radar-ambiguity-calculator Documentation

Release 0.1

Daniel Kastinen & Felipe Betancourt

December 11, 2018

CONTENTS:

CHAPTER ONE

INTRODUCTION

An ambiguity problem arises then determining the position and motion of objects with a radar system. The ambiguity problem is translated in the fact that different Direction of Arrival (DOA) can lead to the same response. In the paper *Determining all ambiguities in direction of arrival measured by radar systems* by Daniel Kastinen (http://www.ursi.org/content/RSB/RSB_365_2018_06.pdf), a mathematical framework and practical method to find all ambiguities in any multichannel system is described. The new formulation allows for an efficient implementation using the numerical Moore-Penrose inverse to find all ambiguities and approximate ambiguities.

Here two main functions composed by other several functions are developed to implement the algorithm proposed and retrieve solutions. The *ambiguity_calculator.py* file contains a function that by entering the name of a listed radar configuration, described in *radarconf.py* and operating frequency is able to calculate all ambiguities of a radar system and saves the results into an HDF5 file for later processing. *plots_generator.py* contains a function that generates the solution and plots for a certain radar configuration, with a certain operating frequency and for a given signal wave (described as a wave vector). All of the use user defined functions in *functions.py*.

This algorithm will help scientist to account on radar ambiguities in applications where determining DOA is critical, like is the cse of interferometry, which is used to calculate precise meteoroid trajectories and orbits.

Written and tested in Python 3.7.

CHAPTER

TWO

TOOL FOR RADAR AMBIGUITY SOLUTION

`ambiguity_calculator.`**`ambiguities_calculate`**`(`*radar_name*, *frequency*`)`

Calculates the solution for the radar ambiguity problem by implementing developed by Daniel Kastinen in his paper "Determining all ambiguities in direction of arrival measured by radar systems". The output of the calculations is summarized in a .h5 file with HDF5 format and saved into the /processed_data folder. Also a .log file is generated with a summary of the calculation process.

Parameters

- **`radar_name`** – Name of the radar to be studied out of a list of defined configurations.
- **`frequency`** – Operating frequency [MHz]

CHAPTER

THREE

TOOL FOR GENERATING PLOTS

plots_generator.**generate_plots**(*radar_name*, *frequency*, *elevation*, *azimuth*)

Import results summary from ambiguity_calculator algorithm and plot results for a DOA given by azimuth and elevation angles.

Parameters

- **radar_name** – choose radar configuration
- **elevation** – DOA elevation angle [º]
- **azimuth** – DOA azimuth angle [º]

CHAPTER

FOUR

LIST OF FUNCTIONS USED DURING THE CALCULATIONS

`functions.`**`R_cal`**(*sensor_groups*, *xycoords*)

Parameters

- **`sensor_groups`** – how many sensor groups are there in the radar configuration. Do not count on the one located at the origin
- **`xycoords`** – coordinates of the subgroups centers in wave lengths

Return R subgroup phase center

`functions.`**`explicit`**(*intersection_line*, *intersections_ind*, *cap_intersections_of_slines*, *xy*, *k0*)

Calculation of ambiguities

Parameters

- **`intersection_line`** – matrix of intersection lines as columns
- **`intersections_ind`** – valid intersection indexes
- **`cap_intersections_of_slines`** – cap?
- **`xy`** – xy coordinates of radar subgroups
- **`k0`** – signal wave vector

Returns

- **ambiguity_distances_explicit** - ambiguity distances
- **ambiguity_normal_explicit** -
- **k_finds** -

`functions.`**`intersections_cal`**(*pinv_norm*, *PERMS_J*, *intersection_line*, *R*, ***kwargs*)

Given that the Moore-Penrose solution gives a solution for any case. It is necessary to choose the ones whose error is below a given tolerance value.

Parameters

- **`pinv_norm`** – distance from real b_vector and the one obtained by the moore_penrose solution
- **`PERMS_J`** – Valid combinations
- **`intersection_line`** – matrix of intersection lines
- **`R`** – subgroup phase center
- **`kwargs`** – if 'norm' an extra condition will be applied and is that if pinv_norm is greater than zero, it will disregarded.

Return intersections dictionary with the so far valid intersection combinations.

`functions.`**`k0_cal`**`(`*el0*, *az0*`)`

Calculation of wave vector defined defined in paper.

Parameters

- **`el0`** – elevation angle [°]
- **`az0`** – azimuth angle [°]

`functions.`**`lambda_cal`**`(`*frequency*`)`

Calculate the wave length of electromagnetic radiation given its frequency.

Parameters **`frequency`** – Frequency of electromagnetic radiation [MHz]

Return wavelength wave length of electromagnetic radiation [m]

`functions.`**`linCoeff_cal`**`(`*R*`)`

Calculate linear coefficients given R.

Parameters **`R`** – matrix of subgroup phase centers

`functions.`**`mooore_penrose_solution`**`(`*W*, *b*`)`

Calculate the difference that produces the use of the Moore-Penrose solution matrix to the algebraic equation

Parameters

- **`W`** – W matrix as described in paper
- **`b`** – b vector as described in paper

Return intersection_line matrix of intersection lines

Return pinv_norm difference after solution check

`functions.`**`mooore_penrose_solution_par`**`(`*W*, *b_set*, *pnum*, *niter*, *intersection_line_set*, *pinv_norm_set*`)`

Calculate the Moore-Penrose solution for each case making use of parallel threading to parallelize task and joining results. Calls the function moore_penrose_solution_ptr

Parameters

- **W** – Matrix representation of all the normal vector of the planes going through the intersection.
- **b_set** – Set ob b vectors for which the solution is going to be computed.
- **pnum** – Number of cores for parallelizing
- **niter** – number of permutations.
- **intersection_line_set** – initial array with zeros where the intersection lines will be allocated.
- **pinv_norm_set** – error in the MP solution approximation

`functions.`**`mooore_penrose_solution_ptr`**(*W*, *Wpinv*, *b_set*, *intersection_line_set*, *pinv_norm_set*, *ind_range*)

Calculate the difference tha produces the use of the Moore-Penrose solution matrix to the algebraic equation in paper.

Parameters

- **W** – W matrix
- **Wpinv** – Pseudo-inverser of W matrix
- **b_set** – set of b vctors
- **intersection_line_set** – matrix of intersection lines
- **pinv_norm_set** – matrix of pinv_norm vectors
- **ind_range** – index

`functions.`**`nvec_j`**(*j*, *R*)

Normalized vector normal to plane j. Each plane is given as a column in R.

Parameters

- **j** – index
- **R** – subgroup phase center

`functions.`**`p0_jk`**(*j*, *k*, *R*, *n0*, *K*)

Displacement point

Parameters

- **j** –
- **k** –
- **R** –
- **n0** –

- **`K`** –

`functions.`**`permutations_create`**(*permutations_base*, *intersections_ind*, *k_length*, *permutation_index*)

Create all possible permutations by combining current permutation combinations with valid indexes and new

Parameters

- **`permutations_base`** – set of combinations created so far
- **`intersections_ind`** – indexes of valid combinations
- **`k_length`** – set of elements to create new permutations, k
- **`permutation_index`** – index of current permutation. Chooses k_j

`functions.`**`slines_intersections`**(*k0*, *intersections_ind*, *intersection_line*, *cut-off_ph_ang*)

Find all s-lines that intersect with the cap by range check.

Parameters

- **`k0`** – wave vector
- **`intersections_ind`** – indexes of valid combinations
- **`intersection_line`** – intersection lines matrix
- **`cutoff_ph_ang`** – cut-off angle

CHAPTER FIVE

TESTED RADAR CONFIGURATIONS

The user can test the algorithms with these radar configurations.

`radarconf.`**`radar_conf`**`(`*radar_name*, *frequency*`)`

Select radar radar configuration after name.

Parameters

- **`radar_name`** – input by user
- **`frequency`** – frequency at which the radar array is being operated [Mhz]

Return lambda0 wave lenght corresponding to radar frequency [m]

Return xycoords coordinates of subarray centers w.r.t. center of radar configuration in wavelengths.

CHAPTER

SIX

EXAMPLE OF UTILIZATION AND EXPECTED OUTPUTS.

6.1 Solve the problem for a certain radar configuration

As an example take one of the radar configurations, **JONES** in this case, with a frequency of 31 MHz.

The coordinates of the subarray in wave lengths are

x	y
0	2
0	-2.5
-2	0
2.5	0
0	0

By running a python script with

```python
from ambiguity_calculator import ambiguities_calculate

ambiguities_calculate(radar_name='Ydist', frequency=31)
```

a HDF5 called JONES.h5 containing the calculation results is generated in the folder ../processed_data/JONES.

JONES.h5 contains several items, data sets. Organized between two main HDF5 groups.

- root:
  - trivial_calculations: holds results from calculations which are straight forward and whose results can be used for to track the results.
    - *sensor_groups*
    - *subgroup_phase_center*
    - *linear_coefficients*

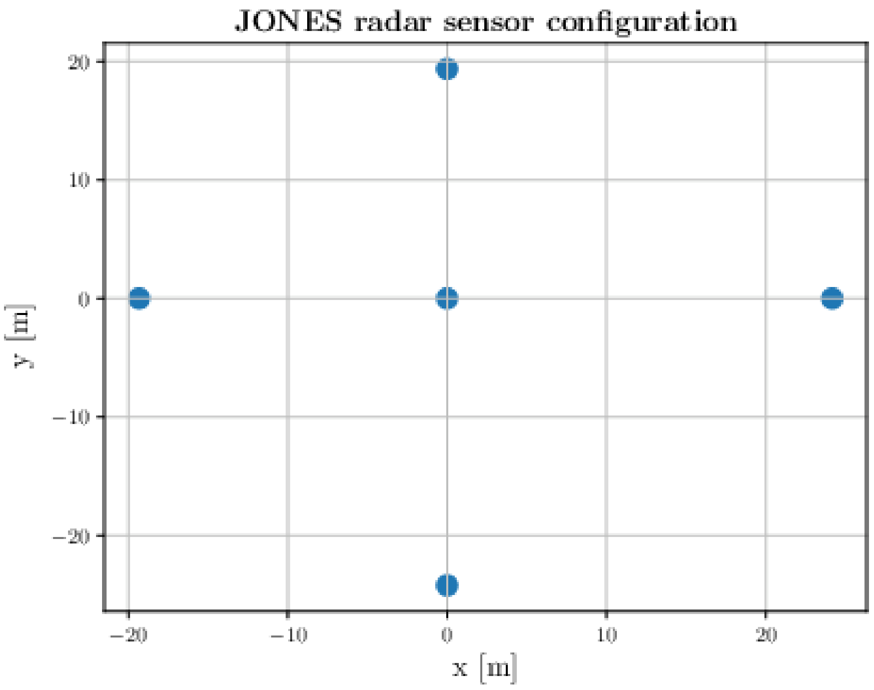
    - *base_numbers*
    - *k_length*
  - results_permutations: holds the results of the permutations and ambiguities.
    - *intersections_integers_complete*
    - *ambiguity_distances_INT_FORM_MAT*
    - *ambiguity_distances_INT_FORM_mean*
    - *ambiguity_distances_WAVE_FORM_MAT*
    - *ambiguity_distances_WAVE_FORM*
    - *intersection_line*
    - *survivors*

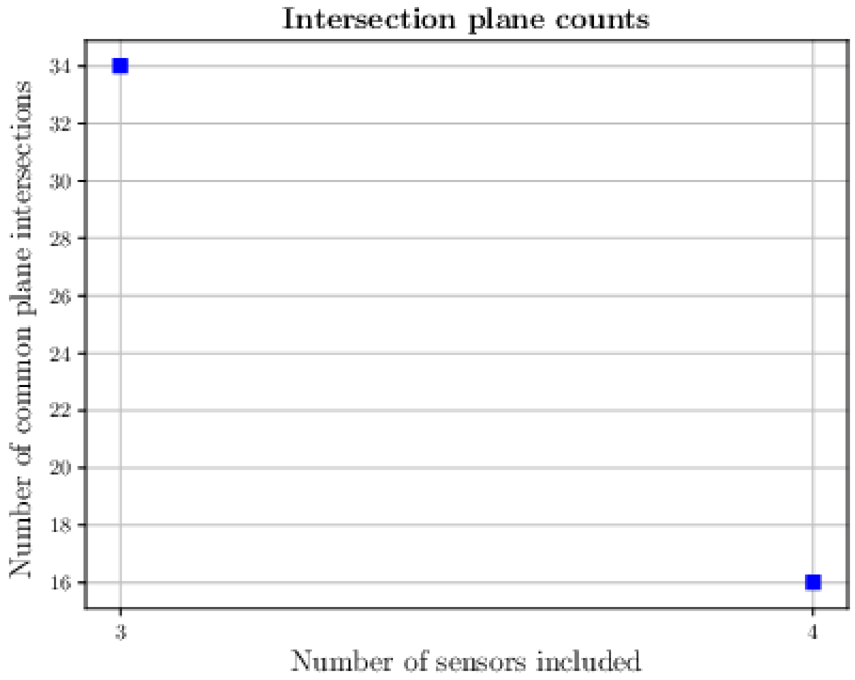
6.2 Use the results to see the ambiguities for a DOA.

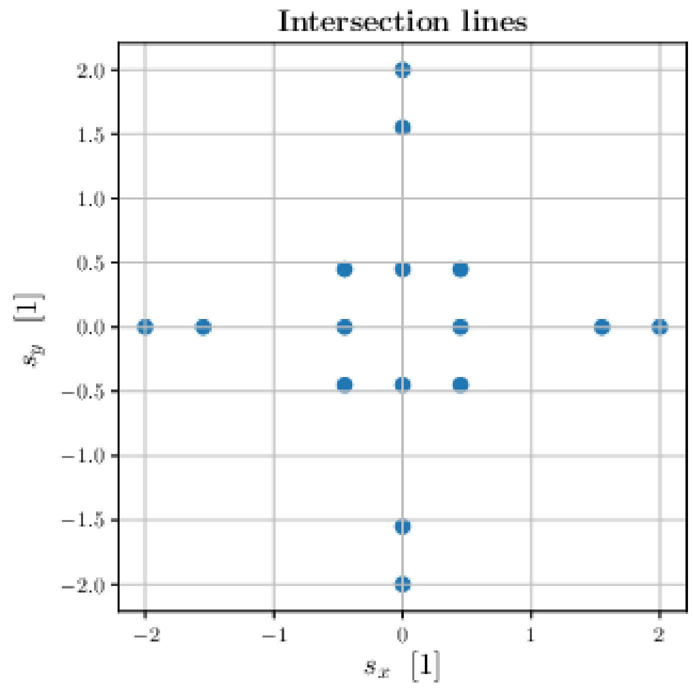
Continuing with JONES radar configuration, if one runs another script with

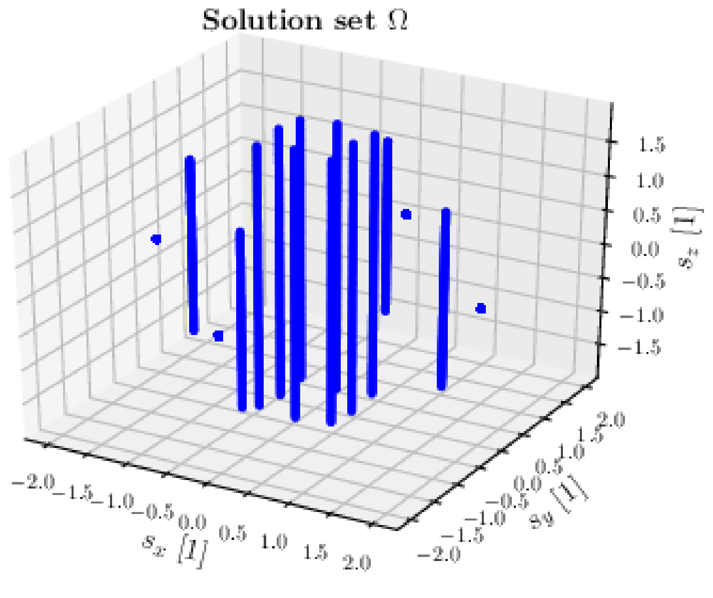
```
from plots_generator import generate_plots

generate_plots(radar_name='JONES', frequency=31, elevation=50,␣
↪azimuth=270)
```

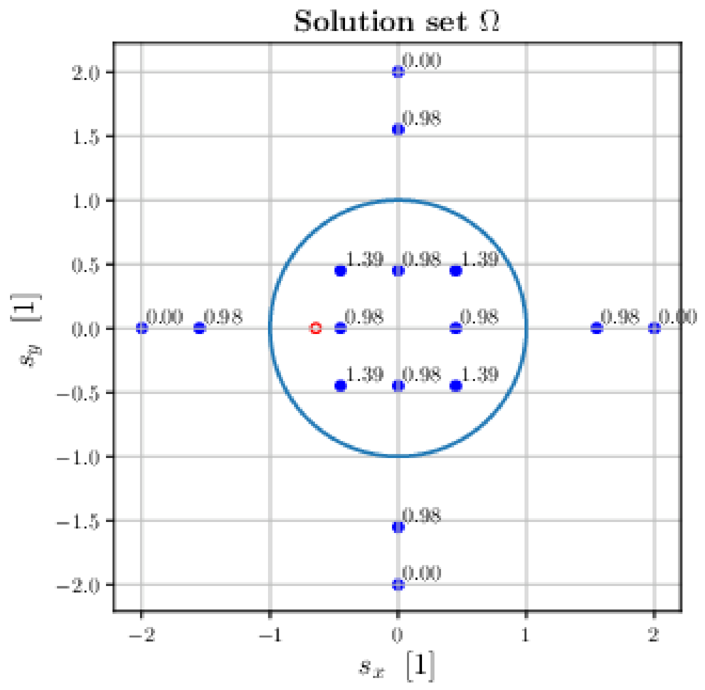
the same HDF5 file will the imported and plots with the results will be generated.

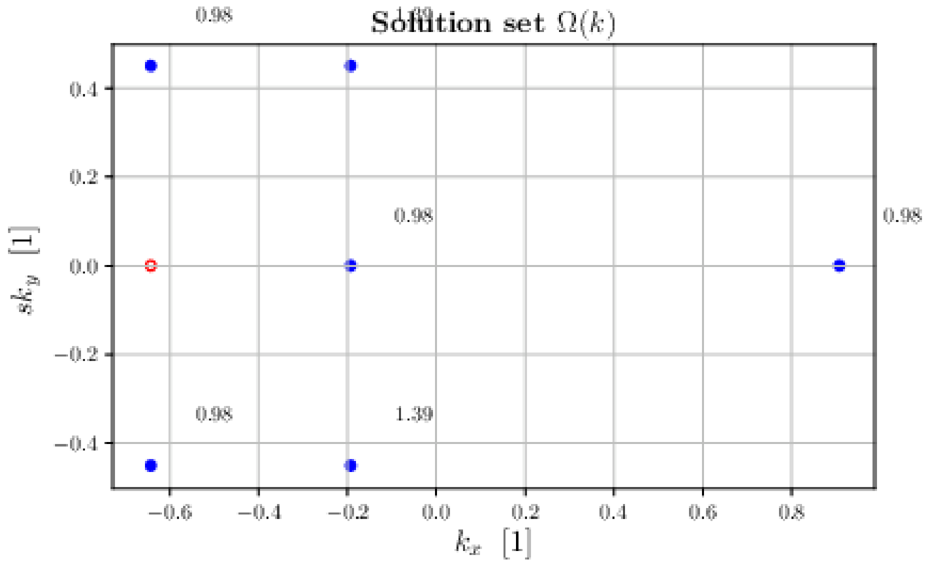

JONES radar sensor configuration
y [m]
x [m]
20
10
0
−10
−20
−20
−10
0
10
20


Intersection plane counts
Number of common plane intersections
Number of sensors included
34
32
30
28
26
24
22
20
18
16
3
4


Intersection lines
s_y [1]
s_x [1]
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2


Solution set Ω
s_x [1]
s_y [1]
s_z [1]
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Solution set Ω
s_x [1]
s_y [1]
0.00
0.98
1.39


Solution set $\Omega(k)$
k_x [1]
sk_y [1]
0.98
1.39

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX