Autor: 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 <Vermessung-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 3.12.4> für Windows kann im Internet unter https://www.python.org/downloads/ heruntergeladen werden[2]. Grundlage der hier vorliegenden Programme ist die <Release version 3.12.4 Jun 6 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). 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.

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 tabulate import tabulate | import numpy as np |
| from sympy import symbols, Eq, sqrt, solve | 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. Durch die solve-Funktion in *numpy* werden diese nicht befriedigend gelöst. Es folgen hier dann iterative Methoden. Die Anzahl der Iterationen wurden auf i.d.R. n = 200 beschränkt. Weitergehende Iterationen dürften zu keiner Lösung führen. Eingangswerte oder Näherungskoordinaten wären dann zu überprüfen.

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 <Vermessung-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.

Die Programme wurden mit <autopep8> formatiert, das auch in <spyder> enthalten ist.

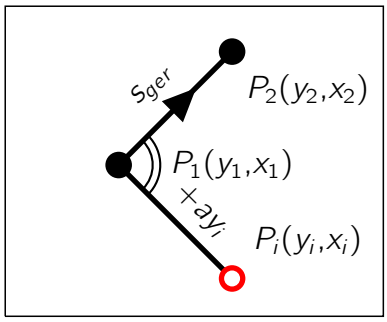
---

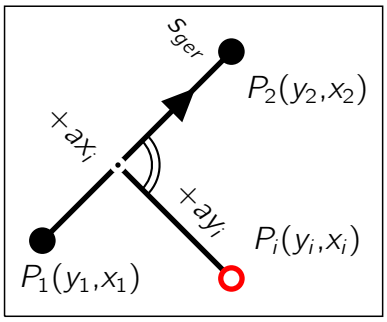
Installation für Android:
Für die Anwendung auf einem Smartphone bzw. Tablet (mit Android) muss im Google <Play store> unter dem Stichwort <python 3 for android> die App <Pydroid 3 - IDE for Python 3>[3] heruntergeladen werden.

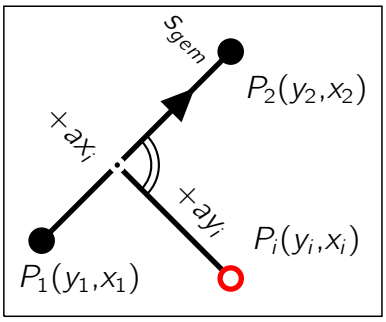
Für Android muss von GooglePlay <*Pydroid repository plugin*> geladen werden, um die Ergänzungspakete **numpy, sympy** und **tabulate** installieren zu können:
Symbol ≡ (oben links) drücken, unter >Pip< in >Libary name< numpy, sympy bzw. tabulate eingeben und >INSTALL< drücken.

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

[3] ⟹ <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.)



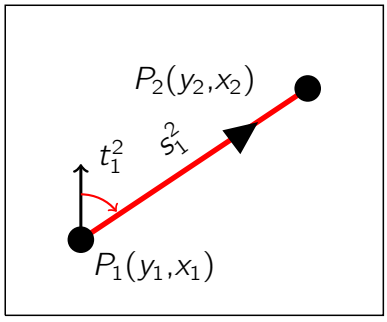

**1** **Orthogon. Punkt 1**

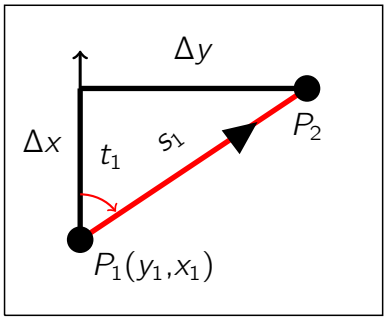
**2** **Orthogon. Punkt 2**
**4** **Orthogon. Punkte 4 ohne Fehlerverteilung**

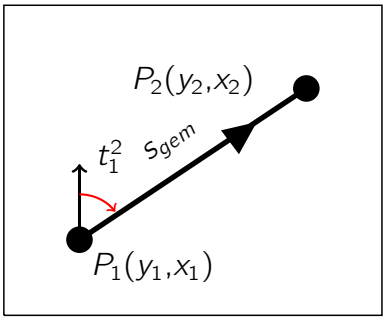
**3** **Orthogon. Punkt 3**
**5** **Orthogon. Punkte 5 mit Fehlerverteilung**

**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**]



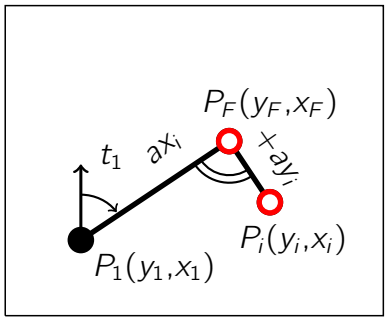

**6** **Richtungswinkel und Entfernung**

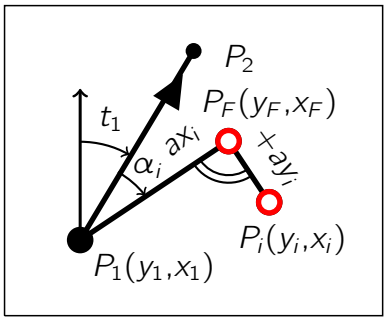
**7** **Richtungswinkel aus Koordinatendifferenzen**

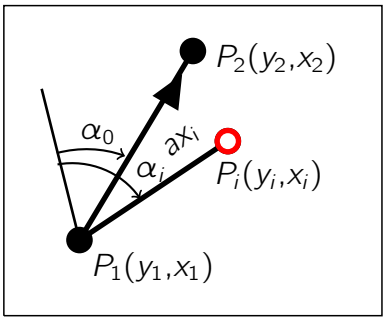
**8** **Richtungsfunktionen, Maßstab**

**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)



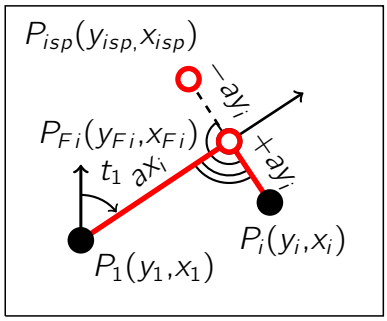



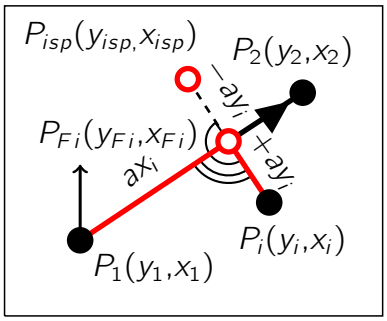



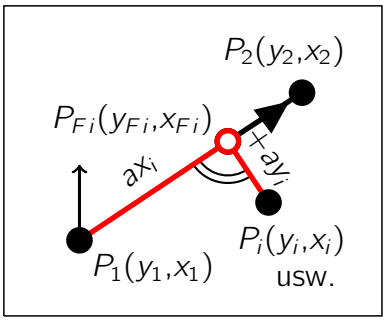

**9** **10** **Polarer Punkt 1, 2** **11** **Polarer Punkt 3** **12** **Polare Punkte 4**

**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**]



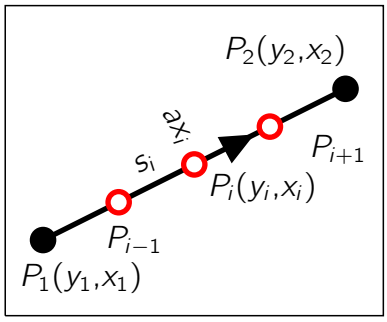



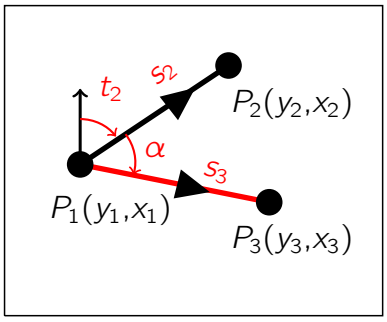



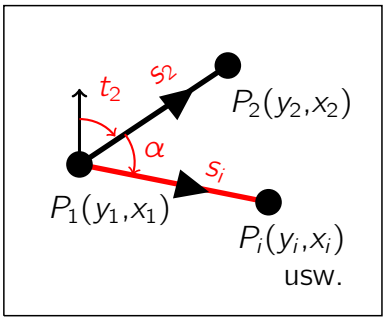

**13** **Abstand Richtung Spiegelpunkt** **14** **Lot, Lotfußpunkt Orthog. Element, Spiegelpunkt** **15** **Lote, Lotfußpunkte Orthog. Elemente**

**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**]



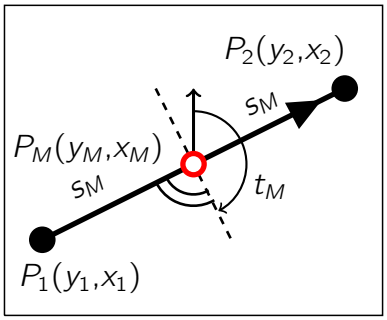



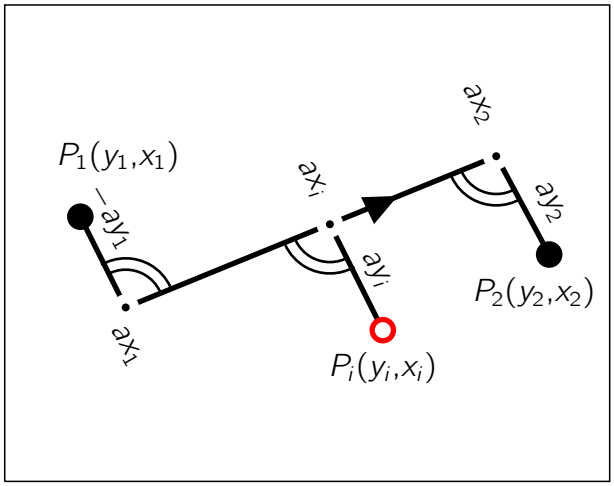




**16** **Strecke teilen** **17** **Polarwinkel, Strecke** **18** **Polarwinkel, Strecken**

**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**]



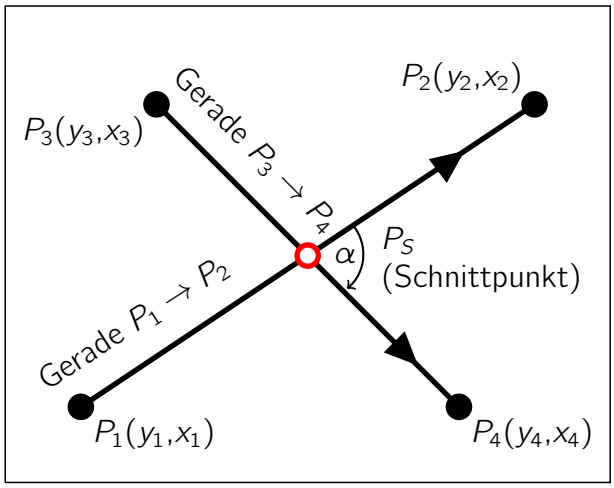



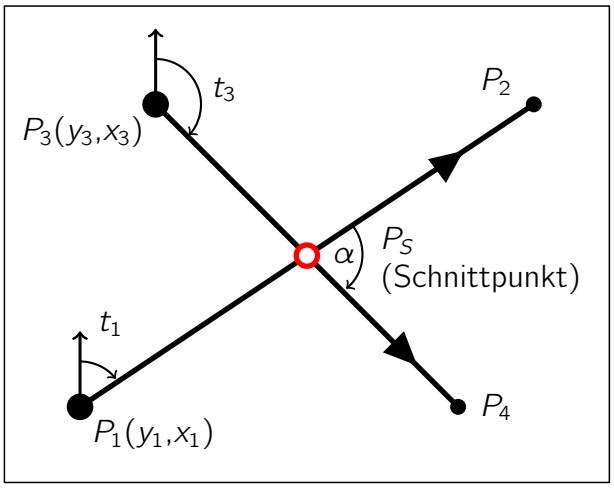

**19** **Strecke halbieren, Mittelsenkrechte** **20** **21** **Transformation**

**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:**



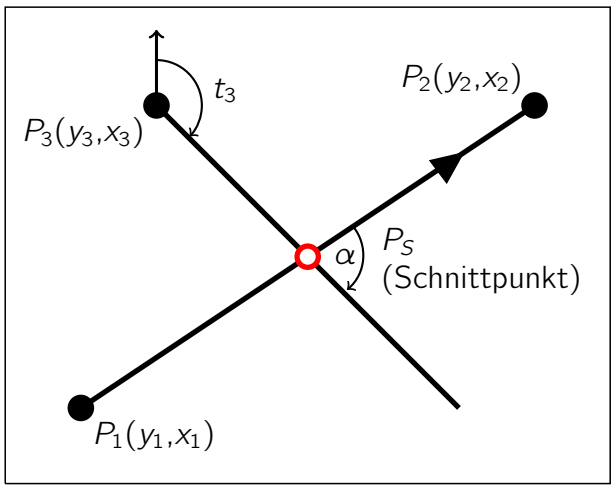



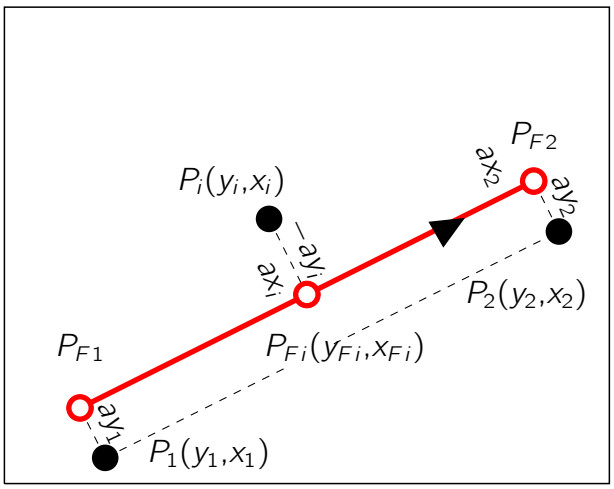

**22** **Einfacher Geradenschnitt**

**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)



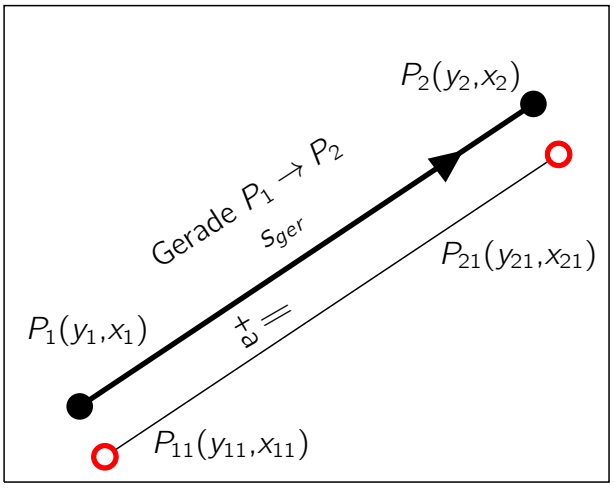



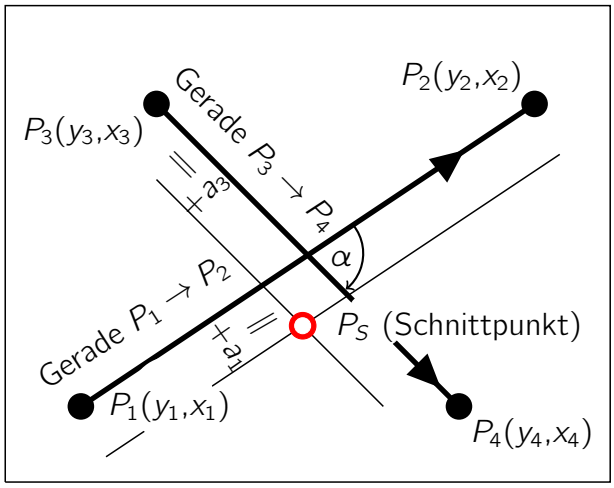

**24** **Geradenschnitt über einen Richtungswinkel**

**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**]

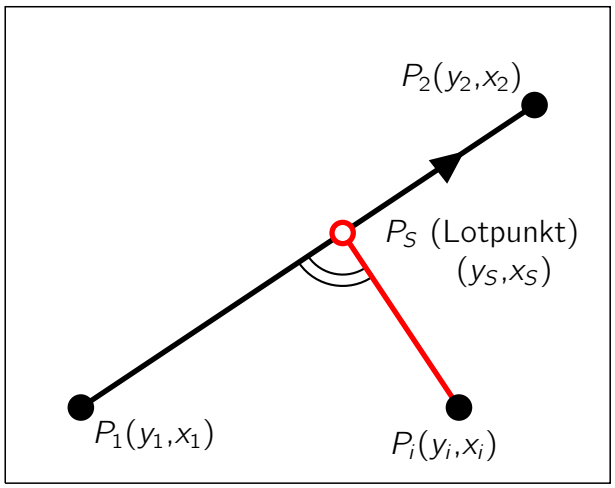


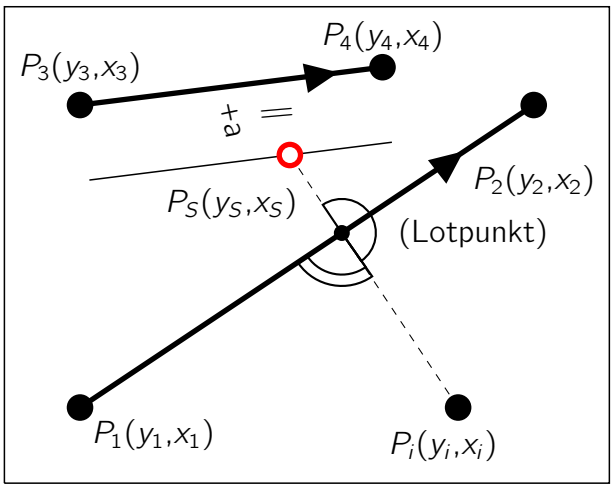




**26** **Parallele** **27** **Parallelschnitt**

**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)

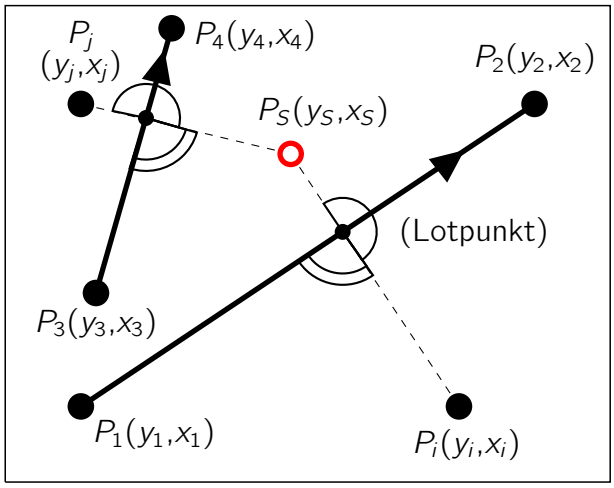


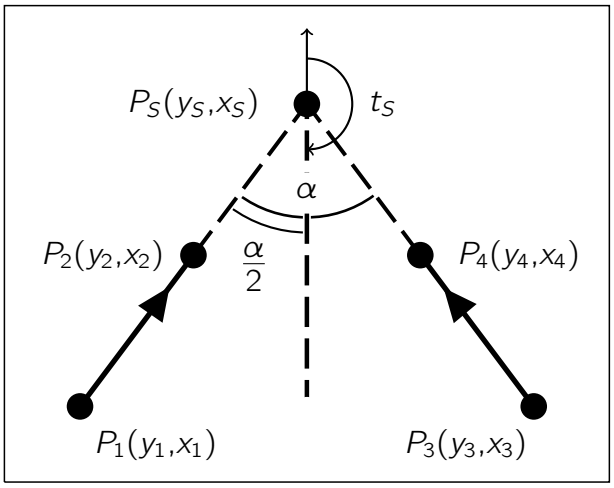




**28** **Senkrechtschnitt** **29** **Senkrecht-Parallelschnitt**

**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)



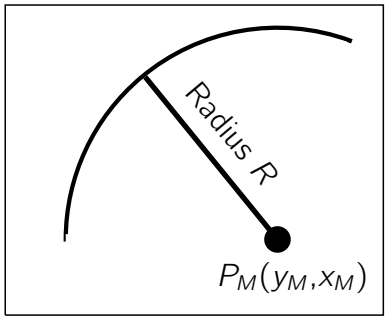



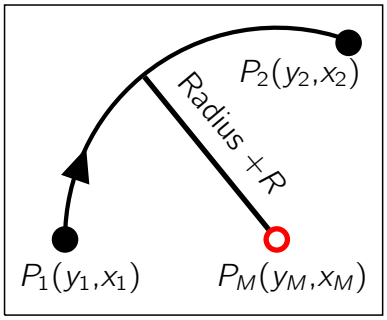

**30** **Doppelter Senkrechtschnitt** **31** **Winkelhalbierende**

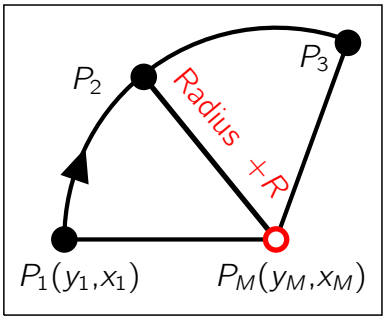
**30** $y_S, x_S = doppelt\_senkrecht(y_1, x_1, y_2, x_2, y_i, x_i, y_3, x_3, y_4, x_4, y_j, x_j)$
**31** $y_S, x_S, t_S, \alpha = winkelhalbierende(y_1, x_1, y_2, x_2, y_3, x_3, y_4, x_4)$

▶ 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:**



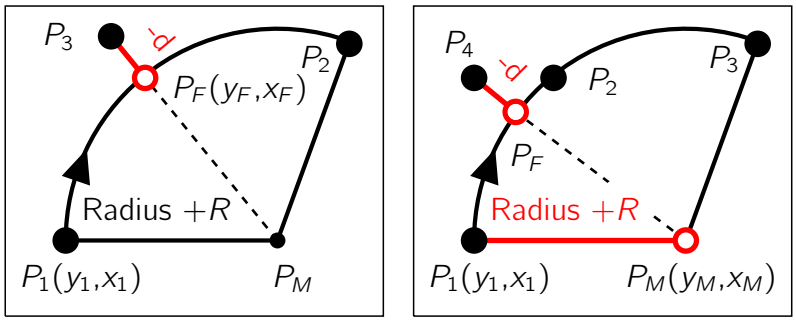




**Kreis 1** **32** **Kreis 2** **33** **Kreis 3**

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

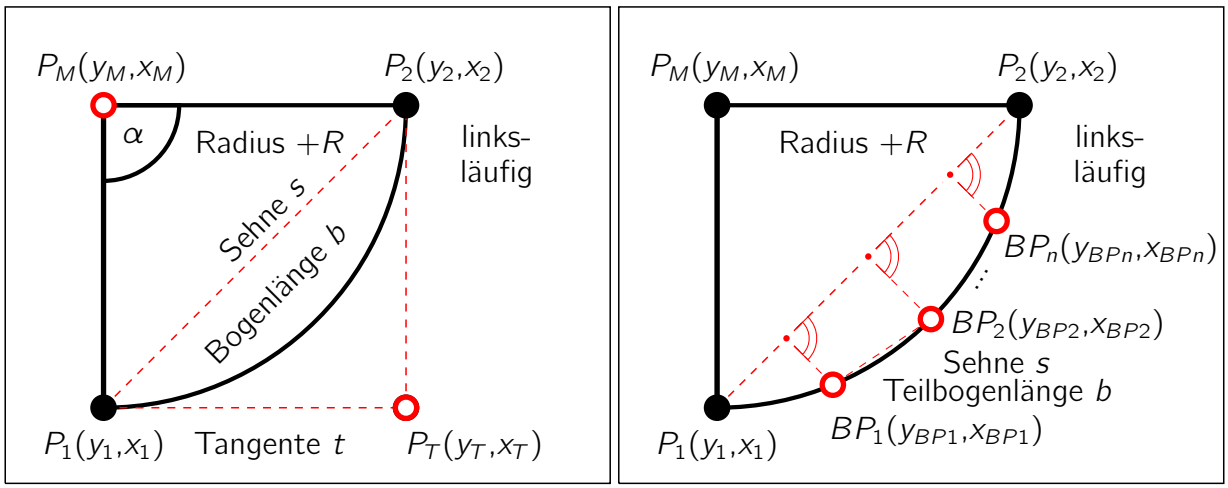
▶ Aufrufe: yM, xM = kreis(y1, x1, y2, x2, ± R)
yM, xM, ± R = kreis(y1, x1, y2, x2, y3, x3)




**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.

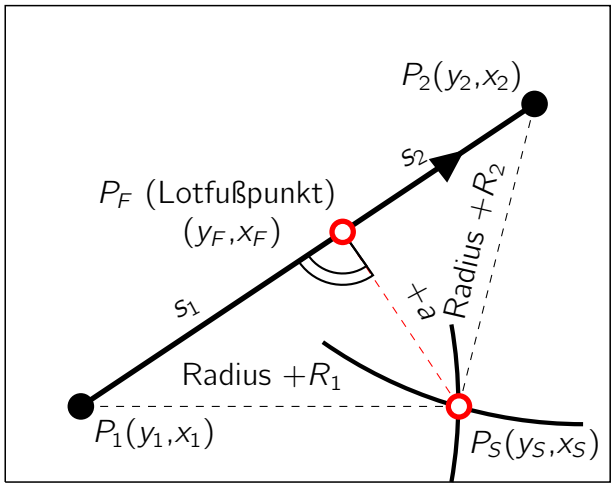
▶ Aufrufe: yF, xF, d = lot_kreis(y1, x1, y2, x2, y3, x3, ± R)
yF, xF, d, yM, xM, ± R = lot_kreis(y1, x1, y2, x2, y3, x3, y4, x4)

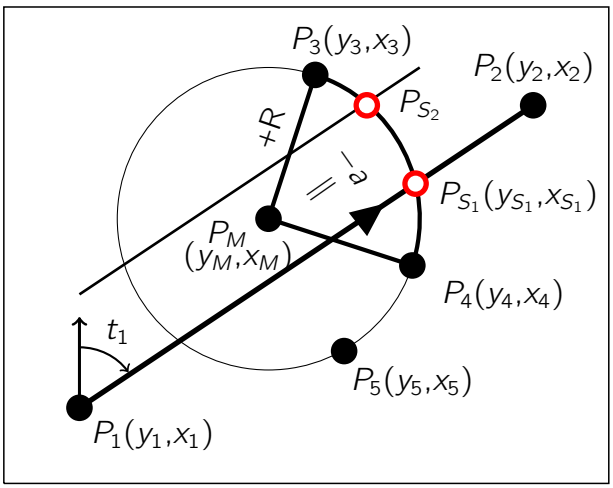



**36** **Kreisbogen** **37** **Bogenteilung**

**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**]



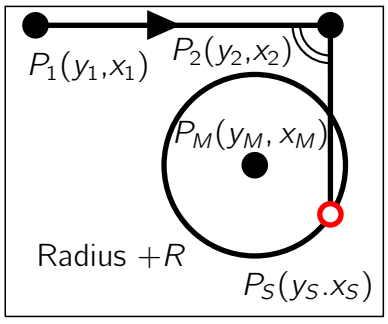




**38** **Kreisbogenschnitt**

**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** $y_S, x_S, y_M, x_M, R = kreis\_gerade(y_1, x_1, y_2, x_2, y_3, x_3, y_4, x_4, y_5, x_5, \pm a)$

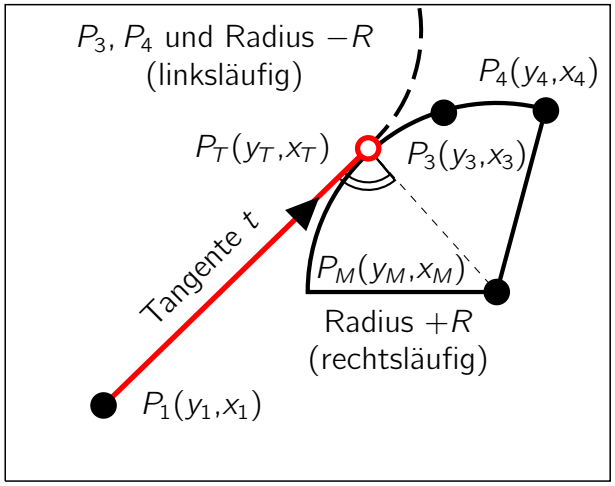
▶ Aufrufe: yS, xS, yF, xF, s1, s2, a = kreis_kreis(y1, x1, R1, y2, x2, +R2)
yS, xS = kreis_gerade(y1, x1, t1, yM, xM, +R)
yS, xS = kreis_gerade(y1, x1, y2, x2, yM, xM, +R)
yS, xS = kreis_gerade(y1, x1, y2, x2, yM, xM, +R, ± a)
yS, xS, yM, xM = kreis_gerade(y1, x1, y2, x2, y3, x3, y4, x4, +R)
yS, xS,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

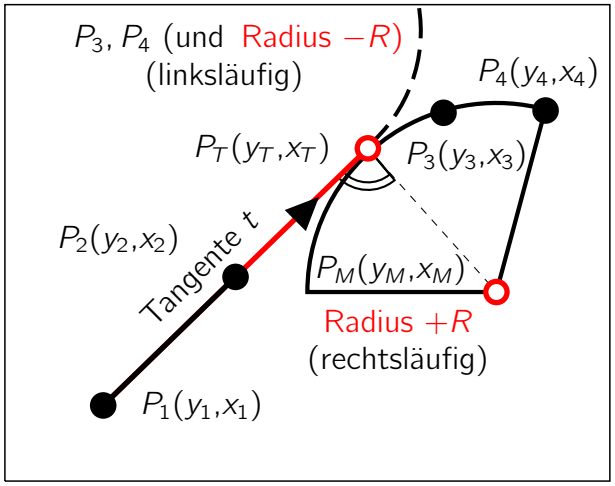



**45** **Senkrechtschnitt Kreis**

**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**

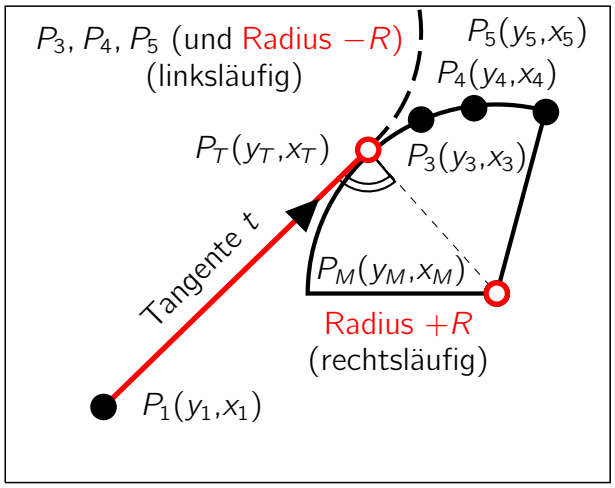
**46** $y_T, x_T, t = tangente1(y_1, x_1, y_M, x_M, \pm R)$

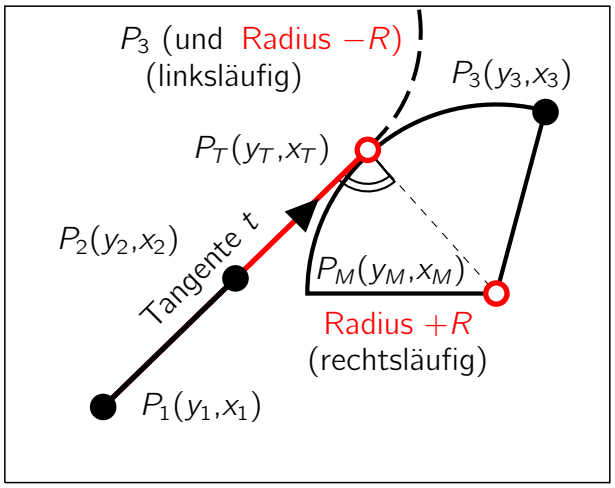
**47** $y_T, x_T, t, y_M, x_M = tangente1(y_1, x_1, y_3, x_3, y_4, x_4, \pm R)$

**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



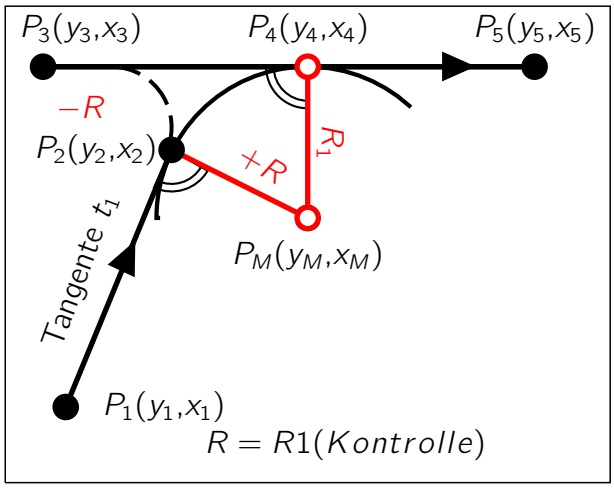

**49** **Kreis, bewegliche Tangente und 3 Kreispunkte**

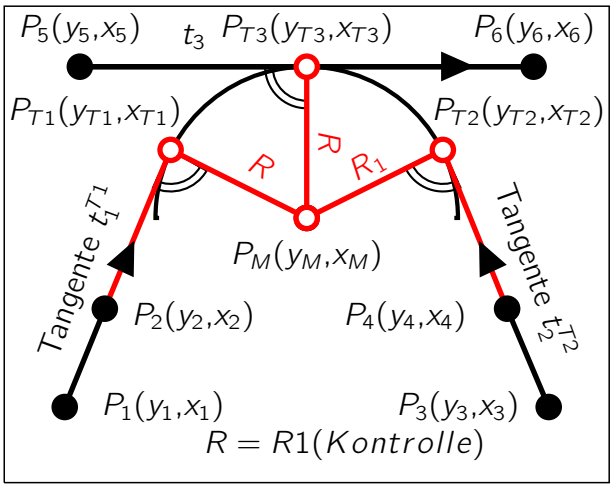
**50** **Kreis, feste, tangentiale Gerade mit Kreispunkt und Radius**

**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)



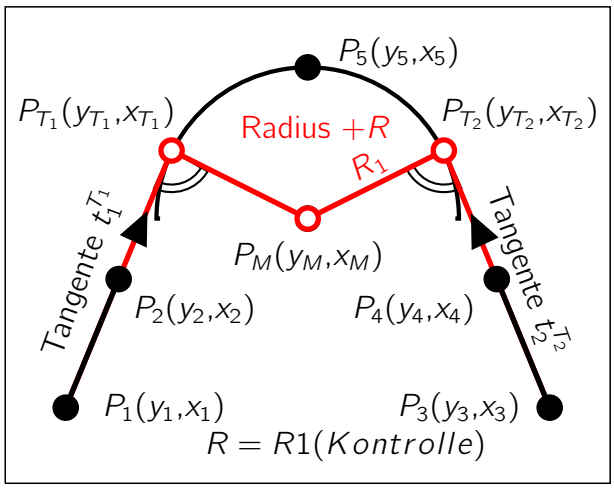

**51** **Kreis, Tangente mit fester Länge und feste, tangentiale Gerade**

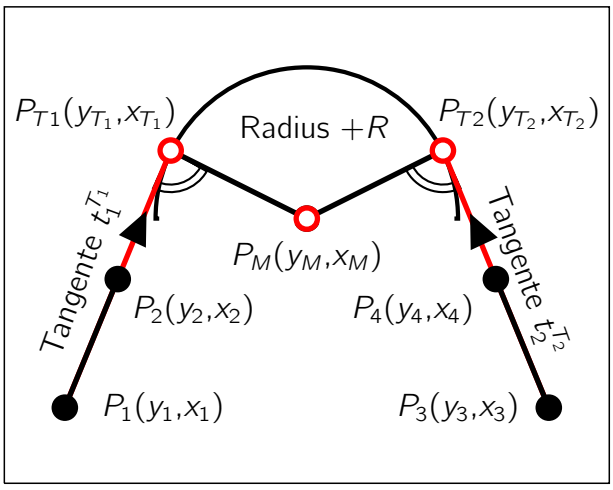
**52** **Kreis, 3 feste, tangentiale Geraden**

**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)



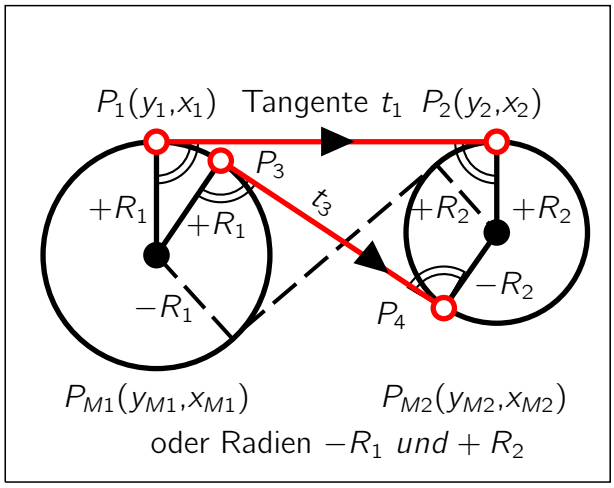

**53** **Kreis, 2 feste, tangentiale Geraden und 1 Kreispunkt**

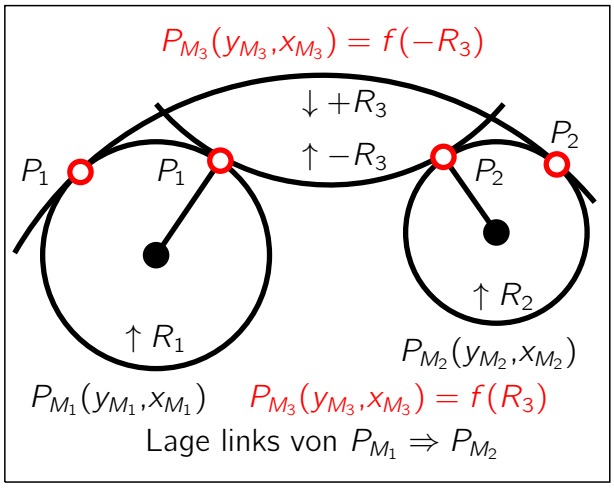
**54** **Kreis, 2 feste, tangentiale Geraden und Radius**

**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)



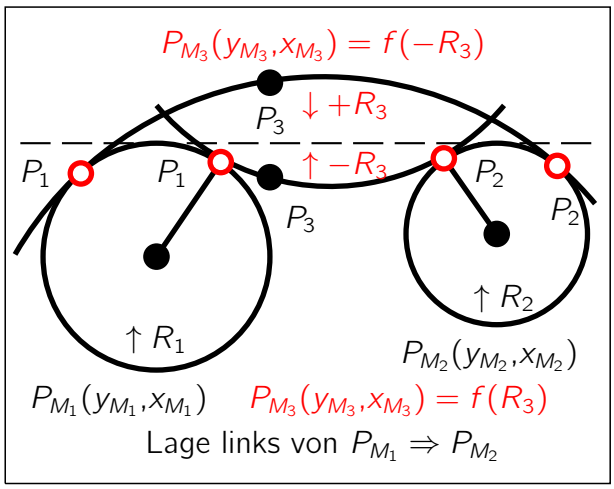




## 55 56 Tangenten, 2 Kreise außen und innen

## 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** $y_1, x_1, y_2, x_2, y_{M_3}, x_{M_3} = tangente11(y_{M_1}, x_{M_1}, R_1, y_{M_2}, x_{M_2}, R_2, \pm R_3)$

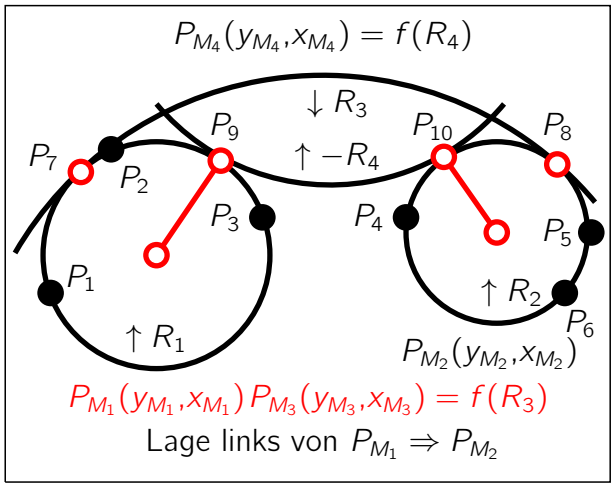
▶ Aufrufe: y1, x1, y2, x2, t1 = tangente9(yM1, xM1, +R1, yM2, xM2, +R2)
y1, x1, y2, x2, t1 = tangente9(yM1, xM1, -R1, yM2, xM2, -R2)
y3, x3, y4, x4, t3 = tangente10(yM1, xM1, +R1, yM2, xM2, -R2)
y3, x3, y4, x4, t3 = tangente10(yM1, xM1, -R1, yM2, xM2, +R2)
y1, x1, y2, x2, yM3, xM3 = tangente11(yM1, xM1,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

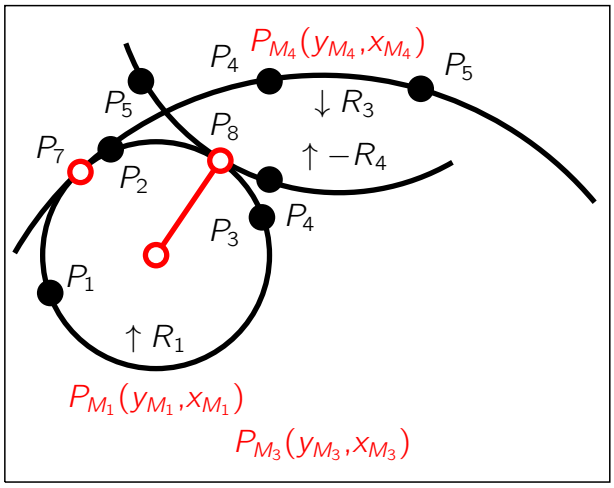



## 58 Tangential, 2 Kreise, 1 Punkt, Umkreis und Ankreis

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

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

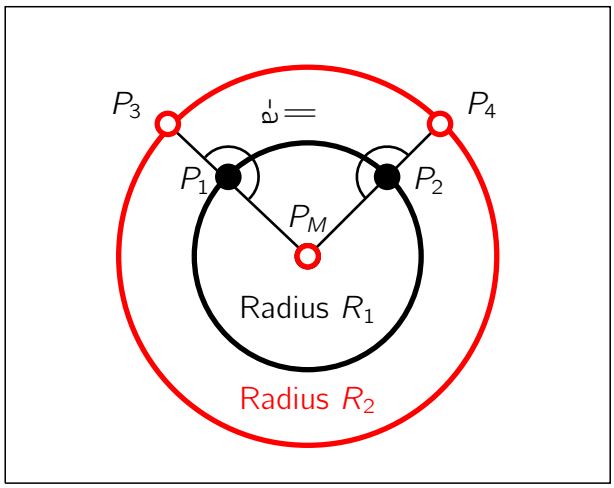
**60** $y_9, x_9, R_1, y_{10}, x_{10}, R_2, y_{M_4}, x_{M4} = tangente14(y_1, x_1\ bis\ y_6, x_6, -R_4)$

**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

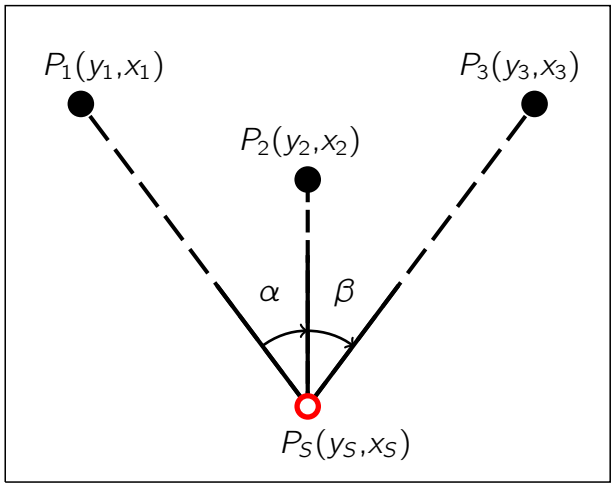
▶ Aufrufe:
```
y7, x7, R1, y8, x8, R2, yM3, xM3 = tangente13(y1, x1 bis y6, x6, R3)
y9, x9, R1, y10, x10, R2, yM4, xM4 = tangente14(y1, x1 bis y6, x6, -R4)
y7, x7, yM1, xM1, R1, yM3, xM3, R3 = tangente15(y1, x1 bis y5, x5)
y8, x8, yM1, xM1, R1, yM4, xM4, -R4 = tangente16(y1, x1 bis y5, x5)
```




**63** **Paralleler Kreisbogen**

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

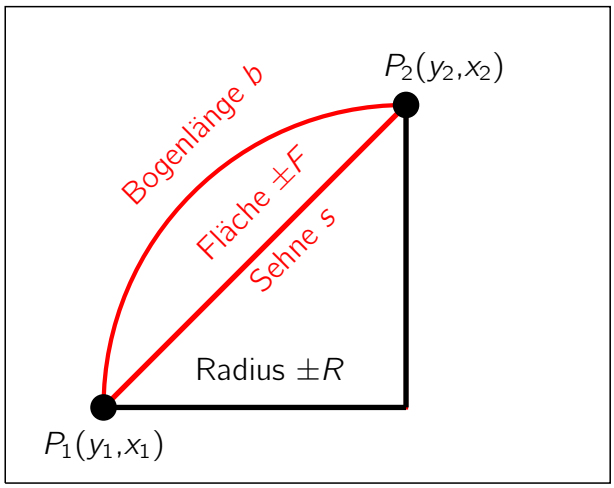
▶ Aufruf:
```
y3, x3, y4, x4, yM, xM, R2 = parallelkreis(y1, x1, y2, x2, R1, a)
```

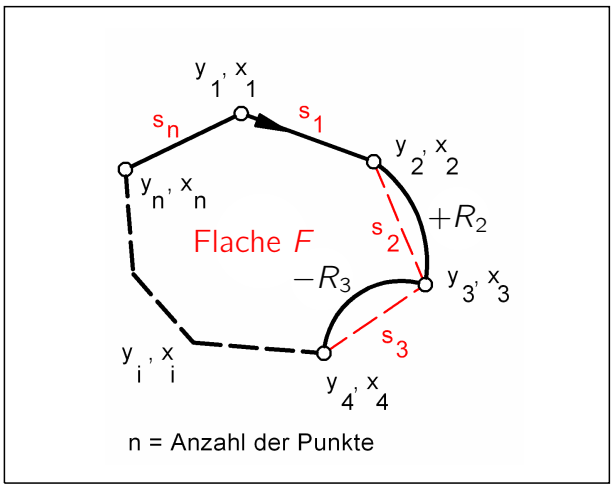



### 64 Rückwärtsschnitt

**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)



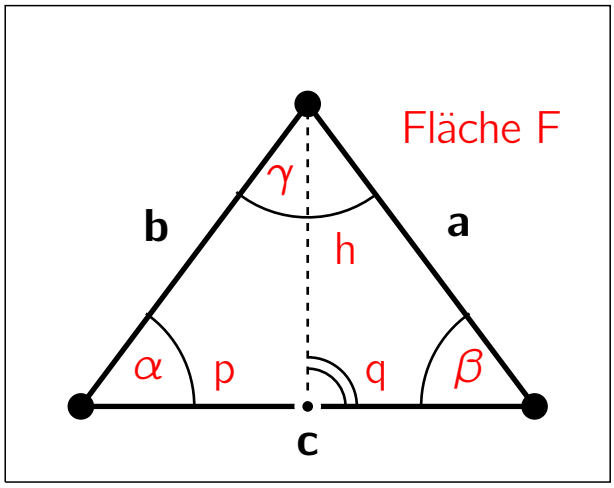

### 65 66 Segment, rechtsläufig

### 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:**

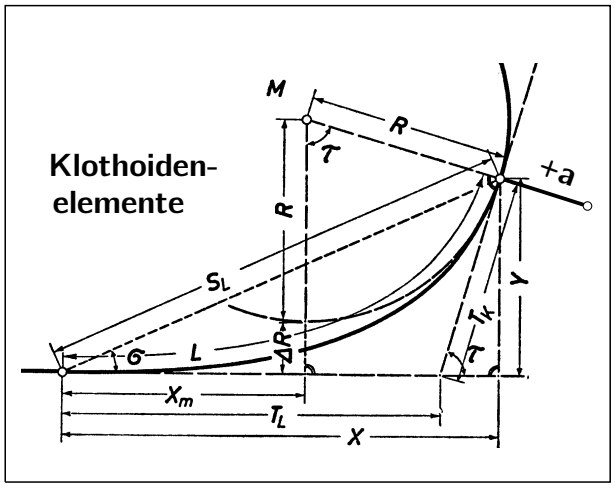
**70** **Strecke: Reduktion von wahrer, örtlicher Strecke nach UTM**

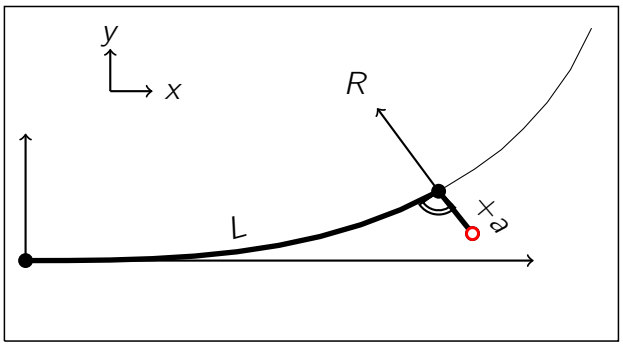
**71** **Strecke: Reduktion von UTM (Koordinaten) in wahre, örtliche Strecke**

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

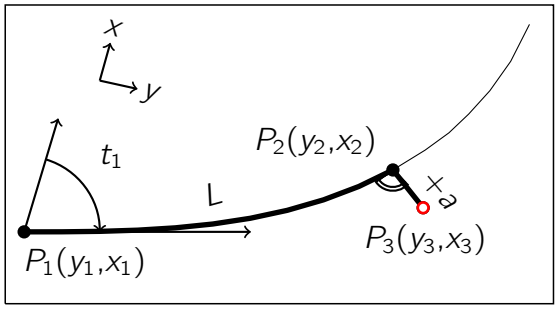
▶ Aufrufe:

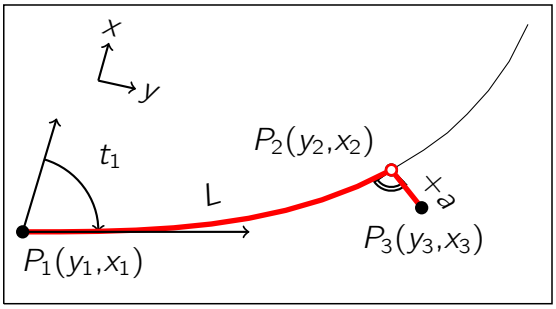
x, y, R, sigma, tau, xM_lokal, yM_lokal, deltaR, TL, TK, y2, x2, y3, x3, yM, xM
= klothoidenelemente(y1, x1, t1, A, L, ± a)
X, Y, R = klothoide(A, L) (vorrangig)
X, Y, R = klothoide2(A, L) (2. Variante)
X, Y, R = klothoide3(A, L) (3. Variante) (Tschebyschow-Approximation)

Erläuterungen:

| | |
|---|---|
| y1 | Rechtswert (übergeordnetes, geodätisches System), Anfangspunkt mit R = ∞ |
| x1 | Hochwert (übergeordnetes System) |
| 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 System |
| sigma_gon | $\sigma$ in gon |
| tau_gon | $\tau$ in gon, Richtungsänderung am Endpunkt |

| | |
|---|---|
| xM_lokal | Rechswert, Kreismittelpunkt des Endpunktes, lokal |
| yM_lokal | Hochwert, Kreismittelpunkt des Endpunktes, lokal |
| 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) |
| x2 | Hochwert, Endpunkt auf der Klothoide (übergeordnetes System) |
| y3 | Rechtswert, Seitwärts gelegender Punkt am Endpunkt (übergeordnetes System) |
| x3 | Hochwert, Seitwärts gelegender Punkt am Endpunkt (übergeordnetes System) |
| yM | Rechtswert, Kreismittelpunkt am Endpunkt (übergeordnetes System) |
| xM | Hochwert, Kreismittelpunkt am Endpunkt (übergeordnetes System) |

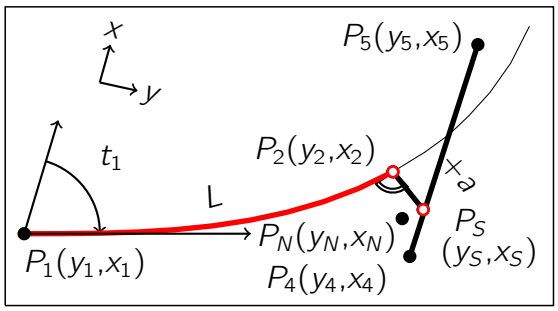


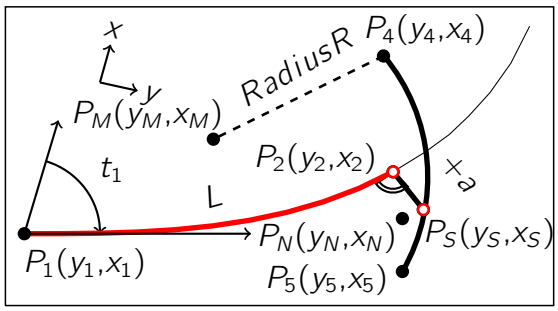

**75** **Klothoide, seitwärtsliegender Punkt** **76** **Klothoide, Punktabstand**

**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**



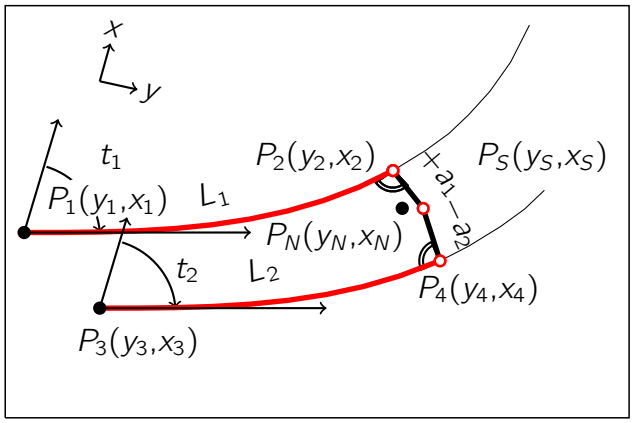

**78** **79** **Klothoide, seitwärts liegender Punkt mit Kreis**

**77**

**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 (red.) | axi |
|---|---|---|---|---|---|
| 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 Beispiel als Array:

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

```
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) |