GAMBIT-light
1. What is GAMBIT-light?
2. Why do we need it?
3. For users: how does it work?
4. For GAMBIT developers: how do we maintain it?
5. Future plans
1. What is GAMBIT-light?
- **GAMBIT-light**: **GAMBIT without all the physics**
- A lightweight yet powerful tool for statistical fits and optimisation tasks
- **What GAMBIT-light is not**: A full-blown tool for global fits in <your discipline> — for that you’d want more of the full GAMBIT functionality
- **Key design principles:**
  - Users should never need to modify and rebuild any GAMBIT code
  - Minimise the extra maintenance work for GAMBIT developers
Models <-> Core <-> ScannerBit

Physics modules:
- SpecBit
- ColliderBit
- NeutrinoBit
- DecayBit
- DarkBit
- CosmoBit
- PrecisionBit
- FlavBit

Backends:
- CapnGeneral
- DarkSUSY
- DDCalc
- FeynHiggs
- FlexibleSUSY
- gamLike
- gm2calc
- HEPLike
- HiggsBounds
- HiggsSignals
- MicrOmegas
- nlike
- Pythia
- SPheno
- SUSYHD
- SUSYHIT
- Superso
- Vevacious
- MontePython
- CLASS
- AlterBBN

Scanners:
- Diver
- GreAT
- MultiNest
- PolyChord
- TWalk
- grid
- random
- postprocessor

Anders Kvellestad
Models ↔ Core ↔ ScannerBit

LightBit

Minimal C, C++, Fortran, Python interface libraries

Scanners

Diver, GreAT, MultiNest, PolyChord, TWalk, grid, random, postprocessor, …

Your target function code here (C, C++, Python, Fortran)
2. Why do we need it?
• **Background:**
  ○ We designed GAMBIT to be very general and physics-agnostic
  ○ We put a lot of effort into the main code framework
    (Core, ScannerBit, Printers, CMake system, …)
  ○ → GAMBIT can be used for optimisation/fits outside particle/astro/cosmo

• **Practical experience:**
  ○ GAMBIT is a particle physics power tool → fairly heavyweight
  ○ Considerable threshold for non-experts to pick up and use/modify
  ○ In particular: frequent and slow recompilation kills the flow of the early development/experimentation stage of projects
● External motivation for GAMBIT-light:
  ○ Help projects outside particle/astro/cosmo use GAMBIT
  ○ In particle/astro/cosmo: suitable for quick experimentation, MSc projects, etc.

● Internal motivation for GAMBIT-light
  ○ Increase visibility and impact of Core & ScannerBit work
  ○ Increase visibility for Core & ScannerBit papers
    (GAMBIT-light should not have a separate code paper – users should cite the main GAMBIT & ScannerBit papers)
  ○ Sandbox for quick experimentation
3. For users: how does it work?
1. Build GAMBIT once

    mkdir build
    cd build
    cmake -DCMAKE_BUILD_TYPE=Release -DWITH_MPI=On -DCMAKE_CXX_COMPILER=g++-11 -DCMAKE_C_COMPILER=gcc-11
    make -jN scanners  # where N is the number of cores to use for the build, e.g. 4
    cmake ..  # this step is needed for GAMBIT to detect the built scanners
    make -jN gambit
2. Develop your target/likelihood function code

```python
# To import gambit_light_interface, first append the directory containing
# gambit_light_interface.so to sys.path. (Alternatively, add this directory
# to the PYTHONPATH environment variable.)
import sys
import os
current_dir = os.path.dirname(os.path.abspath(__file__))
sys.path.append(os.path.join(current_dir, "../lib"))
import gambit_light_interface as gambit_light

# User-side log-likelihood function, which can be called by GAMBIT-light

def user_loglike(input_names, input_vals, output):
    # Make a dictionary of the inputs?
    input = {input_names[i]: input_vals[i] for i in range(len(input_names))}

    # Error handling: Report an invalid point using gambit_light.invalid_point.
    # gambit_light.invalid_point("This input point is no good.")

    # Error handling: Report a warning using gambit_light.warning.
    gambit_light.warning("Some warning.")

    # Error handling: Report an error using gambit_light.error.
    # gambit_light.error("Some error.")

    # Error handling: Error handling, alternative to using gambit_light.error: Throw an exception.
    # raise Exception("Some exception.")

    # Compute loglike
    loglike = input["param_name_1"] + input["param_name_2"] + input["param_name_4"]

    # Save some extra outputs
    output["py_user_loglike_output_1"] = 1
    output["py_user_loglike_output_2"] = 2
    output["py_user_loglike_output_3"] = 3

    return loglike
```
2. Develop your target/likelihood function code
(C++/C/Fortran: build as shared library)
3. Configure GAMBIT run with a YAML file

```yaml
UserModel:
  p1:
    name: param_name_1
    prior_type: flat
    range: [0.0, 5.0]
  p2:
    name: param_name_2
    prior_type: flat
    range: [0.0, 5.0]
  p3:
    name: param_name_3
    fixed_value: 3.0
  p4:
    name: param_name_4
    same_as: UserModel::p1
  p5-p7:
    name: param_name_
    prior_type: flat
    range: [-1.0, 1.0]

UserLogLikes:
  py_user_loglike:
    lang: python
    user_lib: gambit_light_interface/example_python/example.py
    Func_name: user_loglike
    output:
    - py_user_loglike_output_1
    - py_user_loglike_output_2
    - py_user_loglike_output_3
  cpp_user_loglike:
    lang: c++
    user_lib: gambit_light_interface/example_cpp/example.so
    Func_name: user_loglike
    input:
    - param_name_2
    - param_name_3
    output:
    - cpp_user_loglike_output_1
    - cpp_user_loglike_output_2
    - cpp_user_loglike_output_3
```
4. Run GAMBIT

```
mpiexec -np 4 ./gambit -f yaml_files/your_configuration_file.yaml
```
5. Modify your own code, rerun GAMBIT, modify your own code, rerun GAMBIT, …
6. Analyse output samples (saved in HDF5 or ascii format)
Also: user-supplied prior transformation

Python

```python
# User-side prior transform function, which can be called by GAMBIT-light.
def user_prior(input_names, input_vals, output):
    for i, v in enumerate(input_vals):
        output[i] = v * 10.
```

C++

```cpp
// User-side prior transform function, which can be called by GAMBIT-light.
void user_prior(const std::vector<std::string>& input_names,
                const std::vector<double>& input_vals,
                std::vector<double>& output)
{
    for (size_t i = 0; i < input_vals.size(); i++)
    {
        output[i] = input_vals[i] * 10.;
    }

    GAMBIT_LIGHT_REGISTER_PRIOR(user_prior)
```
Also: user-supplied prior transformation
4. For GAMBIT developers: how do we maintain it?
- Will describe the current setup – we can change this as needed
- [github.com/GambitBSM/gambit_light](http://github.com/GambitBSM/gambit_light) is a fork from our main repo [github.com/GambitBSM/gambit](http://github.com/GambitBSM/gambit):
- Will describe the current setup – we can change this as needed
- [github.com/GambitBSM/gambit_light](github.com/GambitBSM/gambit_light) is a fork from our main repo [github.com/GambitBSM/gambit](github.com/GambitBSM/gambit)
- Only some updates on gambit are relevant for gambit_light (Core, ScannerBit, Printers, …)
- The branch gambit:gambit_light_sync contains
  - a list of the gambit files that should be identical on gambit_light
  - a GitHub Actions workflow that autogenerates pull requests on gambit_light whenever some of these files are updated on gambit
- Workflow is currently set to sync gambit:gambit_light_sync → gambit_light:gambit_light_sync
- **Example:**
  - On gambit: a merge master → gambit_light_sync will generate a pull request on gambit_light with relevant updates for gambit_light:gambit_light_sync
  - On gambit_light: when PR is checked and merged, we can merge gambit_light_sync → master
- Using the branches gambit_light_sync is just to make syncing a bit more manual for now – in the future we can directly sync gambit:master → gambit_light:master
• All files are either **fully synced** or **not synced at all** – no partial syncing

• Most files on gambit don’t exist on gambit_light

• Many files on gambit_light (e.g. CMakeLists.txt) are completely detached from the corresponding file on gambit

• All such non-synced gambit_light files can be modified directly on the gambit_light repo

• For synced files where we need small modifications between gambit and gambit_light, we can implement the changes on gambit with

  ```
  #ifdef GAMBIT_LIGHT
  ...
  #endif
  ```

• From the GitHub perspective, the file is fully synced between gambit and gambit_light. (But it will generate different behaviour when compiled on gambit_light.)
This is an experiment – we’ll have to tweak things to find the best system for easy development + minimal duplicated maintenance

Revisit and evaluate at the next face-to-face GAMBIT meetings
5. Future plans
● Code development
  ○ Keep testing the gambit → gambit_light syncing
    ■ First big test: the PR for Python scanners
  ○ Polish the gambit_light examples and documentation
  ○ PR for initial code updates on gambit
  ○ Document gambit_light in the “GAMBIT 2” paper
  ○ Make first public release
  ○ Next: absorb GAMBIT bugfixes and improvements as they arrive
    (fast-slow, continual learning, …)

● Some ongoing projects that will use GAMBIT-light:
  ○ With the nuclear physics group in Oslo: unfolding of gamma spectra
    (Andreas Mjøs, Erlend Lima, Lasse Braseth, Ann-Cecilie Larsen, Morten Hjorth-Jensen + me)
  ○ With the Norwegian Institute of Public Health: optimisation of Monte Carlo simulations of disease spread
    (Ida-Marie Johanson, Jørgen Midtbø, Francesco Di Ruscio, Yat Hin Chan, Birgitte Freiesleben de Blasio + Are and me)