# Grundlagen zur Computertomografie

Die Computertomografie beschäftigt sich mit der Rekonstruktion des inneren Aufbaus eines Gegenstandes. Mit "inneren Aufbau" ist hier die räumliche Verteilung einer Materialeigenschaft gemeint. Die bekannteste Möglichkeit ist vermutlich die Vermessung des Absorptionskoeffiezienten $\alpha$ beim Durchschicken eines "dünnen" Röntgenstrahls durch menschliches Gewebe. Ziel also ist, das skalare Feld $\alpha(\vec{x})$ zu rekonstruieren. Aber statt Absorption könnten auch andere Eigenschaften wie "time-of-flight" (Durchdringungszeit), Phasenverschiebung, Streuung, elektrischer Widerstand, usw. herangezogen werden.

Die Wikipedia zählt heute (2014) ca. 30 verschieden Tomografietypen auf. Alle benützen natürlich irgendeine physikalische Eigenschaft des "zu scannenden Gegenstandes" wie z.B. den elektrischen Widerstand. Die durch das "Scannen" gewonnenen Daten lassen sich bearbeiten (wo sind starke Veränderungen - Umrisse) und Hand von Vergleichsdaten lassen sich bekannte Komponenten(Organe) identifizieren.

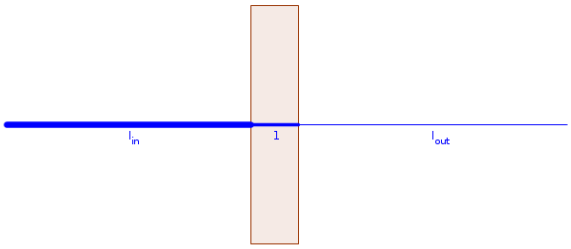
Die CT ist im Wesentlichen(Ausnahme: Einbringen von Kontrastmittel, radioaktives Material, usw.) ein "nicht-invasives" Verfahren, d.h. wir brauchen den Gegenstand nicht "Aufschneiden", um sein Inneres zu kennen. Wie weit das Einbringen ionisierender Strahlung als "nicht-invasiv" zu bezeichnen ist, ist natürlich Ansichtssache - aber klarerweise müssen wir irgendeine Wechselwirkung zwischen Gegenstand und Messgerät herbeiführen. Ultraschall schneidet in punkto "biologische Gefährlichkeit" besser ab, durch die weit größere Wellenlänge ist die Auflösung dafür weit geringer.

Im Folgenden werden wir die Röntgen-CT im Auge haben, da wir Reflexion und Streuung vernachlässigen können. Wir können die Welleneigenschaften vernachlässigen. Zur weiteren Vereinfachung unseres Modelles werden wir 1 Dimension vernachlässigen und nur Raumpunkte betrachten, die in einer Ebene liegen.Außerdem werden wir das skalare Feld $\alpha(\vec{x})$ diskretisieren, d.h. wie bei einem Fernseher das Bild aus Pixel (Bildpunkten) aufgebaut ist, so werden wir unser kontinuierliches skalare Feld $\alpha(\vec{x})$ durch $\alpha(\vec{x}_i)$ annähern. Wir können die "Pixel" von links oben nach rechts unten durchnummerieren und so eine Folge von Variablen schaffen, die es zu berechnen gilt. Für die Vorstellung ist es allerdings leichter sich einem Matrix-Schema zu bedienen, also $\alpha(\vec{x}_{ij})$ kurz $\alpha_{ij}$ für $i$-te Zeile und $j$-te Spalte. Dies kann leicht in den Folgenindex umgerechnet werden.

Aber jetzt zu den physikalischen Grundlagen der Röntgen-CT:

# 1 Physikalische Grundlagen

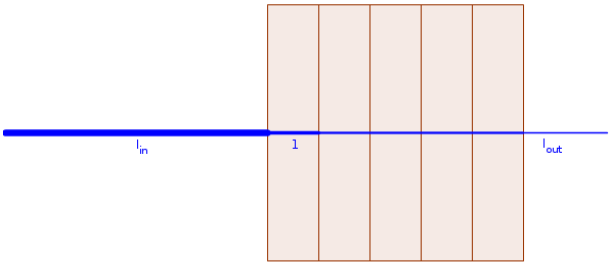
1)




Die Dicke der Schicht sei 1 Längeneinheit.
90% der Strahlung werden durchgelassen, daraus ergibt sich:

$$I_{out} = 0,9 \cdot I_{in}$$

2)




Die Dicke der Schicht sei d Längeneinheiten.
90% der Strahlung werden durchgelassen(pro Schichteinheit), daraus ergibt sich:

$$I_{out} = 0,9^d \cdot I_{in}$$

mit folgender Ersetzung

$$0,9 = e^{\ln 0,9}$$

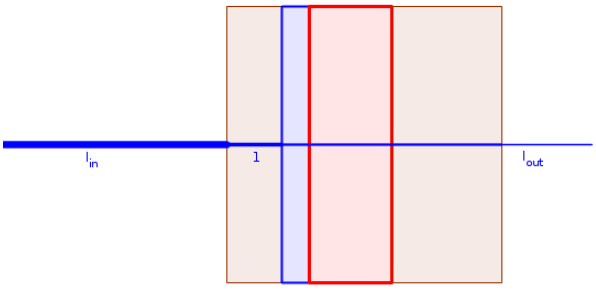
ergibt sich

$$I_{out} = e^{(\ln 0,9)d} \cdot I_{in}$$

mit $\alpha = -\ln 0,9$ (Absorptionskoeffizient) ergibt sich

$$I_{out} = e^{-\alpha d} \cdot I_{in}$$

**Absorption an verschiedenen Schichten**




Die Dicke der Schichten sei $d_1, d_2 \ldots d_4$ Längeneinheiten mit den Absorptionskoeffizienten $\alpha_1, \alpha_2, \ldots \alpha_4$. Daraus ergibt sich:

$$I_{out} = I_{in} e^{-\alpha_1 d_1} e^{-\alpha_2 d_2} \ldots e^{-\alpha_4 d_4} = I_{in} e^{-\sum_i \alpha_i d_i}$$

dividieren durch $I_{in}$ und logarithmieren liefert

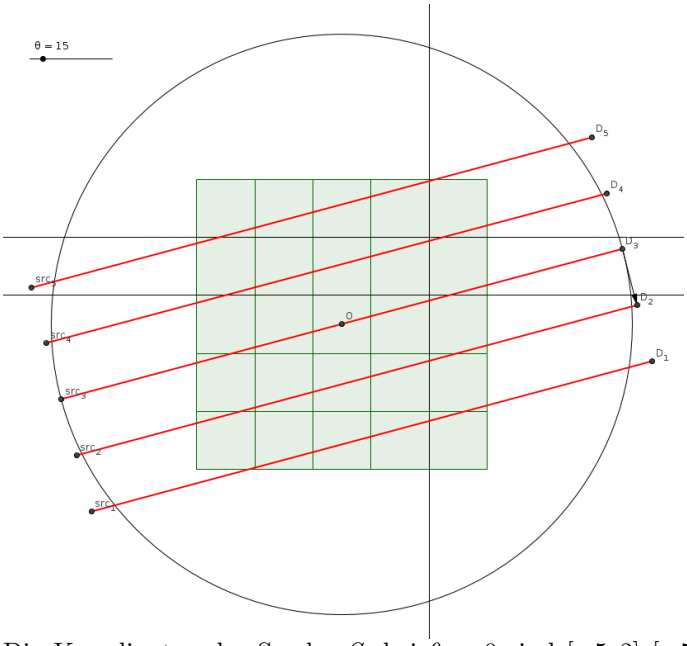
$$\frac{I_{out}}{I_{in}} = e^{-\sum_i \alpha_i d_i} \Rightarrow \boxed{-\ln \frac{I_{out}}{I_{in}} = \sum_i \alpha_i d_i}$$

Bei bekannter linker Seite (Eingangs- und Ausgangsintensität) und bekannten Schichtdicken($d_i$) ist dies eine lineare Gleichung für die Absorptionskoeffiezienten des Materials. Man kann natürlich nicht hoffen dieses Gleichungssystem durch variieren des Strahlwinkels zu lösen, weil die Anordnung der Schichten beliebig permutiert werden kann!
Allerdings ändert sich das durch hinzufügen einer Dimension:

## 2 Geometrische Grundlagen

Diese Situation wurde vom Mathematiker Radon bereits geklärt vor dem 1. Weltkrieg - ein typisches Beispiel, dass Grundlagenforschung gar nicht hoch genug einzuschätzen ist. Niemand kann ahnen, ob die Forschungergebnisse von "Nutzen" sind ( ein anderes wichtiges Beispiel wären die " gekrümmten Mannigfaltigkeiten" von Riemann - ohne die Einstein niemals seine Allgemeine Relativitätstheorie formulieren hätte können). Wer sich näher damit aueinandersetzen möchte, müsste sich mit der "Radon-Transformation" beschäftigen.

Jetzt zu unserer "Versuchsanordnung":

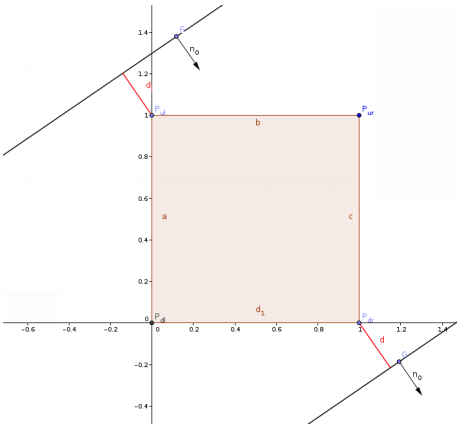



Der Einfachheit halber besteht unser Objekt nur aus 25 quadratischen "Pixeln", die zu einem Quadrat angeordnet sind. Da die Anordnung eine Matrixschreibweise nahelegt, bezeichnen wir die unbekannten Absorptionskoeffiezienten mit

$$\begin{pmatrix} \alpha_{11} & \alpha_{12} & \dots & \alpha_{15} \\ \alpha_{21} & \alpha_{22} & \dots & \alpha_{25} \\ \vdots & & & \vdots \\ \alpha_{51} & \alpha_{52} & \dots & \alpha_{55} \end{pmatrix}$$

Die Seitenlänge der Pixel sei unsere Einheitslänge. Gescannt wird unserer Target durch ein paralleles Strahlenbüschel (5 Stück im Abstand 1), die den Winkel $\theta$ mit der x-Achse bilden. Der Winkel $\theta$ kann durch Drehung der Sender bzw. des Targets variiert werden. Der Richtungsvektor der Geraden $\vec{a} = (\cos\theta, \sin\theta)$, der nach rechts geklappte Einheitsnormalvektor beträgt $\vec{n}_0 = (\sin\theta, -\cos\theta)$
Der Koordinatenursprung ist eingezeichnet.
Die Koordinaten der Sender $S_i$ bei $\theta = 0$ sind $[-5, 2], [-5, 1], [-5, 0], [-5, -1], [-5, -2]$ und ihre aktuelle Position wird durch eine Rotationsmatrix bestimmt. Jetzt gilt es zu überlegen, welche Strecke die einzelnen Strahlen in den Pixeln zurücklegen:

### 1) Kein Treffer bei $0 \leq \theta \leq 90°$

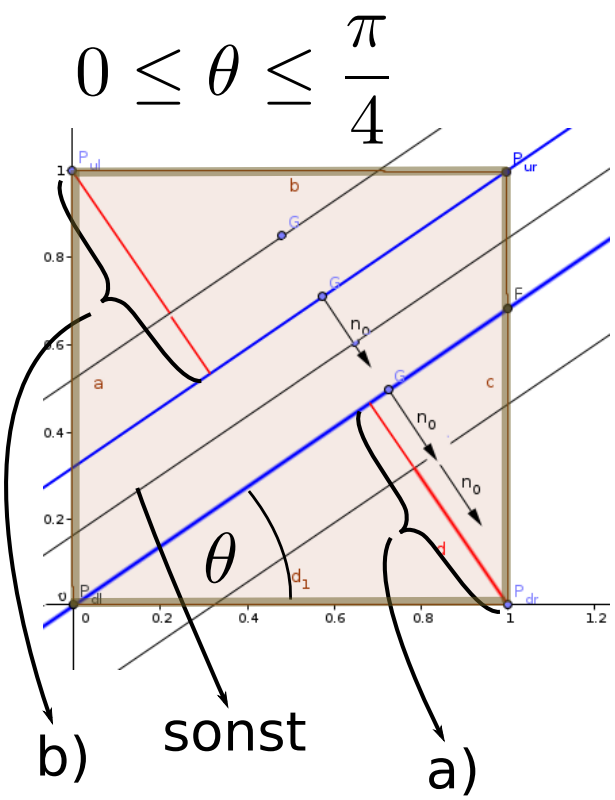



Der Richtungsvektor der Geraden $\vec{a} = (\cos\theta, \sin\theta)$, der nach rechts geklappte Einheitsnormalvektor beträgt $\vec{n}_0 = (\sin\theta, -\cos\theta)$
Wenn der Strahl von "unten" hereinwandert, stößt er zuerst auf $P_{dr}$ (down-right) und verlässt das Quadrat wieder bei $P_{ul}$(up-left). Das Ereignis "kein Treffer" lässt sich also so formulieren:

$$\boxed{\overrightarrow{SP}_{dr} \cdot \vec{n}_0 \leq 0 \qquad \text{oder} \qquad \overrightarrow{SP}_{ul} \cdot \vec{n}_0 \geq 0}$$

$S$ ist dabei der Ausgangspunkt des Strahls (Sender, Source). Also verbal ausgedrückt:
$P_{dr}$ liegt links vom Strahl oder $P_{ul}$ liegt rechts von ihm!

**2) Einfallswinkel** $\theta \leq 45°$




Wenn der Strahl von "unten" hereinwandert, gibt es 2 Grenzfälle (blau) bei denen er eine horizontale und eine vertikale Grenze durchkreuzt, dazwischen liegt der Fall mit 2 vertikalen Schnitten.
Kümmern wir uns um den Grenzfall $a$): (Wir schreiben statt dem Skalarprodukt in Zukunft für den Abstand eines Punktes vom Strahl einfach $d(P)$)

$$d(P_{dr}) = \sin\theta$$

Also bei $d(P_{dr}) \leq \sin\theta$ sind wir unterhalb des unteren blauen Grenzstrahls, bei $|d(P_{ul})| \leq \sin\theta$ sind wir oberhalb des oberen blauen Grenzstrahls, sonst durchschneiden wir das ganze Quadrat! Für die Länge $\ell$, die der Strahl im "Pixel" zubringt bedeutet das:

$$\left.\begin{array}{ll} d(P_{dr}) & \leq \sin\theta \\ or & \\ |d(P_{ul})| & \leq \sin\theta \end{array}\right\} \Rightarrow \ell = d_x \frac{2}{\sin 2\theta}$$

$d_x$ ist dabei der entsprechende positive Abstand. In allen anderen Fällen (also zwischen den blauen Linien) ergibt sich:

$$\ell = \frac{1}{\cos\theta}$$

**3) Einfallswinkel** $45° \leq \theta \leq 90°$

Die genaue Analyse überlasse ich jetzt dem Leser - aber sie funktioniert nach obigem Vorbild. Nach wie vor sind unten-rechts(dr) und oben-links (ul) die 2 Bezugspunkte, damit ergeben sich die Fälle:

$$\begin{array}{llll} a) & d(P_{dr}) & \leq \cos\theta & \Rightarrow \quad \ell = d(P_{dr})\frac{2}{\sin 2\theta} \\ b) & |d(P_{ul})| & \leq \cos\theta & \Rightarrow \quad \ell = |d(P_{ul})|\frac{2}{\sin 2\theta} \\ c) & & sonst & \Rightarrow \quad \ell = \frac{1}{\sin\theta} \end{array}$$

Um dies in die obigen Formeln überzuführen können wir $\theta$ einfach durch $\theta' = \frac{\pi}{2} - \theta$ ersetzen Es bleibt dem Leser überlassen, dass dadurch der Ausdruck $\sin 2\theta$ "unverändert" bleibt.

**4) Kein Treffer bei** $90° \leq \theta \leq 180°$

Ersetze in obiger Formel $P_{dr}$ durch $P_{ur}$ (up-right) und $P_{ul}$ durch $P_{dl}$(down-left) oder kurz: up $\leftrightarrow$ down
**Diese Ersetzung gilt auch für die folgenden Punkte!**

**5) Einfallswinkel** $90° \leq \theta \leq 135°$

obige Ersetzung und $\theta$ durch $\theta' = \theta - \frac{\pi}{2}$ in **2)**

**6) Einfallswinkel** $135° \leq \theta \leq 180°$

obige Ersetzung und $\theta$ durch $\theta' = \pi - \theta$ in **2)**

## 3 Mathematische Grundlagen

### Rotationen in der Ebene

Hier gibt es nicht viel zu sagen:
Rotation ist eine Abbildung des $\mathbb{R}^2 \to \mathbb{R}^2$ und kann mit folgender Matrix durchgeführt werden:

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

also $P_\theta = R_\theta \cdot P$

Versuchen Sie das in Geogebra

A = {{cos(θ), sin(θ)},
{-sin(θ), cos(θ) }}

indem Sie vorher einen Schieberegler $\theta$ definieren. Einen Punkt $P$ in der Nähe des Ursprungs setzen und $P_1 = A * P$ festlegen und anschl. den Schieberegler betätigen. Damit haben Sie den Befehl **Drehe** von Geogebra mathematisch nachgebildet!

### Mathematik der linearen Gleichungssysteme - LA

LA steht hier natürlich nicht für *Los Angeles* sondern für *Lineare Algebra*. Ein eindeutig lösbares lineares Gleichungssystem wird i. a. beschrieben als

$$A \cdot \vec{x} = \vec{b} \,|\, A^{-1}. \qquad \Rightarrow I \cdot \vec{x} = A^{-1} \cdot \vec{b}$$

Das Auffinden des Lösungsvektors läuft sich auf das Invertieren einer quadratischen Matrix hinaus. Jedes CAS kann das - so auch das mit dem wir arbeiten wollen: *wxMaxima* - es ist für alle Plattformen zu haben und im Gegensatz zu *Mathematica* oder *Mapple* ist es **kostenlos** verfügbar. Sprache für die Dinger muss man sowieso für jedes lernen. (Stichwort: “Lebenslanges Lernen”)

Jetzt ist die Sache bei uns etwas komplizierter, denn wir haben es mit einem überbestimmten (hoffentlich - Rang der Matrix ist gleich Spaltenzahl) Gleichungssystem zu tun, d. h. unsere Koeffizientenmatrix $A$ hat mehr Zeilen als Spalten (Anzahl der Messungen übersteigt die Anzahl der gesuchten Absorptionskoeffiezienten). Natürlich gibt es hier keine Lösung im eigentlichen Sinne, sondern wir brauchen einen

Vektor $\vec{x}$, der alle Gleichungen "einigermaßen gut" erfüllt. Ein Verfahren, dass dies erfüllt heißt *Q-R-Faktorisierung* bzw. Q-R-Zerlegung von A - und wxMaxima kann das mit `load(lapack)` und der Befehl dafür lautet `dgeqrf(A)` und liefert eine Liste mit $Q$ und $R$.

Welche Eigenschaften haben nun die Matrizen $Q$ und $R$?

1. $Q \cdot R = A$ - klar, wegen Faktorisierung
2. $Q$ ist quadratisch mit derselben Zeilenanzahl von $A$
3. $Q$ ist orthonormal, also $Q^T = Q^{-1}$
4. $R$ hat dieselbe Zeilenanzahl uns Spaltenzahl wie $A$
5. $R$ ist eine obere Dreiecksmatrix (engl. "upper triangular matrix") - restliche Zeilen sind 0!

Die Vorgangsweise ist im Kern dann einfach:

$$\boxed{A \cdot \vec{x} = \vec{b} \quad \Rightarrow \quad Q \cdot R \cdot \vec{x} = \vec{b} \quad \Rightarrow \quad R \cdot \vec{x} = Q^T \cdot \vec{b}}$$

Die letze Gleichung ist durch Rücksubstitution aber leicht zu lösen.

Stellen wir das Verfahren in wxMaxima auf die Probe: wir nehmen $x = 1$ und $y = 2$ und basteln ein nicht-widersprüchliches überbestimmtes Gleichungssystem aus 4 Gleichungen

$$\begin{array}{llllll} 1) & x & + & y & = & 3 \\ 2) & 2x & + & y & = & 4 \\ 3) & 2x & + & 2y & = & 6 \\ 4) & 5x & + & 2y & = & 9 \end{array} \quad \Rightarrow \quad \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 2 & 2 \\ 5 & 2 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \\ 6 \\ 9 \end{pmatrix}$$

Bevor wir uns die Lösung in wxMaxima anschauen, noch ein kurzer Hinweis wie in diesem Programm Funktionen aufgebaut sind:

```
<funcName>(<par1>,<par2>, ...):=block([<localVar1>: <value1>,<localVar2>: <value2>],
  for c:10 thru 1 step -1 do  <command>,
<var>)$
```

Also Funktionsname mit Übergabeparametern, `block(` ist dasselbe wie in vielen Programmiersprachen eine geschwungene Klammer und wird dort meist als *scope* bezeichnet (Gültigkeitsbereich von lokalen Variablen). Wie in anderen Programmiersprachen auch, kann `block(` weggelassen werden, wenn nur ein Befehl kommt. Innerhalb von `block(` können in eckigen Klammern *lokale Variablen* definiert werden (und ev. initialisiert), um globale Variablen nicht zu beeinflussen. Als Beispiel habe ich hier einmal eine Schleife mit der Schleifenvariablen c gewählt, wobei von 10 heruntergezählt wird. Die einzelnen Befehle werden durch Beistriche getrennt (in Javascript ist dies ein ";"). Der Wert der letzten Variable wird zurückgegeben!

Das folgende Programm ist selbsterklärend bis auf die Funktion `backwardSubstitution` . Übergabeparameter sind die Dreiecksmatrix $r$ und der Konstantenvektor $b$. In den lokalen Variablen holt man sich mit der Funktion `matrix_size` und `second` die Spaltenanzahl von $r$ und definiert einen Nullvektor $x$ mit 1 Spalte und `cols` Zeilen. Die Berechnung des Lösungsvektors durch "Rückeinsetzen" ergibt (bitte nachrechnen):

$$x_c = \left( b_c - \underbrace{\sum_{k=c-1}^{n} r_{ck} \cdot x_k}_{\vec{r}_c \cdot \vec{x}} \right) / r_{cc}$$

Die Summe kann also mit dem skalaren Produkt des c-ten Zeilenvektors von $r$(=`row(r,c)`) und dem bisher bekannten Lösungsvektor $x$ berechnet werden.

```
(%i11) load(lapack)$
```

...

```
(%i12) A:matrix([1,1],[2,1],[2,2],[5,2]);
```

$$(\%o12)\quad \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 2 & 2 \\ 5 & 2 \end{pmatrix}$$

```
(%i13) b:matrix([3],[4],[6],[9])$
```

```
(%i14) [q,r]:dgeqrf(A)$
```

```
(%i15) b_T:transpose(q) . b;
```

$$(\%o15)\quad \begin{pmatrix} -11.6619037896906 \\ 2.44948974278318 \\ -1.554312234475219\,10^{-15} \\ -8.881784197001252\,10^{-16} \end{pmatrix}$$

```
(%i16) backwardSubstitution(r,b):=block([cols:second(matrix_size(r)),
          x:zeromatrix(second(matrix_size(r)),1)],
       for c:cols thru 1 step -1 do  x[c]:((b[c]-row(r,c) . x)/r[c,c]),
       x)$
```

```
(%i17) backwardSubstitution(r,b_T);
```

$$(\%o17)\quad \begin{pmatrix} 1.0 \\ 2.0 \end{pmatrix}$$

Hat doch ganz gut geklappt! Jetzt könnte man noch den Vektor $b$ verändern und so das System widersprüchlich machen - diese Experiment überlasse ich dem Leser.

Fazit: Weichen die Werte nicht allzustark ab, bleibt der Lösungsvektor in der Nähe von $(1, 2)$

Nun zum eigentlichen Programm - wxMaxima braucht dazu bei mir ca. 1 Minute - es handelt sich schließlich um 36 verschiedene Winkel bei 5 Strahlen - macht 180 Gleichungen für 25 Variable.

```
(%i79) ratprint:false$ loadprint:false$ load(lapack)$
```

$0 errors,$

Bezugspunkt eines quadratischen Pixels ist unten-links, für die anderen Eckpunkte werden die Koordinaten entsprechend erhöht!

```
(%i82) down_left:[0,0]$ down_right:[1,0]$ up_right:[1,1]$ up_left:[0,1]$
```

```
(%i86) vecLength(P,Q):=sqrt((P-Q) . (P-Q))$
```

Eingabe Matrix-Indices und position z.B. `down_right` Rückgabe:$[x, y]$

```
(%i87) getPoint(i,j, position):=[-3.5+j,2.5-i]+position$
```

```
(%i88) getNormalUnitVec(theta):=block([t:float(theta*%pi/180)],
    [sin(t), -cos(t)])$
```

Hier bekommt man die Rotationsmatrix für einen bestimmten Winkel $\theta$

```
(%i89) getRotMat(theta):=block([t:float(theta*%pi/180),m],
  m:matrix([cos(t), -sin(t)],[sin(t),cos(t)]), m)$
```

Der rotierte Punkt wird zurückgegeben

```
(%i90) rotatePoint(P,theta):=block([m, P_r],
    m:getRotMat(theta),
    P_r: m . P,
    P_r)$
```

Hier wird einfach eine Liste der 5 Scannerpositionen bei $\theta = 0$ generiert

```
(%i91) createSourceList():=block([src:[],ystart:-2],
      for j: 0 thru 4 do src:cons([-5,ystart+j], src),
  src)$
```

Ausprobiert!

```
(%i92) srcL:createSourceList();
```

$$(\%o92)\ [[-5, 2], [-5, 1], [-5, 0], [-5, -1], [-5, -2]]$$

Der Befehl `map` wendet eine Funktion auf alle Listenelemente an - in diesem Fall `rotatePoint(P,theta)` - und gibt diese Liste zurück.
Übergabeparameter sind der Winkel und die Punktliste

```
(%i93) rotListBy(theta,l):=map(lambda([P],rotatePoint(P,theta)),l)$
```

Verwandelt die übergebene $5 \times 5$ Matrix in einen Zeilenvektor. Das brauchen wir um die Absorptionskoeff.-Matrix und Penetrationslängen-Matrix in Zeilen der Koeffizientenmatrix umzuformen. `cons` fügt ein Listenelement vorne an eine Liste. Wichtig ist die Reihenfolge: $[a_{11}, a_{12}, \ldots a_{15}, a_{21}, \ldots a_{55}]$

```
(%i94) convert2List(M):=block([H:[]],
for i: 1 thru 5 do
  for j: 1 thru 5 do
      H: cons(M[6-i,6-j],H),
H)$
```

Probe - man kann mit Geogebra überprüfen, ob das passt!

```
(%i95) srcL_Rot:reverse(rotListBy(30,srcL));
```

$$(\%o95)\ \left[\begin{pmatrix}-3.330127018922194\\-4.232050807568877\end{pmatrix},\begin{pmatrix}-3.830127018922194\\-3.366025403784438\end{pmatrix},\ldots\right]$$

$\overrightarrow{SP}\cdot\vec{n}_0$ wird berechnet, wobei $P$ mit den Marixkoordinaten angegeben wird. $S$ ist eine "Senderposition"

```
(%i96) getScalarProd(S,n0,i,j, position):=block([P, vecSP],
    P:getPoint(i, j, position),
    vecSP:P - S,
    n0 . vecSP)$
```

Hier wird jetzt bestimmt, welche Weglänge in einem Pixel zurückgelegt wird(die Fallunterscheidungen) - die Theorie dazu haben wir oben besprochen

```
(%i97) evalScalarProd(dr,ul,theta):=block([t:float(theta*%pi/180), pi2:float(%pi/2),
                                           pi4:float(%pi/4), pi34:float(3*%pi/4)],
  if (t > pi4) then t:pi2 - t else
    if (t > pi2) then t: t - pi2 else
      if (t > pi34) then t: 2*pi2 - t,
  if ((ul >= 0) or (dr <= 0)) then 0 else
    if (dr <= sin(t)) then dr*2/sin(2*t) else
      if  (abs(ul) <= sin(t)) then abs(ul)*2/sin(2*t) else
     1/cos(t))$
```

Hier wird jetzt für einen bestimmten Sender die Weglängenmatrix berechnet - die meisten Elemente werden natürlich Null sein

```
(%i98) getPenetrationMatrix(S,theta):=block([m:zeromatrix(5,5), n0],
  n0:getNormalUnitVec(theta),
  for i thru 5 do
    for j thru 5 do block(
      d_dr:getScalarProd(S,n0, i,j, down_right),
      d_ur:getScalarProd(S,n0, i,j, down_right),
      d_ul:getScalarProd(S,n0, i,j, up_left),
      d_dl:getScalarProd(S,n0, i,j, down_left),
      if (theta <= 90) then m[i,j]: evalScalarProd(d_dr, d_ul, theta)
      else m[i,j]: evalScalarProd(d_ur, d_dl, theta)
  ),
  m)$
```

Hier wieder eine Probeausgabe, die man mit Geogebra verifizieren kann!

```
(%i99) getPenetrationMatrix(srcL_Rot[3],30);
```

$$(\%o99)\ \begin{pmatrix}0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & .7320508075688776 & 1.154700538379251\\ 0 & 0.422649730810375 & 1.154700538379251 & .4226497308103739 & 0\\ 1.154700538379251 & .7320508075688766 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\end{pmatrix}$$

Hier haben wir das “Arbeitspferd” vor uns:
Auf die rotierte Senderliste wird die Funktion `getPenetrationMatrix(S,angle)` angewendet und diese Matrix wird mit `append` an die Liste `l` angehängt und diese dann zurückgegeben,

```
--> getMatrixList():=block([srcL:createSourceList(), l:[]],
    for angle:0 thru 175 step 5 do
      l:append(map(lambda([S],
         getPenetrationMatrix(S,angle)),rotListBy(angle,srcL)),l)
    ,l)$
```

Jetzt wird gearbeitet - die Matrizenlist wird erstellt!

```
(%i101)mList:getMatrixList()$
```

Die Liste der Penetrationslängen-Matrizen wird zur Koeffizientenmatrix umgestaltet. Schlüsselbefehl ist hier `addrow`: nimmt eine Matrix und fügt eine Zeile an

```
(%i102)createCoeffMatrix(ml):=block([m:matrix(convert2List(ml[1])), mat],
      for i:2 thru length(ml) do m:addrow(m, convert2List(ml[i])),
      m)$
```

Wird jetzt ausgeführt!

```
(%i103)MM:createCoeffMatrix(mList)$
```

Zur Sicherheit schauen wir uns den Rang der Matrix an - sind wir nicht unterbestimmt?

```
(%i104)rank(MM);
```

$$(\%o104)\ 25$$

Wir denken uns Versuchsdaten aus!

```
(%i105)absorbMat:matrix(
          [0.1,0.2,0.3,0.4,0.5],
          [0.3,0.4,0.5,0.6,0.7],
          [0.5,0.6,0.7,0.8,0.9],
          [1.1,1.2,1.3,1.4,1.5],
          [1.9,2.0,2.1,2.2,5.3])$
```

Verwandeln sie in einen Vektor

```
(%i106)absorbVec:matrix(convert2List(absorbMat));
```

$$(\%o106)\ \begin{pmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.3 & \dots & 2.0 & 2.1 & 2.2 & 5.3 \end{pmatrix}$$

Bestimmen die “Ablesewerte”

```
(%i107)readingVec:MM . absorbVec$
```

Um den Median zu berechnen brauchen wir ein Paket

```
(%i108)load(descriptive)$
```

Wir fügen zu den exakten Daten zufällige Ablesefehler im Ausmaß von 5% bei 10 “Messungen” und nehmen dann aus dieser Liste den Medianwert

```
(%i117)addNoise(rv):=block(
         for i thru length(rv) do block([sign:0, readingList:[]],
           for j thru 10 do block(
              if (floor(random(2.0)) = 1) then sign:1 else sign: -1,
              readingList:cons(rv[i,1]+sign*random(abs(rv[i,1])/20.0+0.001), readingList)),
          rv[i,1]:median(readingList)),
        rv)$
```

Es ist soweit: $Q-R$–Faktorisierung. Jetzt heißt es warten!

```
(%i110)[q, r]: dgeqrf(MM)$
```

```
(%i111)b:addNoise(readingVec)$
```

Den Rest kennen wir von unserem Einführungsgleichungssystem

```
(%i112)q_t:transpose(q)$
```

```
(%i113)b_1:q_t . b$
```

```
(%i114)transposeAndRound(x):=block([x1:transpose(x), l:[]],
         l:map(lambda([p], floor((100*p+0.5))/100.0), x1),
          l)$
```

```
-->    backwardSubstitution(r,b):=block([cols:second(matrix_size(r)),
                          x:zeromatrix(second(matrix_size(r)),1)],
       for c:cols thru 1 step -1 do  x[c]:((b[c]-row(r,c) . x)/r[c,c]),
       transposeAndRound(x))$
```

Schaut nicht schlecht aus!

```
(%i116)solutionVec:backwardSubstitution(r,b_1);
```

(%o116) $\begin{pmatrix} 0.09 & 0.2 & 0.31 & 0.41 & 0.5 & \dots & 1.41 & 1.48 & 1.91 & 2.03 & 2.14 & 2.22 & 5.34 \end{pmatrix}$

Hier das vollständige Programm “zum runterladen”

Bis jetzt haben wir einige Aufgaben (Median, Q-R-Faktorisierung) *wxMaxima* ausführen lassen (packages: descriptive, lapack), aber wir können natürlich auch wissen wollen, was dahintersteckt:

3 Anhänge sollen dies klären:

1. Median
2. Q-R-Faktorisierung
3. Backtracking