



**GEANT4**  
A SIMULATION TOOLKIT

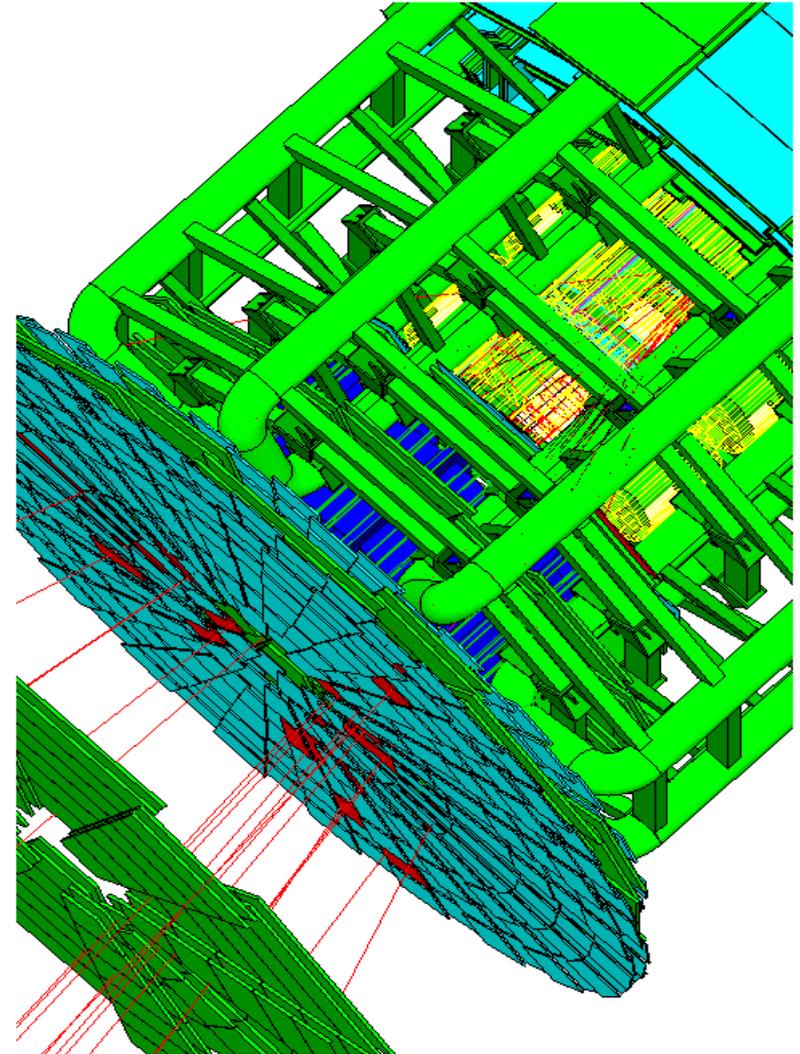
**Version 10.5**

# Magnetic Fields

John Apostolakis (CERN)  
Geant4 Advanced Course at CERN  
26-27 March 2019

# Contents

- Magnetic field
- Integration of trajectories in field
- Other types of field



Many slides adapted from M. Asai (SLAC)



**Version 10.5**

## Defining a magnetic field

# Basics 1: Defining a Magnetic field

- How to create a (magnetic) field ? Instantiate it in the *ConstructSDandField()* method of your *DetectorConstruction*

- Uniform field :

- Use an object of the G4UniformMagField class

```
G4MagneticField* magField =
```

```
    new G4UniformMagField(G4ThreeVector(1.*Tesla,0.,0.));
```

- Non-uniform field :

- Create your own concrete class derived from G4MagneticField and implement `GetFieldValue` method.

```
void MyField::GetFieldValue(
```

```
    const double Point[4], double *field) const
```

- Point[0..2] are x,y,z **position in global coordinates**, Point[3] is **time**
- field[0..2] are output x,y,z components of magnetic field (in G4 units)

## Basics 2: How to assign a field to the whole detector

- A global field manager is the one associated with the 'world'
  - it already exists, before G4VUserDetectorConstruction is called,
  - it is created / set in G4TransportationManager.
- To associate your field with the world, you must obtain that global field manager:

```
G4Fieldmanager* globalFieldMgr = G4TransportationManager::  
    GetTransportationManager()->GetFieldManager();  
  
globalFieldMgr->SetDetectorField(field);
```

- Other volumes can override a global field
  - An alternative field manager can be associated with any logical volume – or a ‘region’
    - The field must accept **position in global coordinates** and return the value of the **field in global coordinates**

```
auto localFieldMgr = new G4FieldManager(magField);  
Region->setFieldManager(localFieldMgr);
```

The region Field Manager overrides the global one (if any.)

```
logVolume->setFieldManager(localFieldMgr, true);
```

The logical Volume’s Field Manager overrides the region and the global (if any.)

Note that this affects also volumes contained in `logVolume`:

- By default only sub-volumes that do not have yet have a field manager.
- Using ‘**true**’ for the second argument asks it to push the field to all the sub-volumes, even if a daughter volume has its own field manager.

Choosing an appropriate stepper for your field

Setting precision parameters

# CUSTOMIZING FIELD PROPAGATION

```
MyField* fMyField = new MyField();

G4Fieldmanager* fieldMgr = new G4FieldManager();

fieldMgr->SetDetectorField(fMyField);

fieldMgr->CreateChordFinder(fMyField); // for B-field only

G4bool forceToAllContained = true;

fMagneticLogical->SetFieldManager(fieldMgr,

                                   forceToAllContained);

// Register the field and its manager for deleting

G4AutoDelete::Register(fMagneticField);

G4AutoDelete::Register(fieldMgr);
```

- `/example/basic/B5` is a good starting point

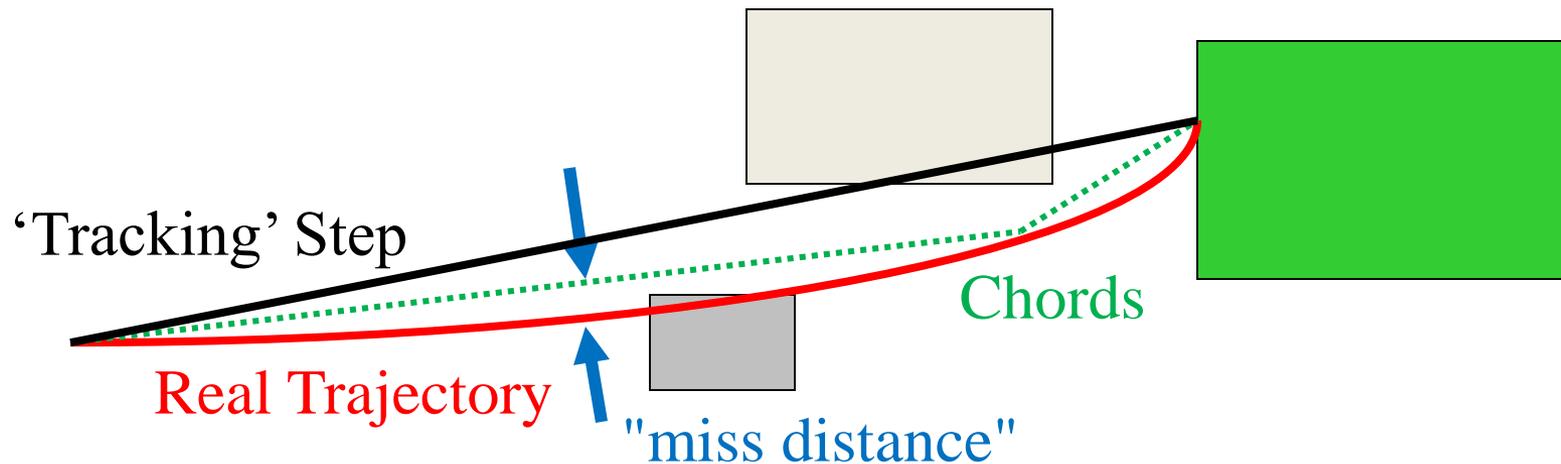
# Field integration

- In order to propagate a particle inside a field (e.g. magnetic, electric or both), we solve **the equation of motion** of the particle in the field.
- We use a Runge-Kutta method for the integration of the ordinary differential equations of motion.
  - Several Runge-Kutta ‘steppers’ are available.
- In specific cases other solvers can also be used:
  - In a uniform field, using the analytical solution – a helix.
  - In a smooth but varying field, with RK+helix.
- Using the method to calculate the track's motion in a field, Geant4 breaks up this curved path into linear chord segments.
  - We determine the chord segments so that they closely approximate the curved path.

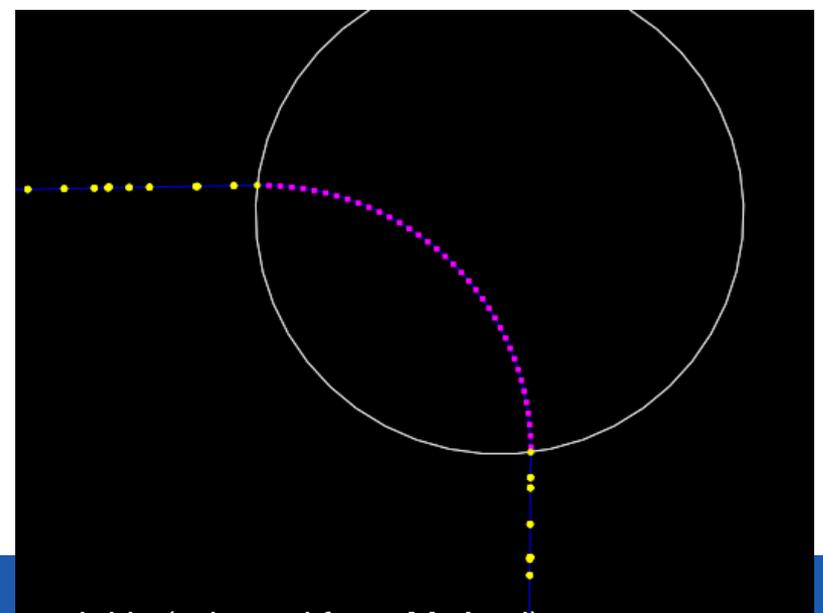
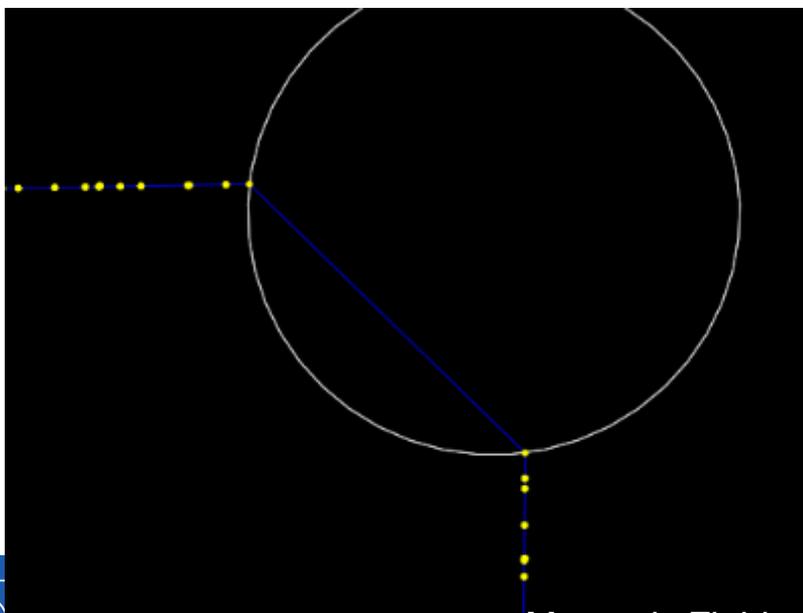
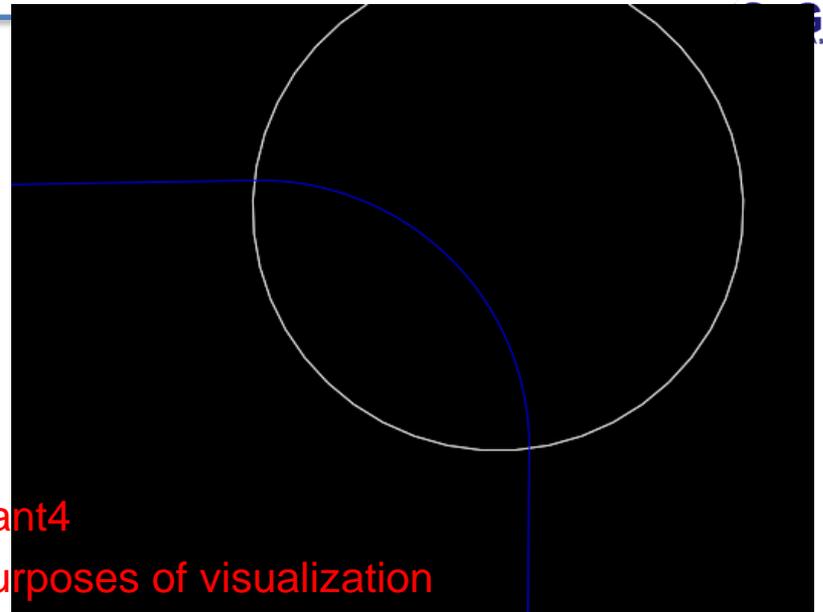
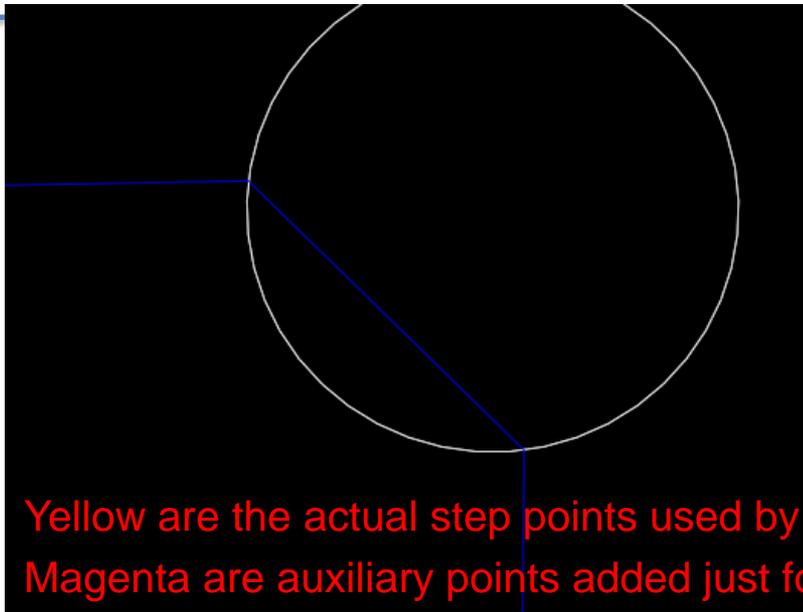


# Tracking in field

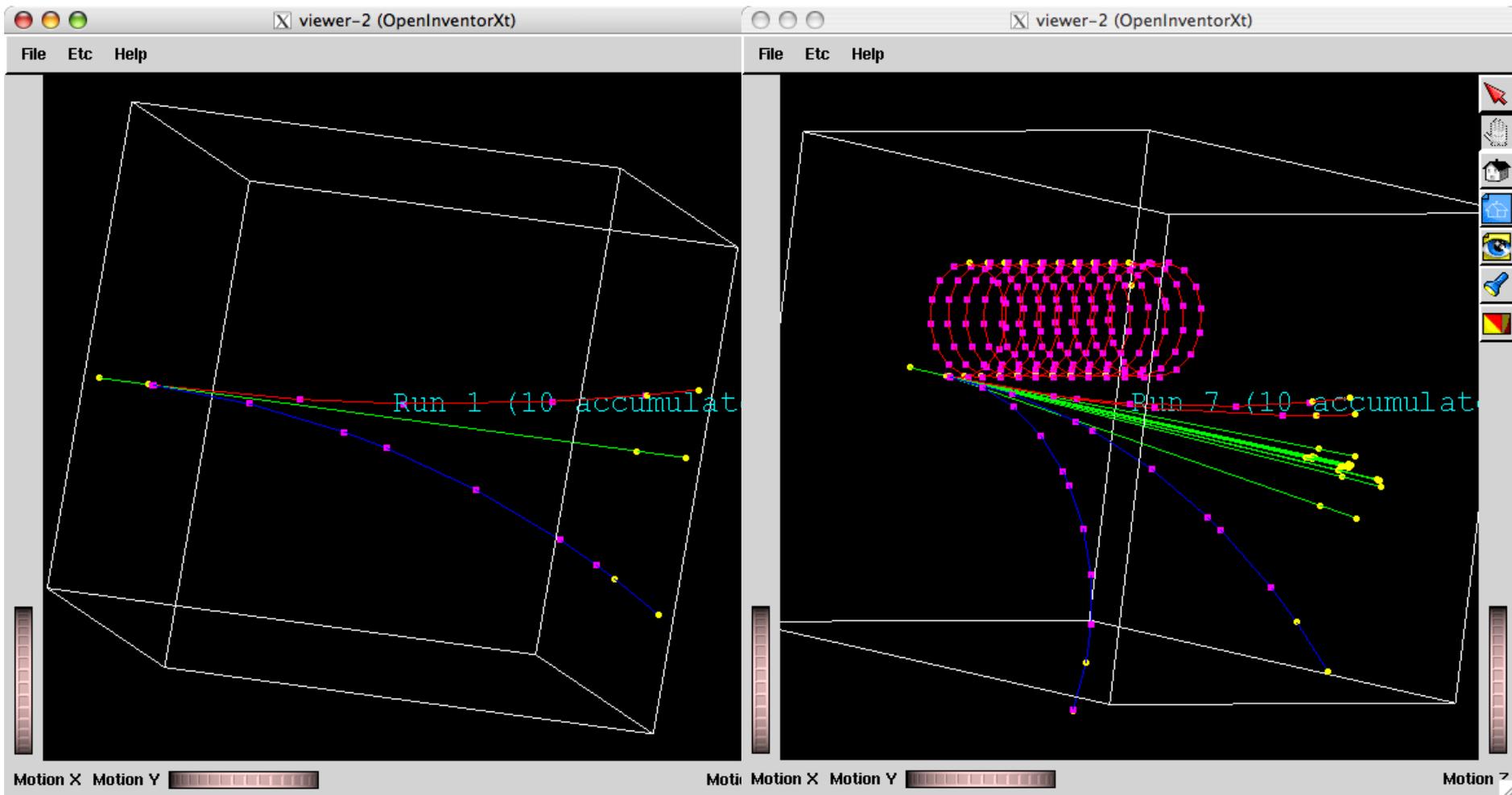
- We use the chords to interrogate the **G4Navigator**, to see whether the track has crossed a volume boundary.
- One physics/tracking step can create several chords.
  - In some cases, one step consists of several helix turns.
- User can set the accuracy of the volume intersection,
  - By setting a parameter called the **“miss distance”**
    - It is a measure of the error in whether the approximate track intersects a volume.
    - It is quite expensive in CPU performance to set too small “miss distance”.



# Regular versus Smooth Trajectory



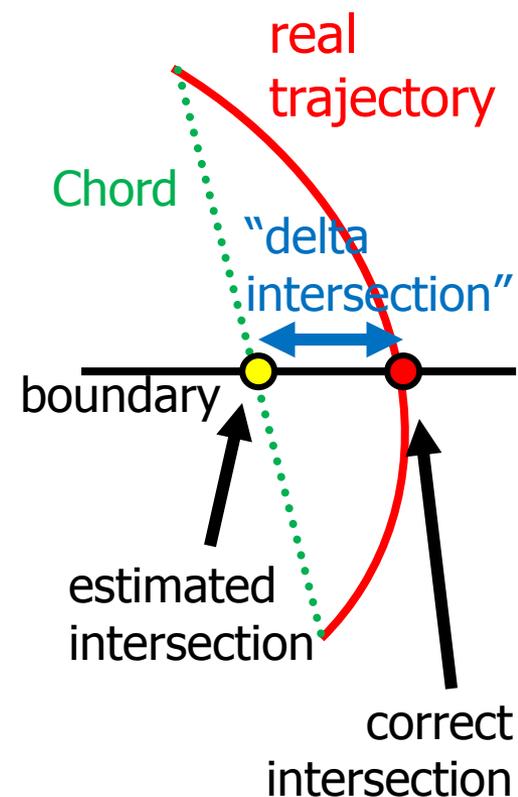
# Smooth Trajectory Makes Big Difference for Trajectories that Loop in a Magnetic Field



- Yellow dots are the actual step points used by Geant4
- Magenta dots are auxiliary points added just for purposes of visualization

# Tunable parameters

- In addition to the “miss distance” there are two more parameters which the user can set in order to adjust the accuracy (and performance) of tracking in a field.
  - These parameters govern the accuracy of the intersection with a volume boundary and the accuracy of the integration of other steps.
- The “delta intersection” parameter is the accuracy to which an intersection with a volume boundary is calculated.
  - **Important:** it is used to **limit the bias** that our algorithm (for boundary crossing in a field) exhibits: the intersection point is always on the 'inside' of the curve.
  - Set its **value** much **smaller** than your acceptable error, to limit the effect of this bias.



- The “**delta one step**” parameter is accuracy for endpoint of integration steps that do **not intersect** a volume boundary.
  - It also limits on the estimated error of the endpoint of each physics step (essentially it is  $< 1000 \delta_{1 \text{ step}}$ .)
  - Values of  $\delta_{\text{intersection}}$  and  $\delta_{1 \text{ step}}$  should be within one order of magnitude.
- We recommend to use a **maximum relative tolerance**  $\epsilon_{\text{max}}$  for the integration error
  - $\epsilon_{\text{max}}$  limits the estimated error for large steps:  $|\Delta x| < \epsilon_{\text{max}} s$  and  $|\Delta p| < \epsilon_{\text{max}} |p|$
- These tunable parameters can be set by

```
theChordFinder->SetDeltaChord( missDistance );
theFieldManager->SetDeltaIntersection( deltaIntersection );
theFieldManager->SetDeltaOneStep( deltaOneStep );
theFieldManager->SetEpsilonMax( epsilonMax );
theFieldManager->SetEpsilonMin( 0.1 * epsilonMax );
```
- Further details are described in **Section 4.3 (Electromagnetic Field) of the Application Developers Manual.**



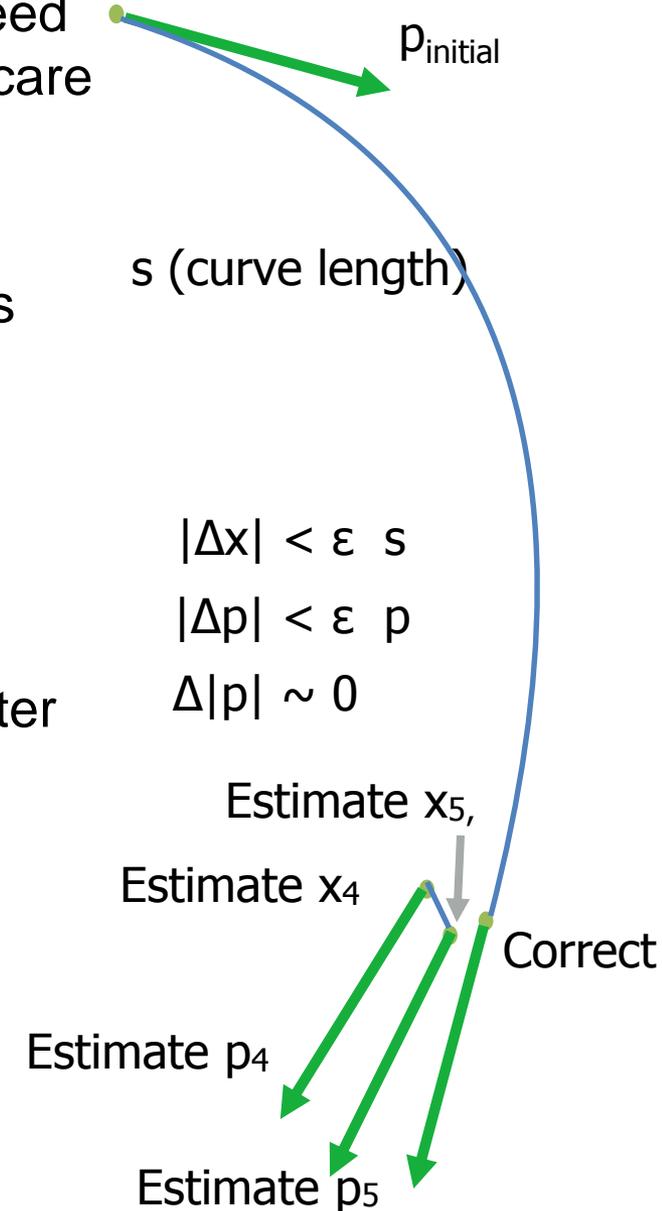
Version 10.5

Field integration – usual and custom methods  
Other types of field

- Runge-Kutta (RK) integration is used to compute the motion of a charged track in a any type of field: magnetic, electric, combined EM, gravitational or a mix.
- There are two kinds of RK steppers are available in Geant4:
  - **General-purpose** steppers, applicable to any field type, usable for any ODE.
  - **Specialized** steppers applicable only to charged particle motion in **pure magnetic** fields.
- RK steppers related to the order of the Taylor expansion from which they derive:
  - low 1<sup>st</sup>, 2<sup>nd</sup> or 3<sup>rd</sup> order are considered ‘low-order’ steppers – one field evaluation per stage is needed
  - 4<sup>th</sup> order (4 evals/step) & 5<sup>th</sup> order (6-7 eval/step) are the typical ‘sweet’ spot
  - higher order require many more stages (evaluations) per step – their increased accuracy is relevant for very specialized applications.
- An integration method must estimate both an end state (position, momentum, maybe polarisation) and an integration error for each state variable
  - Old (pre 1967) ‘simple’ methods estimated error by having the step

# Integrating efficiently

- Given a detector's field  $B(x,y,z)$  [or  $B+E$  ] we need to integrate the trajectory of each track, taking care
  - to stay within a relative accuracy  $\epsilon$
  - to be fast – in particular using as few calls as possible to the field evaluation method
- Typically choose Runge-Kutta methods
  - No history needed – self starts
  - Adjusts easily to change(s) of momentum after (frequent) collisions / interactions
  - Ability to adjust step size



- Runge-Kutta (RK) integration is used to compute the motion of a charged track in a any type of field: magnetic, electric, combined EM, gravitational or a mix.
- Many general steppers are available applicable to any equation / field. Some steppers are specialized, applicable only to pure magnetic fields.
- of low and high order, and specialized steppers for pure magnetic fields.
- Default in G4 is the general purpose [G4DormandPrinceRKF45](#) an embedded 4<sup>th</sup>-5<sup>th</sup> order RK stepper. ( Embedded = compares 4<sup>th</sup> & 5<sup>th</sup> order to estimate error.)
  - It typically uses 6 field evaluates per field step, and provides the derivative .
  - Earlier Geant4 versions (<10.4) had [G4ClassicalRK4](#) as default – this is just as general and very robust, but requires 11 field evaluations per step.
- If the field has very rough or smooth, lower or higher order steppers may obtain the results of same quality using fewer computing cycles.

# Explicit Runge-Kutta methods

Text

- “Integrate”  $dy/dx = F(x, y)$  from  $x_0$  to  $x_0+h$
- Uses evaluations of  $F(x, y)$ 
  - $f_i = F(x_0 + a_i h, y_0 + h \sum_{j<i} b_{ij} f_j)$
  - $y_{estim}(x_0 + h) = \sum_i c_i f_i$
- Each method has its ‘tableau’ made up by  $a_i, b_{ij}, c_i$
- Key Parameters of an RK method:
  - Number of ‘stages’ = number of evaluations of  $f()$
  - ‘Order’ = the expected scaling of the error  $\sim h^N$
  - Embedded method = 2nd ‘line’ to estimate error

$$f_1 = F(x_0, y_0)$$

$$f_2 = F(x_0 + a_2 h, y_0 + h b_{21} f_1)$$

$$f_3 = F(x_0 + a_3 h, y_0 + h b_{31} f_1 + h b_{32} f_2)$$

0			
$a_1$			
$a_2$	0	$b_{21}$	
$a_3$	$b_{31}$	$b_{32}$	
1	$c_1$	$c_2$	$c_3$

$c_j$

$$y_{RBS3} = 2f_1/9 + f_2/3 + 4f_3/9$$

$c'_j$	$2/9$	$1/3$	$4/9$
--------	-------	-------	-------

$$y'(x_0 + h) = \sum_i c'_i f_i$$

$$\Delta y = \sum_i (c'_i - c_i) f_i$$



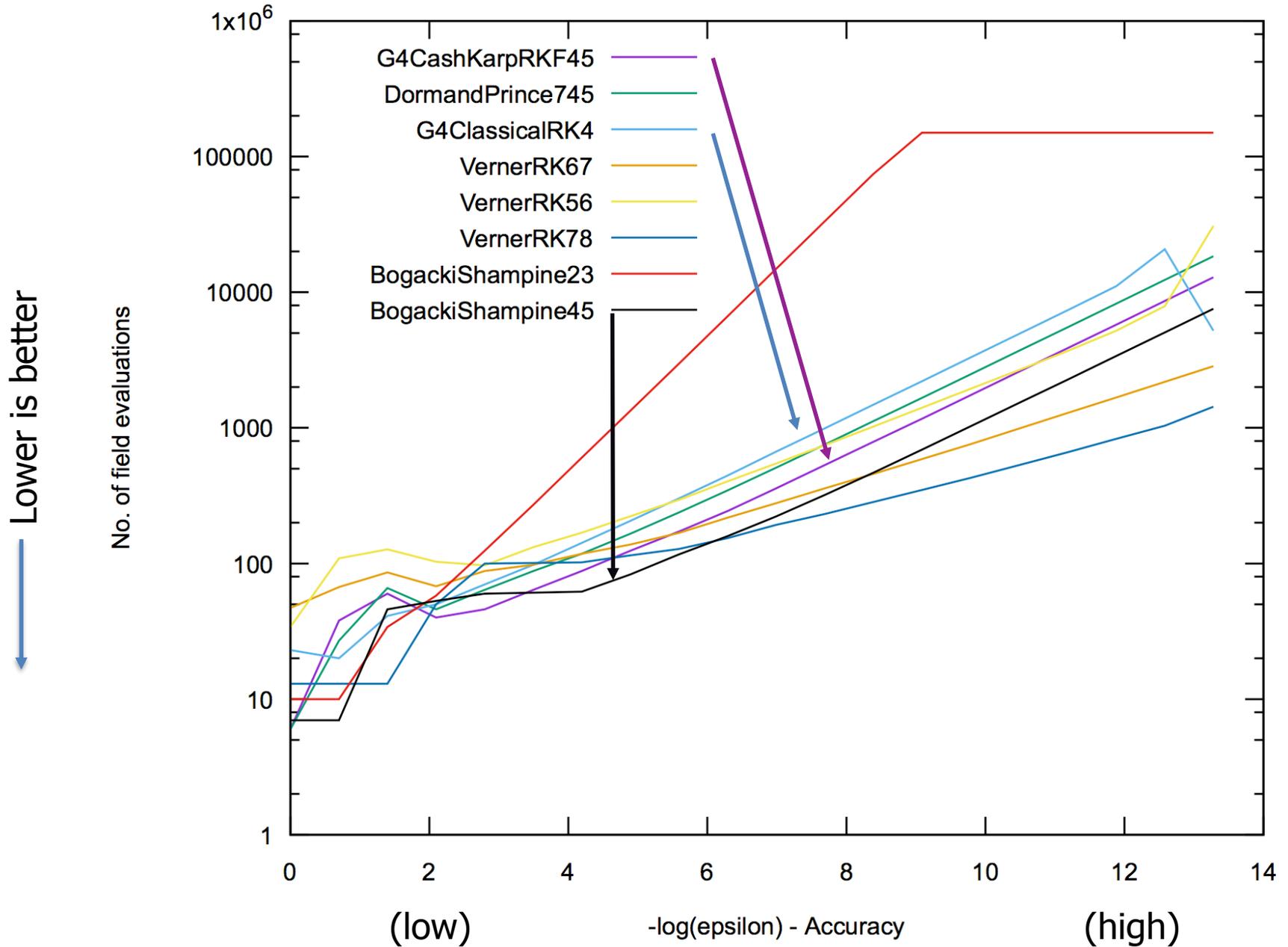
# Alternative Runge-Kutta steppers – some context

- ‘Classical RK4’ = original 4th order method, 4 stages (i.e. evaluations)
  - Error estimate from breaking step in two => 1(initial)+10 evaluations
  - General and very robust, but costly (due to 10 field evaluations per step.)
- Embedded methods (invented by [Fehlberg](#)) provide built-in error estimates by comparing estimators of 2 different orders (that share evaluation points).
  - [Cash-Karp](#) (1990) - uses difference of 5th & 4th order method, six stages = 6 evaluations of derivative
  - [Dormand Prince](#) (DoPri5), the most widely used method improves stability and performance. It’s the default also in MATLAB, GNU Octave (“ode45”)
    - It evaluates the field value and derivative at the final point (with  $p_{\text{final}}$ ), called FSAL. So it’s ready for use in the next step (=> 1 less eval.)
- AtlasRK4/NystromRK4: 3 field evaluations + evaluation of error using numerical estimate of 4th derivative - restricted to B-fields
- Helix - for constant field *FSAL = First Same As Last*
- Lower order RK methods for short steps, and/or lower accuracy

# Choices of Runge-Kutta methods (steppers)

- The default is the general purpose **G4DormandPrinceRKF45** an embedded 4<sup>th</sup>-5<sup>th</sup> order RK stepper. ( Embedded = compares 4<sup>th</sup> & 5<sup>th</sup> order to estimate error.)
  - It typically uses 6 field evaluates per field step, and provides the derivative .
  - Note that earlier Geant4 versions (<10.4) had **G4ClassicalRK4** as default – it is a good robust backup stepper.
- If the field is very smooth, you may consider higher order steppers
  - of most potential interest in large volumes filled with gas or vacuum.
- If the field is rough, 3<sup>rd</sup> order steppers may obtain the results of same quality using fewer computing cycles
  - 3<sup>rd</sup> order stepper(s): **G4SimpleHeum**, **G4BogackiShampine23** (FSAL)
- For reasonably smooth (or not very rough) fields, the choice between 3<sup>rd</sup> , 4<sup>th</sup> or 5<sup>th</sup> order steppers should be made by trial and error.
- In a field calculated from a 'rough' field map, a lower order stepper can be more robust.
  - 2<sup>nd</sup> order **G4SimpleRunge** and **G4ImplicitEuler**
- The less smooth the field is, we would expect that a lower the order of the stepper would be more robust (but no lower than 2<sup>nd</sup> order.) However the relative performance may again depend on many factors, and trial/error is recommended to identify the best performing stepper.

Thanks to Somanth Banerjee (Google Summer of Code student 2015)



- Trying a few different types of steppers for a particular field or application is suggested, if maximum performance is a goal.
  - What is the most performant option may also be different in different regions
    - e.g. depending on whether the field is varying greatly.
- Specialized steppers for pure magnetic fields are available. Some assume that a local trajectory in a slowly varying field will resemble a helix.
  - Combining this in with a variation, the Runge-Kutta method can provide higher accuracy at lower computational cost when large steps are possible
  - Suggested are [G4HelixSimpleRunge](#) and [G4HelixSimpleHeum](#)
- To change the stepper  
**theChordFinder**
  - `->GetIntegrationDriver()`
  - `->RenewStepperAndAdjust( newStepper );`
- Further details are described in [Section 4.3 \(Electromagnetic Field\) of the Application Developers Manual](#).

# Other types of field

- For pure electric field, Geant4 has `G4ElectricField` (base) and the simple `G4UniformElectricField` (concrete) classes.
- `G4ElectroMagneticField` is the base class for combined electro-magnetic fields.
- The *equation of motion* class for electromagnetic field is `G4MagElectricField`.

An example:

```
G4ElectricField* EMfield
```

```
    = new G4UniformElectricField( G4ThreeVector(0., 100000.*kilovolt/cm, 0.) );
```

```
auto equation = new G4EqMagElectricField(EMfield);
```

```
auto stepper = new G4CashKarpRKF45( equation, nvar );
```

```
G4FieldManager* fieldMgr
```

```
    = G4TransportationManager::GetTransportationManager()-> GetFieldManager();
```

```
fieldMgr->SetDetectorField( EMfield );
```

```
auto integrDriver= new G4MagInt_Driver( fMinStep, stepper,  
                                         stepper->GetNumberOfVariables() );
```

```
G4ChordFinder* fChordFinder = new G4ChordFinder(integrDriver);
```

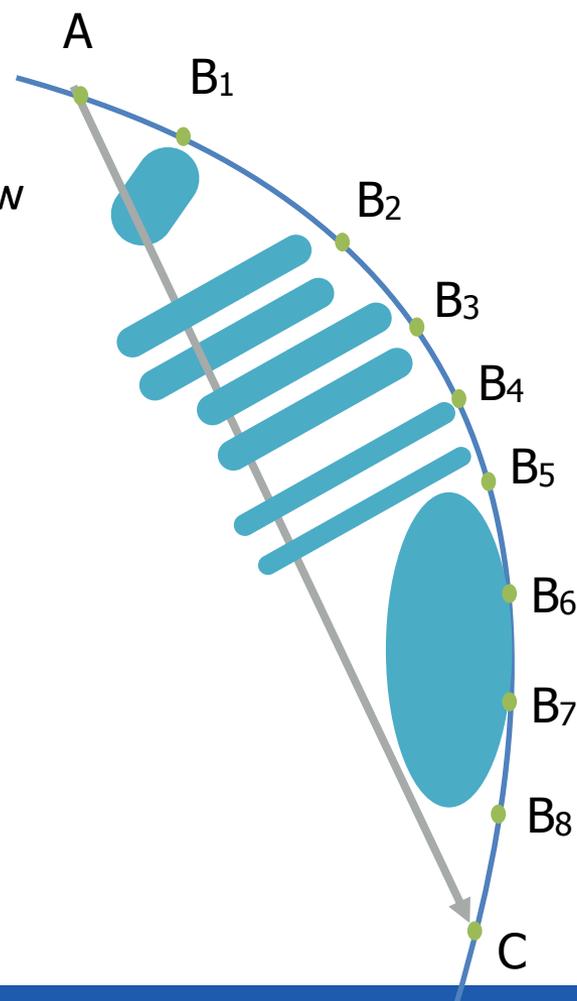
*Notes:*

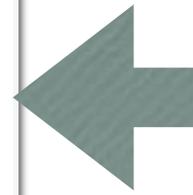
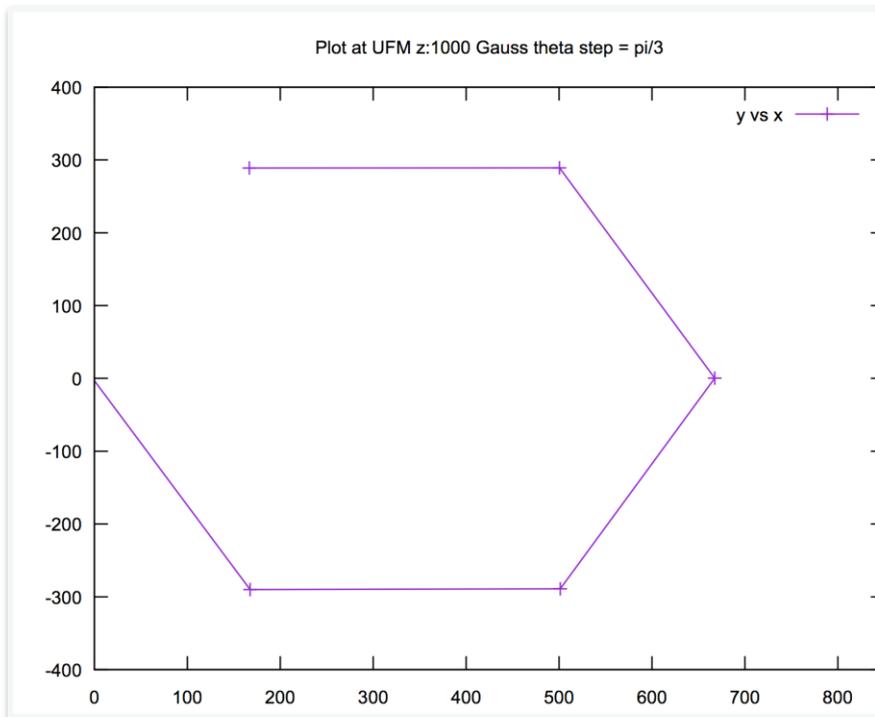
1. A user-defined field type can be time-dependent (its value can change with time).
2. A user can create their own type of field, inheriting from `G4VField`, and an associated *Equation of Motion* class (that inherits from `G4EqRhs`).

A variety of promising Runge-Kutta methods, seeking improvements in

- ▶ efficiency - accuracy of error estimation (fewer 'bad' steps)
- ▶ accuracy of solution - extending it further (larger steps)
- ▶ speed - reduce number of evaluations of derivative/field, e.g. reuse derivatives
- ▶ **interpolation**: obtain the state  $(x,p)$  at any intermediate point

- ▶ Selected RK methods offer capability of estimating any intermediate point given its 'distance' along the curve
  - ▶ One-time cost of a few extra field evaluations
- ▶ Reduced cost of evaluating intermediate points (vs new integration)
  - ▶ Enable faster location of intersection point with surface boundary
- ▶ Enabled using a new type of IntegrationDriver:
  - ▶ `auto interpStepper= new G4DormandPrince745( equation, nvar );`
  - ▶ `auto * integrDriver= new G4InterpolationDriver(fMinStep, interpStepper, interpStepper->GetNumberOfVariables() );`

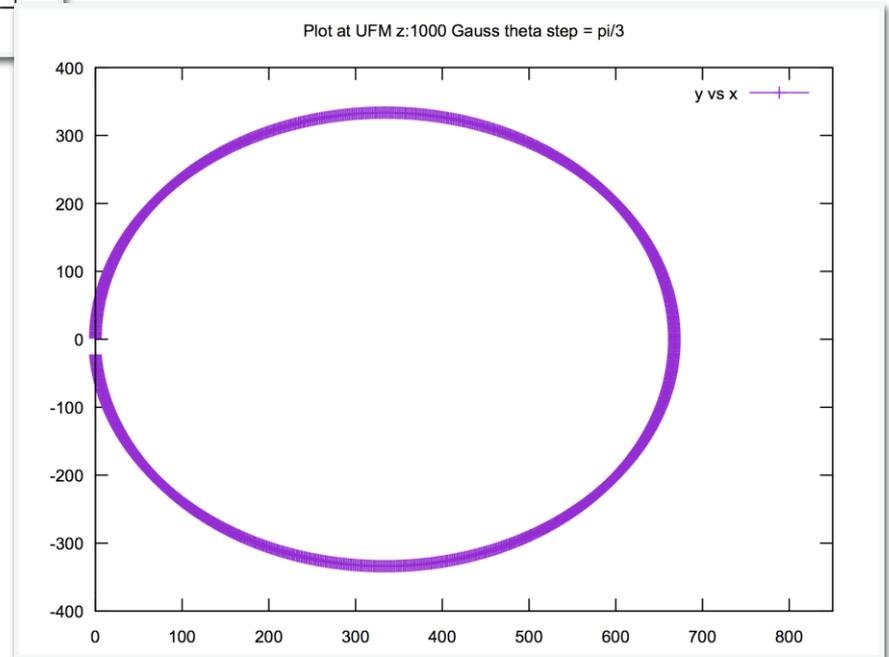




6 computed points in a circular trajectory

Calling 100 times between each pair of points, to give "Dense" output

Interpolated result



These promising Runge-Kutta methods are currently optional, and can be tried out as follows:

- ▶ Interpolation via the G4InterpolationDriver class and a compatible stepper:

```
#include "G4FSALInterpolationDriver.hh"
```

```
...
```

```
auto interpolatingStepper= new G4DormandPrince745( equation, nvar );
```

```
auto integrDriver= new G4InterpolationDriver(fMinStep, interpolatingStepper,  
                                             interpolatingStepper->GetNumberOfVariables() );
```

- ▶ The FSAL property of a stepper is available for a few selected steppers. To use it we also use an alternative 'Driver' class:

```
#include "G4RK547FEq1.hh"
```

```
#include "G4FSALIntegrationDriver.hh"
```

```
..
```

```
stepper1 = new G4RK547FEq1( equation );
```

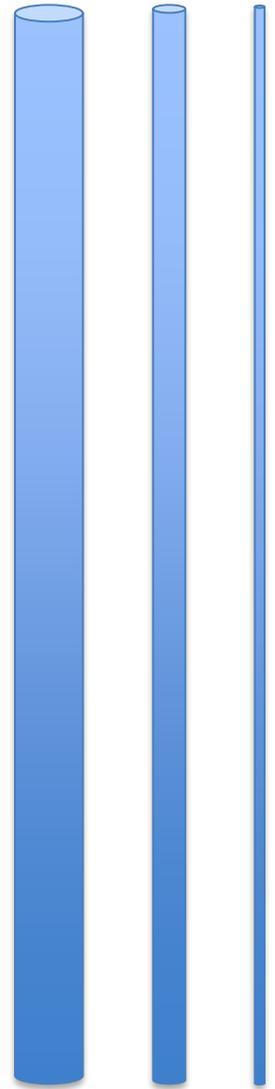
```
auto fsalDriver = new G4FSALIntegrationDriver<G4RK547FEq1>( fMinStep, stepper1 );
```

```
// Note: Soon it will usable also for/with Dormand Prince 4/5
```

*(Thanks to Dmitry Savin,  
GSoC 2016 & 2017)*

# Dealing with looping particles

- In volumes with magnetic field and either vacuum or gas as material some tracks may need a large ( $10^4$ ) or very large ( $>10^6$ ) number of integration steps
  - This can be a major sink of CPU time
- So within a physics step there is a limit of 1,000 integration steps (tunable.)
  - When a track reaches this limit, it is marked as ‘looping’. It is now a candidate for being killed.
- Geant4 will kill particles found to be looping:
  - If  $E < E_{\text{warning}}$  a track is killed immediately without warning
  - If  $E_{\text{warning}} < E < E_{\text{important}}$  the track is killed, and a small warning is printed (to cout in Geant4 versions  $< 10.5$  )
  - If  $E > E_{\text{important}}$  the track is given an extra number of chances (by default 10) before being killed.
- Their values can be changed using methods of G4Transportation. Default values are
  - $E_{\text{warning}} = 100 \text{ MeV}$
  - $E_{\text{important}} = 250 \text{ MeV}$



- In Geant4 10.5 several changes were implemented:
  - only stable particles are killed
  - each particle with energy above the warning energy which is killed **generates a detailed warning** (using G4Exception) with location, volume, material, particle momentum and energy.
  - for the first 5 tracks killed a detailed description is printed that describes the criteria and parameters used to decide what tracks are killed, and guidance (next).
- Guidance regarding how to ‘save’ tracks:
  - by changing the values of thresholds or
  - by adopting different integration methods.

- Runge-Kutta (RK) integration is used to compute the motion of a charged track in a any type of field: magnetic, electric, combined EM, gravitational or a mix.
  - Many general steppers are available applicable to any equation / field
  - A few specialised steppers can be used only for pure magnetic fields.
- Default in G4 is the general purpose [G4DormandPrinceRKF45](#) an embedded 4<sup>th</sup>-5<sup>th</sup> order RK stepper
  - It typically uses 6 field evaluates per field step, and provides the derivative .
- If the field has very rough or smooth, consider lower or higher order steppers
  - Expect same quality using fewer computing cycles,
  - Try a different stepper (or two) to see whether it improves CPU time.
- RK steppers with interpolation will reduce the number of field calls for each intersection boundary
  - Currently optional, with plans to introduce them as default in 2019.
- Different types of fields available, and user can create their own
  - A field must be accompanied by its corresponding equation of motion.