SIP_SENSAI users guide

Simon Tavener

May 25, 2017

1 Problem definition

The stochastic inverse problem is defined by

1. The input space $\lambda \in \Omega$.
2. The forward map $g(\lambda)$ (which may be the solution of a differential equation).
3. the output functionals $f(\lambda) = q(g(\lambda)) \in \mathcal{D}$.
4. The observed distributions $\rho_{\mathcal{D}}(f)$.

The forward map $f(\lambda)$ may be either deterministic or stochastic. A stochastic map requires multiple realizations of the forward map for a given $\lambda \in \Omega$. A deterministic map requires only a single realization for a given location in $\lambda \in \Omega$.

2 The algorithm

First

1. Partition the input space Ω in to cells $P_i,\ i = 1, \ldots, nvol$ with cell volumes $v_i,\ i = 1, \ldots, nvol$.
2. Partition the output space $\mathcal{D}$ in to intervals or bins $Q_j,\ j = 1, \ldots, nbin$.

Let $L_{ij},\ i = 1, \ldots, nvol, j = 1, \ldots, nbin$ be an **input to output map** constructed by repeated application of the forward map, where L_{ij} is the number of forward simulations from input cell P_i that map to output bin Q_j. Let

$$s_i = \sum_{j=1}^{nbin} L_{ij},\ i = 1, \ldots, nvol.$$

Then

$$M_{ij} = \frac{L_{ij}}{s_i}$$

is the *fraction* of forward simulations from input cell P_i that map to output bin Q_j. (In the deterministic case $M_{ij} = \delta_{ik}$ where $1 \leq k \leq nbin$, i.e., M_{ij} is zero for $(nbin - 1)$ values of j and one for a single value of $j(= k)$).

In the MATLAB code, `SIP.m`

- nvol is `nvol`
- nbin is `(fnbin-1)` (for reasons of how histc works)
- L_{ij} is stored in `Lmap(i,j)`,
- M_{ij} is stored in `Mmap(i,j)`,
- s_i is stored in `ns(i)`,

2.1 Interpretation of the input to output map

Rows L_{ij} [or M_{ij}] provide approximations to the conditional probability $p(Q_j|P_i)$. This is zero or one for deterministic maps.

Columns of L_{ij} [or M_{ij}] describe *generalized contours* since all the values of the QoI lie within a given range. (This identification is strict in the deterministic case, but a little woolly in the stochastic case.)

We construct an *ansatz* to distribute the probability within a generalized contour. In other words, we construct an ansatz to compute the conditional probability $p(P_i|Q_j)$.

2.2 Ansatz

We distribute the probability *within* the generalized contour corresponding to output Q_j, i.e., we approximate the conditional probability $p(P_i|Q_j)$, as

$$p(P_i|Q_j) \approx \frac{M_{ij}}{\sum_{k=1}^{nvol} M_{kj}} \tag{1}$$

i.e., the fraction of simulations from input volume i that map to output bin j divided by the fraction of simulations from *all* input volumes k that map to output bin j. (Note that M_{ij} will be zero if cell P_i does not map to output bin Q_j.)

2.3 Algorithm

If $f(\lambda)$ is a stochastic map, then the output $F_{il} = f(P_i), l = 1, \ldots, s(i)$ is a random variable.

Generate approximating sets $\{P_i\}_{i=1}^{nvol}$ and $\{Q_j\}_{j=1}^{nbin}$ that partition Ω and $\mathcal{D}$ respectively

Construct a simple function approximation to $\rho_{\mathcal{D}}$, namely

$$\rho_{\mathcal{D},nbin} = \sum_{j=1}^{nbin} p(Q_j)\mathbf{1}_{Q_j}$$

for $i = 1, \ldots, nvol$ **do**
    for $l = 1, \ldots, s_i$ **do**
        $F_{il} = f(P_i)$
        if $F_{il} \in Q_j$ **then**
            $L_{ij} \leftarrow L_{ij} + 1$
        end if
    end for
end for
Compute approximate likelihood function
for $j = 1, \ldots, nbin$ **do**
    for $i = 1, \ldots, nvol$ **do**
        $M_{ij} = \dfrac{L_{ij}}{s_i}$
    end for
    $N_j = \sum_{k=1}^{nvol} M_{kj}$
    for $i = 1, \ldots, nvol$ **do**
        $p(P_i|Q_j) = \dfrac{M_{ij}}{N_j}$
    end for
end for
for $i = 1, \ldots, nvol$ **do**
    $p(P_i) = \sum_{j=1}^{nbin} p(Q_j) * p(P_i|Q_j)$
end for

3 Matlab codes

Two Matlab codes were used to perform the computations in Daly et. al (2017). The first generates and stores repeated function evaluations. The second code restores the function evaluations and performs an inverse sensitivity analysis. The codes are carefully documented at the top of important routines. Convergence of the deterministic version of the SIP algorithm is studied in [1, 2, 3].

1. SENSAI_MuPad solves the system of ordinary differential equations, computes sensitivity and elasticities and investigates the properties of the sensitivity matrix (the "square root" of the Fisher Information Matrix) either at all times or using selected (time) points [imap=0]. The code also performs repeated function evaluations which it stores for later evaluation by SIP_SENSAI [imap=15].

2. SIP_SENSAI reads the function evaluation information created by SENSAI_MuPad and performs an inverse sensitivity analysis.

4 SENSAI_MuPad

4.1 Creating the Matlab routines

SENSAI_MuPad uses the symbolic capabilities of MuPad to create Matlab code for the RHS of the system of differential equations `gvec.m`, and to differentiate the RHS with respect to the variables and parameters to create Matlab files `dgvec_dxvec.m, dgvec_dparam.m`. It also creates Matlab code `qoi.m` to evaluate the specified quantity of interest. Sensitivities of the quantity of interest are calculated using the chain rule.

1. Run MuPad files, e.g., `Logistic.mu` and `HodgkinHuxley.mu`

2. >> sensai

3. Add directory containing MuPad file

4. Use "Create Matlab files using MuPad" button

Note: Perhaps I should create a script `run_MuPad` to avoid using the GUI.

4.2 Investigating the Fisher Information Matrix

To perform sensitivity analysis and investigate the properties of the Fisher Information matrix, use the commands >> run_gtype('Examples/ODE_examples/Logistic',0)

>> run_gtype('Examples/ODE_examples/HodgkinHuxley-4-SIP',0)

Control is provided through the two input files

- user_inputs.m
- user_FIM.m

4.3 Construct and save function evaluations for SIP

To span the input parameter space and compute and store values of the QoI, use the commands

>> run_gtype('Examples/ODE_examples/Logistic',15)
>> run_gtype('Examples/ODE_examples/HodgkinHuxley-4-SIP',15)

Control is provided through the two input files

- user_inputs.m
- user_SIP.m

4.4 Brute force forward sensitivity computations

To demonstrate that normally distributed noise in the voltage trace does not result in normally distributed values for the summary statistics, we perform a brute force sensitivity analysis as follows.

>> [q,fmin,fmax,fedges,pout]= ...
brute_force(x0,p0,tfinal,ntsteps,xdim,kdim,stype,rtype,nr,idim,nss,nedges);

5 SIP_SENSAI

>> [pinp]=SIP_SENSAI(input_source,output_source,ExampleName,JobName,iprint,iplot)

Inputfiles

- SIP_SENSAI_input

6 Details of the Matlab routines

6.1 SENSAI_MuPaD

The command `run_gtype` is more efficient that the "Execute Matlab file created by MuPaD" button on the GUI

1. run_gtype
    - user_inputs
    - user_QoI
    - user_bifndata
    - user_FIMdata
    - user_plotdata
2. gtype
3. gtype_SIP
    - user_SIP
    - calculate_grid
    - ...Loops over all cells in the mesh and evaluates QoIs...
    - SIP_datafiles creates output_directory/SIP_matfile.mat

6.2 SIP

1. SIP_SENSAI
    - input_output_map
    - SIP_SENSAI_input
    - input_checks
    - Make_Output_Directory

   Uses histc to perform binning.

References

[1] J. Breidt, T. Butler, and D. Estep. A measure-theoretic computational method for inverse sensitivity problems I: Method and analysis. *SIAM Journal on Numerical Analysis*, 49(5):1836–1859, 2011.

[2] T. Butler, D. Estep, and J. Sandelin. A computational measure theoretic approach to inverse sensitivity problems II: a posteriori error analysis. *SIAM Journal on Numerical Analysis*, 50(1):22–45, 2012.

[3] T. Butler, D. Estep, S.J. Tavener, C. Dawson, and J. Westerink. A measure-theoretic computational method for inverse sensitivity problems III: Multiple quantities of interest. *SIAM Journal on Uncertainty Quantification*, pages 1–27, 2014.