Zero-mass Structure Function Package

ZMSTF version 2021-08-04

M. Botje*
Nikhef, Science Park 105, 1098XG Amsterdam, the Netherlands

September 14, 2021

Abstract

The ZMSTF package is a QCDNUM add-on that computes zero-mass unpolarised structure functions up to NNLO in the strong coupling constant.

*email m.botje@nikhef.nl

Contents

1 Introduction **3**

2 Subroutine calls **3**

3 Performance **6**

A List of FORTRAN routines and C++ prototypes **7**

1 Introduction

The ZMSTF package is a QCDNUM add-on with routines that calculate up to NNLO the structure functions F_2, F_L, xF_3 and F'_L in un-polarised deep-inelastic scattering. To estimate scale errors it is possible to vary the renormalisation scale μ_R^2 with respect to the factorisation scale μ_F^2 (QCDNUM routine) and the Q^2 scale with respect to μ_F^2 (ZMSTF routine), but not both at the same time. For the formalism underlying the computation of structure functions and their scale variations we refer to the QCDNUM write-up. There you can also find the definition of F'_L (elevated perturbative order of F_L).

The ZMSTF package is written in FORTRAN77 but interfaces are provided so that all FORTRAN routines can be called from a C++ program.

C++ The C++ wrappers reside in the namespace `QCDNUM` and the routine names are written in lower case, as is the QCDNUM convention. We refer to the QCDNUM manual for more on C++ interfaces.

Floating-point arguments are in double precision and input numbers must, in FORTRAN, be given in double precision format like `2.5D0` instead of `2.5`. In C++ the input format is free since the data-type is specified in the function prototype and the conversion is done automatically, if necessary.

Weight tables are stored in the ZMSTF memory (an internal array). The memory size is specified by the parameter `nzmstor` in the file `zmstf.inc`; if you run out of space (error message) then you must set `nzmstor` as needed, and recompile ZMSTF.

The ZMSTF code is based on the QCDNUM toolbox and error messages are, in most cases, issued by the toolbox routines and not by ZMSTF itself. However, the calling ZMSTF routine is mentioned in the error message so that you know where it came from.

The call `ivers = IZMVERS()` gives you the current ZMSTF version number.

2 Subroutine calls

In this section we describe all available ZMSTF routines. Output arguments are pre-fixed by an ampersand (`&`), like in C++. As mentioned above, all routines have C++ wrappers with the same routine name (in lower case) and argument list as in FORTRAN:

```
call SUB(arguments)     →     QCDNUM::sub(arguments)
```

The C++ prototypes are listed in Appendix A.

```
call ZMWORDS ( &ntotal, &nused )
```

`ntotal` Number of words available in the ZMSTF workspace (`nzmstor` in `zmstf.inc`).

`nused` Number of words used (set to `0` before the call to `zmfillw` or `zmreadw`).

```
call ZMFILLW ( &nused )
```

Fill the weight tables of F_L, F_2, xF_3 and F'_L. The tables are calculated for all flavours $3 \leq n_f \leq 6$ and for all orders LO, NLO, NNLO. On exit, the number of words occupied by the workspace is returned in `nused`. If you get an error message that the internal workspace is too small to contain the weight tables, you should increase the value of the parameter `nzmstor` in the include file `zmstf.inc` and recompile ZMSTF.

Weight tables should be filled after an x-μ^2 grid is defined in QCDNUM and before a structure function is computed. The splitting function weight tables are needed as an input so that also the QCDNUM routines `fillwt`, `readwt` or `wtfile` must have been called before (error message if not).

```
call ZMDUMPW ( lun, 'filename' )
```

Dump the weights in memory via logical unit number `lun` to a disk file. The dump is un-formatted so that the weight file cannot be exchanged across machines.

```
call ZMREADW ( lun, 'filename', &nused, &ierr )
```

Read weights from a disk file via logical unit number `lun`. On exit, `nused` contains the number of words read into the workspace (fatal error if not enough space, see above) and the flag `ierr` is set as follows.

`0`	Weights are successfully read in.
`1`	Read error or input file does not exist.
`2`	Incompatible QCDNUM version.
`3`	Incompatible ZMSTF version.
`4`	Incompatible x-μ^2 grid definition.

These errors will not generate a program abort so that one should check the value of `ierr`, and take the appropriate action if it is non-zero.

```
call ZMWFILE ( 'filename' )
```

This routine reads a weight file from disk and if that fails (see `zmreadw` above) it computes the weights from scratch and dumps them on the file.

```
call ZMDEFQ2 ( a, b )
```

Define the relation between the factorisation scale μ_F^2 and Q^2

$$Q^2 = a\mu_F^2 + b.$$

Default `a` $= 1$ and `b` $= 0$. The ranges are limited to `0.1` $\leq$ `a` $\leq$ `10` and `-100` $\leq$ `b` $\leq$ `100`. A call to `zmabval(a,b)` reads the coefficients back from memory. To convert between the scales use:

```
Q2   = zmqfrmu(qmu2)          qmu2 = zmufrmq(Q2)
```

The Q^2 scale can only be varied when the renormalisation and factorisation scales are set equal in QCDNUM. So you can vary the Q^2 scale or the μ_R^2 scale but not both.

```
call ZSWITCH ( iset )
```

By default, the structure functions are calculated from the QCDNUM pdf-set `iset` = 1. With this routine you can switch to another set, provided that it contains unpolarised pdfs. For imported pdf-sets you have to make sure of that yourself.

It is also possible to switch pdf-sets via the `istf` argument in one of the structure function routines `zmstfij` or `zmstfun`, see below.

```
stf = ZMSTFIJ ( istf, def, ix, iq, ichk )
```

Calculate a structure function at a grid point (`ix`,`iq`) for a linear combination of parton densities. A linear combination is specified in the input array `def(-6:6)` in FORTRAN or `def[13]` in C++. In these arrays the quark flavours are indexed as follows.

	$\bar{t}$	$\bar{b}$	$\bar{c}$	$\bar{s}$	$\bar{u}$	$\bar{d}$	g	d	u	s	c	b	t
C++	0	1	2	3	4	5	6	7	8	9	10	11	12
FORTRAN	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6

`istf` Structure function index (1,2,3,4) = (F_L, F_2, xF_3, F'_L).

`def` Array with the coefficients of the (anti-)quark linear combination. The value of `def(0)`, or `def[6]` in C++, refers to the gluon and is ignored.

`ix`, `iq` Grid-point indices.

`ichk` If `ix` or `iq` are outside the grid boundaries or below the QCD scale Λ^2 then either return a `null` value (`ichk` = 0) or give an error message (`ichk` = 1).

It is not possible to vary the Q^2 scale (error message if $Q^2 \neq \mu^2$).

By default, the structure function is computed for pdf-set 1, or for the set selected by `zswitch`. Alternatively you can encode the pdf-set index in `istf`:

```
istf   →   10*iset + istf
```

```
call ZMSTFUN ( istf, def, x, Q2, &f, n, ichk )
```

Compute a structure function for a list of x and Q^2 points. Here you cannot vary both the μ_R^2 and Q^2 scales at the same time. The arguments are as for `zmstfij` except:

`x`, `Q2` Input arrays containing a list of x and Q^2 (not μ^2) values.

`f` Output array containing the list of structure functions.

`n` Number of items in `x`, `Q2` and `f`.

To calculate a structure function for more than one interpolation point, it is recommended to not execute `zmstfun` in a loop but to pass the entire list of interpolation points in a single call. The loop is then internally optimised for much greater speed.

Another way to calculate structure functions is by calling the routine `zmslowf`, with the same argument list as `zmstfun`. This routine was used for prototyping and runs quite slow but provides the possibility to calculate the quark and gluon contributions separately, order by order. This is achieved by setting `ichk` to one of the values given below; a positive (negative) value switches the boundary check on (off).

Contribution	LO	NLO	NNLO
Quark and gluon	±101	±102	±103
Quark only	±201	±202	±203
Gluon only	±301	±302	±303

3 Performance

We computed the proton F_2 structure function at NNLO on a set of randomly chosen grid-points and, to see the effect of interpolation, also on a random set of x-Q^2 points. The F_2 were calculated either in a loop over n points or, in case of `zmstfun`, for the list of points. The table shows for a few running modes the CPU time needed per F_2.

It is seen that for a loop calculation `zmstfij` is about a factor of 3 faster than `zmstfun`. For the latter routine it does not matter much whether the structure function is interpolated or not, indicating that the overhead in `zmstfun` dominates the cost in CPU.
This overhead is largely eliminated when processing a list of structure functions: for small n, list processing by `zmstfun` is about 10 times faster than a loop computation and this increases to a factor of 100 when n becomes large.

Routine	Mode		n	t/n [ms]
`ZMSTFIJ`	on-grid	loop	10	0.117
`ZMSTFUN`	on-grid	loop	10	0.417
	off-grid	loop	10	0.437
`ZMSTFUN`	on-grid	list	10	0.050
			100	0.009
			1000	0.003
`ZMSTFUN`	off-grid	list	10	0.065
			100	0.014
			1000	0.004
`SPLINE`	off-grid	loop	10	0.108
			100	0.014
			1000	0.001

However, lists are tedious to implement when `zmstfun` is not called directly, but indirectly inside some wrapper function `f(x,Q2)`. Instead of processing lists it is then better to construct beforehand, with the SPLINT package, an interpolation spline. Spline interpolation gives fast access to the structure function at any x and Q^2.

To investigate the timing we created an F_2 spline on a 20×10 x-Q^2 grid giving a relative interpolation accuracy of better than 10^{-3}. The 1 ms overhead of spline creation is included in the timing results listed in the table. It is seen that interpolation with this spline is already faster than `zmstfij` for a few F_2 calculations, and out-runs `zmstfun` list processing for $n = 100$–200, becoming 4 times faster than `zmstfun` for large n.

A List of FORTRAN routines and C++ prototypes

Subroutine or function	Description
`IZMVERS ( )`	Get the ZMSTF version number
`ZMWORDS ( &ntotal, &nused )`	Words available, used
`ZMFILLW ( &nused )`	Fill weight tables
`ZMDUMPW ( lun, 'filename' )`	Dump weight tables
`ZMREADW ( lun, 'filename', &nused, &ierr )`	Read weight tables
`ZMWFILE ( 'filename' )`	Maintain weight file on disk
`ZMDEFQ2 ( a, b )`	Define Q^2
`ZMABVAL ( &a, &b )`	Retrieve a and b coefficients
`ZMQFRMU ( qmu2 )`	Convert $\mu_{\rm F}^2$ to Q^2
`ZMUFRMQ ( Q2 )`	Convert Q^2 to $\mu_{\rm F}^2$
`ZSWITCH ( iset )`	Switch pdf set
`ZMSTFIJ ( iset, def, ix, iq, ichk )`	Structure function on grid-point
`ZMSTFUN ( istf, def, x, Q2, &f, n, ichk )`	Structure function for x-Q^2 list

Output arguments are prefixed with an ampersand (`&`).

C++ prototype

```
int     izmvers ( )
void    zmwords ( int &ntotal, int &nused )
void    zmfillw ( int &nused )
void    zmdumpw ( int lun, string filename )
void    zmreadw ( int lun, string filename, int &nused, int &ierr )
void    zmwfile ( string filename )
void    zmdefq2 ( double a, double b )
void    zmabval ( double &a, double &b )
double  zmqfrmu ( double qmu2 )
double  zmufrmq ( double Q2 )
void    zswitch ( int iset )
double  zmstfij ( int istf, double *def, int ix, int iq, int ichk )
void    zmstfun ( int istf, double *def, double *x, double *Q2,
                  double *f, int n, int ichk )
```