# Quick Introduction

There are two main folders in this repository: 1) <code\_X> and 2) <data>. <code\_X> folder can be as many as you like to run multiple Matlab instances within each <code\_X> folder.

## Data Folder

In the data folder, random input files are generated and kept. The random input files are beamforming vectors (i.e., transmit, relay, and receive beamforming vectors), and channels (i.e., transmit - relay, relay - receiver, and transmit - receiver channels). For instance, same set of input files can be fed into the benchmarking algorithms.

This approach can be useful for two main reasons:

1. Benchmarking can be achieved with lesser number of Monte Carlo simulations.
2. Debugging and scrutinizing of algorithms are more convenient.

Nevertheless, the line of AI\_v05.m, where Channels\_v01 is called, can be modified if the user does not wish to work with input files.

**Generator\_v02.m:** This function generates the random input files.

**NetworkList.xlsx:**

* In this excel file, the parameters of networks to be tested are entered. For instance, *K*3-*M*10-*N*10-*R*3 network at 12-12 dB (*Tx*=12 dB - transmitter SNR, *R*=12 dB - relay SNR).
* *MC* in the excel file stands for the maximum Monte Carlo runs and *MCT* stands for the Monte Carlo target. Sometimes, some Monte Carlo channels are skipped and the next set of Monte Carlo channels are tested as explained in the paper. For instance, for a Monte Carlo run, the algorithm might be slow in convergence, which can be time consuming for the overall simulation process. Thus, such Monte Carlo runs can be skipped. Therefore, *MC* number should be chosen safely larger than the *MCT* number in the excel file.
* *Runner* in the excel file indicates the Matlab instances. By duplicating the code folder as <code\_01>, <code\_02>, …, <code\_X>, multiple Matlab instances can be executed to test multiple networks. The important point here is to modify the name of the Runners\_v03\_X.m file according to the <code\_X> folder. For instance, *Runner* 3 in the excel file will be executed by Runners\_v03\_03.m file under the <code\_03> folder.

**runme\_v05.xlsx:**

* In this excel file, the parameters of the networks to be tested are pulled from the NetworkList.xlsx automatically (i.e., the user does not need to enter the parameters).
* However, the user needs to enter other network parameters in the runme\_v05.xlsx file, e.g., *faultlimit* (to be explained later). It is highly advised that different sheets of runme\_v05.xlsx file are used by different Matlab instances. For this reason, 6 sheets are created in the excel file as code\_01 … code\_06.

The reason for using NetworkList.xlsx and runme\_v05.xlsx files is as follows: There are two main categories of parameters in the simulations. The first category is relevant to network architecture including the number of nodes, antennas, SNRs, and Monte Carlo numbers. The second category is relevant to algorithmic parameters, which can significantly change the outputs, including *faultlimit* andstep sizes. The same network architecture can be tested by different algorithms, e.g., ADMM and ADAL, and different algorithmic parameters, e.g., *faultlimit*=3, 10. By creating multiple sheets in runme\_v05.xlsx that are called by multiple Matlab instances, these different algorithms and algorithmic parameters can be tested.

## <code\_X> Folder

In the <code\_X> folder, there are 3 (+1) main files and 1 main folder.

**Runners\_v03\_X.m:** This is the main file of the Matlab codes. As mentioned earlier, the X values in <code\_X> folder, Runners\_v03\_X.m file, and *Runner* X in NetworkList.xlsx must match. Runners\_v03\_X.m file reads the NetworkList.xlsx file, writes the network parameters into the runme\_v05.xlsx file, and also calls the Main\_AI\_v03.m file.

**Main\_AI\_v03.m:** This file reads all the parameters from the runme\_v05.xlsx file, checks the <data/Inputs> folder whether the inputs files exist, and calls the Generator\_v02.m file if the input files do not exist. Finally, it calls the AI\_v05.m (AI\_v05M.m) file.

**AI\_v05.m:** This file calls the main sub-functions of the proposed distributed algorithm in the paper. This file also serves as the flowchart (i.e., pseudocode) of the proposed algorithm, e.g., which matrices and variables should be updated when, and so on.

The file name Main\_AI indicates that it calls the sub-function AI. The file name AI is chosen such that different codes that correspond to the different algorithms can be differentiated. For instance, for the algorithm A, different receive filters can be tried, e.g., AI for MMSE filter and AII for max-SINR filter. Benchmarking algorithms can be named by different letters such as BX and CX. For instance, BI for ADAL algorithm, and CI for consensus-based primal-dual perturbation algorithm can be used.

**Info.xlsx:** In this excel file, modifications on the file versions can be noted for future reference.

**AI\_v05M.m:** As mentioned earlier, some Monte Carlo runs can be skipped for the sake of shorter simulation processes. Therefore, while benchmarking different algorithms, it is important to exclude the skipped Monte Carlo runs for all algorithms. For instance, if Monte Carlo runs 3, 8 and 6, 7 are skipped for ADMM and ADAL, respectively, then Monte Carlo runs 3, 6, 7, and 8 should be excluded from both ADMM and ADAL simulations. Otherwise, many Monte Carlo runs have to be executed which conflicts the purpose of using input files, i.e., input files help to benchmark with less number of Monte Carlo runs. The missed channels for each algorithm can be obtained from \*\_MissedChannels\_vX.log files. These files can be combined by the tools in the Excel software (see <code\_X/Files/MissedChannelsCombiner.xlsx>). Then the final list can be copy pasted to <data/AllMissedChannels.log> file. Finally, AI\_v05M.m (capital M stands for missed) is called instead of AI\_v05.m from the Main\_AI\_v04.m file.

Note that AI\_v05M.m is better used only for benchmarking of different algorithms. For instance, the same algorithm can miss different channels for different network architectures at different SNRs. AI\_v05M.m is not needed in such cases.

## Sample Executions

In this example, two algorithms are benchmarked for different network architectures at different SNRs. The missed channels for these two algorithms are listed as follows:

ADMM for K3-M10-N10-R3 at 12-12: 1 6 11

ADMM for K3-M10-N10-R3 at 21-21: 3

ADMM for K3-M8-N8-R4 at 12-12: 10

ADMM for K3-M8-N8-R4 at 21-21: 2

ADAL for K3-M10-N10-R3 at 12-12: 8 9

ADAL for K3-M10-N10-R3 at 21-21: 2 7

ADAL for K3-M8-N8-R4 at 12-12: 5 10

ADAL for K3-M8-N8-R4 at 21-21: 4

The following channels must be excluded from both algorithms:

ADMM & ADAL for K3-M10-N10-R3 at 12-12: 1 6 8 9 11

ADMM & ADAL for K3-M10-N10-R3 at 21-21: 2 3 7

ADMM & ADAL for K3-M8-N8-R4 at 12-12: 5 10

ADMM & ADAL for K3-M8-N8-R4 at 21-21: 2 4

For the above common missed channels for both algorithm, AllMissedChannels.log file must be updated and AI\_v05M.m must be called as explained earlier.

## Sample Executions

In this example, for the same algorithm, different network architectures at different SNRs are tested.

**Step 1:** In the NetworkList.xlsx file, enter the following to test three network architectures by three Matlab instances.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| K | M | N | R | Tx dB | R dB | MC | MCT | Runners |
| 3 | 10 | 10 | 3 | 12 | 12 | 50 | 10 | 1 |
| 3 | 8 | 8 | 4 | 12 | 12 | 50 | 10 | 2 |
| 3 | 5 | 8 | 3 | 12 | 12 | 50 | 10 | 3 |

**Step 2:** Copy Paste the <code\_01> folder as <code\_02> and <code\_03> folders. Rename the Runners\_v03\_X.m files in these folders as Runners\_v03\_01.m, Runners\_v03\_02.m, and Runners\_v03\_03.m accordingly.

**Step 3:** Enter the algorithmic parameters into the runme\_v05.xlsx file. As noted earlier, it is advised to use different sheets of runme\_v05.xlsx file for different Matlab instances.

**Step 4:** Before executing the Runners\_v03\_X.m files on different Matlab instances, paths for one of the Matlab instance need to be set (preferably, <code\_01>):

<code\_X\library>

<code\_X\library\generic>

<code\_X\library\specific>

The path for the common <data> folder must also be set.

## Important Notes

1. During the Matlab executions, open the excel files by using the LibreOffice instead of Microsoft Excel software.
2. Assume that the *K*3-*M*10-*N*10-*R*3 network is to be tested at two different SNRs by two different Matlab instances. Also, assume that the input files for the *K*3-*M*10-*N*10-*R*3 network have not been generated earlier. Do not execute the second Matlab instance until the first Matlab instance finishes with generating the input files.
3. Remove any file in the folders that will not be used, e.g., remove the old versions of the codes. For instance, remove AI\_v01.m file if there is a newer version AI\_v02.m file.

# List of Some MATLAB Variables

The variables are listed in alphabetical order. Cited equation numbers are from the IEEE Access paper “Distributed Multi-Stream Beamforming in MIMO Multi-Relay Interference Networks” by C. M. Yetis and R. Y. Chang in IEEE Access, vol. 7, pp. 7535–7554, Jan. 2019.

## B

**B:** Total number of streams in the network. See in Remark 1

## C

**checkwhilei:** While loop is terminated when each stream achieves the SINR target, i.e., when $\Delta\_{k,l}\leq\Delta\_{k,l}^\text{max}, \forall k,l$ . To lower the iteration number, checkwhile(*k*)=1 can be set when , where . In other words, the average deviation of a user’s streams is less than a threshold.

**Cklrin:** Cklrin matrix stores the $\mathbf{C}\_{klrin}$ matrixes in (58), see Appendix B of the IEEE Access paper, as follows:

## D

**d:** Number of streams per user. See $d\_i$ in the paragraph above (1). Note that in the Matlab codes, only the dparameter cannot be changed and it is fixed to 2. In fact, codes are almost ready for the d parameter to be modified.

**Delta:** Thresholdof achievedSINR deviation of a stream from its SINR target, i.e., see $\Delta\_{kl}^\text{max}$ in Step 7 of Algorithm 1

**DeltaM:** DeltaM vector stores the sum of $\Delta\_{k,l}$ values (See step 6 in Algorithm 1) for each iteration. Algorithm is assumed to converge when $\Delta\_{k,l}\leq\Delta\_{k,l}^\text{max}, \forall k,l$

**DeltaMin:** Initialization of average value of deltas . Optimum beamforming vectors are updated when smaller average value of deltas (SINR target deviations) is obtained

**Drkl:** Drkl matrix stores the $\mathbf{D}\_{rkl}$ matrices, see the second equality in (44), as follows:

## F

**F:** Relay beamforming matrices. See $\mathbf{F}\_r$ in (3). F matrix stores the matrices as follows:

**faultlimit:** Maximum number of faults allowed during the fault monitoring window. See Section W of the list of MATLAB variables

## G

**G:** Channels between the relays and receivers. See $\mathbf{G}\_{kr}$ in (2). G matrix stores the matrices as follows:

## H

**Hpp:** Channels between the transmitters and relays. See $\mathbf{H}^{\prime\prime}\_{ri}$ in (2). Hpp matrix stores the matrices as follows:

**hYr:** hYr matrix stores the $\mathbf{Y}\_r$ matrices in (43) as follows:

## J

**J:** Channels between the transmitters and receivers. See $\mathbf{J}\_{ki}$ in (2). J matrix stores the matrices as follows

:

## K

**K/Kr:** Number of users (Number of transmitter and receiver pairs) – NL

## L

**lambdavF**: Lagrangian dual variable vector of the sum of auxiliary variables for the design of relay beamforming filters. See $\dot{\lambda}\_r^{k,l}$ in (47)

**lambdavU:** Lagrangian dual variable vector of the sum of auxiliary variables for the design of transmit beamforming filters. See $\lambda\_{k,l}$ in (19)

## M

**M/Mr:** Number of antennas at each transmitter and receiver – NL

**iterlimit:** Maximum number of iterations

**missed:** This vector stores information about the missed channels.

missed(1): Total number of missed channels, i.e., missed(1)=sum(missed(2:end))

missed(2): Number of non-converging channels (i.e., too many oscillations during the fault monitoring window),

missed(3) slow converging channels (i.e., the algorithm does not converge within the allowed maximum number of iterations),

missed(4): The algorithm converges. However, the resultant powers of transmit (U) beamforming matrices violate the maximum power limits.

missed(5): The algorithm converges. However, the resultant powers of relay (F) beamforming matrices violate the maximum power limits.

**MonteCarlo/MC:** The number of Monte Carlo runs - NL

**MonteCarloT/MCT:** and The number of Monte Carlo runs – target - NL

Due to the missed channels MonteCarlo variable is set to be a higher value than the MonteCarloT (Monte Carlo target) variable. For instance, if you want to test 20 Monte Carlo channels, you should set MonteCarloT=20 and a safe MonteCarlo>MonteCarloT value, such as MonteCarlo=30. Therefore, in case, 10 channels are missed, 20 channels can still be tested

**muvF**: ADMM dual variable of the zetavF auxiliary variable for the design of relay beamforming filters, i.e., $\dot{\mu}\_r^{k,l}$

**muvU:** ADMM dual variable of the zetavU auxiliary variable for the design of transmit beamforming filters. See $\mu\_{k,l}$ in (24b)

**munvF**: ADMM dual variable of the zetanvF auxiliary variable for the design of relay beamforming filters, i.e., $\dot{\mu}\_r^{k,l\backprime}$

**munvU:** ADMM dual variable of the zetanvU auxiliary variable for the design of transmit beamforming filters. See $\mu\_{k,l}^\backprime$ in (24c)

## N

**N/Nr:** Number of antennas at each relay – NL

## P

**PTot:** PTot matrix stores the total power consumption of transmitters and relays per stream $\text{tr}\mathbf{X}\_{k,l}\mathbf{R}\_{.k}$ in (13a) written in terms of transmit filters

**PPdBTxRx/TxdB:** SNR between a transmitter and a receiver in dB - NL

**PPdBRRx/RdB:** SNR between a relay and a receiver in dB – NL

**Pr:** Pr vector stores the relay transmission powers. See $p\_r$ in (3)

**PTx:** Pt matrix stores the power consumption of transmitters per stream $\text{tr}\mathbf{X}\_{k,l}$

## R

**R:** Number of relays – NL

**Rbkl:** Rbkl matrix stores the $\mathbf{R}\_{k,l}^i$ matrices in (13b) as follows: . Rbkl matrix stores both the desired $\mathbf{R}\_{k,l}^k$ and interference $\mathbf{R}\_{k,l}^j$ terms

**Rdk:** Rdk matrix stores the $\mathbf{R}\_{.k}$ matrices in (13a) as follows:

**rhoF:** ADMM Lagrangian dual update step size for relay filter optimization. See $\dot{\rho}$ in (47)

**rhocF:**  Conventional Lagrangian dual update step size for relay filter optimization, i.e., $dot{\rho}\_c$

**rhoU:** ADMM Lagrangian dual update step size for transmit filter optimization. See $\dot{\rho}$ in (19)

**rhocU:**  Conventional Lagrangian dual update step size for transmit filter optimization. See $dot{\rho}\_c$ in (24b)

## S

**sigma2k1:** Noise variance at receiver k at time instance 1. See $\sigma\_k^2(1)$ in (9)

**sigma2k2:** Noise variance at receiver k at time instance 2. See $\sigma\_k^2(2)$ in (9)

**sigma2kl:** sigma2kl is the noise vector. See $\sigma\_{n\_{k,l}}^2$ two lines above (9)

**sigma2r:** Noise variance at relay r. See $\sigma\_r^2$ in (3)

**sinr:** sinr matrix stores the SINR values of each stream as follows:

**stepi:** Total number of steps (iterations) needed by the algorithm to converge

## T

**tCklrin:** tCklrin matrix stores the $\tilde{C}\_{klrin}$ matrixes in (43). See Appendix B of the IEEE Access paper. tCklrin matrix stores both the desired $\tilde{C}\_{klrkl}$ and interference $\tilde{C}\_{klrjm}$ terms as follows:

## U

**U:** U matrix stores the transmit beamforming vectors. See $\mathbf{U}\_{i}$ in (1). U matrix stores the matrices as follows:

## X

**Xkl (Xklopt):** Xkl (Xklopt) matrix store the (optimum) covariance matrices of transmit beamforming vectors. See $\mathbf{X}\_{k,l}$ in (13a). Xkl matrix stores the matrices as follows:

## V

**V:** Concatenated receive beamforming vectors of the two time slots. See $\bar{\mathbf{V}}\_k\in\mathbb{C}^{2M\_k\times d\_k}$ above (8). V matrix stores the matrices as follows:

## W

**wmax:** Maximum value of the fault monitoring window

**wmin:** Minimum value of the fault monitoring window

**wstep:** Step size of the fault monitoring window

faultlimit, wmax, wmin, and wstep are fault monitoring window parameters. For some channels, convergence can be too slow, for instance, due to high number of oscillations. To detect these channels earlier and to save time, a fault monitoring window is used.

In fact, each of the algorithmic parameters can significantly alter the results for any algorithm, any network architecture at any SNR. For instance, see the results K3M5N8B6R3\_MC50\_T12R12\_v05\_faultlimit3.zip and K3M5N8B6R3\_MC50\_T12R12\_v05\_faultlimit10.zip

In general, *faultlimit* is advised to keep small such as 3 to early detect which Monte Carlo runs to skip. However, in some cases as given in the examples above, keeping *faultlimit* large such as 10 can be advantageous because 3 can be small in a way that many non-converging cases can occur.

## Z

**zetavF**: Auxiliary variables of the desired SINR plus noise terms for the design of relay beamforming filters. See $\dot{\zeta}\_r^{k,l}$ in (46c)

**zetavU:** Auxiliary variables of the desired SINR plus noise terms for the design of transmit beamforming filters. See $\zeta\_{k,l}$ in (17)

**zetanvF**: Auxiliary variables of the interference SINR terms for the design of relay beamforming filters. See $\dot{\zeta}\_r^{k,l\backprime}$ in (46d)

**zetanvU:** Auxiliary variables of the interference SINR terms for the design of transmit beamforming filters. See $\zeta\_{k,l}^\backprime$ in (17)

# MATLAB Functions by Their Calling Orders

The functions are listed in their calling orders.

**I - Runners\_v03\_X.m**

1 - warnings.m

2 - CodeFolder\_SheetName\_DataFolder

3 - NetworkList.xlsx (Excel file)

**II- Main\_AI\_v04.m**

1 - CodeFolder\_SheetName\_DataFolder

2 - runme\_v05.xlsx (Excel file)

3 - Generator\_v02.m

4 - InputFolder\_v01.m

**III- AI\_v05.m (AI\_v05M.m)**

1 - CodeVersion\_v01.m

2 - IOFiles\_v03.m

- (DataFolder\_v01.m for AI\_v05M.m)

3 - ExplanatoryFiles\_UFIterII\_v02

4 - Channels\_v01.m

5 - F\_hYr\_v02.m

6 - Xkl\_sinrx\_v02.m

7 - VarInits\_UFIterII\_v01.m

8 - Rdk\_Rbkl\_sigma2kl\_UFIterII\_v02.m

9 - U\_Xkl\_UFIterII\_v04.m

10 - sinrxNew\_v01.m

11 - Delta\_Logging\_v02.m

12 - NewVarsU\_v01.m

13 - tCklrin\_v01.m

14 - Drkl\_v02.m

15 - F\_hYr\_UFIterII\_v03.m

16 - NewVarsF\_UFIterII\_v01.m

17 - PTot\_PTx\_PR\_UFIterII\_v02.m

18 - Logging\_v04.m

- (IncreaseLoaders\_v01.m for AI\_v05M.m)

19 - Outputs\_UFIterII\_v03.m

1 - output\_v03.xlsx (Excel file)

2 - closexlsxfiles.m

3 - secs2hms.m

# MATLAB Functions by Their Locations

The functions are listed by the folders and ordered alphabetically.

## <code\_X> folder

1 - AI\_v05.m (AI\_v05M.m)

2 - Main\_AI\_v03.m

3 - output\_v03.xlsx (Excel file)

4 - Runners\_v03\_X.m

<library> folder

<generic> folder

1 - Channels\_v01.m

2 - closexlsxfiles.m

3 - CodeFolder\_SheetName\_DataFolder

4 - CodeVersion\_v01

- (DataFolder\_v01.m for AI\_v05M.m)

5 - Delta\_Logging\_v02.m

6 - InputFolder\_v01.m

7 - IOFiles\_v03.m

- (IncreaseLoaders\_v01.m for AI\_v05M.m)

8 - Logging\_v04.m

9 - NewVarsU\_v01.m

10 - OutputExcelFiles\_v02.m

11 - secs2hms.m

12 - sinrxNew\_v01.m

13 - warnings.m

<specific> folder

1 - Drkl\_v02.m

2 - ExplanatoryFiles\_UFIterII\_v02.m

3 - F\_hYr\_UFIterII\_v03.m

4 - F\_hYr\_v02.m

5 - NewVarsF\_UFIterII\_v01.m

6 - NewVarsU\_v01.m

7 - Outputs\_UFIterII\_v03.m

8 - PTot\_PTx\_PR\_UFIterII\_v02.m

9 - Rdk\_Rbkl\_sigma2kl\_UFIterII\_v02.m

10 - tCklrin\_v01.m

11 - U\_Xkl\_UFIterII\_v04.m

12 - VarInits\_UFIterII\_v01.m

13 - Xkl\_sinrx\_v02.m

## <data> folder

1 - Generator\_v02

2 - NetworkList.xlsx (Excel file)

3 - runme\_v05.xlsx (Excel file)