ACTS Vertexing Tutorial - Example: Adaptive Multi-Vertex Finder (AMVF)

This tutorial sets up and runs the ACTS Adaptive Multi-Vertex Finder on smeared ACTS Pythia8 truth tracks.
Note: You have to have Pythia8 available on your machine for following this tutorial.

Setting up ACTS

If you already have an ACTS clone, skip Step 1 and jump directly to Step 2.
Step 1:
First, create a tutorial directory:

```
mkdir actsVertexingTutorial; cd actsVertexingTutorial
```

Clone ACTS and checkout the vertexing tutorial branch:

```
git clone https://github.com/baschlag/acts.git
cd acts
git checkout vertex_tutorial
```

If you have followed Step 1, skip Step 2 and move on with Step 3.

Step 2:
Go to your ACTS clone directory, fetch and checkout the vertexing tutorial branch:

```
git remote add vertexing https://github.com/baschlag/acts.git
git fetch vertexing; git checkout vertex_tutorial
```

Step 3:
On e.g. lxplus you might want to set up an LCG release (for e.g. making Pythia8 available):

```
source CI/setup_lcg95.sh # or setup_lcg96.sh
```

Let's try and build ACTS *without* the vertexing example algorithm for now:

```
mkdir build; cd build
cmake -DACTS_BUILD_EXAMPLES=ON -DACTS_BUILD_EXAMPLES_PYTHIA8=ON ..
make -j4
```

Note: On your local machine you might want to specify your Pythia8 location by adding

```
-DCMAKE_INSTALL_PREFIX=<path_to_your_pythia8>
```

to your cmake command above.

Note: By adding

```
-DCMAKE_BUILD_TYPE=Release
```

to your cmake above, you will significantly speed up the execution time of the vertexing algorithm below, it will however slow down the compilation process.

If the previous step worked fine (or in the meantime), we can now start to set up the ACTS Adaptive Multi-Vertex Finder in an example algorithm.

Setting up an ACTS Adaptive Multi-Vertex Finder Algorithm

A template algorithm file with an (almost) empty `execute()` method to be filled in the following is provided here:

```
../Examples/Algorithms/Vertexing/src/ExampleAMVFAlgorithm.cpp
```

Open the file in your editor and let's start setting up the AMVF. We will start setting up all necessary components in the `execute()` method.
Note: You would normally **not** want to do all the following setup steps in the `execute()` method (that is run on every single event), but rather in some kind of initialization method that is executed only once (which is however not available in the lightweight example framework we will be using here).

Setting up required tools: Magnetic field and propagator

Let's start with setting up a constant magnetic field:

```
// Set up the magnetic field
Acts::ConstantBField bField(Acts::Vector3D(0., 0., 2_T));
```

We need the `Acts::Propagator` with the `Acts::EigenStepper`:

```
// Set up EigenStepper
Acts::EigenStepper<Acts::ConstantBField> stepper(bField);
// Set up the propagator
using Propagator = Acts::Propagator<Acts::EigenStepper<Acts::ConstantBField>>;
auto propagator = std::make_shared<Propagator>(stepper);
```

Setting up required tools for the vertex fitter

Now, set up an impact point estimator...

```
// Set up ImpactPointEstimator
using IPEstimator = Acts::ImpactPointEstimator<Acts::BoundParameters,
Propagator>;
IPEstimator::Config ipEstimatorCfg(bField, propagator);
IPEstimator ipEstimator(ipEstimatorCfg);
```

... and track linearizer for helical track parameters:

```
// Set up the helical track linearizer
using Linearizer = Acts::HelicalTrackLinearizer<Propagator>;
Linearizer::Config ltConfig(bField, propagator);
Linearizer linearizer(ltConfig);
```

Now, for the sake of this example, let's specify a user-defined annealing scheme for the AVMF:

```
// Set up deterministic annealing with user-defined temperatures
std::vector<double> temperatures{8.0, 4.0, 2.0, 1.4142136, 1.2247449, 1.0};
Acts::AnnealingUtility::Config annealingConfig(temperatures);
Acts::AnnealingUtility annealingUtility(annealingConfig);
```

The AMVF strongly interplays with its dedicated vertex fitter, the *Adaptive Multi-Vertex Fitter*. Let's configure and set it up with the annealing utility defined above:

```
// Set up the vertex fitter with user-defined annealing
using Fitter = Acts::AdaptiveMultiVertexFitter<Acts::BoundParameters,
Linearizer>;
Fitter::Config fitterCfg(ipEstimator);
fitterCfg.annealingTool = annealingUtility;
Fitter fitter(fitterCfg);
```

Setting up required tools: Vertex seed finder

The last tool we need to set up (before finally setting up the AMVF) is a vertex seed finder:

```
// Set up the vertex seed finder
using SeedFinder = Acts::TrackDensityVertexFinder<Fitter,
Acts::GaussianTrackDensity>;
SeedFinder seedFinder;
```

Setting up the AMVF tool

Now we are ready to set up the Adaptive Multi-Vertex Finder. ACTS vertex finders are templated on the vertex fitter and vertex seed finder type:

```
// The vertex finder type
using Finder = Acts::AdaptiveMultiVertexFinder<Fitter, SeedFinder>;
```

We configure the vertex finder in such a way that we do *not* use a beam spot constraint here:

```
Finder::Config finderConfig(std::move(fitter), seedFinder, ipEstimator,
linearizer);
// We do not want to use a beamspot constraint here
finderConfig.useBeamSpotConstraint = false;
```

Create the AMVF instance and a finder state to be passed to the `find()` method below:

```
// Instantiate the finder
Finder finder(finderConfig);
// The vertex finder state
Finder::State state;
```

Lastly, we need to provide vertexing options. Here, we could e.g. set a beam spot constraint to the vertexing.

```
// Default vertexing options, this is where e.g. a constraint could be set
using VertexingOptions = Acts::VertexingOptions<Acts::BoundParameters>;
VertexingOptions finderOpts(ctx.geoContext, ctx.magFieldContext);
```

Deploying the vertex finder on the track collection

Now we're ready to actually use the AMVF tool that we have set up above to find vertices on our input track collection. The `find()` methods on ACTS vertex finders return an `Acts::Result` object that we can use to check if any errors occured and to retrieve the vertex collection:

```cpp
// Find vertices
auto res = finder.find(inputTrackPtrCollection, finderOpts, state);

if (res.ok()) {
  // Retrieve vertices found by vertex finder
  auto vertexCollection = *res;
  ACTS_INFO("Found " << vertexCollection.size() << " vertices in event.");

  unsigned int count = 0;
  for (const auto& vtx : vertexCollection) {
    ACTS_INFO("\t" << ++count << ". vertex at "
                   << "(" << vtx.position().x() << "," << vtx.position().y()
                   << "," << vtx.position().z() << ") with "
                   << vtx.tracks().size() << " tracks.");
  }
} else {
  ACTS_ERROR("Error in vertex finder: " << res.error().message());
}
```

For reference, the full tutorial code can also be found in a file called `FullExampleAMVFAlgorithm.cpp` in the same directory as `ExampleAMVFAlgorithm.cpp`.

Running the example algorithm

In your build directory, recompile and run the example on three pileup-50 pythia events to get your first ACTS vertices:

```
make -j4
./bin/ActsAMVFTutorial --evg-pileup 50 -n 3
```