CNRS
Centre National de la Recherche Scientifique

INFN
Istituto Nazionale di Fisica Nucleare

MIST
M*odal* **I***nterferometer* **S***imulation* **T***ool*
User Manual

version 2010-05-06

VIR-0321A-10

Gabriele Vajente
INFN sezione di Pisa and Pisa University

Issue: 1

Date: May 6, 2010

VIRGO * A joint CNRS-INFN Project
Via E. Amaldi, I-56021 S. Stefano a Macerata - Cascina (Pisa)
Secretariat: Telephone (39) 050 752 521 * FAX (39) 050 752 550 * Email W3@virgo.infn.it

Contents

1 Introduction

MIST is a Hermite-Gauss simulation software specialized to the simulation of dual-recycled Fabry-Perot interferometers, like Advanced Virgo. The software is developed using MATLAB code. It consists in a collection of functions that can be used to perform several kind of computations. For details on the simulation principles please refer to [1].

1.1 What MIST can do

It is possible to simulate single Fabry-Perot arms, power recycled interferometer (just putting to zero the signal recycling mirror reflectivity) and dual recycling interferometer. MIST can compute:

- field amplitudes in all Hermite-Gauss modes at few points: inside PRC (field at PRM propagating toward BS), inside SRC (field impinging on SRM), at dark port, inside the two arms (fields impinging on end mirrors), field reflected by the interferometer;
- transfer function from differential motion of the end mirrors to dark port,
- transfer function from input field modulation to any of the above mentioned ports.

The kind of defects that can be simulated are:

- mismatched radii of curvature of arm cavity and recycling mirrors;
- astigmatism in NDRC configuration;
- thermally induced lens in input mirror substrate;
- misalignment of arm cavity and recycling mirrors;
- real mirror maps can be used (not yet well tested).

In addition the five longitudinal degrees of freedom (DARM, CARM, MICH, PRCL, SRCL) can be independently tuned.

1.2 What MIST can't do

There are of course many things, but the main limitations at the moment are:

- Transfer functions from auxiliary degrees of freedom can not be computed;
- Transfer functions from DARM to other ports than DF can not be computed;
- Radiation pressure effects are neglected;

1.3 How to use this manual

Please consider that MIST is still in a preliminary version and still ongoing heavy development. Please report any bug to the author at gabriele.vajente@ego-gw.it

Section 2 describes some examples that can be found in the `examples` folder. They are useful to understand the logic of setting up the simulation parameters and running a simulation.

Section 3 collects detailed explanation of the purpose and functioning of the main MATLAB functions used in MIST.

1.4 How to install and use MIST

The latest version of MIST code can be found at the following link:

`https://workarea.ego-gw.it/ego2/virgo/Locking/mist/`

To use it it is enough to add the `src` directory to the MATLAB path list, using for example the command

```
addpath /home/user/MIST/src
```

that can alos be added to the MATLAB `startup.m` file.

2 Commented examples

These examples were modified version of codes used for the main Advanced Virgo simulations and are meant to give the first time user a reasonable starting point. All examples can be found in the `examples` folder inside MIST main one.

2.1 Power recycling MSRC interferometer

The code is in the file `examples/doublecav_msrc.m`. In the following pieces of the code are reported on commented.

```
% Parameters used for mirror matrix computations %%%%%%%%%%%%%%%%%%%%%%%%%%
global rmir npt thr
rmir = 0.6;          % mirror radius in meters
npt = 151;           % number of points for numerical integration on a grid npt x npt
thr = 0;             % threshold to kill small matrix elements (safer to stay at 0!)
```

These are global parameters used in all functions, which defines the mirror radius and the number of points used in numerical integrations to compute the coupling matrix of mismatched mirrors and lenses. The last parameter is a threshold that can be set to put to zero small elements in the coupling matrices. It is safe to maintain it to zero.

```
% Modes parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global mm M
mm = list_even_modes(8);    % use only even modes for mismatch
M = length(mm);
```

Here one can define the modes to be used in the simulation. The function `list_even_modes` create a list of all Hermite-Gauss modes of even order, which are the only one created with thermal lensing and mismatching. A similar function `list_modes` returns instead all modes, odd end even.

```
% Arm cavity parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

itf.RCi = 1416;  % input mirror ROC
itf.RCe = 1646;  % end mirror ROC
itf.L   = 3000;  % length

itf.ti = sqrt(0.0069);                % input mirror  amplitude transmission
itf.ri = sqrt(0.9931);                % input mirror amplitude reflection
itf.li = 0;                           % input mirror losses

itf.te = sqrt(75e-6);                 % end mirror amplitude transmission
itf.le = 0;                           % end mirror losses
itf.re = sqrt(1 - itf.le - itf.te^2); % end mirror amplitude reflection

% compute the arm cavity mode for latter computations
cav = cavity_mode(itf.RCi, itf.RCe, itf.L);

```

```
% compute finesse (just for cross-check, not used in computations)
itf.Finesse = pi * sqrt(itf.ri*itf.re) / (1 - itf.ri * itf.re);
```

Here the parameters for the arm cavities are defined. These are the ideal, not aberrated parameters, which are used to compute the correct resonant Gaussian beam parameter which will be used to express fields at all points inside the ITF. The function `cavity_mode` returns a MATLAB structure containing several fields which describes all parameters of the cavity (Gouy phase, spot sizes on mirrors, waist size and position, etc.).

Note that all simulation parameters are embedded in a MATLAB structure called `itf` which will be the input to all simulation functions.

```
% Recycling cavities parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

itf.tp = sqrt(0.04);                       % power recycling mirror amplitude transmission
itf.lp = 0;                                % power recycling mirror losses
itf.rp = sqrt(1 - itf.lp - itf.tp^2);      % power recycling mirror amplitude reflection

itf.ts = 1;                        % signal recycling mirror amplitude transmission
itf.ls = 0;                        % signal recycling mirror losses
itf.rs = 0;                        % signal recycling mirror amplitude reflection

itf.Lprc = 11.952;                 % power recycling cavity length
itf.Lsrc = 11.0;                   % signal recycling cavity length
itf.Schnupp = 0.0;                 % Schnupp asymmetry

itf.fmod1 = 6270659;               % modulation frequency
itf.m = 0.3;
```

These commands define the parameters of power and signal recycling cavities. In this case the signal recycling transmission is set to 1 and the reflection to 0. In this way no signal recycling cavity is simulated. The Schnupp asymmetry is also put to zero, in order to simulate a coupled double cavity, without losses through dark fringe even for sidebands.

```
% compute the optimal curvature of recycling mirrors
Lp = itf.Lprc + cav.L1;
Ls = itf.Lsrc + cav.L1;
itf.RCp = Lp + cav.z0^2 / Lp;
itf.RCs = Ls + cav.z0^2 / Ls;
% compute beam size on recycling mirrors
itf.wp = cav.w0 * sqrt(1 + (Lp/cav.z0)^2);
itf.ws = cav.w0 * sqrt(1 + (Ls/cav.z0)^2);
% compute Gouy phase in recycling cavity propagation
itf.phig_p = atan(Lp/cav.z0) - atan(cav.L1/cav.z0);
itf.phig_s = atan(Ls/cav.z0) - atan(cav.L1/cav.z0);
```

These commands are used to compute the recycling cavity parameters, namely the matched radius of curvature, the spot size on recycling mirrors and the Gouy phase in propagation inside the cavities. Note that, as explained in [1], the lens made by the input mirror substrate is not considered.

```
% Initial length tunings %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
itf.phi_src = 0;                   % signal recycling cavity tuning in radian
itf.DARM = 0;                      % microscopic tunings in meters of the main
itf.CARM = 0;                      % degrees of freedom
itf.PRCL = 0;
itf.MICH = 0;

% Aberrations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
itf.aRCin = itf.RCi;   % radii of curvature
itf.aRCen = itf.RCe;
itf.aRCiw = itf.RCi;   % radii of curvature
itf.aRCew = itf.RCe;
itf.aRCp = itf.RCp;
itf.aRCs = itf.RCs;

itf.fNI = 100e10;      % focal length of input mirror thermal lenses
itf.fWI = 100e10;

% end of parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Here the initial microscopic tunings of longitudinal degrees of freedom are defined. The simulation works in such a way that for a perfect interferometer, without mismatching or lensings, the zero tuning correspond to the nominal working point of the ITF: carrier resonant in arm cavities and PRC and destructive interference at dark port. The second part of parameters defines the aberrated radius of curvature of all mirrors and the focal length of the input mirror thermal lenses. At this point all simulation parameters are defined.

```
% Build arm cavities (with aberrations if defined)
itf.ncav = build_cavity(itf.RCi, itf.RCe, itf.L, itf.aRCin, ...
                        itf.aRCen, itf.ri, itf.re, itf.li, itf.le, itf.fNI, mm);
itf.wcav = build_cavity(itf.RCi, itf.RCe, itf.L, itf.aRCiw, ...
                        itf.aRCew, itf.ri, itf.re, itf.li, itf.le, itf.fWI, mm);

% build recycling mirrors (with aberrations if defined)
itf.Rp = itf.rp * mirror_reflection(itf.aRCp, itf.RCp, itf.wp, mm, rmir, npt, thr);
itf.Rs = 0;
itf.Tp = itf.tp * eye(length(mm));
itf.Ts = eye(length(mm));
```

These functions computes the arm cavity and recycling cavity mirrors coupling matrices, considering the aberrations defined above. Here again the signal recycling mirror is removed.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% compute all thermal lenses for different powers
tic;
NPT = 300;
pabs = linspace(1e-6, 0.3, NPT);
f = 2700 ./ pabs;
L = lenses(f, itf.ncav.w1, list_even_modes(8), rmir, npt, thr);
el = toc;
disp(['Elapsed time: ' num2str(el) ' s'])

% rescale lenses to maintain 00 resonant
for i=1:NPT
    LL(:,:,i) = squeeze(L(:,:,i)) / exp(1j * angle(L(1,1,i)));
end
```

This part of the code defines a list of thermal lens focal lengths and compute for each of them the corresponding coupling matrix. The function `lenses` returns a list of coupling matrices computed for all the focal lengths in the list. This function works for a general list of modes and is not well optimized, so it is pretty slow. More optimized function exits, please refer to section 3. Each matrix is rescaled with the phase of the coupling coefficient of the TEM_{00} mode. In this way the fundamental mode resonance position is not shifted by the lens.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% NO LOCKING ALGORITHM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% compute fields
tic;
Crc0 = zeros(M,M,NPT);
Csc0 = zeros(M,M,NPT);
Cdf0 = zeros(M,M,NPT);
Cn0 = zeros(M,M,NPT);
Cw0  = zeros(M,M,NPT);

Urc0 = zeros(M,M,NPT);
Usc0 = zeros(M,M,NPT);
Udf0 = zeros(M,M,NPT);
Un0  = zeros(M,M,NPT);
Uw0  = zeros(M,M,NPT);

Lrc0 = zeros(M,M,NPT);
Lsc0 = zeros(M,M,NPT);
Ldf0 = zeros(M,M,NPT);
Ln0  = zeros(M,M,NPT);
Lw0  = zeros(M,M,NPT);

itf.DARM = 0;
itf.CARM = 0;
itf.MICH = 0;
itf.PRCL = 0;

% loop over all lenses
h = waitbar(0, 'Computing static fields...');
for i=1:NPT
    waitbar(i/NPT, h, ['Computing static fields (', num2str(floor(100*i/NPT)), '%)']);

    % update ITF input thermal lenses
    itf.ncav.Lens = squeeze(LL(:,:,i));
    itf.wcav.Lens = squeeze(LL(:,:,i));

    % compute fields (PRM, SRM, DF, North, West)
    [Crc0(:,:,i), Csc0(:,:,i), Cdf0(:,:,i), Cn0(:,:,i), Cw0(:,:,i)] = ...
                                        dualrecycled_fields(itf, 0);
    [Urc0(:,:,i), Usc0(:,:,i), Udf0(:,:,i), Un0(:,:,i), Uw0(:,:,i)] = ...
                                        dualrecycled_fields(itf, +itf.fmod1);
    [Lrc0(:,:,i), Lsc0(:,:,i), Ldf0(:,:,i), Ln0(:,:,i), Lw0(:,:,i)] = ...
                                        dualrecycled_fields(itf, -itf.fmod1);
end
close(h);

% compute total DC fields
dccar0 = dc(squeeze(Crc0(:,1,:)), 1);
dcusb0 = dc(squeeze(Urc0(:,1,:)), 1);
dclsb0 = dc(squeeze(Lrc0(:,1,:)), 1);

```

```
54 % compute demodulated signals
55 [p0,q0] = ac(squeeze(Crc0(:,1,:)), squeeze(Urc0(:,1,:)), squeeze(Lrc0(:,1,:)), itf.m, 0);
56
57 el = toc;
58 disp(['Elapsed time: ' num2str(el) ' s'])
```

This section is the core of the simulation. The first commands, up to line 23, just pre-allocate MATLAB variables to store the results. Lines between 30 and 44 consist in a loop over all focal lengths. For each one the two input mirror cavities are set to the previously computed matrices (lines 36 and 37). Then static fields inside the interferometer are computed, for the carrier (line 40) and for the sidebands (lines 42 and 44). The function `dual_recycled_fields` accepts as input an interferometer structure and the field frequency and it returns field amplitudes at several ports. To be more precise the output is a set of $M \times M$ matrices, each one describing the full interferometer coupling matrix. For example `Crc0(:,1,n)` returns the fields coefficient for all modes inside the power recycling cavity when the interferometer input is a TEM_{00} mode for the nth lensing. This means that `Crc0(1,1,n)` is the TEM_{00} component, while `Crc0(2,1,n)` is the TEM_{02} component and so on.

Lines from 47 to 49 used the function `dc` which computes to total power at a given port, summing up all modes. Similarly the function `ac` on line 52 computes demodulated signals starting from both carrier and sideband fields.

```
% some plots
semilogy(pabs, abs(besselj(0,itf.m) * squeeze(Crc0(1,1,:))).^2)
hold all
semilogy(pabs, abs(besselj(1,itf.m) * squeeze(Urc0(1,1,:))).^2)
semilogy(pabs, abs(besselj(1,itf.m) * squeeze(Lrc0(1,1,:))).^2)
xlabel('Input mirror absorbed power [W]')
ylabel('PRC powers [W]')
grid
legend('CAR','USB','LSB')

figure()
plot(pabs, 2*(abs(squeeze(Cn0(1,1,:))) ./ abs(squeeze(Crc0(1,1,:)))).^2)
xlabel('Input mirror absorbed power [W]')
ylabel('FP recycling gain (00 mode)')
grid

figure()
semilogy(pabs, abs(squeeze(Crc0(1,1,:)).^2))
hold all
semilogy(pabs, abs(squeeze(Urc0(1,1,:)).^2))
semilogy(pabs, abs(squeeze(Lrc0(1,1,:)).^2))
xlabel('Input mirror absorbed power [W]')
ylabel('PRC recycling gains')
grid
legend('CAR','USB','LSB')

figure()
plot(pabs, abs(squeeze(Crc0(1,1,:)).^2), 'r--')
hold all
plot(pabs, abs(squeeze(Urc0(1,1,:)).^2), 'g--')
plot(pabs, abs(squeeze(Lrc0(1,1,:)).^2), 'b--')
plot(pabs, dccar0, 'r')
plot(pabs, dcusb0, 'g')
plot(pabs, dclsb0, 'b')
xlabel('Input mirror absorbed power [W]', 'fontsize', 15)
```

```
ylabel('PRC recycling gains', 'fontsize', 15)
grid
legend('CAR','USB','LSB')
title('Carrier TEM00 always resonant inside PRC and arms', 'fontsize', 15)
```

This part of the code is just devoted to create some plots to show the simulation result.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% compute fields, trying PRCL and CARM lock
tic;
Crc = zeros(M,M,NPT);
Csc = zeros(M,M,NPT);
Cdf = zeros(M,M,NPT);
Cn  = zeros(M,M,NPT);
Cw  = zeros(M,M,NPT);

Urc = zeros(M,M,NPT);
Usc = zeros(M,M,NPT);
Udf = zeros(M,M,NPT);
Un  = zeros(M,M,NPT);
Uw  = zeros(M,M,NPT);

Lrc = zeros(M,M,NPT);
Lsc = zeros(M,M,NPT);
Ldf = zeros(M,M,NPT);
Ln  = zeros(M,M,NPT);
Lw  = zeros(M,M,NPT);

dccar = zeros(1,NPT);
dcusb = zeros(1,NPT);
dclsb = zeros(1,NPT);

Z = zeros(2, NPT);

% loop over all lenses
h = waitbar(0, 'Computing static fields...');
for i=1:NPT
    waitbar(i/NPT, h, ['Computing static fields (', num2str(floor(100*i/NPT)), '%)']);

    % update ITF input thermal lenses
    itf.ncav.Lens = squeeze(LL(:,:,i));
    itf.wcav.Lens = squeeze(LL(:,:,i));

    % use last locking point as a starting point
    if i>1
        Z(:,i) = Z(:,i-1);
    end

    % maximize sb in PRC
    Z(1,i) = fminsearch(@(x) -maximize_sb(itf, x, Z(2,i)), Z(1,i));
    % then maximize car in arm
```

```
48      Z(2,i) = fminsearch(@(x) -maximize_car(itf, Z(1,i), x), Z(2,i));
49
50      itf.PRCL = 1e-12 * Z(1,i);
51      itf.CARM = 1e-12 * Z(2,i);
52
53      % compute fields
54      [Crc(:,:,i), Csc(:,:,i), Cdf(:,:,i), Cn(:,:,i), Cw(:,:,i)] = ...
55                                          dualrecycled_fields(itf, 0);
56      [Urc(:,:,i), Usc(:,:,i), Udf(:,:,i), Un(:,:,i), Uw(:,:,i)] = ...
57                                          dualrecycled_fields(itf, +itf.fmod);
58      [Lrc(:,:,i), Lsc(:,:,i), Ldf(:,:,i), Ln(:,:,i), Lw(:,:,i)] = ...
59                                          dualrecycled_fields(itf, -itf.fmod);
60
61      % compute total DC powers
62      dccar(i) = dc(squeeze(Crc(:,1,i)), 1);
63      dcusb(i) = dc(squeeze(Urc(:,1,i)), 1);
64      dclsb(i) = dc(squeeze(Lrc(:,1,i)), 1);
65
66      fprintf(2, 'PRCL = %g CARM = %g \n', itf.PRCL, itf.CARM);
67      fprintf(2, '     PRC car = %g ARM car = %g  PRC USB = %g PRC LSB = %g\n', ...
68                       dccar(i), abs(Cn(1,1,i))^2, dcusb(i), dclsb(i));
69
70  end
71  close(h);
72  el = toc;
73  disp(['Elapsed time: ' num2str(el) ' s'])
```

This part of the code rerun again the simulation looping over all lenses. The difference is that for each focal length a pseudo-locking algorithm is used to find the best operating point. Starting from the operating point found in the previous iteration step (if any) the PRCL position is tuned to maximize the sideband inside PRC and after the CARM position is tuned to recover high carrier power inside the long arms. In details, lines between 40 and 43 set the initial operating point to the one obtained in the previous iteration. Then line 46 search for the point where the sideband power inside PRC is maximum. Indeed the function `maximize_sb` returns the sideband amplitude inside PRC given the PRCL and CARM tunings. After this, line 48 uses the newly found PRCL tuning and searches for the CARM tuning which maximizes the carrier amplitude inside the long arms. Again the function `maximize_car` computes the carrier amplitude inside the north arm as a function of CARM and PRCL tunings.

The rest of the file is not reported here, since it consists only of code devoted to produce plots similar to the one already reported above.

2.2 Power recycling NDRC interferometer

The simulation of NDRC power recycling interferometer can be found in the code file `examples/doublecav_ndrc.m`. Most of the code is exactly equal to the MSRC case, except in the definition of the power recycling cavity. The fact that the cavity is non-degenerate is simulated without introducing the full telescope, but simply forcing the PRC Gouy phase to he desired value:

```
% compute Gouy phase in recycling cavity propagation
%itf.phig_p = atan(Lp/cav.z0) - atan(cav.L1/cav.z0);
itf.phig_s = atan(Ls/cav.z0) - atan(cav.L1/cav.z0);

% to impose Non Degenerate Recycling cavities, set manually Gouy phase
itf.phig_p = 19.3 / 180 * pi;
```

2.3 Full simulation of astigmatic dual-recycled NDRC

A full simulation of a dual-recycled interferometer with NDRC (including astigmatism) and independent thermal lensings can be found in the file `examples/dualrec_ndrc.m`. The first part which defines the arm cavity is equal as before:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%% CONFIGURATION PART %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Parameters used for mirror matrix computations %%%%%%%%%%%%%%%%%%%%%%%%%%
global rmir npt thr
rmir = 0.6;         % mirror radius in meters
npt = 151;          % number of points for numerical integration on a grid npt x npt
thr = 0;            % threshold to kill small matrix elements (safer to stay at 0!)
c = 299792458;

% Modes parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global mm M
mm = list_even_modes(8);    % use only even modes for mismatch
M = length(mm);

% Arm cavity parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

itf.RCi = 1416;  % input mirror ROC
itf.RCe = 1646;  % end mirror ROC
itf.L   = 3000;  % length

itf.ti = sqrt(0.0069);                  % input mirror  amplitude transmission
itf.ri = sqrt(0.9931);                  % input mirror amplitude reflection
itf.li = 0;                             % input mirror losses

itf.te = sqrt(75e-6);                   % end mirror amplitude transmission
itf.le = 0;                             % end mirror losses
itf.re = sqrt(1 - itf.le - itf.te^2);   % end mirror amplitude reflection

% compute the arm cavity mode for latter computations
cav = cavity_mode(itf.RCi, itf.RCe, itf.L);

% compute finesse
itf.Finesse = pi * sqrt(itf.ri*itf.re) / (1 - itf.ri * itf.re);
```

The definition of recycling cavities is instead different, since it simulates the full telescope:

```
% Recycling cavities parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

itf.tp = sqrt(0.04);                    % power recycling mirror amplitude transmission
itf.lp = 0;                             % power recycling mirror losses
itf.rp = sqrt(1 - itf.lp - itf.tp^2);   % power recycling mirror amplitude reflection

itf.ts = sqrt(0.11);            % signal recycling mirror amplitude transmission
itf.ls = 35e-6;                 % signal recycling mirror losses
itf.rs = sqrt(1 - itf.ls - itf.ts^2);   % signal recycling mirror amplitude reflection

```

```
11 itf.Lprc = 37.3444;
12 itf.Lsrc = 35.9613;
13 itf.Schnupp = 0.10;              % Schnupp asymmetry
14
15 itf.fmod1 = 6020832;
16 itf.fmod2 = 54187479;
17 itf.m = 0.3;
18
19 % Define astigmatic recycling cavities
20 % (total power recycling length 37.25718 m)
21
22 itf.prec.L1 = 10.413;           % distances between mirrors
23 itf.prec.L2 = 10.7;
24 itf.prec.L3 = 10.6742 + 5.55718;
25 itf.prec.R2 = -2.0;             % radii of curvature
26 itf.prec.R3 = 23.0304;
27 itf.prec.alpha = 1.36;          % incidence angle (degrees)
28
29 itf.srec.L1 = 10.413;           % distances between mirrors
30 itf.srec.L2 = 10.7;
31 itf.srec.L3 = 9.29 + 5.55718;
32 itf.srec.R2 = -2.0;             % radii of curvature
33 itf.srec.R3 = 23.0304;
34 itf.srec.alpha = 1.36;          % incidence angle (degrees)
35
36 % Build recycling mirrors (which incorporates the description of NDRC)
37 [R,T, itf.RCp, itf.wp] = astigmatic_power_recycling(itf.prec, cav, mm);
38 itf.Rp = itf.rp * R;
39 itf.Tp = itf.tp * T;
40
41 [R,T, itf.RCs, itf.ws] = astigmatic_signal_recycling(itf.srec, cav, mm);
42 itf.Rs = itf.rs * R;
43 itf.Ts = itf.ts * T;
44
45 % Gouy phase already considered in model of astigmatic cavities
46 itf.phig_p = 0;
47 itf.phig_s = 0;
```

Lines 1 to 17 defines the global parameters of the recycling cavity as before. The definition of the NDRC telescopes is in lines between 19 and 34, in form of mirror radii of curvature and distances between mirrors. The angle of incidence is defined on line 34. All these parameters are collected in MATLAB structures. A full model of the NDRC cavity is then build, using beam parameter propagation with ABCD matrices by the functions called in lines 36 to 43. The recycling cavity is described all together with a coupling matrix (including astigmatism and Gouy phase in propagation) which replaces the recycling mirror matrix. In addition the Gouy phase is put to zero (lines 46-47) to avoid double counting.

```
% compute all thermal lenses for different powers
tic;
NPT = 300;
pabs = linspace(1e-6, 0.3, NPT);
f = 2700 ./ pabs;
L = lenses(f, itf.ncav.w1, list_even_modes(8), rmir, npt, thr);
el = toc;
disp(['Elapsed time: ' num2str(el) ' s'])
```

```
9
10  %  rescale lenses to maintain 00 resonant
11  for i=1:length(L)
12      LL(:,:,i) = squeeze(L(:,:,i)) / exp(1j * angle(L(1,1,i)));
13  end
14
15  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
16  %%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEFINE LENSINGS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
17  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
18
19  % build up the 2d map of lenses
20
21  nlens = 70;
22  LN = zeros(M,M,nlens*nlens);
23  LW = zeros(M,M,nlens*nlens);
24  pabs_ni = zeros(nlens*nlens,1);
25  pabs_wi = zeros(nlens*nlens,1);
26
27  for i=1:nlens
28      for j=1:nlens
29          pabs_ni(nlens*(i-1)+j) = pabs(2*i - 1);
30          LN(:,:,nlens*(i-1)+j) = LL(:,:,2*i - 1);
31          if mod(i,2)
32              pabs_wi(nlens*(i-1)+j) = pabs(2*j - 1);
33              LW(:,:,nlens*(i-1)+j) = LL(:,:,2*j - 1);
34          else
35              pabs_wi(nlens*(i-1)+j) = pabs(2*nlens - 2*j + 1);
36              LW(:,:,nlens*(i-1)+j) = LL(:,:,2*nlens - 2*j + 1);
37          end
38      end
39  end
40
41  NPT = nlens^2;
```

The definition of lenses is performed in this part of the code. The first part (lines 1-13) is similar as before. Since the goal of this simulation is to create a 2-dimensional map of fields as a function of independent thermal lenses in the two input mirrors. To obtain this, the space of lensings corresponding to NI and WI absorption going from 0 to 140 mW is covered by a continuous scan: keeping one mirror at a constant lensing, the other one is scanned from 0 to the maximum, then the first mirror is moved up one step and the second one is scanned back from the maximum to 0. In this way some continuity of parameters is reached. This is done in lines 19-41. The output is a list of pairs of lensing matrices which will be used in the following loop.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% LOCKED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

tic;
Crc = zeros(M,M,NPT);
Csc = zeros(M,M,NPT);
Cdf = zeros(M,M,NPT);
Cn = zeros(M,M,NPT);
Cw  = zeros(M,M,NPT);

Urc = zeros(M,M,NPT);
```

```
Usc = zeros(M,M,NPT);
Udf = zeros(M,M,NPT);
Un  = zeros(M,M,NPT);
Uw  = zeros(M,M,NPT);

Lrc = zeros(M,M,NPT);
Lsc = zeros(M,M,NPT);
Ldf = zeros(M,M,NPT);
Ln  = zeros(M,M,NPT);
Lw  = zeros(M,M,NPT);

U2rc = zeros(M,M,NPT);
U2sc = zeros(M,M,NPT);
U2df = zeros(M,M,NPT);
U2n  = zeros(M,M,NPT);
U2w  = zeros(M,M,NPT);

L2rc = zeros(M,M,NPT);
L2sc = zeros(M,M,NPT);
L2df = zeros(M,M,NPT);
L2n  = zeros(M,M,NPT);
L2w  = zeros(M,M,NPT);

NFR = 100;
fr = logspace(1,4,NFR);

TFp = zeros(NFR,NPT);
TFm = zeros(NFR,NPT);

Z = zeros(2, NPT);
phisr = zeros(1,NPT);
phisr(1) = 0.15;
zdarm = zeros(1,NPT);
zdarm(1) = 1e-11;

itf.phi_src = 0.15;
itf.CARM = 0;
itf.MICH = 0;
itf.PRCL = 0;

% loop over all lenses
h = waitbar(0, 'Dual recycled locked');
for i=1:NPT
    % update ITF input thermal lenses
    itf.ncav.Lens = squeeze(LN(:,:,i));
    itf.wcav.Lens = squeeze(LW(:,:,i));

    % FIRST LOCK PRCL AND CARM %%%%%%%%%%%%%

    % use last locking point as a starting point
    if i>1
        Z(:,i) = Z(:,i-1);
    end

    % maximize sb in PRC
    Z(1,i) = fminsearch(@(x) -maximize_sb(itf, x, Z(2,i)), Z(1,i));
    % then maximize car in arm
    Z(2,i) = fminsearch(@(x) -maximize_car(itf, Z(1,i), x), Z(2,i));

    itf.PRCL = 1e-12 * Z(1,i);
    itf.CARM = 1e-12 * Z(2,i);

    % THEN LOCK DARM %%%%%%%%%%%%%%%%%%%%%%%%

    % maintain the same TEM00 power at dark port
    if i>1
        zdarm(i) = zdarm(i-1);
    end
    zdarm(i) = fzero(@(x) error_darm(itf, x), zdarm(i));
    itf.DARM = 1e-12 * zdarm(i);

    % THEN LOCK SRCL %%%%%%%%%%%%%%%%%%%%%%%%

    % try to lock SRC by maximizing TF at peak frequency
    if i>1
        phisr(i) = phisr(i-1);
    end

    phisr(i) = fminsearch(@(x) -maximize_sens(itf,x), phisr(i));
    itf.phi_src = phisr(i);

    % compute fields
    [Crc(:,:,i), Csc(:,:,i), Cdf(:,:,i), Cn(:,:,i), Cw(:,:,i)] = ...
                                     dualrecycled_fields(itf, 0);
    [U1rc(:,:,i), U1sc(:,:,i), U1df(:,:,i), U1n(:,:,i), U1w(:,:,i)] = ...
                                     dualrecycled_fields(itf, +itf.fmod1);
    [L1rc(:,:,i), L1sc(:,:,i), L1df(:,:,i), L1n(:,:,i), L1w(:,:,i)] = ...
                                     dualrecycled_fields(itf, -itf.fmod1);
    [U2rc(:,:,i), U2sc(:,:,i), U2df(:,:,i), U2n(:,:,i), U2w(:,:,i)] = ...
                                     dualrecycled_fields(itf, +itf.fmod2);
    [L2rc(:,:,i), L2sc(:,:,i), L2df(:,:,i), L2n(:,:,i), L2w(:,:,i)] = ...
                                     dualrecycled_fields(itf, -itf.fmod2);

    dccar(i) = dc(squeeze(Crc(:,1,i)), 1);
    dcusb(i) = dc(squeeze(U1rc(:,1,i)), 1);
    dclsb(i) = dc(squeeze(L1rc(:,1,i)), 1);

    % compute DARM TF to dark fringe
    x = darm_signal(itf,fr);
    TFp(:,i) = x(1,:).';
    x = darm_signal(itf,-fr);
    TFm(:,i) = x(1,:).';
    tf(:,i) = Cdf(1,1,i) * conj(TFm(:,i)) + conj(Cdf(1,1,i)) * TFp(:,i);

    el = toc;
    eta = floor((el / i) * (NPT - i));
```

```
119     waitbar(i/NPT, h, ['Dual recycled locked (', num2str(floor(100*i/NPT)), '% ETA = ', ...
120                                             num2str(eta), 's)']);
121
122     fprintf(2, 'PRCL = %g CARM = %g DARM = %g\n', itf.PRCL, itf.CARM, itf.DARM);
123     fprintf(2, '     PRC car = %g ARM car = %g  PRC USB = %g PRC LSB = %g\n', ...
124                 dccar(i), abs(Cn(1,1,i))^2, dcusb(i), dclsb(i));
125     fprintf(2, 'phi_src = %f TF = %g\n\n', phisr(i), abs(TFm(52,1)));
126 end
127 close(h);
```

Lines from 1 to 46 allocates the MATLAB variables that will be filled during the loop. For each step of the loop, the two input mirror lenses are updated (lines 57 and 58) and then a locking procedure is carried out. Lines 60-73 computes the optimal working point for CARM and PRCL as explained before. Then DARM is locked to maintain a constant carrier TEM_{00} at dark port, which corresponds to the nominal value expected for an offset of 10^{-11} m. The function `error_darm` returns indeed the difference between the computed carrier power at dark port and the expected one, as a function of the DARM tuning.

Lines between 84 and 92 then optimize the signal recycling tuning. This is done by maximizing the interferometer response around 300 Hz, which corresponds to the region of maximum sensitivity. The function `maximize_sens` returns the absolute value of the ITF response to a differential motion around 300 Hz.

Then lines 95-103 computes static fields for carrier and both sideband pair. Lines 110-115 compute the transfer function from a DARM motion to the dark fringe carrier power in the TEM_{00} mode, which is the gravitational wave channel. This is done by first computing the propagation of signal sidebands generated at the end mirrors to the dark port, for both positive and negative frequencies (using the function `darm_signal` on lines 111 and 114). These are then combined with the DC carrier power at dark port to obtain the transfer function (line 115).

```
% compute total DC fields
df_car = dc(squeeze(Cdf(:,1,:)), 1);
sr_car = dc(squeeze(Csc(:,1,:)), 1);
pr_car = dc(squeeze(Crc(:,1,:)), 1);
n_car = dc(squeeze(Cn(:,1,:)), 1);
w_car = dc(squeeze(Cw(:,1,:)), 1);

pr_usb1 = dc(squeeze(U1rc(:,1,:)), 1);
pr_lsb1= dc(squeeze(L1rc(:,1,:)), 1);
pr_usb2 = dc(squeeze(U2rc(:,1,:)), 1);
pr_lsb2= dc(squeeze(L2rc(:,1,:)), 1);
sr_usb2 = dc(squeeze(U2sc(:,1,:)), 1);
sr_lsb2= dc(squeeze(L2sc(:,1,:)), 1);

pr_usb1_00 = abs(squeeze(U1rc(1,1,:))).^2;
pr_lsb1_00 = abs(squeeze(L1rc(1,1,:))).^2;
pr_usb2_00 = abs(squeeze(U2rc(1,1,:))).^2;
pr_lsb2_00 = abs(squeeze(L2rc(1,1,:))).^2;

sr_usb2_00 = abs(squeeze(U2sc(1,1,:))).^2;
sr_lsb2_00 = abs(squeeze(L2sc(1,1,:))).^2;

df_car_00 = abs(squeeze(Cdf(1,1,:))).^2;
```

Here the `dc` function is used to compute the total power for all fields. Afterward the power in the fundamental mode only is computed.

```
% remap all powers in 2x2 grids
PABS_NI = zeros(nlens, nlens);
PABS_WI = zeros(nlens, nlens);
DF_CAR = zeros(nlens, nlens);
DF_CAR_00 = zeros(nlens, nlens);
SR_CAR = zeros(nlens, nlens);
PR_CAR = zeros(nlens, nlens);
PR_USB1 = zeros(nlens, nlens);
PR_LSB1 = zeros(nlens, nlens);
PR_USB1_00 = zeros(nlens, nlens);
PR_LSB1_00 = zeros(nlens, nlens);
PR_USB2 = zeros(nlens, nlens);
PR_LSB2 = zeros(nlens, nlens);
PR_USB2_00 = zeros(nlens, nlens);
PR_LSB2_00 = zeros(nlens, nlens);
SR_USB2 = zeros(nlens, nlens);
SR_LSB2 = zeros(nlens, nlens);
SR_USB2_00 = zeros(nlens, nlens);
SR_LSB2_00 = zeros(nlens, nlens);
N_CAR = zeros(nlens, nlens);
W_CAR = zeros(nlens, nlens);
ZPRCL  = zeros(nlens, nlens);
ZCARM  = zeros(nlens, nlens);
ZDARM  = zeros(nlens, nlens);
PHISRC = zeros(nlens, nlens);

for i=1:nlens
    for j=1:nlens
        if mod(i,2)
            PABS_NI(i,j) = pabs_ni(nlens*(i-1)+j);
            PABS_WI(i,j) = pabs_wi(nlens*(i-1)+j);
            DF_CAR(i,j) = df_car(nlens*(i-1)+j);
            DF_CAR_00(i,j) = df_car_00(nlens*(i-1)+j);
            SR_CAR(i,j) = sr_car(nlens*(i-1)+j);
            PR_CAR(i,j) = pr_car(nlens*(i-1)+j);
            PR_USB1(i,j) = pr_usb1(nlens*(i-1)+j);
            PR_LSB1(i,j) = pr_lsb1(nlens*(i-1)+j);
            PR_USB2(i,j) = pr_usb2(nlens*(i-1)+j);
            PR_LSB2(i,j) = pr_lsb2(nlens*(i-1)+j);
            PR_USB1_00(i,j) = pr_usb1_00(nlens*(i-1)+j);
            PR_LSB1_00(i,j) = pr_lsb1_00(nlens*(i-1)+j);
            PR_USB2_00(i,j) = pr_usb2_00(nlens*(i-1)+j);
            PR_LSB2_00(i,j) = pr_lsb2_00(nlens*(i-1)+j);
            SR_USB2(i,j) = sr_usb2(nlens*(i-1)+j);
            SR_LSB2(i,j) = sr_lsb2(nlens*(i-1)+j);
            SR_USB2_00(i,j) = sr_usb2_00(nlens*(i-1)+j);
            SR_LSB2_00(i,j) = sr_lsb2_00(nlens*(i-1)+j);
            N_CAR(i,j)  = n_car(nlens*(i-1)+j);
            W_CAR(i,j)  = w_car(nlens*(i-1)+j);
            ZPRCL(i,j)   = Z(1,nlens*(i-1)+j);
            ZCARM(i,j)   = Z(2,nlens*(i-1)+j);
            ZDARM(i,j)   = zdarm(nlens*(i-1)+j);
            PHISRC(i,j)  = phisr(nlens*(i-1)+j);
        else
            PABS_NI(i,nlens - j + 1) = pabs_ni(nlens*(i-1)+j);
            PABS_WI(i,nlens - j + 1) = pabs_wi(nlens*(i-1)+j);
            DF_CAR(i,nlens - j + 1) = df_car(nlens*(i-1)+j);
            DF_CAR_00(i,nlens - j + 1) = df_car_00(nlens*(i-1)+j);
            SR_CAR(i,nlens - j + 1) = sr_car(nlens*(i-1)+j);
            PR_CAR(i,nlens - j + 1) = pr_car(nlens*(i-1)+j);
            PR_USB1(i,nlens - j + 1) = pr_usb1(nlens*(i-1)+j);
            PR_LSB1(i,nlens - j + 1) = pr_lsb1(nlens*(i-1)+j);
            PR_USB2(i,nlens - j + 1) = pr_usb2(nlens*(i-1)+j);
            PR_LSB2(i,nlens - j + 1) = pr_lsb2(nlens*(i-1)+j);
            PR_USB1_00(i,nlens - j + 1) = pr_usb1_00(nlens*(i-1)+j);
            PR_LSB1_00(i,nlens - j + 1) = pr_lsb1_00(nlens*(i-1)+j);
            PR_USB2_00(i,nlens - j + 1) = pr_usb2_00(nlens*(i-1)+j);
            PR_LSB2_00(i,nlens - j + 1) = pr_lsb2_00(nlens*(i-1)+j);
            SR_USB2(i,nlens - j + 1) = sr_usb2(nlens*(i-1)+j);
            SR_LSB2(i,nlens - j + 1) = sr_lsb2(nlens*(i-1)+j);
            SR_USB2_00(i,nlens - j + 1) = sr_usb2_00(nlens*(i-1)+j);
            SR_LSB2_00(i,nlens - j + 1) = sr_lsb2_00(nlens*(i-1)+j);
            N_CAR(i,nlens - j + 1)  = n_car(nlens*(i-1)+j);
            W_CAR(i,nlens - j + 1)  = w_car(nlens*(i-1)+j);
            ZPRCL(i,nlens - j + 1)   = Z(1,nlens*(i-1)+j);
            ZCARM(i,nlens - j + 1)   = Z(2,nlens*(i-1)+j);
            ZDARM(i,nlens - j + 1)   = zdarm(nlens*(i-1)+j);
            PHISRC(i,nlens - j + 1)  = phisr(nlens*(i-1)+j);
        end
    end
end
```

This code is used to remap all results in $M \times M$ matrices which can be easily used to produce 3d-plots with MATLAB. The code takes into account the particular scanning used in the simulation.

```
% remap transfer functions and compute sensitivity
hplanck = 6.626068e-34;
nu = 299792458 / 1.064e-6;
Pin = 125;
shot = sqrt(2*hplanck*nu*abs(squeeze(Cdf(1,1,:))).^2 * Pin);

TF = zeros(NFR, nlens, nlens);
SENS = zeros(NFR, nlens, nlens);

for i=1:nlens
    for j=1:nlens
        if mod(i,2)
            TF(:,i,j) = abs(tf(:,nlens*(i-1)+j));
            SENS(:,i,j) = abs(shot(nlens*(i-1)+j) ./ tf(:,nlens*(i-1)+j)./ 3000);
        else
            TF(:,i,nlens - j + 1) = abs(tf(:,nlens*(i-1)+j));
            SENS(:,i,nlens - j + 1) = abs(shot(nlens*(i-1)+j) ./ tf(:,nlens*(i-1)+j)./ 3000);
        end
    end
end
```

Here the shot-noise-limited sensitivity of the detector is computed, starting from the simulated power at dark port and DARM transfer function. Then the result is converted in 2-dimensional maps.

```
figure('Position', [0,0,1200,1200])                % dark fringe total power
h = pcolor(1000*PABS_NI, 1000*PABS_WI, DF_CAR);
colorbar;
set(h, 'EdgeColor', 'none')
hold on
[c,h] = contour3(1000*PABS_NI, 1000*PABS_WI, DF_CAR, 'LineColor', 'k');
clabel(c,h, 'FontSize', 14);
set(gca, 'fontsize', 15)
title({'Dark fringe power (total) [W]';''}, 'fontsize', 20)
xlabel('NI absorbed power [mW]', 'fontsize', 15)
ylabel('WI absorbed power [mW]', 'fontsize', 15)
pbaspect([1 1 1])
ax1 = gca;
ax2 = axes('Position', get(ax1, 'Position'), 'XAxisLocation','top', ...
    'YAxisLocation','right', 'Color', 'none', 'XColor','k','YColor','k');
set(ax2, 'XLim', get(ax1, 'XLim'));
set(ax2, 'YLim', get(ax1, 'YLim'));
flens = [20e3, 30e3, 40e3, 60e3, 100e3, 200e3];
ftic  = {'200km'; '100km'; '60km'; '40km'; '30km'; '20km'};
plens = 2700./flens;
set(ax2, 'XTick', 1000*plens(end:-1:1))
set(ax2, 'YTick', 1000*plens(end:-1:1))
set(ax2, 'XTickLabel', ftic)
set(ax2, 'YTickLabel', ftic)
pbaspect([1 1 1])
set(ax2, 'Position', get(ax1, 'Position'))
```

This is just an example of how to plot 2-dimensional maps, including contour plots and multiple scales.

2.4 Simulation of mis-alignment in NDRC cavities

This example can be found in `examples/alignment_ndrc.m`. It uses a different function to build the recycling cavities, which computes the coupling matrix for each mirror of the telescope separately and then combine everything together. This allows defining the mis-alignment of all three mirrors:

```
% Build perfectly aligned recycling mirrors
[R, T, itf.RCp, itf.wp] = ndrc_power_recycling(itf.prec, cav, mm, 0, 0, 0);
itf.Rp = itf.rp * R;
itf.Tp = itf.tp * T;


[R,T, itf.RCs, itf.ws] = ndrc_signal_recycling(itf.srec, cav, mm, 0, 0, 0);;
itf.Rs = itf.rs * R;
itf.Ts = itf.ts * T;
```

Here the functions `ndrc_power_recycling` returns the reflection and transmission matrices which describes the recycling cavity, provided the cavity parameters, the modes to be used and the three tilt angles.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% MISALIGN PRM1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

NPT = 30;
a = linspace(0, 5e-6, NPT);
```

```

Crc_prm1 = zeros(M,M,NPT);
Csc_prm1 = zeros(M,M,NPT);
Cdf_prm1 = zeros(M,M,NPT);
Cn_prm1 = zeros(M,M,NPT);
Cw_prm1  = zeros(M,M,NPT);
U1rc_prm1 = zeros(M,M,NPT);
U1sc_prm1 = zeros(M,M,NPT);
U1df_prm1 = zeros(M,M,NPT);
U1n_prm1  = zeros(M,M,NPT);
U1w_prm1  = zeros(M,M,NPT);
L1rc_prm1 = zeros(M,M,NPT);
L1sc_prm1 = zeros(M,M,NPT);
L1df_prm1 = zeros(M,M,NPT);
L1n_prm1  = zeros(M,M,NPT);
L1w_prm1  = zeros(M,M,NPT);
U2rc_prm1 = zeros(M,M,NPT);
U2sc_prm1 = zeros(M,M,NPT);
U2df_prm1 = zeros(M,M,NPT);
U2n_prm1  = zeros(M,M,NPT);
U2w_prm1 = zeros(M,M,NPT);
L2rc_prm1 = zeros(M,M,NPT);
L2sc_prm1 = zeros(M,M,NPT);
L2df_prm1 = zeros(M,M,NPT);
L2n_prm1  = zeros(M,M,NPT);
L2w_prm1  = zeros(M,M,NPT);

tic;
h = waitbar(0, 'Computing');
for i=1:NPT
    % define the recycling cavity with tilted PRM1
    [R, T, itf.RCp, itf.wp] = ndrc_power_recycling(itf.prec, cav, mm, a(i), 0, 0);
    itf.Rp = itf.rp * R;
    itf.Tp = itf.tp * T;

    % lock interferometer
    zprcl(i) = fminsearch(@(x) -maximize_sb(itf, x, 0), 0);
    zcarm(i) = fminsearch(@(x) -maximize_car(itf, zprcl(i), x), 0);
    itf.PRCL = 1e-12 * zprcl(i);
    itf.CARM = 1e-12 * zcarm(i);
    zdarm(i) = fzero(@(x) error_darm(itf, x), 10);
    itf.DARM = 1e-12 * zdarm(i);
    phisr(i) = fminsearch(@(x) -maximize_sens(itf,x), 0.16);
    itf.phi_src = phisr(i);

    % compute fields
    [Crc_prm1(:,:,i), Csc_prm1(:,:,i), Cdf_prm1(:,:,i), Cn_prm1(:,:,i), Cw_prm1(:,:,i)] = ...
                                                   dualrecycled_fields(itf, 0);
    [U1rc_prm1(:,:,i), U1sc_prm1(:,:,i), U1df_prm1(:,:,i), U1n_prm1(:,:,i), U1w_prm1(:,:,i)] = ...
                                                   dualrecycled_fields(itf, +itf.fmod1);
    [L1rc_prm1(:,:,i), L1sc_prm1(:,:,i), L1df_prm1(:,:,i), L1n_prm1(:,:,i), L1w_prm1(:,:,i)] = ...
                                                   dualrecycled_fields(itf, -itf.fmod1);
    [U2rc_prm1(:,:,i), U2sc_prm1(:,:,i), U2df_prm1(:,:,i), U2n_prm1(:,:,i), U2w_prm1(:,:,i)] = ...
```

```
60                                                          dualrecycled_fields(itf, +itf.fmod2);
61     [L2rc_prm1(:,:,i), L2sc_prm1(:,:,i), L2df_prm1(:,:,i), L2n_prm1(:,:,i), L2w_prm1(:,:,i)] = ...
62                                                          dualrecycled_fields(itf, -itf.fmod2);
63 
64     el = toc;
65     eta = floor((el / i) * (NPT - i));
66     waitbar(i/NPT, h, ['Dual recycled locked (', num2str(floor(100*i/NPT)), '% ETA = ', num2str(eta), '
67 end
68 close(h)
```

Lines 5 and 6 define the desired mis-alignments of PRM3. Lines from 8 to 32 as usual pre-allocate MATLAB variables. The loop on lines 35-62 goes over all angles. Lines from 38 to 40 redefine the power recycling cavity taking into account the fact that PRM3 is mis-aligned. The other lines simply lock the ITF as before and compute all fields.

3 Function reference

3.1 Structures

Some basic MATLAB structures are used to define the basic parameters of the simulation. The main one is the `itf` structure, which contains references to other structures (`cav`, `prec` and `srec`). The following sections define the members of these structures.

3.1.1 `itf`

`RCi` radius of curvature of the ideal arm cavity input mirror;

`RCe` radius of curvature of the ideal arm cavity end mirror;

`L` arm cavity length;

`ti` amplitude transmittance of arm cavity input mirror;

`ri` amplitude reflection of arm cavity input mirror;

`li` losses in arm cavity input mirror;

`te` amplitude transmittance of arm cavity end mirror;

`re` amplitude reflection of arm cavity end mirror;

`le` losses in arm cavity end mirror;

`cav` structure containing ideal cavity parameters (built with `cavity_mode` function);

`tp` amplitude transmittance of power recycling mirror;

`rp` amplitude reflection of power recycling mirror;

`lp` losses in power recycling mirror;

`ts` amplitude transmittance of signal recycling mirror;

`rs` amplitude reflection of signal recycling mirror;

`ls` losses in signal recycling mirror;

`Lprc` length of power recycling cavity;

`Lsrc` length of signal recycling cavity;

`Schnupp` Schnupp asymmetry;

`fmod1, fmod2` modulation frequencies;

`prec` structure containing power recycling NDRC parameters;

`srec` structure containing signal recycling NDRC parameters;

`Rp` matrix reflection coefficient of power recycling cavity (to be built using functions like `mirror_reflection`, `astigmatic_power_recycling`, etc.);

`Tp` matrix transmission coefficient of power recycling cavity (to be built using functions like `mirror_reflection`, `astigmatic_power_recycling`, etc.);

`Rs` matrix reflection coefficient of signal recycling cavity (to be built using functions like `mirror_reflection`, `astigmatic_signal_recycling`, etc.);

`Ts` matrix transmission coefficient of signal recycling cavity (to be built using functions like `mirror_reflection`, `astigmatic_signal_recycling`, etc.);

`phig_p` Gouy phase in propagation inside power recycling cavity (set to zero if NDRC are fully simulated);

`phig_s` Gouy phase in propagation inside signal recycling cavity (set to zero if NDRC are fully simulated);

`aRCin` real radius of curvature of the north arm input mirror;

`aRCen` real radius of curvature of the north arm end mirror;

`aRCiw` real radius of curvature of the west arm input mirror;

`aRCew` real radius of curvature of the west arm end mirror;

`aRCp` real radius of curvature of the power recycling mirror;

`aRCs` real radius of curvature of the signal recycling mirror;

`fNI` focal length of the north input mirror thermal lens;

`fWI` focal length of the west input mirror thermal lens;

`ncav` structure describing north arm cavity (built with `build_cavity`);

`wcav` structure describing west arm cavity (built with `build_cavity`);

`DARM` microscopic tuning of DARM degree of freedom (in meters);

`CARM` microscopic tuning of CARM degree of freedom (in meters);

`PRCL` microscopic tuning of PRCL degree of freedom (in meters);

`MICH` microscopic tuning of MICH degree of freedom (in meters);

`phi_src` signal recycling cavity tuning;

3.1.2 `cav`

This structure should not be modified by the user. It must be build with the function `cavity_mode`.

3.1.3 `prec` and `srec`

These two structures are similar and contains all the information to describe the optical telescope inside NDRC:

`L1` distance between RM1 and RM2;

`L2` distance between RM2 and RM3;

`L3` distance between RM3 and BS plus mean value of distances between BS and arm cavity input mirrors;

`R2` radius of curvature of PRM2;

`R3` radius of curvature of PRM3;

`alpha` angle of incidence of beams on folding mirrors.

3.2 Functions to define single optical elements

3.2.1 `mirror_reflection`

`M = mirror_reflection(Rc, R, w, modes, rmir, npt, thr)`

Returns the mode coupling matrix for reflection off a mirror, compute numerically from the parameters `rmir, npt, thr` (radius of the mirror, number of point per dimension for numerical integration, threshold to put t zero small matrix elements). `Rc` is the mirror radius of curvature. `R` and `w` the beam wave-front curvature and spot size at the mirror. `modes` is a list of the modes to be used in the computation.

3.2.2 `tilted_mirror_reflection`

`M = tilted_mirror_reflection(Rc, theta, R, w, modes, rmir, npt, thr)`

Similar to `mirror_reflection`, but can include a microscopic tilt of the mirror, defined by the variable `theta`.

3.2.3 `mirror_reflection_qxy`

`M = mirror_reflection_qxy(Rc, aoi, theta, qix, qiy, qox, qoy, modes, rm, npt, thr)`

Similar to `mirror_reflection`, but can use different and astigmatic basis for incident and reflected beam. `Rc` is the mirror radius of curvature, `aoi` the angle of incidence of the beam on the mirror, `theta` is the mirror microscopic tilt with respect to the reference position. `qix` and `qiy` are the input beam complex parameters for the two directions, and `qox` and `qoy` are the same for the output beam.

3.2.4 `mirror_reflection_map`

`M = mirror_reflection_map(mapfile, s, c, w, modes)`

This function is not yet well tested!

This function can be used to build the mirror reflection matrix given the mirror map to be read from the file `mapfile`. `c` must be a vector containing the index coordinates of the optical center of the map. `s` is the step size of the map. The surface map is considered as the deviation of the mirror surface from the ideal one which is well matched to the beam wave-front curvature.

3.2.5 `mirror_transmission_map`

`M = mirror_transmission_map(mapfile, s, c, w, modes)`

This function is not yet well tested!

Same as `mirror_reflection_map`, but compute the transmission matrix.

3.2.6 `lens`

`M = lens(f, w, modes, rlens, npt, thr)`

Compute the transmission matrix of a lens with focal length `f`, given the beam size at the lens `w`, the list of modes to be used `m` and the usual parameters fro numerical integration `rlens, npt, thr` (radius of the lens, number of point per dimension for numerical integration, threshold to put t zero small matrix elements). This function is quite slow, not being optimized yet. It can accept arbitrary modes.

3.2.7 `lens_optimized_even_modes`

`M = lens_optimized_even_modes(f, w, nM, rlens, npt)`

Compute the transmission matrix of a lens with focal length `f`, like `lens`. This function is quite fast since it is optimized for spherical lenses and the use of even modes only. The maximum mode index is `nM`.

3.2.8 `lenses`

`L = lenses(f, w, modes, rlens, npt, thr)`

Similar to `lens`, but accept as input a vector of focal lengths `f` and returns the list of corresponding matrices.

3.2.9 `lenses_optimized`

`L = lenses_optimized(f, w, nM, rlens, npt, thr)`

Similar to `lens_optimized_even_modes`, but accept as input a vector of focal lengths `f` and returns the list of corresponding matrices.

3.3 Functions to define optical configurations

3.3.1 `cavity_mode`

`cav = cavity_mode(R1, R2, L)`

Given the cavity length `L` and the two mirror radii of curvature `R1` (input) and `R2` (end) return a cavity structure containing all informations on the resonant mode.

3.3.2 `build_cavity`

`cav = build_cavity(R1_0, R2_0, L, R1, R2, r1, r2, l1, l2, flens, modes)`

Compute completely one cavity mode and the mirror matrices given: the ideal radii of curvature `R1_0` and`R2_0`, the cavity length `L`, the real radii of curvature `R1` and`R2`, the amplitude reflection coefficients of the two mirrors `r1` and `r2` and their losses `l1` and `l2`, the focal length of the input mirror substrate `flens`, the list of modes to be used `modes`.

3.3.3 `build_cavity_tilted`

`cav = build_cavity_tilted(R1_0, R2_0, L, R1, R2, r1, r2, l1, l2, flens, thetai, thetae, modes)`

The same as `build_cavity`, but including possible tilts of the two mirrors, defined by `thetai` and `thetae`.

3.3.4 `astigmatic_power_recycling`

```
[R, T, RC, w] = astigmatic_power_recycling(rec, cav, modes)
```

Compute a reflection and transmission coupling matrix which completely models the behavior of a non-degenerate power recycling cavity, given the power recycling cavity structure`rec` and the arm cavity structure `cav`. `R` and `T` are the reflection and transmission matrices, while `RC` and `w` are the mean curvature and size of the beam at PRM1 (computed in absence of astigmatism).

3.3.5 `astigmatic_signal_recycling`

```
[R, T, RC, w] = astigmatic_signal_recycling(rec, cav, modes)
```

The same as `astigmatic_power_recycling`, but for the signal recycling cavity.

3.3.6 `ndrc_power_recycling`

```
[R, T, R1, w] = ndrc_power_recycling(rec, cav, modes, theta1, theta2, theta3)
```

This function is similar to `astigmatic_power_recycling`, but can take into account possible tilts of the various recycling mirrors, defined by `theta1`, `theta2`, `theta3`.

3.3.7 `ndrc_signal_recycling`

```
[R, T, R1, w] = ndrc_signal_recycling(rec, cav, modes, theta1, theta2, theta3)
```

This function is similar to `astigmatic_signal_recycling`, but can take into account possible tilts of the various recycling mirrors, defined by `theta1`, `theta2`, `theta3`.

3.4 Functions to compute fields

3.4.1 `FX`

```
F = FX(cav, dL, f)
```

Computes the reflection matrix of a Fabry-Perot cavity, given its cavity structure `cav`, the microscopic tuning `dL` and the frequency of the input field `f`.

3.4.2 `MAA`, `MSA`, `MAS`, `MSS`

```
M = MAA(itf, f)
```

Compute Michelson interferometer operators (see [1]) given the interferometer structure `itf` and the field frequency `f`.

3.4.3 `dualrecycled_fields`

```
[psi_rc, psi_sc, psi_df, psi_n, psi_w, psi_ref] = dualrecycled_fields(itf, f)
```

Compute static fields for a dual recycled interferometer, given the structure `itf` and the field frequency `f`. The output are transmission matrices from the input of the interferometer to the variouos ports: `psi_rc` inside power recycling cavity, field propagating from PRM to BS; `psi_sc` inside signal recycling cavity, field impinging on SRM; `psi_df` at dark port; `psi_n` inside north arm, field impinging on north end mirror; `psi_w` inside west arm, field impinging on west end mirror; `psi_ref` field reflected by the full interferometer.

3.4.4 darm_signal

```
t = darm_signal(itf, f)
```

Returns the transfer function from a DARM motion to the field at dark port, given the frequency `f` which can be positive or negative.

3.4.5 dc

```
d = dc(a, b)
```

Compute the total DC power given a vector `a` of amplitude for different modes and a global scaling factor `b`.

3.4.6 ac

```
[p,q] = ac(car, usb, lsb, m, phase)
```

Compute demodulated signals in both quadratures given the carrier `car` and sidebands `usb` and `lsb` vectors of field amplitudes, the modulation index `m` and the demodulation phase `phase`.

3.4.7 beamprofile

```
[x,y,b] = beamprofile(modes, a, w, R, r, npt)
```

Return an image of the beam given the list of modes `modes`, a vector of amplitudes `a` of these modes. Also needed are the beam size `a` and curvature `R`. `r` is the radius of the image and `npt` the number of points (resolution).

4 Contact

Please contact the author (gabriele.vajente@ego-gw.it) for any comment or question.

References

[1] G. Vajente, *Modal Interferometer Simulation*, VIR-0142A-10 (2010) 2, 4, 24