3D/2D modelling suite for integral water solutions

DELFT3D

Deltares systems

DRAFT

MATLAB

Deltares

Enabling Delta Life

User Manual

Delft3D-MATLAB

Interface to MATLAB for flexibility in visualisation and data analysis

User Manual

Hydro-Morphodynamics & Water Quality

Version: 2.15
SVN Revision: 52614

April 18, 2018

Delft3D-MATLAB, User Manual

Published and printed by:

Deltares
Boussinesqweg 1
2629 HV Delft
P.O. 177
2600 MH Delft
The Netherlands

telephone: +31 88 335 82 73
fax: +31 88 335 85 82
e-mail: info@deltares.nl
www: https://www.deltares.nl

For sales contact:

telephone: +31 88 335 81 88
fax: +31 88 335 81 11
e-mail: software@deltares.nl
www: https://www.deltares.nl/software

For support contact:

telephone: +31 88 335 81 00
fax: +31 88 335 81 11
e-mail: software.support@deltares.nl
www: https://www.deltares.nl/software

Contents

List of Figures

1 Introduction

The MATLAB interface for Delft3D has been developed to expand the flexibility in the processing of Delft3D data in general and of NEFIS data files in particular. The interface consists of three parts:

◇ First, low level routines to access data stored in NEFIS files. The routines are described in the second chapter of this manual. A thorough knowledge of the Delft3D system is required.

◇ Second, high level routines to access the data. These functions support a pre-defined (but extensive) set of spatial data stored in the NEFIS files. Routines for reading and writing flow restart files, grid and depth files are also provided. These routines are described in Chapter 3.

◇ Third, a graphical user interface for basic plotting. The QUICKPLOT interface is described in Chapter 4.

All functions are provided in source code. The routines described in chapters three and four can be used as examples for development of other routines.

1.1 Version information

This manual describes the functionality of Delft3D-MATLAB interface version 2.14 and later versions with minor revisions.

1.2 Compatibility

The routines have been tested for compatibility with MATLAB versions ranging from 6.5 (R13) until 2009b. Recent MATLAB versions give some warnings because 'v6' compatibility arguments are used in `quiver` and `contourf` calls; these warnings do not influence the results.

1.3 List of major changes

Version	Description
2.15	Function to process Delft3D-FLOW mdf file Suport for more Delft3D-FLOW attribute files New function to convert cell centred data to cell corners
2.14	New functions for tidal analysis, horizontal slices and vertical averaging MATLAB version before 6.5 no longer supported.
2.07	Update of QUICKPLOT interface to match standalone version 2.07 of Delft3D-QUICKPLOT
2.05	Update of QUICKPLOT interface to match standalone version 2.05 of Delft3D-QUICKPLOT

2 Low level routines

A set of MATLAB files has been created to access NEFIS files from within MATLAB. The following functions have been implemented:

`vs_use`	open NEFIS file
`vs_disp`	display information of groups and elements contained in a NEFIS file
`vs_let/vs_get`	get data from a NEFIS file
`vs_type`	determine type of a NEFIS file
`vs_find`	find group to which an element belongs
`vs_diff`	find differences between two NEFIS files

If no NEFIS file is specified when using the commands `vs_disp`, `vs_let`, `vs_get`, `vs_type` and `vs_find` the last opened NEFIS is used. The commands are discussed below in more detail.

Remark:
  ◇ Spatial data sets are stored in the NEFIS file with the M and N dimensions reversed.

2.1 Open NEFIS file: `vs_use`

Purpose
Open/scan NEFIS file

Syntax

```
Nfs = vs_use(FileName)
```

Description
`Nfs = vs_use(FileName)` opens a NEFIS file where the filename is the name of either the definition or the data file, and returns a handle `Nfs` to the file, which must be used when calling the other commands. When no filename is specified, the command will ask for one.

The `vs_use` command scans the structure of the definition and data files; this will take some time. MATLAB can store the structure information in a file with the same name base (as indicated by filename) and extension $<*$.mat$>$, and read that information when the file is re-opened a next time. This used to be the default behaviour but is now optional: use option usemat. The old option nomat is still accepted and overrules usemat. If changes were made to the $<*$.def$>$ and/or $<*$.dat$>$ files, you should add refresh as additional argument to prevent reading the existing mat-file. The addition of the argument quiet as last argument reads the file structure without showing the wait bar. When the option debug is used a file $<$vs_use.dbg$>$ is created in the default temporary directory, containing information on the reading of the data structure from the NEFIS file.

The returned `Nfs` can be interpreted as a file handle as used in all programming environments for file access. However, it is not an integer but a structure containing the internal structure of the NEFIS file.

Example

```
>> Nfs=vs_use
```

```
Nfs =

    FileName: 'I:\OCTOPUS\sbod_gf3.66_tot\output\TS11520000\com-d16'
      DatExt: '.dat'
      DefExt: '.def'
      Format: 'b'
      gNames: [21x368 char  ]
       gData: [21x27  double]
      eNames: [89x112 char  ]
       eData: [89x11  double]
```

2.2 Display information of groups and elements: vs_disp

Purpose
Display NEFIS file contents.

Syntax

```
vs_disp(Nfs)
Output = vs_disp(Nfs)
vs_disp(Nfs, [])
Output = vs_disp(Nfs, [])
vs_disp(Nfs, GroupName)
Output = vs_disp(Nfs, GroupName)
Output = vs_disp(Nfs, GroupName, [])
Output = vs_disp(Nfs, GroupName, ElementName)
```

Description
vs_disp(Nfs) lists all group names, group elements, and all group and element prop-
erties like the disp stat command of the Viewer Selector. vs_disp(Nfs, []) lists
group names, group dimensions, and number of elements per group only. This can be used
as a short summary of the file contents. Output = vs_disp(Nfs) and Output =
vs_disp(Nfs, []) return a char matrix containing the names of the groups.

vs_disp(Nfs, GroupName) lists all group elements, and all element properties like the
disp GroupName command of Viewer Selector. Output = vs_disp(Nfs, GroupName)
returns a char matrix containing the names of the elements of the group.

Output = vs_disp(Nfs, GroupName, []) gives detailed data about the specified
group contained in the Nfs structure. The information includes group data name, cell name,
definition name, number of elements, dimensions, and the offsets within the NEFIS file at
which locations data of the group are stored. Output = vs_disp(Nfs, GroupName,
ElementName) gives detailed data about the specified element contained in the Nfs struc-
ture. The information includes element description, unit and size.

Examples

```
>> vs_disp(Nfs)
Groupname:BOTNT          Dimensions:(1)
  No attributes
    NTBOT          INTEGER * 4              [  -  ]        ( 1 )
        Number of bottom fields in group BOTTIM

Groupname:BOTTIM         Dimensions:(24)
  No attributes
```

```
    TIMBOT          INTEGER * 4            [ TSCALE]       ( 1 )
        Communication times bottom fields rel. to reference date/time
    DP              REAL   * 4            [  M   ]       ( 98 62 )
        Bottom depth in bottom points, positive downwards
```

and so on ...

```
>> vs_disp(Nfs,[])
BOTNT           (1) - No attributes, 1 element(s).
BOTTIM          (24)  No attributes, 2 element(s).
PARAMS          (1) - No attributes, 8 element(s).
GRID            (1) - No attributes, 16 element(s).
SPECPOINTS      (1) - No attributes, 4 element(s).
BOUNDCNST       (1) - No attributes, 8 element(s).
KENMCNST        (1) - No attributes, 3 element(s).
INITBOT         (1) - No attributes, 4 element(s).
ROUGHNESS       (1) - No attributes, 3 element(s).
TEMPOUT         (1) - No attributes, 4 element(s).
com-version     (1) - No attributes, 3 element(s).
KENMNT          (1) - No attributes, 1 element(s).
KENMTIM         (1) - No attributes, 3 element(s).
CURNT           (1) - No attributes, 1 element(s).
CURTIM          (1) - No attributes, 7 element(s).
DWQTIM          (1) - No attributes, 5 element(s).
TAUTIM          (1) - No attributes, 1 element(s).
INITI           (1) - No attributes, 1 element(s).
TRANNT          (1) - No attributes, 2 element(s).
BOUNDMOR        (1) - No attributes, 1 element(s).
TRANSTIM        (1) - No attributes, 11 element(s).


>> Output=vs_disp(Nfs) %or Output=vs_disp(Nfs,[])

Output =

BOTNT
BOTTIM
PARAMS
GRID
SPECPOINTS
BOUNDCNST
KENMCNST
INITBOT
ROUGHNESS
TEMPOUT
com-version
KENMNT
KENMTIM
CURNT
CURTIM
DWQTIM
TAUTIM
INITI
TRANNT
BOUNDMOR
TRANSTIM


>> vs_disp(Nfs,'BOTTIM')
Groupname:BOTTIM         Dimensions:(24)
  No attributes
    TIMBOT          INTEGER * 4            [ TSCALE]       ( 1 )
```

```
        Communication times bottom fields rel. to reference date/time
    DP              REAL   * 4          [  M  ]        ( 98 62 )
        Bottom depth in bottom points, positive downwards


>> Output=vs_disp(Nfs,'BOTTIM')

Output =

TIMBOT
DP


>> Output=vs_disp(Nfs,'BOTTIM',[]) % get group information

Output =

              Name: 'BOTTIM'
    GroupDatOffset: 8560
           DefName: 'BOTTIM'
    GroupDefOffset: 12728
          CellName: 'BOTTIM'
     CellDefOffset: 12660
         CellNByte: 24308
          CellNElm: 2
              NDim: 1
            VarDim: 1
           SizeDim: 24
          OrderDim: 1
            Attrib: [1x1 struct]


>> Output=vs_disp(Nfs,'BOTTIM','DP') % get element information

Output =

         GrpNname: 'BOTTIM'
          ElmName: 'DP'
      ElmQuantity: ''
         ElmUnits: '[   M   ]'
    ElmDescription: 'Bottom depth in bottom points, positive downwards'
      ElmDefOffset: 12496
             NDim: 2
          SizeDim: [98 62]
          TypeVal: 5
         NByteVal: 4
            NByte: 24304
```

2.3   Get data from a NEFIS file: `vs_let`/`vs_get`

Purpose
Read data from file (groups as columns or cells).

Syntax

```
[Data, Succes] = vs_let(Nfs, GroupName, GroupIndex, ElementName, ElementIndex)
[Data, Succes] = vs_get(Nfs, GroupName, GroupIndex, ElementName, ElementIndex)
```

Description

```
Data = vs_let(Nfs, GroupName, GroupIndex, ElementName, ElementIndex)
Data = vs_get(Nfs, GroupName, GroupIndex, ElementName, ElementIndex)
```

These commands read the requested element from the specified group from the specified file. The `GroupIndex` should be a 1xN cell array, where N equals the number of dimensions of the group. Each element of the cell array should contain either 0 (zero) or the indices of the cells that should be read. The 0 means that all cells should be read. The `ElementIndex` should be a 1xM cell array, where M equals the number of dimensions of the element. Again, each element of the cell array should contain either 0 (zero) or the indices of the elements that should be read, where 0 means that the complete dimension should be read. When the option debug is used a file <vs_let.dbg> is created in the default temporary directory, containing information on the reading of the data from the NEFIS file.

The command `vs_let` reads all data in one big matrix, using the first dimensions for the groups and the last dimensions for the elements. The command `vs_get` returns a matrix containing the data if data from only one group is read, and a cell array of matrices when data from more than one group is read. An additional output option succes `[Data,Succes] = ...` indicates whether the data has been read successfully.

When the `GroupIndex` and/or `ElementIndex` are skipped all fields of the group/element are selected. When no `GroupName` or `ElementName` is specified or when an error is encountered while interpreting the input a graphical user interface is presented to enter the selection. To help new users to get acquainted to the command line syntax of the `vs_get` and `vs_let` routines, the command option line −cmdhelp is provided. When used, the routine does not actually load the data, but it displays the command line equivalent of the selection that you have made in the user interface.

Examples
For example: To read from the depth data the first 10 rows (and of those rows all columns) of the 1st, 3rd, 5th, 7th, and 9th time steps, you should type:

```
>> Data=vs_let(Nfs,'BOTTIM',{1:2:10},'DP',{1:10 0});
```

which returns a 5x10x62 matrix; or

```
>> Data=vs_get(Nfs,'BOTTIM',{1:2:10},'DP',{1:10 0})

Data =

    [10x62 double]
    [10x62 double]
    [10x62 double]
    [10x62 double]
    [10x62 double]
```

Indicating that a 5 x 1 cell array is returned, each cell containing one bottom field (10 x 62 data points). Character strings are contained in cell arrays. The most common cases of reading from are:

```
Data=vs_get(Nfs,'BOTTIM',{0},'DP',{0 0})
```

Read all bottom data completely. Group/Element indices containing only zeros don't have to be specified. So, the following is also accepted:

```
Data=vs_get(Nfs,'BOTTIM','DP')

Data=vs_get(Nfs,'BOTTIM',{0},'DP',{n 0}), or
Data=vs_get(Nfs,'BOTTIM','DP',{n 0})
```

Read cross-section n at all time steps

```
Data=vs_get(Nfs,'BOTTIM',{t},'DP',{0 0}), or
Data=vs_get(Nfs,'BOTTIM',{t},'DP')
```

Read the bottom at time step t. This will return a 10 x 62 matrix if `vs_get` is used, and a 1x10x62 matrix if `vs_let` is used.

```
Data=vs_get(Nfs,'BOTTIM',{0},'DP',{n m}), or
Data=vs_get(Nfs,'BOTTIM','DP',{n m})
```

Read the variation of the bottom at indicated (n,m)-point. If this is followed by a `Data = [Data:];` command to convert the cell array containing single values into a vector; it can be replaced by

```
Data=vs_let(Nfs,'BOTTIM','DP',{m n})
```

which gives that result immediately and is faster.

When besides the `Nfs` no further arguments are used (or when no element name is specified), an interface will appear asking for the other arguments. Adding the string `quiet` as last input argument will suppress the wait bar during loading. When `'*'` is specified as element name all elements of the group are returned; if an element index is specified it is ignored.

Example

```
Data=vs_let(Nfs,'BOTTIM',{1:2:10},'*')
```

returns

```
Data =

   TIMBOT: [5x1     double]
       DP: [5x10x62 double]
```

and

```
Data=vs_get(Nfs,'BOTTIM',{1:2:10},'*')
```

returns the data of five time steps

```
Data =

5x1 struct array with fields:
    TIMBOT
    DP
```

To assist the novice user with the syntax of the `vs_let` and `vs_get` commands a help functionality has been added to these functions which allows the interactive generation of command lines. This functionality is activated by supplying `-cmdhelp` as one of the input arguments. For instance:

```
>> vs_let -cmdhelp
```

results after the following data has been entered in the user interface

in the following output string

```
Data=vs_let(NFStruct,'BOTTIM',{[ 1:2:9 ]},'DP',{[ 1:10 ],0});
```

You can copy this string into a text editor for use in a script. As indicated above, variations on the syntax are allowed. The following line is also valid

```
Data=vs_let('BOTTIM',{1:2:9},'DP',{1:10,1:73});
```

The result will be exactly the same provided that the variable NFStruct in the first command line contains the information of the last opened NEFIS file.

2.4   Determine the type of a NEFIS file: `vs_type`

Purpose
Show type of NEFIS file.

Syntax

```
Type=vs_type(Nfs)
```

Description
The function returns the type of the loaded NEFIS file. Currently the following NEFIS files are detected: 'Delft3D-com', 'Delft3D-trim', 'Delft3D-trih', 'Delft3D-trid', 'Delft3D-tram', 'Delft3D-trah', 'Delft3D-botm'. Other NEFIS files will return 'unknown'. For example:

Example

```
>> Type=vs_type(Nfs)

Type =

Delft3D-com
```

2.5 Find group to which element belongs: `vs_find`

Purpose
Find elements in a NEFIS file.

Syntax

```
Groups=vs_find(Nfs, ElementName)
```

Description
The function returns a list of group names containing the specified element. For example the element 'TIMCUR' occurs in several groups of the com-file:

Example

```
>> Groups=vs_find(Nfs,'TIMCUR')

Groups =

KENMTIM
CURTIM
DWQTIM
```

2.6 Find differences between two NEFIS files: `vs_diff`

Purpose
Compare NEFIS file contents and find differences in groups, elements and data.

Syntax

```
vs_diff(Nfs1,Nfs2)
```

Description
The function analyses the specified NEFIS files for differences. The analyses searches first for differences in the contained groups. If it finds differences it stops, if not it continues to search for differences in the elements, element properties, and element data.

Example
The following comparison between two com-files, of which the first was obtained by a restart run from the second one, indicates

◇ differences in the water and sediment transport data fields;
◇ a difference in the group properties of the BOTTIM group (which is caused by one additional depth field); and
◇ differences in the 'RUNTXT' and 'SIMDAT' texts.

```
>> vs_diff(Nfs,Nfs2)
Comparing NEFIS files ...
File 1: I:\OCTOPUS\sbod_gf3.66_tot\output\TS11520000\com-d16
File 2: I:\OCTOPUS\sbod_gf3.66_tot\output\TS11160000\com-d16
```

```
Data of element NTBOT of group BOTNT differ.
Group properties differ for: BOTTIM
Data of element QU of group CURTIM differ.
Data of element QV of group CURTIM differ.
Data of element S1 of group CURTIM differ.
Data of element TIMCUR of group CURTIM differ.
Data of element U1 of group CURTIM differ.
Data of element V1 of group CURTIM differ.
Data of element TIMCUR of group DWQTIM differ.
Data of element KFU of group KENMTIM differ.
Data of element KFV of group KENMTIM differ.
Data of element TIMCUR of group KENMTIM differ.
Data of element TAUMAX of group TAUTIM differ.
Data of element TSEDB of group TRANSTIM differ.
Data of element TSEDE of group TRANSTIM differ.
Data of element TTXA of group TRANSTIM differ.
Data of element TTYA of group TRANSTIM differ.
Data of element RUNTXT of group com-version differ.
Data of element SIMDAT of group com-version differ.
... comparison finished.
```

3 High level routines

Although the low level routines provide access to all data in the NEFIS files, obtaining a useful, consistent data set can be difficult. For instance, to plot a 2D vector field you have to combine the information stored at about seven locations in the Delft3D communication file. Therefore, a number of high level routines has been provided to simplify the plotting and data retrieval operations. Routines are provided for plotting a grid (with grid number labels), plotting thin dams and inactive velocity points, and retrieving data to plot velocity vectors. Functions are also provided for reading and writing RGFGRID grid files (RGFGRID manual (RGFGRID UM, 2016)), QUICKIN depth files (QUICKIN manual (QUICKIN UM, 2013)), and Delft3D-FLOW restart files (Delft3D-FLOW manual (Delft3D-FLOW UM, 2013)).

3.1 Plot a grid: `drawgrid`

Purpose
Plot the morphological grid.

Syntax

```
drawgrid(NFStruct)
[X, Y] = drawgrid(NFStruct);
drawgrid(X, Y)
```

Description
`drawgrid(NFStruct)` where `NFStruct` is the structure obtained from the `vs_use` command. The function adds the morphological grid (that is, the user-defined grid) to the current axes. <trim-*>, <tram-*>, <botm-*>, and <com-*> files are supported. The morphological grid equals the grid specified in the input file; the bed level points are specified on the grid line crossings.

`[X, Y] = drawgrid(NFStruct)` returns the X and Y co-ordinates, but it does not actually draw the morphological grid.

`drawgrid(X, Y)` draws the user specified grid in the current axes.

This function accepts a number of optional arguments, which consist of property - value pairs similar to those use by the MATLAB `set` command: `drawgrid(..., 'optionname1', optionval1, 'optionname2', optionval2, ...)`. The supported options are

'color'	The value can be an RGB-triplet, a colour character ('r', 'g', 'b', 'c', 'm', 'y', 'k', 'w'), or the string 'ortho' for colouring based on the orthogonality of grid, 'msmo' for the M-smoothness of grid, 'nsmo' for the N-smoothness of grid
'm1n1'	$[NM]$ number of grid point(1,1). Useful when plotting only part of the grid.
'fontsize'	The size of the font used for grid numbering (default 4)
'gridstep'	Label step used for grid numbering (default 10). For instance, 5 will number every fifth grid line. If grid step is [], grid lines are not labelled.
'thicklength'	Length of ticks in axes co-ordinates or 'auto' (default)
'parent'	The axes handle in which to plot grid (default the current axes, gca)
'clipzero'	By default co-ordinate pairs of (0,0) are clipped. Set to 'off' to plot co-ordinates at (0,0))
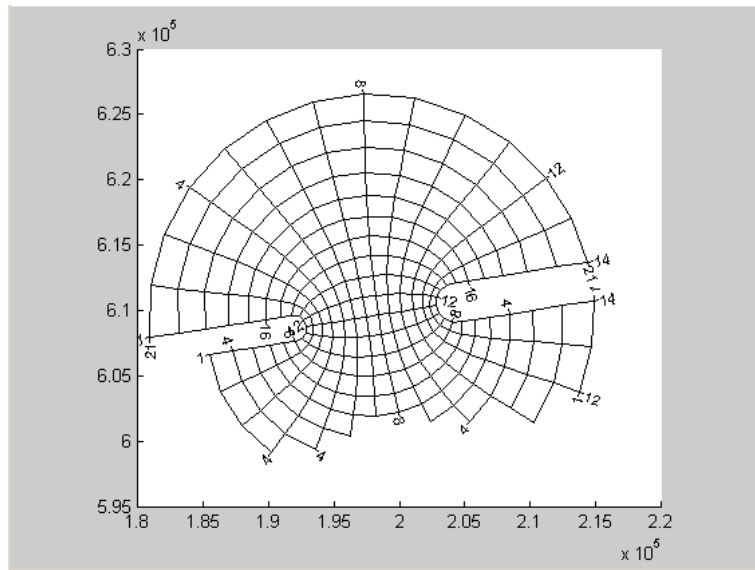
Figure 3.1: Grid layout from the Friesian Tidal Inlet model.

Example, Figure 3.1

```
>> G=wlgrid('read','d:\f-disk\users\mooiman\delft3d\models\f34\f34\f34.grd')

G =

                X: [14x21 double]
                Y: [14x21 double]
        Enclosure: [19x2 double]
         FileName: 'D:\delft3d\tutorial\flow\f34_demo\fti_02.grd'
 CoordinateSystem: 'Cartesian'
     MissingValue: 0
             Type: 'RGF'


>> drawgrid(G.X,G.Y,'fontsize',8,'gridstep',4)
>> set(gca,'dataaspectratio',[1 1 1])
```

3.2 Plot thin dams: `thindam`

Purpose
Plot dams, weirs and vanes.

Syntax

```
thindam(NFStruct, T)
thindam(NFStruct, T, '3D')
thindam(XCOR, YCOR, UDAM, VDAM)
thindam('xyw', XDAM, YDAM, WDAM)
thindam(..., 'PropertyName', PropertyValue)
H = thindam(...)
[X, Y] = thindam(...)
[X, Y, M, N] = thindam(...)
[X, Y, Z] = thindam(...)
[X, Y, BOTTOM, TOP] = thindam(...)
[X, Y, BOTTOM, TOP, M, N] = thindam(...)
```

Description

`thindam(NFStruct,T)` plots the KFU/V (drying/flooding) dams for time step T if T>0. It plots the KCU/V thin dams if T==0 (default if T is not specified). The function supports <trim-*> and <com-*> files.

`thindam(NFStruct, T, '3D')` same as above but now the dams will be plotted as 3D surfaces based on the bottom and water level information in the specified NEFIS file, <trim-*> or <com-*> file.

`thindam(XCOR, YCOR, UDAM, VDAM)` plots the specified U and V dams on the specified grid. The XCOR and YCOR represent the bottom points. Valid entries for UDAM and VDAM are:

◇ matrix of same size as XCOR/YCOR with a 1 representing a dam and a 0 indicating no dam.
◇ D x 2 matrix specifying the N, M co-ordinates of the dams
◇ D x 4 matrix specifying the begin/end N, M co-ordinates of the dams like in the Delft3D input files.

`thindam('xyw', XDAM, YDAM, WDAM, ...)` plots dams with their centre at specified (XDAM, YDAM) locations with specified width/length WDAM. This function is useful for non-grid oriented dams.

A 3D effect can be added to these user-specified dams by specifying bottom and top levels using `thindam(..., 'bottom', <bottom_args>, 'top', <top_args>)`. It plots the dams as 3D surfaces with specified top and bottom elevations. The elevation data should be specified with positive direction up. Additional supported options are:

◇ `'angle', <angle_args>` The dams/vanes are rotated to match the specified angle in degrees with respect to the positive X-axis (positive angle in anti-clockwise direction).
◇ `'color', <color_args>` The dams are coloured using the specified data.
◇ `'thickness', <thickness_args>` The thickness of the dams is specified (default 0).

Valid entries for all options are:

◇ a constant, uniform value for all dams
◇ a matrix of same size as XCOR/YCOR containing data in the depth points.
◇ a matrix of same size as XCOR/YCOR containing data in the water level points. To distinguish this entry from the former you need to add the string 'H' or 'S' as an extra argument after the matrix. For example `thindam(..., 'top', ELEVMATRIX, 'H', ...)` indicates that the top level of the dams are derived from the specified data in water level points (the value will be used uniformly along each elementary dam).
◇ two matrices of same size as XCOR/YCOR containing data in the U and V points.
◇ two D x s arrays specifying the property of the individual dams. If the array is D x 1 the value is constant along each elementary dam. If the array is D x 2 the first value is taken for the dam end with lowest (N,M) while the second elevation for the dam end with highest (N,M). The first array specifies the elevation for dams in U direction the second array for dams in the V direction. For example `thindam(..., 'top', ELEVU, ELEVV, ...)`. This option cannot be used in combination with option 1 for the UDAM and VDAM entries. The number of values should match the number of dam records/elementary dams.

If a colour data field has been specified, there are a few more options available:

◇ `'shape',<type>`
The dam shape: 'dam' (default) or 'rhombus'.
◇ `'drawdams',<onoff>`
Flag to draw dams. Can be set to 'on' (default) or 'off'.
◇ `'drawlabels',<onoff>`
Flag to draw the colour values as text labels. Can be set to 'on' or 'off' (default).
◇ `'labelformat',<format>`
Defines the format for displaying the values (default '%g')
◇ `'fontsize',<size>`
Defines the size of the labels.

If the 'xyw' syntax is used, then valid entries for the options are

◇ a constant (uniform value for all dams), and
◇ a matrix of same size as XDAM/YDAM/WDAM.

`H = thindam(...)` returns the handle of the line/surface object used for plotting the dams.

`[X, Y] = thindam(...)` returns the x,y-arrays normally used for plotting. The dams are not plotted. This syntax is only valid if the input arguments indicate a 2D dam plot.

`[X, Y, M, N] = thindam(...)` returns four nx2 matrices containing per row a pair of end points for an elementary dam (X and Y co-ordinates, M and N index). This syntax is only valid if the input arguments indicate a 2D dam plot.

`[X, Y, Z] = thindam(...)` returns the x,y,z-arrays normally used for plotting. The dams are not plotted. This syntax is only valid if the input arguments indicate a 3D dam plot.

`[X, Y, BOTTOM, TOP] = thindam(...)` returns four nx2 matrices containing per row a pair of end points for an elementary dam: X co-ordinates, Y co-ordinates, bottom elevation and top elevation. This syntax is only valid if the input arguments indicate a 3D dam plot.

`[X, Y, BOTTOM, TOP, M, N] = thindam(...)` returns six nx2 matrices containing per row a pair of end points for an elementary dam: X co-ordinates, Y co-ordinates, bottom elevation, top elevation, M index, N index. This syntax is only valid if the input arguments indicate a 3D dam plot.

Example, Figure 3.2

```
>> vs_use D:\wl\delft3d\tutorial\wave\Siu-Lam\Com-siu
>> drawgrid('color','g')
>> set(gca,'DataAspectRatio',[1 1 1])
>> thindam(1)
>> l=thindam; set(l,'color','r')
```
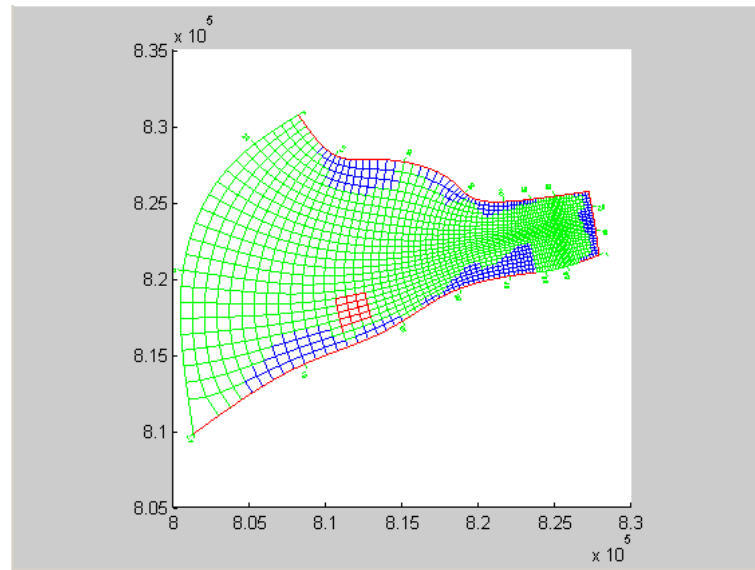
Figure 3.2: Example of grid and thin dams.

3.3 Read/write restart file: `trirst`

Purpose
Read/write Delft3D-FLOW restart file.

Syntax

```
[H, U, V, ...] = trirst('read', FILENAME, SIZE)
[H, U, V, ...] = trirst('read', FILENAME, GRID)
[H, U, V, ...] = trirst('read', FILENAME, GRID, NLAYERS)
trirst('write', FILENAME, H, U, V, ...)
trirst('write', FILENAME, PLATFORM, H, U, V, ...)
```

Description
`[H, U, V, ...] = trirst('read', FILENAME, SIZE)` and `[H, U, V, ...] = trirst('read', FILENAME, GRID)` where GRID was read in from the `<*.grd>` file by WLGRID read data from a Delft3D-FLOW restart file of a 2D simulation. The dimensions of the underlying grid must be specified. This can be done either by manually specifying the grid size as [M N], or by automatic derivation from a grid.

`[H, U, V, ...] = trirst('read', FILENAME, GRID, NLAYERS)` reads data from a Delft3D-FLOW restart file of a 3D simulation. The NLAYERS arrays indicate the number of layers used for the various variables. In the example syntax: `NLAYERS = [NLAYERH, NLAYERU, NLAYERV, ...]`.

`trirst('write', FILENAME, H, U, V, ...)` writes the specified matrices to the indicated restart file. Because the restart file is stored in a platform dependent manner, care should be taken to write the appropriate type of restart file. By default the routine writes the data in the format compatible with the computer on which MATLAB is running. A different platform can be chosen by using the following syntax: `trirst('write', FILENAME, PLATFORM, H, U, V, ...)` where PLATFORM can be 'pc' or 'unix'. Both Windows and Linux versions of Delft3D use the byte order indicated by 'pc'.

3.4 **Read/write depth file:** `wldep`

Purpose
Read/write Delft3D field files (e.g. depth files).

Syntax

```
DEPTH = wldep('read', FILENAME, SIZE)
DEPTH = wldep('read', FILENAME, GRID)
STRUCT = wldep(..., 'multiple')
[FLD1, FLD2, ...] = wldep(..., 'multiple')
wldep('write', FILENAME, MATRIX)
```

Description
`wldep` can be used to read and write Delft3D field files used for specifying depth and rough-ness data.

`DEPTH = wldep('read', FILENAME, SIZE)` or `DEPTH = wldep('read', FILENAME, GRID)` where the dimensions [M N] of the data is either user-specified or derived from the GRID data set that was obtained from `wlgrid`. `STRUCT = wldep(..., 'multiple')` to read multiple fields from the file (e.g. in case of a roughness file). The function returns a structure vector with one field: Data. Use `[FLD1, FLD2, ...] = wldep(..., 'multiple')` to read multiple fields from the file and store the result in separate output arguments.

Use `wldep('write', FILENAME, MATRIX)` or `wldep('write', FILENAME, STRUCT)` to write data to a $<*$.dep$>$ file. Multiple data sets can be written by specifying a structure vector STRUCT with one field data.

Example, Figure 3.3

```
>> G=wlgrid('read','D:\wl\delft3d\tutorial\mor\ymu\r1-nri.grd');
Warning: Grid enclosure not found.
>> D=wldep('read','D:\wl\delft3d\tutorial\mor\ymu\r1-nri.dep',G);
>> surf(G.X,G.Y,-D(1:end-1,1:end-1));
>> shading flat
>> view(-60,30)
>> set(gca,'da',[1 1 1/300])
>> camlight
```
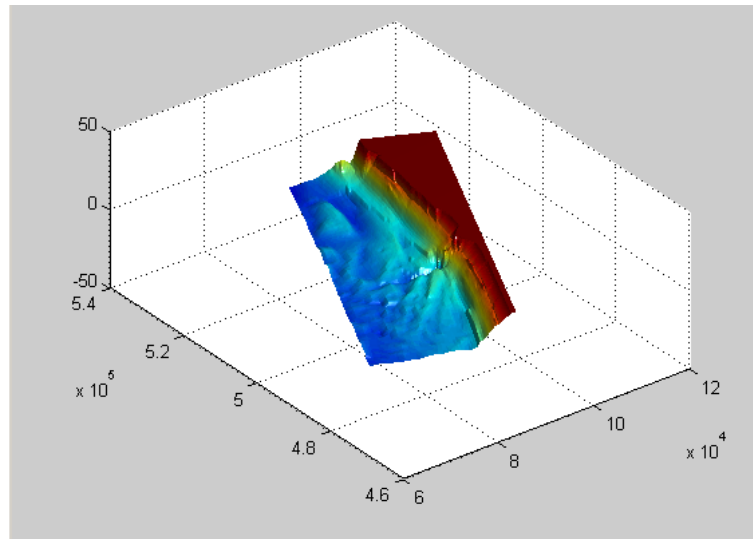
3.5 **Read/write grid file:** `wlgrid`

Purpose
Read/write a Delft3D-FLOW grid file.

Syntax

```
[X,Y,ENC] = wlgrid('read', FILENAME)
GRID = wlgrid('read', FILENAME)
OK = wlgrid('write', FILENAME, X, Y, ENC)
```

Figure 3.3: Example of 3D depth plot.

Description

`[X, Y, ENC] = wlgrid('read', FILENAME)` reads the grid and enclosure from the Delft3D <∗.grd> and <∗.enc> files. X, Y, and enclosure are returned as separate output arguments. `GRID = wlgrid('read', FILENAME)` returns a structure with X, Y, and Enclosure fields.

`OK = wlgrid('write', FILENAME, X, Y, ENC)` writes the GRID to <∗.grd> and <∗.enc> files that can be used by Delft3D. If the last argument (ENC) is not specified, the enclosure file is not written.

3.6 Read/plot velocity field: `xyveloc`

Purpose

Read velocity data from a <trim-*> or <com-*> file.

Syntax

```
[U, V] = xyveloc(NFStruct, TimeStep)
[U, V, W] = xyveloc(NFStruct, Index)
[X, Y, U, V] = xyveloc(NFStruct, TimeStep)
[X, Y, Z, U, V, W] = xyveloc(NFStruct, Index)
[...] = xyveloc(..., 'option')
```

Description

`[U, V] = xyveloc(NFStruct, TimeStep)` reads the velocity field at the specified time step from the specified NEFIS file. By default TimeStep is the last field of the CURTIM or map-series group. The default NEFIS file is the last opened NEFIS file.

`[X, Y, U, V] = xyveloc(NFStruct, TimeStep)` returns not only the velocities, but also the co-ordinates of the grid points at which the velocities are given (water level points).

`[U, V, W] = xyveloc(NFStruct, Index)` reads a 3D velocity field (x,y,z components). `[X, Y, Z, U, V, W] = xyveloc(NFStruct, Index)` reads the 3D velocity field (x,y,z components) and returns the 3D co-ordinates of the velocity values.
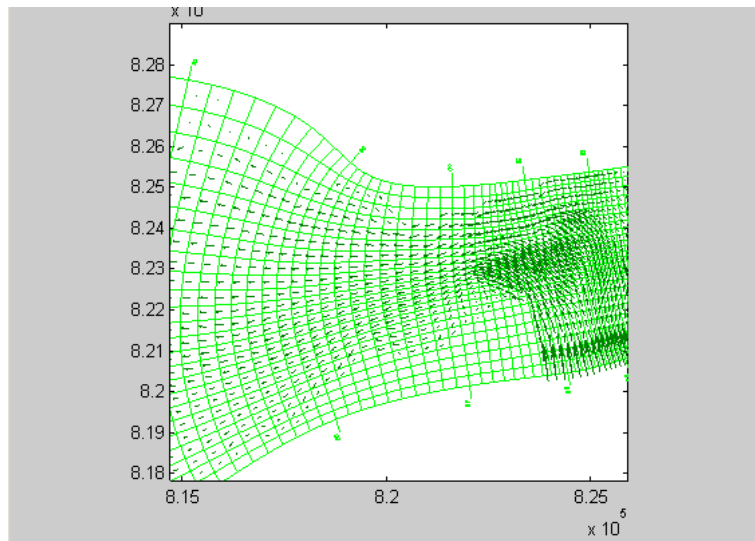
Figure 3.4: Example of vector velocity plot.

`[...] = XYVELOC(..., 'option')` where the string 'option' equals

◇ 'vort': computes the z-component of the vorticity. Only one output argument is required in this case.
◇ 'total', 'fluc', 'mean': reads the total velocity, fluctuation component, or mean velocity field in case of an HLES simulation (trim-file, 2D only). The default setting is 'total'.

Example, Figure 3.4

```
>> vs_use D:\wl\delft3d\tutorial\wave\Siu-Lam\Com-siu
>> [x,y,u,v]=xyveloc;
>> quiver(x,y,u,v)
>> drawgrid('color','g')
>> set(gca,'da',[1 1 1])
>> hold on
>> quiver(x,y,u,v)
```

3.7 Generic read data: `qpread`

Purpose
Get information/data from a data file. Provides access to the functionality of the QUICKPLOT interface.

Syntax

```
[DataFields, Dims, NVal] = qpread(FileInfo)
DataProps = qpread(FileInfo)
Size   = qpread(FileInfo, DataProp, 'size')
Size   = qpread(FileInfo, DataFld, 'size')
Times  = qpread(FileInfo, DataProp, 'times')
Times  = qpread(FileInfo, DataFld, 'times')
StNames = qpread(FileInfo, DataProp, 'stations')
StNames = qpread(FileInfo, DataFld, 'stations')
Data   = qpread(FileInfo, DataProp, 'data', t, station, m, n, k)
```

```
Data     = qpread(FileInfo, DataFld, 'data', t, station, m, n, k)
Data     = qpread(FileInfo, DataProp, 'griddata', t, station, m, n, k)
Data     = qpread(FileInfo, DataFld, 'griddata', t, station, m, n, k)
```

Description

The `qpread` function supports reading data from various data files. Which data sets can be read from a specific data file can be obtained from the function by one of the following two commands:

`[DataFields, Dims, NVal] = qpread(FileInfo)` returns the data fields available from the data file specified using the FileInfo structure. An acceptable FileInfo can be obtained from any of the following three commands: `qpfopen`, `qpfile`, `vs_use`. If FileInfo is not specified the last opened NEFIS file is used (same condition applies to the other calls to `qpread`). The variable DataFields is a cell array of strings containing the names of the data sets in the file. The variable Dims indicates relevant dimensions for the data fields:

First column of Dims:	Time step
Second column of Dims:	Station name
Third column of Dims:	M
Fourth column of Dims:	N
Fifth column of Dims:	K

NVal equals one for scalar data sets, 2 (or 3) for vector data sets.

`DataProps = qpread(FileInfo)` returns a structure containing fields with the above mentioned data and some extra information for internal use by the `qpread` and `d3d_qp` functions.

Based on the information returned by either one of these two `qpread` statements you may select a data field. This data field can be identified using its name provided in the DataFields cell array (DataFieldsi) or its record in the DataProps structure (DataProps(i)). For each data field you can obtain more information by subsequent calls to `qpread`.

Remark:

⬦ You should not request data for the separator fields with name '——'.

$\bigcirc$!

`Size = qpread(FileInfo, DataProp/DataFldName, 'size')` returns a 1 x 5 array containing the maximum size of the five dimensions mentioned above. The second parameter can be either a string (data field name) or a record from the DataProps structure. The latter is necessary if the data field name is not unique.

`Times = qpread(FileInfo, DataProp/DataFldName, 'times')` returns an array of the times for the specified data field if relevant.

`StNames = qpread(FileInfo, DataProp/DataFldName, 'stations')` returns an array of the station names for the specified data field if relevant.

`Data = qpread(FileInfo, DataProp/DataFld, 'data', t, station, m, n, k)` returns the data for the specified data field at the indicated time steps, specified station, and M, N, K co-ordinates. Only those parameters should be specified that have non-zero dimension in Dims and Size. For a data field at a station often only time and station. For spatial data sets not the station but a selection of M, N and K. Following restrictions apply to the retrieval of the data

◇ station: index of the station in the StNames array (scalar).

◇ 0 (all points) or scalar value (m index).

◇ 0 (all points) or scalar value (n index).

◇ 0 (all points) or scalar value (k index).

`Data = qpread(FileInfo, DataProp, 'griddata', t, station, m, n, k)` returns the data and grid for the specified data field at the indicated time steps, specified station, and M, N, K co-ordinates. In both cases the output argument Data is a structure containing a Time field. If the data is scalar, the data is returned in the Val field. If the data is vector data, it is returned as Xcomp and Ycomp fields.

Example, Figure 3.5

```
>> F=qpfopen('D:\wl\delft3d\tutorial\gpp\trim-edw.dat');
>> I=qpread(F);
>> I(9)

ans =

        Name: 'water level'
     DimFlag: [1 0 1 1 0]
   DataInCell: 1
        NVal: 1
     VecType: ''
         Loc: 'z'
      ReqLoc: 'z'
       Loc3D: ''
       Group: 'map-series'
        Val1: 'S1'
        Val2: ''
       SubFld: []
         MNK: 0

>> S=qpread(F,I(9),'griddata')

S =

      X: [87x227 double]
      Y: [87x227 double]
    Val: [87x227 double]
   Time: 7.2942e+005

>> contour(S.X,S.Y,S.Val,50)
>> view(90,90)
>> set(gca,'da',[1 1 100])
```
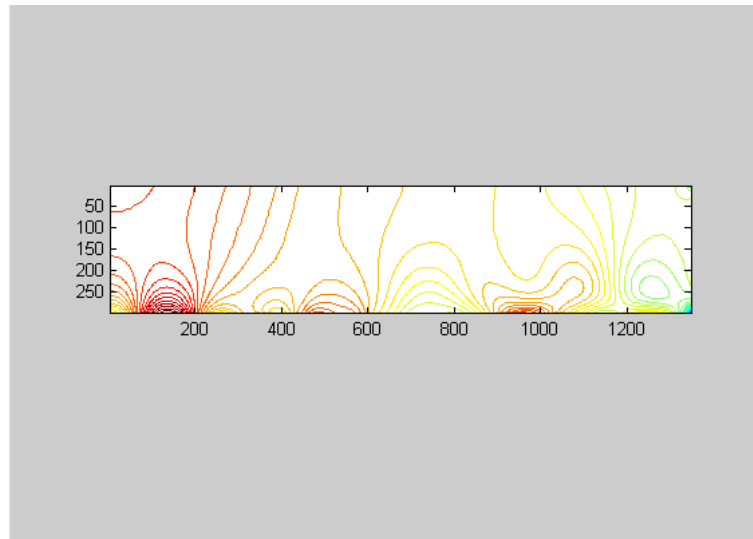
3.8 Generic open file: `qpfopen`

Purpose
Opening a data file. Provides access to the functionality of the QUICKPLOT interface.

Syntax

```
FileInfo=qpfopen('Filename')
FileInfo=qpfopen('DataFile','GridFile')
FileInfo=qpfopen
```

Figure 3.5: Example of contour plot.

Description

The qpfopen functions can be used to open all data files that QUICKPLOT can open (see the appendix of the QUICKPLOT manual for an overview of all file formats).

FileInfo=qpfopen('Filename') opens the specified output file and returns a structure containing data used by the qpread function.

For WAQ or PART <∗.map> files you should also specify an <∗.lga> grid file: FileInfo = qpfopen('DataFile', 'GridFile') If no grid file is specified then the map file is treated as a history file with an observation point at each location.

FileInfo=qpfopen When no arguments are passed to the function, you are asked to specify the file using a standard file selection dialog window. If you try to open a WAQ or PART <∗.map> file, the function asks for the additional grid file.

3.9 Get file opened by QUICKPLOT: qpfile

Purpose

Get information about the currently selected file in QUICKPLOT. Provides access to the functionality of the QUICKPLOT interface.

Syntax

FileInfo=qpfile

Description

To make switching between the user interface of QUICKPLOT and MATLAB's command line easier, the qpfile function provides easy access to the currently selected file in the QUICK-PLOT interface.

FileInfo=qpfile returns the structure of the data file selected in the QUICKPLOT interface.

4 Delft3D-QUICKPLOT functionality

The Delft3D-QUICKPLOT interface can be started by typing `d3d_qp` at the MATLAB command prompt. This interface is intended to simplify the creation of plots with MATLAB. Tooltips provide online help for the interface. The same interface is also available as a standalone program. This chapter gives only a short introduction; for a detailed description of the interface, you are referred to the Delft3D-QUICKPLOT manual (QUICKPLOT UM, 2013).

The interface allows to open NEFIS files and select data fields. After making a selection of time steps, stations and m, n, k indices, you can either load the data (*Load Data* button) or plot the data (by clicking the *Quick View* or *Quick Animate* button).

The *Load Data* button in the lower left corner of the interface is not available in the standalone version, so it is described here. When the data is loaded using the *Load Data* button, a workspace variable called 'data' is created. If the variable 'data' was already defined any data contained in it will be overwritten without confirmation. The variable 'data' contains:

◇ a field X with the x co-ordinates
◇ a field Y with the y co-ordinates
◇ a field Z with the z co-ordinates (for 3D data sets only)
◇ a field Val with the data is the data is scalar
◇ fields XComp, YComp and ZComp when the data is a vector quantity.

The same functionality can be accessed from the command line using the command

```
X=d3d_qp('loaddata')
```

In this case the data is assigned to the user-specified variable X. Many of the other functionalities of Delft3D-QUICKPLOT can also be accessed from the command line. The general syntax of these commands is:

```
d3d_qp(command_string,...command_parameters...)
```

The `command_string` identifies the functionality that should be activated, e.g. 'loaddata', 'openfile' or 'colourbar'. The `command_parameters` depend on the command chosen, e.g. 'loaddata' does not need any parameter, 'openfile' does not need any parameter either but if a parameter is specified then it should be the name of the file to be opened, and 'colourbar' requires one parameter namely a 1 or 0 to indicate whether a colourbar should be drawn or not. The details of all these commands is not described here. The best way to learn these commands is by doing. Therefore, Delft3D-QUICKPLOT has several options to display or export the equivalent commands that it is executing when you click on its buttons. These display options can be accessed from the Macro menu of Delft3D-QUICKPLOT as shown in Figure 4.2.

For exporting/displaying the MATLAB commands associated with the various functionalities of Delft3D-QUICKPLOT, the following three options are available:

◇ Record MATLAB Script. Records the commands directly to a MATLAB .m file.
◇ Record to Message Window, MATLAB Script Commands. Records the commands in a separate window of Delft3D-QUICKPLOT. The commands cannot be copied from this window, but the recorded commands can be saved to a text file.
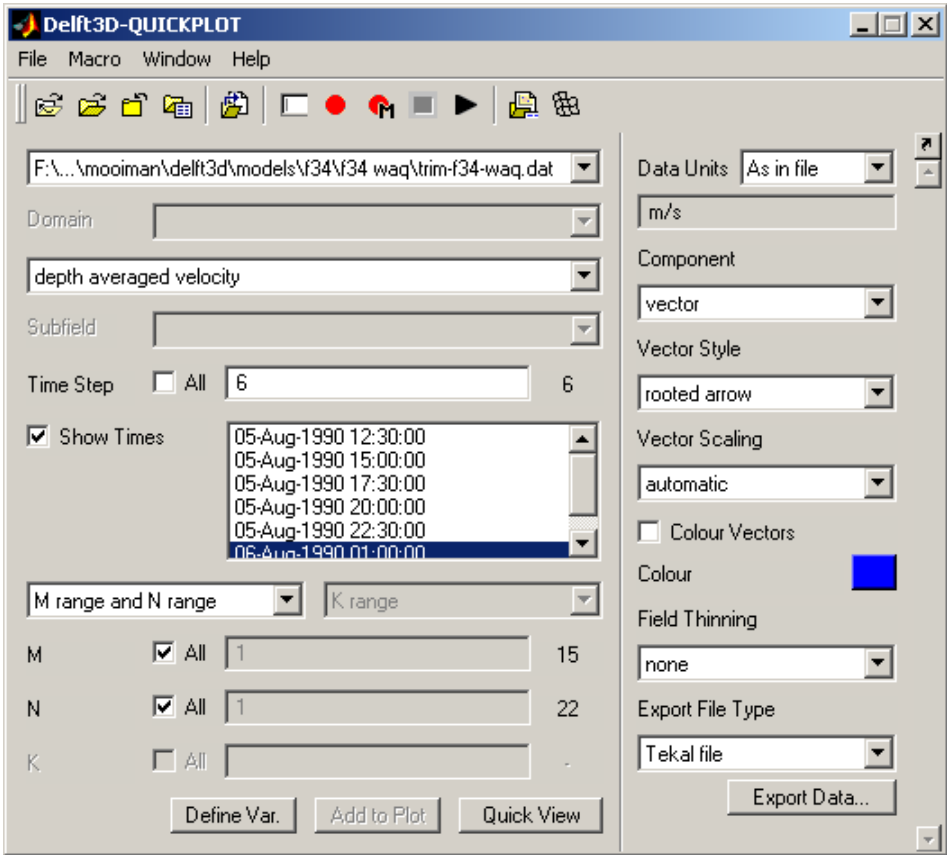
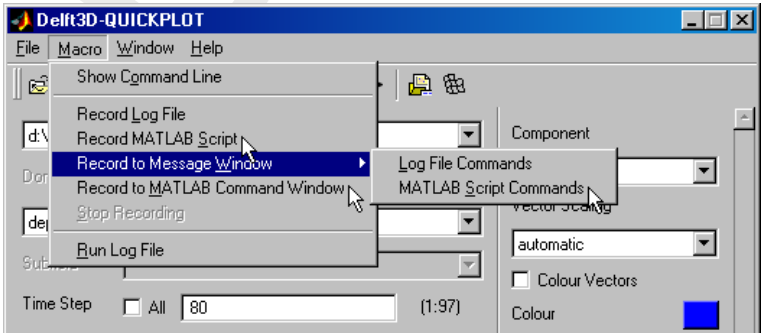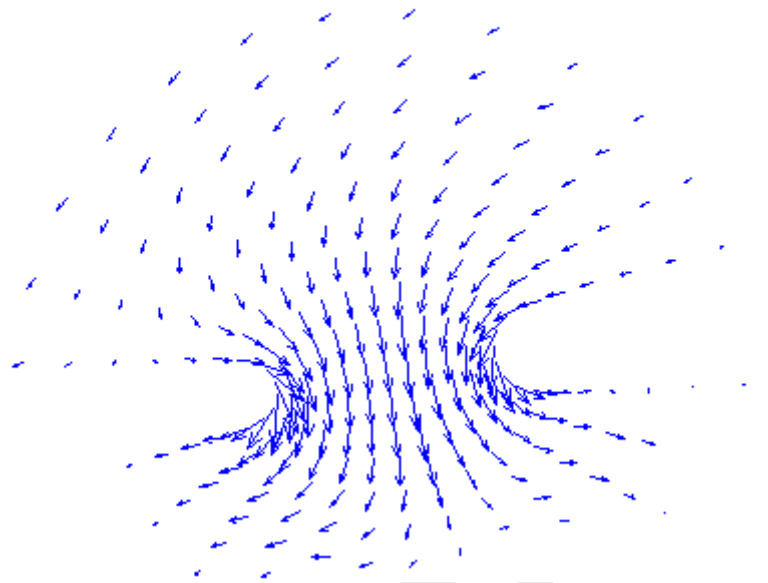*Figure 4.1: Selection window of Delft3D-QUICKPLOT.*

Figure 4.2: Display/export options for QUICKPLOT commands.

Figure 4.3: *Example of Delft3D-QUICKPLOT figure.*

◇ Record to MATLAB Command Window (not available in the standalone version of Delft3D-QUICKPLOT). Echoes the commands to the MATLAB Command Window from which they can be easily copied. When using MATLAB's diary option, the echoed commands will also end up in the diary file.

Both Delft3D-QUICKPLOT log files and MATLAB script files containing Delft3D-QUICKPLOT calls can be run using the Run Log File menu item of Delft3D-QUICKPLOT. The MATLAB script files may also contain other single line MATLAB function calls (not supported in standalone mode). Scripts containing multi-line MATLAB constructs cannot be run from within the Delft3D-QUICKPLOT interface.

When the data is plotted or animated a new figure is opened. Scaling of axes, vector length and colours is automatic. This implies that during an animation colour scaling, vector length scaling and plot extent may vary. The plot extent and colour scaling can be set using the MATLAB commands `axis`, `set`, `gca` and `colorbar`. For example:

```
axis([xmin xmax ymin ymax]) % set the axes extent
```

and

```
set(gca,'clim',[cmin cmax]) % set the colour limits
colorbar % update the colourbar
```

In case of an animation, plotting can be stopped by either closing the figure or selecting the Stop option from the Animation menu in the figure of the plot. When the data set contains multiple time steps, a scroll bar will be plotted in the lower left corner of the plot figure. You can use this to change the time step of the plot.

Remarks:

◇ The figure files saved by the standalone version of Delft3D-QUICKPLOT are compatible with MATLAB 6.5. It does not support files created with newer MATLAB versions.

◇ Access to netCDF files and data on OPeNDAP enabled websites requires separate installation of the 'mexnc' and 'snctools' toolboxes available from
`http://mexcdf.sourceforge.net/`

References

Delft3D-FLOW UM, 2013. *Delft3D-FLOW User Manual*. Deltares, 3.14 ed.

QUICKIN UM, 2013. *Delft3D-QUICKIN User Manual*. Deltares, 4.00 ed.

QUICKPLOT UM, 2013. *Delft3D-QUICKPLOT User Manual*. Deltares, 2.14 ed.

RGFGRID UM, 2016. *Delft3D-RGFGRID User Manual*. Deltares, 5.00 ed.

Deltares systems