Computational Statistics
M.Sc. Mathematical Engineering
Prof. A. Manzoni

A.A. 2021 – 2022
Politecnico di Milano

Hands-On 8
Sensitivity Analysis

We rely on the `SAFE` toolbox in `Matlab` for sensitivity analysis, see `http://www.safetoolbox.info`, where the package can be freely downloaded. The SAFE Toolbox provides a set of functions to perform Global Sensitivity Analysis. It implements several methods, including the Elementary Effects Test, Variance-Based (Sobol') sensitivity analysis and the PAWN method. Originally developed for the `Matlab` environment, `SAFE` is also available in R and Python. An introduction to the SAFE Toolbox is provided in:

Pianosi, F., Sarrazin, F., Wagener, T. (2015), A Matlab toolbox for Global Sensitivity Analysis, Environmental Modelling & Software, 70, 80-85.

1 Warm-up: the Ishigami-Homma function

Using a relatively simple example, let us first show step-by-step how the code works. We consider the widely-used Ishigami-Homma function,

$$f(\mathbf{X}) = \sin(X_1) + a\sin^2(X_2) + bX_3^4\sin(X_1)$$

where all $X_i \sim \mathcal{U}(-\pi, \pi)$. It can be noticed that: *(i)* X_1 seems to be the most influential input; *(ii)* X_2 seems to be non-influential.

To visualize the scatterplots (see the script `workflow_visual_ishigami_homma.m`) it is possible to use the following commands; results are reported in Figure 1.

```
fun_test  = 'ishigami_homma_function' ;
M = 3 ;
distr_fun = 'unif' ;
distrpar =  [ -pi pi ];

% sampling and model evaluation
N = 3000;
X = AAT_sampling('lhs',M,'unif',distrpar,N);
Y = model_evaluation(fun_test,X)          ;

% Scatter plots
scatter_plots(X,Y)
```

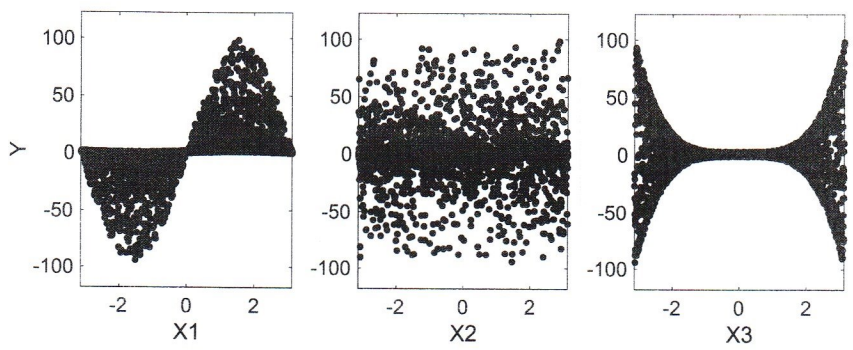


Figure 1: Scatterplot of the function values against each of the input factors.

This is confirmed by variance-based SA: in fact, the total effects index of X_1 is the highest and the total effects of X_2 is almost zero; these numbers can be also derived analytically.

Indeed, it is possible to show that

$$\mathrm{Var}\,[Y] = \frac{a^2}{8} + \frac{b\pi^4}{5} + \frac{b^2\pi^8}{18} + \frac{1}{2}$$

$$V_1 = \mathrm{Var}_1[\mathbb{E}_{\mathbf{X}_{\sim 1}}[Y \mid X_1]] = \frac{1}{2}\left(1 + \frac{b\pi^4}{5}\right)^2, \quad V_2 = \mathrm{Var}_2[\mathbb{E}_{\mathbf{X}_{\sim 2}}[Y \mid X_2]] = \frac{a^2}{8}, \quad V_3 = \mathrm{Var}_3[\mathbb{E}_{\mathbf{X}_{\sim 3}}[Y \mid X_3]] = 0$$

and that

$$\mathbb{E}_{\mathbf{X}_{\sim 1}}[\mathrm{Var}_1[Y|\mathbf{X}_{\sim 1}]] = \frac{1}{2}\left(1 + \frac{b\pi^4}{5}\right)^2 + \frac{8b^2\pi^8}{225}, \qquad \mathbb{E}_{\mathbf{X}_{\sim 2}}[\mathrm{Var}_2[Y|\mathbf{X}_{\sim 2}]] = \frac{a^2}{8}, \qquad \mathbb{E}_{\mathbf{X}_{\sim 3}}[\mathrm{Var}_3[Y|\mathbf{X}_{\sim 3}]] = \frac{8b^2\pi^8}{225}.$$

In `Matlab` (see the script `workflow_vbsa_ishigami_homma.m`) we can compute Sobol' indices using the function `vbsa_indices` and, in this case, compare the approximate results with the analytical expressions of those indices, which can be computed analytically for the case at hand.

```
% Setup the model and define input ranges
fun_test  = 'ishigami_homma_function' ;
M = 3 ;
distr_fun = 'unif' ;
distrpar =  [ -pi pi ];

% Compute the exact values of the output variance (V) and of the first-order
% (Si_ex) and total-order (STi_ex) variance-based sensitivity indices
[ tmp, V, Si_ex, STi_ex ] = ishigami_homma_function(rand(1,M)) ;

% Sample parameter space:
SampStrategy = 'lhs' ;
N = 3000 ;
X = AAT_sampling(SampStrategy,M,distr_fun,distrpar,2*N);

% Apply resampling strategy for the efficient approximation of the indices:
[ XA, XB, XC ] = vbsa_resampling(X) ;

% Run the model and compute selected model output at sampled parameter sets:
YA = model_evaluation(fun_test,XA) ; % size (N,1)
YB = model_evaluation(fun_test,XB) ; % size (N,1)
YC = model_evaluation(fun_test,XC) ; % size (N*M,1)

% Compute main (first-order) and total effects:
[ Si, STi ] = vbsa_indices(YA,YB,YC);
```

In particular, we run the model and compute selected model output at sampled parameter; note the formal similarity with the quantities involved in MC formulas seen during Lectures to approximate Sobol' indices. We obtain

```
   main    total
X1:   0.3778   1.0677
X2:  -0.0116  -0.0477
X3:  -0.0259   0.6175

sum:  0.3403  1.6375
```

We can then plot main and total effects and compare the values estimated by the function `vbsa_indices` with the exact values, also analyzing convergence of sensitivity indices; results are reported in Figures 2– 3.

```
% Plot main and total effects and compare the values estimated
% by the function 'vbsa_indices' with the exact values
figure
```

```
subplot(121); boxplot2([ Si; Si_ex])
subplot(122); boxplot2([STi; STi_ex])
legend('estimated','exact')% add legend

% Analyze convergence of sensitivity indices:
NN = [N/5:N/5:N] ;
[ Si, STi ] = vbsa_convergence([YA;YB;YC],M,NN);
figure
subplot(121); plot_convergence(Si,NN*(M+2),[],[],Si_ex, ...
     'model evals','main effect')
subplot(122); plot_convergence(STi,NN*(M+2),[],[],STi_ex, ...
     'model evals','total effect')
```

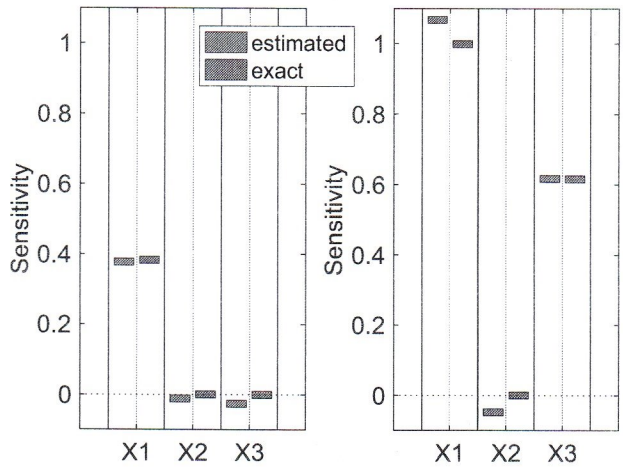


Figure 2: Computed Sobol' indices and analytical expressions (available for the case at hand).

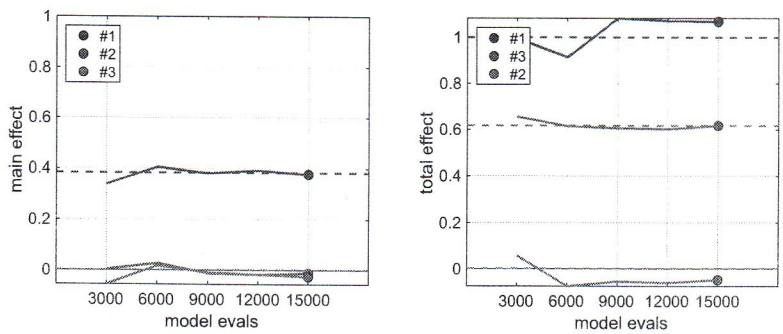


Figure 3: Convergence of sensitivity indices with respect to the number of output evaluations.

We now apply the PAWN approach, obtaining the three sets of conditional CDFs in the panels of Figure 4. In all panels, the red dashed line is the empirical unconditional output CDF $\hat{F}_Y(\cdot)$ while the grey lines are the conditional CDFs $\hat{F}_{Y|X_i}(\cdot)$ $(i = 1, 2, 3)$ obtained at $n = 10$ different conditioning values of X_i. These values can be read on the horizontal axis of the panels of Figure 5, which report the (estimated) Kolomogorov-Smirnov statistics . The horizontal line is the value $c(\alpha)\sqrt{(N_u + N_c)/(N_u N_c)}$ (with $\alpha = 0.05$) for the two-sample Kolmogorov-Smirnov test. We consider, as experimental setup: $n = 10$ conditioning points, $N_u = 150$, and $N_c = 100$.

The Figures show that:

- Visual analysis of the CDFs (Figure 4) immediately shows that X_1 and X_3 are both much more influential than X_2, as their conditional CDFs are more widespread around the unconditional one (red line), while those of X_2 almost overlap with the unconditional CDF.
- In quantitative terms, X_1 is the most influential, as shown by the analysis of the Kolmogorov-Smirnov statistic. In particular, the PAWN sensitivity indices are equal to

$$T_1 = 0.48, \qquad T_2 = 0.14, \qquad T_3 = 0.3$$

  if considering the median (stat = median) and

$$T_1 = 0.53, \qquad T_2 = 0.19, \qquad T_3 = 0.35$$

  if considering the max (stat = max). In other words, when used for Factor Prioritisation, PAWN provides the same ranking as variance-based SA.
- By applying the two-sample Kolmogorov-Smirnov test for Factor Fixing, one would conclude that X_2 is non-influential (at confidence level $\alpha = 0.05$) because its KS statistics are consistently below the threshold value (see Figure 5). This is again consistent with a qualitative judgement that might be formulated based on variance-based SA results where the total effects index of X_2 is very low ($S_{T_2} = 0.0009$).

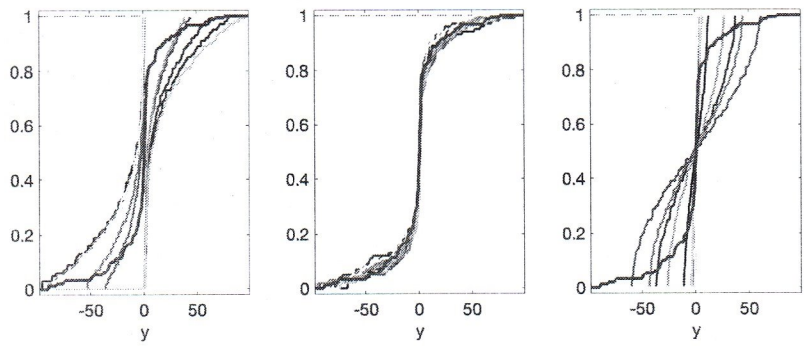


Figure 4: Empirical unconditional output distribution $\hat{F}_Y$ (red dashed lines) and conditional ones $\hat{F}_{Y|X_i}$ (grey solid lines).

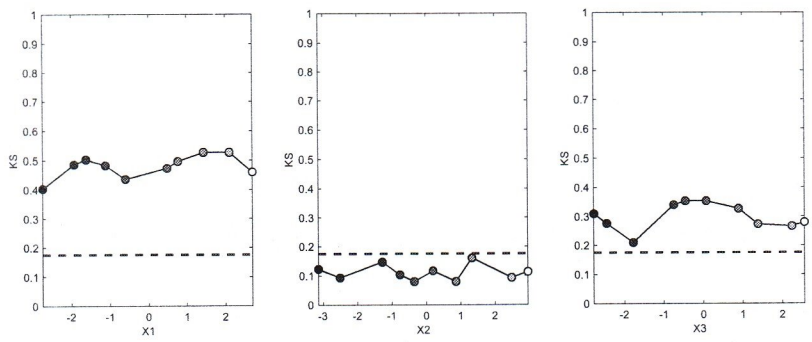


Figure 5: Kolmogorov-Smirnov statistic $\hat{KS}(x_i)$ at different conditioning values of x_i. The dashed horizontal line is the critical value of the KS statistic at confidence level of 0.05.

```
% Define uncertain inputs (parameters):
M = 3 ; % number of inputs

labelparams={ 'X1','X2','X3'} ; % input names
% parameter ranges:
```

```
xmin = [-pi -pi -pi ];
xmax = [ pi  pi  pi ];
distrpar=cell(M,1); for i=1:M; distrpar{i}=[xmin(i) xmax(i)]; end

% Define model output:
fun_test  = 'ishigami_homma_function' ;

% Apply PAWN
NU = 150 ; % number of samples to estimate unconditional CDF
NC = 100 ; % number of samples to estimate conditional CDFs
n  = 10 ; % number of conditioning points

% Create input/output samples to estimate the unconditional output CDF:
Xu = AAT_sampling('lhs',M,'unif',distrpar,NU); % matrix (NU,M)
Yu = model_evaluation(fun_test,Xu)  ; % vector (1,M)

% Create input/output samples to estimate the conditional output CDFs:
[ XX, xc ] = pawn_sampling('lhs',M,'unif',distrpar,n,NC);
YY = pawn_ishigami_homma(fun_test,XX) ;

% Estimate unconditional and conditional CDFs:
[ YF, Fu, Fc  ] = pawn_cdfs(Yu,YY) ;

% Plot CDFs:
figure
for i=1:M
   subplot(1,M,i)
   pawn_plot_cdf(YF, Fu, Fc(i,:),[],'y')
end

% Compute KS statistics:
KS = pawn_ks(YF,Fu,Fc) ;

% Plot KS statistics:
figure
for i=1:M
   subplot(1,M,i)
   pawn_plot_kstest(KS(:,i),NC,NU,0.05,xc{i},labelparams{i})
end

% Compute PAWN index by taking a statistic of KSs (e.g. max):
Pi = max(KS);
```

The function `pawn_ishigami_homma` is a slight variation of the function `pawn_model_evaluation` already provided by the `SAFE` toolbox. With the following commands, we can plot the PAWN indices, see Figure 6, left, then we can use bootstrapping to assess robustness of PAWN indices; results are reported in Figure 6, right.

```
% Plot
figure
boxplot1(Pi,labelparams)

% Use bootstrapping to assess robustness of PAWN indices:
stat = 'max' ; % statistic to be applied to KSs
Nboot = 100  ; % number of boostrap resamples
[ T_m, T_lb, T_ub ] = pawn_indices(Yu,YY,stat,[],Nboot);

% Plot:
figure; boxplot1(T_m,labelparams,[],T_lb,T_ub)
```

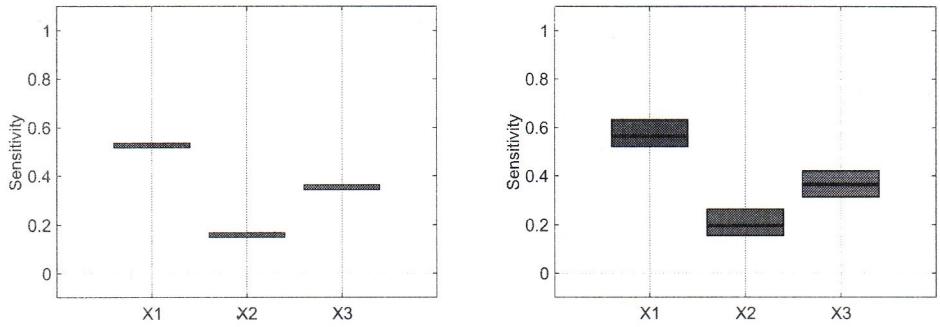


Figure 6: Left: PAWN indices. Right: robustness of PAWN indices through bootstrapping.

Finally, we perform a convergence analysis, choosing 5 different values for N_c and N_u (see Figure 7).

```
% Convergence analysis:
stat = 'max' ; % statistic to be applied to KSs
NCb = [ NC/10 NC/2 NC ] ;
NUb = [ NU/10 NU/2 NU ] ;

[ T_m_n, T_lb_n, T_ub_n ] = pawn_convergence( Yu, YY, stat, ...
       NUb, NCb,[],Nboot );
NN = NUb+n*NCb ;
figure; plot_convergence(T_m_n,NN,T_lb_n,T_ub_n,[], ...
      'no of evals',[],labelparams)
```

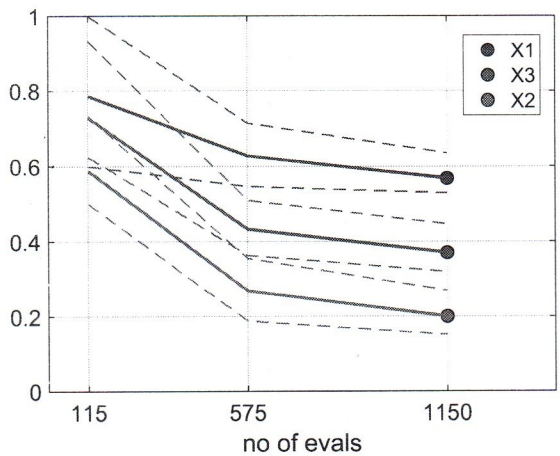


Figure 7: Convergence of PAWN indices.

For the sake of completeness, below we report the function `ishigami_homma_function`.

```
function [ y, V, Si_ex, STi_ex ] = ishigami_homma_function( x )

% [ y, V, Si_ex, STi_ex ] = ishigami_homma_function(x)
%
% x = vector of inputs x(1),x(2),x(3)                    - vector (1,3)
%       y = output                                              - scalar
%       V = output variance (*)                                 - scalar
%  Si_ex = first-order sensitivity indices (*)             - vector (1,3)
% STi_ex = total-order sensitivity indices (*)             - vector (1,3)
% (*) = exact value computed analytically
```

```
a = 2 ;
b = 1 ;
y = sin(x(1)) + a * sin(x(2))^2 + b*x(3)^4*sin(x(1)) ;

% By model definition, we should get:
% VARy = VAR(Y) = 1/2 + a^2/8 + b*pi^4/5 + b^2*pi^8/18
% V1 = VAR(E(Y|X1)) = 1/2 + b*pi^4/5 + b^2*pi^8/50
% V2 = VAR(E(Y|X2)) = a^2/8
% V3 = VAR(E(Y|X3)) = 0
% V13 = VAR(E(Y|X1,X3)) = b^2*pi^4/18 - b^2*pi^8/50
% V12 = V23 = V123 = 0
% and thus:
% ST1 = S1 + S13       ST2 = S2       ST3 = S13

V  = 1/2+a^2/8+b*pi^4/5+b^2*pi^8/18 ;
Si_ex(1) = ( 1/2 + b*pi^4/5 + b^2*pi^8/50 )/V ;
Si_ex(2) = a^2/8 / V ;
Si_ex(3) = 0 ;
STi_ex(1) = Si_ex(1) + ( b^2*pi^8/18 - b^2*pi^8/50 )/V ;
STi_ex(2) = Si_ex(2) ;
STi_ex(3) = ( b^2*pi^8/18 - b^2*pi^8/50 )/V ;
```

2 A time-dependent model: HyMod

A second example deals with a time-dependent model, called *HyMod*, which is a lumped conceptual hydrological model that can be used to simulate rainfall-runoff processes at the catchment scale[1]. This model is characterised by five storages, the soil moisture reservoir e represented by a Pareto distribution function to describe the rainfall excess model, three linear reservoirs in series mimicking the fast runoff component, and one slow reservoir, see Figure 26. HyMod has five parameters:

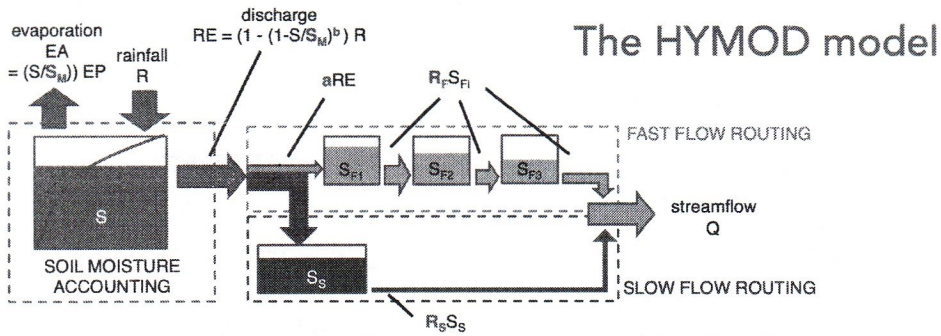
- X_1, the maximum soil moisture storage capacity $S_m \in [0, 400]$;
- X_2, the coefficient of spatial variability of soil moisture $b \in [0, 2]$;
- X_3, the ratio of effective rainfall to the fast pathway $a \in [0, 1]$;
- X_4, the discharge coefficient of the slow reservoirs $R_S \in [0, 0.1]$;
- X_5, the discharge coefficient of the fast reservoirs $R_F \in [0.1, 1]$.

During simulation, differential equations are solved using the forward explicit Euler method with a daily resolution time series of rainfall (mm/day) and evaporation (mm/ day) over a simulation horizon of two years starting from 10/10/ 1948. We consider:

- as a scalar output Y the maximum predicted flow over the simulation horizon, and
- as inputs X_i, $i = 1, \ldots, 5$ the model parameters, in their feasible ranges of variation.

The idea is to carry out a sensitivity analysis of this model, exploiting variance-based GSA techniques, the elementary effects method and a moment-independent method (the PAWN approach).

[1] See, e.g., Francesca Pianosi and Thorsten Wagener. A simple and efficient method for global sensitivity analysis based on cumulative distribution functions. Environmental Modelling & Software, 67:1–11, 2015, and references therein.



		meaning	min	max
1	SM	maximum soil moisture (mm)	0	400
2	beta	exponent in the soil moisture routine (-)	0	2
3	alfa	partition coefficient (-)	0	1
4	Rs	slow reservoir coefficient (-)	0	0.1
5	Rf	fast reservoir coefficient (-)	0.1	1

Boyle, D. (2001). Multicriteria calibration of hydrological models. PhD thesis, Dep. of Hydrol. and Water Resour., Univ. of Ariz., Tucson.

Wagener, T., D. Boyle, M. J. Lees, H. S. Wheater, H. V. Gupta, and S. Sorooshian (2001), A framework for development and application of hydrological models, *Hydrol. Earth Syst. Sci.*, 5(1), 13-26.

Figure 8: Hymod model: idea, input parameters and output.

Problem 8.1 (Variance-based, Elementary Effects and PAWN sensitivity analysis, built-in example). Run the scripts provided in the `SAFE` toolbox, in order to:

1. Compute first-order (main effects) and total-order (total effects) variance-based indices. The script `workflow_vbsa_hymod` also shows how to repeat computations after adding up new input/output samples, and how to compute indices when dealing with multiple outputs.
2. Apply the elementary effects method, using the script `workflow_eet_hymod`. The EE method computes two indices for each input, the mean of the EEs, which measures the total effect of an input over the output, and the standard deviation of the EEs, which measures the degree of interactions with the other inputs.
3. Apply the PAWN sensitivity analysis approach, which exploits empirical CDFs and KS statistics, to perform moment-independent sensitivity analysis. using the script `workflow_pawn_hymod`. Recall that one advantage of the PAWN sensitivity index is that it can be very easily focused on specific sub-ranges of the output. For instance estimate the PAWN indices over the range $Y > 50$.

1. It is possible to run the Steps 2 and 3 of the `Matlab` script `workflow_vbsa_hymod` using as input-output function `hymod_max`. This latter function runs the rainfall-runoff Hymod model and returns the maximum flow in the simulated time series. Below we report the main commands, as well as the plots obtained.

```
% Load data:
load -ascii LeafCatch.txt
rain = LeafCatch(1:365,1)    ;
evap = LeafCatch(1:365,2)    ;
flow = LeafCatch(1:365,3)    ;

% Define input distribution and ranges:
M   = 5 ; % number of uncertain parameters [ Sm beta alfa Rs Rf ]
```

```
DistrFun   = 'unif'   ; % Parameter distribution and ranges
DistrPar   = { [ 0 400 ]; [ 0 2 ]; [ 0 1 ]; [ 0 0.1] ; [ 0.1 1 ] } ;
myfun = 'hymod_max'; %'hymod_nse' ;

SampStrategy = 'lhs' ;
N = 3000 ; % Base sample size.
X = AAT_sampling(SampStrategy,M,DistrFun,DistrPar,2*N);
[ XA, XB, XC ] = vbsa_resampling(X) ;

YA = model_evaluation(myfun,XA,rain,evap); %,flow) ; % size (N,1)
YB = model_evaluation(myfun,XB,rain,evap); %,flow) ; % size (N,1)
YC = model_evaluation(myfun,XC,rain,evap); %,flow) ; % size (N*M,1)

% Compute main (first-order) and total effects:
[ Si, STi ] = vbsa_indices(YA,YB,YC);

% Plot results:
X_Labels = {'Sm','beta','alfa','Rs','Rf'} ;
figure% plot main and total separately
subplot(121); boxplot1(Si,X_Labels,'main effects')
subplot(122); boxplot1(STi,X_Labels,'total effects')
figure % plot both in one plot:
boxplot2([Si; STi],X_Labels)
legend('main effects','total effects')% add legend
```

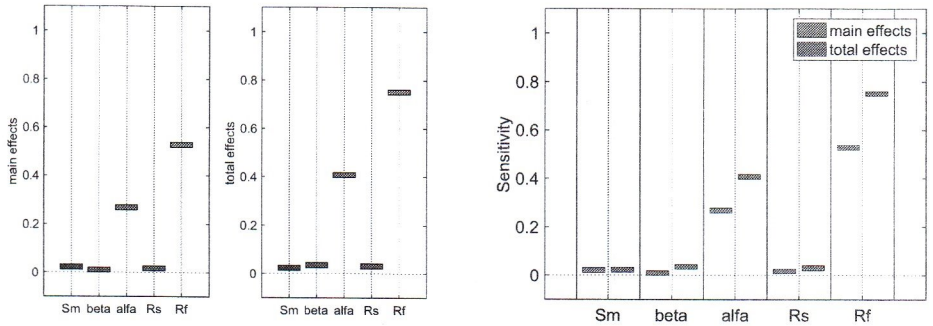


Figure 9: Hymod model: main effects and total effects.

```
% Check the model output distribution (if multi-modal or highly skewed,
% the variance-based approach may not be adequate):
Y = [ YA; YC ] ;
figure; plot_cdf(Y,'MAX') % 'NSE'
figure; plot_pdf(Y,'MAX');% 'NSE'
```

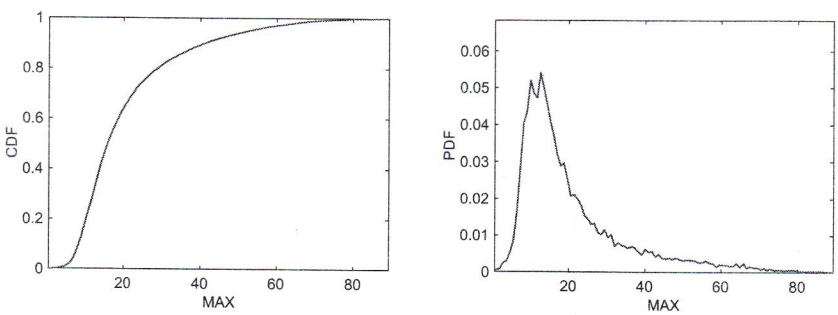


Figure 10: Hymod model: empirical CDF and PDF of the output quantity of interest.

```
% Compute confidence bounds:
Nboot = 500 ; % number of resamples used for boostrapping
[ Si, STi, Si_sd, STi_sd, Si_lb, STi_lb, Si_ub, STi_ub ] = ...
                                        vbsa_indices(YA,YB,YC,Nboot);

% Plot:
figure % plot main and total separately
subplot(121); boxplot1(Si,X_Labels,'main effects',Si_lb,Si_ub)
subplot(122); boxplot1(STi,X_Labels,'total effects',STi_lb,STi_ub)
figure % plot both in one plot:
boxplot2([Si; STi],X_Labels,[ Si_lb; STi_lb ],[ Si_ub; STi_ub ])
legend('main effects','total effects')
```

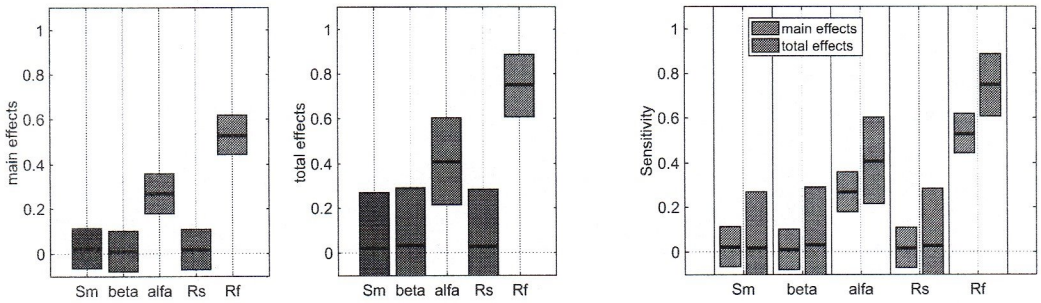


Figure 11: Hymod model: robustness of Sobol' indices through bootstrapping

```
% Analyze convergence of sensitivity indices:
NN = [N/10:N/10:N] ;
[ Sic, STic ] = vbsa_convergence([YA;YB;YC],M,NN);
figure
subplot(121);
plot_convergence(Sic,NN*(M+2),[],[],[],'model evals', ...
                 'main effect',X_Labels)
subplot(122);
plot_convergence(STic,NN*(M+2),[],[],[],'model evals', ...
                 'total effect',X_Labels);
```

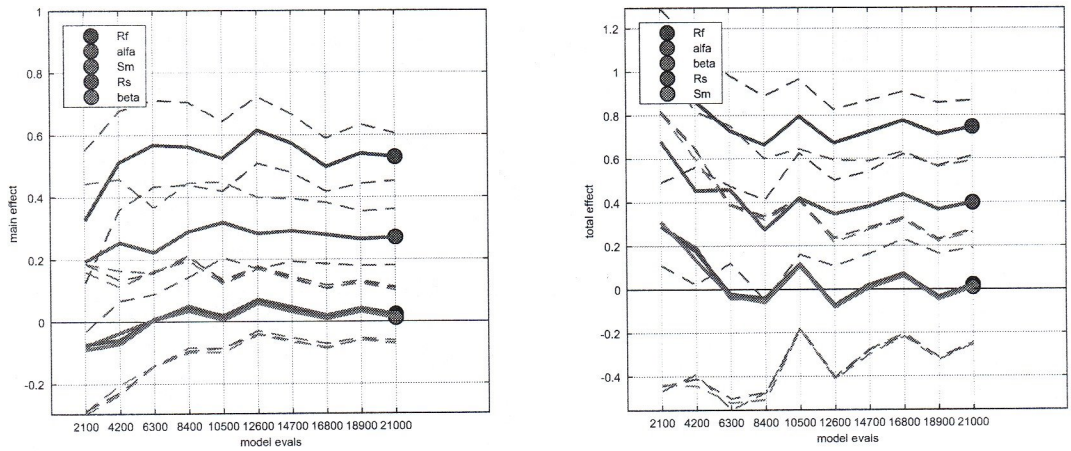


Figure 12: Hymod model: convergence of the Sobol' indices (with confidence bound).

It is also possible to perform a time-dependent (variance-based) sensitivity analysis, computing Sobol' indices at each time step (without introducing the generalized Sobol' indices) following the `Matlab` script `workflow_tvsa_hymod`. In this case we consider the simulation of the Hymod rainfall-runoff model over time, which returns the time series of simulated flow (a vector of size $(T, 1)$, where T is the number of time steps) – the time-varying model output is thus the model prediction (estimated flow) at each time step of the simulation. For the first warmup period (30 days, over 730) sensitivity indices are not computed.

```
% Load data:
load -ascii LeafCatch.txt
rain = LeafCatch(1:730,1)    ;
evap = LeafCatch(1:730,2)    ;
flow = LeafCatch(1:730,3)    ;

% Define inputs:
DistrFun   = 'unif'   ; % Parameter distribution
% Parameter ranges (from literature):
xmin = [   0 0 0 0    0.1 ] ;
xmax = [ 400 2 1 0.1 1    ] ;
% Parameter names (needed for plots):
x_labels = {'Sm','beta','alfa','Rs','Rf'} ;

myfun = 'hymod_sim' ;
warmup = 30 ; % lenght of model warmup period (days)
% (sensitivity indices will not be computed for the warmup period)

GSA_met = 'VBSA' ;
N = 3001 ; % sample size (needed for all other methods)
SampStrategy = 'lhs' ; % Sampling strategy (needed for EET and VBSA)

M = length(xmin) ; % Number of inputs
DistrPar = cell(M,1) ; for i=1:M; DistrPar{i} = [ xmin(i) xmax(i) ] ; end

    X = AAT_sampling(SampStrategy,M,DistrFun,DistrPar,2*N);
    [ XA, XB, XC ] = vbsa_resampling(X) ;
    YA = model_evaluation(myfun,XA,rain,evap) ; % size (N,T)
    YB = model_evaluation(myfun,XB,rain,evap) ; % size (N,T)
    YC = model_evaluation(myfun,XC,rain,evap) ; % size (N*M,T)
    Y = [ YA; YB; YC ] ; % (N*(2+M),T) ... only needed for the next plot!

% Plot results:
figure
plot(Y','b')
xlabel('time')
ylabel('flow')
axis([1,size(Y,2),0,max(max(Y))])

%% Step 5 Compute time-varying Sensitivity Indices

T = size(Y,2) ;

    Si  = nan(T,M) ;
    STi = nan(T,M) ;
    for t=warmup:T
        [ Si(t,:), STi(t,:) ] = vbsa_indices(YA(:,t),YB(:,t),YC(:,t));
    end
    % select sensitivity index to be plotted in the next figure:
    S_plot = Si' ;
    %S_plot = STi' ;

% Plot results:
```

```
figure
clrs = autumn ;
clrs = clrs(end:-1:1,:);
colormap(clrs)
imagesc(S_plot)
xlabel('time')
ylabel('inputs')
set(gca,'YTick',1:M,'YTickLabel',x_labels)
colorbar
title(['Sensitivity indices - GSA meth: ' GSA_met ])
hold on
plot(M+0.5-flow/max(flow)*M,'k','LineWidth',2)
```

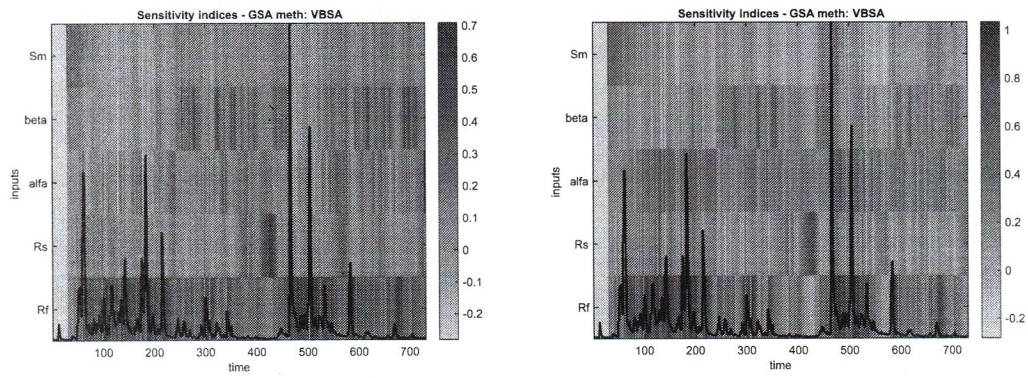


Figure 13: Hymod model: time-dependent main effects (left) and total effects (right).

2. We consider now the elementary effect (EE) method, again considering as quantity of interest the maximum flux.

```
% Load data:
load -ascii LeafCatch.txt
rain = LeafCatch(1:365,1)    ;
evap = LeafCatch(1:365,2)    ;
flow = LeafCatch(1:365,3)    ;

% Number of uncertain parameters subject to SA:
M    = 5 ;
% Parameter ranges (from literature):
xmin = [  0 0 0 0   0.1 ];
xmax = [400 2 1 0.1 1   ];
% Parameter distributions:
DistrFun  = 'unif'  ;
DistrPar = cell(M,1);
for i=1:M; DistrPar{i} = [ xmin(i) xmax(i) ] ; end
% Name of parameters (will be used to costumize plots):
X_labels = {'Sm','beta','alfa','Rs','Rf'} ;

% Define output:
myfun = 'hymod_max' ; % nse

%% Step 3 (sample inputs space)

r = 100 ; % Number of Elementary Effects
% [notice that the final number of model evaluations will be equal to
% r*(M+1)]
```

```
% option 1: use the sampling method originally proposed by Morris (1991):
% L = 6  ; % number of levels in the uniform grid
% design_type  = 'trajectory'; % (note used here but required later)
% X = Morris_sampling(r,xmin,xmax,L); % (r*(M+1),M)

% option 2: Latin Hypercube sampling strategy
SampStrategy = 'lhs' ; % Latin Hypercube
design_type = 'radial';
% other options for design type:
%design_type  = 'trajectory';
X = OAT_sampling(r,M,DistrFun,DistrPar,SampStrategy,design_type);

%% Step 4 (run the model)
Y = model_evaluation(myfun,X,rain,evap); %,flow) ; % size (r*(M+1),1)

%% Step 5 (Computation of the Elementary effects)

% Compute Elementary Effects:
[ mi, sigma ] = EET_indices(r,xmin,xmax,X,Y,design_type);

% Plot results in the plane (mean(EE),std(EE)):
EET_plot(mi, sigma,X_labels )

% Use bootstrapping to derive confidence bounds:
Nboot=100;
[mi,sigma,EE,mi_sd,sigma_sd,mi_lb,sigma_lb,mi_ub,sigma_ub] = ...
EET_indices(r,xmin,xmax,X,Y,design_type,Nboot);

% Plot bootstrapping results in the plane (mean(EE),std(EE)):
EET_plot(mi,sigma,X_labels,mi_lb,mi_ub,sigma_lb,sigma_ub)
```

We obtain the following means and standard deviations of the EEs:

	mean(EE)	std(EE)
X1:	3.906	5.707
X2:	3.833	5.750
X3:	23.937	21.515
X4:	5.080	4.070
X5:	36.179	30.560

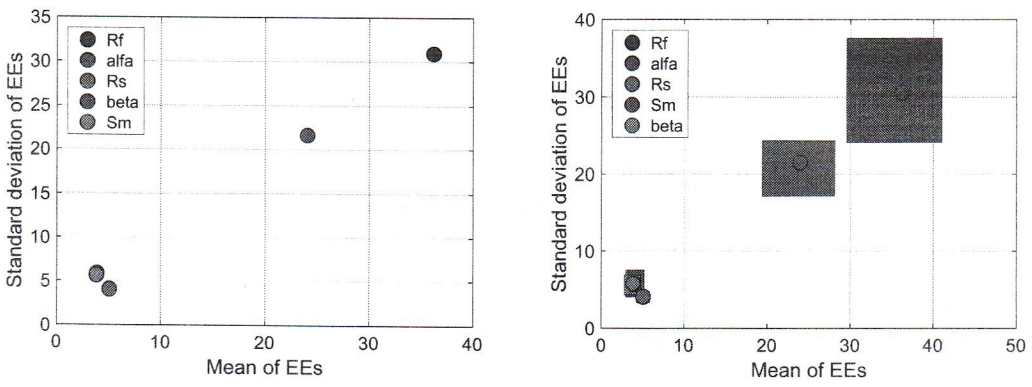


Figure 14: Left: Elementary effects are reported in the plane (mean(EE),std(EE)) . Right: bootstrapping has been used to derive confidence bounds.

```
% Repeat computations using a decreasing number of samples so as to assess
% if convergence was reached within the available dataset:
rr = [ r/5:r/5:r ] ;
m_r = EET_convergence(EE,rr);
```

```
% Plot the sensitivity measure (mean of elementary effects) as a function
% of model evaluations:
figure; plot_convergence(m_r,rr*(M+1),[],[],[],...
'no of model evaluations','mean of EEs',X_labels)

% Repeat convergence analysis using bootstrapping:
Nboot = 100;
rr = [ r/5:r/5:r ] ;
[m_r,s_r,m_lb_r,m_ub_r] = EET_convergence(EE,rr,Nboot);
% Plot the sensitivity measure (mean of elementary effects) as a function
% of model evaluations:
figure; plot_convergence(m_r,rr*(M+1),m_lb_r,m_ub_r,[],...
'no of model evaluations','mean of EEs',X_labels)
```

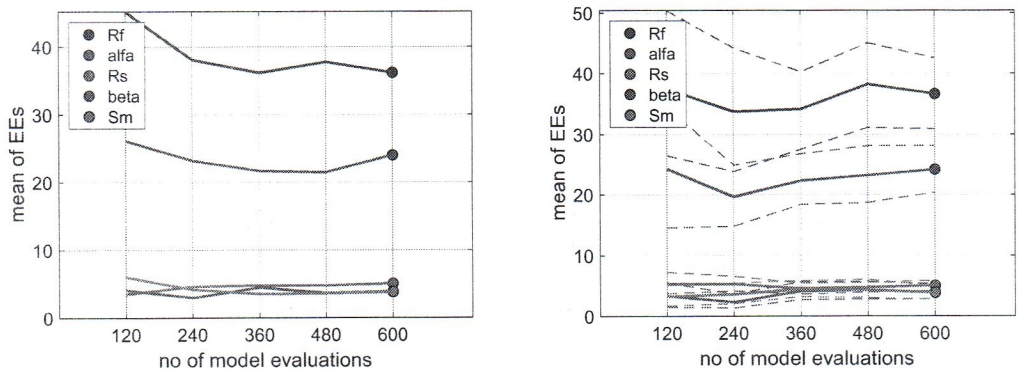


Figure 15: Left: sensitivity measure (mean of elementary effects) as a function of model evaluations. Here computations are repeated using a decreasing number of samples to assess convergence. Right: convergence analysis has been repeated using bootstrapping.

3. We now consider the evaluation of the PAWN indices.

```
%% Step 2: setup the Hymod model
load -ascii LeafCatch.txt
rain = LeafCatch(1:730,1)    ;
evap = LeafCatch(1:730,2)    ;
flow = LeafCatch(1:730,3)    ;

% Define uncertain inputs (parameters):
M = 5 ; % number of inputs
labelparams={ 'SM','beta','alfa','Rs','Rf' } ; % input names
% parameter ranges:
xmin = [  0 0 0 0   0.1 ];
xmax = [400 2 1 0.1 1   ];
distrpar=cell(M,1); for i=1:M; distrpar{i}=[xmin(i) xmax(i)]; end

% Define model output:
fun_test = 'hymod_max';

NU = 150 ; % number of samples to estimate unconditional CDF
NC = 100 ; % number of samples to estimate conditional CDFs
n  = 10 ; % number of conditioning points

% Create input/output samples to estimate the unconditional output CDF:
Xu = AAT_sampling('lhs',M,'unif',distrpar,NU); % matrix (NU,M)
Yu = model_evaluation(fun_test,Xu,rain,evap)  ; % vector (1,M)
```

```
% Create input/output samples to estimate the conditional output CDFs:
[ XX, xc ] = pawn_sampling('lhs',M,'unif',distrpar,n,NC);
YY = pawn_model_evaluation(fun_test,XX,rain,evap) ;

% Estimate unconditional and conditional CDFs:
[ YF, Fu, Fc  ] = pawn_cdfs(Yu,YY) ;

% Plot CDFs:
figure
for i=1:M
   subplot(1,M,i)
   pawn_plot_cdf(YF, Fu, Fc(i,:),[],'y (max flow)')
end
```

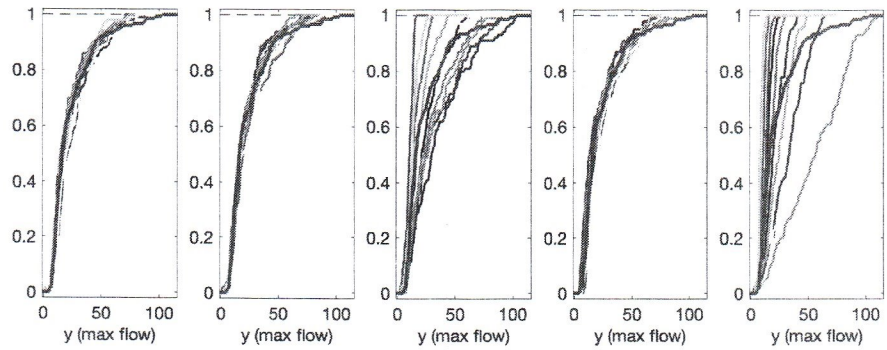


Figure 16: Empirical unconditional output distribution $\hat{F}_Y$ (red dashed lines) and conditional ones $\hat{F}_{Y|X_i}$ (grey solid lines).

```
% Compute KS statistics:
KS = pawn_ks(YF,Fu,Fc) ;

% Plot KS statistics:
figure
for i=1:M
   subplot(1,M,i)
   pawn_plot_kstest(KS(:,i),NC,NU,0.05,xc{i},labelparams{i})
end
```

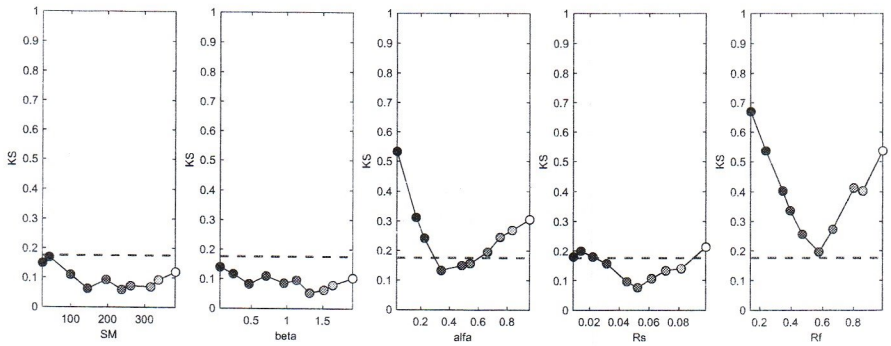


Figure 17: Kolmogorov-Smirnov statistic $\hat{KS}(x_i)$ at different conditioning values of x_i. The dashed horizontal line is the critical value of the KS statistic at confidence level of 0.05.

```
% Compute PAWN index by taking a statistic of KSs (e.g. max):
Pi = max(KS);

% Plot:
figure; boxplot1(Pi,labelparams)

% Use bootstrapping to assess robustness of PAWN indices:
stat = 'max' ; % statistic to be applied to KSs
Nboot = 100  ; % number of boostrap resamples
[ T_m, T_lb, T_ub ] = pawn_indices(Yu,YY,stat,[],Nboot);

% Plot:
figure; boxplot1(T_m,labelparams,[],T_lb,T_ub)
```

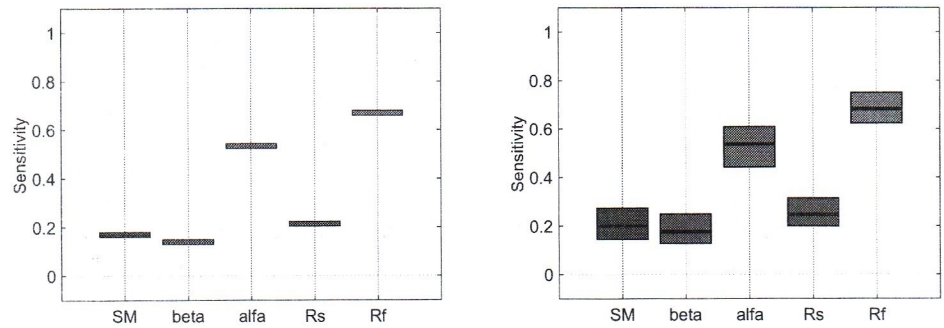


Figure 18: Left: PAWN indices. Right: robustness of PAWN indices through bootstrapping.

```
% Convergence analysis:
stat = 'max' ; % statistic to be applied to KSs
NCb = [ NC/10 NC/2 NC ] ;
NUb = [ NU/10 NU/2 NU ] ;

[ T_m_n, T_lb_n, T_ub_n ] = pawn_convergence( Yu, YY, ...
                                stat, NUb, NCb,[],Nboot );
NN = NUb+n*NCb ;
figure; plot_convergence(T_m_n,NN,T_lb_n, ...
                                T_ub_n,[],'no of evals',[],labelparams)
```

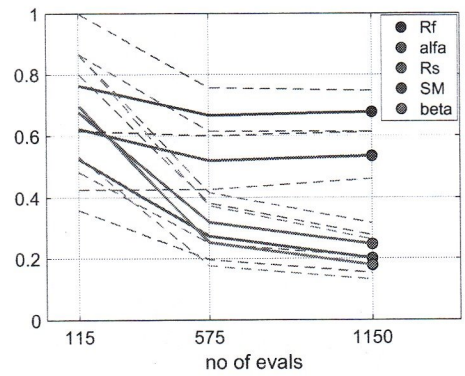


Figure 19: Convergence of PAWN indices.

In order to apply the PAWN method to a sub-region of the output range, for instance considering only output values above a given threshold, it is sufficient to use the commands below.

```
thres = 50 ;
[ T_m2, T_lb2, T_ub2 ]= pawn_indices( Yu, YY, ...
                                    stat,[], Nboot,[],'above',thres ) ;

% Plot:
figure; boxplot1(T_m2,labelparams,[],T_lb2,T_ub2)
```

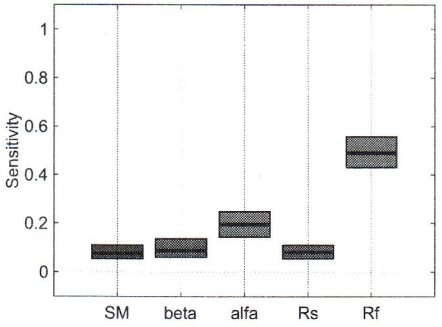


Figure 20: PAWN indices in the constrained case $Y > 50$.

Note that the influence of the parameters on the output constrained above a given threshold can be different compared to the unconstrained case.

3 A cantilever beam model

Let us consider a uniform cantilever beam with horizontal and vertical loads. The beam length L and displacement tolerance D_0 at the free end of the beam are problem constants, with values $L = 100\ in$, and $D_0 = 2.2535\ in$; the parameters w and t are the width and the thickness of the cross-section, respectively (see Figure 21).

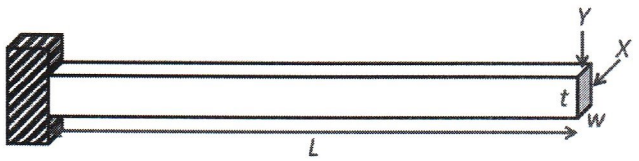


Figure 21: Uniform cantilever beam with horizontal and vertical loads.

Consider, for the time being, $w = t = 1\ in$. The output quantities of interest are the stress (lb/in^2)

$$S = \frac{600}{wt^2}Y + \frac{600}{w^2t}X, \qquad S \leq R$$

and the displacement (in)

$$D = \frac{4L^3}{Ewt}\sqrt{\left(\frac{Y}{t^2}\right)^2 + \left(\frac{X}{w^2}\right)^2}, \qquad D \leq D_0,$$

depending on four random input parameters:

- the Young's modulus[2] $E \sim N(2.9 \cdot 10^7, (1.45 \cdot 10^6)^2)$ (lb/in^2);
- the horizontal load $X \sim N(500, 100^2)$ (lb);
- the vertical load $Y \sim N(1000, 100^2)$ (lb).

[2]The Young's modulus (also referred to as modulus of elasticity) is the ratio of stress to strain in elastic range of deformation. In the American system, its unit of measure is pound per square inch, abbreviation psi. For typical metals, modulus of elasticity is in the range between $6.5 \cdot 10^6$ psi (45 GPa) to $59 \cdot 10^6$ psi (407 GPa).

Problem 8.2 (A cantilever beam model).

1. Under `Examples`, create a subfolder `cantilever` where to save the function `cantilever_function` to be implemented, returning the two outputs S and D depending on the three inputs E, X, Y.
2. Perform a variance-based sensitivity analysis computing the Sobol' indices for both output quantities of interest, and rank the input parameters by decreasing importance.
3. Perform a PAWN sensitivity analysis for the displacement D first, then focusing on a specific sub-range of the output, as the range $D > 1.5$.
4. Now include also width $w \sim \mathcal{U}(0.8, 1.2)$ (in) and thickness $t \sim \mathcal{U}(0.8, 1.2)$ (in) as possible input parameters, and perform again the variance-based sensitivity analysis.

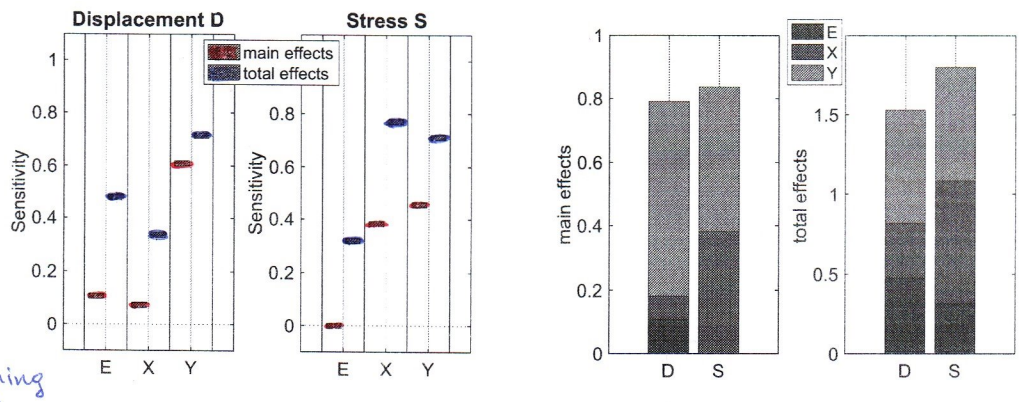
1. The following code provides the evaluations of the model:

```
function [output] = cantilever_function(xx, w, t)

E = xx(1);
X = xx(2);
Y = xx(3);

L = 100;
D_0 = 2.2535;

if (nargin == 1)
    w = 1;
    t = 1;
elseif (nargin == 2)
    t = 1;
end

Sterm1 = 600*Y / (w*(t^2));
Sterm2 = 600*X / ((w^2)*t);

Dfact1 = 4*(L^3) / (E*w*t);
Dfact2 = sqrt((Y/(t^2))^2 + (X/(w^2))^2);

output(1) = Dfact1 * Dfact2; % D
output(2) = Sterm1 + Sterm2; % S

end
```

2. With the following commands, we can perform a variance-based sensitivity analysis.

```
%% Step 1: setup the model and define input ranges

% Define input distribution and ranges:
M   = 3 ; % number of uncertain parameters [ Sm beta alfa Rs Rf ]
DistrFun    = 'norm'   ; % Parameter distribution and ranges
DistrPar    = { [ 2.9e7 1.45e6 ]; [ 500 100 ]; [ 1000 100 ] } ;

%% Step 2: Compute first-order and total-order variance-based indices
myfun = 'cantilever_function' ;
SampStrategy = 'lhs' ;
N = 15000 ; % Base sample size.

X = AAT_sampling(SampStrategy,M,DistrFun,DistrPar,2*N);
[ XA, XB, XC ] = vbsa_resampling(X) ;

% Run the model and compute selected model output at sampled parameters:
```

```
YA = model_evaluation(myfun,XA); %,flow) ; % size (N,1)
YB = model_evaluation(myfun,XB); %,flow) ; % size (N,1)
YC = model_evaluation(myfun,XC); %,flow) ; % size (N*M,1)

%% Step 3: Evaluation of the Sobol indices

% % select the j-th model output:
j = 1 ;  % D
[ Si1, STi1 ] = vbsa_indices(YA(:,j),YB(:,j),YC(:,j));
j = 2 ;  % S
[ Si2, STi2 ] = vbsa_indices(YA(:,j),YB(:,j),YC(:,j));

% check: S does not depend on E
Si2(1) = 0;

% Compare boxplots:
X_Labels = {'E','X','Y'} ;

figure
subplot(121)
boxplot2([Si1; STi1],X_Labels)
title('Displacement D')
subplot(122)
boxplot2([Si2; STi2],X_Labels)
legend('main effects','total effects')
title('Stress S')

% Use stacked bar to put all outputs on one plot:
figure
subplot(121)
stackedbar([Si1; Si2],[],'main effects',[],{'D','S'})
subplot(122)
stackedbar([STi1; STi2],X_Labels,'total effects',[],{'D','S'})
```



here we're performing sensitivity analysis for 2 different outputs (D,S) therefore the analisi are different!

Figure 22: Sobol indices (main effects and total effects) for the displacement D and the stress S as outputs.

We can remark how the stress S does not depend on E; the Monte Carlo evaluation provides a slightly negative value of S_E, and a positive value of S_{T_E}; this is a point that should be further investigated, from the standpoint of convergence of MC estimates.

3. Regarding the PAWN method, we only consider the displacement D, forcing the `cantilever_function` function to return just `output(1)`.

```
% Define uncertain inputs (parameters):
M = 3 ; % number of inputs
labelparams={ 'E','X','Y' } ; % input names
```

```
DistrFun  = 'norm'  ; % Parameter distribution
DistrPar  = { [ 2.9e7 1.45e6 ]; [ 500 100 ]; [ 1000 100 ] } ;

% Define model output:
fun_test = 'cantilever_function';

NU = 150 ; % number of samples to estimate unconditional CDF
NC = 100 ; % number of samples to estimate conditional CDFs
n  = 10 ; % number of conditioning points

% Create input/output samples to estimate the unconditional output CDF:
Xu = AAT_sampling('lhs',M,'norm',DistrPar,NU); % matrix (NU,M)
Yu = model_evaluation(fun_test,Xu)  ; % vector (1,M)

% Create input/output samples to estimate the conditional output CDFs:
[ XX, xc ] = pawn_sampling('lhs',M,'norm',DistrPar,n,NC);
YY = pawn_cantilever(fun_test,XX) ;

% Estimate unconditional and conditional CDFs:
[ YF, Fu, Fc  ] = pawn_cdfs(Yu,YY) ;

% Plot CDFs:
figure
for i=1:M
   subplot(1,M,i)
   pawn_plot_cdf(YF, Fu, Fc(i,:),[],'D (displacement)')
end
```

action of one output (the displacement D) of the 3 inputs

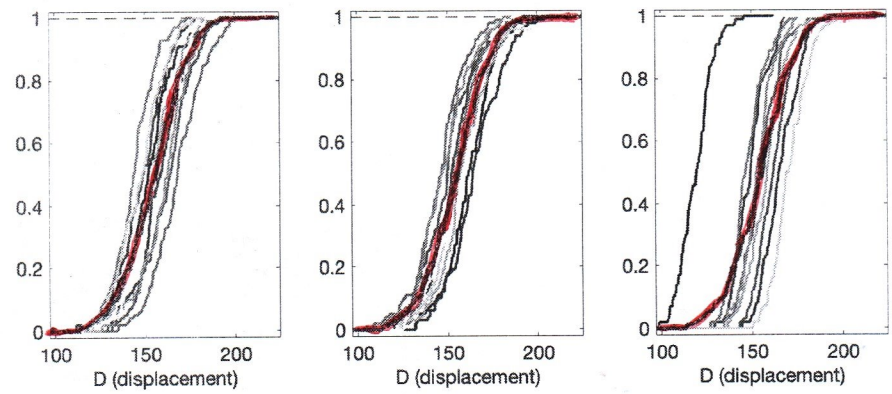


Figure 23: Empirical unconditional output distribution $\hat{F}_Y$ (red dashed lines) and conditional ones $\hat{F}_{Y|X_i}$ (grey solid lines).

```
% Compute KS statistics:
KS = pawn_ks(YF,Fu,Fc) ;

% Plot KS statistics:
figure
for i=1:M
   subplot(1,M,i)
   pawn_plot_kstest(KS(:,i),NC,NU,0.05,xc{i},labelparams{i})
end
```

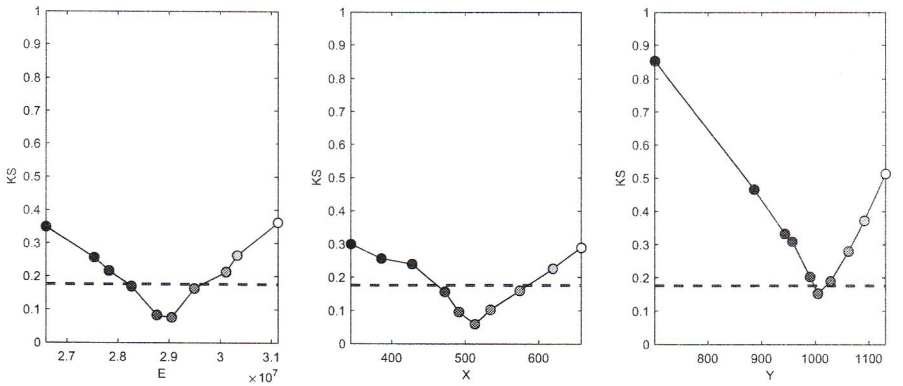


Figure 24: Kolmogorov-Smirnov statistic $\hat{KS}(x_i)$ at different conditioning values of x_i. The dashed horizontal line is the critical value of the KS statistic at confidence level of 0.05.

```
% Compute PAWN index by taking a statistic of KSs (e.g. max):
Pi = max(KS);

% Plot:
figure
boxplot1(Pi,labelparams)

% Use bootstrapping to assess robustness of PAWN indices:
stat = 'max' ; % statistic to be applied to KSs
Nboot = 100  ; % number of boostrap resamples
[ T_m, T_lb, T_ub ] = pawn_indices(Yu,YY,stat,[],Nboot);

% Plot:
figure; boxplot1(T_m,labelparams,[],T_lb,T_ub)

% Convergence analysis:
stat = 'max' ; % statistic to be applied to KSs
NCb = [ NC/10 NC/2 NC ] ;
NUb = [ NU/10 NU/2 NU ] ;

[ T_m_n, T_lb_n, T_ub_n ] = pawn_convergence( Yu, YY, ...
                                    stat, NUb, NCb,[],Nboot );
NN = NUb+n*NCb ;
figure; plot_convergence(T_m_n, ...
              NN,T_lb_n,T_ub_n,[],'no of evals',[],labelparams)
```

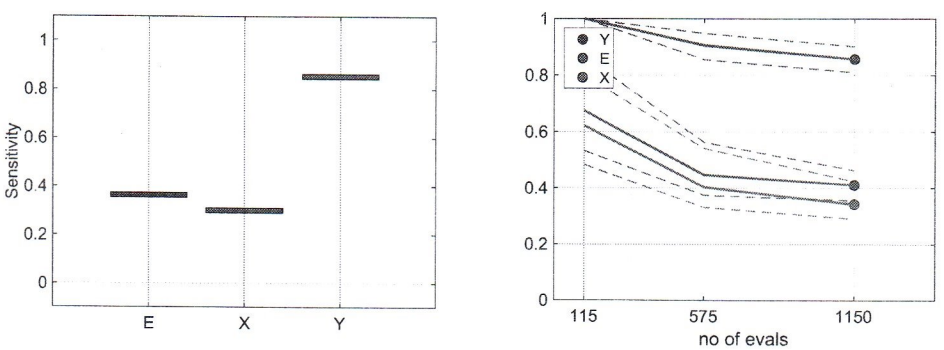


Figure 25: Left: PAWN indices. Right: convergence of PAWN indices.

```
% Compute the PAWN index over a sub-range of the output distribution
thres = 1.5 ;
[ T_m2, T_lb2, T_ub2 ]= pawn_indices( Yu, YY, ...
            stat,[], Nboot,[],'above',thres ) ;

% Plot:
figure; boxplot1(T_m2,labelparams,[],T_lb2,T_ub2)
```

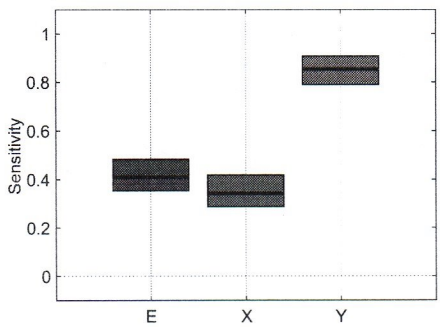


Figure 26: PAWN indices in the constrained case $D > 1.5$.

From the PAWN analysis, the most relevant parameter on the output D seems to be Y, coherently with the Sobol' indices evaluation. Restricting the output to the range $D > 1.5$ does not seem to affect the sensitivity analysis.

4. The analysis including the two additional parameters can be carried out by slightly modifying the script above and for the sake of brevity is not reported.

4 The simplest model from epidemiology, yet not that easy...

The SIR model

sort of Lodka-Volterra model with 3 species rather than 2 : there are parameters and we want to understand if β or r are relevant or not (and for which quantities)

$$\begin{cases} \dot{S} = & -\dfrac{\beta}{N_{pop}} IS \\ \dot{I} = & \dfrac{\beta}{N_{pop}} IS - rI \\ \dot{R} = & rI \end{cases}$$

is the simplest epidemiological model, and describes the time-evolution of three compartments: individuals (S)usceptible to the disease, individuals (I)nfected with the disease, and finally individuals (R)emoved from the disease dynamics (either because they recovered, assuming immunity after having contracted the disease, or died). The total number of individuals in the population $N_pop = S+I+R$ is supposed constant, and individuals transition from one compartment to the next one with certain transition rates β, r.

SIR-like models can be written as ODE systems for a state vector X with N_{states} components. The evolution of the system depends on N_{coef} coefficients $\mathbf{p} = (p_1, \ldots, p_{N_{coef}})$ – in the case of the SIR model, $\mathbf{p} = (\beta, r)$ – and on the N_{states} initial conditions $\mathbf{q} = (q_1, \ldots, q_{N_{states}})$. Our goal is to monitor some related quantities Y (Quantities of Interest) which can be derived from X by an observation operator G, that in turn might depend on some hyper-parameters $\mathbf{h}$:

$$\begin{cases} \dot{X} = f(X, \mathbf{p}), & t \in (0, T) \\ X(0) = \mathbf{q} \\ Y(t) = G(X(t), \mathbf{h}) \end{cases}$$

Possible quantities of interest might be

- the peak-time of number of infected persons,

$$G(X) = \arg \max_{t \in [0,T]} I(t);$$

- the cumulative number of infected persons:

$$G(X) = \int_0^T I(t)dt,$$

also called incidence data (as opposed to prevalence (instantaneous) data $G(X(t)) = I(t)$);

- the peak-time of the new infected persons in a time-window of length Δ:

$$G(X, \Delta) = \arg\max \int_t^{t+\Delta} \frac{\beta}{N_{pop}} S(s)I(s)ds.$$

Problem 8.3 (A SIR epidemic model).
Consider a SIR model with initial conditions

$$S(0) = 0.95, \qquad I(0) = 0.05, \qquad R(0) = 0.$$

and choose these ranges for the parameters:

$$\beta \in [0.25, 0.35], \qquad r \in [0.06, 0.18].$$

We assume that a-priori we have no knowledge that any value of β, r is more plausible than others; therefore, we assume that β, r are uniform independent random variables; moreover, we solve the SIR system with Matlab's `ode45` up to final time $T = 150$. In this case, the Sobol decomposition of any quantity of interest Y reads

$$1 = s_\beta + s_r + s_{\beta r}$$

and the total Sobol indices can be computed as

$$s_{T_\beta} = s_\beta + s_{\beta r}, \qquad s_{T_r} = s_r + s_{\beta r}.$$

1. Implement a solver of the SIR model using `Matlab`'s `ode45`.
2. Compute the SIR dynamics by 100 Monte Carlo samples, obtained by sampling β and r independently, plot the obtained trajectories of S, I and B, highlighting the sample average trajectories.
3. Report the PDFs of the following quantities of interest: SIR states at $T = 30$ (after the average peak position) and $T = 100$ (when the dynamics is over); peak time for I; peak value for I.
4. Compute the Sobol indices for the SIR compartments S, I and R, as functions of time. For this specific application, evaluating Sobol' indices (and not their *generalized* version) is a good option.
5. Show the variability of the trajectories if the range of r is reduced to $[0.06, 0.1]$, again evaluating the SIR dynamics by 100 Monte Carlo samples, and highlighting the sample average trajectories.

The quantities computed in this exercises are partially inspired by the recent work [1]

1. To use `ode45` in `Matlab` we must define a function, `SIR_model`, which returns the right-hand side $\mathbf{f}$ of the ODE system $\mathbf{y}' = \mathbf{f}(t, \mathbf{y}(t))$; here this function depends on the parameters β and r.

```
function F = SIR_model(t,y,N_pop,param_vector)

% param_vector = [beta;r] = [contact prob; 1/recovery time]
        % y = [S; I; R]
        %
        % S' = - beta/N SI
        % I' =   beta/N SI - r I
        % R' =   r I

        beta = param_vector(1);
        r = param_vector(2);
        S = y(1);
        I = y(2);
```

```
        F = [ -beta/N_pop*S*I;
               beta/N_pop*S*I - r*I;
               r*I];
return
```

Since ode45 returns times and first output, and the values of the solution are given as row vectors, we can implement a simple wrapper in order to return the solution values as column vectors for UQ purposes. Moreover, it can be useful to get values at the specified times (such as the final time, or the entire time span). In particular, if we want the solution at the final time, the output is

```
[ S(t_end);
  I(t_end);
  R(t_end)]
```

while if we want the entire profile, it is helpful to have it as a single (huge) column vector where we pile up

```
[S(t0);
 S(t1);
 S(t2);
 ...
 I(t0);
 ...
 R(t0);
 ...]
```

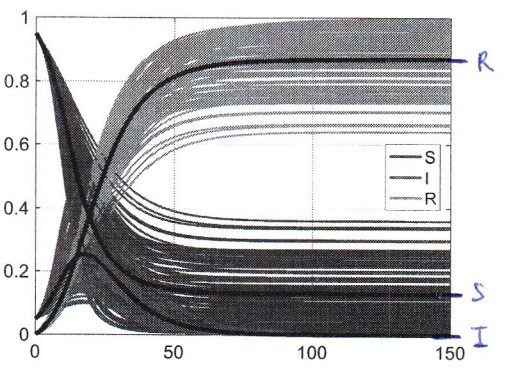
Then, the function `fold_sol_to_matrix` folds this vector to

```
[ S(t0)  S(t1)  S(t2) ...
  I(t0)  I(t1)  I(t2) ...
  R(t0)  R(t1)  R(t2) ...]
```

```
function [qoi,time_steps] = ode45_wrap(N_pop,param_vector,Tspan,y0, ...
                                        value_to_extract,with_time_steps)

[T_SIR,SIR] = ode45(@(t,y) SIR_model(t,y,N_pop,param_vector),Tspan,y0);

switch value_to_extract
    case 'final_time'
        qoi = SIR(end,:)';
    case 'all_times'
        qoi = SIR(:);
    otherwise
        error('unknown kind of output')
end

if nargin == 6 && strcmp(with_time_steps,'with_time_steps')
    time_steps = T_SIR;
else
    time_steps = [];
end
end

function SOL = fold_sol_to_matrix(sol_vect,N_var)

% SOL = fold_sol_to_matrix(sol_vect,N_var)
% Reshape the vector sol_vect into the matrix SOL

len = length(sol_vect);
each_len = len/N_var;

SOL = reshape(sol_vect,[each_len N_var])';
```

2. Here is the code to compute the SIR dynamics by 100 MC samples, plotting the obtained trajectories and the sample mean. The result is reported in Figure 27.

```
% model setting
N_pop = 1;    N_UQ = 2;
N_states = 3;
y0 = [0.95; 0.05; 0];

Tspan = 0:0.5:150;

% parameter bounds
beta_min = 0.25;    beta_max = 0.35;
r_min   = 0.06;       r_max   = 0.18;

domain = [beta_min r_min; beta_max r_max];
MCsize = 100;

% sample uniformly
unit_set = rand(MCsize,2);
MC(:,1) = beta_min + (beta_max - beta_min)*unit_set(:,1);
MC(:,2) = r_min + (r_max - r_min)*unit_set(:,2);

SIR_evals_MC = zeros(N_states*size(Tspan,2),MCsize);

for j = 1: MCsize
    SIR_evals_MC(:,j) = ode45_wrap(N_pop, MC(j,:), Tspan,y0,'all_times');
end
mean_SIR = mean(SIR_evals_MC,2);

% plot MC realizations
for i=1:MCsize
    plot(Tspan,fold_sol_to_matrix(SIR_evals_MC(:,i),N_states),'LineWidth',2)
    hold on
    set(gca,'ColorOrderIndex',1)
end

% plot MC expected value
plot(Tspan,fold_sol_to_matrix(mean_SIR,N_states),'LineWidth',4,'Color','k')

legend('S','I','R')
set(legend,'Location','East')
grid on
set(gca,'FontSize',20)
```



– we notice that the uncertainty on the long term prediction can be remarkable (any of the trajectory has been computed by picking up a value from the intervals regarding β and r)

Figure 27: 100 realizations of the SIR dynamics and MC means of S, I and R obtained sampling uniformly the parameter space.

3. Let us now compute the PDFs of the following quantities of interest: SIR states at $T = 30$ and $T = 100$. The following code allows us to exploit the command `ksdensity` and plot the computed PDFs:

```
figure
selected_t = [100];
for curr_t = selected_t

    [~,j] = find(Tspan==curr_t);
    [S_values_MC,I_values_MC,R_values_MC] = ...
                        get_values_MC_at_t(SIR_evals_MC,j,MCsize,N_states);

    [S_pdf_MC,S_xi_MC] = ksdensity(S_values_MC, ...
                        'support',[0 1],'NumPoints',40);

    plot(S_xi_MC,S_pdf_MC,'o','LineWidth',2,'MarkerSize',10, ...
                        'DisplayName','pdf of S, MC')
    hold on
    switch curr_t
        case 30
            [I_pdf_MC,I_xi_MC] = ksdensity(I_values_MC, ...
                        'support',[0 1],'NumPoints',40);
            plot(I_xi_MC,I_pdf_MC,'o','LineWidth',2,'MarkerSize',10, ...
                        'DisplayName','pdf of I, MC')

        case 100
            set(gca,'ColorOrderIndex',3)
    end

    [R_pdf_MC,R_xi_MC] = ksdensity(R_values_MC, ...
                        'support',[0 1],'NumPoints',40);

    plot(R_xi_MC,R_pdf_MC,'o','LineWidth',2,'MarkerSize',10, ...
                        'DisplayName','pdf of R, MC')
    legend show
    switch curr_t
        case 30
            set(legend,'Location','NorthEast')
        case 100
           set(legend,'Location','North')
    end
    grid on
    ylim([0 12])
    set(gca,'FontSize',20)
    set(gca,'ColorOrderIndex',1)
end
```

Two comments are in order:

- the function `[f,xi] = ksdensity(x)` returns a probability density estimate, `f`, for the sample data in the vector `x`. The estimate is based on a normal kernel function, and is evaluated at equally-spaced points, `xi`, that cover the range of the data in `x`. In our case, `ksdensity` estimates the PDF at 40 equally spaced points that covers the range of data, prescribing the support to be $[0, 1]$.
- We have used the following function, which in a MC experiment (with `N_MC` samples) extracts the j-th time value for each compartment S, I, R and stores them in the array `varargout`.

```
function varargout = get_values_MC_at_t(model_evals_Sr,j,N_MC,N_states)

for pt = 1:N_MC
    vals_Sr = fold_sol_to_matrix(model_evals_Sr(:,pt),N_states);
    for st = 1:N_states
        varargout{st}(pt) = vals_Sr(st,j);
    end
end
```

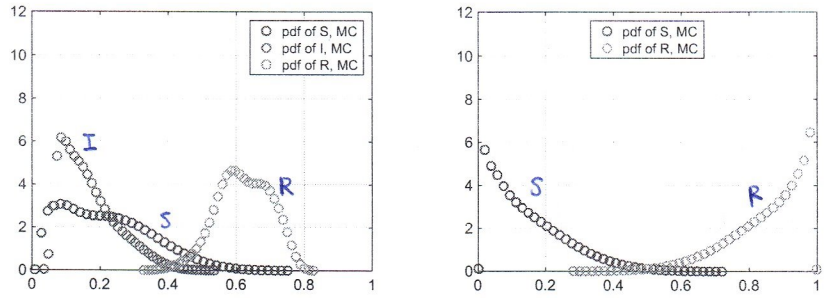


Figure 28: Left: PDFs of S, I and R at $t = 30$. Right: PDFs of S and R at $t = 100$.

4. Let us now compute the PDFs of the following quantities of interest: peak time for I and peak value for I. Here is the code:

```
Tspan = 0:0.5:150;

% derive peaks
peak_I_values = zeros(1,MCsize);
peak_I_times = zeros(1,MCsize);

for pt = 1:MCsize
    vals_Sr = fold_sol_to_matrix(SIR_evals_MC(:,pt),N_states);
    [vI,jI] = max(vals_Sr(2,:));
    peak_I_values(pt) = vI;
    peak_I_times(pt) = Tspan(jI);
end

% peak time pdf
figure
[peak_I_times_pdf,peak_I_times_xi] = ksdensity(peak_I_times);
plot(peak_I_times_xi,peak_I_times_pdf,'-','LineWidth',4, ...
                    'DisplayName','peak time for I, pdf');
grid on
xlim([0 150])
set(gca,'FontSize',30)

% peak value pdf
figure
[peak_I_values_pdf,peak_I_values_xi,bw] = ksdensity(peak_I_values);
plot(peak_I_values_xi,peak_I_values_pdf,'-','LineWidth',4, ...
                    'DisplayName','peak value for I, pdf')
xlim([0 1])
grid on
set(gca,'FontSize',30)
```

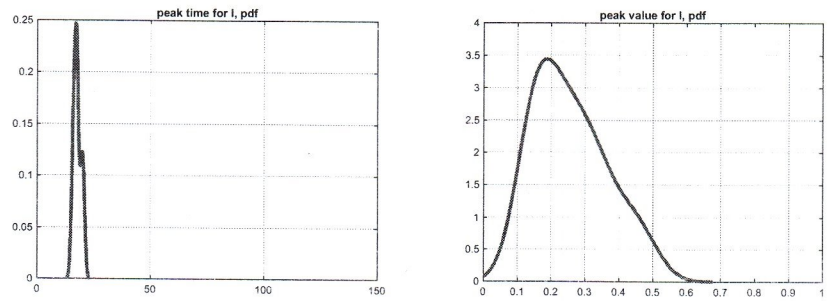


Figure 29: PDFs of the peak time (left) and the peak value (right).

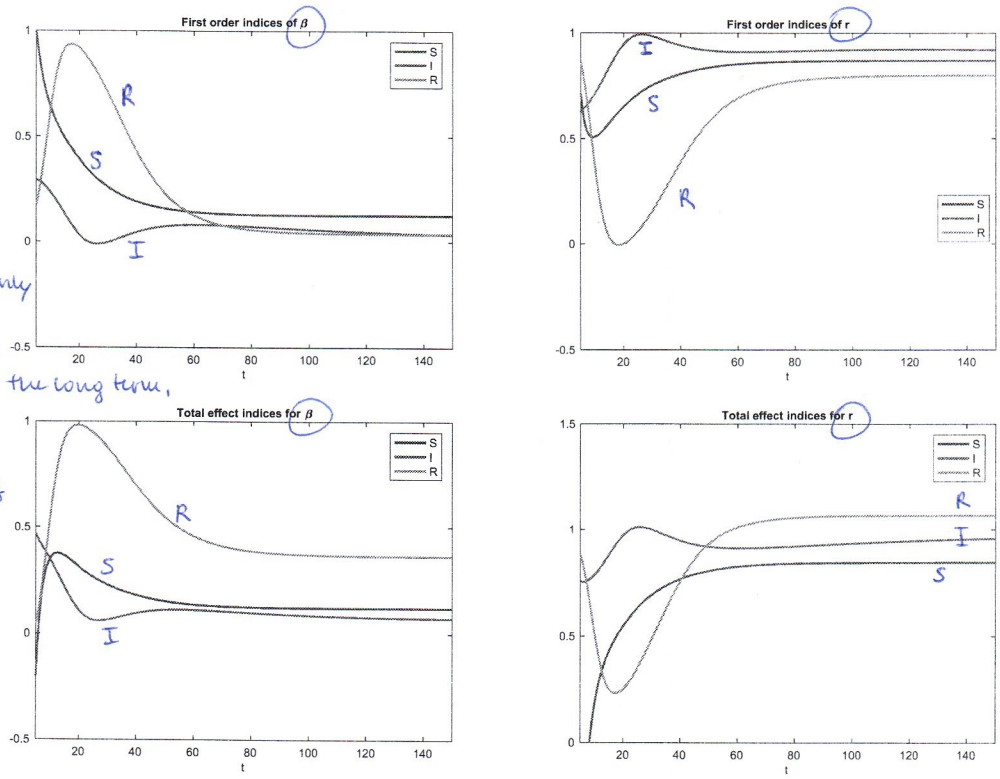
5. We can now perform variance-based global sensitivity analysis using the **SAFE** toolbox. The only difficulty is represented by the fact that we need to evaluate sensitivity indices of two inputs (β and r) over time, on three different variables (S, I and R). Also the model evaluation is now different with respect to previous exercises.

```
%% Step 1 Define inputs and distributions
DistrFun  = 'unif'  ;
xmin = [  beta_min r_min ] ;
xmax = [  beta_max r_max ] ;

x_labels = {'beta','r'} ;
warmup = 1 ;  % time from which indices are computed
GSA_met = 'VBSA' ;

%% Step 2 Sampling
N = 10000 ; % sample size
SampStrategy = 'lhs' ; % Sampling strategy

M = length(xmin) ; % Number of inputs

DistrPar = cell(M,1) ;
for i=1:M
   DistrPar{i} = [ xmin(i) xmax(i) ]
end

X = AAT_sampling(SampStrategy,M,DistrFun,DistrPar,2*N); % Sampling
[ XA, XB, XC ] = vbsa_resampling(X) ;

%% Step 3 Model evaluation
for j = 1: N
    sol = ode45_wrap(N_pop, XA(j,:), Tspan, y0, 'all_times');
    folded_sol = fold_sol_to_matrix(sol,N_states) ;
    YA_S(j,:) = folded_sol(1,:);
    YA_I(j,:) = folded_sol(2,:);
    YA_R(j,:) = folded_sol(3,:);
end

for j = 1: N
    sol = ode45_wrap(N_pop, XB(j,:), Tspan, y0, 'all_times');
    folded_sol = fold_sol_to_matrix(sol,N_states) ;
    YB_S(j,:) = folded_sol(1,:);
    YB_I(j,:) = folded_sol(2,:);
    YB_R(j,:) = folded_sol(3,:);
end

for j = 1: 2*N
    sol = ode45_wrap(N_pop, XC(j,:), Tspan, y0, 'all_times');
    folded_sol = fold_sol_to_matrix(sol,N_states) ;
    YC_S(j,:) = folded_sol(1,:);
    YC_I(j,:) = folded_sol(2,:);
    YC_R(j,:) = folded_sol(3,:);
end

%% Step 4 Compute time-varying Sensitivity Indices
T = size(Tspan,2) ;

Si_S  = nan(T,M) ;   STi_S = nan(T,M) ;
Si_I  = nan(T,M) ;     STi_I = nan(T,M) ;
Si_R  = nan(T,M) ;   STi_R = nan(T,M) ;
```

```
for t=warmup:T
  [ Si_S(t,:), STi_S(t,:) ] = vbsa_indices(YA_S(:,t),YB_S(:,t),YC_S(:,t));
  [ Si_I(t,:), STi_I(t,:) ] = vbsa_indices(YA_I(:,t),YB_I(:,t),YC_I(:,t));
  [ Si_R(t,:), STi_R(t,:) ] = vbsa_indices(YA_R(:,t),YB_R(:,t),YC_R(:,t));
end

%% Step 5 Plot
figure(21)
plot(Tspan,Si_S(:,1),Tspan,Si_I(:,1),Tspan,Si_R(:,1),'-','LineWidth',2)
legend('S','I','R')
figure(22)
plot(Tspan,Si_S(:,2),Tspan,Si_I(:,2),Tspan,Si_R(:,2),'-','LineWidth',2)
legend('S','I','R')
figure(23)
plot(Tspan,STi_S(:,1),Tspan,STi_I(:,1),Tspan,STi_R(:,1),'-','LineWidth',2)
legend('S','I','R')
figure(24)
plot(Tspan,STi_S(:,2),Tspan,STi_I(:,2),Tspan,STi_R(:,2),'-','LineWidth',2)
legend('S','I','R')
```

Obtained first-order indices and total effect indices of the inputs β and r on S, I and R are reported in Figure 30.

which are the most significant parameters?



if we look at β only from the POV of first order effect then it seems that β is not going to interact in the long term, however, if we see what happens in the total effect we see that β is going to interact with R

Figure 30: From top, left to bottom, right: First order indices of β on S, I, R as a function of time; first order indices of r on S, I, R as a function of time; total effect indices of β on S, I, R as a function of time; total effect indices of r on S, I, R as a function of time.

6. The variability of the trajectories is hugely decreased if the range of r is reduced to $[0.06, 0.1]$. To evaluate the SIR dynamics by 100 Monte Carlo samples, and highlight the sample average trajectories, we can exploit the same code used at point 2, by setting `r_max = 0.1`. The results are reported in Figure 31.

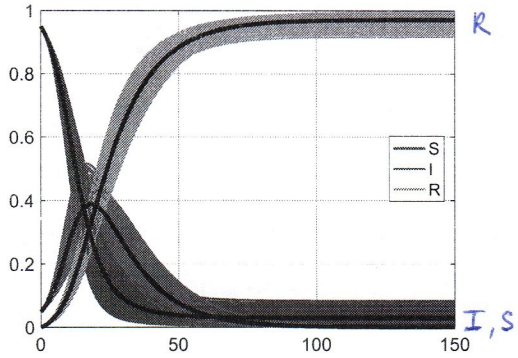


Figure 31: 100 realizations of the SIR dynamics and MC means of S, I and R obtained sampling uniformly the parameter space, in the case of a reduced range on the parameter r.

References

[1] Chiara Piazzola, Lorenzo Tamellini, Raul Tempone. A note on tools for prediction under uncertainty and identifiability of SIR-like dynamical systems for epidemiology. `arXiv:2008.01400`, 2020.