Status for generalised PDF parametrisation in xFitter

Andrey Sapronov, SANC, JINR

xFitter workshop in Dubna, 20 Feb 2016

```
&Parametrisation
  InputPDFs = 'uval dval usea dsea str glu'

  ! Flavor labels: 'bbar  cbar  sbar  ubar  dbar g  d  u  s  c  b'
  ! Only linear combinations, with alphanumeric coefficients
  ! No brackets.
  ! Every input pdf can be only once in each composition (i.e. no 'uval+uval'
  ! and alike).
  Composition  = 'u=uval+usea',
                 'd=dval+dsea',
                 'ubar=usea',
                 'dbar=dsea',
                 'sbar=0.5*str'
                 's=0.5*str'
                 'g=glu'
```

```
! The parameter definition is similar to the fortran MINUIT input.
! Variable (fitted) parameters have single '=' sign followed by 2 or 4
! numbers. In both cases first is the starting value and second is the step.
! The limits, if supplied, are the last two of four numbers given in square
! brackets []. Either both or none limits should be given.
! Fixed parameters are set with double '==' sign. They can be expressed via
! other parameters defined earlier (!) with a numeric coefficient.
! The step value and [] brackets after '==' sign will be ignored.
! Parameters defined via sumrules are only listed (no '=' sign)
Parameters    =
                'Buv =  0.5 0.001 [ 0.1  20.]',
                'Cuv =  0.3 0.01',
                'Duv =  0.5 0.005 [ 0.4  20.]',
                'Euv == 0.0 0.001 [-0.1   1.]',
                'Auv',
                'Adv',
                'Ag',
                'Bdv == Buv',
                'Cdv == 0.9*Cuv',
                'Ddv == 42.'
```

```
Parametrisation = 'uval = Auv * x^Buv * (1-x)^Cuv * (1 + Fuv*x^0.5 + Duv*x + Euv*x*x)',
                  'dval = Adv * x^Bdv * (1-x)^Cdv * (1 + Fdv*x^0.5 + Ddv*x + Edv*x*x)',
                  'usea = AUs * x^BUs * (1-x)^CUs * (1 + FUs*x^0.5 + DUs*x + EUs*x*x)',
                  'dsea = ADs * x^BDs * (1-x)^CDs * (1 + FDs*x^0.5 + DDs*x + EDs*x*x)',
                  'str  = Rs*(dsea+dval)',
                  'glu  = Ag  * x^Bg  * (1-x)^Cg  * (1 +  Fg*x^0.5 +  Dg*x +  Eg*x*x)'
```

```
template <typename Tval>
class AlgExpr
{
  AlgExpr(){ _ndim = 0 };
  ~AlgExpr(){};

 public:
  void Evaluate(Tval &result)
+--- 44 lines: {-----------------------------------------

 private:
  int assignTokens(const string &expr)
+--- 84 lines: {-----------------------------------------

  int convertToRPN()
+--- 35 lines: {-----------------------------------------
```

```
/**
  @class PDFParametrization

  @brief Class manages PDF parametrization

  This class takes care of PDF parametrization. It contains list of algebraic
  expressions for it, with corresponding parameters and sum rules. It is a
  singleton class to be created only once and called by getInstance() method
  afterwards.

  @author A.Sapronov <sapronov@ifh.de>

  @version 0.1
  @date 2015/08/25
  */


class PDFParametrization
{
 public:
  static PDFParametrization &getInstance(){
    static PDFParametrization paramInstance;
    return paramInstance;
  }
```

```
/**
  @class PDFComposition

  @brief Class manages PDF composition

  The class creates composition of flavor labels basing on input pdfs. The
  internal representation is nfl x npdf matrix, which transforms pdfs to
  flavors when required. The composition has to be given as a linear function of pdfs
  with numeric coefficients.

  @author A.Sapronov <sapronov@ifh.de>

  @version 0.1
  @date 2015/08/26
  */

class PDFComposition
{
 public:
  PDFComposition(const vector<string> &basis, const vector<string> &compn);
```

```
/**
  @class PDFParameters

  @brief Class manages PDF parameters

  This class keeps list of PDF parameters given in the config file. It
  destingueshes variable, fixed, derived and defined by the sum rule
  parameters. The derived parameter is equal to some other, variable or fixed.

  The class generates MINUIT data file minuit.in.txt with a list of fitted
  parameters containing their name, initial value, step and limits. During fit
  it gets the parameter values from MINUIT, calculates dependencies and
  substitutes it to the parametrization expressions.

  @author A.Sapronov <sapronov@ifh.de>

  @version 0.1
  @date 2015/08/27
  */

class PDFParameters
{
 public:
  PDFParameters();
  ~PDFParameters();
```

```
typedef struct {
  int type; // 0 - variable, 1 - fixed, 2 - derived, 3 - sum rule
  string name;
  double value; // parameter value
  double step; // parameter fit step
  double lo; // parameter limits
  double hi;
  double coeff; // numeric coefficient for derived par
  tParameter* der; // parameter to derive from
} tParameter;
```

TODO:

- Finish parametrisation
- Sumrules (some ideas)
- MINUIT interface
- + output