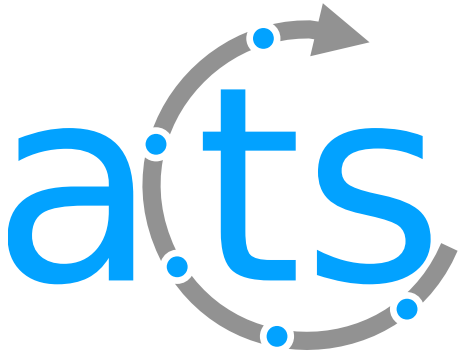
Differentiable Programming for High-Performance, Data-Intensive computations, CERN, 2021

Use of auto-differentiation within the ACTS toolkit

07.04.2021



Benjamin Huth
Universität Regensburg

Lukas Heinrich (CERN), Andreas Salzburger (CERN), Xiaocong Ai (DESY)

Content

1. **The `autodiff` library**
2. **Overview over ACTS**
3. **Autodiff in ACTS**
   a) Covariance Transport in Propagation
   b) Detector Alignment

The `autodiff` library

- C++ 17 library for autodiff
- https://github.com/autodiff/autodiff/
- Autodiff implementation:
  - Expression templates in forward mode -> **fast**
  - Graph of `std::unique_ptr` in reverse mode -> **slow**

The `autodiff` library

- Header-only
- operator-overload
- Very simple and handy API
- Eigen support

```cpp
#include <autodiff/forward.hpp>
using namespace autodiff;

auto f = [](auto x, auto y){ return x*x*sin(x*y); };
dual x = 1., y = 2.;

auto df_dx = derivative(f, wrt(x), at(x,y));
```

```cpp
#include <autodiff/forward/eigen.hpp>
using namespace Eigen;

auto f = [](auto x){ return x[0]*x[0]*sin(x[0]*y[1]); }
Vector2dual x = { 1., 2. };

auto jac = autodiff::jacobian(f, wrt(x), at(x));
```

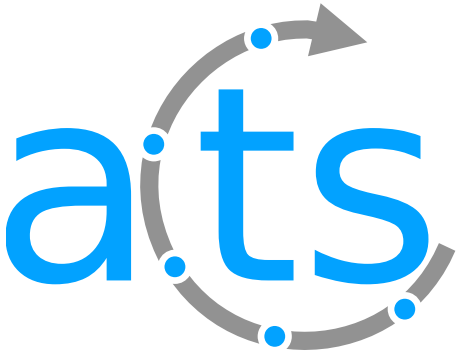
The `autodiff` library

- Performance (some matrix multiplications, 10x10 jacobian)

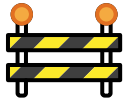
Autodiff Lib	Time [us]	Factor to best
Autodiff (forward)	6.7	1
Autodiff (reverse)	82327.5	~12k
Adept (forward)	24.1	3.6
Adept (reverse)	9.6	1.4
ADOL-C (fwd/rev chosen internally)	98.2	14.6

ACTS

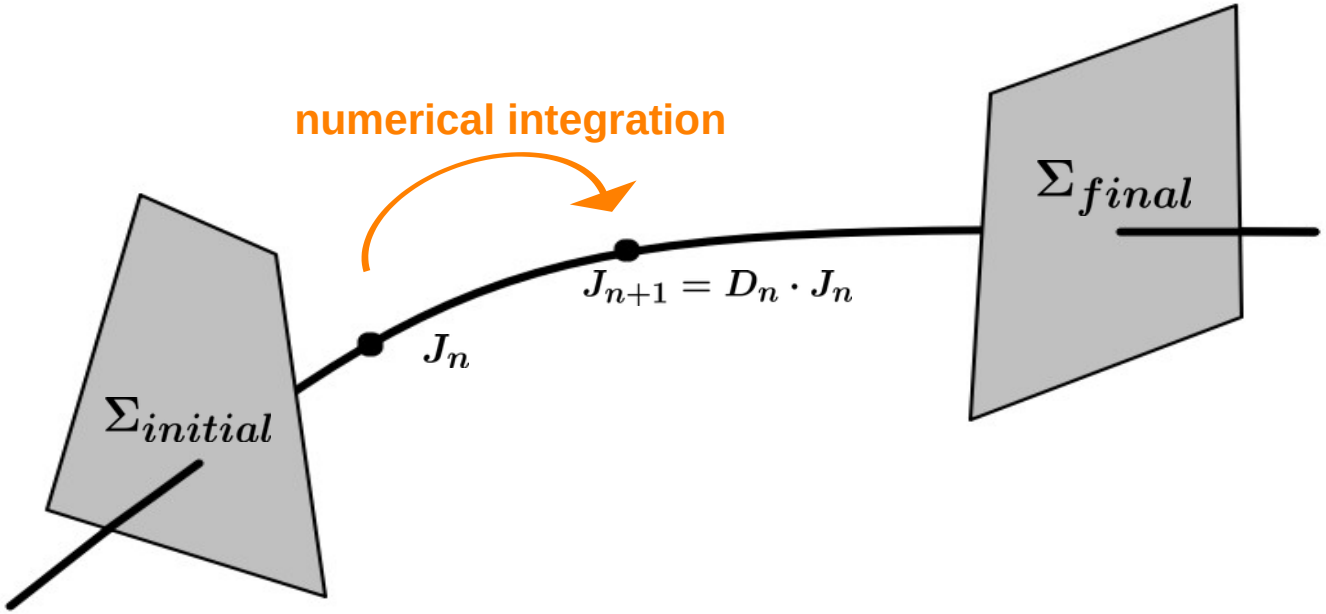
- Tracking Framework written in C++17
  - Track finding, Track fitting, vertexing, ...
  - Production quality implementations (parts are used in ATLAS software stack)
  - R&D platform (parallelization, ML, ...)
  - https://github.com/acts-project/acts



Autodiff in ACTS

- Autodiff is implemented as a plugin for differential validation
  - As far as we know, first use of autodiff in HEP reconstruction code
- Fokus on interface, not necessariely optimal performance
- At the moment: 2 applications:
  - Covariance transport in propagation ✓
  - Alignment derivatives (WIP) 

Covariance transport



Free track parameters:

$$\vec{u} = (\vec{x}, t, \vec{d}, {}^{q}/_{p})$$

numerical integration:

$$\vec{u}_n \rightarrow \vec{u}_{n+1}$$

use autodiff here:

$$D_n = \frac{\partial \vec{u}_n}{\partial \vec{u}_{n+1}}$$

Dimension of D: 6x6 – 8x8 (depending on parameterization)

$$\Sigma_{final} = J \cdot \Sigma_{initial} \cdot J^T$$

Implementation

High level interface

performs integration step

encapsulates physics (EOM)

- `Propagator< Stepper<Bfield, Extensions>, Navigator >`

Implementation

High level interface

performs integration step

encapsulates physics (EOM)

- `Propagator< Stepper<Bfield, Extensions>, Navigator >`
- **Implementation Strategy:**
  - Replace `double` with `autodiff::dual` in the `Extensions`
  - Use `autodiff::jacobian` to compute $\mathbf{D_n}$:

```
auto integration_step(const auto &in_params)
{
    // do integration
    return out_params;
}

// ...

Eigen::Vector8dual in_params;

auto D = autodiff::jacobian(integration_step, wrt(in_params), at(in_params));
```

Implemented in stepper, but must be rewritten for autodiff

Implementation

- Create **„Wrapper-Extension“** which is compatible to Stepper:

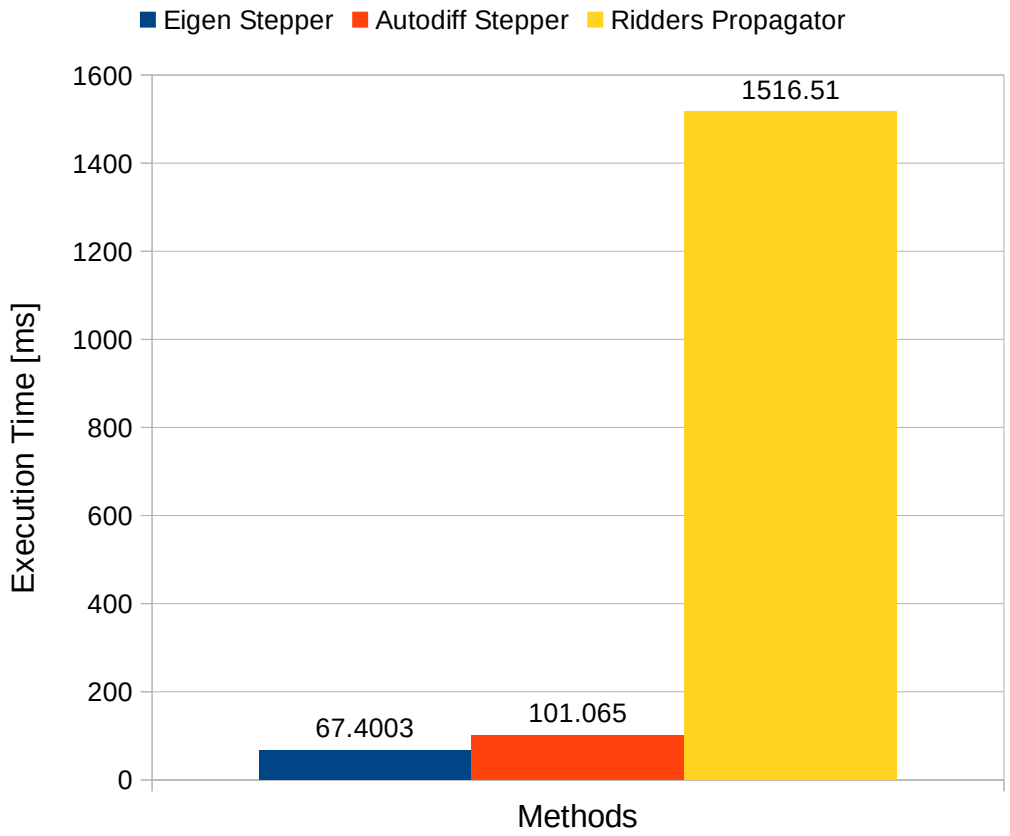
```
template< template<typename> typename basic_extension_t >
class WrapperExtension
{
public:
    // Provides public interface for Stepper
    basic_extension_t<double> m_doubleExtension;

private:
    // Used internally to compute D with autodiff
    basic_extension_t<autodiff::dual> m_autodiffExtension;

    // From this we build the jacobian
    auto integration_step(const auto &params);
};
```

- Drop-in-replacement for „normal“ extension:

```
using DefaultExtension = WrapperExtension< GenericDefaultExtension >;
using DenseExtension = WrapperExtension< GenericDenseEnvironmentExtension >;
```

Performance

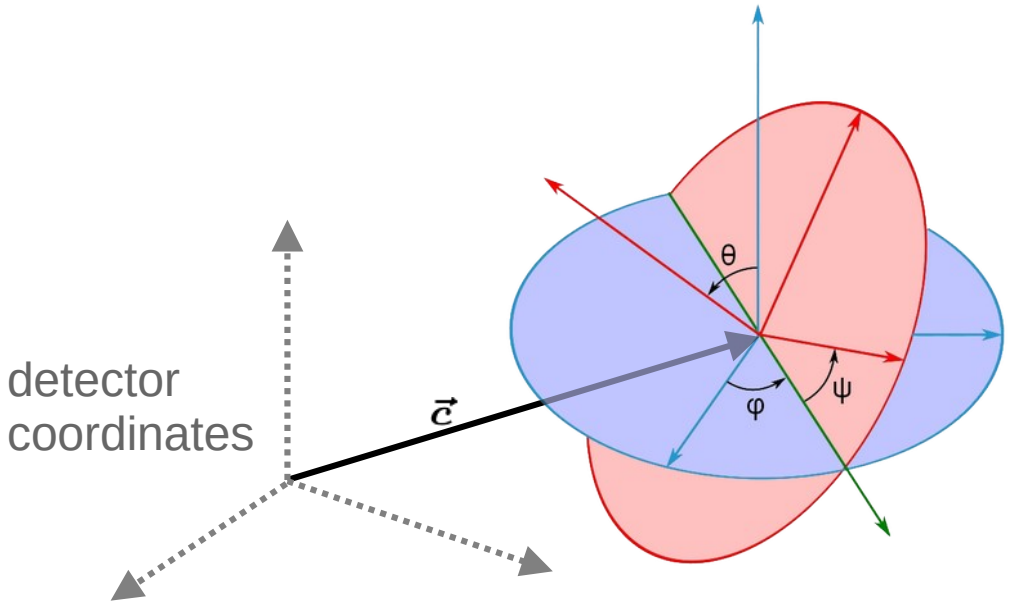
- Autodiff is not as efficient as hand-coded derivatives
- Much faster than numeric validation (Ridders algorithm)
- **Testcase:** Acts Propagation Example
  - B-Field = {0,0,2} T
  - TrackML detector
  - 1000 particles



Alignment derivatives (WIP)

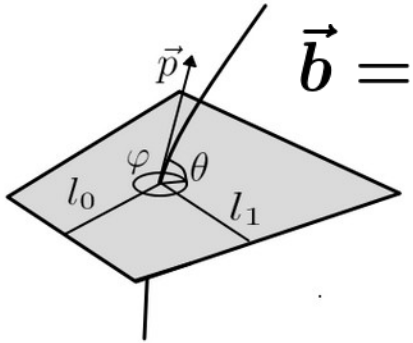
6 Alignment parameters per Surface:

$$\vec{\alpha} = (\underbrace{c_0, c_1, c_2}_{\text{center}}, \underbrace{\varphi, \theta, \psi}_{\text{rotation}})$$



This derivative is needed for estimation of **α**:

$$\frac{d\vec{b}}{d\vec{\alpha}} = \frac{\partial \vec{b}}{\partial \vec{\alpha}} + \frac{\partial \vec{b}}{\partial \vec{u}} \frac{\partial \vec{u}}{\partial s} \frac{\partial s}{\partial \vec{\alpha}}$$



bound parameters:

$$\vec{b} = (l_0, l_1, \varphi, \theta, {}^{q}/_{p}, t)$$

free parameters:

$$\vec{u} = (\vec{x}, t, \vec{d}, {}^{q}/_{p})$$

Implementation sketch

```
auto alignmentToBoundDerivativeAutodiff(
    const FreeVector& free,
    const FreeVector& pathDerivative)
{
    // free params + alignment params -> bound_parameters
    auto f = [](const auto &align, const auto &free)
    {
        return bound;
    };

    // free params + alignment params -> path correction
    auto g = [](const auto &align, const auto &free)
    {
        return path_correction;
    };

    // 1) alignToBound
    auto alignToBound = jacobian(f, wrt(align), at(align, free)).cast<double>();

    // 2) jacToLocal
    auto jacToLocal = jacobian(f, wrt(free), at(align, free)).cast<double>();

    // 3) alignToPath
    auto alignToPath = gradient(g, wrt(align), at(align, free)).cast<double>();

    // Combine the results
    return alignToBound + jacToLocal * pathDerivative * alignToPath;
}
```

user supplied *(in principle depends on EOM & B-field)*

$$\frac{d\vec{b}}{d\vec{\alpha}} = \frac{\partial \vec{b}}{\partial \vec{\alpha}} + \frac{\partial \vec{b}}{\partial \vec{u}} \frac{\partial \vec{u}}{\partial s} \frac{\partial s}{\partial \vec{\alpha}}$$

Conclusion

- Thanks to header-only libraries, very easy to include in a big project such as ACTS
- Autodiff is an easy to use tool to validate derivatives in all kinds of situations
- Autodiff cannot outperform hand-coded derivatives (?)
  - Are there solutions to use autodiff in such cases without performance loss?

Thank you for your attention!

Any Questions?