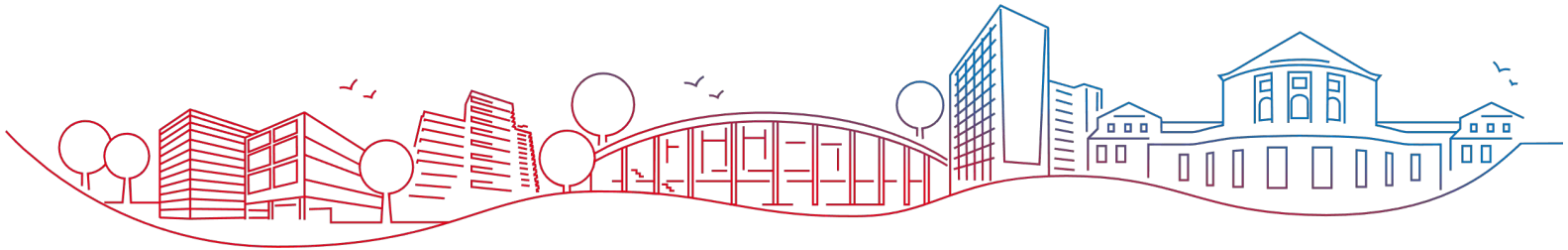
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG



HS^3 - Features, Limitations and the Future

Carsten Burgard

HS^3: The state of the union

- HS^3 has matured significantly over the last year
  - first citable publication available
  - first independent implementation with pyhs3 fully validated
  - usable as a backend in high-level tools already (spey)
  - several ATLAS analyses published
  - CMS combine in the process of implementing support
- Good time to reflect:
  - What is still missing / not working?
  - What are the needs by the DEMOS community that are not sufficiently addressed yet?
  - Where does HS^3 stand in comparison with follow-up/derivative/extension projects (XS3, FlatPPL)?

Part 1

Open design questions

The forward model paradigm

- HS^3 describes forward models
  - knowing the causes (parameter values), describe the effects (distributions)
  - backward / inverse modeling requires inference: algorithms acting on the model
  - the algorithms are not part of HS^3, they are part of the engines implementing HS^3
- A “model” in HS^3 is a computational graph describing which mathematical operations lead from the known parameter values to the (expected) distributions
  - is static on disk - no statement on how to represent it in memory / in your engine
  - can be used to draw (toy) data from the distributions (→ engine)
  - can be combined with “data” (observed or simulated) to form a likelihood
  - likelihoods can be used for inference of best-fit-values (→ engine)

Things that HS3 does not do, but might be interesting for DEMOS:

- prescriptions of how models are to be used (operations on models)
- unified runtime API for evaluating models
- encode results of inference
- … ?

Combined likelihoods & distributions

- HS^3 already supports combined likelihoods
  - data and distribution vectors are mapped by index
  - aux_distributions act standalone without data
- Combined likelihoods are good enough for pure inference
  - find minimum, profile, project, scan…
  - not useful for toy sampling from combined models
- Long-term, need
  - combined distributions
  - combined datasets
- For this, need first-class notion of categorical indices (integer variables)
  - iterator index over regions of model/dataset (“observable-type”)
  - for CMS combine, also need integer parameters (“parameter-type”)
- Steffen currently works on a proposal how this could be realized for HS^3

```
"likelihoods": [
  {
    "aux_distributions": [
       "aux_dist1",
       …
    ],
    "data": [
       "data1",
       …
    ],
    "distributions": [
       "measurement_dist1",
       …
    ],
    "name": "likelihood"
  }
],
```

Why do distributions not declare observables?

- Consider a simple gaussian response model: *R(reco,truth)=N(reco;truth,σ)*
- The detector response itself does not know which quantity is
  - observable,
  - parameter of interest,
  - nuisance parameter.
- Those roles belong to the analysis, not to the model component
- Some Statistical Frameworks will need 3 different models for different tasks
  - but they are all identical!
- RooFit and HS^3 both respect this identity
  - can we retain this feature?

follow link for example

Analysis task	Observed	Inferred
Simulation	truth	reco
Calibration	reco	truth
Unfolding	reco	truth distribution
Response studies	reco, truth	none

Hierarchical Priors in HS^3

- Can HS^3 represent hierarchical priors?
- HS^3 does not yet have mature support for Bayesian prior objects
- … but it can represent hierarchical structures!

Example: Calorimeter Calibration

- global scale G
- cell-to-cell spread τ
- individual cell scales g_i
- calibration measurements x_i

Model	HS^3 representation
$\tau \sim N(0.03, 0.01)$	measurement distributions
$g_i \sim N(G, \tau)$	auxiliary distributions
$x_i \sim N(g_i, \sigma_{meas})$	aux. dist. on hyperparameter

Interpretation as "prior" vs "auxiliary measurement" remains a semantic question, but Hierarchical Models are Already Representable in HS^3

follow link for example

Complex numbers & vectors

- HS^3 currently assumes that all variables are real numbers
  - no support for complex numbers
  - no support for vector-valued variables
- Based on past discussions, this seems to be an absolute necessity
  - suggested solutions included introducing “slots” to functions / distributions
  - syntax remains to be decided, but would require significant overhaul of parser/emitter
- Open questions
  - do we want to allow decomposing composite objects at all?
  - perhaps, we do not need a syntax for addressing components when we prescribe that only the entire object is allowed to be processed (SIMD-like)

- *I have generally tried to follow the philosophy that in order to come up with a useful design, you first need the use case*
- *Can we agree to create a suite of simple usage examples (any language/syntax) to act as “challenges”?*

Syntax suggestions:

- f.xreal, f.ximag, f.y1, f.y2
- { source: “f”, slot: “xreal” }
- …

Metadata

```
"metadata": {
     "hs3_version": "0.2",
     "packages": [
         {
             "name": "ROOT",
             "version": "6.41.01"
         }
     ]
},
```

- Currently, HS^3 only supports very limited metadata
  - hs3_version
  - packages (basically, “source”)
- I think we need much more
  - model provenance?
  - publication links / DOIs?
- Again, hard to decide without a use case
  - have initiated collaboration with hepdata.org development team to improve HS^3 integration on the site (recent talk by Steffen)
  - hope that this will result in use cases to inspire metadata
  - will have to follow up to do the same for zenodo

Part 2

Relations to evolving DEMOS projects

HS^3 and FlatPPL

- From what I can see, FlatPPL is very likely going to be feature-equivalent to HS^3
  - Are we aware of any features of FlatPPL that HS^3 does not / can not have?
- Even with full equivalence, still very interesting
  - FlatPPL has more intuitive syntax (more “programming language like”)
- Do we have / can we have a converter?
  - this would help me understand FlatPPL better
  - would immediately provide a full-blown interpreter for FlatPPL via $HS^3 \rightarrow$ROOT
  - bijective converter would be ideal to study equivalence!

HS^3 and XS3

- XS3 seems to be a generalization / wrapper framework for HS^3
  - would it be fair to say that HS^3 is one realization of XS3?
- Structurally, it seems that XS3 has more “constraints” than HS^3
  - can anything that we can represent in HS^3 be represented in XS3?
  - do we have any complete models in XS3 yet?
  - how different are they from the same model represented in HS^3?

Test Suite

- From the UHH/TUDO team, we have started to work on a test suite
  - follow the link to take a look
  - for now, test suite constructed from RooFit examples
  - in the future, plan to extend with more advanced examples also
    - this includes the gists linked in this presentation
  - please provide more use cases / examples for the suite!
    - in the future, might also include models published on hepdata
- Goals of the suite
  - provide unit tests for any interpreter implementing HS^3: ROOT, pyhs3, …
  - provide use cases for any HS^3 extension / evolution
  - also act as unit tests for any converter

HS3TestSuite in ROOT

courtesy of Steffen & Simon

adding to ROOTs ctests (already part of CI on github)

```
if(pyroot AND hs3testsuite)
    set(hs3testsuite_dir ${CMAKE_CURRENT_SOURCE_DIR}/HS3TestSuite)
    if(NOT EXISTS ${hs3testsuite_dir})
        find_package(Git QUIET REQUIRED)
        execute_process(COMMAND ${GIT_EXECUTABLE} clone https://github.com/Phmonski/HS3TestSuite
        WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR})
    endif()

    if(EXISTS ${hs3testsuite_dir})
        ROOT_ADD_PYUNITTEST(hs3-suite test_hs3_suite.py PYTHON_DEPS jsonschema ENVIRONMENT HS3TESTSUITE_ROOT=${hs3testsuite_dir})
    endif()
endif()
```

wrapping in unittest TestCase

```
import os
from pathlib import Path
import sys
import unittest


hs3_root_dir = os.environ.get("HS3TESTSUITE_ROOT")
if hs3_root_dir:
    sys.path.insert(0, hs3_root_dir)

try:
    from hs3suite.runner import run_suite

except ImportError as e:
    TestHS3Suite = type(
        "TestHS3Suite",
        (unittest.TestCase,),
        dict(test_dependencies=lambda self, e=e: self.fail(f"Missing dependency: {e}")),
    )
else:
    hs3_root_path = Path(hs3_root_dir)
    try:
        results = run_suite(hs3_root_path, "roofit")
    except Exception as e:

        def test(self, e=e):
            self.fail(f"HS3TestSuite failed to run: {e}")

        test.__name__ = "test_hs3testsuite_execution"
        TestHS3Suite = type("TestHS3Suite", (unittest.TestCase,), {test.__name__: test})
    else:
        namespace = dict(__module__=__name__)
        for r in results:

            def _make(result):
                def test(self):
                    if result.status == "failed":
                        self.fail(result.message)
                    return

                test.__name__ = f"test_{result.test_id}__{result.check_id}"
                test.__doc__ = f"{result.test_id}::{result.check_id}"
                return test

            func = _make(r)
            namespace[func.__name__] = func
        TestHS3Suite = type("TestHS3Suite", (unittest.TestCase,), namespace)

if __name__ == "__main__":
    unittest.main()
```

PR to ROOT already opened [PR 22567]

- in the future, move backend to engine (ROOT) so test suite can be agnostic
- pyhs3 backend currently WIP

Conclusions

- The ecosystem is currently in flux
  - good thing, hope to learn from the past and craft something better / more comprehensive for the future
- HS3 provides a solid baseline
  - anything we develop needs to be able to cover at least the feature set already provided by HS3 (reflected by the test suite)
  - plus all the needs we have identified (and will identify) in DEMOS
- Curious to see updates from FlatPPL & XS3
  - How far have the definitions converged?
  - How likely is it that we can have a full-featured bidirectional converter soon?