IBI Group and NRCan

SOFDA 3.0.4

User Manual

Seth Bryant
4-4-2020

Contents

Notes and Acknowledgements 3
1. Introduction 4
1.1. Workflow 4
1.2. Alberta Curves 5
2. Description 8
2.1. Modes 10
2.2. Hierarchy 11
2.3. Flood Risk Simulations 13
2.3.1. Simulation Timeline 16
2.3.2. Flood Damage Module (Fdmg) 17
2.3.3. Urban Re-Development Module (Udev) 19
3. Running SOFDA 20
3.1. Logging 21
3.2. User Data Files 22
3.3. Outputs 22
3.3.1. Hierarchy 23
3.3.2. Types 24
3.3.3. Formats 24
3.4. Tutorial 24
4. Building a SOFDA Model 25
4.1. Global Parameters 28
4.2. Basic Data Files 29
4.2.1. Building Inventory 29
4.2.2. Damage Feature Tables 30
4.2.3. FHZ tables 31
4.3. Houses 32
4.4. Dfuncs 36
4.4.1. Depth-Damage Arrays 37
4.5. Flood Tables (Exposure Variables) 40

4.6. Flood Objects 42
4.7. Timeline 42
4.8. Actions 42
4.9. Selectors 44
4.9.1. Special Functions 44
4.10. Dynamic Parameters (Dynp) 47
4.11. Outputs 48
5. SOFDA Model Modes 49
5.1. Sensitivity Analysis (SA) 49
5.1.1. Focus Parameters 50
5.1.2. Sensitivity Metrics 51
Bibliography 52
Attachment B: Sample Control File 53
Attachment C: Damage Feature Table 54
Attachment D: Alberta Curves 55

Notes and Acknowledgements

Disclaimer

SOFDA is an experimental flood risk assessment research tool with limited testing and validation. Natural Resources Canada and IBI Group assume no liability for any errors or inaccuracies. Any user of SOFDA is responsible for performing their own tests and developing their own confidence in the results of the model. For questions and concerns, please contact the development team via the project GitHub page.

Development Acknowledgements

The original implementation of SOFDA was developed by Seth Bryant with guidance from David Sol (IBI Group) and Dr. Evan Davies (University of Alberta). This work was made possible by the generous support of Natural Sciences and Engineering Research Council of Canada’s Engage and Discovery grants; as well as in-kind contributions from IBI Group through their ‘IBI Think’ initiative. The City of Calgary provided most of the data used to develop early SOFDA models and was instrumental in guiding the work (esp. Sandra Davis).

This implementation of SOFDA was developed for the CanFlood plugin by IBI Group under contract with Natural Resources Canada (NRCan). Copyright is held by NRCan and the software is distributed under the MIT License.

1. Introduction

The Stochastic Object-based Flood damage Dynamic Assessment model framework (SOFDA) was developed to simulate flood risk over time using depth-damage functions and a residential re-development forecast. Framework development was motivated by a desire to quantify the benefits of FHRs and to help incorporate the dynamics of risk into decision-making. This manual provides guidance to a risk analyst seeking to use SOFDA to develop a model that estimates flood risk. Evaluating flood risk is challenging and requires experience and knowledge well beyond what is provided in this manual. The reader should be well informed of the flood risk assessment process before attempting to use SOFDA.

SOFDA is written in python 3.7. Implementation in python allows the model to leverage a vast array of publicly available modules and promotes readability and reusability. For more information on the development and application of SOFDA, refer to Bryant (2019).

1.1. Workflow

A typical workflow for risk assessments executed with SOFDA is presented in Figure 1-1. This manual provides guidance to apply SOFDA once all the required inputs are collected, the hazard analysis complete, and the study objectives well established. Post-processing is not addressed.

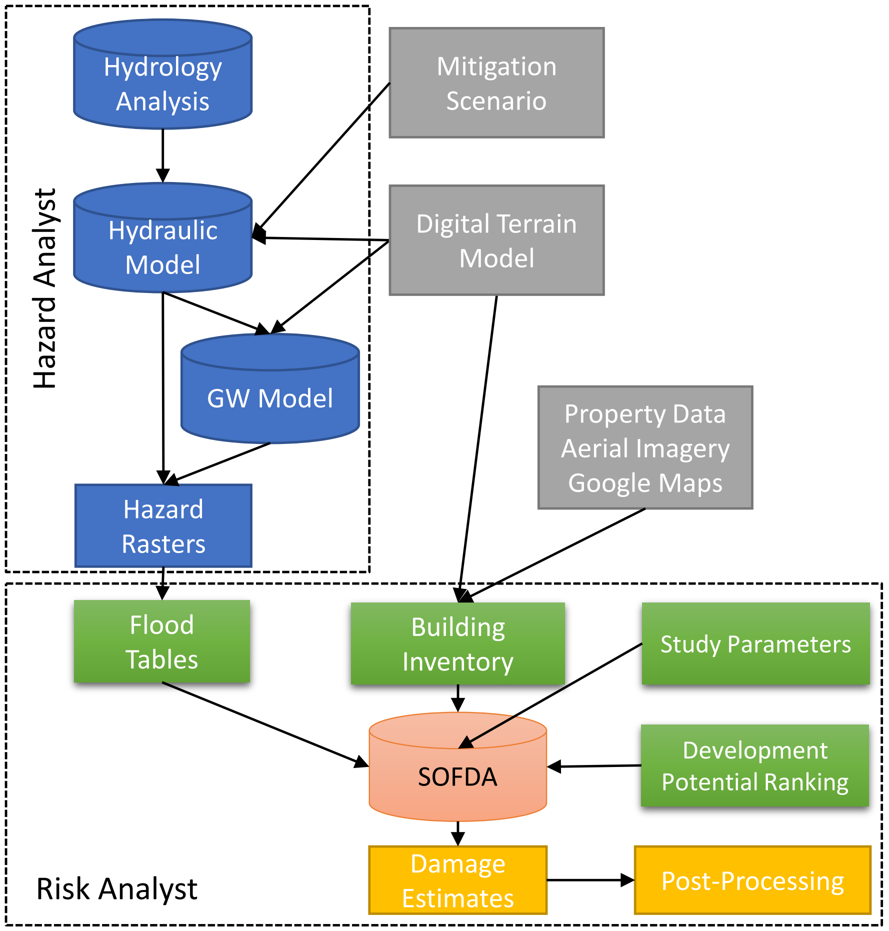


Figure 1-1: Risk assessment workflow for studies implementing SOFDA. Green boxes denote primary inputs to SOFDA.

1.2. Alberta Curves

The Alberta Curves are a set of eleven residential (Figure 1-4) and 20 non-residential loss functions which were developed to predict direct structural (S) and contents (C) building damages from flood depth. These are further divided into main floor (M) and basement (B) damages[1]. Residential categories were developed from expert knowledge of typical Canadian building typology and divide buildings by: 1) size; 2) quality; 3) construction technique; and 4) number of stories. In practice, analysts use government datasets (e.g. property assessment records), aerial imagery, and Google Street View to assign categories to houses. To develop the single-residential curves, 83 in person surveys were conducted of representative flood-unaffected homes and their contents during

[1] Garage damages were also tabulated and reported separately, but combined into both curves (M, B) with the assumption that the garage floor is 2' below the main floor elevation.

2014 in Calgary and Edmonton. For more information on the Alberta Curves, the reader is referred to IBI Group and Golder Associates (2015).

Table 1.1: Alberta Curve building types in study area adapted from IBI Group and Golder Associates(2015).

Class[b]	**Type**	**Building Type**[a]	**Class Description**	**Type Description**
A	A	AA	Home with living space defined as equal to or between 3,999 and 2,400 ft^2.	1 storey
A	D	AD	Home with living space defined as equal to or between 3,999 and 2,400 ft^2.	2 storeys
B	A	BA	Home with living space defined as equal to or between 2,399 and 1,200 ft^2.	1 storey
B	D	BD	Home with living space defined as equal to or between 2,399 and 1,200 ft^2.	2 storeys
C	A	CA	Home with living space defined equal to or less than 1,199 ft^2.	1 storey
C	D	CD	Home with living space defined equal to or less than 1,199 ft^2.	2 storeys
a) RFDA requires class and type variables from each entry in the building inventory. These are combined to select the appropriate damage curve. SOFDA only requires one variable.				
b) See fig XXX for photographs of typical homes.				

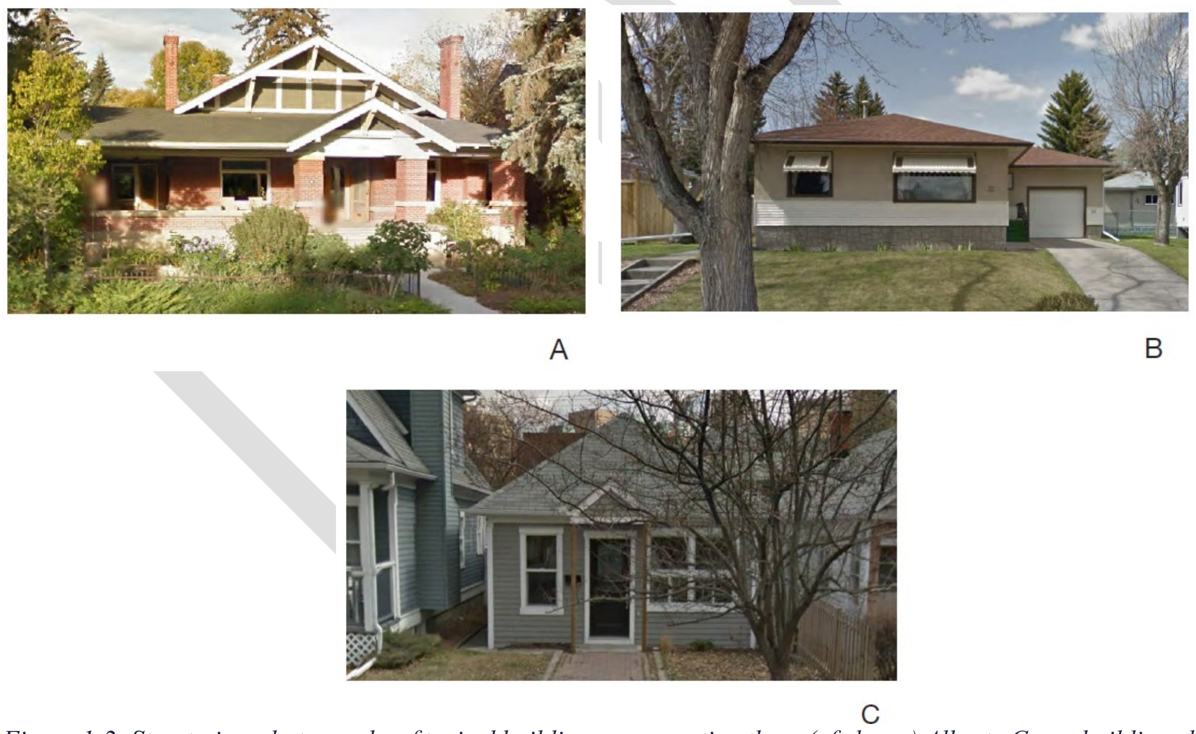
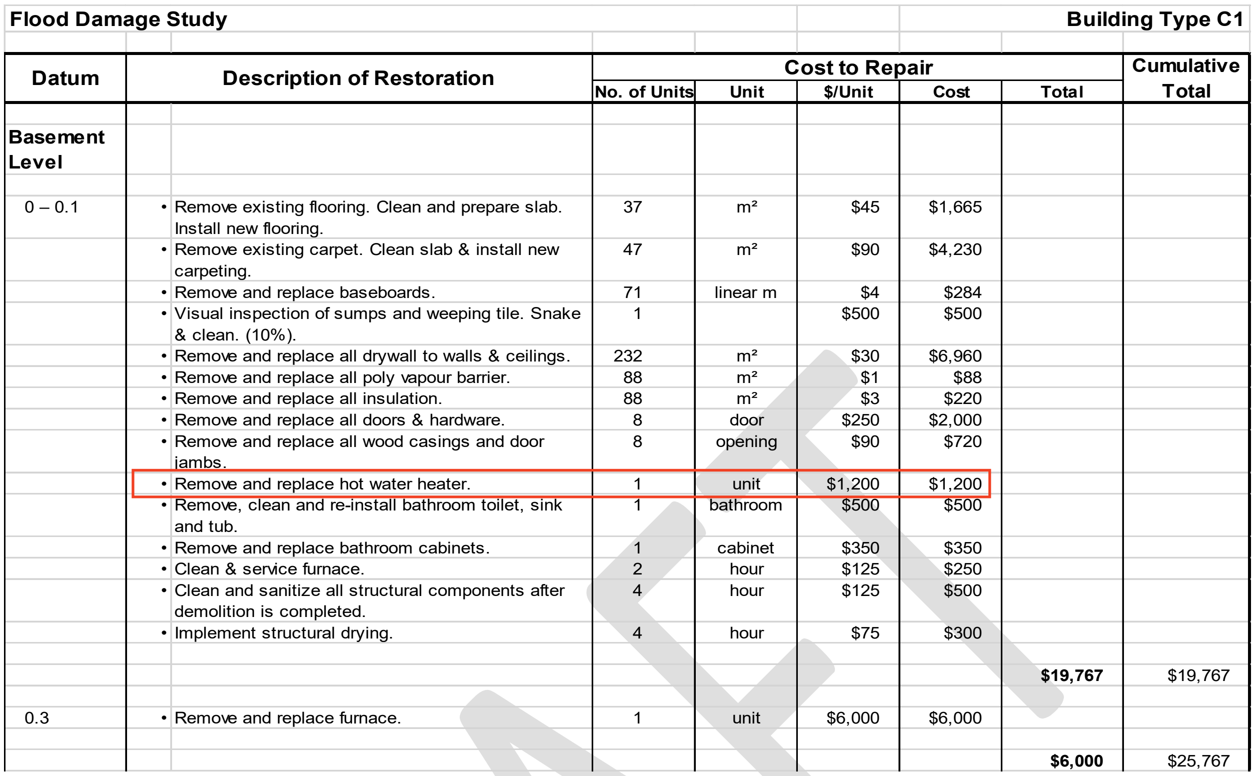


Figure 1-2: Street-view photographs of typical buildings representing three (of eleven) Alberta Curve building classes from IBI Group and Golder Associates (2015).



Flood Damage Study							Building Type C1
Datum	Description of Restoration	Cost to Repair					Cumulative Total
		No. of Units	Unit	$/Unit	Cost	Total	
Basement Level							
0 – 0.1	• Remove existing flooring. Clean and prepare slab. Install new flooring.	37	m²	$45	$1,665		
	• Remove existing carpet. Clean slab & install new carpeting.	47	m²	$90	$4,230		
	• Remove and replace baseboards.	71	linear m	$4	$284		
	• Visual inspection of sumps and weeping tile. Snake & clean. (10%).	1		$500	$500		
	• Remove and replace all drywall to walls & ceilings.	232	m²	$30	$6,960		
	• Remove and replace all poly vapour barrier.	88	m²	$1	$88		
	• Remove and replace all insulation.	88	m²	$3	$220		
	• Remove and replace all doors & hardware.	8	door	$250	$2,000		
	• Remove and replace all wood casings and door jambs.	8	opening	$90	$720		
	• Remove and replace hot water heater.	1	unit	$1,200	$1,200		
	• Remove, clean and re-install bathroom toilet, sink and tub.	1	bathroom	$500	$500		
	• Remove and replace bathroom cabinets.	1	cabinet	$350	$350		
	• Clean & service furnace.	2	hour	$125	$250		
	• Clean and sanitize all structural components after demolition is completed.	4	hour	$125	$500		
	• Implement structural drying.	4	hour	$75	$300		
						$19,767	$19,767
0.3	• Remove and replace furnace.	1	unit	$6,000	$6,000		
						$6,000	$25,767

Figure 1-3: Sample structural damage feature table for a 'C' class house from IBI Group and Golder Associates (2015). Red box denotes a single damage feature.

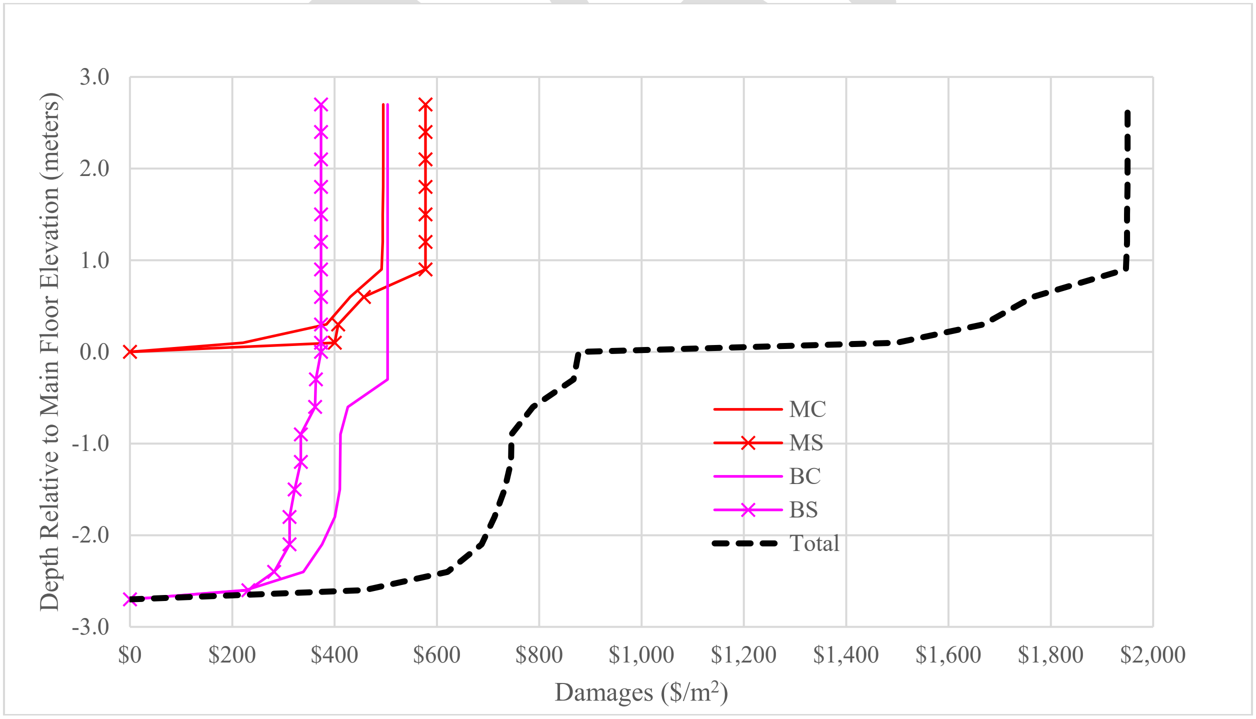


Figure 1-4: RFDA depth-damage curve for a class C house.

The Alberta Curves datafile is provided in Attachment D.

2. Description

SOFDA is designed to provide property level, direct-damage, financial flood risk estimates from a flood water surface level (WSL) table. To accomplish this, the study area is discretized into assets categorized by loss function type (Figure 2-1). While SOFDA can support any depth-damage loss function (in the correct format), the framework was designed for use with the *Alberta Curves* — a set of 11 residential and 20 commercial engineered depth-damage loss functions (section 1.2). Using the Alberta Curves, SOFDA can provide static risk assessments for the study area with the following major assumptions:

- direct financial damage as the significant flood risk indicator;
- the loss functions accurately predict the total damage to each asset from depth at its discrete anchor point;
- the flood tables accurately describe the WSL to frequency relation at each asset;
- the range of floods considered accurately represents the set of all flood hazards possible in the study area;

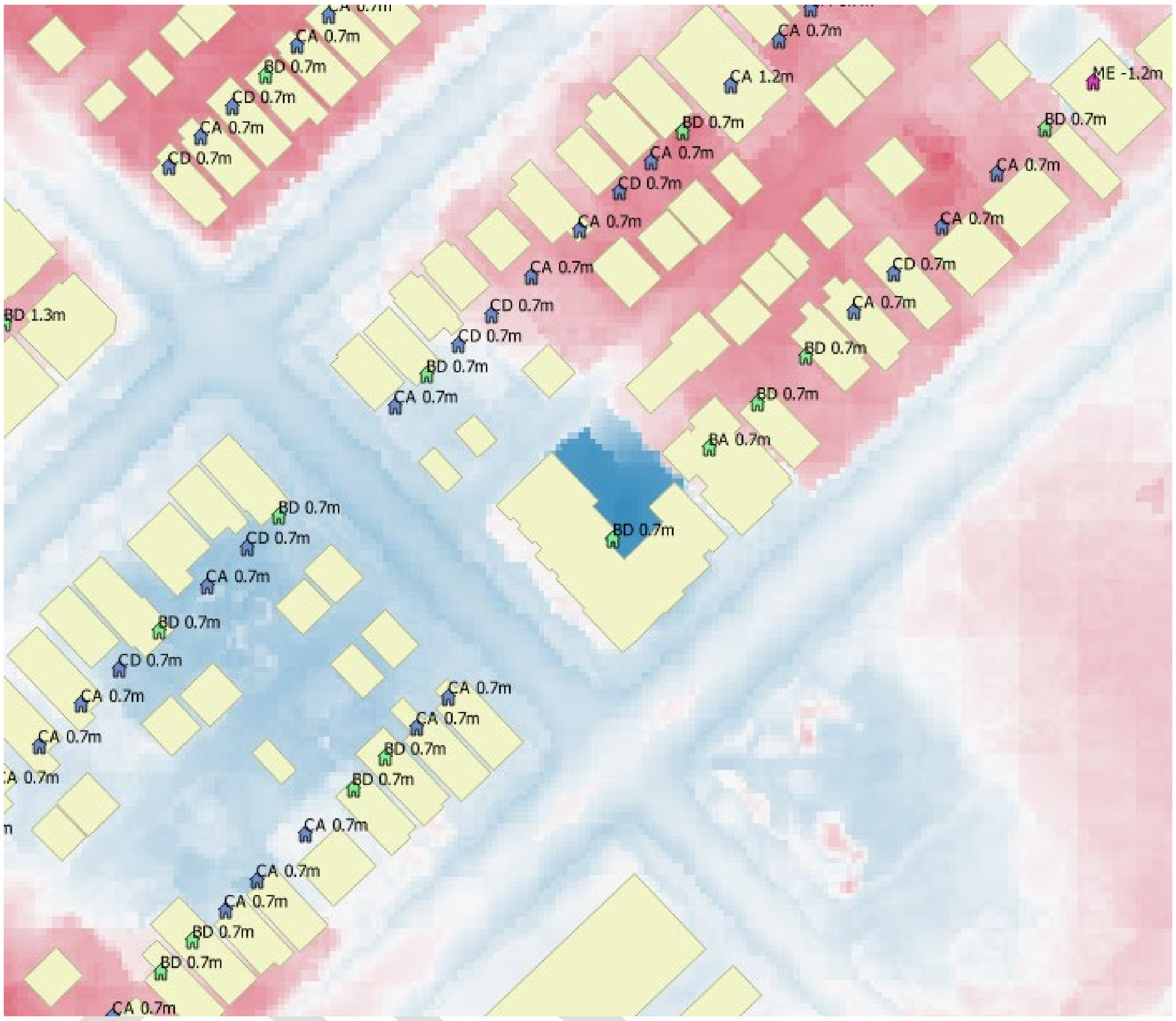


Figure 2-1: Example of spatially represented building inventory showing building type and main floor height. Yellow geometry shows the measured building footprints, red/blue shading shows a WSL raster shaded by depth above the DEM.

The primary advantage of SOFDA is the adoption of a dynamic-view of risk, or the ability to simulate the accumulation of flood risk over time. To accomplish this, SOFDA was designed as a flexible framework, within which the user can apply a wide range of changes at any point in the simulation to nearly all model objects. For example, a user can simulate the stochastic redevelopment of 10% (of assets every year) and/or an increase in flood depths every third year. Such flexibility allows for study objectives to quantify the accumulation of risk as a result of urban re-development or infilling. Such a study may be useful to explore the balance of risk increasing mechanisms, like the infilling with larger houses, against risk reducing mechanisms, like flood hazard regulations (FHRs) and structural mitigations. However, the current version of SOFDA does not support asset joining or splitting, limiting such a study to the evaluation of single-structure infills.

2.1. Modes

SOFDA can generate flood risk simulations in the following modes:

- *Stochastic:* The default mode for SOFDA, this leverages those Dynps the user provided stochastic parameters to stochastically simulate an ensemble of predictions for flood risk. The number of simulations SOFDA executes is controlled by the 'run_cnt' parameter (Table 4.3).
- *Deterministic*: Useful for testing and debugging, this mode only leverages the mean values for each Dynp to deterministically calculate one prediction for flood risk. This mode is controlled via the 'glbl_stoch_f' parameter (Table 4.3).
- *Sensitivity analysis (SA)*: Useful for identifying the sensitivity of model predications to specific parameters, this mode executes a set of deterministic simulations based on the user provided extremes for each Dynp. This mode is controlled via the 'sensi_f' flag (Table 4.3).
- *Debugging*: Useful for debugging model crashes, this mode can be layered on top of any of the above, does not influence the results, but includes more routines for error checking and log file outputting. This mode significantly reduces performance and is controlled via the 'dbg_fld_cnt'

The remainder of this section focuses on executing SOFDA in stochastic mode (glbl_stoch_f=TRUE); except for section 5.1 which discusses the SA mode.

2.2. Hierarchy

To reflect the view that a flood damage prediction should be calculated for each asset in the study area, before summing to obtain the area estimate, SOFDA spawns digital objects for each model feature, placing them in a logical hierarchy. This leverages python's object-oriented programing style. Figure 2-2 provides a simplified diagram of this hierarchy, the required inputs and how they relate to objects.

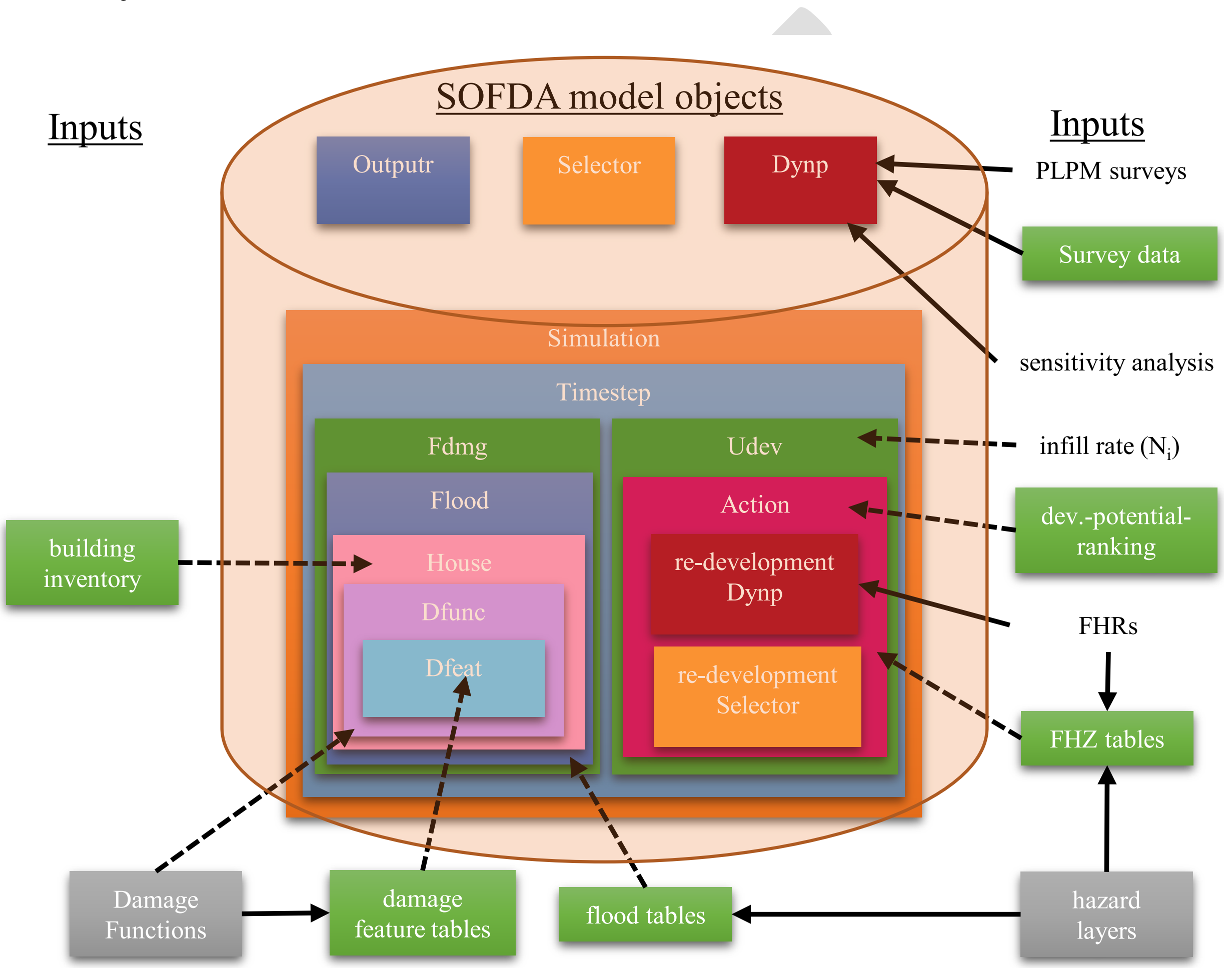


Figure 2-2: SOFDA object simplified hierarchy and input requirements conceptual diagram. Solid arrows denote 'leads' or 'contributes to' while dashed arrows represent information flow. Inputs drawn with boxes represent data sets, with gray boxes sourced from third parties while green boxes were generated by this thesis work. Some arrows and inputs omitted for clarity.

As shown, SOFDA is organized into 13 modules and objects with the following hierarchy:

- *Scenario module*: The highest level contains parameters broadcast down to all other sub-modules. These parameters are generally used to define scenarios (e.g. development rates, # of simulations).
  - *Simulation object*: As a stochastic model, numerous simulation objects are spawned by a scenario (using the same scenario parameters) to randomize key parameters.
    - *Timestep object*: As a simulation model, calculations are performed under a timestep object which contains a string of Action objects parameterized on the timeline.
      - *Flood damage module (**Fdmg**)*: This main module generates an annualized damage estimate (EAD) on the building inventory from the user provided parameters.
        - *Flood object*: Spawned for each annualized recurrence interval (ARI) provided by the user, these objects are associated with one column from the flood table.
          - *House object*: Spawned for each entry in the building inventory, these objects represent one of the 652 properties in the study area.
            - *Damage function object (**Dfunc**):* Generally, five Dfuncs are spawned for each house (MC, MS, BC, BS, GS). These build, own, and execute the loss functions predicting direct-damage from flood depths for each house.
              - *Damage feature object (**Dfeat**)*: On select Dfuncs (generally MS, BS, and GS), a Dfeat is spawned for each entry found in the corresponding damage feature table (e.g. 'replace dry-wall'). The set of Dfeats are used to generate the loss function for the parent Dfunc.
    - *Urban re-development module (**Udev**)*: This module executes Action objects to modify some other objects in the model (e.g. re-development of a house).
  - *Action object*: These objects carry out some change during model simulation of the timeline. Actions are specified with: 1) object class to be modified (e.g. House); 2) selector name (see below); 3) child actions (for chaining multiple actions together); and 4) triggered dynamic parameters (see below).
  - *Dynamic parameter workers (**Dynp**)*: These flexible worker objects change a single attribute on a group of objects (e.g. all House backflow valves = True).
  - *Selector objects*: These flexible worker objects select a group of objects based on some user provided logic (e.g. all houses inside the FHZ) for use by a Dynp, an Action, or an Outputer.
  - *Outputer objects*: These objects specify the model object attributes to output (e.g. total flood damage)

Additional object descriptions are provided in Section 4.

2.3. Flood Risk Simulations

Two diagrams are provided to demonstrate how SOFDA leverages the hierarchy to estimate EAD over time. Figure 2-3 shows how, within the highest model level (Session object), Simulation objects, then Timestep objects are nested. These Timestep objects control how the urban re-development module (Udev) updates the building inventory, before the flood damage module (Fdmg) re-calculates EAD.

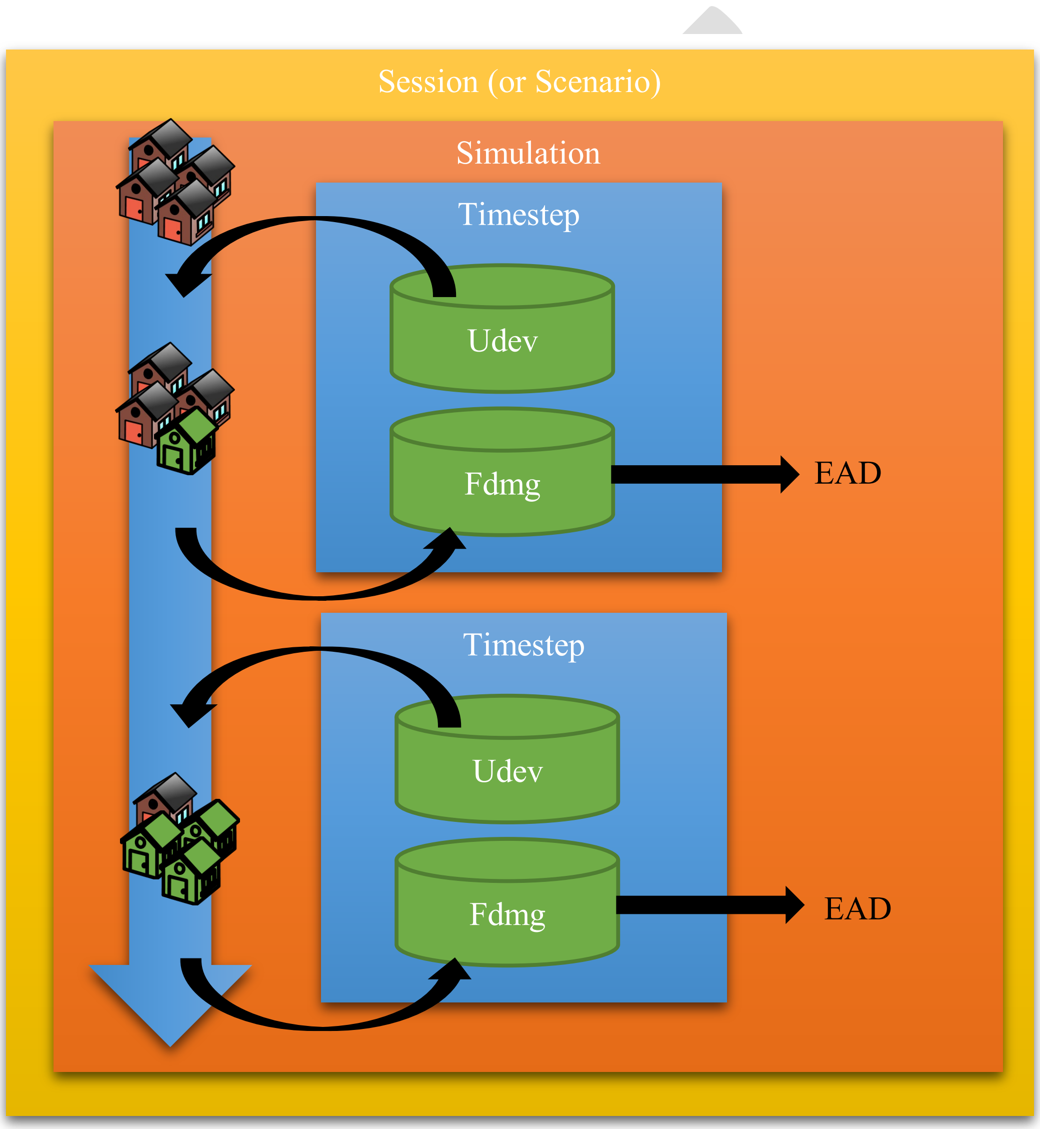


Figure 2-3: SOFDA module hierarchy conceptual diagram.

.

Figure 2-4 shows how, within this Fdmg module, Flood objects, then House objects, then Damage function objects (Dfuncs) are nested. The figure also shows how each Flood object controls the evaluation of damages from flood depths by the Dfuncs, before Fdmg calculates the total EAD.

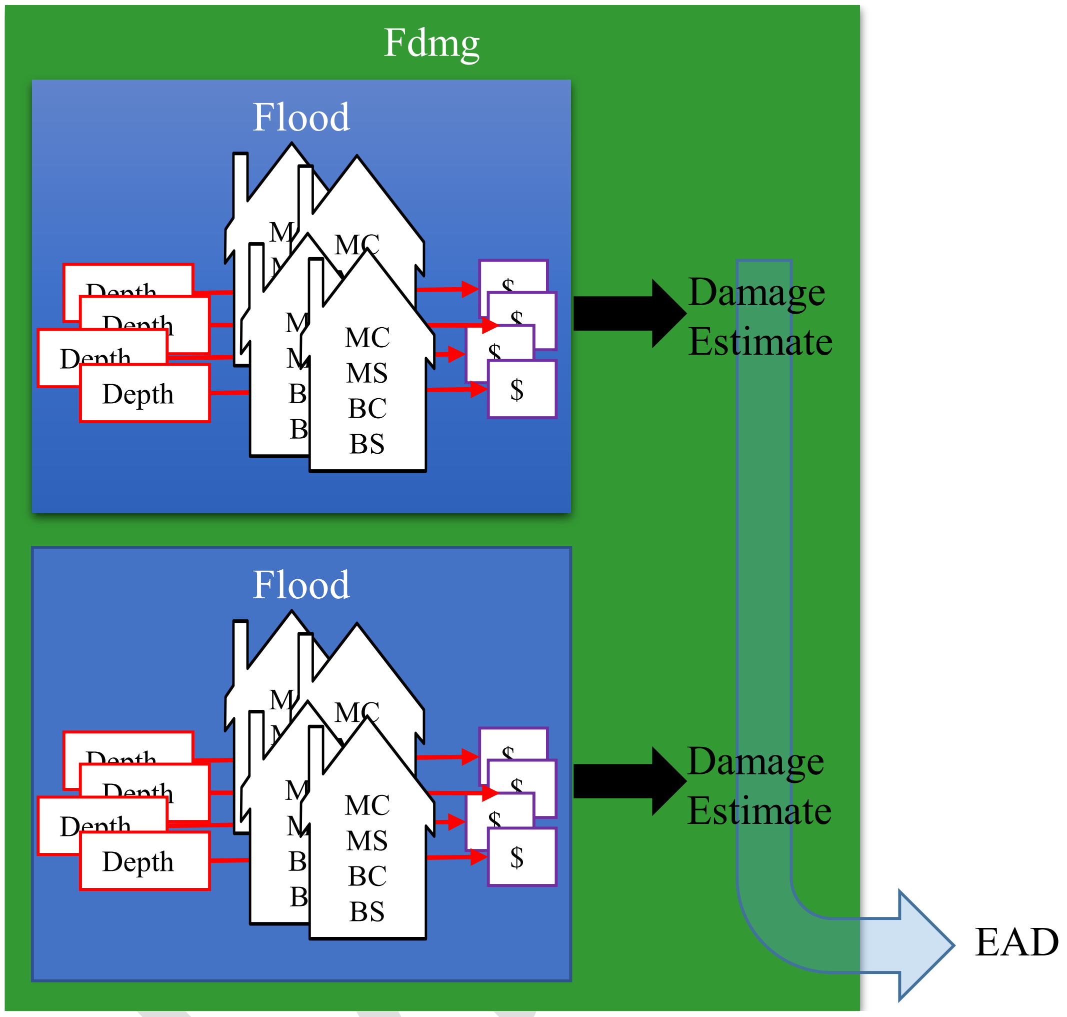


Figure 2-4: Conceptual diagram for the Fdmg module.

Session Run Sequence

SOFDA executes a stochastic model in this basic, Monte-Carlo style, sequence:

- *Build Session*: Set parameter values (e.g. number of runs) or probability distributions (e.g. normal distribution for infill basement finish height). All objects are built during this step.
    - *Run simulation*: Within the scenario parameters, a simulation is spawned with discrete parameter values generated as samples from the scenario distributions to simulate one possible forecast for the study area's dynamic flood risk. This simulation controls the execution of all the modules.
        - *Execute timeline*: During this step, the user provided timeline is executed (see next section).

Upkeep Sequence

Between each of these steps (and the timesteps of the timeline), SOFDA executes the following upkeep sequence:

1. run active Selectors;
2. run active Dynps;
3. execute the module or object; then
4. calculate post-run metrics and execute Outputrs.

Dynp and Selector Activation

Dynps and Selectors can be activated in two ways:

- *explicitly*: These are only activated when named and called by some other model object (e.g. by the timeline). This facilitates intermittent or irregular model updates.
- *periodically*: These are activated by the upkeep sequence, where the model simulation level is less than or equal to the objects 'upd_sim_lvl' (Table 2.1). For example, a Selector with upd_sim_lvl = 2 would re calculate the objects within its selection at the start of each timestep.

Table 2.1: Periodic activation upd_sim_lvl.

Calling Object	**upd_sim_lvl**
Session	0 (never updates)
Simulation	1
Timestep	2
Model	3

2.3.1. Simulation Timeline

The timeline is how the user schematizes the time dimension in SOFDA from the Control File. This is where the user tells SOFDA what to do for each timestep. With the timeline, the user can pass a complex and customizable sequence of operations to the Fdmg module and the Udev module for each timestep via the 'run_seq_d' list. This 'run_seq_d' list accepts a string of length 2 tuples: 1) module name to call; and 2) command sequence to execute on that module. Each module command sequence accepts Action names, or special commands ('*'), as shown in the following table:

Table 2.2: Accepted commands in the timeline module command sequence.

Command	**Description**
*run	Execute the module's main 'run()' method.
*model.[some module function]	Execute the provided method (e.g. when paired with the Fdmg module, '*model.plot_dmgs()' calls Fdmg.plot_dmgs().
[some Action name]	Execute the named action's 'run()' method (see section 4.7).

The below table gives a simple example of a timeline with three timesteps.

Table 2.3: Example timeline in SOFDA.

Timestep name	**Timestep execution description**	**run_seq_d**
t0	Calculate EAD on current inventory	[('Fdmg',['*run'])]
t1	1) Call the 'a_redev' Action to simulate urban re-development on the inventory; 2) calculate the new EAD	[('Udev',['a_redev']), ('Fdmg', ['*run'])]
t2	same	[('Udev',['a_redev']), ('Fdmg', ['*run'])]

During startup, a timestep is spawned for each row of the timeline. These timesteps are executed in sequence by each simulation:

1) Update model objects;
2) Run Selectors;
3) Run Dynps;
4) Execute each module in the run sequence;
    a. Run Selectors;
    b. Execute each command in the command sequence;
        i. Update model objects;
        ii. Execute command (see Table 2.2);
    c. Get module results;
5) Get Timestep results;

2.3.2. Flood Damage Module (Fdmg)

Figure 2-5 shows the calculation loop of the for Fdmg's main 'run()' method.

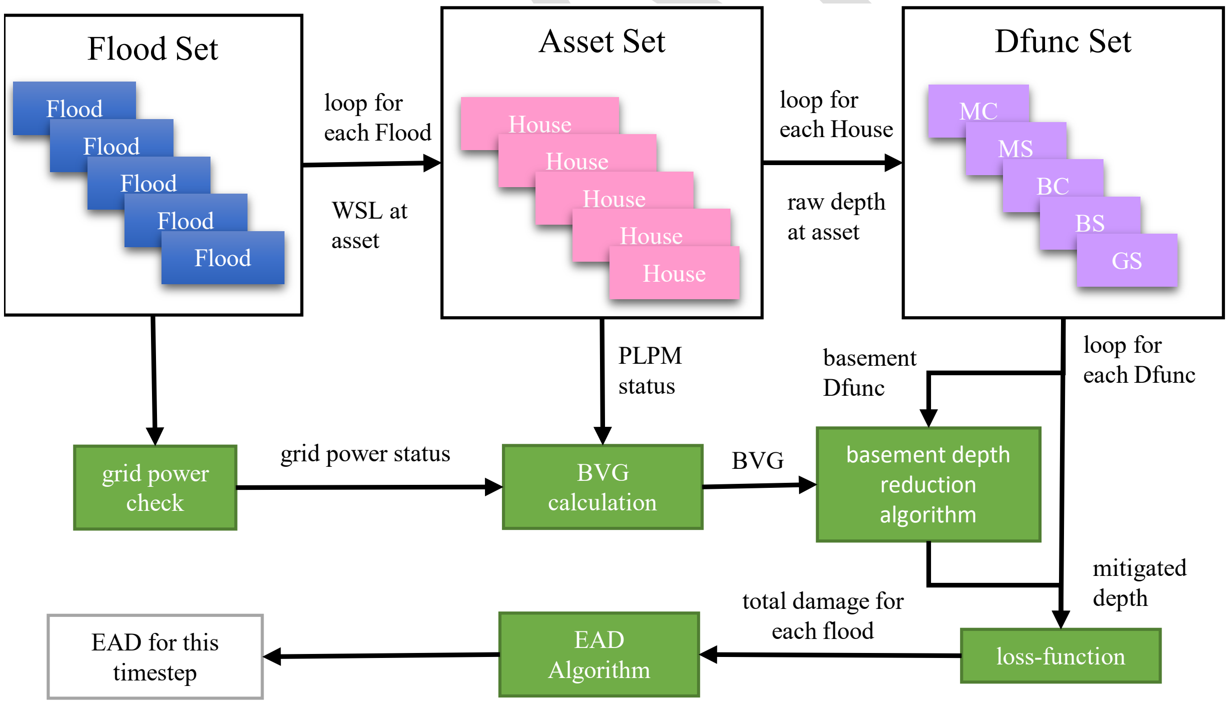


Figure 2-5: SOFDA's Fdmg 'run()' method calculation diagram.

Area Protection Reliability Approximations

To incorporate the reliability of area protections like stormwater pumps and levees into the risk estimate, SOFDA can accept a hazard scenario (via the flood tables) with two datasets: 1) area protections 'fail' (wet); and 2) area protections 'perform' (dry). From these two, a third intermediate case can also be estimated (damp) using the 'damp_build_code' on the 'flood_tbls' tab. Once these three datasets are loaded for each flood table, the desired WSL is applied to each asset based on the logic provided on the 'floods' tab and the asset attribute 'area_prot_lvl' (specified on the 'aprot' tab of the flood table file). SOFDA provides the user three options to calculate the WSL selection logic based on the House's 'area_prot_lvl' attribute:

1. *Uniform*: by specifying 'dry', 'damp', or 'wet,' all Houses will receive WSLs from the version of the flood table (ignoring its area_egrd code). This is useful for simple simulations where the user wants to specify which area_prot_lvls should receive which flood table version for which Floods.
2. *From the flood_tbls tab:* This tells SOFDA to use the codes from the flood_tbls tab based on the current flood table.
3. *From an Fdmg attribute*: This tells SOFDA to use whatever attribute it finds on the Fdmg object. Generally, this is a uniform user provided value (via the 'gen' tab for area_egrd00, 01, and 02).

As a simple example of #1 with only one structural protection scenario (i.e. only one flood table), consider a situation where 5 houses are protected by levee 'A' and another 5 by levee 'B'. We trust that levee 'A' will perform for all floods, but as levee 'B' is old and unreliable, we assume it will fail 50% of the time for flood ARIs > 100. To approximate this, we would code the 5 houses behind levee 'A' with area_prot_lvl = 0 and the 5 behind levee 'B' with area_prot_lvl = 1 (in the flood table file 'aprot' tab). We can now manipulate which version of the flood table is applied to each group separately through the *area_egrd00_code* and *area_egrd01_code* on the 'floods' tab for house group 'A' and 'B' respectively. For *area_egrd00_code* we set to 'dry' for all floods, because we expect those houses to always be protected. To simulate the increased exposure of the 'B' group, we have two options: 1) apply the 'damp' WSLs to these; or 2) use a Dynp to stochastically manipulate the *area_egrd01_code* during each time step.

In closing, the user is reminded that the above described algorithm is a simple approximation to handle hazard model results that show WSL versions with and without area protection failure. This approach is only appropriate during minor floods (low WSL gradients) where the reliability of all area protections influencing the exposure of a single asset can be expressed with a single value. However, it is preferable in most situations to obtain a single likelihood-WSL function for each asset from the hazard study. The hazard study is better suited for this, as it allows a mechanistic (or empirical) evaluation that considers the connectivity between overlapping protections (e.g. areas protected by a levee and a pump) and the forces that influence reliability (e.g. water velocity).

2.3.3. Urban Re-Development Module (Udev)

Unlike the Fdmg module, the Udev module is mostly a vessel for Action objects. Udev has no 'run()' method, but does collect metrics on changes to the building inventory following Action executions.

3. Running SOFDA

Once SOFDA is installed and a model defined with a control file (Section 4), a user can execute the model in the SOFDA platform. In the CanFlood implementation, two options are available:

1) Executing the model from the plugin
2) Executing the model from command line (or batch file)

Command Line

Running SOFDA from command line provides more flexibility and improved performance. To run SOFDA from command line, open a command shell in the installation folder and call:

```
sofda
```

This will launch SOFDA with all the default flags, prompting the user for a control file (double clicking sofda.exe has the same effect).

The SOFDA syntax is:

```
sofda [options]
```

All of the available program flags are documented and can be displayed with the help flag:

```
sofda -h
```

For example, to execute a SOFDA model with debugging enabled and partial data loading (useful for troubleshooting during model development), the partial data loading and master debugging flags can be specified:

```
sofda -part -db all
```

For the experienced user, windows batch files can be created to select input files, direct outputting and logging, supress GUI prompts, schedule multiple runs, etc.

Depending on the complexity of the analysis, SODFA can be resource intensive and slow. This is amplified because the tool has poor memory cleanup and is single threaded. As a workaround, we recommend splitting the simulation and executing in parallel using separate batch files.

3.1. Logging

During command line execution, SOFDA generates messages to notify the user what is happening when. By default, high level messages are displayed in the windows command shell. Log files store these messages to communicate model tasks to the user, providing a window into the model. Understanding and using log files is essential for model debugging. However, large models can produce extremely long and repetitive log files. Each log file entry records the system time, logging level, the logger, and the model object hierarchy generating the message. For example, the log entry:

```
2019-01-16 01:07:32,497.INFO.S.sim_0.t0.fdmg.calc_fld_set:  calc flood damage (12) floods w/ wsl_delta = 0.00
```

Tells us that the first simulation's first timestep has called the 'fdmg.calc_fld_set' method, the main flood damage estimator call, on 12 flood objects.

By default, SOFDA creates two log files:

- *Standard*: This logs all high-level messages (INFO and WARNING) and, if executed in debugging mode, also logs low-level messages (DEBUG).
- *Warning only*: This only logs WARNING messages.

By default, these two log files are written in two places each (for a total of 4 log files):

- *Root logger*: this records the most recent SOFDA run and is found in the working directory (root.log and rootwarn.log).
- *Session logger*: this only records a single session and is stored with the session outputs (S.log and S_WARN.log).

This dual-recording facilitates parallel execution of SOFDA and debugging.

3.2. User Data Files

A fully defined SOFDA model requires numerous user data files, which are referenced in the control file. The types of user data files accepted by SOFDA are summarized in Table 3.1.

Table 3.1: User data file summary

Data file types	**Control file tab**	**Description**	**File format**	**Section**
Basic data files	datos	Four basic data files used by the Fdmg module.	.xls	4.2
Flood tables	flood_tbls	Simulated flood depth per asset per ARI	.xls	4.5
Special selector lists	selectors	Ranked list of object names	.csv	4.9

3.3. Outputs

SOFDA provides user customizable outputs hierarchically with varying levels of detail and format. How and what outputs are generated are set on the 'gen' and 'outputs' tab of the control file.

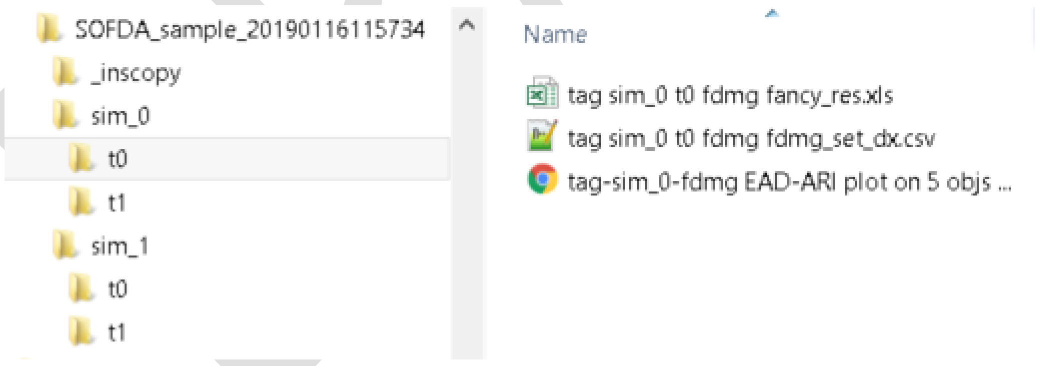
The available output files are summarized in Table 3.2 and further classified in the following sections.

Table 3.2: Output file summary table. See text for column descriptions.

Output type	Format	Level	Description	Name syntax	Control parameter
Outputr	fancy	Session	Main session Outputr results summary	TAG fancy_res.xls	write_fancy_outs
Outputr	fancy	Simulation	Fancy Outputr results detailed for each Simulation.	TAG SIM sim_res.xls	write_sim_res_f
Outputr	raw, fly	Session	Included in the fancy results, this provides Outputr results per simulation.	TAG fly_res.csv	write_fly_f
Outputr	raw, fly, dx	Session	Like the above, but includes the time dimension.	TAG fly_res_dx.csv	write_fly_f
Fdmg	raw, fly	Session	Reports flood damage estimates per flood ARI, timestep, and simulation	TAG fdmg_res_fly.csv	write_fdmg_sum_fly
Fdmg	fancy	Timestep	Reports the full results for a Fdmg run, showing asset attribute values (including damage) for each flood ARI.	TAG SIM TIME fdmg fancy_res.xls	write_fdmg_fancy
Fdmg	raw	Timestep	Like the above, but compressed into one csv	TAG SIM TIME fdmg res_fld.csv	write_fdmg_set_dx
Fdmg	raw, fly	House			write_fly_bdmg_dx

3.3.1. Hierarchy

To better organize outputs, SOFDA generates a folder structure in the output directory matching the basic hierarchy of SOFDA. For example, a session with two simulations with two timesteps each would produce a folder tree like this:


SOFDA_sample_20190116115734
_inscopy
sim_0
t0
t1
sim_1
t0
t1
Name
tag sim_0 t0 fdmg fancy_res.xls
tag sim_0 t0 fdmg fdmg_set_dx.csv
tag-sim_0-fdmg EAD-ARI plot on 5 objs ...

General or session level output files are stored in the main output folder ('SOFDA_sample_20190116115734' in the above example), and inputs are duplicated to the _inscopy subfolder (with the _write_ins flag).

3.3.2. Types

The main output type of SOFDA is configured and generated by the 'Outputr' objects described in Section 4.11. These allow the user to select specific attributes on specific objects at specific time steps, control how these data are summarized, and facilitate simple post-processing (e.g. figure creation). Additional simple, non-configurable output types are also available by module (e.g. 'Fdmg'). The user can specify which output files are generated by specifying 'TRUE' for the corresponding control parameter.

3.3.3. Formats

The two basic stylistic formats are:

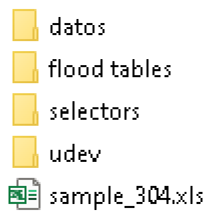
- *Fancy*: These are spreadsheets (.xls) generated for easy direct reading by the user. Numerous tabs are provided to further divide the data. This information is generally duplicated in the raw outputs.
- *Raw*: These are .csv files generated for easy post processing and data manipulation.

Raw data files may have two additional options:

- *On-the-fly ('fly'):* Rather than write the results file at the end of program completion, this mode updates the output file as soon as the information is available. This is useful for large Sessions that may crash.
- *Time dimension ('dx'):* Rather than write a single output value per session, these include a second dimension to report output values at each time step.

3.4. Tutorial

The SOFDA distribution comes with a sample control file and user input data files in the sub folder '_CanFlood\tutorials\3\' which should look like this:


datos
flood tables
selectors
udev
sample_304.xls

To run this sample model, run the provided 'sample.xls' control file.

4. Building a SOFDA Model

Once the user has prepared the data files and familiarized themselves with the basic functions and purpose of SOFDA, the next step is to build the flood risk model using the SOFDA framework. An efficient model builder should progressively run and test the model before it achieves the desired final complexity. As discussed in Section 2, a model is defined with the control file, which points to all other user data files. Model construction is an iterative process which typically follows these basic steps:

1. Format all data files for use in SOFDA;
2. Assign partial interim values for the global parameters which facilitate model development (e.g. glbl_stoch_f=False, run_cnt=2, _parlo_f=True);
3. Define file locations for all user data files;
4. Define static parameters on ‘fdmg’, ‘dfunc’, ‘hse_geo’, ‘floods’ tabs;
5. Define a simple interim static timeline (e.g. '[('Fdmg',['*run'])]);
6. Define a set of basic outputs (and Selectors if necessary);
7. Test interim partial static model — debugging if necessary;
8. Define parameters on ‘timeline’, ‘actions’, ‘selectors’, ‘dynp’ tabs;
9. Test interim partial dynamic model — debugging if necessary;
10. Assign final global parameter values for a single simulation (e.g. glbl_stoch_f=True, run_cnt=1, _parlo_f=False);
11. Test interim dynamic model — debug if necessary;

In this way, a single model scenario can be built in SOFDA. Section 3 describes how to execute the model once built. For multiple scenarios, a basic ‘parent’ model can be developed first before branching off ‘child’ scenarios or models. However, to save yourself many headaches, be sure the ‘parent’ model has no errors before branching. Additional testing and interim model values should be considered depending on data quality, model complexity, and user judgement. The remainder of this section provides guidance on how to parameterize each of the tabs on the control file.

Inputs Summary

Information that controls the execution of SOFDA, which is separate from the source code, can be divided by user interaction type:

- *Execution parameters*: These are high level parameters that control how the model is executed in python. Except for those parameters described in Table 4.1, these are typically only modified for debugging and source code development.
- *Program data files:* These are internal data files that are outside the source code. Typically, the user does not interact with these.
- *Model input parameters*: These define a SOFDA Session and parameterize how it is executed. The user provides these through the SOFDA Control File, an excel spreadsheet with 11 (active) tabs described in Table 4.2. A complete sample control file is provided in Attachment B and section 0 gives a description of each tab.
- *Model data files*: Connected to a SOFDA Session via special user input parameters, these external data files and their parameters help define a SOFDA model.

Table 4.1: Execution parameter summary table.

Input name	**Code**	**Description**
User control file name	pars_filename	File name for the user control file found in the input folder.
Global debugging mode control	_dbgmstr	Parameter to control the debugging mode.
Output folder name	out_fldr	Folder name to place output files.
Input folder name	in_fldr	Folder name to search for the user control file.

This manual focuses on model input parameters and data files summarized in Table 4.2. The remainder of this section details the inputs required for each tab in the Control File, with some additional object description.

Table 4.2: SOFDA control file description. See below for a detailed description of each tab.

Input name	**Code**	**Description**
Global parameters	gen	Tab with high level control parameters for model function (e.g. debug mode control, number of simulations) and some default values (e.g. basement opening height).
Fdmg datasets	fdmg	Tab assigning model data files for the Fdmg module
Dfunc parameters	dfunc	Tab for assigning Dfunc properties for each acode
House geometry parameters	hse_geo	Tab for assigning default geometry logic for houses.
Flood table datasets	flood_tbls	Tab assigning flood tables and configuring their area protection grades.
Flood table	flood_tbl	Model data file with entries for each building in the inventory and tabs: *wet*) WSL for area protection failure; *dry*) WSL with area protections performing; *aprot*) area protection level.
Flood object set parameters	floods	Tab to provide the ARI for each flood event (and area protection code)
Action set parameters	actions	Tab to schematize each model Action object.
Selector set parameters	selectors	Tab to schematize the Selector objects.
Ranked choice list		Model data file with a ranked list of object names (for re-development selection).
Dynp parameters	dynp	Tab to schematize each Dynp.
Timeline	timeline	Tab to specify the sequence of model Actions.
Output setup	outputs	Tab to schematize Outputr objects.

4.1. Global Parameters

Global parameters provide the high-level control parameters for a SOFDA model (e.g. debug mode control, number of simulations). Additionally, some key default values (to pass to the modules) can be entered here (e.g. basement opening height).

Table 4.3: Global parameter summary table. For a complete set, see Attachment B.

Input name	**Code**	**Description**
Session run count	**run_cnt**	number of simulations to run --for deterministic runs: set to 1 --for stochastic (monte-carlo): set to many --for sensitivity analysis: set the maximum number of toggles to evaluate
Session sensitivity analysis mode flag	**sensi_f**	flag whether to run in sensitivity analysis mode --TRUE: ignores run_cnt. instead does 1 run for each value on each variable on the pars tab --FALSE: (default) execute with normal Dynp behavior
Session stochastic mode flag	**glbl_stoch_f**	flag whether to use [TRUE] stochastic Dynps (default) or [FALSE] deterministic Dynps.

4.2. Basic Data Files

On the 'datos' tab in the control file, the user specifies file locations for the four basic datafiles used by SOFDA:

Input name	Code	Description
Alberta Curves	rfda_curve	Model data file with Alberta Curve depth-damage values formatted for use in RFDA.
Building Inventory	binv	Model data file with vulnerability data on each asset (e.g. main floor height, building type). The first 26 columns can be formatted for use in RFDA.
Damage feature tables	dfeat_tbl	Model data file and tabs with damage feature data. Attachment C provides the default tables, developed from the original Alberta Curve damage feature tables.
FHZ tables	fhr_tbl	Model data file with FHZ and BFE on each asset for each FHR.

These data files are Microsoft Excel spreadsheets (.xls). Two of these are described below.

4.2.1. Building Inventory

The building inventory contains vulnerability attributes for each asset in the study area, with variables like building type, main floor height, and basement presence. This dataset is meant to describe the study area as it is today, in terms that are relevant for flood risk modeling of direct building damages. SOFDA builds a digital model of the study area by spawning a House object (and its nested hierarchy) from the values in each row of this building inventory during model startup.

For backwards compatibility of model data files, SOFDA can convert building inventories from the legacy RFDA format with the legacy_binv_f=True parameter value. This tells SOFDA to read the building inventory based on location (column index) rather than the header value, as shown in the following table. Those attributes required by the basic Fdmg.run() method are highlighted in green.

Table 4.4: Typical building inventory attribute description. Green rows indicate those attributes required for the Fdmg.run() method.

Attribute name	**Typical legacy datafile code**	**Legacy datafile index**[a]	**SOFDA code**	**Attribute description**
Identifier	ID	0	ID	Arbitrary unique asset identifier
Asset address		1	address	
Data identifier	CPID	2	bid[b]	Arbitrary unique asset identifier that corresponds to other model data files
Asset class	ClAss	10	class	
Asset stories	StruCt_Typ	11	struct_type	
Building area	area_GIS_m	13	gis_area	Asset area for scaling Dfuncs
Basement status	Bsmt-Prkd	18	bsmt_f	Flag indicating whether House should spawn Dfuncs with place_code = 'B'.
			acode_s	
			acode_c	
Main floor height	Height_m	19	ff_height	Height used to calculate House.anchor_el (added to House.dem_el).
X-coordinate		20	xcoord	
Y-coordinate		21	ycoord	
DEM elevation	integrated	25	dem_el	Ground elevation from which to calculate House.anchor_el
Property land value			land_value	
Property total value			value	
Development-potential-ranking			devpot_rnk	
Year of construction			ayoc	
Asset's parcel area			parcel_area	

a) When legacy_binv_f=True, these attributes are loaded from the building inventory data file based on these index values, rather than the header value.

b) The main asset identifier is controlled with the 'mind' variable on the gen tab.

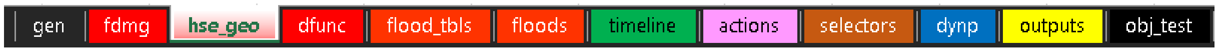
4.2.2. Damage Feature Tables

To facilitate manipulation, and allow for more accurate object scaling, SOFDA includes the 'damage feature curve' mode for Dfuncs (dfunc_type = dfeats). This is assigned on the dfunc tab. With this novel algorithm, depth-damage curves are generated directly from damage feature tables. Typically, these tables use the results from the 2014 Alberta Curve surveys. These tables record typical restoration activates that may be required for a given depth of flooding (e.g. 'remove and replace water heater'). Typical damage feature tables are provided in Attachment C. Each entry (i.e. line) of these tables is referred to as a ‘damage feature’ (and each column provides the Dfeat attributes). Damage features are discussed further in section 4.4.1.

4.2.3. FHZ tables

The FHZ Tables model data file allows the user to provide ‘bfe’ and ‘fhz’ attributes to each asset via the model index value (e.g. ‘CPID’). Each tab in this data file is loaded by name, then accessed via the global parameter ‘fhr_nm’. In this way, the user can simulate the influence of different FHRs by changing the BFE and FHZ of attributes.

4.3. Houses



House objects in SOFDA correspond to the object-based asset-level predictions of flood damage, as framed by the Alberta Curves. For single-family homes, these correspond to a single parcel and facilitate the estimation of damages for the primary structure and any accessory structures (i.e. garages) on this parcel.

Table 4.5: House object key attributes.

Name	Code	Description	Source
Name	name	Unique string identifying the House.	If no 'name' column is found in the building inventory, the 'mind' column is used (with an 'h' appended to the head).
Basic attributes	n/a	see Table 4.4(green items)	Building inventory
Geometric attributes container	geo_dxcol	Dataframe (owned by each House) containing the values of each geometric attribute (looks similar to 'hse_geo' tab) (see description below)	Logic provided on the 'hse_geo' tab.
Basement finish height	B_f_height	Height to ceiling in basement.	Unlike the other values in the geo_dxcol, the basement finish height is referenced explicitly.
House's anchor elevation	anchor_el	The elevation at which the House's relative flood depths are considered zero.	set_hse_anchor()
House type or class	acode_s	Type or class of the asset (generally corresponding to the Alberta Curves)	For legacy building inventories, this is a concatenation of the 'class' and 'struct_type' columns. Otherwise, this must come straight from the building inventory.
	acode_c		
Backflow valve presence	bkflowv_f	Flag indicating the presence of a backflow valve	building inventory
	sumpump_f	Flag indicating the presence of a sump pump	building inventory
	genorat_f	Flag indicating the presence of a generator	building inventory
Basement vulnerability grade	bsmt_egrd	'dry', 'damp', or 'wet', grade indicating the performance of PLPMs during basement flooding.	Calculated based on the bsmt_egrd_code
Garage anchor height	G_anchor_ht	Height of garage anchor relative to the House.anchor_el	Generally, a default value is provided by the user via the control file 'gen' tab (0.6 m).
Joist spacing	joist_space	Distance between the basement ceiling and the House.anchor_el.	Generally, a default value is provided by the user via the control file 'gen' tab
Basement opening height	bsmt_opn_ht	Height from the basement floor to the lowest outside opening.	set_bsmt_opn_ht()
Spill height for bsmt_egrd=damp	damp_spill_ht	Height used for bsmt_egrd = damp & damp_func_code = spill	half the bsmt_open_ht
Area protection level	area_prot_lvl	Integer indicating the level of area protections for this asset (0, 1, or 2)	flood tables 'aprot' tab.

Geometry

During startup, Houses are spawned, and given attributes from each entry in the building inventory. Typically, only the floor area (gis_area) attribute is provided to describe the geometry of a House. To facilitate accurate scaling of Dfeat price attributes, SOFDA provides algorithms to calculated more nuanced House geometry from simple geometric assumptions (scaled from the floor area). For example, the price of a Dfeat like 'remove and replace all drywall to walls & ceilings' should not be scaled by the floor area of the house, but by the interior area ('f_inta'). To calculate these secondary geometric attributes, the 'set_geo_dxcol' method is called during startup (and re-called if a base attribute is modified).

This 'set_geo_dxcol' method calculates the floor area, height, perimeter (per), and interior area (floor + walls; m^2) or the main floor (M), basement (B), and garage (G) from the user supplied parameters on the 'hse_geo' tab. Once calculated, these attributes are stored in the 3 dimensional 'geo_dxcol' dataframe to facilitate access by other routines.

Startup

Once the basic attributes are assigned from the Building Inventory, each house executes the following functions to calculate their secondary attributes:

1. *Calculate geometry (set_geo_dxcol):* see above
2. *Calculate anchor elevation (set_hse_anchor):* calculates the House's anchor elevation (House.anchor_el) based on the DEM elevation and the main floor height (first provided in the building inventory).
3. *Calculate the basement vulnerability grade (set_bsmt_egrd)*: calculates the 'bsmt_egrd' based on the grid power (model.gpwr_f) and PLPM status as shown in Table 4.6.

Then, if the House has a basement, these secondary basement attributes are calculated:

4. *Calculate basement opening height (set_bsmt_opn_ht):* calculates the 'bsmt_opn_ht' value based on the global 'bsmt_opn_ht_code'. This value is used to calculate the basemetnt spill height and during the Dfunc. get_depth() method to modify the flood depth when House.bsmt_egrd = 'dry' .
5. *Calculate the basement spill height (set_damp_spill_ht):* calculates the 'damp_spill_ht' value as half the 'bsmt_opn_ht' parameter. This value is used during the Dfunc.get_depth() method to modify the flood depth when House.bsmt_egrd = 'damp' and model.damp_func_code = 'spill'.

During the upkeep sequence, each of these functions can be queued and re-run as a result of some modification instigated by a Dynp.

Table 4.6: Basement vulnerability grade (BVG) calculation logic used by set_bsmt_egrd().

grid power = ON	**grid power = OFF**	**BVG**
valve & pump	valve & pump & generator	**dry**
valve OR sump	valve OR (pump & generator)	**damp**
otherwise[a]	otherwise[a]	**wet**

c) any combination not included in the above

4.4. Dfuncs

Damage Function objects (Dfuncs) generate damage predictions from flood depths in consideration of an asset's vulnerability. Each House object spawns Dfuncs based on the input parameters for place (e.g. garage, basement, main floor) and damage type (e.g. structural or contents). Following this, Dfuncs specified as 'damage feature curves' (discussed below) spawn a collection of Dfeats from the damage feature tables, as shown in the following figure:

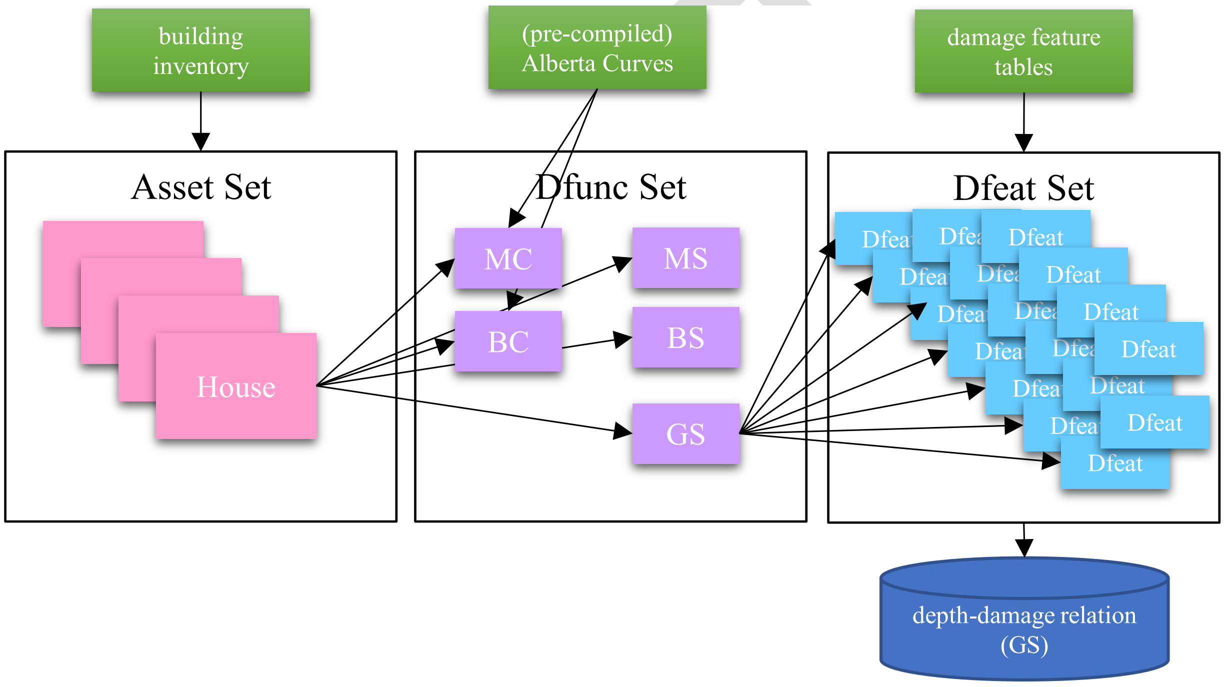


Figure 5-1: Typical house object hierarchy.

Generally, four Dfuncs are specified for each House to simulate Main/Basement Structural/Contents damages considered by the Alberta Curves. The parameters controlling the operation of each of these dfuncs by asset acode are configured on the 'dfunc' tab as shown below.

Table 4.7: Typical Dfunc model parameters.

acode	asector	place_code	dmg_code	dfunc_type	rat_attn	anchor_ht_code
AA	sres	M	C	rfda	self.parent.gis_area	*hse
AA	sres	M	S	rfda	self.parent.gis_area	*hse
AA	sres	B	C	rfda	self.parent.gis_area	*hse
AA	sres	B	S	rfda	self.parent.gis_area	*hse
AD	sres	M	C	rfda	self.parent.gis_area	*hse
AD	sres	M	S	dfeats	*none	*hse
AD	sres	B	C	rfda	self.parent.gis_area	*hse
AD	sres	B	S	dfeats	*none	*hse
BA	sres	M	C	rfda	self.parent.gis_area	*hse
BA	sres	M	S	dfeats	*none	*hse
BA	sres	B	C	rfda	self.parent.gis_area	*hse
BA	sres	B	S	dfeats	*none	*hse

Startup

During startup, each Dfunc executes the following functions:

1. *Calculate the anchor elevation:* The anchor elevation of the house, and the place code of the Dfunc are used to calculate the anchor elevation of the Dfunc, as shown on Table 4.8.
2. *Build the loss function:* depth-damage arrays are compiled based on the House and Dfunc attributes — especially the user provided 'dfunc_type'

During the upkeep sequence, each of these functions can be queued and re-run as a result of some modification instigated by a Dynp.

Table 4.8: Dfunc anchor elevation formulas used in this study. See Appendix E for a complete description of Dfunc parameters.

Place code	Anchor elevation formula
Main Floor (M)	House main floor elevation (HMFE)
Basements (B)	HMFE – basement finish height – joist spacing

4.4.1. Depth-Damage Arrays

To improve performance, discrete loss functions are compiled during startup (and, if necessary, during updating) for each Dfunc. These are numpy arrays with 'depth' and 'damage' columns. during the Fdmg.run() method, these depth-damage arrays can be quickly interpolated (by the Dfunc.get_dmg() method) to calculate the damage corresponding to the passed depth. The method used to compile these depth-damage arrays is controlled with the 'dfunc_type' parameter for each Dfunc class on the 'dfunc' tab in the Control File as shown in the following table:

Table 4.9: Dfunc depth-damage array (dd_ar) compilation method options by dfunc_type parameter.

dfunc_type	Method name	Method description
rfda	get_ddar_rfda	Load the dd_ar directly from the 'rfda_curve' model data file (in the Alberta Curve format).
dfeats	raise_dfeats	Build the dd_ar as damage feature curves from the child Dfeats initially loaded from the damage feature tables (section 4.2.2).
depdmg	get_ddar_depdmg	Load the dd_ar directly from the specified model data file (in the headpath/tailpath columns).

Damage Feature Curves

With the 'dfeats' parameter, SOFDA builds a custom loss function where each component is exposed to the user for manipulation. This loss function is comprised of a set of Damage Feature objects (Dfeat) spawned from each line (on the corresponding tab) of the damage feature tables described in section 4.2.2. Using the global parameter 'dfeat_xclud_price', a filter for the minimum Dfeat value can be set to improve performance. Once the full Damage Feature object set is spawned, the Dfeat prices are summed at each depth and the depth-damage array is generated and made ready for the first damage estimate.

A sample collection of the loss functions generated by five Dfuncs for a class 'C' house is provided in Figure 4-2.

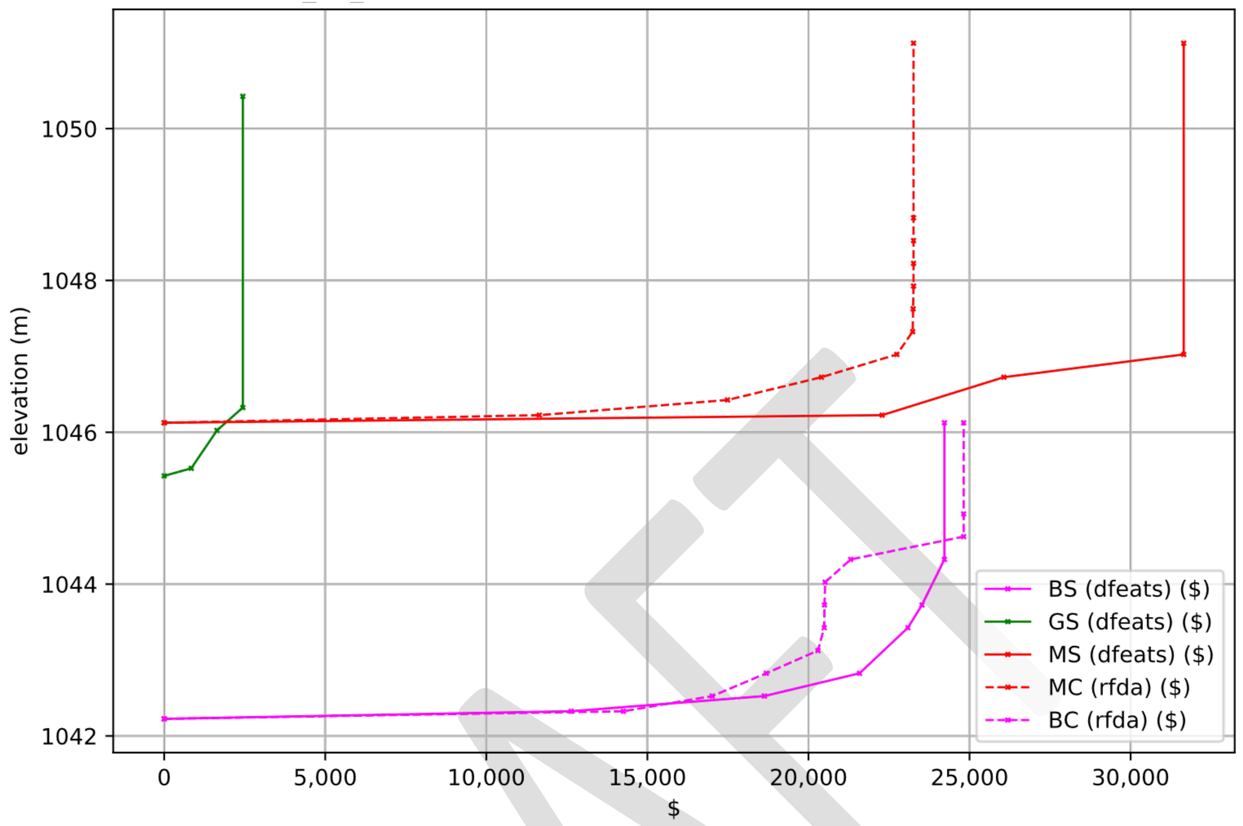


Figure 4-2: Depth-damage relations on five Dfuncs for a typical asset in SOFDA. Dfuncs marked with '(dfeats)' are generated in mode 'Damage Feature Curves' while those marked with '(rfda)' are 'Alberta Curves.'

4.5. Flood Tables (Exposure Variables)

Flood Tables contain hazard attributes for each asset in the study area, with flood WSL provided for each of the Floods considered. These datasets are generated from river model predictions for flood WSL under different scenarios for discharge and structural protections.

Multiple flood tables can be loaded into SOFDA to simulate changes in hazard or area protections. The flood tables are queried during the Fdmg.run() method based on the 'flood_tbl_nm' parameter value.

Area Protections

To facilitate simulations that consider the reliability (i.e. likelihood of failure) of area protections, SOFDA considers three versions of the WSLs in each flood table — similar to the House.bsmt_egrd discussed above. These three WSL versions are described in the following table:

Table 4.10: Flood table area exposure grades and corresponding WSL generation method.

area_egrd	**Typical performance scenario**	**WSL generation method**
wet	Failure of area protections	WSL values from the 'wet tab on the data file
damp	Partial performance of area protections	Calculated based on the 'damp_build_code' parameter.
dry	Full performance of area protections	WSL values from the 'dry' tab on the data file

Which of these three WSL versions are used during the Fdmg.run() method is controlled via the 'area_egrd' matrix on the 'flood' tab of the Control File. Further, on the third 'aprot' tab of the flood tables, each asset is assigned an 'area_prot_lvl' of 0, 1, or 2. For study areas with heterogeneous structural protections (e.g. a levee only protecting a few houses), these asset scale area_prot_lvl attributes can be used to assign different versions of the flood table WSLs to different assets (i.e. failure in one area and performance in another). Using a Dynp, either of these area protection levels can be modified, and reliability simulated stochastically. In this way, the approximate performance of area protections can be simulated on two dimensions:

- spatially: by assigning assets different area_prot_lvl values on the 'aprot' tab; and
- as a function of flood magnitude: by assigning area_egrd (of each area_prot_lvl) to each Flood object on the 'flood' tab.

While this three-WSL-version approach provides a means to approximate the exposure considering the reliability of an area protection, it does not account for the mechanisms and complexities associated with the failure of different area protections (i.e. levee over-topping vs breaching) and their corresponding likelihoods. Therefore, a more robust approach is to incorporate failure mechanics directly into the hazard analysis.

4.6. Flood Objects

Flood objects facilitate the estimation of damages to calculate risk using the EAD metric. These Flood objects are parameterized on the 'flood' tab, where each row corresponds to a column in the flood table (see previous). The primary parameter supplied on this tab is the flood ARI. This ARI is used to scale the Flood object damages by the EAD algorithm. Further, this flood ARI is used by the grid power algorithm, to predict whether grid power will be active or not during a flood event of that magnitude. This threshold is controlled via the 'gpwr_ari' global parameter. For example, when gpwr_ari = 100, SOFDA simulates all flood events more extreme than 100 ARI as having grid power failure (grid power = OFF).

4.7. Timeline

For timeline schematization, see section 2.3.1.

4.8. Actions

In SOFDA, Action objects bundle Dynps to simulate a unified change to the model via the timeline. Actions can also reference Selectors to apply changes to a subset of model objects. To make a more complex model change, Action objects can be bundled or nested so that one Action can call many other child Actions — passing down the parent's object subset to the child Actions. Further, the object hierarchy is respected, so that an Action passed a group of House objects knows which Dfuncs to target with its Dynps. In this way, changes can be made to subsets of subsets of subsets, etc. Similarly, complexity can be added by calling specific Actions in sequence on the timeline. It is up to the user to decide how best to manage the temporal, spatial, and object complexity Action objects afford to best suite the model objectives.

Each Action accepts the following parameters, specified via the 'actions' tab:

Table 4.11: Action attribute parameters.

Name	**Code**	**description**
Name	name	Unique string identifying the Action. Used to reference this Action by a command on the timeline or by another Action.
Target object class name	pclass_n	Class name of objects on which this Action applies
Selector name	sel_n	Selector to apply to passed object set (should match the pclass_n) to generate the selected object set.
Child Action names list	act_n_l	List of other Action names to execute (in sequence) n the selected object set. — prior to executing the dynp_n_l.
Dynp names list	dynp_n_l	List of Dynp names to execute (in sequence) on the selected object set.

4.9. Selectors

gen | fdmg | dfunc | hse_geo | flood_tbls | floods | actions | selectors | dynp | timeline | outputs | obj_test

Selectors are flexible worker objects used to identify a subset of objects for manipulation by some other 'subscriber' object. Selectors can be used, or subscribed to, by Actions, Dynps, and Outputrs to refine which model objects these subscribers apply to. Similar to Dynps, Selectors can be activated periodically (via the 'upd_sim_lvl' parameter) or explicitly (via some subscriber). The attributes used to schematize selectors are provided in the following table:

Table 4.12: Selector main attribute parameters.

Name	Code	Description
Name	name	Unique string identifying the Selector. Used by Action, Dynp, and Outputr objects to subscribe to the Selector.
Target object class name	pclass_n	Class name of objects on which to make selection.
Simulation level for periodic updates	upd_sim_lvl	Sim_lvl on which to recalculate selection

The three main options provided in SOFDA to execute object selection are provided in the following table:

Table 4.13: Selector selection method options.

Name	Code	Description
Object selection by meta-data	metadf_bool_exe	Using the metadata stored on the object's parent (in a pandas dataframe), the user can provide code snippets in this cell to generate a boolean array (where each True result will be included in the selection).
Object selection by local Boolean	obj_bool_exe	Looping through each target object in the session, the code snippet provided in this cell should generate a boolean (where each True result will be included in the selection)
Special object selection	spcl_f_exe_str	See next section

4.9.1. Special Functions

Special function selectors are parametrized by specifying some custom script for execution as an object method in the 'spcl_f_exe_str' column (e.g. self.foobar()). This allows the user to define more complex selection than may otherwise be available in the standard selection methods. The available special functions are summarized in the following sections.

Ranked Choice

A common application for the ‘ranked_choice’ function (Table 4.14) is the downscaling of urban re-development from some user provided list. To include some stochasticity in how objects are selected from this list, a non-zero positive ‘model.bucket_size’ parameter can be specified. This tells the ranked_choice function to use the 'get_random_pick_from_bucket' function (Table 4.15) to select from the user provided list.

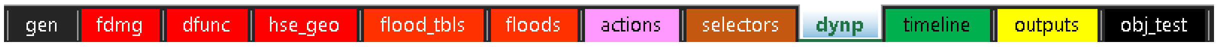
Table 4.14: Selector special function 'ranked_choice'.

Attribute	**Description**	**Default or typical values**
Name	List based selection	
Function code	**ranked_choice**	self.ranked_choice(n ='udev')
Function description	select objects based on some user provided list (ranked_l) and the model bucket size parameter (model.bucket_size).	
Inputs and Parameters		
n	number of entries to select from list	‘udev’: use value from ‘session.udev.infil_cnt’
update	whether to update the master list (remove recent picks)	True
Major dependencies		
self.ranked_l	list of object names for this selector object loaded via specifying some csv with the ‘headpath’ and ‘tailpath’ parameters.	
model.bucket_size	int for the bucket size to use for random bucket selection. Specified on the ‘gen’ tab.	0: no random bucket sampling. Just select the top ‘n’ objects from the list. >0: execute ‘self.get_random_pick_from_bucket’
Output/Result	Returns a dictionary of selected objects	return pick_d

Table 4.15: Selector special function 'get_random_pick_from_bucket'

Attribute	**Description**	**Default or typical values**
Name	Random bucket selection	
Function code	**get_random_pick_from_bucket**	self.get_random_pick_from_bucket(self.ranked_l, n + self.model.bucket_size, n)
Function description	generate a random sample from a bucket built from a passed list	
Parameters/Inputs		
full_l	ranked list of objects used to construct bucket from	
bucket_size	int for size of bucket to build from full_l	
pick_cnt	int for count to randomly select from bucket	
Major dependencies		
Output/Result	Returns a randomly selected list of length ‘pick_cnt’	return pick_l

4.10. Dynamic Parameters (Dynp)



Dynamic Parameter objects are session level workers that make some change to another object within the session. They can be triggered at irregular intervals on the simulation timeline via an Action, or at regular intervals (e.g. every timestep). Attribute changes can be: 1) stochastic (e.g. pulling a random sample from a Scipy distribution), formulaic (e.g. new value = DEM elevation + 0.6 m), or a simple value. Typical applications of Dynps may be; 1) assigning initial conditions at simulation startup; 2) setting a House object's current year to zero to simulate redevelopment; or 3) stochastically assigning the urban development rate for the timestep.

4.11. Outputs

Outputr objects facilitate the selection and outputting of any model object attribute in the simulation. This allows the user to optimize what metrics are reported by SOFDA. During the upkeep sequence, Outputrs scan through the session objects and collect their attributes of interest (i.e. results values). These results values are stored by each Outputr and held for the full simulation. At the end of the simulation, all Outputrs are cleared and readied for the next simulation.

5. SOFDA Model Modes

Section 2.1 describes the different run modes of SOFDA. For the default, stochastic mode, SOFDA program execution is described in section 2.3 the deterministic mode is similar.

5.1. Sensitivity Analysis (SA)

Model sensitivity analysis (SA) can provide valuable insight into model performance and variable interaction. Such insight can help inform model development and data collection, by identifying those parameters which are most significant in driving the model outcomes. Once known, a modeller can focus resources on refining these parameters — reducing the predictive uncertainty (and increasing the utility) of the model. In a stochastic model, like SOFDA, such parameter refinement may support the tightening of model input parameter distributions — reducing the spread of the results ensemble.

The SA is a set of special deterministic model runs on the study area that, instead of seeking a reasonable prediction for flood risk, seeks to quantify the importance of each parameter in making such a prediction. Generally, all parameters are kept at some median or 'best-guess' value — except for the focus parameter, which is toggled between extremes to explore its significance.

The SA mode in SOFDA is controlled with the 'sensi_f=TRUE' global parameter on the 'gen' tab of the Control File. In SA mode, SOFDA executes a model in these basic steps:

1. build the parameter matrix (Dynp parameters that will be applied to each simulation);
2. execute the baseline simulation (first row of parameter matrix; all default values);
3. execute all focus simulations; then
4. post process delta metrics.

In general, this approach does not consider connections between parameters, but instead explores deviations from some 'baseline' that represents an 'average' scenario. To evaluate non-linear connections between parameters (e.g. where both deviate from the baseline), the full stochastic mode of SOFDA should be employed. Further, the SA method of SOFDA assumes all Dynp parameters are independent.

5.1.1. Focus Parameters

For SA sessions, the user selects those Dynps on which to quantify sensitivity via the 'dynp' tab's 'sensi' columns (Table 5.1). Based on these user provided values, SOFDA builds a set of deterministic simulations from the user provided extremes — holding all other Dynps at their base value. These values are calculated during startup and stored in the parameter matrix (included as a tab in the output file).

Table 5.1: Sensitivity analysis control parameter options. Set via the 'dynp' tab columns 'sensi1', 'sensi2', 'sensi3'.

Option description	'sensi' input code	Description
Numerical extremes	*min/max	Generate two SA focus simulations on this parameter from the values provided in the min/max columns
Empty	(blank)	Apply this parameter deterministically and do not include any focus simulations
Custom value	(any other value)	Generate a SA focus simulation for this parameter using the value provided

The user is free to select any of the model Dynps for inclusion as a focus parameter in the SA, with any range of extremes. When selecting Dynps for inclusion, the following provides a useful framework by *application type*:

- *Model structure*: a replacement of some concise model sub-function with some functional alternative (e.g. swapping damage depth functions from 'seep' to 'damp');
- *Model parameter*: replacing a single parameter value with some alternative (e.g. swapping infill_cnt from 300 to 500);
- *Model parameter delta:* adding some delta to a model parameter (e.g. adding 0.5 m to all House.anchor_el). These are always 'zero' value when not the focal Dynp.

The results of any SA are dependent upon;

- *Model structure*: How the model is conceptualized and formalized determines the results generated from different input values.
- *Parameter extremes*: What values the user provides for the analysis determines the results generated by the deterministic model structure.

Considering this, the user should ensure the selected parameter extremes accurately represent the range of 'reasonable' parameter values of the system.

5.1.2. Sensitivity Metrics

To quantify model sensitivity, the user must schematize Outputrs to capture the desired model object attributes (section 4.11). Generally, flood risk — expressed as EAD — is the metric of most interest for a SOFDA SA. For a SA, these key metrics (calculated for a focus simulation) are compared against the baseline simulation to obtain a delta value (i.e. change in risk between focus and baseline). As SOFDA is dynamic, typically two metrics are leveraged to quantify sensitivity:

- **EAD baseline delta at the start (EAD_0)**: This is the risk in the first year (before redevelopment) compared against the baseline. This quantifies how much the focus parameter influences risk estimates. This metric is not influenced by vulnerability dynamics.
- **EAD baseline delta change (EAD_d):** This is the risk change (last year minus first year) compared against the baseline (delta of a delta). This quantifies how much this parameter influences the simulation of risk over time.

To output these two metrics in SOFDA, three Outputr objects are required:

name	desc	pclass_n	out_attn	post_exe	sim_stats_exe	dt_n
od1a	EAD year 1	Fdmg	aad_tot		raw	*first
od1b	EAD year last	Fdmg	aad_tot		raw	*last
od1c	EAD change (dt1 - dt0)	Outputr		outs_d['od1b'].data - outs_d['od1a'].data	raw	*last

These calculate EAD at the first and last timestep, before calculating the difference on a single simulation. To compare these to the baseline simulation's value, these Outputr names are referenced as a list in the global parameter 'delta_compare_col_nl' to calculate the final delta value.

References

Bryant, Seth. 2019. “Accumulating Flood Risk.” University of Alberta. https://era.library.ualberta.ca/items/1e033c0d-6c4c-4749-9195-e46ce9eb3e2b.

IBI Group, and Golder Associates. 2015. “Provincial Flood Damage Assessment Study.” Government of Alberta. http://www.alberta.ca/albertacode/images/pfdas-alberta-main.pdf.

Attachment B: Sample Control File

Attachment C: Damage Feature Table

Attachment D: Alberta Curves