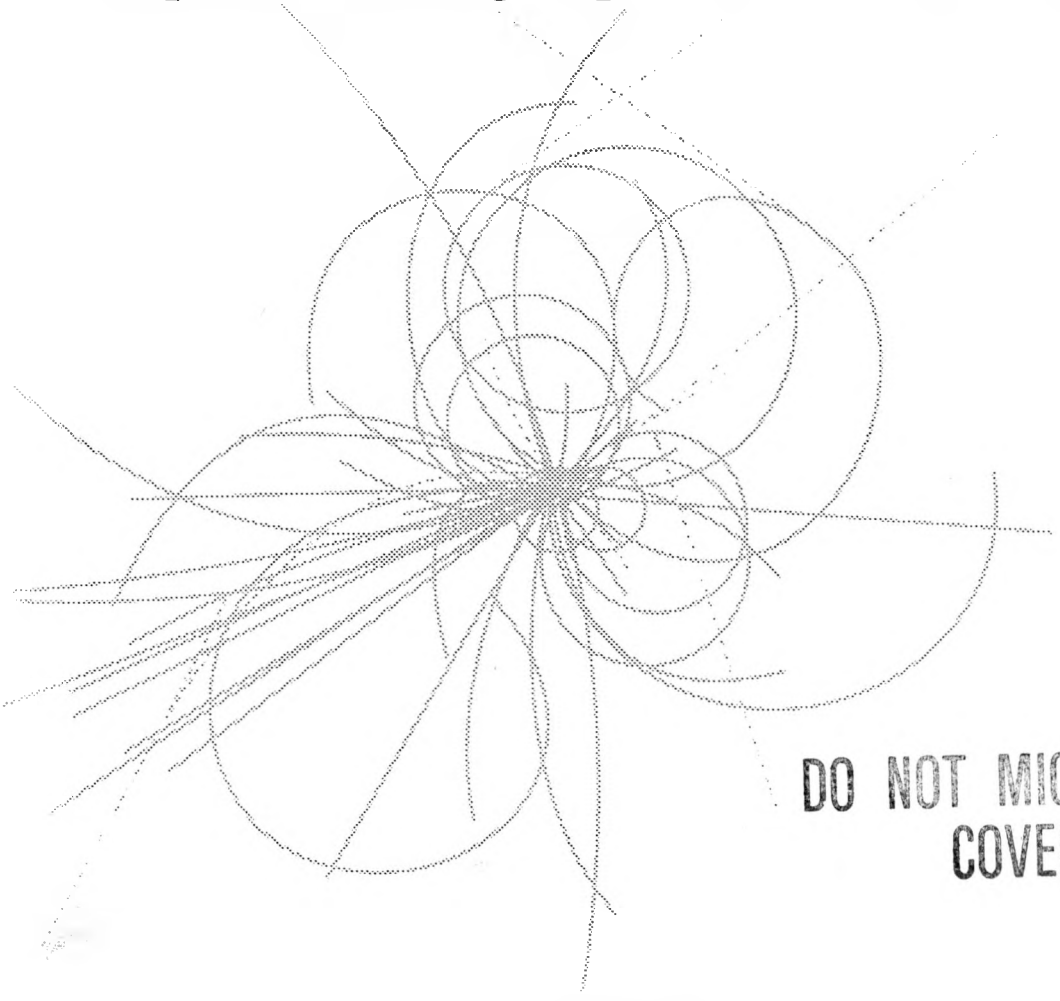
SSCL-300

Superconducting Super Collider Laboratory



"ZLIB"
A Numerical Library for Differential Algebra
(A User's Guide for Version 1.0)

Y. Yan and Chiung-Ying Yan

December 1990

SSCL--300

DE91 012552

"ZLIB"
A NUMERICAL LIBRARY FOR DIFFERENTIAL ALGEBRA
(A User's Guide for Version 1.0)

Yiton Yan

Superconducting Super Collider Laboratory*
2550 Beckleymeade Avenue
Dallas, TX 75237

and

Chiung-Ying Yan

1823 Beaver Creek Drive
Duncanville, TX 75237

December 1990

* Operated by the Universities Research Association, Inc., for the U.S. Department of Energy under Contract No. DE-AC02-89ER40486.

SSCL-300

"ZLIB"
A Numerical Library for Differential Algebra
(A User's Guide for Version 1.0)

Yiton Yan

Superconducting Super Collider Laboratory*

2550 Beckleymeade Avenue

Dallas, TX 75237

and

Chiung-Ying Yan

1823 Beaver Creek Drive

Duncanville, TX 75237

December 1990

Abstract

Given an efficient numerical method and a supercomputer, differential algebra can be a powerful tool for the study of accelerator physics. "ZLIB", which has a style similar to the numerical library "IMSL", has been developed to offer efficient numerical routines on supercomputers for differential algebra. "ZLIB" uses dynamic memory and is both vectorized and parallelized (multi-tasked) besides being scalarly optimized. There are two sub-libraries in "ZLIB", "TPALIB" and "ZPLIB", with unique data structures for flexibility. The "TPALIB" is more flexible in dealing with a different number of variables, and therefore is more suitable for use in extracting maps. The "ZPLIB" is more flexible in dealing with a different number of orders, and therefore is more suitable for use in analyzing a map. Use of "ZLIB" in a scalar computer is also recommended.

* Operated by the Universities Research Association, Inc., for the U.S. Department of Energy under Contract No. DE-AC02-89ER40486.

1. INTRODUCTION

With limited computer memory, and limited computational speed, differential algebra should be treated as the algebra of truncated power series. The algebra of low order truncated power series can be easily accomplished with a simple data structure. However, in most cases, high-order truncated power series is desirable. Therefore, a special data structure is necessary to optimize both the allocation of the computer memory and the numerical speed.

"ZLIB" has been developed for differential algebra, mainly for use on supercomputers. The use of "ZLIB" is similar to the use of the "IMSL" library. Routines in "ZLIB" are vectorized, multi-tasked and use dynamic memory. There are two sub-libraries in "ZLIB", the "TPALIB" and the "ZPLIB", with unique data structures. The "TPALIB" is more flexible in dealing with a different number of variables, and therefore is more suitable for use in extracting a one-turn (or one-period) map for a storage ring such as the SSC. The "ZPLIB" is more flexible in dealing with a different number of orders, and therefore is more suitable for use in analyzing a map. The two sub-libraries can be used simultaneously through a structure-translation routine. Although "ZLIB" is developed mainly for supercomputers, the authors have simultaneously tried to optimize the routines for scalar computers and therefore the use of "ZLIB" in scalar computers is also recommended.

It is not the authors' attempt to describe the data structure of the two sub-libraries, but rather to introduce the use of "ZLIB" to the users. In Section 2, the truncated power series is briefly introduced. Readers who are familiar with differential algebra should skip this section and go to Section 3, where a brief general description is given for the "ZLIB". Available routines in each of the sub-libraries, the "ZPLIB" and the "TPALIB", are discussed in Section 4 and in Section 5, respectively.

2. THE ALGEBRA OF TRUNCATED POWER SERIES

In this section, the authors are not trying to be mathematically rigorous. Once a variable, a function, or an operation is mentioned, its existence is assumed.

(a) Symbolic convention

Let $\vec{\mathbf{z}}$ be an n-dimensional vector, i.e. its transpose can be expressed as

$$\vec{\mathbf{z}}^T = [z_1, z_2, \ldots, z_n] \,,$$

where z_i, for $i = 1, \ldots, n$, are scalar variables. For example, we can consider

$$\vec{\mathbf{z}}^T = [z_1, z_2, \ldots, z_6] = [x, p_x, y, p_y, t, p_t]$$

as the transpose of a vector representing the 6-dimensional phase space coordinates for an accelerator.

Let U be a function of $\vec{\mathbf{z}}$. This means U is a function of $z_1, z_2, \ldots, z_n$. Its truncated power series (TPS) expansion up to an integer Ω order is expressed as

$$U(\vec{\mathbf{z}}) = \sum_{k=0}^{\Omega} u(\vec{\mathbf{k}}) \vec{\mathbf{z}}^{\vec{\mathbf{k}}} \,,$$

where

$$\vec{\mathbf{z}}^{\vec{\mathbf{k}}} \equiv z_1^{k_1} z_2^{k_2} \ldots z_n^{k_n} \,,$$

$$k = \sum_{i=1}^{n} k_i \,, \qquad \text{for } 0 \le k_i \le \Omega \,,$$

and

$$\sum_{k=0}^{\Omega} \equiv \text{summation over all } \vec{\mathbf{k}}\text{'s for } k = 0, 1, \ldots, \Omega \,.$$

Note that $U(\vec{\mathbf{z}})$ is called an n-variable TPS, of order Ω. The number of monomials for an n-variable TPS, of order Ω, is given by

$$\eta = \frac{(n+\Omega)!}{n!\Omega!} .$$

A unit TPS is defined as

$$I(\vec{\mathbf{z}}) = \sum_{k=0}^{\Omega} i(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}} = 1 ,$$

i.e.

$$i(\vec{\mathbf{k}}) = 1 \qquad \text{for } k = 0 ,$$

and

$$i(\vec{\mathbf{k}}) = 0 \qquad \text{for } k > 0 .$$

Let $\vec{\mathbf{U}}(\vec{\mathbf{z}})$ be an m-dimensional vector TPS (VTPS), of n variables, and of Ω order. It is expressed as

$$\vec{\mathbf{U}}(\vec{\mathbf{z}}) = \sum_{k=0}^{\Omega} \vec{\mathbf{u}}(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}} ,$$

(i.e. $U_i(\vec{\mathbf{z}}) = \sum_{k=0}^{\Omega} u_i(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}}$, for $i = 1, 2, \ldots, m$) where the transpose of $\vec{\mathbf{u}}(\vec{\mathbf{k}})$ is given by

$$\vec{\mathbf{u}}^T(\vec{\mathbf{k}}) = [u_1(\vec{\mathbf{k}}), u_2(\vec{\mathbf{k}}), \ldots, u_m(\vec{\mathbf{k}})] .$$

One can consider $\vec{\mathbf{U}}(\vec{\mathbf{z}})$ as a map in accelerator physics.

A unit n-dimensional, n-variable VTPS of order Ω, is defined as

$$\vec{\mathbf{I}}(\vec{\mathbf{z}}) = \sum_{k=0}^{\Omega} \vec{\mathbf{i}}(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}} = \vec{\mathbf{z}} .$$

Its transpose is given by

$$\vec{\mathbf{I}}^T(\vec{\mathbf{z}}) = \vec{\mathbf{z}}^T = [z_1, z_2, \ldots, z_n] \,.$$

Numerically, $u(\vec{\mathbf{k}})$, $i(\vec{\mathbf{k}})$, $\vec{\mathbf{u}}(\vec{\mathbf{k}})$, $\vec{\mathbf{i}}(\vec{\mathbf{k}})$ are used for representing $U(\vec{\mathbf{k}})$, $I(\vec{\mathbf{k}})$, $\vec{\mathbf{U}}(\vec{\mathbf{k}})$, $\vec{\mathbf{I}}(\vec{\mathbf{k}})$, respectively.

(b) TPS Operations

Addition: $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) + V(\vec{\mathbf{z}})$

$$\sum_{k=0}^{\Omega} w(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}} = \sum_{k=0}^{\Omega} u(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}} + \sum_{k=0}^{\Omega} v(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\mathbf{k}}$$

$$= \sum_{k=0}^{\Omega} (u(\vec{\mathbf{k}}) + v(\vec{\mathbf{k}}))\vec{\mathbf{z}}^{\vec{\mathbf{k}}}$$

so

$$w(\vec{\mathbf{k}}) = u(\vec{\mathbf{k}}) + v(\vec{\mathbf{k}})$$

Subtraction: $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) - V(\vec{\mathbf{z}})$

$$\sum_{k=0}^{\Omega} w(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}} = \sum_{k=0}^{\Omega} u(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}} - \sum_{k=0}^{\Omega} v(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\mathbf{k}}$$

$$= \sum_{k=0}^{\Omega} (u(\vec{\mathbf{k}}) - v(\vec{\mathbf{k}}))\vec{\mathbf{z}}^{\vec{\mathbf{k}}}$$

so

$$w(\vec{\mathbf{k}}) = u(\vec{\mathbf{k}}) - v(\vec{\mathbf{k}})$$

Multiplication: $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) * V(\vec{\mathbf{z}})$

$$\sum_{j=0}^{\Omega} w(\vec{\mathbf{j}})\vec{\mathbf{z}}^{\vec{\mathbf{j}}} = \left(\sum_{k=0}^{\Omega} u(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}}\right) * \left(\sum_{h=0}^{\Omega} v(\vec{\mathbf{h}})\vec{\mathbf{z}}^{\vec{\mathbf{h}}}\right)$$

$$= \sum_{j=0}^{\Omega} \left(\sum_{k=0}^{\Omega} (u(\vec{\mathbf{k}}) * v(\vec{\mathbf{j}} - \vec{\mathbf{k}})\right) \vec{\mathbf{z}}^{\vec{\mathbf{j}}} .$$

So

$$w(\vec{\mathbf{j}}) = \sum_{k=0}^{\Omega} u(\vec{\mathbf{k}}) * v(\vec{\mathbf{j}} - \vec{\mathbf{k}}) ,$$
$$\text{with } (\vec{\mathbf{j}} - \vec{\mathbf{k}})_i \geq 0 \text{ for } i = 1, 2, \ldots, n .$$

Partial derivative: $W(\vec{\mathbf{z}}) = (\partial/\partial z_i)U(\vec{\mathbf{z}})$, where $i = 1, 2, \ldots,$ or n

$$\sum_{j=0}^{\Omega} w(\vec{\mathbf{j}})\vec{\mathbf{z}}^{\vec{\mathbf{j}}} = \frac{\partial}{\partial z_i} \sum_{k=0}^{\Omega+1} u(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}}$$

$$= \sum_{k=1}^{\Omega+1} k_i * u(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}-\vec{\mathbf{1}}_i}$$

$$= \sum_{j=0}^{\Omega} (j_i + 1) * u(\vec{\mathbf{j}} + \vec{\mathbf{1}}_i)\vec{\mathbf{z}}^{\vec{\mathbf{j}}}$$

so

$$w(\vec{\mathbf{j}}) = (j_i + 1) * u(\vec{\mathbf{j}} + \vec{\mathbf{1}}_i) ,$$

where $\vec{\mathbf{1}}_i$ is a unit vector in the i^{th} dimension.

Partial integration: $W(\vec{\mathbf{z}}) = \int U(\vec{\mathbf{z}})dz_i$, where $i = 1, 2, \ldots,$ or n

$$\sum_{j=0}^{\Omega+1} w(\vec{\mathbf{j}})\vec{\mathbf{z}}^{\vec{\mathbf{j}}} = \int \sum_{k=0}^{\Omega} u(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}}\, dz_i$$

$$= \sum_{k=0}^{\Omega} \frac{u(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}+\vec{\mathbf{1}}_i}}{k_i+1}$$

$$= 0 + \sum_{j=1}^{\Omega+1} \left(\frac{1}{j_i}\right) * u(\vec{\mathbf{j}} - \vec{\mathbf{1}}_i)\vec{\mathbf{z}}^{\vec{\mathbf{j}}}$$

so

$$w(\vec{\mathbf{j}}) = 0 \qquad \text{for } j_i = 0\ ,$$

$$w(\vec{\mathbf{j}}) = \left(\frac{1}{j_i}\right) * u(\vec{\mathbf{j}} - \vec{\mathbf{1}}_i) \text{for } j_i > 0\ .$$

Using the above fundamental operations for the TPS, $w(\vec{\mathbf{j}})$ can be obtained for the following basic TPS operations:

Square:	$W(\vec{\mathbf{z}}) = U^2(\vec{\mathbf{z}})$,
Inversion:	$W(\vec{\mathbf{z}}) = 1/U(\vec{\mathbf{z}})$,
Division:	$W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}})/V(\vec{\mathbf{z}})$,
Power:	$W(\vec{\mathbf{z}}) = U^p(\vec{\mathbf{z}})$, where p is an integer.
Square root:	$W(\vec{\mathbf{z}}) = \text{sqrt}(U(\vec{\mathbf{z}}))$,
Exponentiation:	$W(\vec{\mathbf{z}}) = \exp(U(\vec{\mathbf{z}}))$,
Logarithm:	$W(\vec{\mathbf{z}}) = \ln(U(\vec{\mathbf{z}}))$,
Trigonometry:	$W(\vec{\mathbf{z}}) = \sin(U(\vec{\mathbf{z}}))$, or $W(\vec{\mathbf{z}}) = \cos(U(\vec{\mathbf{z}}))$,
Poisson bracket:	$W(\vec{\mathbf{z}}) = [U(\vec{\mathbf{z}}), V(\vec{\mathbf{z}})]$.

(c) VTPS Operations

With the fundamental and the basic TPS operations ready, $\vec{\mathbf{w}}(\vec{\mathbf{j}})$ can be obtained for the following basic VTPS operations.

Concatenation: $\qquad \vec{\mathbf{W}}(\vec{\mathbf{z}}) = \vec{\mathbf{V}}(\vec{\mathbf{U}}(\vec{\mathbf{z}}))$,

where, in the usual case, $\vec{\mathbf{U}}$ is an n-dimensional n-variable VTPS, $\vec{\mathbf{V}}$ and $\vec{\mathbf{W}}$ are m-dimensional, n-variable VTPS, m and n may or may not be equal.

Inversion:

Given an n-dimensional, n-variable $\vec{\mathbf{U}}(\vec{\mathbf{z}})$, an n-dimensional, n-variable $\vec{\mathbf{U}}^{-1}(\vec{\mathbf{z}})$ can be obtained such that

$$\vec{\mathbf{U}}^{-1}(\vec{\mathbf{U}}(\vec{\mathbf{z}})) = \vec{\mathbf{U}}(\vec{\mathbf{U}}^{-1}(\vec{\mathbf{z}})) = \vec{\mathbf{I}}(\vec{\mathbf{z}}) \, .$$

All the above basic TPS or VTPS operations have been implemented in "ZLIB".

(d) Tracking:

$\vec{\mathbf{z}}' = \vec{\mathbf{U}}(\vec{\mathbf{z}})$

In conjunction with the implementation of the fundamental and basic TPS and VTPS operations, substitution of a numerical vector $\vec{z}$ into a VTPS (or a map) is implemented in the "ZLIB".

3. THE "ZLIB"

"ZLIB" is a member of the Z-family programs which include (other than ZLIB): Zmap (a map extraction program), Ztrack (a vectorized and parallelized post-Teapot tracking program), Zremc1 and Zremc2 ($1\frac{2}{2}$ - and $2\frac{1}{2}$- dimensional relativistic electromagnetic particle simulation programs), and Zpcomp (a macro precompiler for fortran). Similar to the routines in the IMSL library which perform linear algebra through matrix operations, routines in "ZLIB" perform differential algebra through the operations of expanded power series, truncated at a pre-set order, to include nonlinear effects. Unlike linear algebra which has a domain idealized to be unlimited, differential algebra has a narrow domain where the power series converge at a reasonable rate, that is, the scope of differential algebra is restricted to problems for which an interest region (domain) can be identified to have a reasonable convergent rate for the power series expansion of the governing equations. Presently "ZLIB" finds its application in accelerator

physics, since particles in an accelerator can only be stable in a region where the expanded power series of the nonlinear equations governing the system converge with a reasonable rate. Applications of "ZLIB" to other branches of physics, such as optics, should be possible.

Since "ZLIB" uses dynamic memory and includes most fundamental operations for differential algebra, a binary "ZLIB" is generally adequate for users. Users are welcome to contact the authors for free implementation of a binary ZLIB in their computers. Users are also encouraged to make suggestions and comments.

The general convention of the names of the subroutine arguments (the term "argument" instead of "parameter" is used to avoid the possible confusion between parameters in a parameter statement and in a subroutine statement) are:

nv : number of variables, an input integer; nv > 0.

nvw : number of variables actually used in the subprogram, an input integer; $0 <$ nvw $\leq$ nv.

no : order of a TPS or a VTPS, an input integer; no ≥ 0.

no? : such as "nou" or "now", order of a TPS or a VTPS actually used in the subprogram; an input integer; $0 \leq$ no? $\leq$ no.

nm : number of monomials of a TPS, i.e. nm $=$ (nv + no)!/(nv!no!).

nmw : number of monomials of a TPS actually used in the subprogram, an input integer; $0 <$ nmw $\leq$ nm.

np : number of vectors (or number of particles in accelerator physics), an input integer; np > 0.

c : an input scalar (such as c $= 5.5$).

d : an input scalar.

x : an input vector or vectors (particle phase space coordinates in accelerator physics), usually an array x(nv) or an array x(nx,np).

y : an output vector or vectors (particle phase space coordinates in accelerator physics); usually an array y(nv) or an array y(ny,np). Note that the user may (if desired) let "y" share the memory with "x" within a subprogram that has both "x" and "y" as its subroutine arguments.

u : an input TPS, it is the coefficients of a TPS $U(\vec{\mathbf{z}})$; usually an array u(nm).

v : an input TPS, it is the coefficients of a TPS $V(\vec{\mathbf{z}})$; usually an array v(nm).

w : an output TPS, it is the coefficients of a TPS $W(\vec{\mathbf{z}})$; usually an array w(nm). Note that the user may (if desired) let "w" share the memory with either "u" or "v" within a subprogram that has "w" and either "u" or "v" or both as its subroutine arguments.

uu : an input VTPS (or a map); usually an array uu(nm $*$ nu), where nu is the dimension of the VTPS uu, which is either specified in the subroutine parameter or implicitly assumed to be nu = nv.

nu : dimensions of the VTPS uu, an input integer; nu > 0.

vv : an input VTPS, usually an array vv(nm$*$nw), where nw is the dimension of the VTPS vv, which is either specified in the subroutine parameter or implicitly assumed to be nw = nv.

ww : an output VTPS, usually an array ww(nm$*$nw), where nw is the dimension of the VTPS ww, which is either specified in the subroutine parameter or implicitly assumed to be nw = nv. Note that the user may (if desired) let "ww" share the memory with either "uu" or "vv" within a subprogram that has "ww" and either "uu" or "vv" or both as its subroutine arguments.

nw : dimensions of the VTPS's vv and/or ww, an input integer; nw > 0.

nou : order to be used for the TPS u or the VTPS uu, an input integer; $0 \leq$ nou $\leq$ no.

nov : order to be used for the TPS v or the VTPS vv, an input integer; $0 \leq$ nov $\leq$ no.

now : order desired for the TPS w or the VTPS ww, an input integer; $0 \leq$ now $\leq$ no.

nok : actual order for the TPS w, an input integer; $0 \leq$ nok $\leq$ now.

nd : sets of canonically conjugate variables, an input integer; $0 <$ nd $\leq$ nv/2.

npwr : power to be performed for a TPS (such as u ** npwr), an input integer; $-\infty <$ npwr $< \infty$.

Some of the above subroutine arguments might be commented again upon its appearance. Subroutine arguments which are not described above will be commented upon when they appear.

4. THE SUB-LIBRARY "ZPLIB"

To use the "ZPLIB" routines, users must obtain the compiled "ZLIB" from the authors so that their program can be loaded with the routines in the "ZPLIB". Before any subroutine using the data structure of the "ZPLIB" is called, the user should include the following statement (assuming ZLIB 1.0 is used)

"call zpprep(nv,no,nm,npm),"

where "nv' and "no" are the number of variables and the maximum order the user desires; "nm", is a returned value for the number of monomials, i.e. nm = (nv + no)!/(nv!no!), is returned for the user; "npm" is the maximum number of particles. The user should set a small integer or 0 for npm if tracking is not desired. Occasionally, the user may wish to use routines in the sub-library "ZPLIB" to perform initialization (reading in a VTPS) and tracking only. In

such a case, he may replace the statement "call zpprep(nv,no,nm,npm)," with the calling statement "call zptrkp(nv,no,nm,npm)," to save computer memory.

Once the statement, "call zpprep(nv,no,nm,npm)," is executed, all the TPS's (u, v, and w) are assumed to be nv-variable TPS's of order smaller than or equal to "no", and all the VTPS's (uu, vv, and ww) are assumed to be nv-variable VTPS's of order "no", although operations can be performed up to orders that are lower than "no".

The subroutines available in the "ZPLIB" are as follows.

(a) TPS Operation

Initialization:

(1) subroutine zpzro(w,nmw)

for performing $W(\vec{\mathbf{z}}) = 0$

(2) subroutine zpconst(c,w,nmw)

for performing $W(\vec{\mathbf{z}}) = c$

if $c = 1$, $W(\vec{\mathbf{z}}) = I(\vec{\mathbf{z}}) = 1$

(3) subroutine zpcpy(u,w,nmw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}})$

(4) subroutine zpsgn(u,w,nmw)

for performing $W(\vec{\mathbf{z}}) = -U(\vec{\mathbf{z}})$

(5) subroutine zpok1(w,c,iv,nmw)

for performing $W(\vec{\mathbf{z}}) = c * z_{\mathrm{iv}}$

iv: an input positive integer; iv $\leq$ nv

(6) subroutine zppok(w,c,js)

for performing $W(\vec{\mathbf{z}}) = c * \vec{\mathbf{z}}^{\vec{\mathbf{k}}}$

js: an input nv-dimensional array;

$\mathrm{js}(i) = k_i$ for $i = 1, 2, \ldots,$ nv

Addition and subtraction:

(7) subroutine zpadd(u,v,w,nmw)

for performing $W(\vec{\mathbf{z}}) = U(\mathbf{z}) + V(\vec{\mathbf{z}})$

(8) subroutine zpcadd(c,u,w,nmw)

for performing $W(\vec{\mathbf{z}}) = c + U(\vec{\mathbf{z}})$

(9) subroutine zpsub(u,v,w,nmw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) - V(\vec{\mathbf{z}})$

(10) subroutine zpsubc(u,c,w,nmw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) - c$

(11) subroutine zpcsub(c,u,w,nmw)

for performing $W(\vec{\mathbf{z}}) = c - U(\vec{\mathbf{z}})$

Multiplication and division with scalars:

(12) subroutine zpcmul(c,u,w,nmw)

for performing $W(\vec{\mathbf{z}}) = c * U(\vec{\mathbf{z}})$

(13) subroutine zpdivc(u,c,w,nmw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}})/c$

(14) subroutine zplin(u,c,v,w,nmw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) + c * V(\vec{\mathbf{z}})$

(15) subroutine zpblin(d,u,c,v,w,nmw)

for performing $W(\vec{\mathbf{z}}) = d * U(\vec{\mathbf{z}}) + c * V(\vec{\mathbf{z}})$

** As an example: "call zpblin(3.3,u,−1.1,v,w,nm)"

Multiplication and division:

(16) subroutine zpmul(u,nou,v,nov,w,now,nok)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) * V(\vec{\mathbf{z}})$

(17) subroutine zpdiv(u,nou,v,nov,w,now)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}})/V(\vec{\mathbf{z}})$

(18) subroutine zpinv(u,nou,w,now)

for performing $W(\vec{\mathbf{z}}) = 1/U(\vec{\mathbf{z}})$

(19) subroutine zpsq(u,nou,w,now,nok)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) * U(\vec{\mathbf{z}})$

(20) subroutine zppwr(u,nou,npwr,w,now)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) ** \text{npwr}$

** As an example to show that "w" can share memory with "u", one can have a statement such as "call zppwr(u,3,−4,u,5)"

Derivative and Integral:

(21) subroutine zpdrv(u,nou,w,now,iv,nok)

for performing $W(\vec{\mathbf{z}}) = (d/dz_{\text{iv}})U(\vec{\mathbf{z}})$

iv ($\leq$ nv): an input integer.

(22) subroutine zpintg(u,nou,w,now,iv,nok)

for performing $W(\vec{\mathbf{z}}) = \int U(\vec{\mathbf{z}})dz_{\text{iv}}$,

iv ($\leq$ nv): an input integer.

(23) subroutine zpbrac(u,nou,v,nov,w,now,nok,nd)

for performing $W(\vec{\mathbf{z}}) = [U(\vec{\mathbf{z}}), V(\vec{\mathbf{z}})]$,

the Poisson bracket of U and V.

nd: sets of the canonically conjugate variables; nd $\leq$ nv/2.

Functions:

(24) subroutine zpsin(u,nou,w,now)

for performing $W(\vec{\mathbf{z}}) = \sin(U(\vec{\mathbf{z}}))$

(25) subroutine zpcos(u,nou,w,now)

for performing $W(\vec{\mathbf{z}}) = \cos(U(\vec{\mathbf{z}}))$

(26) subroutine zpexp(u,nou,w,now)

for performing $W(\vec{\mathbf{z}}) = \exp(U(\vec{\mathbf{z}}))$

(27) subroutine zplog(u,nou,w,now)

for performing $W(\vec{\mathbf{z}}) = \ln(U(\vec{\mathbf{z}}))$

(28) subroutine zpsqrt(u,nou,w,now)

for performing $W(\vec{\mathbf{z}}) = \text{sqrt}\ (U(\vec{\mathbf{z}}))$

(b) VTPS Operations:

Initialization:

(29) subroutine zpunit(ww,now)

for performing $\vec{\mathbf{W}}(\vec{\mathbf{z}}) = \vec{\mathbf{I}}(\vec{\mathbf{z}}) = \vec{\mathbf{z}}$

ww : an nv-dimensional, nv-variable VTPS of order "no", but only up to order "now $\leq$ no" is operated; array ww(nm * nv).

(30) subroutine zpmok1(ww,nw,now,c,iw)

for performing $W_{iw}(\vec{\mathbf{z}}) = cz_{iw}$

ww : an nw-dimensional, nv-variable VTPS of order "no", but only up to order "now $\leq$ no" is operated; array ww(nm*nw).

Note: Only the iw$^{\text{th}}$ dimension is initiated.

(31) subroutine zpmpok(ww,nw,now,c,js,iw)

for performing $W_{iw}(\vec{\mathbf{z}}) = c\vec{\mathbf{z}}^{\vec{\mathbf{k}}}$

js: an input nv-dimensional array;

js($\vec{i}$) = k_i for $i = 1, 2, \ldots,$ nv

ww : an nw-dimensional, nv-variable VTPS of order "no", but only up to order "now $\leq$ no" is operated; array ww(nm*nw).

Note: Only the iw$^{\text{th}}$ dimension is initiated.

(32) subroutine rdmaptpa(ww,nwb,nw,now,nomap,imap)

for initializing $\vec{\mathbf{W}}(\vec{\mathbf{z}}) = \sum_{k=0}^{\Omega} \vec{\mathbf{w}}(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}}$

Initialize an nw-dimensional VTPS of order "no" from its nwb$^{\text{th}}$ dimension to nw$^{\text{th}}$ dimension up to order "now" by reading a "TPALIB" structured VTPS (of order "nomap") file (specified by the number "imap") where data are stored from the nwb$^{\text{th}}$ dimension to the nw$^{\text{th}}$ dimension.

(33) subroutine rdmapzp(ww,nw,now,imap)

for initializing $\vec{\mathbf{W}}(\vec{\mathbf{z}}) = \sum_{k=0}^{\Omega} \vec{\mathbf{w}}(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}}$

Initialize an nw-dimensional VTPS of order "no" up to a desired dimension "nw" and a desired order "now" by reading a "ZPLIB" structured VTPS file (specified by the number "imap") where data are stored to any dimension and to any order.

Writing out the VTPS:

(34) subroutine wrmapzp(uu,nub,nu,nou,imap)

output the coefficients $\vec{\mathbf{u}}(\vec{\mathbf{k}})$ in a file for $\vec{\mathbf{U}}(\vec{\mathbf{z}}) = \sum_{k=0}^{\Omega} \vec{\mathbf{u}}(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}}$

Write out an nu-dimensional VTPS, uu, of order "no", from its nub^{th} dimension to nu^{th} dimension up to order "nou" in "ZPLIB" form to a file (specified by the number "imap").

(35) subroutine wrmapzp1(uu,nub,nu,nou,imap)

output the coefficients $\vec{\mathbf{u}}(\vec{\mathbf{k}})$ in a file for $\vec{\mathbf{U}}(\vec{\mathbf{z}}) = \sum_{k=1}^{\Omega} \vec{\mathbf{u}}(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}}$

Write out an nu-dimensional VTPS (a map), uu, of order "no" from its nub^{th} dimension to nu^{th} dimension up to order "nou" in "ZPLIB" form to a file (specified by the number "imap"). Note that the zeroth order is assumed to be 0 and is not written in the file.

Concatenation of VTPS's:

(36) subroutine zpcnct(uu,nou,vv,nov,ww,now,nw)

for performing $\vec{\mathbf{W}}(\vec{\mathbf{z}}) = \vec{\mathbf{V}}(\vec{\mathbf{U}}(\vec{\mathbf{z}}))$

uu : an input nv-dimensional VTPS of order no; uu represent $\vec{\mathbf{U}}(\vec{\mathbf{z}})$

vv : an input nw-dimensional VTPS of order no; vv represent $\vec{\mathbf{V}}(\vec{\mathbf{z}})$

ww : an output nw-dimensional VTPS of order no; ww represent $\vec{\mathbf{W}}(\vec{\mathbf{z}})$

nou : order to be used for the VTPS uu; nou $\leq$ no

nov : order to be used for the VTPS vv; nov $\leq$ no

now : order desired for the VTPS ww; now $\leq$ no

$***$ An example: "call zpcnct(uu,14,uu,14,uu,15,4)"

Inversion of a VTPS:

(37) subroutine zpmapinv(uu,nou,ww,now)

for performing $\vec{\mathbf{W}}(\vec{\mathbf{z}}) = \vec{\mathbf{U}}^{-1}(\vec{\mathbf{z}})$

uu : an input nv-dimensional VTPS of order no; uu represent $\vec{\mathbf{U}}(\vec{\mathbf{z}})$

ww : an output nv-dimensional VTPS of order no; ww represent $\vec{\mathbf{W}}(\vec{\mathbf{z}}) = \vec{\mathbf{U}}^{-1}(\vec{\mathbf{z}})$

nou : an input; order to be used for the VTPS uu; nou $\leq$ no

now : an input; order desired for the VTPS ww; now $\leq$ no; usually now = nou.

(c) Tracking

Single-particle tracking:

(38) subroutine zpmtrk(uu,nub,nu,nou,x,y)

for performing $\vec{\mathbf{y}} = \vec{\mathbf{U}}(\vec{\mathbf{x}})$

uu : an input nu-dimensional, nv-variable VTPS of order no, although only up to order nou ($\leq$ no) is operated; uu represent $\vec{\mathbf{U}}(\vec{\mathbf{x}})$.

x : an input vector of dimension nv.

y : an output vector of dimension nu.

actual operations are for $y_i = uu_i(\vec{\mathbf{x}})$ for i =nub, nub + 1, ..., nu.

Multi-particle tracking:

(39) subroutine zpmtrks(uu,nub,nu,nou,np,x,nx,y,ny)

for performing $\vec{\mathbf{y}}^p = \vec{\mathbf{U}}(\vec{\mathbf{x}}^p)$, $p = 1, 2, \ldots,$ np

uu : an input nu-dimensional, nv-variable VTPS of order no, although only up to order nou ($\leq$ no) is operated; uu represent $\vec{\mathbf{U}}(\vec{\mathbf{x}})$.

x : array x(nx,np) where nx $\geq$ nv, an input; users should consider it as np particles, each with nx-dimensional phase-space coordinates.

y : array y(nx,np) where ny $\geq$ nv, an output; users should consider it as np particles, each with ny-dimensional phase-space coordinates.

actual operations are for $y_i^p = uu_i(\vec{\mathbf{x}}^p)$

for $i =$ nub, nub + 1, ..., nu, and $p = 1, 2, \ldots,$ np.

(40) subroutine zpmtrkq(uu,nub,nu,nou,np,x,nx,y,ny)

for performing $\vec{\mathbf{y}}^p = \vec{\mathbf{U}}(\vec{\mathbf{x}}^p)$, $p = 1, 2, \ldots,$ np

uu : an input nu-dimensional, nv-variable VTPS of order no, although only up to order nou ($\leq$ no) is operated; uu represent $\vec{\mathbf{U}}(\vec{\mathbf{x}})$.

x : array x(nx,np) where nx $\geq$ nv, an input; users should consider it as np particles, each with nx-dimensional phase-space coordinates.

y : array y(nx,np) where ny $\geq$ nv, an output; users should consider it as np particles, each with ny-dimensional phase-space coordinates.

actual operations are for $y_i^p = uu_i(\vec{\mathbf{x}}^p)$

for $i =$ nub, nub + 1, ..., nu, and $p = 1, 2, \ldots,$ np.

** Note that the internal structures in "zpmtrks" and in "zpmtrkq" are different. Vectorization is over particles in "zpmtrks" while vectorization is within a particle and parallel (multi-tasking) computing can be chosen over particles in "zpmtrkq."

(41) subroutine zpmtrkw(uu,nub,nu,nou,np,npm,x,y)

for performing $\vec{\mathbf{y}}^p = \vec{\mathbf{U}}(\vec{\mathbf{x}}^p)$, $p = 1, 2, \ldots,$ np

uu : an input nu-dimensional, nv-variable VTPS of order no, although only up to order nou ($\leq$ no) is operated; uu represent $\vec{\mathbf{U}}(\vec{\mathbf{x}})$.

x : array x(npm,nv) where npm $\geq$ np, an input; users should consider it as npm particles, each with nv-dimensional phase-space coordinates.

y : array y(npm,nu) where npm $\geq$ np, an output; users should consider it as npm particles, each with nu-dimensional phase-space coordinates.

actual operations are for $y_i^p = uu_i(\vec{\mathbf{x}}^p)$

for $i =$ nub, nub + 1, ..., nu, and $p = 1, 2, \ldots,$ np.

(42) subroutine zpmtrkp(uu,nub,nu,nou,np,npm,x,y)

for performing $\vec{\mathbf{y}}^p = \vec{\mathbf{U}}(\vec{\mathbf{x}}^p)$, $p = 1, 2, \ldots,$ np

uu : an input nu-dimensional, nv-variable VTPS of order no, although only up to order nou ($\leq$ no) is operated; uu represent $\vec{\mathbf{U}}(\vec{\mathbf{x}})$.

x : array x(npm,nv) where npm $\geq$ np, an input; users should consider it as npm particles, each with nv-dimensional phase-space coordinates.

y : array y(npm,nu) where npm $\geq$ np, an output; users should consider it as npm particles, each with nu-dimensional phase-space coordinates.

actual operations are for $y_i^p = uu_i(\vec{\mathbf{x}}^p)$

for $i =$ nub, nub + 1, ..., nu, and $p = 1, 2, \ldots,$ np.

** Note that the internal structures in "zpmtrkw" and in "zpmtrkp" are different. Vectorization is over particles in "zpmtrkw," while vectorization is within a particle and parallel (multi-tasking) computing can be chosen over particles in "zpmtrkp."

Scaling:

(43) subroutine zpmscle(uu,nw,now,ww,s)

$$\text{for performing } \vec{\mathbf{W}}(\vec{\mathbf{z}}') = \sum_{k=0}^{\Omega} \vec{\mathbf{w}}(\vec{\mathbf{k}}) \vec{\mathbf{z}}'^{k} = \vec{\mathbf{U}}(\vec{\mathbf{z}}) = \sum_{k=0}^{\Omega} \vec{\mathbf{u}}(\vec{\mathbf{k}}) \vec{\mathbf{z}}^{\vec{\mathbf{k}}},$$

$$\text{where } z_i = z_i' * s(i) \text{ for } i = 1, 2, \ldots, \text{nv}$$

uu, ww : nw-dimensional, nv-variable VTPS's of order "no", but only up to order "now" (now≤no) is scaled; uu represents $\vec{\mathbf{U}}(\vec{\mathbf{z}})$, ww represents $\vec{\mathbf{W}}(\vec{\mathbf{z}})$.

s : an input nv-dimensional vector.

(d) Structure translation between "ZPLIB" and "TPALIB"

(44) subroutine zptpa(uu,nw,now,ww,iflag)

Translate an nw-dimensional, nv-variable VTPS of order "no" between its "ZPLIB" structure and its "TPALIB" structure.

now : desired order to be performed; now≤no.

iflag : an input integer;

iflag = 1 : translate the VTPS from its "ZPLIB" structure, uu, to its "TPALIB" structure, ww;

iflag ≠ 1 : translate the VTPS from its "TPALIB" structure, uu, to its "ZPLIB" structure, ww; setting "iflag = 0" would be good.

5. THE SUB-LIBRARY "TPALIB"

Similar to the use of the sub-library "ZPLIB", to use the "TPALIB", the user has to obtain the compiled "ZLIB" and make a calling statement "call tpaprp(nv,no,nm,npm)" before any subroutine using the data structure of TPALIB is called. Note that slightly different from the ZPLIB, "no" is the order (not the maximum order) while "nv" is the maximum number of variables the user

desires. The same as in the ZPLIB, "nm", is a returned value for the number of monomials, i.e. $nm = (nv + no)!/(nv!no!)$, is returned for the user; "npm" is the maximum number of particles desired for tracking.

Once the statement "call tpaprp(nv,no,nm,npm)" is executed, all the TPS's (u, v, and w) and the VTPS (uu,vv, and ww) are assumed to be order of "no" (although operations may be performed up to orders lower than "no"), but not necessarily to be of nv variables. Usually they are nvw-variable TPS's or VTPS's, where nvw (smaller or equal to nv) is specified as one of the subroutine arguments.

The routines available in the "TPALIB" are as follows.

(a) TPS Operations

Initialization:

(1) subroutine tpazro(w,nmw)

for performing $W(\vec{\mathbf{z}}) = 0$

(2) subroutine tpaconst(c,w,nmw)

for performing $W(\vec{\mathbf{z}}) = c$

if $c = 1$, $W(\vec{\mathbf{z}}) = I(\vec{\mathbf{z}}) = 1$

(3) subroutine tpacpy(u,w,nmw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}})$

(4) subroutine tpasgn(u,w,nmw)

for performing $W(\vec{\mathbf{z}}) = -U(\vec{\mathbf{z}})$

(5) subroutine tpapok1(w,c,iv,nmw)

for performing $W(\vec{\mathbf{z}}) = c * z_{\mathrm{iv}}$

iv: an input positive integer; iv $\leq$ nv

Addition and subtraction:

(6) subroutine tpaadd(u,v,w,nmw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) + V(\vec{\mathbf{z}})$

(7) subroutine tpacadd(c,u,w,nmw)

for performing $W(\vec{\mathbf{z}}) = c + U(\vec{\mathbf{z}})$

(8) subroutine tpasub(u,v,w,nmw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) - V(\vec{\mathbf{z}})$

(9) subroutine tpasubc(u,c,w,nmw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) - c$

(10) subroutine tpacsub(c,u,w,nmw)

for performing $W(\vec{\mathbf{z}}) = c - U(\vec{\mathbf{z}})$

Multiplication and division with scalars:

(11) subroutine tpacmul(c,u,w,nmw)

for performing $W(\vec{\mathbf{z}}) = c * U(\vec{\mathbf{z}})$

(12) subroutine tpadivc(u,c,w,nmw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}})/c$

(13) subroutine tpalin(u,c,v,w,nmw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) + c * V(\vec{\mathbf{z}})$

(14) subroutine tpablin(d,u,c,v,w,nmw)

for performing
$W(\vec{\mathbf{z}}) = d * U(\vec{\mathbf{z}}) + c * V(\vec{\mathbf{z}})$

Multiplication and division:

(15) subroutine tpamul(u,v,w,nvw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) * V(\vec{\mathbf{z}})$

u,v,w: "nvw"-variable TPS's of order "no"; nvw $\leq$ nv.

(16) subroutine tpamulo(u,v,w,now,nvw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) * V(\vec{\mathbf{z}})$

u,v,w: "nvw"-variable TPS's of order "no"; nvw $\leq$ nv.

now: order involved in the operation of the TPS's U,V,W; now $\leq$ no.

(17) subroutine tpadiv(u,v,w,nvw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}})/V(\vec{\mathbf{z}})$

u,v,w: "nvw"-variable TPS's of order "no"; nvw $\leq$ nv.

(18) subroutine tpainv(u,w,nvw)

for performing $W(\vec{\mathbf{z}}) = 1/U(\vec{\mathbf{z}})$

u,w: "nvw"-variable TPS's of order "no"; nvw $\leq$ nv.

(19) subroutine tpasq(u,w,nvw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) * U(\vec{\mathbf{z}})$

u,w: "nvw"-variable TPS's of order "no"; nvw $\leq$ nv.

(20) subroutine tpapwr(u,npwr,w,nvw)

for performing $W(\vec{\mathbf{z}}) = U(\vec{\mathbf{z}}) * *$ npwr

u,w: nvw-variable TPS's of order "no"; nvw $\leq$ nv.

npwr: an input integer for the power to be performed for the TPS U.

Derivative and Integral:

(21) subroutine tpadrv(u,w,nvw,iv)

for performing $W(\vec{\mathbf{z}}) = (d/dz_{\mathrm{iv}})U(\vec{\mathbf{z}})$

iv ($\leq$ nvw): an input integer.

u,w: "nvw"-variable TPS's of order "no"; nvw $\leq$ nv.

(22) subroutine tpabrac(u,v,w,nvw,nd)

for performing $W(\vec{\mathbf{z}}) = \{U(\vec{\mathbf{z}}), V(\vec{\mathbf{z}})\}$,

the poison bracket of U and V.

u,v,w: "nvw"-variable TPS's of order "no"; nvw $\leq$ nv.

nd: sets of canonically conjugate variables; nd $\leq$ nvw/2.

Functions:

(23) subroutine tpasin(u,w,nvw)

for performing $W(\vec{\mathbf{z}}) = \sin(U(\vec{\mathbf{z}}))$

(24) subroutine tpacos(u,w,nvw)

for performing $W(\vec{\mathbf{z}}) = \cos(U(\vec{\mathbf{z}}))$

(25) subroutine tpaexp(u,w,nvw)

for performing $W(\vec{\mathbf{z}}) = \exp(U(\vec{\mathbf{z}}))$

(26) subroutine tpalog(u,w,nvw)

for performing $W(\vec{\mathbf{z}}) = \ln(U(\vec{\mathbf{z}}))$

(27) subroutine tpasqrt(u,w,nvw)

for performing $W(\vec{\mathbf{z}}) = \text{sqrt}\ (U(\vec{\mathbf{z}}))$

u,w: nvw-variable TPS's of order "no"; nvw $\leq$ nv.

(b) VTPS Operations

Initialization:

(28) subroutine tpaunit(ww,nvw)

for performing $W(\vec{\mathbf{z}}) = I(\vec{\mathbf{z}}) = \vec{\mathbf{z}}$

ww: nvw-dimensional, nvw-variable VTPS of order "no".

(29) subroutine rdtpamap(ww,nwb,nw,nvw,imap)

for initializing $\vec{\mathbf{W}}(\vec{\mathbf{z}}) = \sum_{k=0}^{\Omega} \vec{\mathbf{w}}(\vec{\mathbf{k}}) \vec{\mathbf{z}}^{\vec{\mathbf{k}}}$

Initialize an nw-dimensional, nvw-variable VTPS from its nwb$^{\text{th}}$ dimension to nw$^{\text{th}}$ dimension by reading a "TPALIB" structured VTPS (order of "no") file (specified by the number "imap") where data are stored from the nwb$^{\text{th}}$ dimension to the nw$^{\text{th}}$ dimension.

Writing out a VTPS:

(30) subroutine wrtpamap(uu,nub,nu,nvw,imap)

output the coefficients $\vec{\mathbf{u}}(\vec{\mathbf{k}})$ in a file for $\vec{\mathbf{U}}(\vec{\mathbf{z}}) = \sum_{k=0}^{\Omega} \vec{\mathbf{u}}(\vec{\mathbf{k}}) \vec{\mathbf{z}}^{\vec{\mathbf{k}}}$

Write out an nu-dimensional, nvw-variable VTPS from its nub$^{\text{th}}$ dimension to nu$^{\text{th}}$ dimension up to order "no" in "TPALIB" form to a file (specified by the number "imap").

Concatenation of VTPS's:

(31) subroutine tpacncat(uu,vv,ww,nvw)

for performing $\vec{\mathbf{W}}(\vec{\mathbf{z}}) = \vec{\mathbf{V}}(\vec{\mathbf{U}}(\vec{\mathbf{z}}))$

uu, vv, ww : nvw-dimensional, nvw-variable VTPS's of order no; uu represents $\vec{\mathbf{U}}(\vec{\mathbf{z}})$, vv represents $\vec{\mathbf{V}}(\vec{\mathbf{z}})$, ww represents $\vec{\mathbf{W}}(\vec{\mathbf{z}})$.

(00) subroutine tpacnct(uu,vv,nov,ww,nvw)

for performing $\vec{\mathbf{W}}(\vec{\mathbf{z}}) = \vec{\mathbf{V}}(\vec{\mathbf{U}}(\vec{\mathbf{z}}))$

uu, vv, ww : nvw-dimensional, nvw-variable VTPS's of order no; uu represents $\vec{\mathbf{U}}(\vec{\mathbf{z}})$, vv represents $\vec{\mathbf{V}}(\vec{\mathbf{z}})$, ww represents $\vec{\mathbf{W}}(\vec{\mathbf{z}})$.

nov : order of vv actually used in the operation; nov $\leq$ no.

(00) subroutine tpacnctw(uu,vv,ww,now,nvw)

for performing $\vec{\mathbf{W}}(\vec{\mathbf{z}}) = \vec{\mathbf{V}}(\vec{\mathbf{U}}(\vec{\mathbf{z}}))$

uu, vv, ww : nvw-dimensional, nvw-variable VTPS's of order no; uu represents $\vec{\mathbf{U}}(\vec{\mathbf{z}})$, vv represents $\vec{\mathbf{V}}(\vec{\mathbf{z}})$, ww represents $\vec{\mathbf{W}}(\vec{\mathbf{z}})$.

now : order of ww actually desired; now $\leq$ no.

(32) subroutine tpacncto(uu,vv,nov,ww,now,nvw)

for performing $\vec{\mathbf{W}}(\vec{\mathbf{z}}) = \vec{\mathbf{V}}(\vec{\mathbf{U}}(\vec{\mathbf{z}}))$

uu, vv, ww : nvw-dimensional, nvw-variable VTPS's of order no; uu represents $\vec{\mathbf{U}}(\vec{\mathbf{z}})$, vv represents $\vec{\mathbf{V}}(\vec{\mathbf{z}})$, ww represents $\vec{\mathbf{W}}(\vec{\mathbf{z}})$.

nov : order of vv actually used in the operation; nov $\leq$ no.

now : order of ww actually desired; now $\leq$ no.

Inversion of a VTPS:

(33) subroutine tpaminv(uu,ww,now,nvw)

for performing $\vec{\mathbf{W}}(\vec{\mathbf{z}}) = \vec{\mathbf{U}}^{-1}(\vec{\mathbf{z}})$

uu, ww : nvw-dimensional, nvw-variable VTPS's of order no; uu represents $\vec{\mathbf{U}}(\vec{\mathbf{z}})$, ww represents $\vec{\mathbf{W}}(\vec{\mathbf{z}})$.

now : order of ww actually desired; now $\leq$ no.

(c) Tracking

Single-particle tracking:

(34) subroutine tpamtrk(uu,nu,nvw,nou,x,y)

for performing $\vec{\mathbf{y}} = \vec{\mathbf{U}}(\vec{\mathbf{x}})$

uu : an input nu-dimensional, nvw-variable VTPS of order no, although only up to order nou ($\leq$ no) is operated; uu represent $\vec{\mathbf{U}}(\vec{\mathbf{x}})$.

x : an input vector of dimension nvw.

y : an output vector of dimension nu.

actual operations are for $y_i = uu_i(\vec{\mathbf{x}})$ for $i = 1, 2, \ldots,$ nu.

(35) subroutine tpamtrko(uu,nu,nvw,x,y)

for performing $\vec{\mathbf{y}} = \vec{\mathbf{U}}(\vec{\mathbf{x}})$

uu : an input nu-dimensional, nvw-variable VTPS of order no and up to order no is operated; uu represent $\vec{\mathbf{U}}(\vec{\mathbf{x}})$.

x : an input vector of dimension nvw.

y : an output vector of dimension nu.

actual operations are for $y_i = uu_i(\vec{\mathbf{x}})$ for $i = 1, 2, \ldots,$ nu.

** Note that subroutine tpamtrko is faster than subroutine tpamtrk. However tpamtrko cannot be used for tracking up to an order nou smaller than no.

Multi-particle tracking:

(36) subroutine tpamtrks(uu,nu,nvw,nou,np,x,nx,y,ny)

for performing $\vec{\mathbf{y}}^p = \vec{\mathbf{U}}(\vec{\mathbf{x}}^p)$, $p = 1, 2, \ldots,$ np.

uu : an input nu-dimensional, nvw-variable VTPS of order no, although only up to order nou ($\leq$ no) is operated; uu represent $\vec{\mathbf{U}}(\vec{\mathbf{x}})$.

x : array x(nx,np) where nx $\geq$ nv; users should consider it as np input nx-dimensional vectors.

y : array x(nx,np) where nx $\geq$ nv; users should consider it as np output ny-dimensional vectors.

actual operations are for $y_i^p = uu_i(\vec{\mathbf{x}}^p)$

for $i = 1, 2, \ldots,$ nu, and $p = 1, 2, \ldots,$ np.

Scaling:

(37) subroutine tpamscle(uu,nw,nvw,ww,s)

$$\text{for performing } \vec{\mathbf{W}}(\vec{\mathbf{z}}') = \sum_{k=0}^{\Omega} \vec{\mathbf{w}}(\vec{\mathbf{k}})\vec{\mathbf{z}}'^{\vec{\mathbf{k}}} = \vec{\mathbf{U}}(\vec{\mathbf{z}}) = \sum_{k=0}^{\Omega} \vec{\mathbf{u}}(\vec{\mathbf{k}})\vec{\mathbf{z}}^{\vec{\mathbf{k}}} ,$$

$$\text{where } z_i = z_i' * s(i) \text{ for } i = 1, 2, \ldots, \text{nvw}$$

uu, ww : nw-dimensional, nvw-variable VTPS's of order no; uu represents $\vec{\mathbf{U}}(\vec{\mathbf{z}})$, ww represents $\vec{\mathbf{W}}(\vec{\mathbf{z}})$.

s : an input nvw-dimensional vector.

6. SUMMARY AND SUGGESTION

The fundamental and basic operations for the algebra of truncated power series (TPS) have been numerically programmed and gathered in a library entitled "ZLIB". There are two sub-libraries in "ZLIB", the "ZPLIB" and the "TPALIB", with different data structures to provide more flexibility in dealing with a different number of variables and orders simultaneously. The style of the library "ZLIB", being similar to the library "IMSL", may offer the advantage of familiarity to some users. Sample programs using "ZLIB" are available, which could help beginning users. Occasionally, users may need specific operations that cannot be performed with the available routines described in Section 4 and Section 5. Under such a circumstance, users are welcome to call the authors for help.

Beginning users are advised to concentrate on one of the sub-libraries. An NERSC (MFE) Cray computer user who wishes to use routines in the "ZPLIB"

of "ZLIB 1.0" should follow the steps below or its equivalence (assuming the user's file name is "map").

Step 1 : Obtaining "ZLIB" (use one of the following commands)

cfs get zlib:/yan/zlib1.0/zlib for Cray-2
cfs get zlib:/yan/zlib1.0/zlibd for Cray-2 double precision
cfs get zlib:/yan/zlib1.0/zlibe for Cray-XMP
cfs get zlib:/yan/zlib1.0/zlibed for Cray-XMP double precision

Step 2 : cft77 i = map,b = bmap, ...

Step 3 : ldr b = bmap,lib = (zlib,imsl,...), x = xmap

An example of using the routines in the sub-library "TPALIB" is the program "Zmap" which was programmed to extract Taylor maps in a beam line. In particular, "Zmap" can extract one-turn maps from a post-Teapot tracking program "Ztrack".

An example of using the routines in the sub-library "ZPLIB" is the subprogram "OPSMAP" which one (YY) of the authors and his colleagues Ken Kauffman and David Ritson programmed to extract one-turn maps in a tracking program "SSCTRK".

"ZLIB 2.0", which includes routines for the performance of Lie algebraic treatment of beam dynamics such as Dragt-Finn factorization (subroutine zpdragt and zpfinn), nonlinear norm form (subroutine zpforest), etc. will be released once it is well tested.

ACKNOWLEDGEMENTS

The authors wish to point out that there is a "da-package" developed by M. Berz, which also performs differential algebra. Besides the difference in data structure, there are two major differences between "ZLIB" and "da-package":

(a) Optimization for "ZLIB" is primarily on vector and parallel computing while optimization for "da-package" is primarily on scalar computing.

(b) "ZLIB" uses dynamic memory while "da-package" uses decks of memories.

One of the authors (Yiton Yan) thanks E. Forest for many valuable suggestions, J. Irwin for many valuable conversations, and K. Kauffman, T. Sen, and R. Talman for valuable comments. He thanks Alex Chao for continuous support and encouragement.

APPENDIX A

MEMORY PREPARATION SUBPROGRAMS FOR "ZLIB 1.1"

Ocasionally, users would like to prepare ZLIB working memory themselves just as they prepare working memories for some of the "IMSL" routines. In that case, "ZLIB 1.1" instead of "ZLIB 1.0" should be used. In order to reduce errors that might occur due to inappropriate preparation of working memories, the author has written suitable working memory preparation subprograms for the "ZLIB 1.1". To use the routines in "ZLIB", users must load their programs with "ZLIB" just as the "IMSL" library is loaded when "IMSL" routines are used. Users must include at least one subprogram allocating the working memory needed for "ZLIB 1.1" in their program, and must assign suitable integers for four parameters in the parameter statement, of the working memory preparation subprogram(s). The four parameters are:

"nvm": the maximum number of variables.
"nom": the maximum order.
"nmm": the maximum number of monomials;
nmm = (nvm + nom)!/(nvm!nom!).
"npm": the maximum number of particles.

A user program must have a statement that calls the working memory preparation subprogram(s) before the corresponding routines in "ZLIB" are called. For example, to use "ZPLIB" of "ZLIB 1.1", the following statement should be included in the user's program at the very beginning of the executable statements.

"call zpprep(nv,no,nm,np),"

where "nv", "no", and "np" are the number of variables, the order, and the number of particles, which should always be equal to or smaller than "nvm", "nom", and "npm" respectively; "nm", is a returned value for the number of monomials, i.e. nm = (nv + no)!/(nv!no!), is returned for the user. At this stage, if the user makes a mistake in assigning the integer numbers for "nvm", "nom", or "nmm", "ZLIB" will provide messages that will help the user make corrections.

The parameter "nkpm" is calculated in the parameter statement that is guaranteed to be large enough for the corresponding working memories. However, if

both "nvm" and "nom" are large, "nkpm" may become unnecessarily too large. Under such a circumstance, a warning message will be provided but the execution continues. To save computer memory, the user may choose to stop the execution to assign a number suggested by the message for "nkpm" in the parameter statement directly. The user can also look up the table given in Appendix B where "nmm" and "nkpm" are given for given sets of "nvm" and "nom" to assign suitable integer numbers for "nmm" and "nkpm". For a beginning user, try not to be bothered by the warning message.

The following subprogram "zpprep" prepares the working memories for routines in the sub-library "ZPLIB" of "ZLIB 1.1".

```
 subroutine zpprep(nv,no,nm,np)
implicit double precision(a-h,o-z)
parameter (nvm=6,nom=9,nmm=5005, npm=6,
+      no1=nom+1,
+      nov=(nom+2)*no1*nvm,
+      nvno=nom*nvm,
+      njv=(nvm+1)*nmm,
+      nikpm=no1*(nmm-1),
+      navgm=nom/nvm,nrm=nom-navgm*nvm,
+      nkpmx=(navgm+2)**nrm*(navgm+1)**(nvm-nrm),
+      nkpm=nmm*nkpmx,
+      njdm=nvm*nmm*nom/(nvm+nom),
+      nvmsq=nvm*nvm,
+      nmmnp=max(nmm,nvm*npm),
+      nmw=nmm*nvm+6,
+      nmwnp=max(nmw,nmm*npm)
common /strc1/ nmo(no1)
common /strc2/ nmob(no1)
common /strc3/ nmov(nov)
common /strc4/ jv(njv)
common /strc5/ js(nvm)
common /strc6/ nmvo(nvno)
common /strc7/ ivp(nmm)
common /strc8/ jpp(nmm)
common /zptps/ jtpa(nmm)
common /mulp1/ ikp(nikpm)
```

```
common /mulb1/ ikb(nikpm)
common /mulp2/ kp(nkpm)
common /mulp3/ lp(nkpm)
common /drvp1/ jd(njdm)
common /conp1/ jpc(nmm)
common /conp2/ ivpc(nmm)
common /conp3/ ivppc(nmm)
common /conp4/ mp(nov)
common /conloc/ noc(nvm)
common /rdtpa1/ jjp(nmm)
common /mulwk/ wkmul(nmmnp)
common /divwk/ wkdiv(nmw)
common /conwk/ work(nmwnp)
common /mtrx1/ aa(nvmsq)
common /mtrx2/ bb(nvmsq)
common /ccsqrt/ csqrt(nom)
common /ccinvs/ cinv(nom)
common /ccclns/ cln(nom)
common /ccexps/ cexp(nom)
common /csccoe/ csc(nom)
call zpprp(nv,no,nmm,no1,nov,njv,nikpm,nmw,nkpm,njdm,nm)
return
end
```

The following subprogram "tpaprp" prepares the working memories for routines in the sub-library "TPALIB" of "ZLIB 1.1".

```
subroutine tpaprp(nv,no,nm,np)
implicit double precision(a-h,o-z)
parameter (nvm=6,nom=9,nmm=5005,
+      njpm=nmm*nvm,
+      navgm=nom/nvm,nrm=nom-navgm*nvm,
+      nkpm=nmm*(navgm+2)**nrm*(navgm+1)**(nvm-nrm),
+      nmw=nmm*nvm+6)
common /pmul1/ nklp(nmm)
common /bmul1/ nklpb(nmm)
common /pmul2/ kp(nkpm)
common /pmul3/ lp(nkpm)
common /pmul4/ iop(njpm)
```

```
common /pdrv1/ jd(njpm)
common /pdrv2/ jp(njpm)
common /pdrv3/ jo(njpm)
common /mulwk/ wkmul(nmm)
common /divwk/ wkdiv(nmw)
common /conwk/ work(nmw)
call tpa626(nv,no,nmm,nmw,nkpm,njpm,nm)
return
end
```

Occasionally, the user may wish to use routines in the sub-library "ZPLIB" to perform initialization (reading in a VTPS) and tracking only. In such a case, he may choose to use the following working memory preparation subprogram "zptrkp" to save computer memory.

```
subroutine zptrkp(nv,no,nm,np)
implicit double precision(a-h,o-z)
parameter (nvm=6,nom=9,nmm=5005, npm=1,
+      no1=nom+1,
+      nov=(nom+2)*no1*nvm,
+      nvno=nom*nvm,
+      njv=(nvm+1)*nmm,
+      nvp=nvm*npm,
+      nmw=nmm*npm)
common /strc1/ nmo(no1)
common /strc2/ nmob(no1)
common /strc3/ nmov(nov)
common /strc4/ jv(njv)
common /strc5/ js(nvm)
common /strc6/ nmvo(nvno)
common /strc7/ ivp(nmm)
common /strc8/ jpp(nmm)
common /rdtpa1/ jjp(nmm)
common /mulwk/ wkmul(nvp)
common /conwk/ work(nmw)
call trkprp(nv,no,np,nmm,npm,no1,nov,njv,nvp,nmw,nm)
return
end
```

APPENDIX B

PARAMETERS FOR THE PREPARATION OF "ZLIB" WORKING MEMORY

"ZLIB" WORKING MEMORY PARAMETERS

nvm	nom	nmm	nkpm	nvm	nom	nmm	nkpm	nvm	nom	nmm	nkpm	nvm	nom	nmm	nkpm	nvm	nom	nmm	nkpm
10	1	11	21	4	1	5	9	2	1	3	5	1	1	2	3	1	91	92	4278
10	2	66	231	4	2	15	45	2	2	6	15	1	2	3	6	1	92	93	4371
10	3	286	1771	4	3	35	165	2	3	10	35	1	3	4	10	1	93	94	4465
10	4	1001	10626	4	4	70	495	2	4	15	70	1	4	5	15	1	94	95	4560
10	5	3003	53130	4	5	126	1287	2	5	21	126	1	5	6	21	1	95	96	4656
10	6	8008	230230	4	6	210	3003	2	6	28	210	1	6	7	28	1	96	97	4753
10	7	19448	888030	4	7	330	6435	2	7	36	330	1	7	8	36	1	97	98	4851
10	8	43758	3108105	4	8	495	12870	2	8	45	495	1	8	9	45	1	98	99	4950
10	9	92378	10015005	4	9	715	24310	2	9	55	715	1	9	10	55	1	99	100	5050
10	10	184756	30045015	4	10	1001	43758	2	10	66	1001	1	10	11	66	1	100	101	5151
10	11	352716	84672315	4	11	1365	75582	2	11	78	1365	1	11	12	78	1	101	102	5253
9	1	10	19	4	12	1820	125970	2	12	91	1820	1	12	13	91	1	102	103	5356
9	2	55	190	4	13	2380	203490	2	13	105	2380	1	13	14	105	1	103	104	5460
9	3	220	1330	4	14	3060	319770	2	14	120	3060	1	14	15	120	1	104	105	5565
9	4	715	7315	4	15	3876	490314	2	15	136	3876	1	15	16	136	1	105	106	5671
9	5	2002	33649	4	16	4845	735471	2	16	153	4845	1	16	17	153	1	106	107	5778
9	6	5005	134596	4	17	5985	1081575	2	17	171	5985	1	17	18	171	1	107	108	5886
9	7	11440	480700	4	18	7315	1562275	2	18	190	7315	1	18	19	190	1	108	109	5995
9	8	24310	1562275	4	19	8855	2220075	2	19	210	8855	1	19	20	210	1	109	110	6105
9	9	48620	4686825	4	20	10626	3108105	2	20	231	10626	1	20	21	231	1	110	111	6216
9	10	92378	13123110	4	21	12650	4292145	2	21	253	12650	1	21	22	253	1	111	112	6328
9	11	167960	34597290	4	22	14950	5852925	2	22	276	14950	1	22	23	276	1	112	113	6441
9	12	293930	86493225	4	23	17550	7888725	2	23	300	17550	1	23	24	300	1	113	114	6555
8	1	9	17	4	24	20475	10518300	2	24	325	20475	1	24	25	325	1	114	115	6670
8	2	45	153	4	25	23751	13884156	2	25	351	23751	1	25	26	351	1	115	116	6786
8	3	165	969	4	26	27405	18156204	2	26	378	27405	1	26	27	378	1	116	117	6903
8	4	495	4845	4	27	31465	23535820	2	27	406	31465	1	27	28	406	1	117	118	7021
8	5	1287	20349	4	28	35960	30260340	2	28	435	35960	1	28	29	435	1	118	119	7140
8	6	3003	74613	4	29	40920	38608020	2	29	465	40920	1	29	30	465	1	119	120	7260
8	7	6435	245157	4	30	46376	48903492	2	30	496	46376	1	30	31	496	1	120	121	7381
8	8	12870	735471	3	1	4	7	2	31	528	52360	1	31	32	528	1	121	122	7503
8	9	24310	2042975	3	2	10	28	2	32	561	58905	1	32	33	561	1	122	123	7626
8	10	43758	5311735	3	3	20	84	2	33	595	66045	1	33	34	595	1	123	124	7750
8	11	75582	13037895	3	4	35	210	2	34	630	73815	1	34	35	630	1	124	125	7875
8	12	125970	30421755	3	5	56	462	2	35	666	82251	1	35	36	666	1	125	126	8001
8	13	203490	67863915	3	6	84	924	2	36	703	91390	1	36	37	703	1	126	127	8128
7	1	8	15	3	7	120	1716	2	37	741	101270	1	37	38	741	1	127	128	8256
7	2	36	120	3	8	165	3003	2	38	780	111930	1	38	39	780	1	128	129	8385
7	3	120	680	3	9	220	5005	2	39	820	123410	1	39	40	820	1	129	130	8515
7	4	330	3060	3	10	286	8008	2	40	861	135751	1	40	41	861	1	130	131	8646
7	5	792	11628	3	11	364	12376	2	41	903	148995	1	41	42	903	1	131	132	8778
7	6	1716	38760	3	12	455	18564	2	42	946	163185	1	42	43	946	1	132	133	8911
7	7	3432	116280	3	13	560	27132	2	43	990	178365	1	43	44	990	1	133	134	9045
7	8	6435	319770	3	14	680	38760	2	44	1035	194580	1	44	45	1035	1	134	135	9180
7	9	11440	817190	3	15	816	54264	2	45	1081	211876	1	45	46	1081	1	135	136	9316
7	10	19448	1961256	3	16	969	74613	2	46	1128	230300	1	46	47	1128	1	136	137	9453
7	11	31824	4457400	3	17	1140	100947	2	47	1176	249900	1	47	48	1176	1	137	138	9591
7	12	50388	9657700	3	18	1330	134596	2	48	1225	270725	1	48	49	1225	1	138	139	9730
7	13	77520	20058300	3	19	1540	177100	2	49	1275	292825	1	49	50	1275	1	139	140	9870
7	14	116280	40116600	3	20	1771	230230	2	50	1326	316251	1	50	51	1326	1	140	141	10011
7	15	170544	77558760	3	21	2024	296010	2	51	1378	341055	1	51	52	1378	1	141	142	10153
6	1	7	13	3	22	2300	376740	2	52	1431	367290	1	52	53	1431	1	142	143	10296
6	2	28	91	3	23	2600	475020	2	53	1485	395010	1	53	54	1485	1	143	144	10440
6	3	84	455	3	24	2925	593775	2	54	1540	424270	1	54	55	1540	1	144	145	10585
6	4	210	1820	3	25	3276	736281	2	55	1596	455126	1	55	56	1596	1	145	146	10731
6	5	462	6188	3	26	3654	906192	2	56	1653	487635	1	56	57	1653	1	146	147	10878
6	6	924	18564	3	27	4060	1107568	2	57	1711	521855	1	57	58	1711	1	147	148	11026
6	7	1716	50388	3	28	4495	1344904	2	58	1770	557845	1	58	59	1770	1	148	149	11175
6	8	3003	125970	3	29	4960	1623160	2	59	1830	595665	1	59	60	1830	1	149	150	11325
6	9	5005	293930	3	30	5456	1947792	2	60	1891	635376	1	60	61	1891	1	150	151	11476
6	10	8008	646646	3	31	5984	2324784	2	61	1953	677040	1	61	62	1953	1	151	152	11628
6	11	12376	1352078	3	32	6545	2760681	2	62	2016	720720	1	62	63	2016	1	152	153	11781
6	12	18564	2704156	3	33	7140	3262623	2	63	2080	766480	1	63	64	2080	1	153	154	11935
6	13	27132	5200300	3	34	7770	3838380	2	64	2145	814385	1	64	65	2145	1	154	155	12090
6	14	38760	9657700	3	35	8436	4496388	2	65	2211	864501	1	65	66	2211	1	155	156	12246
6	15	54264	17383860	3	36	9139	5245786	2	66	2278	916895	1	66	67	2278	1	156	157	12403
6	16	74613	30421755	3	37	9880	6096454	2	67	2346	971635	1	67	68	2346	1	157	158	12561
6	17	100947	51895935	3	38	10660	7059052	2	68	2415	1028790	1	68	69	2415	1	158	159	12720
6	18	134596	86493225	3	39	11480	8145060	2	69	2485	1088430	1	69	70	2485	1	159	160	12880
5	1	6	11	3	40	12341	9366819	2	70	2556	1150626	1	70	71	2556	1	160	161	13041
5	2	21	66	3	41	13244	10737573	2	71	2628	1215450	1	71	72	2628	1	161	162	13203
5	3	56	286	3	42	14190	12271512	2	72	2701	1282975	1	72	73	2701	1	162	163	13366
5	4	126	1001	3	43	15180	13983816	2	73	2775	1353275	1	73	74	2775	1	163	164	13530
5	5	252	3003	3	44	16215	15890700	2	74	2850	1426425	1	74	75	2850	1	164	165	13695
5	6	462	8008	3	45	17296	18009460	2	75	2926	1502501	1	75	76	2926	1	165	166	13861
5	7	792	19448	3	46	18424	20358520	2	76	3003	1581580	1	76	77	3003	1	166	167	14028
5	8	1287	43758	3	47	19600	22957480	2	77	3081	1663740	1	77	78	3081	1	167	168	14196
5	9	2002	92378	3	48	20825	25827165	2	78	3160	1749060	1	78	79	3160	1	168	169	14365
5	10	3003	184756	3	49	22100	28989675	2	79	3240	1837620	1	79	80	3240	1	169	170	14535
5	11	4368	352716	3	50	23426	32468436	2	80	3321	1929501	1	80	81	3321	1	170	171	14706
5	12	6188	646646	3	51	24804	36288252	2	81	3403	2024785	1	81	82	3403	1	171	172	14878
5	13	8568	1144066	3	52	26235	40475358	2	82	3486	2123555	1	82	83	3486	1	172	173	15051
5	14	11628	1961256	3	53	27720	45057474	2	83	3570	2225895	1	83	84	3570	1	173	174	15225
5	15	15504	3268760	3	54	29260	50063860	2	84	3655	2331890	1	84	85	3655	1	174	175	15400
5	16	20349	5311735	3	55	30856	55525372	2	85	3741	2441626	1	85	86	3741	1	175	176	15576
5	17	26334	8436285	3	56	32509	61474519	2	86	3828	2555190	1	86	87	3828	1	176	177	15753
5	18	33649	13123110	3	57	34220	67945521	2	87	3916	2672670	1	87	88	3916	1	177	178	15931
5	19	42504	20030010	3	58	35990	74974368	2	88	4005	2794155	1	88	89	4005	1	178	179	16110
5	20	53130	30045015	3	59	37820	82598880	2	89	4095	2919735	1	89	90	4095	1	179	180	16290
5	21	65780	44352165	3	60	39711	90858768	2	90	4186	3049501	1	90	91	4186	1	180	181	16471