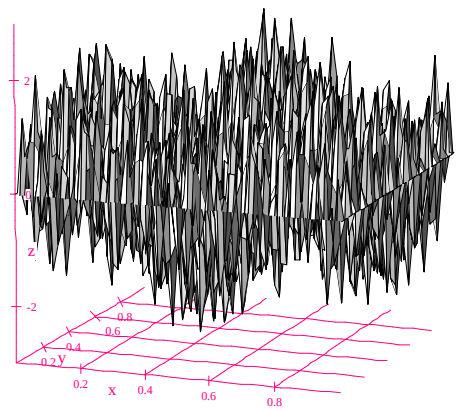
Glättungseigenschaft III

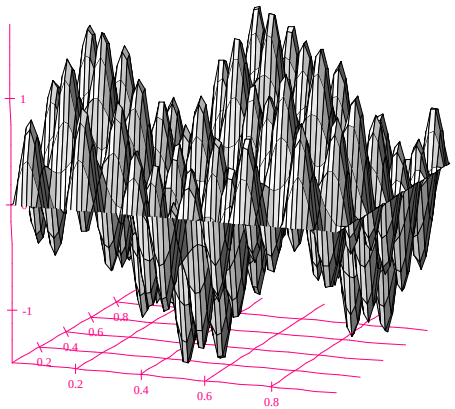
Zur Veranschaulichung betrachen wir folgendes Beispiel:

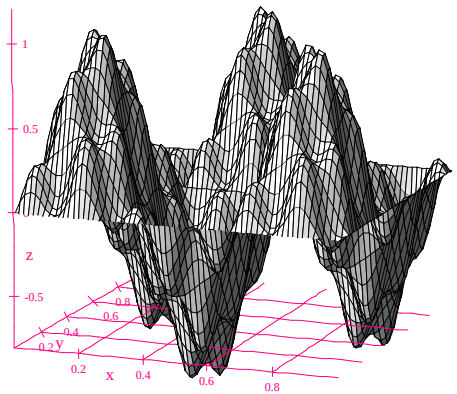
Die Laplacegleichung $-\Delta u = f$ *wird mittels eines Fünf-Punkte-Sterns auf einem strukturierten Gitter diskretisiert. Die zugehörigen Eigenfunktionen sind* $\sin(\nu\pi x)\sin(\mu\pi y)$, *wobei* $1 \leq \nu$ *und* $\mu \leq h^{-1} - 1$ *gelten. Wir setzen* $h = \frac{1}{32}$ *und den Initialisierungsfehler auf*

$$e^0 = \sin(3\pi x)\sin(2\pi y) + \sin(12\pi x)\sin(12\pi y) + \sin(31\pi x)\sin(31\pi y).$$

Mit $\omega = \frac{1}{\lambda_{max}}$ *erhält man die Dämpfungfaktoren (pro Iteration) für die Richardson Iteration durch 0.984, 0.691 und 0 für die einzelnen Eigenfunktionen. Die untenstehenden Graphen zeigen den Initialisierungsfehler und den Fehler nach einer bzw. fünf Iterationen.*

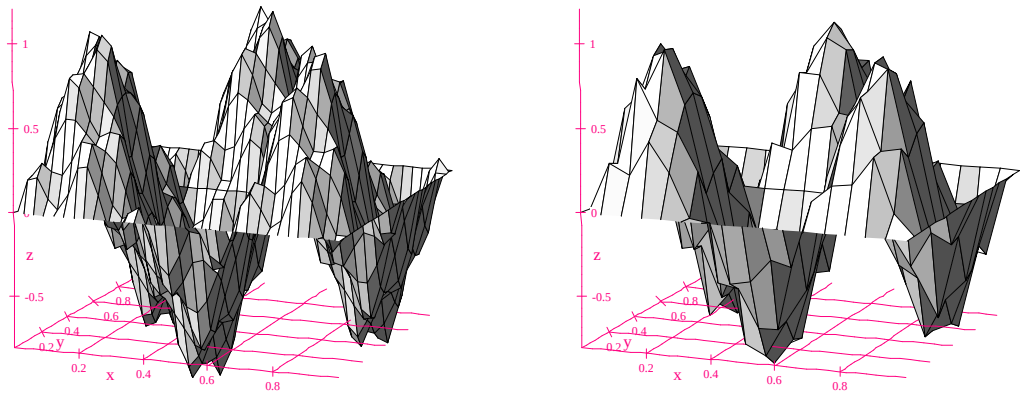






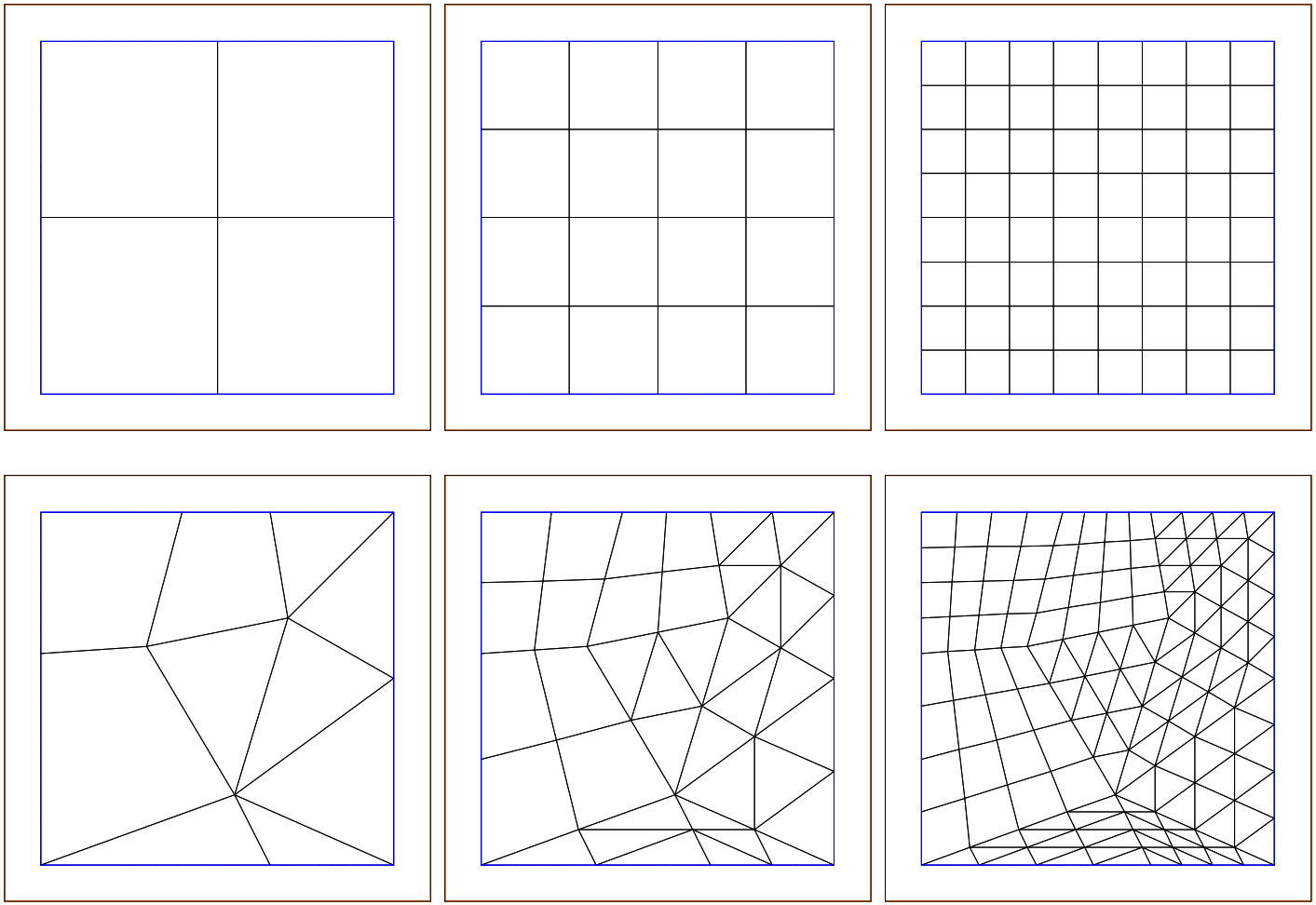
Glättungseigenschaft IV

- Daher kommt man auf die Idee, diese langwelligen Fehler auf gröberen Gittern darzustellen, nachdem die schnellen Ozillationen herausgeglättet worden sind.
- Auf diesen gröberen Gittern ist dann der Aufwand, um die Fehlerkurve zu glätten, geringer.
- Weil die Kurve nach der Vorglättung einigermaßen glatt ist, ist diese Restriktion auf weniger Gitterpunkte gut möglich.



Gitterhierarchie I

- Man erstellt also eine ganze Folge von Gittern verschiedener Feinheit.
- Glättet zunächst auf dem feinsten die kurzwelligen Fehlerfunktionen heraus.
- Geht dann auf das nächst gröbere Gitter über usw. .



Gitterhierarchie II

- Entsprechend erhält man eine ganze Folge linearer Systeme

$$A_l x_l = b_l,$$

  da ja die Anzahl der Gitterpunkte N und damit die Länge von x auf gröberen Gittern schwindet.
- Natürlich will man nach dieser Restriktion wieder auf das ursprüngliche feine Gitter zurückkehren.
- Dazu führt man eine Grobgitterkorrektur durch.
- Nehmen wir dazu an, wir seinen auf Gitterstufe l, betrachten also das LGS

$$A_l x_l = b_l$$

- Auf dieser Stufe ist eine Iterierte x_l^i mit Fehler e_l^i gegeben, also die Fehlergleichung

$$A e_l^i = b_l - A_l x_l^i$$

  Nehmen wir an, x_l^i ist Ergebnis von ν_1 Jacobi, Gauß-Seidel oder ähnlichen Iterationen.
- Dann ist e_l^i verhältnismäßig glatt und somit auch gut auf einem gröberen Gitter darstellbar, das heißt, er kann von einem gröberen Gitter gut auf das feinere interpoliert werden.

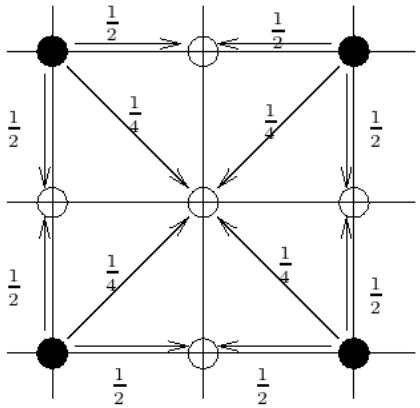
Gitterhierarchie III

- Sei dazu v_{l-1} der Fehler auf dem gröberen Gitter.
- Dann ist in guter Näherung

$$e_l^i \approx P_l v_{l-1}$$

- Dabei ist P_l eine Interpolationsmatrix (Prolongation), die eine linerare Interpolation ausführt und so aus dem Grobgittervektor einen Feingittervektor macht.

1	0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	0	0
0	1	0	0
$\frac{1}{2}$	0	$\frac{1}{2}$	0
$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
0	$\frac{1}{2}$	0	$\frac{1}{2}$
0	0	1	0
0	0	$\frac{1}{2}$	$\frac{1}{2}$
0	0	0	1



Zweigitter- und Mehrgitterverfahren I

- Durch Kombination der obigen Gleichungen erhält man die Gleichung für v_{l-1} durch

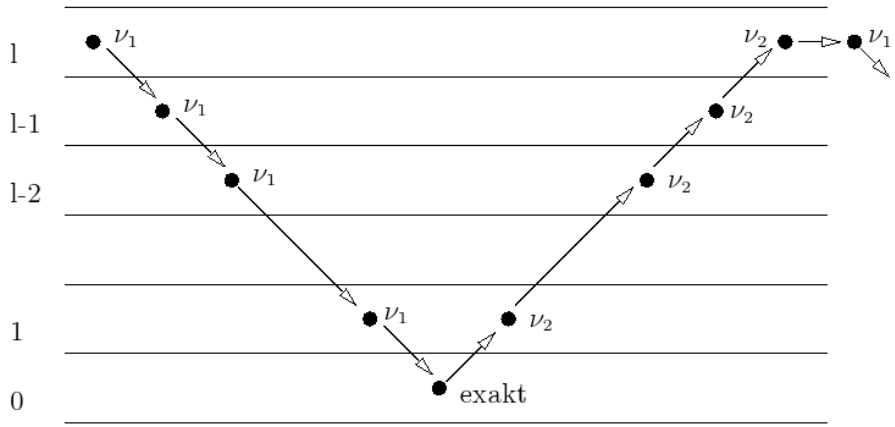
$$R_l A P_l v_{l-1} = R_l (b_l - A_l x_l^i)$$

- Dabei ist $R_l A P_l =: A_{l-1} \in \mathbb{R}^{N_{l-1} \times N_{l-1}}$ und R_l ist die Restriktionsmatrix, für die man z.B. $R_l = P_l^T$ nimmt.
- Ein sogenanntes Zweigitterverfahren besteht nun aus den beiden Schritten:
  1. ν_1 Jacobi-Iterationen (auf Stufe l)
  2. Grobgitterkorrektur $x_l^{i+1} = x_l^i + P_l A_{l-1}^{-1} R_l (b_l - A_l x_l^i)$
- Die rekursive Anwendung führt auf die Mehrgitterverfahren.

mgc(l, x_l, b_l)
{
  if (l == 0) $x_0 = A_0^{-1} b_0$;
  else {
    ν_1 Iterationen Eingitterverfahren auf $A_l x_l = b_l$; //Vorglättung
    $d_{l-1} = R_l (b_l - A_l x_l)$;
    $v_{l-1} = 0$;
    for ($g = 1, \ldots, \gamma$)
      mgc($l - 1, v_{l-1}, d_{l-1}$);
    $x_l = x_l + P_l v_{l-1}$;
    ν_2 Iterationen Eingitterverfahren auf $A_l x_l = b_l$; //Nachglättung
  }
}

Zweigitter- und Mehrgitterverfahren II

- Es ist ausreichend, $\gamma = 1, \nu_1 = 1, \nu_2 = 0$ zu setzen, damit die Iterationszahl $\mathcal{O}(1)$ ist.
- Der einmalige Durchlauf von Stufe l bis Stufe 0 und zurück heißt V-Zyklus: bezeichnet.



- Aufwand für zweidimensionales strukturiertes Gitter: N ist Anzahl der Gitterpunkte in einer Zeile auf der feinsten Stufe, und $C := t_{\text{op}}$:

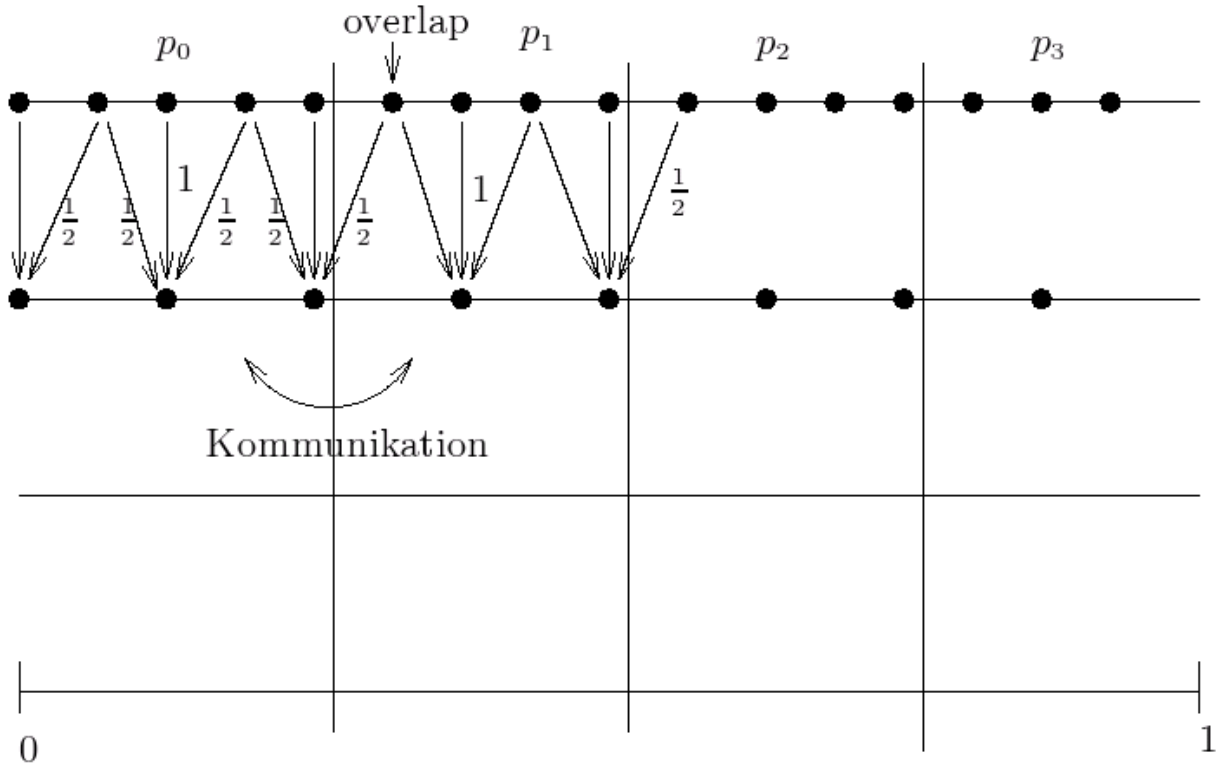
$$T_{\text{IT}}(N) = \underbrace{CN^2}_{\text{Stufe l}} + \underbrace{\frac{CN^2}{4}}_{\text{Stufe l-1}} + \frac{CN^2}{16} + \ldots + \underbrace{G(N_0)}_{\text{Grobgitter}}$$

$$= CN^2 \underbrace{\left(1 + \frac{1}{4} + \frac{1}{16} + \ldots\right)}_{\frac{4}{3}} + G(N_0) = \frac{4}{3} CN_l + G(N_0)$$

Parallelisierung I

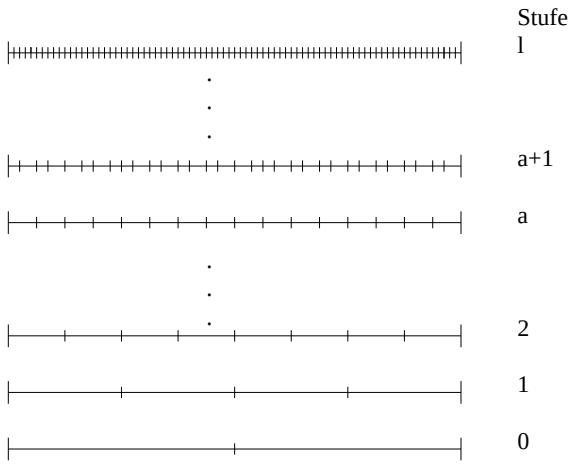
Zur Datenverteilung in der Gitterhierarchie auf die einzelnen Prozessoren ist zu berücksichtigen:

- In der Grobgitterkorrektur muß geprüft werden, ob zur Berechnung der Knotenwerte im Grobgitter Kommunikation erforderlich ist.
- Wie verfährt man mit den gröbsten Gittern, in denen pro Dimensionsrichtung die Anzahl der Unbekannten kleiner als die Anzahl der Prozessoren wird?



Parallelisierung II

- Zur Illustrierung des Verfahrens betrachten wir den eindimensionalen Fall. Die Verteilung im höherdimensionalen Fall erfolgt entsprechend dem Tensorprodukt (schachbrettartig).
- Die Prozessorgrenzen werden gewählt bei $p \cdot \frac{1}{P} + \epsilon$, also die Knoten, die auf dem Rand zwischen zwei Prozessoren liegen, noch dem „vorderen “zugeteilt.
- Zu bemerken ist, daß der Defekt, der in der Grobgitterkorrektur restringiert wird, nur auf den eigenen Knoten berechnet werden kann, nicht aber im Overlap!
- Um dem Problem der schwindenden Anzahl von Knoten in den gröbsten Gittern zu begegnen, nutzt man sukzessive weniger Prozessoren. Sei dazu $a := \mathrm{ld}\, P$ und wieder $C := t_{\mathrm{op}}$. Auf Stufe 0 wird nur ein Prozessor beschäftigt, erst auf Stufe a sind alle beschäftigt.



Parallelisierung III

Stufe	Knoten	Prozessoren	Aufwand
l	$N_l = 2^{l-a} N_a$	$P_l = P$	$T = 2^{l-a} C N_0$
a+1	$N_{a+1} = 2N_a$	$P_{a+1} = P$	$T = 2CN_0$
a	N_a	$P_a = P$	$T = CN_0$
2	$N_2 = 4N_0$	$P_2 = 4P_0$	$T = \frac{CN_2}{4} = CN_0$
1	$N_1 = 2N_0$	$P_1 = 2P_0$	$T = \frac{CN_1}{2} = \frac{C2N_0}{2} = CN_0$
0	N_0	$P_0 = 1$	$G(N_0) \overset{\text{sei}}{\approx} CN_0$

- Betrachten wir den Gesamtaufwand: Von Stufe 0 bis Stufe a ist $\frac{N}{P}$ konstant, also wächst T_P wie $\mathrm{ld}\, P$. Daher bekommen wir

$$T_P = \mathrm{ld}\, P \cdot CN_0$$

- In den höheren Stufen bekommen wir

$$T_P = C \cdot \frac{N_l}{P} \cdot \left(1 + \frac{1}{2} + \frac{1}{4} + \ldots\right) = 2C\frac{N_l}{P}$$

- Den Gesamtaufwand bildet dann die Summe der beiden Teilaufwände.
- Nicht berücksichtigt ist dabei die Kommunikation zwischen den Prozessoren.

Parallelisierung IV

Wie wirkt sich der Einsatz des Mehrgitterverfahrens auf die Anzahl der auszuführenden Iterationsschritte aus?

- Aufgezeichnet wird die Anzahl der eingesetzten Prozessoren gegen die Wahl der Gitterfeinheit, die verwendet wurde.
- Es wurde eine Reduktion des Fehlers auf 10^{-6} durchgeführt.

P/l	5	6	7	8
1	10			
4	10	10		
16	11	12	12	
64	13	12	12	12

- Die Tabelle zeigt die entsprechenden Iterationzeiten in Sekunden für 2D (Faktor 4 Gitterwachstum):

P/l	5	6	7	8
1	3.39			
4	0.95	3.56		
16	0.32	1.00	3.74	
64	0.15	0.34	1.03	3.85

Die wichtigsten Erkenntnisse

- Jacobiverfahren ist eines der simpelsten Iterationsverfahren zur Lösung von linearen Gleichungssystemen.
- Bei festem Reduktionsfaktor ϵ benötigt man eine bestimmte Zahl von Iterationen IT um diese Fehlerreduktion zu erreichen.
- IT ist *unabhängig* von der Wahl des Startwertes, hängt aber unmittelbar von der Wahl des Verfahrens (z.B. Jacobiverfahren) und der Gitterweite h (also N) ab.
- Beim Jacobiverfahren gilt $IT = O(h^{-2})$. Bei einer Halbierung der Gitterweite h benötigt man die vierfache Zahl von Iterationen, um die Fehlerreduktion ϵ zu erreichen. Da eine Iteration auch viermal mehr kostet, ist der Aufwand um den Faktor 16 gestiegen!
- Es gibt eine Reihe von besseren Iterationsverfahren bei denen z.B. $IT = O(h^{-1})$, $IT = O(\log(h^{-1}))$ oder gar $IT = O(1)$ gilt (CG-Verfahren, hierarchische Basis, Mehrgitterverfahren).
- Asymptotisch ($h \to \infty$) ist natürlich jedes dieser Verfahren einem parallelen naiven Verfahren überlegen.
- Man sollte deshalb unbedingt Verfahren optimaler sequentieller Komplexität parallelisieren, insbesondere weil man auf dem Parallelrechner große Probleme (h klein) lösen möchte.