Table of Contents

PAPER

```
% Title: Multi-RIS-aided Wireless Systems: Statistical
 Characterization and Performance Analysis
% Authors : Tri Nhu Do, Georges Kaddoum, Thanh Luan Nguyen, Daniel
 Benevides da Costa, Zygmunt J. Haas
% Online: https://github.com/trinhudo/Multi-RIS
% Version: 12-Sep-2021

% Multi-cell multi-RIS with detailed phase-shift configuration
% --ERA scheme with inter-cell interference: all RISs participate
% --ORA scheme with inter-cell interference: only the best RIS
 participates
% --Simulation only

tic
clear all
close all
```

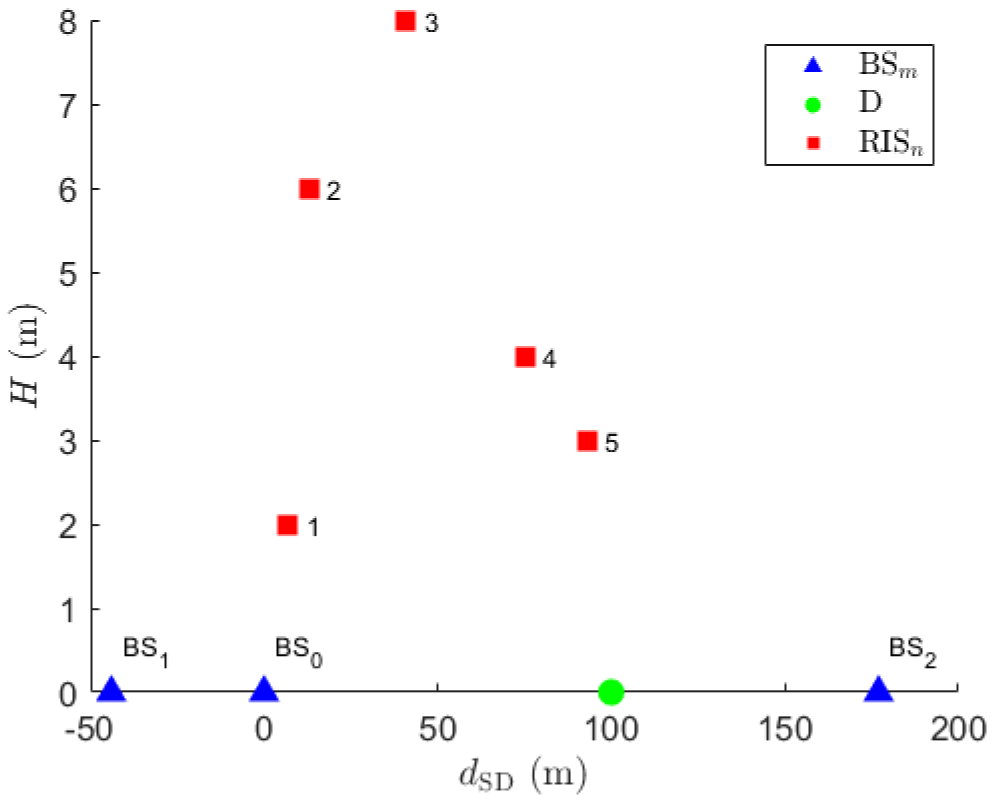
SETTING

```
sim_times = 1e5; % Number of simulation trails

R_th = 3; % Predefined target spectral efficiency [b/s/Hz]
SNR_th = 2^R_th-1; % Predefined SNR threshold

N_RIS = 5; % number of distributed RIS

L_single = 40; % Number of elements at each RIS

L = L_single*ones(1,N_RIS); % all RISs

kappa_nl = 1; % Amplitude reflection coefficient

% Network area
x_area_min = 0;
x_area_max = 100; % in meters
y_area_min = 0;
y_area_max = 10;
```

```
% Source location, base station 1 (BS1)
x_S1 = x_area_min;
y_S1 = y_area_min;

% BS2
x_S2 = x_area_min-44;
y_S2 = y_area_min;

% BS3
x_S3 = x_area_min+177;
y_S3 = y_area_min;

% Destination location
x_des = x_area_max;
y_des = y_area_min;

% % Random location setting
% x_RIS = x_area_min + (x_area_max-x_area_min)*rand(N_RIS, 1); %
 [num_RIS x 1] vector
% y_RIS = y_area_min + (y_area_max-y_area_min)*rand(N_RIS, 1);

%Location setting D1
x_RIS = [7; 13; 41; 75; 93];
y_RIS = [2; 6; 8; 4; 3];

% Compute location of nodes
pos_S1 = [x_S1, y_S1];
pos_S2 = [x_S2, y_S2];
pos_S3 = [x_S3, y_S3];

pos_des = [x_des, y_des];

pos_RIS = [x_RIS, y_RIS]; % [num_RIS x 2] matrix

% Compute distances
d_S1R = sqrt(sum((pos_S1 - pos_RIS).^2 , 2)); % [num_RIS x 1] vector
d_S2R = sqrt(sum((pos_S2 - pos_RIS).^2 , 2)); % [num_RIS x 1] vector
d_S3R = sqrt(sum((pos_S3 - pos_RIS).^2 , 2)); % [num_RIS x 1] vector

d_RD = sqrt(sum((pos_RIS - pos_des).^2 , 2));

d_S1D = sqrt(sum((pos_S1 - pos_des).^2 , 2));
d_S2D = sqrt(sum((pos_S2 - pos_des).^2 , 2));
d_S3D = sqrt(sum((pos_S3 - pos_des).^2 , 2));
```

NETWORK TOPOLOGY

```
figure;

scatter(x_S1, y_S1, 100, 'b^', 'filled'); hold on
scatter(x_des, y_des, 100, 'go', 'filled'); hold on
scatter(x_RIS, y_RIS, 100, 'rs', 'filled'); hold on
scatter(x_S2, y_S2, 100, 'b^', 'filled'); hold on
scatter(x_S3, y_S3, 100, 'b^', 'filled'); hold on

text(x_S1+3, y_S1+.5, 'BS_0')
text(x_S2+3, y_S2+.5, 'BS_1')
text(x_S3+3, y_S3+.5, 'BS_2')

for kk = 1:N_RIS
    text(x_RIS(kk)+5, y_RIS(kk), num2str(kk));
    hold on
end

xlabel('$d_{\rm SD}$ (m)', 'Interpreter', 'Latex')
ylabel('$H$ (m)', 'Interpreter', 'Latex')
legend('$\mathrm{BS}_m$', '$\rm D$', '$\mathrm{RIS}_n$',...
    'Interpreter', 'Latex',...
    'Location', 'best')

% set(gca, 'LooseInset', get(gca, 'TightInset')) % remove plot padding
set(gca,'fontsize',13);
hold off

% Path-loss model
% ---------------

% Carrier frequency (in GHz)
fc = 3; % GHz

% 3GPP Urban Micro in 3GPP TS 36.814, Mar. 2010.
% Note that x is measured in meter

% NLoS path-loss component based on distance
pathloss_NLOS = @(x) db2pow(-22.7 - 26*log10(fc) - 36.7*log10(x));

antenna_gain_S = db2pow(5); % Source antenna gain, dBi
antenna_gain_RIS = db2pow(5); % Gain of each element of a RIS, dBi
antenna_gain_D = db2pow(0); % Destination antenna gain, dBi

% Noise power and Transmit power P_S
% ----------------------------------

BW = 10e6; % Bandwidth : 10 MHz

noiseFiguredB = 10; % Noise figure (in dB)

% Compute the noise power in dBm
sigma2dBm = -174 + 10*log10(BW) + noiseFiguredB; % -94 dBm
sigma2 = db2pow(sigma2dBm);

P_S_dB = -5:.1:25; % Transmit power of the source, dBm, e.g., 200mW =
 23dBm

SNRdB = P_S_dB - sigma2dBm; % Average transmit SNR, dB = dBm - dBm,
 bar{rho} = P_S / sigma2
```



CO-CHANNEL INTER-CELL INTERFERENCE | CHANNEL MODELING

```
% Nakagami scale parameter
m_0_S1 = 2.5 + rand; % S->D, scale parameter, heuristic setting
m_0_S2 = 2.5 + rand;
m_0_S3 = 2.5 + rand;

m_h_S1 = 2.5 + rand(N_RIS, 1); % S->R
m_h_S2 = 2.5 + rand(N_RIS, 1);
m_h_S3 = 2.5 + rand(N_RIS, 1);

m_g = 2.5 + rand(N_RIS, 1); % R->D, for all sources

% Nakagami spread parameter
% can be used for all sources
Omega_0 = 1; % Normalized spread parameter of S->D link
Omega_h = 1; % Normalized spread parameter of S->RIS link
Omega_g = 1; % Normalized spread parameter of RIS->D link

% Path-loss
path_loss_0_S1 = pathloss_NLOS(d_S1D)*antenna_gain_S; % S->D link
path_loss_0_S2 = pathloss_NLOS(d_S2D)*antenna_gain_S;
path_loss_0_S3 = pathloss_NLOS(d_S3D)*antenna_gain_S;
```

```
path_loss_h_S1 = pathloss_NLOS(d_S1R) *
 antenna_gain_S*antenna_gain_RIS*L_single; % Source -> RIS
path_loss_h_S2 = pathloss_NLOS(d_S2R) *
 antenna_gain_S*antenna_gain_RIS*L_single;
path_loss_h_S3 = pathloss_NLOS(d_S3R) *
 antenna_gain_S*antenna_gain_RIS*L_single;

path_loss_g = pathloss_NLOS(d_RD) *
 antenna_gain_RIS*L_single*antenna_gain_D; % RIS -> Des

% phase of channels
phase_h_S1D = 2*pi*rand(1, sim_times); % domain [0,2pi)
phase_h_S2D = 2*pi*rand(1, sim_times);
phase_h_S3D = 2*pi*rand(1, sim_times);

phase_h_S1R = 2*pi*rand(N_RIS, L_single, sim_times); % domain [0,2pi)
phase_h_S2R = 2*pi*rand(N_RIS, L_single, sim_times);
phase_h_S3R = 2*pi*rand(N_RIS, L_single, sim_times);

phase_g_RD = 2*pi*rand(N_RIS, L_single, sim_times); % domain [0,2pi)

phase_h_S1R_eachRIS = zeros(L_single, sim_times);
phase_h_S2R_eachRIS = zeros(L_single, sim_times);
phase_h_S3R_eachRIS = zeros(L_single, sim_times);

phase_g_RD_eachRIS = zeros(L_single, sim_times);

% Channel modeling

h_S1D = sqrt(path_loss_0_S1) * random('Naka', m_0_S1, Omega_0, [1,
 sim_times]) .* exp(1i*phase_h_S1D);
h_S2D = sqrt(path_loss_0_S2) * random('Naka', m_0_S2, Omega_0, [1,
 sim_times]) .* exp(1i*phase_h_S2D);
h_S3D = sqrt(path_loss_0_S3) * random('Naka', m_0_S3, Omega_0, [1,
 sim_times]) .* exp(1i*phase_h_S3D);

h_S1R = zeros(N_RIS,L_single,sim_times);
h_S2R = zeros(N_RIS,L_single,sim_times);
h_S3R = zeros(N_RIS,L_single,sim_times);

g_RD = zeros(N_RIS,L_single,sim_times);

for nn=1:N_RIS
    phase_h_S1R_eachRIS = squeeze(phase_h_S1R(nn,:,:)); % random()
 just uses 2D
    phase_h_S2R_eachRIS = squeeze(phase_h_S2R(nn,:,:));
    phase_h_S3R_eachRIS = squeeze(phase_h_S3R(nn,:,:));

    phase_g_RD_eachRIS = squeeze(phase_g_RD(nn,:,:)); % random() just
 uses 2D

    for kk=1:L(nn)
```

```
        h_S1R(nn,kk,:) = sqrt(path_loss_h_S1(nn)) .* ... % need sqrt
 because path-loss is outside of random()
            random('Naka', m_h_S1(nn), Omega_h, [1, sim_times]) .* ...
            exp(1i*phase_h_S1R_eachRIS(kk,:));

        h_S2R(nn,kk,:) = sqrt(path_loss_h_S2(nn)) .* ...
            random('Naka', m_h_S2(nn), Omega_h, [1, sim_times]) .* ...
            exp(1i*phase_h_S2R_eachRIS(kk,:));

        h_S3R(nn,kk,:) = sqrt(path_loss_h_S3(nn)) .* ...
            random('Naka', m_h_S3(nn), Omega_h, [1, sim_times]) .* ...
            exp(1i*phase_h_S3R_eachRIS(kk,:));

        g_RD(nn,kk,:) = sqrt(path_loss_g(nn)) .* ... % need sqrt
 because path-loss is outside of random()
            random('Naka', m_g(nn), Omega_g, [1, sim_times]) .* ...
            exp(1i*phase_g_RD_eachRIS(kk,:));
    end
end
```

CO-CHANNEL INTER-CELL INTERFERENCE | ERA SCHEME

Phase-shift configuration based on channels of S1

```
h_ERA_cascade_S1 = zeros(N_RIS, sim_times);
h_ERA_cascade_S2 = zeros(N_RIS, sim_times);
h_ERA_cascade_S3 = zeros(N_RIS, sim_times);

for ss = 1:sim_times % loop over simulation trials
    for nn=1:N_RIS % loop over each RIS
        phase_shift_config_ideal = zeros(L_single,1);
        phase_shift_config_ideal_normalized = zeros(L_single,1);
        phase_shift_complex_vector = zeros(L_single,1);
        for ll = 1:L_single % loop over each elements of one RIS
            % unknown domain phase-shift
            phase_shift_config_ideal(ll) = phase_h_S1D(ss) -
 phase_h_S1R(nn,ll,ss) - phase_g_RD(nn,ll,ss);
            % convert to domain of [0, 2pi)
            phase_shift_config_ideal_normalized(ll) =
 wrapTo2Pi(phase_shift_config_ideal(ll));
            phase_shift_complex_vector(ll) =
 exp(1i*phase_shift_config_ideal_normalized(ll));
        end

        phase_shift_matrix = kappa_nl .*
 diag(phase_shift_complex_vector); % diagonal matrix

        % Cascade channel (complex, not magnitude)
        h_ERA_cascade_S1(nn,ss) =  h_S1R(nn,:,ss) * phase_shift_matrix
 * g_RD(nn,:,ss).';
```

```
        h_ERA_cascade_S2(nn,ss) =  h_S2R(nn,:,ss) * phase_shift_matrix
 * g_RD(nn,:,ss).';
        h_ERA_cascade_S3(nn,ss) =  h_S3R(nn,:,ss) * phase_shift_matrix
 * g_RD(nn,:,ss).';
    end
end

h_ERA_e2e_magnitude_S1 = abs(h_S1D + sum(h_ERA_cascade_S1,1)); %
 Magnitude of e2e channel
h_ERA_e2e_magnitude_S2 = abs(h_S2D + sum(h_ERA_cascade_S2,1));
h_ERA_e2e_magnitude_S3 = abs(h_S3D + sum(h_ERA_cascade_S3,1));

Z2_ERA = h_ERA_e2e_magnitude_S1.^2;

% Components of SINR

S_ERA = h_ERA_e2e_magnitude_S1.^2;
I_ERA = h_ERA_e2e_magnitude_S2.^2 + h_ERA_e2e_magnitude_S3.^2;
W = 1;
```

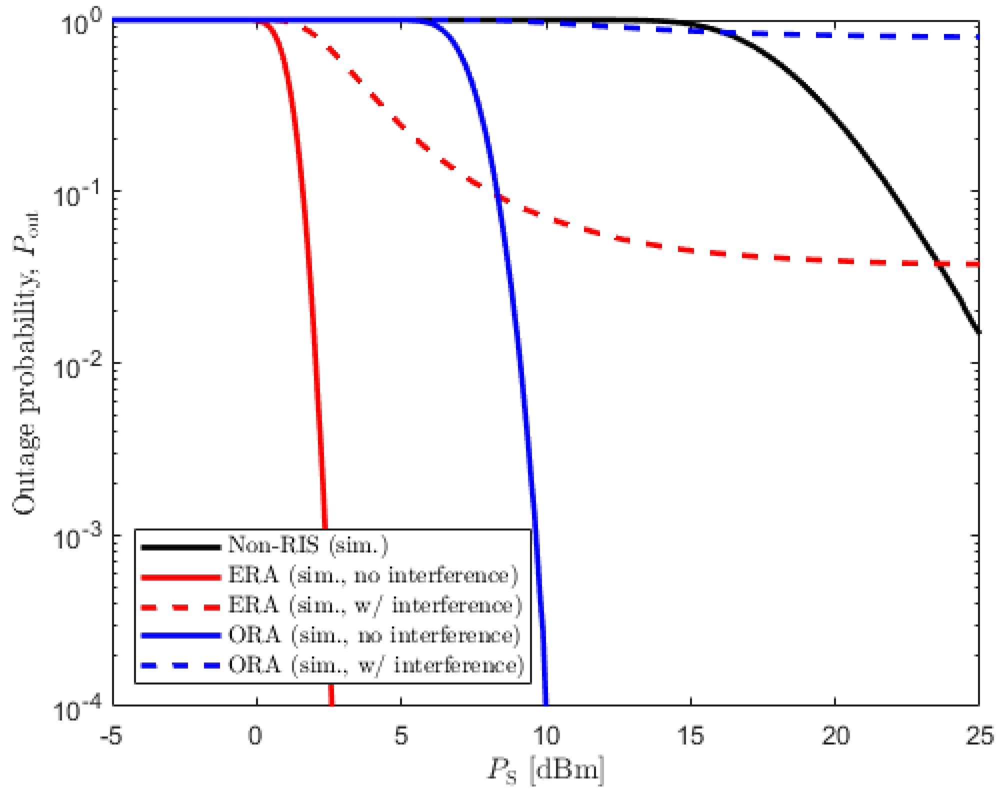
CO-CHANNEL INTER-CELL INTERFERENCE | ORA SCHEME

```
h_ORA_cascade_S1 = zeros(1,sim_times);
h_ORA_cascade_S2 = zeros(1,sim_times);
h_ORA_cascade_S3 = zeros(1,sim_times);

[~,idx] = max(h_ERA_cascade_S1,[],1); % find the best RIS associated
 with BS1

for ss = 1:sim_times
    phase_shift_config_ideal = zeros(L_single,1);
    phase_shift_config_ideal_normalized = zeros(L_single,1);
    phase_shift_complex_vector = zeros(L_single,1);
    for ll = 1:L_single % loop over each elements of one RIS
        % unknown domain phase-shift
        phase_shift_config_ideal(ll) = phase_h_S1D(ss) -
 phase_h_S1R(idx(ss),ll,ss) - phase_g_RD(idx(ss),ll,ss);
        % convert to domain of [0, 2pi)
        phase_shift_config_ideal_normalized(ll) =
 wrapTo2Pi(phase_shift_config_ideal(ll));
        phase_shift_complex_vector(ll) =
 exp(1i*phase_shift_config_ideal_normalized(ll));
    end

    phase_shift_matrix = kappa_nl .* diag(phase_shift_complex_vector);

    % cascade channel (complex number, not magnitude)
    h_ERA_cascade_S1(idx(ss),ss) =  h_S1R(idx(ss),:,ss) *
 phase_shift_matrix * g_RD(idx(ss),:,ss).'; % returns a single number
    h_ERA_cascade_S2(idx(ss),ss) =  h_S2R(idx(ss),:,ss) *
 phase_shift_matrix * g_RD(idx(ss),:,ss).';
```

```
    h_ERA_cascade_S3(idx(ss),ss) =  h_S3R(idx(ss),:,ss) *
 phase_shift_matrix * g_RD(idx(ss),:,ss).';

    h_ORA_cascade_S1(ss) = h_ERA_cascade_S1(idx(ss),ss);
    h_ORA_cascade_S2(ss) = h_ERA_cascade_S2(idx(ss),ss);
    h_ORA_cascade_S3(ss) = h_ERA_cascade_S3(idx(ss),ss);
end

h_ORA_e2e_magnitude_S1 = abs(h_S1D + h_ORA_cascade_S1);
h_ORA_e2e_magnitude_S2 = abs(h_S2D + h_ORA_cascade_S2);
h_ORA_e2e_magnitude_S3 = abs(h_S3D + h_ORA_cascade_S3);

R2_ORA = h_ORA_e2e_magnitude_S1.^2;

S_ORA = h_ORA_e2e_magnitude_S1.^2;
I_ORA = h_ORA_e2e_magnitude_S2.^2 + h_ORA_e2e_magnitude_S3.^2;
```

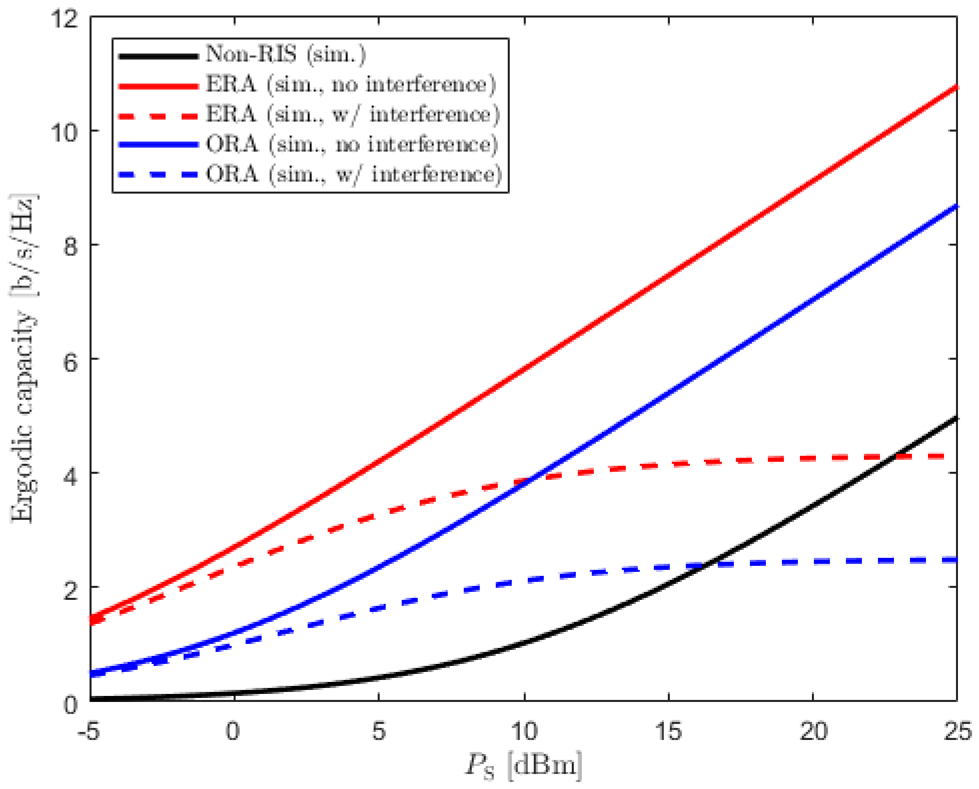
OUTAGE PROBABILITY

```
SNR_h0 = abs(h_S1D).^2;

for rr = 1:length(SNRdB)
    avgSNR = db2pow(SNRdB(rr)); % i.e., 10^(SNRdB/10)

    OP_SISO(rr) = mean(avgSNR*SNR_h0 < SNR_th);

    OP_ERA_no_interference_sim(rr) = mean(avgSNR.*Z2_ERA < SNR_th);
    OP_ERA_interference_sim(rr) = mean(avgSNR*S_ERA./(avgSNR*I_ERA+1)
 < SNR_th);

    OP_ORA_no_interference_sim(rr) = mean(avgSNR.*R2_ORA < SNR_th);
    OP_ORA_interference_sim(rr) = mean(avgSNR*S_ORA./(avgSNR*I_ORA+1)
 < SNR_th);

%     fprintf('Outage probability, SNR = % d \n', round(SNRdB(rr)));
end

figure;
semilogy(P_S_dB, OP_SISO, 'k-', 'linewidth', 2); hold on;
semilogy(P_S_dB, OP_ERA_no_interference_sim, 'r-', 'linewidth', 2);
 hold on;
semilogy(P_S_dB, OP_ERA_interference_sim, 'r--', 'linewidth', 2);
 hold on;
semilogy(P_S_dB, OP_ORA_no_interference_sim, 'b-', 'linewidth', 2);
 hold on;
semilogy(P_S_dB, OP_ORA_interference_sim, 'b--', 'linewidth', 2);
 hold on;

xlabel('$P_{\rm S}$ [dBm]', 'Interpreter', 'Latex');
ylabel('Outage probability, $P_{\rm out}$', 'Interpreter', 'Latex');
legend('Non-RIS (sim.)',...
    'ERA (sim., no interference)', ...
    'ERA (sim., w/ interference)', ...
    'ORA (sim., no interference)', ...
    'ORA (sim., w/ interference)', ...
    'Location','SW',...
    'Interpreter', 'Latex');
axis([-Inf Inf 10^(-4) 10^(0)]);
```



ERGODIC CAPACITY

```
for rr = 1:length(SNRdB)
    avgSNR = db2pow(SNRdB(rr)); % 10^(SNRdB(idx)/10)

    EC_non_RIS(rr) = mean(log2(1 + avgSNR*SNR_h0));

    EC_ERA_no_interference_sim(rr) = mean(log2(1+avgSNR*Z2_ERA));
    EC_ERA_interference_sim(rr) = mean(log2(1 + avgSNR*S_ERA./
(avgSNR*I_ERA+1) ));

    EC_ORA_no_interference_sim(rr) = mean(log2(1+avgSNR*R2_ORA));
    EC_ORA_interference_sim(rr) = mean(log2(1 + avgSNR*S_ORA./
(avgSNR*I_ORA+1) ));

%     fprintf('Ergodic capacity, SNR = % d \n', round(SNRdB(rr)));
end

figure;
plot(P_S_dB, EC_non_RIS, 'k-', 'linewidth', 2); hold on;
plot(P_S_dB, EC_ERA_no_interference_sim, 'r-', 'linewidth', 2);
 hold on;
plot(P_S_dB, EC_ERA_interference_sim, 'r--', 'linewidth', 2); hold on;
plot(P_S_dB, EC_ORA_no_interference_sim, 'b-', 'linewidth', 2);
 hold on;
plot(P_S_dB, EC_ORA_interference_sim, 'b--', 'linewidth', 2); hold on;


xlabel('$P_{\rm S}$ [dBm]', 'Interpreter', 'Latex');
ylabel('Ergodic capacity [b/s/Hz]', 'Interpreter', 'Latex');
legend('Non-RIS (sim.)',...
    'ERA (sim., no interference)', ...
    'ERA (sim., w/ interference)', ...
    'ORA (sim., no interference)', ...
    'ORA (sim., w/ interference)', ...
    'Interpreter', 'Latex',...
    'Location','NW');

% save('data_Gamma_setL2_interference.mat',...
%      'OP_ERA_no_interference_sim',...
%      'OP_ERA_interference_sim',...
%      'OP_ORA_no_interference_sim',...
%      'OP_ORA_interference_sim',...
%      'EC_ERA_no_interference_sim',...
%      'EC_ERA_interference_sim',...
%      'EC_ORA_no_interference_sim',...
%      'EC_ORA_interference_sim')
toc

Elapsed time is 64.686074 seconds.
```



Published with MATLAB® R2020b