Norbert Fuhrmann

Vermessung
Python-Programme
Programmbeschreibungen

Vermessungstechnische Berechnungen
Programme für Smartphone, Tablet und PC (Android und Windows)
auf Python-Basis
Version 1.01, 2023
Seite 1 - 25

Helmerttransformation
Programm für Windows auf Python-Basis
Version 1.02, 2023 (2025)
Seite 1 – 19

Manual für das Modul pyvermessung.py
Version 1.02, 2024
Seite 1 – 32

www.grenzuntersuchung.de
https://www.deutsche-digitale-bibliothek.de/item/BWUYGE33YW27PDHHQYSQ3HHIRZOXZBZG

Kerpen

Vermessungstechnische Berechnungen

Programme für Smartphone, Tablet und PC (Android bzw. Windows) auf Python-Basis
Version 1.01

Norbert Fuhrmann, Kerpen

1 Orthogonale und polare Elemente

2 Polaraufnahme (lokale Geradlinigkeit)

3 Polaraufnahme (Koordinaten)

4 Transformation, Kleinpunkte

5 Geradenschnitte

6 Kreis-Geraden-Schnitt

7 Kreisbogenteilung

8 Flächenberechnung

9 Polygonzug

10 Streckenreduktion örtlich ⇒ UTM

11 Strecke UTM ⇒ örtlich

12 Bogenschnitt, Höhe und Höhenfußpunkt

13 Vorwärtsschnitt

14 Pythagorasprobe

Nachtrag Beispiel (2024)

© 2023 Norbert Fuhrmann, Kerpen (All Rights Reserved) (Gesetzt in LaTeX)

Allgemeine Hinweise

Die hier vorliegenden Programme mit vermessungstechnischen Standardaufgaben sind für den Einsatz auf einem Android-Smartphone oder -Tablet sowie für Windows gedacht. Sie können und sollen große vermessungstechnische Programmsysteme mit Datenbankanbindungen nicht ersetzen. Die Programme sind in Python geschrieben worden. Python ist eine höhere, interpretierende Programmiersprache von der <Python Software Foundation>. In Python geschriebener Code lässt sich in diversen Betriebssystemen anwenden. Der Python-Interpreter <IDLE Shell 3.11.0> für Windows kann im Internet unter https://www.python.org/downloads/ heruntergeladen werden. Ein Python-Datei für Windows wie auch für Android hat die Endung *.pyw.

Für die Verwendung in **Windows** gilt (in <Vermessung1-windows.zip>):

Die Datei <Vermessung1.exe> ist ausschließlich für Windows gedacht; dies am besten auf dem Desktop. Sie wurde mit <pyinstaller> Version 5.6.2 aus dem Python-Code erzeugt. **In Windows können mehrere gleiche und/oder unterschiedliche Instanzen geöffnet und nebeneinander benutzt werden.** Die Grafik-Dateien brauchen nicht installiert werden.

Für die Verwendung in **Android** gilt (in <Vermessung1-android.zip>):

Für die Anwendung auf einem Smartphone bzw. Tablet (mit Android) muss im Google <Play store> unter dem Stichwort <python 3 for android> die App <Pydroid 3 - IDE for Python 3>[1] heruntergeladen werden; diese enthält Werbung. Für derzeit € 9,99 kann diese abgeschaltet werden, was für die Leistungsfähigkeit des Programmes hinnehmbar wäre.

Die Programme in Android sind auf eine einzige, sichtbare Instanz hin ausgerichtet (Fullscreen-Modus). Dies ist erforderlich, weil das kleine Display eines Smartphones eine solide Handhabung mit mehreren Fenstern nicht möglich macht.

[1] $\Longrightarrow$ Pydroid_3_-_IDE_for_Python_3.apk (Version 4.01_arm64 bzw. 5.00_arm64).
https://apkpure.com/pydroid-3-ide-for-python-3/ru.iiec.pydroid3/versions
https://pydroid-3-ide-for-python-3.de.softonic.com/android
https://apkfab.com/de/pydroid-3-ide-for-python-3/ru.iiec.pydroid3
Andere, auch neuere Versionen können mitunter Dateipfade nicht finden oder verändern. (Ein allgemeines Problem bei Android; was auch bei Dateimanager auftritt.)

Die gepackte Datei <Vermessung1-android.zip> enthält die Dateien <Vermessung1.pyw> und die Grafiken <v01.png> bis <v13.png>. Die entpackten Dateien müssen sich unter Android in demselben Ordner befinden bzw. in dem gleichen Pfad installiert bzw. kopiert werden, in dem sich auch das Hauptprogramm <Vermessung1.pyw> befindet, dies am Besten im Hauptspeicher. Sollte das Programm in Android nicht direkt geöffnet werden können, so ist über die Pydroid 3 IDE oben rechts zu starten mit:

> Open > Interal storage > <folder>[2] > <Vermessung1.pyw>

In Windows kann das Quellen-Programm auch in der Python <IDLE Shell 3.11.0> aufgerufen werden mit:

Open ... > Dateipfad und Datei (mit der Endung *.pyw)

mit der Befehlszeile: android = False

Da dies eine frei zugängliche Codierung ist, ist dringend davon abzuraten, sie ändern zu wollen. Ein reibungsloser Verlauf des Programms kann eingeschränkt oder verhindert werden. Eine Gewähr kann nicht übernommen werden.

Eine Druckausgabe ist nicht möglich. Eine dauerhafte Dokumentation dürfte mit einem <screenshot> möglich sein[3]. Allerdings sollten Screenshots, und damit eine dauerhafte Dokumentation, nur dann gemacht werden, wenn sich ein grüner Haken im Display neben der Programmüberschrift zeigt **und keine** Fehlermeldung angezeigt wird; nur um fehlerhafte Dokumentationen zu vermeiden.

Hinweise zur Programmbedienung

Dezimalzahlen müssen durch einen Punkt und nicht mit einem Komma eingegeben werden (pythonspezifisch)[4]. Die Eingabe von Buchstaben ist nicht erlaubt. Durch eine programmspezifische Ausnahmebehandlung würde eine Berechnung nur durchgeführt werden, wenn eine Umwandlung der Eingabe in Gleitkommazahlen (ohne Buchstaben, Komma oder sonstige Zeichen) möglich wäre. Ansonsten müssen Veränderungen in der Eingabe erfolgen.

Vielfach ist in den Eingabefeldern ein Defaultwert (z. B. 100) angezeigt, der durch eine neue Eingabe überschrieben werden kann.

[2] directory, Ordner mit Programmen und Abbildungen (normalerweise im Hauptspeicher)

[3] Beispiele: für Windows <microsoft swift>; für Android <Screenshot touch> aus PlayStore

[4] Eine Einstellung des Punktes im Ziffernblock der Tastatur für Windows ist sinnvoll. Man wähle dazu in Windows: >Einstellungen > Zeit und Sprache > Sprache > Sprache hinzufügen > Deutsch-Optionen > Deutsch(Punkt). Hinweise im Internet, einen Punkt im Ziffernblock der Tastatur mit Tastaturtreiber anderer Länder (z.B. Schweiz) erzeugen zu wollen, wechseln nach Tastendruck zwar das Komma in einen Punkt, haben aber die Eigenschaft andere Sonderzeichen der Tastatur zu verändern. Deshalb kann es sinnvoll sein, softwaremäßig ausschließlich nur das Komma im Ziffernblock zu ändern: Die herunter ladbare Software PatchKeyboard_6.1 vermag das und erzeugt einen Tastaturtreiber <Deutsch(Punkt)>, der zu wählen wäre.

Durch den Button Grafik kann zur besseren Übersicht und zum Verständnis der Eingabefelder eine wieder schließbare Zeichnung im Display geöffnet werden. Grafiken sollten durch den unteren, sich rot färbenden Button geschlossen werden, weil sie sonst in der entsprechenden Instanz nicht erneut geöffnet werden können.

Ein gerade durchgeführter Berechnungsablauf wird neben der Titelüberschrift durch einen grünen Haken ✓ angezeigt. Ein Warnzeichen ⚠ taucht bei einer nachträglichen Betätigung einer Taste auf. Dann ist ein erneuter Berechnungsablauf **stets erforderlich**, da sonst im Zusammenhang fehlerhafte, nicht zusammen gehörige Angaben im Display erscheinen können.

Die Ausgabe erfolgt in [0,1 mgon] bzw. [mm] , wobei rechtsseitige Nullen nicht angezeigt werden.

Gelbe oder blassgelbe Eingabefelder dienen **ausschließlich einer Anzeige**. Eingaben darin haben keinerlei Einfluss auf die Auswertung.

Die einzelnen Klassen (class xxx()) sind modular und autark aufgebaut. Sie können eigenständig aus dem Verbund (Vermessung1.pyw) kopiert werden und in das folgende Skriptgerüst für Android eingefügt werden:

```
from tkinter import *
import math, decimal, tkinter.messagebox, os, sys
global android
android = True
class = xxx ()
    Inhalt der Klasse xxx (z. B. wie Inhalt von class StreckeUTM() ≙ <11 Strecke UTM ⇒ örtlich>
i = xxx () (Erzeugung einer Instanz)
```

Vermessungstechnische Hinweise

Ein Maßstabsfaktor kann Eigenschaften einer Korrektion wie auch einer Reduktion haben: Ein Korrektionsfaktor berichtigt die ursprüngliche Messung in Annäherung an den wahren Wert (z. B. Kalibrierungswerte), ein Reduktionsfaktor behandelt z. B. die Horizontierung einer Strecke oder eine Abbildungsverzerrung (Verebnung).

Dem hier vielfach in der Eingabe vorkommenden Reduktionsfaktor r (oder auch Maßstabsfaktor m) bei Strecken bzw. Koordinaten liegt folgende Definition zu Grunde:

$$reale\ \ddot{o}rtliche,\ horizontierte\ Strecke\,(\mathrm{hor.})\ \times\ r\ =\ Strecke\,(\mathrm{UTM})$$

1 Orthogonale und polare Elemente

Das Programm dient ebenso zur Überprüfung von Geradlinigkeiten, zur Ermittlung von Abständen und für einen Senkrechtschnitt. Es können auch ausschließlich nur Richtungswinkel und Entfernung berechnet werden.

Gegeben:

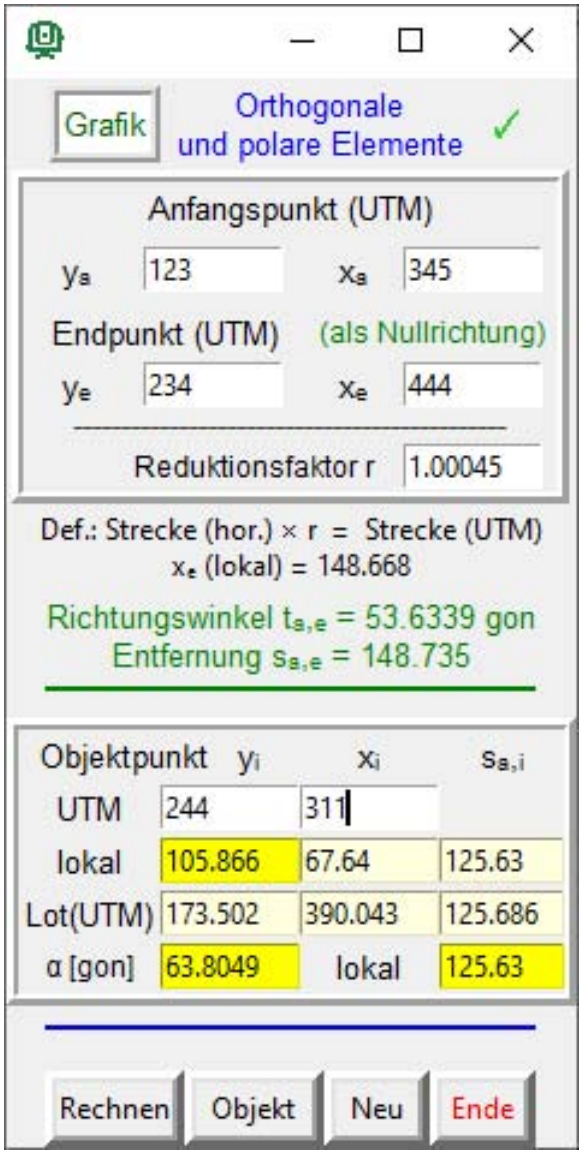
$y_\mathrm{a}, x_\mathrm{a}$ Anfangspunkt (i.d.R. UTM)
$y_\mathrm{e}, x_\mathrm{e}$ Endpunkt (UTM)
y_i, x_i Objektpunkt (UTM)
r Reduktionsfaktor (Default = 1)

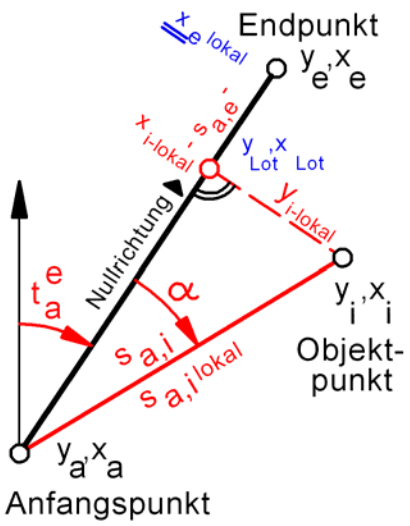
Gesucht:

$t_\mathrm{a}^\mathrm{e}, s_\mathrm{a}^\mathrm{e}$, Richtungswinkel [gon] und Entfernung
$y_{i-\mathrm{lokal}}, x_{i-\mathrm{lokal}}$ lokale orthogonale Elemente
($y_{i-\mathrm{lokal}}$ zur Überprüfung der Geradlinigkeit)
(point to line)
$\alpha, s_{\mathrm{a},i}$ bzw. $s_{\mathrm{a},i\,\mathrm{lokal}}$ Polarwinkel, Strecken
(Polarelemente)

Zusätzliche Ergebnisse:

$y_{i-\mathrm{Lot}}, x_{i-\mathrm{Lot}}$ Koordinaten des Lotfußpunktes (UTM)
x_e lokales Endmaß





Die Nullrichtung für die Polarelemente $\alpha, s_{\mathrm{a},i\,\mathrm{lokal}}$ ist von $y_\mathrm{a}, x_\mathrm{a}$ nach $y_\mathrm{e}, x_\mathrm{e}$ gegeben.

Achtung:

Orthogonale und polare Elemente werden i.d.R. für örtliche Absteckungen benötigt. Werden als Ausgangswerte für $y_\mathrm{a}, x_\mathrm{a}, y_\mathrm{e}, x_\mathrm{e}$ UTM-Koordinaten benutzt, so bewirkt die Eingabe des Reduktionsfaktors r laut der Definition

$$Strecke(\mathrm{hor.}) \times r = Strecke(\mathrm{UTM})$$

hier eine korrekte Berechnung[5], also

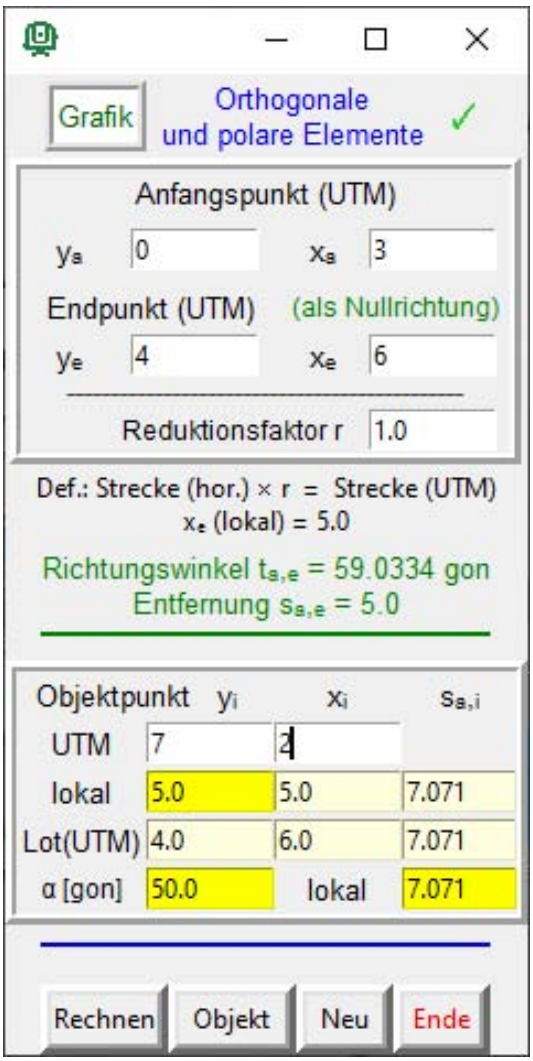
$$Elemente(\mathrm{horizontal}, \mathrm{lokal}) \Longleftrightarrow Koordinaten(\mathrm{UTM}) : r$$

Sollte der wohl seltene, umgekehrte Weg beschritten werden, ist als Reduktionsfaktor dessen reziproker Wert (also $1/r$) zu nehmen.

Bei einem Reduktionsfaktor $r = 1$ liegen eingegebene Koordinaten und die Rechenergebnisse auf einer Rechenebene (s. rechte Abbildung).

Für die alleinige Berechnung von Richtungswinkel und Entfernung reichen die Angaben $y_\mathrm{a}, x_\mathrm{a}, y_\mathrm{e}, x_\mathrm{e}$. Desgleichen können Richtungswinkel und Entfernung können auch mit dem Programm 3 <Polaraufnahme (Koordinaten)> berechnet werden.

Die Berechnung von Senkrechtschnitten (über den Lotfußpunkt) sind auch mit diesem Programmteil möglich.

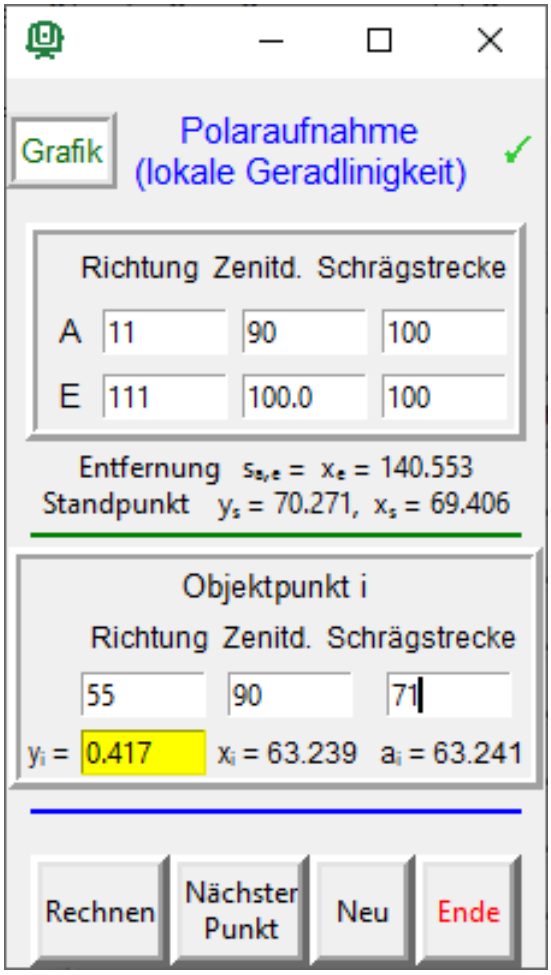


[5] d. h. $s_{\mathrm{a,e}}$ wird im übergeordneten System (i.d.R. UTM) gerechnet, dagegen $x_\mathrm{e}(\mathrm{lokal})$ im örtlichen, unter Berücksichtigung des Reduktionsfaktors r. Die Werte für $s_{\mathrm{a,e}}$ und $x_\mathrm{e}(\mathrm{lokal})$ wie auch $s_{\mathrm{a,i}}$ und $s_{\mathrm{a},i(\mathrm{lokal})}$ müssen deshalb nicht identisch sein.

2 Polaraufnahme (lokale Geradlinigkeit)

Gegeben:

lokale Richtungen und Zenitdistanzen [gon] und Schrägstrecken zum
$r_a, Zenitdistanz_a, s_a$ Anfangspunkt
$r_e, Zenitdistanz_e, s_e$ Endpunkt
$r_i, Zenitdistanz_i, s_i$ Objektpunkt
im örtlichen System
Defaultwerte für Zenitdistanzen = 100 [gon]

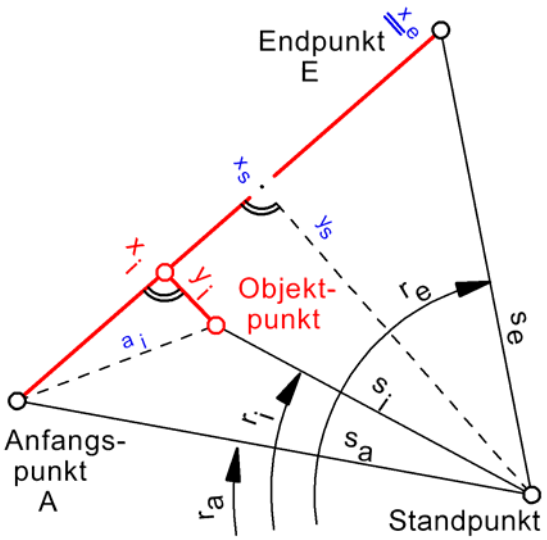


Gesucht:

y_i, x_i (lokale orthogonale Elemente)
(y_i zur Überprüfung der Geradlinigkeit und der Absteckung von Geraden und Parallelen)

Zusätzliche Ergebnisse:

x_e (lokal) Endmaß
y_s, x_s (lokal) Standpunktkoordinaten
a_i (lokal) Strecke



3 Polaraufnahme (Koordinaten)

Gegeben:

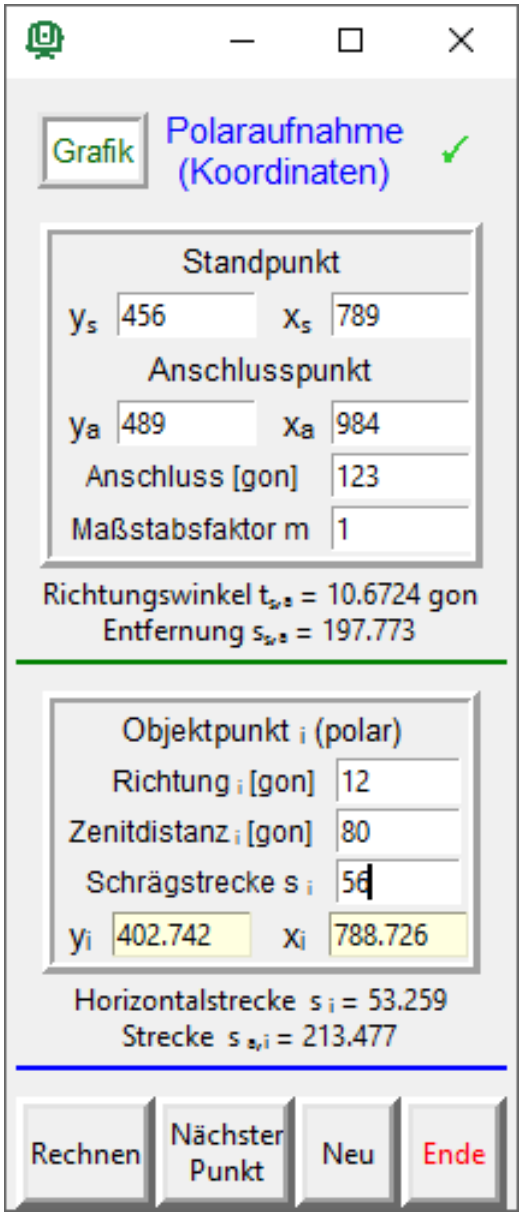
y_s, x_s Standpunkt
y_a, x_a, r_a Anschlusspunkt mit Richtung [gon]
m Maßstabsfaktor (Default = 1)
$r_i, Zenitdistanz_i, s_i$ Objektpunkt mit Richtung und Zenitdistanz [gon] (Default = 100 [gon]) und Schrägstrecke

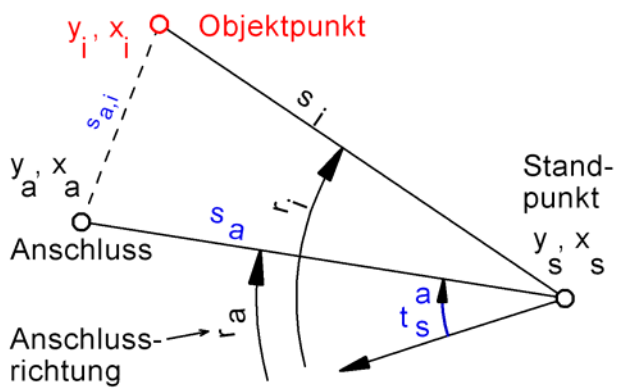
Gesucht:

y_i, x_i Objektpunkt

Zusätzliche Ergebnisse:

t_s^a Anschlussrichtungswinkel [gon] (aus Koordinaten)
$s_{s,a}$ Strecke Stand- zum Anschlusspunkt (Entfernung)
$s_{a,i}$ Strecke Anschluss- zum Objektpunkt





Soll eine örtliche Aufnahme in UTM-Koordinaten berechnet werden, ist zunächst der Reduktions-(Maßstabs-)faktor zu ermitteln und einzutragen.

4 Transformation, Kleinpunkte

Gegeben:

$\mathrm{A}y_\mathrm{a}, \mathrm{A}x_\mathrm{a}$ Anfangspunkt System A
$\mathrm{A}y_\mathrm{e}, \mathrm{A}x_\mathrm{e}$ Endpunkt System A
$\mathrm{A}y_i, \mathrm{A}xi_i$ Objektpunkt

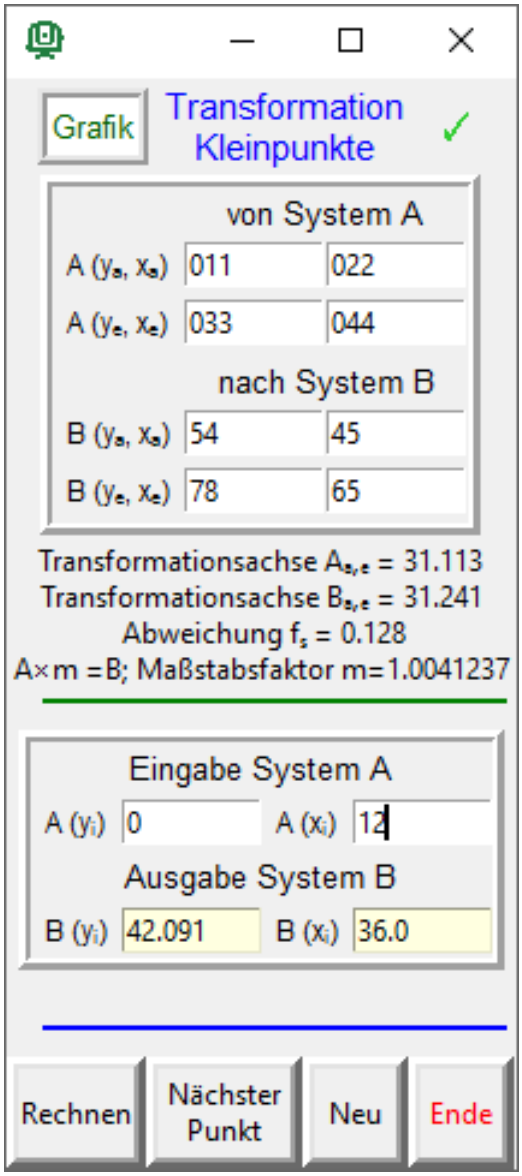
$\mathrm{B}y_\mathrm{a}, \mathrm{B}x_\mathrm{a}$ Anfangspunkt System B
$\mathrm{B}y_\mathrm{e}, \mathrm{B}x_\mathrm{e}$ Endpunkt System B

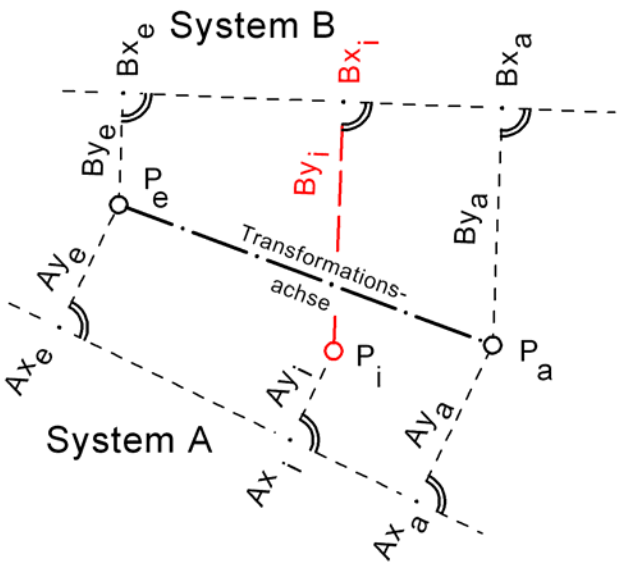
Gesucht:

$\mathrm{B}y_i, \mathrm{B}x_i$ Objektpunkt im System B

Zusätzliche Ergebnisse:

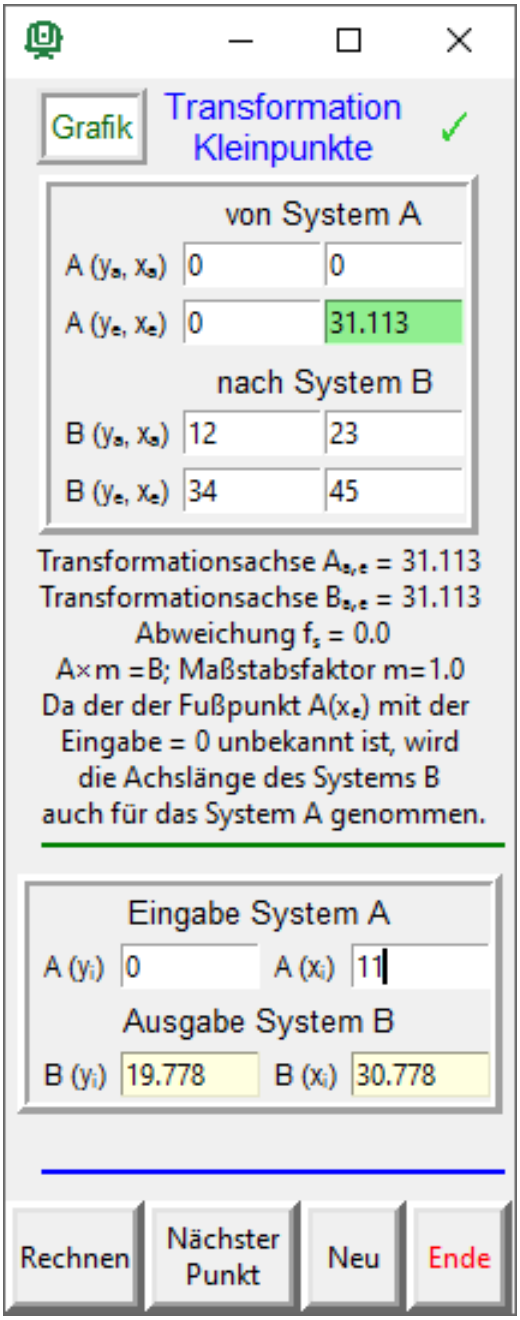
Längen der Transformationsachse
im System A und B $\rightarrow \mathrm{A}s_{\mathrm{a,e}}, \mathrm{B}s_{\mathrm{a,e}}$
f_s = Abweichung $(\mathrm{B}s_{\mathrm{a,e}} - \mathrm{A}s_{\mathrm{a,e}})$
m Maßstabsfaktor



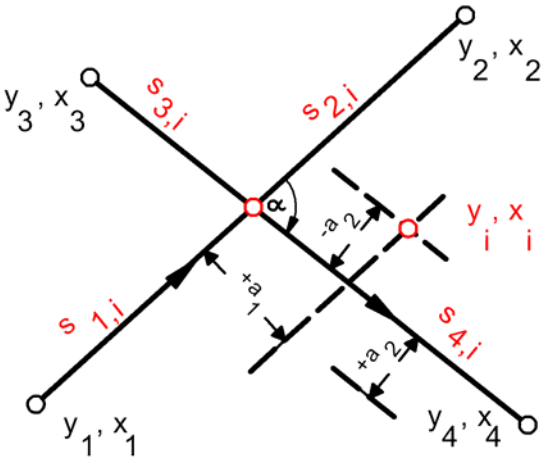


Hiermit können lineare Achstransformationen wie auch übliche Kleinpunktberechnungen durchgeführt werden; Kleinpunktberechnungen derart, dass in den entsprechenden Eingabefeldern Nullen eingetragen werden, die hier i. d. R. als Defaultwerte zur Verfügung stehen.

Nebenstehend ist ein Beispiel einer Kleinpunktberechnung minimalster Form, mit nicht bekanntem Endmaß $\mathrm{A}x_\mathrm{e}$ im System A (Eingabe $\mathrm{A}x_\mathrm{e} = 0$) und mit vielen Nullen als Defaultwerte.



5 Geradenschnitte



Gegeben:

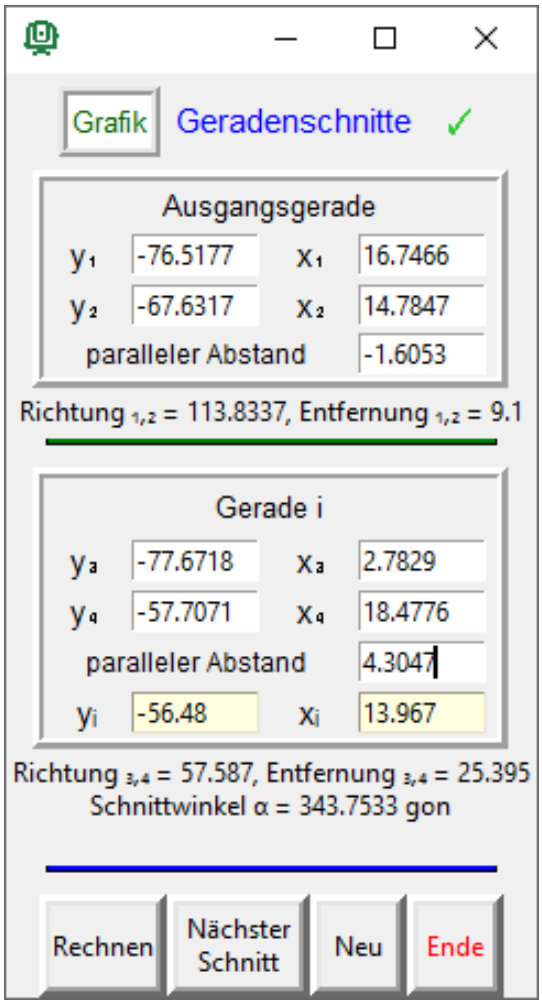
$(y_1, x_1) \rightarrow (y_2, x_2)$ Gerade 1
$\pm a_1$ Abstand
$(y_3, x_3) \rightarrow (y_4, x_4)$ Gerade 2
$\pm a_2$ Abstand

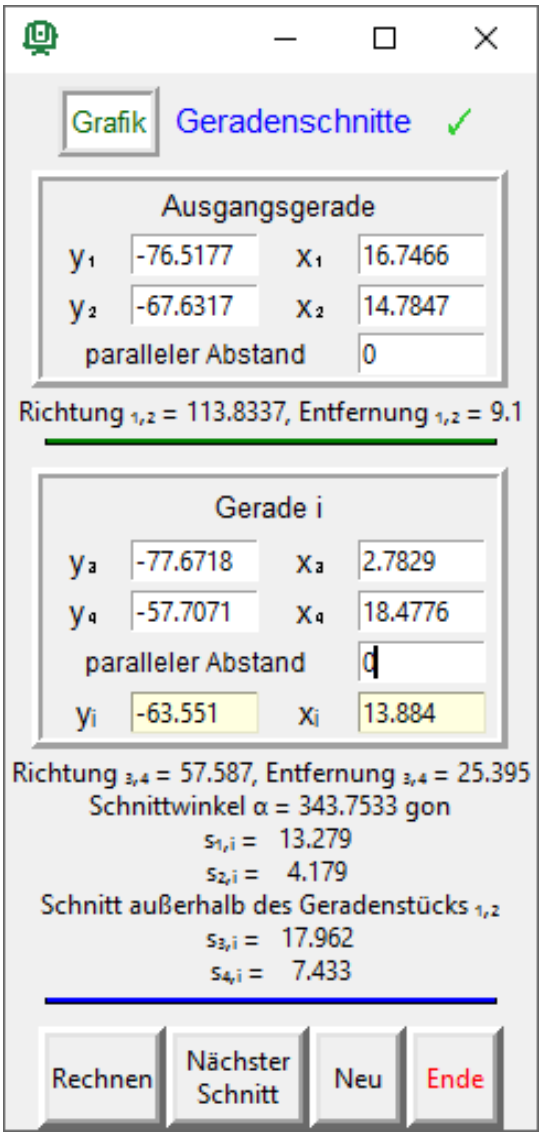
Gesucht:

y_s, y_s Schnittpunkt

Zusätzliche Ergebnisse:

α Schnittwinkel [gon]
$s_{1,i}, s_{2,i}, s_{3,i}, s_{4,i}$ Streckenanteile, sofern keine Abstände gegeben sind

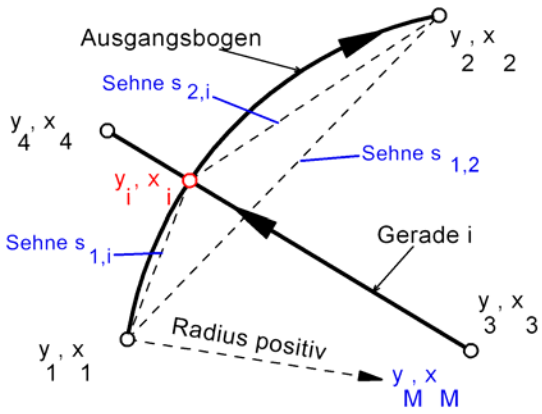




6 Kreis-Geraden-Schnitt

Gegeben:

y_1, x_1 Anfang des Bogens (rechtsläufig)
y_2, x_2 Ende des Bogens
$Radius$ +
$(y_3, x_3) \rightarrow (y_4, x_4)$ Gerade

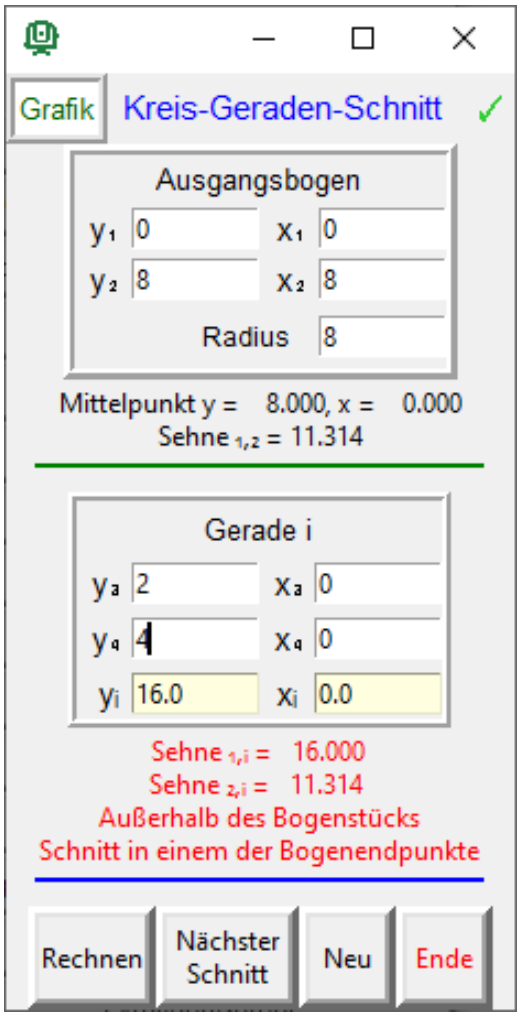


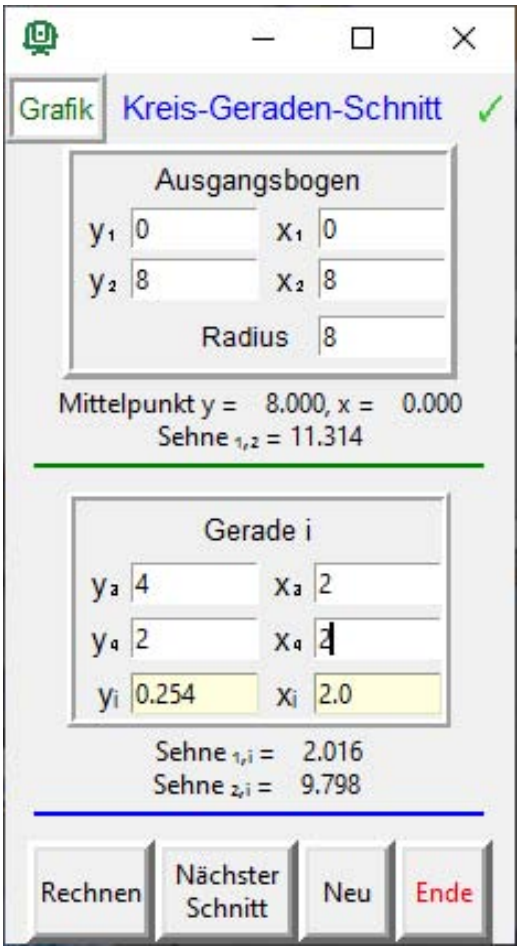
Gesucht:

y_i, y_i Schnittpunkt

Zusätzliche Ergebnisse:

$s_{1,i}, s_{2,i}, s_{1,2}$ Sehnenlängen
y, x Mittelpunktkoordinaten



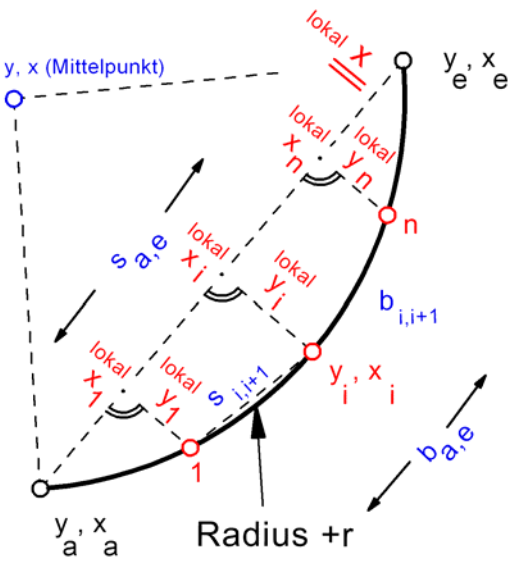


Werden beide Schnittpunkte benötigt, sind im nächsten Durchlauf (y_3, x_3) und (y_4, x_4) zu vertauschen.

7 Kreisbogenteilung

Gegeben[6]:

y_a, x_a Anfang des Bogens (linksläufig)
y_e, x_e Ende des Bogens
oder $s_{a,e}$
m Maßstabsfaktor (Default = 1)
$Radius$ +
n Anzahl der Zwischenpunkte (für $n < 10$)

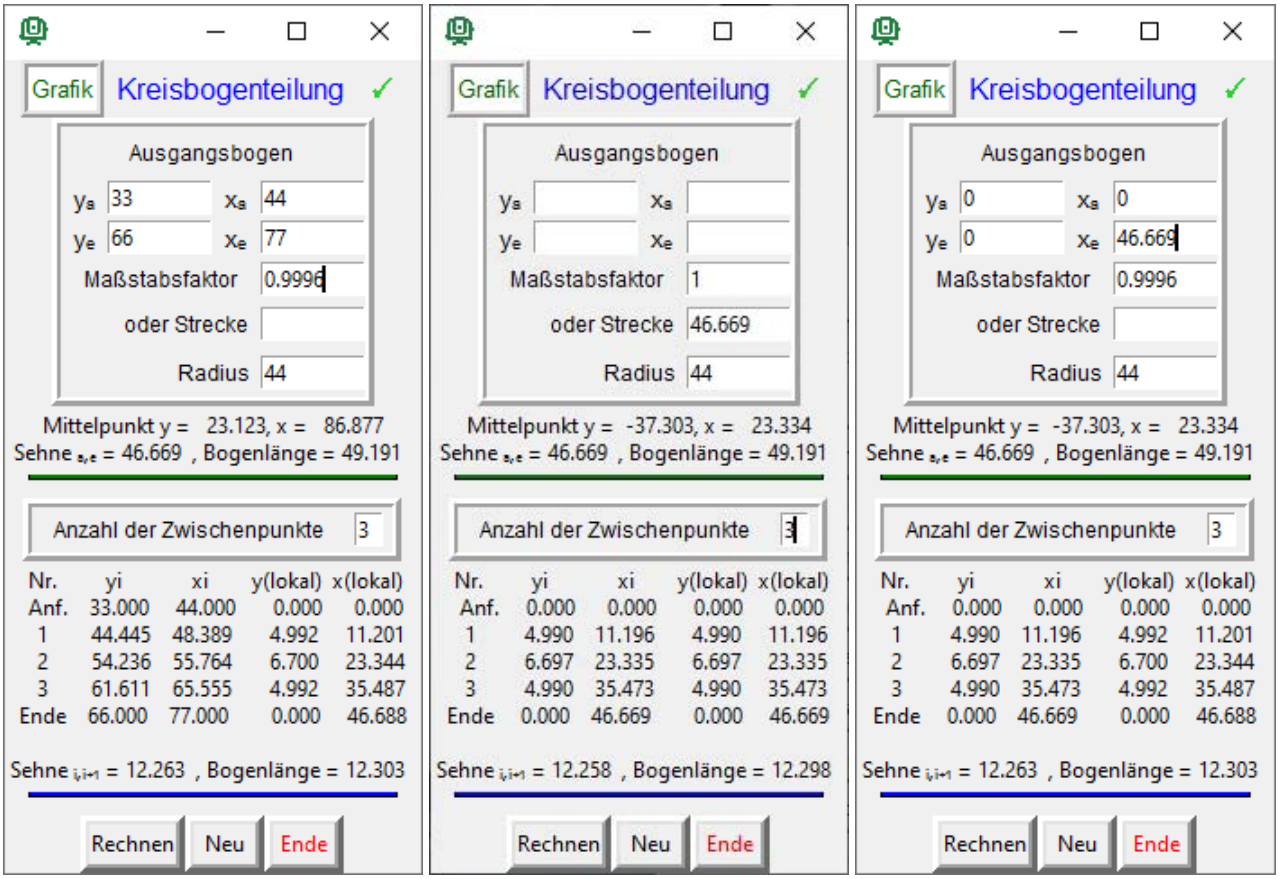


Gesucht:

y_i, x_i Objektpunkt

Zusätzliche Ergebnisse:

$s_{a,e}, s_{i,i+1}$ Sehnenlängen
$b_{a,e}, b_{i,i+1}$ Bogenlängen
y, x Mittelpunktkoordinaten



a) b) c)

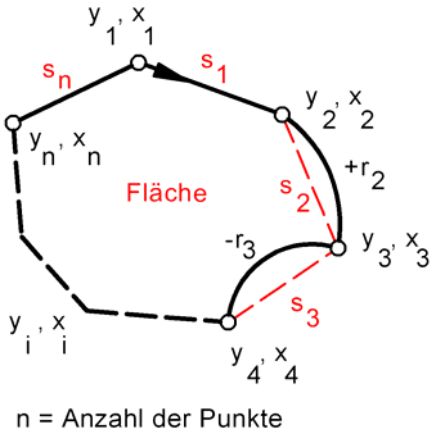
[6]Damit die lokalen Ordinaten positiv werden, wurde der Bogen linksläufig eingeführt.

Sollen örtliche Absteckungselemente aus UTM-Koordinaten berechnet werden, ist zunächst der Reduktions-(Maßstabs-)faktor zu ermitteln und neben den Koordinaten des Ausgangsbogens einzugeben [Beispiel a)]; lt. der Definition (s. Seite 4 und 18):

$$Elemente(\text{horizontal}, \text{lokal}) \Longleftrightarrow Koordinaten(\text{UTM}) : r \text{ (bzw. } m).$$

Dagegen wird die Eingabe einer Ausgangsstrecke **unbedingt vorrangig** gegenüber den Eingabekoordinaten berücksichtigt und erzeugt automatisch einen Maßstabsfaktor 1. y, x sind dann örtliche, lokale Koordinaten des Mittelpunktes [Beispiel b)]. Liegt eine UTM-Strecke als Sehne vor und sollen örtliche, lokale Elemente berechnet werden, sind neben dem Reduktionsfaktor folgende Werte einzugeben: $y_a, x_a, y_e = 0$ und die Strecke als x_e [Beispiel c)].

8 Flächenberechnung

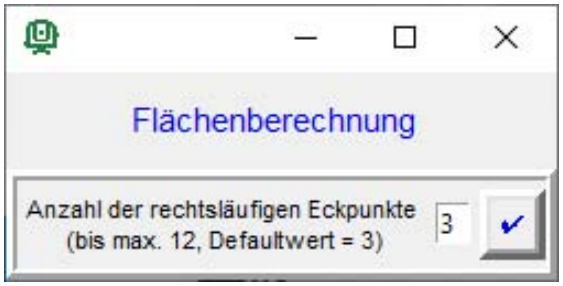


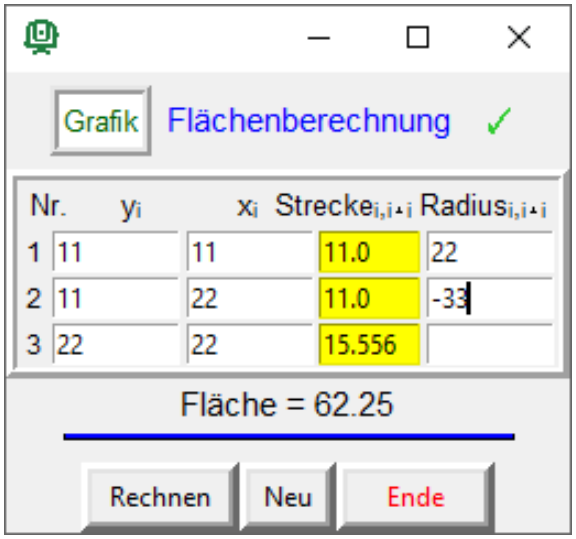
Gegeben:

y_i, x_i rechtsläufige n Eckpunkte
$+Radius \rightarrow$ Das Kreissegment wird addiert.
$-Radius \rightarrow$ Das Kreissegment wird subtrahiert.
n Anzahl der rechtsläufigen Eckpunkte
$(3 \leq n \leq 12)$

Gesucht:

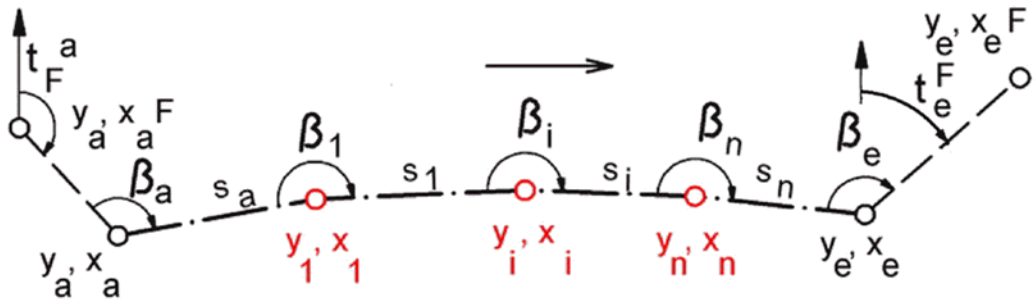
$Fläche$
s_i Seiten- bzw. Sehnenlängen





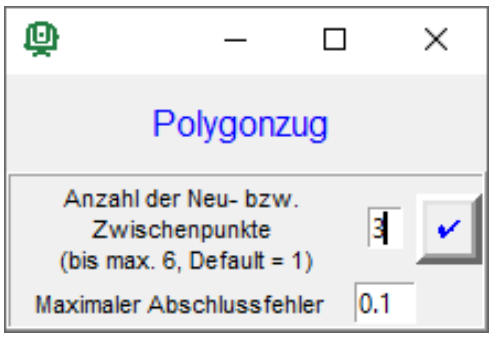
Bei eine aus örtlichen Koordinaten ermittelten Fläche auf eine beispielsweise UTM-Bezugsfläche zu transformieren, ist diese mit dem Quadrat des Reduktionsparameters zu multiplizieren.

9 Polygonzug



Gegeben (je nach Variante):

$y_\mathrm{a}^\mathrm{F}, x_\mathrm{a}^\mathrm{F}$ Fernziel (Koordinaten)
oder t_F^a Anschlussrichtungswinkel
$y_\mathrm{a}, x_\mathrm{a}$ Anfangspunkt
$y_\mathrm{e}, x_\mathrm{e}, y_\mathrm{e}^\mathrm{F}, x_\mathrm{e}^\mathrm{F}$ Endpunkt, Fernziel
oder t_e^F Abschlussrichtungswinkel
$\beta_\mathrm{a}, \beta_i, \beta_\mathrm{e}$ Brechungswinkel [gon]
n Anzahl der Neu- bzw. Zwischenpunkte $(1 \leq n \leq 6)$
$maximaler Abschlussfehler$
$s_\mathrm{a}, \mathrm{s}_i$ Polygonseiten $(i = 1$ nach $n)$
Maßstabsfaktor (Default = 1) **vor** der Transformation



Gesucht:

y_i, x_i

Zusätzliche Ergebnisse:

Abweichungen des Koordinatenabschlusses f_y, f_x
Winkelschlussabweichung $f\beta$, Längsabweichung L, Querabweichung W
Maßstabsfaktor nach der Transformation

Das Programm erfordert zwei Durchläufe. Beim ersten Durchlauf werden vorläufige Koordinaten berechnet, auch zur Anzeige von Längs-, Quer- und Winkelschlussabweichung [Beispiel a)] je nach Eingabevariante.

Endgültige Koordinaten werden im zweiten Durchlauf [Beispiel b)] durch eine Transformation (Drehung und Streckung) erzeugt, womit eine Verteilung der Abweichungen erfolgt. Werden nach dieser Transformation $\mathrm{f}_y = 0, \mathrm{f}_x = 0, L = 0, W = 0$ angezeigt, ist die Transformation korrekt verlaufen. Der zweite, grüne Haken neben der Titelüberschrift <Polygonzug> kennzeichnet die Durchführung des zweiten Programmdurchlaufs. Die

anfängliche Eingabe eines maximalen Abschlussfehlers ist aus Sicherheitsgründen erforderlich [Beispiel c)].

Die Eingabefelder dienen teilweise gleichzeitig zur Programmsteuerung und bieten Berechnungs**varianten**:

- Der Anschluss kann wahlweise über den Anschlussrichtungswinkel $t_{\mathrm{F}}^{\mathrm{a}}$ oder über die Koordinaten von $y_{\mathrm{a}}^{\mathrm{F}}, x_{\mathrm{a}}^{\mathrm{F}}$ erfolgen. Treten beide Angaben auf, wird der Anschlussrichtungswinkel $t_{\mathrm{F}}^{\mathrm{a}}$ gegenüber den Koordinaten von $y_{\mathrm{a}}^{\mathrm{F}}, x_{\mathrm{a}}^{\mathrm{F}}$ vorrangig bearbeitet. Das gilt analog auch für den Abschluss bezüglich $y_{\mathrm{e}}^{\mathrm{F}}, x_{\mathrm{e}}^{\mathrm{F}}$ bzw. $t_{\mathrm{e}}^{\mathrm{F}}$.

- $\beta_{\mathrm{a}} = 0; \Longrightarrow$ ohne Winkelanschluss (Eingaben $y_{\mathrm{a}}^{\mathrm{F}}, x_{\mathrm{a}}^{\mathrm{F}}$ bzw. $t_{\mathrm{F}}^{\mathrm{a}}$ sind unnötig)

  Wegen der fehlenden Anschlussrichtung wird hier programmintern eine fingierte in Richtung des Endpunktes genommen. Die vor der Transformation ermittelte Werte für f_y, f_x, L, W sind deshalb **irregulär**. Ringpolygone können nicht gerechnet werden.

- $\beta_e = 0; \Longrightarrow$ ohne Winkelabschluss (Eingaben $y_{\mathrm{e}}\mathrm{F}, x_{\mathrm{e}}\mathrm{F}$ sind unnötig)

- y_{e} und $x_{\mathrm{e}} = 0; \Longrightarrow$ ohne Koordinatenabschluss (Eingaben von $\beta_{\mathrm{e}}, y_{\mathrm{e}}^{\mathrm{F}}, x_{\mathrm{e}}^{\mathrm{F}}$ sind unnötig)
  y_{e} und x_{e} werden berechnet.

- $y_{\mathrm{a}} = y_{\mathrm{e}}$ und $x_{\mathrm{a}} = x_{\mathrm{e}}; \Longrightarrow$ Ringpolygon

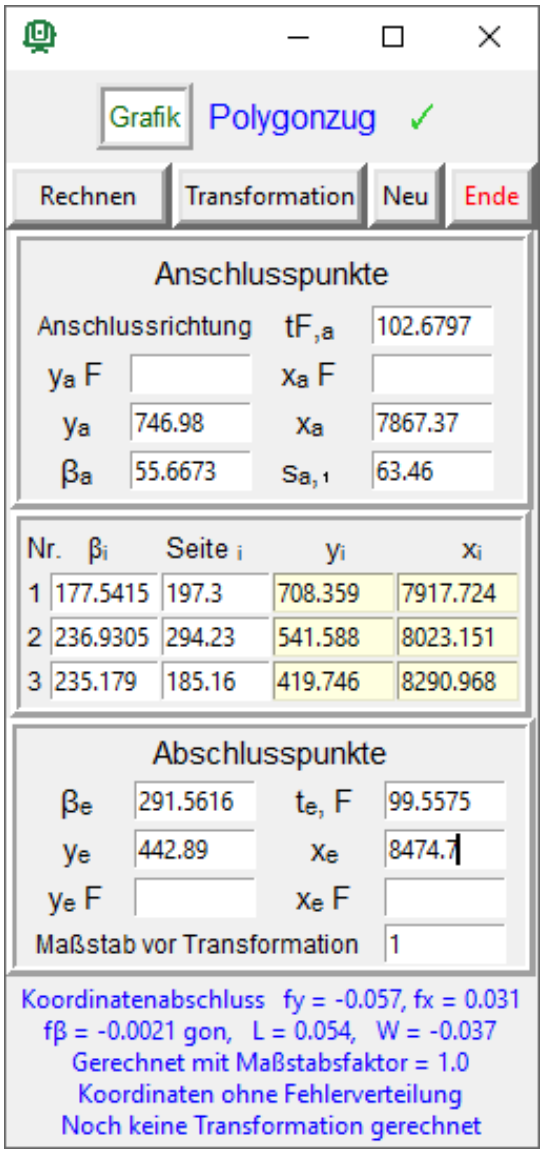
  (Anschlüsse mit $y_{\mathrm{a}}^{\mathrm{F}}, x_{\mathrm{a}}^{\mathrm{F}}$ oder $t_{\mathrm{F}}^{\mathrm{a}}$ und $\beta_{\mathrm{a}} \neq 0$ müssen gegeben sein.)

  $\Longrightarrow$ Ringpolygone sollten möglichst, wegen der unglücklichen Verteilung der Abweichungen und nicht machbarer Maßstabskontrolle vermieden werden. Eine Transformation zu deren Verteilung kann in diesem Fall nicht gerechnet werden. Bei der Betätigung der Taste <Transformation> erfolgt die Verteilung der Abweichungen dann proportional zur Länge der Polygonseiten: $\mathrm{d}(\Delta y_i) = \frac{s_i}{[s]} * fy$ und $\mathrm{d}(\Delta x_i) = \frac{s_i}{[s]} * fx$.

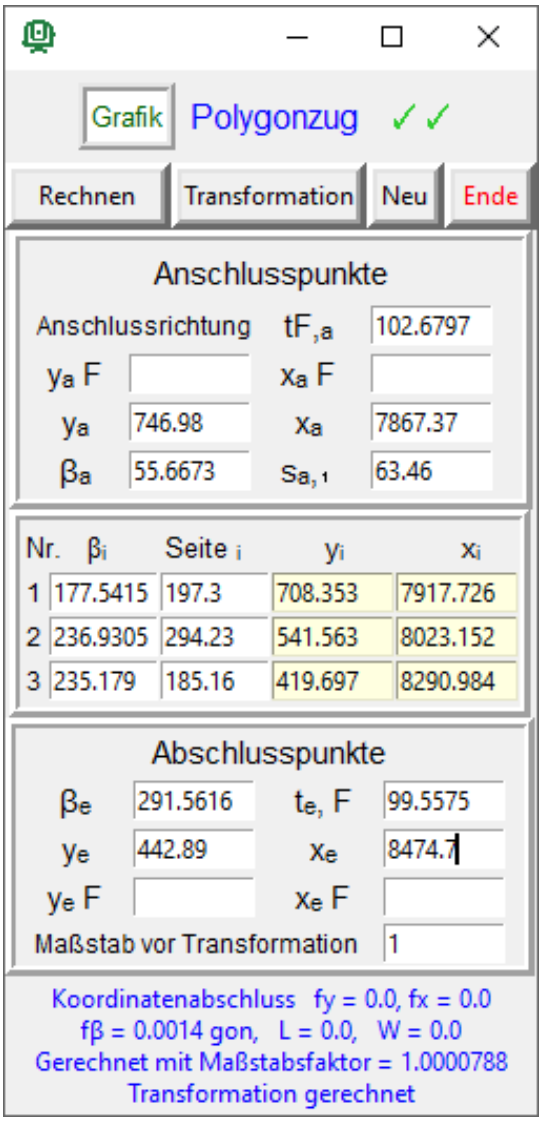
Eine einfache Fehleranalyse kann durchgeführt werden, wenn der Polygonzug nicht mit dem Defaultwert von 1 für den Maßstabsfaktor gerechnet wird, sondern zunächst als Maßstabsfaktor die zuvor ermittelte Maßstabsverzerrung (lokal nach UTM) genommen wird. Erst im zweiten Durchlauf erfolgt die endgültige Verteilung der Abweichungen dann durch die Transformation.

Vor (eingegeben) und nach (berechnet) der Transformation:

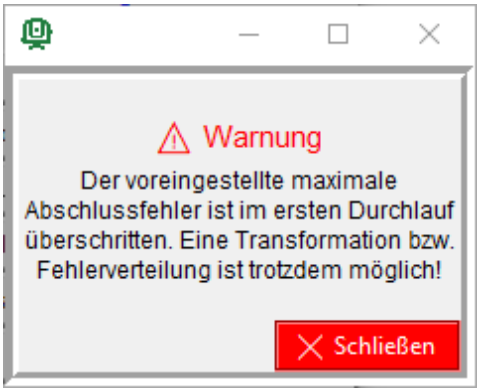
$f\beta =$ Winkelabschlussabweichung [gon] $\Longrightarrow t_{\mathrm{e,Soll}} - t_{\mathrm{e,Ist}}$
$f_y, f_x =$ Abweichungen des Koordinatenabschlusses
$L =$ Längsabweichung
$W =$ lineare Querabweichung
$f_s =$ lineare Abschlussabweichung $\Longrightarrow \sqrt{f_y^2 + f_x^2} = \sqrt{L^2 + W^2}$
(Vor der Transformation erfolgt keine Verteilung der Winkelabschlussabweichung;
eine Verteilung der Winkelabweichung kann zu einer Verbiegung des Zuges führen.)
Maßstabsfaktor, $\Longrightarrow L = (q - 1) * s_{\mathrm{a,e}}$


Grafik
Polygonzug ✓
Rechnen
Transformation
Neu
Ende
Anschlusspunkte
Anschlussrichtung tF,a 102.6797
ya F
xa F
ya 746.98
xa 7867.37
βa 55.6673
Sa,1 63.46
Nr. βi Seite i yi xi
1 177.5415 197.3 708.359 7917.724
2 236.9305 294.23 541.588 8023.151
3 235.179 185.16 419.746 8290.968
Abschlusspunkte
βe 291.5616
te, F 99.5575
ye 442.89
xe 8474.7
ye F
xe F
Maßstab vor Transformation 1
Koordinatenabschluss fy = -0.057, fx = 0.031
fβ = -0.0021 gon, L = 0.054, W = -0.037
Gerechnet mit Maßstabsfaktor = 1.0
Koordinaten ohne Fehlerverteilung
Noch keine Transformation gerechnet

a)

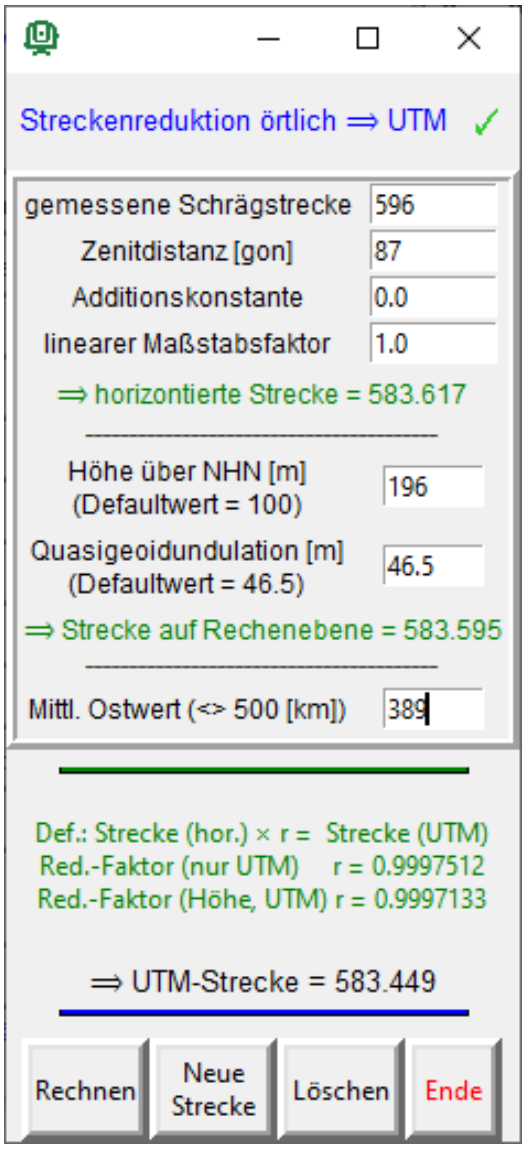

Grafik
Polygonzug ✓✓
Rechnen
Transformation
Neu
Ende
Anschlusspunkte
Anschlussrichtung tF,a 102.6797
ya F
xa F
ya 746.98
xa 7867.37
βa 55.6673
Sa,1 63.46
Nr. βi Seite i yi xi
1 177.5415 197.3 708.353 7917.726
2 236.9305 294.23 541.563 8023.152
3 235.179 185.16 419.697 8290.984
Abschlusspunkte
βe 291.5616
te, F 99.5575
ye 442.89
xe 8474.7
ye F
xe F
Maßstab vor Transformation 1
Koordinatenabschluss fy = 0.0, fx = 0.0
fβ = 0.0014 gon, L = 0.0, W = 0.0
Gerechnet mit Maßstabsfaktor = 1.0000788
Transformation gerechnet

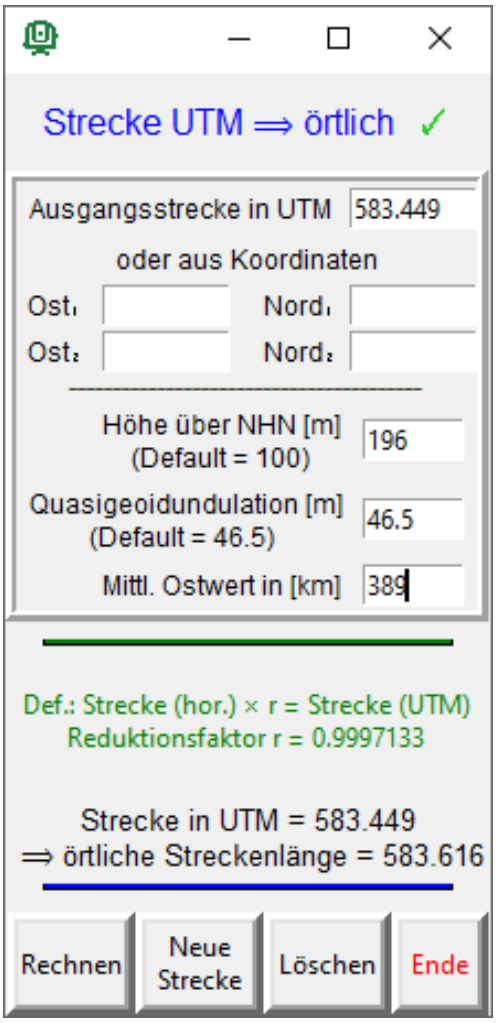
b)


Warnung
Der voreingestellte maximale Abschlussfehler ist im ersten Durchlauf überschritten. Eine Transformation bzw. Fehlerverteilung ist trotzdem möglich!
Schließen

c)

10 Streckenreduktion örtlich ⇒ UTM
11 Strecke UTM ⇒ örtlich



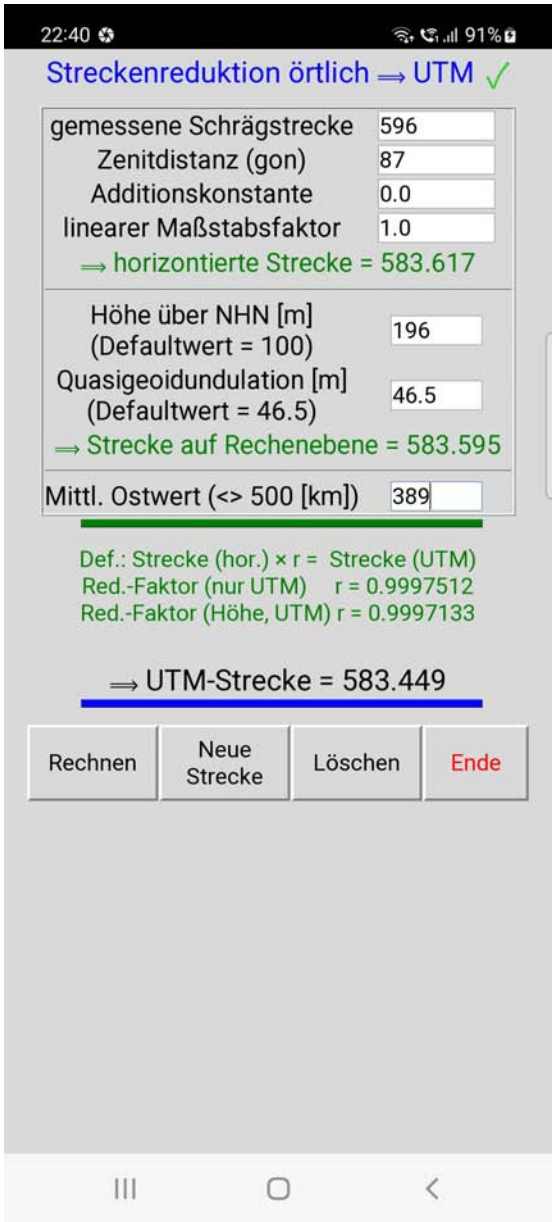


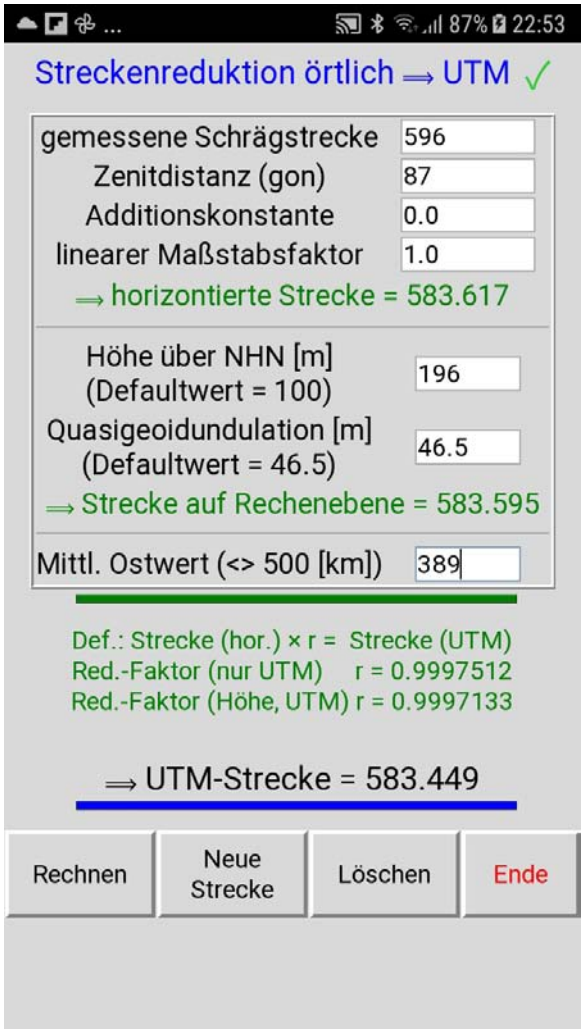
$$S_{[\text{örtlich}]} = \frac{\sqrt{\left((Ost_2 - Ost_1)^2 + (Nord_2 - Nord_1)^2\right)} = S_{[\text{UTM}]}}{\left(1 + \frac{\left(Ost_{\text{m[km]}} - 500\right)^2}{2R^2_{[\text{km}]}} - \frac{h_{\text{NHN[m]}} + h_{\text{Undulation[m]}}}{R_{[\text{km}]} * 1000}\right) * 0.9996} = r$$

Eine nicht eingegebene Höhe über NHN erzeugt einen Defaultwert von 100. Eine fehlerhafte Höhenangabe von 100 m würde an der Streckenreduktion bei 100 m einen Fehler von 1,6 mm bewirken.

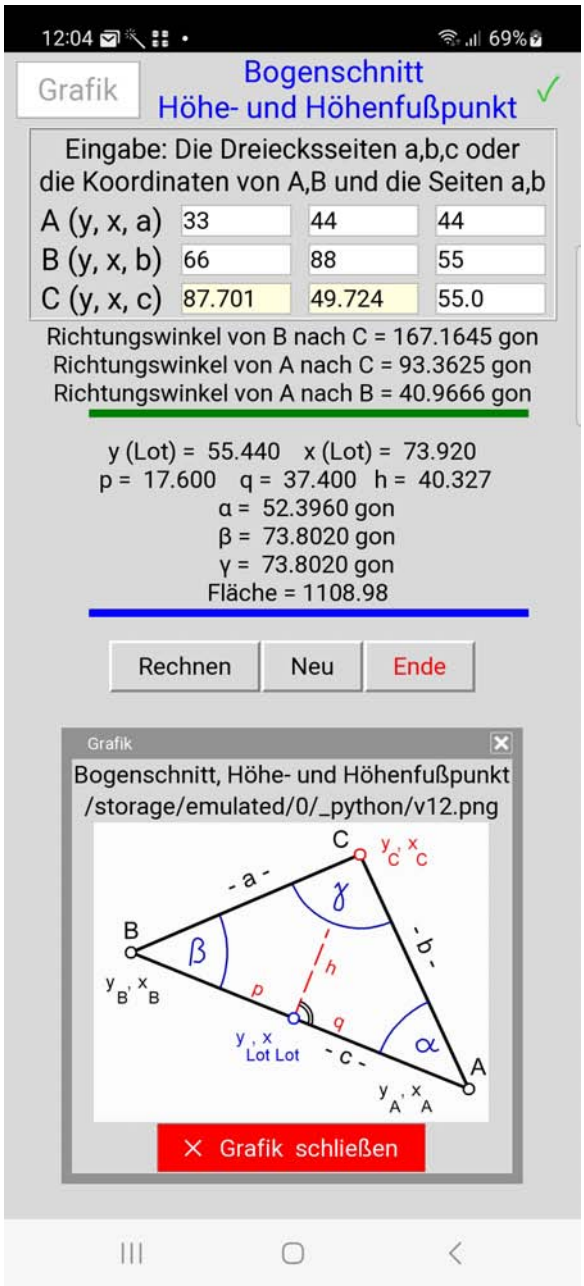
Für die Undulation wurde ein veränderbarer Defaultwert von 46.5 vorgegeben (Kölner Dom). Intern wurde ein mittlerer Erdradius von 6381 km verwendet (NRW).

Ein mittlerer Ostwert in [km] ist unbedingt einzugeben. Bei einem mittl. Ostwert von 319.5 bzw. 680.5 ist die Abbildungsverzerrung = 0.





Screenshot von Samsung Galaxy A52s 5G (links) und Samsung Galaxy J5 (rechts)



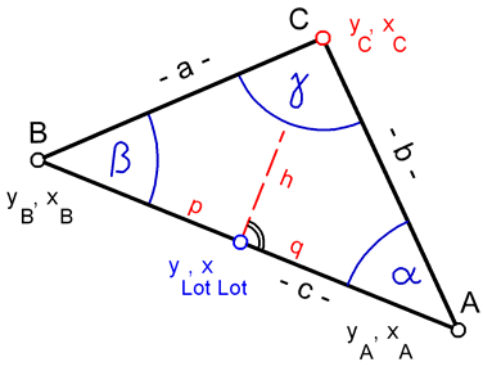
Screenshot von Samsung Galaxy Tab S4 (links) und von Samsung Galaxy A52s 5G (rechts)

12 Bogenschnitt

Höhe und Höhenfußpunkt

Gegeben:

y_A, x_A, y_B, x, a, b Koordinaten und Seiten
oder
a, b, c Dreiecksseiten

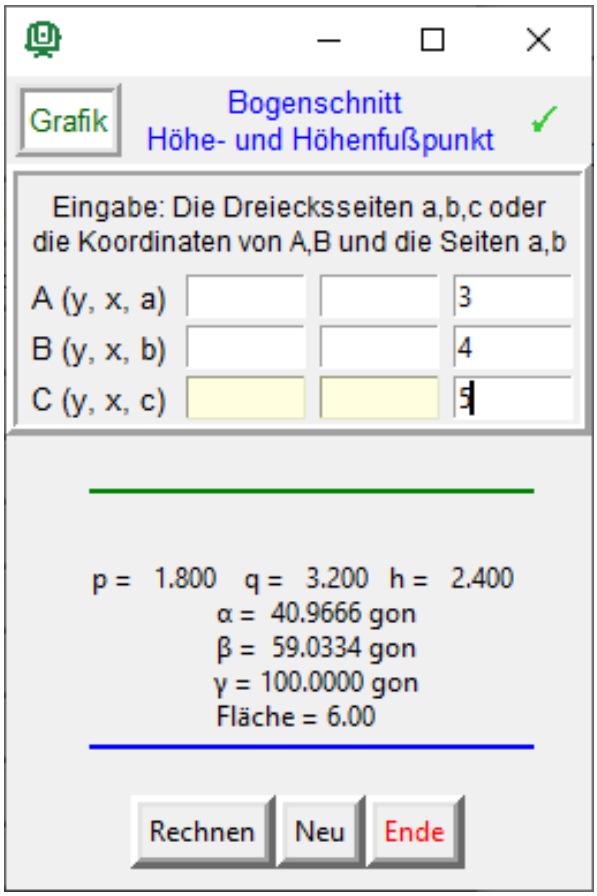


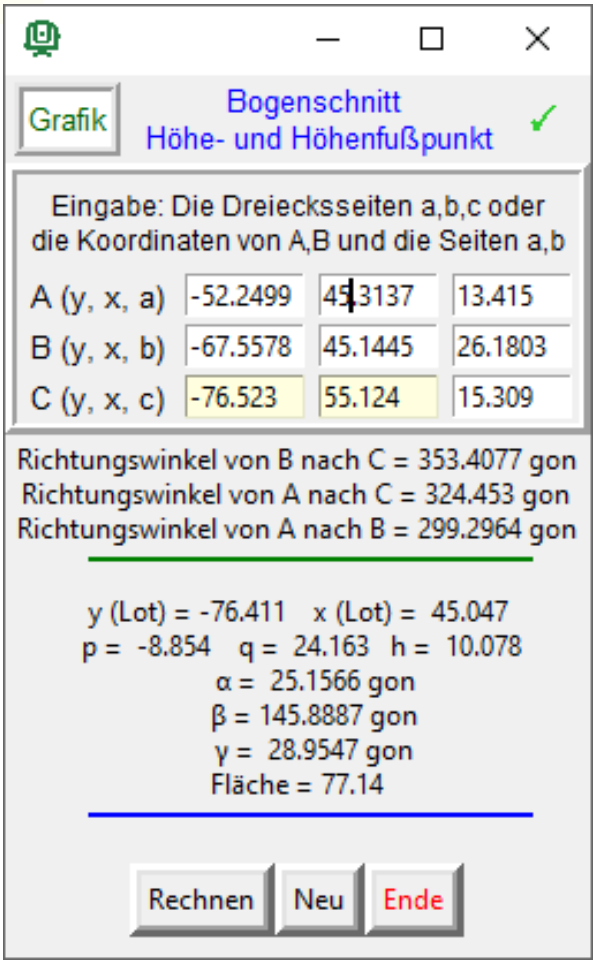
Gesucht:

y_C, x_C Schnittpunkt C
p, q, h Höhe und Höhenfußpunkt

Zusätzliche Ergebnisse:

y_{Lot}, x_{Lot} Fußpunktkoordinaten
eventuell auch c
α, β, γ Dreieckswinkel [gon]
$Fläche$





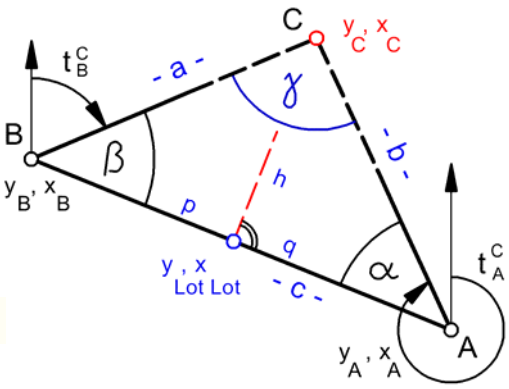
Eine mögliche Berechnung mit y_A, x_A, y_B, x_B, a, b (Koordinaten und Seiten) ist vorrangig gegenüber der Berechnung alleinig aus den Dreiecksseiten a, b, c .

13 Vorwärtsschnitt

über Dreiecks- oder Richtungswinkel

Gegeben:

y_A, x_A, y_B, x_B Koordinaten
α, β Dreieckswinkel [gon]
oder
t_A^C, t_B^C Richtungswinkel [gon]

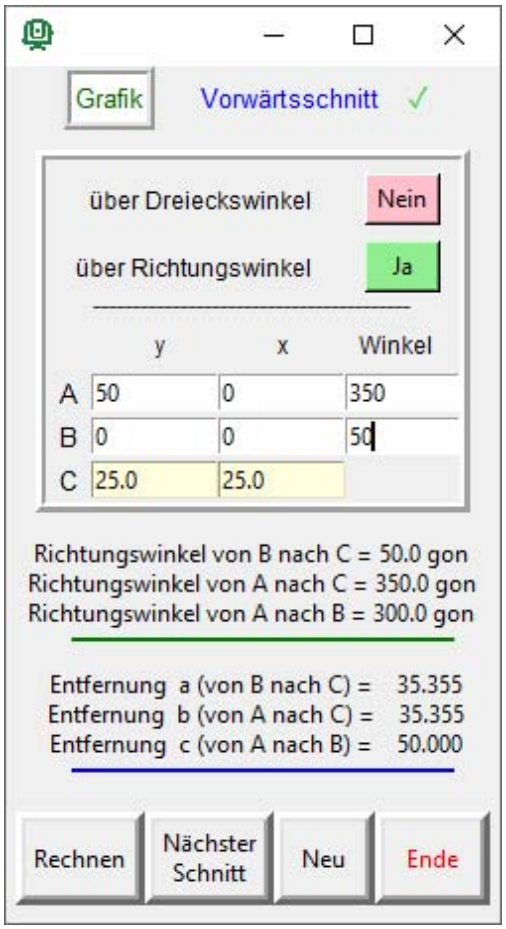


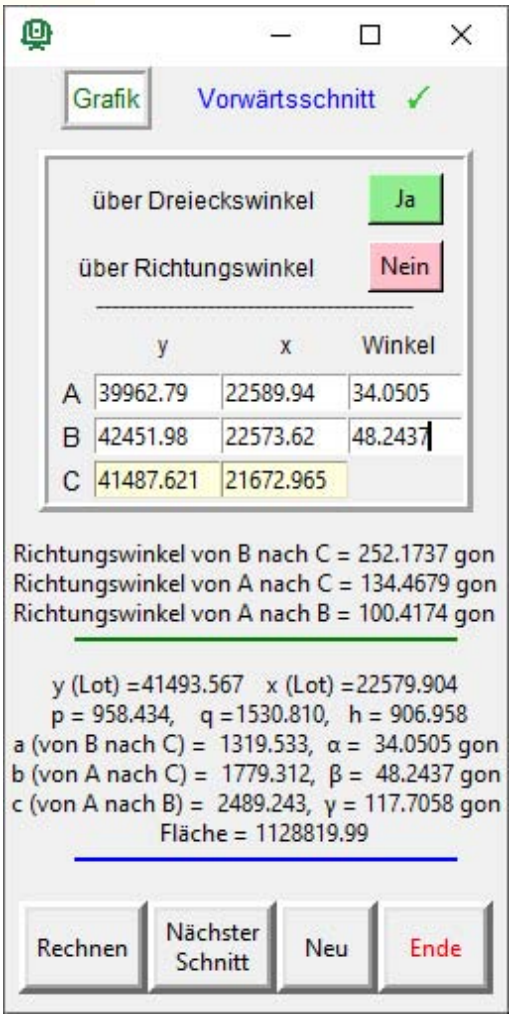
Gesucht:

y_C, x_C Schnittpunkt C

Zusätzliche Ergebnisse:

α, β, γ Dreieckswinkel [gon]
a, b, c (Basis) Dreiecksseiten [gon]
p, q, h Höhe und Höhenfußpunkt
y_{Lot}, x_{Lot} Fußpunktkoordinaten
Fläche





14 Pythagorasprobe

Gegeben:

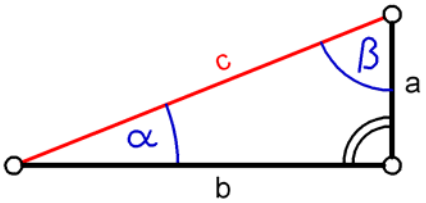
a, b Katheten

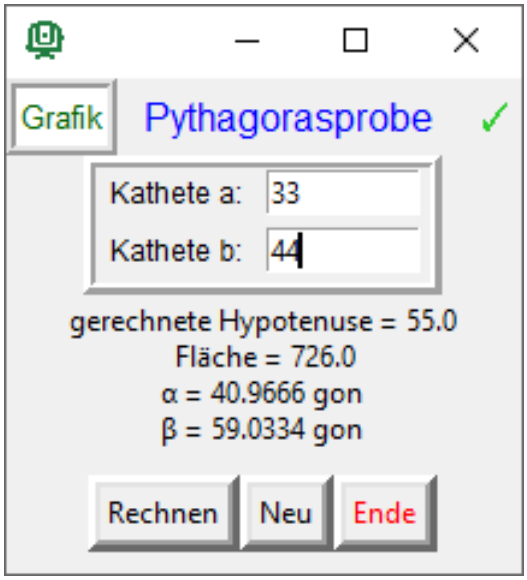
Gesucht:

c Hypotenuse

Zusätzliche Ergebnisse:

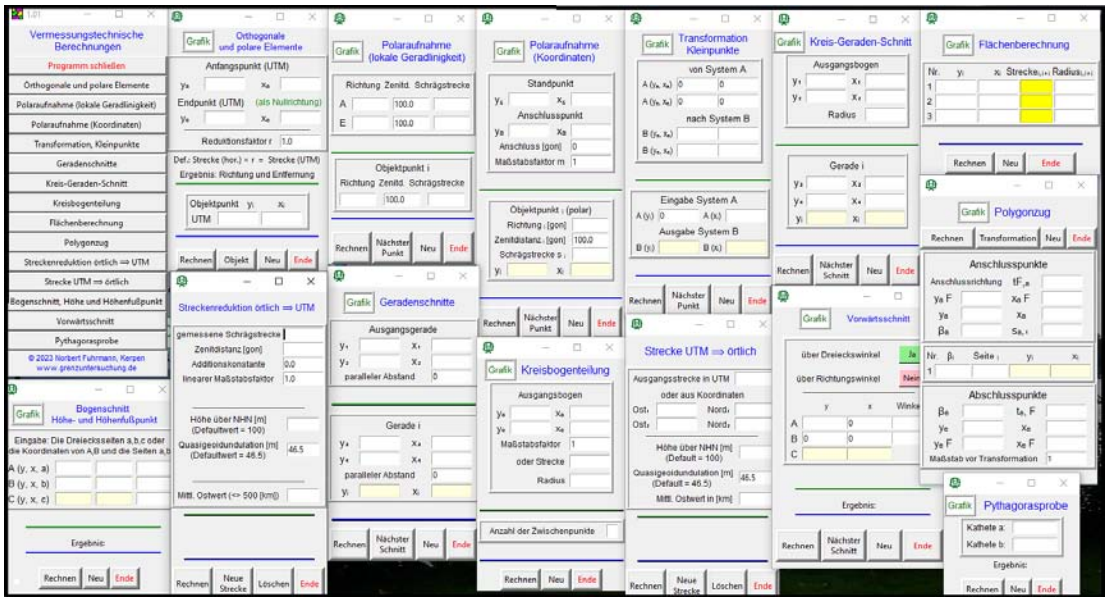
$Fläche$
α, β Dreieckswinkel





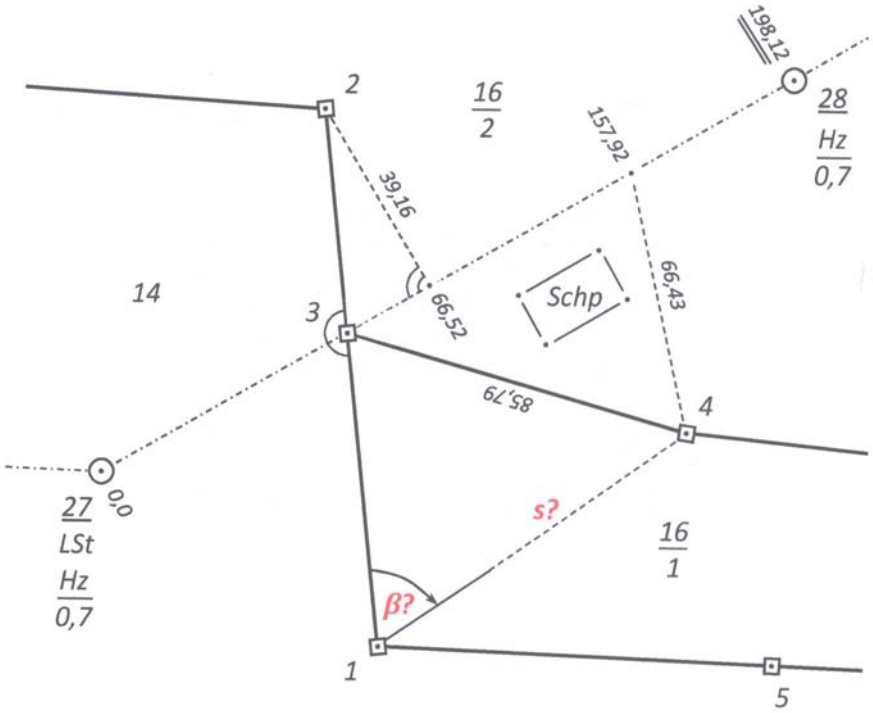
Das Programm wurde getestet mit

Samsung Galaxy Tab S4 – Android Vers. 10, python 3.9.7 Oct 6 2021
Samsung Galaxy A52s 5G (1080x2400 Pixel) – Android Vers. 12
Samsung Galaxy J5 (2015) (720x1480 Pixel) – Android Vers. 7.1.1
Windows 10 Pro 21H1



Nachtrag: Anhang Beispiel

Das Beispiel entstammt aus dem Buch von **Sieland, Sammlung vermessungstechnischer Aufgaben mit ausführlichen Lösungen**, 3. Auflage 2022, Verlag Wichmann, Heidelberg: Beispiel Aufgabe 9.3 auf Seite 84. Lösung der Aufgabe daselbst auf Seite 199f.

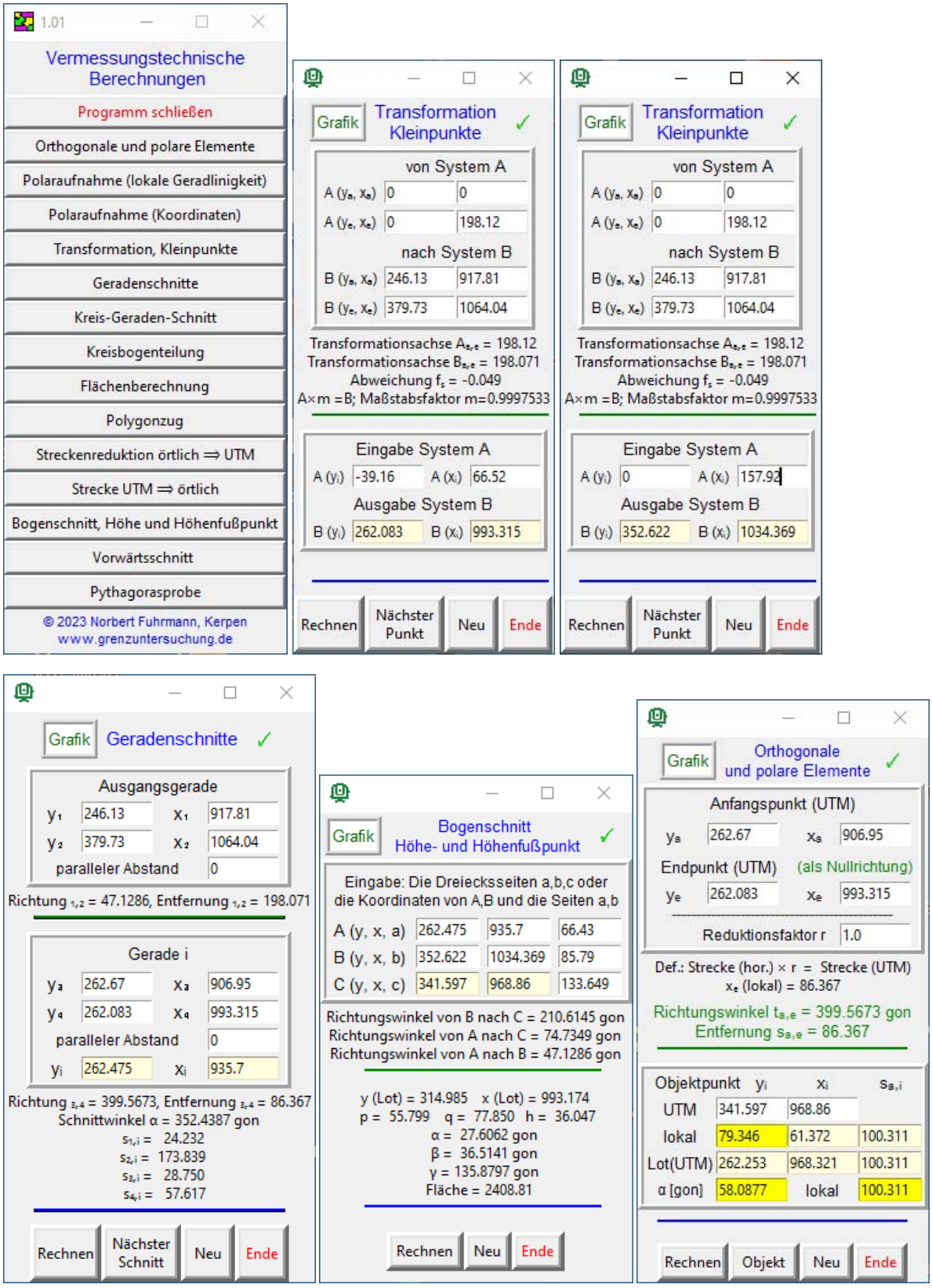


In der Örtlichkeit wurden nur die Grenzpunkte 1 und 2 vorgefunden. Vom Standpunkt 1 soll der Grenzpunkt 4 wiederhergestellt werden. Er soll polar abgesteckt werden; als Nullrichtung ist der Punkt 2 zu verwenden.

Rechts	Hoch	Punkt-Nr
246.13	917.81	27
379.73	1064.04	28
262.67	906.95	1

Die polaren Absteckungselemente sind zu berechnen.

Das Beispiel (s. folgende Seite) wurde in Windows mit zugleich geöffneten Programminstanzen (Tasks) berechnet. Gleiche oder unterschiedliche Instanzen der Programmteile können gleichzeitig geöffnet, darin gerechnet oder geschlossen werden. Daten können wahlweise mit Strg-C bzw. Strg-V in verschiedene Instanzen übertragen werden.



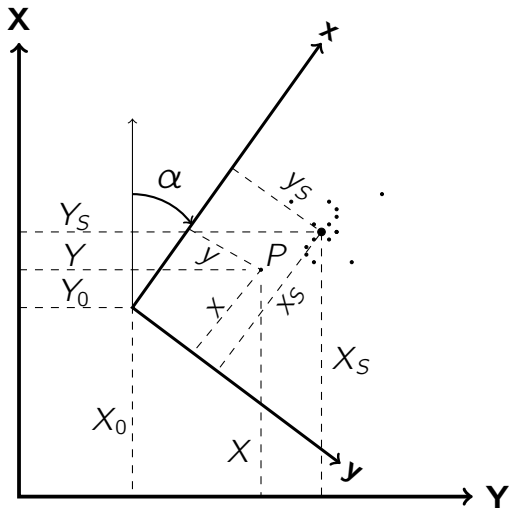
Ergebnis der Aufgabe 9.3: Standpunkt: 1

$$\beta_2 = 0,0000gon, s_2 = 86,367m$$
$$\beta_4 = 58,0877gon, s_4 = 100,311m$$

Helmerttransformation

Programm für Windows auf Python-Basis
Version 1.02

Norbert Fuhrmann, Kerpen



© 2023 Norbert Fuhrmann, Kerpen (All Rights Reserved) (Gesetzt in LaTeX)
www.grenzuntersuchung.de

Programminformation

Das Programm ist in Standardpython geschrieben. Python ist eine höhere, interpretierende Programmiersprache von der <Python Software Foundation>. In Python geschriebener Code lässt sich in diversen Betriebssystemen anwenden. Der Python-Interpreter <IDLE Shell 3.11.0> für Windows kann im Internet unter https://www.python.org/downloads/ heruntergeladen werden.

Das Programm <Helmerttransformation.pyw> ist mit Hilfe des Python-Interpreters auf einen PC mit **Windows** lauffähig. Oder man nimmt das kompilierte Programm <Helmerttransformation.exe>. Dieses Programm ist ausschließlich für Windows gedacht; dies wegen der Ausgabe am besten auf dem Desktop. Es wurde mit <pyinstaller> Version 5.6.2 aus dem Python-Code erzeugt. Unter <Android> ist es nicht einsatzfähig; für die Steuerung des Programms würde eine Funktion erforderlich, die von einem android-kompatiblen Python-Interpreter nicht unterstützt wird.

Nach Programmaufruf können Daten manuell in eine Tabelle eingetragen und editiert werden, wobei die Eingangswerte in der Tabelle nach einem erfolgreichen Berechnungsdurchlauf durch eine Datenausgabe in eine Datei gesichert werden kann. Oder aber die Eingabe der Daten erfolgt von vornherein über eine Datei mit entsprechender Codierung der Elemente.

Ordnungsgemäße Berechnungen bzw. Durchläufe werden angezeigt, wenn sich ein **grüner Haken** im Display neben der Programmüberschrift zeigt **und keine** Fehlermeldung angezeigt wird.

Eingabewerte können editiert werden. Es zeigt sich ein Warnzeichen, wenn eine Taste betätigt wurde und eine neue Berechnung erfolgen muss. Blassgelb hinterlegte Felder zeigen Ergebnisse, die nicht editiert werden können.

Eine Punkte- und Vektorenübersicht der Restklaffen ermöglicht eine visuelle Begutachtung der Transformation. Eine Druckausgabe dieser Grafik ist nicht möglich. Eine dauerhafte Dokumentation dürfte mit einem <screenshot> möglich sein[1]. Allerdings sollten screenshots, und damit eine dauerhafte Dokumentation, nur dann gemacht werden, wenn sich ein grüner Haken am Display neben der Programmüberschrift zeigt und **keine Fehlermeldung** erscheint; nur um fehlerhafte Dokumentationen zu vermeiden. Dies gilt ebenso bei der Übernahme von Daten- bzw. Printdateien.

[1] Beispiele für Windows: <microsoft swift>

Helmerttransformation

Eine Helmerttransformation dient der Überführung von Koordinaten eines Startsystems (System A) in ein Zielsystem (System B) mit Hilfe bekannter Koordinaten in beiden Systemen (identische Punkte, Pass- oder Stützpunkte) durch die vorangehende Ermittlung von Transformationsparametern, wobei unvermeidliche Abweichungen minimiert werden sollen. Mit den Parametern können die Koordinaten der zu transformierenden Punkte für das Zielsystem berechnet werden. Eine 3-Parametertransformation besteht aus 2 Verschiebungen und einer Drehung, die 4-Parametertransformation aus 2 Verschiebungen, einer Drehung und einer Maßstabsanpassung. Weitere Parameter sind denkbar, werden hier aber nicht behandelt.

Durch mit Überbestimmungen erzeugte Transformationsparameter können die Zielkoordinaten der Passpunkte nicht übereinstimmend berechnet werden; es entstehen Differenzen, die als Restklaffen bezeichnet werden. Deshalb werden oder können die Koordinaten der zu transformierenden Punkte durch eine entsprechende Verteilung angepasst werden. Die Verteilung der Restklaffen auf umzuformende Punkte ist kein Bestandteil der Helmerttransformation mit Minimumsbedingungen, sondern mehr oder weniger willkürlich. Hier werden streckenabhängige Verteilungen angeboten, nämlich proportional mit den Gewichten $1/s$, $1/(s*\sqrt{s})$ bzw. $1/s^2$. Näheres entnehme man dem Kapitel: Formelentwicklung.

Mit dem Button besteht die Möglichkeit, ausgewählte Passpunkte in den Eingabefeldern bis zur minimalsten Anzahl der Passpunkte (= 2) deaktiv zu stellen, was andererseits zur Fehlersuche und Ausschaltung fehlerhafter Messungen durch die hilfreiche Restklaffenanzeige möglich ist. Inaktiv gestellte Punkte können auch wieder reaktiviert werden. Deaktive Passpunkte werden auf roten Hintergrund angezeigt. Die dabei angegebenen Restklaffen werden mit den ermittelten Parametern der aktiven Passpunkte berechnet, sodass die Abweichungen klarer hervortreten, weil die Elemente der deaktivierten Passpunkte nicht in die Parameterberechnung einflossen.

Eine Limitierung von jeweils maximal 10 Punkten bei manueller Eingabe dürfte aus praktischen Gründen vernünftig sein. Bei Überschreitung der Limits werden Defaultwerte eingesetzt. Ansonsten würde anschließend die Datei der Datenausgabe zu editieren und einzulesen. Eine Limitierung der Anzahl der Punkte gilt nicht bei der Eingabe von Dateien.

Programmbeschreibung

Ein Maßstabsfaktor kann Eigenschaften einer Korrektion wie auch einer Reduktion haben: Ein Korrektionsfaktor berichtigt die ursprüngliche Messung in Annäherung an den wahren Wert (z. B. Kalibrierungswerte), ein Reduktionsfaktor behandelt z. B. die Horizontierung einer Strecke oder eine Abbildungsverzerrung (Verebnung).

Dem hier vielfach in der Eingabe vorkommenden Reduktionsfaktor r (oder auch Maßstabsfaktor m) bei Strecken bzw. Koordinaten liegt folgende Definition zu Grunde:

$$\textit{reale örtliche, horizontierte Strecke}\,(\text{hor.}) \times r = \textit{Strecke in einer Abbildung}\,(\text{z.B. UTM})$$

Beispiel: Eine gemessene Strecke wird korrigiert durch Additionskonstante, instrumentell bedingtem Maßstab; eine Strecke wird reduziert durch Horizontierung, Höhenlage, Abbildung, Abbildungsmaßstab.)

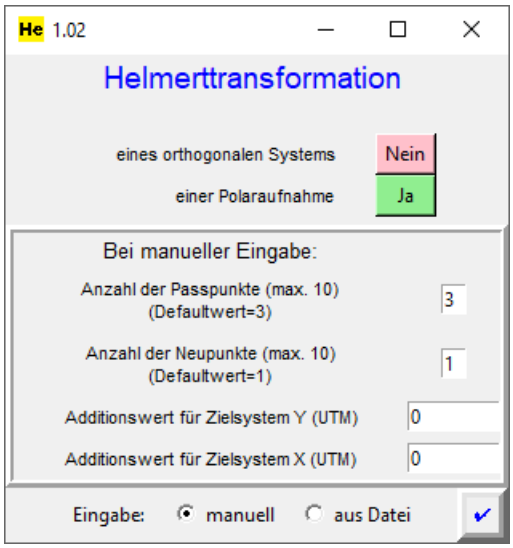


Abbildung 1
Entrée-Fenster, Beispiel der Wahl einer Polaraufnahme

Die Daten können manuell eingegeben oder per Datendatei (*.txt) eingelesen werden und später nach der Berechnung in einer Datenausgabe (*.txt) wieder aktuell gespeichert werden. Ob das umzuformende System eine polare Aufnahme oder orthogonale Koordinaten sind, muss entsprechend im Entrée-Fenster ausgewählt werden. Beim Einlesen einer Datei entscheidet das Programm aufgrund der Codierung in den *.txt-Dateien welche Daten einzulesen sind.

Bei einer manuellen Eingabe muss vorher angegeben werden, wie viele Passpunkte und umzuformende Punkte vorhanden sind, damit die Maske korrekt aufgerufen wird. Bei einer manuellen Eingabe ist die Maske leer, bei einer Dateneingabe entsprechend besetzt. Eine Eingabe durch eine Datei erfordert keine Angabe einer Punktanzahl. Diese wird dann automatisch ermittelt.

Die Syntax für die Dateieingaben in den *.txt Dateien sind in der nachfolgenden Aufstellung (vgl. auch die Beispieldaten auf Seite 16) genannt. Die Elemente werden durch den Separator ; getrennt. Kommas als Dezimalzeichen sind erlaubt, werden aber nach einem Berechnungsdurchlauf in der Dateiausgabe in einen Punkt umgewandelt, der in Python benutzt wird.

Codierungen der Dateieingabe:

01;							(Projektbezeichnung)
C bzw. 99; oder Leerzeile							(Kommentar, wird nicht bearbeitet)
10;	PKZ;	Ayi;	Axi;		BYi;	BXi;	(Koordinaten Passpunkte, System A und B)
20;	PKZ;	Ayj;	Axj;				(Koordinaten zu transformierende Punkte System A)
30;	PKZ;	ARichtung i;	AZenitdistanz i;	AStrecke i;	BYi;	BXi;	(Polarelemente System A, Koordinaten Sytem B)
40;	PKZ;	ARichtung j;	AZenitdistanz j;	AStrecke j;			(zu transformierende Polarelemente System B)

Codierungen der Dateiausgabe:

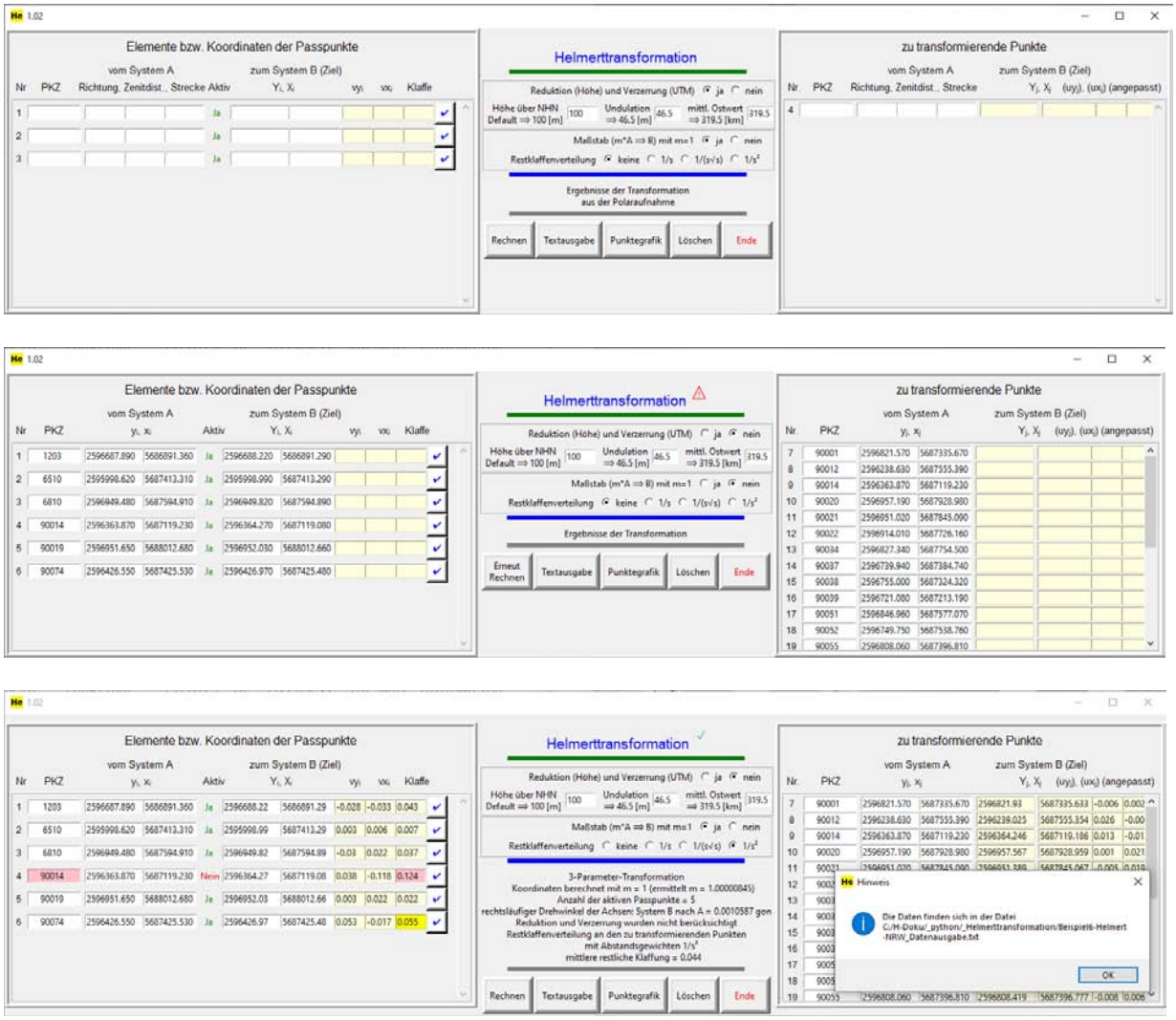
Bei einer erneuten Eingabe werden sie überlesen. Sie haben auf den neuen Berechnungsdurchlauf keinen Einfluss.

C bzw. 99; oder Leerzeile								**(Kommentar, wird nicht bearbeitet)**
02; Berechnungsdatum								
11;	**PKZ;**	**Ayi;**	**Axi;**	**BYi;**	**BXi;**	**vyi;**	**vxi;**	**Klaffe; (Passpunkte, System A, B)**
21;	**PKZ;**	**Ayj;**	**Axj;**	**BYj;**	**BXj;**	**uyj;**	**uxj;**	**(zu transformierende Punkte, System A, B [berechnet])**
31;	**PKZ;**	**Aylokal i;**	**Axlokal i;**	**BYi;**	**BXi;**	**vyi;**	**vxi;**	**Klaffe; (System A(lokale Kood., System B [berechnet])**
41;	**PKZ;**	**Aylokal j;**	**Axlokal j;**	**BYj;**	**BXj;**	**uyj;**	**uxj;**	**(System A(lokale Koord., System B [berechnet])**

Die Eingabemaske ist in 3 Teile gegliedert. Links stehen die Elemente bzw. Koordinaten der Passpunkte, rechts die der umzuformenden Punkte. In der Mitte befinden sich die Voreingaben zur Art der Transformation, die anschließenden Ergebnisse und weiter unten die Buttons zur Wahl der Rechengänge und Ausgaben.

Abbildung 2

Oben: *Leere Maske zur manuellen Eingabe* - **Mitte:** *Maske nach Einlesen einer Datei und Festlegung der Transformationsart zur weiteren manuellen Berichtigung/Ergänzung, Beispiel zu LVA NRW [5] S. 31-33* **Unten:** *Abgeschlossener Berechnungsdurchlauf*



Zur manuellen Eingabe:

> Dezimalzahlen müssen durch einen Punkt und nicht mit einem Komma eingegeben werden (pythonspezifisch)[2]. Die Eingabe von Buchstaben ist nicht erlaubt. Durch eine programmspezifische Ausnahmebehandlung würde eine Berechnung nur durchgeführt werden, wenn eine Umwandlung der Eingabe in Gleitkommazahlen (ohne Buchstaben, Komma oder sonstige Zeichen) möglich wäre. Ansonsten müssen Veränderungen in der Eingabe erfolgen. Bei der Dateneingabe über eine Datei können auch Kommas statt Punkte eingegeben werden; sie werden beim Einlesevorgang automatisch in Punkte umgewandelt. Leerfelder müssen besetzt werden (deshalb ist auch eine präzise Angabe der Passpunkte bzw. umzuformenden Punkte erforderlich).

Reduktions- und Korrektionswerte sind durch Defaultwerte in der Eingabe vorgegeben, die bei Notwendigkeit verändert werden können. Bei leeren Feldern für Zenitdistanzen bei Streckenreduktionen werden automatisch Winkel von 100 gon erzeugt.

Punktkennzeichen (PKZ) mit maximal 10 Zeichen sind alphanumerisch. Punkte können deaktiviert und wieder reaktiviert werden. Bei einem deaktivierten Punkt wird das Feld des PKZ rot hinterlegt. Bei einer Datenausgabe werden die Koordinaten dieses Punktes nicht gelöscht, sondern mit einer Codierung 99 versehen.

Durch den Button <Punktegrafik> kann zur besseren Übersicht und zum Verständnis der Eingabefelder eine wieder schließbare Zeichnung im Display geöffnet werden. Die Grafik sollte durch den unteren, roten Button geschlossen werden.

Ein gerade durchgeführter Berechnungsablauf wird neben der Titelüberschrift durch einen grünen Haken ✓ angezeigt. Ein Warnzeichen ⚠ taucht bei einer nachträglichen Betätigung einer Taste auf. Dann ist ein erneuter Berechnungsablauf **stets erforderlich**, da sonst im Zusammenhang fehlerhafte, nicht zusammen gehörige Angaben im Display erscheinen können.

Blassgelbe Eingabefelder dienen **ausschließlich einer Anzeige**. Eingaben darin haben keinerlei Einfluss auf die Auswertung. Rot hinterlegte Felder deuten auf Fehler oder Abweichungen hin, die aber keinen Einfluss auf die Berechnungsergebnisse haben. Die größte Klaffe bei den aktiven Passpunkten wird in gelb angezeigt. Die Ausgabe erfolgt in [0,1 mgon] bzw. [mm], wobei rechtsseitige Nullen nicht angezeigt werden.

[2] Eine Einstellung des Punktes im Ziffernblock der Tastatur für Windows ist sinnvoll. Man wähle dazu in Windows: >Einstellungen > Zeit und Sprache > Sprache > Sprache hinzufügen > Deutsch-Optionen > Deutsch(Punkt). Hinweise im Internet, einen Punkt im Ziffernblock der Tastatur mit Tastaturtreiber anderer Länder (z.B. Schweiz) erzeugen zu wollen, wechseln nach Tastendruck zwar das Komma in einen Punkt, haben aber die Eigenschaft andere Sonderzeichen der Tastatur zu verändern. Deshalb kann es sinnvoll sein, softwaremäßig ausschließlich nur das Komma im Ziffernblock zu ändern: Die herunter ladbare Software PatchKeyboard_6.1 vermag das und erzeugt einen Tastaturtreiber <Deutsch(Punkt)>, der zu wählen würde.

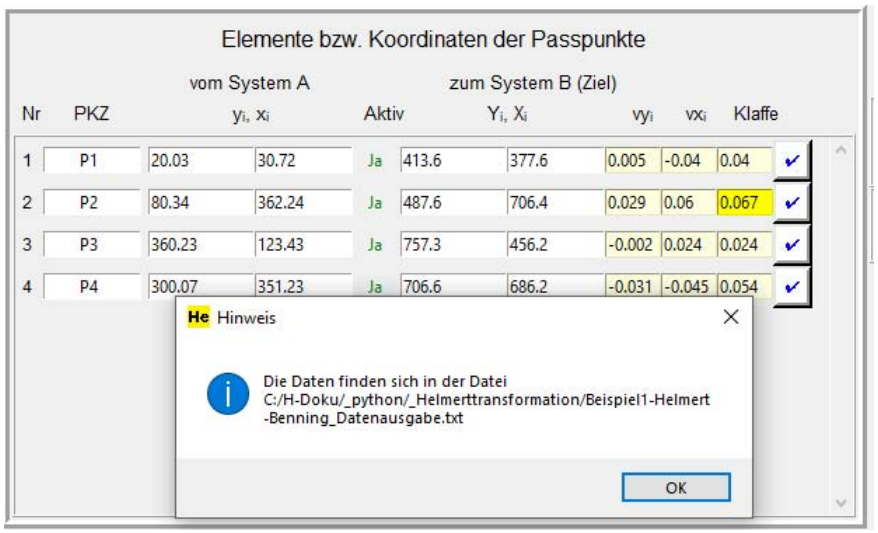


Abbildung 3

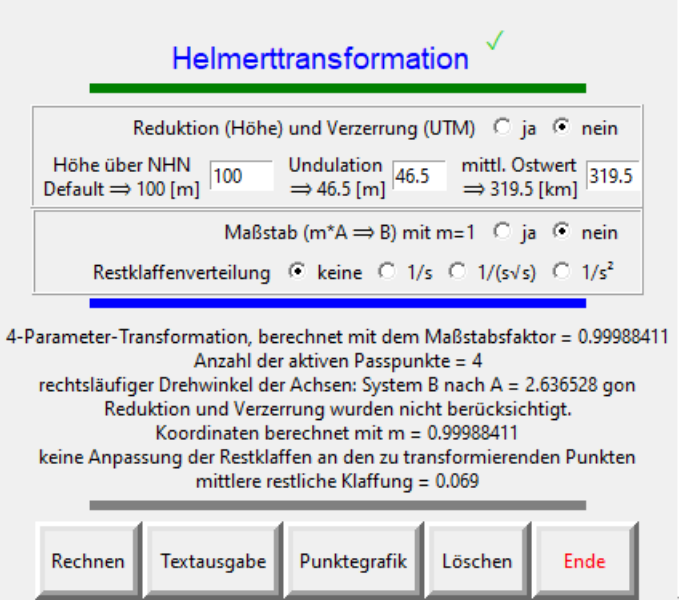


Abbildung 4

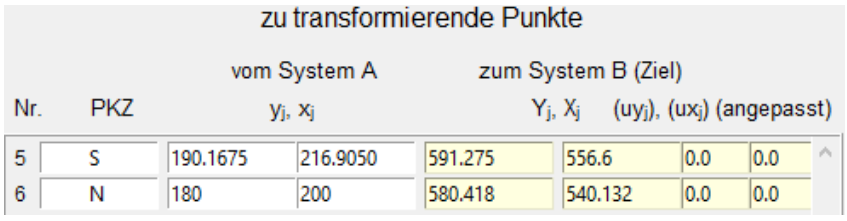


Abbildung 5
Beispiele aus Benning *[1] S. 264-266*

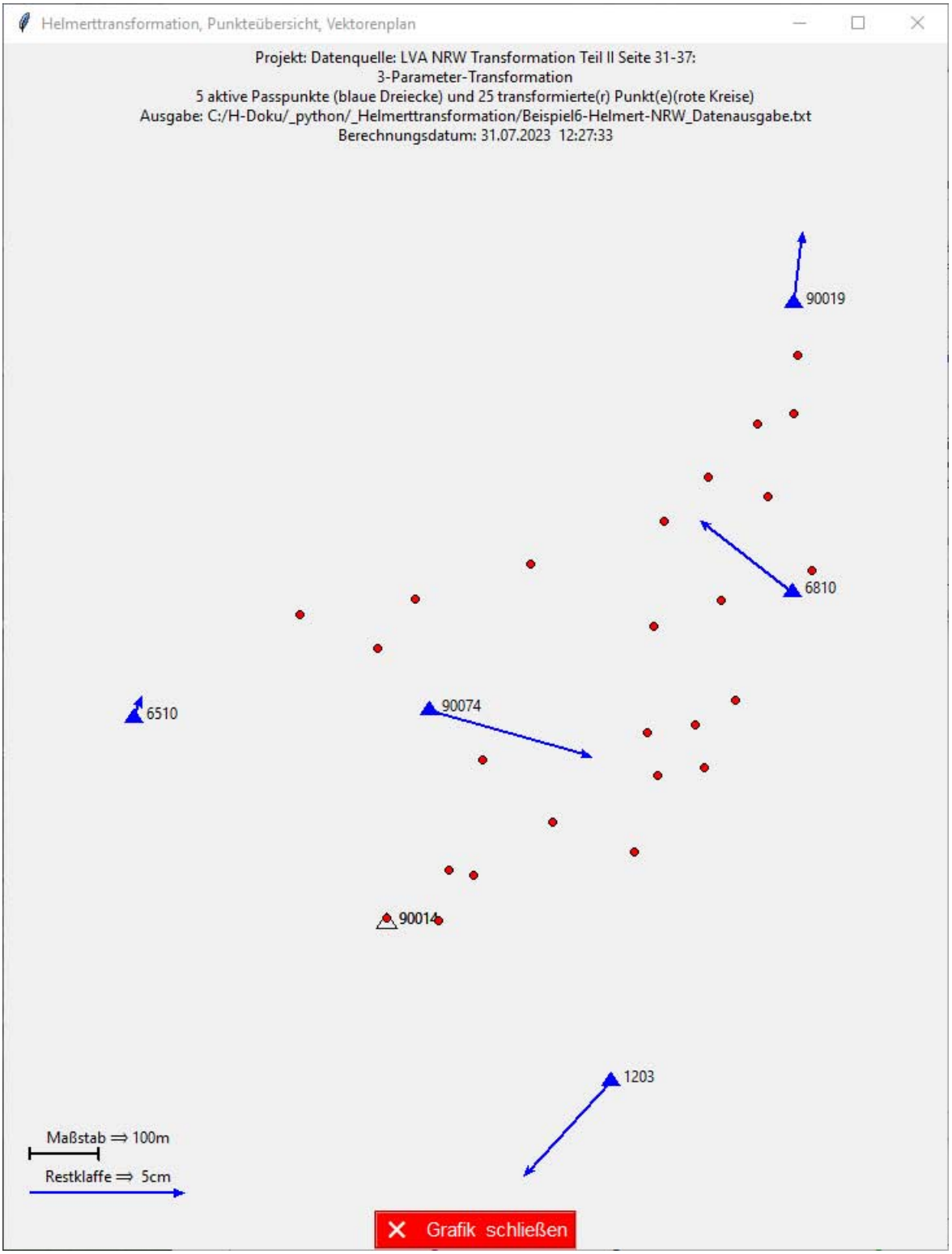


Abbildung 6
zu Beispiel aus LVA NRW [5] S. 34-37

Elemente bzw. Koordinaten der Passpunkte

Nr	PKZ	vom System A y_i, x_i		Aktiv	zum System B (Ziel) Y_i, X_i		vy_i	vx_i	Klaffe	
1	1203	2596687.890	5686891.360	Nein	2596688.22	5686891.29	-0.027	-0.056	0.062	✔
2	6510	2595998.620	5687413.310	Ja	2595998.99	5687413.29	0.004	-0.007	0.008	✔
3	6810	2596949.480	5687594.910	Ja	2596949.82	5687594.89	-0.02	0.002	0.02	✔
4	90014	2596363.870	5687119.230	Nein	2596364.27	5687119.08	0.039	-0.136	0.142	✔
5	90019	2596951.650	5688012.680	Ja	2596952.03	5688012.66	0.017	0.005	0.017	✔
6	90074	2596426.550	5687425.530	Nein	2596426.97	5687425.48	0.057	-0.033	0.066	✔

Abbildung 7
4-Parameter-Transformation mit von 6 auf 3 aktive Passpunkte reduzierte Transformation, zu Beispiel aus LVA NRW [5] S. 34-37

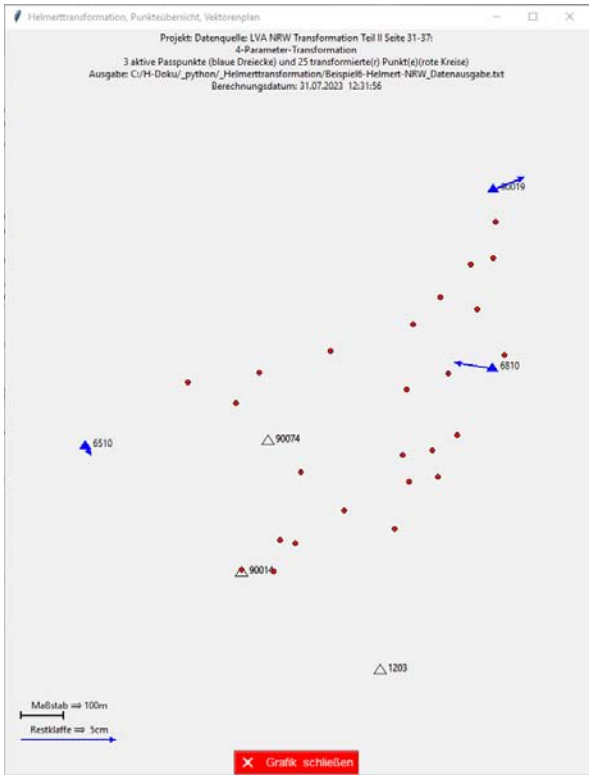


Abbildung 8
Grafik zum obigen Beispiel

Als extrapolierte Punkte (siehe [4]) in einer Helmerttransformation werden die zu transformierenden Punkte bezeichnet, die außerhalb eines die Stützpunkte umhüllenden Polygons liegen. Deren Kennzeichnung ist wenig aussagekräftig, weil die Form des Stützpunktgebietes einen großen Einfluss auf das Ergebnis hat, außerdem werden Clusterbildungen nicht hinreichend erkannt.

Eine wichtige Kontrolle ist die Ansicht der Grafik der Punkte. Sind die Passpunkte gleichmäßig verteilt? Wurde eine Clusterbildung vermieden, insbesondere bei umzuformenden Punkten für die Restklaffenverteilung? Ist das Umformungsgebiet möglichst nicht langgestreckt?

Ausreißer bei den zu transformierenden Punkten werden einer Abstandskontrolle zum Schwerpunkt unterworfen; dies im Vergleich zum Streckenmedian der Passpunkte zu dessen Schwerpunkt.

Zur Fehlersuche wurde auf die Implementierung von Data-Snooping (1968) oder Cook-Abstand (1977) verzichtet, weil die Angaben der Restklaffen und die Möglichkeit der unmittelbaren Deaktivierung von Stützpunkten für eine Fehlersuche ausreichend erscheinen. Auch deshalb, weil hier keine hybriden Messungen vorliegen und keine a-priori-Genauigkeiten verwendet werden. Eine wichtige Kontrolle ist die grafische bzw. durch Augenschein, um auf eine gewogene, gut gestreute Punktverteilung zu achten. Die maximale Restklaffe wird in der Tabelle durch einen verstärkten Gelbton kenntlich gemacht. Eine Deaktivierung dieses Punktes mit einem anschließenden, erneutem Durchlauf mit den neu ermittelten Transformationsparametern der anderen Punkte zeigt die quasiwahre Restklaffe (in rotem Hintergrund) dieses auffälligen Punktes. Weitere Punkte können deaktiviert werden, aber auch die Zurücknahme einer Deaktivierung ist möglich.

Die Redundanzen zeigen bei einer Überbestimmung inwieweit die Eingangswerte kontrolliert sind. Geringe Redundanzwerte sprechen für eine geringe Kontrolle, höhere für eine bessere. Alle Redundanzen im gleichen Bereich sprechen für ein ausgewogenes Ergebnis. Die Redundanzen bei der Polaraufnahme beziehen sich auf die Koordinaten der in das örtliche, orthogonale System umgeformten Punkte. Alle haben demnach das Gewicht 1.

Die Aussage von Y_0 und X_0 ist gering, je nachdem ob man durch Konstanten gekürzte Koordinaten zur Transformation benützt oder nicht. Ein direkter Einfluss auf das Koordinatenergebnis besteht nicht.

Formelentwicklung

Streckenreduktion auf UTM-Bezugsebene:

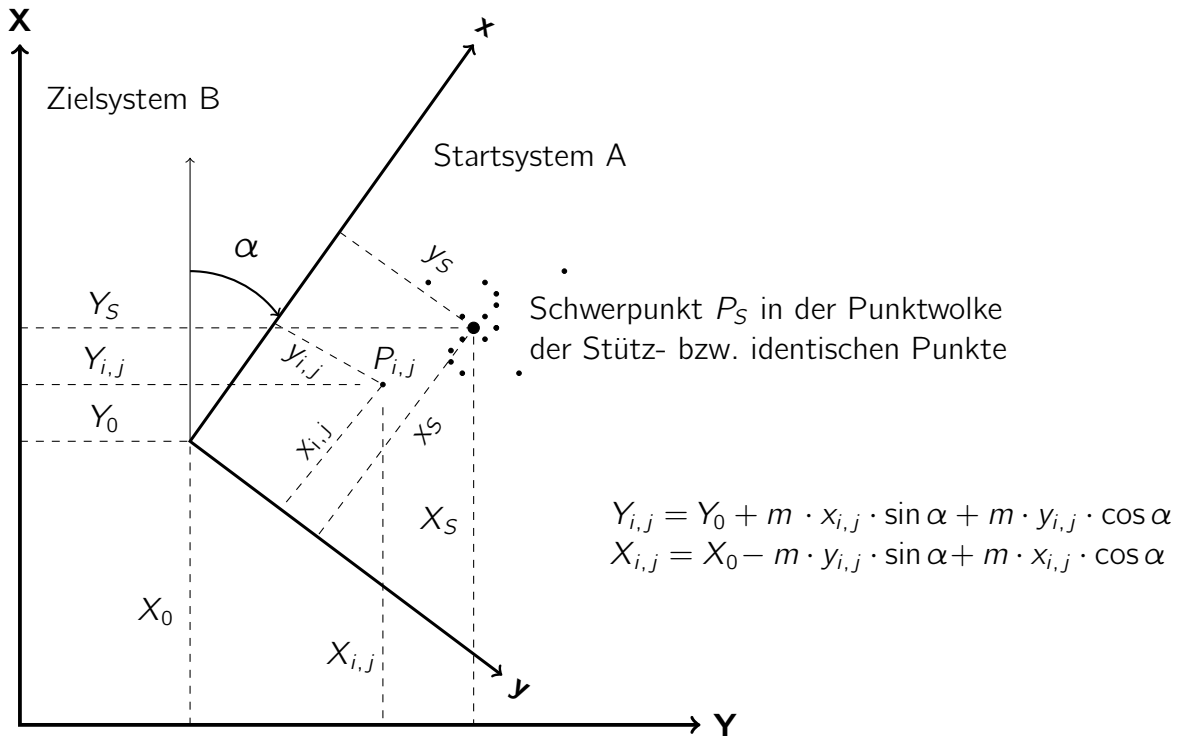
$$S_{[\text{örtlich}]} = \frac{\sqrt{\left((Ost_2 - Ost_1)^2 + (Nord_2 - Nord_1)^2\right)} = S_{[\text{UTM}]}}{\left(1 + \frac{\left(Ost_{\text{m[km]}} - 500\right)^2}{2R^2_{[\text{km}]}} - \frac{h_{\text{NHN[m]}} + h_{\text{Undulation[m]}}}{R_{[\text{km}]} * 1000}\right) * 0.9996 = r}$$

$Ost_i, Nord_i$ = UTM-Koordinaten
$S_{[\text{örtlich}]}, S_{[\text{UTM}]}$ = Strecke im örtlichen System, in UTM
r = Reduktionsfaktor
$R = 6381\ [km]$, mittlerer Erdradius für die mittlere Breite in Deutschland
$S_{[\text{örtlich}]} \cdot r = S_{[\text{UTM}]}$

Bei vorgegebener polaren Aufnahme im System A erfolgt eine Umrechnung polarer in orthogonale Koordinaten mit:

$y_i = s_i \cdot sin(Zenitdistanz_i) \cdot sin(Richtung_i)$
$x_i = s_i \cdot sin(Zenitdistanz_i) \cdot cos(Richtung_i)$

Das System A ist das **A**usgangs- oder Startsystem, dass u. U. die ursprünglichen, umzuformenden Elemente oder Messergebnisse enthält. Das System B ist das Zielsystem, i.d.R. ein übergeordnetes System, in das hinein transformiert werden soll.



Notation[3]:

y, x	Startsystem A
Y, X	Zielsystem B
i	Punktindex Passpunkte
j	Punktindex umzuformende Punkte
$\bar{y}$	Mittelwert
$\hat{y}$	Schätzwert, berechneter Wert
y'	reduzierter Wert
y_S	Schwerpunkt, auf den Schwerpunkt bezogen
y_{iS}	lokaler, vom Punkt i auf den Schwerpunkt reduzierter Wert
m	Maßstab

[3]Die Ziffer **1** bzw. der Kleinbuchstabe **l** bzw. der Großbuchstabe **I** lassen sich vor allem im Druck nur schwer unterscheiden. Wird das nicht beachtet – besonders in Lehrbüchern – führt das beim Leser leicht zu Missverständnissen.
Falls erforderlich wird deshalb hier ein Einsvektor mit $\mathbf{e}$, eine Einheitsmatrix mit $\mathbf{E}$, eine Beobachtung mit ℓ, ein Beobachtungsvektor mit $\boldsymbol{\ell}$, eine Residue mit ε, ein Residuenvektor mit $\boldsymbol{\varepsilon}$ bezeichnet.
Auch die Vertauschung von X und Y in den Lehrbüchern erschweren das Verständnis. Chaotischer kann Lehre nicht sein!

Benning [1] Syntax aus der Statistik $\boldsymbol{y} = \mathbf{X} \cdot \boldsymbol{\beta} + \boldsymbol{\varepsilon}$	**Niemeier** [2] Syntax aus der klassischen Ausgleichungsrechnung $\boldsymbol{\ell} + \boldsymbol{v} = \mathbf{A} \cdot \boldsymbol{x}$	**Helmerttransformation**

Passpunkte im Startsystem A (**A**usgang) = $P_i(y_i, x_i)$ und im Zielsystem B = $P_i(Y_i, X_i)$ und zu transformierende Punkte $P_j(y_j, x_j)$ in das Zielsystem B = $P_j(Y_j, X_j)$

$Y_i = Y_0 + o \cdot x_i + a \cdot y_i$ (Ansatz Benning, 4-Parameter)
$X_i = X_0 - o \cdot y_i + a \cdot x_i$

Translations und Transformationsparameter Y_0, X_0, o, a

$$\boxed{\boldsymbol{y} = \mathbf{X} \cdot \boldsymbol{\beta} + \boldsymbol{\varepsilon}}$$

$$\boxed{\boldsymbol{\ell} + \boldsymbol{v} = \mathbf{A} \cdot \boldsymbol{x}}$$

$v_i = -\varepsilon_i$ bzw. $\mathbf{v} = -\boldsymbol{\varepsilon}$

$$\underset{(2n,1)}{\boldsymbol{y}} = \begin{bmatrix} Y_1 \\ X_1 \\ \vdots \\ Y_n \\ X_n \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} Y_0 \\ X_0 \\ o \\ a \end{bmatrix}$$

$$\underset{(2n,4)}{\mathbf{A}} = \mathbf{X} = \begin{bmatrix} 1 & 0 & x_1 & y_1 \\ 0 & 1 & -y_1 & x_1 \\ 1 & 0 & x_i & y_i \\ 0 & 1 & -y_i & x_i \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & x_n & y_n \\ 0 & 1 & -y_n & x_n \end{bmatrix}$$

i = Index Stützpunkt
j = Index umzuformender Punkt

$$\mathbf{A} = \mathbf{X}, \quad \boldsymbol{\beta} = \boldsymbol{x}, \quad \boldsymbol{y} = \boldsymbol{\ell}$$

$$\boldsymbol{N} = \mathbf{A}^T\mathbf{A}, \quad \boldsymbol{n} = \mathbf{A}^T\boldsymbol{\ell}$$

Reduktion der Koordinaten auf die Schwerpunkte (Mittelwerte) der Passpunkte

$$Y_S = \frac{1}{n}\sum_{i=1}^{n} Y_i, \quad X_S = \frac{1}{n}\sum_{i=1}^{n} X_i$$

$$y_S = \frac{1}{n}\sum_{i=1}^{n} y_i, \quad x_S = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$Y_S = Y_0 + a \cdot y_S + o \cdot x_S, \quad X_S = X_0 - o \cdot y_S + a \cdot x_S$$

$$Y_{iS} = Y_i - Y_S, \quad X_{iS} = X_i - Y_S$$

$$y_{iS} = y_i - y_S, \quad x_{iS} = x_i - x_S$$

$$Y_i = Y_S + a \cdot y_{iS} + o \cdot x_{iS}$$

$$X_i = X_S - o \cdot y_{iS} + a \cdot x_{iS}$$

Schwerpunkte y_S, x_S, Y_S, X_S
n= Anzahl der aktiven Passpunkte

Reduktionen auf den Schwerpunkt bezogen

$$\boxed{\boldsymbol{y} = \mathbf{X}_S \cdot \hat{\boldsymbol{\beta}} + \boldsymbol{\varepsilon}}$$

$$\boxed{\boldsymbol{\ell} + \boldsymbol{v} = \mathbf{A}_S \cdot \hat{\boldsymbol{x}}}$$

$v_i = -\varepsilon_i$ bzw. $\mathbf{v} = -\boldsymbol{\varepsilon}$

$$\hat{\boldsymbol{\beta}} = \begin{bmatrix} Y_S \\ X_S \\ o \\ a \end{bmatrix}$$

$$\underset{(2n,4)}{\mathbf{A}_S} = \begin{bmatrix} 1 & 0 & x_{1S} & y_{1S} \\ 0 & 1 & -y_{1S} & x_{1S} \\ 1 & 0 & x_{iS} & y_{iS} \\ 0 & 1 & -y_{iS} & x_{iS} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & x_{nS} & y_{nS} \\ 0 & 1 & -y_{nS} & x_{nS} \end{bmatrix}$$

$$\mathbf{A}_S = \mathbf{X}_S, \quad \hat{\boldsymbol{\beta}} = \hat{\boldsymbol{x}}, \quad \boldsymbol{y} = \boldsymbol{\ell}$$

$$\boldsymbol{N}_S = \mathbf{A}_S^T\mathbf{A}_S, \quad \boldsymbol{n}_S = \mathbf{A}_S^T\boldsymbol{\ell}$$

$$\mathbf{N}_S = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & \sum & 0 \\ 0 & 0 & 0 & \sum \end{bmatrix}$$

wobei $\sum \hat{=} \sum(x_{iS}^2 + y_{iS}^2)$

$\hat{\boldsymbol{x}} = (\mathbf{A}_S^T \mathbf{A}_S)^{-1} \mathbf{A}_S^T \boldsymbol{\ell} + \boldsymbol{v}$ $\hat{\boldsymbol{x}} = \mathbf{N}_S^{-1} \mathbf{n}_S$	Transformationsparameter
$\boldsymbol{v} = \mathbf{A}_S \hat{\boldsymbol{x}} - \boldsymbol{\ell}$	Restklaffen

Berechnung der Transformationsparameter

$$m = \text{Maßstab} = \frac{Strecke\ (Zielsystem)}{Strecke\ (Startsystem)}$$

$$\hat{o} = m \cdot sin\ \alpha, \quad \hat{a} = m \cdot cos\ \alpha, \quad m = \sqrt{\hat{o}^2 + \hat{a}^2}, \quad \alpha = \arctan \frac{\hat{o}}{\hat{a}}$$

α = Drehwinkel vom System B zum System A

α = Richtungswinkel der x-Achse (System A) gegen die X-Achse (System B)

$$\hat{o} = \frac{\sum_{i=1}^{n}(Y_{iS} * x_{iS}) - \sum_{i=1}^{n}(X_{iS} * y_{iS})}{\sum_{i=1}^{n}(y_{iS}^2 + x_{iS}^2)} \qquad \hat{a} = \frac{\sum_{i=1}^{n}(Y_{iS} * y_{iS}) + \sum_{i=1}^{n}(X_{iS} * x_{iS})}{\sum_{i=1}^{n}(y_{iS}^2 + x_{iS}^2)}$$

$o' = \frac{\hat{o}}{m} = sin\ \alpha, \quad a' = \frac{\hat{a}}{m} = cos\ \alpha$	
$\hat{Y}_0 = Y_S - o \cdot x_S - a \cdot y_S$ $\hat{X}_0 = X_S + o \cdot y_S - a \cdot x_S$	Translationsunbekannte Y_0, X_0

Die Translationsunbekannten werden ausgleichsunabhängig mitunter bei nahezu achsparallelen Systemen berechnet durch : $Y_0 = Y_S - y_S, \quad X_0 = X_S - x_S$ (so auch in [5])

Transformation, Restklaffen

$\hat{Y}_i$ (gerechnet) $= \hat{Y}_S + \hat{a}y_i + \hat{o}x_i$	4-Parameter-Transformation, konform
$\hat{X}_i$ (gerechnet) $= \hat{X}_S + \hat{a}x_i - \hat{o}y_i$	Koordinaten nach Transformation
$\hat{Y}_i$ (gerechnet) $= \hat{Y}_S + a'y_i + o'x_i$	3-Parameter-Transformation, isometrisch
$\hat{X}_i$ (gerechnet) $= \hat{X}_S + a'x_i - o'y_i$	Koordinaten nach Transformation
$\hat{v}_{yi} = Y_i - \hat{Y}_i$ (gerechnet) $\hat{v}_{xi} = X_i - \hat{X}_i$ (gerechnet) Klaffe $= \sqrt{\hat{v}_{yi}^2 + \hat{v}_{xi}^2}$	Restklaffen

Berechnung der Neupunktkoordinaten ins Zielsystem

$\hat{Y}_j$ (gerechnet) $= \hat{Y}_S + \hat{a}y_j + \hat{o}x_j$ $\hat{X}_j$ (gerechnet) $= \hat{X}_S + \hat{a}x_j - \hat{o}y_j$	4-Parameter-Transformation
$\hat{Y}_j$ (gerechnet) $= \hat{Y}_S + a'y_j + o'x_j$ $\hat{X}_j$ (gerechnet) $= \hat{X}_S + a'x_j - o'y_j$	3-Parameter-Transformation

Fehlerberechnung, Redundanzen

$s_0^2 = \frac{\boldsymbol{v}^T\boldsymbol{v}}{2n-3}, \quad s_0^2 = \frac{\boldsymbol{v}^T\boldsymbol{v}}{2n-4}$	3- und 4-Parameter, empirische Varianz der Gewichtseinheit
$\hat{s}_0 = \sqrt{\frac{\sum_{i=1}^n \hat{v}_{y\,i}{}^2 + \sum_{i=1}^n \hat{v}_{x\,i}{}^2}{2n-4}}$	4-Parameter, empirische Standardabw. der Gewichtseinheit
$\hat{s}_0 = \sqrt{\frac{\sum_{i=1}^n \hat{v}_{y\,i}{}^2 + \sum_{i=1}^n \hat{v}_{x\,i}{}^2}{2n-3}}$	3-Parameter, empirische Standardabw. der Gewichtseinheit
$\hat{m}_{P_i} = \hat{s}_0\sqrt{2}$	m_{P_i} = mittlere restliche Klaffung eines Punktes
$r_{y\,i},\ r_{x\,i\,[\%]} = 100 \cdot \left(1 - \frac{1}{n} - \frac{x_{i\,S}^2 + y_{i\,S}^2}{\sum_{i=1}^n (x_{i\,S}^2 + y_{i\,S}^2)}\right)$	Redundanzanteil in %, 4-Parameter
$r_{y\,i\,[\%]} = 100 \cdot \left(1 - \frac{1}{n} - \frac{(x_{i\,S} \cdot \cos\alpha - y_{i\,S} \cdot \sin\alpha)^2}{\sum_{i=1}^n (x_{i\,S}^2 + y_{i\,S}^2)}\right)$	Redundanzanteil in %, 3-Parameter
$r_{x\,i\,[\%]} = 100 \cdot \left(1 - \frac{1}{n} - \frac{(y_{i\,S} \cdot \cos\alpha + x_{i\,S} \cdot \sin\alpha)^2}{\sum_{i=1}^n (x_{i\,S}^2 + y_{i\,S}^2)}\right)$	Redundanzanteil in %, 3-Parameter

Gewichtete Restklaffenverteilung

$$p_{ij} = \frac{1}{s_{ij}} \text{ bzw. } p_{ij} = \frac{1}{s_{ij} * \sqrt{s_{ij}}} \text{ bzw. } p_{ij} = \frac{1}{s_{ij}^2}$$

$$u_{yj} = \frac{1}{\sum p_{ij}} \sum_{i=1}^{n} (p_{ij} \cdot \hat{v}_{y\,i})$$

$$u_{xj} = \frac{1}{\sum p_{ij}} \sum_{i=1}^{n} (p_{ij} \cdot \hat{v}_{x\,i})$$

$u_{yj}, u_{xj}, = 0$ (keine Restklaffenanpassung)

$$\hat{Y}_j(\text{endgültig}) = \hat{Y}_j(\text{gerechnet}) + u_{yj}$$
$$\hat{X}_j(\text{endgültig}) = \hat{X}_j(\text{gerechnet}) + u_{xj}$$

p_{ij} = Abstandsgewicht
s_{ij} = Strecke zwischen i und j
v = Restklaffe
u = Anpassungsbetrag
(Dieser Betrag ist in den endgültigen Koordinaten bereits enthalten)

Anhang – 3-Parameter Helmerttransformation (Redundanzen)

$$Y_i = Y_0 \overbrace{+x_i \cdot \sin\alpha + y_i \cdot \cos\alpha}^{a_i} = Y_0 + a_i \qquad \text{(Transformationsgleichungen)}$$

Ansatz analog zu Benning [1] S. 260f

$$X_i = X_0 \overbrace{-y_i \cdot \sin\alpha + x_i \cdot \cos\alpha}^{b_i} = X_0 + b_i \qquad i = 1, \ldots, n$$

$$\frac{\delta Y_i}{\delta Y_0} = 1; \quad \frac{\delta Y_i}{\delta \alpha} = \frac{\delta a_i}{\delta \alpha} = x_i \cdot \cos\alpha - y_i \cdot \sin\alpha = c_i$$

$$\frac{\delta X_i}{\delta X_0} = 1; \quad \frac{\delta X_i}{\delta \alpha} = \frac{\delta b_i}{\delta \alpha} = -y_i \cdot \cos\alpha - x_i \cdot \sin\alpha = d_i$$

$$\underset{(2n,3)}{\mathbf{A}} = \mathbf{X} = \begin{bmatrix} 1 & 0 & x_1 \cdot \cos\alpha - y_1 \cdot \sin\alpha \\ 0 & 1 & -y_1 \cdot \cos\alpha - x_1 \cdot \sin\alpha \\ 1 & 0 & c_i \\ 0 & 1 & d_i \\ \vdots & \vdots & \vdots \\ 1 & 0 & c_n \\ 0 & 1 & d_n \end{bmatrix} \quad \text{(Funktionalmatrix)}$$

$$\mathbf{A}^T = \begin{bmatrix} 1 & 0 & \ldots \\ 0 & 1 & \ldots \\ c_1 & d_1 & \ldots \end{bmatrix} \begin{bmatrix} n & & \\ & n & \\ & & \sum_{i=1}^n ({c_i}^2 + {d_i}^2) \end{bmatrix} = \mathbf{A}^T\,\mathbf{A} = \mathbf{N}$$

(Normalgleichungsmatrix)
(Falkschema)

$$h = \sum_{i=1}^n ({c_i}^2 + {d_i}^2) = \sum_{i=1}^n ({x_i}^2 + {y_i}^2)$$

$$\mathbf{N}^{-1} = \begin{bmatrix} 1/n & & \\ & 1/n & \\ & & 1/h \end{bmatrix} = diag\,\{1/n, 1/n, 1/h\}$$

$$\mathbf{H} = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T = \mathbf{A}\mathbf{N}^{-1}\mathbf{A}^T \qquad \text{(Projektionsmatrix)}$$

$$\mathbf{N}^{-1} = \begin{bmatrix} 1/n & & \\ & 1/n & \\ & & 1/h \end{bmatrix} \begin{bmatrix} 1 & 0 & \ldots \\ 0 & 1 & \ldots \\ c_1 & d_1 & \ldots \end{bmatrix} = \mathbf{A}^T$$

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & c_1 \\ 0 & 1 & d_1 \\ \vdots & \vdots & \vdots \\ 1 & 0 & c_n \\ 0 & 1 & d_n \end{bmatrix} \begin{bmatrix} 1/n & 0 & c_1/h \\ 0 & 1/n & d_1/h \\ \vdots & \vdots & \vdots \\ 1/n & 0 & c_n/h \\ 0 & 1/n & d_n/h \end{bmatrix} \begin{bmatrix} \frac{1}{n} + \frac{c_1^2}{h} & \ldots & \ldots \\ \ldots & \frac{1}{n} + \frac{d_1^2}{h} & \ldots \\ \vdots & \vdots & \vdots \\ \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots \end{bmatrix} = \mathbf{H} \quad \text{(Falkschema)}$$

$$r_{ii} = 1 - h_{ii}, \quad r_{yi} = 1 - (\tfrac{1}{n} + \tfrac{c_i^2}{h}), \quad r_{xi} = 1 - (\tfrac{1}{n} + \tfrac{d_i^2}{h})$$

$$r_{y\,i\,[\%]} = 100 \cdot \left(1 - \frac{1}{n} - \frac{(x_i \cdot \cos\alpha - y_i \cdot \sin\alpha)^2}{\sum_{i=1}^n ({x_i}^2 + {y_i}^2)}\right) \qquad \text{(Redundanzanteile in \%)}$$

$$r_{x\,i\,[\%]} = 100 \cdot \left(1 - \frac{1}{n} - \frac{(y_i \cdot \cos\alpha + x_i \cdot \sin\alpha)^2}{\sum_{i=1}^n ({x_i}^2 + {y_i}^2)}\right)$$

Ausgabe, Beispieldaten und -ausdrucke

Die Daten- bzw. die Druckausgaben finden sich im gleichen Verzeichnis, in dem auch die Eingabedaten (xxx.txt) vorhanden sind. Sie haben jeweils die Ergänzung xxx_Datenausgabe.txt bzw. xxx_Printausgabe.txt. Bei manueller Eingabe ist die Ausgabe in C:/tmp/_Datenausgabe.txt bzw. C:/tmp/_Printausgabe.txt.

Beispiel einer erzeugten Datenausgabe, die wiederum auch editiert zur Dateneingabe verwendet werden kann:

```
C ***** Eingabedaten (manuell oder ursprünglich aus einer Datei) *****
01;Projekt: Beispieldaten zur Helmerttransformation. 4
02;Berechnungsdatum: 31.07.2023   12:40:29
99 Lagesystem: ETRS89 (489); Bezugsellipsoid: GRS80; Abbildung: UTM
99       Undulation = 46.5 m, mittl. Höhe ü.NHN =  49.5 m, mittl. UTM-Ostwert[km] = 360
99       Reduktion und Verzerrung (r = 0.99982555) wurden berücksichtigt
C Code 30: Polare bzw. orthogonale Passpunkte
C Code;   PKZ; ARichtung i; AZenitdistanz i; AStrecke i;  BYi;   BXi;
30;    104569;     0.0;100.0777;  91.857;2360502.036;5649727.404;
30;    104778; 11.4171;100.4669;  31.551; 2360466.71;  5649677.6;
30;    104207;103.009 ;102.3127;   6.677; 2360447.08;5649653.125;
30;    104779;192.3501;101.0408;  76.459;2360399.768;5649594.142;
99
C Code 40: Polare zu transformierende Punkte
C Code;   PKZ; ARichtung j; AZenitdistanz j; AStrecke j;
40;      2000;300.5159;100.5466;  64.418;
40;    104569;399.9989;100.0784;  91.857;
99
99 ***** Ergebnisse der Transformation *****
99
99 4-Parameter-Transformation
99       Koordinaten berechnet mit m = 1.00003529
99 Anzahl der aktiven Passpunkte = 4
99 rechtsläufiger Drehwinkel der Achsen: System B nach A = 45.144445 gon
99 Standpunktkoordinaten im System B: Y = 2360442.228, X = 5649657.703
99       Reduktion und Verzerrung (r = 0.99982555) wurden berücksichtigt
99 keine Anpassung der Restklaffen an den zu transformierenden Punkten
99 Standardabweichung der Gewichtseinheit = 0.003
99       mittlere restliche Klaffung = 0.004
99
C Code 31: Polare bzw. orthogonale Passpunkte, nach der Transformation
C Code;   PKZ; Aylokal i; Axlokal i;         BYi;         BXi;      vyi;     vxi;  Klaffe;
31;    104569;   -5.363;   80.171; 2360502.036; 5649727.404;   0.001;  -0.003;   0.003;
31;    104778;    0.264;   19.369;  2360466.71;   5649677.6;  -0.001;   0.004;   0.004;
31;    104207;    1.301;  -11.985;  2360447.08; 5649653.125;  -0.001;   0.000;   0.001;
31;    104779;    3.799;  -87.554; 2360399.768; 5649594.142;   0.001;  -0.002;   0.002;
99
C Code 41: Polare zu transformierende Punkte, nach der Transformation
C Code;   PKZ; Aylokal j; Axlokal j;         BYj;         BXj;      uyj; uxj(angepasst);
41;      2000;  -69.766;  -11.148; 2360393.690; 5649700.038;   0.000;   0.000;
41;    104569;   -5.365;   80.171; 2360502.034; 5649727.408;   0.000;   0.000;
99
```

Printbeispiel mit Daten nach [5] Seiten 31, 32:

```
                        *** Helmerttransformation ***                    Seite 1
Projekt: Datenquelle: LVA NRW Transformation Teil II Seite 31-37:
Berechnungsdatum: 31.07.2023

-------------------------------------------------------------------------------

3-Parameter-Transformation
      Koordinaten berechnet mit m = 1 (ermittelt m = 1.00000845)
Restklaffenverteilung an den zu transformierenden Punkten
      mit Abstandsgewichten 1/s²
Standardabweichung der Gewichtseinheit = 0.031
      mittlere restliche Klaffung = 0.044

Auf die Stützpunkte bezogen:
      mittlere Streckenlänge zum Schwerpunkt des lokalen Systems A = 477.357
      maximale Streckenlänge zum Schwerpunkt des lokalen Systems A = 647.169
      Median der Streckenlängen zum Schwerpunkt des lokalen Systems A = 475.868
Auf die zu transformierenden Punkte bezogen:
      maximale Streckenlänge zum Schwerpunkt des lokalen Systems A = 581.787
      Der nach <ex> angezeigte Wert ist das Verh"altnis von Strecke/Median System A
            wenn es größer als 1,5 ist.

rechtsläufiger Drehwinkel der Achsen: System B nach A = 0.0010587 gon
Schwerpunkt der Passpunkte, System A, ys =2596602.838, xs =5687467.558
Schwerpunkt der Passpunkte, System B, Ys =2596603.206, Xs =5687467.522
Translationsunbekannte Y0 =   -116.152, X0 =     -4.903
-------------------------------------------------------------------------------


      Transformation der Elemente bzw. Koordinaten der gegebenen Passpunkte

                  vom System A            zum System B (Ziel)

Punktkennzeichen  yi          xi           Yi          Xi      vyi     vxi    Klaffe
-------------------------------------------------------------------------------

      1203 2596687.890 5686891.360  2596688.220 5686891.290  -0.028  -0.033   0.043
                                                            ry= 54% rx= 79%
      6510 2595998.620 5687413.310  2595998.990 5687413.290   0.003   0.006   0.007
                                                            ry= 80% rx= 52%
      6810 2596949.480 5687594.910  2596949.820 5687594.890  -0.030   0.022   0.037
                                                            ry= 79% rx= 71%
     90019 2596951.650 5688012.680  2596952.030 5688012.660   0.003   0.022   0.022
                                                            ry= 57% rx= 71%
     90074 2596426.550 5687425.530  2596426.970 5687425.480   0.053  -0.017   0.055
                                                            ry= 80% rx= 78%
-------------------------------------------------------------------------------


                 zu transformierende Punkte des Systems A

                    vom System A           zum System B (Ziel, berechnet)

Punktkennzeichen  yj          xj           Yj          Xj     uyj, uxj (angepasst)
-------------------------------------------------------------------------------

     90001 2596821.570 5687335.670  2596821.930 5687335.633  -0.006   0.002
     90012 2596238.630 5687555.390  2596239.025 5687555.354   0.026  -0.006
     90014 2596363.870 5687119.230  2596364.246 5687119.186   0.013  -0.012
     90020 2596957.190 5687928.980  2596957.567 5687928.959   0.001   0.021
     90021 2596951.020 5687845.090  2596951.389 5687845.067  -0.005   0.019
     90022 2596914.010 5687726.160  2596914.362 5687726.138  -0.021   0.019
     90034 2596827.340 5687754.500  2596827.700 5687754.477  -0.013   0.016
     90037 2596739.940 5687384.740  2596740.309 5687384.702   0.003   0.000
usw.
```

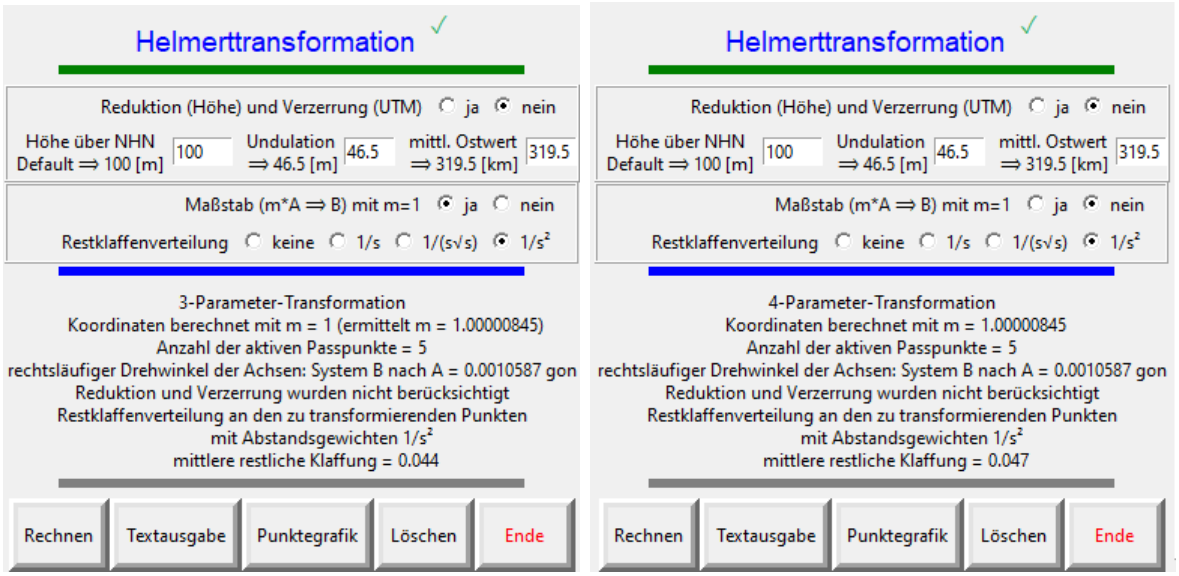


Abbildung 9
Ergebnisbeispiel mit Daten aus LVA NRW *[5] Seite 31-37*

Schlussbemerkungen

Der Quellcode dieses Programmes wird neben der compilierten Fassung ebenfalls zur Verfügung gestellt. Es soll vor allem den vermessungstechnischen oder geodätischen Nachwuchs dazu anregen, sich durchaus mit einem komplexeren Programm auseinander zu setzen. Man kann alles immer besser machen. Eine Haftung und Gewährleistung kann nicht übernommen werden. Wenn man selber programmieren möchte, sind einige Passagen des Programms als Hilfestellungen sicherlich nützlich. Auch sollte man sich nicht scheuen, Python-Erweiterungen wie <NumPy> zu nutzen.

Auch eine Nachfrage bei ChatGPD bietet zumindest eine Lösungsrichtung; nicht alle Vorschläge sind sofort brauchbar. Da ChatGPD von Programmierern geschaffen sein muss, sind dort Fragen und Antworten zur Programmierung erstaunlich gut vertreten.

Python ist eigentlich leicht zu erlernen – eine gewisse Geduld vorausgesetzt –, hat aber als ältere Sprache auch seine zeit- und entwicklungsabhängige Tücken, was sich in der nicht einheitlichen, verschiedenfachen Syntax ausdrückt.

Liste der Beispieldaten

Daten für Eingabe und Printdateien:

- Beispiel 1: orthogonal, entnommen aus Benning [1], Statistik...,
  3. Auflage, S. 264
- Beispiel 2: orthogonal, entnommen aus Formelsammlung Nds 2012 S. 79, 89
- Beispiel 3: orthogonal, entnommen aus Niemeyer [2]. Ausgleichungsrechnung,
  2. Aufl. S. 373
- Beispiel 4: orthogonales Extrembeispiel; Polaraufnahme
- Beispiel 5: orthogonales und polares Extrembeispiel
- Beispiel 6: orthogonal, entnommen aus Transformation von Koordinaten und Höhen in der Landesvermessung - Teil II: Praktische Anwendungsfälle, Seite 31-37

Literatur

[1] Benning, Statistik in Geodäsie, Geoinformation und Bauwesen, Verlag Wichmann, Heidelberg, 3. Aufl. 2010

[2] Niemeier, Ausgleichungsrechnung – Statistische Auswertemethoden, Verlag Walter de Gruyter, 2. Aufl. 2008

[3] Lenkungsgruppe AFIS–ALKIS–ATKIS Niedersachsen, Formelsammlung unter Berücksichtigung des amtlichen Bezugssystems ETRS89 mit UTM-Abbildung – Entwurf –, Landesbetrieb Landesvermessung und Geobasisinformation Niedersachsen (LGN), Stand: 1.7.2012 [Nds], Seiten 34, 35 und 42
Internet

[4] Transformation von Koordinaten und Höhen in der Landesvermessung Teil 1: Theoretische Grundlagen LVA NRW, 2. Aufl. 1999, Seiten 51, 52
http://nbn-resolving.de/urn:nbn:de:hbz:5:2-958

[5] Transformation von Koordinaten und Höhen in der Landesvermessung Teil 2: Praktische Anwendungsfälle LVA NRW, 1. Aufl. 1999
http://nbn-resolving.de/urn:nbn:de:hbz:5:2-973

Norbert Fuhrmann

Python ⇒ Vermessungspaket – Manual für pyvermessung.py

Vermessungstechnisches Erweiterungsmodul für Python auf NumPy-Basis mit linearen und trigonometrischen Aufgaben, Kreisbögen und Klothoiden für das Vermessungs- und Katasterwesen.

In dem beigefügten Programm <Vermessung102-numpy-Beispiele.py> sind alle Funktionen aus dem Paket <pyvermessung.py> durch Rechenbeispiele mit Ein- und Ausgaben enthalten. Das Ergebnis aus diesem Programm findet sich in <Vermessung-numpy-Beispiele.txt> wieder; oder nach Installation und Ausführung in der Datei <output.txt> im gleichen Programmverzeichnis.

<pyvermessung.py> als Erweiterungsmodul für Python ist für den Einsatz in Windows gedacht sind und kann im beigefügten Programm <Vermessung-numpy-Beispiele.py> direkt oder modifiziert ausgeführt werden[1]. Das Modul <pyvermessung.py> kann in andere Python-Programme eingebaut werden. Große vermessungstechnische Programmsysteme mit Datenbankanbindungen soll es nicht ersetzen. Die Programme sind in Python geschrieben worden. Python ist eine höhere, interpretierende Programmiersprache von der <Python Software Foundation>. Der Python-Interpreter <IDLE Shell 3.13.0> für Windows kann im Internet unter https://www.python.org/downloads/ heruntergeladen werden[2]. Grundlage der hier vorliegenden Programme ist die <Release version 3.13.0 Oct 7 2024>. NumPy ist ein Open-Source-Projekt, wissenschaftlich orientiert, das numerisches Rechnen mit Python ermöglicht. Das Programm ist auch unter <Android> mit dem Programm <Pydroid 3> lauffähig (s. Seite 3).

Nach der Installation des Python-Interpreters:
Das Paketverwaltungsprogramm *pip* dient zur Installation zusätzlicher Module. Unter *windows* wird *pip* bereits gemeinsam mit Python installiert. Zur Nutzung von *pip* öffne man unter *windows* die Eingabeaufforderung (WindowsPowerShell)[3]. Welche Module bereits installiert sind, kann in der Eingabeaufforderung mit <pip list> (bzw. <pip3 list> ab Python-Version 3) getestet werden. Wenn nicht installiert, gebe man dort ein:

pip install numpy
pip install tabulate
pip install sympy (für den Programmteil Nr. 58)

[1] Wer nicht programmieren möchte, nutze das Modul für separate Aufgaben, indem man eine Kopie von <Vermessung-numpy-Beispiele.py> macht und die dortigen Beispieldaten ändert. Anfang und Ende von Teilen, die nicht gebraucht werden, können durch drei Anführungszeichen (obere Gänsefüßchen) """ auskommentiert werden.

[2] Hinweis: Eine Alternative besteht auch in der Python-Installation von *Anaconda* mit der ausführlichen Programmieroberfläche <Spyder> an Stelle von <IDLE> aus https://www.anaconda.com/download. Für den Anfang sei aber <IDLE> empfohlen. Die Programme wurden mit <autopep8> formatiert, das auch in <spyder> enthalten ist.

[3] Sind die Pfade bei der Python-Installation nicht gesetzt worden, muss in der Windows-Eingabeaufforderung mit cd-Befehlen in das Unterverzeichnis <python-Installation><Scripts> gewechselt werden.

Im Header von <pyvermessung.py> bzw. des Hauptprogramms müssen folgende Befehlsfolgen enthalten sein:

<pyvermessung.py>	**Hauptprogramm**
import sys	import sys
from tkinter import messagebox	import time
import numpy as np	import pyvermessung as ve
from sympy import symbols, Eq, sqrt, solve	import numpy as np
	import tabulate

Die einzelnen Funktionen in diesem Paket sind in ihrer Struktur sehr unterschiedlich. Sie sind aber vielfach auf *numpy* abgestellt. *numpy* ist im Wesentlichen ein mathematisches, in der zeitlichen Ausführung und Speichermethodik optimiertes Paket. Es ist für die Verarbeitung großer Datenmassen ausgelegt, wie dies bei Vektoren und Matrizen vielfach vorkommt. Im vermessungstechnischen, geometrischen Bereich, wie hier behandelt, ist dies weniger von Bedeutung. Es kann durchaus nützlich und klarer sein, hier in der Vermessungstechnik gebräuchliche Rechenwege zu gehen. Von der mathematischen Betrachtungsweise her kann es beispielsweise bei Aufgaben für Radien mehrere Lösungen geben, diese sind aber stets positiv. Von der vermessungstechnischen Seite ist nur ein Radius von Bedeutung, der in Rechenrichtung gesehen positiv oder negativ definiert werden kann, sowohl ob eine Rechts- oder Linkskurve vorliegt. Für die Lösungsmöglichkeiten in den Funktionen wurde auf vielfältige Varianten geachtet.

Um eine mögliche Massenverarbeitung zu realisieren, muss in manchen Modulen die Dateneingabe in einem Array erfolgen. Die Strukturierung der Arrays ist vorgegeben. Damit dieses Manual nicht unübersichtlich wird, werden die entsprechenden Arrays für gekennzeichnete Programmteile in einem Anhang separat ausgegeben. Die Namen der Arrays für die Eingabe und auch der Ausgabe werden im Manual fett mit einem weiteren Hinweis angegeben. Man kann auch das Beispiel <Vermessung-numpy-Beispiele.py> als Erläuterung zur Hand nehmen, um zu vergleichen.

Ein Array (hier mit dem Namen <koordinaten>) ist beispielsweise wie folgt formatiert. In den Unterprogrammen werden die Daten entsprechend in eine Matrix umgesetzt.

```
koordinaten = [          | → Pktn. (num), y, x,
1, 1000.1,   2000.3,     | → P1  i.d.R. der Anfangspunkt
2, 1052.2,   2155.,      | → P2  i.d.R. der Endpunkt
3, 1029.3,   2002.23,    |
4, 1022.32,  2023.1,     |
5, 1033.,    2013.3      |
]                        |
```

Punkte mit aufsteigender Punktnummerierung bzw. das Symbol ▶ in den Abbildungen geben die Rechenrichtung im Element an, wenn nicht anders angegeben. Rechts davon liegende Elemente sind positiv (+), links davon liegende Elemente sind negativ (−).
Beispiel: Lokale orthogonale Elemente → Anfangspunkt = P_1, Endpunkt = P_2; ⟶ die Ordinaten rechts davon liegender Punkte sind positiv (+), links davon liegende negativ (−).

Gleichnamige Funktionen werden durch verschiedene Anzahl der Parameter in der Eingabe unterschiedlich gesteuert. In manchen Funktionen entstehen durch gemischt-quadratische

Gleichungen komplexere Lösungsansätze. Es folgen hier dann iterative Methoden. Die Anzahl der Iterationen wurden i.d.R. auf n = 200 beschränkt. Weitergehende Iterationen dürften zu keiner Lösung führen. Eingangswerte oder Näherungskoordinaten wären dann zu überprüfen. Auch Symbolische Algebra (sympy) bietet hier Lösungen, die aber rechenintensiver sind und damit länger andauern

Die Funktionen <klothoide2(A, L)> und <klothoide3(A, L)> sind lediglich zum Experimentieren als Variationen in <pyvermessung.py> eingefügt. Die Benutzung von <klothoide(<Parameter>)> wäre vorzuziehen.

In dem beigefügten Programm <Vermessung102-numpy-Beispiele.py> sind alle Funktionen aus dem Paket <pyvermessung.py> durch Rechenbeispiele mit Ein- und Ausgabe enthalten. Das Ergebnis aus diesem Programm findet sich in <Vermessung-numpy-Beispiele.txt> als Muster wieder oder aber die Ausgabe erfolgt bei aktiver Handhabung in IDLE direkt auf dem Bildschirm oder als Textdatei (ut-8-Code) in der Datei <output.txt> im Programmverzeichnis.

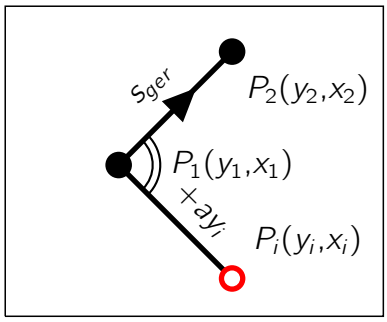
Wenn Fehler auftreten, sind zunächst die Koordinateneingaben der Punkte zu überprüfen (Plausibilität der Punktlage und auch der Konstruktion für eine Lösung). Fehler sind auch angezeigt, wenn unerlaubt dieselben Koordinaten für verschiedene Punkte eingegeben werden, keine Schnittbildungen möglich sind (beispielsweise bei Parallelen), Kreise einen zu kleinen oder großen Radius haben, Parameter fehlen usw. Werden nan-Werte anstatt Zahlen ausgegeben, liegen keine Lösungen vor; so zum Beispiel bei Wurzeln aus negativen Zahlen.

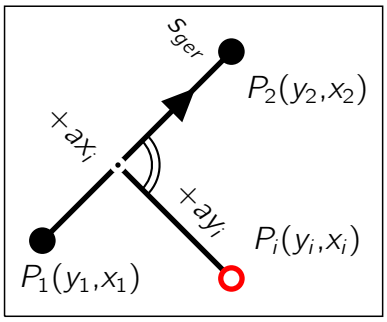
Installation für Android:
Für die Anwendung auf einem Smartphone bzw. Tablet (mit Android) muss im Google <Play store> unter dem Stichwort <python 3 for android> die App <Pydroid 3 - IDE for Python 3>[4] heruntergeladen werden.

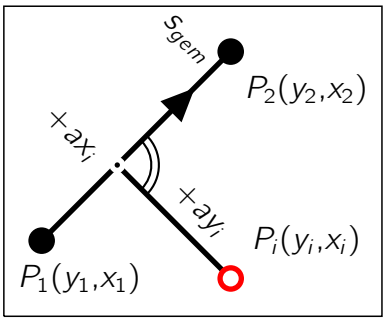
Für Android muss von GooglePlay <*Pydroid repository plugin*> geladen werden, um die Ergänzungspakete **numpy, sympy** und **tabulate** installieren zu können:
Symbol ≡ (oben links) drücken, unter >Pip< in >Libary name< numpy, sympy bzw. tabulate eingeben und >INSTALL< drücken.

Sonstige Einstellungen in <Pydroid 3>: (Textgröße) ⟹ Symbol ≡ (oben links) ⟹ Terminal settings ⟹ Text - Font size ⟹ ◉14 pt

[4] ⟹ <Pydroid 3 - IDE for Python 3.apk> (Version 4.01 arm64 bzw. 5.00 arm64).
https://apkpure.com/pydroid-3-ide-for-python-3/ru.iiec.pydroid3/versions
https://pydroid-3-ide-for-python-3.de.softonic.com/android
https://apkfab.com/de/pydroid-3-ide-for-python-3/ru.iiec.pydroid3
Andere, auch neuere Versionen können mitunter Dateipfade nicht finden oder verändern. (Ein allgemeines Problem bei Android; was auch bei Dateimanager auftritt.)







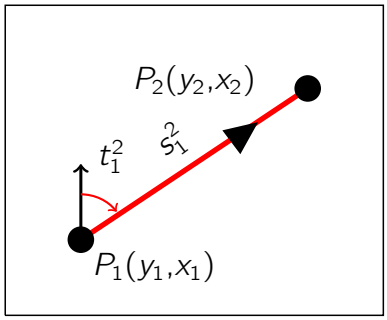
1 **Orthogon. Punkt 1**

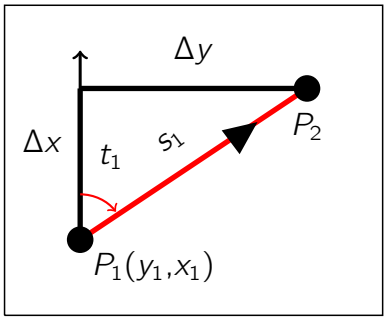
2 **Orthogon. Punkt 2**
4 **Orthogon. Punkte 4 ohne Fehlerverteilung**

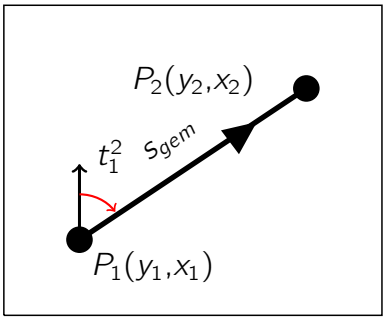
3 **Orthogon. Punkt 3**
5 **Orthogon. Punkte 5 mit Fehlerverteilung**

1 $y_i, x_i, s_{ger} = orthogonal(y_1, x_1, y_2, x_2, ay_i)$
2 $y_i, x_i, s_{ger} = orthogonal(y_1, x_1, y_2, x_2, ay_i, ax_i)$ (ohne Fehlerverteilung)
3 $y_i, x_i, s_{ger} = orthogonal(y_1, x_1, y_2, x_2, ay_i, ax_i, s_{gem})$ (mit Fehlerverteilung)
4 $y_i, x_i = orthogonal4(y_1, x_1, y_2, x_2, ay_i, ax_i)$ (ohne Verteilung) ← [**Array**]
5 $y_i, x_i, f_S = orthogonal5(y_1, x_1, y_2, x_2, ay_i, ax_i, s_{gem})$ (mit Verteilung) ← [**Array**]

▶ Aufrufe:
```
yi, xi, sger = orthogonal(y1, x1, y2, x2, ayi)
yi, xi, sger = orthogonal(y1, x1, y2, x2, ayi, axi)
yi, xi, sger = orthogonal(y1, x1, y2, x2, ayi, axi, sgem)
```
koord1 = orthogonal4(**koordinaten**) ← [**Array**]
koord1, fs = orthogonal5(**koordinaten**, sgem) ← [**Array**]







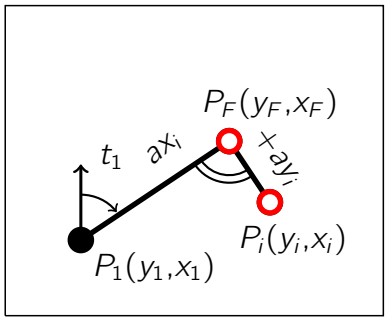
6 **Richtungswinkel und Entfernung**

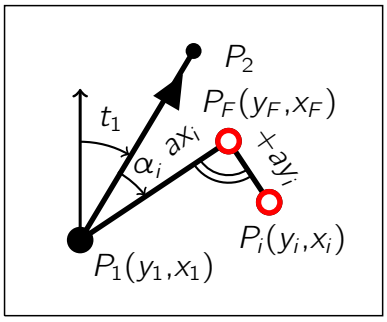
7 **Richtungswinkel aus Koordinatendifferenzen**

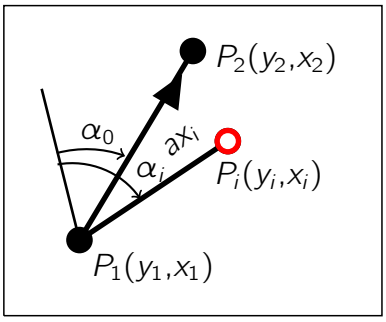
8 **Richtungsfunktionen, Maßstab**

6 $t_1^2, s_1^2 = richtung(y_1, x_1, y_2, x_2)$
7 $t_1, s_1 = richtung(\Delta y, \Delta x)$
8 $sin(t_1^2), cos(t_1^2), s_{ger}, Maßstab_{s_{ger}/s_{gem}} = richtung3(y_1, x_1, y_2, x_2, s_{gem})$

▶ Aufrufe:
```
t1, s1 = richtung(y1, x1, y2, x2)
t1, s1 = richtung(dy, dx)
sint, cost, sger, massstab = richtung3(y1, x1, y2, x2, sgem)
```



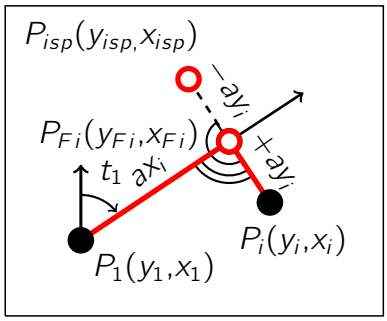


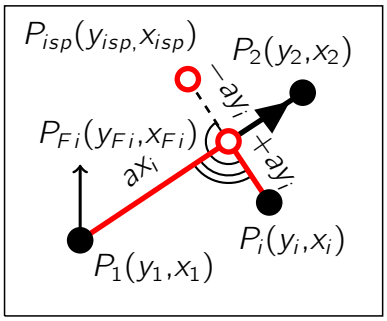


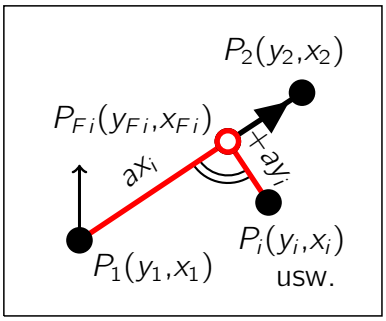
9 **10** **Polarer Punkt 1, 2** **11** **Polarer Punkt 3** **12** **Polare Punkte 4**

9 $y_i, x_i = polar(y_1, x_1, t_1, ax_i)$ (für $ay_1 = 0$)
10 $y_i, x_i, y_F, x_F = polar(y_1, x_1, t_1, ay_i, ax_i)$
11 $y_i, x_i, y_F, x_F = polar(y_1, x_1, t_1, \alpha_i, ay_i, ax_i)$
12 $y_i, x_i = polar4(y_1, x_1, y_2, x_2, \alpha_i, ax_i, \alpha_0)$ ← [**Array**]

▶ Aufrufe: yi, xi = polar(y1, x1, t1, axi)
yi, xi, yF, xF = polar(y1, x1, t1, ayi, axi)
yi, xi, yF, xF = polar(y1, x1, t1, alphai, ayi, axi)
koord2 = polar4(**polaraufnahme**, alpha0) ← [**Array**]



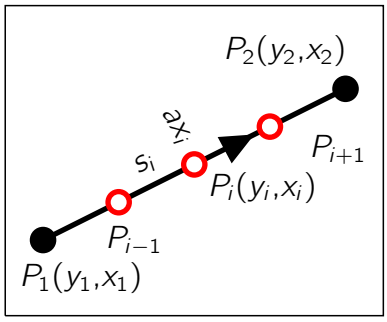


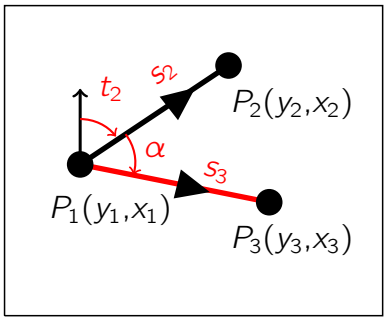


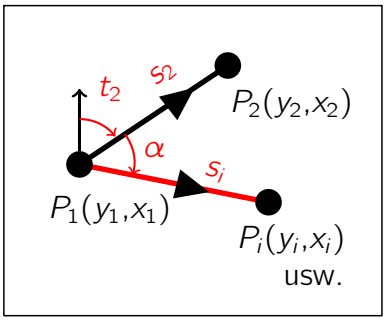
13 **Abstand Richtung Spiegelpunkt** **14** **Lot, Lotfußpunkt Orthog. Element, Spiegelpunkt** **15** **Lote, Lotfußpunkte Orthog. Elemente**

13 $y_F, x_F, a_{y_i}, a_{x_i}, y_{isp}, x_{isp} = lot(y_1, x_1, t_1, y_i, x_i)$
14 $y_F, x_F, a_{y_i}, a_{x_i}, y_{isp}, x_{isp} = lot(y_1, x_1, y_2, x_2, y_i, x_i)$
15 $y_F, x_F, a_{y_i}, a_{x_i} = lot3(y_1, x_1, y_2, x_2, y_i, x_i, usw.)$ ← [**Array**])

▶ Aufrufe: yF, xF, ayi, axi, yisp, xisp = lot(y1, x1, t1, xi, yi)
yF, xF, ayi, axi, yisp, xisp = lot(y1, x1, y2, x2, y3, x3)
koord3 = lot3(**koordinaten**) ← [**Array**]



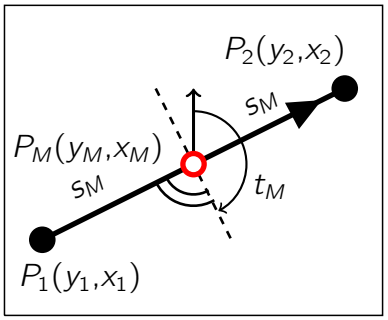


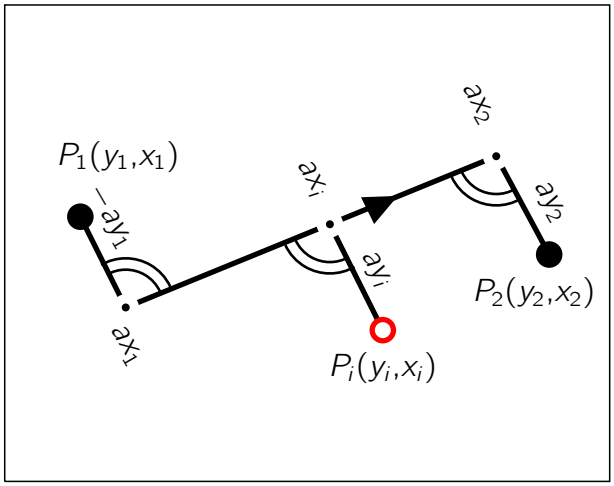


16 **Strecke teilen** **17** **Polarwinkel, Strecke** **18** **Polarwinkel, Strecken**

16 $y_i, x_i, s_i, s_{ger} = teilstrecken(y_1, x_1, y_2, x_2, n)$ ← [**Array**]
Teilung einer Strecke in (n+1) gleiche Teile; n = Anzahl der Zwischenpunkte
17 $t_2, s_2, \alpha, s_3 = polarwinkel1(y_1, x_1, y_2, x_2, y_3, x_3)$
18 $t_2, s_2, \alpha, s_3 = polarwinkel2(y_1, x_1, y_2, x_2, y_i, x_i)$ ← [**Array**]

▶ Aufrufe: **koord4** = teilstrecken(y1, x1, y2, x2, n) ← [**Array**]
t2, s2, alpha, s3 = polarwinkel1(y1, x1, y2, x2, y3, x3)
koord5 = polarwinkel2(**koordinaten**) ← [**Array**]



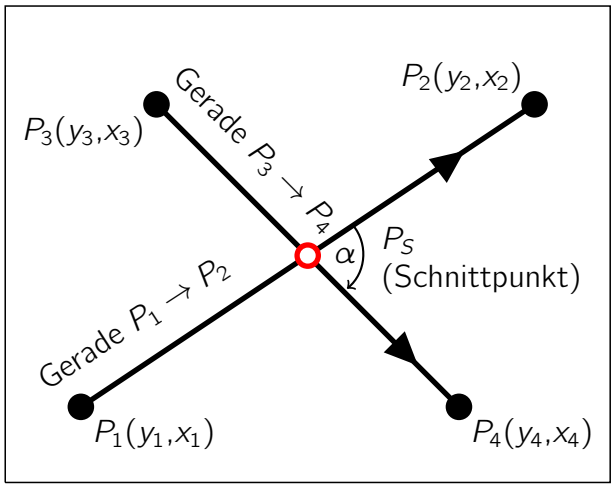


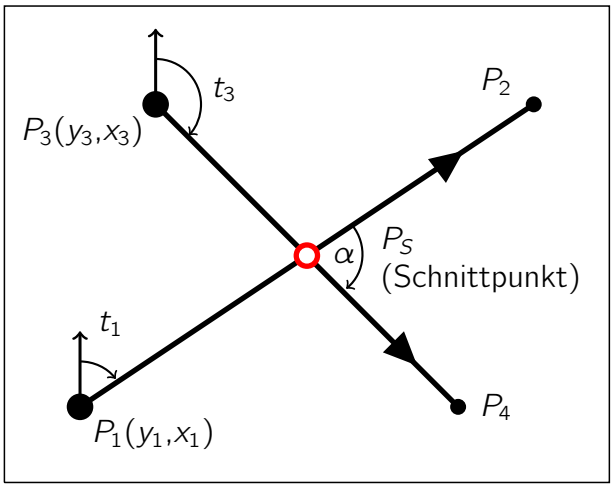
19 **Strecke halbieren, Mittelsenkrechte** **20** **21** **Transformation**

19 $y_M, x_M, t_M, s_M = mittelsenkrechte(y_1, x_1, y_2, x_2)$
20 $y_i, x_i, fs = transformation1(y_1, x_1, y_2, x_2, ay_1, ax_1, ay_2, ax_2, ay_i, ax_i)$
21 $y_i, x_i, fs = transformation2(y_1, x_1, y_2, x_2, ay_1, ax_1, ay_2, ax_2, ay_i, ax_i)$ ← [**Array**]

▶ Aufrufe:
yM, xM, tM, sM = mittelsenkrechte(y1, x1, y2, x2)
yi, xi, fs = transformation1(y1, x1, y2, x2, ay1, ax1, ay2, ax2, ayi, axi)
koord6, fs = transformation2(**transkoordinaten**) ← [**Array**]

Geradenschnitte:





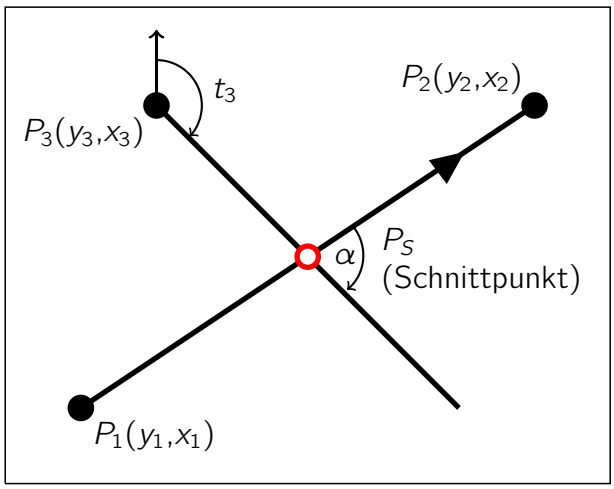
22 **Einfacher Geradenschnitt**

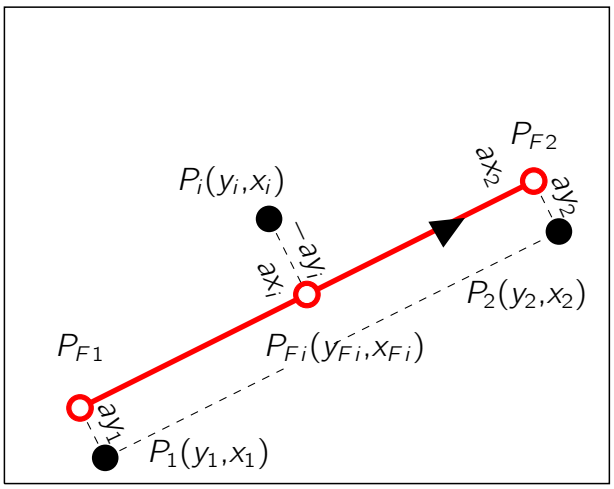
23 **Vorwärtsschnitt über Richtungswinkel**

22 $y_s, x_s, \alpha = geradenschnitt(y_1, x_1, y_2, x_2, y_3, x_3, y_4, x_4)$

23 $y_s, x_s, \alpha = geradenschnitt(y_1, x_1, t_1, y_3, x_3, t_3)$

▶ Aufrufe: yS, xS, alpha = geradenschnitt(y1, x1, y2, x2, y3, x3, y4, x4)
yS, xS, alpha = geradenschnitt(y1, x1, t1, y3, x3, t3)





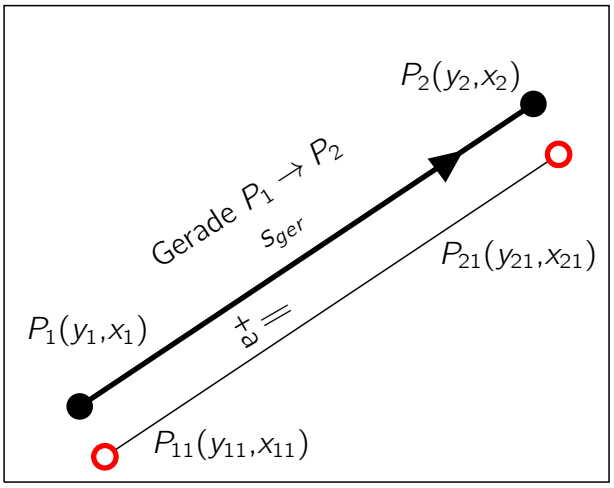
24 **Geradenschnitt über einen Richtungswinkel**

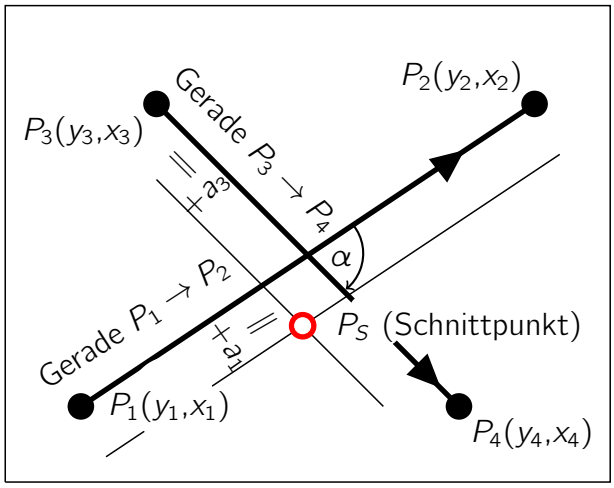
25 **Ausgleichende Gerade**

24 $y_S, x_S, \alpha = geradenschnitt(y_1, x_1, y_2, x_2, y_3, x_3, t_3)$

25 $ay_1, ax_1, usw. = geradenausgleich(y_1, x_1, y_2, x_2, ...y_n, x_n)$ ← [**Array**]

▶ Aufrufe: yS, xS, alpha = geradenschnitt(y1, x1, y2, x2, y3, x3, t3)
koord7 = geradenausgleich(**koordinaten**) ← [**Array**]

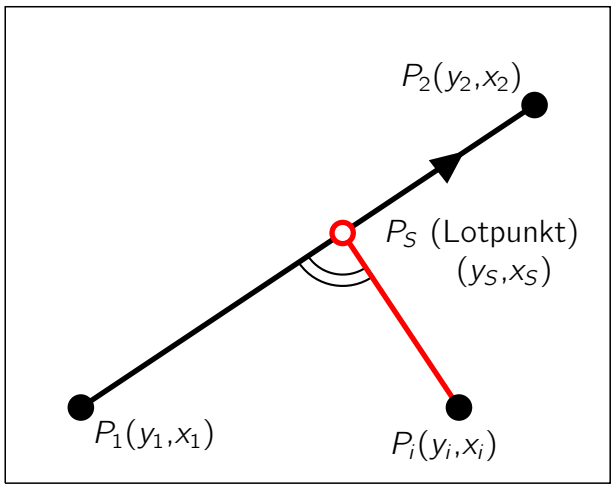


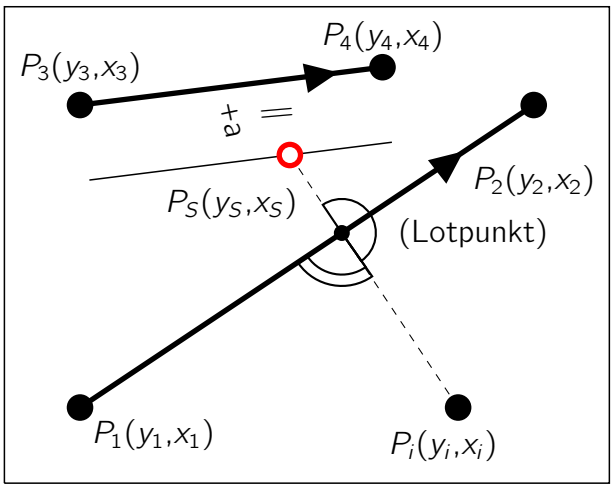


26 **Parallele** **27** **Parallelschnitt**

26 $y_{11}, x_{11}, y_{21}, x_{21}, s_{ger} = parallel(y_1, x_1, y_2, x_2, \pm a)$
27 $y_S, x_S, \alpha = parallelschnitt(y_1, x_1, y_2, x_2, \pm a_1, y_3, x_3, y_4, x_4, \pm a_3)$

▶ Aufrufe: y11, x21, y21, sger = parallel(y1, x1, y2, x2, ± a)
yS, xS, alpha = parallelschnitt(y1, x1, y2, x2, ± a1, y3, x3, y4, x4, ± a3)

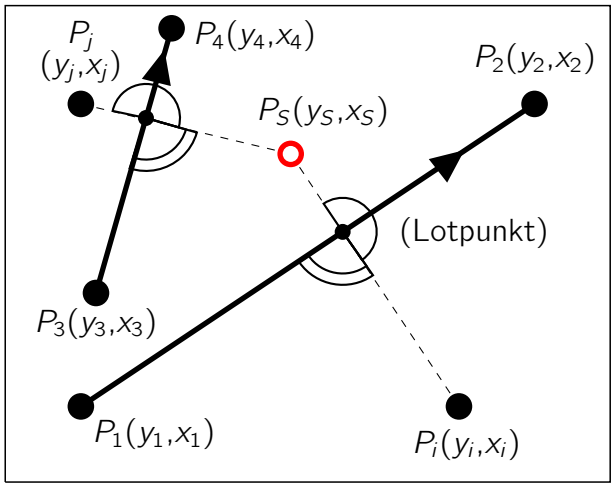


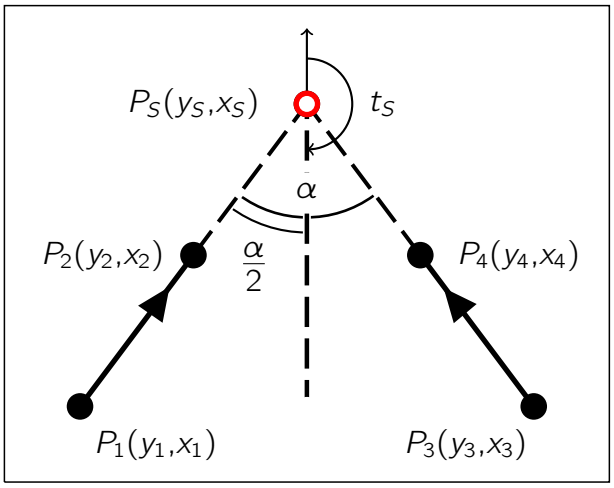


28 **Senkrechtschnitt** **29** **Senkrecht-Parallelschnitt**

28 $y_S, x_S = senkrechtschnitt(y_1, x_1, y_2, x_2, y_i, x_i)$
29 $y_S, x_S = senkrecht_parallel(y_1, x_1, y_2, x_2, y_3, x_3, y_4, x_4, y_i, x_i, \pm a)$

▶ Aufrufe: yS, xS = senkrechtschnitt(y1, x1, y2, x2, yi, xi)
yS, xS = senkrecht_parallel(y1, x1, y2, x2, y3, x3, y4, x4, yi, xi, ± a)



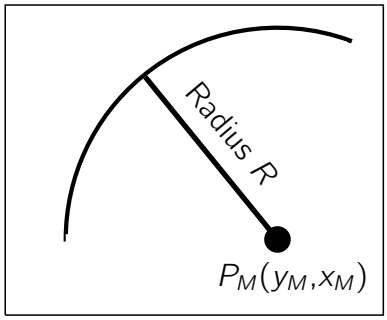


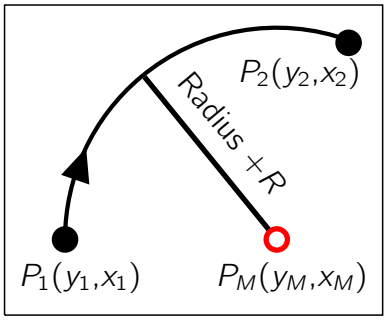
30 **Doppelter Senkrechtschnitt** **31** **Winkelhalbierende**

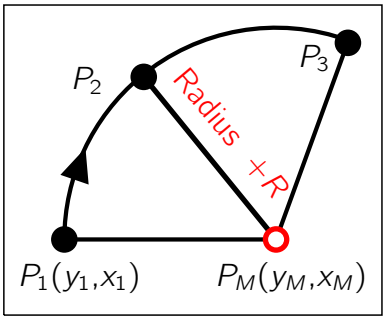
30 $y_S, x_S = doppelt_senkrecht(y_1, x_1, y_2, x_2, y_i, x_i, y_3, x_3, y_4, x_4, y_j, x_j)$
31 $y_S, x_S, t_S, \alpha = winkelhalbierende(y_1, x_1, y_2, x_2, y_3, x_3, y_4, x_4)$

▶ Aufrufe: yS, xS = doppelt_senkrecht(y1, x1, y2, x2, yi, xi, y3, x3, y4, x4, yj, xj)
yS, xS, tS, alpha = winkelhalbierende(y1, x1, y2, x2, y3, x3, y4, x4)

Kreisdefinitionen:



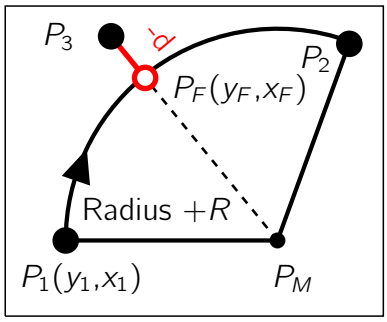


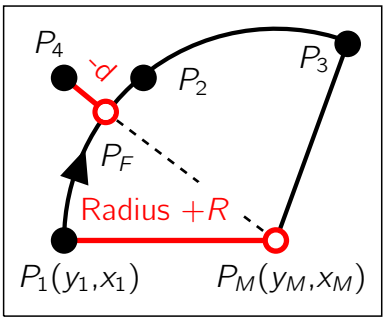


Kreis 1 **32** **Kreis 2** **33** **Kreis 3**

32 $y_M, x_M = kreis(y_1, x_1, y_2, x_2, \pm R)$
33 $y_M, x_M, \pm R = kreis(y_1, x_1, y_2, x_2, y_3, x_3)$

▶ Aufrufe: yM, xM = kreis(y1, x1, y2, x2, ± R)
yM, xM, ± R = kreis(y1, x1, y2, x2, y3, x3)

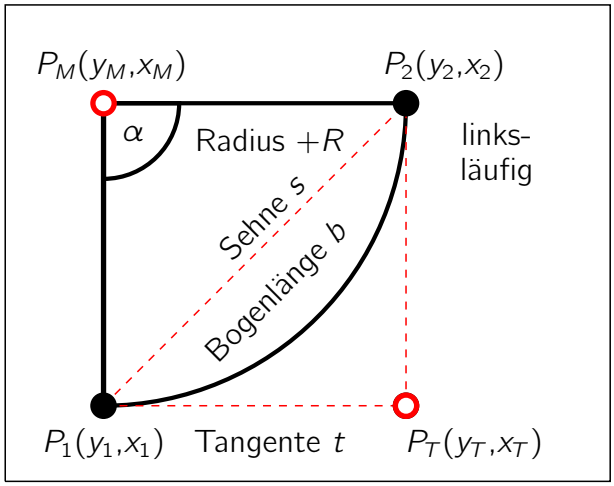


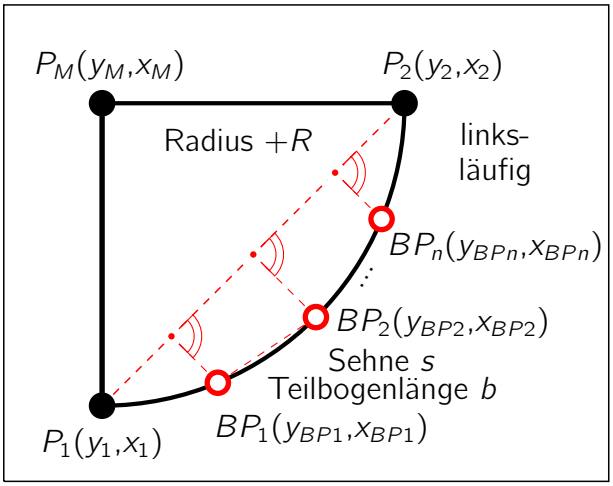


34 **Lot auf Kreis 1** **35** **Lot auf Kreis 2**

34 $y_F, x_F, d = lot_kreis(y_1, x_1, y_2, x_2, y_3, x_3, \pm R)$
35 $y_F, x_F, d, y_M, x_M, \pm R = lot_kreis(y_1, x_1, y_2, x_2, y_3, x_3, y_4, x_4)$
Bei $-d$ liegen P_3 bzw. P_4 außerhalb des Kreisbogens, bei $+d$ innerhalb.

▶ Aufrufe: yF, xF, d = lot_kreis(y1, x1, y2, x2, y3, x3, ± R)
yF, xF, d, yM, xM, ± R = lot_kreis(y1, x1, y2, x2, y3, x3, y4, x4)

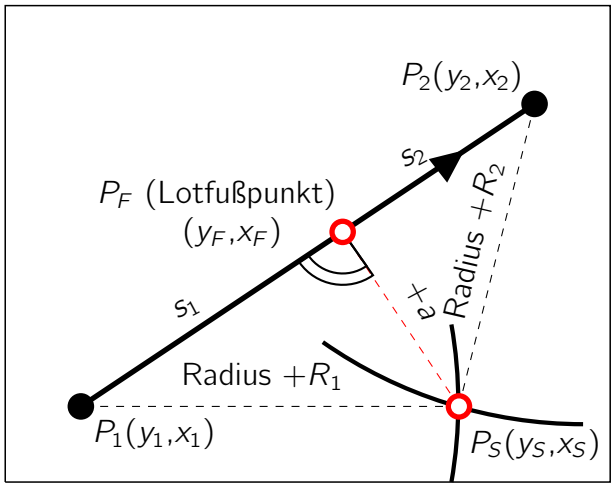


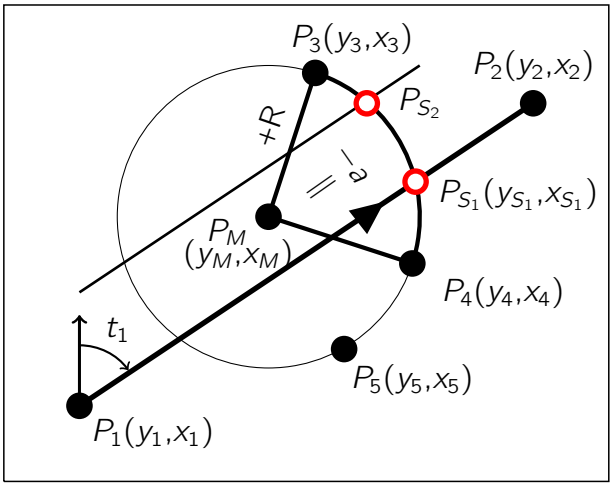


36 **Kreisbogen** **37** **Bogenteilung**

36 $y_M, x_M, y_T, x_T, t, s, b, \alpha = kreisbogen(y_1, x_1, y_2, x_2, +R)$
37 $y_M, x_M, s, b, BPx_1, BPx_1... = bogenteilung(y_1, x_1, y_2, x_2, +R, n)$
n = Anzahl der Zwischenpunkte

▶ Aufrufe: yM, xM, yT, xT, t, s, b, alpha = kreisbogen(y1, x1, y2, x2, +R)
yM, xM, s, b, **elemente** = bogenteilung(y1, x1, y2, x2, +R, n) ← [**Array**]



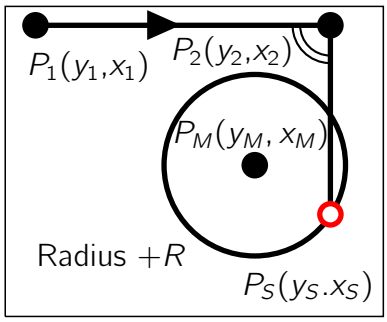


38 **Kreisbogenschnitt**

39 **40** **41** **42** **43** **44**
Schnitt Kreis-Gerade

38 $y_S, x_S, y_F, x_F, s_1, s_2, a = kreis_kreis(y_1, x_1, +R_1, y_2, x_2, +R_2)$
39 $y_S, x_S = kreis_gerade(y_1, x_1, t_1, y_M, x_M, +R)$
40 $y_S, x_S = kreis_gerade(y_1, x_1, y_2, x_2, y_M, x_M, +R)$
41 $y_S, x_S = kreis_gerade(y_1, x_1, y_2, x_2, y_M, x_M, +R, \pm a)$
42 $y_S, x_S, y_M, x_M = kreis_gerade(y_1, x_1, y_2, x_2, y_3, x_3, y_4, x_4, +R)$
43 $y_S, x_S, y_M, x_M = kreis_gerade(y_1, x_1, y_2, x_2, y_3, x_3, y_4, x_4, +R, \pm a)$
44 $y_S, x_S, y_M, x_M, R = kreis_gerade(y_1, x_1, y_2, x_2, y_3, x_3, y_4, x_4, y_5, x_5, \pm a)$

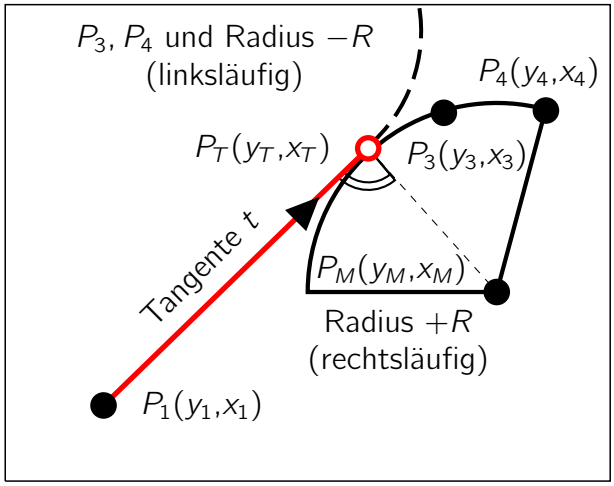
▶ Aufrufe:
```
yS, xS, yF, xF, s1, s2, a = kreis_kreis(y1, x1, R1, y2, x2, +R2)
yS, xS = kreis_gerade(y1, x1, t1, yM, xM, +R)
yS, xS = kreis_gerade(y1, x1, y2, x2, yM, xM, +R)
yS, xS = kreis_gerade(y1, x1, y2, x2, yM, xM, +R, ± a)
yS, xS, yM, xM = kreis_gerade(y1, x1, y2, x2, y3, x3, y4, x4, +R)
yS, xS, yM, xM = kreis_gerade(y1, x1, y2, x2, y3, x3, y4, x4, +R, ± a)
yS, xS, yM, xM, R = kreis_gerade(y1, x1, y2, x2, y3, x3, y4, x4, y5, x5, ± a)
```

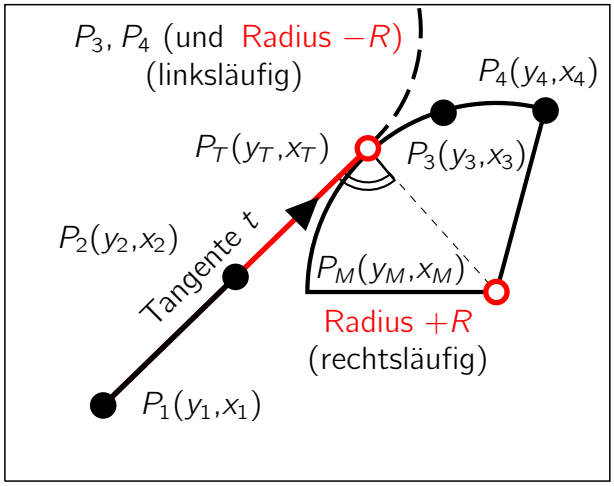


45 **Senkrechtschnitt Kreis**

45 $y_S, x_S = senkrecht_kreis(y_1, x_1, y_2, x_2, y_M, x_M, +R)$

▶ Aufruf:
```
yS, xS = senkrecht_kreis(y1, x1, y2, x2, yM, xM, +R)
```





46 **47** **Kreis, bewegliche Tangente mit Kreismittelpunkt und Radius**

48 **Kreis, feste, tangentiale Gerade und 2 Kreispunkte**

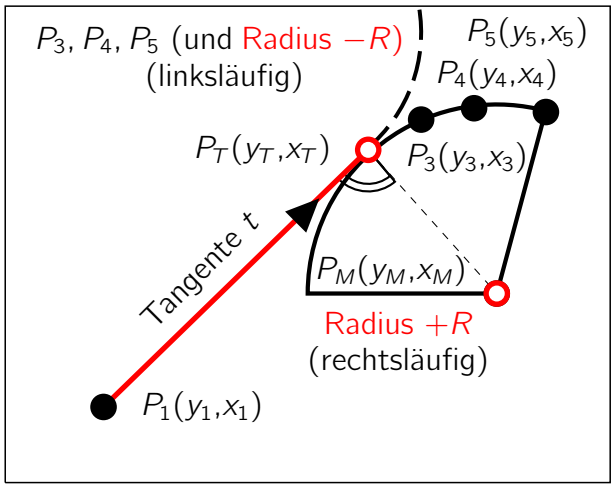
46 $y_T, x_T, t = tangente1(y_1, x_1, y_M, x_M, \pm R)$

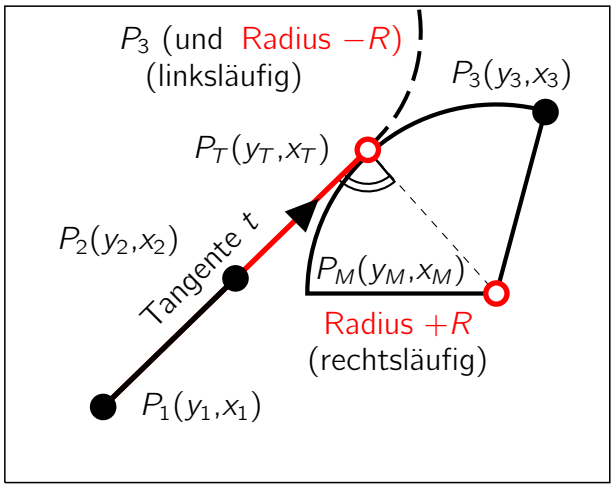
47 $y_T, x_T, t, y_M, x_M = tangente1(y_1, x_1, y_3, x_3, y_4, x_4, \pm R)$

48 $y_T, x_T, t, y_M, x_M, \pm R = tangente2(y_1, x_1, y_2, x_2, y_3, x_3, y_4, x_4)$

▶ Aufrufe: yT, xT, t = tangente1(y1, x1, yM, xM, ± R)
yT, xT, t, yM, xM = tangente1(y1, x1, y3, x3, y4, x4, ± R)
yT, xT, t, yM, xM, ± R = tangente2 (y1, x1, y2, x2, y3, x3, y4, x4)

$+R$: P_M rechts von der Tangente oder $-R$: P_M links von der Tangente





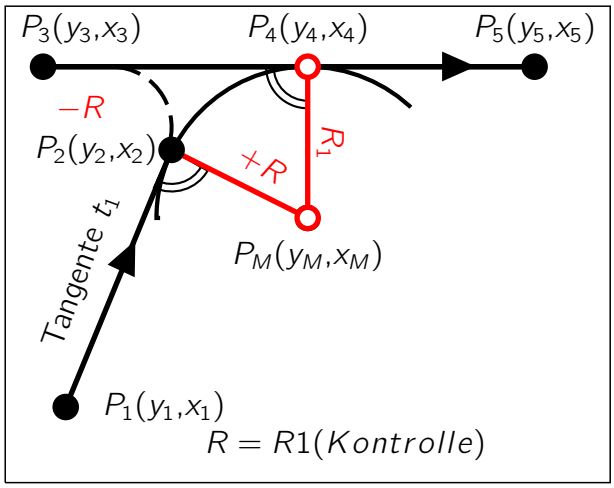
49 **Kreis, bewegliche Tangente und 3 Kreispunkte**

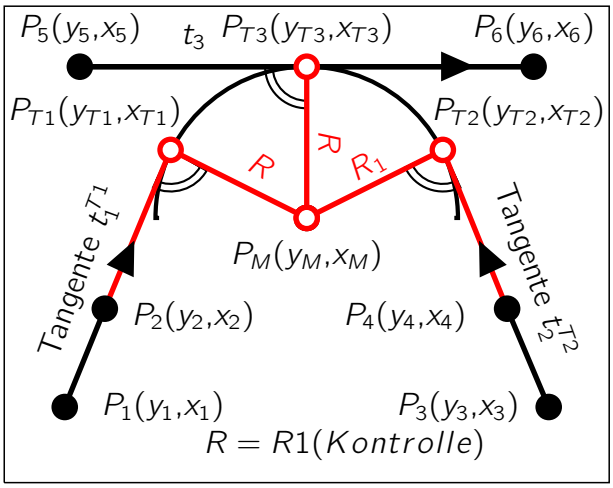
50 **Kreis, feste, tangentiale Gerade mit Kreispunkt und Radius**

49 $y_T, x_T, t, y_M, x_M, \pm R = tangente3(y_1, x_1, y_3, x_3, y_4, x_4, y_5, x_5)$

50 $y_T, x_T, t, y_M, x_M = tangente4(y_1, x_1, y_2, x_2, y_3, x_3, \pm R)$

▶ Aufrufe: yT, xT, t, yM, xM, ± R = tangente3(y1, x1, y3, x3, y4, x4, y5, x5)
yT, xT, t, yM, xM = tangente4(y1, x1, y2, x2, y3, x3, ±R)





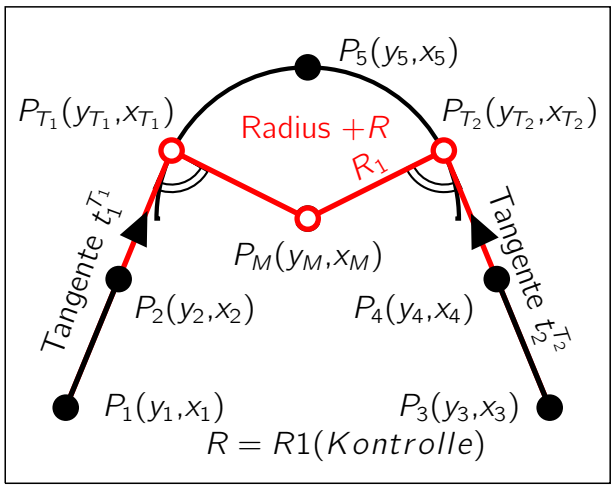
51 **Kreis, Tangente mit fester Länge und feste, tangentiale Gerade**

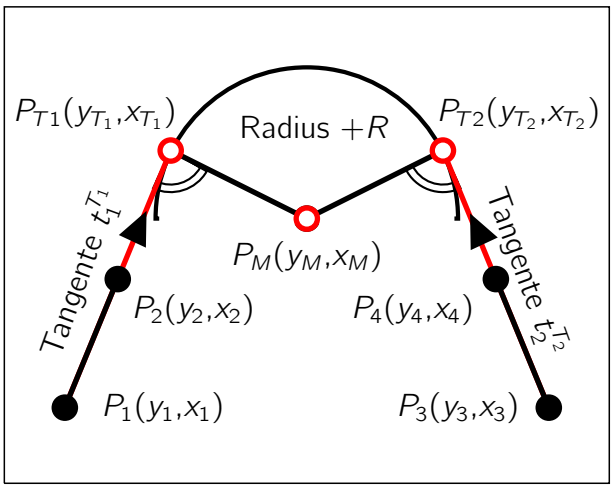
52 **Kreis, 3 feste, tangentiale Geraden**

51 $y_4, x_4, y_M, x_M, R, R_1 = tangente5(y_1, x_1, y_2, x_2, y_3, x_3, y_5, x_5)$ $(Rechtskurve)$
$y_4, x_4, y_M, x_M, -R, -R_1 = tangente5(y_1, x_1, y_2, x_2, y_3, x_3, y_5, x_5, -1)$ $(Linkskurve)$

52 $y_{T_1}, x_{T_1}, y_{T_2}, x_{T_2}, y_{T_3}, x_{T_3}, y_M, x_M, t_1, t_2, t_3, R, R_1 = tangente6(y_1, x_1 bis\ y_6, x_6)$

▶ Aufrufe:
```
y4, x4, yM, xM, R, R1 = tangente5(y1, x1, y2, x2, y3, x3, y5, x5)
y4, x4, yM, xM, R, R1 = tangente5(y1, x1, y2, x2, y3, x3, y5, x5, -1)
yT1, xT1, yT2, xT2, yT3, xT3, yM, xM, t1, t2, t3, R, R1 =
    tangente6(y1, x1 bis y6, x6)
```





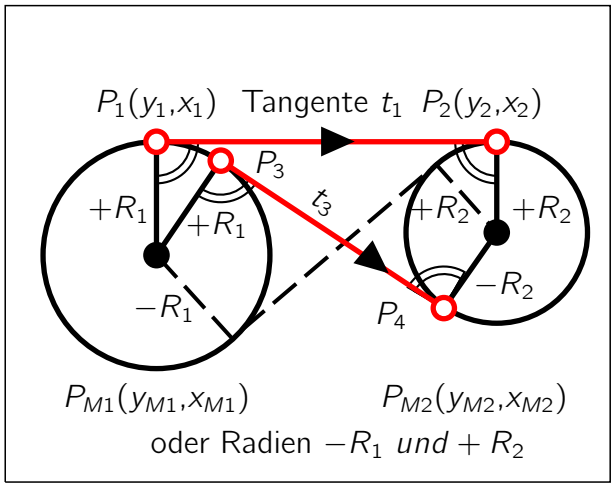
53 **Kreis, 2 feste, tangentiale Geraden und 1 Kreispunkt**

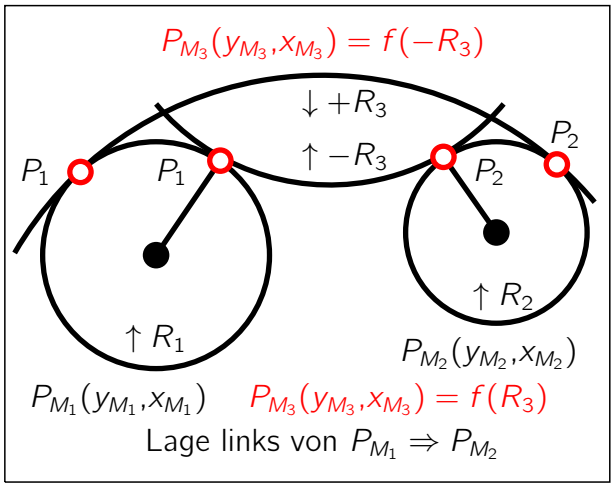
54 **Kreis, 2 feste, tangentiale Geraden und Radius**

53 $y_{T_1}, x_{T_1}, y_{T_2}, x_{T_2}, y_M, x_M, t1, t2, R, R_1 = tangente7(y_1, x_1, y_2, x_2, y_3, x_3, y_4, x_4, y_5, x_5)$

54 $y_{T_1}, x_{T_1}, y_{T_2}, x_{T_2}, y_M, x_M, t1, t2 = tangente8(y_1, x_1, y_2, x_2, y_3, x_3, y_4, x_4, +R)$

▶ Aufrufe:
```
yT1, xT1, yT2, xT2, yM, xM, t1, t2, R, R1 =
    tangente7(y1, x1, y2, x2, y3, x3, y4, x4, y5, x5)
yT1, xT1, yT2, xT2, yM, xM, t1, t2 =
    tangente8(y1, x1, y2, x2, y3, x3, y4, x4, +R)
```



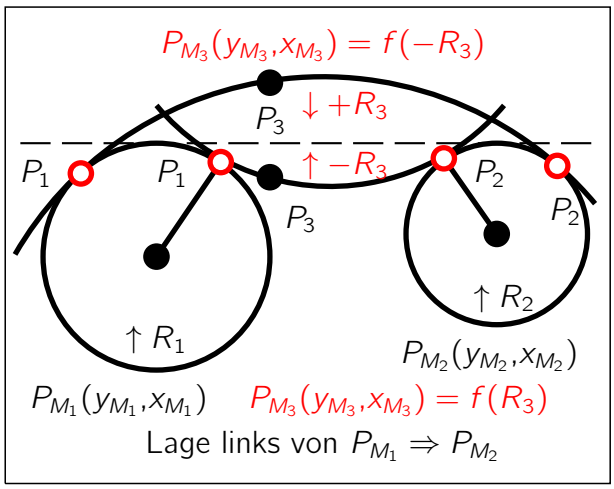


55 **56** **Tangenten, 2 Kreise außen und innen**

57 **Tangential, 3 Kreise Umkreis und Ankreis**

55 $y_1, x_1, y_2, x_2, t_1 = tangente9(y_{M_1}, x_{M_1}, \pm R_1, y_{M_2}, x_{M_2}, \pm R_2)$
56 $y_3, x_3, y_4, x_4, t_3 = tangente10(y_{M_1}, x_{M_1}, \pm R_1, y_{M_2}, x_{M_2}, \mp R_2)$
57 $y_1, x_1, y_2, x_2, y_{M_3}, x_{M_3} = tangente11(y_{M_1}, x_{M_1}, R_1, y_{M_2}, x_{M_2}, R_2, \pm R_3)$

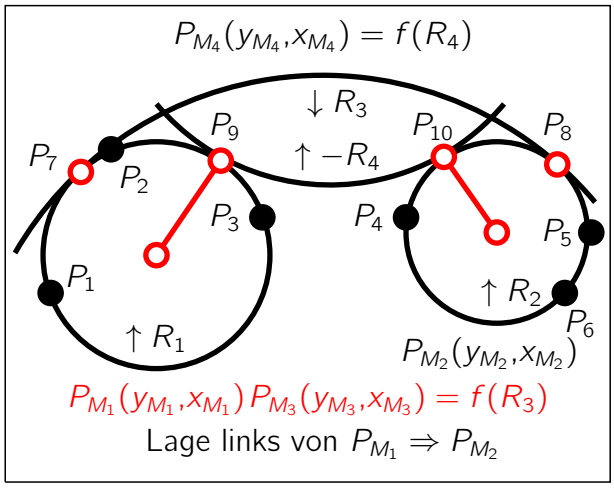
▶ Aufrufe:
```
y1, x1, y2, x2, t1 = tangente9(yM1, xM1, +R1, yM2, xM2, +R2)
y1, x1, y2, x2, t1 = tangente9(yM1, xM1, -R1, yM2, xM2, -R2)
y3, x3, y4, x4, t3 = tangente10(yM1, xM1, +R1, yM2, xM2, -R2)
y3, x3, y4, x4, t3 = tangente10(yM1, xM1, -R1, yM2, xM2, +R2)
y1, x1, y2, x2, yM3, xM3 = tangente11(yM1, xM1, R1, yM2, xM2, R2,± R3)
```

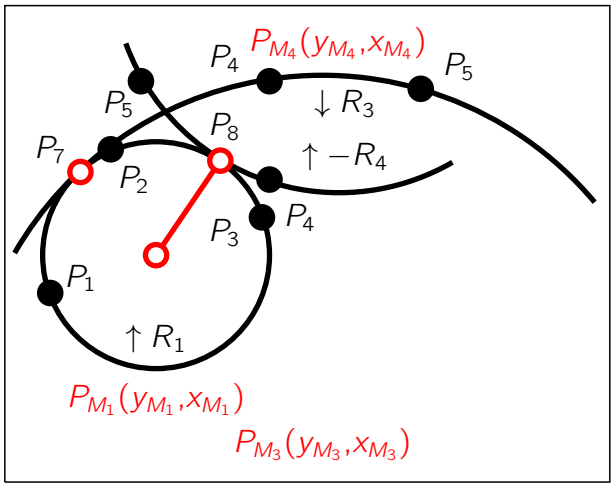


58 **Tangential, 2 Kreise, 1 Punkt, Umkreis und Ankreis**

58 $y_1, x_1, y_2, x_2, y_{M_3}, x_{M_3}, \pm R_3 = tangente12(y_{M_1}, x_{M_1}, R_1, y_{M_2}, x_{M_2}, R_2, y_3, x_3)$

▶ Aufruf:
```
y1, x1, y2, x2, yM3, xM3, ± R3 =
                tangente12(yM1, xM1, R1, yM2, xM2, R2, y3, x3)
```





59 **60** **tangentiale 3 Kreise, 3 Punkte Umkreis und Ankreis**

61 **62** **tangentiale Anschlussbögen Umkreis und Ankreis**

59 $y_7, x_7, R_1, y_8, x_8, R_2, y_{M_3}, x_{M_3} = tangente13(y_1, x_1\ bis\ y_6, x_6, R_3)$

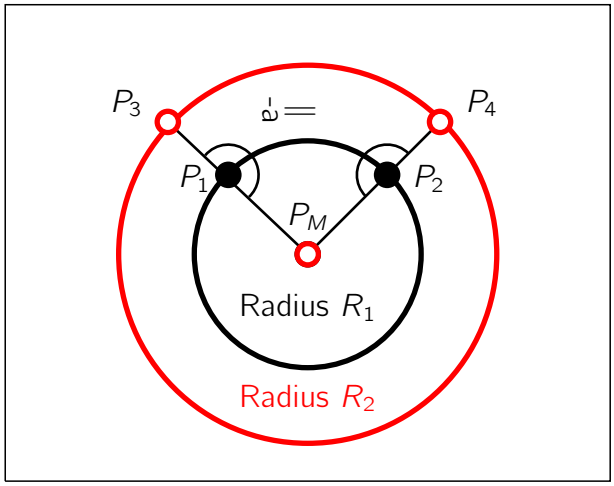
60 $y_9, x_9, R_1, y_{10}, x_{10}, R_2, y_{M_4}, x_{M4} = tangente14(y_1, x_1\ bis\ y_6, x_6, -R_4)$

61 $y_7, x_7, y_{M_1}, x_{M_1}, R_1, y_{M_3}, x_{M_3}, R_3 = tangente15(y_1, x_1\ bis\ y_5, x_5)$

62 $y_8, x_8, y_{M_1}, x_{M_1}, R_1, y_{M_4}, x_{M_4}, -R_4 = tangente16(y_1, x_1\ bis\ y_5, x_5)$

$\rightarrow R_3 =$ Umkreis, $\rightarrow R_4 =$ Ankreis

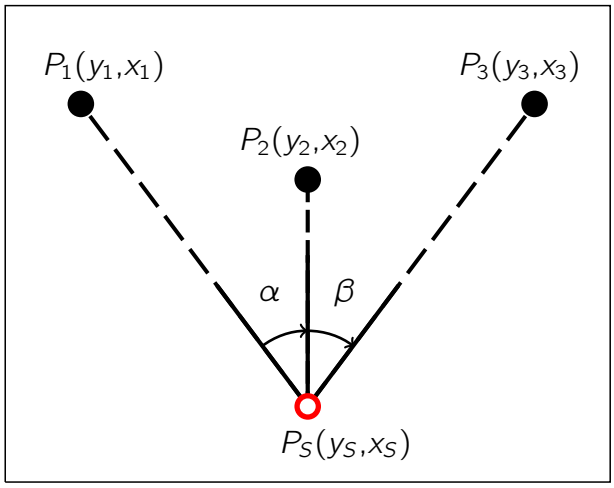
▶ Aufrufe:
```
y7, x7, R1, y8, x8, R2, yM3, xM3 = tangente13(y1, x1 bis y6, x6, R3)
y9, x9, R1, y10, x10, R2, yM4, xM4 = tangente14(y1, x1 bis y6, x6, -R4)
y7, x7, yM1, xM1, R1, yM3, xM3, R3 = tangente15(y1, x1 bis y5, x5)
y8, x8, yM1, xM1, R1, yM4, xM4, -R4 = tangente16(y1, x1 bis y5, x5)
```



63 **Paralleler Kreisbogen**

63 $y_3, x_3, y_4, x_4, y_M, x_M, R_2 = parallelkreis(y_1, x_1, y_2, x_2, R_1, a)$

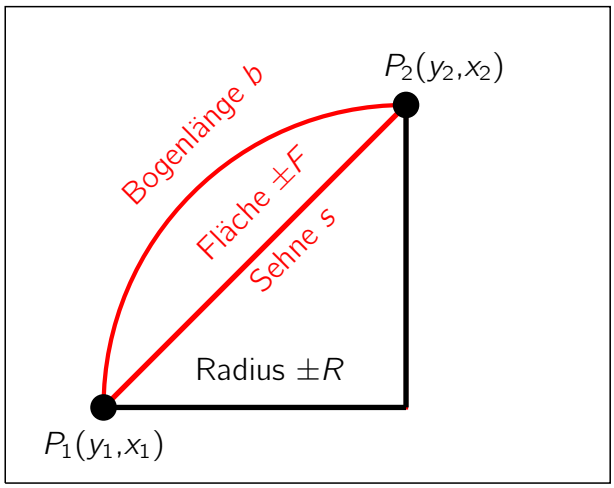
▶ Aufruf:
```
y3, x3, y4, x4, yM, xM, R2 = parallelkreis(y1, x1, y2, x2, R1, a)
```

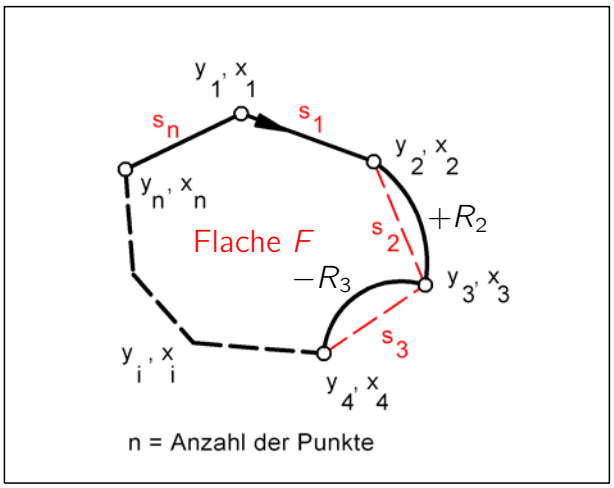


64 **Rückwärtsschnitt**

64 $y_S, x_S = rueckwaertsschnitt1(y_1, x_1, y_2, x_2, y_3, x_3, \alpha, \beta)$ (Collins, 1671)
$y_S, x_S = rueckwaertsschnitt2(y_1, x_1, y_2, x_2, y_3, x_3, \alpha, \beta)$ (Cassini, 1625-1712)

▶ Aufrufe: yS, xS = rueckwaertsschnitt1(y1, x1, y2, x2, y3, x3, alpha, beta)
yS, xS = rueckwaertsschnitt2(y1, x1, y2, x2, y3, x3, alpha, beta)



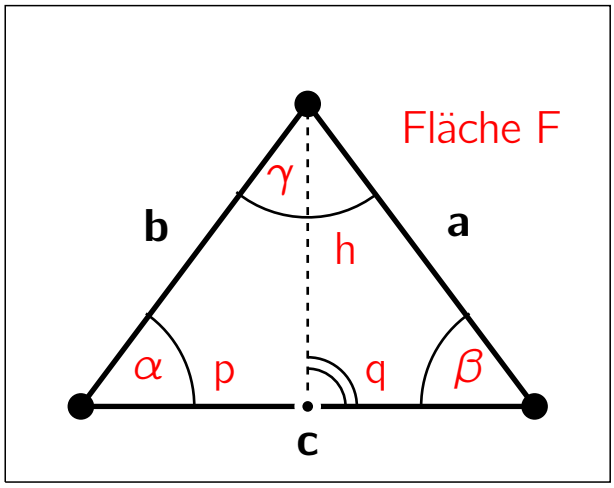


65 **66** **Segment, rechtsläufig** **67** **68** **Flächeninhalt, Seitenlängen**

65 $F, s, b = segment(y_1, x_1, y_2, x_2, R)$
66 $F, s, b = segment(s, R)$
67 $F, PktNr, y, x, R, s = flaeche(PktNr, y, x, R)$ ← [**Array**]
68 $F, PktNr, y, x, R, s = flaeche2(PktNr, y, x, R)$ ← [**Array**](Variante)

▶ Aufrufe: F, s, b = segment(y1, x1, y2, x2, R)
F, s, b = segment(s, R)
F, **koord8** = flaeche(**flaeche_polygon**) ← [**Array**]
F, **koord8** = flaeche2(**flaeche_polygon**) ← [**Array**](Variante)

Flächenberechnung, Gegeben: y_i, x_i rechtsläufige n Eckpunkte
$+Radius$ → Das Kreissegment wird addiert., $-Radius$ → Das Kreissegment wird subtrahiert.
Gesucht werden: $Fläche$ = F, s_i = Seiten- bzw. Sehnenlängen



69 **Dreieck, Höhe, Höhenfußpunkt**

69 $h, p, q, \alpha, \beta, \gamma, F = dreieck(a, b, c)$

▶ Aufruf: h, p, q, alpha, beta, gamma, F = dreieck(a, b, c)

Reduktion von Strecken bezüglich UTM:

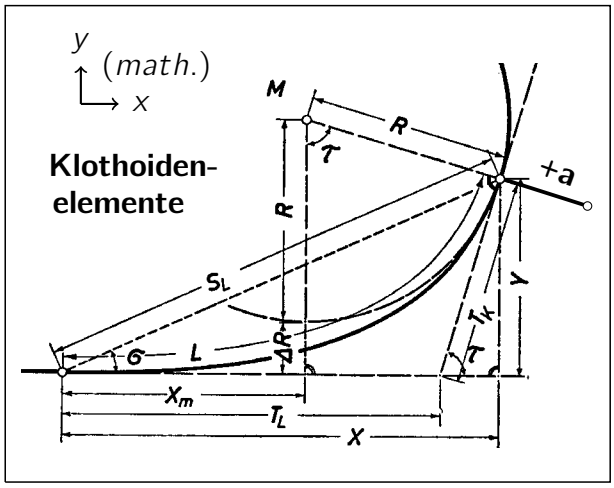
70 **Strecke: Reduktion von wahrer, örtlicher Strecke nach UTM**

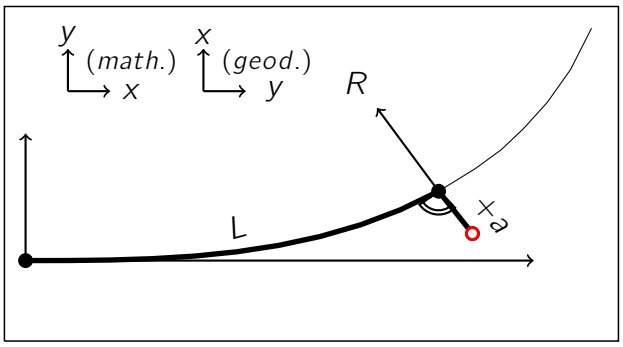
71 **Strecke: Reduktion von UTM (Koordinaten) in wahre, örtliche Strecke**

72 **Strecke: Reduktion von UTM (Streckenlänge) in wahre, örtliche Strecke**

▶ Aufrufe: streckeUTM = streckenreduktion(s, most, hoehe, undulation))
streckewahr = streckeUTM(y1, x1, y2, x2, most, hoehe, undulation)
streckewahr = streckeUTM(s, most, hoehe, undulation)

strecke, s	Strecke aus UTM-Koordinaten
streckewahr	Wahre Streckenlänge in der Örtlichkeit
most	Mittl. Rechtswert in [km]
hoehe	Höhe über NHN
undulation	Der Abstand des Geoids von dem Bezugsellipsoid im betrachteten Ellipsoidpunkt, gemessen entlang der Ellipsoidnormalen
y1, x1, y2, x2	Anfangs- und Endkoordinaten einer Strecke in UTM





73 **74** **Klothoidenelemente lokal**

73 $x, y, R, sigma_{gon}, tau_{gon}, xM_{lokal}, yM_{lokal}, delta_R, TL, TK, y2, x2, y3, x3, yM, xM$
$= klothoidenelemente(y1, x1, t1, A, L, \pm a)$

Für Elemente im örtlichen System setze man y1, x1 = 0, 0; t1 = 100.

Man beachte, das örtliche, lokale System der Klothoide ist mathematisch linksläufig, das des übergeordneten Systems vermessungstechnisch, also rechtsläufig.

74 $X, Y, R = klothoide(A, L)$

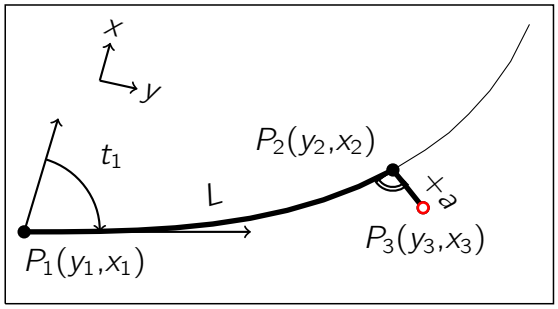
▶ Aufrufe:

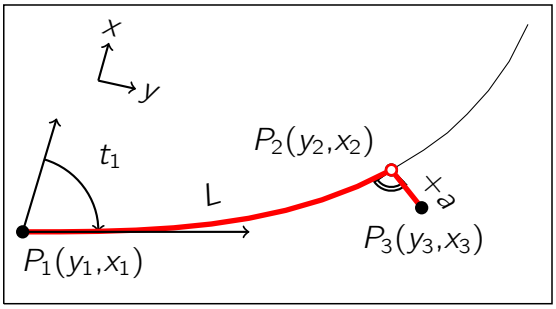
x, y, R, sigma, tau, xM_lokal, yM_lokal, deltaR, TL, TK, y2, x2, y3, x3, yM, xM
= klothoidenelemente(y1, x1, t1, A, L, ± a)
X, Y, R = klothoide(A, L) (vorrangig)
X, Y, R = klothoide2(A, L) (2. Variante)
X, Y, R = klothoide3(A, L) (3. Variante) (Tschebyschow-Approximation)

Erläuterungen:

y1	Rechtswert (übergeordnetes, geodätisches System), Anfangspunkt mit R = ∞
x1	Hochwert (übergeordnetes System, geod.)
t1	Richtungswinkel der Anfangstangente mit R = ∞ im übergeordneten System
A	Klothoidenparameter
L	Bogenlänge der Klothoide
a	Orthogonaler Abstand im Endpunkt
x	Endpunkt, Rechtswert im lokalen, mathematischen System
y	Endpunkt, Hochwert im lokalen, mathematischen System
sigma_gon	σ in gon
tau_gon	τ in gon, Richtungsänderung am Endpunkt

xM_lokal	Rechswert, Kreismittelpunkt des Endpunktes, lokal (math.)
yM_lokal	Hochwert, Kreismittelpunkt des Endpunktes, lokal (math.)
deltaR	Abrückung des Kreises von den Tangente
R	Radius am Endpunkt
TL	Tangentenlänge, tangential zum Anfangspunkt
TK	Tangentenlänge, tangential zum Endpunkt
y2	Rechtswert, Endpunkt auf der Klothoide (übergeordnetes System, geod.)
x2	Hochwert, Endpunkt auf der Klothoide (übergeordnetes System, geod.)
y3	Rechtswert, Seitwärts gelegender Punkt am Endpunkt (geod.)
x3	Hochwert, Seitwärts gelegender Punkt am Endpunkt (geod.)
yM	Rechtswert, Kreismittelpunkt am Endpunkt (geod.)
xM	Hochwert, Kreismittelpunkt am Endpunkt (geod.)

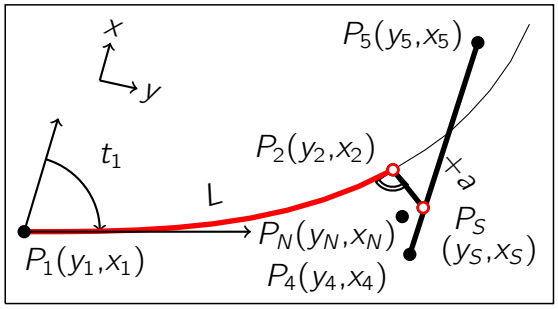




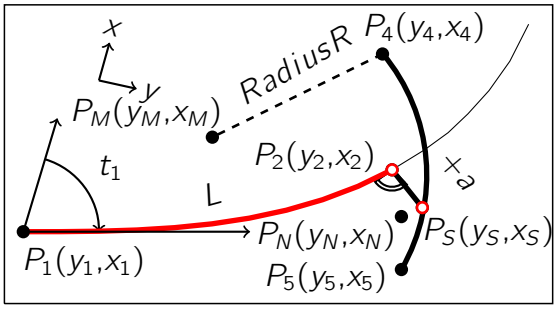
75 **Klothoide, seitwärtsliegender Punkt** **76** **Klothoide, Punktabstand**

75 $y_3, x_3, y_2, x_2 = klothoide4(y_1, x_1, t_1, A, L, \pm a)$
76 $y_2, x_2, L, a = klothoide5(y_1, x_1, t_1, A, y_3, x_3)$

▶ Aufrufe: y3, x3, y2, x3 = klothoide4(y1, x1, t1, A, L, ± a)
y2, x2, L, a = klothoide5(y1, x1, t1, A, y3, x3)



Schnitt Klothoide, seitwärts liegender Punkt mit Gerade



Klothoide, seitwärts liegender Punkt mit Kreis

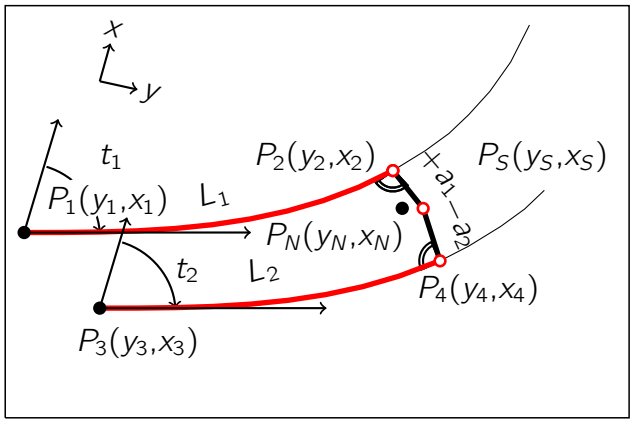
77 $y_S, x_S, y_2, x_2, L = klothoide_gerade(y_1, x_1, t_1, A, \pm a, y_4, x_4, y_5, x_5, y_N, x_N)$

78 $y_S, x_S, y_2, x_2, L = klothoide_kreis(y_1, x_1, t_1, A, \pm a, y_M, x_M, R, y_N, x_N)$

79 $y_S, x_S, y_2, x_2, L, y_M, x_M = klothoide_kreis(y_1, x_1, t_1, A, \pm a, y_4, x_4, y_5, x_5, R, y_N, x_N)$

$P_N(y_N,x_N)$ = Punkt mit Näherungskoordinaten

▶ Aufrufe:
yS, xS, y2, x2, L = klothoide_gerade(y1, x1, t1, A,± a, y4, x4, y5, x5, yN, xN)
yS, xS, y2, x2, L = klothoide_kreis(y1, x1, t1, A, ± a, yM, xM, R, yN, xN)
yS, xS, y2, x2, L, yM, xM = klothoide_kreis(y1, x1, t1, A, ± a, y4, x4, y5, x5, R, yN, xN)



80 **Schnitt Klothoide, seitwärts liegender Punkt mit Klothoide**

80 $y_S, x_S, y_2, x_2, L_1, y_4, x_4, L_2 =$
$klothoide_klothoide(y_1, x_1, t_1, A_1, \pm a_1, y_3, x_3, t_2, A_2, \pm a_2, y_N, x_N)$

$P_N(y_N,x_N)$ = Punkt mit Näherungskoordinaten

▶ Aufruf:
yS, xS, y2, x2, L1, y4, x4, L2 =
klothoide_klothoide(y1, x1, t1, A1, ± a1, y3, x3, t2, A2, ± a2, yN, xN)

Name und Aufbau der Eingabe- und Ausgabe-Arrays für diverse Programmmodule

Eingabeformat als Array für die Module 4, 5

4 Seitwärtsliegende Punkte auf Gerade ohne Fehlerverteilung
5 Seitwärtsliegende Punkte auf Gerade mit Fehlerverteilung

```
koordinaten = [
  1,    y1,    x1,   # Anfangspunkt (num. PktNr., y, x)
  2,    y2,    x2,   # Endpunkt (num. PktNr, y, x)
  3,   ay1,   ax1,   # lokale Daten
...,   ...,   ...,
  i,   ayi,    axi
...,   ...,   ...,
  n,   ayn,    axn
  ]
```

Beispiel als Array:

```
koordinaten = [
1,   100,   200,
2,   300,   400,
3,     1,     2,
4,   -10,    20,
5,    50,   -50
 ]
```

in **5** auch sgem

➡ **Aufrufe**: koord1 = ve.orthogonal4(koordinaten) bzw.
koord1, fs = ve.orthogonal5(koordinaten, sgem)

Ergebnis für die Funktion 4:

PktNr	y	x	ay	ax
1	100.000	200.000	0.000	0.000
2	300.000	400.000	0.000	282.843
3	102.121	200.707	1.000	2.000
4	107.071	221.213	−10.000	20.000
5	100.000	129.289	50.000	−50.000

in **5** auch fs

Ausgabeformat als Array koord1 hier für die Funktion 4

```
[[   1.      100.      200.        0.        0.     ]
[   2.      300.      400.        0.      282.8427]
[   3.      102.1213  200.7071    1.        2.     ]
[   4.      107.0711  221.2132  −10.       20.     ]
[   5.      100.      129.2893   50.      −50.     ]]
```

fs = -0.157 (Funktion 5)

Eingabeformat als Array für Modul 12

12 Polare Punkte mit Anschlusswinkel

```
polaraufnahme = [     # Pktn. (num), y, x
  1,       y1,    x1,  # Anfangspunkt
  2,       y2,    x2,  # Endpunkt
  3,   alpha1,   ax1,  # lokale Daten
...,       ...,   ...,
  i,    alphai,    axi
...,       ...,   ...,
  n,   alphan,    axn
  ],    alpha0
```

Beispiel als Array:

```
polaraufnahme = [
        1,        100,   200,
        2,        300,   400,
        3,   111.1111,    12,
        4,       144.,    20,
        5,        350,     50
], 40.023
```

➡ **Aufrufe**: koord2 = ve.polar3(polaraufnahme, alpha0)

```
Ergebnis:

  PktNr          y          x     alphai     alphai        axi
                                           (red.)
_______  ________  ________  _________  _________  _______
      1   100.000   200.000     0.0000     0.0000    0.000
      2   300.000   400.000    40.0230     0.0000  282.843
      3   111.354   196.117   111.0000    70.9770   12.000
      4   113.232   185.003   144.0000   103.9770   20.000
      5    70.596   240.440   350.0000   309.9770   50.000
alpha0 = 40.023
```

Ausgabeformat als Array koord2 für Modul 12

```
[[  1.      100.      200.        0.        0.        0.     ]
[   2.      300.      400.       40.023     0.      282.8427]
[   3.      110.7836  205.2644  111.1111   71.0881   12.     ]
[   4.      119.961   198.7514  144.      103.977    20.     ]
[   5.       50.6128  207.8039  350.      309.977    50.     ]]
```

Eingabeformat als Array für Modul 15

15 Lote, Lotfußpunkte, orthogonales Elemente über array

```
koordinaten = [    # Pktn. (num), y, x
  1,    y1,    x1,  # Anfangspunkt
  2,    y2,    x2,  # Endpunkt
...,   ...,   ...,
  i,    y1,    x1,  # Daten
...,   ...,   ...,
  n,    yn,    xn
  ]
```

Beispiel als Array:

```
koordinaten = [
1,     100,     100,
2,     900,     900,
3,     201,     203,
4,   300.2,   300.3,
5,   400.1,    400.3
 ]
```

➡ **Aufruf**: koord3 = ve.lot3(koordinaten)

Ergebnis:

PktNr	y	x	yF	xF	ay	ax
1	100.000	100.000	100.000	100.000	0.000	0.000
2	900.000	900.000	900.000	900.000	0.000	1131.371
3	201.000	203.000	202.000	202.000	−1.414	144.250
4	300.200	300.300	300.250	300.250	−0.071	283.196
5	400.100	400.300	400.200	400.200	−0.141	424.547

Ausgabeformat als Array koord3 für Modul 15

```
[[ 1.    100.     100.     100.     100.      0.        0.     ]
[ 2.    900.     900.     900.     900.      0.      1131.3708]
[ 3.    201.     203.     202.     202.     −1.4142   144.2498]
[ 4.    300.2    300.3    300.25   300.25   −0.0707   283.1963]
[ 5.    400.1    400.3    400.2    400.2    −0.1414   424.5469]]
```

Eingabeformat als Array für Modul 16

16 Teilung einer Strecke in (n+1) gleiche Teile

➡ **Aufruf**: koord4 = ve.lot3(y1, x1, y2, x2, n)

Beispieldaten:
y1, x1 = 10000, 10000 Anfangspunkt
y2, x2 = 10500, 10500 Endpunkt
n = 4 Anzahl der Zwischenpunkte

```
Ergebnis:

  Pktnr       y       x     ayi       axi
-------  -----  -----  -----  -------
      1  10000  10000      0    0.000
      2  10500  10500      0  707.107
      3  10200  10200      0  141.421
      4  10300  10300      0  282.843
      5  10400  10400      0  424.264
      6  10500  10500      0  565.685
```

Teilstrecke, Gesamtstrecke = 141.421 707.107

Ausgabeformat als Array koord4 für Modul 16

```
[[     1.      10000.      10000.          0.          0.     ]
[     2.      10500.      10500.          0.        707.1068]
[     3.      10200.      10200.          0.        141.4214]
[     4.      10300.      10300.          0.        282.8427]
[     5.      10400.      10400.          0.        424.2641]
[     6.      10500.      10500.          0.        565.6854]]
```

Eingabeformat als Array für Modul 18

18 Polar, Punkte mit Anschlußwinkel, Strecken

➡ **Aufruf**: koord5 = ve.polarwinkel2(koordinaten)

Beispieldaten als Array:
koordinaten = [
1, 100, 100,
2, 400, 400.486,
3, 251, 202,
4, 144, 254,
5, 350, 445]

```
Ergebnis:

  PktNr        y        x        t     alpha        s
-------  -------  -------  -------  --------  -------
      1  100.000  100.000   0.0000    0.0000    0.000
      2  400.000  400.486  49.9485    0.0000  424.608
      3  351.000  202.000  75.4271   25.4787  270.934
      4  144.000  254.000  17.7171  367.7686  160.162
      5  350.000  445.000  39.9206  389.9721  426.058
```

Ausgabeformat als Array koord5 für Modul 18

```
[[  1.     100.     100.       0.       0.       0.    ]
[  2.     400.     400.486  49.9485   0.     424.6079]
[  3.     351.     202.     75.4271  25.4787 270.9336]
[  4.     144.     254.     17.7171 367.7686 160.1624]
[  5.     350.     445.     39.9206 389.9721 426.0575]]
```

Eingabeformat als Array für Modul 21

21 Transformation

```
  1,   y1,   x1,   ay1,   ax1,   # Anfangspunkt
  2,   y2,   x2,   ay2,   ax2,   # Endpunkt
  3,    0,    0,   ay3,   ax3,   # umzuformende
  i,    0,    0,   ayi,   axi,   # Punkte
...     0,    0,   ...    ...
```

Beispiel als Array:

```
transkoordinaten = [
1001,   100.,   200.,    500.09,    600.05,
1002,   300.,   400.,     700.2,    800.00,
   3,      0,      0,       546,   675.345,
   4,      0,      0,       999,       767,
   5,      0,      0,   786.234,       678,
    ]
```

➡ **Aufruf**: koord6, fs = ve.transformation2(transkoordinaten)

```
Ergebnis:

  PktNr         y         x        ay        ax
-------  -------  -------  -------  -------
   1001  100.000  200.000  500.090  600.050
   1002  300.000  400.000  700.200  800.000
      3  145.873  275.302  546.000  675.345
      4  598.768  367.124  999.000  767.000
      5  386.070  278.053  786.234  678.000
```

fs = -0.042449030893749296

Ausgabeformat als Array koord6 für Modul 21

```
[[1001.      100.      200.      500.09    600.05   ]
[1002.      300.      400.      700.2     800.     ]
[    3.     145.873   275.3021  546.      675.345  ]
[    4.     598.7683  367.1244  999.      767.     ]
[    5.     386.0699  278.0527  786.234   678.     ]]
```

Eingabeformat als Array für Modul 25

25 Geradenausgleich

```
koordinaten = [     # Pktn. (num), y, x
  1,    y1,    x1,  # Anfangspunkt
  2,    y2,    x2,  # Endpunkt
  3,    y3,    x3,  # lokale Daten
...,   ...,   ...,
  i,     yi,     xi
...,   ...,   ...,
  n,    yn,    xn
   ]
```

Beispiel als Array:

```
koordinaten = [
1   100.    100.
2   700.    700.
3   102.     99.
4   102.    105.
5   101.    100.
6   101.    105.
7   698.   698.5
]
```

➡ **Aufruf**: koord7 = ve.geradenausgleich(koordinaten)

Ergebnis:

PktNr	y	x	Fy	Fx	ay	ax
1	100.000	100.000	99.699	100.301	0.425	0.000
2	700.000	700.000	699.876	700.124	0.176	848.528
3	102.000	99.000	100.200	100.801	2.546	0.708
4	102.000	105.000	103.200	103.799	−1.698	4.949
5	101.000	100.000	100.200	100.801	1.132	0.707
6	101.000	105.000	102.700	103.299	−2.405	4.242
7	698.000	698.500	698.125	698.375	−0.177	846.053

Ausgabeformat als Array koord7 für Modul 25

```
[[  1.    100.    100.    99.699 100.301   0.425   0.    ]
[  2.    700.    700.    699.876 700.124   0.176 848.528]
[  3.    102.     99.    100.2   100.801   2.546   0.708]
[  4.    102.    105.    103.2   103.799  −1.698   4.949]
[  5.    101.    100.    100.2   100.801   1.132   0.707]
[  6.    101.    105.    102.7   103.299  −2.405   4.242]
[  7.    698.    698.5   698.125 698.375  −0.177 846.053]]
```

Ausgabeformat als Array für Modul 37

37 Bogenteilung

➡ **Aufruf**:

yM, xM, s, b, elemente = ve.bogenteilung(y1, x1, y2, x2, +R, n)

Beispieldaten:
y1, x1 = 10000, 10000
y2, x2 = 10500, 10500
R = 500, n = 4

```
Ergebnis:

        y1          x1        ayi        axi
__________  __________  _______  _______
10000.000   10000.000     0.000    0.000
10154.508   10024.472    91.950  126.558
10293.893   10095.492   140.291  275.336
10404.508   10206.107   140.291  431.771
10475.528   10345.492    91.950  580.549
10500.000   10500.000     0.000  707.107
```

Ausgabeformat als Array elemente für Modul 37

```
[[10000.      10000.          0.          0.     ]
[10154.5085 10024.4717    91.9499   126.5581]
[10293.8926 10095.4915   140.2908   275.3362]
[10404.5085 10206.1074   140.2908   431.7706]
[10475.5283 10345.4915    91.9499   580.5486]
[10500.      10500.          0.        707.1068]]
```

Ein- und Ausgabeformat als Array für die Module 67. 68

67, 68 Flächeninhalt, Seitenlängen

Beispieldaten als Array:

```
flaeche_polygon =  [
1,    0,    0,    5,
2,    0,    3,   -6,
3,    4,    3,    0,
4,    2,    0,    0
]
```

➡ **Aufruf**: F, **koord** = ve.flaeche(**flaeche_polygon**) ← [**Array**]

Ergebnis:

PktNr	y	x	Radien	Strecken
1	0.000	0.000	5.000	3.000
2	0.000	3.000	−6.000	4.000
3	4.000	3.000	0.000	3.606
4	2.000	0.000	0.000	2.000

Fläche = 8.54

➡ **Aufruf**: F, **koord** = ve.flaeche2(**flaeche_polygon**) ← [**Array**](2. Variante)

Ergebnis:

PktNr	y	x	Radien	Strecken
1	0.000	0.000	5.000	3.000
2	0.000	3.000	−6.000	4.000
3	4.000	3.000	0.000	3.606
4	2.000	0.000	0.000	2.000

Fläche2 = 8.54

Funktionen von pyvermessung.py

Nr.	Aufruf der Funktion:
intern (0)	t_gon, s = RichtEntf(y1, x1, y2, x2, fehlernummer)
1 ▶	yi, xi, sger = orthogonal(y1, x1, y2, x2, ayi)
2 ▶	yi, xi, sger = orthogonal(y1, x1, y2, x2, ayi, axi)
3 ▶	yi, xi, sger = orthogonal(y1, x1, y2, x2, ayi, axi, sgem)
4 ▶	**koord1** = orthogonal4(**koordinaten**) ← [**Array**]
5 ▶	**koord1**, fs = orthogonal5(**koordinaten**, sgem) ← [**Array**]
6 ▶	t1, s1 = richtung(y1, x1, y2, x2)
intern (6)	t1, s1 = richtung1(y1, x1, y2, x2)
7 ▶	t1, s1 = richtung(dy, dx)
intern (7)	t1, s1 = richtung2(dy, dx)
8 ▶	sint, cost, sger, massstab = richtung3(y1, x1, y2, x2, sgem)
9 ▶	yi, xi = polar(y1, x1, t1, axi)
10 ▶	yi, xi, yF, xF = polar(y1, x1, t1, ayi, axi)
11 ▶	yi, xi, yF, xF = polar(y1, x1, t1, alphai, ayi, axi)
12 ▶	**koord2** = polar4(**polaraufnahme**, alpha0)
13 ▶	yF, xF, ayi, axi, yisp, xisp = lot(y1, x1, t1, xi, yi)
intern (13)	yF, xF, ayi, axi = lot1(y1, x1, t1, xi, yi)
intern (13)	ayi = lot2(y1, x1, t1, xi, yi)
14 ▶	yF, xF, ayi, axi, yisp, xisp = lot(y1, x1, y2, x2, yi, xi)
15 ▶	**koord3** = lot3(**koordinaten**) ← [**Array**]
16 ▶	**koord4** = teilstrecken(y1, x1, y2, x2, n) ← [**Array**]
17 ▶	t2, s2, alpha, s3 = polarwinkel1(y1, x1, y2, x2, y3, x3)
18 ▶	**koord5** = polarwinkel2(**koordinaten**) ← [**Array**]
19 ▶	yM, xM, tM, sM = mittelsenkrechte(y1, x1, y2, x2)
20 ▶	yi, xi, fs = transformation1(y1, x1, y2, x2, ay1, ax1, ay2, ax2, ayi, axi)
21 ▶	**koord6**, fs = transformation2(**transkoordinaten**) ← [**Array**]
22 ▶	yS, xS, alpha = geradenschnitt(y1, x1, y2, x2, y3, x3, y4, x4)
intern (22)	yS, xS = geradenschnitt1(y1, x1, y2, x2, y3, x3, y4, x4)
23 ▶	yS, xS, alpha = geradenschnitt(y1, x1, t1, y3, x3, t3) (Vorwärtsschnitt)
intern (23)	yS, xS, alpha = geradenschnitt2(y1, x1, t1, y3, x3, t3)
24 ▶	yS, xS, alpha = geradenschnitt(y1, x1, y2, x2, y3, x3, t3)
25 ▶	**koord7** = geradenausgleich(**koordinaten**) ← [**Array**]
26 ▶	y11, x11, y21, x21, sger = parallel(y1, x1, y2, x2, ± a)
27 ▶	yS, xS, alpha = parallelschnitt(y1, x1, y2, x2, ± a1, y3, x3, y4, x4, ± a3)
28 ▶	yS, xS = senkrechtschnitt(y1, x1, y2, x2, yi, xi)
29 ▶	yS, xS = senkrecht_parallel(y1, x1, y2, x2, y3, x3, y4, x4, yi, xi, ± a)
30 ▶	yS, xS = doppelt_senkrecht(y1, x1, y2, x2, yi, xi, y3, x3, y4, x4, yj, xj)

Funktionen von pyvermessung.py

Nr.	Aufruf der Funktion:
31 ►	yS, xS, tS, alpha = winkelhalbierende(y1, x1, y2, x2, y3, x3, y4, x4)
32 ►	yM, xM = kreis(y1, x1, y2, x2, ± R)
intern (32)	yM, xM = kreismitte(y1, x1, y2, x2, ± R)
33 ►	yM, xM, ± R = kreis(y1, x1, y2, x2, y3, x3)
intern (33)	yM, xM, R = kreis3punkte(y1, x1, y2, x2, y3, x3)
34 ►	yF, xF, d = lot_kreis(y1, x1, y2, x2, y3, x3, ± R)
35 ►	yF, xF, d, yM, xM, ± R = lot_kreis(y1, x1, y2, x2, y3, x3, y4, x4)
36 ►	yM, xM, yT, xT, t, s, b, alpha = kreisbogen(y1, x1, y2, x2, +R) (linksläufig)
intern	hi, ji, se = ortho(dxe, dxe, dyi, dxi) (einfache Projektion)
37 ►	yM, xM, s, b, **elemente** = bogenteilung(y1, x1, y2, x2, +R, n) ← [**Array**]
38 ►	yS, xS, yF, xF, s1, s2, a = kreis_kreis(y1, x1, +R1, y2, x2, +R2)
39 ►	yS, xS = kreis_gerade(y1, x1, t1, yM, xM, +R)
40 ►	yS, xS = kreis_gerade(y1, x1, y2, x2, yM, xM, +R)
41 ►	yS, xS = kreis_gerade(y1, x1, y2, x2, yM, xM, +R, ± a)
42 ►	yS, xS, yM, xM = kreis_gerade(y1, x1, y2, x2, y3, x3, y4, x4, +R)
43 ►	yS, xS, yM, xM = kreis_gerade(y1, x1, y2, x2, y3, x3, y4, x4, +R, ± a)
44 ►	yS, xS, yM, xM, R = kreis_gerade(y1, x1, y2, x2, y3, x3, y4, x4, y5, x5, ± a)
45 ►	yS, xS = senkrecht_kreis(y1, x1, y2, x2, yM, xM, +R)
46 ►	yT, xT, t = tangente1(y1, x1, yM, xM, ±R)
47 ►	yT, xT, t, yM, xM = tangente1(y1, x1, y3, x3, y4, x4, ±R)
48 ►	yT, xT, t, yM, xM, ± R = tangente2 (y1, x1, y2, x2, y3, x3, y4, x4)
49 ►	yT, xT, t, yM, xM, ± R = tangente3(y1, x1, y3, x3, y4, x4, y5, x5)
50 ►	yT, xT, t, yM, xM = tangente4 (y1, x1, y2, x2, y3, x3, ±R)
51 ►	y4, x4, yM, xM, R, R1 = tangente5(y1, x1, y2, x2, y3, x3, y5, x5) (rechts)
51 ►	y4, x4, yM, xM, R, R1 = tangente5(y1, x1, y2, x2, y3, x3, y5, x5, -1) (links)
52 ►	yT1, xT1, yT2, xT2, yT3, xT3, yM, xM, t1, t2, R, R1 = tangente6(y1, x1 bis y6, x6)
53 ►	yT1, xT1, yT2, xT2, yM, xM, t1, t2, R, R1 = tangente7(y1, x1, y2, x2, y3, x3, y4, x4, y5, x5)
54 ►	yT1, xT1, yT2, xT2, yM, xM, t1, t2 = tangente8 (y1, x1, y2, x2, y3, x3, y4, x4, ± R)
55 ►	y1, x1, y2, x2, t1 = tangente9(yM1, xM1, +R1, yM2, xM2, +R2)
55 ►	y1, x1, y2, x2, t1 = tangente9(yM1, xM1, -R1, yM2, xM2, -R2)
56 ►	y3, x3, y4, x4, t3 = tangente10(yM1, xM1, +R1, yM2, xM2, -R2)
56 ►	y3, x3, y4, x4, t3 = tangente10(yM1, xM1, -R1, yM2, xM2, +R2)
57 ►	y1, x1, y2, x2, yM3, xM3 = tangente11(yM1, xM1, R1, yM2, xM2, R2, +R3)
57 ►	y1, x1, y2, x2, yM3, xM3 = tangente11(yM1, xM1, R1, yM2, xM2, R2, -R3)
58 ►	y1, x1, y2, x2, yM3, xM3, ±R = tangente12(yM1, xM1, R1, yM2, xM2, R2, y3, x3)
59 ►	y7, x7, R1, y8, x8, R2, yM3, xM3 = tangente13(y1, x1 bis y6, x6, R3)
60 ►	y9, x9, R1, y10, x10, R2, yM4, xM4 = tangente14(y1, x1 bis y6, x6, -R4)

Funktionen von pyvermessung.py

Nr.	Aufruf der Funktion:
61 ▶	y7, x7, yM1, xM1, R1, yM3, xM3, R3 = tangente15(y1, x1 bis y5, x5)
62 ▶	y8, x8, yM1, xM1, R1, yM4, xM4, -R4 = tangente16(y1, x1 bis y5, x5)
63 ▶	y3, x3, y4, x4, yM, xM, R2 = parallelkreis(y1, x1, y2, x2, R1, a)
64 ▶	yS, xS = rueckwaertsschnitt1(y1, x1, y2, x2, y3, x3, alpha, beta) (Collins)
64 ▶	yS, xS = rueckwaertsschnitt2(y1, x1, y2, x2, y3, x3, alpha, beta) (Cassini)
65 ▶	F, s, b = segment(y1, x1, y2, x2, R)
66 ▶	F, s, b = segment(s, R)
67 ▶	F, **koord8** = flaeche(**flaeche_polygon**) ← [**Array**]
68 ▶	F, **koord8** = flaeche2(**flaeche_polygon**) ← [**Array**] (Variante)
69 ▶	h, p, q, alpha, beta, gamma, F = dreieck(a, b, c)
70 ▶	streckeUTM = streckenreduktion(s, most, hoehe, undulation))
71 ▶	streckewahr = streckeUTM(y1, x1, y2, x2, most, hoehe, undulation)
72 ▶	streckewahr = streckeUTM(s, most, hoehe, undulation)
73 ▶	x, y, R, sigma, tau, xM_lokal, yM_lokal, deltaR, TL, TK, y2, x2, y3, x3, yM, xM = klothoidenelemente(y1, x1, t1, A, L, ± a)
74 ▶	X, Y, R = klothoide(A, L) (vorrangig)
intern (74)	X, Y, R = klothoide2(A, L) (2. Variante)
intern (74)	X, Y, R = klothoide3(A, L) (3. Variante) (Tschebyschow-Approximation)
75 ▶	y3, x3, y2, x2 = klothoide4(y1, x1, t1, A, L, ± a)
76 ▶	y2, x2, L, a = klothoide5(y1, x1, t1, A, y3, x3)
77 ▶	yS, xS, y2, x2, L = klothoide_gerade(y1, x1, t1, A, ± a, y4, x4, y5, x5, yN, xN)
78 ▶	yS, xS, y2, x2, L = klothoide_kreis(y1, x1, t1, A, ± a, yM, xM, R, yN, xN)
79 ▶	yS, xS, y2, x2, L, yM, xM = klothoide_kreis(y1, x1, t1, A, ± a, y4, x4, y5, x5, R, yN, xN)
80 ▶	yS, xS, y2, x2, L1, y4, x4, L2 = klothoide_klothoide(y1, x1, t1, A1, ± a1, y3, x3, t2, A2, ± a2, yN, xN)
intern (81)	yT1, xT1, yM, xM, R, t1 = regula_falsi_kreis_gerade(y1, x1, t1, yW, xW, tM, y5, x5, yS, xS)
intern (82)	x = regula_falsi_nullstelle(Funktion, LinkeS, RechteS) (mit Lambda-Funktion)

© 2024, Norbert Fuhrmann, 50169 Kerpen (Gesetzt in LaTeX, TikZ)
www.grenzuntersuchung.de