CCAMA: Software for solving the covariance completion problem using alternating minimization algorithm

Armin Zare and Mihailo R. Jovanović
Department of Electrical and Computer Engineering,
University of Minnesota, Minneapolis, MN 55455, USA
(Dated: April 19, 2016)

We provide a brief description of a MATLAB implementation of a customized alternating minimization algorithm considered for solving the covariance completion problem. Additional information about the examples, along with MATLAB source codes, can be found at:

http://www.ece.umn.edu/users/mihailo/software/ccama/

* http://www.ece.umn.edu/users/arminzare/arminzare@umn.edu
† http://www.ece.umn.edu/users/mihailo/mihailo@umn.edu
ccama.zip – contains all Matlab functions and problem data required to run CCAMA and reproduce all results reported in the IEEE TAC paper. These include m-files required for optimization, modeling, stochastic simulation, and plotting.

DESCRIPTION OF MATLAB FILES

• m-files

  ccama.m – customized algorithms for solving the covariance completion problem (CC). The user has the option to choose between AMA and ADMM solvers;

  run_ccama.m – explains how to run ccama for the mass-spring-damper system;

  linfilter.m – realization of the filter dynamics based on the solution to problem (CC);

  run_sim.m – explains how to run linear stochastic simulations to verify the modeling procedure;

  plots.m – plots figures shown in the paper.

A. Description of ccama.m

• MATLAB SYNTAX

  output = ccama(A,C,E,G,gamma,n,m,options);

• DESCRIPTION: The Matlab function ccama.m takes the problem data \(\{A, C, E, G, \gamma, n, m\}\) and input options and returns the solution to the covariance completion problem

\[
\begin{align*}
\minimize_{X, Z} & \quad - \log \det (X) + \gamma \|Z\|_* \\
\text{subject to} & \quad AX + XA^* + Z = 0 \\
& \quad (CXC^*) \circ E - G = 0
\end{align*}
\]

(CC)

where \(n\) and \(m\) denote the number of the states and the outputs, respectively.

• Input options allows users to specify the following parameters:

  – options.rho – initial step-size \(\rho\);
  – options.eps_prim – tolerance on primal constraints;
  – options.eps_dual – tolerance on duality gap;
  – options.maxiter – maximum number of iterations;
  – options.Xinit – feasible initial value for matrix \(X\);
  – options.Zinit – feasible initial value for matrix \(Z\);
  – options.Y1init – dual-feasible initial value for \(Y_1\);
  – options.Y2init – dual-feasible initial value for \(Y_2\);
  – options.method – method = 1, alternating minimization algorithm (default)
    method = 2, alternating direction method of multipliers.

• If options argument is omitted, the default values are set to:

  – options.rho = 10;
  – options.eps_prim = 1.e-5;
  – options.eps_dual = 1.e-4;
  – options.maxiter = 10^5;
  – \(X_{\text{init}} = \text{lyap}(A, I_{m \times m})\),
  – options.Xinit = \(X_{\text{init}}\);
  – options.Zinit = \(I_{m \times m}\);
\[ Y_{1,\text{init}} = \text{lyap}(A^*, -X_{\text{init}}), \]
\[ \text{options.Y1init} = \gamma \frac{Y_{1,\text{init}}}{\| Y_{1,\text{init}} \|_2}; \]
\[ \text{options.Y2init} = I_{n \times n}; \]
\[ \text{options.method} = 1. \]

- The output is a structured array that contains
  - `output.X` – optimal state covariance matrix \( X \) resulting from the optimization problem \( (CC) \);
  - `output.Z` – optimal forcing correlation matrix \( Z \) resulting from the optimization problem \( (CC) \);
  - `output.Y1` – optimal dual variable \( Y_1 \) resulting from the optimization problem \( (CC) \);
  - `output.Y2` – optimal dual variable \( Y_2 \) resulting from the optimization problem \( (CC) \);
  - `output.Jp` – value of the primal objective function at each step;
  - `output.Jd` – value of the dual objective function at each step;
  - `output.Rp` – primal residual at each step;
  - `output.dg` – duality gap at each step;
  - `output.steps` – number of steps required to solve \( (CC) \);
  - `output.time` – cumulative solve time per outer iteration (in seconds);
  - `output.flag` – flag = 0, iteration counter reaches its maximum
    flag = 1, problem \( (CC) \) is solved before iteration counter reaches its maximum.

B. Description of `run_ccama.m`

- Matlab script `run_ccama.m` allows users to:
  - choose the number of masses \( N \);
  - form the dynamic matrix \( A \);
  - form the filter dynamics that generate colored-in-time excitation for the mass-spring-damper system;
  - compute the true state covariance matrix of the mass-spring-damper system;
  - form the matrix \( G \) of available correlations and the structural identity \( E \);
  - choose the low-rank parameter \( \gamma \);
  - choose the optimization parameters through the structured array `options`;
  - call the customized AMA or ADMM algorithms by calling the function `ccama.m`.

C. Description of `linfilter.m`

- MATLAB SYNTAX
  
  \[ [Af,Bf,Cf,Df] = \text{linfilter}(A,X,Z); \]

- DESCRIPTION: Matlab function `linfilter.m` takes the linear dynamical generator \( A \) and the correlation matrices \( X \) and \( Z \) which results from the function `ccama.m` and returns the state-space realization of the linear filter which generates the suitable colored-in-time forcing into the linear dynamics.

D. Description of `run_sim.m`

- Matlab script `run_sim.m` performs linear stochastic simulations of the linear filter driven by band-limited white noise. This is done via the Matlab function `sim` which calls the Simulink model `sim.mdl.mdl`. \( T_s \) is the sampling period and \( t \) is the time vector. After running the simulations, the script averages over output measurements and computes one-point and two-point correlations. This allows the user to verify the modeling procedure by comparing the computed statistics with the available data in optimization problem \( (CC) \).

E. Description of `plots.m`

- Matlab script `plots.m` allows users to reproduce some of the figures shown in the IEEE TAC paper.