

EPT: Error Propagation Toolbox

F. Antonacci [§], P. Bestagini [§], A. Canclini [§], M. Compagnoni [†], A. Sarti [§]

[§] Dipartimento di Elettronica ed Informazione

[†] Dipartimento di Matematica “Francesco Brioschi”

Politecnico di Milano, piazza Leonardo da Vinci, 20133, Milano, Italy

marco.compagnoni@polimi.it, antonacci/bestagini/canclini/sarti@elet.polimi.it

July 27, 2012

If you use the Error Propagation Toolbox in your research, please include in your publications references specific to [1] from which the toolbox is inspired.

1 Introduction

EPT is a Matlab[®] toolbox for analyzing the propagation of the measurement error on an arbitrary unbiased (or small-biased) estimation problem based on the optimization of a cost function. Such problems are, in general, formulated as the combination of multiple measurements (or observations) into a cost function, whose global minimum gives the best estimate of the variable of interest. Provided the covariance matrix of the observations and the symbolic equation of the cost function, EPT calculates the covariance of the estimation, assuming negligible bias on the estimated variable. EPT is based on basic concepts of the catastrophe theory [2], which makes possible to derive a linear relationship between the measurement error and the estimation error. For details on the development of the error propagation analysis, the reader is referred to [1].

It is interesting to observe that the theory implemented in EPT is more general than the well-known Cramer-Rao Lower Bound (CRLB). In fact, while CRLB provides a bound for an estimation problem, the approach based on catastrophe theory gives the theoretical limit for a specific cost function applied to that problem. Moreover, it can be shown that CRLB corresponds to the error propagation analysis applied to a maximum-likelihood cost function, and therefore it can be interpreted as a particular case of the error propagation analysis.

This document is structured as follows. In Section 2 a description of the functionalities of EPT is reported. Section 3 contains a set of usage examples included in EPT.

2 Error propagation analysis using EPT

EPT is suitable for predicting the accuracy of estimation problems in the form

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} J(\mathbf{x}, \mathbf{c}) , \quad (1)$$

where:

- $J(\mathbf{x}, \mathbf{c})$ is a generic cost function, whose Hessian $H_{\mathbf{x}, \mathbf{x}}$ evaluated at the global minimum is non-singular;
- $\mathbf{x} = [x_1, x_2, \dots, x_M]^T \in \mathbb{R}^M$ is a vector of M unknowns to be estimated;
- $\mathbf{c} = [c_1, c_2, \dots, c_N]^T \in \mathbb{R}^N$ is a vector of N measurements.

In an ideal situation, the measurements $\mathbf{c} = \mathbf{c}_0$ are error-free, so that the cost function reaches its global minimum at $J(\mathbf{x}_0, \mathbf{c}_0)$. However, when the measurements $\mathbf{c} = \mathbf{c}_0 + \delta\mathbf{c}$ are corrupted with additive noise $\delta\mathbf{c}$, the global minimum moves to the new position

$$\bar{\mathbf{x}} = \mathbf{x}_0 + \delta\mathbf{x}, \quad (2)$$

where $\delta\mathbf{x}$ is an error term applied to the error-free minimum \mathbf{x}_0 . Provided the covariance matrix $\Sigma_{\mathbf{c}}$ of the measurement error $\delta\mathbf{c}$, EPT predicts the covariance $\Sigma_{\mathbf{x}}$ of the estimation error \mathbf{x} . An highlight of the input and outputs of EPT is illustrated in Figure 1.



Figure 1: Input and outputs of EPT.

2.1 Using the toolbox

The toolbox provides the following set of Matlab[®] functions:

- `EPT_install.p`,
- `EPT_problem.p`,
- `EPT_COV.p`.

We now see in details how to use them, following a step-by-step approach based on the template available in the folder `EPT_template` of the toolbox.

2.1.1 Installation

First of all, the toolbox needs to be installed. This is done trough the function `EPT_install.p`, which asks for temporary or permanent installation on the computer.

2.1.2 Definition of a cost function

We now give an algorithmic guideline for the definition of an arbitrary cost function compliant with EPT. The toolbox provides a template for defining a cost function, available in the file `EPT_template/my_cost_function.m`. For a fully comprehension the reader is suggested to consider the examples in Section 3 as a complement to this guideline. An EPT cost function is a Matlab[®] function in the form

```
function J_sym = my_cost_function(M,N,params)
```

saved into the file `my_cost_function.m`. The input arguments are

- `M`, corresponding to the number of variables to be estimated
- `N`, corresponding to the number of observations
- `params`, which contains eventual parameters of the cost function. Its use is completely arbitrary: it can be a structure, a vector, a matrix, etc. It can also be empty.

The file begins with the definition of two sets of symbolic variables:

- x_1, x_2, \dots, x_N , specifying the variables to be estimated;
- c_1, c_2, \dots, c_M , specifying the observations.

This is done by means of the following code (included into the template):

```
% symbols for the variables to be estimated
for m=1:M
    eval(['syms x_' num2str(m)]);
end

% symbols for the observations
for n=1:N
    eval(['syms c_' num2str(n)]);
end
```

After these definitions, the symbolic form of the cost function is computed and stored into the variable `J_sym`:

```
J_sym = ..... any function of:
           x_1 ... x_M,
           c_1 ... c_N
           params
```

The usage of `params` may be more clear after reading the Section devoted to the examples.

2.1.3 Setting up a problem

We now describe how to perform an error propagation analysis with EPT. We suggest to refer to the template script `EPT_template/my_test.m`, which implements the following sequence of operations:

1. definition of the handle to the cost function:

```
cost_function_hdl = @my_cost_function;
```

2. definition of the set of K evaluation points:

```
ev_points = ... ; % [K x M] matrix
```

3. computation of the noise-free observations:

```
observations = ... ; % [N x K] matrix
```

4. definition of the eventual parameters of the cost function:

```
params = ... ; % any type of vector/matrix/structure...
```

5. definition of the estimation problem:

```
prob = EPT_problem(cost_function_hdl, ev_points, ...
                   observations, params)
```

6. definition of the observation covariance matrix Σ_c :

```
Pc = ... ; % [N x N] matrix
```

7. prediction of the covariance matrix $\Sigma_{\mathbf{x}}$ of the estimated variables:

```
Px = EPT_cov(prob,Pc); % Px is a [M x M x K] matrix
                        % containing the covariance of
                        % the estimation for each
                        % of the K test points.
```

The function `EPT_problem` called at step 5 returns the structure `prob`, which is an argument of the function `EPT_cov` (step 7) and contains the following fields:

FIELD	DESCRIPTION
<code>prob.J_sym</code>	symbolic form of the cost function
<code>prob.Hxx_sym</code> <code>prob.Hxc_sym</code>	symbolic form of the Hessian (see [1] for theoretical details)
<code>prob.Hxx</code> <code>prob.Hxc</code>	numerical form of the Hessian (see [1] for theoretical details)
<code>prob.ev_points</code> <code>prob.observations</code> <code>prob.params</code>	evaluation points, noise-free observations, and parameters of the cost function

3 Examples

In this Section we give a description of the examples contained in the folder `EPT_demos` of the toolbox. The first example refers to a very simple estimation problem described in [3]; however it can be very useful for getting familiar with EPT. The second example refers to the typical problem of acoustic source localization. In particular we use EPT for predicting the accuracy of the Squared Range Difference-Least Squares (SRD-LS) algorithm discussed in [4]. Finally, the last example regards the estimation of the pose of an acoustic reflector, and refers to the optimization problem introduced in [5].

3.1 A toy example: error propagation in estimating the phase of a sinusoid

We now consider the estimation problem described in [3], which relies with the estimation of the parameters of discrete signals composed of sum of sinusoids. For the sake of simplicity, we consider a single sinusoid with unitary amplitude

$$c(n) = \cos(\omega n + \phi) + \delta c(n) ,$$

where the frequency $\omega \in [0, \pi]$ is assumed to be known, while the phase term $\phi \in [0, 2\pi]$ has to be estimated from N observations $c(n)$ of the signal at discrete times $n = 0, \dots, N-1$. The term

$\delta c(n)$ denotes an additive noise, which corrupts the observations. As shown in [3], the estimation is based on the solution of a linear least-squares problem:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \|\mathbf{H}\mathbf{x} - \mathbf{c}\|^2, \quad (3)$$

where

$$\mathbf{H} = \begin{bmatrix} 1 & 0 \\ \cos[\omega] & \sin[\omega] \\ \vdots & \vdots \\ \cos[\omega(N-1)] & \sin[\omega(N-1)] \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} c(0) \\ c(1) \\ \vdots \\ c(N-1) \end{bmatrix}.$$

The solution to (3) is given by

$$\hat{\mathbf{x}} = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{c}, \quad \text{where } \hat{\mathbf{x}} = \begin{bmatrix} \hat{x}_1 & = & \cos(\hat{\phi}) \\ \hat{x}_2 & = & -\sin(\hat{\phi}) \end{bmatrix},$$

and an estimate of the phase is therefore obtained as

$$\hat{\phi} = \arctan\left(-\frac{\hat{x}_2}{\hat{x}_1}\right).$$

We now discuss how we can predict the impact of the error $\delta \mathbf{c} = [\delta c(0) \ \delta c(1) \ \dots \ \delta c(N-1)]^T$ on the estimation error $\delta \mathbf{x}$ defined as

$$\delta \mathbf{x} = \begin{bmatrix} \hat{x}_1 - x_1 \\ \hat{x}_2 - x_2 \end{bmatrix} \quad \text{with } x_1 = \cos(\phi), \ x_2 = -\sin(\phi). \quad (4)$$

To this end, we refer to the files included in the folder `EPT_demos/phase_estimation`. We first consider the file `phase_estimation_cost_function.m`, which implements the cost function defined in the problem (3), i.e. $J(\mathbf{x}, \mathbf{c}) = \|\mathbf{H}\mathbf{x} - \mathbf{c}\|^2$, as reported below:

```
function J_sym = phase_estimation_cost_function(M,N,params)

% Symbols for the variables to be estimated
for m=1:M
    eval(['syms x_' num2str(m)]);
end

% Symbols for the observations
for n=1:N
    eval(['syms c_' num2str(n)]);
end

% Read the function parameters
omega = params(1); % the frequency in the range [0,pi]

% Cost function
J_sym = 0;
for n = 1:N
    temp = cos(omega*(n-1))*x_1 + sin(omega*(n-1))*x_2...
        - eval(['c_', num2str(n)]);
    J_sym = J_sym + temp^2;
end
```

The problem is then defined in the file `test_phase_estimation.m`, which also implements a comparison between the result predicted by EPT and results from Monte-Carlo simulations. The first part of that file reflects the structure of the template:

1. first of all, it defines the handle to the cost function:

```
cost_function_hdl = @phase_estimation_cost_function;
```

2. then, it defines the evaluation points, i.e. a vector containing a set of phase values to be tested. Since in this formulation the phase is estimated as two components (see eq.(4)), the number of unknowns is posed to $M = 2$ and therefore the evaluation points are stored into a $K \times 2$ matrix:

```
K = 100;
phi = linspace(0,2*pi,K)';
ev_points = [cos(phi) -sin(phi)];
```

3. after that, the noise-free observations are calculated:

```
N = 6;          % number of observations
omega = pi/4;   % frequency
observations = zeros(N,K);
for n = 1:N
    for k = 1:K
        observations(n,k) = cos(omega*(n-1)+phi(k));
    end
end
```

4. in this case, the only parameter of the cost function is the frequency:

```
params = omega;
```

5. the estimation problem is defined calling:

```
prob = EPT_problem(cost_function_hdl, ev_points,...
                   observations, params);
```

6. before predicting the propagation of the error, we need to define the covariance matrix of the observations. Assuming a Gaussian noise whose components are independent on each observation, the covariance matrix is diagonal:

```
var_noise = 0.001;      % noise variance
Pc = var_noise * eye(N); % [N x N] matrix
```

7. finally, EPT is used for calculating the covariance of the estimation variables:

```
Px = EPT_cov(prob,Pc);
```

From the covariance $\Sigma_{\mathbf{x}}$ of the estimated variable $\hat{\mathbf{x}}$ it is possible to estimate the variance $\sigma_{\hat{\phi}}^2$ of $\hat{\phi}$ adopting the simple transformation¹

$$\sigma_{\hat{\phi}}^2 = \mathbf{T} \Sigma_{\mathbf{x}} \mathbf{T}^T, \text{ with } \mathbf{T} = \begin{bmatrix} \frac{x_2}{x_1^2 + x_2^2} & -\frac{x_1}{x_1^2 + x_2^2} \end{bmatrix}. \quad (5)$$

¹Given the set of variables $\mathbf{x} = [x_1, \dots, x_M]^T$, and a further set defined as functions of \mathbf{x} , i.e. $\mathbf{y} = [g_1(\mathbf{x}), \dots, g_L(\mathbf{x})]^T$, the covariance matrix $\Sigma_{\mathbf{y}}$ of \mathbf{y} can be approximated as [6]

$$\Sigma_{\mathbf{y}} = \mathbf{T} \Sigma_{\mathbf{x}} \mathbf{T}^T, \quad \mathbf{T} = \begin{bmatrix} \frac{\partial g_1(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial g_1(\mathbf{x})}{\partial x_M} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_L(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial g_L(\mathbf{x})}{\partial x_M} \end{bmatrix},$$

where $\Sigma_{\mathbf{x}}$ is the covariance matrix of \mathbf{x} .

This is implemented in the following code, which computes the standard deviation $\sigma_\phi = \sqrt{\sigma_\phi^2}$:

```
EPT_std_phi = zeros(K,1);
for k = 1:K
    x1 = cos(phi(k));
    x2 = -sin(phi(k));
    T = [x2/(x1^2 + x2^2) -x1/(x1^2 + x2^2)];
    EPT_std_phi(k) = sqrt(T*Px(:, :, k)*T');
end
```

Finally, the script performs a set of Monte-Carlo simulations, whose results are compared with the theoretical ones. An example is shown in Figure 2, where the simulation were performed considering $N = 6$ observations of a sinusoidal signal at frequency $\omega = \pi/4$, with phase assuming $K = 100$ values in the range $\phi \in [0, 2\pi]$. The observations are corrupted with 1000 realizations of a zero-mean white noise with variance $\sigma_c^2 = 0.001$.

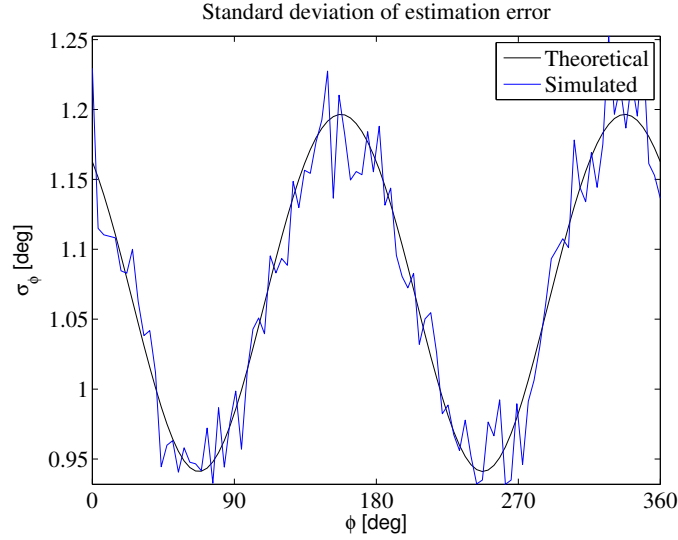


Figure 2: Standard deviation of the estimation: comparison between the theoretical value and the one computed from simulations.

3.2 Error propagation for a source localization problem

We now show how to apply EPT to a source localization problem. At this purpose we refer to [4], where the authors deal with the problem of localizing an acoustic source given a set of range-differences of arrival observations at pairs of microphones. In particular they provide a procedure for efficiently estimating the source location based on the Squared Range Difference-Least Squares cost function.

Let us consider an array of $N + 1$ sensors placed at \mathbf{a}_i , $i = 0, \dots, N$. The first sensor is placed at the origin ($\mathbf{a}_0 = \mathbf{0}$). The acoustic source to be localized is placed at \mathbf{x} (Figure 3). The noise-free range-differences between sensor i and sensor 0 are given by

$$c_{0,i} = \|\mathbf{x} - \mathbf{a}_i\| - \|\mathbf{x}\| \quad i = 1, \dots, N. \quad (6)$$

However the range-differences observations are assumed to be corrupted by a noise term so that they can be written as

$$c_i = c_{0,i} + \delta c_i, \quad i = 1, \dots, N, \quad (7)$$

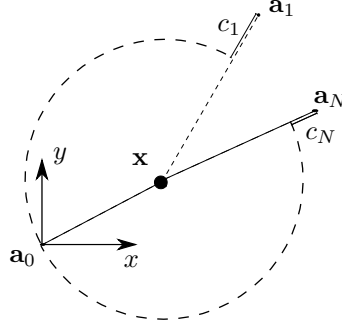


Figure 3: Setup for a source localization problem. The source is represented by \mathbf{x} , microphones are represented by \mathbf{a}_i , and the range differences of arrival are denoted by c_i .

where δc_i accounts for the noise. It can be shown that a solution to the problem of estimating \mathbf{x} given the sensors locations \mathbf{a}_i and the set of noisy range-differences c_i is given by

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} J(\mathbf{x}, \mathbf{c}), \quad (8)$$

where the cost function is defined as

$$J(\mathbf{x}, \mathbf{c}) = \sum_{i=1}^N \left(-2\mathbf{a}_i^T \mathbf{x} - 2c_i \|\mathbf{x}\| - c_i^2 - \|\mathbf{a}_i\|^2 \right)^2, \quad (9)$$

and

$$\mathbf{c} = [c_1 \ c_2 \ \dots \ c_N]^T. \quad (10)$$

As a matter of fact, because of the noise on the observations, the value of $\hat{\mathbf{x}}$ will be different from the real source position \mathbf{x} . Using EPT, the knowledge of the covariance matrix $\Sigma_{\mathbf{c}}$ of the measurements error makes possible to compute the covariance matrix $\Sigma_{\mathbf{x}}$ of the estimation error without the need of heavy time-demanding techniques such as Monte-Carlo simulations.

For a tutorial purpose, we now give a brief overview of the code contained in the folder `EPT_demos/source_localization`. The first file we consider is `SRDLS_cost_function.m`, which implements the cost function $J(\mathbf{x}, \mathbf{c})$ shown in eq.(9) as reported below:

```
function J_sym = SRDLS_cost_function(M,N,params)

% Symbols for the variables to be estimated
%     the variables to be estimated are the 2 coordinates
%     of the source position [x_1, x_2].
for m=1:M
    eval(['syms x_' num2str(m)]);
end

% Symbols for the observations
%     the observations are the N noisy range-differences
%     [c_1, ..., c_N] measured at microphones pairs.
for n=1:N
    eval(['syms c_' num2str(n)]);
end

% Read the function parameters
%     params contains the microphones coordinates as:
%     params = [xm_1    ym_1;
```



```

%           xm_2    ym_2;
%           ...     ... ;
%           xm_N+1  ym_N+1].
mic_pos = params;

% Cost Function
J_sym = 0;
x0 = mic_pos(1,1); % reference microphone x
y0 = mic_pos(1,2); % reference microphone y

for i = 2:N+1
    ax = mic_pos(i,1)-x0;
    ay = mic_pos(i,2)-y0;
    di = eval(['c_' num2str(i-1)]);
    gi = di^2 - (ax^2 + ay^2);
    x = x_1 - x0;
    y = x_2 - y0;
    J_sym = J_sym + (-2*(ax*x + ay*y)-2*di*sqrt(x^2+y^2)-gi)^2;
end

```

The problem is then defined in the file `test_SRDLS.m`, which also contains the code to plot some results predicted by EPT. The first part of that file reflects the structure of the template:

1. first we define the cost function handle:

```
cost_function = @SRDLS_cost_function;
```

2. then, we define the evaluation points, i.e. a matrix containing the coordinates of the tested source positions. Since in this formulation the source and microphones are placed on a plane, the number of coordinates to be estimated is $M = 2$ and therefore the evaluation points are stored into a $K \times 2$ matrix. In this example we place the sources on a `l_x`×`l_y` grid centered at the origin (shown as crosses in Figure 4a):

```

% We first define the grid parameters.
l_x = 3.9; % x grid length in [m]
l_y = 3.9; % y grid length in [m]
num_x = 10; % number of sources on x
num_y = 10; % number of sources on y

% We then define the grid saving each source position
% in the matrix ev_points = [xs_1 ys_1;
%                             ...   ...;
%                             xs_K ys_K];
delta_x = l_x / (num_x-1);
delta_y = l_y / (num_y-1);
num_ev = num_x*num_y;
ev_points = zeros(num_ev,2);
counter = 1;
for i = 1:num_x
    for j = 1:num_y
        ev_points(counter,:) = [delta_x*(i-1) - l_x/2, ...
                                -(-delta_y*(j-1) + l_y/2)];
        counter = counter + 1;
    end
end

```

```

end
end

```

3. after that, we compute the noise-free observations. They coincide with the range-differences between the source and microphone pairs. At this purpose we also define microphones location (shown as circles in Figure 4a):

```

% We define the microphones positions
mic_pos = [ 2  2; ...
            -2  2; ...
            -2 -2; ...
            2 -2];

% We compute the microphone-source distances
distance = zeros(size(mic_pos,1),size(ev_points,1));
for n = 1:size(mic_pos,1),
    for k = 1:size(ev_points,1),
        distance(n,k) = norm(mic_pos(n,:)-ev_points(k,:));
    end
end

% We compute the range differences
observations = zeros(size(mic_pos,1),size(ev_points,1));
for n = 1:size(mic_pos,1)
    for k = 1:size(ev_points,1)
        observations(n,k) = -(distance(1,k)-distance(n,k));
    end
end
observations = observations(2:end,:);

```

4. we now define the parameters. For this problem, the only parameters we use are the microphones positions:

```

params = mic_pos;

```

5. the estimation problem is now defined calling:

```

prob = EPT_problem(cost_function, ev_points, ...
                   observations, params);

```

6. in order to predict the propagation of the error, we need to define the covariance matrix of the observations. Assuming a Gaussian noise whose components are independent on each observation, the covariance matrix is diagonal:

```

var_noise = 0.03^2; % noise variance
Pc = var_noise * eye(size(observations,1)); % [N x N] matrix

```

7. finally, EPT is used for calculating the covariance of the estimation variables:

```

Px = EPT_cov(prob,Pc);

```

Once we have computed the estimation error covariance matrix Σ_x (Px in the code), we can extract the RMSE for each source location. The RMSE for the x and y coordinates of each source position is obtained by computing the square-root of the diagonal element of Σ_x as:

```

RMSE = zeros(size(Px,3),size(Px,1));
for m = 1:size(Px,1)
    for k = 1:size(Px,3)
        RMSE(k,m) = sqrt(Px(m,m,k));
    end
end

```

In order to visualize these results, we plot the x coordinate RMSE of the source over the source grid. In this way we have a visual feedback of the estimation error for each source position as shown in Figure 4b.

```

% Plot the RMSE on x coordinate for every source position
ax_lim = [min(prob.ev_points(:,1)) max(prob.ev_points(:,1)) ...
          min(prob.ev_points(:,2)) max(prob.ev_points(:,2))];
xRMSE = reshape(RMSE(:,1),num_y,num_x);

figure(),
imagesc(ax_lim(1:2),ax_lim(3:4),xRMSE);
colormap('gray'), colorbar
set(gca,'YDir','normal')
set(gca,'Fontname','Times new roman','FontSize',18);
title('RMSE on x [m]')
xlabel('x [m]','interpreter','tex','fontsize',16)
ylabel('y [m]','interpreter','tex','fontsize',16)

```

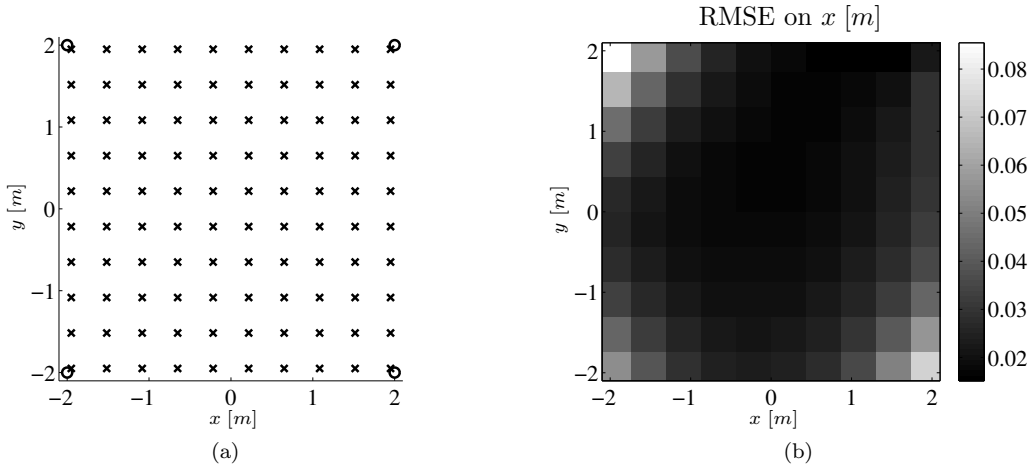


Figure 4: (a): setup used for testing EPT . Crosses represent source positions, circles represent microphones. (b): RMSE on the x coordinate of the source for every source position.

3.3 Error propagation for a reflector localization problem

As a final example, we discuss how to replicate the results shown in [5], relative to the problem of estimating the position of an acoustic planar reflector from Times of Arrival measurements. The estimation algorithm is detailed in [5]. Here we limit to report the simulation setup and discuss some implementation issues.

We consider the scenario depicted in Figure 5. A microphone array is composed of $N = 5$

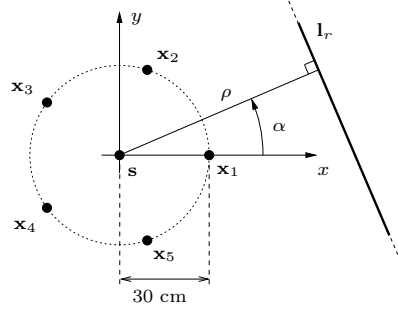


Figure 5: Simulation setup for the problem of reflector localization.

sensors uniformly spaced on a circumference with radius 30 cm, at positions $\mathbf{x}_1, \dots, \mathbf{x}_5$. The line $\mathbf{l}_r = [l_{1_r}, l_{2_r}, l_{3_r}]^T$ of equation $l_{1_r}x + l_{2_r}y + l_{3_r} = 0$ models a planar reflector perpendicular to the plane of the microphones and passing through that line. An acoustic source is placed at \mathbf{s} , whose location coincides with the origin of the reference frame. As discussed in [5], the goal is to estimate the reflector line parameters from the knowledge of the length of the reflected paths between the source and the microphones. In mathematical terms, this problem is formulated as the minimization of a cost function that collects $N = 5$ homogeneous elliptic constraints [5]:

$$J(\mathbf{l}, \mathbf{c}) = \sum_{n=1}^N (\mathbf{l}^T \mathbf{C}_n^* \mathbf{l})^2,$$

where $\mathbf{c} = [c_1, \dots, c_N]^T$ are the length of the reflective paths; and \mathbf{C}_n^* is the dual-conic matrix representing the n th elliptic constraint (see [5] for details). The reflector line is finally estimated as [5]

$$\hat{\mathbf{l}} = \underset{\mathbf{l}}{\operatorname{argmin}} J(\mathbf{l}, \mathbf{c})|_{l_3=1}, \quad (11)$$

where $\hat{\mathbf{l}} = [\hat{l}_{1_r}, \hat{l}_{2_r}, 1]^T$. Notice that, in order to avoid the redundancy determined by the homogeneous coordinates², the problem (11) forces the third line parameter to be $l_3 = 1$.

We now discuss how to define this problem using EPT. We start with the definition of the cost function. In particular, the number of unknowns is $M = 2$, corresponding to the pair (l_1, l_2) . The observations correspond to the vector $\mathbf{c} = [c_1, \dots, c_N]^T$, which includes the length of the $N = 5$ reflective paths. The microphone positions are passed as parameters to the cost function, as it can be noticed in the code listed below:

```
function J_sym = ref_loc_cost_function(M,N,params)

% Symbols for the variables to be estimated
for m=1:M
    eval(['syms x_' num2str(m)]);
end

% Symbols for the observations
for n=1:N
    eval(['syms c_' num2str(n)]);
end

% Cost function
J_sym = 0;
```

²The line parameter vector \mathbf{l} is homogeneous. This means that all the vectors $\lambda \mathbf{l}$, $\lambda \neq 0$, represent the same line. Posing $l_3 = 1$ corresponds to fixing the value of λ .

```

for n = 1:N

    % microphone positions (read from 'params')
    xm = params.mic_pos(n,1); % current x coordinate
    ym = params.mic_pos(n,2); % current y coordinate

    % reflective TOA (corresponding to the observation c_n)
    t = eval(['c_' num2str(n)]);

    % compute the dual ellipse parameters
    a = -4*t^2*(- t^2 + xm^2 + ym^2)*(- t^2 + xm^2 + ym^2);
    b = 0;
    c = a;
    f = -16*t^2*(- t^2 + xm^2 + ym^2);
    d = f*xm;
    e = f*ym;

    % construct the dual ellipse matrix
    CS = [a b/2 d/2;b/2 c e/2;d/2 e/2 f];

    % update the cost function
    temp = ([x_1 x_2 1]*CS*[x_1;x_2;1])^2;
    J_sym = J_sym + simplify(temp); % simplify speeds up the computation
                                   % at the next steps

end

```

After the definition of the cost function, the problem is defined, as usual, by the code:

```

%% 1. definition of the cost function handle

cost_function_hdl = @ref_loc_cost_function;

%% 2. definition of the set of evaluation points

% We define a grid of test reflector lines defined by their
% distance 'rho' and orientation 'alpha' with respect to the origin
%-----

num_o = 9; % number of orientations
num_d = 10; % number of distances
K = num_o*num_d; % total number of test reflector positions

or_ax = linspace(0,2*pi,num_o); % the orientation ranges...
                                   % ...from 0 to 2*pi

min_d = 1;
max_d = 4;
dist_ax = linspace(min_d,max_d,num_d); % the distance ranges...
                                           % ...from 'min_d' to 'max_d'

% grid definition
[grid_o,grid_d] = meshgrid(or_ax,dist_ax);

```

```

pts = [grid_o(:) grid_d(:)];

% conversion of the reflector lines in cartesian form, i.e. in the
% parameter vector [l_1 l_2 l_3] defining the line of equation
%  $l_1x + l_2y + l_3 = 0$ .
% Notice that we force  $l_3 = 1$ , since we no test reflector lines pass
% trough the origin.

ref_line = zeros(K,3);
for k=1:K
    rho = pts(k,2);
    alpha = pts(k,1);
    ref_line(k,:) = [cos(alpha) sin(alpha) -rho];
    ref_line(k,:) = ref_line(k,+)/ref_line(k,3); % force l3 = 1
end

% We now define the evaluation points. The variables to be estimated
% are  $l_1$  and  $l_2$ , thus  $M = 2$ 

ev_points = ref_line(:,1:2); % a [K x 2] matrix

%% 3. definition of the observations

% The theoretical observations are the euclidean distances between the
% microphones and the mirror source generated by the real one
% -----

% source position
src_pos = [0 0]; % the source is in (0,0);

% microphone positions (N sensors on a circle):
N = 5; % number of microphones
angs = linspace(0,2*pi,N+1)';
angs = angs(1:end-1);
r = 0.3;
mic_pos = r*[cos(angs) sin(angs)]; % mic. positions

% calculate the noise-free reflective distances
observations = zeros(N,K);

for k = 1:K

    % mirror source position
    l1 = ev_points(k,1);
    l2 = ev_points(k,2);

    xs = src_pos(1);
    ys = src_pos(2);

    xr = -(l1^2*xs+2*l1+2*l1*l2*ys-l2^2*xs)/(l1^2 + l2^2);
    yr = (l1^2*ys-2*l1*l2*xs-2*l2-l2^2*ys)/(l1^2 + l2^2);

    mir_src = [xr yr];

```

```

        % length of the reflective paths
        for n = 1:N
            observations(n,k) = norm(mir_src-mic_pos(n,:));
        end
    end
end

%% 4. definition of params

% embed the microphone positions in params:
params.mic_pos = mic_pos;

%% 5 .definition of the estimation problem

prob = EPT_problem(cost_function_hdl, ev_points, observations, params);

%% 6. definition of the observation covariance

sig = 0.01;          % standard deviation of distance error
Pc = sig^2*eye(N); % diagonal matrix

%% 7. prediction of the covariance matrix

Px = EPT_cov(prob,Pc);

```

Although the comments inserted in the code should be sufficient for its comprehension, it may be helpful to discuss some implementation choices:

- **definition of the evaluation points (step 2):** in order to give a physical interpretation to the reflector position, the reflector lines are expressed in polar coordinates, i.e. in term of their distance ρ and orientation α with respect to the origin (see Figure 5). The evaluation points are then converted into the Cartesian representation using the relationship

$$\begin{cases} l_1 = \cos \alpha \\ l_2 = \sin \alpha \\ l_3 = -\rho \end{cases} ; \quad (12)$$

- **definition of the observations (step 3):** the observations correspond to the length of the N reflective paths. Their values are computed as the distance between the microphones and the mirror source position, obtained as the specular reflection of the real source over each reflector line.

The remainder of the code simply rearranges and plots the results:

```

% Convert the results from cartesian paramaters
% to polar (distance/orientation) parameters

Pp = zeros(2,2,K);          % covariance matrices of the estimated
                             % polar parameters

P_std_dev = zeros(2,K); % standard deviation of the

```

```

                                % estimated polar parameters

% Compute covariance and standard deviation
for k = 1:K

    l1 = ev_points(k,1);
    l2 = ev_points(k,2);
    l3 = 1;

    nr2 = l1^2 + l2^2;
    T = [-l1/(nr2^(3/2)), -l2/(nr2^(3/2));...
         -l2/nr2          , l1/nr2          ];

    Pp(:,:,k) = T*Px(:,:,k)*T';          % covariance matrix

    P_std_dev(1,k) = sqrt(Pp(1,1,k)); % standard deviation
    P_std_dev(2,k) = sqrt(Pp(2,2,k));

end

% Plot results (standard deviation)
std_o = reshape(P_std_dev(1,:),num_o,num_d);
std_d = reshape(P_std_dev(2,:),num_o,num_d);

hfig = figure;
set(hfig,'Units', 'Normalized', 'OuterPosition', [0 0 1 1]);

subplot(1,2,1);
colormap('gray');
imagesc([min_d max_d],[0 360],std_o*180/pi);colorbar;
xlabel('Distance [m]','interpreter','tex','fontsize',15);
ylabel('Orientation [deg]','interpreter','tex','fontsize',15);
set(gca,'xtick',[1 2 3 4]);
set(gca,'ytick',[0 90 180 270 360]);
set(gca,'fontname','times new roman','fontsize',15)
title('Std. dev. of orientation error [deg]','interpreter','tex','fontsize',15);
axis xy; axis square

subplot(1,2,2);
colormap('gray');
imagesc([min_d max_d],[0 360],std_d);colorbar;
xlabel('Distance [m]','interpreter','tex','fontsize',15);
ylabel('Orientation [deg]','interpreter','tex','fontsize',15);
set(gca,'xtick',[1 2 3 4]);
set(gca,'ytick',[0 90 180 270 360]);
set(gca,'fontname','times new roman','fontsize',15)
title('Std. dev. of distance error [m]','interpreter','tex','fontsize',15);
axis xy; axis square

```

In particular, it is important to notice that:

- in order to bring back to the polar representation (i.e., from the Cartesian coordinates (l_1, l_2) to the polar pair (ρ, α)), we need to invert eq.(12). First, we force $l_3 = 1$ by dividing all the

terms by $-\rho$, so that

$$\begin{cases} l_1 &= -\frac{1}{\rho} \cos \alpha \\ l_2 &= -\frac{1}{\rho} \sin \alpha \\ l_3 &= 1 \end{cases} .$$

Consequently, we obtain

$$\begin{cases} \rho &= \frac{1}{\sqrt{l_1^2 + l_2^2}} \\ \alpha &= \arctan\left(\frac{l_2}{l_1}\right) \end{cases} . \quad (13)$$

- the covariance matrix of the estimation Σ_1 (denoted with \mathbf{Px} in the code) is converted into the covariance $\Sigma_{\rho,\alpha}$ relative to the polar representation of the estimated variables:

$$\Sigma_{\rho,\alpha} = \mathbf{T} \Sigma_1 \mathbf{T}^T, \quad \mathbf{T} = \begin{bmatrix} \frac{-l_1}{(l_1^2 + l_2^2)^{\frac{3}{2}}} & \frac{-l_2}{(l_1^2 + l_2^2)^{\frac{3}{2}}} \\ \frac{-l_2}{l_1^2 + l_2^2} & \frac{l_1}{l_1^2 + l_2^2} \end{bmatrix} . \quad (14)$$

Figure 6 shows the result produced by the example discussed above. In particular, it implements the scenario in Figure 11, where the reflector are selected from a grid of 90 test reflector lines. The standard deviation of the measurement error is 1 cm.

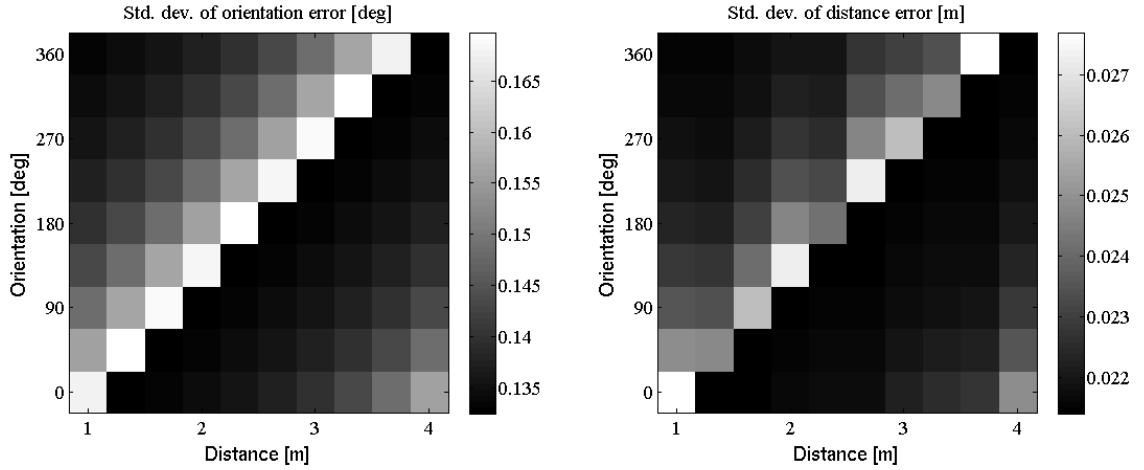


Figure 6: Error propagation analysis relative to the reflector localization.

As a final remark, notice that the results reported in [5] can be replicated using a larger grid of test reflectors on the same setup, posing `num_o=90` and `num_d=100` at step 2 of the problem definition.

References

- [1] M. Compagnoni, P. Bestagini, F. Antonacci, A. Sarti, and S. Tubaro, “Localization of acoustic sources through the fitting of propagation cones using multiple independent arrays,” *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 7, pp. 1964–1975, sept. 2012.
- [2] R. Gilmore, *Catastrophe theory for scientists and engineers*. Dover Publications, Inc., 1993, reprint of the 1981 original.
- [3] H.-C. So, “On linear least squares approach for phase estimation of real sinusoidal signals,” *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E88-A, pp. 3654–3657, December 2005. [Online]. Available: <http://dx.doi.org/10.1093/ietfec/e88-a.12.3654>
- [4] A. Beck, P. Stoica, and J. Li, “Exact and approximate solutions of source localization problems,” *Signal Processing, IEEE Transactions on*, vol. 56, no. 5, pp. 1770–1778, May 2008.
- [5] A. Canclini, F. Antonacci, M. R. P. Thomas, J. Filos, A. Sarti, P. A. Naylor, and S. Tubaro, “Exact localization of acoustic reflectors from quadratic constraints,” in *Applications of Signal Processing to Audio and Acoustics, 2011. WASPAA '11. IEEE Workshop on*, oct. 2011.
- [6] B. Ochoa and S. Belongie, “Covariance propagation for guided matching.”