INT 189/96 February 1996

THE TCVXTI SOFTWARE PACKAGE IN MATLAB

M. Anton

with the contributions from

M.J. Dutch, W. Von der Linden, J.-M. Moret, Y. Peysson & S. Sagbo

CENTRE DE RECHERCHES EN PHYSIQUE DES PLASMAS
ASSOCIATION EURATOM - CONFEDERATION SUISSE
ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE
21 Av. des Bains - CH-1007 Lausanne - Switzerland

The tcvxti Software Package in MATLAB

M.Anton

with contributions from

M. J. Dutch
W. von der Linden
J. M. Moret
Y. Peysson
S. Sagbo

A kind of manual. 3rd version 6/2/1996

Contents

Overview

A set of MATLAB functions has been written to perform the data retrieval, calibration and the tomographic inversion of the Soft-X-ray data obtained on the TCV tokamak. The architecture and the use of this set of functions is the subject of section 1. Some datafiles which are needed for the calibration have to be provided by an external routine `eff_calibs_new` which uses spectral and angular dependences of the diode output to determine the detector efficiency wich will not be described here. Simulated data to test the performance of the tomography algorithms are provided by another external routine, the function `tcvxti_simulant`, which will briefly be described in a second section.

A second program serves to analyse and display the inversion results in a comfortable way. The structure and the use of this routine called `tcvxti_guck` will be described in section 3.

Finally, a complete alphabetic list of the functions with their `MATLAB` help texts is appended as well as some theoretical details on the calculation of the so-called T-matrix which maps the 2D (pixel-) emissivity to 1D line integrated data and on the inversion algorithms. A list of useful literature concludes this booklet.

1 The Inversion Package: `tcvxti` and `tcvxti_uifun`

1.1 Some Generalities on the Architecture

A general overview of the inversion routine is given in figure 1: The main parts are the **MATLAB** function **tcvxti_uifun** (a graphical user interface) and a set of 'external functions', also written in **MATLAB**. The working space, *i.e.* all variables are set up and declared as **global** by the scriptfile **tcvxti**.

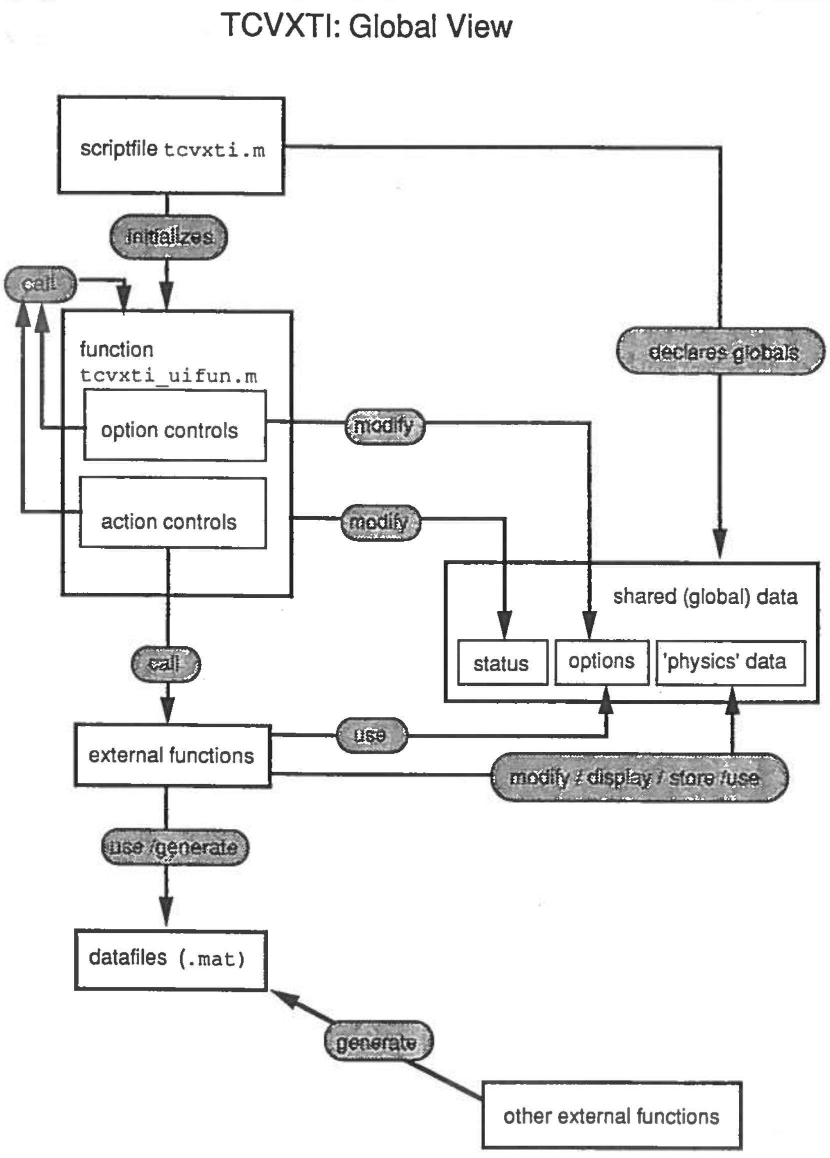


Figure 1: A global view of the **tcvxti** inversion package. Explanations see text

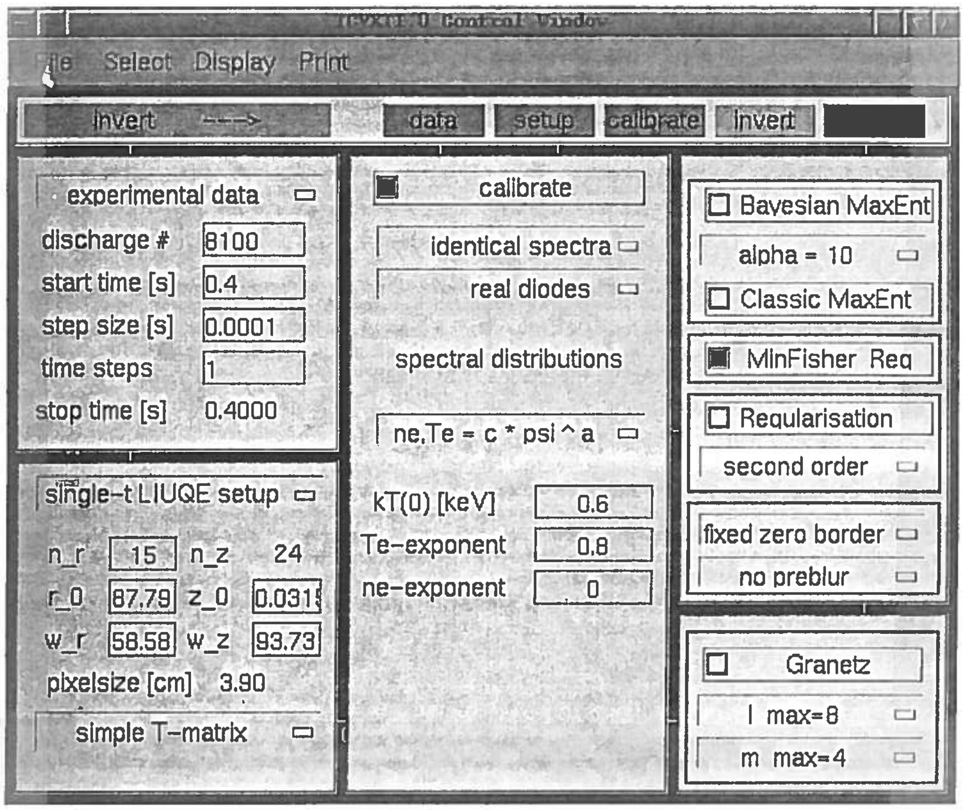
After the definition of the variables, **tcvxti** calls the graphical user interface **tcvxti_uifun** for the first time, passing as an argument the string **'initialize'**, which causes the function to initialize all graphical user interfaces (pulldown-menus, pushbuttons, popup menus, edit boxes, text and check boxes) in the first figure of the actual `MATLAB` session. This figure, in the following referred to as 'control window' is displayed in figure 2.

Every gui-interface calls back the main function `tcvxti_uifun`, passing an argument string 'action', which determines which action is to be performed. Most of these gui-controls modify option variables, *e.g.* concerning the inversion method or the display. These are referred to as `option controls` in figure 1. The change of some options may also affect the status variables. If, for example, a new shotnumber is chosen, a new setup for the inversion has to be calculated as well, so the status variables are set back to force the user to get a new setup. In general, the status variables determine which buttons are enabled or disabled, which message is to be displayed in the top left corner of the control figure (see fig.2), or simply, what can be done next and what has to be done next. The handles of all gui–controls are stored in the `UserData` matrix of the control window (see the `MATLAB` manual 'How to build a graphic user interface').

Only the `FILE` menu and the row of pushbuttons on top of the control window (see fig. 2) call external functions, *e.g.* the function `get_xtomo_data` to retrieve experimental data from the `MDS+` database (see [17]). The external functions generally make use of the option variables defined before. The input arguments of the functions are options and 'physics' data. The output of these functions only affect the 'physics' data. The external functions occasionally read data from `.mat` data files or from other `ASCII` data files with the extension `.res`, which again may have been provided by other functions, not included in the `tcvxti` package. After a successfull function call, the status is updated.

Before going into further details, the next section describes a typical session with `tcvxti`.

1.2 How to use tcvxti



experimental data
discharge # 8100
start time [s] 0.4
step size [s] 0.0001
time steps 1
stop time [s] 0.4000

single-t LIUQE setup
n_r 15 n_z 24
r_0 87.79 z_0 0.031
w_r 58.58 w_z 93.73
pixelsize [cm] 3.90
simple T-matrix

calibrate
identical spectra
real diodes
spectral distributions
ne,Te = c * psi ^ a
kT(0) [keV] 0.6
Te-exponent 0.8
ne-exponent 0

Bayesian MaxEnt
alpha = 10
Classic MaxEnt
MinFisher Reg
Regularisation
second order
fixed zero border
no preblur
Granetz
l max=8
m max=4

Figure 2: If you launch tcvxti, this is the first window which should appear. Explanations see text

After opening a MATLAB session, you have to make sure that the paths needed are included in your `matlabpath` (see below). If that's settled, you just type `tcvxti <CR>` which starts the program like described above. Three windows should now appear, one which looks like fig.2 and two others which are labelled `TCVXTI setup display` and `TCVXTI results display`, which will later on serve to display the setup and the results.

```
[anton.efficiency]
[anton.etendue]
[anton.matlab]
[anton.maxent]
[anton.public]
[anton.tcvti]
[anton.tcvxti]
[anton.tomofour]
```

```
[anton.xallg]
[moret.matlab]
```

First, you should decide wether you whish to work with simulated or with experimental data, which is done using the first popup menu named `experimental data` top left corner of the control figure (see fig. 2). If it's simulated data, a gui interface pops up which lets you choose a `.mat` file containing (hopefully) a simulated emissivity distribution and the line integrated data as well as the data on which TCV shot the simulation is based (for details on such a file see section 2).

If you want to invert experimental data, you choose a shot number and a start time, the latter in seconds, a step size (in seconds) and a number of time steps. The stop time is then calculated and displayed automatically.

The data to be inverted are actually retrieved by pressing on the `data` pushbutton. This pushbutton either calls `get_xtomo_data` for the experimental data, or sets the variables according to what was read from the file with the simulated data.

Now, a setup for the tomographic inversion should be chosen. There are four options in a popup menu:

```
      manual setup
single-t LIUQE setup
all-shot LIUQE setup
    setup from MDS
```

The second and third item on that list make use of the inverse equilibrium reconstruction code `LIUQE` [12]. For the `single-t LIUQE`, the last closed flux surface at the starttime specified in the shot data (see above) serves to determine the size and position of the pixelgrid. The pixelgrid is chosen such that the *entire* pixels of the first and last rows and columns of the grid are outside the last closed flux surface (LCFS). If the `all-shot LIUQE` setup is chosen, a pixelgrid is posed such that the last rows and columns of the grid are outside of *all* last closed flux surfaces available for that shot. If you want to add some inversions to ones already existing in the `MDS+ results` tree, it is a good idea to retrieve the pixelgrid from there using the `setup from MDS` item. With the manual setup, you can place and size the grid anywhere you like.

Via the first edit box in the setup frame, you modify the number of radial pixels. In case of a manual setup, the center of the grid as well as the horizontal and the vertical width can be put in using the other edit boxes. Otherwise the latter just work as displays. On the bottom,the actual pixelsize can be found. Below that, there is a popup menu which allows a choice between `simple T-matrix` and the `3D grid T-matrix`. Usually, the simple version should be sufficient, which just calculates the lenght of every line of sight in every pixel. The more complicated version (see Appendix) may be useful once systematic errors (detector calibration) are reduced.

Pressing the `setup` button now causes the call of several functions: first, the grid of quadratic pixels is set up with the help of `tcvxti_setgrid`. The contours of the last closed flux surface as well as the normalized flux contours on the pixelgrid are retrieved from `MDS+` using `tcvxti_getlcfs` and `tcvxti_getpsi`. After that, either the function `tmat_standard` or `t_omgrid` are called to calculate the T-matrix. Some statistics concerning the pixelgrid as well as the pixelgrid itself are then displayed in the `setup display` window. By the way: You may select ore deselect certain cameras or even single diodes with the help of the `SELECT` menu (top of the window).

With the check box on top of the middle part of the control window, you decide if you want to calibrate the detectors or no. A clear advantage of doing a calibration is that you get *Watts per cubic meter* for your emissivity instead of arbitrary units. What's more, sometimes the reconstruction might look a bit funny if you don't.

The calibration is performed calculating spectrum averaged efficiencies for the assumed spectral distribution of the plasma soft-X-ray radiation, as detailed in [2] or LRP-515. For that purpose, you can use the data like diffusion length and dead layer thickness of the actual detectors mounted on TCV. Or you may choose some standard parameters, for example 'all have $200\mu m$ diffusion length and half a micron dead layer'. This choice is implemented in the `real diodes / standard diodes` popup menu. The next option concerns the spectral distribution. You may want to assume identical distributions for all detectors, or you may want to do a simulation, taking into account that *e.g.* some detectors look right through the hot core of the plasma whereas some others just see the cold border. You choose this clicking on the `identical spectra / individual spectra` pushbutton. A further detail is the way you want to model your emissivity profile. Generally, you will assume that electron density and temperature profiles follow the normalised flux like $T_e, n_e = const \cdot (\psi/\psi_{axis})^a$, where the exponents a have to be guessed and put into the corresponding edit boxes below. The central electron temperature can be taken from `XTe` or Thomson scattering. A good choice for the profile parameters is 0 for the density, *i.e.* a flat density profile, and a more ore less peaked temperature profile. Experience shows that for a plasma without an X-point, $a = 0.8$ is generally a good choice whereas for diverted plasmas or pears $a = 0.5$ gives reasonable results. After that, you have to press the `calibrate` button. If you deselected the `calibrate` check box, you're directly switched to `invert`, then.

With the block of options on the right side of the control window, you choose your inversion method as well as some options or parameters concerning the inversion. If your `run` options are like they should be (or you think so, at least) you click on `invert`. The inversion routine chosen is launched for every timeslice. The progress report(*e.g.* `70 percent done`) is displayed in the top left corner of the control window (see fig.2) and in the window of your `MATLAB` session.

The `Bayesian MaxEnt` button lets you select a Bayesian Maximum Entropy algorithm (see *e.g.* [4]) originally written in `FORTRAN` by *von der Linden* [24]. The option `alpha` refers to this routine, it's an initial value for the regularization parameter to be optimized. For both MaxEnt methods (the second one is a 'classical' MaxEnt by *Peysson* [18]) a `preblur` of the T-matrix may be adequate to smooth the reconstruction a bit.

Considering the linear Regularisation [19], you have the choice between first or second order, which comes to minimizing the gradients or the curvature of the reconstructed emissivity profile together with the χ^2 of the fit. If you choose `fixed zero border`, the emissivity on the outermost pixels is forced to zero in the algorithm. This is reasonable if the border of the pixelgrid is outside the last closed flux surface, which should always be the case if you use `LIUQE` to get your setup.

The `MinFisher_Reg` algorithm is somewhat more sophisticated than the regularization, because the smoothing is weighted by the distribution itself: small signal, big smoothing and vice versa. This was originally introduced by *von der Linden* [25] and *Reinmuth* [20] where the Fisher Information of a probability distribution was minimised together with the χ^2 for one-dimensional deconvolution. The algorithm has been implemented by *Sagbo.* This method seems to provide the best compromise between calculation speed, precision and "beauty" as detailed in the TP report by *Sagbo.*

The last item in that list is the `Granetz` method as outlined in [9]. `l_max` and `m_max` denote the maximum numbers of radial nodes and poloidal harmonics, respectively.

Once the inversion is done, you may choose some display options in the menu `DISPLAY` on top of the control window. Click on `show` and you see what you got. It's displayed in the `results` window. You may have to click `show` several times, if there are many timeslices.

If you like your results, you may feel the desire to do two things: Have a printout, which is possible with the `PRINT` menu. Or you even want to keep your results, which is possible in two different ways: Either you put them in a `.mat` file or you put them in the `MDS+ results` tree. Both can be done using the items in the `FILE` menu. Just a few remarks on `FILE`: if you select `save results as .mat file`, the result should be clear. You are asked for a filename, it's saved and that's it. If you select `overwrite/write new MDS trace`, you have to type `yes` in your matlab session to convince the program to actually do so. If your not too sure about this point, you rather select `add inversion to existing results`. This operation may not be successfull if there are already data and the inversion method or the pixelgrid are not compatible. Should this be the case, you either repeat your inversion with the parameters you find in the `MDS` tree, ore you save your results to a `.mat` file. Tricky detail: before saving, all length units are converted to `m` instead of `cm` as before.

Below, you see a list of all traces used to store the results in the `MDS+` results tree

```
\RESULTS::XTOMO:CALIBR_PARMS
\RESULTS::XTOMO:CALIBR_TEXT
\RESULTS::XTOMO:CALIBR_TIMES
\RESULTS::XTOMO:CHI_SQUARED
\RESULTS::XTOMO:COMMENT
```

```
\RESULTS::XTOMO:EMISSIVITY
\RESULTS::XTOMO:METHOD
\RESULTS::XTOMO:RMESH
\RESULTS::XTOMO:SCALE_FACTOR
\RESULTS::XTOMO:TIME
\RESULTS::XTOMO:ZMESH
```

There are traces for the pixelgrid, the times where a reconstruction has been done, the times where different calibrations were necessary, the corresponding calibration parameters used, a string giving details on the calibration, a general comment, a string on the method employed, the χ^2 of the result and finally the emissivity itself. Attention: It's stored normalised (max=1), you recover the physical units and the real 'size' by a multiplication by the `SCALE_FACTOR`. For uncalibrated data this scale factor is set to negative unity. Should you want to stop, you also find the `quit` button in the `FILE` menu.

Once you have a lot of results, you preferentially should use the program `tcvxti_guck` to display and analyse them. How this works will be described later on.

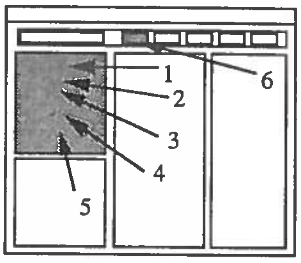
1.3 A fairly complete list of callbacks and function calls by `tcvxti`

The following tables give a complete list of all the callbacks of `tcvxti_uifun` and all other functions which `tcvxti_uifun` calls. The tables are grouped according to the main groups of `gui` controls described above. For almost every group of controls, there are two tables. The first shows the labels of the uicontrols together with the callback string they pass when calling `tcvxti_uifun`. The small diagram at their side indicates where you find these buttons on figure 2. The second table of every group lists all the subsequent calls of other `MATLAB` functions which are launched by pressing the button with the number given in the first column of all tables. For the initialization, the function call(s) are listed in table 1

tcvxti_uifun.m: init function calls			
#	callback	calls/reads	calls/reads
	'initialize'	xtomo_geometry.m	etendue_n2.m
			angular_fact_1.mat

Table 1: Function calls during initialization. The scriptfile `tcvxti` calls `tcvxti_uifun` and passes the callback string `'initialize'`. During this procedure, a function gets the geometry of the lines of sight and the corresponding values for the *étendue géometrique.*

tcvxti_uifun: data callbacks			
#	guicontrol	label	callback string
1	popup menu	experimental data	'data'
2	editable field	discharge #	'shot'
3	editable field	start time [s]	'starttime'
4	editable field	step size [s]	'shotpara2'
5	editable field	time steps	'shotpara2'
6	pushbutton	data	'getdata'



tcvxti_uifun: data function calls				
#	callback	calls/reads	calls/reads	calls/reads
6	'getdata'	get_xtomo_data.m	get_xtomo_gains.m MDS+	MDS+
6	'getdata'	sim_****_*.mat		

Table 2: Callbacks and function calls of `tcvxti_uifun` during data selection and retrieval.

tcvxti_uifun: SELECT menu and setup callbacks			
#	guicontrol	label	callback string
15	pulldown menu item	array # **	'selectcam'
	submenu (15) item	channel # **	'selectch'
	submenu (15) item	all channels on	'selectch'
	submenu (15) item	all channels off	'selectch'
16	popup menu	all-shot LIUQE setup	'whatsetup'
			'setgrid'
17	editable field	n_r	'pixelnum'
18	editable fields	r_0,z_0,w_r,w_z	'pixelpos'
18a	popup menu	simple T-matrix	'what_tmatrix'
			'tmatrix'
19	pushbutton	setup	'setgrid'
			'tmatrix'

tcvxti_uifun: setup function calls			
#	callback	calls/reads	calls/reads
16	'whatsetup'	tcvxti_chk_mds.m	
19	'setgrid'	tcvxti_setgrid.m	MDS+
		tcvxti_getlcfs.m	MDS+
		tcvxti_getpsi.m	MDS+
19	'tmatrix'	tmat_standard.m	
19	'tmatrix'	t_omgrid.m	raumwinkel_***.mat
		tcvti_testphip.m	
		plot_vessel	

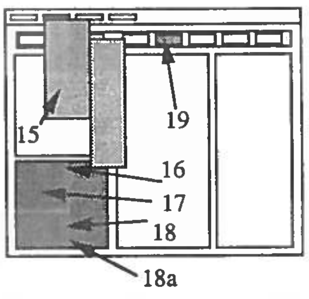
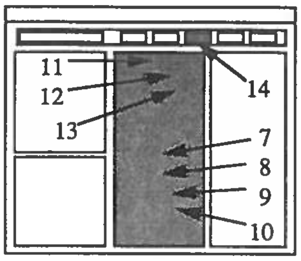


Table 3: Callbacks and function calls of `tcvxti_uifun` to determine the setup, *i.e.* lines of sight and pixelgrid. The SELECT menu lets you switch cameras or single diodes on and off. Setup otions can be chosen using the bottom left area of the control window. The pushbutton 'setup' and the two two popup menus may cause two callbacks.

tcvxti_uifun: simulation and calibration callbacks			
#	guicontrol	label	callback string
7	popup menu	ne,Te = c*psi ^ a	'simopt'
8	editable field	kT(0) $[keV]$	'simopt'
9	editable field	Te-exponent	'simopt'
10	editable field	ne-exponent	'simopt'
11	check box	calibrate	'ifcalib'
12	popup menu	identical spectra	'calspec'
13	popup menu	real diodes	'caldet'
14	pushbutton	calibrate	'calibrate'

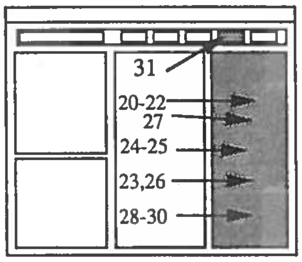


tcvxti_uifun: simulation and calibration function calls					
#	callback	calls/reads	calls/reads	calls/reads	calls/reads
14	'calibrate'	xtomo_calibrate.m	xtomo_geometry.m	etendue_n2.m	
				angular_fact_1.mat	
			get_detector.m		
			get_filters.m		
			get_spectrum.m	xrs_spectrum.m	
			eta_spec_av.m	linabs.m	cros2.m
				dxml.m	
			linabs.m	cros2.m	
			eff_calibs_10kV.res		
			eff_calibs_30kV.res		
14	'calibrate'	xtomo_simcal.m	xtomo_geometry.m	etendue_n2.m	
				angular_fact_1.mat	
			get_detector.m		
			get_filters.m		
			eta_theta_kev.m	linabs.m	cros2.m
				dxml.m	
			linabs.m	cros2.m	
			eff_calibs_10kV.res		
			eff_calibs_30kV.res		
			cut_lines.m		

Table 4: Callbacks and function calls concerning calibration or simulation options. The 'calibrate' pushbutton calls functions which calculate spectrum averaged efficiencies of photodiodes for the spectral distribution(s) chosen.

tcvxti_uifun: inversion callbacks

#	guicontrol	label	callback string
20	check box	New MaxEnt	'runopt'
21	check box	Classic MaxEnt	'runopt'
22	popup menu	alpha = 10	'runopt'
23	popup menu	no preblur	'runopt'
24	check box	Regularisation	'runopt'
25	popup menu	second order	'runopt'
26	popup menu	fixed zero border	'runopt'
27	check box	MinFisher_Reg	'runopt'
28	check box	Granetz	'runopt'
29	popup menu	l max=8	'runopt'
30	popup menu	m max=2	'runopt'
31	pushbutton	invert	'run'



tcvxti_uifun: 'invert' function calls

#	callback	calls/reads	calls/reads	calls/reads	calls/reads
31	'run'	preblur.m			
		makem_*.m			
		maxenttcv.m			
31	'run'	preblur.m			
		makem_*.m			
		mem_wvl.m	pfixedalpha.m	dxml.m	
			pgoldsec	pfixedalpha.m	dxml.m
31	'run'	makem_*.m			
		regulo_2d_tcvti.m	dxml.m		
31	'run'	makem_*.m			
		minfisher_reg.m	dxml.m		
31	'run'	granfun	rect_cont.m		
			phip.m		
			rml		
			wml		

Table 5: The right part of the control window (see fig.2) serves to switch between different inversion methods and to tune some parameters of the different methods. The 'invert' pushbutton finally launches the inversion.

tcvxti_uifun: DISPLAY and 'show' callbacks			
#	guicontrol	label	callback string
32	pulldown menu item	– > display inversion results	'dispwhat'
33	pulldown menu item	– > show some more, if possible	'disphow'
34	pulldown menu item	contour	'dispmode1'
35	pulldown menu item	pseudocolor	'dispmode2'
36	pulldown menu item	mesh	'dispmode3'
37	pulldown menu item	surface	'dispmode4'
38	pushbutton	show	'display'

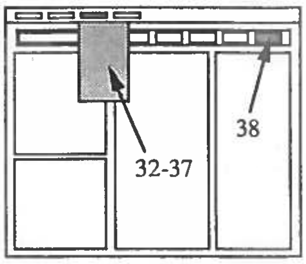


Table 6: Display options are modified with the help of the DISPLAY menu.

tcvxti_uifun: PRINT menu callbacks			
#	guicontrol	label	callback string
39	pulldown menu item	results display	'printres'
40	pulldown menu item	setup display	'printset'
41	pulldown menu item	PS_TCV_A	'selecprt1'
42	pulldown menu item	PS_PPH277	'selecprt2'

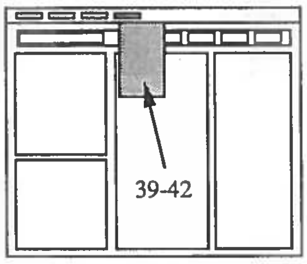


Table 7: Hardcopies of the setup and of the inversion results can be otained on two different printers using the PRINT menu.

tcvxti_uifun: FILE menu callbacks			
#	guicontrol	label	callback string
43	pulldown menu item	save results as .mat-file	'save'
44	pulldown menu item	overwrite/write new MDS trace	'store_mds'
45	pulldown menu item	add inversions to existing...	'merge_mds'
46	pulldown menu item	quit	'quit'

tcvxti_uifun: FILE menu function calls				
#	callback	calls/reads	calls/reads	calls/reads
43	'save'			
44	'store_mds'	tcvti_store_mds	MDS+	
45	'merge_mds'	tcvti_merge_mds	tcvti_get_mds	MDS+
			tcvti_store_mds	MDS+

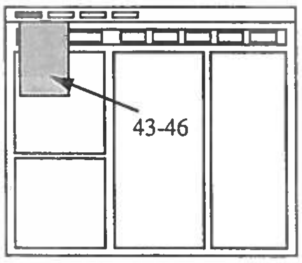


Table 8: Results can be stored as .mat using the 'save as mat-file' item in the FILE menu. They may as well be stored in the MDS+ tree.

2 Generating Simulated data using tcvxti_simulant

Simulated data sets are quite useful in testing the performance of a certain setup for the inversion, *i.e.* the combination of pixelgrid, lines of sight and the particular method chosen. They can be generated with the help of the simple function **tcvxti_simulant**. The function has to be called passing a string which contains a filename, like for example `tcvxti_simulant('sim_8100_1')`, where a matlab script file `sim_8100_1.m` has to exist somewhere containing the parameters of the simulation to be performed. A template for such a file can be found in

```
[anton.tcvxti.simulations]sim_template.m
```

The function `tcvxti_simulant` then calls several of the functions mentioned in the preceding sections, as the most important one notably `xtomo_simcal.m`, passing as arguments the parameters specified in the .m-file. After about 20 seconds a file `sim_8100_1.mat` is then written to the directory `[anton.tcvxti.simulations]` which should also contain the corresponding .m-file. The parameters of the simulations stored can thus easily be checked or printed out.

In addition to the parameters which are used for the calibration procedure (see above), the `funtype=3`-option allows to simulate mode structures. On top of a simulation using the equilibrium reconstruction at the starttime specified, a mode-like perturbation of the emissivity can be added. The mode number, the normalised flux coordinate, a relative amplitude (with respect to the unperturbed emissivity), a width (in terms of the flux, assuming a Lorentzian shape in ψ-direction) and a vector of phase angles have to be specified. If the increment of the phase angle is chosen not constant with time, a mode rotating with changing frequency or even a blocking mode can be simulated.

Below, a specimen parameter scriptfile for `tcvxti_simulant` is listed, namely `sim_9243_2.m`.

```
% [anton.tcvxti.simulations]sim_9243_2.m
%
%
% ==========  a parameter file for tcvxti_simulant =====================

% ------------------------------------
% ---- discharge number
%

shot=9243;

% ------------------------------------
% ---- start time [s]
%

tstart=0.746;

% ------------------------------------
% ---- step size  [s]
%

dt=0.0001;

% ------------------------------------
% ---- time steps
%

schritte=41;

% ------------------------------------------------
% ---- setup type for meshgrid
%
%  i_setup=2:  single t LIUQE setup @start time
%  i_setup=3:  all-shot LIUQE setup
%

i_setup=2;


% ------------------------------------------------
% ------------------------------------
% ---- number of radial pixels (suggested: 15)
%

nx=15;


% ------------------------------------------------------
% ---- type of diodes to be simulated
%
% i_dioda=1: real detectors (LD20-5T installed on TCV)
% i_dioda=2: standard detectors (L=200mu, dp=0.5mu)
```

```
%

i_dioda=1;

% --------------------------------------------------------
% --- function type
%     funtype = 1         single step, no modes
%     funtype = 3         several steps possible, modes
%

funtype=3;

% ------------------------------------------------------------------
% --- spectral distribution described by
%
%   kTe0         central electron temper. in [keV]
%   Te = const1 * psi^a_te   temparature profile
%   ne = const2 * psi^a_ne   density profile
%   with psi being the normalised flux
%

kTe0=0.6;
a_te=0.8;
a_ne=0;

% -----------------------------------------------------------------------------

if funtype == 3

   % --- mode structures can be added: it is assumed that the emissivity X(psi)
   %     changes following
   %     delta_X = X0*(1/(1+(psi-psi_mode)^2/HWHM^2)^2*cos(m_mode*theta + phi)
   %

   % ---------------------------------------------------------------
   % --- poloidal mode number m
   %

   m_mode=2;
   % --------------------------------------------------------------
   % --- relative amplitude of the mode r0 so that X0 = r0 * X(psi)
   %

   amp_mode=0.1;

   % --------------------------------------------------------------
   % --- psi - coordinate of the mode
   %

   psi_mode=0.85;

   % --------------------------------------------------------------------
   % --- HWHM of the mode, assuming a Lorentzian shape with respect to psi
   %

   HWHM_psi_mode=0.15;

   % --------------------------------------------------------------------
   % --- phase angles of the mode
   %
   phi1=0;
   phi2=6*pi;

%   phi=linspace(phi1,phi2,schritte)

   dphi=(phi2-phi1)*(1-linspace(1,0,schritte)).^2;
   dphi=dphi(length(dphi):-1:1);
   phi=phi2-dphi;

   % other variations of phi with time are possible!

end % end if funtype


% ======================== end of the parametfile ========================
```

3 Analysis and Display using `tcvxti_guck`

3.1 How `tcvxti_guck` is built and how it works

With **tcvxti_guck** you can look at the results of your previously finished tomographic inversion and analyse the features of the inversion with the help of the Singular Value Decomposition, further referred to as SVD (see *e.g.* [19, 8]).

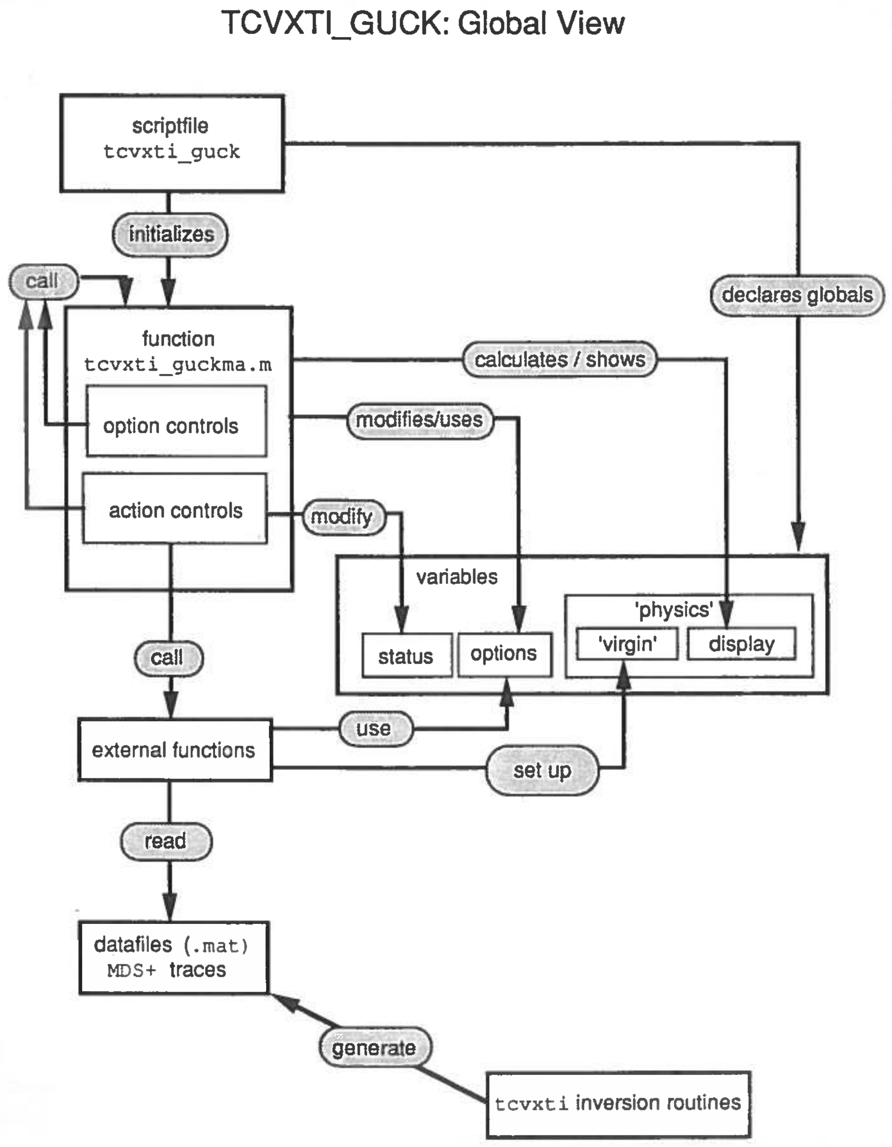


Figure 3: A global view of the display and analyses routine **tcvxti_guck**. Explanations see text.

Like above, a coarse overview is at hand. Launching the **MATLAB** script file **tcvxti_guck** initializes the graphic interface by a call to **tcvxti_guckma**, passing the argument string **'initialize'**. This causes a set of gui interfaces to pop up in the first window of the **MATLAB** session. As for **tcvxti_uifun**, the handles of all controls are stored and retrieved from the `UserData` matrix of the control window.

At the same time, the shared data are declared as global. As above, there are status variables, options and 'physics' data. Here, only three external functions are needed, namely `tcvxti_get_mds`, `tcvxti_getlcfs` and `tcvxti_getpsi` to retrieve the inversion results as well as the last closed flux surface and the normalized flux contours from the `MDS+` results tree. The data retrieved from the tree remain untouched in memory, which is indicated in fig. 3 as `'virgin'`. All manipulations which can be performed with the different `gui` buttons, *e.g.* an SVD filtering of the result, generate a treated subset of the original data for display purposes. This set of data is referred to as `display` in figure 3. Once the data are there, `tcvxti_guckma` keeps calling itself, passing different string arguments which determine the actions to be performed next.

3.2 How to spend plenty of time with tcvxti_guck looking at tomographic inversions

Things tend to repeat themselves. In that sense I shall now describe how a typical session with tcvxti_guck takes place.

After you launched your MATLAB session, you have to make sure all the paths are there. This time it's just two, namely

```
[anton.tcvti]
[anton.tcvxti]
```

Now you start the program by typing tcvxti_guck, which initializes the gui-interfaces by calling tcvxti_guckma('initialize'). (This is analogous to tcvxti / tcvxti_uifun.) Two figures should pop up. The first is the control window and is labelled TCVXTI_GUCKMA.O. You see a hardcopy of this window in figure 4. The second simply serves as the display.

On the left bottom of the control window, you see an edit box with the label shot #. That's where you can type in the number of the TCV discharge you want to analyse (you're not surprised, I guess...). If you then go to the File menu top left, you have three options:

```
load results from .mat file
load results from MDS
quit
```

Since you've just started, the third point on the list is probably not what you want. If you select the first point, the shot number you typed in is worthless because then a menu pops up where you can choose any .mat file with your results. If you select the second point on the above list, the MDS+ results tree is opened for the shotnumber you chose and *all* reconstructions which can be found there are transferred to your MATLAB working space. The minimum and maximum of the time *with* available reconstructions are displayed right next to the labels tmin and tmax (see figure 4). At the same time, data from the equilibrium reconstruction, which you may want to compare to the XTOMO results, are fetched from MDS+ as well.

Now, you should remark that two graphs have appeared in the control window showing the maximum of the emissivity versus time for the reconstructions you loaded. At the beginning, both graphs should look the same. But you can change that by typing different times in the edit boxes which are labelled tstart and tstop. With the help of these guicontrols you choose the subset of the reconstructions you're specially interested in. In your controlwindow you should see then that the selected part of the reconstruction is displayed in the right figure and is highlighted (or rather 'dark'lighted in fig. 4) in the left part, where you still see all reconstruction timeslices available in memory.

Before you actually can see the results, I have to draw your attention to the two pushbuttons on the bottom right of the control window, the ones which are labelled **APPLY** and **SHOW**, respectively. You *have* to click on **APPLY** to encourage the program to do a lot of things for you:

- take the selected subset of the reconstructions
- do an SVD of the matrix which contains the corresponding reconstructions
- calculate one horizontal cut through the emissivity distribution and
- calculate two vertical cuts through the emissivity, one where on the average the position of the maximum emissivity can be found (like in the horizontal case) and one along the path where the YAG Laser of the Thomson scattering diagnostic usually fires through the vessel.
- enable the **SHOW** button

Depending on the number of reconstructions involved, this may take some time.

What **SHOW** does, should be evident, somehow. The **SHOW** button, together with the **APPLY** button, the `File` and the `Print` menus constitute the whole set of 'action' controls (compare figure 3). All the rest is just options,a part of which has already been discussed. We'll get the rest in what follows now.

With the sliders `row`, `column` and `speed` you influence the number of rows and columns you want your display window to have. This enables you to see several reconstructions at once. If you set nonzero `speed`, you automatically reset the number of rows and columns both to 1, in which case the reconstructions are automatically displayed one after another[1]. Vice versa the speed is reset to zero if you choose to see more than one reconstruction at once.

To understand the SVD stuff and what the **SVD** menu as well as the `topos/chronos` item in the `Display` menu means, I suggest you first have a look at a paper by *Dudok de Wit et al.* [8]. Other helpful informations may be found in the **MATLAB** reference manual or in [19]. Here, I can just give a *very* brief overview:

The variable **X**, a matrix, contains the reconstructed emissivity for all pixels and all analysed timeslices. Every column of **X** represents a different timeslice, every row corresponds to a certain pixel. If you do an SVD in `matlab`, which reads

```
[u,s,v]=svd(X);
```

you can recover X from the three matrices `u,s,v` like this:

```
X=u*s*v';
```

[1]The **MATLAB** movie command has not been used because there's a certain tendency to crash. Maybe there are reasons to that.

The columns of u are a kind of spatial 'eigenmodes' (topos), which are linked to their temporal 'eigenmodes' (chronos), the columns of v via the corresponding 'singular value' in the diagonal matrix s. This means, there are always pairs of spatial and temporal vectors which represent a certain 'feature' of the spatial and temporal evolution of the emissivity in the time interval under consideration. The elements of the 'diagonal' matrix s are sorted in descending order, so that the first pair of topos/chronos will contain the most important information on the reconstruction. You can look at just this if you do not recover the whole matrix X like above, but if you leave all higher components aside, *e.g.* like this:

```
Ximportant=u(:,1)*s(1,1)*v(:,1)';
```

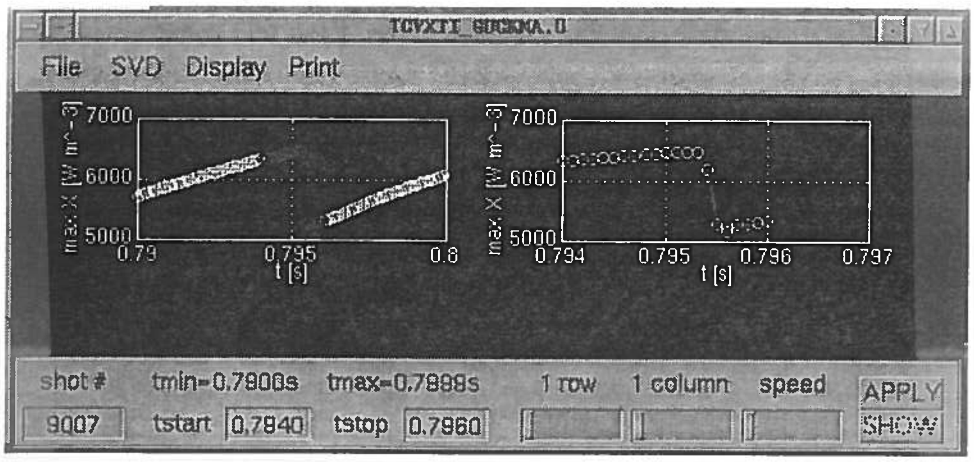
This represents a kind of filtering. On the other hand, you may want to see just the changes, not the gross features. This can be done in the following way:

```
Xchanges=u(:,2:20)*s(2:20,2:20)*v(:,2:20)';
```

You can display the topos/chronos pairs by choosing the corresponding item in the `Display` menu and then clicking on `SHOW`. The filtering or suppressing of gross features is done using the items of the `SVD` menu. You should by now have an idea what this means. Just try it. It won't hurt neither you nor the computer. Just keep in mind that you have to click `APPLY` after you selected or deselected something in the `SVD` menu.

All the rest in the `Display` should be evident. Find out by trying. The last three items on the list let you add the vessel, the last closed flux surface or the flux contours.

The `Print` menu finally just brings up *Jean Marc Morets* printmenu in your display figure. Just in case you want a hardcopy. If you've had enough, there's always the `quit` item in the `File` menu.



shot # 9007 tmin=0.7900s tmax=0.7999s tstart 0.7840 tstop 0.7960 1 row 1 column speed

Figure 4: The control window of `tcvxti_guck`. Explanations see text.

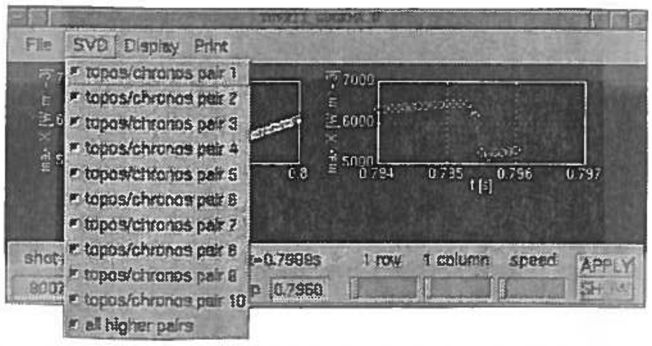


Figure 5: The `Display` menu. Discover the wonderful world of tomographic reconstructions .... the names are hopefully self-explanatory.

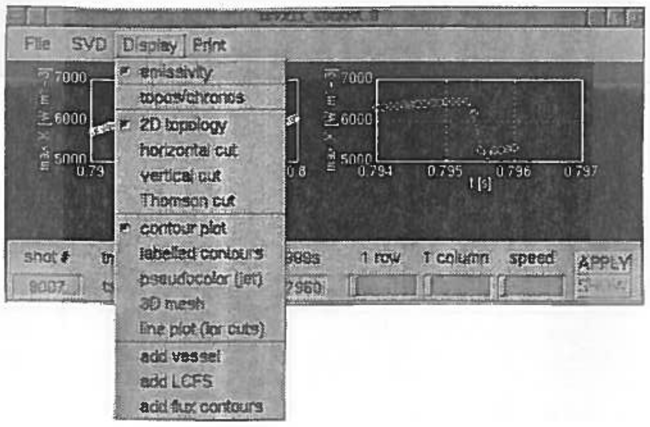


Figure 6: The `SVD` menu. You can select or deselect certain pairs of spatial and temporal 'eigenmodes' of the reconstruction. But please look at them first, using the `topos/chronos` item in the `Display` menu. See above.

3.3 A list of all callbacks

In the following table, a list of all callbacks and calls to other `MATLAB` functions is given. Since `tcvxti_guck` is not that complex, we just have this one.

tcvxti_guckma callbacks				
#	guicontrol	label	callback string	functions called
1	editable field	shot #	'shotnumber'	
2	pd menu item	load res from .mat	'load_mat'	
3	pd menu item	load res from MDS	'load_mds'	tcvti_get_mds tcvxti_getlcfs tcvxti_getlpsi
4	editable field	tstart	'timewin'	
5	editable field	tstop	'timewin'	
6	slider	* row	'rows'	
7	slider	* column	'cols'	
8	slider	speed	'dispspeed'	
9	pushbutton	APPLY	'apply'	
10	pushbutton	SHOW	'guckma'	
11	pulldown menu	SVD	'svdselect'	
12	pulldown menu	Display	'dispopt'	
13	pulldown menu	Print		printmenu

Table 9: Callback strings and external function calls by numbers of gui controls of `tcvxti_guckma`

4 The Function Dictionary

The following pages contain the help texts of all matlab functions used in the `tcvxti` environment in alphabetical order. In some cases, the program listings will have to be consulted as well as some of the publications cited throughout the text of this document to get a full understanding of what is going on. I hope the list will be helpful in spite of many imperfections.

The program `eff_calibs_new.m` with the help of which the files `eff_calibs_*.res` containing the diode parameters were generated, has not been included.

cut_lines

```
----- [anton.xallg]

function

 [xchord,ychord,relevant]=cut_lines(xchord,ychord,xmin,xmax,ymin,ymax);

      "xchord,ychord"  matrices are 'squeezed' into the box specified by
      "xmin,xmax,ymin,ymax". Chords which DO cross this rectangle are
      sorted out, their numbers are specified in 'relevant' and returned.
      i.e. the size of the output "xchord,ychord" matrices may be smaller
      than the corresponding inputs.

      see XTOMO_GEOMETRY for the definition of xchord,ychord

-------- MA 1995
```

cros2

```
----[anton.efficiency]

function [RHO,NER,E1,E2,A1,A2,A3,A4]=cros2(filter)

CROS.DAT, 23/6/1993
routine to provide cross-section data originally
stored in Fortran database file CROS.DAT
              M. J. Dutch Feb 1992

 SYNTAX:      function [RHO,NER,E1,E2,A1,A2,A3,A4]=cros2(filter)

  INPUT:
                  FILTER = string containing name of filter material
                     e.g. 'BE' , 'AL' , 'FE' etc 'O ', 'N '
OUTPUTS:
             RHO = density in g/cm^3
             NER = number of energy ranges
             E1,E2 = limits of energy range (keV)
             A1,A2,A3,A4 = coeffs of polynomial fit
                    to log10(XS) vs log10(E)
```

DXML

```
 DXML    Dec-eXtended-Math-Library interface
      DXML('*',A,B[,opa,opb,a,b,C])  a*opa(A)*opb(B)+b*C
      DXML('\',A,B[,op,nbloc])       op(A)\B
      DXML('chol',A[,B])             A\B or inv(A), A being sym pos def
      DXML('gsep',A,B)             eig(A,B) A,B being sym
      DXML('exp',A)                    exp(A)
      op = 'N'ormal 'T'ranspose
```

------ JMM-----------

eta_spec_av

```
------[anton.efficiency]

 function etabar=eta_spec_av(ABS,D,dd,w,L,FILTER,fd,ff,theta,KEV,EDIST,model)

 is a subroutine used by eff_calib.m

 output: a table length(L) x length(theta) of spectrum averaged efficiencies

 inputs:      ABS         -the absorber Material , e.g. 'SI'
              D           -thickness of the absorber
              dd          -dead layer thickness in microns
              w           -width of depletion zone in microns
              L           -a col. vector of diffusion lengths in microns
              FILTER      -a col. vector of filter materials, eg.
                            ['SI';'N ';'O ']
              fd          -a col vector of filter thickness in microns
              ff          -a col vector of flags: ff=1: no change of filter
                            thickness with theta, ff=0: fd->fd/cos(theta)
              theta       -a row vector of angles theta in radians
              KEV         -a row vector of photon energies in keV
              EDIST       -normalized spectral distribution, sum(EDIST)=1
              model       -a number to specify the model used (1,2,3):

        1: width w neglected, D very large (Kingston)
        2: w,D and L taken into account (own mod. of Kingston's formula)
        3: only w and L (formula from Sze, Physics of Semiconductur Devices)

--------------------M.Anton 8.7.1994-----------------------------------------
```

eta_theta_kev

```
------[anton.efficiency]

 function eta=eta_theta_kev(ABS,D,dd,w,L,FILTER,fd,ff,theta,KEV,model)


 output: a table length(theta) x length(KEV) of efficiencies

 inputs:      ABS     -the absorber Material , e.g. 'SI'
              D       -thickness of the absorber
              dd      -dead layer thickness in microns
              w       -width of depletion zone in microns
              L       -a col. vector of diffusion lengths in microns
              FILTER  -a col. vector of filter materials, eg.
                        ['SI';'N ';'O ']
              fd      -a col vector of filter thickness in microns
              ff      -a col vector of flags: ff=1: no change of filter
                      thickness with theta, ff=0: fd->fd/cos(theta)
              theta   -a row vector of angles theta in radians
              KEV     -a row vector of photon energies in keV
              model   -a number to specify the model used (1,2,3):

        1: width w neglected, D very large (Kingston)
        2: w,D and L taken into account (own mod. of Kingston's formula)
        3: only w and L (formula from Sze, Physics of Semiconductur Devices)

--------------------M.Anton 5.1.1995----------------------------------------
```

etendue_n2

```
-------[ANTON.ETENDUE]

function

AOMEGA=etendue_n2(b1x,b1y,b1z,b2x,b2y,b2z,z01,z02,X0,cw);

calculates 'etendue geometrique' A*OMEGA for two rectangular apertures

 ! ALL LENGTHS IN  MM !

input: b1x b1y b1z : width, height and thickness of ap. 1
b2x b2y b2z :   ap. 2
z01 : distance detector - ap.1
z02 : distance detector - ap.2
X0 : shift centers of detector arr. - ap1
cw : det # clockwise if cw==1

output: AOMEGA : etendue A x OMEGA in mm^2 x steradians (20x1)

------M.Anton 24.8.93 & 29.9.1994 & 11.4.1995 -----------------------------
```

get_detector

```
[anton.efficiency]--------------

function [ABS,D,w,dd,dp,infostr]=get_detector(whatdiode,dp)

      input:whatdiode      1:LD20-5T       (Centronic)
                           2:OSD-50-4X     (Centronic)
                           3:AXUV100       (IRD)

            dp             p+ layer thickness, if not specified
                           the program asks for input

      output:ABS           absorber Material, e.g. 'SI'
             D             absorber thickness in microns
             w             width of depletion zone
             dd            Si3N4 thickness
             dp            see above
             infostr       string with the name of the diode

---- MA 3/1/95 ------------------
```

get_filters

```
[anton.efficiency] ----------

function [FILTER,fd,ff,filt,infostr]=get_filters(dd,dp,spec,filt)

input:        spec     -optional, only important if used for visible light
              dd       -thickness of Si3N4 dead layer
              dp       -thickness of p+ dead layer
              filt     -integer,if already decided which one to use
                1       47u Be, Si3N4
                2       125u Be,air,Si3N4
                3       Be, B, Si3N4
                4       only Si3N4
                5       47/cos u Be,Si3N4
                6       a la carte
                7       Be+Oxides, Si3N4
                8       47u Be + SiO2
                9       only SiO2
               10       only 125u Be and air


output:       FILTER   -a nfilter x 2 string matrix
              fd       -filter thickness in microns, column matrix
              ff       -fixed filters 0:angular dependent thickness, 1:fixed
              infostr -string describing the filter set

--------------MA 6/10/94 ----------------------------------------------------
```

get_spectrum

```
[anton.efficiency] -------------

function [KEV,EDIST,spec,infostr]=get_spectrum(spec,parm,peak,dens)

input:
             spec, an integer:              parm:
             1:  visible - IR                wavelength in nm
             2:  Plasma Bremsstrahlung       temperature in keV
             3:  line integrated Plasma       central T in keV
             4:  W-Xray source              source voltage in kV
             peak       peaking factor
             dens       density exponent

outputs:     KEV                  vector of photon energies in keV
             EDIST                normalised energy distribution
                                  (sum(EDIST)=1)
             infostr              a string describing the spectrum

subroutines:  xrs_spectrum(U,KEV) gets fits to experimental XRS-PHA-spectra

MA 6/10/94-------------------------------------------------------------------
```

get_xtomo_data

```
----[anton.public]

function

      [sig,t]=get_xtomo_data(shot,t1,t2,dt,fans,angfact);

      input  shot     TCV shot #
             t1       start time
             t2       stop time
             dt       timestep
             fans     camera switch, e.g. [0 0 0 0 0 0 1 0 1 0];
             angfact  etendue, size:       [20 x 10]

      output sig     xtomo signals, size:       [sum(fans) x length(t)]
             t       time
             ATTENTION: length(time) may be shorter than foreseen !!

----- MA 1995
```

get_xtomo_gains

```
-----[anton.public]

  function gains = get_xtomo_gains(shot);

  emerged from MJD's  XTOMOSEQ/XTOMO_LOAD_GAINS
  purpose: load gains of a certain shot nr. from the database
  usage:   'gains' is a row vector containing the gains of
           [array_001(det.1..020), array_002, det. 001...020 etc]



-------------------------MA 23/2/94-----------------------------------------
```

granfun

```
--- [anton.tomofour]granfun.m

function [g,xgran,ygran,p,phi,radius,nlines,nzcho,nzpts]=...
granfun(f,m_max,l_max,i_m,i_p1,i_p2,xchord,ychord,xpix,ypix,con_x,con_y,cx,cy)

output:
      g               emissivity, column vector
      p,phi           angle and impact parameter of lines of sight
      radius          radius of the inversion region
      nlines          numbers of chords which are taken into account
      nzcho           number of zero chords added
      nzpts           number of zero points added

input:
      f               chord brightness (data,column vector)
      m_max           max number of poloidal harmonics
      l_max           max degree of Zernike polynomials (radial function)
      i_m             1: use pseudoinverse from SVD to get aml-coeffs
                      2: use regularization method to get aml-coeffs
                      3: use '\' - dxml routine
      i_p1            1: plot p,phi 0: don't
      i_p2            1: plot 5 most important harmonics ,0: don't
      x/ychord        lines of sight
      x/ypix          pixel grid for display
      con_x,con_y,
      cx,cy           optional, LCFS and magnetic axis from LIUQE

subroutines: [phi,p]=phip(xchord,ychord,cx,cy);          phi, p of chords
             zpol_ml=rml(m_max,l_max,rr);                Zernike polyn.
             W=wml(m_max,l_max,p,phi,nl,nzcho,nzpts);    W-matrix
             [con_x,con_y,cx,cy]=rect_cont(xmin,xmax,ymin,ymax)

algorithm: Granetz method for tomographic inversion

---- M.Anton 23/1/95 -----------------------------------------------
```

linabs

```
---------[ANTON.EFFICIENCY]

   function LINA=LINABS(filt,KEV)

   arguments     FILT:    column vector of elements like ['BE';..]
                 KEV:     vector of photon energies in keV
   returns       LINA:    a matrix with as much rows as FILT and as much
                          columns as KEV

   calculates linear absorption coefficients (unit 1/mu)
   in the energy range given by KEV
   cross section calculations are based on
   SX-ray cross-section data in file [ANTON.MATLAB]CROS2.M
                                  M. Anton Jun 1993
   the whole thing is extracted from MICHAEL DUTCH's diode_resp.m
   modif for matlab4 23.6.94

-----M.ANTON------------------------------------------------------------
```

maxenttcv

```
-------[anton.maxent]-------

 function

 [X,chi2]=maxenttcv(mode,Y,dY,T,zrs,display,lambda,eps1,eps2,gamma,Xinit)

       Maximum entropy algorithm

       Input: - mode: maximum entropy algorithm        [1,1]
                                   - (1) Paraboloid
                                   - (2) Davidon-Fletcher-Powell
                                   - (3) Gull-Daniell-Delsuc
                                   - (4) Gull-Daniell-Wu
              - Y: normalized line-integrated emission [nchord,1]
              - dY: relative error of the line-integrated emission [nchord,1]
              - T: matrix of transfer between the local emissivity and the line
                integrated emission [nchord,ncell]
              - zrs: a fictive chord is added to ensure that cells which are not
                crossed by any chord do not contribute to the local emission [1,1]
                (optional,default = 1)
              - display: plot the convergence in real-time (default = 0) [1,1]
              - eps1: convergence parameter (optional,default = 0.005)
              - eps2: convergence parameter (optional,default = 0.005)
              - gamma: convergence parameter (optional,default = 2)
              - lambda: Lagrange multiplier (optional,default = 1e-9)
              - Xinit: initial guess for the local emissivity (optional, default = flat
                profile) [1,ncell]
        Output:
              - X: local emissivity [1,ncell]
              - chi2: global error between Y and T*X (chi2<= 1 for an accurate
                inversion) [1,1]

 by Y.PEYSSON CEA-DRFC 13/07/1994 <peysson@fedv09.cad.cea.fr>
```

makem_1

```
----[anton.tcvti]---

function M=makem_1(Y,T,xpix,ypix,flat)

set up flat default model with zero border
(used for different  Xtomo  algorithms, regulo_ , max_ent)

      input: Y         line int data
             T         transfer matrix
             xpix
             ypix      pixel coordinates
             flat      1: totally flat model, just borders set zero
                       0: simple estimation used (see HOlland and Navratil)

      output:M       default model for the emissivities
-------------MA 2/12/94 -------------------
```

makem_2

```
----[anton.tcvti]---

function M=makem_2(Y,T,xpix,ypix,con_x,con_y,cx,cy,flat);

set up flat default model with zero border
(used for different  Xtomo  algorithms, regulo_ , max_ent)

      input: Y          line int data
             T          transfer matrix
             xpix
             ypix       pixel coordinates
             con_x...   LCFS contour and magnetic axis
             flat       1: totally flat model, just borders set zero
                        0: simple estimation used (see HOlland and Navratil)

      output:M          default model for the emissivities

-------------MA 31/8/95 -------------------
```

mem_wvl

```
---[anton.maxent]

function [X,chi2,alphaopt,S,levi,ppdb]=mem_wvl(Y,dY,T,alpha0,M)

Bayesian MaxEnt algorithm

input:  Y        line integrated data       size: nl x 1
       dY        RELATIVE errors of Y    size: nl x 1
        T        transfer matrix        size: nl x npix
       alpha0    initial value for regularisation parameter alpha  size: 1x1
       M         default model              size: npix x 1

output: X         inversion result       size: npix x 1
        chi2      its chisquare
        Smem      its information entropy
        alphaopt  the regularisation parameter
        levi      the logarithm of the alpha evidence
        ppdb      posterior probability of the solution in dB

subroutines:
        makem          make the default model if not supplied
        pgoldsec       golden section search
        pfixedalpha    probability of result for fixed alpha, MAIN SUBROUTINE

Algorithms:        W.v.d.Linden, IPP Garching Ber. OP & NUMERICAL RECIPES
Matlab implementation:        M.Anton       CRPP
 --------- 11/94 & 12/10/95------------
```

minfisher_reg

```
tomographic inversion using the minimum fisher formalism

use:          [g,chi2]=minfisher_reg(f,df,T,xmesh,ymesh,i_disp,i_zero,g_model)
outputs:      g         [nx*ny x 1]    reconstructed emissivity distribution
              chi2      [1 x 1]        reduced chisquare, i.e.
                                       chi2=(Ts*g-fs)'*(Ts*g-fs)/length(f)
                                       where Ts=diag(1./(df.*f))*T,fs=1./df
 inputs:       f        [nl x 1]        chord brightness
              df       [nl x 1]        RELATIVE errors of f
              T        [nl x nx*ny]    transfer matrix
              xmesh    [1 x nx]        x-coordinates of pixel centers
              ymesh    [1 x ny]        y-coordinates of pixel centers
              i_disp   [1 x 1]         a flag,
                                       if 0, no output during iteration
                                       if 1, the actual value for chi2 is
                                       displayed during each iteration.
              i_bord   [1 x 1]         a flag,
                                       if 0, boundary conditions are not
                                       taken into account; if 1, the default
                                       model m is used to modify the T-matrix
                                       to assure g=zero where the default
                                       model g_model is zero
                                       (see regulo_2d_tcvti.m)
              g_model [nx*ny x 1]      the default model for g; g_model is OPTIONAL
                                       if not specified, g_model is calculated
                                       within the routine. used only if
                                       i_zero==1.(see regulo_2d_tcvti.m)

---- Serge Sagbo -------------------
---- Informatique 4 eme annee ------
-----Projet de 7e semestre    ------
-----Responsable: Mathias Anton ----
---- CRPP/EPFL/1995-1996      ------
---- Creation: 21-11-95       ------
----- Version : 23-1-96     --------
```

omgrid_main

```
----[anton.public]

function [OMEGA,rho_grid,zet_grid]=omgrid_main(i_detec,fans)

inputs:

 i_detec:     =2: Xtomo prototype cameras (shot# < 6768)
              =1: Xtomo 9-cameras         (shot# > 682x)

 fans:    camera switch, 1=on,0=off (1x10)

outputs:

 files named 'raumwinkel_###.mat' in [anton.public.raumwinkel]
 which are used by the function t_omgrid

uses:

 omgrid_3d.m

---------------- M.Anton 29/5/95 -------------------------------------------
```

omgrid_3d

```
---- [anton.tcvti]

function
[omega,rho_grid,zet_grid,dV]=omgrid_3d(Kb1,Kb2,Kb3,Kb4,Kd1,Kd2,Kd3,Kd4,ivert);

Calculates a 2D-matrix omega from a 3D grid definde inside omgrid_3d.
The grid fills approximately a 40cm 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:                                       detector
                             each K has three components:
                                  K..(1): radial     rho
                                     (2): vertical   zet
                                     (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:       not used in the main routine omgrid_main.m. Just dummies....
uses:         projbl.m

------ MA 29/5/95 --------------------------------------------------------
```

pfixedalpha

```
---[anton.maxent]---

function [lPa,chi2,S,E]=pfixedalpha(D,Dtild,sigma,T,Ttild,E,M,alpha)

input        D:        line integrated data
             sigma:    their standard deviations
             E:        initial guess for the emissivity
             M:        default model
             T:        Transfer matrix
             alpha:    regularisation parameter
             Ttild:    T/sigma
             Dtild:    D/sigma  convenient definitions
output       lPa:      log posterior probability
             chi2:
             S:        information entropy of the solution with resp. to M
             E:        emissivity solution for given alpha

LPS-approach. Newton iteration for every call.

Algorithm by W.von der Linden, MPG-IPP Garching

---matlab: MA 30/11/94---------------------------------------------
```

pgoldsec

```
---[anton.maxent]---

function [lPopt,alphaopt,chi2opt,Sopt,Eopt]= ...
    pgoldsec(ax,bx,cx,fa,fb,fc,D,Dtild,sigma,T,Ttild,E,M);

golden section search (NUMERICAL RECIPES) modified for p(alpha), alpha>0

subroutine pfixedalpha calculates the posterior probability
           Palpha for fixed regularisation parameter alpha

--- matlab implementation: MA 1/12/94
```

phip

```
--- [anton.tomofour]

 function [phi,p]=phip(xchord,ychord,cx,cy)

 subroutine of granfun.m

 calculate phi and p for the cormack tomography method
 from xchord,ychord and the center of plasma cx, cy

 ----- M.Anton 23/1/95
```

plot_vessel

```
---[anton.matlab]

function plot_vessel(rzvin,rzvout)

rzvin,rzvout contain the coordinates of the vessel in cm, stored
in tcv_vesc.mat
---------MA 1994
```

preblur

```
--- [anton.maxent]

function Tb=preblur(T,b,nx,ny)

      output:Tb      preblurred T-matrix
      inputs:T       'virgin' T matrix
             b       preblur width (in pixels)
             nx      number of hori. pixels
             ny      number of vert. pixel

------ MA 15/2/95 -----------------------------------------------------
```

projbl

```
--- [anton.public] -----------------------------------------------------

  function [xl1,yl1,xl2,yl2]=projbl(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. 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

  used by omgrid_3d.m

  ---- MA 29/5/95 ---------------------------------------------------
```

rect_cont

```
--- [anton.maxent]

       function

      [con_x,con_y,cx,cy]=rect_cont(xmin,xmax,ymin,ymax)

       returns x/y pairs  of a  rectangular contour whose corners
      are given by the input

--- M.Anton --26.1.95--
```

regulo_2d_tcvti

```
----[anton.tcvti]-----

function [Xro,chi2]=regulo_2d_tcvti(Y,dY,T,xpix,ypix,ord,zrs,display,M)

linear regularisation methods of degree 0..3 (NUMERICAL RECIPES)
version ..2d: gradients and laplacian in really 2 dimensions


inputs:         -Y:       line-integrated measurements [nl x n_timesteps]
               -dY:       errors
                -T:       transfer matrix (corresponds to a matrix)
             -xpix:       pixel coordinates
             -ypix:       pixel coordinates
             -ord:        order of linear regularisation (0,1,2)
             -zrs:        add zero chord if 1, optional, default zero
             -display
                          display iteration proceedings if 1, optional
             -M:          default model, optional

output:      -Xro:        inversion result, [npix x n_timesteps]

----MA 31/8/95: some changes concerning M with respect to older versions ---
```

rml

```
--- [anton.tomofour]

function R=rml(m_max,l_max,r)

calculation of Zernike polynomial coefficients, subroutine of granfun.m

input:        r       radial vector (0<=r<=1)
            m_max     maximum m-number
            l_max     maximum l
output:       R       values of Zernike polynomials
                    size [length(r) x (2*(m_max+1)-1)*(l_max+1)]
---- MA 1/95
```

tcvti_chk_mds

```
---- [anton.tcvti]

function status=tcvti_chk_mds(shot)

checks, if there are already Xtomo results written to the MDS results tree
status=1: yes, there are data
       0: no, all is empty

----------- MA 31/8/95 ----------------------------------------------------
```

tcvti_get_mds

```
---- [anton.tcvti]

function [rm,zm,t,X,calf,ct,cpara,ctxt,mtxt,conf]=tcvti_get_mds(shot);

get tcvti - results from the mds- results tree



input: shot                                            [1 x 1]
output:rm              pixel coordinates                 [nr x 1]
       zm              idem                               [nz x 1]
       t               times for the slices inverted         [timesteps x 1]
       X               normalised emissivity (max=1)        [npixels x timesteps]
       calf            calibration factor,                 [1 x 1]
                       if -1: [a.u.], else [W m^-3]
       ct              times where diff. cal. parms
                       had to be chosen                [? x 1]
       cpara           parameters for the calibration         [length(ct) x 6]
       ctxt            string, comment on calibration
       mtxt            string, comment on inversion method
       conf            confidence, -2 ... 2, optional

----------- MA 30/8/95 ---------------------------------------------------
```

tcvti_merge_mds

```
----- [anton.tcvti]

function [mesh_flag,meth_flag,calf_flag,cali_flag]=...
tcvti_merge_mds(shot,rm,zm,t,X,calf,ct,cpara,ctxt,mtxt,conf)

merges new Xtomo data with already existing ones, if possible
if not, see flags. If times are identical, old data are replaced.

out:   mesh_flag       =0 if meshes incompatible
       meth_flag       =0 if different methods were used
       calf_flag       =0 calibrated and uncalibrated data cant be mixed
       mix_flag        =0 if data are just added, 1 if inserted
       cali_flag       =0 inversion res can only be inserted, if the
                       same calibr. parameters are used. if they are
                       just appended, cali_flag=1 by default.
 in:   shot                                         [1 x 1]
       rm        pixel coordinates                  [1 x nr]
       zm        idem                               [1 x nz]
       t         times for the slices inverted         [1 x timesteps]
       X         normalised emissivity (max=1)       [npixels x timesteps]
       calf      calibration factor                 [1 x 1]
       ct        calibration time
       cpara     parameters for the calibration         [1 x 6]
       ctxt      string, comment on calibration
       mtxt      string, comment on inversion method
       conf      confidence, -2 ... 2, optional (def=-2)
 ----------- MA 1/9/95 -----------------------------------------------------
```

tcvti_store_mds

```
----- [anton.tcvti]

function ....
tcvti_store_mds(shot,rm,zm,t,X,calf,ct,cpara,ctxt,mtxt,conf);

store tcvti - results in the mds- results tree.
ACHTUNG: whatever may be there will be overwritten.


                    arguments:
       shot                                          [1 x 1]
       rm             pixel coordinates               [1 x nr]
       zm             idem                             [1 x nz]
       t              times for the slices inverted        [1 x timesteps]
       X              normalised emissivity (max=1)       [npixels x timesteps]
       calf           calibration factor
       cpara          parameters for the calibration        [1 x 6]
       ct             'calibration time'              [1 x 1]
       ctxt           string, comment on calibration
       mtxt           string, comment on inversion method
       conf           confidence, -2 ... 2, optional


      see also TCVTI_MERGE_MDS and TCVTI_CHK_MDS

----------- MA 30/8/95 ----------------------------------------------------
```

tcvxti

```
--- [anton.tcvxti]

 tcvxti.m: scriptfile, launches tcvxti_uifun
           it's possible to peek at all variables
           DANGER: everything's CLEARED and CLOSED if you call this script!
           calls TCVXTI_UIFUN('initialize')

--------- MA 1995
```

tcvxti_getlcfs

```
----- [anton.tcvxti]

      function

      [con_xt,con_yt,c_xt,c_yt,c_times]=tcvxti_getlcfs(shot,times);

      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 e.g. is [npts_contour x length(c_times)], size of
      c_yt is [1 x length(c_times)].

------- MA 1995
```

tcvxti_getpsi

```
----- [anton.tcvxti]

       function

       [psi_mesh,c_times]=tcvxti_getpsi(shot,times,xmesh,ymesh)

       returns matrices of the contours of PSI/PSI_AXIS from LIUQE #shot
       "c_times" contains the LIUQE times which were nearest to the
       values specified in the vector "times". PSI is interpolated on
       the meshgrid "xmesh,ymesh". One timeslice is stored in one column of
       "psi_mesh", to get the values in the right order, you have to do a
       RESHAPE(psi_mesh(:,N),length(YMESH),length(XMESH)).
       N=1...length(c_times).

 ------- MA 1995
```

tcvxti_simulant

```
----- [anton.tcvxti]

simulate a set of x-ray emissivity date and store them.
parameters have to be edited in the script file
named FILENAME without extension,please.

function tcvxti_simulant(filename)
 example: 'filename'='sim_8100_1'

---- MA 25/1/1996
```

tcvxti_setgrid

```
---- [anton.tcvxti]

function

[xmin,xmax,ymin,ymax,xmesh,ymesh]=tcvxti_setgrid(set,nx,arg1,arg2,arg3,arg4);

set up  pixelgrid coordinates for xtomo

inputs        set       specifies the kind of setup, if set equals
                    1: manual setup
                         -> arg1=pcx, the grid center x coordinate
                            arg2=pcy  the grid center y coordinate
                            arg3=wx,  the horizontal width of the grid
                            arg4=wy,  the vert, width. ALL UNITS [CM] !
                    2: use LIUQE for a specified time
                         -> arg1=shot,arg2=time
                                (everything else is ignored)
                    3: use LIUQE for the whole shot
                         -> arg1=shot (everything else is ignored)
                    4: get setup from MDS results tree
                         -> arg1=shot (nx and other args ignored)

output        vectors xmesh, ymesh specifying the CENTER coordinates of
              the pixel grid. xmin ... ymax give the corners of the
              outermost border of the grid

------------ MA 21/11/95 ---------------------------------------------------
```

tcvti_testphip

```
------- [anton.tcvti]

function

tcvti_testphip(xchord,ychord,con_x,con_y,cx,cy,xmin,xmax,ymin,ymax)

utility to plot the (p,phi)-representation (for GRANETZ inversion)
of the chords (xchord,ychord) and the LCFS as well as of a rectangle
specified by xmin ... ymax with respect to the center cx,cy.

uses
               [phi,p]=phip(xchord,ychord,cx,cy);
               [con_x,con_y,cx,cy]=rect_cont(xmin,xmax,ymin,ymax);

---- MA 1995
```

tcvxti_uifun

```
---- [anton.tcvxti]

function tcvxti_uifun(action)

ui interface for x-ray tomography. call preferably via
the script file TCVXTI.M, please. otherwise serious
problems may occur ...

--- MA 1995
```

tmat_standard

```
--- [anton.maxent]

function

[TT,numdet]=tmat_standard(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

input

      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

------------------M.Anton 9.8.94 / 2.12.94
```

t_omgrid

```
---- [anton.public]

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

calculation of t matrix for a rectangular grid using precalculated
matrices of solid angles for all detectors and a grid of 0.5x0.5x1cm^3
matrices are stored in [anton.public.raumwinkel]raumwinkel_#.mat

----- MA 30/5/95 + 23/11/95
```

wml

```
 --- [anton.tomofour]

 function W=wml(m_max,l_max,p,phi,nl,nzcho,nzpts)


 calculation of Granetz' W-matrix

 input:        p,phi       impact parameter and corr. angle of the lines of sight
                           size [nl x 1]
               m_max       maximum m-number
               l_max       maximum l
               nl          number of 'real' lines of sight
               nzcho       number of zero chords added
               nzpts       number of zero points added

 output:        W       W-matrix
                    size [ nl x (2(m_max+1)-1)*(l_max+1)]
---- MA 1/95
```

xrs_spectrum

```
----- [anton.efficiency]

function EDIST=xrs_spectrum(UXS,KEV);

inputs -KEV:   a vector of Energies in keV
       -UXS:   the voltage of the Xray source

output -EDIST: a normalized energy distribution (i.e. integral=1)
               of same size as KEV
               is given only for a restricted set of voltages for
               which the spectra have been measured. xrs_spectrum
               gives a fit to the experimental spectra

-------------------M.Anton--June 1994---------------------------------------
```

xtomo_geometry

```
----[anton.public]

function

[fans,vangle,xchord,ychord,aomega,angfact]=xtomo_geometry(i_detec,fans);

      inputs:

 i_detec:       =2: Xtomo prototype cameras (shot# < 6768)
                =1: Xtomo 9-cameras           (shot# > 682x)

      outputs:

 fans:     camera switch, 1=on,0=off (1x10)
 vangle:   angle between detect. surface normal and pos. x-axis (1x10)
 xchord:   two x-coordinates (2xnl) in [cm] and
 ychord:   two y-coord. for each line (2xnl), they specify start + end points
 aomega:   etendue in mm^2 x steradians
 angfact: angular factors, inverse of relative etendue (throughput) (20x10)

      uses:
             AOMEGA=etendue_n2(b1x,b1y,b1z,b2x,b2y,b2z,z01,z02,X0,cw);
             angular_fact_*.mat , '*'=i_detec


---------------- M.Anton 14/3/95 -------------------------------------------
```

xtomo_calibrate

```
--- [anton.public] ---

function

[corr,eta]=xtomo_calibrate(i_detec,i_dioda,i_spec,fans,funpara,KEV,EDIST)

output       corr         calibration factors, if i_dioda==1:
                          5% correction (up) for 1st and last diode
input
             i_detec      1: Xtomo
                          2: Xtomo Prototypes
             i_dioda      0: L=200,dp+=0.5
                          1: exper. L,dp+
             i_spec       0: same spectral distribution
                          1: simul. spectral dist
             fans         detectors switch
             funpara      [funtype,para1,para2...]
             KEV          photon energies             (if i_spec)
             EDIST        energy distributions      (if i_spec)

---------M.Anton 2/3/95 ---------------------------------------------------
```

xtomo_simcal

```
--- [anton.tcvti] ---

function   [Y_ideal,Y_eta0,Y_eta,etamean,Xinit,KEV,EDIST]=...
xtomo_simcal(i_de,i_di,i_wa,fans,xmesh,ymesh,psi_mesh,funpara);

      output:
             Y_ideal       chord brightness, assuming ideal detectors
             Y_eta0        same, using eta, ignoring real angle of incidence
             Y_eta         line integrated signals, taking all into acc.
             etamean       spectrum averaged efficiency
             Xinit         emissivity distribution as a func of xmesh,ymesn
             KEV,EDIST     energy distribution for every line of sight
      input:
             i_de          1: Xtomo
                           2: Xtomo Prototypes
             i_di          0: L=200mu, dp=0.5mu
                           1: experimental L,dp data
             i_wa          1: only Y_ideal
                           2: Y_ideal & Y_eta0
                           3: Y_ideal & Y_eta0 & Y_eta
             fans          detectors switch
             psi_mesh      flux on meshgrid defined by
             xmesh,ymesh
             funpara = [funtype,para1,para2,...]
             funtype=1: Ne,Te are polynomials of psi_norm
                     2: Ne,Te, from Thomson scattering

---------M.Anton 9/6/95 --------------------------------------------------
```

A Some remarks on inversion methods

A.1 The tomography problem ...

Plasma parameters like temperature, density, and effective charge distribution determine the quantity and the spectral distribution of the emitted radiation. In toroidal magnetic fusion devices like tokamaks, the properties of the plasma radiation in the soft X-ray spectral range are assumed to be constant on surfaces of equal poloidal magnetic flux because of the enhanced transport parallel to these surfaces. The magnetic topology becomes thus accessible via soft X-ray tomography.

A schematic experimental setup for soft X-ray tomography on a tokamak is displayed in figure 7. Several pinhole cameras to observe the soft X-ray emission in a poloidal cross section of the plasma are placed around the torus. Every pinhole camera is equipped with a number of detectors. The "pinhole" itself is a small aperture, usually a slit, to limit the field of view of the detector. The aperture and the sensitive area of the detector define a cone of view, as shown schematically in figure 8. The center of this cone is in the following referred to as "line of sight". A Beryllium foil in front of the aperture usually serves to block off ultraviolet, visible and infrared radiation.

We can define a spectral emissivity $G(\vec{r},\nu)$ which is determined by the plasma radiation itself *and* by the transmission characteristics of the Be foil. The dimension of G is *power per volume and frequency interval*. We assume that the power is radiated isotropically. Let $\Omega_\ell(\vec{r})$ be the solid angle subtended by one of the detectors with the efficiency $\eta_\ell(\nu)$. The total power P_ℓ detected by diode $\#\ell$ ($\ell = 1 \ldots n_\ell$) equals

$$P_\ell = \int d\vec{r} \int d\nu \ \frac{\Omega_\ell(\vec{r})}{4\pi} \cdot G(\vec{r},\nu) \cdot \eta_\ell(\nu) \tag{1}$$

If the field of view of the detector is sufficiently narrow, we may assume that the emissivity does not vary on a surface perpendicular to the line of sight S_ℓ (see figure 8), so that $d\vec{r} \rightarrow A(s) \times ds$, where ds is a line element along the line of sight. This leads to

$$P_\ell \approx \frac{(A\Omega)_\ell}{4\pi} \cdot \int\limits_{S_\ell} ds \int d\nu \ G(\vec{r},\nu)\ \eta_\ell(\nu). \tag{2}$$

where the factor $(A\Omega)_\ell$, the *étendue géometrique* or optical throughput, could be taken outside the integral. This can be explained by the fact that the surface area $A(s)$ increases quadratically with the distance from the detector, while at the same time the solid angle subtended by the same detector decreases quadratically with distance. With the help of (2) we can define the chord brightness f_ℓ as

$$f_\ell = \frac{P_\ell}{(A\Omega)_\ell / 4\pi} \tag{3}$$

The dimension of f is obviously *power per area*.

A further, generally applied approximation is to assume that eventual differences in detector response can be allowed for using a constant calibration factor c_ℓ for every diode, which leads to

$$f_\ell = c_\ell \cdot \int_{S_\ell} ds\; g(\vec{r}) \tag{4}$$

where the emissivity

$$g(\vec{r}) = \int d\nu\; G(\vec{r}, \nu) \tag{5}$$

has been introduced. In the following we shall assume that c_ℓ equals unity for all detectors. (In reality, this is seldom true, see *e.g.* [2]).

The task of X-ray tomography is to reconstruct the two–dimensional distribution of the local emissivity g from a limited number of line integrated measurements f_ℓ. Mathematically, the problem consist of solving the system of integral equations[2]

$$f_\ell = \int_{S_\ell} g\; ds \quad (\ell = 1 \ldots n_\ell) \tag{6}$$

where the integral is along the line of sight (compare figure 8) and n_ℓ is the number of available measurements. This system of equations is always underdetermined, since we would need an infinite number of measurements f_ℓ to be able to determine g exactly. In fusion research, the number of line integrated data is usually limited to the order of some 10^2, which is even farther away from infinity than the 10^5 available in medical tomography.

[2]To be precise: it's a set of inhomogeneous Fredholm equations of the first kind, see *e.g.* [19].

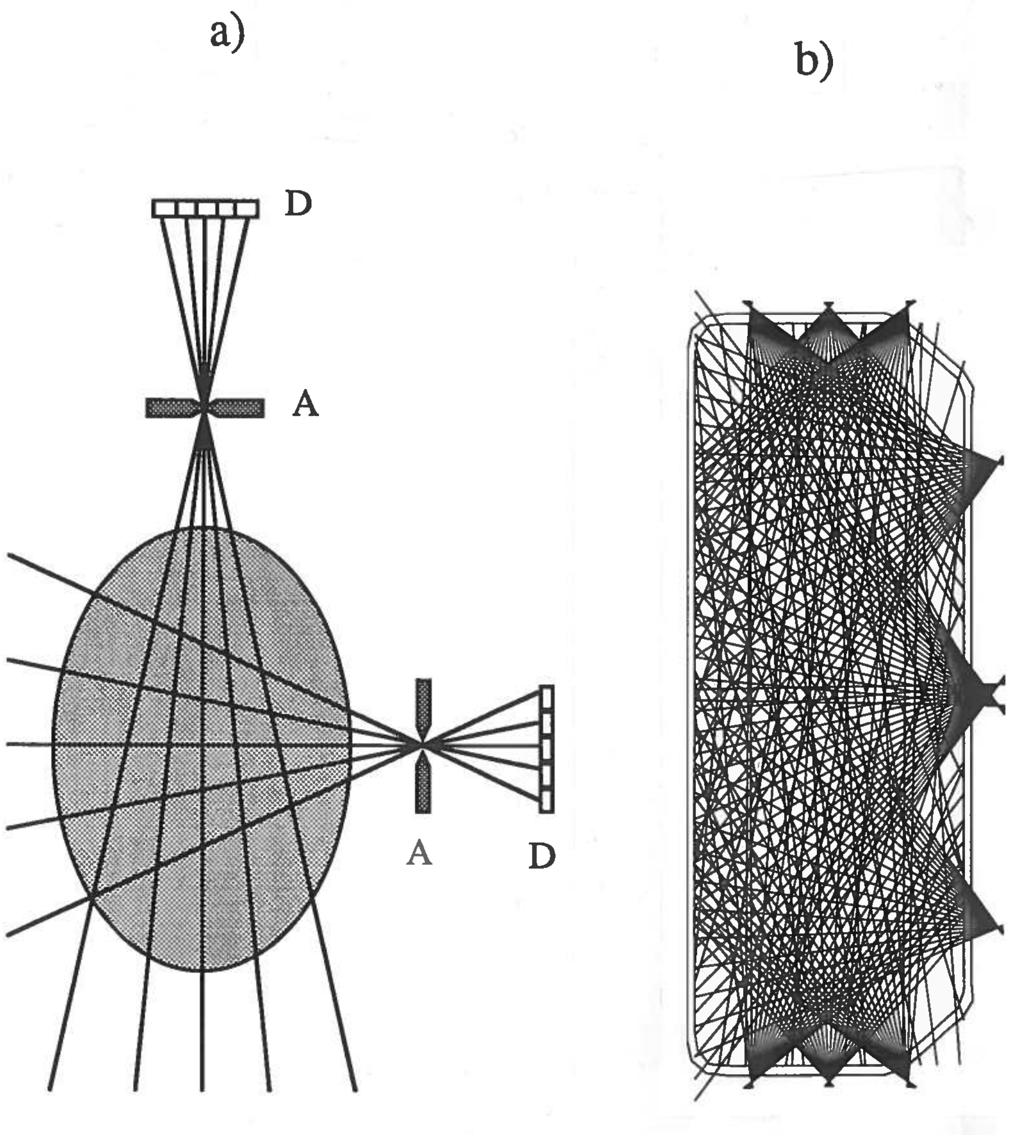


Figure 7: *The tomography problem: a) a schematic setup : radiation from a 2D emissivity distribution g is measured by a set of detectors D. The field of view is assumed to be collimated by apertures A, so as to justify an approximation of the field of view by a line. b) the actual setup of the TCV soft X-ray tomography system*

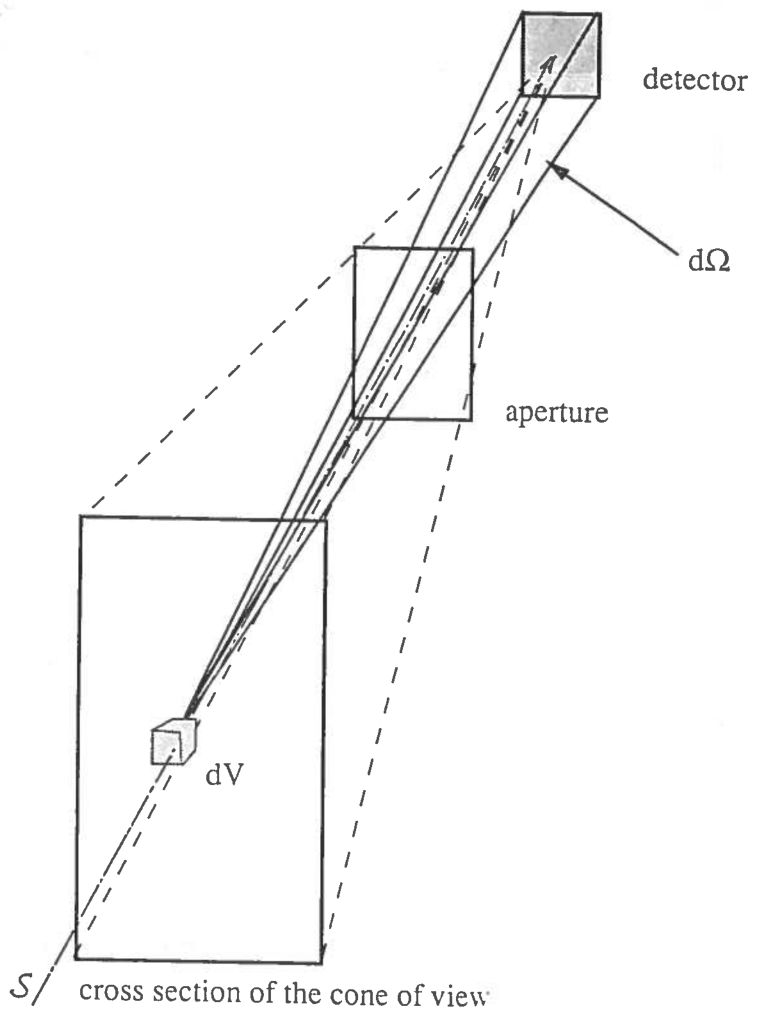


Figure 8: *The field of view defined by the aperture and the sensitive surface area of the detector is usually sufficiently narrow to justify an approximation of the cone by a line. S is the central chord of the field of view, the "line of sight", Ω is the solid angle subtended by the detector as seen from the volume element $d\vec{r}$.*

A.2 ... and some ways to solve it

There are essentially two ways to adress the tomography problem:

- It is possible to reduce the number of degrees of freedom by expanding the emissivity distribution in a set of orthogonal functions. Instead of the distribution g itself, a limited set of parameters, the coefficients of the base functions, have to be determined. If *e.g.* we choose polar coordinates in a plane, a Fourier decomposition for the angular part and a polynomial approximation for the radial part of g can be used. This is done in the *Cormack-Granetz* algorithm [9], which is widely used in fusion research.

- The plane where we want to reconstruct the emissivity distribution is subdivided into quadratic pixels. The size of the pixels has to be sufficiently small to justify the assumption of constant emissivity within one pixel. At the same time, they have to be sufficiently big to obtain a system of equations which *can* be solved.

In the following, we will concentrate on the second possibility.

One advantage of the pixel ansatz is that the system (6) is transformed to a system of algebraic equations in a very natural way. If we have a 2D pixelgrid with n_x horizontal and n_y vertical pixels, we can store the $n_{pixel} = n_x \cdot n_y$ emissivity values g_i as lines of the column vector g. The n_ℓ line integrated data are put into a column vector **f**. We get

$$f_\ell = \sum_{i=1}^{n_{pixel}} T_{\ell i} g_i \quad (\ell = 1 \ldots n_\ell) \tag{7}$$

or simply

$$\mathbf{f} = \mathbf{T} * \mathbf{g} \tag{8}$$

where $*$ denotes usual matrix multiplication. In the simplest approximation, the matrix element $T_{\ell i}$ is equal to the length of the chord $\# \ell$ in pixel $\# i$. The size of **T** is $n_\ell \times n_{pixel}$, the number of lines of sight times the number of pixels.

The most obvious idea to solve (8) is to invert **T**. In most cases this won't work, either because we have less equations than unknowns (*i.e.* the inverse of **T** does not exist) or, even if we have $n_\ell = n_{pixel}$, the matrix **T** might be badly conditioned. We want a smooth, stable and unique solution vector g, which we can not obtain with a simple inversion.

For a start, let us assume that we have *more* line integrated measurements than pixels, *i.e.* $n_\ell > n_{pixel}$. In that case, we would try to

$$minimise \quad \chi^2 \tag{9}$$

with

$$\chi^2 = \sum_\ell \left(\frac{\sum_i T_{\ell i} g_i \; - \; f_\ell}{\sigma_\ell}\right)^2 \tag{10}$$

which is the same as

$$\chi^2 = (\tilde{\mathbf{T}} * \mathbf{g} - \tilde{\mathbf{f}})^T * (\tilde{\mathbf{T}} * \mathbf{g} - \tilde{\mathbf{f}}) \tag{11}$$

The exponent ^{T} denotes transposition. For convenience, we have used the abbreviations $\tilde{T}_{\ell i} = T_{\ell i}/\sigma_\ell$ and $\tilde{f}_\ell = f_\ell/\sigma_\ell$, where σ_ℓ is the standard deviation of f_ℓ.

To minimise χ^2 , we have to derive the normal equations [19] from eqn (11) which read

$$\tilde{\mathbf{T}}^T * \tilde{\mathbf{T}} * \mathbf{g} = \tilde{\mathbf{T}}^T * \tilde{\mathbf{f}} \tag{12}$$

The solution of the set of normal equations then yields a least-squares-fit solution to the tomography problem, see for example [5].

If we reduce the number of line integrated data f_ℓ and keep the number of pixels fixed, we observe that a better χ^2-fit is possible. In the limit of less equations (hence f_ℓ's) than unknown g_i's, we can always achieve $\chi^2 = 0$, because then there are too many degrees of freedom, *i.e.* there is an infinite number of solutions ("overfitting"). To obtain a unique (and sensible) solution, we have to require something in addition to $\chi^2 = min$. The general idea is to look for a minimum of a functional ϕ, which may be written as

$$minimise \quad \phi = \frac{1}{2}\chi^2 + \alpha\,\mathcal{R} \tag{13}$$

where $\mathcal{R}$ is a regularising functional (hence the letter $\mathcal{R}$). The regularisation parameter α is a positive number which determines the weighting between the goodness-of-fit, represented by χ^2, and the requirements imposed on the solution $\mathbf{g}$ by the functional $\mathcal{R}$, *e.g.* the smoothness of the solution. In the limit $\alpha \to 0$, the solution is determined by χ^2 alone as above, in the limit of very large α it is only the smoothing (or whatever we may have required) which determines the solution. Truth must be somewhere in between, so one part of the problem is to find a way to choose the "correct" value of the regularisation parameter, a second part is to find a solution g for a given value of α.

Three different choices of ϕ will be discussed in the following: Linear regularisation, maximum entropy and a method relying on the Fisher information.

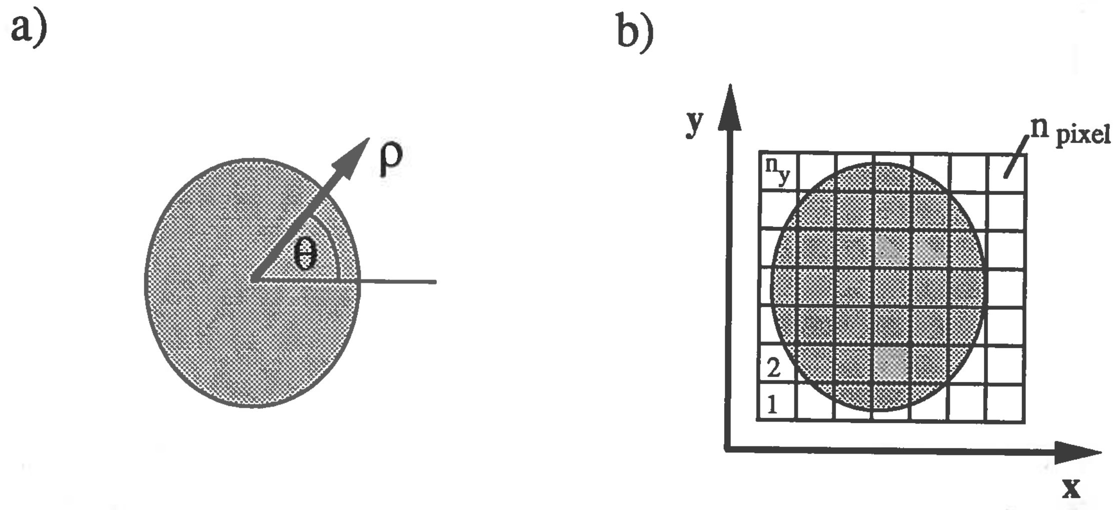


Figure 9: *Two ways to attack the tomography problem: we can a) expand g using a limited set of base functions or b) subdivide the $x-y$ plane into a set of quadratic pixels. The emissivity in each pixel is assumed to be constant. In the simplest approximation, the matrix element $T_{\ell i}$ is the length of the line of sight S_ℓ in pixel # i.*

A.2.1 Linear Regularisation

The first method we want to discuss is the so-called linear regularisation method as detailed in [19]. If we want a smooth solution, the functional $\mathcal{R}$ has to measure the roughness of the solution somehow.

The simplest approach is to require the solution vector $\mathbf{g}$ to have minimum length,

$$\mathcal{R} = ||\mathbf{g}||^2 = \mathbf{g}^T * \mathbf{g} \tag{14}$$

where $||\cdot||$ denotes the usual euclidean vector norm. This is called zeroeth order regularisation.

We skip first order regularization to discuss second order regularisation in some detail: The second order linear regularisation tries to minimise the norm of a vector which contains the values of the second derivative of the solution $\mathbf{g}$,

$$\mathcal{R} = ||\triangle\mathbf{g}||^2 = (\triangle\mathbf{g})^T * (\triangle\mathbf{g}) \tag{15}$$

which means that we look for a solution with minimum curvature. If $\triangle$ denotes a matrix representation (finite differences) of the Laplacian, we can write

$$\mathcal{R} = (\triangle * \mathbf{g})^T * (\triangle * \mathbf{g}) = \mathbf{g}^T * \triangle^T * \triangle * \mathbf{g} \tag{16}$$

With the help of the definition

$$\mathbf{H} = \triangle^T * \triangle \tag{17}$$

we get from (11) and (13)

$$\textit{minimise} \quad \phi = \frac{1}{2}(\tilde{\mathbf{T}} * \mathbf{g} - \tilde{\mathbf{f}})^T * (\tilde{\mathbf{T}} * \mathbf{g} - \tilde{\mathbf{f}}) + \mathbf{g}^T * \mathbf{H} * \mathbf{g} \tag{18}$$

If we set all n_{pixel} partial derivatives $\partial\phi/\partial g_i$ to zero, we get the normal equations

$$(\tilde{\mathbf{T}}^T * \tilde{\mathbf{T}} + \alpha\mathbf{H}) * \mathbf{g} = \tilde{\mathbf{T}}^T * \tilde{\mathbf{f}} \tag{19}$$

which have to be solved for $\mathbf{g}$, *e.g.* by standard methods like LU-decomposition[3]. To determine the "correct" α, you start with [19]

$$\alpha = \mathit{trace}(\tilde{\mathbf{T}}^T * \tilde{\mathbf{T}})/\mathit{trace}(\mathbf{H}) \tag{20}$$

and tune α, until you achieve $\chi^2 \approx n_\ell$, if the measurement errors σ_ℓ are known sufficiently well. For experimental data obtained on TCV, the optimum α is usually found after $2-3$ iterations. In terms of computing time this corresponds to 6-8 seconds on a DECα station.

It has to be stressed that the equation (19) is the *same* for *all* orders of linear regularisation. It's just the matrix $\mathbf{H}$ which has to be modified. For an n'th order regularisation, H contains the (finite difference) matrix form of the n'th derivative of the solution, analogous to (15). For zero order regularisation (see (14)), the matrix $\mathbf{H}$ is simply the unit matrix[4] Higher order algorithms are also possible, one can even think of a mixture of different orders ("solution close to a differential equation", see [19]).

[3] If you use MATLAB, you just employ the '\'- operator.

[4] The pseudoinverse calculated during the solution of (19) for zero order is related to the so-called Moore-Penrose pseudoinverse [23, 1]

A.2.2 Maximum Entropy

There is nothing like *the* Maximum Entropy method, instead there is a whole lot of algorithms exploiting in one way ore another the information entropy of a probability distribution as defined by *Shannon*. The relationship between information theory and statistical physics has been investigated in two excellent publications by *Jaynes* [14, 15], who also references there [14] the "father" of Bayesian statistics, *Sir Harold Jeffreys* [16].

Different degrees of sophistication and depth of philosophical background are possible. A brief introduction is provided by [19], although the *Bayesian* aspect of Maximum Entropy is somewhat turned down by the authors. An abordable survey on the theory and different applications of *Bayesian* MaxEnt is given in a fairly recent book editet by *Buck* and *Mackaulay* [4]. A review of a variety of MaxEnt algorithms has recently been given by *Djafari* [7].

We will start with an algorithm implemented as a MATLAB function by *Peysson* [18].

The entropy S of a probability distribution g'_i is given by

$$S = -\sum_i g'_i \cdot ln(g'_i) \tag{21}$$

with

$$g'_i = \frac{g_i}{\sum_k g_k} \equiv \frac{g_i}{N} \tag{22}$$

where N assures proper normalisation. We use χ^2 as defined above (11) and try to

$$maximise \quad \tilde{\phi} = -\frac{1}{2}\chi^2 + \alpha\ S \tag{23}$$

The maximisation of $-\chi^2$ is clearly equivalent to a minimisation of χ^2. The information entropy S has now taken the place of the regularising functional $\mathcal{R}$, which means our requirement in addition to a reasonable fit to the data is that the entropy of the solution $\mathbf{g}$ attains its maximum. If no experimental data are available, this is the case when all pixels have equal emissivity. In a certain sense, the maximum entropy solution is the least we can expect, it is the most "pessimistic" of all possible solutions.

We can write the expression for the entropy as

$$S = -\mathbf{g}'^T * ln(\mathbf{g}') = -\frac{\mathbf{g}^T}{N} * ln(\frac{\mathbf{g}^T}{N}) \tag{24}$$

where here and in the following functions like ln , exp *etc.* are understood to act *elementwise* on the argument vectors or matrices. The column vector $\mathbf{g}$ contains the pixel emissivities as above. To find an extremal value of $\tilde{\phi}$, we have to set the partial derivatives $\partial\tilde{\phi}/\partial g_i = 0$ for all $i = 1 \ldots n_{pixel}$. The result is

$$\mathbf{g} = N \cdot \exp(-S \ - \ \frac{N}{\alpha}(\tilde{\mathbf{T}}^T * \tilde{\mathbf{T}} * \mathbf{g} + \tilde{\mathbf{T}}^T * \tilde{\mathbf{f}})), \tag{25}$$

which represents an implicit, nonlinear system of equations for the sought–for solution $\mathbf{g}$. Different fixed-point iteration methods to solve (25) exist, four of which are implemented in *Peysson*'s MATLAB routine [18]. The most common one has been described by *Gull and Daniell* [10], modifications of this method to stabilize the convergence have been proposed by *Wu* [27] and *Delsuc* [6]. The "correct" regularisation parameter α is obtained using the criterion

$$\chi^2 \approx n_\ell \tag{26}$$

as for the linear regularisation method (see above). A "typical" tomographic inversion of data obtained with the soft X-ray camera system on TCV will at least take some 30 seconds for about 6-12 iterations to fine tune α. We will not go into further details but rather discuss an algorithm by *von der Linden* [24], which is "really" Bayesian.

It was *Sir Harold Jeffreys* who re-discoverd *Bayes* theorem to approach statistics in a philosophically different way than the usual "frequentist" school of thought, dominated by *Fisher*. The general idea of Bayesian statistics is that we assign probabilities in case of *incomplete knowledge* of the system under consideration, *i.e.* probabilities are rather degrees of plausibility than relative frequencies of occurence as in the usual frequentist point of view. Probability theory thus becomes an extension of logic rather than just a toolbox to handle (seemingly) random data. The original theorem simply relates conditional probabilities P from two events A and B in the following way:

$$P(A|B) = P(A) \cdot \frac{P(B|A)}{P(B)} \tag{27}$$

$P(A|B)$ is the conditional probability of A given that B has occured, $P(A)$ and $P(B)$ are unconditional probabilities (see *e.g.* [19, 11] or any textbook on probability theory).

For so-called inverse problems like tomographic inversion, the theorem can be used in the following way [11, 24]

$$P(\mathbf{g}|\mathbf{f}_{exp}, I) = P(\mathbf{g}|I) \cdot \frac{P(\mathbf{f}_{exp}|\mathbf{g}, I)}{P(\mathbf{f}_{exp}|I)}, \tag{28}$$

where names and meanings of the symbols are listed below:

$\mathbf{f}_{exp}$	experimental, line integrated data (chord brightness)
I	any other available information *a priori*
$\mathbf{g}$	the sought-for emissivity distribution
$P(\mathbf{g}\|I)$	the probability of the solution prior to any experiment
$P(\mathbf{f}_{exp}\|I)$	the probability for the experimental data, given only the prior information I; $P(\mathbf{f}_{exp}\|I)$ usually remains unknown
$P(\mathbf{f}_{exp}\|\mathbf{g}, I)$	the probability for the measured data $\mathbf{f}_{exp}$ if $\mathbf{g}$ and I were known, known as the likelihood function
$P(\mathbf{g}\|\mathbf{f}_{exp}, I)$	the so-called posterior probability for the solution $\mathbf{g}$, given the experimental data $\mathbf{f}_{exp}$ and the *a priori* information I

The general idea is to search for a maximum of the posterior probability $P(\mathbf{g}|\mathbf{f}_{exp}, I)$, *i.e.* to look for the most probable solution in the sense of Bayesian statistics. To be able to do so, we look at the terms on the right-hand side of equation (28). The denominator usually remains unknown and serves only as a normalisation constant. The likelihood function is well-known and can be expressed as

$$P(\mathbf{f}_{exp}|\mathbf{g}, I) \propto exp\left(-\frac{1}{2}\chi^2\right) \tag{29}$$

with χ^2 given by eqn (11), as usual. It can be shown [22] that the most uninformative, unbiased prior is the entropic one, given by

$$P(\mathbf{g}|I) = (\frac{\alpha}{2\pi})^{n_{pixel}/2} \cdot \frac{1}{\sqrt{\prod_i g_i}} \cdot exp(\alpha\ S). \tag{30}$$

Here, the entropy is defined with respect to a default model $\mathbf{m}$ of the emissivity $\mathbf{g}$ following

$$S = \sum_i g_i - m_i - g_i\ ln(\frac{g_i}{m_i}) \tag{31}$$

The entropy S attains its maximum if $\mathbf{g}$ is equal to the default model. If we put equations (29) and (30) together, we see that a maximum of the posterior probability occurs at a maximum of

$$\phi(\alpha, \mathbf{g}) = -\frac{1}{2}\chi^2 + \alpha\ S \tag{32}$$

which looks rather familiar, the positive number α still being the regularisation parameter.

For α fixed, we have to find the solution $\mathbf{g}^*$. Following *von der Linden*, this can be done introducing n_ℓ Lagrangian parameters λ_ℓ, thus reducing the number of unknowns from n_{pixel} to n_ℓ. Maximising (32) under the exact "constraints"

$$\begin{aligned} 0 &= F_\ell - f_\ell(\mathbf{g}) \\ &\text{with} \\ f_\ell(\mathbf{g}) &= \sum_i T_{\ell i}\ g_i \end{aligned} \tag{33}$$

$$\tag{34}$$

is equivalent to maximising

$$\phi_\lambda = \alpha\ S - \frac{1}{2}\sum_\ell(\tilde{f}_\ell^{\,exp} - \tilde{F}_\ell)^2 + \alpha\ \sum_\ell \lambda_\ell(F_\ell - f_\ell(\mathbf{g})) \tag{35}$$

with the $\tilde{}$ designing division by the error σ_ℓ as above. For ϕ_λ to take on an extremal value, all partial derivatives have to be zero:

$$\partial\phi_\lambda/\partial g_i = 0, \quad \partial\phi_\lambda/\partial\lambda_\ell = 0, \quad \text{and} \quad \partial\phi_\lambda/\partial F_\ell = 0 \tag{36}$$

For fixed λ, we get

$$g_i = m_i\ exp(-\sum_\ell \lambda_\ell T_{\ell i}), \tag{37}$$

which automatically assures that the emissivity is nonnegative. The resulting system of n_ℓ implicit equations for the Lagrangian parameters

$$f_\ell^{exp} - \sum_i T_{\ell i}\, m_i\, exp\,(-\sum_{\ell'} \lambda_{\ell'}\, T_{\ell' i}\,) + \alpha\, \sigma_\ell^2\, \lambda_\ell = 0 \tag{38}$$

has a unique solution which can be determined using a Newton-Raphson scheme [19].

Determining the maximum posterior probability (28) as a function of the regularisation parameter is a rather formidable task, because it involves the evaluation of n_{pixel}-dimensional integrals. As demonstrated in [24], the integrals become tractable if we expand the integrand into a Taylor series about the (approximate) solution $\mathbf{g}^*$. In that case the posterior probability can be expressed as

$$P(\mathbf{g}|\mathbf{f}_{exp}, I) = \left(\frac{1}{2\pi}\right)^{n_{pixel}/2} \cdot \left[\, det(\,\delta_{\ell\ell'} + \frac{1}{\alpha}\sum_i \tilde{T}_{\ell i}\; g_i^*\; \tilde{T}_{\ell' i})\,\right]^{-\frac{1}{2}} \cdot exp(\;\phi(\alpha, \mathbf{g}^*)\;) \tag{39}$$

Although quite costly in terms of computer time, this approach to determine α is much more satisfactory than the "historic" criterion (26). It turns out that the χ^2 of a solution $\mathbf{g}$ with maximum posterior probability may be much smaller than n_ℓ. As has been discussed by *von der Linden* [24], the "historic" criterion underestimates the information apported by the data, which is a consequence of the fact that it is simply not adequate. The application of (26) were justified if we had n_ℓ measurements of the *same* quantity, which is clearly not the case in tomography or other fields where inverse methods are used.

Another advantage of MaxEnt becomes very clear from eqn (37): negative emissivities, which do not have a physical sense anyway, are automatically excluded by the algorithm. This is not the case for the linear regularisation method. A disadvantage of MaxEnt is the amount of calculations involved as well as the fact that, at least for tomography, the MaxEnt solution tends to not look "nice" and smooth. This is partly due to the fact that the pixel emissivities are treated as completely indepent, the values g_i are determined without explicitly considering the emissivity of the neighbouring pixels. A smoothing can only be achieved if a so-called "preblur" is added (see *e.g.* [11]).

To do actual calculations, you have to start with some value of α, calculate a solution $\mathbf{g}(\alpha)$, calculate the posterior probability of this solution, try another value of α and so on to finally bracket the maximum of the posterior probability. Even on a DECα station, it usually takes at least one minute to obtain a tomographic inversion of TCV soft X ray data using Bayesian MaxEnt.

A.2.3 Minimum Fisher Information

Reinmuth has shown in his diploma thesis [20] that yet another approach can successfully be used for the assessment of inverse problems, namely the exploit of the so–called Fisher information. The method is based on the *Cramer–Rao* inequality

$$\sigma(g) \geq \frac{1}{I_F} \tag{40}$$

which states that the variance $\sigma(g)$ of a probability distribution g is greater or equal than the Fisher Information I_F of that same distribution. The Fisher Information is defined as

$$I_F = \int \frac{g'(x)^2}{g(x)}\, dx \tag{41}$$

where the prime denotes the derivative with respect to x and we assume that the integral of g equals unity.

$$\int g(x)dx = 1 \tag{42}$$

If we identify the distribution g with the soft X-ray emissivity, the motivation to use the Fisher information as a regularising functional becomes clear. We see immediately from the definition (41), that minimising the Fisher information of g implies a minimisation of the absolute value of the first derivative of g, as is the case for first order linear regularisation (see above). But the denominator of the integrand in eqn (41) *weighs* the smoothing in the sense that for a fixed contribution to the integral, the absolute value of the derivative is allowed to be larger if the value of g itself is also large than in a case where g itself is small. This means that the smoothing is strongest where the values of g are small. For soft-X-ray tomography, this is a very reasonable assumption: small values of g correspond to low emissivity, hence low temperature. The contribution of these areas to the chord brightness f_ℓ are small anyway, so not much information about the low–emissivity regions is contained in the f_ℓ. On the other hand we do not want to smooth eventual structure in the center of the plasma where the emissivity is highest, so the weighted smooth provided by a minimisation of the Fisher information of g seems to suit our purposes very well.

Instead of trying the hard way of using the Fisher information as indicated in *Reinmuth's* diploma thesis [20], we rather suggest to use the central idea for a modified weighted linear regularisation method.

For first order linear regularisation, we would try to minimise (13) with the regularising functional this time given by

$$\mathcal{R} = \|\mathbf{g_x}\|^2 + \|\mathbf{g_y}\|^2 \tag{43}$$

where $\mathbf{g_x}$ and $\mathbf{g_y}$ denote the derivatives with respect to x and y, respectively. If we assume that ∇_x and ∇_y are finite-difference matrix representations of the corresponding differential operators, we can write

$$\mathcal{R} = (\nabla_x\ \mathbf{g})^{\mathbf{T}} * (\nabla_{\mathbf{x}}\ \mathbf{g}) + (\nabla_{\mathbf{y}}\ \mathbf{g})^{\mathbf{T}} * (\nabla_{\mathbf{y}}\ \mathbf{g}) \tag{44}$$

or

$$\mathcal{R} = \mathbf{g}^{\mathbf{T}} * (\nabla_{\mathbf{x}}^{\mathbf{T}} * \nabla_{\mathbf{x}} + \nabla_{\mathbf{y}}^{\mathbf{T}} * \nabla_{\mathbf{y}}) * \mathbf{g}. \tag{45}$$

This means that the matrix $\mathbf{H}$ of eqn (19) is given by

$$\mathbf{H} = \nabla_x^T * \nabla_x + \nabla_y^T * \nabla_y \tag{46}$$

We are free to add a a diagonal weight matrix $\mathbf{W}$, as long as all elements are greater than zero ($W_{ii} > 0$):

$$\mathbf{H} = \nabla_x^T * \mathbf{W} * \nabla_{\mathbf{x}} + \nabla_{\mathbf{y}}^{\mathbf{T}} * \mathbf{W} * \nabla_{\mathbf{y}} \tag{47}$$

If we set W equal to the unit matrix, we have first order linear regularisation in two dimension. To minimise the Fisher information of the distribution $\mathbf{g}$, we can not directly insert $1/g_i$ as a weight, since this would make the method nonlinear. Instead, we propose an iterative process, where we start with $\mathbf{W} \equiv \mathbf{1}$, the unit matrix. We can now solve the normal equations (19) with $\mathbf{H}$ defined by equation (47), and use the solution obtained for $\mathbf{g}$ to determine a new weight matrix $\mathbf{W}$ such that

$$\begin{aligned} W_{ij}^{(1)} &= \delta_{ij} \\ W_{ij}^{(n)} &= \frac{1}{g_i^{(n)}} \cdot \delta_{ij}, \quad g_i^{(n)} > 0 \quad i = 1 \ldots n_{pixel} \\ &\text{and} \\ W_{ij}^{(n)} &= W_{max} \cdot \delta_{ij} > 0, \quad g_i^{(n)} \leq 0 \end{aligned} \tag{48}$$

where the subscript (n) denotes the solution of the n-th iteration. This procedure can be continued until the change in the elements of the weight matrix is less than a certain limit. We have thus introduced a regularisation which remains linear, but the solution of which should have minimum Fisher information. We can write this as

$$(\tilde{\mathbf{T}}^T * \tilde{\mathbf{T}} + \alpha \mathbf{H}^{(n)}) * \mathbf{g}^{(n+1)} = \tilde{\mathbf{T}}^T * \tilde{\mathbf{f}} \tag{49}$$

where $\mathbf{H}^{(n)}$ is defined by (47) and (48) and $\mathbf{g}^{(n+1)}$ is the new solution. It turns out that, although 4-5 iterations are needed to have stable $\mathbf{W}$-matrix elements, the solution remains almost unchanged after the first iteration. *Sagbo* has shown in his TP report that this method provides the best compromise concerning computational time, precision and "beauty" of the solution compared to all other methods included in the `tcvxti`-package. A very nice, smooth and precise solution is obtained after approximately 15-18 seconds CPU on ELTCA1.

B How to calculate the T-matrix

There are just historical reasons for calling the T-matrix 'T-matrix'. which is a complicated way to state that there aren't any. Anyhow, the meaning of the T-matrix should be clear by now (I hope you read appendix A ...): the element $T_{\ell i}$ of the matrix **T** represents the contribution of pixel # i to the chord brightness f_ℓ. The dimension of **T** is *length.*

The simplest approximation is to set $T_{\ell i}$ equal to the length of the line of sight ℓ in pixel i, since we assume that the emissivity is a constant inside every pixel. I will not comment on *how* to calculate the lengths of the chords in every pixel. This is done (rather quickly) by the routine `tmat_standard.m`. Although it's not too obvious how it's done, it should be comprehensive. There is just a small problem: there are two contradictory requirements for this approximation to be valid!

- the pixels must be small to justify the assumption that the emissivity is constant within every pixel
- the pixels must be big enough to avoid problems with the assumption of *lines* of sight, *i.e.* in principle the whole of the cone segment should be within the same pixel

I will explain the second item a bit further: We saw in appendix B, figure 8 how the volumic integral can be transformed to a line integral under the assumption that the emissivity does not vary on a surface perpendicular to the line of sight. However, we have to keep in mind that the lines of sight are rather cones with a certain spacial extent. If, for example, a line of sight is placed just near and parallel to the border of one pixel, the simple approximation of T as length segment of the chord will produce wrong results, since all the emissivity is attributed to one pixel instead of an approximately equal distribution between two pixels, which would be much more realistic. This is illustrated in figure 10. Another aspect of the same problem is that the size of the cone (the field of view) may be much bigger than a pixel. Consider for example the field of view of one of the cameras on top of the vessel and a pixel on the bottom (compare appendix B, figure 7). To overcome this problem, a rather labourious way to calculate "effective chord lengths" has been taken, which can be sketched as follows:

- define a 3D rectangular grid which covers the whole poloidal cross section of the vessel and a toroidal extent of $40cm$. The size of each cell is $0.5 \times 0.5 \times 0.5cm^3$.
- calculate for every cell the solid angles subtended by all detectors
- integrate the toroidal direction, taking the curvature of the vessel into account, to obtain a $0.5cm \times 0.5cm$ 2D mesh of integrated solid angles. Store these grids seperately for all detectors as `raumwinkel_###.mat`–files
- calculate the T-matrix for every meshgrid in the following way:
  - load the `raumwinkel_###.mat` file for the detector under consideration
  - for all pixels, sum up the solid angles for all cells which are inside a pixel and multiply by the surface area of the cell
  - divide by the *étendue géometrique* of the corresponding detector to obtain the "effective length"
  - repeat this procedure for all detectors

The first part of this work, the calculation of the 2D solid angle grids, is done by the routines `omgrid_main.m`, `omgrid_3d.m` and `projbl.m`. The main program, `omgrid_main.m`, should preferably be launched as a batch job. A calculation for the whole set of detectors may take several hours on `ELTCA1`.

The second part can be done using the matlab function `t_omgrid`, which can be used as a subroutine of `tcvxti_uifun`. However, you will not want to do it too often, since a calculation of the T-matrix as described above takes several minutes (the simple method takes some seconds) on `ELTCA1`. But if you calculate both matrices (for reasons of curiosity, for example) and compare them, *e.g.* by doing a pseudocolor plot, the difference is striking enough: the jumps and bumps seen on the simple approximation of the T-matrix are all smoothed out. Once the systematic errors on the Xtomo signals have disappeared (due to the not only new but hopefully better IRD detectors), it will be worthwile to use the "hard" way ....

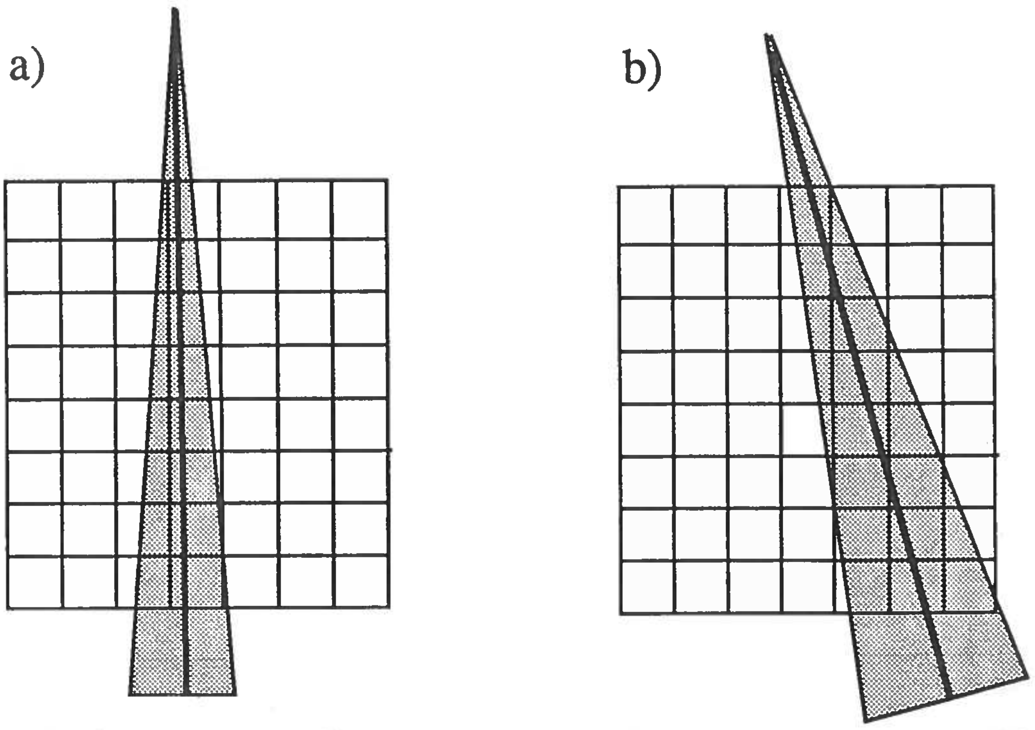


Figure 10: *Some cases where the approximation $T_{\ell i}$ = length of line ℓ in pixel i breaks down*

C Helpful Publications

References

[1] *ABBISS, J.B., ALLEN, J.C., BOCKER, R.P. and WHITEHOUSE, H.J.:* in SPIE Vol. **1767** Inverse Problems in Scattering and Imaging, p93 (1992)

[2] *ANTON, M, DUTCH, M J and WEISEN, H:* Rev. Sci. Instrum. **66** 7 p3762 (July 1995)

[3] *ANTON, M, DUTCH, M J and WEISEN, H:* 22nd EPS conference on CONTROLLED FUSION AND PLASMA PHYSICS, Bournemouth UK, 3rd-7th July 1995, Contributed Papers, Volume19C Part II p389 (1995)

[4] *BUCK, B. and MACKAULAY, V. A. (eds.):* "Maximum Entropy in Action" Oxford University Press (1991)

[5] *DECOSTE, R.:* Rev. Sci. Instrum. **56** 5 p806 (1985)

[6] *DELSUC, M.A.:* in "Maximum Entropy and Bayesian Methods", Ed. J.Skilling, Kluwer Academ. Publ. p285 (1989)

[7] *DJAFARI, A.M.:* Traitement du Signal **11** 2 p87 (1994)

[8] *DUDOK DE WIT, T., PECQUET, A.-L., VALLET, J.C. and LIMA, R.:* Phys. Plasmas **1** (10) p3288 (1994)

[9] *GRANETZ, R. S. and SMEULDERS, P.:* Nuclear Fusion **28** 3 (1989)

[10] *GULL S.F. and DANIELL, G.J.:* Nature **272** p686 (1978)

[11] *GULL, S.:* in "Maximum Entropy and Bayesian Methods", Ed. J.Skilling, Kluwer Academ. Publ. p53 (1989)

[12] *HOFMANN, F. and TONETTI, G.:* Nuclear Fusion **28** p1871 (1988)

[13] *HOLLAND, A. and NAVRATIL, G. A.:* Rev. Sci. Instrum. **57** 8, p1557 (August 1986)

[14] *JAYNES, E.T.:*
Phys. Rev. **106** 4 p620 (1957)

[15] *JAYNES, E.T.:*
Phys. Rev. **108** 2 p171 (1957)

[16] *JEFFREYS, H.:*
"Theory of probability", Oxford University Press (1939)

[17] *LLOBET, X.:* "L'acquisition et la gestion des données du Tokamak TCV", in "Flash informatique", SIC/EPFL Fl 3 (1995)

[18] *PEYSSON, Y.* private communication

[19] *PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T. and FLANNERY, B. P.:*
"Numerical Recipes in FORTRAN: The Art of Scientific Computing", Second Edition, Cambridge University Press (1992)

[20] *REINMUTH, J.:*
Diplomarbeit, TU München / IPP Garching (1994)

[21] *SAKURAI, J.J.:*
"Modern Quantum Mechanics", Addison Wesley Publishing Co. (1985)

[22] *SKILLING, J.:* in
"Maximum Entropy and Bayesian Methods", Ed. J.Skilling, Kluwer Academ. Publ. p45 (1989)

[23] *STERITI, R. and FIDDY, M.A.* in
SPIE Vol. **1767** Inverse Problems in Scattering and Imaging, p112 (1992)

[24] *VON DER LINDEN, W.:*
Appl. Phys. A **60** p155 (1994)

[25] *VON DER LINDEN, W.:*
private communication (1994)

[26] *VON DER LINDEN, W.:*
private communication (1995)

[27] *WU, N.L.:*
Astron. Astrophys. **139** p555 (1984)