User manual for the code package : Modeling generalized scattering matrix containing evanescent modes for wavefront shaping applications in disordered media

Michael Raju,[1,2,*] Baptiste Jayet,[1] and Stefan Andersson-Engels[1,2]

[1] *Tyndall National Institute, Lee Maltings Complex, Dyke Parade, Cork, Ireland, T12 R5CP*
[2] *School of Physics, University College Cork, College Road, Cork, Ireland, T12 K8AF*

This document is the user manual for the Matlab® based code packages associated with the manuscript titled "Modeling generalized scattering matrix containing evanescent modes for wavefront shaping applications in disordered media". There are two stand-alone code packages associated with this Github repository. Code-package-1 deals with the estimation of the generalized S matrix for a single thick diffusive disorder and helps to visualize the transport of various shaped wavefronts (eg: propagating eigenchannels) inside the given disorder using the boundary integral equation. Two options are available for the Code-package-1. Option no 1 is for loading an already saved computational run data (hosted as a Zenodo repository) for the quick visualization of various plots associated with the code package. Option no 2 is for a new computational run with a newly initialized random disorder. Code-package-2 generates an ensemble of generalized S matrices estimated from a collection of thin slices of disorders. Generalized S matrices ensembles of thicker disorders can be quickly estimated through the S-matrix-cascading of thinner disorders by shuffling the order of cascading. Hence, Code-Package-2 is utilized for the ensemble averaging of the results associated with the focusing of an evanescent wave mode presented in the main paper. To summarize, Code-Package-2 is comparatively faster and uses lesser memory with respect to Code-Package-1, if the goal is to generate an ensemble of generalized S matrices for statistical analysis, without visualizing wave transport inside the disorder. On the other hand, Code-Package-1 is used for the purpose of wave transport visualization and S matrix estimation for a single disorder. All the computations were performed on a desktop PC with a configuration of 64 GB of RAM and an eight-core CPU.

I. INTRODUCTION

The main Github code repository contains two self-contained Matlab® code-packages as shown in the FIG. 1 targeting two different applications. Code-Package-1 is mainly used for estimating the generalized S matrix for a single thicker diffusive disorder and the visualization of shaped waves inside that disorder. On the other hand, Code-Package-2 is utilized for the statistical analysis involving the cascading of several generalized S matrices of thinner disorders to estimate the generalized S matrix ensembles of the thicker disorders. Fast generation of the ensembles of cascaded S matrices is possible by randomly shuffling the cascading order of S matrices of thinner disorders estimated just for once. This helps in the fast ensemble averaging for the estimation of the mesoscopic fluctuations or correlations. Code-Package-2 also uses lesser memory compared to Code-Package-1 for similarly sized disorders. This is because Code-Package-2 decomposes the thicker disorder as a cascade of thinner disorders and estimate their generalized S matrices one after the other followed by cascading. Code-Package-2 is also faster in comparison to Code-Package-1 in generating ensembles of generalized S matrices as cascading is performed quickly using the S matrix cascading rule. On the other hand, as Code-Package-2 uses S matrix cascading method, it comes with the limitation that wave fields can't be visualized inside the disorder. The code units contained in Code-Package-1 is reused again in Code-Package-2 for the generation of S matrices. As given in FIG. 1, both Code-Package-1 and Code-Package-2 have the flag `new_run_flag` to be set as 0 or 1 to load an already saved computational run data or to start a fresh run with a newly initialized disorder, respectively. Such a saved run data is hosted in a Zenodo repository as it is recommended to limit the storage of large files associated with Github repositories. Users may download the saved run data from the Zenodo repository and place it within the associated local Github directory as shown in FIG. 2 and FIG. 3. The default setting for `new_run_flag` is 0 so that the already saved run data is loaded and the plots given in the main paper and the supplement are generated.

The computational need for splitting the code packages into two is due to the reason that the Green's function perturbation method in general is memory intensive, especially with increase in size of the disorder. Hence, with Code-Package-1, ensemble averaging becomes challenging especially with thicker disorders. With thinner disorders in Code-Package-2, direct Green's function perturbation method (explained in the supplementary document of the

* michaelraju@umail.ucc.ie

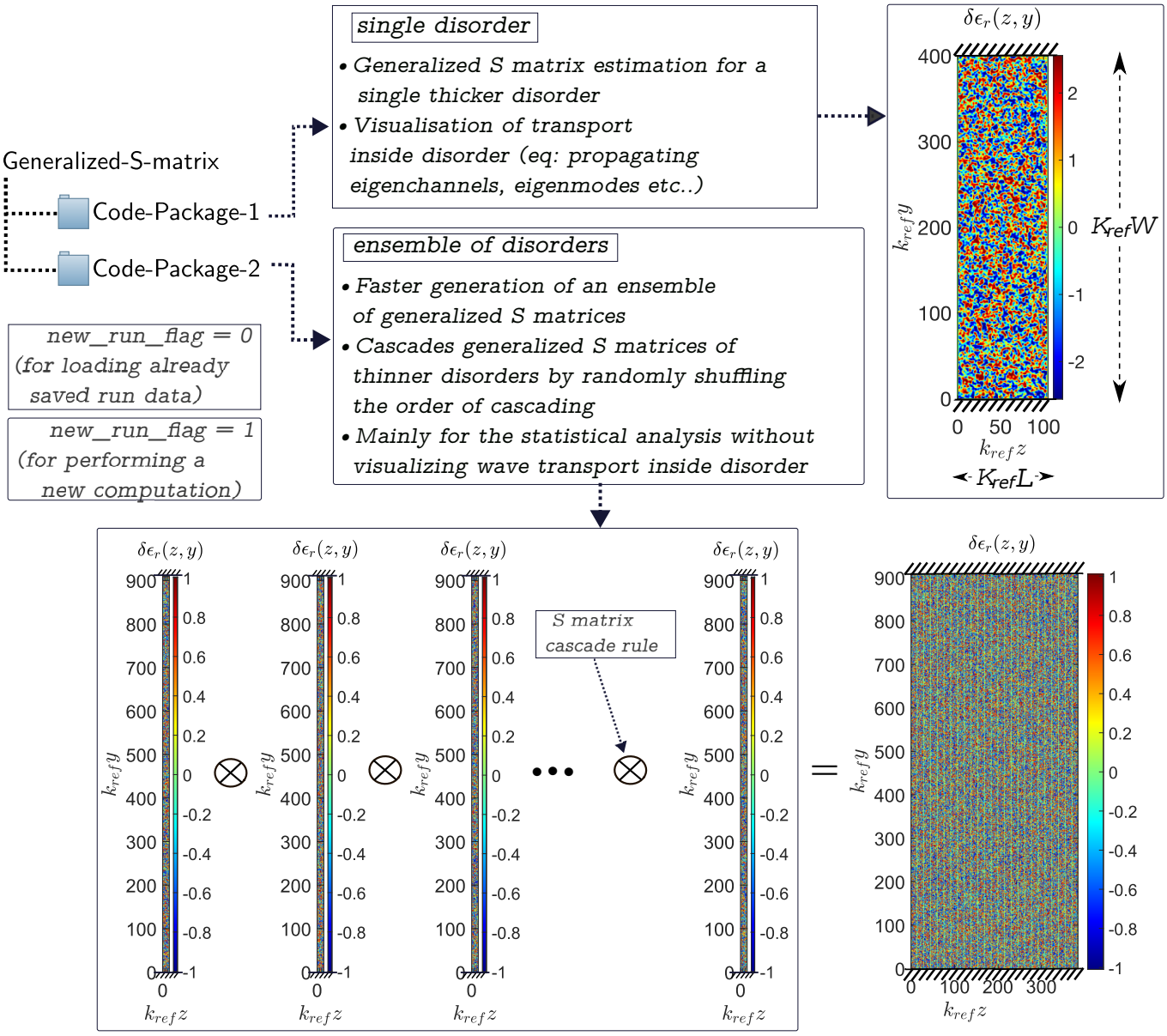

FIG. 1. **Overview of the code repository and the associated code packages.** Two standalone code packages namely, Code-Package-1 and Code-Package-2 are given. Both code packages estimate generalized S matrices, although the end goals are different as given above.

main paper) is used as such a method is comparatively faster for thinner disorders. For the thicker disorders in Code-Package-1, iterative Green's function method (explained in the supplementary document) is used as it is comparatively memory efficient with respect to the direct method.

Larger the transverse size $k_{ref}W$ of the disorder, more the number of propagating modes available and better it represents the analytical results from Random Matrix Theory (R.M.T) where usually the limit of the number of modes is taken to be a large number. Smaller the W value, the finite size and dimensionality effects of the random S matrix components has to be accounted in for comparing with the analytical results from R.M.T. For both the code packages, spatially correlated disorders without any loss or gain are used. Spatially correlated disorders are implemented to provide customizability of the scattering properties of the disorder. A disorder with a finite spatial correlation length would scatter the waves efficiently for a given peak refractive index fluctuation, in comparison to a white-noise like disorder (spatial correlation length approaching zero). The users could also easily modify these code packages to incorporate any kind of scattering structures instead of the disordered medium used for the paper as the Green's function perturbation method is quite general with respect to the nature of the disorder.

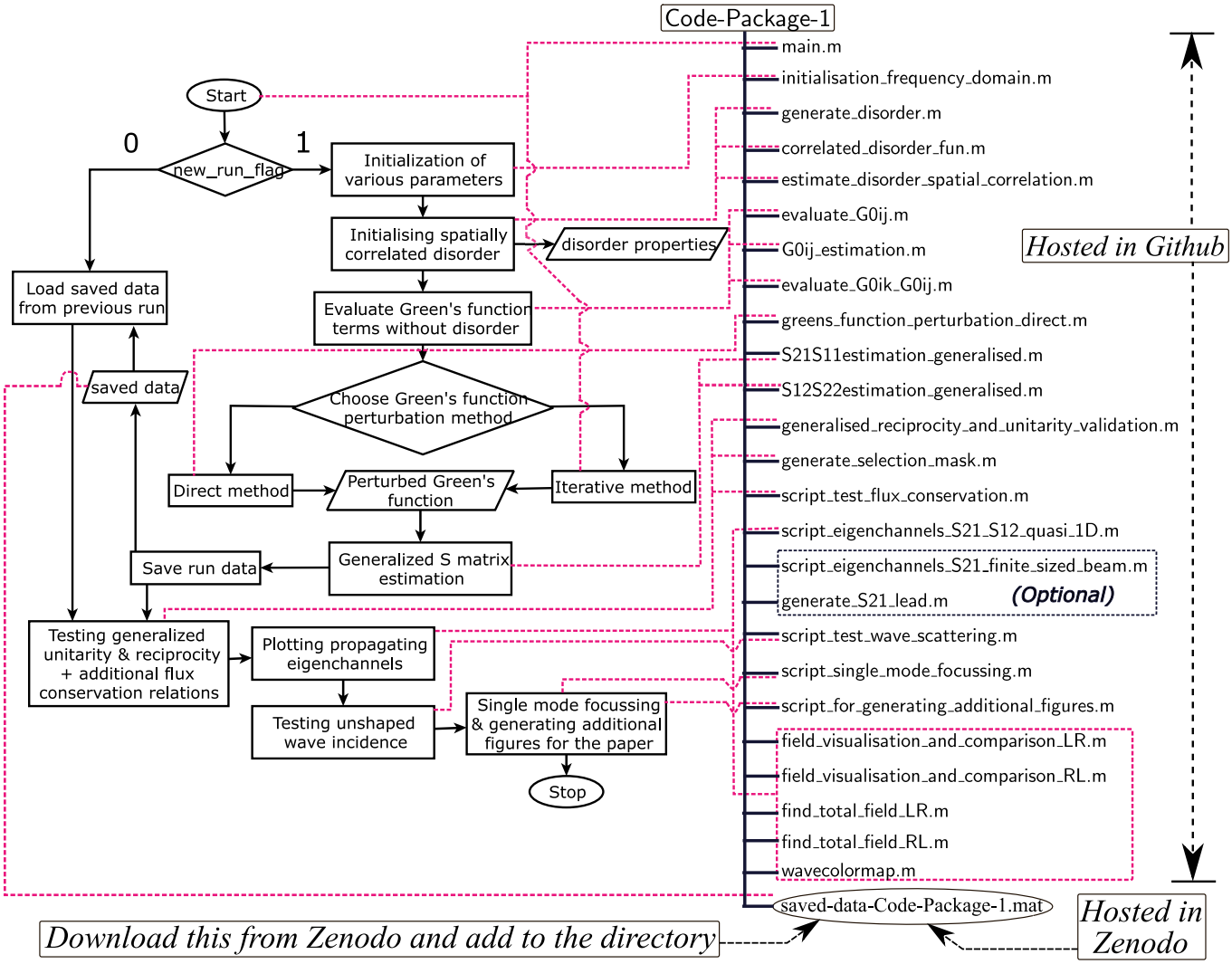

FIG. 2. **Flowchart for summarizing the workflow of Code-Package-1 and its associated files.** The *.m* code files are hosted in Github and the saved run data (`saved-data-Code-Package-1.mat`) is hosted in Zenodo.

Code files within the code packages are commented and this user manual gives a top-level overview of the structure of the code packages. First, the overview and various aspects of Code-Package-1 are described. This is followed by the discussion on Code-Package-2. As Code-Package-2 reuses some of the code units covered in Code-Package-1, only the previously undiscussed aspects of Code-Package-2 are covered. For the implementation of the code, a desktop PC with the configuration of 64 GB of RAM and an eight-core CPU was used. Users who don't have 64 GB of RAM for having a new computational run (`new_run_flag=1`) for the Code-Package-1, may reduce the transverse size of the slab $k_{ref}W$ accordingly. For the option `new_run_flag=0`, one needs only the RAM for loading the saved run data (particularly for ensemble averaging) and performing minimal computation on it.

II. CODE-PACKAGE-1

The flowchart describing the operational flow of Code-Package-1 is given in FIG. 2. The files associated with the workflow are also marked using pink-colored dashed-lines in the FIG. 2. The main.m files starts as the following with the default value of `new_run_flag=0` to load an already-saved data from a previous computation.

```
new_run_flag=0;
% Set the flag=1 if a new run is required.
% If flag=1, one may reduce the size of krefW according
```

```
% to one's memory availability.
```

In case if `new_run_flag=1` for a new computational run from the beginning, the initialization procedure starts as the following.

A. Initialization

Within the main.m file, the following top-level initialization commands are given.

```
%------------------ initialise ------------------------------------------
krefW=400;   % Dimensionless transverse size (along y axis) of the slab
%               kref*W.
krefZ=100;    % Dimensionless longitudinal size of the computational domain
%               kref*Z (along z axis).
lambda0=632.8*10^-9;      % Wavelength in nm.
n0=1.9;                   % Reference(Background) medium refractive index.
offset=2;
% Offset is defined wrt to the left and right sides of the computational
% domain. The Offset value is used to control the dimensionless slab thickness
% The dimensionless slab thickness is krefL=krefZ-2*offset,
% to be initialised later. Larger the offset value for a given krefZ,
% thinner the slab.
disorder_corr_flag=1; % 1 for correlated disorder and 0 for uncorrelated.
disorder_spatial_corr_fact=2; % A factor used to control disorder
% spatial correlation length. Larger the factor, larger the
% correlation length. Exact value of the dimensionless spatial
% correlation length would be estimated later.
% Usually, disorder_spatial_corr_fact is taken greater than  1
disorder_perturbation_strength_parameter=0.99;
% a number between 0 and 1 to adjust the strength of scattering
% or the degree of the disorder perturbation.
num_modes_evanes=30;  % number of evanescent waves, set by the user.
% Larger the number, finer will be the
% spatial resolution, higher the memory
% required for a large sample.
%pertubation_method='D';
pertubation_method='Iter';
% 'D' for direct matrix inversion method : Fast, but memory intensive
%      and mainly used for purposes involving smaller samples.
% 'Iter' for the iterative method : Slower, but uses lower memory in
%      comparison to the direct matrix inversion method.
```

The remaining initialisation parameters derived from the top-level initialization commands are evaluated using the function `initialisation_frequency_domain(...)` which returns a structure `init_data`. Some of the initialization parameters defined before calling the said function are also added to this structure `init_data` later.

```
init_data = initialisation_frequency_domain(krefW,krefZ,...
n0,lambda0,num_modes_evanes);  % Initialisation
init_data.disorder_perturbation_strength_parameter=...
disorder_perturbation_strength_parameter;
init_data.disorder_corr_flag=disorder_corr_flag;
init_data.offset=offset;
init_data.krefL=(init_data.Nz-2*init_data.offset-1)*init_data.kref*init_data.dz;
% Dimensionless thickness of the
% disordered slab along the z axis.
init_data.sigma_disorder= ...
disorder_perturbation_strength_parameter*(init_data.n0^2-1);
% dielectric disorder strength
```

The data fields associated with the initialization structure `init_data` are listed as the following.

init_data.krefW : The dimensionless transverse size $k_{ref}W$ of the slab (along Y axis).
init_data.krefZ : The dimensionless thickness $k_{ref}Z$ of the computational domain (along Z axis).
init_data.lambda0 : free space wavelength λ_0.
init_data.n0 : Background (reference) refractive index η_0 upon which the disorder perturbation is performed.
init_data.kref : Wavevector magnitude in the reference (background) medium without any disorder, $k_{ref} = (2\pi\eta_0)/\lambda_0$.
init_data.W : Transverse size of the slab W.
init_data.Zmax : Thickness of the computational domain along Z axis.
init_data.num_modes_prop : Number of propagating modes possible for the given $k_{ref}W$.
init_data.num_modes_evanes : The truncated number of evanescent modes taken.
init_data.num_modes : Total number of modes = sum of propagating and evanescent number of modes.
init_data.dy : Δ_y is the numerical grid resolution along Y axis.
init_data.dz : Δ_z is the numerical grid resolution along Z axis.
init_data.kydy : Combined term $k_y\Delta_y$ where k_y is the Y component of the wave vector.
init_data.kzdz : Combined term $k_z\Delta_z$ where k_z is the Z component of the wave vector.
init_data.kz_flux : Numerical value of k_z on a discrete grid being $sin(|k_z\Delta_z|)/\Delta_z$. Upon the limit $\Delta_z \to 0$, it yields $|k_z|$ to be used in the generalized Fisher–Lee relations.
init_data.Z : Meshgrid matrix containing the Z values of the numerical grid.
init_data.Y : Meshgrid matrix containing the Y values of the numerical grid.
init_data.Nz : The grid matrix size in terms of the number of grid nodes along Z.
init_data.Ny : The grid matrix size in terms of the number of grid nodes along Y.
init_data.Nmat : Meshgrid matrix containing the position indices corresponding to $1:N_z$.
init_data.Mmat : Meshgrid matrix containing the position indices corresponding to $1:N_y$.
init_data.jth : Column vector position indices corresponding to $1:N_y$.
init_data.Chi : Function handle referring to the transverse component of the eigenmode being $\chi_m(y) = \sqrt{\frac{2}{W}} sin(k_y^{(m)} y)$.
init_data.phi_LR_prop : Left to right propagating eigenmode's Z component $\Phi^{+}_{m,pr}(z)$.
init_data.phi_RL_prop : Right to left propagating eigenmode's Z component $\Phi^{-}_{m,pr}(z)$.
init_data.find_prop_modes_LR : Propagating eigenmode from left to right, the combined form.
init_data.find_prop_modes_RL : Propagating eigenmode from right to left, the combined form.
init_data.phi_evanes : Evanescent eigenmode's Z component $\Phi_{m,ev}(z, z_{L/R})$.
init_data.find_evanes_modes : Evanescent eigenmode combined.
init_data.left_boundary_src : Column vector containing the position indices for the left boundary which acts as source. This is used to index G0ij matrix.
init_data.right_boundary_src : Column vector containing the position indices for the right boundary which acts as source. This is used to index G0ij matrix.
init_data.left_boundary_field : Column vector containing the position indices for accessing the field position values on the left boundary.
init_data.right_boundary_field : Column vector containing the position indices for accessing the field position values on the right boundary.
init_data.Nz0_L : z Position index for the left boundary.
init_data.Nz0_R : z Position index for the right boundary.
%

Next step is to initialize the disorder dielectric constant perturbation profile, $\delta\epsilon(z, y)$ and estimate its spatial correlation properties. As `disorder_corr_flag` is set to be unity by default, option for generating a spatially correlated disorder is selected. The parameter `disorder_spatial_corr_fact` defined before can be used to adjust the spatial correlation length of the disorder. In addition to that, the parameter `sigma_disorder` controls the variance of $\delta\epsilon(z, y)$. Upon passing the `init_data` to the `generate_disorder(...)` function, it returns the $\delta\epsilon(z, y)$ matrix assigned to the variable

`eps_profile`, given as the following.

```
%---------------------- define disorder ----------------------------------------
init_data.sigma_disorder= ...
disorder_perturbation_strength_parameter*(init_data.n0^2-1);
% dielectric disorder strength
if init_data.disorder_corr_flag==1
init_data.kreflc=disorder_spatial_corr_fact*(init_data.kref*init_data.dz);
% kreflc is kref*lc where lc is the spatial correlation length
% (init_data.kref*init_data.dz) is the smallest possible value
if init_data.krefL<10*init_data.kreflc
sprintf('Sample thickness is preferred to be atleast 10kref*lc !!!')
pause;
end
end
eps_profile = generate_disorder(init_data);
sprintf('No of grid elements = %d',numel(eps_profile))
```

Once the $\delta\epsilon(z, y)$ is initialized as `eps_profile`, next is to numerically estimate the spatial correlation properties of the generated disorder using the function `estimate_disorder_spatial_correlation(...)`. Those properties are added to the `init_data` structure.

```
%---------- Estimate the disorder properties ------------------------------
[kreflc_main_numerical] = ...
estimate_disorder_spatial_correlation(eps_profile,init_data);
init_data.kreflc_main_numerical=kreflc_main_numerical;
init_data.eps_profile=eps_profile;
init_data.eps_lin_nonzero=find(eps_profile);
init_data.no_of_disorder_perturbations=numel(init_data.eps_lin_nonzero);
init_data.no_of_grid_elements=(init_data.Nz*init_data.Ny);
```

Upon having the $\delta\epsilon(z, y)$ initialized, the next step is to implement the Green's function perturbation method. The method for implementing the perturbation is provided in the supplementary document.

B. Green's function perturbation method

First, the Green's function terms within the computational domain without any disorder are estimated.

```
%---------------------- Estimate free space greens function ---------------
if init_data.no_of_disorder_perturbations==0
[G0ij] = evaluate_G0ij(init_data); % Estimating the Green's function
%    without the disorder.
else
[G0ik,G0ij] = evaluate_G0ik_G0ij(init_data); % Estimating the Green's
% function without disorder
end
G0ij_LR=reshape(G0ij(:,init_data.left_boundary_src), ...
[init_data.Ny init_data.Nz init_data.Ny]);
% Saving a part of G0ij now to be
%        used for plotting later.
% Saved now because in the iterative method,
%         G0ik and G0ij will be overwritten.
```

Two methods for the perturbation (as given in the supplementary document) are available to choose by setting `pertubation_method` as the following.

```
%pertubation_method='D';
pertubation_method='Iter';
% 'D' for direct matrix inversion method : Fast, but memory intensive
%      and mainly used for purposes involving smaller samples.
% 'Iter' for the iterative method : Slower, but uses lower memory in
```

```
%       comparison to the direct matrix inversion method.
```

For the direction matrix inversion method, the function `greens_function_perturbation_direct(...)` is utilized. For the iterative method, a separate function file is not used, but instead the code portion is provided within the main.m file itself. This particularly helps to save memory as some of the Green's function matrices are overwritten without passing a copy to a function. One may refer the supplementary material for details regarding the implementation algorithm. The code portion of the iterative method in the main.m file is given as the following

```
block_size_fraction=(1/10);
% Block size fraction is the fraction of the total no of disorder particle
% grid points taken per step of the iteration process to solve for
% the perturbed field. For example, if the total no of disorder pertubation
% grid points is 20,000. then block_size_fraction=(1/5) implies,
% 4000 particles are solved  at once per the iteration method and the
% Green's function is updated for every 4000 particles taken together,
% per iteration step. Hence, 5 steps are needed to update the perturbed field.



%---------------------- Estimate perturbed Green's function ---------------------
if init_data.no_of_disorder_perturbations~=0
if strcmp(pertubation_method,'D') % Choosing the direct method.
[Gij_R_LR,Gij_L_LR,Gij_LR,Gij_R_RL,Gij_L_RL,Gij_RL]= ...
greens_function_perturbation_direct(G0ik,G0ij,init_data);
elseif strcmp(pertubation_method,'Iter')% Choosing the iterative method.
init_data.block_size_fraction=block_size_fraction;
src_ind=[init_data.left_boundary_src init_data.right_boundary_src];
no_of_iterations=floor(1/init_data.block_size_fraction);
no_of_particles_per_iteration= ...
floor(block_size_fraction*init_data.no_of_disorder_perturbations);
%---------------------- Start iteration ------------------------------------------
tic
for b_count=1:no_of_iterations
sprintf('Iterative pertubation using scatterer block no %d/%d',...
b_count,no_of_iterations)
if b_count~=no_of_iterations
particle_index= ...
1+(b_count-1)*no_of_particles_per_iteration: ...
b_count*no_of_particles_per_iteration;
elseif b_count==no_of_iterations
particle_index= ...
1+(b_count-1)*no_of_particles_per_iteration: ...
init_data.no_of_disorder_perturbations;
end
% particle_index chooses the block of scatterers taken together
% for the perturbing the Green's function
G0kj=G0ij(init_data.eps_lin_nonzero(particle_index),src_ind);
% For G0kj, field location k is taken only at the chosen
% perturbation particle locations and j the source location at
% the boundary.
G0kk=G0ik(init_data.eps_lin_nonzero(particle_index), ...
particle_index(end)+1:end);
% For G0kk, field location k is taken only at the chosen
% perturbation particle locations and the source location k is
% taken for the remaining particle locations which are yet to be
% brought up for perturbation.
%------ Estimate Mpert matrix (Refer the associated paper/thesis)--
eps_full=(eps_profile(:));
k0=init_data.kref/init_data.n0; % Because in Dysons equation,
```

```
% we use the free space k0, not the kref.
Vkdeltak=...
-(init_data.dz*init_data.dy*k0^2).*eps_full(init_data.eps_lin_nonzero(particle_index
    ));
Mpert=(zeros(length(particle_index),length(particle_index)));
for scount=1:length(Vkdeltak)
for dcount=1:length(Vkdeltak)
Mpert(dcount,scount)=...
G0ik(init_data.eps_lin_nonzero(particle_index(dcount)),...
particle_index(scount))*Vkdeltak(scount);
end
end
%------------------------- Estimate Gkj -------------------------------
G0kj=(eye(length(Vkdeltak))-Mpert)\G0kj;
G0kk=(eye(length(Vkdeltak))-Mpert)\G0kk;
clearvars Mpert;
%--- Estimate Gij ------------------------------
G0ij=G0ij + G0ik(:,particle_index).*repmat(Vkdeltak.',...
init_data.Nz*init_data.Ny,1)*G0kj;
temp=G0ik(:,particle_index).*repmat(Vkdeltak.',...
init_data.Nz*init_data.Ny,1);

if b_count~=no_of_iterations
for r_count=b_count+1:no_of_iterations
r_count
if r_count~=no_of_iterations
remaining_particle_index= ...
1+(r_count-1)*no_of_particles_per_iteration: ...
r_count*no_of_particles_per_iteration;
elseif r_count==no_of_iterations
remaining_particle_index= ...
1+(r_count-1)*no_of_particles_per_iteration: ...
init_data.no_of_disorder_perturbations;
end
G0ik(:,remaining_particle_index)= ...
G0ik(:,remaining_particle_index) + ...
temp*G0kk(:,remaining_particle_index-particle_index(end));
end
end

end
clearvars G0ik G0kj G0kk temp; % This is a partially overwritten matrix.
% Good to clear those matrices now.
%--------- Store the perturbed Green's function matrices ------------
Gij_L_LR=(G0ij(init_data.left_boundary_field,init_data.left_boundary_src));
Gij_R_LR=(G0ij(init_data.right_boundary_field,init_data.left_boundary_src));
Gij_R_RL=(G0ij(init_data.right_boundary_field,init_data.right_boundary_src));
Gij_L_RL=(G0ij(init_data.left_boundary_field,init_data.right_boundary_src));
Gij_LR=reshape(G0ij(:,init_data.left_boundary_src),[init_data.Ny init_data.Nz
    init_data.Ny]);
Gij_RL=reshape(G0ij(:,init_data.right_boundary_src),[init_data.Ny init_data.Nz
    init_data.Ny]);
% Gij_R_LR : LR means wave incident from left to right, R denotes Green's
%            function receiver on the right boundary
% Gij_L_LR : LR means wave incident from left to right, L denotes Green's
%            function receiver on the left boundary
% Gij_LR   : Greens function for the source on the left boundary
% Similarly RL means right to left
```

```
sprintf('Time taken iteratively = %f mins',toc/60)
end
clearvars G0ij; % Good to clear
%-------------------------------------------------------------------------
else
% If there is no perturbation, then take G=G0, the unperturbed Green's
% functions
Gij_LR=reshape(G0ij(:,init_data.left_boundary_src),[init_data.Ny init_data.Nz
    init_data.Ny]);
Gij_RL=reshape(G0ij(:,init_data.right_boundary_src),[init_data.Ny init_data.Nz
    init_data.Ny]);
Gij_L_LR=(G0ij(init_data.left_boundary_field,init_data.left_boundary_src));
Gij_R_LR=(G0ij(init_data.right_boundary_field,init_data.left_boundary_src));
Gij_R_RL=(G0ij(init_data.right_boundary_field,init_data.right_boundary_src));
Gij_L_RL=(G0ij(init_data.left_boundary_field,init_data.right_boundary_src));
end
```

C. Generalized S matrix estimation

Once the perturbed Green's function terms are obtained, the generalized S matrix components S_{11}, S_{21}, S_{12} and S_{22} are estimated using the following functions.

```
%-- Estimate the transmission and reflection matrix, S21 and S11 ------------
[S21,S11] = S21S11estimation_generalised(Gij_L_LR.',Gij_R_LR.',init_data);
%-- Estimate Transmission and reflection matrix, S12 and S22 ----------------
[S12,S22] = S12S22estimation_generalised(Gij_R_RL.',Gij_L_RL.',init_data);
save('S11.mat','S11');save('S12.mat','S12');
save('S21.mat','S21');save('S22.mat','S22');
save('init_data.mat','init_data');
```

One may refer the main paper and the supplementary document for the generalized Fisher-Lee relations and the associated derivations. Next, the data saving/loading portion is given.

```
if new_run_flag==0
% If a new run is not needed, load the existing saved data.
if isfile('saved-data-Code-Package-1.mat')
% File exists.
load('saved-data-Code-Package-1.mat');
else
% File does not exist and needs to be downloaded from the Zenodo
% repository.
disp('Load the saved-data-Code-Package-1.mat file')
[selected_data_name,data_path] = uigetfile('*.mat');
if strcmp(selected_data_name,'saved-data-Code-Package-1.mat')
load(selected_data_name);
disp('Loaded saved-data-Code-Package-1.mat file')
else
disp('Right .mat not selected !!')
end
end

else
disp('Warning : Overwriting the previously saved data...')
pause(5);
save('saved-data-Code-Package-1.mat') % If a new run is needed, save the workspace
    data
end
```

If the `new_run_flag` is set to 1, then the workspace data is saved as `saved-data-Code-Package-1.mat`. On the other hand if `new_run_flag` is set to 0, it loads the said data file.

D. Generalized reciprocity and unitarity relations

Next section is the analysis part of the main.m file. First is the following script for numerically verifying the generalized reciprocity and unitarity relations. The script generates the associated plots given in the main paper.

```
% Validation for the generalised unitarity and reciprocity properties
generalised_reciprocity_and_unitarity_validation(S11,S12,S21,S22,init_data)
```

Within the above given script, the function `generate_selection_mask(...)` is used to generate the matrix terms containing ones and zeros which are used to extract the submatrices from the main S matrix. This results in obtaining `Smatrix_pr_pr`, `Smatrix_pr_ev`, `Smatrix_ev_pr` and `Smatrix_ev_ev` as described in the main paper. Eventually, the partitioned S matrix `Spartitioned` is estimated using which the generalized unitarity relation is verified. In addition to that, the generalized reciprocity relations are also numerically verified. Additionally broken-down forms of the generalized unitarity relations (given in the supplementary document) are numerically verified using the following script.

```
%------- Script for verifying various flux conservation relations ----------
script_test_flux_conservation
```

E. Transmission eigenchannels of $S_{21}^{pr,pr}$ and $S_{12}^{pr,pr}$

As the transmission matrices $S_{21}^{pr,pr}$ and $S_{12}^{pr,pr}$ are available along with the perturbed Green function terms, the transmission eigenchannels can be plotted using the boundary integral equation. The script for such a task is the following.

```
%-- Script for demonstrating the transmission eigenchannel decomposition
%   involving the propagating part of the S matrix, being S_pr_pr. Script
%   demonstrates wave scattering due to incident shaped waves, exciting
%   various eigenchannels in the quasi-1D geometry. Incidence from left side
%   or the right side of the disorder is given.
script_eigenchannels_S21_S12_quasi_1D
```

`quasi_1D` denotes the geometry of the problem being quasi 1 dimensional. Within the script, the eigenchannel decomposition for the $S_{21}^{pr,pr}$ is estimated using the singular value decomposition as

```
%------------------- Part 1 : Wave incidence from left to right -------------
[U_S21,Sigma_S21,V_S21] = svd(S21_pr_pr);% Eigenchannel modes estimated only for
    propagating modes
tau_S21=diag(Sigma_S21).^2;                % Transmission of eigenchannels
```

This is followed by the excitation of the maximally transmitting eigenchannel of $S_{21}^{pr,pr}$.

```
%-------------------------------------------------------------------------
c_inc=V_S21(:,1);            % Maximally transmitting (First) singular vector
c_trans=S21_pr_pr*c_inc;      % Transmitted eigenchannel wavefront
sprintf('Transmission = %f',sum(abs(c_trans(1:init_data.num_modes_prop).^2)))
% S21_pr_pr contains only propagating modes, hence a truncated transmission
% matrix containing both propagating and evanescent waves on the outgoing side
% and only propagating modes on the incident side is defined for
% visualisation purposes for applying the boundary integral equation.
S21_for_eigen=S21(:,1:init_data.num_modes_prop);
S11_for_eigen=S11(:,1:init_data.num_modes_prop);
% maximally transmitting eigenchannel
field_visualisation_and_comparison_LR(c_inc,Gij_LR,S21_for_eigen,S11_for_eigen,
    init_data,plot_type)
```

Similarly, an intermediate transmitting eigenchannel and a minimally transmitting eigenchannel are plotted. Same analysis is performed for the eigenchannels of $S_{12}^{pr,pr}$ as well. All generated figures are given in the supplementary document.

F. Unshaped wave incidence

A script is provided for performing wave transport using unshaped wave incidence. Examples of wave scattering involving incidence of various eigenmodes (basis wave states) is provided. Wave incidence from both left and the right side of the disorder are given.

```
%------- Script for testing wave scattering for an unshaped incident wave
script_test_wave_scattering
```

Within the script, as an example, the following section performs wave incidence from the left where the 3^{rd} propagating mode is incident.

```
% Plot options are 'real_part','imaginary_part','magnitude'
%plot_type='real_part';
%plot_type='imaginary_part';
plot_type='magnitude';
c_inc=zeros(init_data.num_modes,1);
inc_mode_no=3;            % Incident propagating eigenmode no
c_inc(inc_mode_no)=1;     % Coefficient of the incident wave
% The incident mode could also be evanescent as the following
% c_inc(init_data.num_modes_prop+1)=1;
% The incident mode could be a random propagating wave as the following
% c_inc(1:init_data.num_modes_prop)=(2.*rand(init_data.num_modes_prop,1)-1)+ ...
%                                   1i.*(2.*rand(init_data.num_modes_prop,1)-1);
% c_inc=c_inc./norm(c_inc); % flux normalization

if (~isempty(find(c_inc(1:init_data.num_modes_prop),1)))
c_inc=c_inc./norm(c_inc(1:init_data.num_modes_prop));
% Ensure that the flux of the propagating part of the
%             incident wave has to be normalised to unity
end
field_visualisation_and_comparison_LR(c_inc,Gij_LR,S21,S11,init_data,plot_type)
```

As given above in the commented code section, the incident mode could be evanescent or a random wave. Note that the propagating part of the incident wave is usually normalized so that flux injected by the mode is unity. The plot type can be chosen by setting `plot_type` to `'real_part'`, `'imaginary_part'` or `'magnitude'`.

G. Single mode focusing without ensemble averaging

A sample script is given as the following to give an introduction to single mode focusing without ensemble averaging. As there will be sample specific fluctuations on the focused transmission, this script won't be used to generate results for the paper. Instead Code-Package-2 will be used for generating the result given in the main paper.

```
%-------- Script for visualising single mode focussing ----------------------
script_single_mode_focussing
```

H. Generating additional figures

```
%---- Additional plots for publication ----------------------------------------
script_for_generating_additional_figures
```

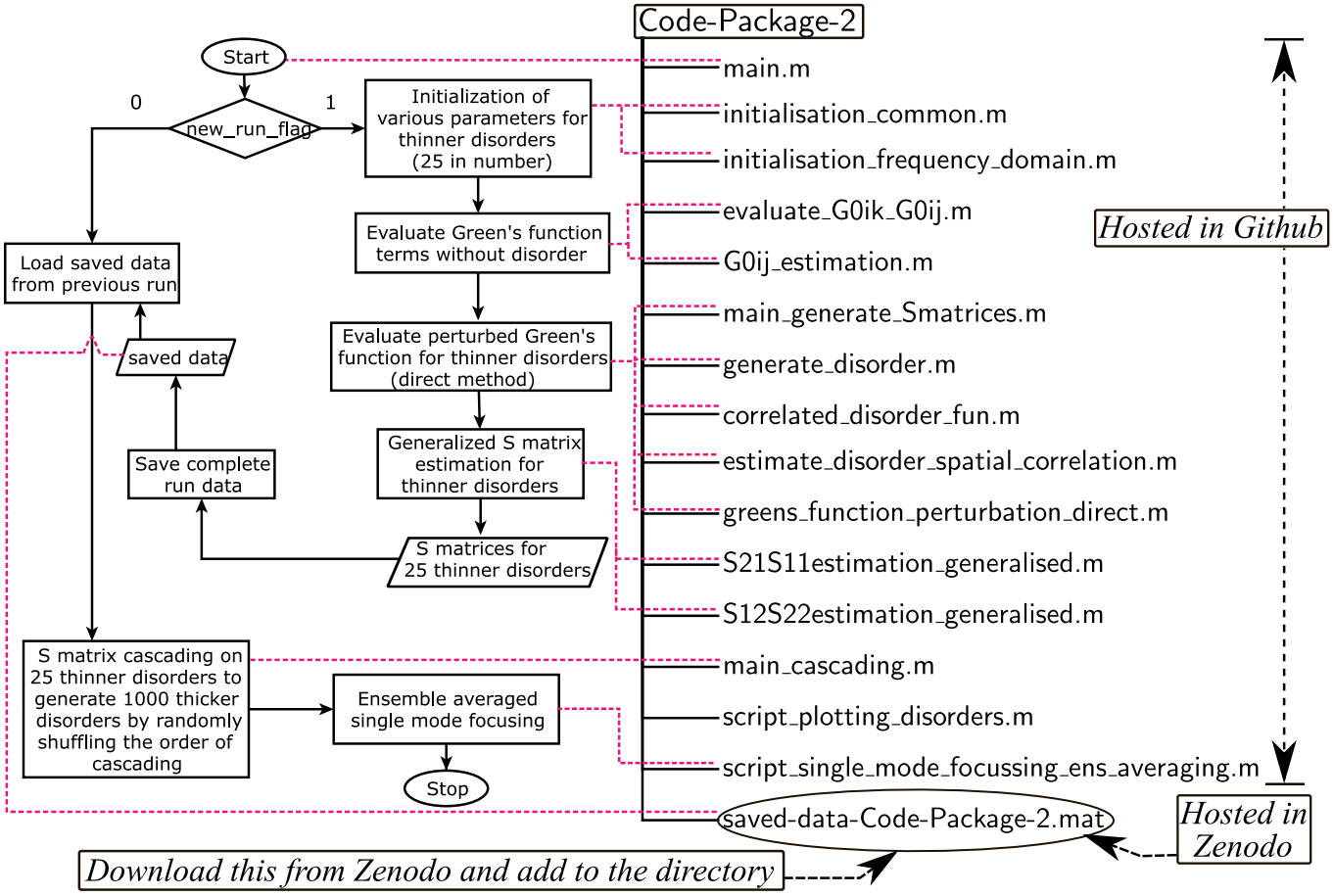

FIG. 3. **Flowchart for summarizing the workflow of Code-Package-1 and its associated** $.m$ **files.**

III. CODE-PACKAGE-2

For Code-Package-2, the flowchart summarizing the operational flow and the associated file structure is given in FIG. 3. The operational flow is similar to that of Code-Package-1 in the sense that if the `new_run_flag` is set to 0, already saved generalized S matrices of thinner disorders will be loaded to generate ensembles of S matrices for thicker cascaded disorders.

A. Initialization

The number of ensemble S matrices (of the thicker disorders) to be generated is set as the following.

```
no_of_thick_slabs=1000; % Ensemble size. These many generalized S matrices
% are generated by randomly shuffling the order of cascading.
```

Once the 1000 generalized S matrices are generated by randomly shuffling the order of cascading by the user, ensemble averaging of the transmission associated with the single mode focusing (presented in the main paper) is performed. In case if the `new_run_flag` is set to 1, the option for a new run from the scratch to generate a set of thin disorders is performed. The script `initialisation_common` is used to initialize the properties of these thinner disorders. Within the script `initialisation_common`, the following is used to initialize 25 thin disorders.

```
%------------------------ Cascading parameters ------------------------------
krefW=909;
no_of_samples=25;
num_modes_evanes=30;
lambda0=632.8*10^-9;        % Wavelength in nm.
```

```
n0=1.6;                          % Reference(Background) medium refractive index.
disorder_perturbation_strength_parameter=0.65; % kept less than 1
krefZ=15;
offset=1;
kreflc_factor=[1];
%-------------------------------------------------------------------------
[init_data] = initialisation_frequency_domain(krefW,krefZ,n0,lambda0,
    num_modes_evanes);
```

Here, `no_of_samples` refers to the number of thin samples of disorders which are taken for cascading later. While defining the thinner disorders, it is recommended not to take the thinner disorders too thin with respect to scattering to avoid numerical precision errors. Once the thinner samples (`krefZ=15` as given above) are defined, the script `main_generate_Smatrices` generates the generalized S matrices of those thinner disorders. Within the script `main_generate_Smatrices`, the following `for` loop estimates the generalized S matrices.

B. Estimation of the generalized S matrices for thinner disorders

Within the following code section, the direct estimation method for the Green's function perturbation (refer the supplementary material) is used. This is implemented using the function `greens_function_perturbation_direct(...)` as the following. Such a direct matrix inversion method is used mainly because it is comparatively faster for thinner disorders.

```
for ens_count=1:no_of_samples
sprintf('Evaluating sample no %d',ens_count)
%--------- estimate the disorder and its properties ----------------------
eps_profile = generate_disorder(init_data);
sprintf('No of grid elements = %d',numel(eps_profile))
%---------- Estimate the disorder properties -----------------------------
[kreflc_main_numerical,var_disorder] = ...
estimate_disorder_spatial_correlation(eps_profile,init_data);
init_data.kreflc_main_numerical_array(ens_count)=kreflc_main_numerical;
init_data.disorder_var_array(ens_count)=var_disorder;
sprintf('Dimensionless spatial correlation length kref*lc= %f',init_data.
    kreflc_main_numerical_array(ens_count))

init_data.eps_profile=eps_profile;
init_data.eps_lin_nonzero=find(eps_profile);
init_data.no_of_disorder_perturbations=numel(init_data.eps_lin_nonzero);
init_data.no_of_grid_elements=(init_data.Nz*init_data.Ny);
%------------------------ Estimate perturbed Green's function -------------
[Gij_R_LR,Gij_L_LR,Gij_LR,Gij_R_RL,Gij_L_RL,Gij_RL]= ...
greens_function_perturbation_direct(G0ik,G0ij,init_data);% G1ij_R_LR : LR means wave
    incident from left to right, R denotes Green's
%              function receiver on the right boundary
% Gij_L_LR : LR means wave incident from left to right, L denotes Green's
%              function receiver on the left boundary
% Gij_LR  : Greens function for the source on the left boundary
% Similarly RL means right to left
%-- Estimate the transmission and reflection matrix, S21 and S11 ---------
[S21,S11] = S21S11estimation_generalised(Gij_L_LR.',Gij_R_LR.',init_data);
%-- Estimate Transmission and reflection matrix, S12 and S22 -------------
[S12,S22] = S12S22estimation_generalised(Gij_R_RL.',Gij_L_RL.',init_data);

S21_array(:,:,ens_count)=S21;
S11_array(:,:,ens_count)=S11;
S12_array(:,:,ens_count)=S12;
S22_array(:,:,ens_count)=S22;
eps_profile_array(:,:,ens_count)=eps_profile;
```

```
pause(1);
close all;
end
```

The workspace data is saved as `saved-data-Code-Package-2.mat` using the following code section within the `main.m` file if it is a new run. If the `new_run_flag` is set to 0, it loads the already saved `saved-data-Code-Package-2.mat` file.

```
if new_run_flag==0
% If a new run is not needed, load the existing saved data.
if isfile('saved-data-Code-Package-2.mat')
% File exists.
load('saved-data-Code-Package-2.mat');
else
% File does not exist and needs to be downloaded from the Zenodo
% repository.
disp('Load the saved-data-Code-Package-2.mat file')
[selected_data_name,data_path] = uigetfile('*.mat');
if strcmp(selected_data_name,'saved-data-Code-Package-2.mat')
load(selected_data_name);
disp('Loaded saved-data-Code-Package-2.mat file')
else
disp('Right .mat not selected !!')
end
end

else
disp('Warning : Overwriting the previously saved data...')
pause(5);
save('saved-data-Code-Package-2.mat') % If a new run is needed, save the workspace
    data
end
```

C. Generalized S matrix cascading of thinner disorders to generate ensembles of thicker diffusive disorders

The script `main_cascading` generates an ensemble of generalized S matrices for thicker diffusive disorders by cascading the S matrices of thinner disorders.

```
%---------- Generate an ensemble of thick scatterers ----------------------
main_cascading;
```

Within the script `main_cascading`, the following code section performs the cascading process. It saves `no_of_thick_slabs` number of generalized S matrix components, the default value being 1000. The generation and data saving of the cascaded ensemble S matrices are left to the user to reduce the size of the code repository. The code section implements the S matrix cascading method given in the supplementary material.

```
%---------------------- load the S matrix elements for cascading ------------
num_modes_prop=init_data.num_modes_prop;
[N,M,ens_tot]=size(S21_array);
%---------------------- Cascading rule --------------------------------------
for ens_cas_count=1:no_of_thick_slabs
index=randperm(ens_tot);
%------------ first cascading -------------------------------------------
S11_1=S11_array(:,:,index(1));
S21_1=S21_array(:,:,index(1));
S12_1=S12_array(:,:,index(1));
S22_1=S22_array(:,:,index(1));
%------------------------------------------------------------------------
S11_2=S11_array(:,:,index(2));
```

```
S21_2=S21_array(:,:,index(2));
S12_2=S12_array(:,:,index(2));
S22_2=S22_array(:,:,index(2));
%-------------------------------------------------------------------------
S11_cas=S11_1 + S12_1*inv(eye(N,M)- S11_2*S22_1)*S11_2*S21_1;
S12_cas=S12_1*inv(eye(N,M)-S11_2*S22_1)*S12_2;
S21_cas=S21_2*inv(eye(N,M)-S22_1*S11_2)*S21_1;
S22_cas=S22_2 + S21_2*inv(eye(N,M)-S22_1*S11_2)*S22_1*S12_2;

%---------- From 3rd slab and beyond ------------------------------------
for cas_count=3:ens_tot

S11_1=S11_cas;
S12_1=S12_cas;
S21_1=S21_cas;
S22_1=S22_cas;

S11_2=S11_array(:,:,index(cas_count));
S21_2=S21_array(:,:,index(cas_count));
S12_2=S12_array(:,:,index(cas_count));
S22_2=S22_array(:,:,index(cas_count));
%---------- cascading ---------------------------------------------------
S11_cas=S11_1 + S12_1*inv(eye(N,M)- S11_2*S22_1)*S11_2*S21_1;
S12_cas=S12_1*inv(eye(N,M)-S11_2*S22_1)*S12_2;
S21_cas=S21_2*inv(eye(N,M)-S22_1*S11_2)*S21_1;
S22_cas=S22_2 + S21_2*inv(eye(N,M)-S22_1*S11_2)*S22_1*S12_2;
%-------------------------------------------------------------------------
S21_prop=S21_cas(1:num_modes_prop,1:num_modes_prop);  %% extract propagating TM
S11_prop=S11_cas(1:num_modes_prop,1:num_modes_prop);  %% extract propagating RM
S12_prop=S12_cas(1:num_modes_prop,1:num_modes_prop);  %% extract propagating TM
S22_prop=S22_cas(1:num_modes_prop,1:num_modes_prop);  %% extract propagating RM

end

tau_avg=trace(S21_prop'*S21_prop)/num_modes_prop;

filename=sprintf('S21_cas_%d.mat',ens_cas_count);
save(filename,'S21_cas');
filename=sprintf('S12_cas_%d.mat',ens_cas_count);
save(filename,'S12_cas');
filename=sprintf('S11_cas_%d.mat',ens_cas_count);
save(filename,'S11_cas');
filename=sprintf('S22_cas_%d.mat',ens_cas_count);
save(filename,'S22_cas');
sprintf('Cascaded sample no %d with tau_avg = %.4f',ens_cas_count,tau_avg)
end
```

D. Ensemble averaging of single mode focusing via phase conjugation

Upon generating 1000 cascaded generalized S matrices, single mode focusing outside the disorder in transmission is implemented next by using the script `script_single_mode_focussing_ens_averaging`.

```
%---------- Ensemble averaging of optimal transmission ------------------
script_single_mode_focussing_ens_averaging
```

Within the script `script_single_mode_focussing_ens_averaging`, first the focusing of a propagating mode is estimated which is followed by the focusing of an evanescent mode. These two separate scenarios of focusing in transmission via phase conjugation are ensemble averaged using the 1000 generalized transmission matrices obtained

before. Focusing of a propagating mode (mode number to be focused is set by `mode_num_to_focus_pr`), is given as the following. One may refer the main paper for the theory behind.

```
%------ Focussing of a propagating mode focussing via phase conjugation ---
mode_num_to_focus_pr=10;                    % The ith mode to be focussed
I_m_pr=zeros(init_data.num_modes_prop,1); % Column vector to choose the mode
%     to focus
I_m_pr(mode_num_to_focus_pr)=1;

clearvars S21_cas S21_big_array S21_prop
c_R_opt_pr_pr=zeros(init_data.num_modes_prop,no_of_thick_slabs);
T_focus_pr_pr=zeros(1,no_of_thick_slabs);
T_background_pr_pr=zeros(1,no_of_thick_slabs);

for scount=1:no_of_thick_slabs
scount
clearvars S12_cas S21_cas
load(sprintf('S12_cas_%d.mat',scount));
load(sprintf('S21_cas_%d.mat',scount));

S12_pr_pr=S12_cas(1:init_data.num_modes_prop,1:init_data.num_modes_prop);
% Purely propagating S21 without evanescent modes
S21_pr_pr=S21_cas(1:init_data.num_modes_prop,1:init_data.num_modes_prop);
% Purely propagating S21 without evanescent modes

%---------- Start by exciting the mode to be focussed from the right side
% of the disorder, to be phase conjugated again from the left to focus back
% on the right side.
c_L_pr_pr=S12_pr_pr*I_m_pr; % outgoing wave on the left due the single mode
% incidence from the right
c1=1/norm(c_L_pr_pr);  % Flux normalization term for phase conjugation to the
% right side.
c_R_opt_pr_pr(:,scount)=S21_pr_pr*(c1.*(conj(S12_pr_pr))*I_m_pr);
%Phase conjugation to right side of the slab
% Here, (c1.*(conj(S12_pr_pr))*I_m_pr) is the incident
% flux-normalized phase conjugate wave from the left.

T_focus_pr_pr(scount)=abs(c_R_opt_pr_pr(mode_num_to_focus_pr,scount))^2;
% Transmission just due to the focus
T_background_pr_pr(scount)=sum(abs(c_R_opt_pr_pr(1:end ~= mode_num_to_focus_pr,
    scount)).^2);

end
```

Next is the focusing onto an evanescent mode, implemented by the following code section. `mode_num_to_focus_ev` is used to set the evanescent mode number to be focused.

```
%------ Focussing of an evanescent mode focussing via phase conjugation ---
mode_num_to_focus_ev=4;                    % The ith mode to be focussed
I_m_ev=zeros(init_data.num_modes_evanes,1); % Column vector to choose the mode
%     to focus
I_m_ev(mode_num_to_focus_ev)=1;

clearvars S21_cas S21_big_array S21_prop S12_cas S21_pr_pr S12_pr_pr
c_R_opt_ev_ev=zeros(init_data.num_modes_evanes,no_of_thick_slabs);
c_R_opt_pr_ev=zeros(init_data.num_modes_prop,no_of_thick_slabs);
T_background_pr_ev=zeros(1,no_of_thick_slabs);

for scount=1:no_of_thick_slabs
scount
```

```
clearvars S12_cas S21_cas
load(sprintf('S12_cas_%d.mat',scount));
load(sprintf('S21_cas_%d.mat',scount));

S12_pr_ev=S12_cas(1:init_data.num_modes_prop,(1+init_data.num_modes_prop):end);
% Incident modes evanescent, transmitted modes propagating
S21_ev_pr=S21_cas((1+init_data.num_modes_prop):end,1:init_data.num_modes_prop);

S21_pr_pr=S21_cas(1:init_data.num_modes_prop,1:init_data.num_modes_prop);
% Purely propagating S21 without evanescent modes


c_L_pr_ev=S12_pr_ev*I_m_ev; % outgoing propagating wave on the left
% due the incident single evanescent mode on
% the right
c1=1/norm(c_L_pr_ev);  % Flux normalization term for phase conjugation to the
% right side.

c_R_opt_ev_ev(:,scount)=S21_ev_pr*(c1.*(conj(S12_pr_ev))*I_m_ev);
% This is the evanescent part of the focussed wave
% Phase conjugation to right side of the slab
% Here, (c1.*(conj(S12_pr_ev))*I_m_ev) is the incident
% flux-normalized phase conjugate wave from the left.
c_R_opt_pr_ev(:,scount)=S21_pr_pr*(c1.*(conj(S12_pr_ev))*I_m_ev);
% This is the propagating part of the focussed wave
T_background_pr_ev(scount)=c_R_opt_pr_ev(:,scount)'*c_R_opt_pr_ev(:,scount);
end
```

Finally, the results given in the main paper are plotted.