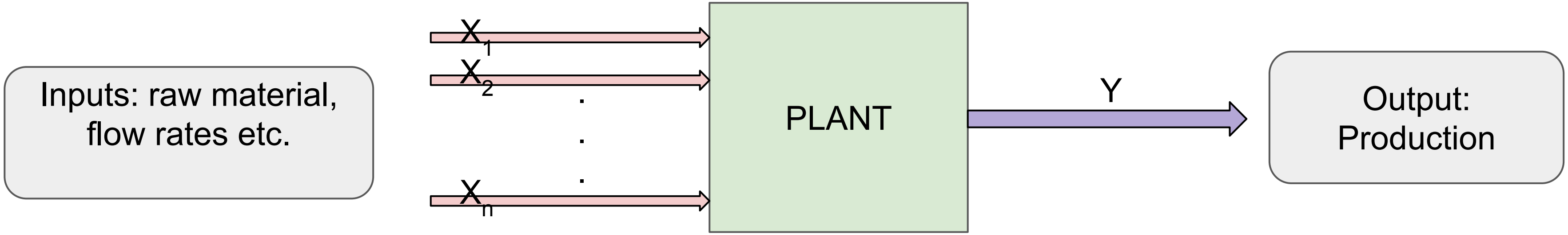
Shapley Effects for Global Sensitivity Analysis

Devang Sehgal and Anurag Vaidya
Department of Health Science and Technology, MIT
5/19/2023

Code: https://github.com/ajv012/shapley_julia

Suppose you have a Plant …



How does the output of a model generally change with a change in the input?

Global Sensitivity Analysis!

Sensitive inputs

Sparsify

Robustness

C. Rackauckas, SciML/scimlbook https://book.sciml.ai/

GlobalSensitivity.jl

- SciML implementation for Global Sensitivity Analysis.

```
res = gsa(f, method, param_range; samples)
```

- Several methods implemented - derivative based, morris, regression etc.
- State of the art method in library: Sobol.
  - First order effects

$$V_i \equiv \mathrm{Var}[\mathrm{E}[Y|X_i]] = \mathrm{Var}[Y] - \mathrm{E}[\mathrm{Var}[Y|X_i]].$$

  - Total effects

$$T_i \equiv \mathrm{Var}[Y] - \mathrm{Var}[\mathrm{E}[Y|\mathbf{X}_{-i}]] = \mathrm{E}[\mathrm{Var}[Y|\mathbf{X}_{-i}]],$$

- Limitations:
  - Methods cannot take into account dependent inputs
  - Sobol may not characterise overall variance correctly

V. Dixit and C. Rackauckas, GlobalSensitivity.jl, 2022, https://github.com/SciML/GlobalSensitivity.jl

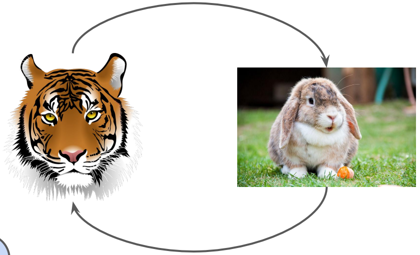
Shapley Effects

- Based on Shapley values from game theory
- Attributes total variance to individual inputs or ‘players’ - interpretable.
- Can handle dependencies between features/ inputs.
- Computationally expensive but tractable with Monte Carlo methods.
- Aim of our project to implement this for GlobalSensitivity.jl .

E. Song, B. Nelson, and J. Staum, SIAM, 2016

Details of our implementation

f can be parametric function (fire spread) or ODE solution (prey-predator system)



What are the inputs?

- Function *f* to be tested
- Marginal distribution of each feature
- How do the marginals relate (Copula)

Generate the sample, *X* → Get $y = f(X)$ → Compute Shapley effects (sum to 1)

How to generate the sample X

Goal: Given a payout (function output), what is the contribution of each player (feature)?

Solution: To find contribution of player P, find the payout with and without P in the coalition

Coalition with P

P	A	B	C

Sample from joint distribution of players P, A, B, C

Coalition without P

P

Sample from distribution of P

A	B	C

Sample from joint distribution of players A, B, C

Repeat over all permutations of players and all coalitions to get X

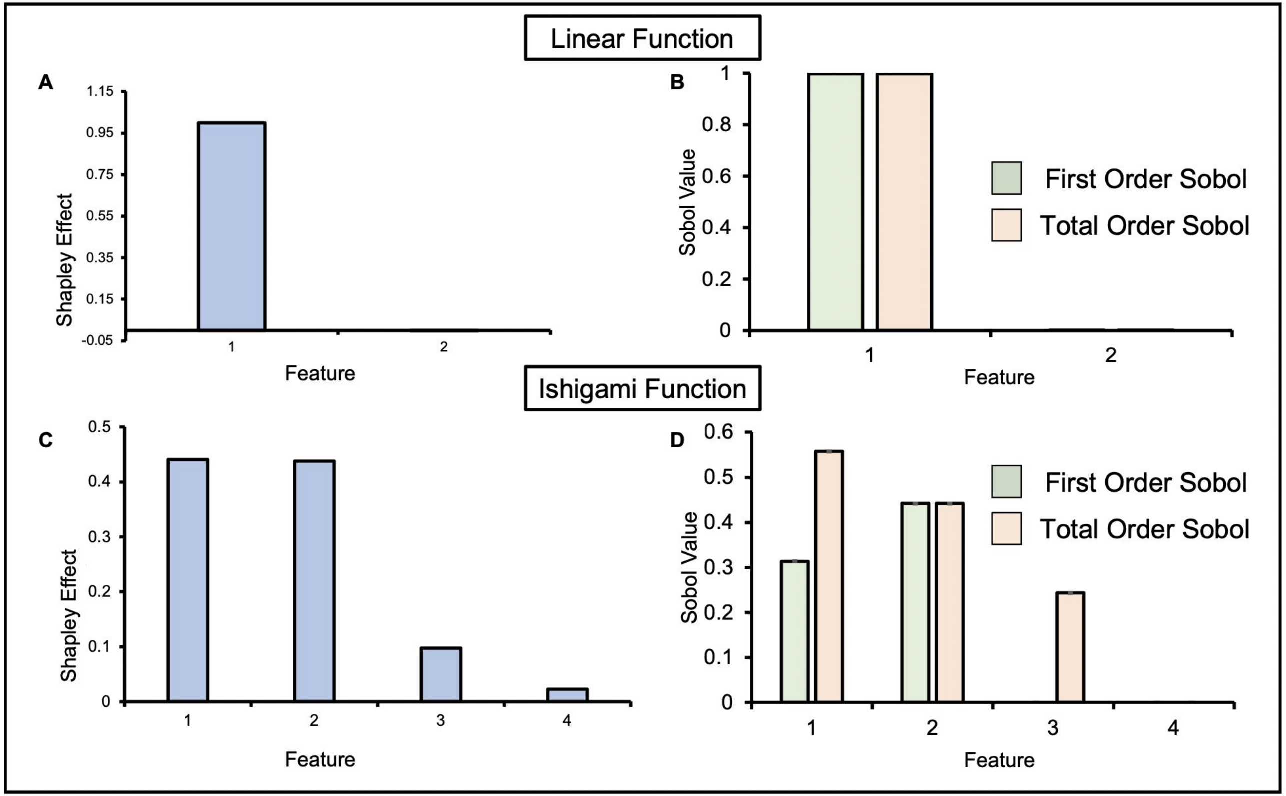
Key results of our implementation

- Demonstrated correctness of our Julia shapley-effects implementation with use of Ishigami and Linear functions, comparing with Sobol and theory.
- Demonstrated utility of shapley-effects over Sobol in interpretability and when inputs are dependent by application on Jackson factory model.
- Demonstrated the functioning of random permutation implementation which makes the problem computationally tractable in higher dimensions.
- Demonstrated versatility of implementation in use with Lotka-Volterra Differential Equations.
- Performance engineering - made code type-stable, reduced allocations and parallelized.
  - Fastest serial implementation is 9x faster than equivalent python-numpy implementation.
  - Parallel implementation 20x faster than equivalent python-numpy on 4 threads.
- Showed impact of hyperparameters on correctness and performance of our code.

Salient features of implementation

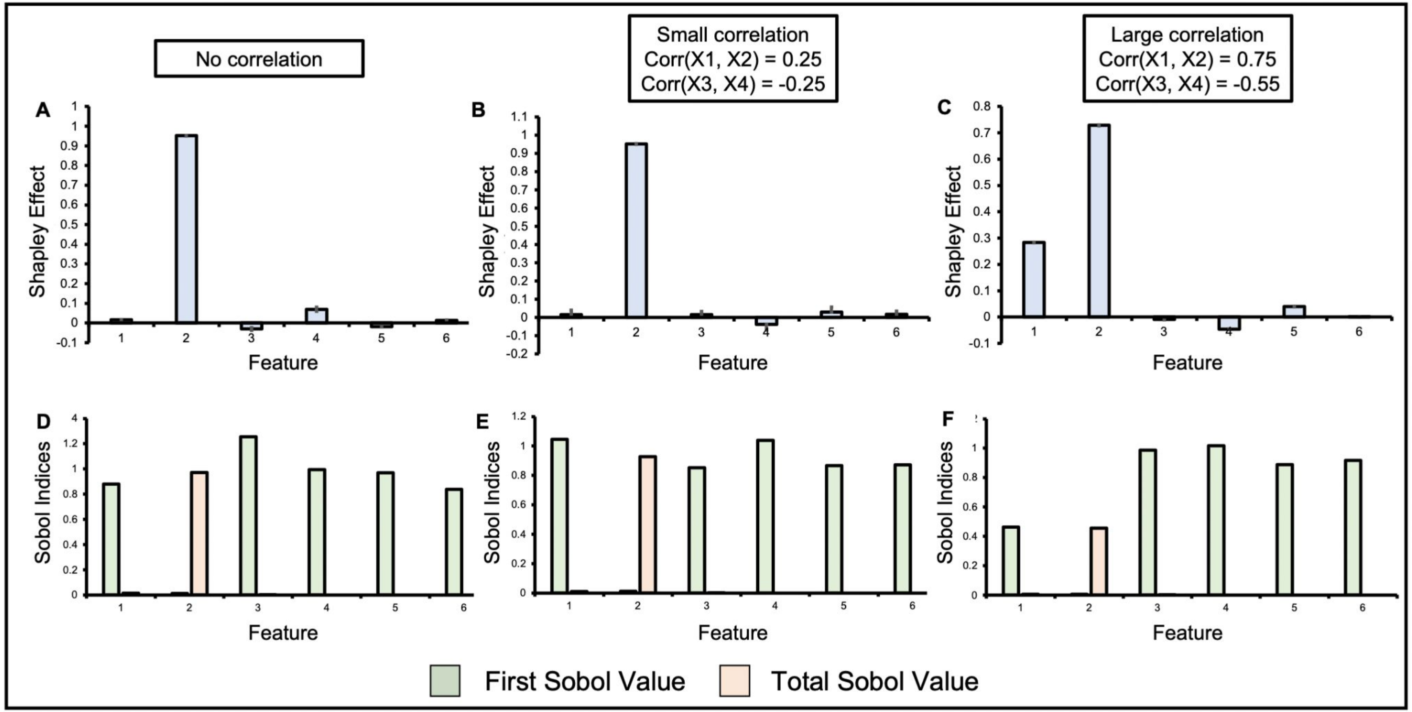
- Decouple the sample generation from Shapley effect calculations
  - Benefit: high performance engineering of each step can be done separately
- All permutations of players infeasible for large number of players (think 10!)
  - Implement two models:
    - random permutation = sample from all permutations
    - Exact permutation = consider all permutations
- Report the median Shapley effect and 95% confidence interval

Correctness experiments



For simple functions, Shapley effects match with Sobol indices. Redundant features are given ~0 attribution by both methods

Applying our algorithm to Jackson model of a manufacturing plant



With greater correlation, we see discrepancies in the Sobol indices but not Shapley effects

Effect of different hyperparams on the memory and time complexity of Shapley algorithm

N_V	N_O	N_I	n_{boot}	Feature 1	Feature 2	Feature 3	Allocation (GB)	Time (ms)
10	100	3	60000	0.316 (0.302, 0.329)	0.497 (0.479, 0.515)	0.125 (0.116, 0.135)	1.77	479.86
100	100	3	60000	0.484 (0.483, 0.485)	0.414 (0.413, 0.415)	0.033 (0.033, 0.034)	1.86	493.34
1000	100	3	60000	0.396 (0.395, 0.396)	0.383 (0.382, 0.383)	0.153 (0.153, 0.153)	2.67	631.20
10000	100	3	60000	0.380 (0.379, 0.381)	0.414 (0.413, 0.414)	0.140 (0.139, 0.140)	10.71	1772.0
1000	**1**	3	60000	0.707 (0.706, 0.708)	0.501 (0.500, 0.501)	-0.253 (-0.254, -0.253)	1.52	333.06
1000	**10**	3	60000	0.124 (0.123, 0.124)	0.716 (0.714, 0.719)	0.067 (0.065, 0.069)	1.63	351.6
1000	**100**	3	60000	0.319 (0.318, 0.319)	0.418 (0.417, 0.418)	0.167 (0.167, 0.169)	2.67	579.71
1000	**1000**	3	60000	0.431 (0.430, 0.432)	0.387 (0.387, 0.387)	0.104 (0.104, 0.104)	12.97	2917.0
1000	100	**2**	60000	0.459 (0.458, 0.460)	0.402 (0.402, 0.403)	0.075 (0.074, 0.075)	2.66	602.27
1000	100	**5**	60000	0.382 (0.381, 0.382)	0.394 (0.393, 0.394)	0.157 (0.157, 0.157)	2.67	714.80
1000	100	**10**	60000	0.459 (0.458, 0.460)	0.395 (0.394, 0.395)	0.083 (0.082, 0.83)	2.67	908.03
1000	100	**100**	60000	0.396 (0.395, 0.396)	0.448 (0.448, 0.449)	0.096 (0.096, 0.097)	2.69	4684.0
1000	100	3	**100**	0.388 (0.375, 0.401)	0.413 (0.399, 0.426)	0.123 (0.115, 0.132)	0.03	7.19
1000	100	3	**1000**	0.437 (0.432, 0.442)	0.412 (0.408, 0.416)	0.078 (0.075, 0.081)	0.07	16.10
1000	100	3	**10000**	0.436 (0.434, 0.437)	0.379 (0.378, 0.381)	0.107 (0.106, 0.108)	0.48	111.98
1000	100	3	**100000**	0.392 (0.392, 0.393)	0.440 (0.439, 0.440)	0.079 (0.079, 0.080)	4.42	1029.0
1000	**100**	**3**	**60000**	**0.398 (0.397, 0.398)**	**0.386 (0.385, 0.386)**	**0.147 (0.146, 0.147)**	**2.67**	**605.61**

Table 1

The effect of different hyper-parameters N_V, N_O, N_I, n_{boot} on the output of the Shapley effects algorithm on the Ishigami function. The values in bold indicate the hyperparameter that is being changed for that set of experiments. The final row shows the set of hyper-parameters used in making Figure 2C. The Shapley effects for the different features are reported with their 95% Confidence Interval in brackets.

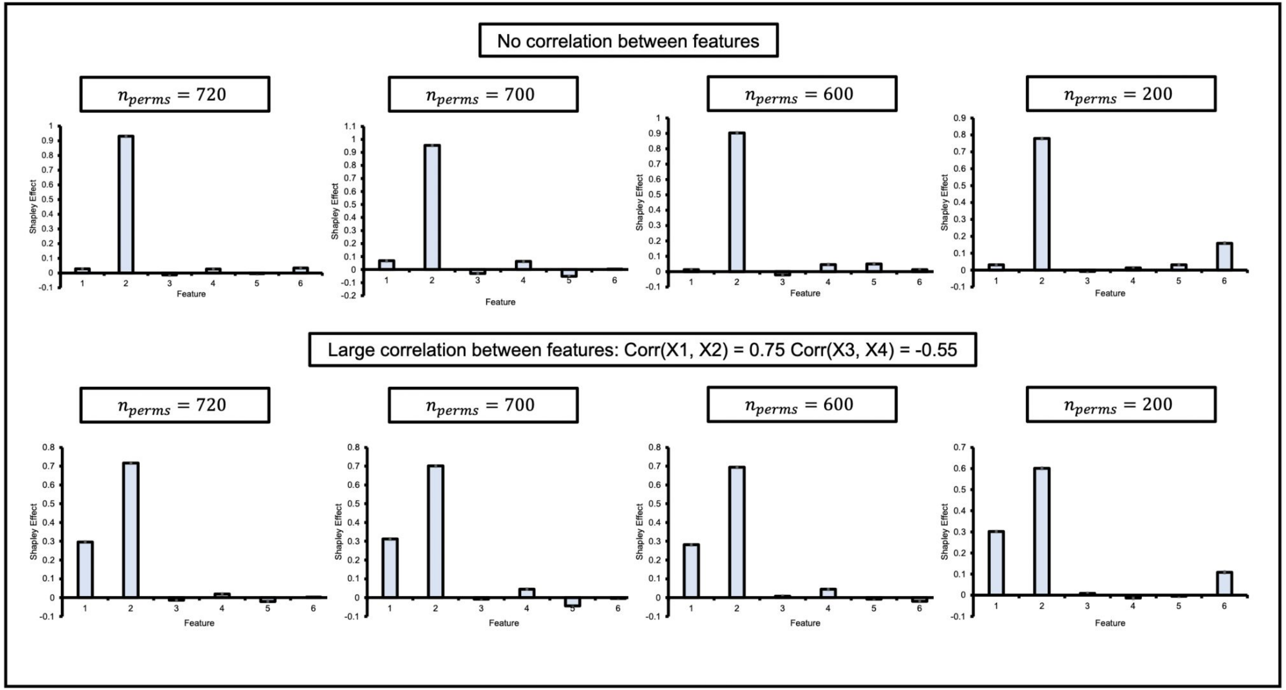
Breaking down the effect of hyperparams on different parts of the algorithm

N_V	N_O	N_I	n_{boot}	dim	**Total runtime (s)**	**Total allocations (GB)**	**Sample time (s)**	**Shapley time(s)**
1000	100	3	1000	**3**	0.014	0.073	0.006	0.008
1000	100	3	1000	**4**	0.075	0.281	0.038	0.038
1000	100	3	1000	**5**	0.531	2.02	0.370	0.161
1000	100	3	1000	**6**	3.590	13.71	2.162	1.428
1000	100	3	1000	**7**	45.083	128.35	22.903	22.180
1000	100	3	100000	**3**	1.054	4.42	0.006	1.048
1000	100	3	100000	**4**	3.387	15.45	0.081	3.305
1000	100	3	100000	**5**	20.629	88.19	0.292	20.337

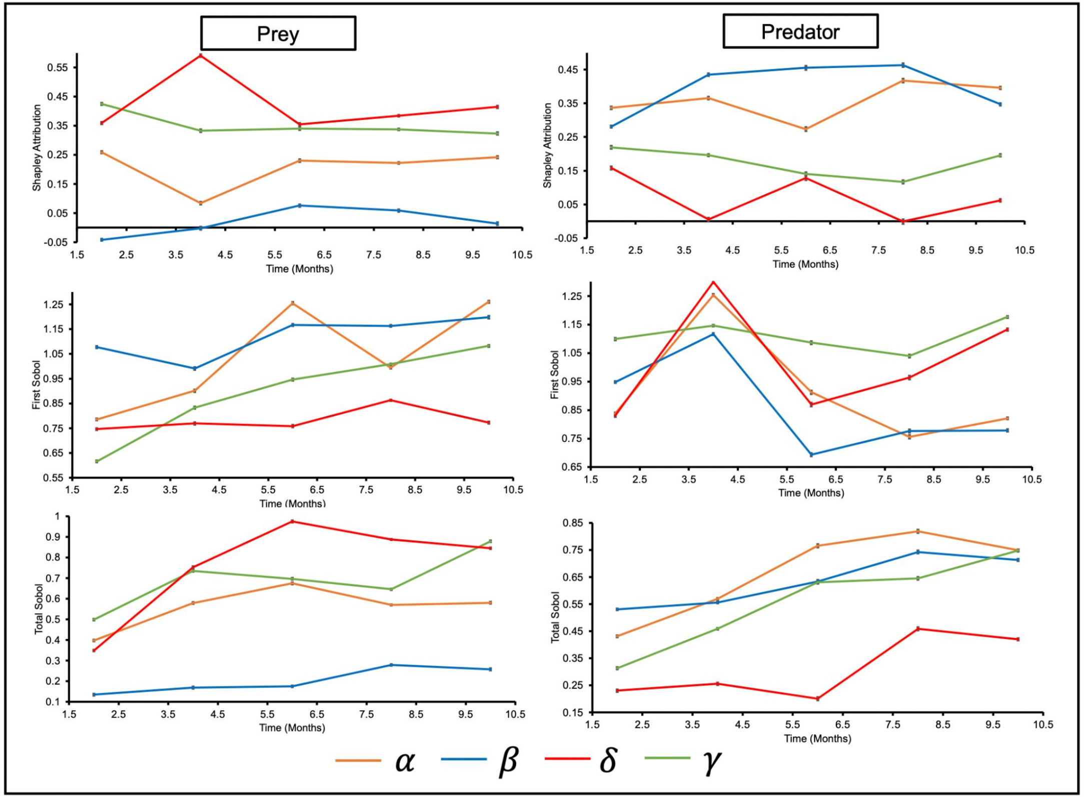
Table 3

Runtime and allocations as a function of number of dimensions and number of bootstraps. The total runtime is broken down into the time taken for generating the sample (sample time) and the time for calculating Shapley indices (shapley time).

We make Shapley effects algorithm tractable for functions with many inputs by randomly sampling permutations of features



Shapley effects can be applied to time-dependent systems, like the Lotka–Volterra equations



Shapley effects have better interpretability because they can be seen as percentages of total variance. Easier to interpret system and see how importance changes at different time points.

Total and First order Sobol cannot be compared against each other because they do not sum up to the total variance.

Significant performance gains made by Julia over python (tested over the Ishigami function)

Implementation	N_V	N_O	N_I	n_{boot}	dim	Runtime (s)	Allocations (GB)
python-shapley [5]	1000	100	3	60000	3	12.009	n/a
julia-baseline	1000	100	3	60000	3	1.917	4.12
julia-optim. serial	1000	100	3	60000	3	1.372	1.99
julia-optim. parallel	1000	100	3	60000	3	0.582	2.67

Table 2

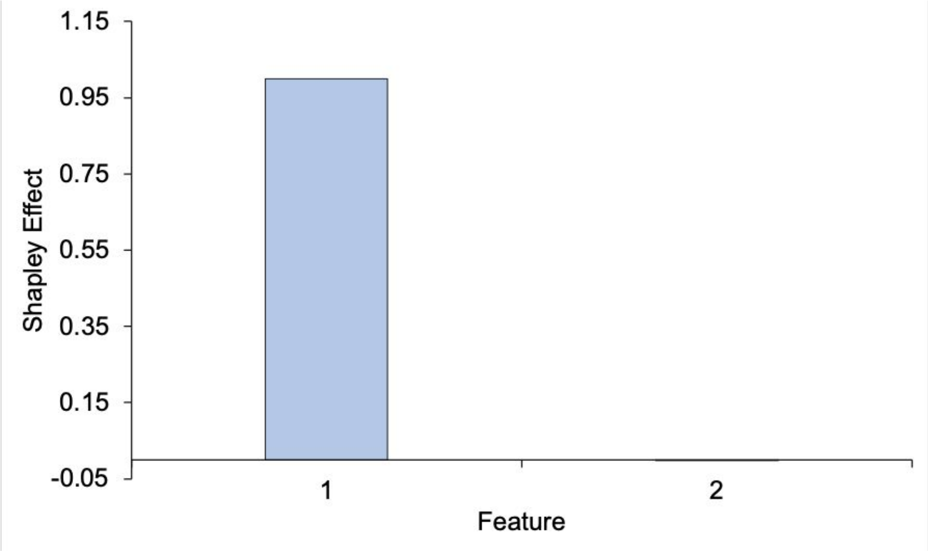
Performance benchmarks, namely runtime and allocations, for 4 implementations of Shapley effects for GSA: 'python-shapley' refers to the implementation of Shapley effects in python [5], 'julia-basline' is our implementation of Shapley-effects in the GSA scaffold, 'julia-optim. serial' refers to our optimised serial implementation while 'julia-optim. parallel' is our optimised parallel implementation, running over 4 threads.

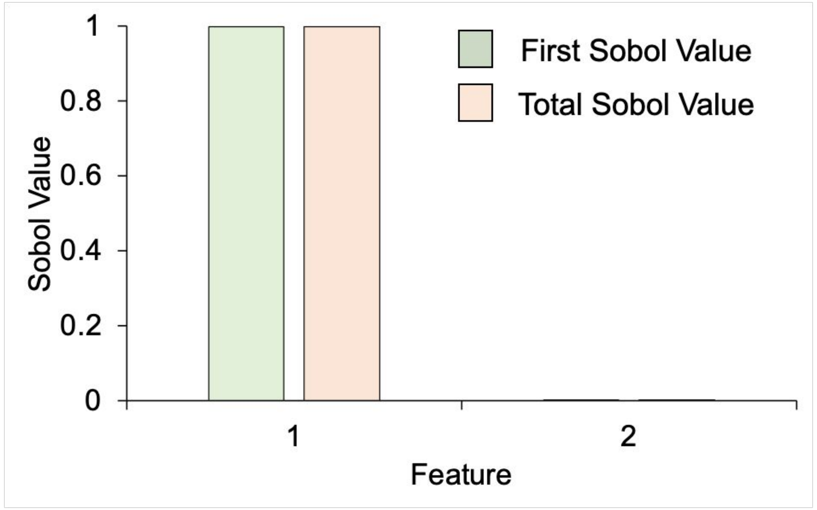
Next steps: Integrate Shapley effects in GlobalSensitivity.jl. Check out PR at: https://github.com/SciML/GlobalSensitivity.jl/pull/105

Archive slides

Linear (Shapley and Sobol)

$$y = 0.7x + 0.1$$



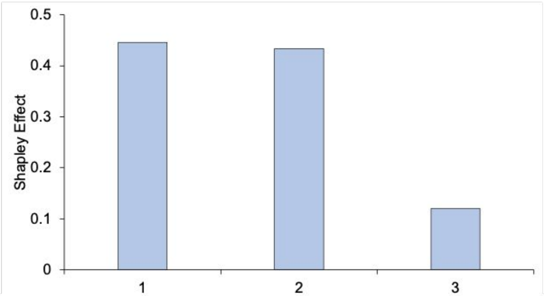


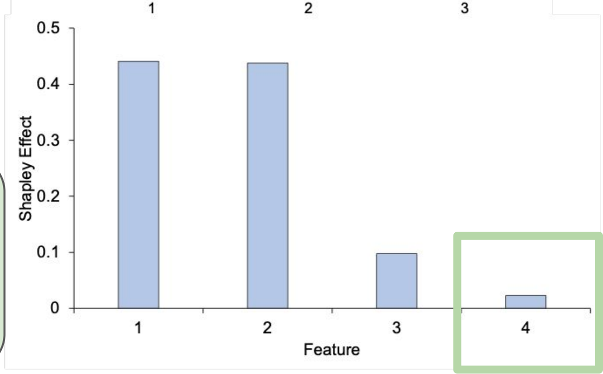
99.9% of payout accounted by feature 1 (0.7)

No interpretation can be made

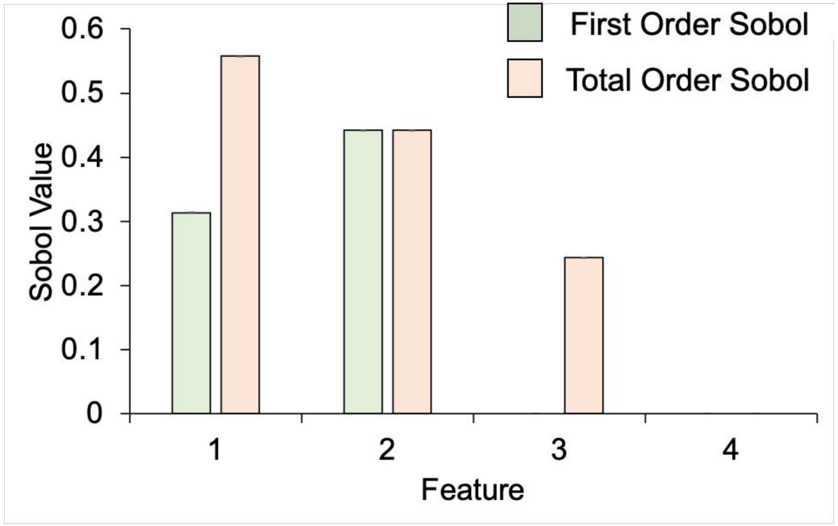
Ishigami (Shapley and Sobol)

$$y = \sin(x_1) + 7\sin^2(x_2) + 0.1x_3^4\sin(x_1)$$





Redundant feature given ~0 attribution



First order and Total order Sobol are inconsistent

Next Steps

- Develop Randperm - implementation to generate and use randomly samples permutations
- Performance Engineering - improve and characterise performance
- Application to other systems - Jackson manufacturing network model.