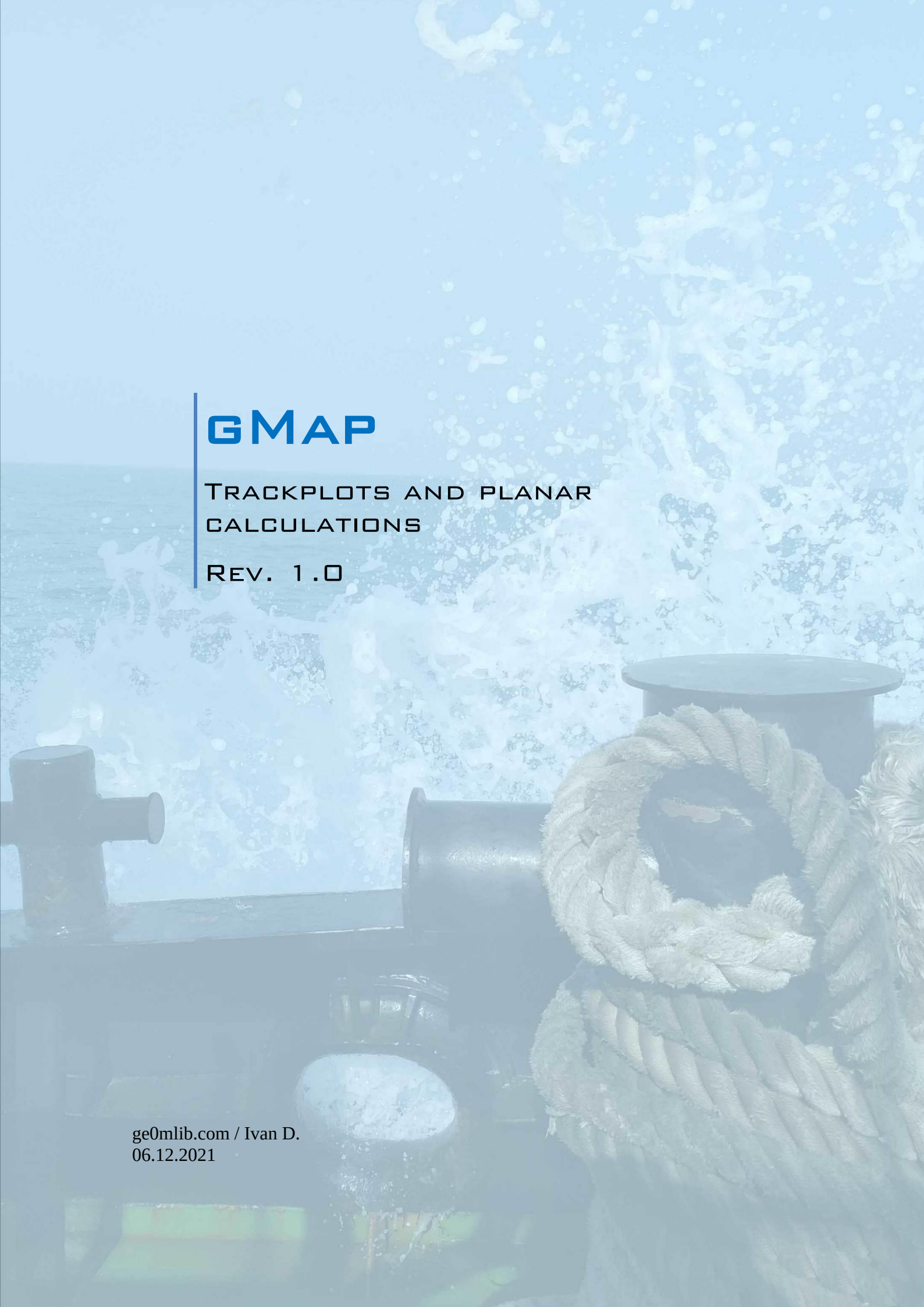




GMAP

TRACKPLOTS AND PLANAR
CALCULATIONS

REV. 1.0

Contents

1. gMap general description	3
1.1 Track-polyline structure types.....	3
1.2 DTEN-fields	5
2. Geometrics tasks decision: minimal distance, cross, normal	6
2.1 Minimal distance for two 2D-polylines.....	6
2.2 Cross points for segments pairs.....	8
2.3 Normal from points to segments	9
2.4 Minimal-distance-normal from points to polyline	11
2.5 Linear approximation for polyline	11
3. Track-polyline structure functions	13
3.1 Read Track-polyline from txt-file	13
3.2 Write Track-polyline to txt-file	14
3.3 Draw Track-polyline	14
3.4 Track-polyline export to AutoCAD	15
3.5 Calculate [length points_num] for Track-polyline	15
3.6 Minimized Z-axis difference for Track-polylines by shift	16
4. Picking and graphics.....	19
4.1 Manual spikes piking for polyline.....	19
4.2 Set Tick Labels format	20
4.3 Set Tips data	20

Tables list

Table 1.1 gMap functions	3
Table 1.2 Track-polyline structure fields' names	3

Figures list

Figure 2.1 gMapGeomPoints2DMinDist using example	8
Figure 2.2 gMapGeomPoints2DMinDist with additional KP using example	8
Figure 2.3 gMapGeomSegments2DCross using example.....	9
Figure 2.4 gGPointsSegments2DNormal using example	10
Figure 2.5 gMapGeomPointsSegments2DNormal using example chart.....	11
Figure 4.1 gMapPickHandleNan tools	20

1. gMap general description

MatLab functions set for simple geometric tasks decision (for example, find cross-point for pipeline and survey line), track-plots drawing (MatLab or AutoCAD scripts) and spikes manual removing. The part of functions was adapted to manipulations with Row-content data; the “Track-polyline structure functions” are depends from structure’s fields names. The set’s functions are shown in [Table 1.1](#).

[Table 1.1](#) gMap functions

Function name	Function description
Geometrics tasks decision: minimal distance, cross, normal	
gMapGeomPoints2DMinDist	Minimal distance for two 2D-polylines
gMapGeomSegments2DCross	Cross points for segments pairs
gMapGeomPointsSegments2DNormal	Normal from points to segments
gMapGeomPointsPolyline2DNormal	Minimal-distance-normal from points to polyline
gMapGeomLineDirect2D	Linear approximation for polyline
Track-polyline structure functions	
gMapPLReadTxt	Read Track-polyline from txt-file
gMapPLWriteTxt	Write Track-polyline to txt-file
gMapPLDraw	Draw Track-polyline
gMapPL2AcadExport	Track-polyline export to AutoCAD
gMapPLLength	Calculate [length points_num] for Track-polyline
gMapPLShiftZaxis	Minimized Z-axis difference for Track-polylines by shift
Picking and graphics	
gMapPickHandleNan	Manual spikes piking for polyline
gMapTickLabel	Set Tick Labels format
gMapTipsLabel	Set Tips data

1.1 Track-polyline structure types

The Track-polyline structure usually is used as 2D-on-plane polyline container. Track-polyline is on-plane polyline structure, which used as “interface structure” for track-plots, line-planning and other “planar objects” drawing and exports to AutoCad. There are Track-polyline structure fields in [Table 1.2](#). Usually Track-polyline is not containing GpsDay and GpsTime fields, but it can be include for compatibility with DTEN-fields functions.

There are follow Track-polyline types: LinePlan, LinePlanKP, PipeLineTrack, Track.

[Table 1.2](#) Track-polyline structure fields’ names

Field name	Field Description
PLName	Track-polyline name.
Type	Track-polyline types: Trackplot, LinePlan, PipeLineTrack, etc.
KeyLineDraw	String key for polyline drawing in MatLab figure (for example: '-r','xb').
GpsE	Polyline’s points rectangular projection Easting.
GpsN	Polyline’s points rectangular projection Northing.

Field name	Field Description
GpsH	Optional field. Polyline's points height created when Ellipsoid-to-Ellipsoid coordinates transformation take place; field used for coordinates transformation's stability.
GpsZ	Optional field. Polyline's points Z-axis coordinate for vertical geodetic datum (pipeline or towing equipment position).
GpsKP	Optional field, polyline KP. This field has different meaning for polyline types: PipeLineTrack – pipeline KP; Trackplot – measurement/ping/shot number.
GpsDay	Optional field. Used as part of DTEN-fields.
GpsTime	Optional field. Used as part of DTEN-fields.
PipeD	Optional field. Pipeline diameter.
WaterDepth	Optional field. Water Depth for current points coordinates (can be “actual” or referenced to some system MLS, LAT, Baltic, etc).

LinePlan (Track-polyline type)

There are used follow field names: PLName, Type='LinePlan', KeyLineDraw, GpsE, GpsN. The LinePlan used for on-plane objects drawing only (gMapPLDraw function) and for objects export to AutoCAD (gMapPL2AcadExport function); there are:

- Line planning net;
- Survey zone area;
- Pipelines (drawing only);
- Platforms and another infrastructure object's borders, which draw with polyline;
- Targets like ADCP, wells and another one-point-targets (the polyline contained only one point).

The LinePlan file format in txt included LineName, E/Lat, N/Lon data:

LineName1, E1, N1, ..., En, Nn

.....

LineNameN, E1, N1, ..., En, Nn

One polyline is one row. The delimiters are: ',' '\t' ';'.

LinePlanKP (Track-polyline type)

There are used follow field names: PLName, Type='LinePlanKP', KeyLineDraw, GpsE, GpsN, GpsKP. The LinePlan used for on-plane objects drawing (gMapPLDraw function) and for objects export to AutoCAD (gMapPL2AcadExport function). The GpsKP-field allows to drawn GpsKP numbers near polyline points and calculates KP for cross-points (for example, for pipeline and survey line).

The GpsKP-field can contain not only Kilometer Points; it can be Shot Number or Measurements Number for Survey line. The LinePlanKP type can use for:

- Pipelines with Kilometer Points;
- Survey Lines track-plots with Measurements Number;
- Line planning net with KP marks (for example, High Resolution seismic).

The LinePlan file format in txt included LineName, E/Lat, N/Lon, KP data:

LineName1, E1, N1, KP1, ..., En, Nn, KPn

.....

LineNameN, E1, N1, KP1, ..., En, Nn, KPn

One polyline-with-KP is one row. The delimiters are: ',' '\t' ';'.

PipeLineTrack (Track-polyline type)

There are used follow field names: PLName, Type='PipeLineTrack', KeyLineDraw, GpsE, GpsN. The optional fields are: GpsKP, GpsZ, PipeD. The PipeLineTrack-type was created for pipe-line drawing and some calculations (for example: cross-point for pipeline and survey line; diffraction point hyperbola from pipeline for SBP-data). The optional fields can be presented or not presented; in the second case the fields' names are equal to LinePlan-type.

The LinePlan file format in txt included LineName, E/Lat, N/Lon, KP, Z, PipeD data:

E, N, KP, Z, PipeD

.....

En, Nn, KPn, Zn, PipeDn

One polyline is presented as single file which includes from 2 to 5 columns. The delimiters are: ',' '\t' ';'.

Track (Track-polyline type)

There are used follow field names: PLName, Type='Track', KeyLineDraw, GpsE, GpsN, GpsDay, GpsTime. The optional fields are: GpsH, GpsKP, GpsZ. The Track-type was created for Equipment's track-plot drawing and some calculations. The Track-type contains DTEN-fields (see below) for track-plot coordinates transformation from datum to datum.

The Track file format in txt included follow data:

YYYY1 MM1 DD1 hh1 mm1 ss.sss1 E1 N1 H1 KP1 Z1 WaterDepth

.....

YYYYn MMn DDn hhn mmn ss.sssn En Nn Hn KPn Zn WaterDepth

Where: YYYY – year, MM – month, DD – day, hh – hour, mm – minute, ss.sss – second; it will converted to GpsDay, GpsTime fields.

One polyline is presented as single file which includes from 8 to 11 columns. The delimiters are: ',' '\t' ';'.

1.2 DTEN-fields

GpsKP, GpsDay, GpsTime, GpsE, GpsN, GpsH are named the DTEN-fields, it is define measurement time and coordinates in planar projection. The fields are used for “universalized” time/coordinates fields for different structures, for example sgy-structure, xtf-structure, jsf-structure (the functions were used gSgyDTEN, gSgyDTENinv, gXtfDTEN, gJsfdTEN).

2. Geometrics tasks decision: minimal distance, cross, normal

There are some geometrics trivial tasks decisions:

- minimal distance from point to polyline/points_set;
- cross points calculation for segments;
- distance from point to segments/lines by normal.

Axis direction: y- up, x- right; rotation: from x, left is +.

0) Lines equations were used

Line general equation is

$$Ax + By + C = 0$$

Parametric equation is

$$\frac{A}{-C}x + \frac{B}{-C}y = 1;$$

1) Across two points Line's equation

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = 0;$$

$$(y_1 - y_2) \cdot x + (x_2 - x_1) \cdot y + (x_1 y_2 - y_1 x_2) = 0;$$

$$A = y_1 - y_2; B = x_2 - x_1; C = x_1 y_2 - y_1 x_2;$$

General line's equation $A_1x + A_2y + A_3 = 0$, for points a1 and a2 code is

>> len=size(xy1,2); %xy1 and xy2 are rows with first and second points' coordinates

>> ABC=cross([xy1(1,:);xy1(2,:);ones(1,len)], [xy2(1,:);xy2(2,:);ones(1,len)],1);

2) Cross lines point equation

$$\begin{vmatrix} A & B & C \\ A_1 & B_1 & C_1 \\ A_2 & B_2 & C_2 \end{vmatrix} = 0;$$

$$A \cdot (B_1 C_2 - C_1 B_2) + B \cdot (C_1 A_2 - A_1 C_2) + C \cdot (A_1 B_2 - B_1 A_2) = 0;$$

Point coordinates are

$$A \cdot \frac{B_1 C_2 - C_1 B_2}{A_1 B_2 - B_1 A_2} + B \cdot \frac{C_1 A_2 - A_1 C_2}{A_1 B_2 - B_1 A_2} + C = 0;$$

$$x = \frac{B_1 C_2 - C_1 B_2}{A_1 B_2 - B_1 A_2}; y = \frac{C_1 A_2 - A_1 C_2}{A_1 B_2 - B_1 A_2};$$

Cross points (x,y) for $A_1x + B_1y + C_1 = 0$ and $A_2x + B_2y + C_2 = 0$ lines code is

>> % ABC1 and ABC2 are rows with crossed lines coefficients;

>> k=cross(ABC1,ABC2,1); xy=[k(1,:)/k(3,:);k(2,:)/k(3,:);

If lines are collinear, then c=[0 0 0];d=[nan nan].

2.1 Minimal distance for two 2D-polylines

function [nk2,nK2,Dmin]=gMapGeomPoints2DMinDist(kxy,dk,KXY,dK,key_one)

Find minimal distance from each kxy-points to all KXY-points (2D only). If key_one parameter set to one, than least distance from all kxy-points to all KXY-points be founded (for example for polyline “cross-point” search).

The kp-column for kxy and KXY can be used as numbers/ID for points. We can consider points as consecutive points of polyline, and kp-column as “distance” between points; using dk and dK parameters we can add additional points (linear interpolation) between existing for more careful distance search.

Parameters:

kxy – rows [kp;x;y] with points_1 (polyline_1) coordinates;

dk – kp-step for points_1 (polyline_1) point to point; if isempty, than kp is not recalculated;

KXY – rows [KP;X;Y] with points_2 or polyline_2 coordinates;

dK – KP-step for polyline_2 point to point; if isempty, than kp is not recalculated;

key_one – if key==1 then the least distance from all kxy-points to all KXY-points be founded.

nk2 – points_1 [numbers;X;Y] or polyline_1 [kp;X;Y] for minimal distance search;

nK2 – points_2 [numbers;X;Y] or polyline_2 [KP;X;Y] for minimal distance with k2(n) was founded;

Dmin – minimal distance value between k2 and nK2;

Example 1.

```
>> [k2,nK2,Dmin]=gMapGeomPoints2DMinDist(kxy,[],KXY,[],0);
```

```
>> [k2,nK2,Dmin]=gMapGeomPoints2DMinDist(kxy,0.1,KXY,0.1,0);
```

Example 2 (**Figure 2.1**). Pipelines KP and coordinates read to ‘a’; SBP pings numbers and coordinates read to ‘xy’. Calculate nearest point’s values.

```
>> a=dlmread('e:\01_Surveys\2015_KpXY_final.txt'); %a=[KP X Y];
```

```
>> JsFHead=gJsFHeaderRead('e:\ROMONA\16_04.000a_CV.jsf',0);
```

```
>> [Head,Data]=gJsF0080Read(JsFHead,0,0); xy=[1:length(Head.X);Head.X./100;Head.Y./100];
```

```
>> [nk2,nK2,Dmin]=gMapGeomPoints2DMinDist(xy,[],a(:,1:3)',[],1);
```

```
>> figure(1); plot(xy(:,2),xy(:,3),'.r'); hold on; plot(a(:,2),a(:,3),'ob'); plot(xy(nk2(1)-  
xy(1,1)+1,2),xy(nk2(1)- xy(1,1)+1,3),'.xr'); hold on; plot(a(nK2(1)-a(1,1)+1,2),a(nK2(1)-a(1,1)+1,3),'.xb');  
hold off;
```

Example 3 (**Figure 2.2**). Pipelines KP and coordinates read to ‘a’; SBP pings numbers and coordinates read to ‘xy’. Calculate nearest point’s values; using smaller step (additional points).

```
>> a=dlmread('e:\01_Surveys\2015_KpXY_final.txt'); %a=[KP X Y];
```

```
>> JsFHead=gJsFHeaderRead('e:\16_04.000a_CV.jsf',0);
```

```
>> [Head,Data]=gJsF0080Read(JsFHead,0,0); xy=[1:length(Head.X);Head.X./100;Head.Y./100];
```

```
>> [nk2,nK2,Dmin]=gMapGeomPoints2DMinDist(xy,0.5,a(:,1:3)',0.1,1);
```

```
>> figure(1); plot(xy(:,2),xy(:,3),'.r'); hold on; plot(a(:,2),a(:,3),'ob'); plot(nk2(2),nk2(3),'.xr');hold on;  
plot(nK2(2),nK2(3),'.xb'); hold off;
```

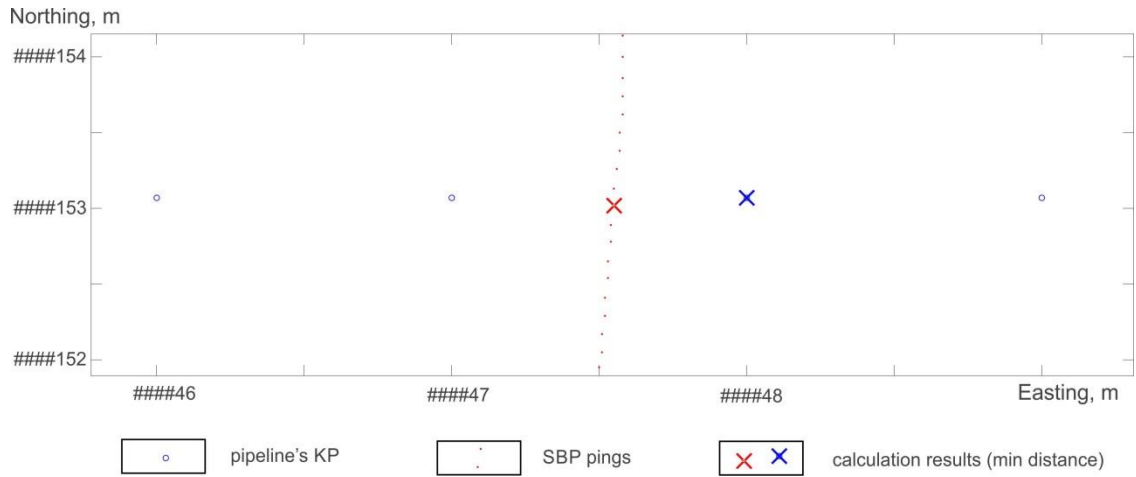


Figure 2.1 gMapGeomPoints2DMinDist using example

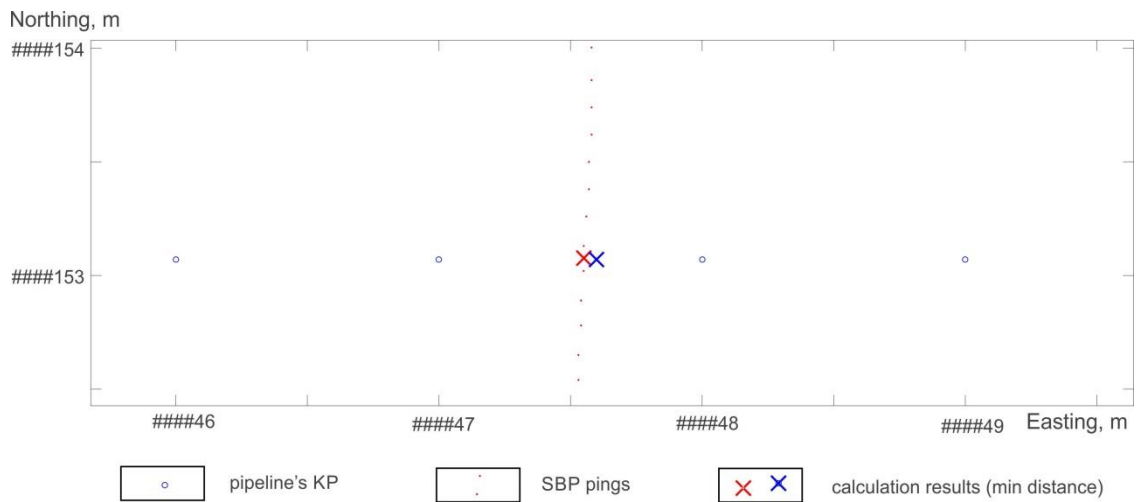


Figure 2.2 gMapGeomPoints2DMinDist with additional KP using example

2.2 Cross points for segments pairs

function [d,mask]=gMapGeomSegments2DCross(a1,a2,b1,b2)

Find cross points for segments pairs a1_to_a2 and b1_to_b2 (2D only); cross point coordinates is nan, if segments in pair are parallel. The a1 and a2 are first and last a-segment's points; b1 and b2 are first and last b-segment's points. The number of a and b segments must be equal; if only one a-segment was define for several b-segments, than a-segment will be replicate to b-segments number.

Parameters:

a1 – rows with a1(x,y) points coordinates for segments a1_to_a2;

a2 – rows with a2(x,y) points coordinates for segments a1_to_a2;

b1 – rows with b1(x,y) points coordinates for segments b1_to_b2;

b2 – rows with b2(x,y) points coordinates for segments b1_to_b2;

d – rows with cross point (x,y) for lines, created on a1_to_a2 and b1_to_b2;

mask – row mask for "cross point in borders of segments a1_to_a2 and b1_to_b2";

Example 1.

```
>> [d,mask]=gMapGeomSegments2DCross([0 0;0 0;0 0],[10 10;10 10;10 10],[-8 9;-3 5;6 -1],[-3 5;6 -1;10 -2]);
>> [d,mask]=gMapGeomSegments2DCross([1 1],[10 10],[-8 9;-3 5;6 -1],[-3 5;6 -1;10 -2]);
```

Example 2 (**Figure 2.3**). Pipelines KP and coordinates read to 'a'; SBP pings numbers and coordinates read to 'xy'. Calculate cross-point.

```
>> a=dlmread('e:\01_Surveys\2015_KpXY_final.txt'); %a=[KP X Y];
>> JsHead=gJsfHeaderRead('e:\16_04.000a_CV.jsf',0);
>> [Head,Data]=gJsf0080Read(JsHead,0,0); xy=[1:length(Head.X);Head.X./100;Head.Y./100];
>> [nk2,nK2,Dmin]=gMapGeomPoints2DMinDist(xy,[],a(:,1:3),[],1);
>> figure(1); plot(xy(:,2),xy(:,3),'r'); hold on; plot(a(:,2),a(:,3),'ob'); plot(xy(nk2(1)-xy(1,1)+1,2),xy(nk2(1)-xy(1,1)+1,3),'xr'); hold on; plot(a(nK2(1)-a(1,1)+1,2),a(nK2(1)-a(1,1)+1,3),'xb'); hold off;
>> [d,mask]=gMapGeomSegments2DCross(a(nK2(1)-a(1,1),2:3),a(nK2(1)-a(1,1)+1,2:3),xy(1:end-1,2:3),xy(2:end,2:3));
>> plot(d(1,find(mask)),d(2,find(mask)),'x');
```

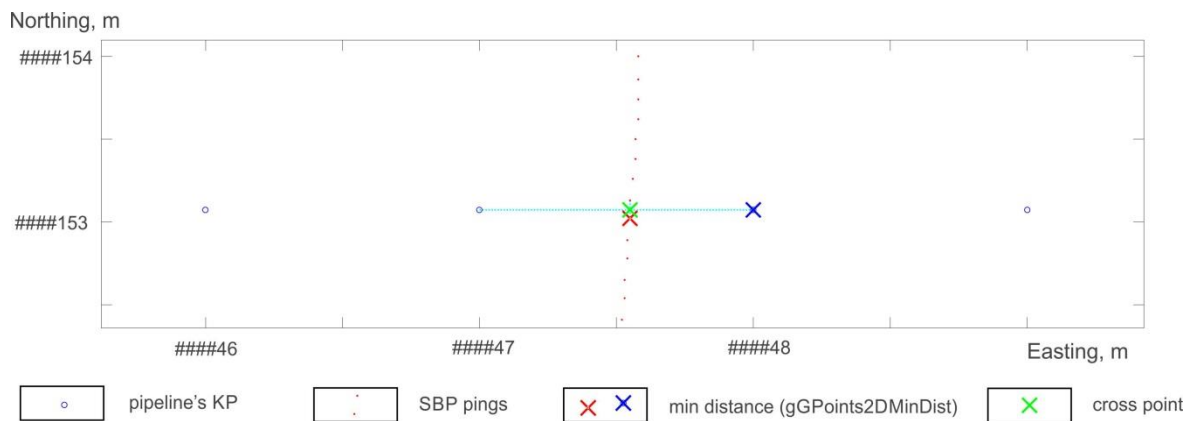


Figure 2.3 gMapGeomSegments2DCross using example

2.3 Normal from points to segments

function [d,r,mask]=gMapGeomPointsSegments2DNormal(a1,b1,b2)

Find cross points for pairs normal from points a1 and segments b1_to_b2 (2D only). The number of a1 points and b segments must be equal; if only one a1-point was define for several b-segments, than a1-point will be replicate to b-segments number. Functions works correctly if point on line formed on b1_to_b2 segment.

Parameters:

a1 – rows with a1(x,y) points coordinates;

b1 – rows with b1(x,y) points coordinates for segments b1_to_b2;

b2 – rows with b2(x,y) points coordinates for segments b1_to_b2;

d – rows with cross point coordinates (x,y) for normal from a1 to line formed on b1_to_b2;
r – row "signed" distance from a1 to cross point (x,y); if we go from first to second Kp then left side "sign" is negativ.
mask – mask for "cross point in borders of b1_to_b2 segment".

Example 1.

```
>> [d,r,mask]=gMapGeomPointsSegments2DNormal([0 0;0 0;0 0]',[-8 9;-3 5;6 -1]',[-3 5;6 -1;10 -2]');
>> [d,r,mask]=gMapGeomPointsSegments2DNormal([1 1]',[-8 9;-3 5;6 -1]',[-3 5;6 -1;10 -2]');
```

Example 2 (**Figure 2.4**). Pipelines KP and coordinates read to 'a'; SBP pings numbers and coordinates read to 'xy'. Calculate normal from pipelines KP to SBP-trackplot-segments.

```
>> a=dlmread('e:\01_Surveys\2015_KpXY_final.txt'); %a=[KP X Y];
>> JsFHead=gJsFHeaderRead('e:\16_04.000a_CV.jsf',0);
>> [Head,Data]=gJsF0080Read(JsFHead,0,0); xy=[1:length(Head.X);Head.X./100;Head.Y./100]';
>> [nk2,nK2,Dmin]=gMapGeomPoints2DMinDist(xy,[],a(:,1:3)',[],1);
>> figure(1); plot(xy(:,2),xy(:,3),'r'); hold on; plot(a(:,2),a(:,3),'ob'); plot(xy(nk2(1)-
xy(1,1)+1,2),xy(nk2(1)-xy(1,1)+1,3),'xr'); hold on; plot(a(nK2(1)-a(1,1)+1,2),a(nK2(1)-a(1,1)+1,3),'xb');
>> [d,r,mask]=gMapGeomPointsSegments2DNormal(a(nK2(1)-a(1,1)+1,2:3)',xy(1:end-
1,2:3)',xy(2:end,2:3)');
>> plot(d(1,find(mask)),d(2,find(mask)),'xg');
```

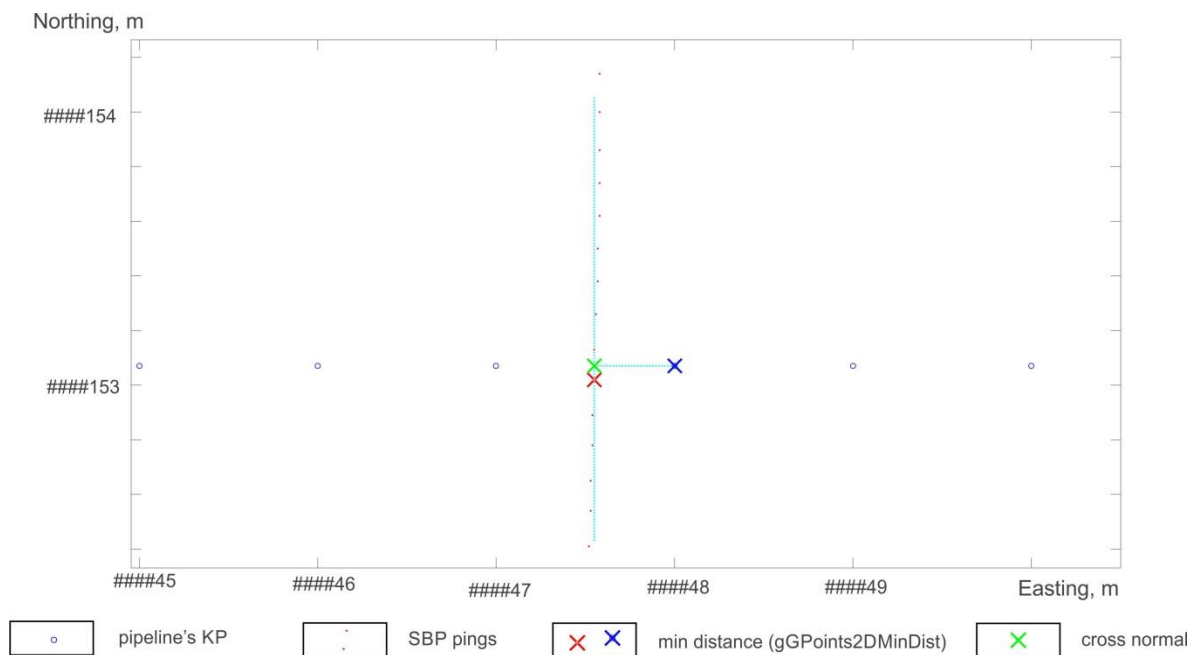


Figure 2.4 gGPointsSegments2DNormal using example

Example 3 (**Figure 2.5**)

```
>> [d,r,mask]=gMapGeomPointsSegments2DNormal([1;1],[0;3],[5;0]);
ans: d = [1.617647058823529; 2.029411764705882]; r = 1.200490095997562; mask = 1
```

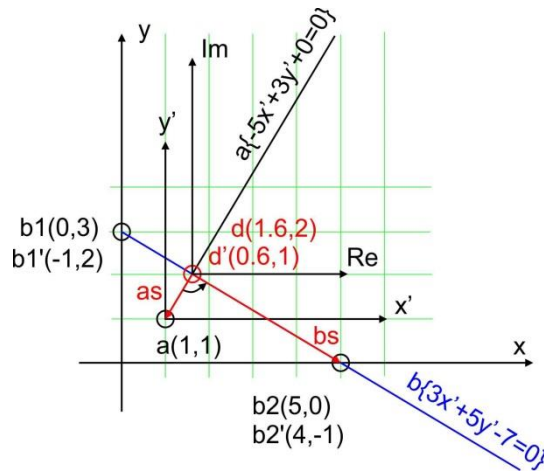


Figure 2.5 gMapGeomPointsSegments2DNormal using example chart

2.4 Minimal-distance-normal from points to polyline

function [dd,rrr]=gMapGeomPointsPolyline2DNormal(a,b)

Find cross points for normal from points a to polyline b (2D only).

Parameters:

a – rows with a(x,y) points coordinates;

b – rows with b(kp,x,y) points coordinates for polyline;

dd – rows with cross point coordinates (x,y) for normal from points to polyline;

r – row "signed" distance from points to cross points (x,y); if we go from first to second kp then left side "sign" is negativ.

Algorithm: 1) find min distance from point to polyline points, 2) try to draw normal for two segments near minimal-distance-point.

Example.

```
>> [dd,r]=gMapGeomPointsPolyline2DNormal([1;1],[1 2 3;-5 -1 7;8 -3 -6]);
>> a=dlmread('d:\pointsXY.txt');b=dlmread('e:\KpXY.txt');
>> [dd,r]=gMapGeomPointsPolyline2DNormal(a,b);
>> plot(b(2,:),b(3,:),'-');hold on;plot(a(1,:),a(2,:),'o');plot(dd(2,:),dd(3,:),'x');axis equal;
```

2.5 Linear approximation for polyline

function [a,b,stdxy,mask]=gMapGeomLineDirect2D(X,Y,rk)

Linear approximation for polyline; robust calculation a,b for $y=ax+b$. Used for Survey line direction calculation.

Parameters:

X – vector x;

Y – vector y;

rk – vector std-coefficients for robust procedure, usually [3 2.5];

a,b – calculated a and b;

stdxy – standard deviation for $Y-ax+b$;

mask – logical mask for elements y and x, which take part in a,b calculation.

Example:

```
>> [a,b,stdxy,mask]=gMapGeomLineDirect2D([1 3 2 5 7 9 3 1],[7 15 90 23 31 39 15 7],[3 2.5 2]);
```

3. Track-polyline structure functions

3.1 Read Track-polyline from txt-file

function PL=gMapPLReadTxt(fName,keyRead,KeyLineDraw)

Read data from txt-file to Track-polyline structure (2D-on-plane polyline).

Parameters:

fName – file name for reading;

keyRead – key for reading: 1) LinePlan format; 2) PipeLineTrack format; 3) LinePlanKP file format; 4) Track in DTEN format;

if keyRead(1)==2, than keyRead(2..5) is columns numbers for [E N KP Z PipeD], set NaN if data not exist for GpsKP or GpsZ

if keyRead(1)==4, than keyRead(2..12) is [YYYY MM DD HH MM SS.SSS E N H KP Z], set NaN if data not exist for GpsH or GpsKP or GpsZ

keyLineDraw – string key for line drawing: '-r','xb', etc;

PL – output structure: PL(n).PLName, PL(n).Type, PL(n).KeyLineDraw, PL(n).GpsE, PL(n).GpsN;

Extended fields are: PL(n).GpsH, PL(n).GpsKP, PL(n).GpsDay, PL(n).GpsTime, PL(n).PipeD.

Function Example:

```
>> PL=gMapPLReadTxt('c:\temp\SSS\V3LinePlan.txt',1,'-c');gMapPLDraw(100,PL);axis equal;
```

```
>> PL=gMapPLReadTxt('c:\temp\SSS\V3LinePlan.txt',[2 1 2 5 nan nan],'-c');
```

1--LinePlan file format

There are rows included LineName and E/Lat, N/Lon coordinates:

LineName1, E1, N1, ..., En, Nn

.....

LineNameN, E1, N1, ..., En, Nn

The delimiters are: ',' '\t' ';'.

2--PipeLineTrack file format

There are a number of columns included E, N, KP(?), Z(?); columns positions are defined in keyRead:

E, N, KP, Z, PipeD

.....

En, Nn, KPn, Zn, PipeDn

The delimiters are: ',' '\t' ';'.

3--LinePlanKP file format

There are rows included E, N, KP:

LineName1, E1, N1, KP1, ..., En, Nn, KPn

.....

LineNameN, E1, N1, KP1, ..., En, Nn, KPn

The delimiters are: ',' '\t' ';'.

4--Track in DTEN format

There are a number of columns included Date, Time, E, N, H(?), KP(?), Z(?), WaterDepth(?):

YYYY1 MM1 DD1 hh1 mm1 ss.sss1 E1 N1 H1 KP1 Z1 WaterDepth

.....

YYYYn MMn DDn hhn mmn ss.sssn En Nn Hn KPn Zn WaterDepth

3.2 Write Track-polyline to txt-file

function gMapPLWriteTxt(fName,PL,keyWrite)

Export Track-polyline structure to txt-file.

Parameters:

fName – file or folder name for export;

PL – polyline structure: PL(n).PLName; PL(n).Type; PL(n).KeyLineDraw; PL(n).GpsE; PL(n).GpsN;
PL(n).GpsKP

keyWrite(1) – Track-polyline type code.

if keyWrite(1)=1 >> 'LinePlan', than all lines write to single file with name "fName", in "LinePlan file format";

if keyWrite(1)=2 >> 'PipeLineTrack', than each line write to own file with names "PL(n).PLName" at folder "fName", in "PipeLineTrack" file format;

if keyWrite(1)=3 >> 'LinePlanKP', than all lines write to single file with name "fName", in "LinePlanKP file format".

keyWrite(2) – digits number for numbers in file.

Function Example:

```
>> gMapPLWriteTxt('c:\temp\PL1.txt',PL,[1 3]);
```

3.3 Draw Track-polyline

function gMapPLDraw(figDraw,PL,flText)

Draw poly lines from Track-polyline structure.

Parameters:

f – figure number or handle;

PL – track-polyline structure, field used: PL(n).PLName(?); PL(n).KeyLineDraw(?); PL(n).GpsE;
PL(n).GpsN;

Function Example:

```
>> PLLine=gMapPLReadTxt('c:\temp\SSS\V3LinePlan.txt',1,'-c');
```

```
>> gMapPLDraw(100,PLLine);axis equal;
```

3.4 Track-polyline export to AutoCAD

function gMapPL2AcadExport

(fName,PL,PLNameAttr,CircleAttr,KPNameAttr,DigitNum,PointsStep,flLayers)

Export Track-polyline structure to AutoCad script

Parameters:

fName – file or folder name for export; if fName(end)=='\', then each PL write to file [PL(n).PLName '.scr'];

PL – polyline structure includes: PL(n).PLName; PL(n).GpsE; PL(n).GpsN; PL(n).GpsKP(?);

PLNameAttr=[Size Angle dE dN CircleRadius] – PL(n).PLName text attributes;

if PLNameAttr=[Size Angle], than dE=0, dN=0;

if PLNameAttr includes CircleRadius, then draw circle for first polyline's point, using CircleRadius;

CircleAttr=[Radius DrawModulio] – circles draw attributes for polyline's points;

KPNameAttr=[Size Angle DrawModulio KPDigitNum] – PL(n).GpsKP text attributes for polyline's points
(combine with circles);

DigitNum – number of digits after floating point for pline/circles;

PointsStep – step of points for polyline draw (gAcadPline);

flLayers – each pline to own layer.

Acad coordinates:

^pY

|

o--->pX

Function Example:

```
>> PLXtf=gXtf000Dir2PLRead('c:\temp\SSS\3\' '-b',[6378137 0.081819190842621],[0 141 0.9996  
500000 0],[1,1,1);
```

```
>> gMapPL2AcadExport('c:\temp\SSS\3\Cad\',PLXtf,[7 0 0 2 3],[5 100],[2 0 500 1],2,1,1);
```

```
>> PL=gMapPLReadTxt('e:\Lazarev170818.txt',3,'.');
```

```
>> gMapPL2AcadExport('c:\temp\333.scr',PL,[3 0],[0.3 1],[1 0 500 2],2,1,0);
```

3.5 Calculate [length points_num] for Track-polyline

function Len=gMapPLLength(PL,sm)

Calculate [length points_num] for poly-lines from Track-polyline structure

Parameters:

PL – track-polyline structure, field used: PL(n).GpsE; PL(n).GpsN; PL(n).GpsZ;

sm – smooth value;

LenNum=[length points_num].

Function Example:

```
>> PL=gMapPLReadTxt('c:\temp\SSS\V3LinePlan.txt',1,'-c');
>> gMapPLDraw(100,PL); axis equal; LenNum=gMapPLLength(PL,10);
```

3.6 Minimized Z-axis difference for Track-polylines by shift

function [PL,varargout]=gMapPLShiftZaxis(PL,dMax,NumIter,drFl) 17/10/2020

Minimized mean Z-axis difference for Track-polylines set in a cross points. The function shifted PLs, using $PL(n).GpsZ = PL(n).GpsZ + \text{difference}$. Each PL must have one cross-point minimum.

Parameters:

PL – input structure: PL(n).PLName; PL(n).GpsKP; PL(n).GpsE; PL(n).GpsN; PL(n).GpsZ; PL(n).Type;

PL(n).KeyLineDraw;

dMax – maximum mean Z-difference between crosses for each PL;

NumIter – maximum number of iteration for PL's shifting;

drFl – flag for draw figure with differences;

AcadFile – file name for Autocad's script (no file will be created if empty);

PL – output structure: PL(n).PLName; PL(n).GpsKP; PL(n).GpsE; PL(n).GpsN; PL(n).GpsZ; PL(n).Type;

PL(n).KeyLineDraw;

varargout{1} – dZshift - adds for each PL;

varargout{2}=[dZpl dZstd dZnum dZmad dZrms] – statistical estimations:

dZpl – mean GpsZ-difference between crosses for each PL;

dZstd – standard deviation around dZpl;

dZnum – the number of cross-points;

dZmd – mean absolute for Z-differences (for all PL's crosses);

dZrm – root-mean-square for Z-differences;

varargout{3} – dZnode - max differens value for all crosses;

varargout{4} – Num - crosses-matrix;

varargout{5} – Num2 - balanced crosses-matrix.

Remark: the "nanmean" function for using without Statistical Toolbox was added as a local.

Function Example:

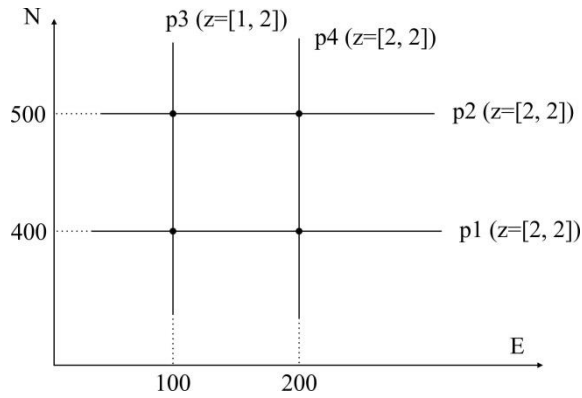
```
>> [PL2,dZshift,dZpl,dZnode,Num,Num2]=gMapPLShiftZaxis(PL,0.01,500000,1,'c:\temp\zshift');
```

Function gMapPLShiftZaxis: short algorithm description

0) Set polyline structure:

```
>> PL(1)=struct('PLName','p1','GpsKP',[180 190],'GpsE',[200 100],'GpsN',[400 400],'GpsZ',[2 2]);
>> PL(2)=struct('PLName','p1','GpsKP',[280 290],'GpsE',[200 100],'GpsN',[500 500],'GpsZ',[2 2]);
>> PL(3)=struct('PLName','p1','GpsKP',[380 390],'GpsE',[100 100],'GpsN',[500 400],'GpsZ',[1 2]);
>> PL(4)=struct('PLName','p1','GpsKP',[480 490],'GpsE',[200 200],'GpsN',[500 400],'GpsZ',[2 2]);
>> [PL,v]=gMapPLShiftZaxis(PL,0.01,10,1,'c:\temp\qzz.txt');
```

The figure of crossed lines



Start calculations

```
>> [PL2,dZshift,dZpl,dZnode,Num,Num2]=gMapPLShiftZaxis(PL,0.01,500000,1,'c:\tmp\qzz.t');
```

1) The cross-points between each line to all other lines are finding.

2) Using cross-points the algorithm formed Num – 4 “layers” matrix.

Num(1) >> contained point number for PL-row which crossed with PL-column.

	PL1	PL2	PL3	PL4
PL1	NaN	NaN	2	1
PL2	NaN	NaN	2	1
PL3	2	1	NaN	NaN
PL4	2	1	NaN	NaN

Num(2) >> contained Z for point number for PL-row which crossed with PL-column.

	PL1	PL2	PL3	PL4
PL1	NaN	NaN	2	2
PL2	NaN	NaN	2	2
PL3	2	1	NaN	NaN
PL4	2	2	NaN	NaN

Num(3) >> contained point number for PL-column which crossed with PL-row.

	PL1	PL2	PL3	PL4
PL1	NaN	NaN	2	2
PL2	NaN	NaN	1	1
PL3	2	2	NaN	NaN
PL4	1	1	NaN	NaN

Num(4) >> contained Z for point number for PL-column which crossed with PL-row.

	PL1	PL2	PL3	PL4
PL1	NaN	NaN	2	2
PL2	NaN	NaN	1	2
PL3	2	2	NaN	NaN
PL4	2	2	NaN	NaN

3) The calculate difference between Z for cross-points.

Num(2) – Num(4) >> difference between Z for PL-row and Z for PL-column.

	PL1	PL2	PL3	PL4
PL1	NaN	NaN	0	0
PL2	NaN	NaN	1	0
PL3	0	-1	NaN	NaN
PL4	0	0	NaN	NaN

4) Calculate the mean for columns with difference.

dZpl >> mean for columns (it means “for each line”)

	PL1	PL2	PL3	PL4
PL1	0	-0.5	0.5	0

5) Find the maximum absolute difference (first value). The PL2 has the max “shift” from other lines.

6) Shift PL2 using “maximum difference”.

Num(2) >> add (-0.5) to Z for PL2-row

	PL1	PL2	PL3	PL4
PL1	NaN	NaN	2	2
PL2	NaN	NaN	1.5	1.5
PL3	2	1	NaN	NaN
PL4	2	1	NaN	NaN

Num(4) >> add (-0.5) to Z for PL2-column

	PL1	PL2	PL3	PL4
PL1	NaN	NaN	2	2
PL2	NaN	NaN	1	1
PL3	2	1.5	NaN	NaN
PL4	2	1.5	NaN	NaN

7) Make iterations for steps 3-to-6 to “small difference” or to “the number of cycles” true flag.

The final “difference matrix” Num(2) – Num(4).

	PL1	PL2	PL3	PL4
PL1	NaN	NaN	-0.25	0.25
PL2	NaN	NaN	0.25	-0.25
PL3	0.25	-0.25	NaN	NaN
PL4	-0.25	0.25	NaN	NaN

The final Num(2) and Num(4)

	PL1	PL2	PL3	PL4
PL1	NaN	NaN	2	2
PL2	NaN	NaN	1.5	1.5
PL3	2.25	1.25	NaN	NaN
PL4	1.75	2.75	NaN	NaN

	PL1	PL2	PL3	PL4
PL1	NaN	NaN	2.25	1.75
PL2	NaN	NaN	1.25	1.75
PL3	2	1.5	NaN	NaN
PL4	2	1.5	NaN	NaN

8) Calculate statistics for “difference matrix” (MAD=0.25) and shift for each PL (Num4_final-Num4)

	PL1	PL2	PL3	PL4
PL1	0	-0.5	0.25	-0.25

4. Picking and graphics

4.1 Manual spikes piking for polyline

function p=gMapPickHandleNan(X,Y,f)

Set to Nan points on curve using manual piking.

Parameters:

X,Y – rows with curve's coordinates;

f – figure number or pointer to figure;

p – pointer to curve;

a=get(p,'Ydata') – get Y-coordinate with NaN;

a=get(s,'UserData') – get logical mask for ~NaN value.

Function Example

```
>> s=gMapPickHandleNan(HGps.GpsLat,HGps.GpsLon,100);a=get(s,'Ydata');a=get(s,'UserData');
```

There are two picking tools (*Figure 4.1*):

- 1) rectangle (first button at added panel) – set to nan all points in selected rectangle;
- 2) curve_part (second button at added panel) – set to nan all points in selected curve's part.

Point at curve set using minimal distance from mouse-click, distance calculates for current axis-scale (axis not equal).

Buttons:

LeftMouseButton – first selection element (first part of rectangle or first point at curve);

RightMouseButton – second selection element (second part of rectangle or second point at curve);

MiddleMouseButton – set to NaN points in selected area (to nan set Y-coordinate value);

z – undo;

x – redo;

q – Exit.

DataTips show: X,Y – coordinate; point number; curve ID.

There can create several independent "nan-pick-button" at one figure for different curves.

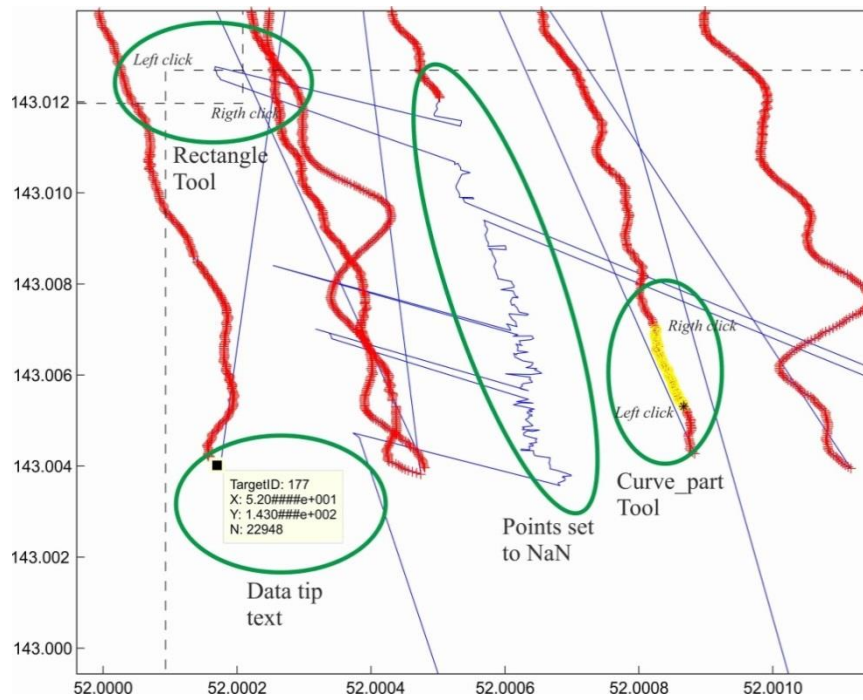


Figure 4.1 gMapPickHandleNan tools

4.2 Set Tick Labels format

function gMapTickLabel(fig,key,fntSize)

Remove exponent, set format and font for figure's Tick Labels.

Parameters:

fig – figure number or handle;

key – format for label: '\$%,.2f', '%g\circ', '%g%%', '%,g', '%,4.4g','%+4.4g','%04.4g','%-4.4g','%#4.4g'
(<https://www.mathworks.com/help/matlab/ref/matlab.graphics.axis.decorator.numericruler-properties.html>)

key can be separate for X and Y axis using cells {'%.1f','%.4f'}; if empty, than format is not changed (font size only).

fntSize – font size. Function Example:

```
>> gMapTickLabel(7,'%.2f',12);
```

4.3 Set Tips data

function output_txt=gMapTipsLabel (~,event_obj,varargin)

Create tips with current position and additional data.

Parameters:

obj – currently not used (empty);

event_obj – handle to event object;

varargin – additional data for tips' text creation (row numbers with length equal to points number);

output_txt – data cursor text string (string or cell array of strings).

Function Example:

```
>> figure(100);dcm_obj=datacursormode(100);set(dcm_obj,'UpdateFcn',{@gMapTipsLabel});
```