

The Warp User and Developer Manual

Arno Swart

11th January 2001

Contents

1	Introduction	2
2	Installation	2
2.1	Installation on Unix systems	2
2.2	Installation on Microsoft Windows systems	2
3	Using Warp	3
3.1	Specifying the geometry	3
3.2	Calculation of the solutions	5
3.3	Visualisation of the solutions.	5
3.3.1	Visualisation of modes	5
3.3.2	Tracing of characteristics	6
3.3.3	Visualisation of the fundamental intervals	6
3.4	Forcing boundary conditions	6
4	Implementation of Warp	8
4.1	Overview	8
4.2	Tracing Characteristics	9
4.3	Grid refinement	11
4.3.1	Brep structure	11
4.3.2	Adding internal edges	13
5	Known bugs	13

1 Introduction

This is the manual for the Warp (**W**ave **A**ttraction **R**eflection and **P**ropagation) software.

Throughout this manual literal (command line) commands will be set in `verbatim`. Controls (such as buttons or textboxes) will be **bold** faced.

2 Installation

The Warp software can be installed on Microsoft Windows systems, or any system running an Unix operating system. Furthermore you need Matlab version 5 or higher.

2.1 Installation on Unix systems

First of all you need to download the QMG 2.0 mesh generation package (Vavasis (1999)) named `qmg2_0.tar.gz` from <http://www.cs.cornell.edu/home/vavasis/qmg2.0/installation.html>. Additional information regarding the installation can also be found on this page. Install the package by creating a directory `~\warp` and placing the files `qmg2_0.tar.gz`, `warpcore.tar.gz` and `warpinstall` in this directory. Execute `warpinstall` and installation will proceed automatically.

The next step is configuring QMG. Go to `~\warp\qmg\build\unixmatlab`. In this directory you need to customize the `custom` file. Open it with your favorite text editor, and make the changes as indicated in the comments. If you are uncertain about certain options, ask your system administrator.

After customisation, issue a `make` command to compile and link the software. Executables will be written to `~\Warp\qmg\ex` and a Matlab startup script `startup.m` file will be created in `~\Warp\qmg\ex`. A directory structure as shown in figure 1 should be created. The main Matlab file `Warp.m` now resides in `~\Warp` and can be executed from the Matlab prompt by issuing

```
Matlab
cd ~/Warp
Warp
```

The following message should appear

```
QMG 2.0 startup file successfully completed.
Type alltests to test QMG. This takes a long time.
Type helpqmg to start web-based documentation.
Warp GUI ready.
```

The Warp gui (graphical user interface) will pop up in figure 1 and a QMG settings-dialog will be created in figure 2. You are now ready to go.

2.2 Installation on Microsoft Windows systems

Extract the self-extracting zip file `warpwin.exe` to `c:.` The directory structure as shown in figure 1) should be created.

To start Warp, open Matlab and give the following commands

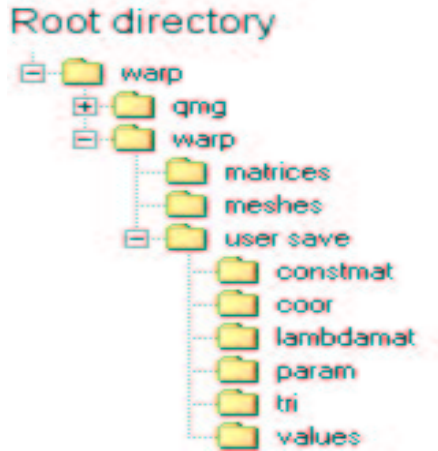


FIGURE 1: Directory structure for the Warp program.

```
cd C:/Warp
Warp
```

The following message should appear

```
QMG 2.0 startup file successfully completed.
Type alltests to test QMG. This takes a long time.
Type helpqmg to start web-based documentation.
Warp GUI ready.
```

The Warp gui will appear in figure 1 and the QMG settings-dialog pops up in figure 2. Warp is now ready for use.

3 Using Warp

The Warp software is controlled by means of a graphical user interface as depicted in figure 2. The following sections will describe the use of the interface for specifying geometries, calculating solutions and visualising solutions. The general philosophy is that you work your way top-down through the choices. The controls that can not yet be used will be greyed out. Before starting Warp, make sure you have no open figures. Warp needs Matlab figure 1 and will only work in this figure. QMG resides in figure 2, you will probably not need this figure so you can push the **Hide** button. Also note the status bar at the bottom of the GUI, which will tell you what task the software is performing. Next to this the time will be displayed it took to execute the last command.

The significance of the various options is discussed in the accompanying report (Swart (2000)).

3.1 Specifying the geometry

The first choice to be made is whether to use the QMG mesh generator or the internal mesh generator. The internal mesh generator creates structured meshes for rectangular geometries while the QMG mesh generator creates unstructured

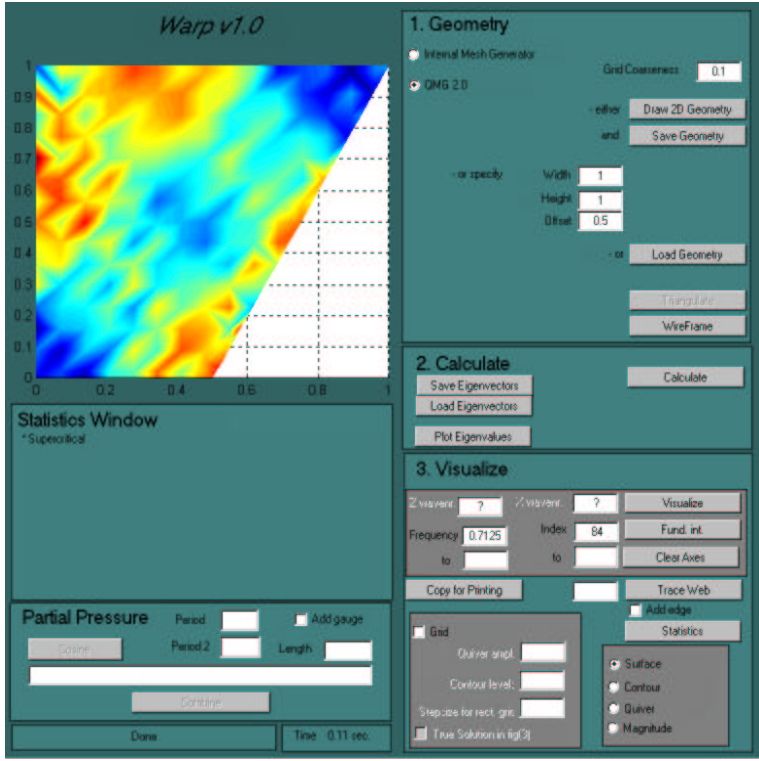


FIGURE 2: Schematic overview of the Warp implementation

meshes for arbitrary geometries. Next, enter the desired coarseness for your mesh. For a geometry of dimensions of order 1×1 this will be typically between 5 and 20 for a structured grid (representing the number of gridpoints on the axes) or between 0.5 and 0.05 for an unstructured grid (representing the approximate size of the elements). Higher values will quickly lead to memory issues on ordinary workstations.

The next step is the specification of the geometry itself. Using **Draw Geometry** one can freely design an arbitrary shape by clicking control points. More useful is the specification of the **Width**, **Height** and **Offset**. This creates a trapezoid, a rectangle with a sloping rightmost side. The **Offset** is a factor such that **Width** times **Offset** is the length of the bottom edge. When using the internal mesh generator the **Offset** will be ignored.

At this point you may also save the geometry by pushing **Save Geometry**. A file dialog will pop up. It is recommended to give your geometries all the `.geom` postfix and save them in the `meshes` directory.

The third option for specifying a geometry is thus pushing the **Load Geometry** button.

The geometry can now be triangulated by means of the **Triangulate** button and the result can be viewed using the **Wireframe** button. The statistics window will show the number of triangles and nodes generated. Make sure the number of nodes is not too high, values over of over 800 nodes can render the calculation rather lengthy.

3.2 Calculation of the solutions

The **Calculate** button will start the calculation. Assuming you entered a typical value for the grid coarseness, this will take somewhere between a few seconds and 10 minutes. After the calculation is finished you have several options (in any order)

- **Save Eigenvectors** This saves the calculated eigenvectors and eigenvalues, plus additional information. The files should be stored in `user save` with the `.mat` postfix. Additional information for internal use will be stored in subdirectories of `user save`.
- **Load Eigenvectors** Load eigenvectors and values from the `user save` directory.
- **Plot Eigenvalues** Plots all eigenvalues λ as $\frac{\lambda}{1-\lambda}$, sorted by size.

3.3 Visualisation of the solutions.

Visualisation can always be performed by entering either a **Angle** in radians or an **Index**. This angle represents the smallest angle with the vertical and must be between 0 and $\pi/2$. The angle is related to the eigenvalue, see the accompanying report. The index represents an index into the eigenvalues (sorted by increasing value) and corresponding eigenvectors. The index must ofcourse be smaller than the number of nodes. When entering an angle or an index, the other value will be calculated by pressing enter¹. The textboxes underneath will be covered in a later section.

When using the internal mesh generator and thus a structured grid you can also enter the **Z wavenumber** and **X wavenumber**. Since the true solutions are known you may opt to check **True Solution in fig(3)**, in order to visually compare solutions. Also the statistics window will give information regarding the found eigenvalue and the error. The non-simple eigenvalues will automatically be combined, and the 'Partial Pressure' window (see section 3.4) will be set with the chosen values.

The **Statistics** button will give information on the current matrix $\mathbf{A} + \lambda\mathbf{B}$ and will visualise the fundamental interval(s).

The **Copy for Printing** button still has a bug. Do not try to edit the copied figure, this will affect the Warp gui.

3.3.1 Visualisation of modes

The initial setting for the visualisation is a plot linearly interpolated on the gridpoints. The grid can be visualised using the **Grid** checkbox. Also the rotation and zoom controls at the top of the screen can be utilized. The other three visualisation options are

- **Contour** This will create a contour plot of the current eigenvector. In order for this to work the mesh will have to be converted to a equidistant

¹For example, when you enter 0.7 for the angle, the *closest* corresponding index will be calculated and displayed. When clicking in the index textbox and hitting enter, the corresponding angle will be shown. This can, however, be unequal to 0.7 (say 0.6954).

rectangular grid. For this, enter the intergridpoint distance in **Stepsize for rect. grid**. After the specification of the desired number of **contour levels** the **Visualise** button will produce contour plots.

- **Quiver** This option plots the velocities in the form of vectors on grid-points. As with the contour plot, the **Stepsize for rect. grid** needs to be specified in order for this to operate. Additionally the length of the vectors must be entered in **Quiver amplitude**.
- **Magnitude** This plots the magnitudes of the velocities. The options needed for the quiver plot must also be specified for the 'magnitude plot'.

3.3.2 Tracing of characteristics

With the **Trace Web** button one can trace a number of characteristics through a number of reflections. These characteristics are superimposed on the current image. The angle entered in the **Angle** textbox is used here. Note that even if, for example, the angle of exactly 7.0 does not exist (as eigenvalue), it can still be traced. Before tracing, the desired number of reflections must be entered in the textbox next to the button. Only the last 20 reflections will be drawn. This is useful if the characteristics converge towards an attractor, one can enter a value of, say 300 reflections and the last 20 will lie practically on the attractor. If nothing seems to happen it is most likely that the characteristics were attracted towards a point, which will mean that the last 20 lines all lie in a region that cannot be resolved by the resolution of the screen.

If you want to see a longer series of characteristics, enter subsequently 20, 40, 60, ... and press the **trace** button each time. For the algorithm used, see section 4.2.

The tracing starts at the point (0, 0), you have to make sure that any geometry you design has this point in the interior or on the boundary. Furthermore, when the geometry was created using **Draw Geometry** the edges must be specified anticlockwise.

If the **Add edge** checkbox is selected, a closed loop consisting of four boundary intersections will be sought in the next computation of the characteristics. This loop is a (1, 1) attractor. Extra edges will be added along the attractor with the coarseness at these edges set to half the current value. To calculate solutions on the new grid, press **Calculate**.

3.3.3 Visualisation of the fundamental intervals

The **Fund. int.** button finds the fundamental intervals and visualises them. This works for rectangular geometries and the trapezoid. This option also uses the raytrace algorithm.

The fundamental intervals are the regions where the boundary conditions need to be specified. This will be covered in the next section.

3.4 Forcing boundary conditions

As discussed in the accompanying report it is possible to construct a solution conforming to boundary conditions by forming a linear combination of a group

of eigenvectors. This function is only supported for the trapezoid and the rectangle in Warp. Specify the group of eigenvectors desired by entering an upper and a lower bound in **Index** and **Index2**. Alternatively one can specify **Frequency** and **Frequency2**. The **Length** field will be updated in the Partial Pressure frame.

Warp takes gridpoints in the fundamental intervals as the points on which to force the boundary conditions. These points are taken from the left to the right. For example, suppose the leftmost fundamental interval consists of 5 points and the rightmost interval has 4 points. Combining 7 eigenvectors will then completely determine the leftmost interval and the first two points of the rightmost interval will be taken.

On these intervals cosines can be prescribed. Enter the period on the left, respectively the right interval using **Period1** and **Period2**. It is up to the user to make sure the reproduced boundary conditions fit smoothly as far as this is possible. Now a pressing the **Cosine** button calculates the values to be applied as boundary conditions. These values are displayed in a textbox and can be manually edited. When entering you own values (ignoring the **Cosine** button) make sure the number of entries is correct.

The **Combine** button takes the group of eigenvalues specified and combines them to conform the fundamental intervals to the values shown in the textbox. The resulting vector is stored in the first eigenvector (overwriting the old first eigenvector), also the first eigenvalue is overwritten with the eigenvalue corresponding to the one from the **Index** field.

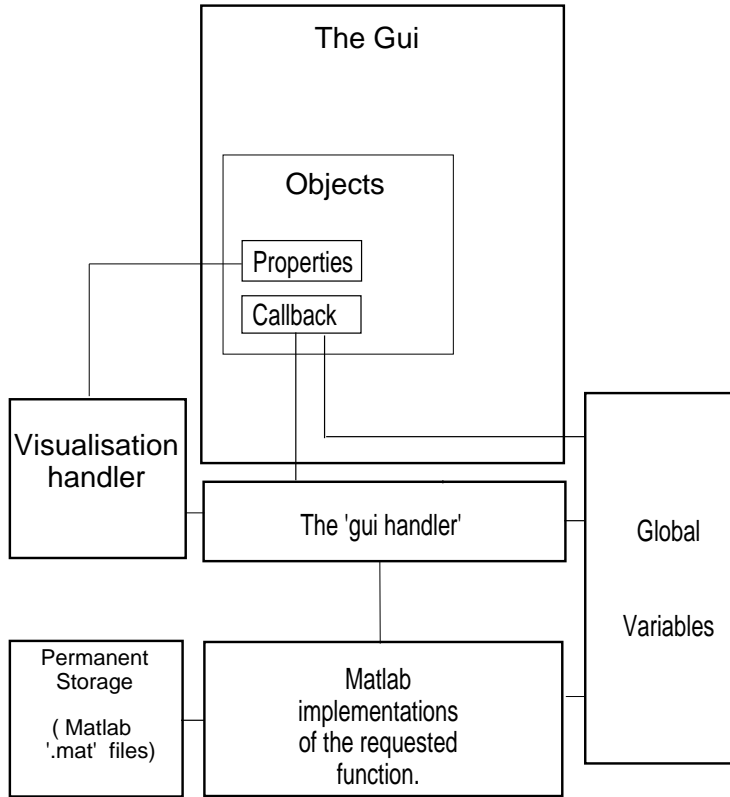


FIGURE 3: Schematic overview of the Warp program

4 Implementation of Warp

The following sections will discuss the general structure of Warp. In addition to this the implementation of some specific functions will be given.

4.1 Overview

This section will give an overview of the structure of the Warp program. The program is implemented as depicted in figure 3. The main part is the graphical user interface itself. It consists of various objects. These objects are defined by their properties, for example a button may have properties like: **Style = 'PushbuttonButton'**, **String = 'Push me'**, **Position = [100 100 50 20]** and so forth. The GUI was designed using the Matlab tool Guide which consists of a property editor, a callback editor and an alignment tool. See asfff (2001) for detailed information on this.

The next important aspect of an object is it's callback. A callback is an user definable function that is executed when the object is activated by the user. In Warp almost every callback sends a request to a Matlab file called **guihandler.m**. For example, when pushing the **Calculate** button the callback executes **guihandler('Calculate')**. Only the most trivial functions have been

handled inside the callback and do not send a request to the guihandler.

The callbacks and guihandler communicate using global variables. As opposed to local variables these can be accessed from any function. By convention global variables are capitalised. An overview of the used global variables can be found in table 2. Some callbacks that only have the task of setting global variables do not engage the guihandler but do so within the callback.

The guihandler has two basic tasks. It's first action is to execute the necessary Matlab files and if needed to set some global variables. Next it will send a message to the visualisation handler, for example `vishandler('Calculatedone')`. Table 4 lists all main Matlab `.m` files used. The utility functions are listed in table 3. The Matlab implementations write some matrices to the harddrive, see table 1 for an overview of these.

The visualisation handler sets properties of objects. Mainly this is used for enabling and disabling buttons and textboxes. This is done to make sure no unavailable actions can be taken by the user. One exception to this procedure is accessing the axes which is an area where drawing is done. Axes are always directly accessed from the Matlab functions.

4.2 Tracing Characteristics

This section describes how the tracing of characteristics in a given geometry is implemented. It is assumed that a list of vertex coordinates is available. This list (v_1, v_2, \dots, v_m) should be such that the line pieces connecting v_i to v_{i+1} for $1 \leq i \leq m - 1$, plus v_m to v_1 , form the geometry. It is imperative to know if vertices specify the geometry clockwise or anticlockwise or there would be no way to distinguish between inward and outward normals. The Warp program assumes an anticlockwise orientation. Furthermore the tracing starts at a specific point (x_0, z_0) , this point must lie in the interior of the geometry. Warp takes this point to be $(x_0, z_0) = (0, 0)$. The final ingredient is the smallest angle with the vertical θ with $0 \leq \theta \leq \pi/4$. This angle θ is chosen as initial direction. As shown in Swart (2000) this smallest angle is a fixed. The allowed angles that the characteristics can make with the positive z -axis are thus $\alpha \in \{\theta, \pi - \theta, \pi + \theta, 2\pi - \theta\}$.

The problem is now: given a position (x_i, z_i) and a direction α , calculate the next intersection with the boundary (this will become the new position) and calculate the new direction α . We will need the following relations that give the x and z coordinates of the intersection of the lines passing through (x_1, z_1) and (x_2, z_2) respectively (x_3, z_3) and (x_4, z_4)

$$x = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & z_1 \\ x_2 & z_2 \end{vmatrix} (x_3 - x_4) \\ \begin{vmatrix} x_3 & z_3 \\ x_4 & z_4 \end{vmatrix} (x_1 - x_2) \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & z_1 - z_2 \\ x_3 - x_4 & z_3 - z_4 \end{vmatrix}} \quad (1)$$

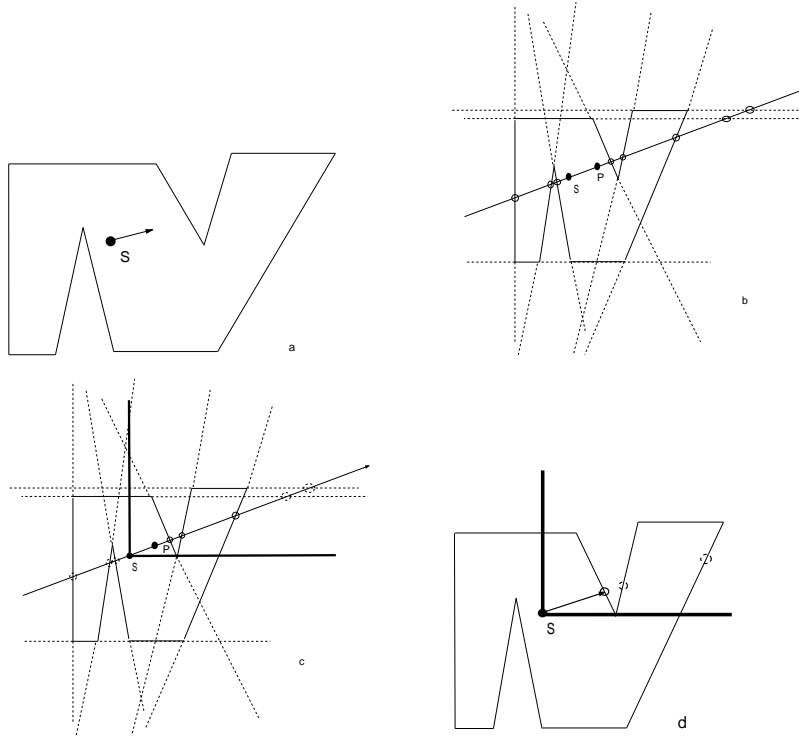


FIGURE 4: Figure (a) shows a geometry with an initial point S from which the tracing will commence in the direction indicated by the arrow. In figure (b) all linepieces are extended to full lines. The algorithm creates an extra point P arbitrary on the characteristic for use in equations (1) and (2). Using these equations all line-characteristic intersections are calculated (shown as circles). In figure (c) all intersections outside the geometry and all intersections in the wrong direction are discarded (shown as dotted circles). Figure (d) depicts the result, the linepiece from S to the closest intersection left.

$$z = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & z_1 \\ x_2 & z_2 \end{vmatrix} & (z_1 - z_2) \\ \begin{vmatrix} x_3 & z_3 \\ x_4 & z_4 \end{vmatrix} & (z_3 - z_4) \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & z_1 - z_2 \\ x_3 - x_4 & z_3 - z_4 \end{vmatrix}} \quad (2)$$

The vertical bars denote determinants. Note that these relations give infinity for parallel lines. With the help of these relations the following algorithm can be used to find the next intersection (see also figure 4):

1. The current position is (x_i, z_i) . Create another set of coordinates (x', z') on the characteristic, using the direction α .
2. Find the intersections of the characteristic with every boundary line-piece,

using (1) and (2). Here the coordinates of the edges are used as coordinates on the line. Intersections outside the geometry will therefore be found.

3. Discard of the intersection that is the current point (x_i, z_i) . This intersection will be present if the current point is on a boundary.
4. Discard of any intersections outside the geometry.
5. Discard of any intersections in the wrong direction. Only intersections that lie in the quadrant given by the direction of the characteristic are preserved. This ensures that the trace will not reverse direction.
6. Select the closest intersection, this is the new current position

The new angle is restricted to four directions. The correct angle is found by the following algorithm:

1. Discard the angle that is equal to the current angle. This would lead us outside the geometry.
2. Discard the angle that is the opposite of the current angle.
3. Select one of the two remaining angles by checking if the boundary is sub- or supercritical. Here the normal on the boundary is used. See figure 5 for an example.

4.3 Grid refinement

4.3.1 Brep structure

The grid refinement was quite difficult to implement due to the complexity of the data structure that describes the geometry. The QMG manual (Vavasis (1999)) gives extensive information but is rather cryptic at certain points. What follows will be a somewhat less extensive but more comprehensive description. It is recommended to have a printout of the file `trace.m` available, this may clarify the steps taken in the following sections.

Geometric entities in QMG are described by boundary representations, or breps for short. Breps are suitable for non-selfintersecting 2D or 3D objects. The following table gives the structure of a 2D brep. Only straight lines are used, no Bezier curves. It is assumed that the reader is familiar with Matlab *cell* data structures. Furthermore QMG adds an extra data type to Matlab; the *zba*. This enables indices to start at zero instead of one.

entry	type	contains
brep{0}	string	always 'brep v2.0'
brep{1}	integer	dimension of the object, 2
brep{2}	integer	dimension of the space the object is in, 2
brep{3}{prop,val}	string	property-value pairs, empty in practice
brep{4}	(<i>verts</i> , 2) array	<i>x, y</i> coordinates of the control points (vertices) In total there are <i>verts</i> vertices
brep{5}{0,i}	string	label for vertex <i>i</i>
brep{5}{1,i}{prop,val}	string	property-value pairs
brep{5}{2,i}{j}	string	list of subfaces, always empty ²
brep{5}{3,i}{j}	string	list of lower-dimensional boundaries, always empty ²
brep{5}{4,i}{0}	string	type of point, use 'vertex'
brep{5}{4,i}{1}	integer	degree, empty for vertices
brep{5}{4,i}{2}	integer array	control point indices, here 1 integer equal to <i>i</i> .
brep{6}{0,i}	string	label for vertex <i>i</i>
brep{6}{1,i}{prop,val}	string	property-value pairs
brep{6}{2,i}{j}	string	list of subfaces, here a list of vertices
brep{6}{3,i}{j}	string	list of lower-dimensional boundaries, here empty ³
brep{6}{4,i}{0,j}	string	type of <i>i</i> th edge, <i>j</i> th subedge, here 'bezier_curve'
brep{6}{4,i}{1,j}	integer	degree of <i>i</i> th edge, <i>j</i> th subedge, here 1 for a line
brep{6}{4,i}{2,j}(0)	integer	from vertex of <i>i</i> th edge, <i>j</i> th subedge, (index into brep{4})
brep{6}{4,i}{2,j}(1)	integer	to vertex of <i>i</i> th edge, <i>j</i> th subedge, (index into brep{4})
brep{7}{0,i}	string	label for regions, here only one
brep{7}{1,i}{prop,val}	string	property-value pairs for region <i>i</i>
brep{7}{2}{i}	string	list of subfaces (boundaries), here a the edges
brep{7}{3}{i}	string	list of lower dimensional internal boundaries, here empty ³
brep{7}{4}{i}	string	list of subregions making up the face, here empty

One will note the occurrence of sub-entities. For example a edge can be composed of two other edges. Also a 2D mesh (which is composed similar to a brep) has for example triangle entities in brep{7}{4}. Also these subdivisions are necessary for adding internal edges, as will be seen later. Another feature that needs extra attention are the property-value pairs. These can be used to various means but the main function is local grid refinement. This is

²A subspace of a point would make no sense. This entry exists to make the brep structure consistent with brep{6} and brep{7}

³A lower dimensional internal boundary is an internal boundary of at least two dimensions lower. For example a point in a face or a line in a region.

done using the **sizecontrol** keyword along with a suitably chosen value, for example **'const 0.2'**. More complicated mesh refinements can be implemented by specifying a function instead of a constant. One could take for example refinement proportional to a certain distance. This sizecontrol property can be associated with a vertex, an edge or a region, so one can refine locally. It is even possible to specify functions, instead of constant values.

4.3.2 Adding internal edges

The addition of extra internal edges along a attractor currently only works for (1,1) attractors. The reason for this is that more complicated attractors intersect *in* the domain, which is not yet taken care of by the implementation.

The first step to take is tracing a series of characteristics and taking the last 4 points visited. These will be an ordered list of intersections with the boundary. Add these points to the list of coordinates in **brep{4}**. Also, add vertex entities to **brep{5}**.

The next entry to be filled is **brep{6}**. Normally one would specify edges here from one vertex to another. However, we now have edges touching the boundary that need to be connected to it. For this reason the boundary edges are composed of two parts; a linepiece from the corner to the intersection point and a linepiece from the intersection point to the next corner. The vertex corresponding to the intersection point is thus listed twice. When all boundary edges are constructed one can proceed with inserting the edges in the domain.

The insertion of the internal edges is straightforward. Suppose **'vert5'**, **'vert6'** **'vert7'** and **'vert8'** are the vertices at the intersection points. Add one edges from **'vert5'** to **'vert6'**, from **'vert6'** to **'vert7'**, from **'vert7'** to **'vert8'** and from **'vert8'** to **'vert5'**. Furthermore the property-value pairs of these edges are set to have the mesh generator generate a triangulation at half the coarseness.

As a final step **brep{7}** must be specified. Insert the four boundary edges as normal. The internal edges along the attractor need to be listed twice. The reason for this is that all vertices must occur an even number of times in the boundary representation.

5 Known bugs

- Zooming in on an eigenvalue plot behaves strangely sometimes. Try visualising an eigenvector first and pressing and depressing the rotate and zoom buttons a few times. This solves the problem sometimes.
- After using **Copy for printing**, do not try to edit Figure 4 where the axes is copied to. This will mess up the GUI window.
- If compiled for Sun Solaris, the QMG package will crash Matlab when exiting. You can manually kill the associated processes by issuing `ps -Af | grep matlab` or `top`. Look up the PID numbers and type `kill -9 PID`.

Furthermore, Warp comes with no error handling at all.

Table 1: Description of matrices used by Warp

Directory / matrices		
File	Contains (from $(\mathbf{A} + \lambda\mathbf{B})\mathbf{x} = 0$)	Matlab variable name
constmat.mat	matrix A	aconst
lambdamat.mat	matrix B	alambda
eigenvalues.mat	vector, containing all eigenvalues $\lambda' = \frac{\lambda}{1-\lambda}$	E
eigenvectors.mat	matrix, columns containing \mathbf{x} 's	X
tmat.mat	triangle connectivity matrix	T
zmat.mat	triangle coordinate matrix	Z
stiffc.mat	element stiffness matrices associated with A	stiffconst
stiffm.mat	element stiffness matrices associated with B	stifflambda

Table 2: Overview of the global variables used in Warp

Global variable	Description
COARSE	Grid coarseness
FREQ	Current angle of characteristics with the vertical
FREQ2	Second angle, for the purpose of combining eigenvectors
GEN	0 = internal mesh generator (messy.m) 1 = QMG mesh generator
GEOMMETHOD	0 = geometry was drawn 1 = coordinates were specified
GRID	0 = Do not overlay the mesh when visualising 1 = Do overlay the mesh when visualising
HEIGHT	Height of the basin
INDEX	Current index into the eigenvalues
INDEX2	Second index, for the purpose of combining eigenvectors
LAST	0 = Last entered value was the angle 1 = Last entered value was the index 2 = Last entered value was a modenumber
LEVELS	Number of contour plot levels
MAG	Multiplication factor for the magnitude of the quiver plot arrows
NRNODES	Number of nodes
NRTRIANGLES	Number of triangles
OFFSET	Length of the bottom of the basin as a fraction
STEP	Number of equidistant gridpoints to generate. For contour and quiver plots.
TEMPBREP	Contains the boundary representation of the current mesh
TRUESOL	0 = Do not visualise true solution 1 = Do visualise true solution
VIZMODE	0 = Ordinary pressure plot 1 = Contour plot 2 = Quiver plot 3 = Magnitude plot
WIDTH	Width of the basin
XMODE	Requested modenumber in the x direction
ZMODE	Requested modenumber in the z direction

Table 3: Description of utility Matlab '.m' files used by Warp

File	Description
a2i.m	Converts angle to index
i2a.m	Converts index to angle
approx.m	Tests if two values are approximately equal
nearest.m	Finds the closest match to a value in a vector
status.m	Writes a string to the status line of the GUI
normalise.m	Normalises the eigenvectors to the range $[-1, 1]$
checkempty.m	Checks for empty (and thus invalid) textboxes

Table 4: Description of main Matlab '.m' files used by Warp

File	Description
assemble.m	Assembles the element stiffness matrices
domode.m	Constructs and visualises a (m, n) mode
eigplot.m	Plots eigenvalues
eigsolve.m	Solves the generalised eigenvalue problem $(\mathbf{A} + \lambda\mathbf{B})\mathbf{x} = 0$
elmstiff2.m	Constructs element stiffness matrices
force.m	Constructs a linear combination of eigenvectors, the result conforms to a boundary condition
fund.m	Find the fundamental intervals
guihandler.m	Responds to user interaction with the GUI
makecosine.m	Constructs a cosine on the fundamental interval(s)
makestiff.m	Subroutine of elmstiff.m
meshgen.m	Triangulates a geometry
messy.m	Subroutine of meshgen.m, for structured meshes
oldviz.m	Handles the various visualisations
stats.m	Statistics on the matrix $\mathbf{A} + \lambda\mathbf{B}$
trace.m	Traces characteristics in a geometry Optionally refines the grid
vishandler.m	Enables/disables GUI objects
wireview.m	Displays a wireframe view of a mesh

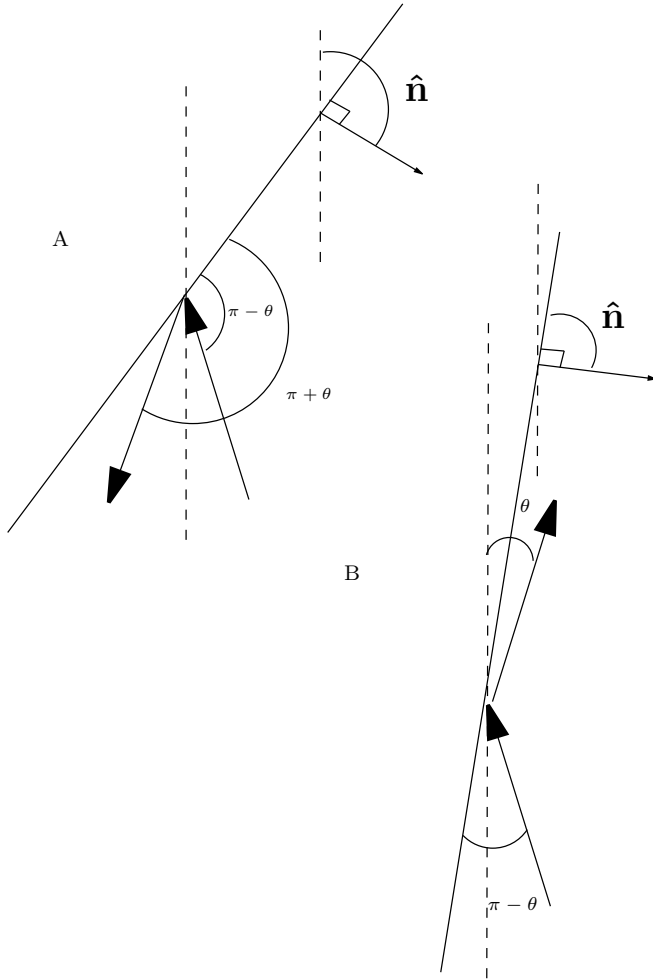


FIGURE 5: This figure gives an example of characteristic reflections. Reflection is as indicated by the arrows. The dotted line is the vertical, the solid line stands for a boundary. The normal direction $\hat{\mathbf{n}}$ is measured with respect to the vertical. In order to determine the new direction of the characteristic we have to compare it with the angle of the boundary ($\hat{\mathbf{n}} - \pi/2$) to check for sub- or supercriticality. This figure gives an example, the angle of the incident characteristic is $\pi - \theta$. In situation A we have that $\hat{\mathbf{n}} - \pi/2 > \theta$ which results in a downward reflection in the $\pi + \theta$ direction. Situation B has $\hat{\mathbf{n}} - \pi/2 < \theta$ which results in an upward reflection.

References

ASFFF 2001 *Building Graphical User Interfaces with Matlab*.

SWART, A. 2000 A finite element method for internal gravity waves .

VAVASIS, S. M. 1999 The qmg 2.0 mesh generator
<http://www.cs.cornell.edu/home/vavasis/qmg-home.html>.