```
1  /**********************************************************************************
2   * Package: GSM                                                                   *
3   * Class  : RunningAlphaQCD                                                       *
4   *                                                                                *
5   * Description:                                                                   *
6   *          Implementation                                                        *
7   *                                                                                *
8   * Source:                                                                        *
9   *      - G. Rodrigo, A. Pich, A. Santamaria,                                     *
10  *                Phys. Lett. B424:367-374, 1998, hep-ph/9707474                  *
11  *      - G. M. Prosperi, M. Raciti, C. Simolo,                                   *
12  *                Prog. Part. Nucl. Phys. 58:387-438, 2007, hep-ph/0607209        *
13  *      - B. A. Magradze, Few Body Syst. 40 (2006) 71-99, hep-ph/0512374.         *
14  **********************************************************************************
15  #include "TMath.h"
16  #include "Riostream.h"
17
18  #include "Gfitter/GMath.h"
19  #include "Gfitter/GParameterRef.h"
20  #include "Gfitter/GTheoryRef.h"
21  #include "Gfitter/GVariable.h"
22  #include "Gfitter/GStore.h"
23  #include "Gfitter/GInterval.h"
24  #include "Gfitter/GRungeKutta.h"
25  #include "Gfitter/GRootFinder.h"
26
27  #include "GSM/RunningAlphaQCD.h"
28  #include "GSM/AlphaQCDAtQ.h"
29
30  using namespace Gfitter;
31
32  ClassImp(GSM::RunningAlphaQCD)
33
34  GSM::RunningAlphaQCD:: RunningAlphaQCD()
35    : Gfitter::GAuxTheory(),
```

Hinweis:

Kommentare mit Hinweisen auf ZFitter sind grün markiert

Übereinstimmungen sind gelb markiert.

Auffällige Stellen bzw. Anmerkungen sind violett markiert

Anhang 13

zum Gutachten DESY ZFitter_GFitter vom 17.03.2014

Lübeck, den 17. März 2014

Dr. Ulrich Obermöller
-Sachverständiger-

Von der Handelskammer Hamburg
Dr. Ulrich Obermöller
Sachverständiger für Systeme und Anwendungen der Informationsverarbeitung, insbesondere Individualsoftware und Software-Qualitätssicherung
öffentlich bestellt und vereidigt

```
36      m_isUpToDate_Update( kFALSE )
37    {
38      const TString& AlphasType = gStore()->GetVariable( "GSMFlags::OptionQCD" )->GetStringValue();
39      m_logger << kINFO << "Evaluate running alpha_s(s) with: \"" << AlphasType << "\" method" << GEndl;
40
41      m_thisPointer = this;
42
43      SetTheoryName( GetName() );
44      SetExistDerivative( kFALSE );
45
46      if (AlphasType == "FitLambda" ){
47        BookParameter( "lambdaMS5", &p_InputValueLambdaMS5 );
48        m_QCDType  = kFitLambda;
49      }
50      else if (AlphasType == "FitAlphas"){
51        BookParameter("alphasMZ", &p_InputValueAlphasMZ );
52        m_QCDType  = kFitAlphas;
53      }
54      else if (AlphasType == "RungeKutta"){
55        m_logger << kINFO << "Booking alphasMZ" << GEndl;
56        BookParameter("alphasMZ", &p_InputValueAlphasMZ );
57        m_QCDType  = kRungeKutta;
58      }
59      else {
60        m_logger << kFATAL << "unknown value for \"GSMFlags::OptionQCD\": \""
61                 << AlphasType << "\"" << ". Possible are: \"FitLambda\", \"FitAlphas\", \"RungeKutta\""
62                 << GEndl;
63      }
64
65      BookParameter( "MZ"       , &p_MZ );
66      BookParameter( "mu_MSb"   , &p_mu );
67      BookParameter( "md_MSb"   , &p_md );
68      BookParameter( "ms_MSb"   , &p_ms );
69      BookParameter( "mc_MSb"   , &p_mc );
70      BookParameter( "mb_MSb"   , &p_mb );
```

```
71      BookParameter( "mt"        , &p_mt );
72      BookParameter( "Scale"     , &p_Scale );
73    }
74
75    void GSM::RunningAlphaQCD::UpdateLocalFlags( GReference& /* ref */ )
76    {}
77
78    void GSM::RunningAlphaQCD::Update()
79    {
80
81      if (m_isUpToDate_Update) return;
82
83    }
84
85    void GSM::RunningAlphaQCD::Initialise()
86    {
87
88      // init coefficients for alphas matching
89      // hep-ph/9707474 (9)
90      m_c20 = -11/72.0;
91      for (Int_t nf = 0; nf < kNfmax; nf++) {
92        m_c30[nf] = ( 82043/27648.0*GMath::Zeta3()
93                    - 575263/124416.0 + 2633/31104.0*nf );
94      }
95
96    //    m_c20 = 7/24.0;
97    //    for (Int_t nf = 0; nf < kNfmax; nf++) {
98    //      m_c30[nf] = ( 80507/27648.0*GMath::Zeta3()
99    //                  + 1/9.0*GMath::Zeta2()*(2.0*TMath::Log(2) + 7.0)
100   //                  + 68849/124416.0 - nf/9.0*(GMath::Zeta2() + 2479/3456.0 ) );
101   //    }
102
103
104
105     Double_t pi = TMath::Pi();
```

Match 1

2.1.2.2

dizet6_42.f

```
5459        ALG=2D0*LOG(AMQ/ALNFN)
5460        ALGALG=LOG(ALG)
5461  *
5462        Bc2=-7D0/24D0
5463        Bc3=-80507D0/27648D0*D3-2D0/3D0*(1D0/3D0*LOG(2D0)+1D0)*D2
5464     &    -58933D0/124416D0+1D0/9D0*(D2+2479D0/3456D0)*4D0
5465  *
```

Match 2

4,1,3.2

```
106
107     // init "beta" functions
108     // from (16) of hep-ph/0607209
109     for (Int_t nf = 0; nf < kNfmax; nf++) {
110       m_b0[nf] = 1/(4.0*pi)*(11.0 - 2/3.0*nf);
111       m_b1[nf] = 1/GMath::IPow(4.0*pi,2)*(102.0 - 38/3.0*nf);
112       m_b2[nf] = 1/GMath::IPow(4.0*pi,3)*(2857/2.0 - 5033/18.0*nf + 325/54.0*nf*nf);
113       m_b3[nf] = ( 1/GMath::IPow(4.0*pi,4)*
114                    ( (149753/6.0 + 3564.0*GMath::Zeta3())
115                      - (1078361/162.0 + 6508/27.0*GMath::Zeta3())*nf
116                      + (50065/162.0 + 6472/81.0*GMath::Zeta3())*nf*nf
117                      + 1093/729.0*nf*nf*nf ) );
118     }
119   }
120
121   // switching between methods
122   // here: Fit methods and Runge-Kutta integration
123   Double_t GSM::RunningAlphaQCD::EvolveAlphas( Double_t mu )
124   {
125     if ((mu == p_MZ)) {
126       if (GetQCDType() == kFitAlphas || GetQCDType() == kRungeKutta ) {
127         // in this case just return the parameter (ie. a_s(M_z) from data card)
128         return p_InputValueAlphasMZ;
129       }
130       else {
131         return AlphaStrong( 5, mu, p_InputValueLambdaMS5 );
132       }
133     }
134
135     Double_t asnew  = 0;
136     Int_t    nf     = 0;
137
138     //calculate number of active flavours
139
140     if (mu < 1 || TMath::IsNaN(mu) ){
```

dizet6_42.f

```
5333        FUNCTION ZALSFL(A)
5334        IMPLICIT REAL*8(A-H,O-Z)
5335        COMMON /CDZZER/UPI,ALPHAQ,ANF,AKC,AKB
5336        COMMON /CDZCON/PI,PI2,F1,D3,ALFAI,AL4PI,AL2PI,AL1PI
5337        B=LOG(A)
5338        B0=1D0/4D0 *(11D0-2D0/3D0*ANF)
5339        B1=1D0/16D0*(102D0-38D0/3D0*ANF)/B0
5340        B2=1D0/64D0*(2857D0/2D0-5033D0/18D0*ANF+325D0/54D0*ANF**2)/B0
5341        B3=1D0/256D0*(149753D0/6D0+3564D0*D3
5342       &  -(1078361D0/162D0+6508D0/27D0*D3)*ANF
5343       &  +(50065D0/162D0+6472D0/81D0*D3)*ANF**2+1093D0/729D0*ANF**3)/B0
5344   *
5345        ZALSFL=ALPHAQ-PI*(1D0/(B0*A)
5346       &                 -1D0/(B0*A)**2* B1*B
5347       &                 +1D0/(B0*A)**3*(B1**2*(B**2-B-1D0)+B2))
5348       &      -1D0/(B0*A)**4*(B1**3*(B**3-2.5D0*B**2-2D0*B+.5D0)
5349       &                      +3D0*B1*B2*B-.5D0*B3)
5350   *
5351        END
```

```
141         m_logger << kFATAL << " <EvolveAlphas>: Energy-Scale is too low ( scale > 1 GeV ) or NaN. Scale = "
142                  << mu << GEndl;
143       }
144       else    nf = 3;
145
146       if (mu >= p_mc) nf = 4;
147       if (mu >= p_mb) nf = 5;
148       if (mu >= p_mt) nf = 6;
149
150       //m_logger << kINFO << "nf = " << nf << GEndl;
151
152       if( GetQCDType() == kFitAlphas || GetQCDType() == kFitLambda ) {
153         Double_t lambda = 0;
154         switch(nf) {
155         case 3:
156           if (TMath::IsNaN(p_mc)) m_logger << kFATAL << "<RunningAlphaQCD::EvolveAlphas> p_mc is NaN !" << GEndl;
157           asnew  = EvolveAlphas( p_mc );
158           asnew  = AlphasMatchDown( 4, asnew, p_Scale*p_mc, p_mc );
159           lambda = LambdaAtThreshold ( 3, asnew, p_Scale*p_ms );
160           asnew  = AlphaStrong ( 3, mu, lambda );
161           break;
162         case 4:
163           asnew  = AlphaStrong( 5, p_Scale*p_mb, GetLambdaMS5() );
164           asnew  = AlphasMatchDown( 5, asnew, p_Scale*p_mb, p_mb );
165           lambda = LambdaAtThreshold   (    4, asnew, p_Scale*p_mb );
166           asnew  = AlphaStrong( 4, mu, lambda );
167           break;
168         case 5:
169           asnew  = AlphaStrong( 5, mu, GetLambdaMS5() );
170           //m_logger << kINFO << "case5fit: " << asnew << " at " << mu << GEndl;
171           break;
172         case 6:
173           asnew  = AlphaStrong(  5, p_Scale*p_mt, GetLambdaMS5() );
174           asnew  = AlphasMatchUp( 6, asnew, p_Scale*p_mt, p_mt );
175           lambda = LambdaAtThreshold   (  6, asnew, p_Scale*p_mt );
```

```
176           asnew  = AlphaStrong( 6, mu, lambda );
177           break;
178         }
179       }
180       else if( GetQCDType() == kRungeKutta ) {
181         if (nf == 5) {
182           GRungeKutta rungekutta5( &GetRGE5 );
183           asnew = rungekutta5.GetYEndValue( 2.0*TMath::Log( p_MZ ), p_InputValueAlphasMZ,
184                                             2.0*TMath::Log( mu ) );
185           //        m_logger << kINFO << " z: "<< p_MZ
186           //       << " inp: " << p_InputValueAlphasMZ
187           //       << " mu: " << mu
188           //       << " case5kut: " << asnew << GEndl;
189         }
190         else if( nf == 4 ) {
191           if (TMath::IsNaN(p_mb)) m_logger << kFATAL << "<RunningAlphaQCD::EvolveAlphas> p_mb is NaN !" << GEndl;
192           asnew = EvolveAlphas( p_mb );
193           asnew = AlphasMatchDown( 5, asnew, p_Scale*p_mb, p_mb );
194           GRungeKutta rungekutta4( &GetRGE4 );
195           asnew = rungekutta4.GetYEndValue( 2.0*TMath::Log( p_mb ), asnew,
196                                             2.0*TMath::Log( mu ) );
197         }
198         else if( nf == 3 ) {
199           if (TMath::IsNaN(p_mc)) m_logger << kFATAL << "<RunningAlphaQCD::EvolveAlphas> p_mc(s) is NaN !" << GEndl;
200           asnew  = EvolveAlphas( p_mc );
201           asnew  = AlphasMatchDown( 4, asnew, p_Scale*p_mc, p_mc );
202           GRungeKutta rungekutta3( &GetRGE3 );
203           asnew = rungekutta3.GetYEndValue( 2.0*TMath::Log( p_ms ), asnew,
204                                             2.0*TMath::Log( mu ) );
205         }
206         else if (nf == 6 ) {
207           GRungeKutta rungekutta5( &GetRGE5 );
208           asnew = rungekutta5.GetYEndValue( 2.0*TMath::Log( p_MZ ), p_InputValueAlphasMZ,
209                                             2.0*TMath::Log( p_mt ) );
210           asnew  = AlphasMatchUp( 6, asnew, p_Scale*p_mt, p_mt );
```

```
211             GRungeKutta rungekutta6( &GetRGE6 );
212             asnew = rungekutta6.GetYEndValue( 2.0*TMath::Log( p_mt ), asnew,
213                                               2.0*TMath::Log( mu ) );
214           }
215         }
216
217         //m_logger << kINFO << " ret val: " << asnew << GEndl;
218
219         return asnew;
220       }
221
222       // Evolve runnind alphas, see (31) of hep-ph/0607209
223       Double_t GSM::RunningAlphaQCD::AlphaStrong(Int_t nf, Double_t mu, Double_t lambda) const
224       {
225         Double_t L    = 2.0*( TMath::Log( mu/lambda ) );
226         Double_t logL = TMath::Log( L );
227         Double_t alphas = ( 1/(m_b0[nf]*L)*
228                           ( 1.0 - m_b1[nf]*logL/(m_b0[nf]*m_b0[nf]*L) + 1/(m_b0[nf]*m_b0[nf]*L*L)*
229                             ( GMath::IPow(m_b1[nf]/m_b0[nf],2)
230                               *( GMath::IPow(logL,2) - logL -1.0 ) + m_b2[nf]/m_b0[nf] )
231                             + 1/GMath::IPow(m_b0[nf]*L,3)*
232                             ( GMath::IPow(m_b1[nf]/m_b0[nf],3)
233                               *( - GMath::IPow(logL,3) +5/2.0*logL*logL + 2.0*logL - 0.5 )
234                               - 3.0*m_b1[nf]*m_b2[nf]*logL/(m_b0[nf]*m_b0[nf])
235                               + m_b3[nf]/(2.0*m_b0[nf]) ) ) );
236         return alphas;
237       }
238
239       // matching condition for decreasing nf
240       // inverted eq. of hep-ph/9707474, see (4) and (6)
241       // nf belongs to the old alphas
242       Double_t GSM::RunningAlphaQCD::AlphasMatchDown(Int_t nf, Double_t alphas, Double_t mu_nf, Double_t qmass) const
243       {
244         Double_t x  = 2.0*TMath::Log( mu_nf/qmass );
245
```

```
246     Double_t C1 = x/6.0;
247     Double_t C2 = m_c20 + 19/24.0*x + x*x/36.0;
248     Double_t C3 = ( m_c30[nf] + (241/54.0 + 13/4.0*m_c20 - (325/1728.0 + m_c20/6.0)*nf)*x
249                     + 511/576.0*x*x + x*x*x/216.0 );
250
251     Double_t A1 = -C1;
252     Double_t A2 = 2*C1*C1 - C2;
253     Double_t A3 = 3.0*C1*C1*C1 + C2*C1 - C3;
254     Double_t as = alphas/TMath::Pi();
255
256     Double_t alphasnew = alphas*(1.0 + A1*as + A2*as*as + A3*as*as*as);
257
258     return alphasnew;
259   }
260
261   // matching condition for increasing nf
262   // eq. of hep-ph/9707474, see (4) and (6)
263   // nf belongs to the new alphas
264   Double_t GSM::RunningAlphaQCD::AlphasMatchUp(Int_t nf, Double_t alphas, Double_t mu_nf, Double_t qmass) const
265   {
266     Double_t x  = 2.0*TMath::Log( mu_nf/qmass );
267
268     Double_t C1 = x/6.0;
269     Double_t C2 = m_c20 + 19/24.0*x + x*x/36.0;
270     Double_t C3 = ( m_c30[nf] + (241/54.0 + 13/4.0*m_c20 - (325/1728.0 + m_c20/6.0)*nf)*x
271                     + 511/576.0*x*x +x*x*x/216.0 );
272
273     Double_t as = alphas/TMath::Pi();
274
275     Double_t alphasnew = alphas*(1 + C1*as + C2*as*as + C3*as*as*as);
276
277     return alphasnew;
278   }
279
280   // computes new Lambda for different nf
```

```
281  Double_t GSM::RunningAlphaQCD::LambdaAtThreshold( Int_t nf, Double_t alphas, Double_t mu) const
282  {
283    Double_t pi   = TMath::Pi();
284    Double_t b0   = m_b0[nf]*pi;
285    Double_t b1   = m_b1[nf]*pi*pi/b0;
286    Double_t b2   = m_b2[nf]*pi*pi*pi/b0;
287    Double_t b3   = m_b3[nf]*pi*pi*pi*pi/b0;
288    Double_t C    = b1/b0*TMath::Log(b0);
289
290    Double_t asPi = alphas/pi;
291
292    Double_t T = 1/b0*( 1.0/asPi + b1*TMath::Log(asPi) + (b2 - b1*b1)*asPi
293                    + (b3/2.0 - b1*b2 + b1*b1*b1/2.0)*asPi*asPi ) + C;
294
295    return mu*TMath::Sqrt(TMath::Exp(-T));
296  }
297
298  // thisPoiter = NULL
299  GSM::RunningAlphaQCD* GSM::RunningAlphaQCD::m_thisPointer = NULL;
300
301  // static functions for Runge-Kutta method
302  Double_t GSM::RunningAlphaQCD::GetRGE1( Double_t x, Double_t y )
303  {
304    return GetThisPointer()->RGE( 1, x, y );
305  }
306
307  Double_t GSM::RunningAlphaQCD::GetRGE2( Double_t x, Double_t y )
308  {
309    return GetThisPointer()->RGE( 2, x, y );
310  }
311
312  Double_t GSM::RunningAlphaQCD::GetRGE3( Double_t x, Double_t y )
313  {
314    return GetThisPointer()->RGE( 3, x, y );
315  }
```

```
316
317   Double_t GSM::RunningAlphaQCD::GetRGE4( Double_t x, Double_t y )
318   {
319     return GetThisPointer()->RGE( 4, x, y );
320   }
321
322   Double_t GSM::RunningAlphaQCD::GetRGE5( Double_t x, Double_t y )
323   {
324     return GetThisPointer()->RGE( 5, x, y );
325   }
326
327   Double_t GSM::RunningAlphaQCD::GetRGE6( Double_t x, Double_t y )
328   {
329     return GetThisPointer()->RGE( 6, x, y );
330   }
331
332   // RGE
333   // eq. (A.1) of  hep-ph/0512374
334   Double_t GSM::RunningAlphaQCD::RGE( Int_t nf, Double_t /* x */, Double_t y )
335   {
336     return -y*y*( m_b0[nf] + m_b1[nf]*y + m_b2[nf]*y*y + m_b3[nf]*y*y*y );
337   }
338
339   Double_t GSM::RunningAlphaQCD::GetLambdaMS5()
340   {
341     // for kFitLambda : return parameter from data card
342     if      ( GetQCDType() == kFitLambda ) return p_InputValueLambdaMS5;
343
344     // for kFitAlphas : numerical calculation of lambdaMS5
345     else if ( GetQCDType() == kFitAlphas ){
346       GRootFinder rootFinder( &RootAlphas, 0.05, 0.5 );
347       return rootFinder.Root( p_InputValueAlphasMZ );
348     }
349     else{
350       m_logger << kFATAL << "unknown value for \"p_QCDType\": \"" << GetQCDType() << "\""
```

```
351                 << GEndl;
352         return 0;
353       }
354     }
355
356     // static function for root finding
357     Double_t GSM::RunningAlphaQCD::RootAlphas( Double_t lambda )
358     {
359       return GetThisPointer()->AlphasAtMZ( lambda );
360     }
361
362     // Get EvolveAlphas and initialise non static member values
363     // particulary p_MZ
364     Double_t GSM::RunningAlphaQCD::AlphasAtMZ( Double_t lambda )
365     {
366       return AlphaStrong( 5, p_MZ, lambda );
367     }
```