FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Algorithmik kontinuierlicher Systeme

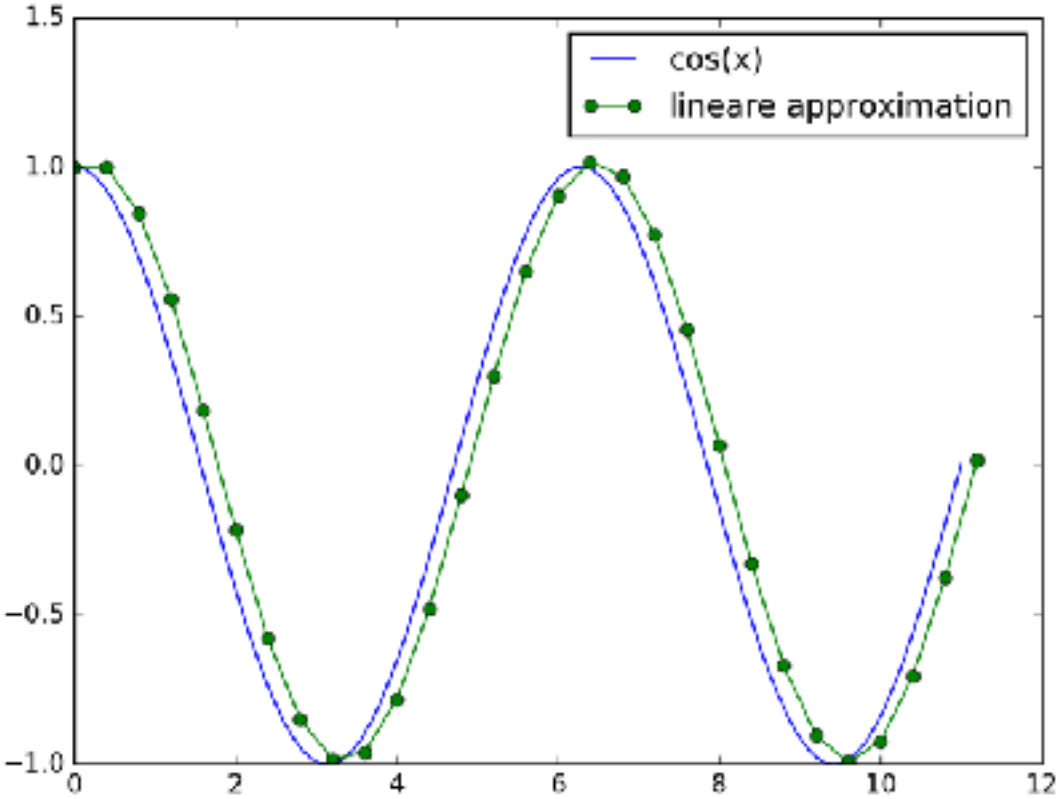
Iterative Verfahren (2/2) Vektoriteration

- Wie schnell können wir Gleichungssysteme lösen?

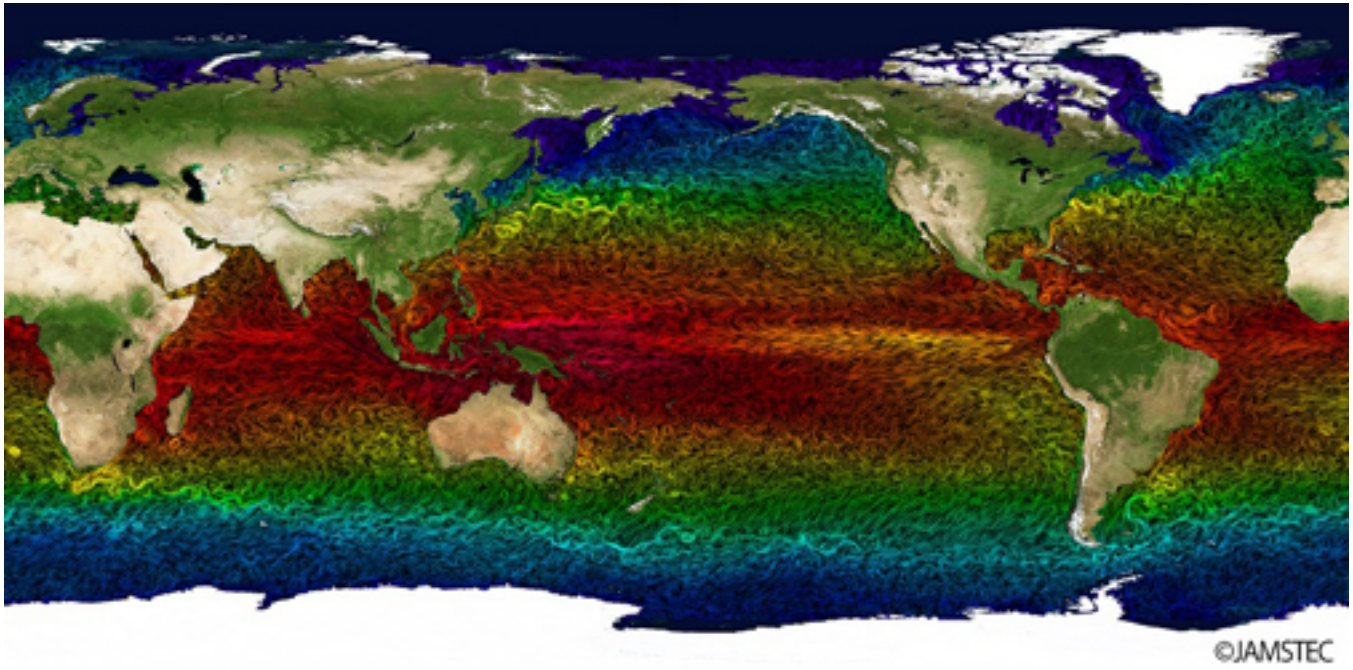
- LR- oder QR-Zerlegung: $\mathcal{O}(n^3)$
  - Immer anwendbar
  - Standardverfahren
  - Aber: n kann sehr groß werden

- Können wir wichtige Spezialfälle schneller lösen?
  - z.B. Bandmatrizen
  - Hier und heute: **Iterative Verfahren**!

- Das einfachste sinnvolle Modell
- Können effizient ausgewertet werden
- Nichtlineare Probleme können linear approximiert werden



- Mit linearen Gleichungssystemen können wir
  - vor Tsunamis und Hurricanes warnen
  - Auswirkungen globaler Erwärmung abschätzen
  - Krebs frühzeitig diagnostizieren
  - Autos, Flugzeuge etc. sicherer machen



- Am 26. Dezember 1823 schreibt C.F. Gauss an seinen Schüler Christian Gerling

Fortsetz[un]g.

Mein Brief ist zu spät zur Post gekommen und mir zurückgebracht. Ich erbreche ihn daher wieder, um noch die praktische Anweisung zur Elimination beizufügen. Freilich gibt es dabei vielfache kleine Lokalvorteile, die sich nur ex usu lernen lassen.

Ich nehme Ihre Messungen auf Orber Reisig zum Beispiel:

[...]

[...]

Die Bedingungsgleichungen sind also:

$$
\begin{array}{rrrrrr}
0 = & +6 & +67a & -13b & -28c & -26d \\
0 = & -7558 & -13a & +69b & -50c & -6d \\
0 = & -14604 & -28a & -50b & +156c & -78d \\
0 = & +22156 & -26a & -6b & -78c & +110d \\
\hline
\text{Summe} = & 0 & & & &
\end{array}
$$

Um nun indirekt zu eliminieren, bemerke ich, daß, wenn drei der Größen a, b, c, d = 0 gesetzt werden, die vierte den größten Wert bekommt, wenn d dafür gewählt wird. Natürlich muß jede Größe aus ihrer eigenen Gleich[un]g, also d aus der vierten, bestimmt werden. Ich setze also d = −201 und substituiere diesen Wert. Die absoluten Teile werden dann +5232, −6352, +1074, +46, das übrige bleibt dasselbe.

Jetzt lasse ich b an die Reihe kommen, finde b = + 92, substituiere u[nd] finde die absoluten Teile + 4036, — 4, — 3526, — 506.

So fahre ich fort, bis nichts mehr zu korrigieren ist. Von dieser ganzen Rechnung schreibe ich aber in Wirklichkeit bloß folgendes Schema:

	d = −201	b = + 92	a = −60	c = + 12	a = + 5	b = − 2	a = − 1
+ 6	+ 5232	+ 4036	+ 16	− 320	+ 15	+ 41	− 26
− 7558	− 6352	− 4	+ 776	+ 176	+ 111	− 27	− 14
− 14604	+ 1074	− 3526	− 1846	+ 26	− 114	− 14	+ 14
+ 22156	+ 46	− 506	+ 1054	+ 118	− 12	0	+ 26

Insofern ich die Rechnung nur auf das nächste $^1/_{10000}$ Sek[unde] führe, sehe ich, daß jetzt nichts mehr zu korrigieren ist. Ich sammle daher

a = − 60	b = + 92	c = + 12	d = − 201
+ 5 − 1	− 2		
− 56	+ 90	+ 12	− 201

- Der Brief von Gauss im Jahre 1823 beschreibt das erste iterative Lösungsverfahren
- Das Verfahren wurde erst 1874 von Ludwig Seidel publiziert
- Varianten vom Gauss-Seidel-Verfahren sind das Jacobi-Verfahren und das SOR-Verfahren

Vorteile:

- Schneller als Gauss-Elimination
- weniger anfällig gegen Rechenfehler

- Algorithmik:
  - Was ist der zugrundeliegende Algorithmus?
  - Wann ist der Algorithmus anwendbar?
  - Wie schnell ist der Algorithmus?
  - Können wir weitere Verbesserungen vornehmen?

Diese Fragen sind gleichzeitig der Fahrplan dieser Vorlesung

- **Was ist der zugrundeliegende Algorithmus?**
- Wann ist der Algorithmus anwendbar?
- Wie schnell ist der Algorithmus?
- Können wir weitere Verbesserungen vornehmen?

Die Bedingungsgleichungen sind also:

$$
\begin{array}{rrrrrr}
0 = & +6 & +67a & -13b & -28c & -26d \\
0 = & -7558 & -13a & +69b & -50c & -6d \\
0 = & -14604 & -28a & -50b & +156c & -78d \\
0 = & +22156 & -26a & -6b & -78c & +110d \\
\hline
\text{Summe} = & 0 & & & &
\end{array}
$$

- Die Matrixnotation wurde erst 1850 eingeführt! Heute würde man schreiben

$$
\begin{bmatrix} +67 & -13 & -28 & -26 \\ -13 & +69 & -50 & -6 \\ -28 & -50 & +156 & -78 \\ -26 & -6 & -78 & +110 \end{bmatrix}
\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}
=
\begin{bmatrix} -6 \\ +7558 \\ +14604 \\ -22156 \end{bmatrix}
$$

Um nun indirekt zu eliminieren, bemerke ich, daß, wenn drei der Größen a, b, c, d = 0 gesetzt werden, die vierte den größten Wert bekommt, wenn d dafür gewählt wird. Natürlich muß jede Größe aus ihrer eigenen Gleich[un]g, also d aus der vierten, bestimmt werden. Ich setze also d = — 201 und substituiere diesen Wert. Die absoluten Teile werden dann + 5232, — 6352, + 1074, + 46, das übrige bleibt dasselbe.

- Startwert: $x_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$
- Koeffizient d optimieren: $x_1 = \begin{bmatrix} 0 & 0 & 0 & -201 \end{bmatrix}^T$
- Residuum berechnen:

  $b - Ax_1 = \begin{bmatrix} -5232 & +6352 & -1074 & -46 \end{bmatrix}^T$
- nächster Koeffizient, Residuum weiter verkleinern

- Definition: $r = b - A\tilde{x}$
- Das Residuum ist ein Maß für den absoluten Fehler, denn

$$A(\tilde{x} - e) = b$$
$$A\tilde{x} - Ae = b$$
$$-Ae = \underbrace{b - A\tilde{x}}_{\mathrm{r}}$$

- Je nach Gestalt von A wird der Fehler mehr oder weniger gut durch das Residuum beschrieben
- In jedem Fall gilt aber

$$r = 0 \quad \Leftrightarrow \quad e = 0$$

So fahre ich fort, bis nichts mehr zu korrigieren ist. Von dieser ganzen Rechnung schreibe ich aber in Wirklichkeit bloß folgendes Schema:

	d = −201	b = +92	a = −60	c = +12	a = +5	b = −2	a = −1
+ 6	+ 5232	+ 4036	+ 16	− 320	+ 15	+ 41	− 26
− 7558	− 6352	− 4	+ 776	+ 176	+ 111	− 27	− 14
− 14604	+ 1074	− 3526	− 1846	+ 26	− 114	− 14	+ 14
+ 22156	+ 46	− 506	+ 1054	+ 118	− 12	0	+ 26

- Tabelle mit Residuen
- In jedem Schritt wir das Residuum kleiner
- Gauss wählt stets den Koeffizienten mit dem größten Residuum

Insofern ich die Rechnung nur auf das nächste $^1/_{10000}$ Sek[unde] führe, sehe ich, daß jetzt nichts mehr zu korrigieren ist. Ich sammle daher

a = − 60	b = + 92	c = + 12	d = − 201
+ 5 − 1	− 2		
− 56	+ 90	+ 12	− 201

- Am Ende werden alle Korrekturen aufsummiert
- Nach 7 Schritten ist das Verfahren konvergiert

Fast jeden Abend mache ich eine neue Auflage des Tableau, wo immer leicht nachzuhelfen ist. Bei der Einförmigkeit des Messungsgeschäfts gibt dies immer eine angenehme Unterhaltung; man sieht daran auch immer gleich, ob etwas Zweifelhaftes eingeschlichen ist, was noch wünschenswert bleibt usw. Ich empfehle Ihnen diesen Modus zur Nachahmung. Schwerlich werden Sie je wieder direkt eliminieren, wenigstens nicht, wenn Sie mehr als zwei Unbekannte haben. Das indirekte Verfahren läßt sich halb im Schlaf ausführen oder man kann während desselben an andere Dinge denken.

- **Gauss-Seidel in Python:**

```
x = np.zeros((n,1))
while np.linalg.norm(b - A @ x) > eps:
    for k in range(n):
        left  = sum(a[k,0:k] * x[0:k,0])
        right = sum(a[k,k+1:] * x[k+1:,0])
        x[k] = (b[k] - left - right) / a[k,k]
return x
```

- Im Wesentlichen wird pro Zeile `a[k,k]` so gesetzt, dass in dieser Zeile das Residuum gleich 0 ist
- Anders als bei Gauss werden die Zeilen einfach zyklisch durchlaufen

- **Problem: inhärent sequentiell**

```
x = np.zeros((n,1))
while np.linalg.norm(b - A @ x) > eps:
    for k in range(n):
        left  = sum(a[k,0:k] * x[0:k,0])
        right = sum(a[k,k+1:] * x[k+1:,0])
        x[k] = (b[k] - left - right) / a[k,k]
return x
```

- Immer wenn **a[k,k-j]** ≠ 0 hängt **x[k]** von **x[k-j]** ab!
- Im allgemeinen nicht parallel ausführbar

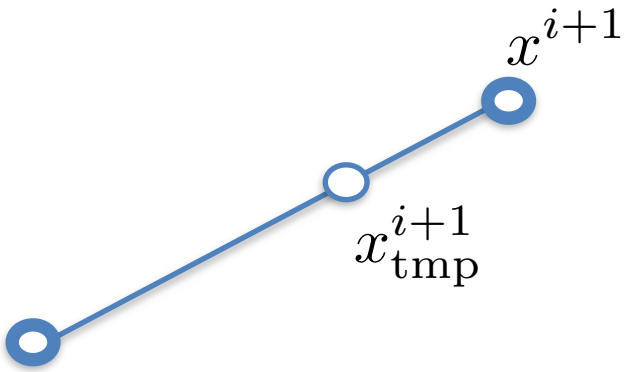
- **Besser parallelisierbar: Jacobi-Verfahren**

```
x = np.zeros((n,1))
while np.linalg.norm(b - A @ x) > eps:
    x_new = np.zeros((n,1))
    for k in range(n):
        left  = sum(a[k,0:k] * x[0:k,0])
        right = sum(a[k,k+1:] * x[k+1:,0])
        x_new[k] = (b[k] - left - right)/a[k,k]
return x_new
```

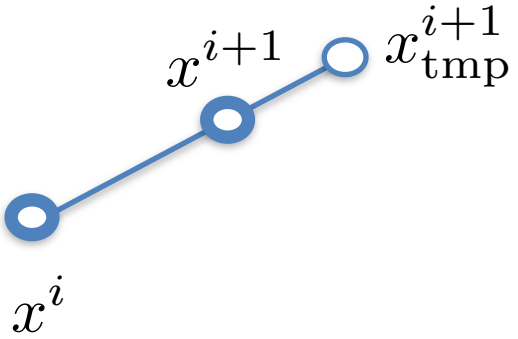
- **Nachteil:** Neue, bessere Werte **x[k-j]** werden ignoriert

- Konvergenzbeschleunigung durch Relaxation:
  Ersetze x^{i+1} durch $\omega\, x^{i+1} + (1-\omega)\, x^i$

$\omega > 1$: Überrelaxation



$0 < \omega < 1$: Unterrelaxation



- SOR = Successive Over Relaxation $1 < \omega < 2$
- Gauss-Seidel $\omega = 1$

- **Schnellere Konvergenz durch Überrelaxation**

```
x = np.zeros((n,1))
while np.linalg.norm(b - A @ x) > eps:
    for k in range(n):
        left  = sum(a[k,0:k] * x[0:k,0])
        right = sum(a[k,k+1:] * x[k+1:,0])
        x_tmp = (b[k] - left - right)/a[k,k]
        x[k] = (1 - w) * x[k] + x_tmp * w
return x
```

- Was ist der zugrundeliegende Algorithmus?
- **Wann ist der Algorithmus anwendbar?**
- Wie schnell ist der Algorithmus?
- Können wir weitere Verbesserungen vornehmen?

- Wir splitten die Matrix $A = L + D + R$
  - strikte untere Dreiecksmatrix L
  - Diagonale D
  - strikte obere Dreiecksmatrix R

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}, \quad L = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}, \quad R = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

- und schreiben
  - Gauss-Seidel: $(L + D)\, x^{i+1} + R\, x^i = b$
  - Jacobi: $D\, x^{i+1} + (L + R)\, x^i = b$

- Gauss-Seidel $(L+D)\,x^{i+1} + R\,x^i = b$

$$x^{i+1} = (L+D)^{-1}\,(b - R\,x^i)$$

```
for k in range(n):
    left  = sum(a[k,0:k] * x[0:k,0])
    right = sum(a[k,k+1:] * x[k+1:,0])
    x[k] = (b[k] - left - right) / a[k,k]
```

Wichtig: $(L+D)$ wird nicht wirklich invertiert!

Stattdessen: Vorwärtssubstitution

- Jacobi $$D\,x^{i+1} + (L+R)\,x^i = b$$

$$x^{i+1} = D^{-1}\,(b - (L+R)\,x^i)$$

```
for k in range(n):
    x_new = np.zeros((n,1))
    left  = sum(a[k,0:k] * x[0:k,0])
    right = sum(a[k,k+1:] * x[k+1:,0])
    x_new[k] = (b[k] - left - right) / a[k,k]
```

- Keine Datenabhängigkeiten, es wird jeweils einfach durch das Diagonalelement geteilt

- SOR $$(\frac{1}{\omega}D + L)x^{i+1} + ((1 - \frac{1}{\omega})D + R)\,x^i = b$$

$$x^{i+1} = (D + \omega L)^{-1}\,(\omega b - (\omega R + (1 - \omega)D)\,x^i)$$

```
for k in range(n):
    left  = sum(a[k,0:k] * x[0:k,0])
    right = sum(a[k,k+1:] * x[k+1:,0])
    x_tmp = (b[k] - left - right)/a[k,k]
    x[k] = (1-w)*x[k] + x_tmp*w
```

Fazit: Iterative Verfahren können mathematisch elegant beschrieben werden

- Allgemein
  - Splitte $A = \hat{A} + E$
  - einfach invertierbare Matrix $\hat{A}$
  - Restmatrix E
  - Iteration $x^{i+1} = \hat{A}^{-1}\left(b - E\,x^i\right)$

Resultat: Fixpunkt-Iteration für affine Abbildungen:

$$\Phi(x) = Vx + d$$

wobei $V = -\hat{A}^{-1}E$

und $d = \hat{A}^{-1}b$

- Fixpunkt-Iteration für affine Abbildungen:

$$\Phi(x) = Vx + d$$

Dabei ist V eine $n \times n$ Matrix und d ein n- Vektor

- Fragen:
  - Wann konvergiert diese Folge?
  - Gegen welchen Wert konvergiert sie?
  - Wie schnell konvergiert sie?

- Wann konvergiert die Folge $x_{i+1} = Vx_i + d$?
  - hinreichende Bedingung: $||V|| < 1$ für irgendeine Operatornorm $||.||$
  - ebenfalls hinreichende Bedingung:
    - ⋆ $|\lambda| < 1$ für alle Eigenwerte von V bzw.
    - ⋆ Spektralradius $\rho(V) < 1$
- Konvergenz-Geschwindigkeit: linear (Banachscher Fixpunktsatz)

$$||x^i - x^0|| = C\,\rho(V)^i$$

In der Praxis:

- Konvergenzgeschwindigkeit ist problemabhängig
- Nur für einfache Beispiele lässt sich der Spektralradius abschätzen

- Ob Jacobi-, Gauss-Seidel oder SOR-Verfahre konvergieren hängt somit vom Spektralradius der jeweiligen Iterationsmatrizen ab:

- Im Einzelnen mit $A = L + D + R$

$$\rho(V_J): \qquad V_J = -D^{-1}(L+R)$$

$$\rho(V_{GS}): \qquad V_{GS} = -(L+D)^{-1}R$$

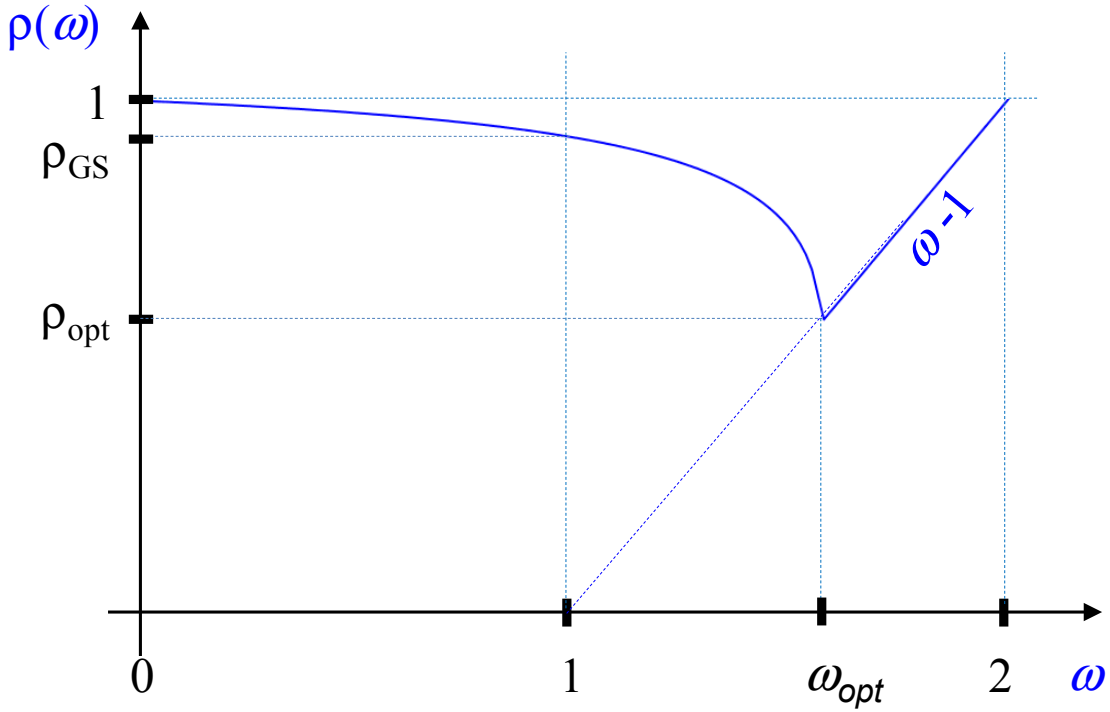
$$\rho(V_{SOR}): \qquad V_{SOR} = -(\frac{1}{\omega}D + L)^{-1}((\omega-1)/\omega D + R)$$

- Ist A strikt diagonaldominant, d.h.

$$|a_{i,i}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{i,j}| \quad \forall i \in 1, \ldots, n$$

  Dann konvergieren Gauss-Seidel, Jacobi und SOR-Verfahren (für $0 < \omega < 2$)
- Ist A positiv definit (alle Eigenwerte positiv), dann konvergiert das SOR-Verfahren für $0 < \omega < 2$ und damit auch das Gauss-Seidel Verfahren
- Ist sowohl A, als auch $D - 2A$ positiv definit, dann konvergiert auch das Jacobi-Verfahren

- Ist $\rho(\omega)$ der Spektralradius der Iterationsmatrix $V_{\mathrm{SOR}}(\omega)$ des SOR-Verfahrens, so ist ein typischer Verlauf des Spektralradius



- Eine Matrix heißt **schwach-diagonaldominant**, wenn

$$|a_{i,i}| \geq \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{i,j}| \quad \forall i \in 1, \ldots, n$$

- Beispiele:

$$\begin{bmatrix} 3 & -2 \\ -4 & 5 \end{bmatrix} \qquad \begin{bmatrix} 3 & 1 & -1 \\ 1 & -5 & 2 \\ 1 & 2 & 4 \end{bmatrix} \qquad \begin{bmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{bmatrix}$$

- Eine Matrix heißt **unzerlegbar**, wenn der Graph von A (Kanten für alle von Null verschiedenen Einträge) stark zusammenhängend ist, d.h. es gibt zwischen beliebigen Knoten einen durchgehenden Pfad.

- Wichtigstes praktisches Resultat:

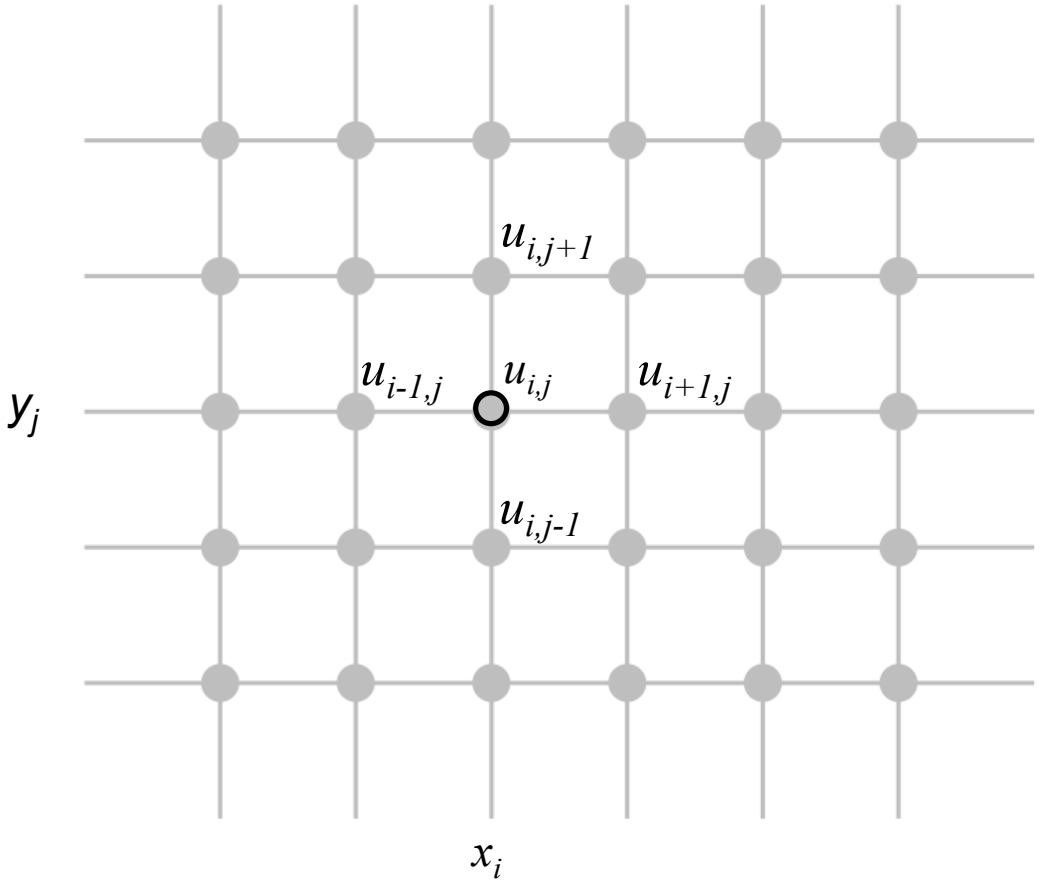
Ist A unzerlegbar und schwach-diagonaldominant, dann konvergieren Gauss-Seidel-, Jacobi- und SOR-Verfahren

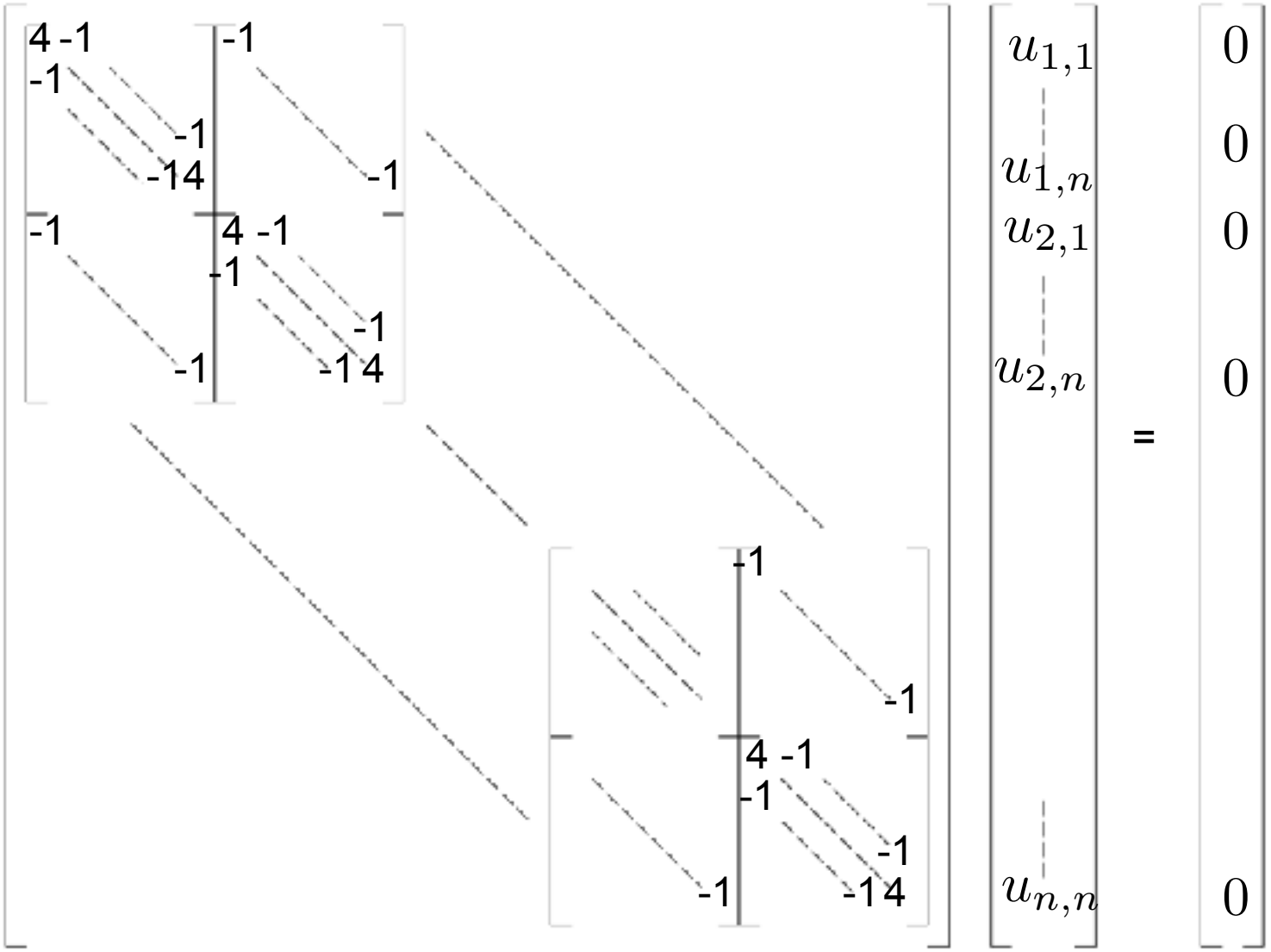
- Schwache Diagonaldominanz ist typisch bei Modellen mit Erhaltungsgrößen (z.B. Energie, Masse, Impuls)
- Realistische Probleme führen meist zu unzerlegbaren Matrizen

- Bei SOR und Gauss-Seidel hängt das Ergebnis auch von der Reihenfolge der Unbekannten ab, bei Jacobi nicht
- In vielen Fällen konvergiert das Gauss-Seidel-Verfahren etwa doppelt so schnell wie das Jacobi-Verfahren
- Splitting-Verfahren konvergieren nicht in jedem Fall, sondern nur für bestimmte Problemklassen
- Oft hängt der Spektralradius auch von der Größe des Gleichungssystems ab, d.h. je größer das Problem desto schlechter konvergieren diese Verfahren

- Was ist der zugrundeliegende Algorithmus?
- Wann ist der Algorithmus anwendbar?
- **Wie schnell ist der Algorithmus?**
- Können wir weitere Verbesserungen vornehmen?

- Beispiel: Diskretisierung der Poisson-Gleichung
- Kontinuierlich: $-\Delta u = 0$
- Diskret: $4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} = 0$





$$4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} = 0$$

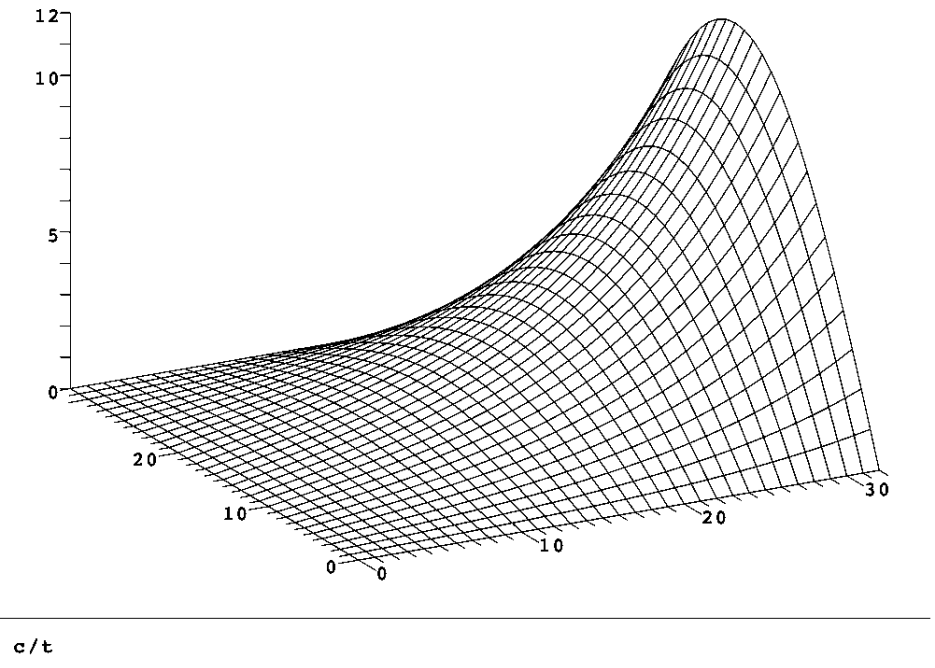
- Achtung: Das $(N+1) \times (N+1)$ Gitter U ist nicht die Koeffizienten-Matrix!
  Die Werte im Innern des Gitters sind die Unbekannten!
- Matrix des Gleichungssystems hat Größe $(N-1)^4$!
- Rechte Seite enthält
  - vorgegebene „Randwerte“ und
  - „Temperaturquellen“ an den Gitterpunkten (dies ist eine Verallgemeinerung des Problems).
- Aufstellen der Matrix in geeignetem Sparse-Matrix-Format
- Alternativ (besser): Direkte (matrix-freie) Implementierung eines iterativen Verfahrens

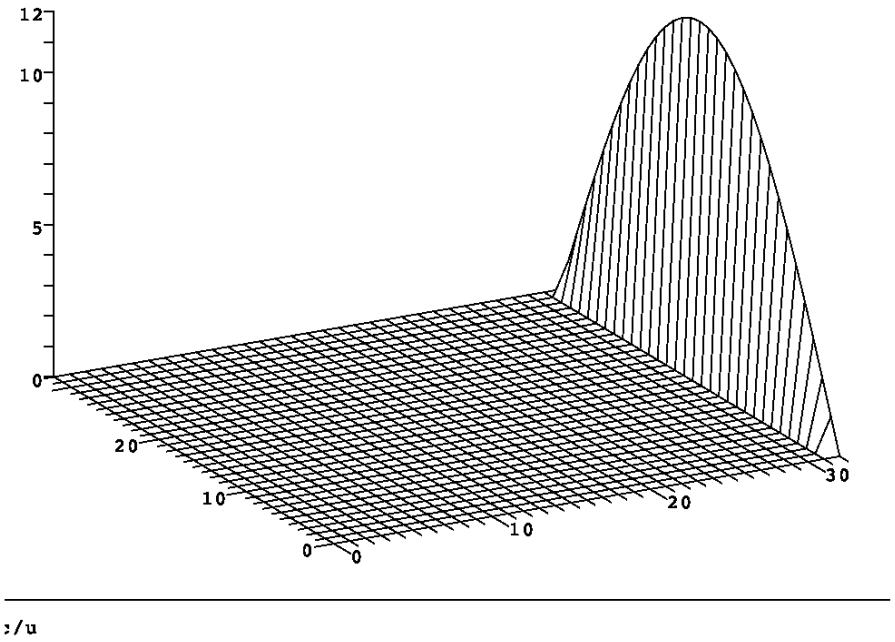
- Matrixfreie Implementierung des SOR-Verfahrens

```
def SOR(U, F, h, omega):
    for iy in range(1, m-1):
        for ix in range(1, n-1):
            U_tmp = (2*h*h * F[iy, ix]
                           + U[iy-1, ix]
                           + U[iy+1, ix]
                           + U[iy, ix-1]
                           + U[iy, ix+1]) * 0.25
            U[iy, ix] = U[iy, ix]*(1-omega) + omega*U_tmp
    return U
```

Graphische Illustration (Visualisierung)

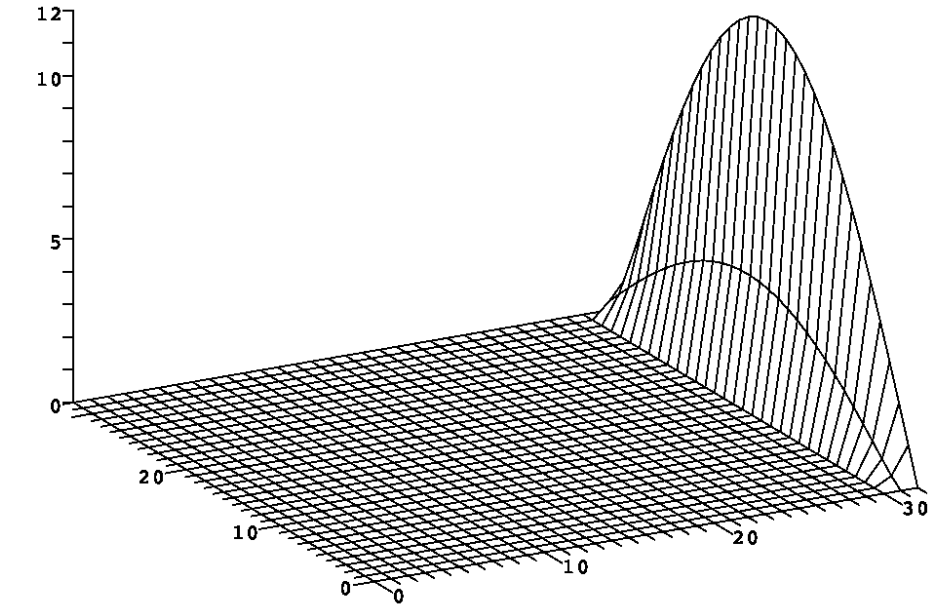
$$u = \sinh(x)\sin(y)$$



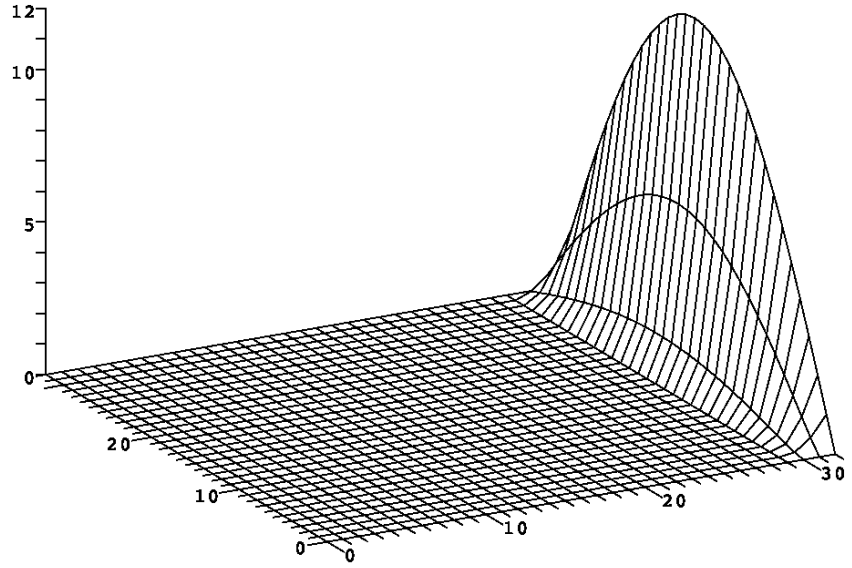


- Exakte Lösung (der PDE)
- Randwerte am Start der Iteration

Visualization of Iterations

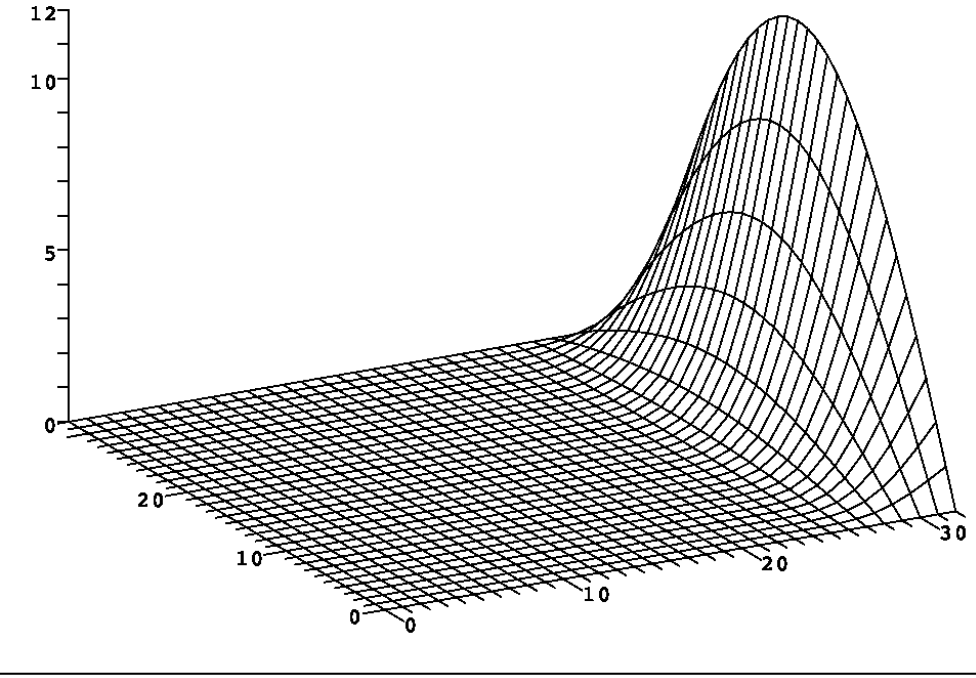


c/u.2

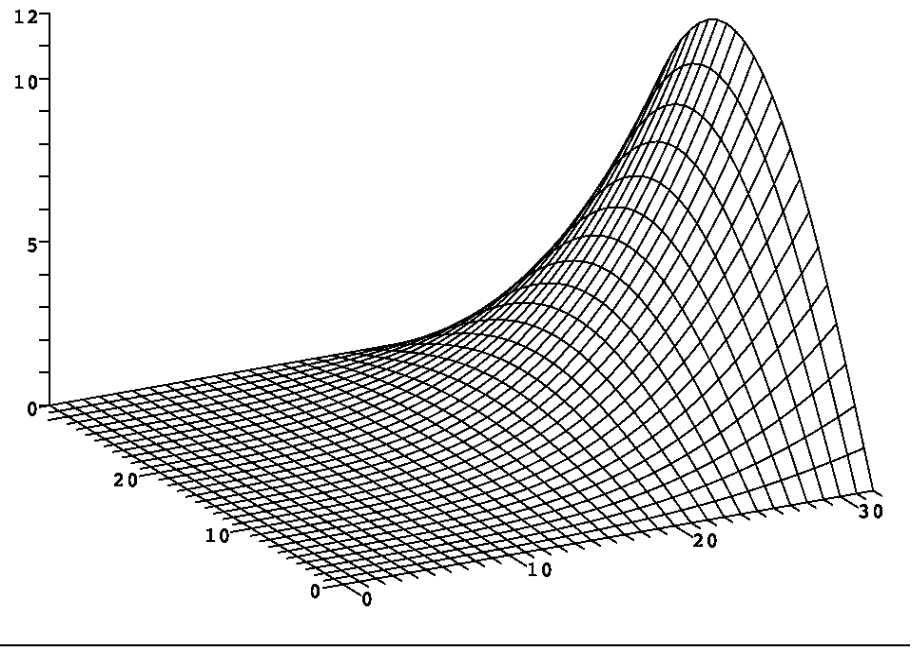


c/u.7

- Bild der Approximationslösung nach 1 bzw. 2 Iterationen

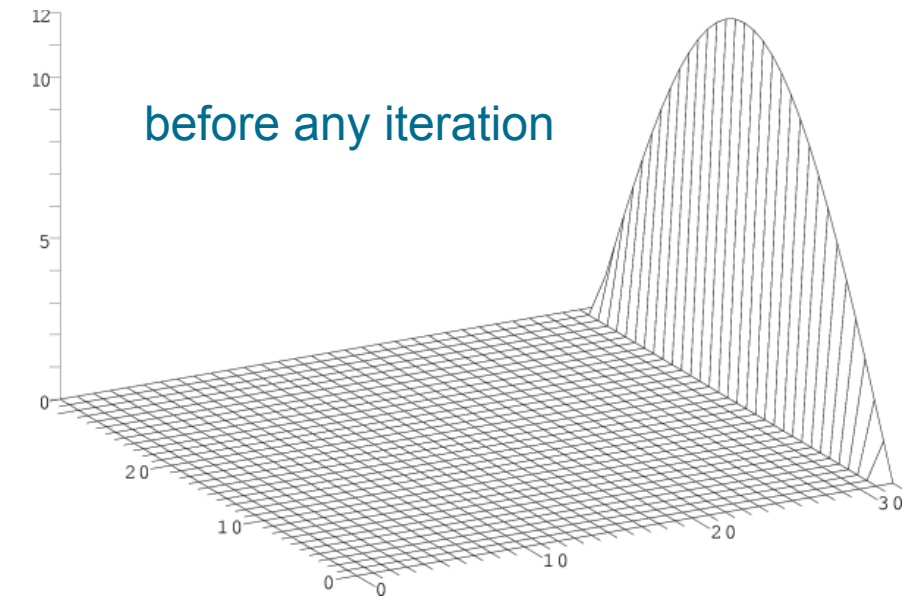


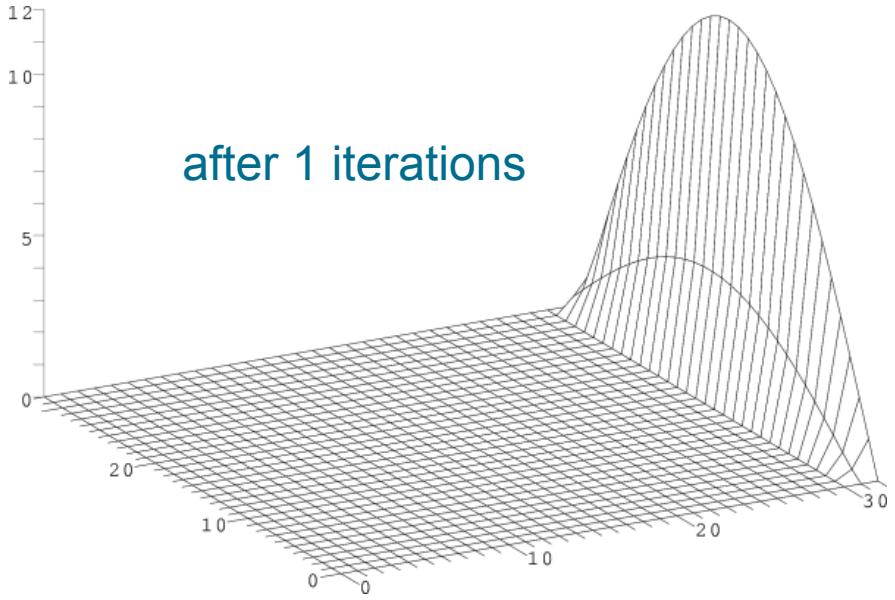
c/u.8

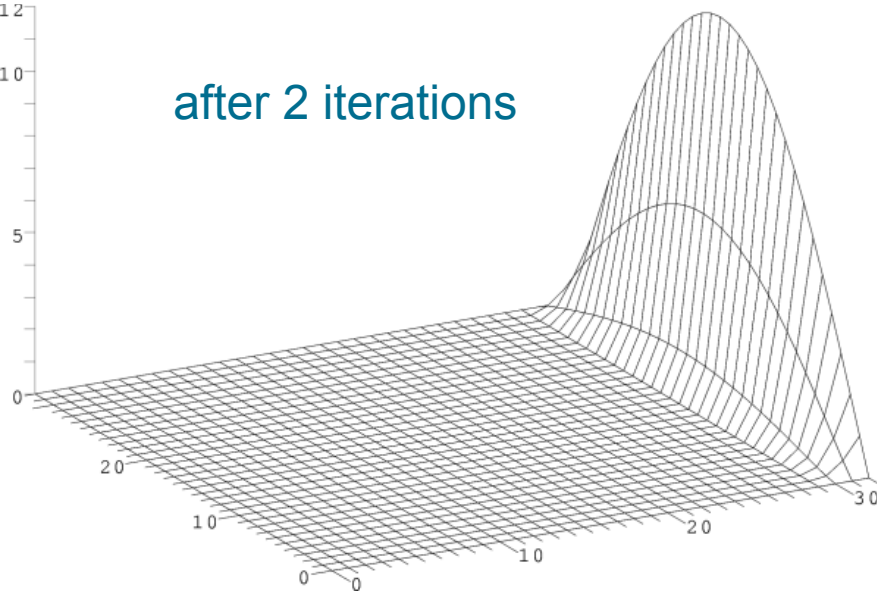


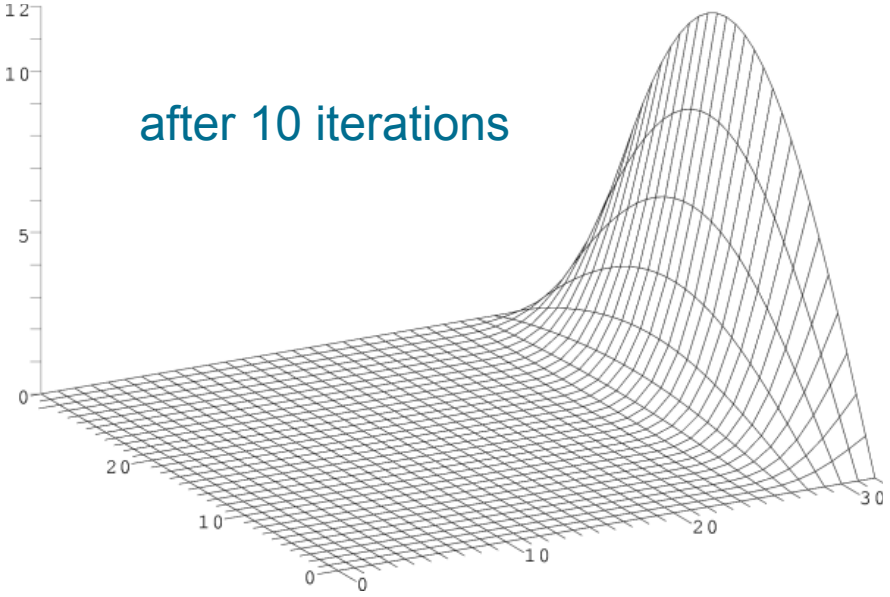
c/u.9

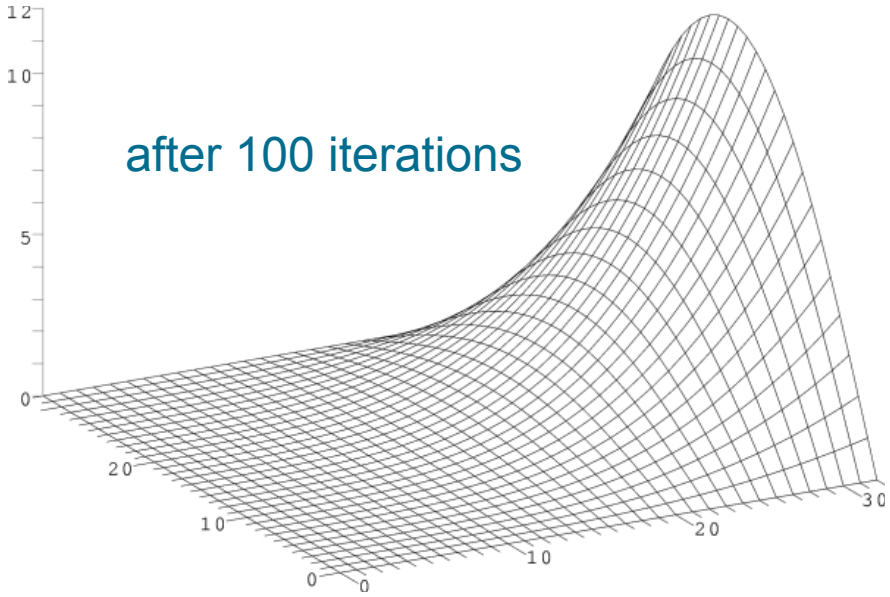
- nach 10 und nach 100 Iterationen

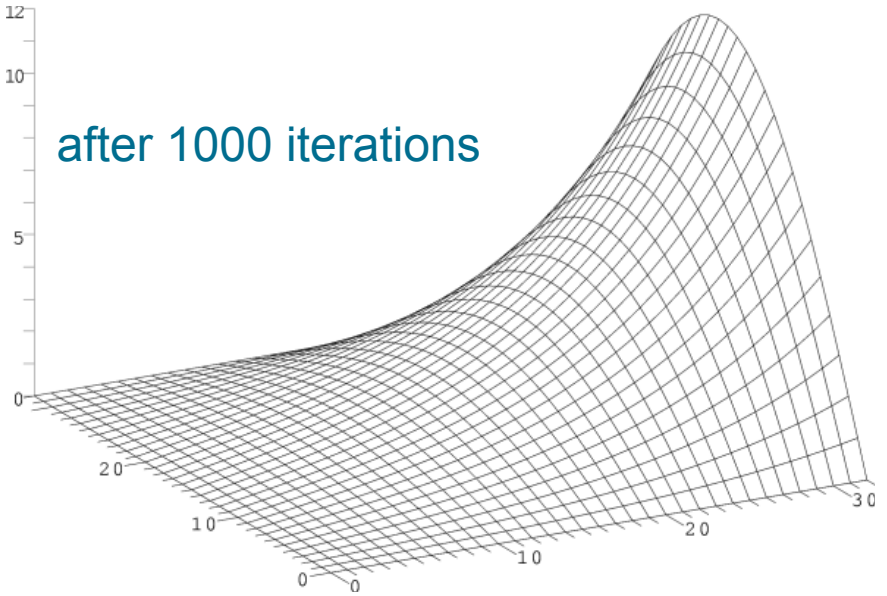

before any iteration
12
10
5
0
20
10
0
0
10
20
30

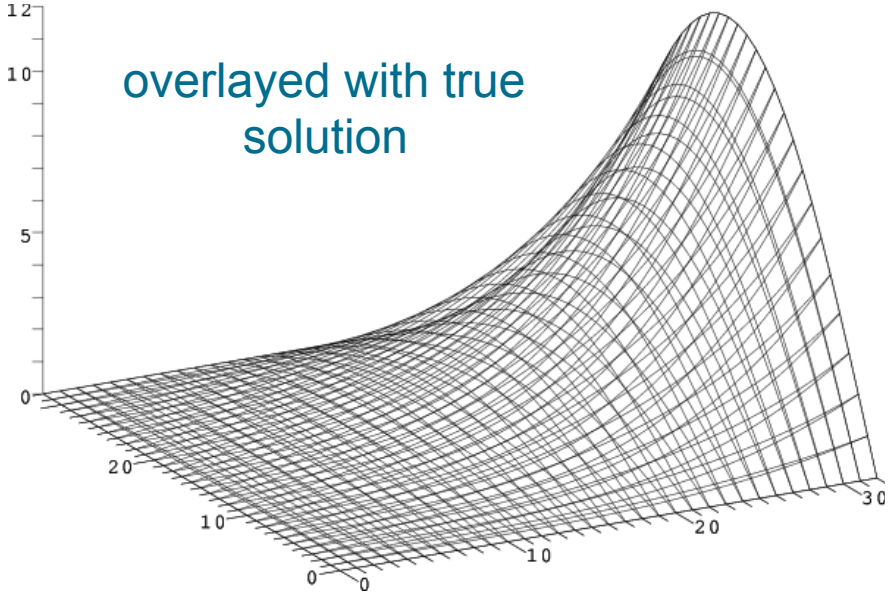

after 1 iterations
12
10
5
0
20
10
0
0
10
20
30

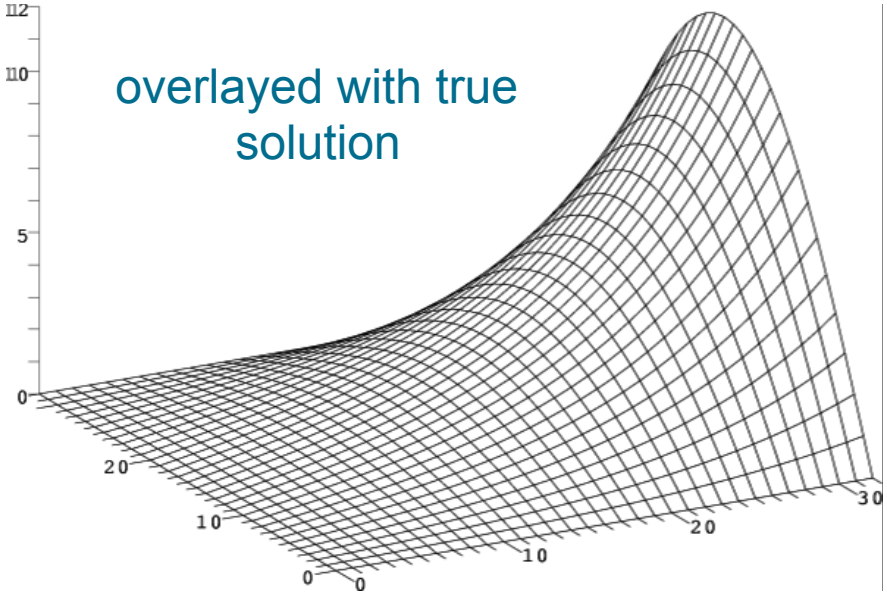

after 2 iterations
12
10
5
0
20
10
0
0
10
20
30


after 10 iterations
12
10
5
0
20
10
0
0
10
20
30

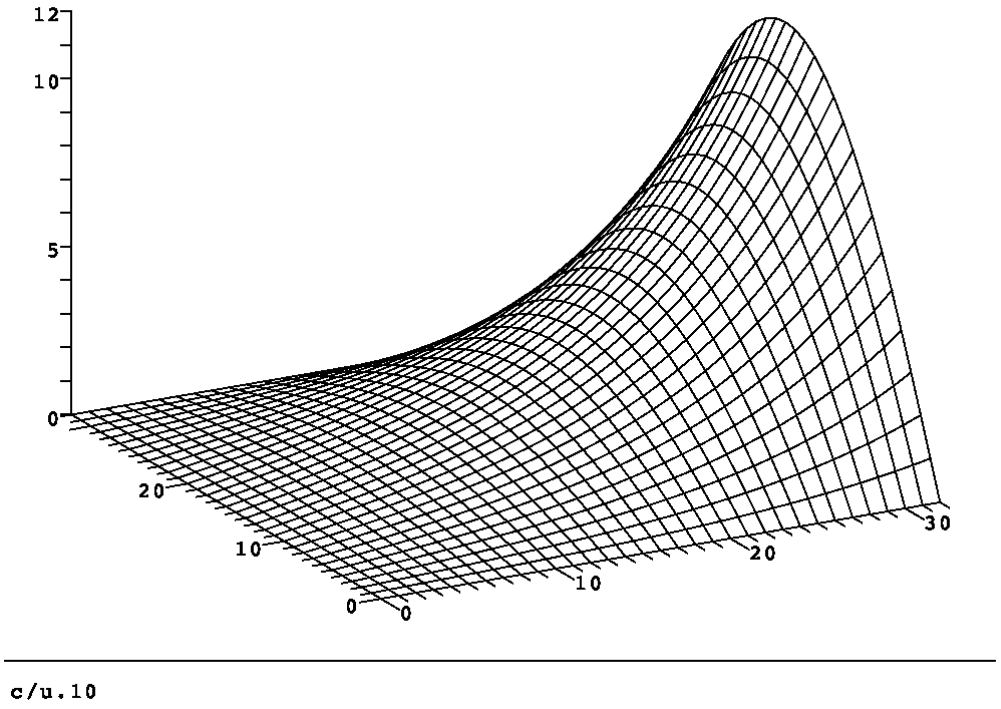

after 100 iterations


after 1000 iterations


overlayed with true solution


overlayed with true solution

- Bild der Iterationslösung nach 1000 Gauss-Seidel-Iterationen
- Das Gauss-Seidel-Verfahren benötigt $\mathrm{O}(N)$ iterations, wobei $N(=n*m)$ die Zahl der Gitterpunkte ist.



- Das Gauss-Seidel-Verfahren ist in etwa doppelt so schnell wie das Jacobi-Verfahren
- Beide benötigen etwa $\mathcal{O}(n)$ Iterationen (bei n Unbekannten)
- Das SOR-Verfahren benötigt bei optimalem Relaxationsparameter nur $\mathcal{O}(\sqrt{n})$ Iterationen
- Der optimale Relaxationsparameter wächst mit der Systemgröße

- Diese Aussagen gelten nicht generell, aber für viele ähnliche Probleme

- In der Praxis kommt es häufig vor, dass der Spektralradius von Iterationsmatrizen nahe 1 ist.
- Die Tendenz ist

$$\rho_{GS/J} \approx 1 - \frac{\alpha}{n}$$

$$\rho_{SOR} \approx 1 - \frac{\beta}{\sqrt{n}}$$

- Fazit: Bei sehr großen Gleichungssystemen braucht man sehr viele Interaktionen und die Effizienz des Algorithmus leidet.

- Was ist der zugrundeliegende Algorithmus?
- Wann ist der Algorithmus anwendbar?
- Wie schnell ist der Algorithmus?
- **Können wir weitere Verbesserungen vornehmen?**

- Unmittelbar anwendbare Techniken
  - Durchlaufreihenfolge geschickt wählen
  - Matrixfreie Implementierung
  - Dynamische Anpassung des Relaxationsparameters

- Fortgeschrittene Techniken
  - Mehrere Iterationen speichern und sinnvoll ausnutzen, führt zu Krylov-Unterraum-Verfahren, z.B. CG, GMRES
  - Hierarchische Gleichungssysteme konstruieren um unterschiedliche Frequenzen jeweils Optimal zu behandeln -> Mehrgitterverfahren
  - Kombinationen aus verschiedenen Verfahren, z.B. Systeme vorkonditionieren

- Vektoriteration
  - Umformung von $Ax = b$ über $A = \hat{A} + E$ zu einem iterativen Verfahren $x^{i+1} = \hat{A}^{-1}(b - E\,x^i)$
  - Allgemeine Form: $\Phi(x) = Vx + d$
  - Entscheidend für die lineare Konvergenz ist $\rho(V) < 1$

- Faustregeln
  - Gauss-Seidel konvergiert doppelt so schnell wie Jacobi
  - Gauss-Seidel und Jacobi benötigen $\mathcal{O}(n)$ Iterationsschritte, SOR nur $\mathcal{O}(\sqrt{n})$ (im Idealfall)

- Nur Jacobi ist einfach parallelisierbar

- In der Theorie: exakte Lösung vs. Näherungslösung
- Rundungsfehler vs. Iterationsfehler
  - DV: Rundungsfehler pflanzen sich fort u.U. mit Verstärkung
  - IV: Iterations- und Rundungsfehler werden abgeschwächt
- IV : schnell eine (grobe) Näherung, kann ggf. abbrechen.
  DV : erst am Ende kennt man die Lösung
- Oftmals genügt eine Näherungslösung
- je größer die Matrix und je dünner sie besetzt ist, desto interessanter werden iterative Verfahren
- Bei dünn besetzten Matrizen sind iterative Verfahren oft effizienter als direkte, d.h. sie finden eine (ausreichend genaue) Lösung in besserer Komplexität als $\mathcal{O}(n^3)$

- Iterative Verfahren können noch wesentlich leistungsfähiger sein
- Verfahren der konjugierten Gradienten (CG-Verfahren), ein nichtlineares Iterationsverfahren, das wir im Zusammenhang mit Optimierung kennen lernen werden
  - Krylov-Raum-Verfahren
  - genaueres in der Vorlesung ANLA (Algorithmen der numerischen Linearen Algebra)
  - gleiche Komplexitätsklasse wir SOR, aber genereller anwendbar
- Mehrgitterverfahren (erreichen optimale Komplexität $\mathrm{O}(N)$)
  - z.B. Grundlage von des TerraNeo Projekts in der Geophysik
  - wird z.B. in den Vorlesungen SiWiR (Simulation und wissenschaftliches Rechnen behandelt)