RooFit redesign

Conclusions from 27-28 June 2019 meeting @ CERN

Three parts

- Rewriting likelihood calculation in "sane" statistics/physics based structure/concepts
- Task management & parallel calculation
- Minimization

Rewrite likelihoods

- RooUnbinnedLikelihood
- RooBinnedLikelihood
- RooMultipleLikelihood
- RooConstraintLikelihood
- Common interface: evaluate_partition with components, partitions and begin + end
- everything with stride = 1 now, no more interleave (bad for vectorization & general performance)
- These are not RooAbsArgs, but there will be a wrapper class (return value of createNLL) which will use the Likelihoods and will again be a RAA ("calculators != RooAbsArg/Real")
  - Conceptual replacement of current RooAbsTestStatistic
  - But also different; user facing replacement, because parallelization will now be done elsewhere
- Optimization (all the stuff that's now in RooAbsOptTestStatistic):
  - Common base class for e.g. binned and unbinned (similar optimization strategy)
  - Separate RooPDFOptimizer class

Parallelization & Task management

- Composition instead of inheritance
- Make possible to use different "back-ends" next to the default RooFit back-end, e.g. TensorFlow, analytical derivations, etc.
- JobLikelihood, JobDerivative, Job2ndDerivative
  - These should somehow be fed to the minimizer
  - Possibilities for implementation:
    - RooMinimizer<JobL, JobD, Job2D>
    - ABClass for all three types that can be implemented per back-end

Parallelization & Task management

- Current situation:
  - Singleton TaskManager
  - Jobs defined through inheritance from Job class
    - Parallel -> Vector<Single> -> {Single, Job}
- Preferred situation:
  - Singleton TaskManager (same)
  - Jobs have RAR* likelihood (wrapper) object as member
  - Minimizer cues calculation of something, e.g. likelihood or gradient
  - Job then takes care of parallelization, i.e. talkes to TaskManager
  - Only generic part of Jobs is the partitions/components division, everything else is back-end specific

Parallelization & Task management

- Nomenclature of Jobs vs Tasks is confusing, can we think of better names?
  - three (/ four) concepts:
    - "Job": the concept of a parallelizable computation, like a Likelihood or a Gradient
    - [unnamed]: a single full execution of that computation
    - "task": individual computational unit of the Job; one part of the Job that can be computed in parallel from the other parts
    - "batch": used for vectorization, overlaps currently with tasks
  - How do others call these things?
    - TBB
      - Tasks: corresponds to our tasks
      - Blocks: range of tasks
      - Iteration space: full range of tasks
      - Not 100% compatible, just different units of the same basic quantity, no semantic link to the thing you're trying to parallelize
    - https://en.wikipedia.org/wiki/Job_(computing)
      - *In computing, a **job** is a unit of work or unit of execution (that performs said work). A component of a job (as a unit of work) is called a task or a step (if sequential, as in a job stream).*
      - **Compatible with our current definition**
  - More (creative) options instead of Job:
    - http://www.namibian.org/travel/misc/collective-nouns.html collective animal names
    - Problem decomposition https://en.wikipedia.org/wiki/Parallel_programming_model#Problem_decomposition
    - "Workflow"
    - Something from graph theory maybe… Tasks could perhaps be Leafs, which would also alleviate the Job/Task degeneracy

Parallelization & Task management

- TaskManager needs to know all tasks of all jobs at start-up time (fork)
- How do we handle tasks that themselves can be split up as a Job into new tasks?
- If the constellation of Jobs is known before fork, it's possible in current framework
- Need some form of dependency management
- We can implement switching between different sets of tasks (e.g. only partial derivatives or also split Gradient by likelihood components or...) as Job "strategies"
- This can then be handled internally in the Jobs
- The only non-Job infra that has to be added is a message to change strategies

Minimization

- For Jobs to implement effective strategies based on minimizer phase, they must be able to somehow detect "phase"
  - E.g. in line search phase, or in gradient calculation phase, or doing Hesse or using Simplex method, etc.
  - Must Minimizer interface be modified to expose this information?
    - As far as we can see: no.
    - The information is already signaled by the call the minimizer makes to the back-end
    - For Minuit, 3 possible calls:
      - FCN.evaluate()
      - GRAD.Gradient()
      - GRAD.G2ndDerivative() **[Is it actually used? If not, can we do this? Does this only calculate the symmetric components (seems like it, it only has one "icoord" parameter)? @Lorenzo]**