

**NIKHEF**



**LHCb**  
**THCP**

**Bender v11r0 as your  
analysis environment**

Vanya BELYAEV

- Bender Pages and Bender pages by Lena Mayatskaya
- Bender mailing list
- Bender Savannah portal (news, bugs, tasks, ...)
- Bender Tutorial TWiki
- **Bender HyperNews, TWiki, FAQ, User Guide and Manual :**
  - ☹ not yet. still in the bottle of inc
- **Bender Examples**
  - including nice scripts from Diego for  $B_s \rightarrow \mu\mu$  background studies
- **"Bender-helpdesk@lhcb.cern.ch"**
  - 13-1-018 at CERN
  - +41 (0) 22 767 40 24

# When use Bender

- **Python:** perfect for prototyping
  - e.g. develop the cuts for preselection/stripping
- **Interactive:** perfect for “short” (“supervising”) tasks
  - resolutions, efficiencies,
  - spectra
  - “reflections”
- **Flexible & Friendly.**
  - good for “the final” analysis of small data sets
  - combine with **Root, Panoramix, RooFit,...**





- Stripping does not support Bender.
- Reasons?
  - ☹️ *Some CPU penalty* for Bender selections vs LoKi selections is unavoidable (Python vs C++)
    - could be visible/sizeable for “minimal” job
      - mainly comes from the explicit loops, N-Tuples and explicit manipulations with dictionaries:  

```
sqrt (p.px () *p.px () +p.py () *p.py ())
```
    - could be *very small* for realistic selection
      - And of course for *well-coded* lines

**Negligible with patterns (C++) ☺️**





# The goal

- To be able in one hour to make following steps
  - selection of particles
  - extraction of the basic information
  - fill the histograms
  - looping over the particles
  - fill N-Tuples
  - save interesting combinations
  - Match to Monte Carlo truth

**Covers >95% of functionality needed for event selection**



```
> SetupProject Bender v11r0 -build-env
> SetupProject Bender v11r0
> getpack Tutorial/BenderTutor v11r0
> cd Tutorial/BenderTutor/cmt
> make
> source ./setup.[c]sh
> ...
```

# Minimal Bender script

```

from Bender.Main import *
from Configurables import DaVinci
DaVinci (
    DataType      = 'DC06' ,
    Simulation     = True   ,
    HltType       = ''     )
gaudi = appMgr()
evtSel = gaudi.evtSel()
evtSel.open ( ... )
gaudi.run(10)
gaudi.exit()

```

Well, It is not  
Bender, it is  
Configurables  
+ GaudiPython

**BUT: Take care about input data!!**

`$BENDERTUTORROOT/solution/Minimalistic_0.py`





```
from Bender.Main import *
def configure() :
    from Configurable import DaVinci
    DaVinci ( ... )
    gaudi=appMgr()
    ...
    return SUCCESS
```

Application and Components Configuration

```
if __name__ == '__main__' :
    configure()
    run(100)
```

Job steering

`$BENDERTUTORROOT/solutions/Minimalistic.py`



# Scripts vs modules

- Dilemma in Python: *scripts vs modules*
- Scripts are a bit more intuitive and a bit easier to write ☺
  - Problems with reusing ☹
- Modules require some discipline & conventions ☹
  - Full power of OO, including classes & extensions
  - Easy to import and reuse ☺
  - The only way to assemble "large" application from pieces
- **Be friendly: code modules**
  - loose nothing
  - gain a lot

- Script above:

```
import myscript
```

Execute everything out of control.

To change something - copy, cut-n-paste

(and of course duplicate your colleague's bugs + introduce some of your own)

- Module above - easy reusable by your colleagues:

```
import mymodule
```

```
mymodule.config()
```

```
gaudi.run(100)
```





# "Hello, World!" (I)

- The simplest possible BENDER "algorithm"
- Follow LoKi's style:
  - *inherit the algorithm from useful **base class***
  - (re)implement the "**analyse**" method

```
class HelloWorld(Algo) :  
    def analyse( self ) :  
        print 'Hello, World!'  
        return SUCCESS
```

```
$BENDERTUTORROOT/solutions/HelloWorld.py
```

# "Hello, World!" (II)

- One needs to instantiate the algorithm:  
`alg = HelloWorld( 'Hello' )`
- Add it to the list of 'active' algorithms (`TopAlg`)  
`gaudi.addAlgorithm ( alg )`
- Or:
- Substitute the current list of algorithms:  
`gaudi.setAlgorithms ( [alg] )`
- Execute 😊  
`gaudi.run(10)`

`$BENDERTUTORROOT/solutions/HelloWorld.py`

- **C++: GaudiAlgorithm/LoKi**

```
const LHCb::MCParticle::Container* mcps  
    =get<LHCb::MCParticle::Container>( "MC/Particles")
```

- **Python: Bender / GaudiAlgs**

```
mcps = self.get( 'MC/Particles' )
```

- **Mnemonic rule:**

**this->      →      self.**

**`$BENDERTUTORROOT/solutions/DataAccess.py`**



- **Inside the algorithm**

```
dataSvc = self.evtSvc()  
hdr      = dataSvc['Header']  
print 'Event #', hdr.evtNum()
```

- **Outside algorithm (script)**

```
dataSvc = gaudi.evtSvc()  
hdr      = dataSvc['Header']  
print 'Run #', hdr.runNum()
```

The only way!

# Attributes and (python) loops

**MCParticle**

```
for p in mcps :
    print `ID=` , name( p )
    print `PX=` , p.momentum().px()
    print `PY=` , p.momentum().py()
    print `decay: ' , p.decay( True )
```

**From Dictionaries**

- All containers from Transient Event store are iterable

**\$BENDERTUTORROOT/solutions/DataAccess.py**

- To know the available attributes:

```
help( obj )
```

```
help( type( obj ) )
```

```
dir ( ), dir ( cpp ), dir ( LoKi ), dir ( LHCb )
```

- ON-LINE help for ALL Python/Bender functions/classes, sometimes it is VERY useful
- Doxygen pages:

Broken, to be fixed ☹️

```
>>> import LoKiCore.doxygenurl as doxy
>>> o = ...
>>> doxy.browse ( o )
>>> doxy.browse ( 'LHCb::MCParticle' )
```



- Objects in python are “more rich”:
  - >>> `from Bender.MainMC import *`
  - >>> `help(LHCb.MCParticle)`
  - >>> `doxy.browser( LHCb.MCParticle )`
- Many new methods (added “on-flight” by Bender):
  - `child`, `children`, `parents`, `mother`, `ascendents`, `descendent`, `printDecay`, ...
- Uniform with `LHCb.Particle` & `HepMC.GenParticle`

```
p = ...
for c in p.children() :
    print c
c1 = p.child(1)
```

```
p = ...
for c in children( p ) :
    print c
c1 = child( p , 1)
```



- <Almost> all of LoKi's idioms are in Bender
  - (No need to write the separate manual)
  - The semantics is very similar
    - In spite of *DIFFERENT* languages
    - Few "obvious" exceptions
- In the game
  - All functors (functions/predicates) "cuts"
  - All (v,mc,mcv.g) select methods
  - Loops, plots
  - For nTuples the functionality is a bit limited
    - A lack of templated methods

# MCselect statement

- Selection of MCParticles which satisfy the certain criteria:

LUG, Tab. 13.4, p.84

```
mcmu = self.mcselect( 'mcmu' ,
                    'mu+' == MCABSID )
```

Select  $\mu^+$  &  $\mu^-$

```
beauty = self.mcselect('beauty' , BEAUTY )
```

Everything which has b or  $\bar{b}$

- Refine criteria:

```
muFromB = self.mcselect ( 'muFromC' ,
                        mcmu ,
                        FROMMCTREE( beauty ) )
```

Everything from  
"decay" trees  
(incl. decay-  
on-flight)

```
muPT = self.mcselect( 'withPT' ,
                    muFromB ,
                    ( MCPT > 1000 ) )
```

# Change the input data

- Get and configure EventSelector component:

```
evtSel = gaudi.evtSel()
```

```
evtSel = open( "file" )
```

Or

```
evtSel.open ( [ "file1", "file2" ] )
```



## Brilliant tool from Olivier Dormond

- find the MC-decay trees:

```
mc = self.mcFinder()
trees = mc.find(
    '[B_s0 -> (J/psi(1S) -> mu+ mu-) phi(1020)]cc' )
```

Container("Range") of  
MCParticles

- find MC-decay tree components:

```
phis = mc.find(
    'phi(1020) : [B_s0 -> (J/psi(1S) -> mu+ mu-) phi(1020)]cc' )
```

Container("Range") of  
MCParticles

- extract 'marked' MC-decay tree components:

```
mus = mc.find(
    '[B_s0 -> (J/psi(1S) -> mu+ ^mu-) phi(1020)]cc' )
```

[\\$BENDERTUTORROOT/solutions/MCTrees.py](#)

## Add simple histos:

```
for mu in mus:
    self.plot ( MCPT(mu) / 1000 ,
                'PT of muon from J/psi' , 0 , 10 )
```

- Configuration :

```
ApplicationMgr ( HistogramPersistency="ROOT" )
HistogramPersistencySvc ( OutputFile = "myhistos.root" )
```

```
gaudi.HistogramPersistency= "ROOT"
hsvc = gaudi.service("HistogramPersistencySvc" )
hsvc.OutputFile = "myhistos.root"
```

# Add the simple N-Tuple

```
tup = self.nTuple( 'My N-Tuple' )
zOrig = MCVXFUN( MCVZ )
for mu in mus :
    tup.column( 'PT' , MCPT ( mu ) )
    tup.column( 'P' , MCP ( mu ) )
    tup.column( 'Z' , zOrig ( mu ) )
    tup.write()
```

```
NTupleSvc ( Output = [
  "MC DATAFILE='tuples.root' TYP='ROOT' OPT='NEW' " ] )
```

- **Configuration:**

```
myAlg.NTupleLUN = 'MC'
```

```
ntsvc = gaudi.service( 'NTupleSvc' )
```

To be improved

```
ntsvc.Output =
```

```
[ "MC DATAFILE='tuples.root' TYP='ROOT' OPT='NEW' " ]
```

[\\$BENDERTUTORROOT/solutions/MCTrees.py](#)



```

#####
## @class MCTrees
## The algorithm itself
class MCTrees( AlgoMC ) :
    """
    The algorithm itself
    """
    ## the main analysis method
    def analyse( self ) :
        """
        The main analysis method
        """
        ## get the MCDecayFinder wrapper
        finder = self.mcFinder()

        ## find all MC trees of interest
        trees = finder.find(
            ' [B_s0 -> ( J/psi(1S) -> mu+ mu- ) phi(1020) ]cc' )

        ## get all kaons from the tree :
        phis = finder.find(
            ' [B_s0 -> ( J/psi(1S) -> mu+ mu- ) ^phi(1020) ]cc' )

        ## get marked particles from the tree:
        mus = finder.find(
            ' [B_s0 -> ( J/psi(1S) -> ^mu+ ^mu- ) phi(1020) ]cc' )

        print ' found MCTrees/Phis/Mus: %s/%s/%s' % ( trees.size () ,
                                                    phis.size () ,
                                                    mus.size () )

        ## fill the histogram
        for mu in mus :
            self.plot ( MCPT( mu ) / 1000 ,
                       ' PT of Muons from J/psi ' ,
                       0 , 10 )

        ## retrieve (bon-on-demand) N-Tuple
        tup = self.nTuple( 'My N-Tuple' )
        zOrig = MCVXFUN( MCVZ )

        for mu in mus :
            tup.column ( 'P' , MCP ( mu ) / 1000 )
            tup.column ( 'PT' , MCPT ( mu ) / 1000 )
            tup.column ( 'ID' , MCID ( mu ) )
            tup.column ( 'Q3' , MC3Q ( mu ) )
            tup.column ( 'ZOR' , zOrig ( mu ) )
            tup.write()

        return SUCCESS
#####

```

```

#####
## configure the job
def configure() :
    """
    Configure the job
    """

    from Configurables import DaVinci

    DaVinci (
        DataType = 'DC06' , # default
        Simulation = True ,
        HltType = '' )

    ## configure histograms & n-tuples
    from Gaudi.Configuration import NTupleSvc, HistogramPersistencySvc
    NTupleSvc ( Output = [ "MC DATAFILE='MCTrees.root' OPT='NEW' TYP='ROOT' ] )
    HistogramPersistencySvc ( OutputFile = 'MCTrees_histos.root' )

    ## get/create application manager
    gaudi = appMgr()

    # 1) create the algorithm
    alg = MCTrees( 'McTree' )

    # 2) replace the list of top level algorithm by only *THIS* algorithm
    gaudi.setAlgorithms ( [ alg ] )

    # configure my own algorithm
    alg.NTupleLUN = 'MC'
    alg.PP2MCs = []

    ## redefine input files
    evtSel = gaudi.evtSel()
    import LoKiExample.Bs2Jpsiphi_mm_data as data
    evtSel.open( data.Files )

    return SUCCESS
#####
#
#####
## Job steering
if __name__ == '__main__' :

    ## job configuration
    configure()

    ## event loop
    run(100)

#####
# The END
#####

```



- At this moment one knows how to
  - Deal with MC trees, decays, particles
  - Perform simple (python) loops
  - Deal with the histograms and N-tuples
    - Some knowledge of 'configuration'
  
- For RC-data one **must** perform **non-trivial** algorithm configuration to be able to run
  - Input for RC-particles (or ParticleMaker)
  - Dependency on 'other' algorithms

# Algorithm configuration

```
myAlg = MyAlg( 'MyAlg' )  
desktop = gaudi.property( 'MyAlg.PhysDesktop' )  
desktop.InputLocations = [  
    'Phys/StdLooseKaons' ,  
    'Phys/StdLooseMuons' ]
```

```
muons = self.select ( 'mu' ,
    ( 'mu+' == ABSID ) & ( PT > (1*GeV) ) )
kaons = self.select ( 'K' ,
    ( 'K+' == ABSID ) & ( PIDK > 0 ) )
```

- **Loops:**

```
psis=self.loop( 'mu mu' , 'J/psi(1S)')
phis=self.loop( 'K K' , 'phi(1020)')
```

`$BENDERTUTORROOT/solution/RCSelect.py`

# Inside the loops (I)

```

dmcut = ADMASS('J/psi(1S)') < 50
for psi in psis :
    if not 2500 < psi.mass(1,2) < 3500 : continue
    if not 0 == SUMQ( psi ) : continue
    if not 0 <= VCHI2( psi ) < 49 : continue
    self.plot ( M(psi)/1000 ,
                " di-muon invariant mass" ,
                2.5 , 3.5 )
    if not dmcut( psi ) : continue
    psi.save('psi')

```

$\Sigma q = 0$

$\chi^2_{\nu\chi} < 49$

$|\Delta M| < 50 \text{ MeV}/c^2$

```

psis = self.selected('psi')
print '# of selected J/psi candidates:', psis.size()

```

[\\$BENDERTUTORROOT/solution/RCSelect.py](#)



# Inside the loops (II)

```
dmcut = ADMASS('phi(1020') < 12
```

```
for phi in phis :
```

```
    if not phi.mass(1,2) < 1050      : continue
```

```
    if not 0 == SUMQ( phi )          : continue
```

$\Sigma q = 0$

```
    if not 0 <= VCHI2( phi ) < 49   : continue
```

$\chi^2_{\nu\chi} < 49$

```
    self.plot ( M( phi ) / 1000 ,
```

```
                " di-kaon invariant mass" ,
```

```
                1.0 , 1.050 )
```

```
    if not dmcut( phi ) : continue
```

$|\Delta M| < 12 \text{ MeV}/c^2$

```
    phi.save('phi')
```

```
phis = self.selected('phi')
```

```
print '# of selected phi candidates:', phis.size()
```

[\\$BENDERTUTORROOT/solutions/RCSelect.py](#)



# Inside the loops (III)

```
dmcut = ADMASS( 'B_s0' ) < 100
bs = self.loop ( 'psi phi' , 'B_s0' )
for B in bs :
    if not 4500 < B.mass(1,2) < 6500 : continue
    if not 0 <= VCHI2( B ) < 49 : continue
    self.plot ( M( B ) / GeV ,
                " J/psi phi invariant mass" ,
                5.0 , 6.0 )
    if not dmcut( B ) : continue
    B.save( 'Bs' )

Bs = self.selected( 'Bs' )
print `# of selected Bs candidates:`, Bs.size()
if not Bs.empty() : self.setFilterPassed ( TRUE )
```

[\\$BENDERTUTORROOT/solutions/RCSelect.py](#)

- The simplest case: check if RC particles originates form certain MC-(sub)tree:
  - The most frequent case
    - Check for efficiency
    - Resolution
- The opposite case ( RC  $\leftrightarrow$  MC )
  - similar `MCTRUTH`  $\leftrightarrow$  `RCTRUTH`

**NB:**

**LoKi ( and therefore Bender) uses own concept of MC loose matching**

Nice tool from Olivier Dormond

```
finder = self.mcFinder('some name')
```

- Select MC-particles

```
mcBs = finder.find(
    '[B_s0 -> (J/psi(1S) -> mu+ mu-) phi(1020)]cc ' )
mcPhi = finder.find(
    'phi(1020) : [B_s0 -> (J/psi(1S) -> mu+ mu-) phi(1020)]cc ' )
mcPsi = finder.find(
    'J/psi(1S) : [B_s0 -> (J/psi(1S) -> mu+ mu-) phi(1020)]cc ' )
```

- Prepare 'MC-Truth cuts'

```
match = self.mcTruth('some name')
mcCutBs = MCTRUTH ( match , mcBs )
mcCutPhi = MCTRUTH ( match , mcPhi )
mcCutPsi = MCTRUTH ( match , mcPsi )
```

`$BENDERTUTORROOT/solution/RCMCSelect.py`



```

for psi in psis :
    if not mcCutPsi ( psi ) : continue      ## use MC-truth predicate
...
for phi in phis :
    if not mcCutPhi ( phi ) : continue     ## use MC-truth predicate
...
for B in bs :
    if not mcCutBs ( B ) : continue        ## use MC-truth predicate

```

```

for B in Bs :
    psi = B ( 1 )      ## get the first daughter
    phi = B ( 2 )     ## get the second daughter
...
tup.column( 'mcpsi' , mcCutPsi ( psi ) )   ## use MC-truth predicate
tup.column( 'mcphi' , mcCutPhi ( phi ) )   ## use MC-truth predicate
tup.column( 'mc' , mcCutBs ( B ) )        ## use MC-truth predicate
tup.write ( )

```

```

=====
## @class RCMCSelect
# my analysis algorithm
class RCMCSelect(AlgoMC):
    """
    My analysis algorithm
    """
    ## the main analysis method
    def analyse( self ) :
        """
        The main analysis method
        """
        ## get MCDecayFinder wrapper:
        finder = self.mcFinder()
        ## find all MC trees
        mcBs = finder.find(
            '[ B_s0 -> ( J/psi(1S) -> mu+ mu- {,gamma} ) phi(1020)]cc' )
        ## get all MCphis from the tree :
        mcPhi = finder.find(
            '[ B_s0 -> ( J/psi(1S) -> mu+ mu- {,gamma} ) ^phi(1020)]cc' )
        ## get all MC psis from the tree :
        mcPsi = finder.find(
            '[ B_s0 -> ( ^J/psi(1S) -> mu+ mu- {,gamma} ) phi(1020)]cc' )
        ## get helper object for MC-match
        match = self.mcTruth( 'MCDecayMatch' )
        # prepare "Monte-Carlo Cuts"
        mcCutBs = MCTRUTH( match , mcBs )
        mcCutPhi = MCTRUTH( match , mcPhi )
        mcCutPsi = MCTRUTH( match , mcPsi )
        ## select muons for J/Psi reconstruction
        muons = self.select( "mu" , ( "mu+" == ABSID ) & ( PT > 500 ) )
        if muons.empty() : return SUCCESS
        ## select kaons for Phi reconstruction
        kaons = self.select( "K" , ( "K+" == ABSID ) & ( PIDK > 0.0 ) )
        if kaons.empty() : return SUCCESS
        ## delta mass cut for J/psi
        dmPsi = ADMASS('J/psi(1S)') < 50
        ## prepare the loop over dimuons
        psis = self.loop( 'mu mu' , 'J/psi(1S)' )
        for psi in psis :
            ## use *ONLY* Monte-Carlo cuts
            if not mcCutPsi( psi ) : continue ## ATTENTION! only true J/psi
            ## rough estimation of the mass
            mass = psi.mass(1,2) / 1000
            if not 2.5 < mass < 3.5 : continue
            ## neutral combination?
            if not 0 == SUMQ( psi ) : continue
            ## check the chi2 of the vertex fit
            if not 0 <= VCHI2( psi ) < 49 : continue
            self.plot( M(psi) / 1000 ,
                " dimuon invariant mass " ,
                2.5 , 3.5 )
            if not dmPsi( psi ) : continue
            psi.save( 'psi' )

```

```

## delta mass cut for phi
dmPhi = ADMASS('phi(1020)') < 20
## prepare the loop over dikaons
phis = self.loop( 'K K' , 'phi(1020)' )
for phi in phis :
    ## use *ONLY* Monte-Carlo cuts
    if not mcCutPhi( phi ) : continue ## ATTENTION: only true phi
    #
    if phi.mass( 1 , 2 ) > 1050 : continue
    # neutral combination ?
    if not 0 == SUMQ( phi ) : continue
    if not 0 <= VCHI2( phi ) < 49 : continue
    self.plot ( M(phi) / 1000 ,
        " dikaon invariant mass " ,
        1.0 , 1.050 )
    if not dmPhi( phi ) : continue
    phi.save('phi')
## delta mass cut for Bs
dmBs = ADMASS('B_s0') < 100
## prepare the loop over psi+phi combinations
bs = self.loop( 'psi phi' , 'B_s0' )
for B in bs :
    ## use *ONLY* Monte-Carlo cuts
    if not mcCutBs( B ) : continue ## ATTENTION: only true Bs
    #
    m = B.mass(1,2) / 1000
    if not 4.5 < m < 6.5 : continue
    if not 0 < VCHI2( B ) < 49 : continue
    self.plot ( M(B) / 1000 ,
        " psi phi invariant mass " ,
        5.0 , 6.0 )
    if not dmBs ( B ) : continue
    B.save('Bs')

Bs = self.selected('Bs')
if not Bs.empty() : self.setFilterPassed ( True ) ## FILTER PASSED

return SUCCESS
=====

```

```

# =====
## Job configuration:
def configure() :
    """
    Job configuration
    """

    from Configurables import DaVinci

    DaVinci (
        DataType = 'DC06'      , # default
        Simulation = True      ,
        HitType = '' )

    from Gaudi.Configuration import HistogramPersistencySvc
    HistogramPersistencySvc ( OutputFile = 'RCMCselect_histos.root' )

    ## get/create application manager
    gaudi = appMgr()

    # modify/update the configuration:
    # 1) create the algorithm
    alg = RCMCSelect( 'RCMCSelect' )
    # 2) add the algorithm
    #gaudi.addAlgorithm( alg )
    gaudi.setAlgorithms( [alg] )

    # 3) configure algorithm
    desktop = gaudi.tool('RCMCSelect.PhysDesktop')
    desktop.InputLocations = [
        'Phys/StdTightKaons' ,
        'Phys/StdTightMuons'
    ]

    ## configure the desktop:
    alg.PP2MCs = ['Relations/Rec/ProtoP/Charged']

    ## redefine input files
    evtSel = gaudi.evtSel()
    import LoKiExample.Bs2Jpsiphi_mm_data as data
    evtSel.open( data.Files )

    return SUCCESS
# =====

## Job steering:
if __name__ == '__main__' :

    ## job configuration
    configure()
    ## event loop
    run(2000)
# =====

```

- Algorithm: 81 lines
  - 55% - comments
- Configuration & steering: 44 lines
  - 40% comments
- Select true "reconstructed" Bs with loose cuts: fine for cuts investigation



- List the histograms:

```
import ROOT
```

```
hsvc = gaudi.histSvc()
```

```
hsvc.dump()
```

- Get the histogram by ID (==title):

```
h = hsvc[ 'Phi/K+K- mass ' ]
```

- Visualize the histogram as ROOT histo:

```
h.plot()
```

```
h.Draw()
```





- Nice printout of trees, particles, events
- Various "extractors" and metafunctions
- HepMC + HepMCParticleMaker
- Jets
- Tools for background origin studies
- Patterns
- and much much more...

As concerns the functionality needed for analysis, Bender is full scale application, widely used for many physics studies.

# Next (obvious) steps

- Integrate with DIRAC/Ganga
- Integrate with Panoramix
- More, more and more “on-flight” decorations:  
`p.name()` , `name(p)` , ...  
`p.printDecay()` , `printDecay(p)` , ...

# References again...

- Bender Pages and Bender pages by Lena Mayatskaya
- Bender mailing list
- Bender Savannah portal (news, bugs, tasks, ...)
- Bender Tutorial TWiki
- Regular informal "hands-on" tutorials
  - You know whom to ask...
- Bender Examples
  - including nice scripts from Diego for  $B_s \rightarrow \mu\mu$  background studies
- "Bender-helpdesk@lhcb.cern.ch"
  - 13-R-018 at CERN
  - +41 (0) 22 767 40 24