MATLAB CODES FOR THE NUMERICAL SIMULATION AS A FUNCTION OF THE TRANSMITTED SNR

For the sake of space saving, only the main m-file for the numerical simulation as a function of the transmitted SNR is provided in this chapter. The simulation for other aspects than the transmitted SNR can be performed by slightly adjusting some lines in the following main m-file as described in each example previously. Moreover, no further change is required to the subfunctions of the LS, the WLS, and the ML estimators.

In the section "The Main m-File," the main m-file for the numerical simulation as a function of the transmitted SNR is explained in detail. The MATLAB subfunctions for the LS, the WLS, and the ML estimators are explained in detail in sections "MATLAB Subfunction 'LS_quick_search'," "MATLAB Subfunction 'WLS_known_gamma_original_search'," and "MATLAB Subfunction 'ML_known_gamma_original_search'," respectively. Finally, a standard reference of the built-in MATLAB optimization function `fminsearch` is given in the section "MATLAB Function Reference: `fminsearch`."

The Main m-File

- ```
  clc
  clear all
  ```
  The instructions are first introduced to clear the MATLAB screen and to clear all the variables that may previously be occupied in MATLAB workspace, respectively.
- `c = 3 * 10^8;`
  The wave propagation speed is assumed to be equivalent to the speed of the light; that is, $c = 3 \times 10^8$ m/s.
- `beta_bar = 2 * pi * (1/sqrt(3)) * 5 * 10^6;`
  The effective bandwidth is assigned as $\overline{\beta} = \left(1/\sqrt{3}\right)10\pi$ MHz.
- `r = 2000;`
  The cell radius is assumed to be $r = 2000$ m.
- `p = (1/2) * r * cos(pi/6) * [cos(pi/6);sin(pi/6)];`
  The mobile position is assigned according to Equation 18.51.
- `B = 7;`
  The number of participating BSs is assigned as $B = 7$.
- `M = 0;`
  The number of BSs that receive the NLOS signals is assigned as $M = 0$.
- ```
  a_range = [4.6 4.0 3.6];
  b_range = [0.0075 0.0065 0.0050];
  ```

```
c_range = [12.6 17.1 20.0];
h_b = 45; % (80+10)/2
```

The parameters a, b, c, and h_b for computing the PLE γ_b are assigned for several scenarios according to Reference 69.

- `p_true = p;`

  The true value of the mobile position is assigned.

- `f_0 = 1.9 * 10^9;`

  The central frequency is assumed to be $f_0 = 1.9\,\text{GHz}$.

- `d_0 = 100;`

  The close-in distance is assigned as $d_0 = 100\,\text{m}$.

- `kappa = c^2 / (16 * pi^2 * f_0^2 * d_0^2);`

  The constant κ is computed according to Equation 18.22.

- `P = r * [0,3/2,0,-3/2,-3/2,0,3/2;0,sqrt(3)/2,sqrt(3),…`
  `sqrt(3)/2,-sqrt(3)/2,-sqrt(3),-sqrt(3)/2].';`

  The BS coordinates are assigned.

- ```
  P_diff = P - repmat(p',B,1);
  phi = atan2(P_diff(:,2),P_diff(:,1));
  Phi = [cos(phi).';sin(phi).'];
  Phi_tilde = Phi(:,1:M);
  Phi_bar = Phi(:,M+1:B);
  ```

  The matrices $\tilde{\mathbf{F}}$ and $\bar{\mathbf{F}}$ are assigned according to Equations 18.61 and 18.30.

- `N_R = 10000;`

  The number of noise realizations is assigned.

- `SNR_range = [50:10:230]; % in dB`

  The SNR range is assigned for varying the SNR values in the simulation.

- `n_SNR = 0;`

  The assignment of the index of the SNR variation before the first loop.

- `for SNR_dB = SNR_range;`

  The for-loop is started according to the SNR range, which is assigned previously.

- `n_SNR = n_SNR + 1;`

  The index of the SNR variation is increased gradually by 1 in. each loop.

- `SNR = 10^(SNR_dB/10);`

  The SNR is at first assigned in decibels and later converted into the linear scale.

- `n_scenario = 0;`

  The number of scenarios is assigned at first to be 0 before the loop of three scenarios according to Reference 69.
  ```

- `for scenario = 1 % scenario_range`

  The for-loop can be started for several scenarios. However, only the first scenario is chosen in this simulation.

- `n_scenario = n_scenario + 1;`

  The index of the scenario variation is increased gradually by 1 in. each loop.

- ```
  gamma_gen = a_range(scenario) - b_range(scenario)*h_b …
              + c_range(scenario)/h_b;
  ```

  The PLE is computed according to a formula in Reference 69.

- `gamma_true = ones(B,1)*gamma_gen;`

  The generated PLE is assigned to the PLEs from all BSs.

- `bar_gamma = gamma_true(M+1:B,1);`

  The PLE vector for all LOS components is drawn from the PLEs from all BSs.

- `for b = 1 : B`

  The for-loop is started for several BSs.

- ```
  x_tilde(b) = P(b,1) - p_true(1);
  y_tilde(b) = P(b,2) - p_true(2);
  ```

  The variables $\tilde{x}_b$ and $\tilde{y}_b$ in Equation 18.2 are computed from the BS position matrix and the true coordinates of the MS.

- ```
  if b <= M
      tau_true(b) = (1/c) * (sqrt(x_tilde(b)^2 …
                    + y_tilde(b)^2) + l_true(b));
  else
      tau_true(b) = (1/c) * (sqrt(x_tilde(b)^2 …
                    + y_tilde(b)^2));
  end
  ```

  The TOAs for either the NLOS or LOS components are computed according to Equation 18.2. The value of the additional path length for the NLOS components is not mentioned herein, since the estimation algorithm and the theoretical performance do not depend on it. This means that we can assign any value for the additional path length of the NLOS components without affecting any simulation result.

- ```
  d_true(b) = c * tau_true(b);
  alpha_true(b) = sqrt( kappa * ( d_0/d_true(b) ) …
                  ^(gamma_true(b)) );
  frac(b) = 1 + (gamma_true(b)^2)/(8 * pi^2 * …
            beta_bar^2 * tau_true(b)^2);
  sigma2_true(b) = 1 / ( 8 * pi^2 * SNR * …
                   alpha_true(b)^2 * …
                   beta_bar^2 * frac(b));
  ```

```
end % b
bar_sigma2 = sigma2_true(M+1:B)';
```

The variance of the TOA perturbation is computed.

- `for n_R = 1 : N_R`

  The for-loop of several noise realizations is started until N_R times.

- `for b = 1 : B`

  The for-loop of several BSs is started until B times.

- `u_1(b) = randn;`

  A realization of zero-mean unit-variance normal random variable is generated for representing the additive noise.

- ```
  hat_tau_gen(b) = tau_true(b) +...
                   sqrt(sigma2_true(b)) * u_1(b);
  ```

- `end % b`

  The for-loop of several BSs is ended.

- `hat_bar_tau = hat_tau_gen(M+1:B)';`

  The LOS TOA vector is drawn from the generated TOAs for all BSs.

- `p_in = p_true + randn(1,1);`

  The initial value of the mobile position for the optimization is assumed to be the true value of the mobile position perturbed by another realization of the zero-mean unit-variance normal random variable.

- `hat_p_LS = fminsearch('LS_quick_search',p_in,[], …`

  `hat_bar_tau,P,M,B,c);`

  The mobile position is estimated from the optimization using the function `fminsearch` over the MATLAB subfunction "LS_quick_search" with the initial value in the previous item and other input values, such as the estimated LOS TOAs, the BS positions, the number of NLOS TOAs, the number of BSs, and the speed of the light.

- `hat_p_WLS_known_gamma = fminsearch( …`

  `'WLS_known_gamma_original_search',p_in,[], …`

  `hat_bar_tau,bar_gamma,beta_bar,d_0,SNR,P,M,B,c,kappa);`

  The mobile position is estimated from the optimization using the function `fminsearch` over the MATLAB subfunction "WLS_known_gamma_original_search" with the initial value in the previous item and other input values, such as the estimated LOS TOAs, the LOS PLEs, the effective bandwidth, the close-in distance, the SNR, the BS positions, the number of NLOS TOAs, the number of BSs, and the speed of the light.

- `hat_p_ML_known_gamma = fminsearch(…`

  `'ML_known_gamma_original_search',p_in,[],…`

  `hat_bar_tau,bar_gamma,beta_bar,d_0,SNR,P,M,B,c,kappa);`
  ```

The mobile position is estimated from the optimization using the function `fminsearch` over the MATLAB subfunction "ML_known_gamma_original_search" with the initial value in the previous item and other input values, such as the estimated LOS TOAs, the LOS PLEs, the effective bandwidth, the close-in distance, the SNR, the BS positions, the number of NLOS TOAs, the number of BSs, and the speed of the light.

- `SE_p_LS(n_SNR,n_R) =...`

  `sum((hat_p_LS - p_true).^2);`

  The error between the LS estimate and the true value of the mobile position is squared for a value of SNR and a noise realization.

- `SE_p_WLS_known_gamma(n_SNR,n_R) = …`

  `sum((hat_p_WLS_known_gamma - p_true).^2);`

  The error between the WLS estimate and the true value of the mobile position is squared for a value of SNR and a noise realization.

- `SE_p_ML_known_gamma(n_SNR,n_R) = …`

  `sum((hat_p_ML_known_gamma - p_true).^2);`

  The error between the ML estimate and the true value of the mobile position is squared for a value of SNR and a noise realization.

- `end % n_R`

  The for-loop of the noise realization is ended.

- `RMSE_p_LS(n_SNR) = sqrt(mean(SE_p_LS(n_SNR,:)));`

  The RMSE is the square root of the average of the computed error square from the LS estimate over all the realization of the noise for a value of the SNR.

- `RMSE_p_WLS_known_gamma(n_SNR) = …`

  `sqrt(mean(SE_p_WLS_known_gamma(n_SNR,:)));`

  The RMSE is the square root of the average of the computed error square from the WLS estimate over all the realization of the noise for a value of the SNR.

- `B_LS_theory = c^2 * inv(Phi_bar * Phi_bar') * …`

  `Phi_bar * diag( sigma2_true(M+1:B) ) * …`

  `Phi_bar' * inv(Phi_bar * Phi_bar');`

  The theoretical error variance by the LS estimate is computed from Equation 18.38 for a value of the SNR.

- `RMSE_p_LS_theory(n_SNR) = sqrt( trace( B_LS_theory ) );`

  The theoretical RMSE by the LS estimate for both axes is computed from the square root of the trace for a value of the SNR.

- `RMSE_p_ML_known_gamma(n_SNR) = …`

  `sqrt(mean(SE_p_ML_known_gamma(n_SNR,:)));`

  The RMSE is the square root of the average of the computed error square from the ML estimate over all the realization of the noise for a value of the SNR.

- `B_pp_given_gamma = c^2 * (1 / ( 8 * pi^2 * …`
  `beta_bar^2 * SNR)) * inv(Phi_bar *...`
  `diag(alpha_true(M+1:B))^2 * (eye(B-M) + …`
  `(1/(16 * pi^2 * beta_bar^2)) *...`
  `diag(gamma_true(M+1:B))^2 * …`
  `inv( diag(tau_true(M+1:B))^2 ) ) * Phi_bar');`

  The theoretical error variance by the WLS and the ML estimates is computed from Equation 18.46 for a value of the SNR.
- `CRB_p_given_gamma(n_SNR) = sqrt(trace(B_pp_given_gamma));`

  The theoretical CRB for both axes is computed from the square root of the trace for a value of the SNR.
- `end % scenario`

  The for-loop for each scenario is ended.
- `end % n_SNR`

  The for-loop for each value of the SNR is ended.
- ```
  figure
  hold on
  grid on
  plot(SNR_range,RMSE_p_LS,'r*','LineWidth',.5, …
  'MarkerEdgeColor','r','MarkerSize',5)
  plot(SNR_range,RMSE_p_LS_theory,':r','LineWidth',.5,…
  'MarkerEdgeColor','r','MarkerSize',5)
  plot(SNR_range,RMSE_p_WLS_known_gamma,'mo','LineWidth',.5,…
  'MarkerEdgeColor','m','MarkerSize',5)
  plot(SNR_range,RMSE_p_ML_known_gamma,'bs','LineWidth',.5,…
  'MarkerEdgeColor','b','MarkerSize',5)
  plot(SNR_range,CRB_p_given_gamma,'-k','LineWidth',.5,…
  'MarkerEdgeColor','k','MarkerSize',5)
  legend('LS: simulation','LS: analysis',…
  'WLS: simulation','ML: simulation','CRB')
  h = findobj(gcf,'type','axes','tag','legend');
  Pos = get(h,'position');
  Pos(3) = 1.9 * Pos(3); % 1.9 times the width
  Pos(4) = 3 * Pos(4); % 3 times the length
  set(h,'position',Pos) % Implement it
  title('RMSE as a function of SNR ($M=0$, $N_R=1{,}000$)')
  ylabel('Root Mean Square Error (m)')
  ```
  ```

```
xlabel('Transmitted SNR …
 $\frac{E_{\mathrm{s}}}{\sigma^2_{\mathrm{n}}}$ (dB)')
hold off
```

Some lines are given to plot experimental and theoretical curves as a function of the SNR.

MATLAB Subfunction "LS_quick_search"

- `function y = LS_quick_search(p_cal,hat_bar_tau,P,M,B,c)`

  The subfunction "LS_quick_search" is declared for searching the variable `p_cal` with the output parameter `y` and the input parameters, such as the estimated TOAs of the LOS components `hat_bar_tau`, the matrix of the BS positions `P`, the number of the NLOS components `M`, the number of BSs `B`, and the speed of the light `c`.

- ```
  bar_tau_theory = (1/c) * sqrt( ( P(M+1:B,1)
                   - p_cal(1,1) ).^2 …
                   + ( P(M+1:B,2) - p_cal(2,1) ).^2 );
  ```

  In terms of `bar_tau_theory`, the desired LOS TOAs are computed for a value of the searching vector `p_cal`.

- `y = sum((hat_bar_tau - bar_tau_theory).^2);`

  The output variable `y` is computed from the summation along all LOS components of the difference between the estimated LOS TOAs and the theoretical LOS TOAs calculated from the searching vector `p_cal`.

MATLAB Subfunction "WLS_known_gamma_original_search"

- `function y = WLS_known_gamma_original_search(p_cal, …`
  `hat_bar_tau,bar_gamma,beta_bar,d_0,SNR,P,M,B,c,kappa)`

  The subfunction "WLS_known_gamma_original_search" is declared for searching the variable `p_cal` with the output parameter y and the input parameters, such as the estimated TOAs of the LOS components `hat_bar_tau`, the PLEs in the LOS components `bar_gamma`, the effective bandwidth `beta_bar`, the close-in distance `d_0`, the matrix of the BS positions `P`, the number of the NLOS components `M`, the number of BSs `B`, the speed of the light `c`, and the constant `kappa`.

- ```
  bar_d_theory = sqrt( ( P(M+1:B,1) - p_cal(1,1) ).^2 + …
                 ( P(M+1:B,2) - p_cal(2,1) ).^2 );
  ```

  The distance between each BS and the MS is computed for all BSs.

- `bar_tau_theory = bar_d_theory / c;`

  The distances from the MS to all BSs are all divided by the speed of light.

- `frac = 1 + (bar_gamma.^2)./(16 * pi^2 * beta_bar^2 * … bar_tau_theory.^2);`

  A ratio is computed at first for preparation.

- `alpha_bar = sqrt( kappa * (d_0./ … (c*bar_tau_theory)).^(bar_gamma) );`

  The path gain is computed for all LOS components.

- `sigma2_b = 1 ./ ( 8 * pi^2 * SNR * beta_bar^2 * … alpha_bar.^2 .* frac);`

  The variance of the TOA estimation error is computed for all LOS components.

- `y = sum( ((hat_bar_tau - bar_tau_theory).^2) ./ sigma2_b );`

  The output variable `y` is computed from the summation along all LOS components of the difference between the estimated LOS TOAs and the theoretical LOS TOAs calculated from the searching vector `p_cal`.

MATLAB Subfunction "ML_known_gamma_original_search"

- `function y = ML_known_gamma_original_search(p_cal, … hat_bar_tau,bar_gamma,beta_bar,d_0,SNR,P,M,B,c,kappa)`

  The subfunction "ML_known_gamma_original_search" is declared for searching the variable `p_cal` with the output parameter `y` and the input parameters, such as the estimated TOAs of the LOS components `hat_bar_tau`, the PLEs in the LOS components `bar_gamma`, the effective bandwidth `beta_bar`, the close-in distance `d_0`, the matrix of the BS positions `P`, the number of the NLOS components `M`, the number of BSs `B`, the speed of the light `c`, and the constant `kappa`.

- `bar_d_theory = sqrt( ( P(M+1:B,1) - p_cal(1,1) ).^2 … + ( P(M+1:B,2) - p_cal(2,1) ).^2 );`

  The distance between each BS and the MS is computed for all BSs.

- `bar_tau_theory = bar_d_theory / c;`

  The distances from the MS to all BSs are all divided by the speed of light.

- `frac = 1 + (bar_gamma.^2)./(16 * pi^2 * beta_bar^2 * … bar_tau_theory.^2);`

  A ratio is computed at first for preparation.

- `alpha_bar = sqrt( kappa * (d_0./(c*bar_tau_theory)) … .^(bar_gamma) );`

  The path gain is computed for all LOS components.

- `sigma2_b = 1 ./ ( 8 * pi^2 * SNR * beta_bar^2 * … alpha_bar.^2 .* frac);`

  The variance of the TOA estimation error is computed for all LOS components.

- `y = sum( log(sigma2_b) + ((hat_bar_tau - … bar_tau_theory).^2) ./ sigma2_b );`

  The output variable `y` is computed from the summation along all LOS components of the difference between the estimated LOS TOAs and the theoretical LOS TOAs calculated from the searching vector `p_cal`.

MATLAB Function Reference: `fminsearch`

`fminsearch`

Minimize a function of several variables. For more details, see MATLAB help.

Limitations: `fminsearch` can often handle discontinuity, particularly if it does not occur near the solution. `fminsearch` may only give local solutions.

`fminsearch` only minimizes over the real numbers, that is, `x` must only consist of real numbers and the function to be minimized, `f(x)`, must only return real numbers. When `x` has complex variables, they must be split into real and imaginary parts.

*For random NLOS excess propagation paths, if the knowledge of the distribution of l_m is available, the parameter estimation can be cast into a *maximum a posteriori* approach [43].