Meeting notes

Next meeting: TBD

Topics for next meeting

To do

- [ ] Check what is the minimal amount of storage required to recreate the GPs (Jeriek)
- [ ] Optimise the GP prediction function in scikit-learn (Jeriek)
- [x] Create working example of parallelisable DGP code (Ingrid)
- [ ] Try connecting GAMBIT to the DGP example (Anders)
- [ ] Look into including PDF uncertainties (Are)
- [ ] ...

13-14.06.2018

1. Overall program structure

- The code is split over two separate repositories: one for the training part (jonvegards/prospino-PointSampler), one for the evaluation part (jeriek/xsec). We need some user interface around Ingrid's code. Need to think what possibilities we want to offer users.
- Future *integration with GAMBIT* is central, as our tool will function as an external library. GAMBIT has an internal C++ representation of SLHA files, as reading/writing many of these is surprisingly costly. ColliderBit has an internal way of passing parameters from the SLHA, which we will have to take this as input. But our tool should also work as a *stand-alone application*, with SLHA files passed by the user and for example processed with PySLHA.
- GAMBIT has a functioning Python interface and its thread-safety should be fine, as fine as possible with Python (and as long as no multiple instances of Python are run on the same kernel). In the interest of reducing development time and easy integration with Ingrid's code and scikit-learn, we will try to write the tool in Python.
- Requiring external libraries should be avoided, but for now, there seems no way around scikit-learn. This should be fine. Joblib automatically follows with scikit-learn, which requires NumPy and SciPy by default.
- The program will be heavy on memory, but if the initial requirements are kept below ~1 GB by reducing file sizes, it should be acceptable.

2. Optimising the GP code

- Ingrid's code has now been extended to all strong production processes.

- Storage of large matrices: use Joblib Persistence and Memory class. Also, what type is stored in the matrices? Currently `float64`. We don't need too high precision, so maybe rounding and saving smaller floats would help. Note we're working on 64-bit architecture. SciPy supports more numerical types than standard Python. Their numerical value limits depend on the machine:

Type	Resolution	Min/max
`float16`	0.001	+/- 6.55040e+04
`float32`	1e-06	+/- 3.4028235e+38
`float64`	1e-15	+/- 1.797931348623157e+308

  Question: why do we want the stored files to be small? Is it just a (longer-term) storage problem or should we prevent scikit-learn from creating large `float64` matrices in the first place? Answer: for now, just limiting the (longer-term) storage space should be enough, so scikit-learn can internally create the `float64` arrays as long as we can round, compress and store them in smaller files.
- Evaluation speed: investigate possibilities to optimise scikit-learn's GP prediction function, making use of the persistence functions provided by Joblib. In particular, `L_inv` and `K_inv` shouldn't be recomputed for every new prediction point. Try to use `GaussianProcessRegressor` as base class and override the `predict` method. Object persistence between points (particularly for the evaluation function) should be alright, as the Python interpreter doesn't get restarted between points in GAMBIT scans.
- For Abel runs, ensure the Intel-compiled math libraries LAPACK and BLAS are picked up rather than any that might be in the user directories. See Abel FAQ and Intel compiler details.

3. Parallellisation and interfacing with GAMBIT

- Ingrid will send Anders some working DGP example, to try hooking this up to GAMBIT.

30.04.2018 (first meeting!)

1. Project name

Currently XSEC (or TUXEDO, The Universal X-section EstimaDOr, code name SMOKING), no final decision was made about the project name. A mythological reference would be nice (but no, Tanngnjóstr 🐐 is too hard). Preferably a name that tab-completes easily in a terminal.

Some other acronym ideas:

- FACET(S): Fast and Accurate Cross-section Estimation Tool (for SUSY)
- ACES: Accelerated Cross-section Estimator for SUSY
- ITHACA: ITHACA Tool for Handling Accelerated Cross-section Approximation (it's a recursive acronym and of course the final destination in Homer's *Odyssey*)
  - Alternatively: Integrated Tool for Handling Accelerated Cross-section Approximation
- TAXES: Tool Accelerating X-section Estimation for SUSY (but no one would like us)

- PEXA: Program for Estimating X-sections Accurately
- <feel free to add!>

2. Project objective

Fast cross-section estimation at higher orders. *Fast* means: $\mathcal{O}(1\ \mathrm{s})$ evaluation time for 1 parameter point. More than 10 s would give trouble. Most points are thrown anyway.

Example: one point, with 10 experts and 36 processes, takes about 16 seconds right now.

Important issues include:

- Large amount of training data required (computationally expensive)
- Training time
- Generated file sizes (should be kept a lot smaller than a gigabyte)
- Obtained accuracy of the estimates
- Knowledge of the uncertainties
- Including all processes required (for strong LHC production): $\tilde{g}\tilde{g}, \tilde{g}\tilde{q}, \tilde{q}^*\tilde{q}, \tilde{q}\tilde{q}$

3. Current status

Boosted Decision Trees (Jon Vegard)

- Fast: due to the tree structure, not a lot of code is effectively executed
- Lots of training data required
- Easier to trim file sizes
- Tested on $\tilde{g}\tilde{g}$ production

(Distributed) Gaussian Processes (Ingrid)

- Slow: evaluation time scales as N^2
- Not a lot of training data required
- Large files, harder to trim
- Tested on $\tilde{q}\tilde{q}$ production (expectation: $\tilde{g}\tilde{g}$ more involved for DGPs since more parameters)

Latest developments

- Set-up for semi-automated MSSM-24 sample generation with Prospino 2.1 (Jon Vegard)
  - On Abel: `/work/projects/nn9284k/xsec`
  - On GitHub: jonvegards/prospino-PointSampler
    - Point sampler calculates NLO cross-sections for strong and weak processes in MSSM-24 at four COM energies with scale variation

4. Future work

- Including PDF uncertainties
  - Will require modifying Prospino with `COMMON` blocks
  - Switch to NNPDF
- Extend Gaussian Processes to all processes
  - Note: Electroweak production processes would take very long and require 4 times as many training points. Currently, we have 100 000 points for strong production

- Writing the actual program
    - C++ or Python?
        - Right now, bottle necks are the file size and evaluation time
        - It's good that the evaluation across experts (even across points) is parallelisable
        - In any case, it should be able to interface to GAMBIT. Is GAMBIT thread-safe?
        - Probably best to have an unparallelised version first, going as fast as possible on one single thread, in order to avoid interfering with GAMBIT's two-level parallelisation