fastNLO v2.0 tableformat

Proposal for an ultra-flexible format to store any fastNLO-like table in x-space.
Original idea:

- later include non-pert. corrections and data (incl uncertainties) in the same table
- later include new physics processes in the same table

Maybe better and more flexible:

- ability to pass more than one table to code
- this allows to use separate files for (LO,NLO), (data, non-pert), (new physics)
- this e.g. allows to supply different new physics tables, depending on which model you want to compute
- the point: it does not matter if certain contributions are in the same physical table - the code combines logically all supplied tables (if they are consistent)

1234567890	Block A1 - fastNLO version (blocks A1 & A2 together are the "table header"
Itabversion	int: table version No. * 10000 (v1.5 -> 15000)
ScenName	string: Scenario Name
Ncontrib	int: No. of Contributions available in this table (includes No. of multiplicative contributions, but not number of data blocks)
Nmult	int: No. of multiplicative contributions
Ndata	int: No. of data blocks (can only be 0 or 1)
1234567890	Block A2 - Scenario Description (blocks A1 & A2 together are the "table header"
Ipublunits	int: negative power of ten for x-sect units in published paper
NScDescript	int: No. of subsequent strings for Scenario Description (<=30)
# for i=1,NScDescript	
ScDescript(i)	string(NscDescript): strings which will be printed during the 1st call of the usercode.

	Proposed format: **- 1st: "dsigma_dpT_dy-(nb_GeV)"** **- 2nd: hep-ex No. of publication** **- 3rd: Collaboration** **- Description of Process** **- Table No. where corresponding data results are listed** **- Fig. No. where corresponding data points are plotted** **- e.g. jet algorithm and parameters used** **- detailed cuts ...** **(note: converted v1.4 tables have always exactly 3 strings)**
# endfor (i)	
Ecms	**dbl: center-of-mass energy of collider**
ILOord	**int: The order of alphas of the LO contribution**
NObsBin	**int: total No. of Observable Bins in Scenario**
NDim	**int: No. of Dimensions in which Observable is presented** **-> e.g. for inclusive pp jets: y is dim=2, pT is dim=1** **-> e.g. for DIS jets: Q2 is dim=2, pT is dim=1**
# for i=1,NDim	
DimLabel(i)	**string(NDim): Labels what the Dimensions are (30 char)**
# endfor(i)	
# for i=1,NDim	
IDiffBin(i)	**int: Flag =1 (differential distribution) =2 (binned distribution) in i-th dimension** **(do we have one bin value / or two bin boundaries?)**
# endfor	
## for i=1,NObsBin	

# for j=1,NDim	
LoBin[i][j]	dbl: Bin Center (IDiffBin=1) or lower boundary (IDiffBin=2) in j-th dimension
*** if IDiffBin(j)=2: UpBin[i][j]	dbl: upper bin boundary in j-th dimension (if j-th dimension is binned)
# endfor(j)	
## endfor(i)	
INormFlag	int: normalization flag =0: the results from this table are the final cross sections >0: the results from this table should be normalized =1: each distribution (in highest dim.) is normalized by its own integral (or, more general: by a sum of contiguous bins from the same table) =2: each distribution is normalized by a single external value - or by a sum of contiguous bins =3: each distribution is normalized bin-by-bin by a second distribution - or by a sum of contiguous bins =-1: result from this table is already a denominator to normalize another table For INormFlag=2 or =3: a second instance of the usercode will be called to compute the denominator For INormFlag=-1 one could include additional cross checks to avoid nonsense
*** If INormFlag=2 or =3: DenomTable	string: Name of 2nd Table which includes calculation for Denominator
*** If INormFlag>0:	
# for i=1,NObsBin	
IDivLoPointer(i)	int: pointer to lower bin in same or second table used to compute the sum by which i-th bin will be divided
IDivUpPointer(i)	int: pointer to upper bin in same or second table used to compute the sum by which i-th bin will be divided
# endfor(i)	

	Block B - All Contributions in Sub-Blocks 01, 02, 03, ...
1234567890	**Block B01 - First Contribution**
IXsectUnits	**int: negative powers of ten for x-sect units (note: although allowed by tableformat v1.5, we never use different x-sect units within one scenario)**
IDataFlag	**int: Flag =1 for data block =0 for theory contributions**
IAddMultFlag	**int: Flag =0 for additive =1 for multiplicative contribution (multiplicative contributions are only numbers w/o dependence on alpha_s, PDFs, scales, etc.) e.g. hadronization corrections**
IContrFlag1	**int: type of contribution 1: fixed order - all 1 contributions can be added/ multiplied - relative order in IContrFlag2 (1=LO, 2=NLO, ...) 2: SM corrections -> more info in IContrFlag2 (1=thresh.cor., 2=e/w, ...) - relative order in IContrflag3 3: "New Physics" corrections -> more info in IContrFlag2 (1=QuarkComp, 2=ADD-LED, 3=TeV-1-ED, ...) - more in IContrFlag3 (1=interference w/ LO term, 2=pure LO NP term, 3= interference w/ NLO, 4= pure NLO NP term)**
IContrFlag2	**int: flag - contributions with identical IContrFlag2 belong to the same calculation/model**
IContrFlag3	**int: additional flag to define contributions (for future use)**
NContrDescr	**int: No. of strings to describe contribution**
# for i=1,NContrDescr	
CtrbDescript(i)	**string(NContrDescr) (30 char): Description of Contribution "LO"**

# endfor(i)	
NCodeDescr	int: No. of strings to describe Code used in calculation
# for i=1,NCodeDescr	
CodeDescript(i)	string(NCodeDescr) (30 char): describe code that was used for computation
# endfor(i)	
# if ((not data) and (not multiplicative)) ... continue	
IRef	int: flag if this contribution is a "standard" fastNLO table (=0) or a "reference table" (=1) which includes alphas, PDFs
IScaleDep	int: flag for scale dependence of coefficients =0: for "Born-type" coefficient with no scale dependence - as in LO, or LO corrections (some "New Physics") =1: for standard fixed-order contributions (NLO, NNLO, ...) where coefficients depend on the scale in a special way which allows a-posteriori scale variations =2: for contributions in which coefficients depend on the scale in a way which does not allow a-posteriori scale variations
Nevt	int: Number of Events (or Integration Points) used (important: in Fortran use Integer*8!!!)
Npow	int: absolute order in alphas
NPDF	int: No. of PDFs involved (only choices: 0,1,2)
# for i=1,NPDF	
NPDFPDG(i)	int: PDG code of particle for i-th PDF (special values for nuclei)
# endfor(i)	

NPDFDim	**int: No. of 'Dimensions' in which PDFs are stored =0 linear (for NPDF=1) =1 half matrix (for NPDF=2 using symmetries) =2 full matrix (for NPDF=2 w/o symmetries)**
NFragFunc	**int: No. of Fragmentation Functions (FFs) involved**
# for i=1,NFRagFunc	
NFFPDG(i)	**int: PDF code of particle for i-th Fragmentation Function**
# endfor(i)	
NFFDim	**int: Flag how FFs are stored (to be defined ...)**
NSubproc	**int: No. of partonic Subprocesses - comment: this is redundant with the next flag, but still kept!)**
IPDFdef1	**int: 1st Flag to define PDF linear combinations corresponding to partonic subprocesses (0:not specified-see single coeff 1:e+e- 2: incl DIS, 3:hh/hh-bar-jets 4:hH different hadrons(gamma-p))**
IPDFdef2	**int: 2nd Flag to define PDF linear combinations if IPDFdef1 = 1 if IPDFdef1 = 2 1: NC DIS 2: direct gammaP if IPDFdef1 = 3 1: hhbar-jets if IPDFdef1 = 4 1: resolved gammaP**
IPDFdef3	**int: 3rd Flag to define PDF linear combinations**
IPDFCoeff (obsolete - not used!!)	**int: Flag for predefined sets of PDF coefficients =0 not specified -> specify all n*13^m coeficients below else: ee <1000000; ep 1000000's; pp 2000000's 1000101: incl DIS LO - 1 subproc ~ sum(e^2 q) 1000102: incl DIS NLO - 2 subproc ~ sum(e^2 q), g**

	1000103: incl DIS N^nLO - 3 subproc ~ sum(e^2 q),g,sum(q) 1000110's: same with e/w corrections 2000101: jets LO (6 subproc) 2000102: jets N^nLO (7 subproc) 2000110's jet w/ e/w corrections (? subproc) 2000201: Z LO 2000202: Z NLO 2000203: Z NNLO 2000300's same for W
*** if IPDFdef1=0	if there is no predefined set for the PDF linear combinations, all single PDF coefficients will be stored here
## for k=1,NSubproc	
	to be filled
>> case NPDF=1	
>> case NPDF=2	
## endfor(k)	
	comment: as far as I see, a similar procedure is not needed for FFs
*** if NPDF>0	
## for i=1,NObsBin	
Nxtot1(i)	int: No. of x-nodes (in 1st dimension of PDF array)
# for j=1,Nxtot1(i)	

XNode1(i)(j)	dbl: x-values of nodes where PDFs are evaluated
# endfor(j)	
## endfor(i)	
*** if NPDFdim=2	if we use a full matrix to store coefficients
## for j=1,NObsBin	
Nxtot2(i)	int: No. of x-nodes in 2nd dimension of PDF array for i-th ObsBin
# for i=1,Nxtot2(i)	
Xnode2(i)(j)	dbl: x-values of nodes in 2nd dim where PDF are evaluated
# endfor(i)	
## endfor(j)	
*** endif (NPDFdim)	
*** endif (NPDF)	
NScales	int: No. of scales involved 1st mur, then muf for all PDFs, then muF for all FFs
NScaleDim	int: No. of dimensions in which scales are stored
# for i=1,Nscales	
Iscale(i)	int: pointer to dimension in which i-th scale is stored
# endfor (i)	

## for i=1,Nscaledim	
NscaleDescript(i)	int: No. of Description strings for i-th scale dimension
# for j=1,Nscaledescript(i)	
ScaleDescript(i)(j)	string: Description how "unity" in i-th scale dimension is defined e.g.: "pT of individual jet", "computed from the two highest pT jets", "fixed value - taken at 45% within the pT bin"
# endfor(j)	
## endfor(i)	
# for i=1,NscaleDim	
Nscalevar(i)	int: No. of scale variations in i-th dimension
Nscalenode(i)	int: No. of scale nodes in i-th dimension used for interpolation (must be the same for all variations)
# endfor	
## for i=1,Nscaledim	
# for j=1,Nscalevar(i)	
ScaleFac(i)(j)	dbl: j-th scale variation factor in i-th dimension
# endfor(i)	
## endfor(j)	

#### for i=1,NObsBin	
### for j=1,Nscaledim	
## for k=1,Nscalevar(j)	
# for l=1,Nscalenode(j)	
ScaleNode(i)(j)(k)(l)	dbl: l-th node in k-th scale variation in j-th dimension for i-th observable bin (in v1.4 this was: murscale*murval)
# endfor(l)	
## endfor(k)	
### endfor(j)	
#### endfor(i)	
###### for i=1,NObsBin	
##### for j=1,Nscaledim (remove!!!)	
#### for k=1,Nscalevar(j) (repeat for each dim)	
### for l=1,Nscalenode(j) (repeat for each dim)	
>>> the following assumes NFragFunc=0	to be worked out for FFs
>> case NPDFdim=0: nxmax = Nxtot(1,i)	linear
>> case NPDFdim=1: nxmax = (Nxtot(1,i)^2+Nxtot(1,i)) /2	half matrix
>> case NPDFdim=2: nxmax = Nxtot(1,i)*Nxtot(2,i)	full matrix
## for m=1,nxmax	
# for n=1,Nsubproc	

SigmaTilde(i)(j)(k)(l)(m)(n)	dbl: sigma tilde (old: buggy def.)
SigmaTilde(i)(k1)(l1)...(k[n])(l[n])(m)(n)	dbl: sigma tilde (n=NScaleDim)
# endfor(n)	
## endfor(m)	
### endfor(l)	
#### endfor(k)	
##### endfor(j)	
###### endfor(i)	
1234567890	Block B02 (second contribution) - optional
	same structure as block B01
1234567890	Block B03 (third contribution) - optional
	same structure as block B01
1234567890	Block Bnn (nn-th contribution) - optional
	same structure as block B01
1234567890	end of table
1234567890	redundant extra check