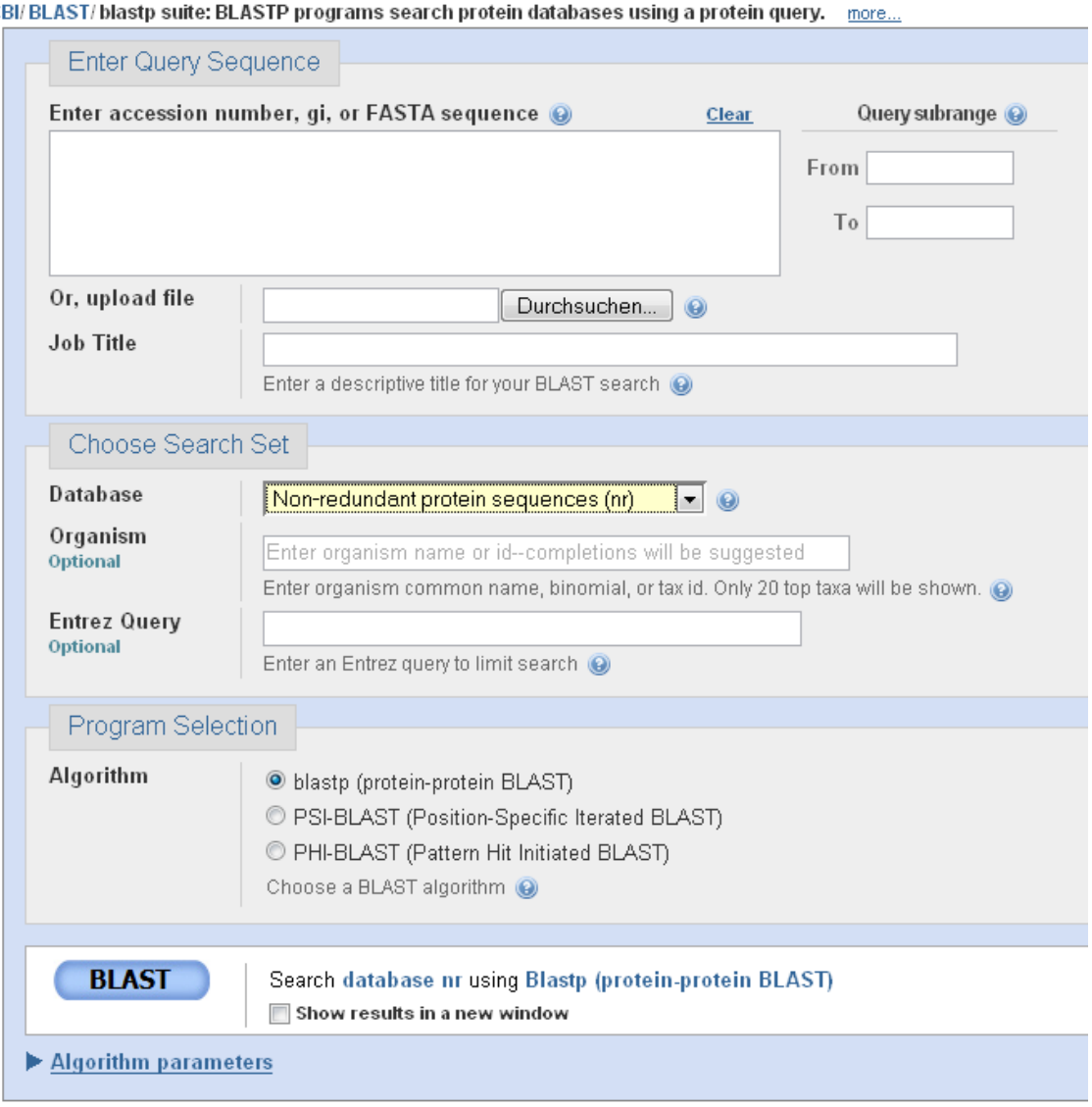
Einführung in die Angewandte Bioinformatik: BLAST E-Werte, Algorithmen, Multiple Alignments, R

17.06.2010

Prof. Dr. Sven Rahmann

Protein BLAST (blastp)



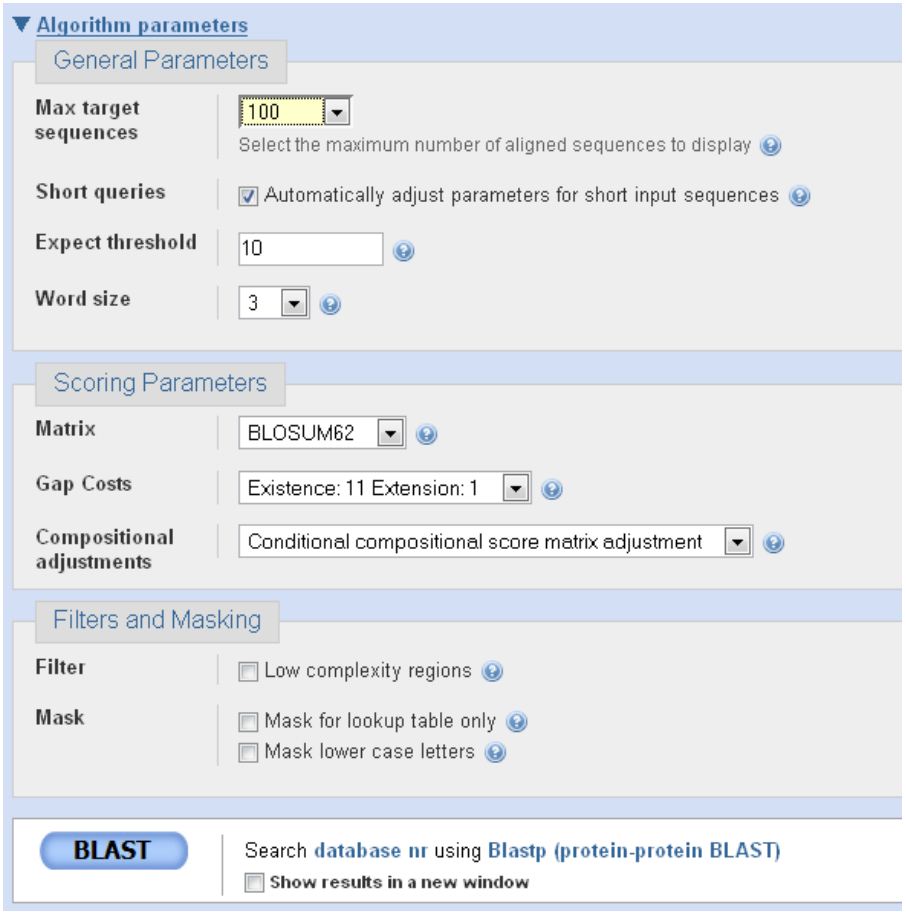
Hier wird die Sequenz eingegeben oder hochgeladen, bzw. ein Teil davon ausgewählt (subrange).

Welche Protein-Datenbank soll durchsucht werden?
Wie werden die Ergebnisse eingeschränkt?

Welche Variante des Algorithmus soll verwendet werden?

weitere Parameter (nächste Folie)

Parameter von blastp



- Wie viele Treffer maximal anzeigen?
- Für kurze Queries anpassen?
- Erwartungswert-Schwellenwert: (*)
- Wortlänge: Übereinstimmung dieser Länge mit Query ist notwendig (grösser=schneller, aber findet nicht entferntere Sequenzen)

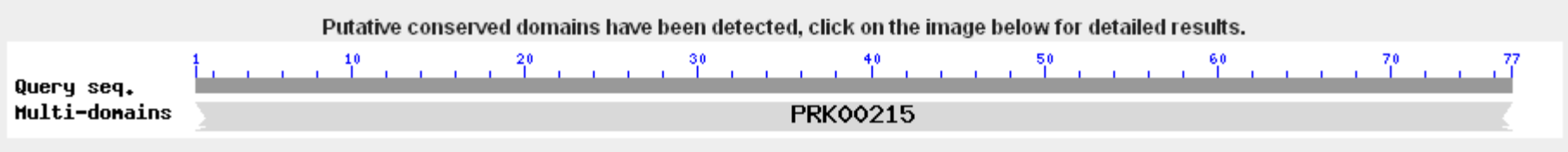
- Welche Scorematrix und Gapkosten sollen verwendet werden?
- Adjustierung: (*)

- Query filtern oder maskieren, z.B. bekannte Repeats?

Zu (*): erfordert Theorie zur BLAST-Statistik

BLAST-Ausgabe

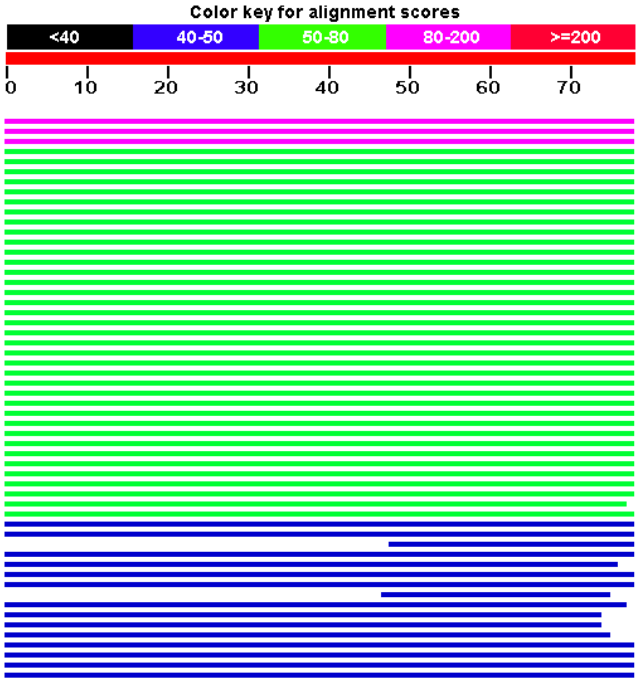
1. Während der Suche: Conserved Domains (Proteindomänen) in der Query



2. Ergebnisliste (graphisch):
beste Treffer in der Datenbank.

Ausdehnung des Balkens zeigt,
wo das ähnlichste Segment
der DB-Sequenz in der Query liegt
(lokales Alignment).

Farbe zeigt Grad der Ähnlichkeit an
(hier gemessen in Prozent Identität).



BLAST-Ausgabe

3. Ergebnisliste (Text): einzeilige Beschreibungen mit 5 Spalten

Sequences producing significant alignments:	Score (Bits)	E Value	
ref|NP_939778.1| LexA repressor [Corynebacterium diphtheriae ...	152	5e-36	G
ref|NP_738433.1| LexA repressor [Corynebacterium efficiens YS...	80.9	3e-14	G
sp|Q8FPF5|LEXA_COREF LexA repressor	80.5	3e-14	
ref|YP_001138656.1| LexA repressor [Corynebacterium glutamicu...	77.4	3e-13	G
ref|NP_601136.1| LexA repressor [Corynebacterium glutamicum A...	76.3	6e-13	G
sp|Q8NP86|LEXA_CORGL LexA repressor >dbj|BAB99323.1| SOS-resp...	76.3	7e-13	
ref|YP_250888.1| LexA repressor [Corynebacterium jeikeium K41...	63.5	4e-09	G
ref|YP_120012.1| LexA repressor [Nocardia farcinica IFM 10152...	62.8	7e-09	G
ref|YP_001800278.1| LexA repressor [Corynebacterium urealytic...	61.6	2e-08	G
ref|YP_907016.1| LexA repressor [Mycobacterium ulcerans Agy99...	61.2	2e-08	G
ref|YP_001850297.1| repressor LexA [Mycobacterium marinum M] ...	61.2	2e-08	G
ref|YP_001703770.1| Probable repressor LexA [Mycobacterium ab...	58.2	2e-07	G
ref|YP_706718.1| LexA repressor [Rhodococcus sp. RHA1] >gb|AB...	58.2	2e-07	G
ref|YP_001135215.1| LexA repressor [Mycobacterium gilvum PYR-...	57.4	3e-07	G
ref|YP_639332.1| LexA repressor [Mycobacterium sp. MCS] >ref|...	57.0	3e-07	G
ref|YP_001536300.1| transcriptional repressor, LexA family [S...	56.6	5e-07	G
ref|NP_823639.1| LexA repressor [Streptomyces avermitilis MA-...	56.6	5e-07	G
emb|CAA12169.1| LexA protein [Streptomyces clavuligerus]	56.6	6e-07	

(1) Link + Schlüssel der gefundenen DB-Sequenz
(2) Name (+ Kurzbeschreibung) der gefundenen DB-Sequenz
(3) Alignment-Score (in Bits [d.h. Logarithmus zur Basis 2 wurde verwendet])
(4) der E-Wert (5e-36 bedeutet $5*10^{-36}$ und gibt die Anzahl der Treffer an, die man rein zufällig mit diesem oder besserem Score **erwarten** würde)
(5) Links zu anderen NCBI-Datenbanken (z.B. G: Gene, S: Structures)

BLAST-Ausgabe

4. Detaillierte Beschreibungen:
insbesondere Alignments & deren Qualität:

- identities, positives, gaps
- Alignment-Score in bits (80.9)
- Alignment-Score in Rohform (198)
- E-Wert (Expect-Wert, $3*10^{-14}$)
- Methode, mit der der E-Wert berechnet wurde

```
>ref|NP_738433.1|  LexA repressor [Corynebacterium efficiens YS-314]
 dbj|BAC18633.1|  putative SOS response repressor LexA [Corynebacterium efficiens
YS-314]
Length=269

 GENE ID: 1034469 CE1823 | LexA repressor [Corynebacterium efficiens YS-314]
(10 or fewer PubMed links)

 Score = 80.9 bits (198),  Expect = 3e-14, Method: Compositional matrix adjust.
 Identities = 43/80 (53%), Positives = 55/80 (68%), Gaps = 6/80 (7%)

Query  1    RDPNKPRAVDVRALPDPIPSKPGRKPGPKKS---SVAISPDPAETSPTSFVPIVGSIAAG  57
            RDPNKPRAVDVR LP+   +    K GPK     +   SP P   S TSF+P+VG IAAG
Sbjct  103  RDPNKPRAVDVRHLPE---TDNRTKAGPKAKARPTAGASPQPELASSTSFIPVVGKIAAG  159

Query  58   NPILAEENVDGYFPFPSEIV  77
            +PILAE+N++ Y+P P++IV
Sbjct  160  SPILAEQNIEEYYPLPADIV  179
```

BLAST-Statistik

```
>ref|NP_738433.1| LexA repressor [Corynebacterium efficiens YS-314]
 dbj|BAC18633.1| putative SOS response repressor LexA [Corynebacterium efficiens
YS-314]
Length=269

 GENE ID: 1034469 CE1823 | LexA repressor [Corynebacterium efficiens YS-314]
(10 or fewer PubMed links)

 Score = 80.9 bits (198),  Expect = 3e-14, Method: Compositional matrix adjust.
 Identities = 43/80 (53%), Positives = 55/80 (68%), Gaps = 6/80 (7%)

Query  1    RDPNKPRAVDVRALPDPIPSKPGRKPGPKKS---SVAISPDPAETSPTSFVPIVGSIAAG  57
            RDPNKPRAVDVR LP+   +    K GPK     +   SP P   S TSF+P+VG IAAG
Sbjct  103  RDPNKPRAVDVRHLPE---TDNRTKAGPKAKARPTAGASPQPELASSTSFIPVVGKIAAG  159

Query  58   NPILAEENVDGYFPFPSEIV  77
            +PILAE+N++ Y+P P++IV
Sbjct  160  SPILAEQNIEEYYPLPADIV  179
```

Score in Rohform (198):

Summe der Scores über Alignment-Spalten

Interpretationsproblem der Roh-Scores:

- Längere Queries haben höhere Chance, gutes lokales(!) Alignment zu bekommen.
- Je länger die Sequenzen, desto länger „zufällige“ lokale Ähnlichkeiten.
- Unklar, auf welcher Skala der Roh-Score angegeben wird.

Bit-Score (80.9):

längennormalisiert, genormt (Log-odds zur Basis 2)

Interpretationsproblem des Bit-Scores:

- Wann ist ein Bit-Score wirklich (sehr) hoch ist?

E-Wert ($3*10^{-14}$):

Lösung der Interpretationsprobleme

Blast-Statistik: E-Wert

Definition (E-Wert):
Wie viele Treffer **mit diesem oder höherem** Score
erwartet man aufgrund zufälliger Sequenzähnlichkeiten?
(mit zufälliger Query gleicher Länge in zufälliger Datenbank gleicher Größe)

Kleiner E-Wert (z.B. $< 10^{-10}$) heißt:
- unwahrscheinlich, dass beobachtete Ähnlichkeit
  auf einem nur zufällig ähnlichen Stück Sequenz beruht.
- plausibel, dass eine evolutionäre Verwandtschaft vorliegt.

Hoher E-Wert (nahe bei oder größer als 1) heißt:
- Such-Treffer lässt sich durch Zufall erklären,
  sollte nicht notwendigerweise als biologisch relevant gelten.

Berechnung des E-Werts

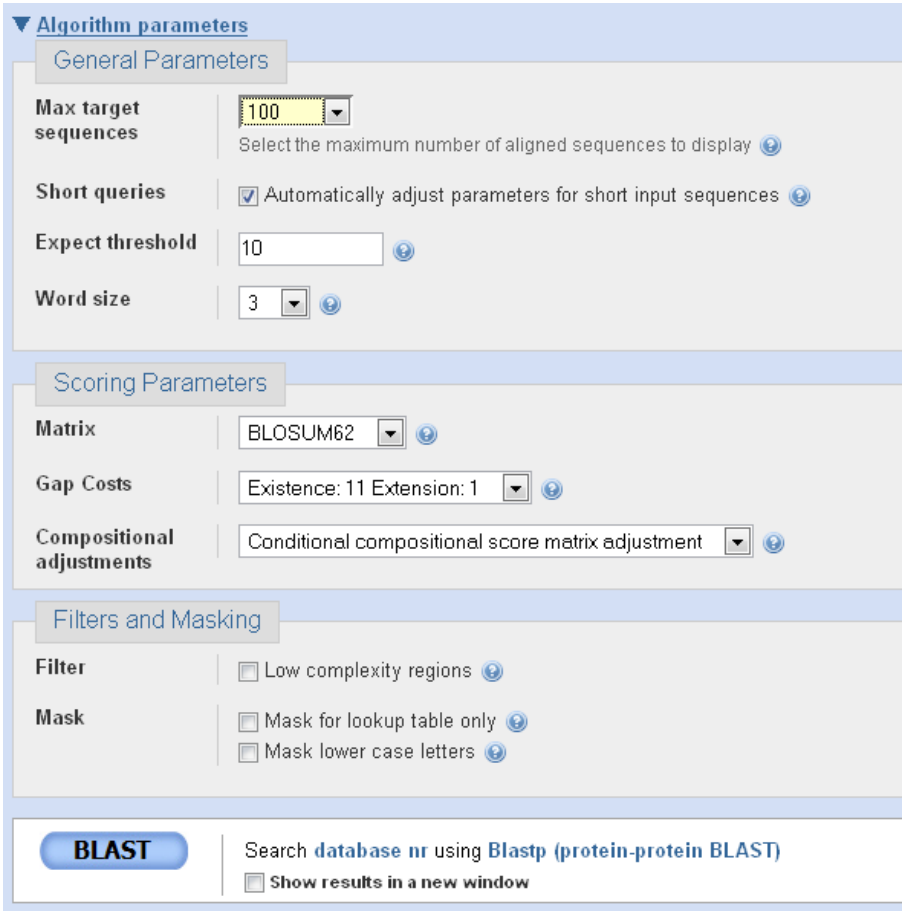
E-Wert (erwartete Anzahl Treffer mit diesem oder besserem Score) hängt ab von:
- Score s
- Querylänge m
- Datenbanklänge n (Gesamtlänge)
- Scorematrix und Gapkosten

E-Wert lässt sich mathematisch berechnen und approximieren durch

$$E(s) \approx Kmn\, e^{-\lambda s}.$$

Dabei hängen $K>0$, $\lambda>0$ von der Scorematrix und den Gapkosten ab.

Verbleibende Parameter von blastp



- **Erwartungswert-Schwellenwert:**
  Ausgabe nur von Treffern mit Score s,
  so dass $E(s) <$ Schwellenwert.
  Kleinerer Schwellenwert:
  schneller, nur nah verwandte Treffer,

- **Adjustierung:**
  E-Wert Berechnung geht von Zufälligkeit von Query und Datenbank aus.
  Query ist aber bei der Suche fest.
  Wenn die Aminosäure-Zusammensetzung der Query von der „typischen“ abweicht, sind die E-Werte unpassend.
  Daher: Adjustiere Parameter K, λ
  im Hinblick auf Query-Zusammensetzung.

E und e

- E-Wert: E heißt Erwartung
- 5E-02 = 5e-02 = 0.05: E=e heißt Exponent
  (3.2e+18 = $3.2 \cdot 10^{18}$, **nicht** $3.2 \cdot e^{18}$)
- e in $e^{-\lambda s}$: e heißt Eulersche Zahl (ca. 2.71)

Grundlegendes zu Algorithmen

Problem – Algorithmus – Programm

Typisches Vorgehen in der Informatik:

1. Spezifikation eines **Problems**
2. „Erfinden“ eines **Algorithmus** (Methode), der das Problem löst
3. Umsetzung („Implementierung“) des Algorithmus als **Programm** in einer Programmiersprache (z.B. C, C++, Java, Perl, Python, R, ...)

Zu jeden Problem gibt es i.d.R. mehrere Algorithmen,
die unterschiedlich gut (schnell oder elegant oder einfach) sein können.

Zu jedem Algorithmus gibt es i.d.R. mehrere mögliche Implementierungen.

Problem – Algorithmus – Programm: Details

1. Spezifikation des **Problems** durch:
- Was ist gegeben?
- Was ist gesucht?
- Welche Bedingungen / Regeln sind einzuhalten?

2. „Erfinden“ eines **Algorithmus**, der das Problem löst.
- Algorithmus gibt endliche Abfolge von auszuführenden Schritten an.
- Beweis (math.), dass die angegebene Schrittfolge das Problem löst.
- Aussagen zur Dauer („Laufzeit“) bis zur Lösung.

3. Umsetzung („Implementierung“) des Algorithmus als **Programm**
in einer bestimmten formalen Sprache (z.B. C, C++, Java, Perl, Python, R, ...)

Zum Algorithmus-Begriff

Schwierig, den Begriff Algorithmus exakt zu formalisieren.
Berechenbarkeitsmodelle: Turing-Maschinen, Registermaschinen, Lambda-Kalkül...

Wichtig sind folgende Eigenschaften eines Rechenverfahrens:

1. mit einem endlichen Text eindeutig beschreibbar (Finitheit).
2. jeder Schritt ausführbar (Ausführbarkeit)
3. stets nur endlich viel Speicher (dynamische Finitheit)
4. nur endlich viele Schritte (Terminierung)
5. gleiches Ergebnis bei gleichen Voraussetzungen (Determiniertheit).
6. nächster Schritt ist stets eindeutig definiert (Determinismus).

Algorithmus leitet sich aus dem Namen
Abu Abdallah Muhammad ibn Musa al-Chwarizmi (أبو عبد الله محمد بن موسى الخوارزمي)
ab; vermutlich iranischer Mathematiker (* um 780; † zwischen 835 und 850).

Problem – Algorithmus – Programm: Beispiel ggT (größter gemeinsamer Teiler)

1. Spezifikation des **Problems:**
- Was ist gegeben? zwei positive ganze Zahlen x, y.
- Was ist gesucht? ihr größter gemeinsamer Teiler.
- Welche Bedingungen? Was ist (gemeinsamer) Teiler; was heißt größer?

2. „Erfinden“ eines **Algorithmus**, der das Problem löst:
- Sei m das Minimum von x und y
- Sei $g := 1$
- Für jede Zahl z von 2 bis m aufsteigend nacheinander:
  Wenn x/z und y/z beide ganzzahlig sind, setze $g := z$
- Gib g aus

Der Algorithmus ist „offensichtlich“ korrekt. (Wirklich? Warum?)
Aber er ist nicht sehr schnell: Man testet $m = \min(x,y)$ Zahlen.
Besser: Euklidischer Algorithmus (Mathe-Vorlesung).

Problem – Algorithmus – Programm: Beispiel ggT (größter gemeinsamer Teiler)

3a. Implementierung in Python:

```
def ggt(x,y):
    """groesster gemeinsamer Teiler von x und y."""
    m = min(x,y)
    g = 1
    for z in range(2,m+1):
        if (x//z)*z == x and (y//z)*z == y:
            g = z
    return g
```

3b. Implementierung in R:

```
ggt = function(x,y) {
  m = min(x,y)
  g = 1
  for (z in 2:m) {
    if (floor(x/z)*z==x && floor(y/z)*z==y) {
      g=z
    }
  }
  g
}
```

Problem – Algorithmus – Programm: Beispiel globales paarweises Sequenzalignment

1. Spezifikation des Problems:

Gegeben:
- zwei Sequenzen über einem Alphabet (i.d.R. Nukleotide oder Aminosäuren),
- eine Scorematrix zu diesem Alphabet,
- Gapkosten (welcher Art? linear, affin, beliebig?)

Gesucht:
Score eines optimalen globalen Alignments; optimales Alignment dazu.

Regeln:
- Alignment heißt: (siehe Definition von paarweisem Alignment)
- optimal heißt: höchstmöglicher Score
- Score eines Alignments berechnet sich als Summer über alle Spalten
- Score einer Spalte ergibt sich aus Scorematrix oder Gapkosten

Problem – Algorithmus – Programm: Beispiel globales paarweises Sequenzalignment

2. Ein Algorithmus („naiv“):

- Für alle möglichen Alignments der Sequenzen:
  - Berechne Score des Alignments.
  - Wenn Score besser als bisher bester Score, merke Score und Alignment.
- Gib besten gemerkten Score und Alignment aus.

Problem dabei:
Zu zwei Sequenzen der Länge n gibt es ca. 4^n mögliche Alignments.
(Die meisten haben einen niedrigen Score und sind irrelevant,
werden aber dabei trotzdem überprüft.)

Der Needleman-Wunsch-Algorithmus („needle“) löst das Problem schneller,
indem sukzessiv längere Präfixe der Sequenzen betrachtet werden.

Laufzeit von Algorithmen: O-Notation
Korrektheit von Algorithmen („exakte Algorithmen“)

Laufzeit des Needleman-Wunsch-Algorithmus

Es werden $(|s|+1) * (|t|+1)$ Werte berechnet.
Jede Berechnung braucht konstante Zeit (Maximierung über 3 Werte)
Insgesamt: $O(st)$ Zeit.

O-Notation: Angabe von Funktionen unter Vernachlässigung von
- konstanten Faktoren
- „kleineren“ (d.h. langsamer wachsenden) Termen

Definition [exakte Lösung in Zeit „groß-O von $f(n)$“]
Ein Algorithmus **löst** ein gegebenes Problem der Größe n
exakt in **Zeit $O(f(n))$**, wenn
- garantiert die korrekte Lösung berechnet wird („Korrektheit“)
- dies immer höchstens $c * f(n)$ Rechenschritte benötigt (c konstanter Faktor).

Beispiel zur O-Notation

Angenommen, es gibt zwei Sortieralgorithmen, die jeweils
- $n^2/2 - n \log n + n/3$ Vergleiche
- $3n \log n$ Vergleiche

machen, um n Zahlen zu sortieren.

Es ist $n^2/2 - n \log n + n/3$ in $O(n^2)$, oder auch in $O(n^{17})$.
Es ist $3n \log n$ in $O(n \log n)$, oder auch in $O(n^2)$, aber nicht in $O(n)$.

Für genügend große n ist der zweite Algorithmus immer schneller.
Für kleine n (bis wohin?) ist der erste Algorithmus schneller.
O-Notation betrachtet nur Verhalten für große n.

Heuristiken

Korrekte („exakte“) Algorithmen garantieren optimale Lösung.
Nicht immer existieren hinreichend schnelle korrekte Algorithmen!
Beispiel gleich: multiples Sequenzalignment

=> Verwendung von **Heuristiken** (gr. *heuriskein*, „(auf-)finden“, „entdecken“)
(Kunst, mit begrenztem Wissen und wenig Zeit zu guten Lösungen zu kommen.)

(Qualitäts-)Heuristik:
- garantierte (schnelle) Laufzeit.
- gefundene Lösung nicht notwendig optimal.

Laufzeit-Heuristik:
- löst das Problem in jedem Fall exakt.
- versucht, durch „Abkürzungen“ die Lösung möglichst schnell zu finden.

Beispiel zu Heuristiken: Smith-Waterman vs. BLAST

Der **Smith-Waterman-Algorithmus** ('water' bei EMBOSS)
löst Problem des lokalen Sequenz-Alignments von s,t **exakt in $O(|s|\,|t|)$** Zeit.
Ist t eine große Datenbank -> mehrere Minuten bis Stunden; zu langsam.

BLAST ist eine **Heuristik** für dasselbe Problem.
Bei unähnlichen Sequenzen wird evtl. nicht das optimale Alignment gefunden.
In der Praxis trotzdem sehr gut, jedoch keine Garantie.
Laufzeit nur etwa $O(|s| + |t|)$.

Für Datenbanksuche wird ausschließlich BLAST verwendet.
Für Alignment einzelner Sequenzen wird Smith-Waterman verwendet.

Multiple Alignments

Bisher: Paarweise Alignments

Optimales Alignment:

- Alignment mit höchstem Score unter allen Alignments
- arbeitet die Ähnlichkeiten zwischen zwei Sequenzen heraus
- Spaltentypen: Match/Mismatch, Insertion, Deletion mit Scores bzw. Kosten

Wichtig für:

- quantitative Bestimmung von Sequenzähnlichkeit
- Übertragen von Informationen (Struktur/Funktion) zw. ähnlichen Sequenzen

Aber:

Proteinfamilien oder Domänen bestehen aus mehr als 2 Sequenzen!

Erinnerung: Proteindomänen und Proteinfamilien (Pfam)

Domänen

sind wiederkehrende modulare Bausteine von Proteinen.
Durch verschiedene Kombinationen von Domänen
entstehen Proteine mit unterschiedlichen Eigenschaften.
Ziel: alle existierenden Domänen katalogisieren, analysieren

Proteinfamilien

Eine Domäne oder eine bestimmte Kombination von Domänen
kann eine bestimmte Familie von Proteinen charakterisieren.

Datenbanken zu Domänen

Pfam:	http://pfam.sanger.ac.uk/	protein families
SMART:	http://smart.embl-heidelberg.de/	simple modular architecture research tool

Modellierung von Proteindomänen

Wie kann man eine Domäne beschreiben?

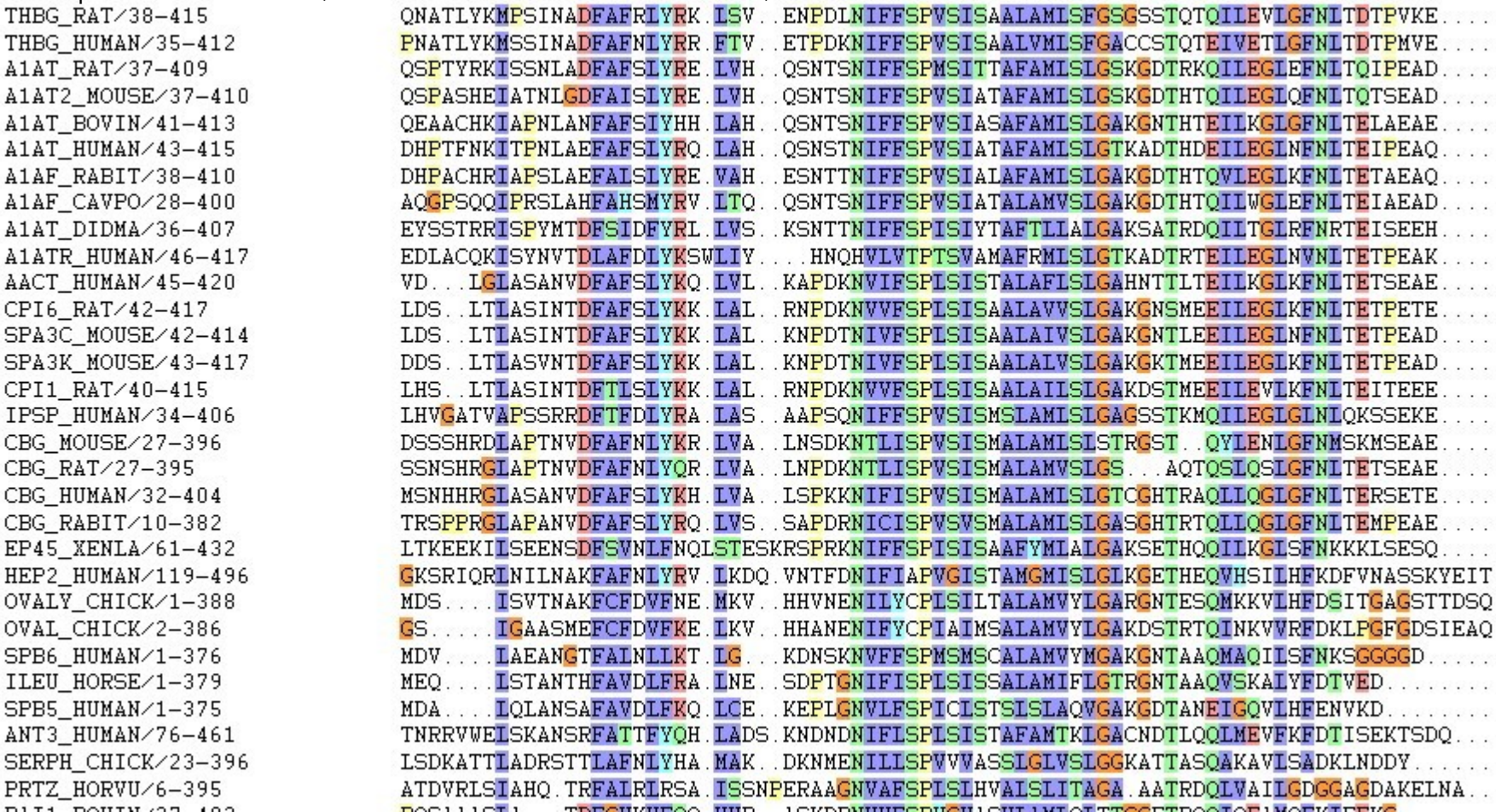
- Aminosäuresequenz (Konsensus + Variationsmöglichkeiten)
  Sequenz angeben, evtl. mehrere Symbole pro Position
  (nicht sehr nützlich wg. Variationen)

- statistisches Modell (Hidden-Markov Model, HMM)

- **multiples Alignment** aus bekannten Beispiel-Sequenzen

Beschreibung durch Multiples Alignment

Serpin-Domäne (Serin Protease Inhibitor)



Multiple Alignments

Motivation

- Möchte alle Mitglieder einer Proteinfamilie auf einmal betrachten, Gemeinsamkeiten / Unterschiede **auf einen Blick** sehen.
- Homologe (evolutionär sich entsprechende) Positionen besser sichtbar.
- Drei optimale paarweise Alignments von drei Sequenzen evtl. inkonsistent: *a* mit *b* aligniert, *b* mit *c*, aber *a* nicht mit *c*.
  In multiplem Alignment unmöglich: Alle Sequenzen gleichzeitig aligniert!

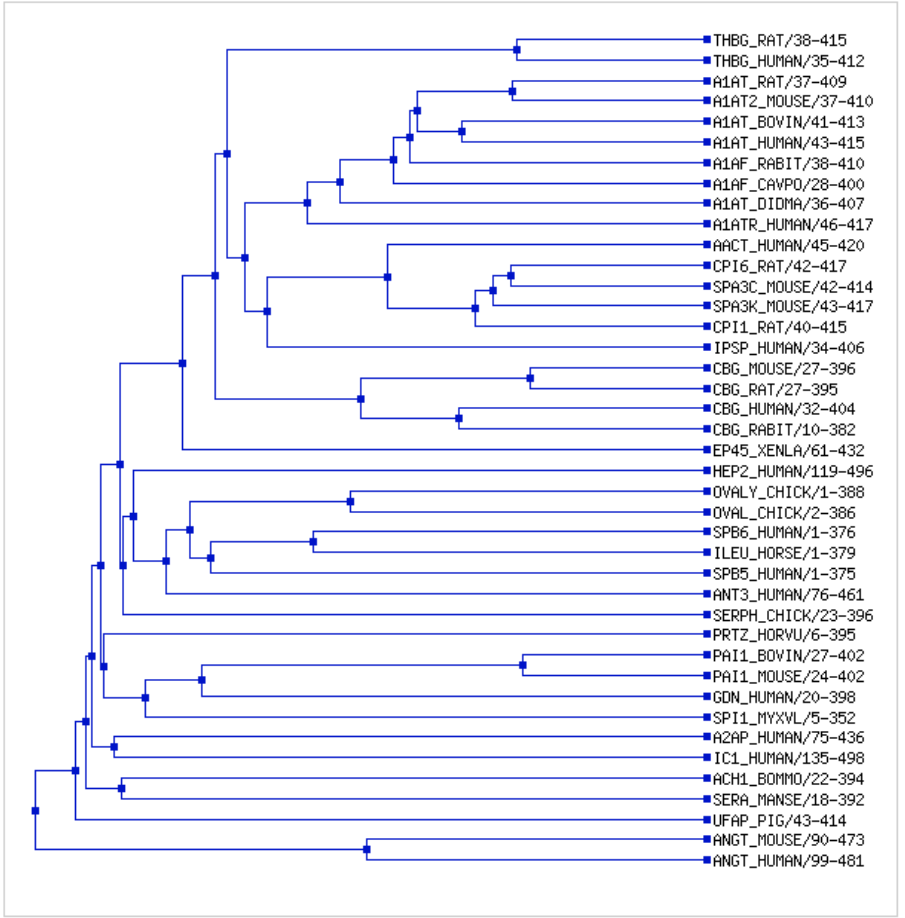
Multiples Alignment :=
Alignment von mindestens 3 Sequenzen.
Jede Zeile entspricht (durch Weglassen der Gaps) einer der Sequenzen.
Bei k Sequenzen kann jede Spalte 0 bis k-1 Gap-Zeichen enthalten.

Wann sind multiple Alignments sinnvoll?

Globales multiples Alignment nur sinnvoll, wenn
- Sequenzen global ähnlich,
- alle Sequenzen evolutionär verwandt

Lokales multiples Alignment sinnvoll:
- hinreichend lange gemeinsame ähnliche Teilsequenz aller Sequenzen (Domäne?)
- diese Bereiche evolutionär verwandt

Verwandtschaftsbeziehungen werden durch **phylogenetischen Baum** beschrieben.
Beispiel: Serpine in Pfam



Berechnung multipler Alignments

Ziel:
Biologisch / evolutionär korrektes multiples Alignment
(evolutionär voneinander abstammende Aminosäuren,
oder solche mit gemeinsamem Vorfahren, stehen untereinander)

Formulierung als Optimierungsproblem:
Definiere (wie schon auf paarweisen Alignments) eine Score-Funktion.
Finde das multiple Alignment, das den Score maximiert.

Problematisch:
Es gibt vermutlich kein Scoring-Verfahren,
das stets das evolutionär korrekte Alignment
zu dem mit dem höchsten Score macht (Beispiel).

Scoring von multiplen Alignments

Sum-of-pairs Score:
Multiples Alignment aus k Sequenzen enthält ~ $k^2/2$ paarweise Alignments.
Summe aller paarweisen Scores ergibt den Score des multiplen Alignments.

Tree score:
Annahme: Zwischen Sequenzen bestehen evolutionäre Verwandtschaften, gegeben durch phylogenetischen Baum.
Summiere paarweise Scores von im Baum benachbarten Sequenzen.

gewichteter Sum-of-pairs Score
wie Sum-of-pairs Score, aber jedes Paar erhält individuelles Gewicht

3 Optimierungsprobleme beim multiplen Alignment

Sum-of-pairs Problem

Gegeben *k* Sequenzen, Scorematrix, Gapkosten.
Finde multiples Alignment, das (gewichteten) sum-of-pairs Score maximiert.

Tree Alignment Problem

Gegeben zusätzlich ein Baum mit den *k* Sequenzen an den Blättern.
Finde Belegung der inneren Knoten mit Sequenzen (gemeinsame Vorfahren)
und multiples Alignment, das den Tree Score maximiert.

Verallgemeinertes Tree Alignment Problem

Gegeben *k* Sequenzen, Scorematrix, Gapkosten (kein Baum!),
finde Baumtopologie, Belegung der inneren Knoten mit Sequenzen
und multiplea Alignment, das den Tree Score maximiert

Komplexität des multiplen Alignment - Problems

Für alle drei Varianten des Problems
- Sum-of-pairs Problem
- Tree Alignment Problem
- Verallgemeinertes Tree Alignment Problem

gilt:

Es gibt exakte Algorithmen,
aber Zeitbedarf exponentiell in der Anzahl der Sequenzen k.
Nur praktikabel für 3 – 7 Sequenzen.

Verwendung von Heuristiken evtl. „nicht so schlimm“, denn:
Score-maximales Alignment ungleich biologisch korrektes Alignment.

Heuristiken für multiples Alignment

Center-star-Methode:

- Wähle Sequenz (mit geringster evolutionärer Abstandssumme zu den anderen).
- Aligniere jede andere Sequenz paarweise daran.
- Setze k-1 paarweise Alignments zu einem multiplen Alignment zusammen.
- Nachteil: Es werden nur k-1 der möglichen paarweisen Alignments betrachtet.

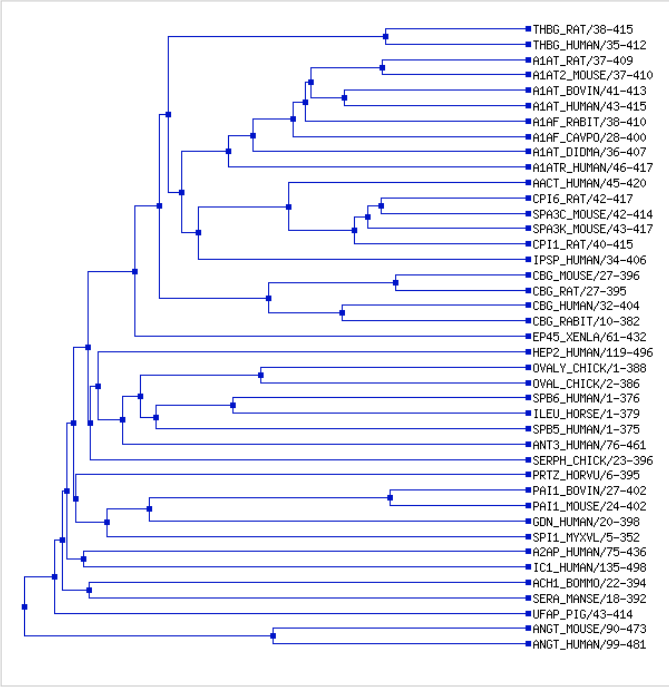
Divide-and-conquer Alignment:

- Suche etwa in der Mitte jeder Sequenz gut konservierte Stellen,
- an der man relativ sicher ist, dass alle diese Stellen eine Alignmentspalte bilden.
- Teile Problem in linkes und rechtes Problem auf, verfahre dort genauso.
- Nachteil: funktioniert nicht gut, wenn es keine solchen Stellen gibt.

Heuristiken für multiples Alignment

Progressives Alignment:

- Beginne mit zwei nah verwandten Sequenzen
- Aligniere in jedem Schritt eine weitere Sequenz optimal paarweise an bestehendes multiples Alignment.
- Reihenfolge durch evolutionäre Distanzen der Sequenzen bestimmt (Baum).
- Nachteil: falsche frühe Entscheidungen können nicht rückgängig gemacht werden.
- Wichtigstes Beispiel: **Clustal**



Clustal (http://www.clustal.org)

- schnelle Heuristik (und Software-Paket) für multiples Alignment

Webserver zum Berechnen von Alignments:
- am EBI: http://www.ebi.ac.uk/Tools/clustalw2/index.html
- am SIB: http://www.ch.embnet.org/index.html

Idee und Verfahren
- Berechne eine Folge von paarweisen Alignments
- Nimm dazu einen Baum zur Hilfe („guide tree“)
  (Art phylogenetischer Baum, aber eher ein Hilfsmittel zur Berechnung).
- Bereits existierende Teil-Alignments werden dabei nicht mehr verändert.

1. Berechne Distanzen zwischen allen Sequenzpaaren
2. Berechne aus den Distanzwerten einen Baum (mehr dazu später)
3. Aligniere Sequenzen und existierende Alignments
   in der Reihenfolge, die der Baum vorgibt (bottom-up).

Abhängigkeit von Baum und Alignment

Baum („guide tree“) bildet Verwandtschaftsverhältnisse der Sequenzen ab.
Wichtiges Hilfsmittel beim Berechnen des multiplen Alignments.
Woher bekommen?

Man schätzt evolutionäre Distanzen (z.B. in PAM) zwischen Sequenzen.
Dazu braucht man aber das Alignment.

Henne-Ei-Problem:

- Berechnung des Baums benötigt Alignment.
- Berechnung des Alignments benötigt Baum.

Abhängigkeit von Baum und Alignment

Lösung des Henne-Ei-Problems:

Beginne mit (groben) Schätzungen für Distanzen,
z.B. aus paarweisen Alignments (unterschätzen wahren Distanzen).
Berechne ersten Baum.
Erstelle erstes multiples Alignment.

Schätze daraus neue Distanzen, neuen Baum, neues Alignment.

Iteriere so lange, bis sich nichts mehr ändert,
oder eine Maximalzahl an Iterations-Schritten gemacht wurde.

Ein wenig mehr R

Wichtige Verteilungen und ihre Eigenschaften

Stetige Verteilungen

norm: Normalverteilung
unif: Gleichverteilung (Uniform-Verteilung)
exp: Exponentialverteilung

Diskrete Verteilungen

binom: Binomialverteilung
pois: Poisson-Verteilung

Funktionen auf Verteilungen

d... Dichte
p... kumulative Verteilungsfunktion
q... Quantile
r... Zufallszahlen

Dichte und kumulative Wahrscheinlichkeiten

Dichte d____(x)
gibt an, wie wahrscheinlich Werte aus der Verteilung in einem Bereich liegen
W-keit, dass zufälliger Wert X (aus Verteilung) in]a,b] liegt, ist:
Integral von a bis b über die Dichte
z.B. dnorm(x) – Dichte der Standard-Normalverteilung an der Stelle x

kumulative Wahrscheinlichkeitsfunktion p____(q)
Integral (von minus unendlich) bis q über die Dichte
W-keit, dass zufälliger Wert (aus Verteilung) <= q ist
z.B. pnorm(x) – Wahrscheinlichkeit, dass normalverteilte Zahl <= x ist

Beispiel Normalverteilung (Gauss-Verteilung)

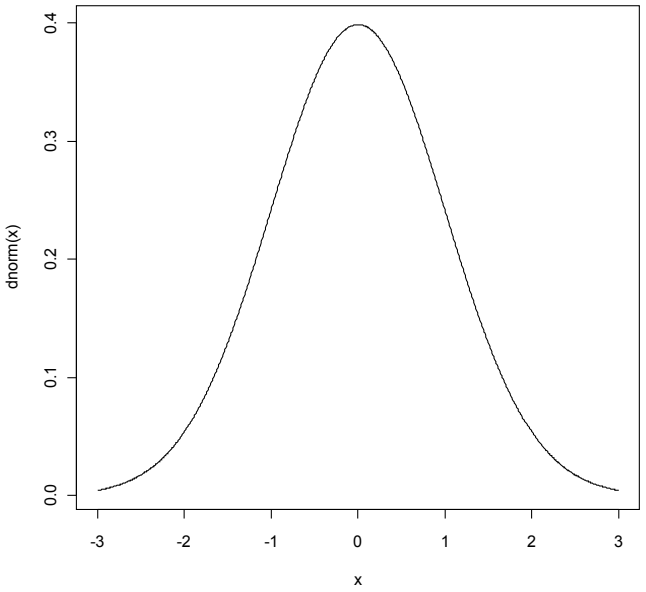
Charakterisiert durch (Mittelwert, Varianz), z.B. bei Fehlern in Messdaten.
Typisch: Geringe W-keit, stark vom Mittelwert abzuweichen („fällt schnell“)

Dichte (dnorm) plotten

```
x = seq(-3,3,by=1/128)
plot(x,dnorm(x), type="l")
```

Alternative

```
curve(dnorm,from=-3,to=3)
```



Beispiel Normalverteilung (Gauss-Verteilung)

Charakteristisch für die Normalverteilung

- Symmetrie um den Punkt maximaler Dichte: Median = Mittelwert
- Konzentration der Dichte um den Erwarungswert („fällt schnell“)

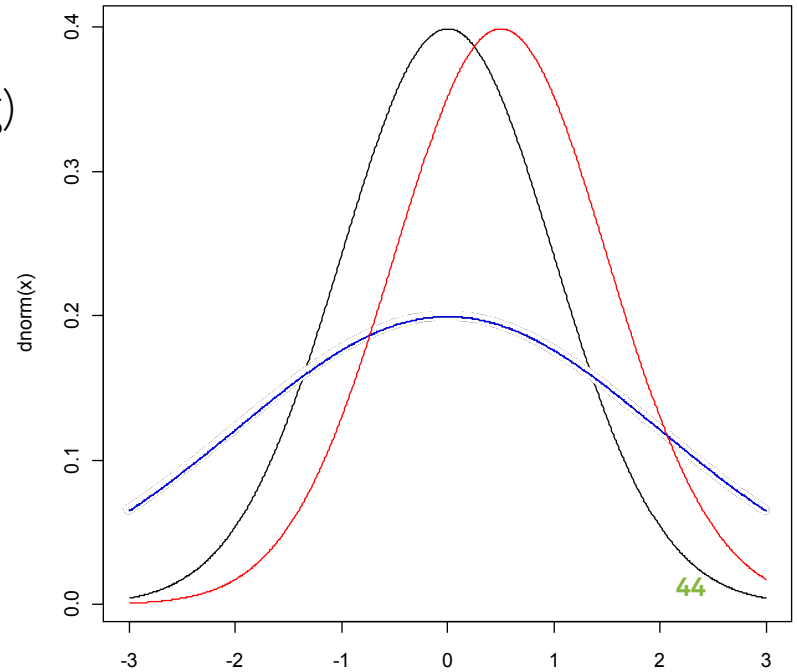
Zwei Parameter

- Mittelwert: beschreibt Lage
- Standardabweichung: beschreibt Skala
  (Varianz: Quadrat der Standardabweichung)

Standardnormalverteilung

Mittelwert 0
Standardabweichung 1

Parameter kann man übergeben an
`dnorm(mean=..., sd=...)`.



Quantile

Quantile von Verteilungen

Bisher: Quantie aus Daten
p-Quantil: Wert x, so dass Anteil p der Daten <= x ist
Jetzt: Quantile für Verteilungen
p-Quantil: Wert x, so dass W-keit p besteht, dass zufällige Zahl <=x ist.

Beispiel

```
qnorm(0.5) = 0    # Median liegt bei 0
```

Verschiebung und Skalierung

Stammt x aus Standardnormalverteilung (mean=0, sd=1)
so verhält sich ax+b wie aus Normalverteilung mit sd=a, mean=b.

Quantil-Quantil-Plot

Stammen zwei Datensätze aus der gleichen Verteilung?
Wenn ja, müssen auch die Quantile (fast) gleich sein.
Wenn die Verteilungen sich nur in Lage (mean) und Skala (sd) unterscheiden, sind auch die Quantile entsprechend skaliert.

qq-Plot

plottet Quantile einer Verteilung gegen Quantile anderer Verteilung.

gleiche Verteilung:	Gerade mit Steigung 1 durch 0
verschobene Verteilung:	Gerade mit Steigung 1 durch ??
skalierte Verteilung:	Gerade mit Steigung ?? durch 0
verschobene skalierte Verteilung:	Gerade
andere Verteilung:	keine Gerade!

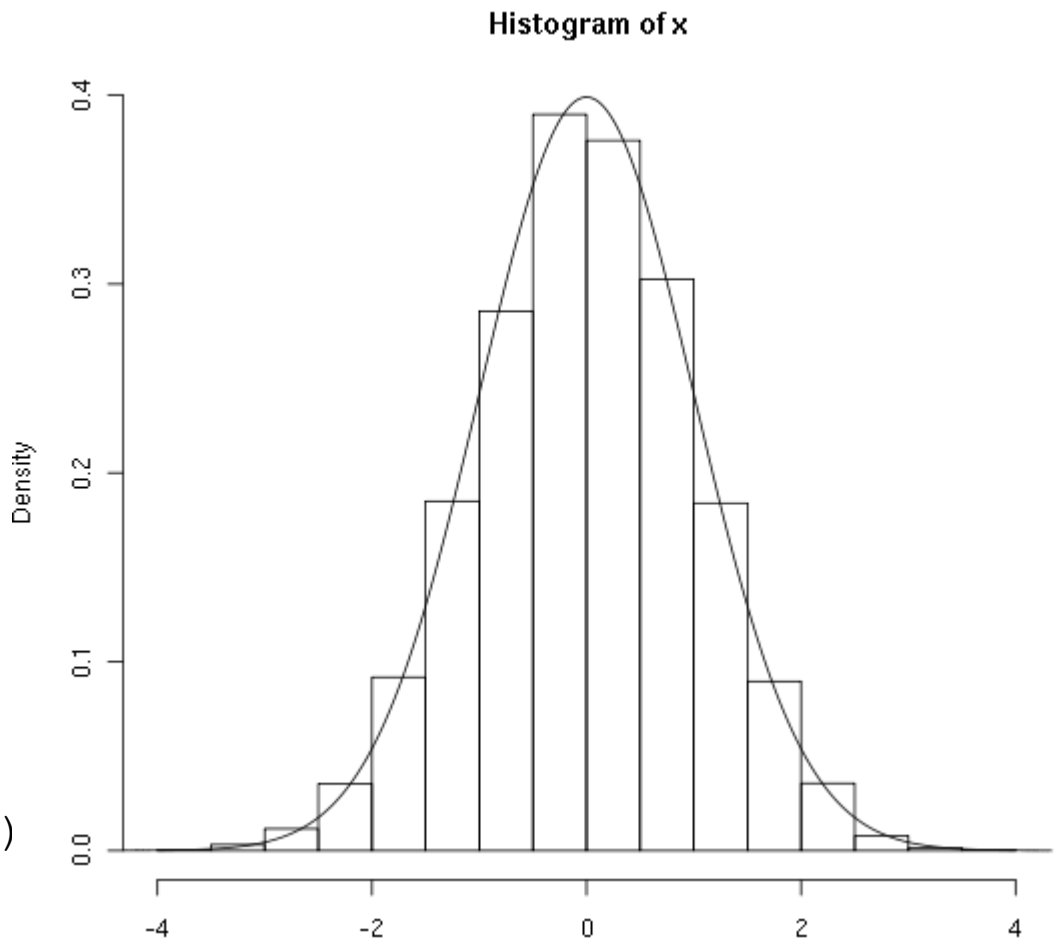
Zufallszahlen

Ziehe n Zufallszahlen
Aufruf: r____(n);
evtl. weitere Parameter angeben
Beispiel

```
x=rnorm(100, mean=5,sd=3)
```

Histogramm vieler Zufallszahlen rekonstruiert Dichte

```
x = rnorm(10000)
hist(x)
z = seq(-5,5,1/128)
points(z,dnorm(z),type="l")
```



qq-Plots mit Zufallszahlen aus der gleichen Verteilung

Experiment

Ziehe 2x aus der gleichen Verteilung viele Zufallszahlen.
(Diese sind nun verschieden, aber gleich verteilt.)

qq-Plot muss eine Gerade sein.
Starke Krümmung bei ganz verschiedenen Verteilungen.

```
> x = rnorm(100)
> y = rnorm(100)
> qqplot(x,y)
# in etwa eine Gerade
```

```
> x = rexp(100)
> y = rnorm(100)
> qqplot(x,y)
# stark gekrümmt
```

Wichtiger Fall: Test auf Normalität

Gegeben: Daten (Vektor) x
Frage: Stammt x aus einer Normalverteilung?

Kann Folgendes tun:

```
y=rnorm(10000)
qqplot(x,y)   # Gerade?
```

Einfacher:

keine Zufallszahlen ziehen, sondern mit theoretischen Quantilen vergleichen

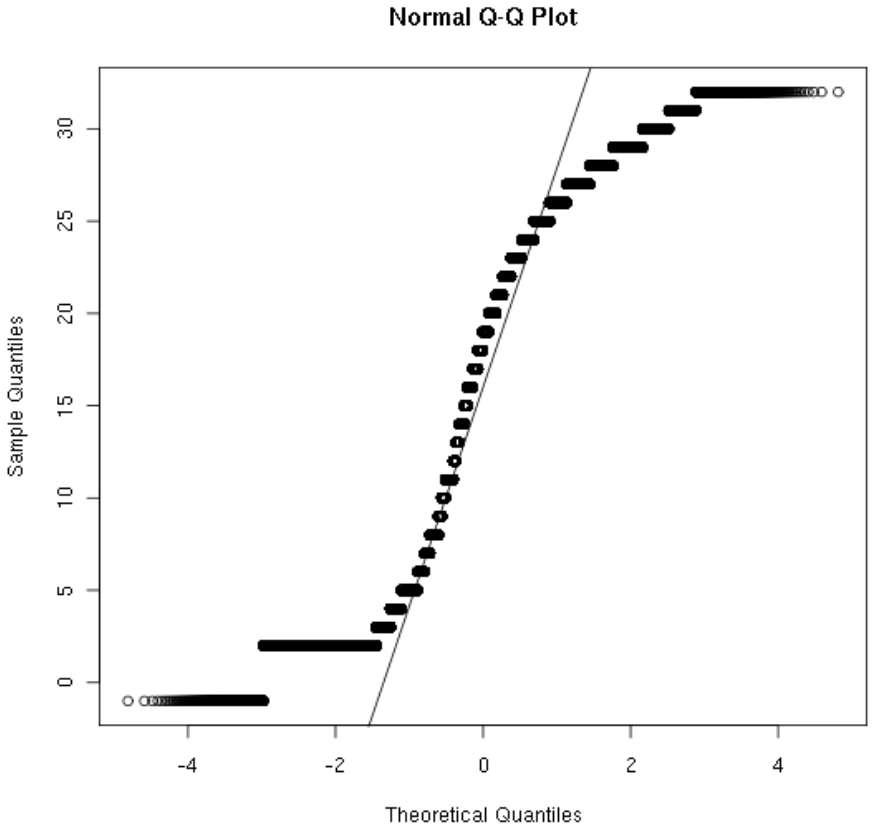
```
qqnorm(x)     # Gerade ??
qqline(x)     # Beste Gerade dazu einzeichnen!
```

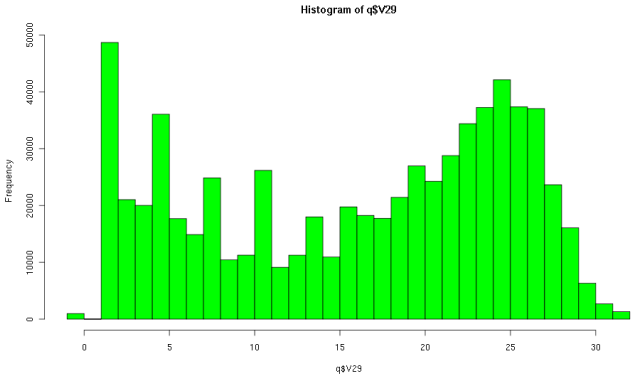
Beispiel:

Sequenzierdaten; Qualitätswerte an Position 29 normalverteilt?

Test auf Normalität mit qqnorm()

```
qqnorm(q$V29) # Gerade??
qqline(q$V29) # beste Gerade dazu
```





Warum nicht Histogramm mit Dichte dnorm() vergleichen?

- qqplot zeigt Abweichungen besser; invariant bei Verschiebung, Skalierung