I.3 Lineare Algebra für (parallele) Programmierung

Vektorisierung der phys. Probleme

Warum Vektorisierung? Viele Hardwarearchitektur, Software-Bibliotheken etc. sind optimiert um Matrixmultiplikation und Vektoroperationen durchzuführen.

Beispiele für typische lineare Probleme (formalisiert):

Lösung linearer Gleichungssysteme

$$\underline{\underline{A}} \cdot \underline{x} = \underline{b}$$

Lösung von zeitabhängigen Problemen

$$\partial_t \underline{v} = \underline{\underline{M}} \cdot \underline{v}$$

Eigenwertproblem

$$\underline{\underline{A}} \cdot \underline{x} = E \, \underline{x}$$

(und weitere Probleme wie spectral transformation, singular value decomposition ...)

Beispiele für physikalische Probleme

Schrödingergl.

$$H |\psi\rangle = E |\psi\rangle \Rightarrow$$

↑ Matrix ↑ Vektor

Wichtig ist es eine geeignete Basis zu finden (s. Finite differenzen und Quantenchemie)

⇙

Übersetzen in Spaltenvektoren

$$|\psi\rangle = \sum_i c_i |i\rangle$$

Liouville-von Neumann Gl.

$$\partial_t \rho = -\frac{i}{\hbar} [H, \rho]_-$$

...

(Hier ist die Dichtematrix ein Vektor
[z.B. $\langle i|\rho|j\rangle = v[i+N\cdot j]$]
und $-\frac{i}{\hbar}[H,\cdot]$ wird auf Matrix abgebildet.

Helmholtzgleichung:

$$\Delta \underline{A}(\underline{r},\omega) + \frac{\omega^2}{c^2}\varepsilon(\omega)\underline{A}(\underline{r},\omega) = 0$$

(Δ und $\frac{\omega^2}{c^2}\varepsilon(\omega)$: Matrix; $\underline{A}$: Vektor)

+ Randbedingungen

Besondere Form des Eigenwertproblems (nicht Hermitesch)
z.B. wichtig für Quantisierung des Lichtfelds
Entwicklung von $\underline{A}(\underline{r},\omega)$ über Gitter oder Finite Elemente.

Außerdem gibts es auch vektorielle Fassungen für nicht lineare Probleme.

Beispiele für Vektor und Matrixformate

Sequentielle Vektor

Vektorformat: Vektor ist ein Array von Fließkommazahlen

Speicher	z[0]	z[1]	z[2]	z[3]	....	z[n]

Abbildung im Speicher single (32 Bit) oder double (64 Bit) precision Fließkommazahlen.
~~z.B.~~ Genaues Datenformat Hardware abhängig.
z.B. IEEE 754

$$x = s\, m\, 2^{e}$$

e ← Exponent (8 bit single $-126 \le e \le 127$; 11 bit double $-1022 \le e \le 1023$)

s ← Vorzeichen

m ← Mantisse (23 bit single, 52 bit double)

Wichtig: Genauigkeit beschränkt
(Multiplikation und Addition sehr großer und sehr kleiner Zahlen ist schwierig)

Bei komplexen Zahlen besteht z [ ] aus zwei Werten reale und imaginär Teil (Ist z.B. C, C++ fertig implementiert ...)

Matrixformate

Einfachste Möglichkeit (als Block)

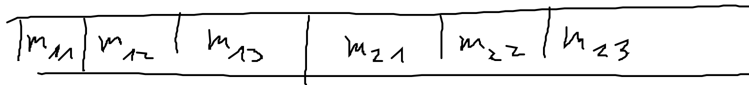
Matrix

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \end{pmatrix}$$

Zwei Möglichkeiten das im Speicher abzulegen.

1) Spalte zuerst (Column major) (z.B. Fortran und BLAS)

2.) Reihe zuerst (Row major) (z.B. Standard bei C, C++)



Low-level Routine für diese Matrixformate findet sich in BLAS. (Das ist meist vom Admin, oder Hersteller des Supercomputers optimiert.)

Dieses Format ist ideal für dicht besetzte Matrix
(viele Einträge von null verschieden)

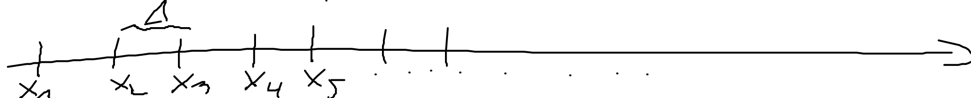
$$\begin{pmatrix} 1 & 2 & 9 \\ 6 & 7 & 8 \\ 9 & 0 & 10 \end{pmatrix} \leftarrow \text{Matrix mit wenig } 0.$$

Problem: (sehr) Oft gibt es dünn besetzte Matrizen

Ein Beispiel

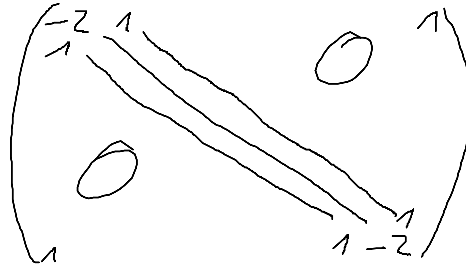
$$\partial_x^2 \psi(x)$$

Wir diskretisieren ψ



$\psi(x_1)\ \psi(x_2)\ \psi(x_3)\ \psi(x_4)\ \psi(x_5)$

$$\partial_x^2 \psi(x_n) \xrightarrow{\approx} \psi(x_{n-1}) - 2\psi(x_n) + \psi(x_{n+1})$$



Anz: Von N^2 Einträgen sind nur $3N$ Einträge besetzt!

$$\frac{3N}{N^2} = \frac{3}{N}$$

Bei großen Gitter wird das beliebig ineffektiv.
Das naive Format ist dann richtig ineffektiv.
(Literatur linalg netlib.org)

Dünn besetze Matrizen

Naivstes Format

Ein Array mit Fließkommazahlen v[ ]
Zwei Array mit ganzen Zahlen i[ ], j[ ]

Beispiel:

	i= 0	1	2	3
j= 0	2		6	
1				
2				8
3		7		
4				

v = {2, 6, 8, 7}
i = {0, 2, 3, 1}
j = {0, 0, 2, 3}

$\Longrightarrow$ Das benutzt keiner, da ineffektiv.

Compressed Row Storage Format
aka Yale sparse format, small sparse AIJ format

Array val[] speichert die Elemente, die nicht null sind.

Array col_ind[] enthält die Spaltenindizes

Array row_ptr[] enthält den Index, in dem col_ind Array, bei dem die Reihe gewechselt wird.

Beispiel (von netlib.org

$$M = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}$$

val = {10, -2, 3, 9, 3, 7, 8, 7, 3 ... 9, 13, 4, 2, -1}

col_ind = {0, 4, 0, 1, 5, 1, 2, 3, 0 .... 4, 5, 1, 4, 5}

row_ptr = {0, 2, 5, 8, 12, 16, 19}

Vorteil, Speicherplatz ersparnis, Algorithmen effektiv, Nachteil: Speicherzugriffsmuster kann ineffektiv

Compressed Column Storage: das gleiche mit vertauschten Zeile und Spalte (z. B. Transponierte zu berechnen)

Weitere Formate für spezielle Anwendungen

Für typische Formate existieren optimierte Fassungen der Matrixmultiplikation

Tipps: Zusammenbau der Matrix ist aufwendig.
In der Praxis muss man bevor man Speicher reserviert, die Anzahl der besetzten Einträge in jeder Zeile zählen.

Blockmatrizen (z.B. bei multiphysik Problem)

$$\begin{pmatrix} A_{aa} & A_{ab} & A_{ac} \\ A_{ba} & A_{bb} & A_{bc} \\ A_{ca} & A_{cb} & A_{cc} \end{pmatrix}$$

Zwei Möglichkeiten

1) monolithische Matrizen
oder
2) verschachtelte Matrizen