

Algorithmik kontinuierlicher Systeme

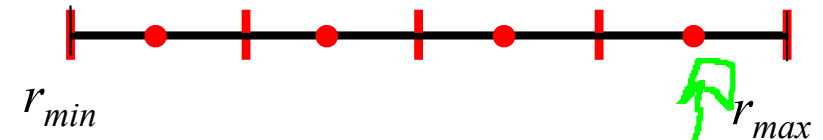
Diskretisierung und Quantisierung (Teil 2)



- Diskretisierung
 - ▶ Faltung
 - ▶ Fourier Transformation
 - ▶ Abtasttheorem
 - ▶ Aliasing / Anti-Aliasing

- Quantisierung
 - ▶ Diskrete Approximation reeller Werte
 - ▶ Gleitpunktzahlen, Maschinenzahlen
 - ▶ Beispiel: Quantisierung von Farbwerten (*median cut*)
 - ▶ Vektor-Quantisierung

- Zu approximieren $r \in \mathbb{R}, r_{min} \leq r \leq r_{max}$
- Wertebereich $W := [r_{min}, r_{max}]$
- Zerlegung in disjunkte, gleich große Teilintervalle
- L Teilintervalle $[w_j, w_{j+1}]$ der Länge $d = (r_{max} - r_{min})/L$
- Repräsentant $r_j \in [w_j, w_{j+1}]$, z.B: $r_j = (w_j + w_{j+1})/2$
- **Quantisierung:** $Q[r] \approx r_j$ falls $r \in [w_j, w_{j+1}]$
- Beobachtung: nicht optimal sofern die Punkte nicht gleichverteilt sind.

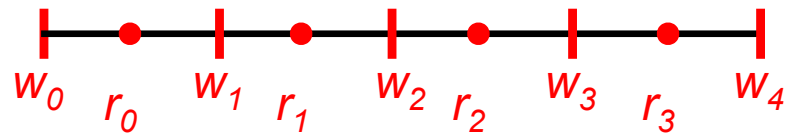


- Nahezu gleichverteilte Punkte:

XXXXXXXXXX XXXX XXXX XXXX XXXX

→ uniforme Quantisierung

|XXXXXXXX|XXXX|X|XXXXX|XXXXX|



XXXXXXXXXX XXXX XXXX XXXX XXXX

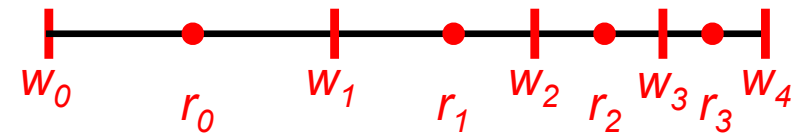
- Teile Intervall in äquidistante Teilintervalle und nimm jeweils Mittelpunkt als Repräsentanten

- Ungleich verteilte Punkte:

X X X X X X X X X X X X X X X X

→ nicht uniforme Quantisierung

|X X X X X X X X X X X X X X X X



X X X X X X X X X X X X X X X X

- Unterschiedliche Strategien

- Median Cut punkte anhand median aufteilen. problem: statist
- Least square minimizer
- Maschinenzahlen

- Definition:

nicht normiert

Die Menge der normalisierten t -stelligen Gleitpunktzahlen zur Basis B sind definiert als

$$FP_{B,t} = \{ \pm M \cdot B^E : M, E \text{ ganze Zahlen,} \\ E \text{ beliebig, } B^{t-1} \leq M < B^t \text{ oder } M = 0 \}$$

- B heißt Basis (eine ganze Zahl > 1)
- t ist die Anzahl der Stellen (ganze Zahl > 0)
- $M \geq B^{t-1}$ bedeutet, dass die erste Stelle $\neq 0$ sein muss

M ist die Mantisse,
 E ist der Exponent

- Beispiel:

Basis $B = 10$, Mantissenlänge $t = 2$

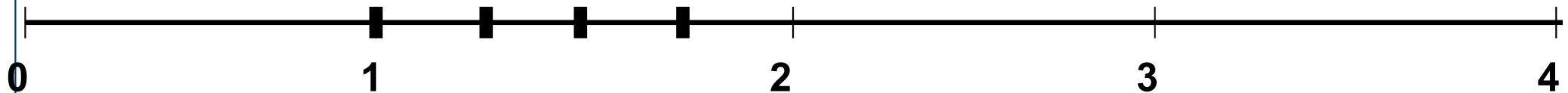
$$B^{t-1} \leq M < B^t : 10, 11, 12, \dots, 19, 20, 21, \dots, 91, 92, \dots, 99$$

- $E = 0 : 10, 11, 12, \dots, 19, 20, 21, \dots, 91, 92, \dots, 99$
- $E = 1 : 100, 110, 120, \dots, 190, 200, 210, \dots, 910, 920, \dots, 990$
- $E = -1 : 1.0, 1.1, 1.2, \dots, 1.9, 2.0, 2.1, \dots, 9.1, 9.2, \dots, 9.9$

- Beispiel:

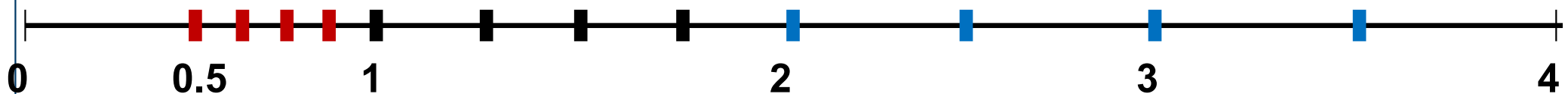
Basis $B = 2$, Mantissenlänge $t = 3$

- $E = -2 : 1.00, 1.01, 1.10, 1.11$ (binär)
 $1.0, 1.25, 1.5, 1.75$ (dezimal)



- $E = -3 : 0.100, 0.101, 0.110, 0.111$ (binär)

- $E = -1 : 10.0, 10.1, 11.0, 11.1$ (binär)



- Logarithmische Verteilung!

halblogarithmisch: Intervalle logarithmisch verteilt, aber innerhalb der Intervalle u

- Definition:

Die Menge der **Maschinenzahlen** ist definiert als

$$MFP_{B,t,\alpha,\beta} = \{ g \in FP_{B,t} : \alpha \leq E \leq \beta \}$$

- Hier ist jetzt auch der **Exponent eingeschränkt.**
 - ▶ B, t, α, β sind durch die Implementierung (also den Hersteller des Computers) festgelegt.
Sie werden **nie explizit gespeichert.**
 - ▶ Für jede Maschinenzahl $\pm M \cdot B^E$ wird **Vorzeichen** und die Zahlen **M (Mantisse)** und **E (Exponent)** gespeichert.

- Im IEEE-754 Format (von 1985, heute fast universell)

(Details zB http://de.wikipedia.org/wiki/IEEE_754)

- ▶ $B=2$
- ▶ Single precision (4 Bytes, 32 Bit): $t=24$ **(nur 23 Bits!)** erstes bit isz bei mantisse 1 mu

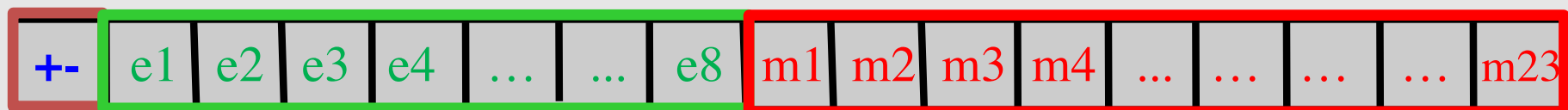
normalisierte Mantisse

$$m = 1 + \frac{m_1}{2} + \frac{m_2}{4} + \frac{m_3}{8} + \dots + \frac{m_{23}}{2^{23}}$$

Single Precision

Exponent

Mantisse



- ▶ Double precision (8 Bytes, 64 Bit): $t=53$ **(nur 52 Bits!)**

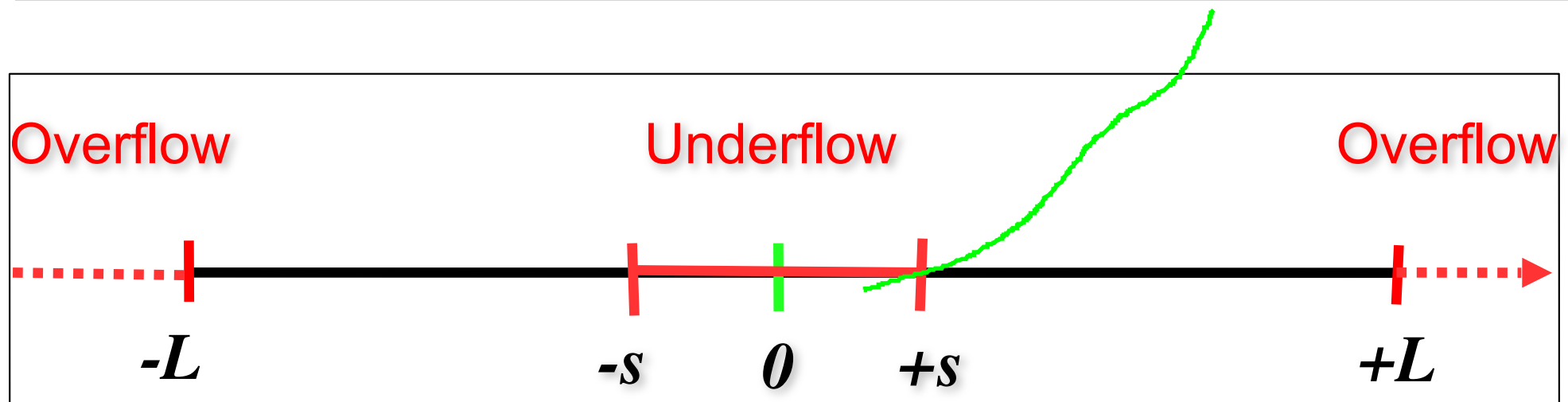
- Kleinste und größte positive Zahl:

$$E_{\max} = 254 - 127 = 127, m_{\max} = 1.111...1: L \approx 2 \cdot 2^{127} \approx 3.40 \cdot 10^{38}$$

$$E_{\min} = 1 - 127 = -126, m_{\min} = 1.000...0: s \approx 1 \cdot 2^{-126} \approx 1.18 \cdot 10^{-38}$$

mindestgenauigkeit

	t	Alpha	Beta	Resolution	s	L
Single	24	-149	104	1,19E-007	1,18E-038	3,40E+038
Double	53	-1074	971	2,22E-016	2,23E-308	2 ⁺¹⁰²⁴
Extend.	64	-16445	16320	1,08E-019	2 ⁻¹⁶³⁸²	2 ⁺¹⁶³⁸⁴



- Einfache Genauigkeit (single) entspricht etwa 7-8 Dezimalstellen, bei doppelter Genauigkeit (double) sind etwa 15-16 Dezimalstellen gespeichert.
- Sonderfälle (Exponent $E=11111111$ und $E = 00000000$)
 - Die Zahl 0
 - NaN (not a number): unbestimmter Wert, implementiert als „quiet“ (vererbt sich stillschweigend) oder „signaling“ (löst Alarm aus)
 - Exponentenüberlauf: $+\infty$ (Absolutwert der Zahl $> L$)
 - Exponentenunterlauf: $+0$ (Absolutwert der Zahl $< s$)
 - ...
 - (siehe auch: Webseite zu IEEE-Standard)

- Gleitpunktzahlen sind die Repräsentanten
- Quantisierung erfolgt durch Runden
 - Aufrunden: nächst größere Gleitpunktzahl
 - Abrunden: nächst kleinere Gleitpunktzahl
 - Exaktes/kaufmännisches Runden: nächst gelegene Gleitpunktzahl
 Was tun bei Mehrdeutigkeit?

- **Achtung: Es entstehen Rundungsfehler**

- **Absoluter Fehler** kann sehr groß werden
- **Relativer Fehler** ist beschränkt durch die Auflösung:

$$\rho = \frac{1}{B^{t-1}} = \frac{1}{2^{23}} \approx 1.19 \cdot 10^{-7}$$

- (das ist der maximale **relative** Abstand zweier benachbarter Zahlen Gleitpunktzahlen)

- Die eingeschränkte Genauigkeit von Gleitpunktzahlen kann zu (vollständig) falschen Ergebnissen führen
- Es geht nicht nur um die Rundungsfehler, auch der **Fehler in Eingabedaten** kann verstärkt werden.
Fast alle Daten aus der Realität haben (Mess-)Fehler. Auch wenn wir eine unbegrenzt genaue Arithmetik hätten, müssten wir vorsichtig sein.
- Algorithmen, die Fehler stark verstärken, heißen “**instabil**”. Algorithmen, die Fehler nicht verstärken, heißen “**stabil**”.
- Endliche Genauigkeit ist ein grundsätzliches Problem, das man nicht vermeiden kann, sondern nur lernen, damit umzugehen.

Fehlerquellen sind (u.a.)

- Messfehler und Rundungsfehler (Quantisierungsfehler) bei Eingabedaten → „einmalig“!
- Zwischenergebnisse arithmetischer Operationen sind keine exakten Gleitpunktzahlen:

Rundung auf nächstgelegene Gleitpunktzahl

Beispiel ($B = 10$, $t = 3$):

$$a = 3.38 \cdot 10^3, b = 7.27 \cdot 10^3, c = 4.73 \cdot 10^1$$

- ▶ $a+b = 10.65 \cdot 10^3$ $1.06 \cdot 10^4$
- ▶ $a+c = 3.4273 \cdot 10^3$ $3.43 \cdot 10^3$
- ▶ $a \cdot b = 2.45726 \cdot 10^7$ $2.46 \cdot 10^7$
- ▶ $a \cdot c = 1.59874 \cdot 10^5$ $1.60 \cdot 10^5$

- **Letzteres erfolgt zig-Millionen(Billionen) mal**
(Bsp LR-Zerlegung $n=1000 \rightarrow \frac{2}{3} \cdot n^3 = 666 \text{ Mio}$)

- Alle guten Implementierungen (z.B. gem IEEE-Standard) der Gleitpunktarithmetik arbeiten so, dass das Ergebnis einer Gleitpunktoperation $\dot{*}$ zu folgendem Prozess gleichwertig ist:

- ▶ Berechne das exakte Resultat
- ▶ Runde das Resultat zur nächsten Maschinenzahl (entsprechend eines der Rundungsmodi)

- Die Auflösung $\rho = \frac{1}{B^{t-1}}$ (rel. Abstand zweier

Gleitpunktzahlen, Maschinengenauigkeit) ist eine Schranke für den relativen Rundungsfehler

- Es gilt dann die starke Hypothese: Es gibt ein $\tilde{\varepsilon} = O(\rho)$

so dass

$$a \dot{*} b = (a * b) \cdot (1 + \varepsilon) \quad \text{mit} \quad |\varepsilon| = |\varepsilon(*, a, b)| \leq \tilde{\varepsilon}$$

punkt exakte operation* ist die m

- Für $*$, $/$ ist der **relative Fehler $1/B^{(t-1)}$** (= Rundungsfehler)
- Für $+$, $-$ kann **Auslöschung (cancellation)** auftreten:

wenn zwei Zahlen fast gleicher Größe subtrahiert werden
(5 Stellen Genauigkeit)

→ 1.23456xxxx E0


man verliert also die xxxx stellen

-1.23455xxxx E0

eine stelle genauigkeit

= 0.00001 = 1.xxxx E-5 (**Normalisierung**)

Die **xxxx-Stellen** werden mit 0en aufgefüllt (weil sie nicht darstellbar sind): Das führt dazu, dass man an **relativer Genauigkeit** (Anzahl gültiger Stellen) verliert.

- Gleitpunktarithmetik ist fehlerbehaftet
- Zusätzlich: Über- / Unterlauf (kleineres Problem)
- Hinsichtlich des relativen Fehlers sind
 - ▶ * und / gutartig;
 - ▶ +, - gefährlich, weil Auslöschung auftreten kann
- Wohlbekannten Rechengesetze gelten nur bedingt:
 - ▶ kommutativ OK, entgegen der landläufigen meinung, aber mit kommutativ kommt meist auch ein implizites as
 - ▶ assoziativ, distributiv hingegen **nicht!** 

- ▶ Beispiel (Rechnung mit 6 Dezimalstellen)

$$(1 + 1\,000\,000) - 1\,000\,000 \approx 1\,000\,000 - 1\,000\,000 = 0$$

$$1 + (1\,000\,000 - 1\,000\,000) \approx 1 - 0 = 1$$

runden

- Frage: Wie pflanzen sich (Rundungs-)Fehler fort?

- Im Golfkrieg 1991 verfehlte eine amerikanische Patriot-Rakete in Saudi-Arabien eine irakische Scud-Rakete. Die Scud-Rakete traf eine amerikanische Kaserne und tötete 28 US-Soldaten
- Ursache waren Rundungsfehler

- ▶ Die interne Uhr verwendete 24 Bit-Register zur Darstellung der Zeit in 1/10 Sekunden seit Systemstart
- ▶ 1/10 ist im Binärsystem nicht exakt darstellbar, nur die ersten 24 Stellen wurden verwendet,

$$0.1 = (0.000\overline{1100})_2 \approx (0.00011001100110011001100)_2$$

- ▶ Der Rundungsfehler entspricht $9.5 \cdot 10^{-8} \text{ sec}$
- ▶ Nach 100 Betriebsstunden (ohne Reboot) akkumulierte sich der Rundungsfehler zu ca. 0.34 sec.
- ▶ In dieser Zeit legte die Scud-Rakete 570 m zurück und war damit außerhalb der Reichweite der Sensoren der Patriot-Rakete

- Die **Anfälligkeit gegenüber Störungen** (Fehlern in Eingabedaten, Rundungsfehler,) eines numerischen Verfahrens wird durch zwei Konzepte beschrieben:

▶ **Kondition**

▶ **Stabilität**

schlechte kondition = schlechtes problem

- Kondition ist eine Eigenschaft des Problems !**
- Stabilität ist eine Eigenschaft des Algorithmus (des Lösungsverfahrens)!**

stabilität schlecht= schlechtes ergebnis

- Formal: ideal: Eingabe $x \rightarrow$ Ergebnis $F(x)$
real: Eingabe $\tilde{x} \rightarrow$ Ergebnis $\tilde{F}(\tilde{x})$

- Formal: ideal: Eingabe $x \rightarrow$ Ergebnis $F(x)$
real: Eingabe $\tilde{x} \rightarrow$ Ergebnis $\tilde{F}(\tilde{x})$

- Ein Problem ist **gut konditioniert**, wenn (bei exakter Rechnung) Fehler in den Eingangsdaten nicht verstärkt werden:

werden:
$$\frac{|F(\tilde{x}) - F(x)|}{|F(x)|} \approx \frac{|\tilde{x} - x|}{|x|}$$

relativer fehler!

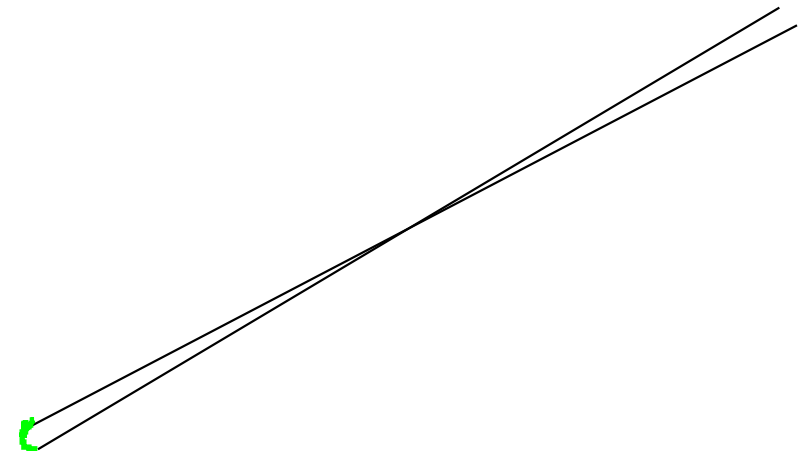
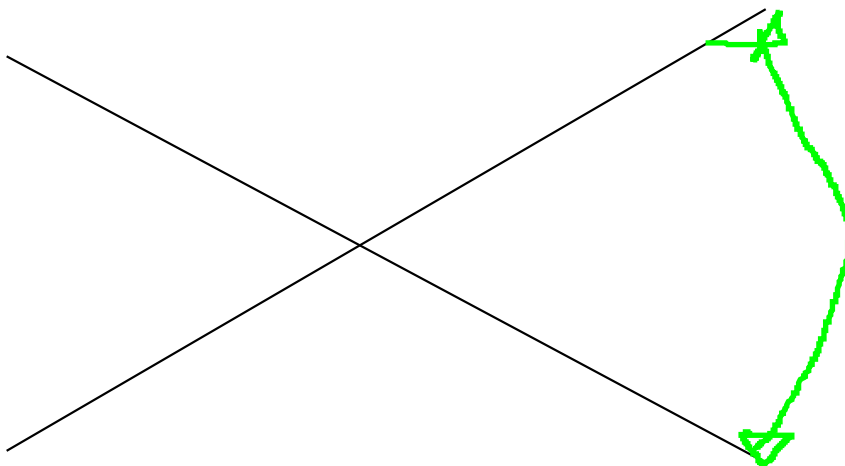
- Ein Algorithmus ist (vorwärts-) **stabil**, wenn die Rechenfehler nicht

akkumulieren:
$$\frac{|\tilde{F}(y) - F(y)|}{|F(y)|} \ll 1$$

idealerweise besser als maschinengenaugkeit. Problem: dies ist i.a. kaum einzuha

Berechnung des Schnittpunktes zweier Geraden

- Geraden „fast orthogonal“: Gute Kondition, d.h. leichtes „Wackeln“ an den Geraden verändert den Schnittpunkt kaum
- Geraden „fast parallel“: Schlechte Kondition, denn auch ein geringes „Wackeln“ kann den Schnittpunkt drastisch verändern.



- Bei schlecht konditionierten Problemen ist die einfache Definition der Vorwärts-Stabilität nicht sachgemäß:
- Eingabewerte x , exaktes Ergebnis sei $y = F(x)$
- Das tatsächlich berechnete Ergebnis y' heißt **akzeptabel** (wenn es als exaktes Ergebnis zu nur leicht gestörten Eingabewerten verstanden werden kann,
also $y' = F(x')$ mit $\|x' - x\| \leq \varepsilon$ also man nur einen kleinen Urbildbereich hat bei sol
- Ein Algorithmus, der akzeptable Ergebnisse liefert heißt rückwärts-stabil
- Bei dieser Definition müssen die Ergebnisse nicht unbedingt sehr genau sein.
- Bei einem schlecht konditionierten Problem können auch hundsmiserabel falsche Werte „akzeptabel“ sein
- Außer den Rundungsfehlern müssen noch andere Verfahrensfehler mit berücksichtigt werden:
 - Diskretisierungsfehler
 - Iterationsfehler

- Störungen der Eingabedaten müssen auch deshalb untersucht werden, weil ja die Eingabe (z.B. aus Messungen) von Haus aus nur ungenau vorliegt. Selbst bei exakter Rechnung muss man (von akademischen Problemen abgesehen) immer Störungen der Daten mit einzukalkulieren.
- Aus Perspektive des Anwenders:
 - ▶ Schlecht konditionierte Probleme sind schwer (im Extremfall gar nicht) lösbar
 - ▶ Bei schlechter Kondition kann jede noch so kleine Störung (ausgelöst durch Rundung, oder Messfehler, ...) in den Daten das Ergebnis vollständig unbrauchbar machen.

- **Gute Kondition:**

- ▶ Kleine Eingabestörungen führen zu kleinen Änderungen der Ergebnisse, Eingabestörungen sind unkritisch
- ▶ Es lohnt sich einen guten Algorithmus zu entwickeln (der die gute Kondition des Problems ausnützt)

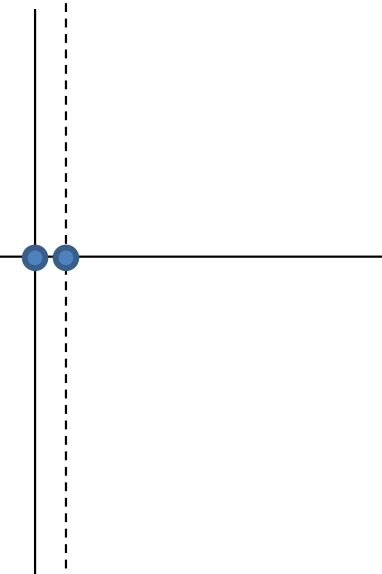
- **Schlechte Kondition**

- ▶ Selbst kleine Eingabestörungen können zu großen Störungen der Ergebnisse führen, Lösung reagiert empfindlich auf kleine Störungen der Eingabe
- ▶ Daran kann auch ein guter Algorithmus nichts ändern!
- ▶ Aus Perspektive des Anwenders:
 - ★ Schwierige Probleme, die im Extremfall gar nicht gelöst werden können
 - ★ Oft ein Zeichen, dass die Aufgabenstellung falsch oder unsinnig ist (dies kritisch hinterfragen!)

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \epsilon \\ 1 \end{bmatrix}$$

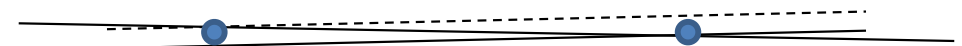
$$x = \epsilon, y = 1$$

gut konditioniert, gestrichelte linie



$$\begin{bmatrix} 0.00000001 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 + \epsilon \\ 1 \end{bmatrix}$$

$$x = 100000000\epsilon, y = 1$$



schlecht konditioniert. Kleine epsilonänderung wirft das gestrichelte komplett durch die gegend

- In beiden Fällen hat das ungestörte Problem (d.h. $\varepsilon = 0$) die exakte Lösung $x = 0$, $y = 1$
- Im ersten Fall hat das gestörte Problem die Lösung $x = \varepsilon$, $y = 1$, d.h. eine kleine Störung der Daten bewirkt eine kleine Störung des Ergebnisses, das Problem ist **gut konditioniert**
- Im zweiten Fall hat das gestörte Problem die Lösung $x = 10000000\varepsilon$, $y = 1$, d.h. eine kleine Störung der Daten bewirkt eine enorme Störung des Ergebnisses, das Problem ist **schlecht konditioniert**.

- Ein (numerischer) Algorithmus heißt **stabil** wenn die durch ihn im Laufe der Rechnung erzeugten Fehler in der Größenordnung des durch die Kondition des Problems bedingten unvermeidbaren Fehlers bleiben.
- **Kondition ist Eigenschaft des Problems**
- **Stabilität ist Eigenschaft des Verfahrens/Algorithmus**
- Wenn ein Problem schlecht konditioniert ist, kann man nicht erwarten, dass die numerische Methode (ein stabiler Algorithmus) gute Ergebnisse liefert.

- Die arithmetischen Grundoperationen sind stabil
- Aber:
Die Hintereinanderausführung stabiler Operationen ist nicht automatisch stabil
(sonst wäre ja alles stabil, was aus den Grundoperationen zusammengesetzt wird)

- Rechnung mit 3 Nachkommastellen
- Problemstellung:
 - ★ Eingabe Werte a, b
 - ★ Gesucht Nullstellen des Polynoms: $(x - a)(x - b) = 0$
- Gegeben (Input): $a = 1/100 = 0.01$, $b = -100$

Algorithmus 1:

- Ergebnis: (Trivialerweise) sind die Nullstellen a und b
- egal mit wie vielen Stellen
- Ergebnis so genau wie die Eingabe
- Resultat „akzeptabel“
- „Algorithmus“ stabil

• Algorithmus 2

- ▶ Multipliziere das Polynom aus: $p(x) := x^2 - (a+b)x + ab = 0$
- ▶ Setze a und b ein: $x^2 + 99.99x - 1 = 0$, wird gerundet zu $x^2 + 100x - 1 = 0$
- ▶ Wende Formel für Nullstellen an (mit 3-stelliger Genauigkeit)

$$\begin{aligned}
 x_{1,2} &= \frac{-100 \pm \sqrt{100^2 - 4}}{2} \\
 &= -50 \pm \frac{\sqrt{10000 - 4}}{2} \\
 &\approx -50 \pm \sqrt{10000}/2 \\
 &= \{0; -100\}
 \end{aligned}$$

- ▶ Man erhält $x_1 = 0 \neq 0.01$, also 100% Fehler
- ▶ Das ist im Sinne unserer Definition kein akzeptables Resultat
- ▶ Für $x = 0$ gilt $p(x) = -1$, also überhaupt keine Nullstelle.

also hohe pertubation des inputs

- **Fazit:**
Sogar „Selbstverständlichkeiten“

hier auslöschung möglich, es gib

z.B. die wohlbekannte Lösungsformel $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

für die quadratische Gleichung $ax^2 + bx + c = 0$

kann (für manche Eingabewerte) zu einem instabilen Algorithmus führen!

Aufgabe: Löse lineares Gleichungssystem $Ax = b$

- mit LR-Zerlegung **ohne** Pivot : evt. instabil
- mit LR-Zerlegung mit Pivot : stabil
- mit QR-Zerlegung : stabil

mind. akzeptabel

- Spezialfall:
Eingabe: reelle Zahl $x \rightarrow$ Ausgabe reelle Zahl $y = F(x)$

- **Kondition κ :**
Um welchen Faktor verstärkt sich der relative Fehler (bei exakter Rechnung).

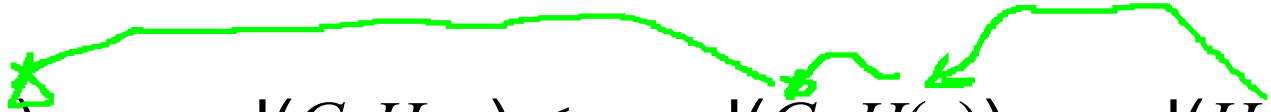
$$\frac{|\tilde{y} - y|}{|y|} \leq \kappa \cdot \frac{|\tilde{x} - x|}{|x|}$$

$$(x \approx \tilde{x}, y = F(x), \tilde{y} = F(\tilde{x}))$$

- Gilt auch allgemein für $F : \mathbb{R}^m \rightarrow \mathbb{R}^n$. Man muss nur den Betrag $|\cdot|$ durch eine Norm $\|\cdot\|$ ersetzen (z.B. Euklidische Norm)

$$\frac{\|\tilde{y} - y\|}{\|y\|} \leq \kappa \cdot \frac{\|\tilde{x} - x\|}{\|x\|}$$

- Ein Problem wird oft in mehrere Teilprobleme zerlegt
→ mehr Lösungsschritte
- Man kann die Gesamtkondition eines solchen zusammengesetzten Problems $F(x) = G(H(x)) = G \circ H(x)$ per Kettenregel auf die Kondition der Teilprobleme zurückführen:

$$\text{cond}(F, x) = \text{cond}(G \circ H, x) \leq \text{cond}(G, H(x)) \cdot \text{cond}(H, x)$$


- **Vorsicht:** Dabei kann „ \leq “ durch aus „ \ll “ sein!

Das heißt, das Gesamtproblem ist zwar gut konditioniert, aber die Teilprobleme sind schlecht konditioniert!

- Problem: Löse $Ax = b$ mit (invertierbarer) $n \times n$ –Matrix A und rechter Seite b .
- input: rechte Seite $b \rightarrow$ output: Lösung x
- Die Kondition dieses Problems ist durch die **Konditionszahl der Matrix A** gegeben:

$$\kappa(A) = \frac{\max \{ \|Ay\| : \|y\| = 1 \}}{\min \{ \|Ay\| : \|y\| = 1 \}}$$

n dim einheitskugel

input

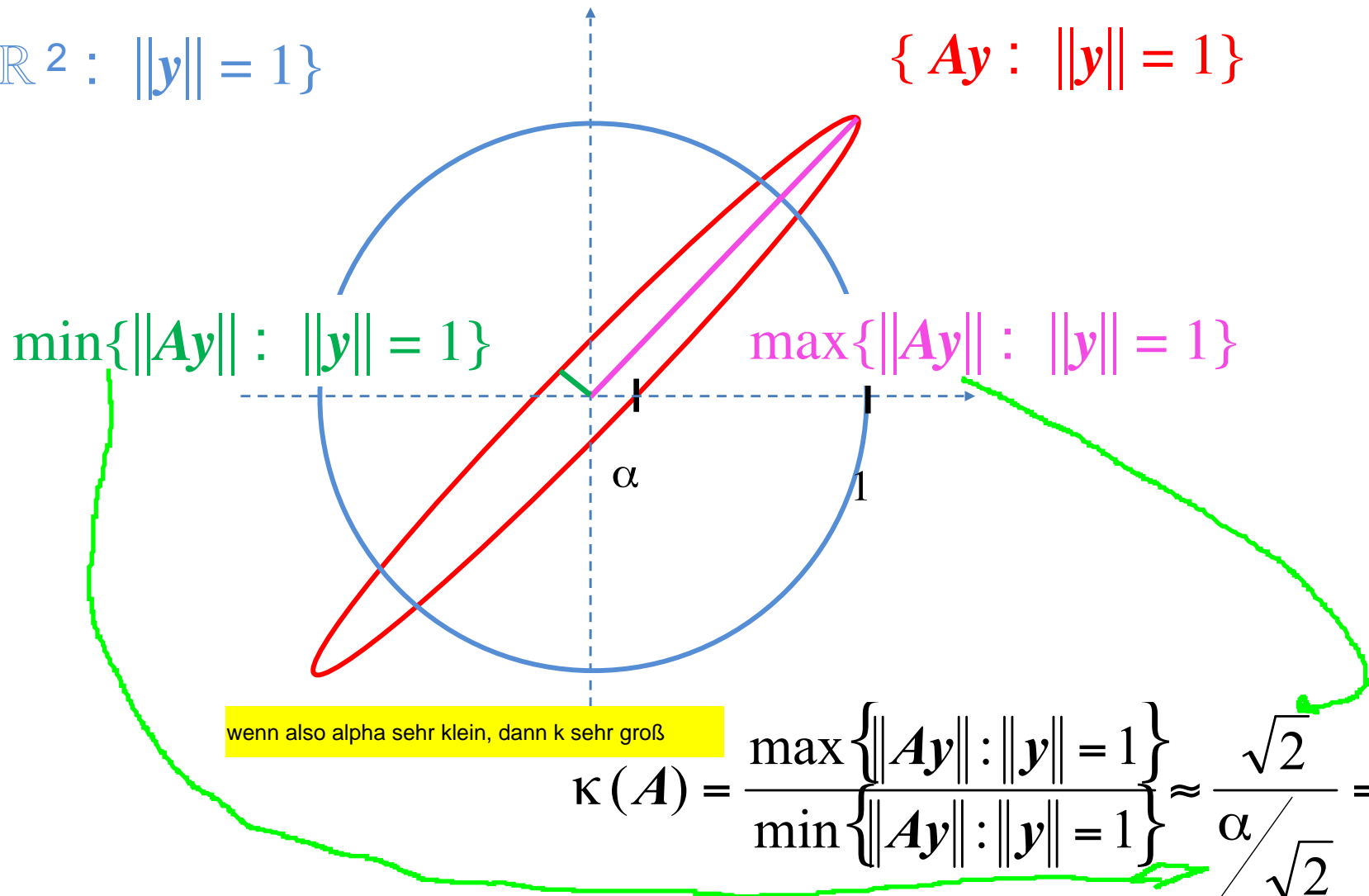
- Falls $Ax = b$ und $A\tilde{x} = \tilde{b}$ gilt dann

$$\frac{\|\tilde{x} - x\|}{\|x\|} \leq \kappa(A) \cdot \frac{\|\tilde{b} - b\|}{\|b\|}$$

- Beispiel: $n=2$, $A = \begin{bmatrix} \alpha & 1 \\ 0 & 1 \end{bmatrix}$ ($\alpha \ll 1$)

$$\{ \mathbf{y} \in \mathbb{R}^2 : \|\mathbf{y}\| = 1 \}$$

$$\{ A\mathbf{y} : \|\mathbf{y}\| = 1 \}$$



Konditionszahl: Beispiel

- Es gilt $\kappa(AB) \leq \kappa(A) \cdot \kappa(B)$ u.U. aber „ \ll “!

- Beispiel: $M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ -99 & -200 & 50 & 40 \\ 900 & 50 & -50 & -80 \\ 100 & -10 & 90 & 150 \end{bmatrix}$ hat $\kappa(M) \approx 1176.1$

- LR-Zerlegung **ohne** Pivotisierung:
 $M = L_1 R_1 : \kappa(L_1) \approx 1.108 \cdot 10^8, \kappa(R_1) \approx 7.188 \cdot 10^5$
weil die zahlen zur elimination der z.B. ersten sp

- LR-Zerlegung **mit** Pivotisierung:
 $M = L_2 R_2 : \kappa(L_2) \approx 1.1833, \kappa(R_2) \approx 1161.5$
 $\kappa(L_2 R_2) = 1.1833 \cdot 1161.5$

- QR-Zerlegung:
 $M = QR_3 : \kappa(Q) = 1, \kappa(R_3) = \kappa(M)$

QR ist ja doppelt so teuer $4/3n^3$ statt $2/3n^3$

LR:
ohne Pivot
instabil (siehe
aber Spezial-
Literatur)

mit Pivot
stabil

QR stabil

- Ungenauigkeit der Eingabewerte
 - Diskretisierungs-/Quantisierungsfehler
 - Rundungsfehler
 - Iterationsfehler
 - Linearisierungsfehler
 - Modellierungsfehler
 - ...
-
- Ist es da nicht überraschend, dass man überhaupt irgend etwas zuverlässig berechnen kann?

- Diskretisierungsfehler, Rundungsfehler
- Gleitpunktarithmetik
 - ▶ Auflösung bzw. Maschinengenauigkeit
 - ▶ Unterschiede zw. „Punkt- und Strich-Rechnung“
 - ▶ Rechengesetze gelten nur eingeschränkt
- Rundungsfehler-Fortplanzung
- Kondition und Stabilität
 - ▶ Probleme sind gut oder schlecht konditioniert
 - ▶ Algorithmen sind stabil oder instabil
- **Kondition linearer Gleichungssysteme, Konditionszahl**