CRPP - INT 205/02 - February 2002

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

CENTRE DE RECHERCHES EN PHYSIQUE DES PLASMAS (CRPP)
Association EURATOM - Confédération suisse

# TCV Diagnostics Documentation

# Fast-Algorithm Bolometric Computer Aided Tomography (FABCAT)

Christian Schlatter[1], Jan Mlynář[2]
CRPP EPFL
PPB
CH-1015 Lausanne

February, 2002

Made with LaTeX

[1]TP IV in physics, christian.schlatter@epfl.ch
[2]jan.mlynar@epfl.ch

# Contents

# List of Figures

# Chapter 1

# Introduction

The **F**ast-**A**lgoritm **B**olometric **C**omputer **A**ided **T**omography (**FABCAT**[1]) reconstruction code is a modified release of the *btomo.m* package [1] without GUI interface and using a fast minimum Fisher regularization algorithm. The *btomo.m* package (see figure 1.1) is a useful tool for tomography reconstruction, but it needs quite a lot of computing time and user interaction before delivering results.

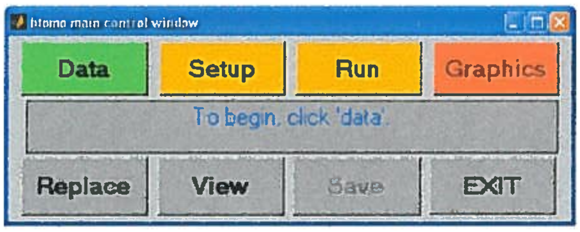


**Figure 1.1: btomo.m package.** Graphical user interface for bolometry tomography reconstruction.

In order to evaluate a plasma shot shortly after the acquisition, one needs a faster tool for the plasma shape reconstruction.

Recently a fast tomography reconstruction algorithm for Soft X-Ray tomography has been developed (the *cattcv.m* package [2]), but an adequate release for bolometry was still missing.

The new *fabcat.m* package is simply calling an adaption to the requirements of bolometry of the fast Minimum Fisher Regularization routine from *cattcv.m*. It also uses an averaged, single reconstruction matrix **M** which can be applied for all timeframes, making the inversion by the number of timeframes faster (about three seconds in the code at issue).

After testing the package for some particular shots we processed shots 19000 up to 21739 and stored their reconstructions in the MDS tree. In this interval there are 1589 shots, and 668 (42 %) have been reconstructed successfully.

The next chapter gives a short overview and memento on bolometry. Chapter 3 describes the parts and organization of the package for a first orientation. For most of the users of the package the chapter 4 giving operating instructions will probably be the most important. Chapter 5 gives more details on the code, mainly it explains the tomography and their fast formulation for bolometry. Chapter 6 shows reconstructed images for selected shots. For those already familiar with other tomography packages (*btomo.m* for bolometry, *cattcv.m* for Soft X-Ray) the chapter A explaining the roots of their codes and its modification will be relevant.

Finally the appendix lists the program code.

This documentation is available online at http://crpplocal/diags/fabcat.pdf (Adobe Acrobat file). The package is administrated by Jan Mlynář and ready to use on HAL under \home\mlynar\matlab\fabcat\.

[1]colloquial: 'fabulous cat'

# Chapter 2

# Bolometry on Tokamak à configuration variable (TCV)

Bolometric measurements on tokamaks are useful for getting the distribution of the sources of radiation inside the vessel. The TCV at CRPP has a major radius of R = 88 cm and an almost rectangular cross section with width and height of r = 55 cm and z = 155 cm respectively. 8 bolometer arrays with 8 channels each (see figure 2.1) are grouped around a poloidal section of the tokamak and arranged in arrays of 5 pinhole cameras as shown in figure 2.2. Each camera measures the line integrated radiation. A tomographic inversion of the radiation signals provides a spatial resolution of the radiation sources (the emissivity profile).

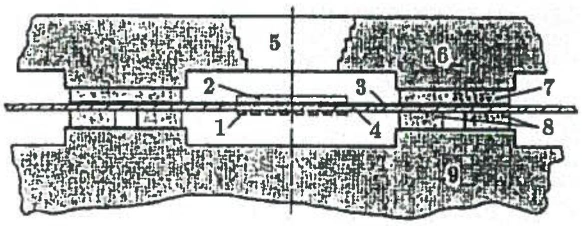


**Figure 2.1: Bolometer design.** Legend: 1 - gold-meander resistor; 2 - gold-absorber layer; 3 - golden heat-resistor layer; 4 - kapton carrier foil; 5 - viewing window; 6 - aluminium cooling plate; 7 - golden thermal-contact layer; 8 - conduction path; 9 - mounting support.

The line integrated energy flux is derived from the temperature rise and thermal capacity of well-defined golden foils. However, their thermal inertia limits the temporal resolution of this type of bolometer to approximately 10 ms. The measured signal corresponds not only to the radiated energy, but also contains the contribution due to neutral particle flow [[3]].

The acquisition electronics works at a frequency of 50 kHz and the signals are stored in the MDS tree.

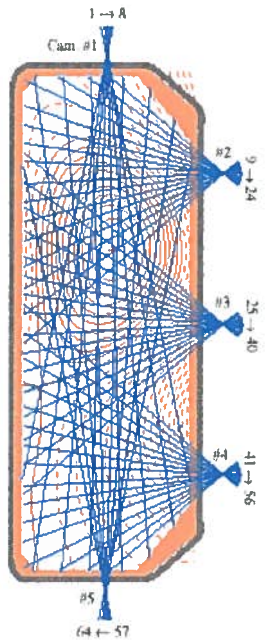
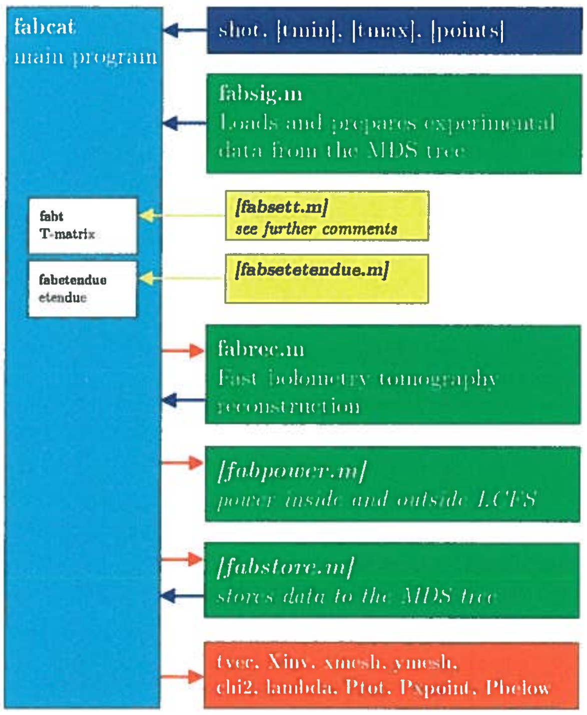


**Figure 2.2: Setup of the foil bolometers.** Poloidal section of the TCV tokamak. 5 pinhole cameras and the viewing lines of the 64 gold foil bolometers.

# Chapter 3

# Organization of the code

The algorithm is based on the btomo package. Most of the parts has been recycled and adapted, mainly by eliminating the GUI related parts of the code. The fast inversion code based on minimizing the Fisher information is essentially the same as that used in Soft-X Ray tomography [2].



fabcat
main program
shot, [tmin], [tmax], [points]
fabsig.m
Loads and prepares experimental data from the MDS tree
fabt
T-matrix
[fabsett.m]
see further comments
fabetendue
etendue
[fabsetetendue.m]
fabrec.m
Fast bolometry tomography reconstruction
[fabpower.m]
power inside and outside LCFS
[fabstore.m]
stores data to the MDS tree
tvec, Xinv, xmesh, ymesh,
chi2, lambda, Ptot, Pxpoint, Pbelow

**Figure 3.1: Main program fabcat.m and its subroutines.** The green blocks represent the indispensable subroutines. Yellow units are executed separately by the user to create the variable files (drawn in white) loaded by the main code.

## 3.1 Main program

Figure 3.1 gives an overview of the structure of the code by and names its subroutines.

*fabcat.m* is the main program and should be called up as described in chapter 4.1 (see paragraph 5.1 for further description).

The subroutine *fabsig.m* gets the measured bolometry signals from the MDS database, checks for bad channels and does data preprocessing, i.e. corrections respecting the thermal inertia of the bolometers (see paragraph 5.2 for further description).

*fabrec.m* calculates the tomography inversion using a time averaged M-matrix based on minimization of the Fisher information as regularization functional (see paragraph 5.3 for further description).

*fabstore.m* stores the reconstruction image (the emissivity $g$) and the total and partial (from the x-point and divertor) power radiation in the MDS tree. (see paragraph 4.3 about reading the MDS tree).

The .mat files *fabchord* (coordinates of the viewing lines of the bolometry detectors), *fabt* (contains the T-matrix) and *fabetendue* (contains the etendue of the detectors for bolometry) are loaded by the main program and can be recalculated using the associated .m files (see the following paragraph). The precalculated T-matrix was established for a rectangular grid of

$n_{pixel} = 280$ pixel[1].

## 3.2 Supporting routines

### 3.2.1 fabsetchord.m

Calculates the coordinates of the viewing lines of the bolometers and stores them to *fabchord.mat*.

### 3.2.2 fabsetetendue.m

Calculates the bolometers etendue and stores them to *fabetendue.mat*. The etendue is used to calculate the errorbars of the signals.

### 3.2.3 fabsett.m

Generates the T-matrix for the inversion, based on the solid angles and etendue of the bolometers. The program loads precalculated *fabsolidangle_#.mat* files and creates the *fabt.mat* file. See figure 3.2 for the programs hierarchy.

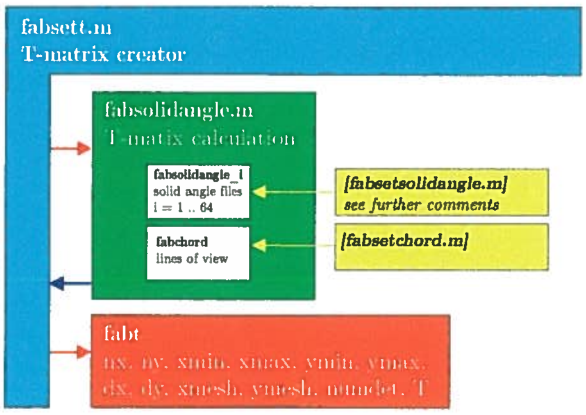


**Figure 3.2: T-matrix calculator.** The T-matrix elements $T_{\ell i}$ give the weight of pixel $i$ in the viewing line of bolometer channel $\ell$.

The program calls the *fabsolidangle.m* file, which is exactly the same as in the *btomo.m* package (*btt_omgrid.m*). The *fabsolidangle_#.mat* files can be recalculated using *fabsetsolidangle.m* as presented in figure 3.3.

[1] $n_x = 10$ in horizontal and $n_y = 28$ in vertical direction.

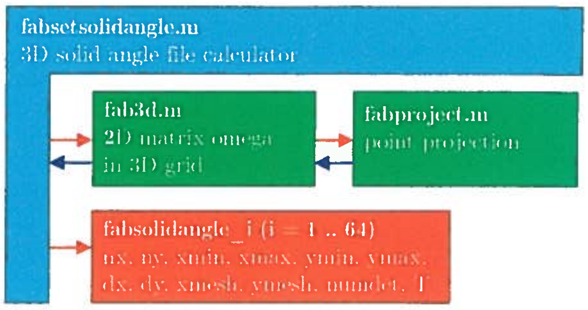
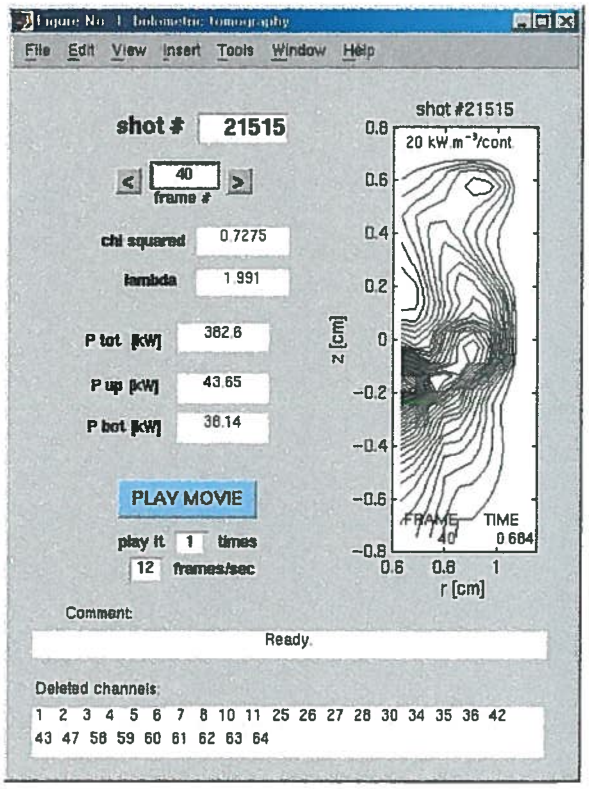

fabsetsolidangle.m
3D solid angle file calculator
fab3d.m
2D matrix omega
in 3D grid
fabproject.m
point projection
fabsolidangle_i (i = 1 .. 64)
nx, ny, xmin, xmax, ymin, ymax,
dx, dy, xmesh, ymesh, numdet, T

**Figure 3.3: Raumwinkel file calculator.** The solid angles of each bolometer is calculated.

### 3.2.4 Bolometry movie player

The movie player *fabmov.m* shows a movie for a given shot and gives the values of $\chi^2$, $\lambda$ and radiated power for each timeframe. It also notes which channels have been deleted.



shot # 21515
frame # 40
chi squared 0.7275
lambda 1.991
P tot [kW] 382.6
P up [kW] 43.65
P bot [kW] 38.14
play it 1 times
12 frames/sec
FRAME 40 TIME 0.684

Comment:
Ready.

Deleted channels:
1 2 3 4 5 6 7 8 10 11 25 26 27 28 30 34 35 36 42
43 47 58 59 60 61 62 63 64

**Figure 3.4: Bolometric movie player.** Permits to check the results for each individual timeframe.

### 3.2.5 Other tools

*fabreader.m*

The program *fabreader.m* extracts the status for a shot or for an interval of shots from the MDS tree. Information about the existence of the shot, its successful reconstruction and if the x-point determination was possible is stored in a file *fabresult.mat*.

*fabanalyze.m*

The program *fabanalyze.m* can be used to analyze the results provided by *fabreader.m*. For the shot interval 19000 to 2173 there are 1589 shots, for 816 of them a x-point was determined and 668 have been reconstructed successfully.

*fabplot.m*

This tool plots time evolution of plasma current, plasma density and radiated power (total power, power from the X-point and divertor region) read from MDS (prior calculation by *fabcat.m* is necessary).

# Chapter 4

# Operating instructions

## 4.1 How to execute 'fabcat.m'

The program can be run in three modes:

- **fabcat(shot):**
  treats the specified plasma shot and delivers the reconstruction at the LIUQE times. The reconstructed image (dimension $n_x \times n_y \times timeframes$, unity is W/m$^3$) is available through the vector $X_{inv}$ (matlab variable name).

- **fabcat(shot, t$_{min}$, t$_{max}$):**
  loads and treats each timeframe located between $t_{min}$ and $t_{max}$ [s] only.

- **fabcat(shot, t$_{min}$, t$_{max}$, points):**
  loads and treats the timeframes located between $t_{min}$ and $t_{max}$ [s] and delivers the results in *points* frames only. Of course *points* has to be smaller than the number of acquired images between $t_{min}$ and $t_{max}$.

## 4.2 How to configure 'fabcat.m'

In the header of the main program file *fabcat.m* several flags are specified to control the behavior of the package. When experiencing problems with the execution of the package the first thing to check are these configuration flags and try to run them using the default settings. When you want to change the default settings copy all files to a local directory and run it from there.

- **f_cam**
  Specifies which pinhole cameras should be used. Each camera gathers eight bolometers looking through the same pinhole of the section (see chapter 2).
  **Default setting = [1 1 1 1 1 1 1 1]**, i.e. all eight cameras are activated.

- **chantest**
  The package can itself try to identify bad channels. The following values are admitted :
  **0:** no automatic testing for bad channels.
  **1:** Testing using Christian Deschenaux's method (backfiltering).
  **2:** Testing using Arno Refke's method (consistency check).
  **3:** Testing using both methods.
  **Default setting = 2.**
  For a description of the testing methods see paragraph 5.2.5. As a new feature, the subroutine *fabrec.m* is also testing for faulty channels (see paragraph 5.5, the configuration is done using the flag LIMIT).

- **smoothmode**
  Specifies the method by which the raw bolometry signals are treated and calibrated.
  **0:** no data treatment is done.
  **1:** Processing using Bernard Joye's method (polynomial fitting).
  **2:** Processing using Christian Deschenaux's method (backfiltering).

**Default setting = 2.**
For a detailed description of the data processing methods see paragraph 5.2.6.

- **delchanuser**
Allows the specification of channel numbers which will be ignored by the reconstruction. Maybe useful if single bolometers are defective.
**Default setting =** [], i.e. no channel excluded.

- **runsigma**
Provides the global multiplication factor for the error bars. The constants unity is in percent. The variance $\sigma_i$ will still be divided by the individual channel etendue (see formula 5.1).
**Default setting = 3.**

- **lambda_init**
Provides the initial $\lambda$ for the fast minimum Fisher regularization routine, i.e. the balance between fitting and smoothing (see paragraph 5.4.2).
**Default setting = 0.1.**

- **lamlim**
Minimum value of the final $\lambda$ which qualifies the shot still as a good one. This parameter was chosen arbitrary.
**Default setting = 0.01.**

- **chitar**
Target $\chi^2$, i.e. the reconstruction tries to obtain this final $\chi^2_{target}$.
**Default setting = 1,** i.e. a reconstruction image having maximum variance $\sigma_i$.

- **errchi**
Supreme difference between the time averaged final $\chi^2$ and the target $\chi^2_{target}$ (see flag **chitar**) which qualifies the shot still as a good one. This parameter was chosen arbitrary.
**Default setting = 0.05.**

- **loopfisher**
Provides the number of times the main loop of *fabrec.m* minimizing the Fisher information is going to be executed.
**Default setting = 3.**

- **iterfisher**
Provides the number of times the subloop of *fabrec.m* (finding $\lambda$ which brings $\chi^2$ next to $\chi^2_{target}$) is going to be executed.
**Default setting = 4.**

- **limit**
Limiting factor for the post-reconstruction bad channel check in *fabrec.m*. See paragraph 5.5 for its definition.
**Default setting = 5.**

- **i_bord**
If set to 1 then emissivity from borders of pixels mesh are constrained to be zero. Although this feature work well for Soft X-ray, it is not suitable in the case of bolometry.
**Default setting =** 0 (disabled).

- **x_blind**
If set to 1 then the radiation originating from the X-point region is subtracted from the bolometry signals and a second reconstruction is initiated. **Please note that this feature depends on geometry.** This part has to be changed if the pixel setup is modified. See the associated subflag **cutoff** for further configuration.
**Default setting =** 0 (disabled).

- **cutoff**
Specifies the expanded X-point geometry:
**0:** the X-point is found in the left center and diverter region and pixels radiating down to the percentage defined by the flag **level** of the maximum radiation are assigned to the expanded X-point.
**1:** pixel 12 and 13 as expanded X-point.
**2:** pixels 10 to 16 as expanded X-point.
**3:** pixels 6 to 18 and 41 as expanded X-point.
**4:** pixles 5 to 20 and 40 to 43 as expanded X-point.
**Default setting =** 0 (automatic determination).

- **level**
  Pixels radiating down to the percentage given by this flag of the maximum radiation are assigned to the expanded X-point.
  **Default setting = 80 [%].**

- **radiation**
  Calls the slow *fabpower.m* (*btboloti_pradinout.m* in the *btomo.m* package) to calculate the radiated power in regard to the LCFS.
  **Default setting = 0** (disabled).

- **saving**
  If set to 1 then stores shotnumber, the emissivity g (*Xinv*) and the timeframe vector (*tvec*) to a .mat file on the hard disk. Please specify the path where the files should be stored in using the flag **savepath**.
  **Default setting = 0** (disabled).

- **MDS**
  If set to 1 then the results are stored in the MDS tree. Make sure that you have MDS writing privileges.
  **Default setting = 1** (enabled).

- **plotting**
  If set to 1 then makes a contour plot for the last timeframe of the reconstructed emissivity distribution on the screen.
  **Default setting = 0** (disabled).

## 4.3 How to read the MDS tree

The subroutine *fabstore.m* (taken from the *btomo.m* package) stores the data in the MDS tree. The results can be read from MDS using the following nodes:

- *results :: btomo : method*
  Information about the used method. There are three possibilities:
  - **"FABCAT with X-point at pixel 11"**. The X-point has been found at pixel 11, but its radiation emissivity has not been excluded during the inversion.
  - **"FABCAT without X-point at pixel 11"**. The X-point has been found at pixel 11 and its radiation emissivity has been excluded during the inversion.
  - **"FABCAT, X-point determination failed"**. No X-point has been determined.

- *results :: btomo : comment*
  Gives the final $\lambda$ and the mean value of $\chi^2$. If the shot was reconstructed successfully, a statement e.g. "shot 15211 OK" is included.

- *results :: btomo : rmesh.*
  Pixels center coordinates in radial direction. Dimension $n_x$. Values in [m].

- *results :: btomo : zmesh.*
  Pixels center coordinates in vertical direction. Dimension $n_y$. Values in [m].

- *results :: btomo : lambda*
  Final smoothing factor $\lambda$, see paragraph 5.4.2 for comments.

- *results :: btomo : sigma*
  The global multiplication factor for the error bars, as defined by the flag **runsigma** in the main program (see chapter 4.1). Value in percent.

- *results :: btomo : chi_squared*
  The final fitting coefficients $\chi^2$. If the individual (i.e. for each timeframe) $\chi^2$ are not close to zero, you may rerun *fabcat.m* for a restricted time interval or verify with the *btomo.m* package. See paragraph 5.3 for more comments. Dimension *timeframes*.

- *results :: btomo : nchord*
  Vector listing the enabled channels.

- *results :: btomo : smoothing*
  Number of rescaled timeframes, only available when the variable *points* was specified within *fabcat.m* call.

- \\ *results :: btomo : time*
  Timeframe times in [s].

- \\ *results :: btomo : emissivity*
  The emissivity matrix **g** $(Xinv)$. Dimension $n_{pixel} \times timeframes$. Values in [W/m$^3$]. Notice that this node can be shortened to \\ *results :: btomo* for future shots to conform with ANASRV standards.

- \\ *results :: btomo : prad_above*
  At present radiation power coming from the X-point (only saved if the determination of the X-point was successful). The power of the pixel with highest emissivity and all neighboring pixels is summed up. Values in [W]. Dimension $timeframes$.

- \\ *results :: btomo : prad_below*
  At present radiation power coming from the divertor region below the X-point (only saved if the determination of the X-point was successful). The two leftmost pixel columns are summed up from the bottom of the vessel up to two pixels below the pixel of the x-point with highest emissivity. Values in [W]. Dimension $timeframes$.

- \\ *results :: btomo : prad_tot*
  Total radiation power of the whole tokamak. Values in [W]. Dimension $timeframes$.

# Chapter 5

# Detailed description of the method

## 5.1 fabcat.m

The header of the main program list all the necessary user adjustable configuration flags. Their signification has been described in chapter 4.

The MDS data is read by *fabsig.m* (see next paragraph) for the specified shot with respect to the chosen cameras, channels and times.

The precalculated T-matrix (including the pixel grid, all values in [cm]) is stored in *fabt.mat* and simply loaded. The error bars *df* for the inversion are calculated using the etendue $(A \cdot \Omega)$ of the bolometry detectors (stored in *fabetendue.mat*):

$$df = \sigma \cdot \frac{\sup(A \cdot \Omega)}{(A \cdot \Omega)} \tag{5.1}$$

The inversion is done using the subroutine *fabrec.m* (see paragraph 5.3).

The resulting tomography reconstruction ($X_{inv}$) is converted to [W/m$^3$]. $\chi^2$ and $\lambda$ are tested for the quality[1] of the reconstruction (by default[2] a good quality image has $\lambda > 0.01$ and $| < \chi^2 > - 1 | < 0.05$) and commented on screen.

According to the configuration flag **plotting** a contour plot of $X_{inv}$ (see figure 6.1 ff.) can be displayed or not.

[1]i.e. the mean $\chi^2$, averaged over all timeframes.

[2]see the empirical parameters **errchi** and **lamlim** as described in chapter 4

The power is calculated from the emissivity by the integration over the ring tubes intersected by the pixels. The power coming from a rectangular shaped ring (cylindrical geometry, cross section $dr \cdot dz$) intersected by pixel positioned in $(r, z)$ is

$$P(r,z) = \int_0^{2\pi} \int_{left}^{right} \int_{low}^{up} g(r,z) \cdot r \cdot d\theta \cdot dr \cdot dz \tag{5.2}$$

where $\theta$ is the toroidal angle which gives the position on the rings central line. With 'right' we mean the right border of the pixel. Assuming that the power emission is constant along the rings central line (cylindrical symmetry) and with pixels having horizonal and vertical extension of dx and dy respectively leads to

$$P(pixel) = 2\pi \cdot g(pixel) \cdot r(pixel) \cdot dx \cdot dy \tag{5.3}$$

Summing up all $n_{pixel}$ pixels provides the total radiated power $P_{tot}$:

$$P_{tot} = \sum_{pixel=1}^{n_{pixel}} P(pixel) \tag{5.4}$$

## 5.2 fabsig.m

This subroutine is responsible for extracting and preparing any information from the MDS tree. It is the only routine using MDS reading access commands.

The behavior of *fabsig.m* is essentially determined by the number of parameters provided when executing *fabcat.m* (see chapter 4.1).

### 5.2.1 The brightness $f_\ell$

During a shot, the plasma radiates. The radiation emitted in $\vec{r}$ is characterized by the emissivity[3] $g(\vec{r})$.

Detector $\ell$ collects the total radiated power $P_\ell$ from its field of view given by the solid angle $\Omega_\ell$. Hence, the total radiated power can be expressed as an integral of the cone of view $S_\ell$ defined by the detector and aperture geometry

$$P_\ell = \int_{S_\ell} d\vec{r}\, g(\vec{r}) \cdot \frac{\Omega_\ell(\vec{r})}{4\pi}, \qquad \ell = 1,..,64 \quad (5.5)$$

The bolometers have a narrow field of view. The emissivity $g(\vec{r})$ doesn't vary considerable on a surface perpendicular to the axis[4] of the cone $S_\ell$. The volume integral can be replaced by a simple line integral $d\vec{r} \rightarrow A(s)\, ds$ where $A$ is the surface of the detector and $ds$ is a line element along the line of sight $s_\ell$.

Using the brilliancy theorem[5] for the etendue $(A \cdot \Omega)$ we can write

$$P_\ell = \frac{(A\Omega)_\ell}{4\pi} \int_{s_\ell} ds\, g(\vec{r}), \qquad \ell = 1,..,64 \quad (5.6)$$

The multiplication with factor $1.78 \cdot 10^4$ in the code corresponds to the determination of the chord brightness $f_\ell$ which is derived from the total radiated power $P_\ell$ as follows

$$f_\ell = \frac{P_\ell}{(A\Omega)_\ell / 4\pi}, \qquad \ell = 1,..,64 \qquad (5.7)$$

In the following the measured chord brightness $f$ will be called signals.

[3]power per volume

[4]the so-called line of sight

[5]$A' \cdot \Omega' = A \cdot \Omega$ through an optical system

### 5.2.2 Reading the MDS tree

The following MDS trees are read :

- \\ *base* :: *bolo* : *signals*
  The matrix (size *channels* $\times$ *timeframes*) contains preprocessed bolometry signals, i.e. the acquired chord brightness $f_\ell$ integrated along the lines of sight of the detector $\ell$.

- \\ *base* :: *bolo* : *source*
  The raw bolometer signals $\check{f}_\ell$ matrix (size *channels*$\times$*timeframes*). They are used for weird channel detection.

- *dim_of*(\\ *base* :: *bolo* : *signals*)
  The normal time base vector (length *timeframes*) containing the the times corresponding to the timeframes of the signals matrix.

- *dim_of*(\\ *results* :: *r_axis*)
  The LIUQE time base vector. These are the times used for the magnetic field reconstruction of the LIUQE package.

- \\ *base* :: *bolo* : *tau*
  $\tau$, the time constants of the bolometers. They are used for data treatment.

### 5.2.3 LIUQE times

When *fabcat.m* has been called providing the shot number only, then the LIUQE times are used for the reconstruction. A new time base consisting of normal times nearest to LIUQE times is calculated.

### 5.2.4 NORMAL times

When *fabcat.m* is called providing a minimum time argument ($t_{min}$) then the normal times are used. If no maximum time ($t_{max}$) was specified all timeframes beginning with $t_{min}$ are loaded. The optional argument *points* is to use only *points* timeframes for the specified shot. This decreases calculation time. Its clear that *points* has to be lower than the total timeframes available for a given shot.

### 5.2.5 Automatic channel deleting

The header of the main program *fabcat.m* allows for the specification of channels which should not be used for reconstruction. Using the flag *channels* in the main program (see chapter 4.1) it is possible to choose all, none or one of the following automatic channel analysis algorithm to identify further faulty channels:

- Arno Refke's method. The raw raw signals are checked for consistency, i.e. the signal range, signal mean value, signal slope and negative signals.
- Christian Deschenaux's method. Filters the raw signals using a butterworth 4th order digital lowpass filter and checks the backfiltered signals mean values for consistency.

These channel deletion methods are the same as those used in the *btomo.m* package.

### 5.2.6 Signal processing

The flag *smoothmode* (in the header of *fabcat.m*) can be used to choice if one, none or one of the following smoothing procedures is used to calibrate the measured signals:

- Bernard Joye's method. Fits the temporal behavior of each channel with a polynomial of 2nd degree over 30 time samples considering the bolometers time constant $\tau$ for the linear term.

  $$f_\ell = at^2 + b\frac{\tau}{\Delta t}t$$

  where $a$ is the quadratic and $b$ the linear fit coefficient. $\Delta t$ is the averaged time step per frame.
- Christian Deschenaux's method. Filters the raw signals again using a butterworth 4th order digital lowpass filter :

  $$f_\ell = \breve{f}_\ell + \tau \cdot \frac{d\breve{f}_\ell}{dt}$$

  where $\breve{f}_\ell$ designates the backfiltered raw signals.

## 5.3 catrec.m

This is the core piece of software of this package. The inversion is done under the control of several parameters specified in the main program and described in chapter 4.1. The routine corresponds to the one used in the Soft X-ray package and has only been modified slightly (see appendix A).

### 5.3.1 Tomography principles

The tomography problem consist of solving the system of integral equations

$$f_\ell = \int_{s_\ell} ds\, g(\vec{r}), \quad \ell = 1,..,64 \qquad (5.8)$$

where $f_\ell$ are known and $g(\vec{r})$ is unknown.

The system 5.8 of inhomogeneous Fredholm equations of the first kind [3] is always underdetermined, since an infinite number of line integrated data measurements $f_\ell$ would be necessary to determine $g(\vec{r})$ exactly. By a subdivision of the area of the poloidal cross section into pixels, the number of degree of freedom is reduced. We use a grid of rectangular pixels (by default $n_x = 10$ pixels in radial and $n_y = 28$ in vertical direction) covering the whole vessel of the tokamak. The system 5.8 has now been transformed to a system of algebraic equations[6]

$$f_\ell = T_{\ell i} \cdot g_i, \quad \ell = 1,..,64 \qquad (5.9)$$

where $T_{\ell i}$ describes the weight of pixel $i$ in the line of sight of detector $\ell$. The **T**-matrix is related to geometry only (detector positions, aperture dimensions and pixel grid) and is calculated using *fabsett.m* (see paragraph 3.2.3).

**g** represents now the two-dimensional emissivity distribution.

[6]sum over repeated indexes (Einsteins convention).

Unfortunately a direct inversion of **T** is not possible because it is badly conditioned. Furthermore, with our grid we have $n_{pixel} = 280 > n_\ell = 64$, so the set of equations 5.9 is overdetermined and none or an infinite number of solutions is possible. To get a unique and sensible solution of the problem a functional

$$\phi = \frac{1}{2}\chi^2 + \lambda\Re \tag{5.10}$$

is minimized (see [4]). $\chi^2$ is calculated by

$$\chi^2 = \sum_i \left[ \left( \frac{T_{\ell i} \cdot g_i - f_i}{\sigma_i} \right)^T \cdot \left( \frac{T_{\ell i} \cdot g_i - f_i}{\sigma_i} \right) \right] \tag{5.11}$$

and controls the goodness[7] of fit. $\Re$ is a regularization functional controlling the smoothness of the fit. The smoothing parameter $\lambda$ weights between goodness of the fit and smoothness and has to be chosen carefully for reliable results.

The $\Re$ functional in *fabrec.m* is the minimum of the Fisher Information $I_F$

$$I_F = \int dx \frac{(g'(x))^2}{g(x)} \tag{5.12}$$

leading to the smoothest [5] solution of the inversion. The division by $g(x)$ in 5.12 assures that regions of low $g$ are particularly smooth, taking care of low-emissivity regions.

## 5.4 Coding of a fast inversion routine

The process to establish the emissivity g was applied to each timeframe $t$ in the *btomo.m* package. The essential reduction in time consumption of *fabrec.m* is related to averaging $\chi^2$ and $\lambda$ and the reconstruction matrix **M** over the whole shot. The routine uses two nested loops which are shown in figure 5.1.

[7]low $\chi^2$ corresponds to 'overfitting', $\chi^2 = 1$ is the solution with maximum variance.

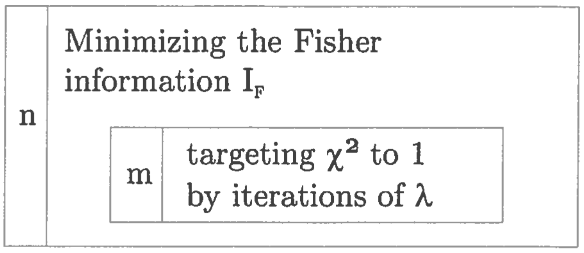

n
Minimizing the Fisher information $I_F$
m
targeting $\chi^2$ to 1 by iterations of $\lambda$

**Figure 5.1: Iterations to find the emissivity g.** For the minimization of the Fisher information n = 3 loops (flag **loopfisher**) are enough, for the smoothing loop 4 iterations (flag **iterfisher**) are executed by default.

### 5.4.1 Minimizing the Fisher information

The algorithm uses a variational principle to minimize the Fisher information. $(g'(x))^2 = (\frac{d}{dx}g(x))^2$ in 5.12 can be evaluated as gradients in respect to x and y. Assuming that $\nabla_x$ and $\nabla_y$ are finite-difference matrix representations of the partial derivatives[8], then

$$(g'(x))^2 = (\nabla_{\mathbf{x}} g)^T * (\nabla_{\mathbf{x}} g) + (\nabla_{\mathbf{y}} g)^T * (\nabla_{\mathbf{y}} g) \tag{5.13}$$

The weight factor $\frac{1}{g(x)}$ in 5.12 can be introduced using a diagonal weight matrix **W**.

But to avoid nonlinearity we start with

$$W_{ij}^{(1)} = \delta_{ij} \tag{5.14}$$

and use the found $g^{(1)}$ (found after the first iteration) and so on, i.e.

$$W_{ij}^{(n)} = \frac{1}{G_i^{(n-1)}} \cdot \delta_{ij}, \quad n \geqslant 2 \tag{5.15}$$

where

$$G^{(n)} = \sum_t g^{(n)}(t) \tag{5.16}$$

is the over all sum of the timeframes of vectors of emissivity $g$ in a shot. By means of

$$\mathbf{H^{(n)}} = (\nabla_x)^T * \mathbf{W^{(n)}} * (\nabla_x) + (\nabla_y)^T * \mathbf{W^{(n)}} * (\nabla_y), \quad n \in \mathbb{N} \tag{5.17}$$

[8]*Bx* and *By* in the code

5.10 can now be rewritten as

$$\phi^{(n)} = \frac{1}{2}\left(\frac{\mathbf{T}*G^{(n+1)}-F}{\sigma}\right)^T * \left(\frac{\mathbf{T}*G^{(n+1)}-F}{\sigma}\right) + \lambda\left(G^{(n+1)^T} * \mathbf{H^{(n)}} * G^{(n+1)}\right) \tag{5.18}$$

where

$$F = \sum_t f(t) \tag{5.19}$$

is the over all sum of the timeframes of vectors of the chord brightness $f$. Minimizing $\phi^{(n)}$ means

$$\frac{\partial\phi^{(n)}}{\partial G_i} = 0 \quad \forall\, i = 1,..,280 \tag{5.20}$$

leading to the normal equations

$$\left(\frac{\mathbf{T}^T * \mathbf{T}}{\sigma^2} + \lambda\mathbf{H}^{(n)}\right) * G^{(n+1)} = \frac{\mathbf{T}^T * F}{\sigma^2} \tag{5.21}$$

which has to be solved iteratively for $G^{(n+1)}$. In *fabrec.m* [2] this is done by calculating a reconstruction matrix $\mathbf{M}$ (\ stands for left division)

$$\mathbf{M^{(n)}} = \left(\frac{\mathbf{T}^T * \mathbf{T}}{\sigma^2} + \lambda\mathbf{H}^{(n)}\right) \backslash \frac{\mathbf{T}^T}{\sigma} \tag{5.22}$$

The shot-averaged reconstruction matrix $\mathbf{M}$ can now simply be applied to the single timeframe

$$g^{(n)}(t) = \mathbf{M^{(n)}} * f(t) \tag{5.23}$$

### 5.4.2 Establishment of the good $\lambda$

Before the next loop minimizing $I_F$ is executed (using a new weight matrix $\mathbf{W}$ determined by $g^{(n-1)}$) a new $\lambda$ which iterates $\chi^2$ to $\chi^2_{target}$ has to be found.

In general a solution with maximum variance $\sigma$ requires $\chi^2 = 1$. But due to our definition of $\chi^2$ (see formula 5.11) we have summed over the $n_\ell$ enabled bolometer channels (by multiplying the matrices) and also over the $n_t$ timeframes of the shot (by time averaging)

$$\chi^2_{target} = n_\ell \cdot n_t \tag{5.24}$$

For a given iteration n of the invoking Fisher loop the m$^{th}$ subloop simply calculates

$$\lambda^{(m+1)} = \lambda^{(m)} + \frac{d\lambda^{(m)}}{d\chi^2}\left(\chi^2_{target} - \chi^2\right) \tag{5.25}$$

with

$$\frac{d\lambda}{d\chi^2} = \frac{\lambda^{(m-1)} - \lambda^{(m-2)}}{\chi^{2(m-1)} - \chi^{2(m-2)}} \tag{5.26}$$

and

$$\lambda^{(n=1,m=1)} = \lambda_{initial} \cdot \frac{Tr(\mathbf{T})}{Tr(\mathbf{H})}$$
$$\lambda^{(n,m=2)} = \frac{1}{2}\lambda^{(n,m=1)} \tag{5.27}$$

Of course evaluating $\chi^2$ requires the recalculation of the new $g^{(n,m)}(t)$ with the new $\lambda^{(m)}$ as described in the previous paragraph.

## 5.5 Postreconstruction test for bad channels

Due to problems with the weird channel detection algorithm described in chapter 5.2.5, we modified the *fabrec.m* routine in order to test for weird channels after the reconstruction, i.e. by comparing the measured signals and reconstructed emissivity. When we define this difference by

$$\chi = \mathbf{T} * g - f \tag{5.28}$$

then a channel is declared as bad using the criterion

$$|m(\chi)| \geqslant limit \cdot s(m(\chi)) \tag{5.29}$$

where $m(\chi)$ is the mean value over all timeframes and $s(m(\chi))$ its standard deviation for a given channel. By default the parameter **limit** is set to 5.

# Chapter 6

# Illustrations for shot 21515

In this chapter some results for shot 21515 are presented.

The left border of the cross section is oriented to the center of the TCV tokamak.

The figures 6.2 up to 6.5 shows the evolution of the emissivity $g$ for the shot 21515. The colorbar of these figures are all scaled identically. The values are given in [W/m$^3$]. The images have been reconstructed using the default settings of the package.

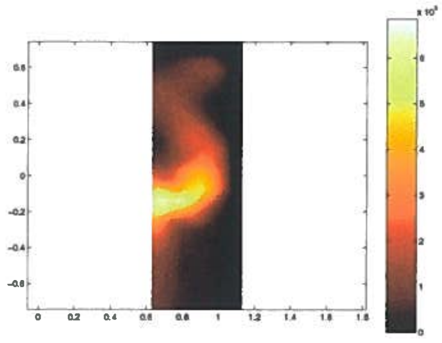


**Figure 6.3: shot 21515 @ t = 0.85 s.**

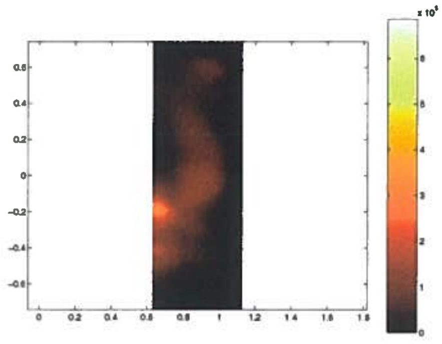


**Figure 6.1: shot 21515 @ t = 0.35 s.**

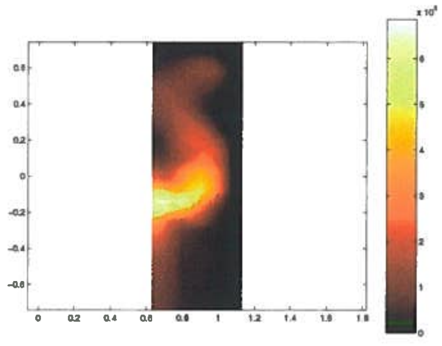


**Figure 6.4: shot 21515 @ t = 1.017 s.**

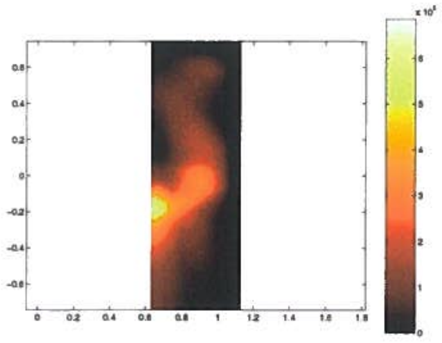


**Figure 6.2: shot 21515 @ t = 0.684 s.**

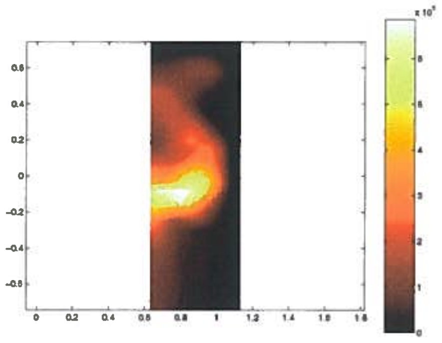


**Figure 6.5: shot 21515 @ t = 1.197 s.**

Figure 6.6 shows the magnetic configuration of the plasma shot at t = 1.197 s. Figure 6.7 shows the evolution of the radiated power for the whole vessel, the x-point and from the divertor region. Figure 6.8 compares the resistive heating power with the total radiated power.

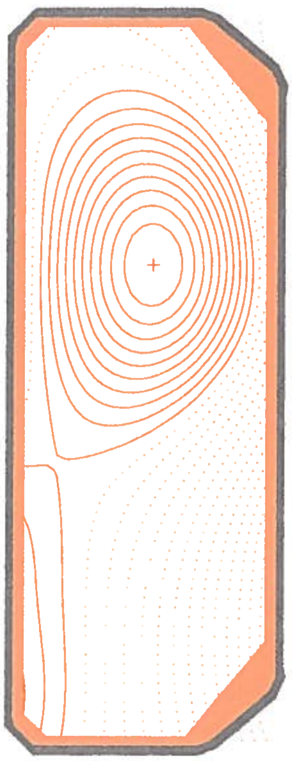


**Figure 6.6: Magnetic flux surface @ 1.197 s.** Plasma shot 21515.

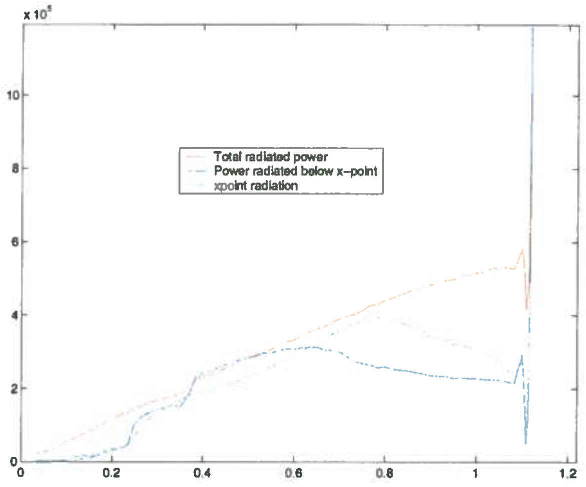


**Figure 6.7: Evolution of the radiated power.** Plasma shot 21515. The three curves represent the total radiated power, the power emitted from the x-point and from the divertor region. Values in [W].

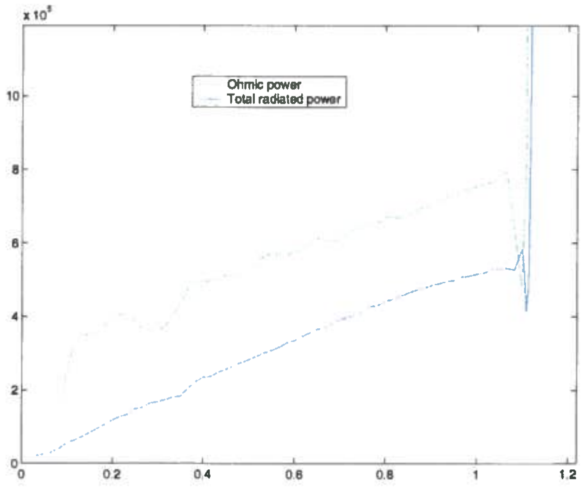


**Figure 6.8: Comparison of the total radiated power and the ohmic heating power.** Plasma shot 21515. Values in [W].

# Chapter 7

# Conclusion

The new package *fabcat.m* has been realized and tested for a large number of shots.

Suitable default parameters were chosen. The results can be stored to the MDS database and simply visualized, so that the package is ready for daily use at CRPP.

The structure of the package is simple enough and its routines have been well commented so that it is ready for future development.

To make the package still more useful, it should be integrated on ANASRV for automatic post-shot processing. Traces of radiated power could then make part of TCV operators scope.

Studies on the latest shots have shown that *fabcat.m* succeeds in about 40 % of the shots to deliver a good quality image reconstruction. Random spot checks have shown that even the *btomo.m* package fails at the same shots. This is due to bad signal data, i.e. systematic problems with the top and bottom bolometer arrays. Regular calibration of the bolometers is necessary, but those above and below the vessel are not easily accessible.

In future we would like to include the flux-shaped pixels method described in [7]. But detailed studies of this novel method within *btomo.m* (where this is already implemented) is necessary first.

## 7.1 Acknowledgements

We would like to thank the TCV group, namely Richard Pitts, Bernard Joye and Basil Duval for their support. Thanks also to all them who contributed with programming to previous packages solving the tomography problem in plasma physics.

Christian Schlatter and Jan Mlynář

Lausanne, february 2002

# References

[1] Iwan Jerjen and Jan Mlynář; **BTOMO: The Software Package Used for the Bolometry on TCV Tokamak**, CRPP EPFL Lausanne, INT 197/99.

[2] Jan Mlynář; **Rapid tomography inversion and its contribution to plasma position measurements at the TCV tokamak**; not published yet.

[3] Jan Mlynář, Ivo Furno, Bernard Joye and Arno Refke: **Bolometry on the TCV Tokamak**, CRPP EPFL Lausanne, INT 196/99.

[4] Martin Anton and al.:**X-ray tomography on the TCV tokamak**, Plasma Physics Controlled Fusion, Volume 38, 1996.

[5] B. R. Frieden: **Application to optics and wave mechanics of the criterion of maximum Cramer-Rao bound**, Journal of Modern Optics, Volume 35, page 1297, 1988.

[6] William H. Press and al:**Numerical recipes in FORTRAN 90**, Cambridge 1996.

[7] S Kalvin and S Zoletnik: **Tomographic Reconstruction of Radiation Intensity Profiles from Bolometric Measurements on the TCV Tokamak**, CAT-Science Bt. for CRPP EPFL Lausanne, INT.

# Appendix A

# Included files and its history

This chapter list all the files included in the *fabcat.m* package. It is also directed at those interested in the history of the code. Here we will explain the modifications of already existing routines (packages *btomo.m* and *cattcv.m*) to meet the requirements of a fast bolometry inversion.

## A.1 Necessary files

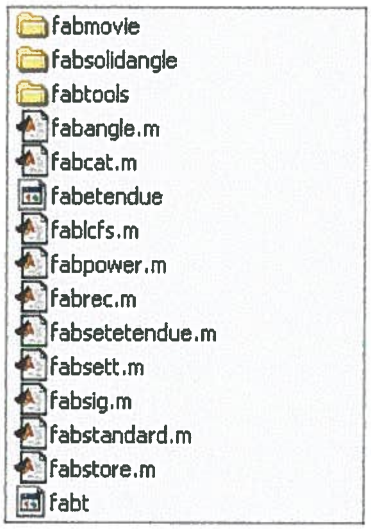

fabmovie
fabsolidangle
fabtools
fabangle.m
fabcat.m
fabetendue
fablcfs.m
fabpower.m
fabrec.m
fabsetetendue.m
fabsett.m
fabsig.m
fabstandard.m
fabstore.m
fabt

**Figure A.1: Necessary package files for** *fabcat.m.* Located in the main directory \ fabcat.

Figure A.1 shows the necessary files to run the package. If the .mat files (the files without an extension in figure A.1) don't exist already, they can be created using the associated .m files.

### A.1.1 fabcat.m

The main program is fundamentally based on the main program of the *cattcv.m* package. All lines of code related to Soft-X ray tomography only have been eliminated. Some elementary setup parameters located in *catsetup.m* for the *cattcv.m* package have been moved to the main program. So an additional setup routine is no longer needed. New, added features are the saving in the MDS tree and the fast calculation of the radiated power.

### A.1.2 fabsig.m

Essential reduction of the routine *btomodata.m* of the *btomo.m* package. All GUI related elements have been removed. Some minor adaption concerning the data types of the variables were necessary.

### A.1.3 fabrec.m

Ancient *fastfish.m* of the *cattcv.m* package. Addition of the i_bord flag. The zero radiation border can now be disabled. Several modifications were necessary (see all '*if i_bord ...*' expressions).

An post-reconstruction $\chi^2$ test disabling bad channels has been added, since the Deschenaux backfiltering method does not always work satisfactory.

The definition of the border pixels has been corrected.

The plasma position measurement has been removed.

### A.1.4 Unchanged routines

The following routines haven't been modified but renamed:

- *btboloti_store_mds.m* ⟶ *fabstore.m* (to store results to MDS),
- *btboloti_pradinout.m* ⟶ *fabpower.m* (to calculate the power radiated inside and outside the LCFS) and
- *bttcv_getlcfs.m* ⟶ *fablcfs.m* (subroutine used by *fabpower.m* to determine the LCFS).

## A.2 Optional files

The files *fabsetetendue.m* and *fabsett.m* are used to create the .mat files *fabetendue* and *fabt* and are located in the main directory (see figure A.1). *fabsett.m* needs the subroutines *fabangle.m*, *fabstandard.m* and the 64 files *fabsolidangle_#.mat* from the subdirectory \*fabsolidangle*. The latter can be created using the *fabsetsolidangle.m* program in this subdirectory (see figure A.2).

The files *fab3d.m*, *fabproject.m* and *fabchord* are necessary to run *fabsetsolidangle.m*. The .mat file *fabchord* can be created using *fabsetchord.m* if it doesn't exist already.

### A.2.1 Unchanged routines

The following routines haven't been modified but renamed:

- *omgrid_3d.m* ⟶ *fab3d.m* and
- *projbl.m* ⟶ *fabproject.m*

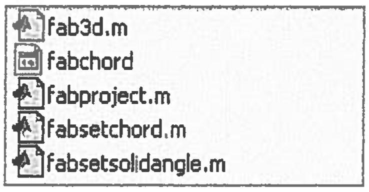

fab3d.m
fabchord
fabproject.m
fabsetchord.m
fabsetsolidangle.m

**Figure A.2: Files to recalculate the solid angle files of the bolometers.** Located in the subdirectory \ fabcat \ fabsolidangle.

- *btt_omgrid_new.m* ⟶ *fabsetsolidangle.m*,
- *omgrid_main_new.m* ⟶ *fabsolidangle.m*

## A.3 Additional tools

In the subdirectories \ fabtools (see figure A.3) and \ fabmovie (see figure A.3) you will find the tools described in chapter 3.2.5.

The *fabmov.m* program is a modified version of the *catmov.m* program for soft X-ray.

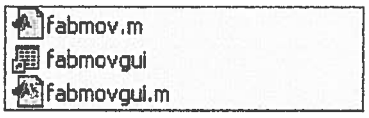

fabmov.m
fabmovgui
fabmovgui.m

**Figure A.3: Fast bolometry result movie player.** Located in the subdirectory \fabcat\fabmovie.

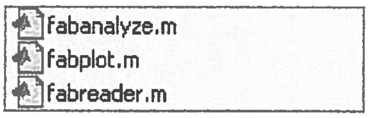

fabanalyze.m
fabplot.m
fabreader.m

**Figure A.4: Additional Tools.** Located in the subdirectory \fabcat\fabtools.

# Appendix B

# The program code

## B.1 Necessary files

### B.1.1 fabcat.m

```
function [Xinv, tvec, xmesh, ymesh, chi2, lambda, ...
          FastPtot, FastPbelow, FastPxpoint, good] = ...
          fabcat(shot,tmin,tmax,points)

% function [Xinv, tvec, xmesh, ymesh, chi2, lambda, ...
%           FastPtot, FastPbelow, FastPxpoint, good] = ...
%           fabcat(shot,tmin,tmax,points)
%
% Algorithm for fast tomography reconstruction for bolometry.
% Package Fast-Algorithm Bolometric Computer Aided Tomography (FABCAT).
% Only one inversion matrix M is calculated per shot.
% If the tmin, tmax times were not specified, all LIUQE times are applied.
% Otherwise all (normal) bolometry time samples within tmin, tmax are
% reconstructed. Points limits the number of timeframes in the reconstruction.
%
% Definition of the X-point:
% The pixel with highest emissivity and its upper, lower and right adjacent
% pixels within pixels 3 to 16 and 31 to 44 (the inferior part of two left
% border columns, the so-called potential X-point region) is are searched
% and excluded.
% ATTENTION : THESE DEFINITIONS DEPEND ON THE CHOSEN PIXEL GRID !!!
%
% See file FABCAT.pdf for further documentation (available on http://crpplocal).
%
% The results can be reached via global variables:
%
% Y              : the processed (calibrated) bolometry signals
% tvec           : the vector of times
% Xinv          : 3-dim emissivity (ny x nx x tvec) reconstruction image [W/m^3]
% xmesh, ymesh : geometrical setup of the pixels [cm]
% chi2, lambda : resulting chi^2 and smoothing factor
%                  length(chi2)=length(tvec)
% FastPtot      : Radiated power inside the whole tokamak [W]
% FastPxpoint   : Radiated power coming from the x-point [W]
% FastPbelow    : Radiated power coming from the divertor region [W]
%                 All radiated power variables have length(tvec)
% good          : Flag indicating if the reconstruction was successful
%
% History:
%
% Christian Schlatter, TPIV CRPP EPFL, February 2002 (first release)
%
% Final release version 1.4 (2002/02/07)

global Y tvec Xinv chi2 lambda xmesh ymesh errflag
global FastPtot FastPbelow FastPxpoint good

disp(' ')
disp('Fast-Algorithm Bolometric Computer Aided Tomography (FABCAT).');
disp('Final release 1.4')
disp('Latest modification on 2002/02/07.')
disp(' ');

debut=cputime;

% User adjustable parameters ****************************************************

f_cam        =        [1 1 1 1 1 1 1 1];
                    % Camera switch. 0 = disabled; 1 = enabled (default).

chantest     = 1;   % 0 : no testing for weird channels.
                    % 1 : testing using Christian Deschenaux's method (default)
                    %     based on backfiltering of the bolometer signals.
                    % 2 : testing using Arno Refke's method
                    %     based on a consistency check of the bolometer signals.
                    % 3 : testing using both methods.

smoothmode = 2;     % 0 : no data processing.
                                % 1 : Bernard Joye's data treatement, based on fitting of
                    %     a polynomial of second degree to the signals (slow).
                    % 2 : Data treatment using Christian Deschenaux's method
                    %     based on backfiltering of the raw signals (default).
                    % All data processing methods calibrate the signal in respect
                    % to the bolometer thermal inertia coefficents TAU.

delchanuser = [];   % User defined channels to exclude from the reconstruction.

runsigma     = 3;   % Errorbar (for the signals) multiplication factor.
                    % The final errorbars are still multiplied by the
                    % bolometer etendue. Default = 3 %

```

```
lambda_init = 0.1; % Initial lambda for the minimum fisher loop (default = 0.1).

lamlim      = .01; % Minimum lambda for a good shot (default = 0.01).

chitar      = 1;   % Target value of chi^2. Default = 1

errchi      = .05; % Supremum of the difference between the time averaged final
                   % chi^2 and the target chi^2 defined by the flag CHITAR.
                   % Default = 0.05

loopfisher  = 3;   % Number of loops of the minimum fisher algorithm to
                   % minimize the Fisher information (default = 3).

 iterfisher = 4;   % Number of iterations of the minimum fisher algorithm
                   % to find the lambda that pushes chi^2 to chitar
                   % (default = 4).

limit       = 5;   % limiting factor for post-reconstruction channel
                   % consistency check (default = 5).

i_bord      = 0;   % 0 : Emissivity contribution coming from the vessels
                   %     border pixels are taken in consideration (default).
                   % 1 : emission form border pixels is set to zero.

x_blind     = 0;   % 1 : Excludes the expanded X-point from the reconstruction
                   %     to increase the contrast of the reconstructed image.
                   %     For the definition of the X-point see flag CUTOFF.
                   % 0 : no X-point exclusion (default).

cutoff      = 0;   % 0 : Automatic mode (default).
                   %     The X-point is searched automatically (see description
                   %     in the header of this file) and finally expanded to
                   %     pixels in the potential X-point region having
                   %     emissivities higher than the percentage defined by
                   %     the flag LEVEL in respect to the X-point.
                   % 1 : Pixel 12 and 13 as expanded X-point.
                   % 2 : Pixel 10 to 16 as expanded X-point.
                   % 3 : Pixel 6 to 18 and 41 as expanded X-point.
                   % 4 : Pixel 5 to 20 and 40 to 43 as expanded X-point.

level       = 80;  % Percentage of the X-points emissivity to establish the
                   % expanded X-point (CUTOFF has to be = 0). Default = 80 [%].

radiation   = 0;   % 1 : The fractions of radiation inside and outside of the
                   %     LCFS are determined using the slow fabpower.m
                   %     routine (originally the btboloti_pradinout.m routine).
                   % 0 : using the fast algorithm to determine FastPtot,
                   %     FastPxpoint and FastPbelow (see header of this file).

```

```
saving      = 0;   % 1 : Stores the variables shot, Xinv, tvec, FastPtot,
                   %     FastPbelow, FastPxpoint and good to the harddisk in
                   %      files  called FABCAT_shotnumber.mat. Use the flag
                   %     SAVEPATH to chose the destination directory.
                   % 0 : no saving to harddisk (default).

savepath    =        '/home/tp47/matlab/FBTR/';
                   % Destination folder for the FABCAT_shotnumber.mat files.
                   % Make sure that you have writing priviliges for the chosen
                   % destination folder. See flag SAVING for more details.

MDS         = 0;   % 1 : The results of the reconstruction are saved to the
                   %     MDS-tree. Make sure you have MDS writing priviliges.
                   % 0 : feature disabled (default).

plotting    = 0;   % 1 : shows a contour plot of the reconstructed emissivity
                   %     for the last timeframe in the shot.
                   % 0 : no plotting (default).

% ******************************************************************************

errflag = 0;
good    = 0;

try

  close('Emissivity matrix for the last timeframe [W/m^3]');

end

if nargin == 1   % LIUQE time mode for a single shot.

  tmin=NaN;
  tmax=NaN;

end

if nargin == 0   % LIUQE time mode for multiple shots.

  tmin=NaN;
  tmax=NaN;
  inputfile=input('Please provide the name of the shot array : ','s');

  try

    load(inputfile);
    disp('The file has successfully been loaded.');

  catch
```

```

    disp('The file doesn''t exist. Try another filename.');
    return

  end

  if ~exist('shot') % Test if the provided file is a reliable shot array

    disp('This is not a shot array.');
    disp('There wasn''t a variable called shot inside.');
    return

  end

  firstshot = shot(1);
  lastshot = shot(length(shot));
  clear shot;

else

  firstshot = shot;
  lastshot = shot;
  clear shot;

end

if nargin < 3

  tmax = NaN;

end

if nargin < 4        % Normal time mode without rescaling of the timeframes.

  points=NaN;

end

for shot = firstshot : lastshot

  % FIRST STEP : GETTING THE SIGNALS

  Y = [];
  tvec = [];
  goodchans = [];
  errstr = [];

  [Y, tvec, errflag, errstr, goodchans] = ...
      fabsig(shot, f_cam, delchanuser, smoothmode, ...
                                    tmin, tmax, points, chantest);

% FABSIG.m gets the signals from the MDS tree, checks for weird channels and
% does data processing (calibration).
% If TMIN, TMAX were not specified, sets TVEC to nearest LIUQE times.
% Otherwise reads the normal times from the MDS tree.
% If POINTS was submitted, the number or timeframes is rescaled to POINTS
% frames.

if errflag           % Shows error message when a problem occured calling FABSIG

  disp('  ');
  disp('  ');
  disp('  ');
  disp(errstr);
  return

end

% SECOND STEP : CHOOSING GEOMETRY

load fabt            % Saved by the optional routine FABSETT.m. Includes
                     % NX, NY (number of pixels); XMIN, XMAX, YMIN, YMAX
                     % (extremal pixel positions); DX, DY (pixel extensions);
                     % XMESH, YMESH (pixels center coordinates); NUMDET
                     % (camera indicator) and T (the T-matrix).

[nt, nl] = size(Y);

                     % NT: number of timeframes.
                     % NL: number of enabled channels.

[usedch, igoodchans, inumdet] = intersect(goodchans, numdet);

                     % Determines the channels which are good and activated.
                     % IGOODCHANS and INUMDET give the index
                                   % of the elements which are common.

% THIRD STEP : DETERMING THE ERRORBARS.

runsigma = runsigma / 100;

                     % change to [%].

dY = runsigma*ones(64,1);

                     % Calculation of DY, the 'relative' error of the signals.

load fabetendue      % Contains the etende of the bolometers in srad*cm^2.
```

```
                    % Calculated using FABSETETENDUE.m.

angfact = 1 ./ etend';

wdy = angfact ./ min(angfact);

                    % Weighting matrix for the errorbars.

dY = dY .* wdy;                  % Errorbars.

Y = Y';                   % because of: size(framedata) = (tvec x goodchans)
                    % but you need size(Y) = (goodchans x tvec).

% FOURTH STEP : EMISSIVITY RECONSTRUCTION.

lambda = [];
chim  = [];
Xinv  = [];
chi2  = [];

xpoint = 1;         % The X-point hasn"t been excluded yet.

[Xinv, lambda, chim, inumdet] = ...
    fabrec(Y, dY, xmesh, ymesh, tvec, inumdet, igoodchans, ...
           T ,lambda_init, loopfisher ,  iterfisher , i_bord, limit );

% FABREC.m is doing the inversion. The reconstructed image is delivered
% by Xinv. Final smoothing factor LAMBDA and chi2 (CHIM) are returned.
% The routine is checking again for weird channels using a post-
% reconstruction test on chi^2 (see flag LIMIT).

Xinv=Xinv*100;  % Conversion to [W/m^3].

if plotting & ~x_blind

                    % Shows a contour plot of the found emissivity of the
                    % first inversion if no second inversion is planned.

  figure('Name','Emissivity matrix for the last timeframe [W/m^3]');
  contourf(xmesh/100, ymesh/100, Xinv(:,:,length(tvec )), 50), ...
                    shading flat, axis equal
  colorbar;

end

% FIFTH STEP : RECONSTRUCTION RELIABILITY TESTING

chi2 = mean(chim); % chi^2 averaged over all timeframes.
```

```
disp('The inversion has finished');

if lambda < lamlim | abs(chi2 − 1) > errchi

                     % Test if shot reconstruction is acceptable.

  disp(' ');
  result = ['Results: lambda=' num2str(lambda) ', mean(chi2)=' num2str(chi2)];
  disp(result);
  disp(['WARNING: reconstruction seems to be wrong in shot ' ...
                 int2str(shot) ' !']);
  disp(' ');
  errflag = 1;

else

  disp(' ');
  result = ['shot ',int2str(shot),' OK. (lambda = ', num2str(lambda), ...
                 ', chi2 = ', num2str(chi2),')'];
  disp(result);
  disp(' ');
  good = 1;

end

% SIXTH STEP : QUEST FOR THE X−POINT.

X = reshape(Xinv, ny ∗ nx, length(tvec));
[K,J] = find(X == max(max(X)));

                     % K : Pixel containing the X−point.

Q = union(3:16,31:44);

                     % Q : Potential X−point region.

K = intersect(K,Q);

                     % Limiting the X−point to the potential X−point region.

if x_blind           % Optional X−point exclusion

  disp(' You have chosen to exlude the X-point');
  disp(' Running a second inversion...');
  disp(' ');

  xpoint = 0;

  if isempty(K) % When the point with highest emissivity lies autside the
```

```
                    % potential X−point region.

      disp('Identification of X-point impossible');
      xpoint = 1;

    else

      switch cutoff % Determination of the expanded X−point (multiple pixels)

        case 0

          [pixel, J] = find(X > (level / 100) * max(max(X)));

                    % LEVEL specifies the inferior radiation percentage limit
                    % for pixels participating to the expanded X−point.

          pixel = intersect(pixel, Q);

        case 1        % Overrides the automatic X−point determination.

          pixel  = [12, 13];

        case 2

          pixel   = [10, 11, 12, 13, 14, 15, 16];

        case 3

          pixel     = [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 41];

        case 4

          pixel      = [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, ...
                       20, 40, 41, 42, 43];

      end

      g = reshape(Xinv / 100, ny * nx, length(tvec));

      contX = T(inumdet, pixel) * g(pixel, :);

                    % Contribution of the expanded X−point to the chord
                    % brightness Y.

      Y = Y − contX;

                    % New chord brightness without emissivity from the expanded
                    % X−point.
      [Xinv, lambda, chim, inumdet] = ...
          fabrec(Y, dY, xmesh, ymesh, tvec, inumdet, igoodchans, ...
                   T ,lambda_init, loopfisher ,  iterfisher , i_bord, limit );

      % Second tomography inversion without expanded X-point contributions.

      Xinv=Xinv*100;  % Conversion to [W/m^3]
      chi2=mean(chim); % chi^2 averaged over all timeframes.

      if  plotting        % Shows a contour plot of the found emissivity.

        figure('Name','Emissivity matrix for the last timeframe [W/m^3]');
        contourf(xmesh/100, ymesh/100, Xinv(:,:,length(tvec )), 50), ...
                    shading flat, axis equal
        colorbar;

      end

      disp('The second inversion has finished');

      if lambda < lamlim | abs(chi2 - 1) > errchi

                    % Test if shot reconstruction is acceptable.

        disp(' ');
        result = ['  Results: lambda=' num2str(lambda) ', mean(chi2)=' num2str(chi2)];
        disp(result );
        disp([' WARNING: second reconstruction not successful for shot ' int2str(shot) ' !'
        disp(' ');
        errflag =1;

      else

        disp(' ');
        result = ['shot ',int2str(shot),' OK. (lambda = ', num2str(lambda), ...
                    ', chi2 = ', num2str(chi2),')'];
        good = 1;
        disp(result );
        disp(' ');

      end

    end

  end

  X = reshape(Xinv, ny*nx, length(tvec));

  rm = xmesh / 100;     % Conversion to [m]
```

```
zm = ymesh / 100;      % Conversion to [m]

if radiation           % Optional calculation of the radiation fractions issued inside and
                       % outside of the LCFS surface.

  disp(['Calculation of the LCFS fractions of the radiation started.']);

     [Ptot, Pin, Pout, Pabove, Pbelow, tvec] = ...
                        fabpower(X, shot, tvec, xmesh, ymesh, 1);

end

mdsclose;

if saving              % Optinal saving of variables on harddisk.

  savefile =['FABCAT_',int2str(shot)];

  try

    eval(['save ',[savepath,savefile ], ' shot Xinv tvec FastPtot FastPbelow FastPxpoint good
    disp(['The following file has been created: ', savepath savefile]);

  end

end

% SEVENTH STEP : Fast calculation of the radiated power

for k = 1 : nx

  Pinv(:,k,:) = Xinv(:,k,:)*xmesh(k);

                          % Integrand for later integration

end

fact=2*pi*dx*dy/1e6;  % Integration factor (over toroidal ring).

P = reshape(Pinv,nx*ny,length(tvec));

                          % Pixel power matrix

FastPtot = sum(P)*fact; % Sum over all pixels

if ~isempty(K)
  FastPbelow = (sum(P(1:(K-2),:)) + sum(P(29:(K+26),:))) * fact;

                          % Power radiated from the divertor region.
```

```

      FastPxpoint = (sum(P(K−1:K+1,:)) + sum(P(K+27:K+29,:))+P(K+56,:)) ∗ fact;

                                  % Power radiated from the expanded X−point.

    else

      FastPbelow = NaN;
      FastPxpoint = NaN;

    end

    if MDS                        % Optional saving to the MDS tree.

      if xpoint                   % Comments for MDS node 'COMMENT'

        mtxt = ['FBTR with x-point at pixel ', num2str(K)];

      else

        mtxt = ['FBTR without x-point at pixel ', num2str(K)];

      end

      if isempty(K)               % No X−point found.

        mtxt = ['FBTR, x-point determination failed'];

      end

      fabstore(shot, rm, zm, tvec, X, lambda, runsigma, chim, points, setdiff (1:64,inumdet), mtxt, resul

    end

end

time_end = cputime − debut;
disp(['CPU runtime: ' num2str(time_end),' s.']);
disp(' ');
```

### B.1.2 fabsig.m

```
function [framedata,frametbase,errflag, errstr ,goodchans] = ...
       fabsig(btshotnum,f_cam,delchanuser,smoothmode,tmin,tmax,points,chantest)

% function [framedata,frametbase,errflag, errstr ,goodchans] = ...
%       fabsig(btshotnum,f_cam,delchanuser,smoothmode,tmin,tmax,points,chantest)
%
% Gets the bolometry signals for the tomography analysis, checks for bad
% bolometers and does data processing (calibration). This is the fast version.
%
% This function has been rewritten to satisfy the needs of bolometry.
% It is based on btomodata.m written by Jan Mlynar in 1999.
%
% If tmin is a number, it takes all time samples within tmin and tmax.
% Otherwise it loads data only at LIUQE times
%
% History:
%
% Christian Schlatter, TPIV CRPP EPFL, February 2002
% major parts stolen from the btomo package
%
% Release 1.4 (2002/02/07)

if nargin<5 tmin = NaN; end
if nargin<7 points = NaN; end

fprintf('FABCAT signal processing routine. Release 1.4 (2002/02/07)\n');

fprintf('Loading of the bolometry signals.\n\n');

errflag = 0;                      % Global error flag
errstr = '';
sig = []; tvec = []; delchans=[];

% Get default setup values

smoofac = 15;                     % Bernard Joyce's data treatment method is
                                  % fitting a polynomial over 2*SMOOFAC timeframes.
                                  % Default = 15 (should not be changed).
 filtfreq = 50;                   % Frequency of the butterworth 4th order digital
                                  % filter used by Christian Deschenaux's channel
                                  % testing and data treatment methods.
                                  % Default = 50 [Hz] (should not be changed).

% ----------------------------------------------------------------
% Signal reading and scaling


if isempty(btshotnum)             % Test if shotnumber was given.

  errflag = 1;
  errstr = (['You didn''t specify a shot number !!!']);
  mdsclose;
  return

end

```

```
mdsopen('tcv_shot',btshotnum);   % Opening the MDS tree for reading.

btalldata=mdsdata('\base::bolo:signals');

                                  % Bolometry signal node.

tau = mdsdata('\base::bolo:tau'); % Loading the bolometer thermal inertia
                                  % constants (for calibration ).

if isempty(btalldata)             % Check if the shot exists in MDS.

  errflag = 1;
  errstr = (['Sorry, no data available for shot ',int2str(btshotnum)]);
  mdsclose;
  return

end

% *** If TMIN was not specified --> LIUQE times are loaded. ********************************

if isnan(tmin)

  bttimebase=mdsdata(['dim_of(\base::bolo:signals)']);

                                  % Loading the NORMAL time base.

  liuqetbase=mdsdata('dim_of(\results::r_axis)');

                                  % Loading the LIUQE time base.

  mdsclose;

  ii=1;                           % Indicator for NORMAL times nearest to LIUQE times.
  base=[];
  tbase=[];

  for k=1:length(liuqetbase)

    base(ii)=max(find(abs(bttimebase-liuqetbase(k))==min(abs(bttimebase-liuqetbase(k)))));

                                  % Index vector of NORMAL times nearest to LIUQE times.

        if ii==1                     % First nearest timeframe.

      tbase(ii)=bttimebase(base(ii));
          ii=ii+1;

    else                          % Following nearest timeframes.

```

```
         if bttimebase(base(ii))>tbase(ii-1)

           tbase( ii )=bttimebase(base(ii));
                         ii=ii+1;

         end

       end

     end

     frametbase=tbase';

     [vec,framebase,ib]=intersect(bttimebase,frametbase);

                                        % Times matching the time interval of the shots timeframes.

     [bttotsteps,bttotchan]=size(btalldata);


   % *** If TMIN was specified -> NORMAL times are loaded. ***********************************

   else

     bttimebase=mdsdata(['dim_of(\base::bolo:signals)']);

                                        % Loading the NORMAL time base.

     mdsclose;

     [bttotsteps,bttotchan]=size(btalldata);

     if isnan(tmax)                     % If TMAX was not specified -> upper time limit is the
                                        % latest timeframe for the given shot.

       tmax = bttimebase(length(bttimebase)-1);

     end

     if tmax > bttimebase(length(bttimebase))

       tmax = bttimebase(length(bttimebase)-1);

     end

     if ~isnan(points)                  % Reducing the number of timeframes to POINTS timeframes
                                        % if points have been submitted.

             minstep = min(find(bttimebase>=tmin));
```

```
            maxstep = max(find(bttimebase<=tmax));
            framebase = round(minstep:(maxstep−minstep)/(points−1):maxstep);

    else                                   % Taking all the timeframes matching the given time interval.

      framebase = find(bttimebase>=tmin & bttimebase<=tmax);
      points = length(framebase);

    end

    frametbase = bttimebase(framebase);

  end


  % −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
  % Channel testing and deleting

  if chantest ~= 0

    fprintf('Looking for bad channels:\n\n');

  end

  mdsopen('tcv_shot',btshotnum);

  bolosig = mdsdata('\base::bolo:source');     % Reading of the unprocessed (raw) bolometry signals
  mdsclose;

  if ~isempty(delchanuser)                     % Channels deleted by the user (flag DELCHANUSEI
                                               % 'fabcat.m'

    fprintf('  You have disabled the following channels:\n ');
    fprintf('%1.0f,', delchanuser');
    fprintf('\n\n');

  end


  % *** Automated deleting of weird channels using Arno Refke's method *********************
  % *** This is the channel consistency check

  if ((chantest == 2) | (chantest == 3))

    offset = mean(bolosig(1:80,:));               % Offset : Mean value of the signals
    bolosig = bolosig − repmat(offset,bttotsteps,1);
    difbolo = diff(bolosig);                      % Slope of the signals.
    gaga1 = find((max(bolosig)−min(bolosig))<0.5); % Signal value range check.
    gaga2 = find(mean(bolosig)<0);                % Negative signal check.
```

```
 gaga3 = find((max(difbolo)−min(difbolo))>4);    % Signal slope check.
 gaga4 = find(abs(offset)>9);                    % Signal offset check.
 gaga = union(gaga1,gaga2);
 gaga = union(gaga,gaga3);
 gaga = union(gaga,gaga4);

 if ~isempty(gaga)

   fprintf(' Arno Refke''s method disabled the following channels:\n ');
       fprintf('%1.0f,', gaga');
       fprintf('\n\n');

 end

 delchans=union(gaga,delchanuser);

end

% *** Automated deleting of weird channels using Christian Deschenaux's method ************
% *** This is the signal backfiltering method

if ((chantest == 1) | (chantest == 3))

 p = btalldata';
 amp=.01;
 [b,a]=butter(4,min(1,filtfreq /(2000/2)));      % Design of a butterworth
                                                 % digital filter of 4th order.

 [py,px]=size(p);
 fltr =zeros(size(bttotchan));

 for i=1:bttotchan

   p(i,:)= filtfilt (b,a,p(i ,:));               % Filtering of the signals.
        fltr1 (i)=mean(p(i,[1:min(100,bttotsteps)]));
       m=abs(mean(p(i,:)));

       if m>0                                       % Filtered signal mean value sign check.

     fltr2 (i)=mean(abs(diff(p(i,:))))/m;        % Filtered signal slope check

   else

     fltr2 (i)=0;

   end;

 end;

```

```
    fltr1 =abs(fltr1)>0.1;
    fltr2 =(fltr2<(mean(fltr2)*amp));
    fltr =find(fltr1 | fltr2 );

    if ~isempty(fltr)

      fprintf('  Christian Deschenaux''s method disabled the following channels:\n ');
      fprintf('%1.0f,', fltr ');
      fprintf('\n\n');

    end

    delchans=union(fltr,delchans);
    delchans=union(delchans,delchanuser);

end

goodchans=setdiff([1:bttotchan],delchans);


%---------------------------------------------------------------------
% Data treatment (calibration)

if smoothmode

  sweep=smoofac;

else

  sweep=1;

end

if framebase(1)<sweep+1                          % Time compatibility check,
                                                 % to test  if methods are applicable.

  errflag = 1;
  errstr = (['The smoothing requires higher start time.']);
  return

elseif length(framebase) > bttotsteps - sweep

  errflag = 1;
  errstr = (['The smoothing requires lower stop time.']);
  return

end

rawdata=btalldata(framebase,goodchans); % The raw, uncalibrated signals.
```

```

framedata=NaN.*ones(size(rawdata));

% *** No data treatment ****************************************************

if smoothmode == 0

  framedata=1.78e4*rawdata;                % Calculation of the chord brigthness

end

% *** Bernard Joye's method *************************************************
% *** This is the second order polynomial fitting method

if smoothmode == 1

  fprintf('  Smoothing using Bernard Joye''s method.\n');
   offset  = mean(btalldata(1:80,goodchans));
  basedata = 1.78e4*(btalldata(:,goodchans)...
                        - repmat(offset,bttotsteps,1));

                                           % Uncalibrated chord brightness.

  per_acq=(bttimebase(bttotsteps)-bttimebase(1))/(bttotsteps-1);

                                           % mean timestep time.
  xvec=-smoofac:smoofac;                   % Fitting index.

  bolocor=NaN.*ones(1,bttotsteps);

  for stn=1:length(framebase)              % Fitting timeframe window.

        k=framebase(stn);
          yvec=basedata((k-smoofac):(k+smoofac),:);

    for chn=1:length(goodchans)            % Fitting signals using a polinomial
                                           % of second degree taking in account
                                           % of the bolometers time constants TAU

      p=polyfit(xvec,yvec(:,chn)',2);
            framedata(stn,chn)=p(3)+...
             tau(goodchans(chn))*p(2)/per_acq;

    end

  end

end

```

```
% *** Christian Deschenaux's method ********************************************
% *** This is the backfiltering method

if smoothmode == 2

  fprintf(' Smoothing using Christian Deschenaux''s method.\n');

  [b,a]=butter(4,min(1,filtfreq/1000));          % Design of a butterworth
                                                 % digital filter of 4th order.
  unity=ones(size(framebase));

  difft=bttimebase(framebase+unity)-bttimebase(framebase-unity);

                                                 % Neighboring fimeframes
                                                 % time intervals.

  for ch=1:length(goodchans)

    pall= filtfilt (b,a,btalldata(:,goodchans(ch)));  % Filtering of the signals.
          psel=pall(framebase);
          diffp=pall(framebase+unity)-pall(framebase-unity);

                                                 % Derivation of the filtered signals.
          framedata(:,ch)=1.78e4*...
                          (psel+tau(goodchans(ch))*diffp./difft);

                                                 % Calibration using time constants TAU.

  end

end

fprintf('\n');
```

### B.1.3 fabrec.m

```
function [gres, lamout, chi2out, actt ] = ...
        fabrec(Y, dY, xmesh, ymesh, tvec, actt, actdet, Tb, ...
               lambdai, ifishmax, iimax, i_bord, limit);

% function [gres, lamout, chi2out, actt ] = ...
%         fabrec(Y, dY, xmesh, ymesh, tvec, actt, actdet, Tb, ...
%                lambdai, ifishmax, iimax, i_bord, limit);
%
% Tomographic reconstruction of the emissivity for the given
% signals and geometry. The reconstruction is based on minimizing
% the Fisher information and uses a single reconstruction matrix
% which is averaged over all timeframes.
%
```

```
% The routine does a post-reconstruction test for bad channels and exclude
% them from the reconstruction.
%
% Outputs: gres     : the emissivity array, size (ny x nx x length(tvec))
%          lamout   : the final smoothing factor LAMBDA
%          chi2out  : The final chi^2, length(chi2out) = lenght(tsteps)
%          actt     : The array of enabled bolometer channels
%
% Please not that the average of chi2 = mean(chi2out) is targeted to 1,
% representing then the solution with maximum variance (dY).
%
% History:
%
% Jan Mlynarr, CRPP EPFL, May 2001 (first release)
% dito, January 2002 (crosscheck of good channels)
%
% Final release version 1.4 (2002/02/07)

fprintf('FABCAT inversion routine. Release 1.4 (2002/02/07)\n');

while 1 % Repetitive execution until no more weird channels are discovered
        % by the post-reconstruction check (see at the end of this routine)

lambda=lambdai;
To=Tb(actt,:);                  % Restrictions to enabled bolometers
sigmao=dY(actt);
YY=Y(actdet,:);

tsteps=length(tvec);
[nl,npix]=size(To);

if i_bord                       % no border condition

  chi2tar=(nl+1)*tsteps;        % chi2 per a sample should be 1
                                % (nl+1) one in plus - see the zero border below
else

  chi2tar=nl*tsteps;            % chi2 per a sample should be 1

end

dx=xmesh(2)-xmesh(1);
dy=ymesh(2)-ymesh(1);
nx=length(xmesh);
ny=length(ymesh);

% START of the fast interim of of the cattcv package 'makem_1.m'
% only used when the no border condition is enabled.

```

```
epsilon=sqrt(eps);

% Border pixel determination

iborder=[1:ny,ny+1:ny:((ny*(nx-2))+1),(2:(nx-1))*ny,(1+(nx-1)*ny):nx*ny];
nborder=length(iborder);

Ymax=max(YY);
D=max(YY./repmat(Ymax,nl,1),epsilon*ones(nl,tsteps));

% Default model for the tought-for emissivity distribution.

Tbar=sum(To(:))/(nl+npix);
Ep=D./repmat(sum(To')',1,tsteps);
nn=find(sum(To)~=0);
Eca=(Ep'*To(:,nn))./repmat(sum(To(:,nn)),tsteps,1);

% Totally flat model with zero border.

I0=sum(Eca');
Emean=I0./(npix-nborder);
g_model=repmat(Emean,npix,1);
g_model(iborder,:)=1e-9*ones(length(iborder),tsteps);
g_model=g_model.*repmat(Ymax,npix,1);

% END of fast interim of "makem_1.m"

tiny=1e-4;
dfadd=min(sigmao)/100; % Relative error of the lines of sight

i_pocs=1;
rgmin=1e-2;

diam=eye(npix);                          % diagonal
diar=diag(ones(1,npix-ny),ny);          % to reference right nearest neighbors
dial=diag(ones(1,npix-ny),-ny);         % reference left nearest neighbors
diao=diag(ones(1,npix-1),1);            % reference upper nearest neighbors
diau=diag(ones(1,npix-1),-1);           % reference lower nearest neighbors

% Indices for borders and coins

i_bl=2:ny-1;                          % left border
i_br=(2+(nx-1)*ny):(npix-1);      % right border
i_bo=(2:(nx-1))*ny;                   % upper border
i_bu=1+(1:nx-2)*ny;                   % lower border
i_ul=1;                                  % lower left corner
i_ol=ny;                                 % upper left corner
i_or=npix;                               % upper right corner
i_ur=1+(nx-1)*ny;                     % lower right corner
```

```
i_oben=[i_ol,i_bo,i_or];
 i_rechts=[i_ur,i_br,i_or];

% Evaluation of the gradients

Bx=-diam+diar;                    % finite-difference matrix representations
By=-diam+diao;                    % of the partial derivates.

By(i_oben,:)=diam(i_oben,:)+diau(i_oben,:);
Bx(i_rechts,:)=diam(i_rechts,:)+dial(i_rechts ,:);

Bx=Bx/dx;
By=By/dy;

% Normalization and division by the variance

fo=Y(actdet,:);
noemiss=[];                       % border fixed to zero:

if i_bord                         % border pixels emissivity set to zero.

  noemiss=find(g_model<=tiny*max(max(g_model)));
  Tadd=zeros(1,npix);
  inx=find(noemiss<npix);
  Tadd(noemiss(inx))=ones(size(inx));
  fadd=zeros(1,tsteps);
  T=[To;Tadd];
  f=[fo;fadd];
  sigma=[sigmao;dfadd];

else

  f = fo;
  T = To;
  sigma = sigmao;

end

[nl,npix]=size(T);      % size of T changes for the fixed border

fmax = max(f);
f = f./repmat(fmax,size(f,1),1);      % Normalization

S1=diag(1./sigma);                % Division by the variance
TT=zeros(npix);
Ts=S1*T;
TT=Ts'*Ts;

f(find(f<0)) = zeros(size(find(f<0))); % no negatives emissions:
```

```
fs = f./repmat(sigma,1,tsteps);

lam=zeros(ifishmax,iimax);
chi2=lam;
chi2det=zeros(ifishmax,iimax,tsteps);
chi2b=zeros(ifishmax,tsteps);

% *** Mainloop to minimize the Fisher information

for i=1:ifishmax

  if i==1 % First loop

    w = ones(npix,1);                 % unity weight matrix
                                      % The first iteration corresponds
                                      % to linear regularization
        H = Bx'*Bx+By'*By;                   % The H-matrix
    coeflam=trace(TT)/trace(H);       % inital LAMBDA
    lam(1,1)=lambda*coeflam;
    A=(TT+lam(1,1)*H)';
        Tpsinv=A\Ts';                     % The M-matrix
        g=Tpsinv*fs;                      % The emissivity

    neg=find(g<0);
    g(neg)=zeros(size(neg));          % Negative emissivity is set to zero
    g(noemiss)=zeros(size(noemiss));  % the same with noemission pixels

  else

        gdummy=sum(g,2)/max(sum(g,2));
        petit=find(gdummy<rgmin);
        gdummy(petit)=rgmin*ones(size(petit));
        w=1./gdummy;                      % The weight matrix W(n), n>1
    Ax=diag(w)*Bx;
    Ay=diag(w)*By;
        H=Bx'*Ax+By'*Ay;                      % H(n), n>1

    lam(i,1)=lambdab(i-1);
    A=(TT+lam(i,1)*H)';
        Tpsinv=A\Ts';                     % M(n), n>1
    g=Tpsinv*fs;

    neg=find(g<0);
    g(neg)=zeros(size(neg));
    g(noemiss)=zeros(size(noemiss));

  end

  lam(i,2)=lam(i,1)/2;
```

```
 fc=Ts*g;                             % Reconstructed chord brightness
 dummy=fc-fs;
 chi2det(i,1,:)=sum(dummy.*dummy)'; % (ifishmax * iimax * tsteps) matrix
 chi2(i,1)=sum(chi2det(i,1,:));

% *** Subloop to find LAMBDA which delivers chi^2 -> target chi^2

 for ii=2:iimax

   A=(TT+lam(i,ii)*H)';
   Tpsinv=A\Ts';
   g=Tpsinv*fs;
       neg=find(g<0);
       g(neg)=zeros(size(neg));
       g(noemiss)=zeros(size(noemiss));
   fc=Ts*g;
   dummy=fc-fs;
       chi2det(i,ii,:)=sum(dummy.*dummy)';
   chi2(i,ii)=sum(chi2det(i,ii,:));

   if ii==iimax break; end           % If last subloop iteration reached

   deriv=(lam(i,ii)-lam(i,ii-1))/(chi2(i,ii)-chi2(i,ii-1));
   coef=lam(i,ii)-deriv*chi2(i,ii);
       lam(i,ii+1)=deriv*chi2tar+coef;  % The new lambda

   if lam(i,ii+1)<=0 lam(i,ii+1)=lam(i,ii)/2; end

       ii=ii+1;

 end % of the subloop

 [noth,stepno]=min(abs(chi2(i,:)-chi2tar*ones(size(chi2(i,:)))));

                                     % find lambda which makes chi2 closest to chi2tar

 chi2b(i,:)=chi2det(i,stepno,:);
 lambdab(i)=lam(i,stepno);           % Best LAMBDA so far
 A=(TT+lambdab(i)*H)';
 Tpsinv=A\Ts';                       % Best M-matrix so far
 g=Tpsinv*fs;                        % Best emissivity reconstruction so far
 neg=find(g<0);
 g(neg)=zeros(size(neg));
 g(noemiss)=zeros(size(noemiss));

end % of the main loop

lambda=lam/coeflam;                  % (ifishmax * iimax) matrix, describes the iterative
                                     % evolution of smoothing
```

```
lamout=lambda(ifishmax,iimax);        % Final LAMBDA
chi2out=chi2b(ifishmax,:)/nl;         % Final chi^2 vector

gres=reshape(g.*repmat(fmax,npix,1),ny,nx,length(tvec));

                                      % Final emissivity

% Post-reconstruction bad channel detection

mdum=mean(dummy,2);                   % Mean chi
stdd=std(mdum);                       % standard deviation of mean chi
goodch=find(abs(mdum)<limit*stdd); % LIMIT determines the confidential channels

 if length(goodch)==length(mdum) break; end

delch=find(abs(mdum)>=limit*stdd);
fprintf('\n');
fprintf('  Channels deleted by the chi2 test : ');
fprintf('%1.0f,', actt(delch));
fprintf('\n\n');

actt=actt(goodch);
actdet=actdet(goodch);

end
```

### B.1.4 fabstore.m

```
function fabstore(shot, rm, zm, time, X, lambda, Sigma, ...
                logchi2, smoothing, nchords, mtxt, ctxt, Ptot , ...
                Pin, Pout, Pabove, Pbelow);

% function fabstore(shot, rm, zm, time, X, lambda, Sigma, ...
%               logchi2, smoothing,       nchords, mtxt, ctxt, Ptot , ...
%               Pin, Pout, Pabove, Pbelow);
%
% Stores the results of the reconstruction in the MDS tree.
% ATTENTION: whatever may be there will be overwritten.
%
% This routine has been taken from the boloti package.
%
% Arguments:
%
%       shot        the shotnumber                        [1 x 1]
%       rm              pixel coordinates                 [1 x nr]
%       zm          idem                                      [1 x nz]
%       time        times for the slices inverted    [1 x timesteps]
%       X               normalised emissivity (max=1)       [npixels x timesteps]
%       lambda      smoothing factor of reconstr.    [1 x timesteps]
%         Sigma         data error  in %                     [1 x 1]
%         logchi2       log10(chi2/length(nchords))        [3 x timesteps]
%         smoothing     number of points averaged in     [1 x 1]
%                           bolo-data treatment
%         nchords       number of bolo-chords used         [1 x 1]
%         mtxt          comment on inversion method            string
%         Ptot          total  radiated power [Pin+Pout] [1 x timesteps]
%         Pin           radiated power inside LCFS         [1 x timesteps]
%         Pout          radiated power outside LCFS      [1 x timesteps]
%         Pabove        radiated power above LCFS          [1 x timesteps]
%         Pbelow        radiated power below LCFS          [1 x timesteps]
%
% All radiation given in [W]
%
% History:
%
%  Arno Refke, CRPP EPFL, 20 of November, 1998 (first release)
%
% Final release  version 1.4 (2002/02/07)

if nargin~=17

   error('fabstore.m: sorry, incorrect number of input arguments')
       return

end

cmd1 = 'build_signal(build_with_units(f_float($1),"W"),*,f_float($2))';

mdsopen('results',shot)

mdsput('\results::btomo:method',mtxt,'t');
mdsput('\results::btomo:comment',ctxt,'t');

mdsput('\results::btomo:rmesh',rm,'f');
mdsput('\results::btomo:zmesh',zm,'f');
mdsput('\results::btomo:nchord',nchords,'f');
mdsput('\results::btomo:lambda',lambda,'f');
mdsput('\results::btomo:sigma',Sigma,'f');
mdsput('\results::btomo:chi_squared',logchi2,'f');
mdsput('\results::btomo:smoothing',smoothing,'f');
mdsput('\results::btomo:time',time,'f');
mdsput('\results::btomo:emissivity',.....
        'Build_signal($1,*,*,\results::btomo:time)','x',X);

mdsput('\results::btomo:prad_above',cmd1,'x',Pabove,time);
mdsput('\results::btomo:prad_below',cmd1,'x',Pbelow,time);
mdsput('\results::btomo:prad_in',cmd1,'x',Pin,time);
mdsput('\results::btomo:prad_out',cmd1,'x',Pout,time);
mdsput('\results::btomo:prad_tot',cmd1,'x',Ptot,time);

mdsclose;

return
```

## B.2 Optional files

### B.2.1 fabpower.m

```
function [Ptot, Pin, Pout, Pabove, Pbelow, tvec] = ...
                fabpower(X, shot, tvec, rm, zm, iplot);

% function [Ptot, Pin, Pout, Pabove, Pbelow, tvec] = ...
%                fabpower(X, shot, tvec, rm, zm, iplot);
%
% Calculates the cell- and grid-points of a selected
% mesh (rm, zm in [cm]) for tomographic inverson of
% the bolometric data lying inside/outside the LCFS.
% Also determines the cutpoints of the lcfs with the tomogrid
% and calculates Ptot, Pin, Pout, and Pdiv from the power
% radiated in-/outside the lcfs from each cell
%
% This routine was taken from the 'btomo.m' package
%
% History:
%
% Arno Refke, CRPP EPFL, 18 of September, 1996 (first release)
% Iwan Jerjen, CRPP EPFL, 26 of August, 1999 (minor modifications)
%
% Final release version 1.4 (2002/02/07)

global ha_figrad          %handle of figure

if nargin < 6
    iplot = 0;
    else iplot=iplot;
end

% Normalize

calf=max(X(:)); % scalar
X=X/calf;

% Create vector Pm to every mesh-point

nx = length(rm);              % number of x-cells
ny = length(zm);              % number of y cells
dx = (rm(2)-rm(1));           % cell size in x
```

```
dy = (zm(2)-zm(1));             % cell size in y
xmax = rm(nx)+dx/2;             % maximum x mesh point
xmin = rm(1)-dx/2;              % minimum x mesh point
ymax = zm(ny)+dy/2;             % maximum y mesh point
ymin = ymax-ny*dy;              % minimum y mesh point
Acell = dx*dy;                  % cell area;

xgrid = [xmin:dx:xmax+0.001];            % vector of x grid points
ygrid = fliplr ([ymin:dy:ymax+0.001]);  % vector of y grid points

mx = length(xgrid);             % number of x meshgrid points
my = length(ygrid);             % number of y meshgrid-points
km = mx*my;                     % total number of mesh points

Pm=zeros(km,3);
for j=1:my
Pm((mx*(j-1)+1):(j*mx),1) = xgrid';
end
for j=1:mx
Pm(j:mx:(mx*(my-1)+j),2) = ygrid';
end

% Create vector Pc to every cell center-point
% Only xcell and kc are used

xcell = rm;                     % vector of x cell center points
kc = nx*ny;                     % number of cell_points

if length(calf) == 1, calf = calf*ones(length(tvec),1); end;

% Loop over all times specified in the shot

for l=1:length(tvec);

% Vector L to every point of the LCFS

[con_xt,con_yt,c_xt,c_yt,times]=fablcfs(shot,tvec(l));

I = find(~isnan(con_xt) & ~isnan(con_yt));
con_xt = con_xt(I);
con_yt = con_yt(I);

lcfs_x = con_xt;                % vector of lcfs x-points
lcfs_y = con_yt;                % vector of lcfs y-points
y_Xpt = min(lcfs_y);    % y - coordinate of the X-point;
y_Tpt = max(lcfs_y);    % y - coordinate of the top-point from LCFS;
i = length(lcfs_x);             % number of lcfs-points
L=zeros(i,2);
L(:,1) = lcfs_x ;
```

```
L(:,2) = lcfs_y ;

% Matrix Rm with vectors from each mesh point to each lcfs-point

Rm = zeros(i,2*km);
for j=1:2:(2*km-1)
Rm(:,j)   = L(:,1)-Pm((j+1)/2,1);
Rm(:,j+1) = L(:,2)-Pm((j+1)/2,2);
end

% Determine for each mesh point the angle phi_m from the polarcoordinates of Rm

phi_m = zeros(i,km);
phi_m = atan2(Rm(:,2:2:2*km),Rm(:,1:2:(2*km)-1));
phi_mn =phi_m; % phi_mn element [-pi,pi]

for j=1:i,
    for k=1:km,
        if phi_m(j,k)<0,
            phi_m(j,k)=phi_m(j,k)+2*pi; % phi_m element [0,2*pi]
        end
    end
end

dif_phi_m = max(phi_m)-min(phi_m);
dif_phi_mn = max(phi_mn)-min(phi_mn);
concavefactor=1.07;

for j=1:km
        if  dif_phi_m(j) > concavefactor*pi & dif_phi_mn(j) > concavefactor*pi
                Pm(j,3) = 1;
        else
                Pm(j,3) = 0;

        end
end

[Mout] = find(Pm(:,3) == 0);
[Min]  = find(Pm(:,3) == 1);

% Define different branches of the lcfs  for use of interpolation

 [XMIN,N1]=min(con_xt);
 [XMAX,N2]=max(con_xt);
 [YMIN,N3]=min(con_yt);
 [YMAX,N4]=max(con_yt);

 lcfs_xu = fliplr(con_xt(N2:N1));
 lcfs_yu = fliplr(con_yt(N2:N1));
 lcfs_xo  = [con_xt(N1:length(con_xt));con_xt(2:N2)];
 lcfs_yo  = [con_yt(N1:length(con_yt));con_yt(2:N2)];

 if N3>N4,
         Nl = [N3:length(con_yt) 1:N4];
         Nr = [N4:N3];
          lcfs_xl  = con_xt(Nl);
          lcfs_yl  = con_yt(Nl);
          lcfs_xr  = con_xt(Nr);
          lcfs_yr  = con_yt(Nr);
 else     lcfs_xl  = con_xt(N3:N4);
          lcfs_yl  = con_yt(N3:N4);
          lcfs_xr  = fliplr ([con_xt(N4:length(con_xt));con_xt(2:N3)]);
          lcfs_yr  = fliplr ([con_yt(N4:length(con_yt));con_yt(2:N3)]);
 end

% Make sure that each branch is monotonic

 lcfs_xo_dif  = diff( lcfs_xo );
[Mxo] = find(lcfs_xo_dif<0);
if ~isempty(Mxo)
    [ lcfs_xo ,I] = sort( lcfs_xo );
    for m=1:length(lcfs_xo),
            lcfs_yo (m) = lcfs_yo(I(m));
    end
end
 lcfs_xo_dif  = diff( lcfs_xo );
[Nxo] = find(lcfs_xo_dif==0);
if ~isempty(Nxo)
 Nxo_dif = diff(Nxo);
 [nxo] = find(Nxo_dif==1);
 if length(Nxo)>0,
    if ~isempty(nxo)
          for m=1:length(nxo),
          lcfs_xo (Nxo(nxo(m))) = (lcfs_xo(Nxo(nxo(m)))+lcfs_xo(Nxo(nxo(m))-1))/1.99999;
          end
    end
    for m=1:length(Nxo),
          if Nxo(m) < length(lcfs_xo)-1,
          lcfs_xo (Nxo(m)+1) = (lcfs_xo(Nxo(m)+1)+lcfs_xo(Nxo(m)+2))/1.99999;
          else lcfs_xo (Nxo(m)) = (lcfs_xo(Nxo(m))+lcfs_xo(Nxo(m)-1))/1.99999;
          end
    end
 end
end
 lcfs_xu_dif  = diff( lcfs_xu );
[Mxu] = find(lcfs_xu_dif<0);
if ~isempty(Mxu)
    [ lcfs_xu ,I] = sort( lcfs_xu );
    for m=1:length(lcfs_xu),
          lcfs_yu (m) = lcfs_yu(I(m));
    end
end
 lcfs_xu_dif  = diff( lcfs_xu );
[Nxu] = find(lcfs_xu_dif==0);
if ~isempty(Nxu)
 Nxu_dif = diff(Nxu);
 [nxu] = find(Nxu_dif==1);
 if length(Nxu)>0,
    if ~isempty(nxu)
         for m=1:length(nxu),
          lcfs_xu (Nxu(nxu(m))) = (lcfs_xu(Nxu(nxu(m)))+lcfs_xu(Nxu(nxu(m))-1))/1.99999;
         end
    end
    for m=1:length(Nxu),
          if Nxu(m) < length(lcfs_xu)-1,
              lcfs_xu (Nxu(m)+1) = (lcfs_xu(Nxu(m)+1)+lcfs_xu(Nxu(m)+2))/1.99999;
         else lcfs_xu (Nxu(m)) = (lcfs_xu(Nxu(m))+lcfs_xu(Nxu(m)-1))/1.99999;
         end
    end
 end
end
 lcfs_yl_dif  = diff( lcfs_yl );
[Myl] = find( lcfs_yl_dif <0);
if ~isempty(Myl)
    [ lcfs_yl ,I] = sort( lcfs_yl );
    for m=1:length(lcfs_yl),
          lcfs_xl (m) = lcfs_xl(I(m));
    end
end
 lcfs_yl_dif  = diff( lcfs_yl );
[Nyl] = find( lcfs_yl_dif ==0);
if ~isempty(Nyl)
 Nyl_dif = diff(Nyl);
 [nyl] = find(Nyl_dif==1);
 if length(Nyl)>0,
    if ~isempty(nyl)
         for m=1:length(nyl),
          lcfs_yl (Nyl(nyl(m))) = (lcfs_yl (Nyl(nyl(m)))+lcfs_yl(Nyl(nyl(m))-1))/1.99999;
         end
    end
    for m=1:length(Nyl),
          if Nyl(m) < length(lcfs_yl)-1,
          lcfs_yl (Nyl(m)+1) = (lcfs_yl(Nyl(m)+1)+lcfs_yl(Nyl(m)+2))/1.99999;
         else lcfs_yl (Nyl(m)) = (lcfs_yl(Nyl(m))+lcfs_yl(Nyl(m)-1))/1.99999;
         end
    end
 end
```

```
end
 lcfs_yr_dif  = diff( lcfs_yr );
[Myr] = find(lcfs_yr_dif<0);
if ~isempty(Myr)
    [ lcfs_yr ,I] = sort( lcfs_yr );
    for m=1:length(lcfs_yr),
            lcfs_xr (m) = lcfs_xr(I(m));
    end
end
 lcfs_yr_dif  = diff( lcfs_yr );
[Nyr] = find( lcfs_yr_dif ==0);
if ~isempty(Nyr)
 Nyr_dif = diff(Nyr);
 [nyr] = find(Nyr_dif==1);
 if length(Nyr)>0,
    if ~isempty(nyr)
          for m=1:length(nyr),
           lcfs_yr (Nyr(nyr(m))) = (lcfs_yr(Nyr(nyr(m)))+lcfs_yr(Nyr(nyr(m))-1))/1.99999;
          end
    end
    for m=1:length(Nyr),
          if Nyr(m) < length(lcfs_yr)-1,
           lcfs_yr (Nyr(m)+1) = (lcfs_yr(Nyr(m)+1)+lcfs_yr(Nyr(m)+2))/1.99999;
          else  lcfs_yr (Nyr(m)) = (lcfs_yr(Nyr(m))+lcfs_yr(Nyr(m)-1))/1.99999;
          end
    end
 end
end

% Calculation of Power radiated inside/outside  lcfs

pin(l) = 0;
pout(l) = 0;
pdiv(l) = 0;
ptop(l) = 0;
P_in(l) = 0;
P_out(l) = 0;
P_div(l) = 0;
P_top(l) = 0;

 cell_inout  = 2*ones(kc);
emiss = reshape(X(:,l),length(zm),length(rm));
emiss = flipud(emiss);

for j=1:ny
     for k=1:nx
          if Pm((j-1)*mx+k,3)==1 & Pm((j-1)*mx+k+1,3)==1 & Pm(j*mx+k,3)==1 & Pm(j*mx+k
               cell_inout ((j-1)*nx+k) = 1;

            pin(l) = pin(l) + emiss(j,k)*xcell(k);
        elseif Pm((j-1)*mx+k,3)==0 & Pm((j-1)*mx+k+1,3)==0 & Pm(j*mx+k,3)==0 & Pm(j*mx+k
             cell_inout ((j-1)*nx+k) = 0;
            pout(l) = pout(l) + emiss(j,k)*xcell(k);

% Determines the cutlines of each cell with the lcfs

elseif Pm((j-1)*mx+k,3)==0 & Pm((j-1)*mx+k+1,3)==0 & Pm(j*mx+k,3)==1 & Pm(j*mx+k+1,

 x_cut(1) = xgrid(k);
 y_cut(1) = interp1(lcfs_xo, lcfs_yo ,x_cut (1));
 x_cut(2) = xgrid(k+1);
 y_cut(2) = interp1(lcfs_xo, lcfs_yo ,x_cut (2));
 Ain  = ((min(y_cut)-ygrid(j+1))*dx + abs(diff(y_cut))*dx/2) / Acell;
 P_in(l)   = P_in(l) + emiss(j,k)*xcell(k)*Ain;
 P_out(l) = P_out(l) + emiss(j,k)*xcell(k)*(1-Ain);


elseif Pm((j-1)*mx+k,3)==1 & Pm((j-1)*mx+k+1,3)==1 & Pm(j*mx+k,3)==0 & Pm(j*mx+k+1,
 x_cut(1) = xgrid(k+1);
 y_cut(1) = interp1(lcfs_xu, lcfs_yu ,x_cut (1));
 x_cut(2) = xgrid(k);
 y_cut(2) = interp1(lcfs_xu, lcfs_yu ,x_cut (2));
 Aout = ((min(y_cut)-ygrid(j+1))*dx + abs(diff(y_cut))*dx/2) / Acell;
 P_in(l) = P_in(l) + emiss(j,k)*xcell(k)*(1-Aout);
 P_out(l) = P_out(l) + emiss(j,k)*xcell(k)*Aout;

elseif Pm((j-1)*mx+k,3)==0 & Pm((j-1)*mx+k+1,3)==1 & Pm(j*mx+k,3)==0 & Pm(j*mx+k+1,
 y_cut(1) = ygrid(j+1);
 x_cut(1) = interp1(lcfs_yl, lcfs_xl ,y_cut (1));
 y_cut(2) = ygrid(j);
 x_cut(2) = interp1(lcfs_yl, lcfs_xl ,y_cut (2));
 Ain  = ((min(x_cut)-xgrid(k))*dy + abs(diff(x_cut))*dy/2) / Acell;
 P_in(l)   = P_in(l) + emiss(j,k)*xcell(k)*Ain;
 P_out(l) = P_out(l) + emiss(j,k)*xcell(k)*(1-Ain);

elseif Pm((j-1)*mx+k,3)==1 & Pm((j-1)*mx+k+1,3)==0 & Pm(j*mx+k,3)==1 & Pm(j*mx+k+1,
 y_cut(1) = ygrid(j);
 x_cut(1) = interp1(lcfs_yr, lcfs_xr ,y_cut (1));
 y_cut(2) = ygrid(j+1);
 x_cut(2) = interp1(lcfs_yr, lcfs_xr ,y_cut (2));
 Aout = ((min(x_cut)-xgrid(k))*dy + abs(diff(x_cut))*dy/2) / Acell;
 P_in(l)   = P_in(l) + emiss(j,k)*xcell(k)*(1-Aout);
 P_out(l) = P_out(l) + emiss(j,k)*xcell(k)*Aout;

elseif Pm((j-1)*mx+k,3)==0 & Pm((j-1)*mx+k+1,3)==0 & Pm(j*mx+k,3)==0 & Pm(j*mx+k+1,
 y_cut(1) = ygrid(j+1);
 x_cut(1) = interp1(lcfs_yl, lcfs_xl ,y_cut (1));
 x_cut(2) = xgrid(k+1);
 y_cut(2) = interp1(lcfs_xo, lcfs_yo ,x_cut (2));
 Ain  = abs(diff(x_cut))*abs(diff(y_cut))/2 / Acell;
 P_in(l)   = P_in(l) + emiss(j,k)*xcell(k)*Ain;
 P_out(l) = P_out(l) + emiss(j,k)*xcell(k)*(1-Ain);

elseif Pm((j-1)*mx+k,3)==0 & Pm((j-1)*mx+k+1,3)==0 & Pm(j*mx+k,3)==1 & Pm(j*mx+k+1,
 y_cut(2) = ygrid(j+1);
 x_cut(2) = interp1(lcfs_yr, lcfs_xr ,y_cut (2));
 x_cut(1) = xgrid(k);
 y_cut(1) = interp1(lcfs_xo, lcfs_yo ,x_cut (1));
 Ain  = abs(diff(x_cut))*abs(diff(y_cut))/2 / Acell;
 P_in(l)   = P_in(l) + emiss(j,k)*xcell(k)*Ain;
 P_out(l) = P_out(l) + emiss(j,k)*xcell(k)*(1-Ain);

elseif Pm((j-1)*mx+k,3)==1 & Pm((j-1)*mx+k+1,3)==0 & Pm(j*mx+k,3)==0 & Pm(j*mx+k+1,
 y_cut(1) = ygrid(j);
 x_cut(1) = interp1(lcfs_yr, lcfs_xr ,y_cut (1));
 x_cut(2) = xgrid(k);
 y_cut(2) = interp1(lcfs_xu, lcfs_yu ,x_cut (2));
 Ain  = abs(diff(x_cut))*abs(diff(y_cut))/2 / Acell;
 P_in(l)   = P_in(l) + emiss(j,k)*xcell(k)*Ain;
 P_out(l) = P_out(l) + emiss(j,k)*xcell(k)*(1-Ain);

elseif Pm((j-1)*mx+k,3)==0 & Pm((j-1)*mx+k+1,3)==1 & Pm(j*mx+k,3)==0 & Pm(j*mx+k+1,
 y_cut(2) = ygrid(j);
 x_cut(2) = interp1(lcfs_yl, lcfs_xl ,y_cut (2));
 x_cut(1) = xgrid(k+1);
 y_cut(1) = interp1(lcfs_xu, lcfs_yu ,x_cut (1));
 Ain  = abs(diff(x_cut))*abs(diff(y_cut))/2 / Acell;
 P_in(l)   = P_in(l) + emiss(j,k)*xcell(k)*Ain;
 P_out(l) = P_out (l)+ emiss(j,k)*xcell(k)*(1-Ain);

elseif Pm((j-1)*mx+k,3)==0 & Pm((j-1)*mx+k+1,3)==1 & Pm(j*mx+k,3)==1 & Pm(j*mx+k+1,
 x_cut(1) = xgrid(k);
 y_cut(1) = interp1(lcfs_xo, lcfs_yo ,x_cut (1));
 y_cut(2) = ygrid(j);
 x_cut(2) = interp1(lcfs_yl, lcfs_xl ,y_cut (2));
 Aout = abs(diff(x_cut))*abs(diff(y_cut))/2 / Acell;
 P_in(l)   = P_in(l) + emiss(j,k)*xcell(k)*(1-Aout);
 P_out(l) = P_out(l)+ emiss(j,k)*xcell(k)*Aout;

elseif Pm((j-1)*mx+k,3)==1 & Pm((j-1)*mx+k+1,3)==0 & Pm(j*mx+k,3)==1 & Pm(j*mx+k+1,
 x_cut(2) = xgrid(k+1);
 y_cut(2) = interp1(lcfs_xo, lcfs_yo ,x_cut (2));
 y_cut(1) = ygrid(j);
 x_cut(1) = interp1(lcfs_yr, lcfs_xr ,y_cut (1));
 Aout = abs(diff(x_cut))*abs(diff(y_cut))/2 / Acell;
 P_in(l) = P_in(l) + emiss(j,k)*xcell(k)*(1-Aout);
 P_out(l) = P_out(l) + emiss(j,k)*xcell(k)*Aout;
```

```
elseif Pm((j−1)∗mx+k,3)==1 & Pm((j−1)∗mx+k+1,3)==1 & Pm(j∗mx+k,3)==1 & Pm(j∗mx+k+1,3
 x_cut(1) = xgrid(k+1);
 y_cut(1) = interp1(lcfs_xu, lcfs_yu ,x_cut (1));
 y_cut(2) = ygrid(j+1);
 x_cut(2) = interp1(lcfs_yr, lcfs_xr ,y_cut (2));
 Aout = abs(diff(x_cut))∗abs(diff(y_cut))/2 / Acell;
 P_in(l)   = P_in(l) + emiss(j,k)∗xcell(k)∗(1−Aout);
 P_out(l) = P_out(l) + emiss(j,k)∗xcell(k)∗Aout;

elseif Pm((j−1)∗mx+k,3)==1 & Pm((j−1)∗mx+k+1,3)==1 & Pm(j∗mx+k,3)==0 & Pm(j∗mx+k+1,3
 x_cut(2) = xgrid(k);
 y_cut(2) = interp1(lcfs_xu, lcfs_yu ,x_cut (2));
 y_cut(1) = ygrid(j+1);
 x_cut(1) = interp1(lcfs_yl, lcfs_xl ,y_cut (1));
 Aout = abs(diff(x_cut))∗abs(diff(y_cut))/2 / Acell;
 P_in(l)   = P_in(l) + emiss(j,k)∗xcell(k)∗(1−Aout);
 P_out(l) = P_out(l) + emiss(j,k)∗xcell(k)∗Aout;

        end
    end
end

% Calculation of power radiated in the divertor (below the X−point)

for j=1:ny
    for k=1:nx

        if  Pm((j−1)∗mx+k,2)<y_Xpt & Pm((j−1)∗mx+k+1,2)<y_Xpt & Pm(j∗mx+k,2)<y_Xpt & Pr
            pdiv(l) = pdiv(l) + emiss(j,k)∗xcell(k);

    elseif  Pm((j−1)∗mx+k,2)>y_Xpt & Pm((j−1)∗mx+k+1,2)>y_Xpt & Pm(j∗mx+k,2)<y_Xpt & Pr
            x_cut(1) = xgrid(k);
            x_cut(2) = xgrid(k+1);
            Adiv  = diff(x_cut)∗(y_Xpt − ygrid(j+1))/Acell;
            P_div(l) = P_div(l) + emiss(j,k)∗xcell(k)∗Adiv;
        end
    end
end

% Calculation of power radiated above the LCFS
for j=1:ny
    for k=1:nx

        if  Pm((j−1)∗mx+k,2)>y_Tpt & Pm((j−1)∗mx+k+1,2)>y_Tpt & Pm(j∗mx+k,2)>y_Tpt & Pn
            ptop(l) = ptop(l) + emiss(j,k)∗xcell(k);

    elseif  Pm((j−1)∗mx+k,2)>y_Tpt & Pm((j−1)∗mx+k+1,2)>y_Tpt & Pm(j∗mx+k,2)<y_Tpt & Pn
            x_cut(1) = xgrid(k);
```

```
                x_cut(2) = xgrid(k+1);
                Atop = diff(x_cut)*(ygrid(j)-y_Tpt)/Acell;
                P_top(l) = P_top(l) + emiss(j,k)*xcell(k)*Atop;
            end
        end
end

[IN]  = find(cell_inout == 1);
[OUT] = find(cell_inout == 0);

% Power radiated inside / outside LCFS from cells

P_in(l)  = calf(l)*P_in(l)*2*pi*dx*dy; % partially inside  lcfs
P_out(l) = calf(l)*P_out(l)*2*pi*dx*dy; % partially outside lcfs
P_div(l) = calf(l)*P_div(l)*2*pi*dx*dy; % partially below  lcfs
P_top(l) = calf(l)*P_top(l)*2*pi*dx*dy; % partially above  lcfs
pin(l)   = calf(l)*pin(l)*2*pi*dx*dy;  % totally inside  lcfs
pout(l)  = calf(l)*pout(l)*2*pi*dx*dy; % totally outside lcfs
pdiv(l)  = calf(l)*pdiv(l)*2*pi*dx*dy; % totally below  lcfs
ptop(l)  = calf(l)*ptop(l)*2*pi*dx*dy; % totally above  lcfs
Pin(l)   = (pin(l) + P_in(l))*1e-6;       % inside  lcfs rad. power
Pout(l)  = (pout(l)+ P_out(l))*1e-6;   % outside lcfs rad. power
Pdiv(l)  = (pdiv(l)+ P_div(l))*1e-6;   % below  lcfs rad. power
Ptop(l)  = (ptop(l)+ P_top(l))*1e-6;   % above  lcfs rad. power
Ptot(l)  = (Pin(l) + Pout(l));         % totally radiated power
end

Pabove = Ptop;
Pbelow = Pdiv;

% Plots radiated power fractions (Ptot, Pin, Pout and Pdiv)
% Calculated from bolometric inversion routine

if iplot
  ha_figrad=figure('Name','btomorun radiation fractions window', ...
        'NumberTitle','off');
  clf
  hmenu=uimenu('Parent',ha_figrad, ...
          'Label','E&XIT', ...
          'Callback','close(gcf)');

  plot(tvec,Ptot/1e+3,'b')
  hold on
  plot(tvec,Pin/1e+3,'r')
  plot(tvec,Pout/1e+3,'g')
  plot(tvec,Pdiv/1e+3,'-.m')
  plot(tvec,Ptop/1e+3,'--c')
  title (['# ',int2str(shot),' bolometric inversion']);
  xlabel('time [s]');
  ylabel('Prad [kW]');
  legend('Ptot','Pin','Pout','Pbelow','Pabove',0);
end
```

### B.2.2 fablcfs.m

```
function [con_xt, con_yt, c_xt, c_yt, c_times] = ...
      fablcfs(shot, times)

% function [con_xt, con_yt, c_xt, c_yt, c_times] = ...
%     fablcfs(shot, times)
%
% This routine returns matrices of the contours of the
% LCFS from LIUQE for #shot as well as the magnetic axis.
% 'c_times' contains the LIUQE times which were nearest
% to the values specified in the vector "times".
%
% size of con_xt is [npts_contour x length(c_times)]
% size of c_yt is [1 x length(c_times)].
%
% This routine has been taken form the 'btomo.m' package
%
% History:
%
% Matthias Anton, CRPP EPFL, 1995 (first release)
% Iwan Jerjen, CRPP EPFL, 23 of August, 1999 (minor modifications)
%
% Final release version 1.4 (2002/02/07)

sshot=int2str(shot);

% GET THE DATA FROM MDS

% check availability of data

shot=mdsopen('tcv_shot',shot);
pts=mdsdata('\results::npts_contour')-1;

% check whether data is available for this shot

if (pts(1) == -1)

  disp(['NO LIUQE DATA AVAILABLE FOR SHOT: ',sshot]);
  return;

end

% get valid timebase

```

```
tbase=mdsdata('dim_of(\results::r_axis)');

% ensure that 'time' is within range

if min(times)<min(tbase)

  c_time1=1.001*min(tbase); s_c_times1=num2str(c_time1);
  disp(['WARNING: min time out of range, using T =' s_c_times1 '[s] instead']);

end

if max(times)>max(tbase)

  c_time2=0.999*max(tbase); s_c_times2=num2str(c_time2);
  disp(['WARNING: max time out of range, using T =' s_c_times2 '[s] instead']);

end

ii=1;

for k=1:length(times)

  indext=max(find(abs(times(k)-tbase)==min(abs(times(k)-tbase))));

  if ii==1

    c_times(ii)=tbase(indext);
         ii=ii+1;

  else

    if tbase(indext)>c_times(ii-1)

      c_times(ii)=tbase(indext);
           ii=ii+1;

    end

  end

end

con_xt=mdsdata('\results::r_contour[*,$]',c_times);
con_yt=mdsdata('\results::z_contour[*,$]',c_times);

con_xt=con_xt*100;
con_yt=con_yt*100;

c_xt=(mdsdata('\results::r_axis[$]',c_times)*100);
```

```
c_yt=mdsdata('\results::z_axis[$]',c_times)*100;

mdsclose;

if length(c_times)>1

  con_xt(find(con_xt==0))=NaN*ones(size(find(con_xt==0)));
  con_yt(find(con_xt==0))=NaN*ones(size(find(con_xt==0)));

else

  con_xt=con_xt(find(con_xt~=0));
  con_yt=con_yt(find(con_xt~=0));

end

return
```

### B.2.3 fabsetetendue.m

```
function fabsetetendue

% function fabsetetendue
%
% Calculates detector etendue for bolometer tomography.
%
% History:
%
%  Christian Schlatter, TPIV CRPP EPFL, February 2002 (first release)
%
% Final release version 1.4 (2002/02/07)

disp(' ');
disp('Etendue calculator for bolometry.');
disp('Final release 1.4');
disp('Latest modification on 2002/02/07.');
disp(' ');

cw=1;                         % detector numbers cw = 1 : clockwise
                              % cw = 0 : counterclockwise

fans =[1 1 1 1 1 1 1 1];      % camera switch
vangle=[90 0 0 0 0 0 0 -90];  % angle of detector surface normal

xpos=[88 123.5 123.5 123.5 123.5 123.5 123.5 88];

                              % x position of the diaphragmas in [cm]

ypos=[81.5 45.5 45.5 -.25 -.25 -46 -46 -81.5];

```

```
                                          % y position of the diaphragmas in [cm]

% Positions of the detectors

rdet  =[0.862,0.867,0.8721,0.8771,0.8829,0.8879,0.893,0.898,...
 1.278,1.2806,1.2833,1.2861,1.2883,1.2893,1.2902,1.2912];

rdet(17:24)=fliplr(rdet (9:16));
rdet(25:40)=rdet(9:24);
rdet(41:56)=rdet(9:24);
rdet(57:64)=fliplr(rdet (1:8));

zdet =[0.9057,0.9063,0.9069,0.9075];
zdet(5:8)=fliplr(zdet (1:4));
zdet  (9:24)=[0.4916,0.4873,0.483,0.4787,0.473,0.468,0.4631,0.4581,...
 0.4519,0.4469,0.442,0.437,0.4313,0.427,0.4227,0.4184];
zdet(25:40)=zdet(9:24)-0.4575;
zdet(41:56)=zdet(9:24)-0.915;
zdet(57:64)=-zdet(1:8);

xdet=rdet*100; ydet=zdet*100;

d1=1.5/10;                              % detector width in cm
d2=4/10;                                % detector length in cm
b1  =[2.6 2.2 2.2 2.2 2.2 2.2 2.2 2.6]/5;

                                    % aperture width in cm (poloidal)
b2=[10 8 8 8 8 8 8 10]/5;              % aperture length in cm (toroidal)

% Calculation of aperture and detector positions

nact=sum(fans);
iact=find(fans);
ndet=8;
ncam=8;

xap=ones(ndet,1)*xpos;
xap=xap(:)';
yap=ones(ndet,1)*ypos;
yap=yap(:)';
be1=ones(ndet,1)*b1;
be1=be1(:)';
be2=ones(ndet,1)*b2;
be2=be2(:)';

x0=xdet-xap;
y0=ydet-yap;

ivert =[1:8,57:64]';
```

```
x0(ivert)=ydet(ivert)-yap(ivert);
y0(ivert)=xdet(ivert)-xap(ivert);

alpha=atan(y0./x0);

rsquare=x0.^2+y0.^2;

etend=d1*d2*be1.*be2.*cos(alpha).^2./rsquare;

                                % The bolometers etendue

save fabetendue.mat etend; % Save to disk

disp('File fabetendue.mat successfully created.');
disp(' ');
```

### B.2.4 fabsetchord.m

```
function catsetchord

% function catsetchord
%
% Detector viewing lines coordinates.
%
% Release 1.4 (07/02/2002)

disp(' ');
disp('FABCAT Detector viewing line calculator for bolometry.');
disp('Final Release 1.4.');
disp('Latest modification on 2002/02/07.');
disp(' ');

xchord(1,01:09) = [   0.8620   0.8670   0.8721   0.8771   0.8829   0.8879   0.8930   0.8980
xchord(2,01:09) = [   1.1370   1.0843   1.0111   0.9291   0.8309   0.7455   0.6572   0.6200
xchord(1,10:18) = [   1.2806   1.2833   1.2861   1.2883   1.2893   1.2902   1.2912   1.2912
xchord(2,10:18) = [   0.6200   0.6200   0.6200   0.6200   0.6200   0.6200   0.6200   0.6200
xchord(1,19:27) = [   1.2893   1.2883   1.2861   1.2833   1.2806   1.2780   1.2780   1.2806
xchord(2,19:27) = [   0.6200   0.6200   0.6200   0.7261   0.8185   0.8884   0.6200   0.6200
xchord(1,28:36) = [   1.2861   1.2883   1.2893   1.2902   1.2912   1.2912   1.2902   1.2893
xchord(2,28:36) = [   0.6200   0.6200   0.6200   0.6200   0.6200   0.6200   0.6200   0.6200
xchord(1,37:45) = [   1.2861   1.2833   1.2806   1.2780   1.2780   1.2806   1.2833   1.2861
xchord(2,37:45) = [   0.6200   0.6200   0.6200   0.6200   0.8943   0.8256   0.7347   0.6200
xchord(1,46:54) = [   1.2893   1.2902   1.2912   1.2912   1.2902   1.2893   1.2883   1.2861
xchord(2,46:54) = [   0.6200   0.6200   0.6200   0.6200   0.6200   0.6200   0.6200   0.6200
xchord(1,55:63) = [   1.2806   1.2780   0.8980   0.8930   0.8879   0.8829   0.8771   0.8721
xchord(2,55:63) = [   0.6200   0.6200   0.6200   0.6572   0.7455   0.8309   0.9291   1.0111
xchord(1,64:64) = [   0.8620];
xchord(2,64:64) = [   1.1370];

```

```
ychord(1,01:09) = [  0.9057   0.9063   0.9069   0.9075   0.9075   0.9069   0.9063   0.9057
ychord(2,01:09) = [ -0.4800  -0.6200  -0.7100  -0.7500  -0.7500  -0.7500  -0.7500  -0.4951
ychord(1,10:18) = [  0.4873   0.4830   0.4787   0.4730   0.4680   0.4631   0.4581   0.4519
ychord(2,10:18) = [  0.0194   0.0985   0.1698   0.2473   0.3078   0.3648   0.4211   0.4889
ychord(1,19:27) = [  0.4420   0.4370   0.4313   0.4270   0.4227   0.4184   0.0341   0.0298
ychord(2,19:27) = [  0.6022   0.6627   0.7402   0.7500   0.7500   0.7500  -0.5260  -0.4381
ychord(1,28:36) = [  0.0212   0.0155   0.0105   0.0056   0.0006  -0.0056  -0.0106  -0.0155
ychord(2,28:36) = [ -0.2877  -0.2102  -0.1497  -0.0927  -0.0364   0.0314   0.0877   0.1447
ychord(1,37:45) = [ -0.0262  -0.0305  -0.0348  -0.0391  -0.4234  -0.4277  -0.4320  -0.4363
ychord(2,37:45) = [  0.2827   0.3540   0.4331   0.5210  -0.7500  -0.7500  -0.7500  -0.7452
ychord(1,46:54) = [ -0.4470  -0.4519  -0.4569  -0.4631  -0.4681  -0.4730  -0.4780  -0.4837
ychord(2,46:54) = [ -0.6072  -0.5502  -0.4939  -0.4261  -0.3698  -0.3128  -0.2523  -0.1748
ychord(1,55:63) = [ -0.4923  -0.4966  -0.9057  -0.9063  -0.9069  -0.9075  -0.9075  -0.9069
ychord(2,55:63) = [ -0.0244   0.0635   0.4951   0.7500   0.7500   0.7500   0.7500   0.7100
ychord(1,64:64) = [ -0.9057];
ychord(2,64:64) = [  0.4800];

save fabchord.mat xchord ychord;

disp('File fabchord.mat has been created successfully.');
disp(' ');
```

### B.2.5 fabsetsolidangle.m

```
function [OMEGA, rho_grid, zet_grid] = fabsetsolidangle

% function [OMEGA, rho_grid, zet_grid] = fabsetsolidangle
%
% 'fabsetsolidangle.m' calculates the individual bolometers solid angles and stores
% them to files called 'fabsolidangle_#.mat' in the current working directory.
%
% The 'fabsolidangle_#.mat' files are used by the routine 'fabsolidangle.m' which helps
% to calculate the T-matrix (initiated by 'fabsett.m')
%
% This program calls 'fab3d.m'
%
% History:
%
%  Matthias Anton, CRPP EPFL, 29 of May, 1995 (first release)
%  Jan Mlynar, CRPP EPFL, 20 of August, 1998 (adaption for bolometry)
%  Christian Schlatter, TPIV CRPP EPFL, February 2002 (comments and renaming)
%
% Final release version 1.4 (2002/02/07)

global Kb1 Kb2 Kb3 Kb4 Kd1 Kd2 Kd3 Kd4

debut=cputime;

disp(' ')
disp('Solid angle calculator for bolometry.');
disp('Final release 1.4')
disp('Latest modification on 2002/02/07.')
disp(' ');

% Detector parameters

cw=1;                      % Detector numbers cw = 1 : clockwise cw = 0 : ccw
fans =[1 1 1 1 1 1 1 1];   % Camera switch
vangle=[90 0 0 0 0 0 0 -90]; % Angle of detector surface normal
xpos=[88 123.5 123.5 123.5 123.5 123.5 123.5 88];
                           % x position of the diaphragmas in [cm]
ypos=[81.5 45.5 45.5 -.25 -.25 -46 -46 -81.5];
                           % y position of the diaphragmas in [cm]

% Position of the detectors

rdet =[0.862,0.867,0.8721,0.8771,0.8829,0.8879,0.893,0.898,...
 1.278,1.2806,1.2833,1.2861,1.2883,1.2893,1.2902,1.2912];

rdet(17:24)=fliplr(rdet(9:16));
rdet(25:40)=rdet(9:24);
rdet(41:56)=rdet(9:24);
rdet(57:64)=fliplr(rdet(1:8));

zdet=[0.9057,0.9063,0.9069,0.9075];
zdet(5:8)=fliplr(zdet(1:4));
zdet (9:24)=[0.4916,0.4873,0.483,0.4787,0.473,0.468,0.4631,0.4581,...
 0.4519,0.4469,0.442,0.437,0.4313,0.427,0.4227,0.4184];
zdet(25:40)=zdet(9:24)-0.4575;
zdet(41:56)=zdet(9:24)-0.915;
zdet(57:64)=-zdet(1:8);

xdet=rdet*100; ydet=zdet*100;

d1=1.5/10;                              % Detector width     in cm
d2=4/10;                                % Detector length    in cm
b1 =[2.6 2.2 2.2 2.2 2.2 2.2 2.2 2.6]/5;
                                           % Aperture width    in cm (pol.)
b2=[10 8 8 8 8 8 8 10]/5;
                                           % Aperture length   in cm (tor.)

% Calculation of aperture and detector positions

nact=sum(fans);
iact=find(fans);
ndet=8;
ncam=8;
```

```
% Aperture 1 (poloidally limiting)

xap=ones(ndet,1)*xpos;
xap=xap(:)';
yap=ones(ndet,1)*ypos;
yap=yap(:)';

% Detectors

dxd1=xdet-xap;
dyd1=ydet-yap;

ivert =[1:8,57:64]';
ihori=[9:56]';

dxd2=dxd1; dyd2=dyd1; % Supp. da = da2 i.e. diaphragma-array distance is
                                   % equal in poloidal (da) and toroidal (da2) directions
xap2=xdet-dxd2;
yap2=ydet-dyd2;

xyw=ones(ndet,1)*b1/2;
xyw=xyw(:)';

idxw(ivert)=ones(size(ivert));
idyw(ihori)=ones(size(ihori));
idyw(64)=0;

dxw=xyw;
dyw=xyw;
dzw=ones(ndet,1)*b2(iact)/2;
dzw=dzw(:)';

% x, y: poloidal cross section ; z : ' toroidal ' coordinate
% Transform to the 'format' used by 'fab3d.m':
 % 1, 2 : midpoints on the straight lines which limit the detector poloidally
 % 3, 4 ; midpoints on the straight lines which limit the detector toroidally

Kb1=[xap-dxw;yap-dyw;zeros(size(xap))];
Kb2=[xap+dxw;yap+dyw;zeros(size(xap))];
Kb3=[xap2;yap2;-dzw];
Kb4=[xap2;yap2;dzw];

Kd1=[xdet-idxw*d1/2;ydet-idyw*d1/2;zeros(size(xap))];
Kd2=[xdet+idxw*d1/2;ydet+idyw*d1/2;zeros(size(xap))];
Kd3=[xdet;ydet;-d2*ones(size(xdet))/2];
Kd4=[xdet;ydet;d2*ones(size(xdet))/2];

% Get the detector names

```

```
detn=[];

for j=1:length(fans)

          if fans(j)

          detn=[detn,(j−1)*8+1:j*8];

     end

end

% Calculate the solid angle for the whole grid and every detector

for kk=1:length(xap)

     disp('');
     disp([' Calculation of the solid angle for detector ', int2str(kk)]);

          [omega, rho_grid, zet_grid , dV] = ...
          fab3d(Kb1(:,kk), Kb2(:,kk), Kb3(:,kk), Kb4(:,kk ), ...
                    Kd1(:,kk), Kd2(:,kk), Kd3(:,kk), Kd4(:,kk), idxw(kk));

          ici =find(omega);
     rhodum=ones(size(zet_grid'))*rho_grid;
          zetdum=zet_grid'*ones(size(rho_grid));
     rhoici =rhodum(ici);
          zetici =zetdum(ici);
     omeici=omega(ici);
          i_detec=1;

     eval(['save fabsolidangle_',int2str(detn(kk)),......
          '.mat ici rho_grid zet_grid rhoici zetici omeici dV i_detec']);

end

OMEGA=[];

disp(' ');
disp('Successfully finished');
disp(' ');

time_end = (cputime − debut)/60;
disp(['CPU runtime: ' num2str(time_end),' min.']);
disp(' ');
```

### B.2.6 fab3d.m

```
function [omega, rho_grid, zet_grid , dV] = ...
```

```
    fab3d(Kb1, Kb2, Kb3, Kb4, Kd1, Kd2, Kd3, Kd4, ivert)

% function [omega, rho_grid, zet_grid , dV] = ...
%    fab3d(Kb1, Kb2, Kb3, Kb4, Kd1, Kd2, Kd3, Kd4, ivert)
%
% Calculates a 2D-matrix omega from a 3D grid definded inside
% 'fabsetangle.m'.
% The grid fills approximately a 40 cm thick poloidal slice
% of the TCV vacuum vessel ('thick': in toroidal direction)
%
% input data:  Kb1..4:        midpoints of the edges of the aperture
%              Kd1..4:        midpoints of the edges of the detector
%                             each K.. has three components:
%                               K..(1): radial rho
%                                  K..(2): vertical    zet
%                                  K..(3): toroidal    tee
%                               K..1&2: midpoints of edge 'lines'
%                                       in rho-zet-plane
%                               K..3&4: midpoints of edges
%                                       in tee-zet-plane
%              ivert :     a flag , determines if the detector 'looks'
%                             horizontally or vertically
% output data : omega:    The detectors solid angles
%              rho_grid : The radial coordinates.
%              zet_grid : The vertical coordinates.
%              dV:        a pixels volume.
%
% This program calls 'fabproject.m'
%
% History:
%
% Matthias Anton, CRPP EPFL, 29 of May, 1995 (first release)
% Christian Schlatter , TPIV CRPP EPFL, February 2002 (comments and renaming)
%
% Final release version 1.4 (2002/02/07)


% parameter: define the 3D grid
% rho: radial
% zet: vertical
% tee: toroidal coordinate


Rho=88;
brho=2*30;
bzet=2*90;
btee=2*20;

drho=0.5;
```

```
dzet=0.5;
dtee=0.5;

dV=dtee*dzet*drho;

rho_grid=Rho-(brho-drho)/2:drho:Rho+(brho-drho)/2;
zet_grid=-(bzet-dzet)/2:dzet:(bzet-dzet)/2;
tee_grid=0:dtee:(btee-dtee)/2;

% get four points of aperture (midpoint of each side)
% 1,2: in the rho-zet plane
% 3,4: in the tee-zet plane

rhob1=Kb1(1);
zetb1=Kb1(2);
teeb1=Kb1(3);

rhob2=Kb2(1);
zetb2=Kb2(2);
teeb2=Kb2(3);

rhob3=Kb3(1);
zetb3=Kb3(2);
teeb3=Kb3(3);

rhob4=Kb4(1);
zetb4=Kb4(2);
teeb4=Kb4(3);

% get four points of detector (midpoint of each side)
% 1,2: in the rho-zet plane
% 3,4: in the tee-zet plane

rhod1=Kd1(1);
zetd1=Kd1(2);
teed1=Kd1(3);

rhod2=Kd2(1);
zetd2=Kd2(2);
teed2=Kd2(3);

rhod3=Kd3(1);
zetd3=Kd3(2);
teed3=Kd3(3);

rhod4=Kd4(1);
zetd4=Kd4(2);
teed4=Kd4(3);
```

```
% init omega

omega=zeros(length(zet_grid),length(rho_grid));

% loop over toroidal coordinate

for it=1:length(tee_grid);

        tee=tee_grid(it);
        rho_shift=Rho-sqrt(Rho^2-tee^2);

        rho=ones(size(zet_grid'))*(rho_grid-rho_shift);
        zet=zet_grid'*ones(size(rho_grid));
    tee=tee*ones(size(zet));

        if ivert

                [rhol1,zetl1,rhol2,zetl2] = ...
                fabproject(rhob1,zetb1,rhob2,zetb2,rhod1,zetd1,rhod2,zetd2,rho,zet);

                [teel3,zetl3,teel4,zetl4] = ...
                fabproject(teeb3,zetb3,teeb4,zetb4,teed3,zetd3,teed4,zetd4,tee,zet);

                die=find(rhol1~=99999 & teel3 ~=99999);

                dis_rho=(rhol1(die)+rhol2(die))/2-rho(die);
                dis_zet=(zetl1(die)+zetl2(die)+zetl3(die)+zetl4(die))/4-zet(die);
                dis_tee=(teel3(die)+teel4(die))/2-tee(die);

                surf_rho=(zetl1(die)-zetl2(die)).*(teel3(die)-teel4(die));
                surf_zet=-(rhol1(die)-rhol2(die)).*(teel3(die)-teel4(die));
                surf_tee=(rhol1(die)-rhol2(die)).*(zetl3(die)-zetl4(die));

        else

                [zetl1,rhol1,zetl2,rhol2]=....
                fabproject(zetb1,rhob1,zetb2,rhob2,zetd1,rhod1,zetd2,rhod2,zet,rho);

                [teel3,rhol3,teel4,rhol4]=....
                fabproject(teeb3,rhob3,teeb4,rhob4,teed3,rhod3,teed4,rhod4,tee,rho);

                die=find(zetl1~=99999 & teel3 ~=99999);

                dis_rho=(rhol1(die)+rhol2(die)+rhol3(die)+rhol4(die))/4-rho(die);
                dis_zet=(zetl1(die)+zetl2(die))/2-zet(die);
                dis_tee=(teel3(die)+teel4(die))/2-tee(die);

                surf_rho=(zetl1(die)-zetl2(die)).*(teel3(die)-teel4(die));
                surf_zet=-(rhol1(die)-rhol2(die)).*(teel3(die)-teel4(die));
```

```
                    surf_tee=-(zetl1(die)-zetl2(die)).*(rhol3(die)-rhol4(die));

            end

            dd=(dis_rho.^2+dis_zet.^2+dis_tee.^2).^0.5;
            surfn=abs(surf_rho.*dis_rho + ...
                    surf_zet .* dis_zet  + ...
                    surf_tee .* dis_tee )./dd;

            omega(die)=omega(die)...
                   +abs(surf_rho.*dis_rho + ...
                    surf_zet .* dis_zet  + ...
                    surf_tee .* dis_tee )./ dd.^3;

            if  it==1;

            omega0=omega;

        end

end

omega=2*omega-omega0;
```

### B.2.7 fabproject.m

```
function [xl1, yl1, xl2, yl2 ] = ...
    fabproject(xb1, yb1, xb2, yb2, xd1, yd1, xd2, yd2, xi, yi)

% function [xl1, yl1, xl2, yl2 ] = ...
%     fabproject(xb1, yb1, xb2, yb2, xd1, yd1, xd2, yd2, xi, yi)
%
% Calculates the projection of two points xb1,xb2,yb1,yb2 on a line
% defined by xd1,yd1,xd2,yd2. The point of projection is xi,yi.
%
% sizes: xb1,yb1,xb2,yb2,xd1,yd1,xd2,yd2        1x1
%                xi,yi                                arbitrary
%                xl1,yl1,xl2,yl2                          same size as xi,yi
%
% This program is a subroutine of 'fab3d.m'
%
% History:
%
% Matthias Anton, CRPP EPFL, 29 of May, 1995 (first release)
% Christian Schlatter, TPIV CRPP EPFL, February 2002 (comments and renaming)
%
% Final release version 1.4 (2002/02/07)

xl1=zeros(size(xi));
```

```
xl2=zeros(size(xi));
yl1=zeros(size(xi));
yl2=zeros(size(xi));

p1=( yi*(xb1-xd1)-xi*(yb1-yd1)+xd1*yb1-yd1*xb1 ) ./ ....
    ( (yb1-yi)*(xd1-xd2)-(xb1-xi)*(yd1-yd2) );

p2=( yi*(xb2-xd1)-xi*(yb2-yd1)+xd1*yb2-yd1*xb2 ) ./ ....
    ( (yb2-yi)*(xd1-xd2)-(xb2-xi)*(yd1-yd2) );

xb1d=xd1+p1*(xd2-xd1);
yb1d=yd1+p1*(yd2-yd1);
xb2d=xd1+p2*(xd2-xd1);
yb2d=yd1+p2*(yd2-yd1);

iia = find( (p1<0 & p2<0) | (p1>1 & p2>1) );
iib = find( p1<=1 & p1>=0 & p2>=0 & p2<=1 );
iic = find( (p1<0 & p2>1) | (p2<0 & p1>1) );
iid = find( p1>=0 & p1<=1 & p2>1 );
iie = find( p1>=0 & p1<=1 & p2<0 );
iif = find( p2>=0 & p2<=1 & p1>1 );
iig = find( p2>=0 & p2<=1 & p1<0 );

if ~isempty(iia);

  xl1(iia)=xi(iia);
  xl2(iia)=xi(iia);
  yl1(iia)=yi(iia);
  yl2(iia)=yi(iia);

end

if ~isempty(iia);

  xl1(iia)=99999*ones(size(iia));
  xl2(iia)=99999*ones(size(iia));
  yl1(iia)=99999*ones(size(iia));
  yl2(iia)=99999*ones(size(iia));

end

if ~isempty(iib);

  xl1(iib)=xb1d(iib);
  xl2(iib)=xb2d(iib);
  yl1(iib)=yb1d(iib);
  yl2(iib)=yb2d(iib);

end
```

```

if ~isempty(iic);

  xl1( iic )=xd1*ones(size(iic));
  xl2( iic )=xd2*ones(size(iic));
  yl1( iic )=yd1*ones(size(iic));
  yl2( iic )=yd2*ones(size(iic));

end

if ~isempty(iid);

  xl1( iid )=xd2*ones(size(iid));
  xl2( iid )=xb1d(iid);
  yl1( iid )=yd2*ones(size(iid));
  yl2( iid )=yb1d(iid);

end

if ~isempty(iie);

  xl1( iie )=xd1*ones(size(iie));
  xl2( iie )=xb1d(iie);
  yl1( iie )=yd1*ones(size(iie));
  yl2( iie )=yb1d(iie);

end

if ~isempty(iif);

  xl1( iif )=xd2*ones(size(iif));
  xl2( iif )=xb2d(iif);
  yl1( iif )=yd2*ones(size(iif));
  yl2( iif )=yb2d(iif);

end

if ~isempty(iig);

  xl1( iig )=xd1*ones(size(iig));
  xl2( iig )=xb2d(iig);
  yl1( iig )=yd1*ones(size(iig));
  yl2( iig )=yb2d(iig);

end
```

### B.2.8 fabsett.m

```
function fabsett
```

```

% function fabsett
%
% Calculation of the T matrix for a rectangular grid using precalculated
% matrices of solid angles for all detectors and a grid of 0.5x0.5x0.5cm^3
% The solid angle matrices has to be called 'fabangle_#.mat' and need to be
% stored in a subdirectory called \fabsolidangle\.
%
% This uses the 3 dimensional omgrid algorithm (fabangle.m)
% This is the version for bolometry.
%
% The variables nx ny xmin xmax ymin ymax dx dy xmesh ymesh numdet T
% are stored to the file fabt.mat for use with fabcat.m.
%
% History:
%
% Christian Schlatter, TPIV CRPP EPFL, November 2001
%
% Final release version 1.4 (2002/02/07)

disp(' ')
disp('FABCAT T-matrix calculator for bolometry.');
disp('Final release 1.4')
disp('Latest modification on 2002/02/07.')
disp(' ');

% Tokamak setup parameters for placing the rectangular grid.
% Values correspond to the default values in the btomo package.
% All values given in [cm]

r_0 = 88;      % radial position of the rectangular grid center
z_0 = 0;       % z-coordinate of the rectangular grid center
w_r = 55;      % horizontal width of the grid
w_z = 154;     % vertical width of the grid
nx = 10;       % hoizontal number of pixels

% Camera setup (bolometry detectors)

f_cam = [1 1 1 1 1 1 1 1];

xmin=r_0-w_r/2;
xmax=r_0+w_r/2;
ymin=z_0-w_z/2;
ymax=z_0+w_z/2;

dx=(xmax-xmin)/nx;
dy=dx;

ny = ceil((ymax-ymin)/dy);
```

```
ymin = z_0-ny/2*dy;
ymax = z_0+ny/2*dy;

xmesh = xmin+dx/2:dx:xmax-dx/2;
ymesh = ymin+dy/2:dy:ymax-dy/2;

[T,numdet] = fabangle(f_cam, xmin, xmax, ymin, ymax, nx, ny);

save fabt.mat nx ny xmin xmax ymin ymax dx dy xmesh ymesh numdet T;

disp('File fabt.mat successfully created.')
disp(' ');
```

### B.2.9 fabangle.m

```
function [T, numdet] = fabangle(fans, xmin, xmax, ymin, ymax, nx, ny)

% function [T, numdet] = fabangle(fans, xmin, xmax, ymin, ymax, nx, ny)
%
% Calculation of the T matrix for a rectangular grid using precalculated
% matrices of solid angles for all detectors and a grid of 0.5x0.5x0.5cm^3
% matrices are stored in fabangle_#.mat files located in the subdirectory
% \fabsolidangle\.
%
% History:
%
% Matthias Anton, CRPP EPFL, 30 of May, 1995 (first release)
% Matthias Anton, CRPP EPFL, 23 of November, 1995
% Iwan Jerjen, CRPP EPFL, 12 of July, 1998
% Jan Mlynar, CRPP EPFL, 20 of August, 1998 (adaption for bolometry)
%
% Final release version 1.4 (2002/02/07)

debut=cputime;
detn=[];

for j=1:length(fans)

  if fans(j)

    detn=[detn,(j-1)*8+1:j*8];

  end

end

ndet=length(detn);

dx=(xmax-xmin)/nx;
```

```
dy=(ymax-ymin)/ny;
xgrid=xmin+dx/2:dx:xmax-dx/2;
ygrid=ymin+dy/2:dy:ymax-dy/2;

xpix=ones(size(ygrid'))*xgrid;
ypix=ygrid'*ones(size(xgrid));
xpix=xpix(:);
ypix=ypix(:);

T=zeros(ndet,nx*ny);

for k=1:ndet

  eval(['load fabsolidangle/fabsolidangle_'...
                              ,int2str(detn(k))])

  for l = 1 : nx*ny

    drin=find(rhoici>=xpix(l)-dx/2 & rhoici<=xpix(l)+dx/2 ....
                                & zetici>=ypix(l)-dy/2 & zetici<=ypix(l)+dy/2);
        T(k,l)=sum(omeici(drin))*dV;

  end

end

load fabchord

[Th,numh]=fabstandard(100.*xchord,100.*ychord,xmin,xmax,ymin,ymax,nx,ny);

for i=1:64

  linst =find(numh==i);

  if ~isempty(linst)

    leng=sum(Th(linst,:));

  else

    leng=0;

  end

  viv=sum(T(i,:));

  if viv~=0

    cor=leng/viv;

  else

    cor=0;

  end

  T(i,:)=T(i,:)*cor;

end

numdet=find(sum(T'));
T=T(numdet,:);
numdet=detn(numdet);

time_end = cputime - debut;
disp(['CPU runtime: ' num2str(time_end),' s.']);
disp(' ');
```

### B.2.10 fabstandard.m

```
function [TT, numdet] = fabstandard(xchord, ychord, ...
    xmin, xmax, ymin, ymax, nx, ny);

% function [TT, numdet] = fabstandard(xchord, ychord, ...
%    xmin, xmax, ymin, ymax, nx, ny);
%
% A fast algorithm to calculate the lengths of the chords given by
% xchord, ychord in pixels of a grid specified by the other inputs.
%
% Determination only based on geometry
%
% Inputs
%
%       xchord, ychord: endpoints of lines of sight, size [2 x nl]
%       xmin ... ymax: corners of pixel grid
%       nx, ny:               number of pixels horizontal, vertical
%
% Output
%
%       TT:            transfermatrix [length(numdet) x nx*ny],
%                      TT(l,i) is the length of chord l in pixel i
%       numdet: numbers of 'active' lines of sight, usually
%                      length(numdet) < nl
%
% History:
%
% Matthias Anton, CRPP EPFL, 9 of August 1994 (first release)
% Matthias Anton, CRPP EPFL, 2nd of December 1994
```

```
%
% Final release version 1.4 (2002/02/07)

[dummy,nl]=size(xchord);

dx=(xmax−xmin)/nx;
dy=(ymax−ymin)/ny;
xgrid=xmin:dx:xmax;
ygrid=ymin:dy:ymax;
xpix=xmin+dx/2:dx:xmax−dx/2;
ypix=ymin+dy/2:dy:xmax−dy/2;
numpix=reshape(1:nx∗ny,ny,nx);

for k=1:nl

  c=polyfit(xchord(:,k),ychord(:,k),1);
  m(k)=c(1);b(k)=c(2);

end

% crossing with vertical lines of grid

Ysec=zeros(nl,nx+1);
X=ones(nl,1)∗xgrid;
Ysec=m'∗xgrid+b'∗ones(1,nx+1);

% crossings with horizontal lines of grid

Xsec=zeros(nl,ny+1);
Y=ones(nl,1)∗ygrid;
Xsec=(ones(nl,1)∗ygrid−b'∗ones(1,ny+1))./(m'∗ones(1,ny+1));

% matrices with x and y coordinates of all crossings with meshgrid

XX=[X,Xsec];
YY=[Ysec,Y];

% sorting in ascending order of X

for k=1:nl

  [XX(k,:),ind]=sort(XX(k,:));
  YY(k,:)=YY(k,ind);

end

dX=[diff(XX')' zeros(nl,1)];
dY=[diff(YY')' zeros(nl,1)];
```

```
% trying to find out the pixel numbers

XXnum=zeros(size(XX));
YYnum=zeros(size(YY));

for k=1:nx

  hx=find(XX+dX/2>xgrid(k) & XX+dX/2<xgrid(k+1) & dX>0);
  XXnum(hx)=k*ones(size(hx));

end

for k=1:ny

  hy=find( (YY+dY/2>ygrid(k) & YY + dY/2 < ygrid(k+1)) & dX>0 );
  YYnum(hy)=k*ones(size(hy));

end

seglength=sqrt(dX.^2+dY.^2);
TT=zeros(nl,nx*ny);

for k=1:nl

  cols=find((YYnum(k,:)~=0)&(XXnum(k,:)~=0)&...
        (seglength(k,:)<=sqrt(dx^2+dy^2)));
  dummy=diag( numpix(YYnum(k,cols),XXnum(k,cols)) );

  if ~isempty(dummy)

    TT(k,dummy(:))=seglength(k,cols);

  end

end

numdet=find(sum(TT'));
TT=TT(numdet,:);
```

## B.3 Tools

### B.3.1 fabmov.m

```
function fabmov(shot, wperclev)

% fabmov(shot, level)
%
% When called without argument runs last shot, default level = 100.
% Loads results of fast bolometry tomography. Uses BLACK AND WHITE contours for
% displaying emission profiles . Shows plasma position, allows MOVIE run.
% By default the emission level is 20 kW.m^(-3) but this can be changed
% by the second function parameter.
%
% History:
%
% Jan Mlynar, CRPP EPFL, September 2001 (first release)
% Jan Mlynar, CRPP EPFL, January 2002 (adaption for bolometry)
%
% Final release version 1.4 (2002/02/07)

global bm_maincat bm_movieaxis bm_shotnum bm_comment bm_framenum bm_chisq ...
       bm_lambda bm_Ptot bm_Pup bm_Pbot bm_play bm_repeat bm_speed ...
       bm_delchans bm_delchans2

global shotnum tsteps comment g_bolo Ptot lambda rpix zpix tvec chisq ...
       rmag zmag moma maxlev clevs wperc Pup Pbot delchans

if nargin < 1

  shot=21518;

end

if isnumeric(shot)

  action='start';
  shotnum=shot;

  if nargin < 2

    wperc=20000; % W.m^(-3) per contour

  else

    wperc=wperclev;

  end

else

  action=shot;

end

switch action

case 'start'

```

```
    bm_maincat=fabmovgui;
    zoom on;

    bm_movieaxis=findobj(bm_maincat,'Tag','movieaxes');
    bm_shotnum=findobj(bm_maincat,'Tag','shotnum');
    bm_comment=findobj(bm_maincat,'Tag','comment');
    bm_framenum=findobj(bm_maincat,'Tag','framenum');
    bm_chisq=findobj(bm_maincat,'Tag','chisq');
    bm_lambda=findobj(bm_maincat,'Tag','lambda');
    bm_Ptot=findobj(bm_maincat,'Tag','Ptot');
    bm_Pup=findobj(bm_maincat,'Tag','Pup');
    bm_Pbot=findobj(bm_maincat,'Tag','Pbot');
    bm_play=findobj(bm_maincat,'Tag','play');
    bm_repeat=findobj(bm_maincat,'Tag','repeat');
    bm_speed=findobj(bm_maincat,'Tag','speed');
    bm_delchans=findobj(bm_maincat,'Tag','delchans');
    bm_delchans2=findobj(bm_maincat,'Tag','delchans2');

    axes(bm_movieaxis);
    axis equal;
    axis ([.6 1.2 -.8 .8]);

    set(bm_shotnum,'String',int2str(shotnum));

    fabmov('shotnum');
    return

case 'shotnum'

    set(bm_comment,'String','...wait please, getting inversions...');
    cla
    drawnow

    shotnum=eval(get(bm_shotnum,'String'));

    mdsopen(shotnum);

    try

      g_bolo=mdsdata('\results::btomo:emissivity');
      comment=mdsdata('\results::btomo:comment');
      delchans=mdsdata('\results::btomo:nchord');
      Ptot=mdsdata('\results::btomo:prad_tot');
      lambda=mdsdata('\results::btomo:lambda');
      rpix=mdsdata('\results::btomo:rmesh');
      zpix=mdsdata('\results::btomo:zmesh');
      tvec=mdsdata('\results::btomo:time');
      chisq=mdsdata('\results::btomo:chi_squared');
      Pup=mdsdata('\results::btomo:prad_above');
```

```
  Pbot=mdsdata('\results::btomo:prad_below');

catch

  comment=' Data problem, sorry.'

end

mdsclose;

if length(delchans) < 20

  delchans1=int2str(delchans');;
  delchans2=[];

else

  delchans1=int2str(delchans(1:19)');
  delchans2= int2str(delchans(20:length(delchans))');

end

nr=length(rpix);
nz=length(zpix);
tsteps=length(tvec);
g_bolo=reshape(g_bolo,nz,nr,tsteps);

set(bm_comment,'String',comment);
pause(3)
set(bm_delchans,'String',delchans1);
set(bm_delchans2,'String',delchans2);
set(bm_lambda,'String',num2str(lambda,4));

if tsteps==0

  set(bm_comment,'String','No data, sorry.');
  return;

end

if isnan(Pup) Pup=NaN*ones(size(tvec)); end
if isnan(Pbot) Pbot=NaN*ones(size(tvec)); end

set(bm_comment,'String',' ...wait please, making frames...');

maxlev=max(g_bolo(:));
clevs=[0:wperc:maxlev];

bfr=5;                   % number of empty frames
```

```

for ti = 1 : tsteps

  cla
      [con,hcon]=contour(rpix,zpix,g_bolo(:,:, ti ), clevs , 'k');

      hold on;

      text(.65,-.68,'FRAME');text(.95,-.68,'TIME');
      text(.78,-.75,int2str(ti )); text(1,-.75,num2str(tvec(ti),4));
      text (.65,.74,[ int2str(wperc/1000) ' kW.m^{-3}/cont.']);

      hold off;
  caxis([0 maxlev]);
      axis equal;
  shading flat;
  axis ([.6 1.15 -.8 .8]);
      title (['shot #' int2str(shotnum)]);
      xlabel('r [cm]');
      ylabel('z [cm]');

      set(bm_framenum,'String',num2str(ti));

      set(bm_Ptot,'String',num2str(Ptot(ti)/1000,4));
      set(bm_Pup,'String',num2str(Pup(ti)/1000,4));
      set(bm_Pbot,'String',num2str(Pbot(ti)/1000,4));
      set(bm_chisq,'String',num2str(chisq(ti),4));

      if ti == 1 % moviein is not necessary in MatLab6

    moma=moviein(tsteps+bfr,bm_movieaxis);

  end

      moma(:,ti)=getframe(bm_movieaxis);

end

cla;

for blanc = 1 : bfr               % to distinguish the end in movie reply

  moma(:,tsteps+blanc)=getframe(bm_movieaxis);

end

set(bm_framenum,'String','1');
fabmov('framenum')
return;
```

```

case 'play'

 set(bm_comment,'String','Playing movie...');

 repeat=eval(get(bm_repeat,'String'));
 speed=eval(get(bm_speed,'String'));
 movie(moma,repeat,speed);

 fabmov('framenum')
 return

case 'chan>'

 framenum=eval(get(bm_framenum,'String'))+1;
 set(bm_framenum,'String',int2str(framenum))
 fabmov('framenum');
 return

case 'chan<'

 framenum=eval(get(bm_framenum,'String'))-1;
 set(bm_framenum,'String',int2str(framenum))
 fabmov('framenum');
 return

case 'framenum'

 set(bm_comment,'String','........');
 ti=eval(get(bm_framenum,'String'));

 if ti<1 ti=1;

   set(bm_comment,'String','..hey, don''t exaggerate!');pause(1);

 end

 if ti>tsteps ti=tsteps;

   set(bm_comment,'String','..hey, don''t exaggerate!');pause(1);

 end

 drawnow
 cla
 contour(rpix,zpix,g_bolo(:,:,ti),clevs,'k');
 hold on;

 text(.65,-.68,'FRAME');text(.95,-.68,'TIME');
```

```
    text(.78,-.75,int2str(ti)); text(1,-.75,num2str(tvec(ti),4));
    text (.65,.74,[ int2str(wperc/1000) ' kW.m^{-3}/cont.']);

    hold off;
    caxis([0 maxlev]);
    axis equal;
    shading flat;
    axis ([.6 1.15 -.8 .8]);
    title (['shot #' int2str(shotnum)]);
    xlabel('r [cm]');
    ylabel('z [cm]');

    set(bm_framenum,'String',num2str(ti));

    set(bm_Ptot,'String',num2str(Ptot(ti)/1000,4));
    set(bm_lambda,'String',num2str(lambda,4));
    set(bm_Pup,'String',num2str(Pup(ti)/1000,4));
    set(bm_Pbot,'String',num2str(Pbot(ti)/1000,4));
    set(bm_chisq,'String',num2str(chisq(ti),4));
    set(bm_comment,'String','Ready.');

    return

end
```

### B.3.2 fabreader.m

```
function fabreader(shot);

disp ('FABCAT MDS Reader. Release 1.4');

if nargin < 1

  startshot = 19000;
  endshot = 21739;

else

  startshot = shot;
  endshot = shot;

end

index = 1;

for shot = startshot : endshot

  try

```

```
    shotnumber(index) = shot;
    exist(index) = 0;
    xpoint(index) = NaN;
    status(index) = NaN;
    lambda(index) = NaN;
    chi2(index) = NaN;

    mdsopen('results',shot);

    disp(['Analyzing shot ', num2str(shot)]);

    method = mdsdata('\results::btomo:method');
    disp (method);

    if method(1:17) == (('FBTR with x-point') | ('FABCAT with X-point'))

      xpoint(index) = 1;

    else

      xpoint(index) = 0;

    end

    lambda(index) = mdsdata('\results::btomo:lambda');

    chim = mdsdata('\results::btomo:chi_squared');

    chi2(index) = mean(chim);

    % Check using the default settings of fabcat !!!
    if lambda(index) < 0.01 | abs(chi2(index) - 1) > 0.05

      status(index) = 0;

    else

      status(index) = 1;

    end

    exist(index) = 1;

    result = mdsdata('\results::btomo:comment');
    disp(result);

  catch

  end

  index = index + 1;
  mdsclose

end

save fabresults shotnumber exist xpoint status lambda chi2

disp ('File fabresults.mat successfully created.');
```

### B.3.3 fabanalyze.m

```
disp('FABCAT Result Analyzer Release 1.4');

try

   load fabresults

catch

   disp('File fabresults.mat not found. Run fabreader.m first.');

end

disp(['Shots out of interval ', num2str(shotnumber(1)), ...
        ' to ', num2str(shotnumber(length(shotnumber)))]);
disp(['There are ', num2str(sum(exist)), ' shots available']);

plus = 0;

for i = 1 : length(xpoint)
  if ~isnan(xpoint(i))
    if xpoint(i) == 1
      plus = plus + 1;
    end
  end
end

disp(['There are ', num2str(plus), ' shots with xpoint available']);

conv = 0;

for i = 1 : length(status)
  if ~isnan(status(i))
    if status(i) == 1
      conv = conv + 1;
    end
  end
end
```

```

disp(['There are ', num2str(conv), ' shots with good reconstruction available']);
```

### B.3.4 fabplot.m

```
function fabplot(shot);

disp ('FABCAT radiation Plotter. Release 1.4');

mdsopen(shot);

disp(['Analyzing shot ', num2str(shot)]);

Pxpoint = mdsdata('\results::btomo:prad_above');
Pbelow = mdsdata('\results::btomo:prad_below');
Ptot = mdsdata('\results::btomo:prad_tot');
time = mdsdata('\results::btomo:time');

% plasma current
Ip=mdsdata('\results::I_p');
tIp = mdsdata('dim_of(\results::I_p)');
uIp = mdsdata('units_of(\results::I_p)');

% plasma average density
nel = mdsdata('\results::fir:n_average');
tnel = mdsdata('dim_of(\results::fir:n_average)');
unel = mdsdata('units_of(\results::fir:n_average)');

[tPoh,Poh]=tcvget('POHM');

figure('Name','Radiation of x-point');
plot(Pxpoint);
figure('Name','Radiation below the x-point');
plot(Pbelow);
figure('Name','Total radiation');
plot(Ptot);
figure('Name','Plasma current');
plot(Ip);
figure('Name','Plasma density');
plot(nel);
figure('Name','Ohmic power');
plot(Poh);
figure('Name','Plasma core flux surface');
tcvview('pvt',shot,.5)
figure('Name','Comparison between Ptot and Pohm');
plot(tPoh,Poh,'g',time,Ptot,'b');
legend('Ohmic power','Total radiated power');
axis([0 time(length(time))+0.1 0 max(max(Ptot),max(Poh))]);
figure('Name','Comparison between radiated powers');
```

```
plot(time,Ptot,'r',time,8*Pbelow,'b',time,8*Pxpoint,'g');
legend('Total radiated power','Power radiated below x-point','xpoint radiation');
axis([0 time(length(time))+0.1 0 max(Ptot)]);

mdsclose;
```