LBNL-49625 — Rev. 1
LCLS-TN-04-03

A User Manual for GINGER and its Post-Processor XPLOTGIN

William M. Fawley
Lawrence Berkeley National Laboratory

Version 1.4f — April 2004

Acknowledgments

The author has benefited from many useful discussions on simulation modeling from his colleagues in the general FEL community. In particular, he would like to acknowledge E.T. Scharlemann, J. Wurtele, P. Pierini, H.-D. Nuhn, K.-J. Kim, M. Xie, J. Eddighoffer, A. Zholents, Z. Huang, S. Reiche, L. Mezi and W. Graves as having contributed in one way or another to GINGER's development over the past 18(!) years.

Work on the GINGER simulation code has been supported by the Director, Office of Science, Offices of Basic Energy Sciences and High Energy and Nuclear Physics, of the U.S. Department of Energy under Contracts No. DE-AC03-76SF00098 to LBNL and DE-AC03-76SF0015 to SLAC. Computational resources have been provided in part by NERSC. The author also wants to thank the Accelerator and Fusion Research Division (AFRD) and the Center for Beam Physics (CBP) at LBNL and the Stanford Synchrotron Research Laboratory (SSRL) for support related to GINGER development.

Copyright & Distribution Restrictions

Disclaimer

Contents

1 Overview of GINGER and this manual

1.1 Introduction to GINGER

GINGER is a multidimensional ($r-z-t$ fields, $x-y-z-t$ macroparticles), polychromatic FEL simulation code developed over the past 18 years. GINGER directly descends from FRED, the original LLNL 2-D FEL simulation code. FRED was a single-pass amplifier particle-in-cell (PIC) code which modeled the interaction between electrons in one ponderomotive well and a monochromatic, r- and z-dependent electromagnetic wave. By monochromatic, we mean that all field quantities (and many particle quantities such as the particle bunching) vary *exactly* as $\exp(-i\omega_o t)$. Other quantities such as beam current and energy are presumed to be approximately time-invariant over "slow" time scales (*i.e.* when averaged over $\sim$ dozens of wave periods). Hence, FRED and its monochromatic descendents (*e.g.* FRED3D and the harmonic code NUTMEG) are useful in modeling FEL's where shot noise, slippage, current and energy variations, and sideband growth may be neglected.

GINGER was developed in the mid-1980's primarily to examine the consequences of sideband growth in single-pass amplifiers. Soon after, a shot noise package was added to examine the minimum excitation level of sidebands and to model SASE growth in the LLNL microwave FEL experiments ELF (35-GHz) and IMP (140- and 250-GHz). In the early 1990's, GINGER began to be used for both SASE x-ray FEL studies (*i.e.* LCLS at SLAC and TESLA-FEL at DESY) and for modeling some of the longer wavelength proof-of-principle SASE experiments which have been done at UCLA/LANL, Brookhaven, and Argonne. In the last couple years modifications to GINGER have been primarily directed toward increasing platform independenc, giving it greater ability to model more exactly actual experimental conditions, and the capability to model relatively complicated scenarios such as harmonic cascade FEL's. GINGER remains a work in progress and the user should check for recent additions/changes by e-mailing the author or talking to other "power" users of the code. Additionally, if the user has access to NERSC (see §2.2), checking the **README** and **CHANGES** files in the public archive space may also be useful.

1.2 Purpose and General Contents of this Manual

This manual is intended to give a brief introduction to the physics and necessary input parameters relevant to the GINGER and its graphical postprocessor XPLOTGIN. The manual presumes that the interested reader/user has a reasonably thorough knowledge of FEL physics, including those of time-dependent (*i.e.* polychromatic) effects such as slippage, sideband generation, and self-amplified spontaneous emission emission (SASE). Throughout the following text, input parameter names are typeset in **red bold Courier font**, input parameter values in **blue**, and user input to the console in **green**. Recent changes since the last manual version are indicated by **orange bars** in the right margin. Since the first version of this manual was written in 1996, GINGER and its

postprocessor have been extensively rewritten in Fortran90 and have become more modularized to aid in porting to different hardware platforms. Additional features, such as the ability to exploit (in certain situations) multiple processor capabilities of the massively parallel machines at NERSC, have been added.

The remainder of this manual is organized as follows. The following subsections detail recent changes/additons to GINGER. Section 2 gives instructions on both how to obtain and how to run GINGER and its post-processor XPLOTGIN while Section 3 gives details concerning the types of runs possible and the input file variables which define and control a specific GINGER run. A number of sample input files are shown in §3.14 which will help a beginning GINGER user get up and running. Section 4 describes the post-processor and a user-modifiable "preferences" file which control the types and details of graphical and text output. Section 5 describes GINGER's physics, structure, underlying assumptions, and thus limitations.

1.3 Changes from User Manual Version 1.3 to 1.4 (December 2003)

- Ability to input namelist variable values directly from a GINGER execute line (§2.3)
- Ability to import time-dependent, multiple slice, 5D phase space information from either user-supplied "simple" ASCII-formatted files or from **`ELEGANT`** output files (§3.3.7)
- Improved namelist error diagnostics (§3.2)
- Improved ability to use externally-generated $t-$dependent electron beam envelope parameters, including those generated by the **`elegant2genesis`** code (§3.3.6).
- Ability to specify initial $t-$dependent radiation power in short pulse mode (§3.4.3)
- Ability to keep the radiation beam artificially aligned with the electron beam centroid in presence of transverse drifts/offsets
- Improved and extended restart capability, both in "time-segmented" mode and for multiple undulators (*e.g.* harmonic cascades) (§3.3.9, §3.4.5)
- Ability to use externally-generated wakefield information (§3.6.1)
- Ability to output macroparticle phase space dumps at user-specified $z-$ *and* $t-$locations (*i.e.* in polychromatic mode); a new ASCII format option now exists also (§3.13.2)
- Ability to artifically "lock" radiation transverse centroid to that of e-beam (§3.5.10)
- Full graphic Windows post-processor using DISLIN graphics (§2.6)

1.4 Changes from User Manual Version 1.2 to 1.3 (Dec. 2001)

- Help information from command line (-h option)

- New template file capability (-t option) to set "base" run parameters
- Windows (including non-graphical post-processor) and Linux ports
- Generation of and use of wiggler error lattice files including steeering and BPM errors
- Output diagnostics of microbunching at higher harmonics
- Multi-processor capability in short pulse mode
- Creating and restarting from z-dependent radiation field files (§3.4.6)
- User-chosen input preference file for post-processor
- Options to generate special purpose SDDS output files from post-processor including $\tilde{E}(r,t)$, $P(\lambda,z)$

1.5 Changes from User Manual Version 1.1 to 1.2 (Dec. 2000)

- Ports to massively-parallel processor platforms (Cray-T3E and IBM-SP)
- Initial capability to output SDDS-formatted files from post-processor
- Discrete external magnet description for periodic lattice
- Ability to input electron beam Twiss parameters
- Improved shot noise initialization including higher harmonics

2 Running GINGER and its post-processor XPLOTGIN

2.1 Hardware Portability and Workstation Access

GINGER and its graphics postprocessor XPLOTGIN are written in Fortran90 and are targeted (preferentially) toward UNIX platforms. Access to and use of GINGER are particularly easy if the user has an account at NERSC (National Energy Research Supercomputer Center), which is funded by the Office of Science in the U.S. Department of Energy.

Alternatively, since both codes compile, link, and run on many UNIX (*e.g.* Sun/Solaris; SGI/IRIX; IBM/AIX) and Linux (*e.g.* X86/Intel&Portland Group F90 compilers; Alpha/DEC F90 Compiler) workstations with standard F90 compilers, arrangements can be made with the author by serious users (from US govt. defined "non-sensitive" countries only, however) to obtain (or make) executables to run on their own desktop/laptop computers. Successful ports of GINGER have also been made Windows NT (Alpha/DEC F90 compiler), DOS-Windows95/98/ME (Lahey F95 compiler), and MacOS-X (PPC/Absoft compiler). Use of the multipass oscillator capability in GINGER currently requires a few matrix subroutines from the commercial IMSL libraries.

Under UNIX/LINUX, postprocessor relies upon a set of graphics subroutines which are built on top of the NCAR graphics libraries. tor graphics for Windows users, a port employing DISLIN routines was been done in 2003. Some output routines in XPLOTGIN also rely upon the publicly available HDF library from NCSA. A non-graphical version of the post-processor which substitutes tabular output (*e.g.* ASCII and/or SDDS files; see §4.7 and §4.8) for graphics is also available for Windows. At present, source code for the postprocessor is freely available from the author. However, source code is normally *not* available for GINGER itself, due both to U.S. Government export control considerations, and the author's desire to avoid the generation/proliferation of incompatible and/or possibly buggy versions.

2.2 Public Access to GINGER at NERSC

At NERSC, executables for both GINGER and XPLOTGIN, together with some sample input files may be copied (using the system **`hsi`** program) from the publicly-readable HPSS directory named **`/nersc/mp40/fawley/pub`**. Currently, executables for the IBM-SP, DOS/Windows(X86) and Linux(X86) should be available from this directory. Since NERSC seems to change its HPSS directory structure surprisingly often, you might need to contact me (e-mail: ***`WMFawley@lbl.gov`***) if you have problems locating the current equivalent of this directory. The **`pub`** directory contains both a **`README`** file with information concerning other files in the directory and also a **`CHANGES`** file which attempts to list the more important modifications made to GINGER and XPLOTGIN. I will attempt to keep both older versions of these codes (in the **`/OLD`** subdirectory which also contains Cray-J90 and Cray-T3E versions) and reasonably "fresh" ones with more current modifications. Please e-mail me if you have problems with either access or version compatibility.

2.3 GINGER Execute Lines and Standard Options

Ignoring temporarily the case of multiprocessor runs on MPP platforms, the command line to begin a GINGER run from a terminal window is: **`xginger -r`** ***`run_name`*** **`[`***Options***`]`**
Typing **`xginger`** alone or **`xginger -h`** will echo to the terminal window the source version and some simple help instructions. Beginning with GINGER versions dated October 2001 or later, the **`-r`** ***`run_name`*** option string *must* be given on the execute line. Here ***`run_name`*** is a 1-24 character alphanumeric string (*e.g.* "**`elf3a`**"). It must not start with a dash("-") nor contain characters such as "**`/*=+()`**" which could play havoc with the system shell. Longer than 24-character ***`run_name`***'s will be truncated with possibly nasty consequences. The ***`run_name`*** both identifies the run and acts as a substring contained within the name of various disk files GINGER creates (*e.g.* "**`pltelf3a`**") or reads. If the input file is *not* specified by the **`-i`** option, GINGER presumes that a proper input file named **`in`*****`run_name`*** (*e.g.* **`inelf3a`**) exists within the directory from which GINGER is being run.

In addition to the **-r** (required) option, other execute line options include:

-i *infile* Here ***infile*** is the name of the input file which will be read by **xginger** and will override the name corresponding to the ***run_name***.

-b *bwfile* Here ***bwfile*** is the name of the tapered wiggler field file containing $a_w(z)$ to be either read (**idesign=0**) or to be created (**idesign=2**) (only possible when running in monochromatic FRED-mode). See §3.5.3 for additional details.

-t template_file A *template_file* is a "normal" GINGER input file which contains the usual header lines and namelists. It is read *before* the usual input file (whose name is set by the ***run_name*** or **-i** option). The purpose of the *template_file* is to set various common input parameters for a general class of runs (*e.g.* standard LCLS parameters). The normal input file is subsequently read and can be used to override some of the specific namelist variable values set by the *template_file* and/or set additional values. For those users of the object-oriented mindset, the *template_file* may be thought of as a "base class" whose namelist variable values the normal input file "inherits" and then can optionally extend.

For example, the *template_file* might have **ntestp=2048** but the input file might set **ntestp=8192**, **nfold_sym=12**, and **nhar_io = 1 3 5** in order to examine fifth harmonic bunching properties. The "final" values actually used in the GINGER run will be properly echoed to the *pltfile* output data file. An example using templates is given in §3.14.5.

-v { special string input } The **-v** option became available in January 2003 and provides a simple mechansim by which the user can specify a small number of input variables in the **&in** namelist *directly* on the console/terminal execute line. For example, **-v { ntestp = 6144 nfold_sym = 12 }** will set **ntestp** to 6144 and **nfold_sym** to 12, irrespective of whatever values were chosen in the input file (and template file if present). The purpose of this feature is to allow the user to make an "on the fly" change to the input values and can also be used in batch scripts for elaborate multi-run parameter scanning. The input string should be curly-bracket delimited (*i.e.* {}). Note that for some UNIX shells, it may be necessary to precede each of the curly brackets by a backslash. Character-type namelist variables may also require "escaping" the shell by preceding single quotes which enclose the chosen value with a backslash. At present a bracket-enclosed special input string should be 80 characters or less (including brackets).

-f fldfile A ***fldfile*** contains radiation field information at the undulator exit from a previous GINGER run and is used to initialize a new runs such as might be required in a radiator/modulator configuration (see §3.4.5 for details) or a multiple segment run (see §3.9).

-rs rstfile A ***rstfile*** contains 6D macroparticle phase space information at the undulator exit of a previous GINGER run. This file will be used (in the present run) to initialize the particle distribution entering a new undulator such as might be required in a modulator/radiator and/or harmonic cascade configurations (see §3.3.9 for details).

-h As mentioned above, this option leads to GINGER typing out some simple "help" instructions to the user console and then exiting.

As GINGER initializes, it first echoes to the user console the underlying source version (*e.g.* **GINGER source version $->$ gnx_20010201a**), the various header lines at the top of the input file, and then some general characteristics of the electron beam, radiation field, and magnetic wiggler. While running, GINGER creates a so-called *pltfile* (*e.g.* **pltelfa3a**) containing the output field and particle diagnostics which will be subsequently analyzed by the postprocessor. This file is currently in ASCII format and can exceed 10 megabytes if **nside**, the number of electron beam slices, is large (*e.g.* ≥ 192). When running on single processors in periodic boundary condition mode, a binary format *parfile* may be created for temporary storage of particle information. This file can be safely deleted at the conclusion of the run. When GINGER is instructed to dump out macroparticle phase space information at various z-locations via the input variables such as **nspec** or **z_scatterplot**, a binary format *spcfile* is created (*e.g.* **spcelfa3a**) which can be read by the post-processor to create macroparticle scatter plots (see §4.13). When requested (via the **l_debug=.t** input switch), GINGER can also create an ASCII *debugfile* which is a catch-bag of normally obscure numerical integrator diagnostics.

2.4 Running GINGER on DOS/Windows Machines

Although the author frowns upon usage of operating systems from the Evil Empire of Redmond, such occurs unfortunately (even by supposedly intelligent FEL scientists who should know better!). GINGER will now run under Windows via a DOS command window/terminal. I strongly suggest getting the free **Cygwin** package (URL **http://cygwin.com**) which provides a reasonably robust UNIX-like environment. Presuming you have obtained a Windows executable named **xginger.exe**, you would run it with the exact same command line as on a UNIX box except obviously replace the name "xginger" with "xginger.exe" (of course, you could always make a soft link under Cygwin to allow you to use the name "xginger"). In single slice FRED-mode, GINGER *itself* can create a simple *datfile* containing simple ASCII tables of items such as radiation power, rms delta gamma, microbunching fraction b versus z via the input parameter **l_datfile**. This logical switch is defaulted to **.true.** for the DOS version. One can then use **gnuplot** or **Excel** to plot this file directly without having to use the post-processor at all.

2.5 Running GINGER on Massively Parallel Processors at NERSC

GINGER was first ported to the NERSC Cray-T3E in 1999 and then to the IBM-SP (*seaborg.nersc.gov*) in 2000. Given the retirement of *all* NERSC Crays (*i.e.* the Cray-2, C90, J90, and the T3E), one has no choice when running at NERSC but to use the IBM-SP (for which both serial and MPP

versions exist). IBM-SP executables are available in the the author's public HPSS file space (see §2.2). GINGER can effectively use multiple processors for the following types of runs: (1) In polychromatic mode, both "long-pulse" runs (*i.e.* periodic boundary conditions in time) and short pulse "transit-time" (non-periodic BC) runs. (2) In monochromatic FRED-mode, multiple slice, "parameter-scanning" runs (see §3.11). In serial (*i.e.* single-processor) mode, all run modes of GINGER should work properly: monochromatic single slice and multi-slice parameter-scanning FRED-mode; single- and multipass, monochromatic and polychromatic oscillator mode; polychromatic short pulse "transit-time" and periodic BC "long-pulse" modes. Earlier tests by H.-D. Nuhn of SLAC on the Cray-T3E for LCLS-type runs showed nearly exact linear speed-up as the number of processors used increased from 2 to 64.

2.5.1 IBM-SP Multiprocessor Execute Lines

Multiple processors can be used effectively to run GINGER on the IBM-SP. Since the IBM-SP version of GINGER (which contains "mpp" in its name) in the public HPSS directory was compiled to be "MPP-ready" (*i.e.* one does not need to use "POE"), one uses a normal (*i.e.* single processor) execute line for GINGER together (in interactive mode) with the additional phrase **`-procs NPROC`** where **`NPROC`** is the number of processors requested. **`NPROC`** should be an integer factor of **`nside`**, the number of electron beam slices in the run. For example, when **`nside=64`**, permitted choices for **`NPROC`** are 2, 4, 8, 16, 32, 64 but not 3, 17, 24, 36, *etc.* . At present, GINGER creates **`NPROC`** separate output *pltfiles*, numbered 000 and up. For example, **`xginger-mpp -r palac -procs 8`** will create 8 separate *pltfiles*, whose names begin with **`plt000palac`** and end with **`plt007palac`**. For later analysis by the postprocessor, these pltfiles *must* be concatenated together into one big, single file. The *plt000...* file also must be at the head of the resultant concatenated file. The UNIX **`cat`** command is probably the simplest way to do this: *e.g.* **`cat plt0*palac > pltpalac`** will put all the subfiles together in the correct order.

With the arrival at NERSC of the newest version of the IBM-SP (*seaborg.nersc.gov*) which is composed of 16-processor SMP nodes, it is most sensible to run in MPP batch mode with multiples of 16 processors (in any case, you will be charged for the full 16 processors of each node). In batch mode, it is not necessary to use **`-procs NPROC`** because most batch scripts will set the number of tasks (*i.e.* processors) used per node (see §3.14.6 for a sample MPP GINGER batch script for use at NERSC).

The IBM-SP supports NCAR graphics and a version of the post-processor XPLOTGIN is available via **`hsi`** access of HPSS. Consequently, it is not necessary to export output *pltfiles* to another machine for post-processing.

2.5.2 Cray-T3E Multiprocessor Execute Lines

Although NERSC has recently retired its Cray-T3E, perhaps certain users have access to other CRAY MPP platforms. To run in multiprocessor mode (at NERSC at least), the user should precede the normal execute line (*i.e.* **`xginger r=...`**) with the phrase **`mpprun -n NPROC`** where ***NPROC*** ≥ 1 is the number of T3E processors requested. The T3E version of GINGER has not been updated since early 2002 and interested users should contact the author concerning the possibility of compiling an up-to-date T3E version.

2.6 Postprocessor XPLOTGIN Execute Line and Options

The execute line for the postprocessor XPLOTGIN is:
`xplotgin -r run_name` **[***OPTIONS***]**
The postprocessor presumes that a file named **`pltrun_name`** (*e.g.* **`pltelf3a`**) exists in the working directory. Presuming that the postprocessor can generate NCAR or DISLIN graphics, the user may choose a particular output device/file by optionally adding to the XPLOTGIN execute line an uppercase mnemonic. For NCAR graphics under UNIX/LINUX, acceptable devices are **`X11`** for direct output to an X-windows screen, **`CGM`** to generate a CGM graphics file, or **`POST`** to generate a Postscript file. When running under Windows with the DISLIN graphics package, permitted output devices include: **`X11`** which produces screen output with a black bankground, **`CONS`** which produces screen output with a white background, **`POST`** for a postscript file, and **`PDF`** for a PDF file readable with the Adobe Acrobat reader. In addition to the DISLIN-capable executable, the user must also have the **`disdll`** library in an appropriate location in the Windows directory space. This library is available either from the author or directly from the DISLIN authors via the URL **`http://www.linmpi.mpg.de/dislin/`** .

When running the postprocessor at NERSC, by default graphical output goes to a CGM file named ***run_name*`.cgm`** (*e.g.* **`elf3a.cgm`**). When running the postprocessor at NERSC (especially under "batch" mode), be sure that the list of your loaded "modules" includes the NCAR package. Use the **`module list`** command to check which modules are actually loaded; if "NCAR" is not listed, type **`module load ncar`**. On UNIX/LINUX workstations, the default is X11 output which leaves no residual file when XPLOTGIN finishes. Consequently, the user should use the option **`CGM`** or **`POST`** instead if one expects to look at or use the output later. For those user running under LINUX or UNIX who do *not* have the full NCAR utilities package loaded, the postscript output option is preferrred because no NCAR utilities are needed to examine the graphics file (see below). For example, the **`gv`** and/or **`ghostview`** programs are normally available under UNIX/LINUX to view and print postscript files.

If NCAR applications are available to the user, one or more of the **`ctrans`** application family can be used both to view output CGM files and to plot individual frames to various device drivers

such as X11, Postscript, *etc.*. The **idt** program is particularly useful under X11 for CGM files because one can scroll interactively through the frames in a given file, simultaneously examine multiple frames from one or more files, and do some rudimentary animation on screen.

A "preferences" file, if present in the local working directory, will be read by the post-processor and then used to control various plotting options and the generation of additional output files (*e.g.* SDDS format files) for further analysis. The default name for the preferences file is **xplotgin.pref** but this name can be optionally overridden by typing on the XPLOTGIN execute line **-pref *pref_file***. Because many specialized features of the post-processor are now controlled by inputing variable values in the preferences file (see §4.1 for more details), the user should make the effort to master the usage of this file.

Significant effort has been spent to ensure *upward* compatibility of *pltfiles* from "old" GINGER runs with new versions of the post-processor. Please communicate any compatibility problems encountered in this area (*i.e.* new postprocessor executable aborting when analyzing an old *pltfile*). On the other hand, old versions of the post-processor can be seriously incompatible with *pltfiles* created by more recent versions of GINGER both because the data format in the *pltfiles* occasionally changes (*e.g.* new variables are written out) and because occasionally new variables are added to the **&POSTPROC** namelist located near the begining of the *pltfile*. GINGER versions since fall 2002 now insert into the *pltfile* the required "minimum" version of the post-processor. Similarly, post-processor versions of comparable or newer vintage check for this and will indicate immediately to the user if a new post-processor version is required to reduce the *pltfile*.

If, due to a user mistake or whatever, either GINGER or the XPLOTGIN tries to open an input or other such file which is *not* present on disk, an error message will be sent to the console terminal and the user can type in a new name. Alternatively, the user can type **end** and the code will exit. Obviously, if one runs the code in either "background" or batch mode, recovery from such an such an error is difficult, if not impossible. Under Unix/Linux, one should kill the process (*i.e.* **kill -9 *pid*** where ***pid*** is the process number). Typical run times for the post-processor are of order one minute or less but can become larger if the *pltfile* is huge (*i.e.* $\geq$ 20 MB) or significant post-processing is needed (*e.g.* far-field mode calculations).

3 GINGER "Modes" and Input File Details

3.1 Overview of GINGER "Run Modes"

Over the past decade, GINGER has evolved from a code targeted solely toward modeling side-band growth in single pass amplifiers to a beast with far greater pretensions. The author considers these as "run modes" and they may be broken down into various classes via different criteria. The first criterion is monochromatic simulation (known as "FRED mode" which is set by the input variable **`lfred=.t`**) as compared with polychromatic simulation ("time-dependent mode" with **`lfred=.f`**) involving a discrete band of wavelengths centered upon a central, usually resonant wavelength.

A second criterion is the type of FEL configuration. Normally, GINGER models single-pass devices but is also capable of modeling (in both FRED- and time-dependent modes) multi-pass oscillators (§3.8) and, with far more effort, more complicated, multiple undulators configurations (*e.g.* oscillator-radiator combinations, harmonic "cascades"). Both simple drift space and dispersive/optical klystron sections (§3.7.2) may comprise part of an undulator.

For time-dependent simulations, the user chooses either the default "long-pulse" mode, in which case periodic boundary conditions are applied in time, or non-periodic mode. Within the latter there are now a number of different cases possible. First, there is "short-pulse" mode (chosen via the input variable **`ltransit=.t`**) in which the electron beam length is comparable to the slippage length in the undulator. Here, the temporal window normally includes the entire electron beam pulse (+ slippage) and the user must specify a longitudinal current profile (§3.3.5). A new option within non-periodic mode is the so-called "segment mode" (chosen via the input variable **`l_segment_mode=.t`**) which divides a relatively long electron beam into multiple, continguous segments, with radiation field information being effectively passed from one segment onto the next. This mode is more fully described in §3.4.6 and §3.9. Each of these time-dependent modes can also incorporate external time-dependent beam envelope or macroparticle data (see §3.3.6 and 3.3.8). There are also options to create and use various types of "restart" files (§3.3.9 and 3.4.5) ; however, beginning GINGER users are advised to postpone their use until they master run the code on simpler problems.

Within FRED-mode, the user normally models single-pass amplifier configurations. However, one can also model multi-pass oscillators or use special restart radiation field files created by time-dependent runs (§3.4.5). One may also do a "parameter scanning" run (§3.11) to study output power sensitivity to individual input parameters such as electron beam current or undulator strength a_w. In FRED-mode, one can also do multislice runs utilizing time-dependent beam enevelope information from external "data" files.

3.2 Namelist Input Information

GINGER uses the "namelist" capability of Fortran which (hopefully) minimizes the work required to keep a given input file "runnable". Fortran90 prefers that the namelist identifier be preceded by an ampersand (*e.g.* **`&in`**) rather than a dollar sign (*e.g.* **`$in`**), and that the end of the namelist be specified by a slash (*e.g.* "**`/END`**") (although, depending upon the F90 compiler, dollar signs may work in both cases). For some versions of the F90 compiler, when giving input to set a one-dimensional multi-element array (*i.e.* vector) it may be necessary to specify indices. For example, **`iseeds(1:2) = 342845 663857`**. Many F90 compilers (*e.g.* NERSC Crays) have quite limited ability to handle arrays of "TYPE" structures in F90 namelists. This limitation has forced some rather inelegant (*i.e.* ugly) coding for inputting items such as periodic drift spaces.

Various examples of working GINGER input files are shown in §3.14. The beginning of an input file (and template file if present — see §2.3) *must* contain a one or more informational header lines whose essential purpose is to be repeated verbatim within the first graphics frame of the postprocessor output. These header lines can help remind the user as to what was so incredibly special or important about this particular set of input parameters. The header must be at least one and no more than 10 lines long; its end is indicated by the mandatory presence of a "**`$`**" symbol. Following the header lines is the first (and, for most runs, only) namelist identifier "**`&in`**". Note that this identifier *must* begin in the *second* column (the Fortran namelist structure prohibits any symbol from appearing within the first column). The namelist should terminated by "**`/END`**", also beginning in the second column. Most compilers permit a simple "**`/`**" but for readability "**`/END`**" is much safer. Likewise, each namelist line may contain multiple input variables but, for readability, one should use this capability sparingly. As explained later, when simulating optical klystron configurations (§3.7.2), multipass oscillators (§3.8) and/or certain other situations, the input file will need a second namelist named **`in_extra`**.

If an error is encountered in the namelist, further reading ceases. Beginning with GINGER versions January 2003 and later, the code attempts to print out to the user terminal window the offending input line from the input file together with one neighboring line. This should make it quite easy for the user to determine and then correct the error. Commonly-made errors in Fortran namelist input include specifying floating point values for integer variables (*e.g.* **`nside=32.0`** rather than **`nside=32`**), ASCII variables (*i.e.* strings) for reals or integers (*e.g.* **`nside='32.0'`**), arrays for scalars (*e.g.* **`omgj = 0.33 0.45`**), and plain vanilla typos for variable names (*e.g.* **`nsise=32`**). All character strings in GINGER namelist input should be single-quote delimited (*e.g.* **`gam_load = 'gaussian'`**).

Please note that for "historical" reasons, nearly all variables involving transverse dimensions (*e.g.* e-beam size) should be given in units of *centimeters* while those associated with longitudinal dimensions (*i.e.* wiggler wavelength) require units of *meters*. In a few rare cases, certain longitudinal variables require units of Rayleigh ranges but for nearly all of these, there are corresponding

alternative input variables for which meters are used (*e.g.* **zmaxsim** is in Rayleigh ranges while **zmxmeter** is in meters). Also, for variables such as transverse emittance and Twiss parameters for which the general accelerator community normally uses MKS units, GINGER attempts to follow suit.

3.3 Electron Beam Input Variables

Setting up the electron beam for a very simple, standard GINGER run requires that the user need only specify the beam current in Amperes (**current**), the MKS normalized emittance in rad-m (**emit_mks**), and the beam energy in MeV (**energy**). Normally, one also gives the number of beam slices (**nside**) to be simulated in either polychromatic mode (default value **nside=4**; see §3.4.2) or monochromatic "FRED" mode (**lfred=.t** has a default of **nside=1**). The other variables will be set to default values which will result in a beam loaded in equilibrium with the wiggler focusing, a uniform ellipsoid distribution in 4-D transverse phase space, and representation by 1024 macroparticles per slice. Usually, the user will want to set many other variables and we discuss the most important ones in the following paragraphs.

3.3.1 Macroparticle Number and Distribution

GINGER uses a moderate number (**ntestp**) of macroparticles (usually 512-8192 is adequate) per slice to represent the actual electrons in each beam slice. The default macroparticle load was changed in versions dated November 2001 and later to a Gaussian (**jmg=+2**). Alternative macroparticle loads include "super-Gaussians" (**jmg**$\geq$**+3**) and hard edge, uniformly-filled ellipsoids (*i.e.* waterbag) in 4-D phase space (**jmg < 0**) (which leads to parabolic radial density profile). If one is interested in accurately diagnosing bunching at higher harmonics, the number of macroparticles needed for good statistics will increase (*e.g.* $\geq$16384 for the 7th harmonic).

3.3.2 Electron Beam Emittance and Size

By default, the ratio of the x, y and x', y' axes of the transverse emittance ellipsoid are chosen such that the electron beam will be in a matched equilibrium (*i.e.* to prevent downstream mismatch oscillations) with the focusing properties of the wiggler at entrance. For equilibrium loads, the e-beam size is normally determined by input of the normalized MKS emittance (**emit_mks** in rad-m) or normalized CGS emittance (**emit0** in rad-cm). By default, the emittance is presumed the same in both transverse planes but a new feature permits one to specify different values for the MKS $x-x'$ (**emitx_mks**) and $y - y'$ (**emity_mks**) projected emittances. If none of the emittance variables is input, the code calculates the equivalent equilibrium emittance value if the namelist contains either (a) the beam radius (**omgj**) in cm; *or* (b) the beam current AND the central beam brightness

(**bright**). For historical reasons, the brightness is defined in "old LLNL" units of Amps/(rad-cm)^2 with $J \equiv 2I/(\gamma\varepsilon_o)^2$ for a uniformly-filled ellipsoid; note the absence of a π^2 factor in the denominator.

GINGER also has a seldom-used capability which determines the beam current when just upstream of the wiggler the electron beam passes through an emittance filter — such a situation was true for the 1980's LLNL/LBNL ELF experiment. To do this, the user must input a *negative* value for the beam current, a *positive* value for the electron beam brightness, and a positive value for *either* the electron beam emittance or radius.

In cases where the focusing strength is different in the x-plane from that in the y-plane, the equilibrium beam radius in each plane will differ if either the emittance or brightness is specified. One may also use the multiplicative scaling factors **xbscale** and **ybscale** to set the beam radius in either plane to a larger or smaller value than that corresponding to equilibrium. For uniform ellipsoid loads, the input emittance is the hard edge value; the corresponding RMS value is smaller by $\sqrt{6}$. For Gaussian loads, the input emittance corresponds to the RMS value in each projected plane (*i.e.* $x - x'$ or $y - y'$), *not* the edge value.

There are at least two ways in which the user can force the beam radius in each plane to particular values. The first is by giving the Twiss parameter beta in *both planes*: **betax_twiss** for x and **betay_twiss** for y, both in *meters* (note that this is an exception to the "centimeter" units rule for transverse quantities). Alternatively, one may set the radius in a given plane by inputting the parameters **omgjx** and/or **omgjy**, both in units of cm. This feature is particularly useful when there is no focusing in the wiggle plane of a linear wiggler. Input of the Twiss beta parameters overrides any values specified for **omgjx** and/or **omgjy**. When specifying the Twiss parameters, one must also specify the emittance. Note: if in one transverse plane there is neither wiggler nor external focusing (as might occur for a linear wiggler without curved pole face focusing — see §3.5.5), it is *highly* advisable for the user to input the initial beam size in that plane manually.

The parameter **rmaxcur** sets the electron beam's cutoff radius (in cm) when a Gaussian distribution has been chosen. The default value for **rmaxcur** is $3*$**omgj**. At present, as **rmaxcur/omgj** approaches 2 or smaller, the resultant particle load will result in an RMS emittance significantly smaller than input.

3.3.3 Initial Tilts and/or Offsets in x, x', y, y'

By default, there is no tilt of the initial emittance ellipse; *i.e.* the averages of $< xx' >$ and $< yy' >$ are zero. One can override this with the Twiss parameters **alphax_twiss** and **alphay_twiss**. One may also input the "thin lens" parameters **xfocus_mtr** and **yfocus_mtr** which set a hypothetical, zero emittance focal point (in meters) in the x- and y-planes respectively. For an individual macroparticle "*n*", this adds a term $-p_z * x_n/$**xfocus_mtr** and $-p_z * y_n/$**yfocus_mtr**, respectively, to the particle's transverse momenta. The net effect of specifying both the Twiss alpha

parameters and the thin lens lengths are additive - *i.e.* one does not override the other. One may also add either a constant transverse offset (**xoff** and **yoff** in cm) or transverse angle (**xprime** and **yprime** in radians) to the electron beam centroid. However, one should remember that for non-waveguide runs GINGER presumes axisymmetric radiation fields and the coupling between an off-axis electron beam and the radiation will not be treated in a self-consistent manner.

3.3.4 Instantaneous Energy Distributions and Chirping

The default instantaneous electron beam energy distribution is a delta function centered at the input-specified Lorentz factor **gammar0** or, alternatively, the beam energy in MeV (**energy**). One may specify a non-zero energy spread by inputting values for the width **dgamma** and the distribution type **gamload**. Permitted values for **gamload** are (1) **'uniform'**, the default; (2) **'random'** or (3) **'gaussian'**. Note that these choices are all lower case. For a gaussian distribution, **dgamma** is the RMS width while for the uniform and random loads, macroparticles are initialized between **gammar0**$\pm$**dgamma**.

In time-dependent mode, one may also place a chirp on $\gamma(t)$ with an amplitude of **gamchirp**. If **chirp_type** is set to its default value of **'sinusoid'**, γ varies sinusoidally with a peak-to-peak amplitude of 2***gamchirp**. When **chirp_type** is **'linear'**, γ increases from a value **gammar0** at the beam tail to a value (**gammar0** + **gamchirp**) at the beam head.

3.3.5 Longitudinal Current Profiles

By default, the electron beam current is time-independent (**pulse_shape='tophat'**). For short pulse, polychromatic simulations with either **ltransit=.t** or **losc=.t** (both of which override periodic boundary conditions in time) OR multislice, monochromatic FRED-mode runs (which suppress slippage effects), various longitudinal current profile options are permitted: (a) parabolic (**pulse_shape='parabolic'**), (b) sawtooth (**pulse_shape='sawtooth'**), (c) Gaussian (**pulse_shape='gaussian'**), (d) hyperbolic tangent (**pulse_shape='tanh'**), or (e) a modified tophat profile in which the current has both an exponential rise and fall with time (**pulse_shape='exptail'**).

For both parabolic and sawtooth profiles, the full pulse width (*i.e.* where $I(t) > 0$) will be equal to the *full* electron beam duration, namely **nside** $\times$ **dt_slice** $\equiv$ (**nside/nphoton**) $\times$ **window**. For Gaussian profiles, the input variable **tbody** sets the RMS electron beam pulse width; one must be careful that **tbody** is appropriately small enough for the choice of **nside** and **window**. For hyperbolic tangent profiles, $I(t) \propto (1 + \tanh[(t-\tau_-)/\texttt{trise}]) \times (1 + \tanh[(\tau_+ - t)/\texttt{trise}])$ with $\tau_\pm \equiv \bar{t} \pm \texttt{tbody}/2$ where $\bar{t}$ is the center of the temporal simulation window. When **pulse_shape='exptail'**, **trise** sets an exponential rise and fall time (presumed identical). In all cases, **tbody** and **trise** are measured in seconds. Similar temporal profiles may be chosen

for the input radiation field (see §3.4.3). Note that if one seeks a shortened "tophat" profile, one can approximate this by choosing a **tanh** profile with a very small **trise** and **tbody** equal to the duration sought. However, it is computationally more efficient just to reduce **nside**.

At present, GINGER does not model the *coherent* microbunching due to a time-varying current (*i.e.* the "shape-factor" term) on the electron beam at its entrance into the wiggler. Such bunching (which produces coherent spontaneous emission) can be important for electron beams whose pulse lengths are a dozen radiation wavelengths or shorter, especially when the longitudinal profile is non-Gaussian. Since this coherent microbunching can in principle be much larger in magnitude than the incoherent shot noise term, its absence is an important limitation in GINGER.

3.3.6 Beam Envelope Initialization from a External "Experimental Data" File

GINGER can read and then use time-resolved electron beam envelope information from a data file to initialize electron beam properties at the undulator entrance. This capability is available in polychromatic mode both for short-pulse (*i.e.* **ltransit=.t**) and segmented run mode. It is also available for multi-slice, monochromatic FRED mode. For the last case, a "parameter-scanning" run is done (see §3.11) with the independent scan parameter being the longitudinal position (*i.e.* time) in the electron beam frame. This type of run permits one to get a quick estimate of how FEL gain might vary over a particular pulse profile.

The external data file must be in ASCII format and can either have the daat in simple columns (see below) or in SDDS format (preferred) as would be produced by the program **elegant2genesis**. The data filename is set by the GINGER input variable **exp_data_file**. GINGER parses the beginning of the data file to determine whether it is an SDDS or simple ASCII file.

The 2-element, real input variable **t_start_end_ebeam** can be used to set (in seconds) the temporal portion of the data which will be used by the simulation; this applies both for short-pulse and multislice FRED mode. In polychromatic mode, setting both elements of **t_start_end_ebeam** together with **nside** defines **dt_slice** and overrides any input value for this or **nsidep**. Similarly, if both **dt_slice** *and* the "tail" position of the simulation **t_start_end_ebeam(2)** are set, these together with the value of **nside** will determine **nsidep** and **t_start_end_ebeam(1)**. If either edge of the simulation temporal window extends beyond the that corresponding to the data file, GINGER attempts to do intelligent extrapolation of the various envelope quantities.

The following envelope parameters can be initialized from "experimental" datafiles: electron beam current (label **'CURRENT'**; units = amps), average Lorentz factor γ (label **'GAMMA'**), instantaneous energy spread $\delta\gamma$ (label **'DGAMMA'**), rms beam size in x (label **'XRMS'**; units = m) and y (label **'YRMS'**; units = m); normalized emittance ε_N (label **'EMIT'**, units = rad-m) when equal in both transverse projections, or for the individual planes $\varepsilon_{x,N}$ (label **'EMITX'**) and $\varepsilon_{y,N}$ (label **'EMITY'**); Twiss parameters α_x (label **'ALPHAX'**), α_y (label **'ALPHAY'**), β_x (label **'BETAX'**; units = m), and β_y (label **'BETAY'**; units = m). There should also be a longititudinal position

column containing time coordinates (column label **'TIME'**; units = seconds) increasing from head to tail. The temporal positions need not be uniformly spaced but they should be monotonically increasing. For SDDS-formatted files, lower case alternatives (*e.g.* **gamma**) to the above labels are permitted as is spatial position (**'S'**; units = meters, decreasing from head to tail). All of the above envelope parameters can be generated in SDDS format from 6D **elegant** tracking code output using the **elegant2genesis** conversion code (or its successor **elegant2slices** currently under development). Once input, the envelope data is interpolated to GINGER's uniform temporal grid.

A recent example of SDDS-formatted envelope information from ELEGANT output (please note that to fit all the necessary columns within the printed page, they have been artificially put on multiple new lines — the actual file has all the column data for a given macroparticle on a single, very large line):

```
SDDS1
&description text="Macroparticle data obtained 27 Jun 03 14:13 from Elegant
output file: LCLS10JUN03_matched.sdds", &end
&column name=t, symbol="Time position", units=s, type=double,  &end
&column name=gamma, type=double,  &end
&column name=dgamma, type=double,  &end
&column name=xemit, symbol=NormalizedEmittance-x, units=m, type=double, &end
&column name=yemit, symbol=NormalizedEmittance-y, units=m, type=double, &end
&column name=xrms, symbol="Beam Size-x", units=m, type=double,  &end
&column name=yrms, symbol="Beam Size-y", units=m, type=double,  &end
&column name=xavg, symbol=Position-x, units=m, type=double,  &end
&column name=yavg, symbol=Position-y, units=m, type=double,  &end
&column name=pxavg, symbol="Average x'", units=rad, type=double,  &end
&column name=pyavg, symbol="Average y'", units=rad, type=double,  &end
&column name=alphax, symbol=Alpha-x, type=double,  &end
&column name=alphay, symbol=Alpha-y, type=double,  &end
&column name=current, symbol=Current, units=Amp, type=double,  &end
&data mode=ascii, &end
! page number 1
                  128
 1.056618000000000e-13  2.806437390513857e+04  2.261781207393367e+00
     3.108103564640305e-07  6.225613157229169e-07  1.727873797185741e-05
     2.685425915162709e-05  2.744201315982141e-05  1.567737775814357e-05
    -1.011066086041603e-06 -2.246429721466592e-06 -2.677069672749939e+00
     2.883181437519945e+00  1.813735625803665e+01
 1.039794236220472e-13  2.806516064411772e+04  3.644894466311403e+00
     1.374355746409790e-06  8.123925763796192e-07  3.066269046451931e-05
```

```
 2.096679996978939e-05  1.324073235334876e-05 -4.288142261091647e-06
-3.625579076306638e-07  6.174544336401088e-07 -2.839956664706138e+00
 1.411162359259141e+00  7.586315874850315e+01
```

et cetera ...

Although SDDS-formatted files are preferred, files with groups of simple columnar output can also be used. In this case, for each envelope parameter, the code searches the data file to find a line which includes the appropriate label (*e.g.* **CURRENT**). If GINGER finds such a line, the next line must contain the integer number n of $(t_i, f(t_i))$ data pairs followed by n rows of 2-column data. For example:

```
  CURRENT
8
0.0    150.
1.e-15 300.
2.e-15 400.
3.e-15 450.
4.e-15 450.
5.e-15 300.
6.e-15 200.
7.e-15 100.

  RMSX
  8
0.0    88.3e-6
1.e-15 115.e-6
```

et cetera ...

3.3.7 Macroparticle Initialization from an Simple ASCII File

GINGER has a limited capability to read in a previously user-generated macroparticle distribution in columnar ASCII-format and use the information as the basis for its own macroparticle generation. The GINGER input variable **trackparfile** should be set to the name of the ASCII file which contains the phase space information. The information can be used both in FRED- and full time-dependent mode; as of mid-2003, GINGER can directly import multiple independent groups of macroparticles from a **trackparfile**.

At present for single group input, the **trackparfile** format should be laid out as follows:
(1) an optional number (0-49) of comment lines
(2) a single post-comment line, identified by the presence of the string **'-----'** ;
(3) the next line should contain the integer number of macroparticles (NP) in the file (this value

will be used to override the value of **ntestp** input in the GINGER namelist
(4) NP single lines, each containing (x, x', y, y', γ) for a single macroparticle, with x and y in *meters*, x' and y' in radians (*i.e.* not normalized transverse momenta $\gamma x'$!), and γ, the Lorentz factor.

To denote multi-group input, the post-comment line which contains the string **'-----'** should *also* contain the string **'MULTISLICE'**. The format is modifed from the single slice case as follows:
(3) A line containing the number of groups, the numbers of particles (NP) per group, and the temporal spacing between groups. These values will be used to set GINGER values of **nside**, **ntestp** and **dt_slice** respectively; consequently, one should bear this in mind when generating the data externally.
(4) Sets of macroparticle data for each group, which should be organized in the following structure:

- (a) A single line with the slice number (integer) and a an arbitrary real number (*e.g.* the time value at slice center) — at present, neither of these values is actually be used by GINGER.
- (b) NP lines of *6* individual particle coordinates $(x, x', y, y', \gamma,\ t_*)$. At present t_* is NOT used and may contain any real value.
- (c) Following each set of individual group particle data (including the last), there must appear a single line containing the string **'----'**. This line serves as a separator between groups to improve eyeball readability of the data file.

For each user-supplied macroparticle, GINGER will supply a longitudinal phase coordinate θ using the same algorithms as its normal load process, which in general is a "quiet start" load — see §3.3.10. In this case, each particle's 5D coordinate will be cloned **nfold_sym** times and each successive clone will have its θ offset by $2\pi/$`nfold_sym`. Thus, if the external distribution file contains 512 distinct particles, each electron beam slice in GINGER will then contain 4096 macroparticles if **nfold_sym** is left at its default value of 8. Consequently, for FEL purposes the microbunching occurs initially only through the application of shot noise effects. For polychromatic runs for which only one macroparticle group has been input in the **trackparfile**, each of the **nside** electron beam slices will have *identical* distributions in (x, x', y, y', γ).

3.3.8 Macroparticle Initialization from ELEGANT tracking code output data

GINGER now has a limited capability to import actual macroparticle data from ELEGANT code output. To do so, *before* making the GINGER run, one uses a separate program (**xconv_eleg**) to read, convert, and finally output ELEGANT-related data in a format usable by GINGER; the Fortran90 source code for **xconv_eleg** is freely available from the GINGER author. When compiled and then run as an executable without any further input, **xconv_eleg** writes a simple "help" file to the output terminal which illuminates details of its use.

Briefly, via namelist input, one instructs **xconv_eleg** to construct either a simple, single slice, ASCII **trackparfile** in the format described above in §3.3.7 or a "summary" file which will be read by GINGER to help construct time-dependent, multi-slice output. The single slice **trackparfile** can be used without further changes by GINGER in either FRED- or time-dependent, polychromatic mode. When multi-slice output is desired, **xconv_eleg** analyzes the ELEGANT macroparticle data to determine both the mean and first-order temporal derivative of time-dependent envelope quantities such as I, $\langle x \rangle$, $\langle x' \rangle$, *etc.* . It also creates a look-up table associating time with time-ordered particle index in the ELEGANT file. Note that this requires that the ELEGANT macroparticle data *must* be sorted in monotonically-*decreasing* time before using **xconv_eleg**; fortunately, the SDDS toolkit program **sddssort** can do this directly. The summary information (and the name of the original ELEGANT file) is output to a special ASCII-formatted file. Setting the input variable **tracksumfile** to the name of this summary file instructs GINGER to read both this file and the original ELEGANT file. Then (within GINGER), a "moving window" scheme is applied whose temporal thickness varies to enclose the wanted number of ELEGANT macroparticles. The scheme also corrects for unphysical, numerical effects which arise from a non-zero window thickness when there are locally, non-zero time-derivatives for mean electron beam properties (such as $\langle x \rangle$). As the temporal macroparticle density increases in the ELEGANT file, the thickness of the moving window and necessary coordinate corrections become correspondingly smaller. GINGER directly extracts $I(t)$ from the **tracksumfile**.

In both single- and multislice cases, only the 5D ELEGANT coordinates (x, x', y, y', γ) are directly used by GINGER. The longitudinal θ position is independently set by GINGER applying its "usual" methods (*e.g.* quiet start with shot noise — see §3.3.10). Please note that the coding and algorithms for importing multi-slice, ELEGANT macroparticles are quite new and may be possibly quite buggy.

3.3.9 Importing Macroparticles from the Output of a Previous GINGER Run

For both single- and multi-slice (*i.e.* scanning mode §3.11)), FRED-mode runs and multi-slice polychromatic runs, GINGER can both write and read a *rstfile* containing a full 6D $(x, x', y, y', \gamma, \theta)$ macroparticle distribution. Normally, one writes such a file at the undulator exit of one run and then reads in this information for a subsequent GINGER run to initialize the particle phase space for a following (in z) undulator. This capability was developed specifically for multiple undulator scenarios such as modulator-radiator and harmonic cascade configurations.

To instruct GINGER to write a macroparticle *rstfile* file at completion, set the input switch **l_write_rst=.t** in the main input namelist. The resultant file will be in binary format and named **rst*run_name***. Note that if the GINGER run uses multiple processors (*e.g.* the IBM-SP), the file will be written using parallel MPI-IO subroutines and can later *only* be read by another multi-processor run employing MPI-IO. In single-processor serial mode, the file is written in sequential

binary format and subsequent GINGER runs in either single processor or multi-processor mode can successfully read the file. For polychromatic runs, the *rstfile* can be very large (≈ 50 kB per slice per 1024 macroparticles).

In addition to the actual macroparticle distribution, the restart file contains a header including the number of macroparticles per slice, the central wavelength, the temporal slice spacing, and various computed envelope parameters (*e.g.* transverse emittance, energy spread, central energy) for the first electron beam slice.

To initialize the macroparticle distribution in a subsequent GINGER run from a previously written *rstfile*, set the input variable **`l_read_rst=.t`** *and* on the execute command line add the string **`-rs rstfile`** where *rstfile* is the file's name (*e.g.* **`rstpalacA1`**). In general, the values read in from the *rstfile* for macroparticle number (**`ntestp`**), temporal slice spacing (**`dt_slice`**), number of e-beam slices (**`nside`**) and beam current will override those set by user input. However, in FRED-mode, if **`nside=1`** in the original run, **`nside`** can exceed one if this a parameter-scanning run. At present, a *rstfile* does not contain longitudinal current profile information. Consequently, if the first undulator GINGER run was done in short-pulse mode (see §3.1), the input file for the second and subsequent undulators should also indicate short-pulse mode and include the necessary information (see §3.3.5) to create an identical $I(t)$ profile.

3.3.10 "Quiet Starts" and Shot Noise

By default (**`lquiet=.t`**), the macroparticles are loaded in phase space with a bit-reversed quiet start with a N-fold symmetry in the longitudinal coordinate θ where N is set by the input variable **`nfold_sym`** with a default value of 8. Thus, each macroparticle at $(x_n, x'_n, y_n, y'_n, \gamma_n, \theta_n)$ will have **`nfold_sym-1`** "shadow" macroparticles with the identical $(x_n, x'_n, y_n, y'_n, \gamma_n)$, but whose longitudinal phase θ is successively incremented by $2\pi/$**`nfold_sym`**. Choosing **`nfold_sym=8`** will eliminate any initial bunching through the fourth harmonic. In studies of third harmonic bunching, we found it necessary to use seven rather than one shadow particle in order to cancel out all initial bunching at the fourth harmonic (which couples to growth of the third harmonic in the exponential gain regime). If one is examining bunching through harmonic N, one should set **`nfold_sym=2N+2`** and simultaneously increase **`ntestp`** appropriately to retain reasonable resolution in γ and transverse phase space quantities. However, if one is not concerned with accurate modeling of higher harmonic bunching, picking **`nfold_sym=2`** will give the best resolution of the transverse phase space and longitudinal energy spread distributions for a given value of **`ntestp`**.

When shot noise fluctuations are desired (**`lshot=.t`**) for SASE and similar studies, a random $\delta\theta_n$, which follows a Poisson distribution, is added to each macroparticle's longitudinal phase θ_n. There are no fluctuations in transverse phase space. In fall 1999, the shot noise algorithm was completely rewritten to try to overcome a small bug observed when **`dgamma`** was non-zero. Now, each group of **`nfold_sym`** macroparticles with the same $(x_n, x'_n, y_n, y'_n, \gamma_n)$ has its own set of random

shot noise variables (*e.g.* bunching phase and amplitude at different harmonics of the fundamental radiation wavelength λ_s). The new algorithm appears to ensure that coarse-grained averages over both $< \exp i\theta >$ and $< \exp 3i\theta >$ are correct (and have identical values in the limit **nside**$\rightarrow \infty$). One may increase/decrease the effective power level of initial shot noise bunching fluctuations by setting the scaling factor **pwrnoise** different from its default value of one.

Shot noise bunching fluctuations may also be included in monochromatic, single-slice FRED-mode runs. This capability is useful if one wants to quickly examine the differences between MOPA's and SASE-like input sources. However, one should remember that, given the absence of slippage in FRED-mode runs, these fluctuations will be "coherent" (and thus monochromatic) in a longitudinal sense and will also produce a much larger effective input laser power than would be true for the equivalent polychromatic run. Finally, when the macroparticle distribution is initialized from a *rstfile* created by a previous GINGER run (see §3.3.9), shot noise is *not* re-initialized nor is a quiet start possible.

3.3.11 Random Number Seeds

Random number seed input variables (which, in principle at least, should allow the user to repeat *exactly* previous simulation runs) include: (1) **iseeds** for longitudinal shot noise (see §3.3.10); (2) **iseed** for the phase and amplitude of the different radiation field spectral components (see §3.4.4); and (3) **iseedp** for loading instantaneous energy spread (applicable when **gam_load='random'**; see §3.3.4). Due to recent changes in the shot noise algorithm, the variable **iseedp** no longer affects the electron beam's transverse phase space distribution.

Each of the random seed variables is a 4-element array of decimal (*i.e.* not octal) integers. A "master seed" is created by a call to the system clock which is then used to generate those seed variables *not* input by the user. In this case, the generated seed will lead to only the first element of the seed array being non-zero. However, if the user inputs a random seed variable (as might be true to recreate a run), this variable is converted into an array of 4 12-bit integers (*i.e.* the effective seed is the input seed modulo 2**48). Hence, one should not input a seed greater than $\sim 2.8 \times 10^{14}$. For runs on the NERSC CRAYs (which have 64-bit size words), the first array input seed element can fully contain the 48-bit effective seed. On platforms with 32-bit integers (including the IBM-SP and most workstations) one would need two array elements to get the full dynamic range of possible input seeds (however, it is not likely any user will do such a large number of runs that 2**32 unique seeds proves insufficient!).

For *all* runs (FRED-mode and polychromatic-mode, single and multiprocessor platforms) with GINGER versions beginning in November 2001, the random numbers have effective lengths of 48-bits and are generated by a special numerical package provided by NERSC consultants. For polychromatic runs, this package has the distinct and needed property that the random numbers used to generate each beam slice will be independent of both the total processor number used in the

run *AND* the particular assignment order of individual processors to individual slices.

3.4 Radiation Field Input Quantities

3.4.1 Optical Beam Size, Transverse Profile, and Waist Position

Paralleling many of the electron beam input variables are those corresponding to the radiation field. The default initial transverse profile is Gaussian (**nmg=+2**) while "supergaussians" may be specified by **nmg** $\geq$ **+3**. The optical waist radius ω_0 ($\equiv$**omg0**) corresponds to the $1/e$ point in $\vec{r}$ of the radiation electric field when at a waist minimum. Normally, **omg0** is determined by the input variable **omg0fac**, whose default value is 0.8, and the relation **omg0**$\equiv$**omgj** $\times$ **omg0fac**, where **omgj** is the electron beam radius. This can be overridden by giving a positive value for **omg0**, Optionally, one may also set the position in z of the focal point (*i.e.* waist) of the input radiation by giving a value either in meters (**zfcmeter**) or in Rayleigh ranges (**zfocus**). The Rayleigh range $Z_r \equiv \pi\omega_o^2/\lambda_s$. A non-zero value for either leads to curved wavefronts at $z = 0$. Note that although the electron beam model includes full 3-D non-axisymmetric dynamics which can result in a non-circular shape, the radiation field and its source terms are presumed axisymmetric (in non-waveguide runs).

3.4.2 Wavelength, Input Power, Slice #, and Temporal Resolution

By default, GINGER presumes a time-dependent, polychromatic problem; this can be overridden by setting either **lfred=.t** or **nside=1**.

The optical wavelength λ_s is specified by **wavels** in meters; this is the numerical value of the central wavelength of the effective bandpass in polychromatic runs. If one is running a microwave problem (**lwavegd=.t**), one may alternatively specify the central frequency in GHz (**ghz**). Normally, one uses the input variable **plaser** to set the initial radiation power in watts.

There are a number of input parameters which define the temporal resolution of the input laser field. The variable **nphoton** sets the total number of photon slices. For short-pulse, non-periodic boundary conditions (*e.g.* oscillator or "transit time" runs), **nphoton** *must* be input. For periodic boundary conditions in time (*e.g.* long pulse amplifier runs), **nphoton = nside** automatically and **nside** rather than **nphoton** should be input. In any case, the resultant **nphoton** should be a power of two or three times a power of two in order for the FFT spectral decomposition in the postprocessor to run properly. To set the total temporal duration followed in z within the simulation and the time interval between individual photon (and electron beam) slices, one should set either **window**, which gives the equivalent longitudinal length (in meters) of the temporal window, or **nsidep** which gives the total number of discrete photon slices with which a particular electron beam slice will interact over the full length of the wiggler. Numerically, **window** $\equiv$ (**nside** /

nsidep) $\times (L_w \times \lambda_s/\lambda_w)$. When **nsidep**=$N_w$, the full frequency span of the simulation equals central frequency ($\equiv c/\lambda_s$). In the great majority of situations one will usually set **nsidep** rather than **window**.

At present with two exceptions, **nsidep** *must* be less or equal to **nphoton**. The first exception exists for the multiple processor runs on MPP platforms (the CRAY-T3E and IBM-SP) where, **nsidep** can exceed **nside**. In cases where the slippage length greatly exceeds the so-called "cooperation" or "coherence" length, one should consider employing this strategy to increase the $z-$resolution and spectral bandpass without having to increase the total number of slices. This option is also useful for optical klystron configurations (§3.7.2).

A second exception exists when the run uses a $z-$dependent field file to initialize the radiation field for the tail slice of a new run (see §3.4.6). In this case, **nside** may be as small as 1.

3.4.3 Setting Temporal Profile of Input Radiation Field in Short Pulse Mode

As was true for the longitidinal current profile $I(t)$, in short pulse mode the user may set a different profile for the input radiation field than the default time-independent "tophat". The input variable **laser_shape** controls $P(t)$; the available choices are:

(a) parabolic (**pulse_shape='parabolic'**)

(b) sawtooth (**pulse_shape='sawtooth'**)

(c) Gaussian (**pulse_shape='gaussian'**)

(d) hyperbolic tangent (**pulse_shape='tanh'**).

The input variable **laser_body** set the full base width (in seconds) in the case of parabolic or sawtooth profiles, and the RMS temporal width for Gaussian profiles. For the hyperbolic tangent profile, **laser_body** is the width between the two points in time where the argument of the tanh function go to zero. A shortened tophat profile may also be chosen by giving a positive value for **laser_body** which is less than the full temporal window. The temporal centroid of $P(t)$ may be moved relative to that of the electron beam's $I(t)$ by giving a non-zero value (in seconds) to the input variable **laser_timing_rel**. A positive value moves the centroid of $P(t)$ toward the electron beam tail, a negative value toward the electron beam head.

3.4.4 "Customizing" the Spectrum of the Input Radiation Field

The user has a fair amount of flexibility in "customizing" the spectrum of the input radiation field. If, for whatever reason, one wants the wavelength of the input radiation field (to which **plaser** refers) to be different from the central wavelength of the simulation (=**wavels**), one can set this by giving a value for **wavelsin** in meters. If one wants the input radiation power spread out equally

over a number of spectral bins, one gives a positive integral value for **nfreqbin**. Both **wavelsin** and **nfreqbin** may be simultaneously input.

To generate a uniform spectrum with equal noise power in each of the **nphoton** frequency bins encompassing the complete frequency span, one sets the power level per bin by either inputting **wattpbin** or **wattpghz**. The latter variable applies only to waveguide runs. If **plaser = 0.**, the input radiation field will consist *only* of noise — note this is also true for the central wavelength **wavels**. The total input power will be **nphoton** $\times$ **wattpbin**. Another means of starting with a uniform power spectrum without creating excess power at the central frequency is by giving a positive value for **plaser** and a *negative* value for **nfreqbin** in which case the total input power is **plaser** exactly. Alternatively, a *negative* value of **ampside** together with a positive value of **plaser** will also generate a flat noise spectrum over the full bandpass (in addition to the power **plaser** at the central wavelength) with the total noise power equaling (**plaser** $\times$ ampside^2). An example of this type of white noise initialization is given in §3.14.2.

Both **ampside** and **wattpbin** generate noise which in the time domain has both a random amplitude and phase. A new input variable, **field_noise**, allows one to generate each component separately. A positive **field_noise(1)** set the normalized RMS strength of the field amplitude noise while a positive **field_noise(2)** gives the RMS phase noise in radians; both elements can simultaneously be positive. This presumes that **plaser** is positive.

If one wants *all* the non-fundamental power at a single frequency or wavelength, one specifies either **sidewave** in meters or **sidefreq** in Hz. The power level is set by a *positive* value of **ampside** with $P_{sideband}$ = **plaser** $\times$ampside^2. A positive **ampside** without specifying either **sidewave** or **sidefreq** sets up a white noise sideband field amplitude $\hat{E}(r,t)$ fluctuating in time between $\pm$ampside $\times$ $E_o(r)$ where $E_o(r)$ is the electric field of the fundamental.

When running in short-pulse mode (*i.e.* **ltransit=.t**), the radiation field is not presumed to be periodic in time and for various reasons GINGER does not initialize all **nphoton** locations in time of the radiation field at $z = 0$ (essentially it needs only to initialize those radiation slices which immediately interact with the electron beam). Consequently, even if one specifies a flat noise spectrum (*e.g.* via the **wattpbin** input variable), the post-processor power spectrum will *not* show a flat spectrum at $z = 0$.

Finally, one may place a sinusoidal temporal chirp on the input radiation's wavelength by giving a positive value to the input variable **ampchirp**, which defines the peak normalized (*i.e.* relative to the simulation bandpass) wavelength shift. In general, one would use such a chirp only for "long pulse" simulations with temporally-periodic boundary conditions.

3.4.5 Saving to and Initializing from Undulator Exit Radiation Field Restart Files

In order to study multiple undulator configurations where the output radiation from one undulator is used to initialize the field in a following undulator, GINGER has the capability to both read

and write $t-$dependent restart field files. This capability exists for both single-slice FRED mode and multi-slice polychromatic mode (including multi-processor mode). To write such a undulator exit field file, set the main namelist input variable **l_write_fld = .t**. The resulting output field file will be named **fld*run_name*** and is written in sequential binary format. In addition to the $r-$ and $t-$dependent radiation field information, a field file also contains various simulation parameters such as the radial grid, the radiation central wavelength **wavels**, the temporal slice spacing **dt_slice**, and the number of radiation slices **nphoton**.

To read a previous undulator's field file, the command line switch **-f fldfile** should be used. The number of radiation slices, temporal slice spacing (*i.e.* **dt_slice**), radial grid dimensions and spacing, *etc.* for the new run will be set from information read from this field file. However, the central wavelength λ_s in the field file is *not* checked at present for self-consistency with the value given in the input file. As explained in §3.3.9, GINGER has an equivalent IO capability for writing and reading macroparticle information from one undulator to the next.

3.4.6 Saving to and Initializing from $z-$dependent "Beam Head" Radiation Field Files

GINGER also allows a user to save to a special disk file the instantaneous state of the radiation field "exiting" from the "head" e-beam slice (*i.e.* slice #**nside** because GINGER numbers slices beginning from the tail). Because GINGER applies slippage at **nsidep** discrete locations in the wiggler (see §5.5), this "exiting" field information is written at the same $z-$ locations. This field information can subsequently be used at the beginning of a new GINGER run to initialize the radiation field for the new "tail" e-beam slice (*i.e.* slice #1). This procedure results in the equivalent of stitching together of two temporally-adjoining electron beam sections; *i.e.* it is only through such radiation slices that adjoining electron beam slices "communicate".

There are two underlying purposes for this feature: (1) To permit a user to "cut" a relatively long electron beam pulse (as compared with the slippage length) into a number of shorter sections, each one of which can then be simulated sequentially with GINGER. In effect, each run following the first is similar to a short pulse, transit-time run but with no radiation field propagation into vacuum beyond the electron beam body. (2) To allow a user to do a monochromatic, single-slice "FRED-mode" run using the radiation field from a polychromatic, time-dependent run. The macroparticle output diagnostics from such a "FRED-mode" run (via the **nspec** input variable) can be used to examine in great detail the $z-$evolution of an electron beam slice's longitudinal $\gamma - \theta$ phase space.

To save a $z-$dependent field file, set the input variable **l_write_fld_z=.t** in the main namelist. This "write"switch may be used in "long-pulse", periodic BC polychromatic mode only (*i.e.* **losc** and **ltransit** are both false) for both single- and multi-processor runs. At the completion of the simulation, an SDDS ASCII-formatted file named **fld*run_name*.sdds** (*e.g.* **fldpalac.sdds**) is written to disk. In addition to the complex radiation field information, the numerical values of the radial grid are also stored (see §3.12.1) and various simulation parameters such as the temporal

spacing of slices **dt_slice**.

To use a pre-existing $z-$dependent field file in a subsequent run, set **l_read_fld_z=.t** in the main namelist. Also, the execute line of GINGER must now include the option **-f *oldfldfile***, where ***oldfldfile*** is the *full* name of the previously created SDDS-formatted field file. The new run can be either in single slice, FRED-mode or multiple slice, polychromatic-mode. In either case, the undulator parameters of the new run should be *absolutely identical* to those of the previous one which created the field file. In polychromatic mode, the value of **dt_slice** read from the field file set the temporal spacing in the new run and override any user input of variables such as **nsidep** or **window**. However, **nside** can now take on any positive value ≥ 4. In polychromatic mode, setting **l_write_fld_z=.t** in the new run's input will lead to the writing of a completely new field file; be careful not to use the same ***run_name*** as that of the previous run or the old field file will get overwritten. In FRED mode, the input variable **ncurve** *must* be set to the previous run's value of **nsidep**.

The post-processor XPLOTGIN also can generate a $z-$dependent field file from the field information contained in the *pltfile* of a previous run; see §4.12 for details.

3.5 Wiggler and Electron Beam Focusing Input Variables

3.5.1 Base Wiggler Input Parameters

Nearly all GINGER runs require the user to set the following input variables to specify the wiggler configuration: polarization type with **llinear=.t** for linearly- or **llinear=.f** for helically-polarized undulators, wiggle period **wavelw** in meters, and whether the wiggler field strength is constant with z (**lcnstwgl=.t**) or not (**lcnstwgl=.f**). The total wiggler length (including drift spaces, if any) is set equal to the simulation's longitudinal span in either Rayleigh ranges (**zmaxsim**) or meters (**zmxmeter**). Beginning in November 2001, an optional "lattice" file may be used to set wiggler parameters (including pole strength errors and corrective steering) — see §3.5.8 for details.

3.5.2 Constant $a_w(z)$ Wiggler Field

The longitudinal dependence of the RMS normalized vector potential $a_w(z)$ can be set in several ways. A *constant strength* wiggler can be specified by giving one of the following: (1) a positive value for **aw0**, the *RMS* normalized vector potential strength of the wiggler on axis; (2) a positive value for **bw0**, the *peak*, on-axis wiggler field strength in *Tesla* (note: changed from kG as of Nov 2001); (3) setting **idesign=1**, following which GINGER computes the appropriate resonant value for a_w corresponding to the input values for **gammar0**, **wavelw**, and **wavels**. In the last case, no allowance is made for detuning effects such as non-zero emittance and/or quadrupole focusing

which normally will make peak gain occur at a slightly lower value of a_w. One can multiplicatively scale the nominally computed a_w up or down by giving a value for the input parameter **awdmult** different from its default value of 1.0.

3.5.3 Using a Predetermined $a_w(z)$ Tapered Wiggler Profile

In some cases one may want to use a previously determined $a_w(z)$ tapered wiggler profile stored in a separate file. To do so, (a) set **idesign=0**; (b) set **lcnstwgl=.f**; and (c) specify the name of the wiggler file by either using the **-b *mybwfile*** option on the execute line *OR* set **bwfile='*mybwfile*'** in the main input namelist. Otherwise, GINGER presumes that a tapered wiggler file (generally referred to as a *bwfile*) exists with the name **bw*run_name*** (*e.g.* **bwelf3a**). The previously mentioned input parameter **awdmult** can also be used to scale the values of the input tapered $a_w(z)$.

Normally, such a tapered wiggler file will have been generated by a previous GINGER run operating in FRED-mode. However, the user can also design such a file (which must be in ASCII format) adopting the following prescription:
(1) the first line is simple text. Normally this contains some details about the GINGER run which generated it but the user may put anything here.
(2) The second line contains a single integer equal to the number ($\equiv$ **N_AW**) of z positions to follow; a minimum of two such positions is needed.
(3) **N_AW** single lines containing two floating point variables: the z position (in meters) and the on-axis value for $a_w(z)$.

By default, GINGER will use a cubic spline fit to determine the local $a_w(z)$ from the discrete values of a_w input in the wiggler file. If one desires abrupt shifts in either a_w or its z-derivative, a spline fit require putting in many additional points to force such a shift in the fit. However, one can avoid this annoyance by setting the input namelist variable **l_bw_linearfit=.t** which replaces the spline fit with simple linear interpolation of a_w between adjacent points in z.

3.5.4 Tapered Wiggler Self-Design in FRED-Mode

In FRED-mode *only*, setting **idesign=2** *and* **lcnstwgl=.f** uses GINGER's self-design algorithm to compute a tapered wiggler. These $a_w(z)$ values will be written out to the ***bwfile*** at **ncurve** equally spaced locations in z. The algorithm determines $a_w(z)$ by keeping its ponderomotive well phase $\psi(z)$ constant for an imaginary test particle initially located at ψ=**psir0**, γ=**gammad0**, and transversely at r=**rdesign**. Default values are **psir0=0.4** and **gammad0 = gammar0**. The switch **lphase10=.t** limits the z-rate of change of $a_w(z)$ caused by the electromagnetic field phase variation $d\phi/dz$; normally, this is only a concern for high gain microwave FEL's. In order to account for the radial increase of a_w off-axis and the radially local

value of the radiation field phase $\phi(r)$, the parameter **rdesign** or **rdesfac** determines the radius of the imaginary design particle, where **rdesign = rdesfac** $\times$ **omgj**. with a default values **rdesfac=0.707**. A sample tapered wiggler input file is given in §3.14.1.

3.5.5 Wiggler Focusing: Simple and Curved Poleface

Wiggler focusing is controlled by a number of optional input parameters. Helical wigglers have natural focusing in both transverse planes while linear wigglers have focusing only in the non-wiggle plane ($\hat{y}$-direction in GINGER for non-waveguide FEL's). If no other focusing than the "simplest", natural wiggler focusing is desired, no additional input parameters are required.

However, for longer wigglers with linear polarization, one generally needs focusing in the non-wiggle plane. One method to achieve this is by using "curved" (*aka* "parabolic") pole piece focusing by inputting a value for **rkxkw** which increases the focusing in the wiggle $\hat{x}$ plane while simultaneously reducing the focusing in the non-wiggle $\hat{y}$ plane. **rkxkw** must lie within the interval $[0, 1]$. Setting **rkxkw=0.707** provides equal strength focusing in both planes, which is normally the desired result.

3.5.6 External Focusing: Continuous Quadrupoles and/or Ion Channels

If one desires external focusing is addition to that provided naturally by the wiggler, GINGER provides a number of choices:

(1) To add constant, z-independent, quadrupole focusing in the wiggle plane (and *simultaneously* equal magnitude *defocusing* in the non-wiggle plane), specify a positive value for **quad0**, which has units of Gauss/cm, or **wavelx**, the desired betatron wavelength (m) in the wiggle plane. Alternatively, a positive value for **focusfac**, which measures the ratio of the quad-induced $k^2_{\beta,x}$ relative to the value of $k^2_{\beta,y}$ associated with the natural wiggler focusing, will also give wiggle plane focusing. If a_w varies with z (*i.e.* a tapered wiggler), so will the actual quad gradient when **focusfac** is non-zero. When **focusfac=0.5**, the net focusing (*i.e.* quadrupolar plus wiggler) will be equal in the wiggle and non-wiggle planes. In general, one will input **quad0**, **focusfac** or **rkxkw** only for linearly-polarized wigglers.

(2) One may add "strong" and equal focusing in both planes by setting **zlion** equal to the wanted (external) betatron wavelength in meters for a particle at $\gamma =$**gammar0**. This is a "kluge" to simulate the long-wavelength effects of AG quadrupole motion but ignores the short wavelength AG flutter. **zlion** also simulates the effects of a focusing ion channel (hence the name).

3.5.7 External Focusing: Discrete Quadrupole Magnet Lattices

GINGER now can include discrete quadrupole magnets in a periodic lattice, which may range from a simple FODO system to more complicated systems such as triplets. At present, only one periodic lattice can be defined but each lattice cell may include up to ten individual quadrupoles, each of which can have its own length and gradient. The necessary input variables are:
(1) **quad_lattice_zperiod**, the full lattice period (m)
(2) **quad_lattice_zstart**, the beginning position (m) of the first lattice cell
(3) **quad_lattice_zend**, the end position (m) of the last lattice cell (the default value of 10^6 m exceeds any reasonable wiggler length)
(4) **quad_lattice_mag_gradient**, an array containing the individual magnet gradients in G/cm. A positive gradient corresponds to focusing in the x-plane
(5) **quad_lattice_mag_start** and **quad_lattice_mag_zend**, arrays defining the beginning and end (m) of the individual quadrupoles *relative* to the beginning of each periodic lattice cell
(6) **quad_lattice_nmag**, the number of magnets in each cell. This is an optional parameter which need not be input because GINGER uses the input gradient values to determine how many magnet truly exist in the lattice.

To make sure that each individual magnet will be "hit" at least once by GINGER's PDE integrator, the maximum step size **hub_m** in meters should be set to approximately 75% or less of the shortest quadrupole's length (see §3.12 for information on numerical integrator parameters). Also, individual magnets should not overlap one another. At present, all magnets have "hard" edges and no fringe field effects are calculated.

3.5.8 Lattice Files, Wiggler and Quadrupole Errors, and Steering Corrections

As of November 2001, GINGER now has the capability for linearly-polarized undulators (only) to model the transport effects of pole excitation errors, quadrupole offset and gradient errors, and dipole steering corrections. As the treatment of these new features matures, they may evolve in various ways including input format, physics description, *etc*. Currently, to input such errors and steering corrections (if any) and model their effects, the user should set the main namelist input variable **lattice_file** equal to the name of special ASCII *lattice file*. This file (which is normally produced by an external program **xwigerr**; see §3.5.9 below) has the following format:

(1) a single line containing the string "GFLD", starting in the first column

(2) an arbitrary number of comment lines with the last one signified by the string "$$" in the second and third

(3) a Fortran90 namelist **&lattice** which should contain the following variables: **gammar0**, **wavelw**, **aw0** (these values should agree exactly without those input in the main GINGER namelist), **nelement** (the total number of discrete sections, including drifts, which describe the undulator) and the variables **nquad** and **nsteer** which define the number of individual quadrupoles and steering elements to be described.

After the namelist, the sets of individual quadrupoles, steering dipoles, and wiggler elements are read (in that exact order). The individual quad format is the string "QUAD" in the first 10 characters, followed by five real numbers giving the beginning and end z-position in meters, the quad gradient in T/m, and the x- and y-offsets in meters. The steering element format is "DIPOLE" in the first 10 characters, followed by 3 real numbers giving the z-position in meters and the x- and y-integrated fields in T-m. The wiggler element format is either "WIGGLER" or "DRIFT" in the first 10 characters, followed by four real numbers. For the case of "WIGGLER", these numbers comprise the initial z-position in meters, the nominal value of aw0 (without errors), the pure "phase error" component of aw0, and the effective "kick error" component of aw0 (which is acts as a transverse dipole magnetic field in the wiggle plane). These error values are usually specially-averaged values over multiple wiggle periods (see §3.5.9). For the "DRIFT" case", only the first two numbers are used which give the beginning and end z positions in meters of the drift.

Note that the configuration described in the **lattice_file**, including drift space and quadrupole lattice information if present, override any values set in the main GINGER input file. Consequently, to avoid inadvertent confusion, the user is advised that when modeling wiggler errors, the **lattice_file** should serve as the primary source of undulator description.

3.5.9 Generation of Lattice File via the XWIGERR Program

A simple F90 program (**xwigerr**), freely available from the author, generates for a linearly-polarized, constant a_w wiggler the individual undulator pole errors and quadrupole gradient and offset errors (if any), computes the appropriately-averaged (see below) errors in phase and drift, the proper transverse steering (if any) including the optional effects of beam position monitor offset errors, and then writes a properly-formatted **lattice_file** (see §3.5.8) to be used by GINGER.

xwigerr uses a simple F90 namelist **&inwigerr** which as much as possible uses the same input variable names as GINGER for equivalent items. However, units in **xwigerr** are MKS for both longitudinal *and* transverse variables, which leads to some inconsistency with GINGER for items such as **xoff**. Electron beam input parameters include **gammar0** for Lorentz factor (alternatively **energy** in MeV may be given), **xoff** and **yoff** in *meters* for initial beam offset, and **xprime** and **yprime** in radians for initial beam angle. Main undulator variables are **wavelw** (or **period**) for undulator period, **zmxmeter** for wiggler length, **rkxkw** to set the amount of curved poleface focusing, and **aw0** for RMS normalized vector potential. Undulator error variables include **aw_rms_err** for RMS *fractional* a_w error, **iseed_aw** for the optional random number

seed for these errors, and **nw_avg** which is the number of undulator periods over which to specially average the pole strength errors (default value 1). One should not average over a length greater than about one-eighth the effective betatron period λ_β of the system. Drift sections are described using the same variables as those in the lattice formulation of the main GINGER namelist (see §3.7.1).

If there are discrete external quadrupoles, the lattice may be described by the exact same variables as in the main GINGER namelist (*e.g.* **quad_lattice_zperiod**; see §3.5.7). Additional quad input variables include **quad_lattice_rms_offset** for the RMS offset in meters, **quad_lattice_rms_grad_err** for the RMS fractional error in quadrupole gradient, and **mag_units** which is either **'T/m'** (default) or **'G/cm'**. The optional random number seed **iseed_quad** can be used to generate the offset and gradient errors.

If discrete dipole steering correction kicks are wanted, the following namelist variables apply: **zsteer_beg_end** for z-interval in which steerer are placed (default full wiggler length), **zsteer_period** for the period interval of steerers, and **zsteer0** which gives location of an optional steerer before the beginning of the periodic lattice — if used, this is normally placed at the beginning of the wiggler. In place of these three variables, the user may instead input discrete steerer $z-$locations via the real array **zsteer**. One may also input RMS beam position monitor transverse errors via **bpm_rms_offset** in meters and a random number seed **iseed_bpm** for such. Presently, the steerers are presumed to have infinitesimal longitudinal length and GINGER applies them as delta-function transverse kicks.

The **xwigerr** namelist variables **outfile** set the main output filename ($\equiv$ the GINGER input **lattice_file**), **sdds_outfile** the SDDS-formatted filename which contains the predicted beam $x-$ and $y-$centroid versus z (including effects of steering), and **description** an ≤ 80-character string may be used to set a comment in the **outfile**.

3.5.10 Checking Beam Transport Properties through the Lattice

In order to check the beam transport (*i.e.* beam envelope size and centroid motion) through the wiggler's focusing lattice, it may be useful to do a single slice, FRED-mode run with the input parameter **nspec**, which controls the number of phase space dumps, being set to a moderate positive value. Then using the **l_plot_xy=.t** switch in the XPLOTGIN preferences file (see §4.13) forces the postprocessor to produce $x-y$ macroparticle scatter plots at various z-locations. History plots of envelope variables σ_x , σ_y , $< x >$, and $< y >$ versus z are also automatically produced in single slice FRED-mode runs. To speed up this sort of debugging run, one can set **lfixfld=.t** to prevent any z-evolution of the radiation field, and **luncoupl=.t** to decouple the macroparticles's z-evolution in γ and θ from the radiation field. Also, the switch **lmovxyt** should be left at its default value of **.true.** in order to retain macroparticle betatron motion.

For debugging and investigatory purposes, GINGER has a number of other logical switches which control details of the physics included in the macroparticle motion. When **lawaxis=.t**,

the radial variation in a_w is ignored in the KMR equation for longitudinal phase advance. Using this switch leads to an unphysical variation in $\gamma_{\|}$ over the betatron orbits of individual macroparticles. The switch **`lmovxyt=.f`** prevents *all* macroparticle betatron motion. The switch **`lxvxfix=.t`** freezes motion *only* in the $\hat{x}$ plane and was implemented for TE0n mode waveguide studies.

A new logical switch added in 2003 can (in effect) force the radiation field centroid to follow that of the electron beam in the transverse plane. When **`l_centroid_axis=.t`**, the transverse center of the axisymmetric radiaiton field is artifically translated to that of the electron beam when evaluating the electric field terms in the energy loss and phase dedrivatives for each individual macroparticle. However, the undulator field a_w is evaluated at the true $x-y$ position. This allows the user to get an indication of phase drift effects for offset/kicked e-beams while still permitting the use of an axisymmtric field solver.

3.6 Wakefield, Space-charge, and Waveguide Specification

3.6.1 Wakefields and Uniform External Accelerating/Decelerating Fields

In some situations, the effects of longitudinal wakefields or other sources of energy loss or gain may be important to simulate. The simplest type of longitudinal field to model is that consisting of a superimposed uniform (in each of t, z and r) accelerating or decelerating longitudinal electric field or equivalent energy loss term (*e.g.* incoherent synchrotron emission). The gain or loss term is set by the input variable **`dgamdz0`**, the change in Lorentz factor γ per meter. A positive value accelerates the macroparticles. A somewhat more complicated, time-dependent energy loss can be chosen by using the input variable **`wake_profile`** in conjunction with **`dgamdz0`**. Setting **`wake_profile='sinusoid'`** will make the external $\partial\gamma/\partial z$ sinusoidal in time with a time period equal to the simulation's periodic window together with a peak amplitude of **`dgamdz0`**. This choice can give some indication of the resonance-detuning effects of longitudinal wakefields.

Beginning in late 2002, GINGER can now use externally-generated wakefield information to simulate the effects of time-dependent (but presently $z-$independent) wakefields. This feature only works in short-pulse mode (*i.e.* not in long pulse mode with temporally periodic boundary conditions). The wake information should be contained in an SDDS-formatted ASCII file whose name is set by the input variable **`wakefile`**. This file must contain at least three columns: one should be the longitudinal position in the beam either given in time (column label **`'t'`** or **`'time'`**, units = seconds) increasing from head to tail or space (**`'S'`** or **`'s'`**; units = meters) decreasing from head to tail. A second column must contain the electron beam current (label **`'current'`**; units = amps) while the third should have the longitudinal wake (label **`'wake_ez'`** or **`'WAKE_EZ'`**; units = volts/m). A *positive* value *decelerates* electrons. Note: the $I(t)$ current profile in the **`wakefile`** will override a profile set by the user either in the input namelist or in an external beam data file (§3.3.6).

Here is an example of SDDS-formatted wakefield file:

```
SDDS1
&description
 text =''lcls-s2e-200pC-CSR-adj-time.slices;pipe r,sigma= 2.50E-03 5.90E+07;
rough period, ampl= 5.00E-05 1.00E-07;geo-wake gap,T= 0.010"   &end
&parameter  name="Peak_Current",  fixed_value=2561.00122,
type=double,  units=Amps,  description="peak current"    &end
&column  name="time",  units=seconds,  type=double
 description="Time"  &end
&column  name="current",  units=Amps,  type=double
 description="Beam Current"  &end
&column  name="wake_EZ",  units=volts/m,  type=double
 description="Total Wakefield Longitudinal Electric Field"  &end
&data  mode="ascii", lines_per_row=0   &end
         128
 -8.900E-14    12.9196   0.000E+00
 -8.763E-14    26.5499  -1.858E+02
 -8.626E-14    52.1701  -5.828E+02
 -8.489E-14   101.8216  -1.328E+03
 -8.352E-14   207.5974  -2.866E+03
 -8.215E-14   306.6741  -4.695E+03
 -8.078E-14   472.4413  -7.986E+03
 -7.941E-14   683.1721  -1.239E+04
 -7.804E-14   882.4681  -1.778E+04
 -7.667E-14  1107.1880  -2.475E+04
```

et cetera ...

The temporal spacing of the wake information should be monotonic in longitudinal position but need not be evenly spaced. A cubic spline fit is applied to the wakefile data for interpolation to the uniform temporal grid of the simulation. A simple Fortran90 code, primarily due to H.-D. Nuhn of SLAC, which calculates the wake associated with resistive losses and beam pipe surface roughness is available from the author.

3.6.2 Longitudinal Space-charge

When modeling high current microwave FEL's such as the LLNL/LBNL ELF experiment, longitudinal space charge forces can become important and affect both particle microbunching and radiation field gain. If **`lspacech=.t`**, GINGER follows these forces using the same approximations as did the original FRED code; the interested user can find physics details in the paper by E. T. Scharlemann *et al.*, *Nucl. Instr. Meth. Phs. Res.* **A250**, 150 (1986). The space charge switch also works for

non-waveguide FEL's. For most problems, inclusion of space charge forces will increase running times by a factor of two or more. The effects of *transverse* space charge fields are not modeled by GINGER; their net force relative to the "pressure" term corresponding to the electron beam transverse emittance is usually very small due to cancellation by the electron beam's azimuthal magnetic field.

3.6.3 Specification of Waveguide Properties

By setting **`lwavegd=.t`**, one instructs GINGER to propagate the radiation within a rectangular waveguide, in which case the radiation's longitudinal wavenumber $k_z \neq \omega_o/c$. The waveguide dimensions are specified via **`xwidth`** and **`ywidth`**, both in cm. Rather than using radial gridding as is true for non-waveguide simulations, in waveguide mode GINGER adopts a Cartesian grid in the y plane and, by default (**`lte21=.f`**), follows the TE01, TE03, TE05, ... modes. For these modes, the electromagnetic field is independent of the wiggle plane direction $\hat{x}$, the (normally) larger waveguide dimension (*i.e.* **`xwidth`** $\geq$ **`ywidth`**). The code does not actually decompose the field into the various TE modes; their strength is diagnosed only in the postprocessor through analysis of the spatial variation of the field on the grid. Note that when doing polychromatic runs, there is *no* variation of group velocity with ω or $k_\perp$; in other words, the slippage velocity is independent of ω.

When **`lte21=.t`**, the code follows the TE01, TE21, TE41, ... modes and, somewhat confusingly, one must set the waveguide "on its side" by making the long dimension in the y direction (this is required because of the internal gridding in y). In this latter case, the TE41 mode will follow a $\cos(\pi x/\texttt{xwidth}) \times \cos(4\pi y/\texttt{ywidth})$ spatial dependence with the coordinate origin lying at the center of the waveguide. Note: if one desires to set a positive quadrupole focusing in the wiggle plane, one still uses a positive value for **`wavelx`** or **`quad0`** even when the wiggle plane is parallel to y for **`lte21=.t`**. This is also true for other focusing parameters such as **`rkxkw`**, and **`focusfac`**. Internally, the code computes k_β with the wiggle plane parallel to x and then exchanges the x and y directions.

Since most microwave FEL's operate at relatively low energies, the approximation $\beta_\parallel \approx 1 - 1/2\gamma_\parallel^2$, which GINGER normally uses, can lead to inaccuracies. These can be prevented by setting the input variable **`lbetapar=.t`**, which forces exact evaluation of $\beta_\parallel$ in the equation determining the macroparticle longitudinal phase derivative $d\theta/dz$.

3.7 Drift Space and Dispersive Section/Optical Klystron Input Variables

3.7.1 Periodic Drift Spaces

GINGER is capable of including spatially *periodic* drift spaces in which the wiggler field strength is zero. Such spaces physically might be used for diagnostics and/or discrete quadrupole focusing.

At present, only one such periodic drift space lattice can be set but within the period, up to five individual drift spaces may be defined. Four input variables in the main namelist specify the drift space lattice: (1) **DL_zperiod** specifies the period in meters of the drift space lattice; (2) **DL_zstart** specifies the point at which the first drift space "lattice" period begins, with a default of the start of the wiggler; (3) **DL_zend** specifies the point at which the last period of the drift space lattice ends, with a default of the end of the wiggler; and (4) the input array **DL_zdrift_beg_end** specifies the physical beginning and end in meters of each of the individual drift spaces *relative* to the beginning of each drift space lattice period.

For example, setting **zmxmeter=100.**, **DL_zperiod=10.**, **DL_zstart=25.**, **DL_zend=75.**, and **DL_zdrift_beg_end = 3. 5. 8.8 9.5** defines a 100-m wiggler containing five drift space 'super-periods' of ten meters each. Within each period, are two individual drift spaces, one 2-m long and the other 0.7-m long. The first 2-m drift space begins at $z = 28\,\mathrm{m}$ and the first 0.7-m long drift space begins at $z = 33.8\,\mathrm{m}$.

Within the drift space, both the electrons and radiation propagate "normally" but with $a_w \equiv 0$ which eliminates any coupling. Hence, there is radiation diffraction but no refraction or gain. In polychromatic mode, GINGER does compute the actual physical slippage occurring in drift sections. In general, this slippage is much smaller than for normal wiggler sections (the opposite from optical klystron sections - see below).

3.7.2 Dispersive Sections/Optical Klystron Configurations

The net gain per pass in a low gain oscillator can often be significantly enhanced placing a dispersive drift section with a non-zero R56 value in the central region of the undulator — this resultant configuration is generally called an "optical klystron". Such devices have also on occasion been proposed for single pass amplifier configurations to enhance gain and reduce overall wiggler length. GINGER has some ability to simulate optical klystron configurations and, at present, a wiggler may include up to 5 individually unique dispersive sections.

To define their properties, one must set **l_extra=.t** to force GINGER the read the **in_extra** namelist within the input (and/or template file). Then, within the **in_extra** namelist, the locations of the individual dispersive sections in meters or are set by giving values in the interval [**zstmeter**,**zmxmeter**] to the array **z_OptKly** (or **t_OptKly** in Rayleigh ranges). The physical lengths of the dispersive sections are controlled by the array **OptKly_zlen** in meters. These lengths define the distance that the radiation field propagates without interacting with the electron beam ($a_w \equiv 0$ as with drift sections). However, diffraction does occur in the dispersive section. Hence, the effective number of wiggle periods available for gain will be reduced from that of the same length wiggler with no drift or dispersive sections.

The strengths of the dispersive sections can be set three different ways: directly by the array **R56** in meters, and indirectly by either the array **OptKly_zwgl_equiv** or the array **d_colson**.

The array **OptKly_zwgl_equiv** (defined in meters) sets R56 in each section to the equivalent dispersion produced by a wiggler of that length. **d_colson** is defined as the ratio of the R56 dispersion produced in each dispersive section divided by that which would be produced by a wiggler of length **zmxmeter** containing only "normal" wiggle periods (*i.e.* an undulator with neither dispersive sections nor simple drift sections). Thus, if **d_colson(2)=0.2**, the wiggler period is 0.1 m, and the total undulator length is 10 m (*i.e.* 100 true periods), the second dispersive section would produce dispersion equivalent to that normally produced in $0.2 \times (10/0.1) = 20$ wiggler periods. Mathematically, `d_colson` $\equiv 0.5 \times$ `R56`$/(N_w \times \lambda_s)$, where N_w is the number of wiggler periods. Note: previous versions of the GINGER manual incorrectly indicated that only the true undulator length (*i.e.* that calculated by ignoring drift and dispersion sections) would be used for calculating the dispersion. However, the code does not do this and thus adding a drift section can change the equivalent **R56** calculated for a given **d_colson**. Consequently, for most simulations, it is probably safest to use **R56** to set the dispersion.

In polychromatic mode, GINGER presumes that each dispersion section will produce extra physical slippage between the e-beam slice and the radiation field. Numerically, the slippage in each section is set to **d_colson** $\times N_w \times \lambda_s$. In reality, the exact value of the slippage will depend upon details of the magnetic design of each optical klystron and might be quite different from this presumed value. Note that when **nsidep** is given as an input parameter (as is normally done for polychromatic runs), the increased slippage in dispersion sections effectively leads to shortened slippage interaction $z-$intervals in each klystron and *lengthened* interaction distances in the normal wiggler sections.

For large values of **d_colson**, one can end up with nearly all the slippage advances occurring in the dispersive sections and very few in normal wiggler sections. This consequently reduces the z-resolution of the diagnostic *pltfile*. To provide partial alleviation for this problem, when running with multiple processors on a MPP platform such as the T3E, one may set **nsidep** larger than **nside** — this permits greater $z-$resolution. Moreover, for each individual dispersion section, diagnostic output will be written out at no more than one $z-$location to the *pltfile*. This change was made to prevent *pltfiles* from being completely dominated by output from the dispersion sections, where there is little physical change in either the e-beam or radiation. To more fully alleviate this problem and give the user more control over the slippage produced by a dispersive section, the new input variable **zslip_equiv_OptKly** sets the slippage in terms equivalent undulator length. Thus, if $\lambda_w = .04$ m and $\lambda_s = 1.0\mu$m, **zslip_equiv_OptKly = 1.0** gives 25.0μm of slippage.

One should note that only the differential phase advance (relative to a hypothetical resonant particle) in a dispersion section is calculated for each macroparticle; *i.e.* the physical slippage mentioned in the previous paragraph does *not* directly affect a macroparticle's longitudinal phase θ. Numerically, $\theta_{out,n} \equiv \theta_{in,n} + (\gamma_n -$ **gammar0**$) \times 2\pi \times$ **d_colson** $\times N_w$. The rationale behind decoupling the physical slippage and the dispersive phase slippage was to avoid having the user

have to worry about zero order (in γ) phase jumps in θ as compared with the more important aspects of the energy-dependent phase jump. Nonetheless, there is an option to include an additional *energy-independent* phase jump by inputting a non-zero value for **theta_drift** in radians - this term does *not* lead to additional physical slippage between the radiation and electron beam in time-dependent mode. One might use **theta_drift** to *unphysically* adjust the net jump in the e-beam longitudinal bunching phase relative to the radiation phase ϕ,

3.8 Oscillator Mode Input Variables

GINGER currently has a limited capability to model FEL oscillator configurations. Setting **losc=.t** in either FRED- or polychromatic mode instructs GINGER to model a "short" electron beam (*i.e.* **nside < nphoton** and non-periodic boundary conditions in time) with the radiation pulse propagating between two mirrors. In fact, when GINGER (when running on a single processor) models a single-pass amplifier with non-periodic boundary conditions (*i.e.* **ltransit=.t**), it secretly forces **losc** to be true and treats the run as a single-pass oscillator. As mentioned previously, oscillator runs require linking the GINGER executable with the IMSL mathematical subroutine library. Presently, GINGER can do oscillator mode runs *only* in serial (single-processor) mode.

Nearly all oscillator input parameters are specified in the *second* namelist **in_extra**; setting **losc=.t** or **l_extra=.t** in the *first* forces the reading of the second namelist. Oscillator input parameters include: **npass** which specifies the total number of passes; **npass_out** (default value $\equiv 1$), the interval (in passes) to write the output radiation field to the postprocessor file; **z_m1_m2**, the intra-mirror cavity length in meters; **rc_m1** and **rc_m2**, the radii of curvature in meters for the two mirrors respectively; **rad_mr**, the actual radius of the mirrors in meters; and **r_cavity**, the total surface reflectance (in power) of the two mirrors taken together. This value of **r_cavity** does *not* include the effects of the finite mirror size; hence, the net reflectivity will be less. The input parameter **d_synch** sets the cavity detuning length measured as a fraction of the slippage length L_s. Note that, following Colson's formalism, a *positive* value of **d_synch** shortens the actual cavity length so that radiation field energy near the head keeps arriving earlier in time relative to the head of the e-beam pulse in successive passes.

3.9 Segment Mode Description and Input Variables

A new type of run mode within GINGER is "segment mode" which permits the user to break the simulation of a very long electron beam pulse into multiple runs. This capability was added to handle the situation in which the electron beam is many hundreds of so-called coherence lengths long which requires the use of many thousands of electron beams slices for adequate simulation. For the ~ 200-fs LCLS, a slice spacing of $\approx 10^{-17}$ sec leads to a need of $\approx 20,000$ slices, generally

more than a thousand CPU hours, and *pltfile* sizes of tens of GB's. Doing all this within a single run is not at all attractive.

In segment mode, one first simulates the tail segment (necessary because the radiation propagates from beam tail toward the beam head) by setting the input variable **l_segment_mode=.t**. This switch prevents periodic boundary conditions and also automatically saves the $z-$dependent radiation field of the "head" slice within the tail segment (see §3.4.6) in an output field file. The user must input the number of e-beam slices via the **nside** variable and also the temporal spacing via either the **nsidep** or **dt_slice** variables. The capability to exploit time-dependent envelope parameters (§3.3.6) or time-dependent macroparticle phase space reconstruction (§3.3.8) is fully available within segment mode. For the first (*i.e.* tail) segment, it is safest to input the exact temporal location of the tail slice by giving a value for **t_start_end_ebeam(2)**.

After the run for the tail segment has been completed, one then simulates the next segment by forcing GINGER to read in the field information (via **l_read_fld_z=.t** in the main input namelist and use of the the option **-f *fldfile*** on the execute line); it is not necessary to set **l_segment_mode=.t** because the use of **l_read_fld_z=.t** instructs GINGER that it will be simulating a beam segment without periodic boundary conditions. For this second segment and all following ones, one should force the writing of a new *fldfile* via the **l_write_fld_z=.t** switch. Because a *fldfile* contains information concerning the temporal spacing between slices and the temporal location of the "head" slice corresponding to the radiation field information, it is not necessary to input either **dt_slice** or **t_start_end_ebeam** for the second and following segments. Likewise, the value of **nside** need not be identical from one segment run to another. For the second segment and beyond, it may also be smaller quantitatively beyond. than **nsidep** since all tne necessary field information is in the *fldfile*. For post-processing purposes, it is best that **nside** be a power of two or three times a power of two. For SASE runs, one should be careful to make sure that a different random number seed for shot noise input **iseeds** is used for each segment (see §3.3.10 and §3.3.11). Also, one should obviously choose some sort of run name sequence by which the different segments can be identified straight-forwardly.

The output *pltfile* from each segment run is fully self-contained and can be post-processed normally. At present (late 2003), there is no automatic way to continguously weave the post-processor output from different segments together. However, via the creation of either ASCII or SDDS-formatted output files, a reasonably adept user can do so manually for time-resolved scalar quantities such as output power or bunching. To date, no such capability exists for linking the spectral output from different segments.

3.10 Harmonic Cascade Capability

Due to the increasing interest in reaching short wavelengths by use of harmonic FEL cascades, a fair amount of work has been put into GINGER to allow the user to simulate such. The typical

configuration of such a cascasde is a series of paired undulators which alternately coherently modulate (*i.e.* microbunch) the electron beam at a wavelength λ_m and then allow the electron beam to radiate coherently at a harmonic wavelength λ_m/N with N ranging from 2 to as much as 5 or 7. The output from each radiator is then used in the following modulator to microbunch the electron beam modulator at wavelength λ_m/N. This sequence can be continued through many stages, with harmonic multiplication occuring in each modulator-radiator pair. In general, the input radiation for the first modulator is provided from a coherent, external source.

At present, a separate GINGER run must be done for *each* modulator and radiator (primarily because no coding exists to permit a jump in harmonic number within a given run). Both monochromatic (FRED-mode) and polychromatic simulation of cascades are supported (although "short-pulse" mode has not been extensively tested).

For each modulator run, one either uses an external radiation source (via **plaser**) or the ouput radiation from the previous radiator stage via a *fldfile* (see §3.4.5) and the **-f fld_file** option on the execute line. If the cascade is configured such that a "fresh" electron beam segment is used for each modulator (via use of a delay chicane - not modeled in GINGER), the electron beam input parameters should be chosen as in a normal (*i.e.* non-cascade) run. If the effects of shot noise are to be included, be sure to set **lshot=.t**. If on the other hand, the same electron beam segment previously used in the upstream radiator will be used, one must read in the restart file via the **l_read_rst=.t** input switch **-rs rstfile** option on the execute line. In order to create a macroparticle restart file for the following radiator run, be sure to set the input switch **l_write_rst=.t** in the main input namelist. Please see §3.3.9 for details concerning *rstfile* IO.

For a radiator run, one *must* read in macroparticle restart file as explained in the previous paragraph. One also *must* instruct GINGER that one is interested in radiation at a harmonic of the previous modulator wavelength by giving an integral value ≥ 2 for the main namelist variable **nhar_mult**. GINGER uses this value to multiply the longitudinal phase θ for each macoparticle in the *rstfile* and also to force the value of the radiators λ_s to be equal to the previous modulator's λ_s/**nhar_mult**. However, due to some coding intricacies, one must also set **wavels** in the radiator input file to the correct value. Finally, one must set **l_write_fld=.t** to write a *fldfile* with the output radiation field from the radiator.

If the "fresh" beam configuration is not used in the cascade (*i.e.* the same macroparticles are used throughout the cascade), one must be careful to set the input vairable **nfold_sym** to a sufficiently high value to account to prevent any artificial numerical microbunching at the *final* wavelength desired to be modeled. For example, if the input wavelength for the first stage modulator is 240 nm and the ultimate final radiator wavelength is 4 nm (*i.e.* a total harmonic upshift of a factor of 60), **nfold_sym** should be at least 120 in the first stage. This can lead to a huge number of particles being necessary to simulate the full cascade in polychromatic mode. Consequently, from both a simulation and gain point of view, the "fresh beam" approach is extremely attractive. For self-consistency in polychromatic simulation, one should adopt periodic boundary conditions in time

(*i.e.* "long pulse" assumption) for this approach.

Finally, in many situations one is likely to use a dispersive section (see §3.7.2) following one or more of the modulator undulators. For purposes of optimizing the R56 parameter in terms of microbuncing, the author has found it most efficient to put the dispersive section at the *beginning* of the following radiator rather than at the end of the preceding modulator because this allows one to do artifically short radiator length runs to optimize the dispersive section. Furthermore, the diagnostics concerning bunching and macroparticle phase space scatterplots may be more informative when examining the wanted harmonic.

3.11 FRED-mode Parameter Scanning Capability

3.11.1 General Parameter Scanning Input Variables

On certain occasions, a user might want to know how sensitive a FRED-mode output parameter (*e.g.* power, saturation position in z, *etc.*) is to a particular input parameter (*e.g.* beam current, energy spread, *etc.*). One could follow a "brute force" approach and do a number of separate FRED-mode runs with the individual input decks scanning the appropriate range in the desired variable input parameter. However, an alternative beckons due to the multidimensional nature of the time-dependent GINGER *pltfile*, It was relatively easy to change the post-processor to produce a *pltfile* with the scanning variable (*e.g.* beam current) replacing time as the independent third dimension (with the first two dimensions being r and z). Likewise, GINGER's monochromatic FRED-mode was then extended to be able to loop through the varying input parameter.

To use this capability, the input deck should be modified as follows: 1) **`nside`** should be set to the number of different values the input variable will take; in general this will probably be in the range 8-32 and need not be related to a power of two; 2) set **`lfred=.t`** to indicate a monochromatic FRED-mode scan rather than a polychromatic run; 3) set **`l_extra=.t`** to force the reading of the **`&in_extra`** namelist; 4) within the **`&in_extra`** namelist, set the input variable **`param_name`** to one of the following allowed strings:

`AW0 PLASER CURRENT BRIGHT EMIT EMIT_MKS GAMMA DGAM DGAMDZ0 BETA_TWISS ALPHA_TWISS R_MISMATCH THETA_DRIFT D_COLSON R56`

Note: the variables *must* be given in upper case (capital letters); 5) set the scanning range for the input variable by giving values to **`param_min`** and **`param_max`** in the **`&in_extra`** namelist.

The default is a linear scan over the full range of the input variable. If **`l_log_param=.t`**, then the scan is done with a geometric scaling (*i.e.* equal increments in the logarithm as one probably would want to use if scanning **`PLASER`** over the range 1 kW to 1 GW). **`DGAM`** refers to scanning in the instantaneous energy spread while **`AW0`** is for scanning a_w in a fixed (untapered) wiggler. Scans in a_w can be very useful if the FEL parameter ρ is quite small and one is trying to maximize gain in a situation where 2D or emittance effects are important. If either **`BETA_TWISS`** or **`ALPHA_TWISS`**

are given, the values will be used for *both* the x and y planes (which may be inconsistent for unequal focusing or a FODO lattice). As mentioned in §3.3.6, use of an "experimental data" file containing $t-$resolved beam envelope data in a multislice FRED-mode run results in parameter-scanning run with time being the independent variable.

If one wants to do a multistage undulator run involving restart files, one can do a scanning mode run over a downstream stage so long as the immediate upstream stage (*i.e.* that which produced a macroparticle or field restart file) was done in single slice mode with **nside=1**. However, even in this situation the downstream run cannot scan in a quantity which attempts to change e-beam parameters such as **GAMMA**. One is allowed to do downstream scans in quantities such as **AW0** or **R56**. Such might be useful in trying to optimize a radiator wiggler or dispersion section immediately following a modulating section (for which only a single slice run needed to be done). Note, when the beam current is the scanning parameter in an multislice upstream run, all downstream runs will use the same current variation. If a multislice upstream run used any other scanning variable (*e.g.* laser input power), a downstream run involving a macroparticle or field restart file can only "scan" over the effective variation of the restart file form slice to slice (*e.g.* one cannot scan over a_w in the downstream undulator).

3.11.2 FRED-Mode Parameter Scanning using Multiple Processors

When doing a FRED-mode parameter scan using the NERSC IBM-SP with its massively-parallel architecture, it is possible to use multiple processors simultaneously for a significant speedup. For the IBM-SP, one need only specify the wanted number of processors on the GINGER execute line (see §2.5.2), making sure that **nside** is evenly divisible by the number of processors. Since the FRED-mode scanning main loop is "embarrassingly" parallel (*i.e.* no slippage effects), multiprocessing here does not involve the complexity of message processing from one processor to another.

3.12 Grid and Numerical Integrator Input Parameters

3.12.1 Simulation Grid

For non-waveguide FEL simulations, GINGER uses a non-linear, expanding radial grid that near the axis is nearly linear in r and then exponentially expands for large r. The outer grid boundary is given by **rmaxsim** in centimeters while the number of radial grid zones is set by **nnd** ≤ 63. The region over which the grid is linear is controlled by the input parameter **rlinear** such that

$$r_n = \texttt{rlinear} \times \sinh\left[(n-1)\,\zeta\right]$$

where ζ is determined by the condition r_{nnd} = **rmaxsim**. A good choice for **rlinear** is the expected radiation mode size. In waveguide mode, the gridding is uniform in the $\hat{y}$ plane.

3.12.2 Numerical Integrator Input Variables

The most important numerical integrator input parameter is **eps**, the maximum normalized error allowed by the predictor-corrector for a step in z to be successful. Typical values lie in the range 10^{-5} to 10^{-4} with the average step size being proportional to $\texttt{eps}^{+1/2}$. One may also set the initial step size (**dt**) the lower bound on step size (**hlb**), and the upper bound on step size (**hub**, default value $0.5\lambda_w/Z_r$), all measured in Rayleigh lengths. **hub_m** is the equivalent upper bound in meters, The final value **hlb** also has a lower bound set by machine accuracy. The input parameter **maxstep** is an upper limit to the number of integrator steps permitted during a full integration interval in z ($\equiv L_w$ / **ncurve** for FRED mode and L_w/**nsidep** for polychromatic mode). If this limit is exceeded the code will abort. The default value is 10,000 and a user should normally not need to set **maxstep** any higher but there can be situations where one might need a larger value. Running with a very high value is potentially dangerous in terms of excessive CPU costs if the code begins acting bizarrely.

3.13 Output Diagnostics Control Variables

There are relatively few input variables to GINGER which control the contents and details of the output files generated for later analysis by the post-processor code XPLOTGIN. When GINGER is run in a polychromatic, multi-slice mode for a single pass amplifier, radially-resolved radiation field quantities and various slice-averaged electron beam quantities are written at the end of every "interaction" interval in z, including $z = 0$. Thus, there are **nsidep + 1** positions in z, **nphoton** positions in time, and **nnd** positions in r for which both the real and imaginary components of E are written to the *pltfile*. As explained in §3.4.5, one can set a switch in polychromatic mode to write out a special radiation field file for restart purposes. For all modes, macroparticle diagnostics written to the *pltfile* for each slice as a function of (t, z) currently include the values of average macroparticle energy, the RMS energy spread, and the average bunching at the fundamental and possibly additional harmonics.

When GINGER is run in monochromatic "FRED" mode (**lfred=.t**), one uses the input variable **ncurve** to set the number of positions in z at which electron beam and radiation field diagnostics will be written to the *pltfile*. In this mode, one can also request the creation of a simple ASCII *datfile* via the **l_datfile=.t** switch. This file will contain simple columnar outputs of items such as the radiation power, macroparticle bunching, RMS delta gamma, *etc.* . For platforms such as DOS/Windows for which presently the post-processor has no direct graphical capability, this switch is default to **.true.**. However, it can be used for all platforms in single slice, FRED-mode. This file can be used directly by **gnuplot** as the non-data rows are begun by "##".

3.13.1 Macroparticle Bunching Diagnostics

Historically previous to 2001, GINGER had always computed macroparticle microbunching at the fundamental wavelength relative to the ponderomotive phase $\psi \equiv \theta + \phi$. The original rationale for this choice was that in many cases there could be significant radiation phase front curvature $\phi(r)$; this is especially true when the radiation mode size is significantly smaller than electron beam size. However, for comparison to coherent transition radiation (CTR) measurements, the bunching should be calculated relative to a a hypothetical plane wave (*i.e.* θ rather than ψ). and it is necessary to do so at higher harmonics (because the field phase ϕ is unknown in GINGER at present). Consequently, beginning in April 2001, the microbunching at the fundamental wavelength will now be calculated relative to a plane wave.

Beginning in GINGER versions of April 2001 and later, one can output diagnostics of microbunching at harmonic number greater than 3. To do so, the vector input variable **nhar_io** should be set to the integer values wanted; note **nhar_io(1)=1** (*i.e.* the fundamental) is hard-wired and cannot be overridden. For example, **nhar_io(1)=1 2 3 4 5 7** will cause microbunching at harmonics from 1 through 5 plus the 7th to be output. The microbunching is a "complex" quantity with both amplitude and phase; a new feature of the postprocessor allows one to calculate the power spectrum $b(\lambda)$ of the microbunching (see §4.5).

Another new diagnostic introduced in late 2003 is one which determines the correlation between particle phase θ and energy γ. If the string **'GAMBUNCH'** is included in the input for the main namelist string variable **scalar_diags**, GINGER will calculate and ouput to the *pltfile* at each $z-$diagnostic location the value of $\sum_j \exp(i\theta_j)\,(\gamma_j - \bar{\gamma})$. This diagnostic is useful for determining the bunching in a modulator section of a multistage FEL.

3.13.2 Macroparticle Phase Space Scatterplot Output

Beginning in 2003, GINGER can now write macroparticle phase space information in a dump *spcfile* for both monochromatic FRED-mode runs and multislice polychromatic mode runs. Since each scatterplot involves $\tilde{5}$0kB of information for each 1024 macroparticles, one should be sparing in the number of slices in time for which output will be written. For all modes, the discrete z locations for the phase space information can be set in multiple ways:

(a) a positive integral value for either **nspec** or **nz_scatterplot** sets the number of such locations which include and are evenly spaced between $z = 0$ and $z =$**zmxmeter**.

(b) a positive real value to **dz_scatterplot** defines the $z-$spacing of the scatterplots (again beginning with $z = 0$ and ending with $z =$**zmxmeter**

(c) inputting discrete values to real array **z_scatterplot**

For multislice polychromatic (time-dependent) runs, the user has the choice of two positive integer input variables to define the t interval between scatterplots at a given z:

(a) **`nt_scatterplot`** which defines the total number of slices (approximately evenly spaced in t) for which scatterplot output will be generated

(b) **`nt_scatterplot_mod`** defines the interval in slice number between diagnosed slices.

As of 2002, the output longitudinal phase of the individual macroparticles is now measured relative to a hypothetical plane wave (*i.e.* θ_i) rather than their position relative to the instantaneous ponderomotive well (*i.e.* ψ_i). However, the latest versions of the *spcfile* now include both radial grid information and the complex radiation field for each output slice which allows the calculation of the ponderomotive phase ψ if wanted.

3.13.3 Reducing the z-Frequency of Diagnostic Output to the *Pltfile*

For a polychromatic, non-oscillator mode run, GINGER by default writes the radiation field and macroparticle diagnostics for each slice at the end of every interaction (slippage advance) interval in z. One can reduce the $z-$frequency of these diagnostics (and thus the total size of the *pltfile*), by setting **`n_diag_mod`** to an integral value greater than one. A choice of three, for example, would produce output at the end of every third interaction interval for each slice. Runs with long optical klystron dispersive drift sections will generally produce output at fewer than (**`nsidep + 1`**) $z-$locations (see §3.7.2). The postprocessor automatically deals with this change.

In multi-pass, oscillator mode, GINGER writes out field and particle quantities at only one z location (the physical end of the wiggler) for each pass. This is true for both monochromatic and polychromatic oscillator runs. Hence, there is no z-resolved information within the wiggler. The input parameter **`npass_out`** controls the frequency at which diagnostic information is written to the *pltfile*. The default of **`npass_out=1`** is equivalent to output being written every single pass; a choice of **`npass_out=2`** would lead to output every other pass, *etc.* .

3.14 Sample GINGER and XWIGERR Input Files

The following input files were chosen as examples of the various types of FEL configurations which can be modeled with GINGER. They and some post-processor output CGM files are available with in the "tar" archive **`GNX_TEST.tar`** which is stored in the public HPSS space at NERSC described in §2.2.

3.14.1 Monochromatic, "Paladin" Tapered Wiggler Self-Design: `inpalSD`

This input file adopts "nominal" LLNL Paladin parameters and calculates the time-independent (*i.e.* monochromatic) power gain from a 10-MW input laser at $\lambda_s = 10.6\mu$m over a 80-mm period,

linearly-polarized 25-m long wiggler. Curved pole faces (**rkxkw=0.707**) are adopted for e-beam focusing. This run also uses the GINGER tapered wiggler self-design algorithm (via **idesign=2**) to compute $a_w(z)$ and sets the design macroparticle at 0.285 cm. There will be 1024 macroparticles with only 2-fold longitudinal symmetry as we are not concerned with third harmonic growth. In addition to the creation of a "normal" post-processor file (named **pltpalSD**) which will have data at 51 $z-$locations, GINGER will also output a *bwfile* named **bwpalSD** containing $a_w(z)$, a macroparticle dump file (**spcpalSD**) containing 6-D phase space snapshots at six equally-spaced positions in z down the wiggler (*i.e.* every 5 meters beginning at $z = 0$). This problem requires 3.9 seconds on the Killeen Cray-J90 (SV1 processor), 8.0 seconds on a 1.1-GHz Pentium-3M under Linux, and 5.2 seconds on the IBM-SP at NERSC. To run this problem one would type: **xginger r=palSD**

```
Paladin FRED-mode run - B=2.e5 ; 2.5 kA ; 10 MW input ;
25-meter tapered wiggler self-design ; eps=6.7e-5 ;
inpalSD $
 &in
 jmg = -1
 current = 2.5e3
 bright = 2.5e5
 gammar0 = 98.85
 lshot = .f
 ntestp = 1024
 nfold_sym = 2
 wavels = 1.06e-5
 plaser = 1.0e7
 wavelw = .08
 llinear = .t
 rkxkw = 0.70711
 lcnstwgl = .f
 idesign = 2
 rdesign = 0.285
 lfred = .t
 ncurve = 50
 nz_scatterplot = 6
 rmaxsim = 4.
 nnd = 45
 zmxmeter = 25.0
 eps = 6.708e-5
 /END
```

3.14.2 Paladin Sideband Growth in a Tapered Wiggler: inpalacSTD

This simulation initializes a white noise spectrum to provide a seed for sideband growth to a Paladin-like e-beam. With **wattpbin=2000.**), the total noise power is 192 kW. The input value **idesign=0** and **lcnstwgl=.f** indicate that the $a_w(z)$ is pre-determined and tapered; it will be

read from an external file (**bwpalac**). By default, there are periodic temporal boundary conditions with 80 slippage advances over 25 m (or approximately every 30 cm, much smaller than the gain length) and 96 separate e-beam slices. This example took 1250 seconds in serial mode (1 CPU) on the IBM-SP, 230 seconds in 6-CPU parallel mode on the IBM-SP, 1280 seconds in 4-CPU parallel mode on the Cray-T3E, and 6630 seconds on a 400-MHz Pentium II under Linux. To run this case on a single processor, the user would type: **xginger r=palacSTD b=bwpalac** To run this case on an IBM-SP employing 4 processors in parallel, the execute line would be: **xginger-mpp -procs 4 r=palacSTD b=bwpalac**

```
Test case to study Paladin sideband growth -
needs wiggler file bwpalac
B=2.e5 ; 2.5 kA ; 10 MW input ; 25 meter wiggler
nsidep = 80 ; 96 e- slices ; eps=5.e-5 ;
inpalacSTD $
&in
jmg = -1
current = 2.5e3
bright = 2.5e5
gammar0 = 98.85
ntestp = 2048
lshot = .f
wavels = 1.06e-5
plaser = 1.0e7
zfocus = 0.0
zmxmeter = 25.0
wavelw = .08
llinear = .t
rkxkw = 0.70711
lcnstwgl = .f
idesign = 0
lfred = .f
nside = 96
nsidep = 80
wattpbin = 2000.
iseed = 453774
rmaxsim = 4.
nnd = 47
dt = 1.e-6
eps = 5.e-5
/END
```

3.14.3 Long pulse, LCLS 1.5Å SASE x-ray FEL: `inlcls_fodoSb`

This particular input file sets up a full polychromatic simulation of the proposed LCLS x-ray FEL at SSRL/SLAC, operating at 1.5 Angstroms with a 14.35-GeV electron beam. By default, this sim-

ulatioin will employ periodic boundary conditions in time. Each electron beam slice has a small amount of instantaneous energy spread. Although particle shot noise provides the dominant seed for radiation growth, a small level of white noise power is put in at $z = 0$ via the **wattpbin** input parameter; at present this is need to keep GINGER's predictor-corrector happy at $z = 0$. Note that the random number seeds for both the shot noise and the random radiation field noise are preset by the user. The longitudinal window length is set by **nsidep**, the total number of slippage advances occuring for each of the 192 electron beam slices over the full wiggler length. The linearly polarized wiggler has a constant a_w strength whose on-axis value is preset at 2.622 by the input variable **aw0**. The initial transverse phase space orientation for the electron beam in each plane is set by the input values of the Twiss parameters. The external focusing is a FODO-type lattice with a period of 4.31 meters. Each magnet is associated with a drift space whose length is 0.135 meters. The radial simulation grid is approximately linear out to 40 microns and has an outer boundary of 150 microns. The full simulation length (including drift spaces) is 112.5 meters with an initial step-size of 10^{-6} Rayleigh ranges, a minimum step-size of 10^{-8} Rayleigh ranges, and an upper step-size limit of 0.03 meters (this ensures each quad magnet will be "hit" at least once by the adaptive $z-$stepsize integrator). To run this case (on a single processor), one would type: **xginger r=lcls_fodoSb** To use 8 processors on the IBM-SP (presuming the MPP version of the GINGER executable is named **xginger-mpp**), one would type: **xginger-mpp -procs 8 r=lcls_fodoSb**

```
LCLS 1.5-A FEL ; SASE initiated (5W/bin);
112.5-m wiggler; wavelw=3.0 cm; Gaussian profile e-beam
rlinear=40um ; rmaxsim=150 um ; no radial smoothing
external focusing from standard FODO lattice; enhanced resolution
eps_n=1.5e-6 m-rad ; 3.4 kA ; 1024 particles ; inlcls_fodoSb $
 &in
 jmg = 2
 current = 3400.
 emit_mks = 1.5e-6
 gammar0 = 28077.
 gam_load = 'gaussian'
 dgamma = 6.0
 ntestp = 1024
 lshot = .t
 iseed(1:2) = 845517 926417
 iseeds(1:2) = 345823 482366
 lfred = .f
 nside = 192
 nsidep = 150
 wavels = 1.5e-10
 plaser = 0.e4
 wattpbin = 5.
 omg0fac = 0.8
 wavelw = .03
```

```
llinear = .t
lcnstwgl = .t
idesign = 1
aw0 = 2.622
rkxkw = 0.
quad_lattice_nmag = 2
quad_lattice_zperiod = 4.310
quad_lattice_zend = 500.
quad_lattice_mag_zstart = 1.9775 4.1325
quad_lattice_mag_zend = 2.0975 4.2525
quad_lattice_mag_gradient = 4440.0 -4413.0
betax_twiss = 16.0552
alphax_twiss = -0.8918
betay_twiss = 19.8175
alphay_twiss = 1.0952
DL_zperiod = 2.155
DL_zdrift_beg_end = 1.920 2.155
rlinear = 40.0e-4
rmaxsim = 150.e-4
nnd = 47
zmxmeter = 112.5
dt = 1.e-6
hlb = 1.e-8
hub_m = 0.03
eps = 4.0e-05
/END
```

3.14.4 Sample LCLS Wiggler Error Input File for XWIGERR Program

This input file is used by the **xwigerr** program to generate wiggler and quadrupole errors together with corresponding steering for an LCLS run with parameters similar to the previous input file. The output **lattice_file** *lcls_bpm-50off.lat* can subsequently be used by GINGER. The net effects of the pole errors (normalized RMS size = 1 part in 10^3, much larger than actually wanted for the LCLS) are averaged over 6 wiggle periods, which leads to an output interval of 18 cm. Note that one needs to specify **mag_units = 'G/cm'** for the input quad magnet gradients because the default unit used in **xwigerr** is T/m. The steering lattice has a 4.31-m period which begins at 2.04 m; there is an additional steerer at $z = 0$. The rms transverse position error for the beam position monitors and the quadrupole centroids is set to 50 microns (note: this is more than 10X greater than is expected for the actual LCLS). To run this input file, one would type **xwigerr i=inlcls_wigerr-A**

```
sample input file for XWIGERR program
LCLS-type undulator; 0.16-period undulator period averaging
inlcls_wiggerr-A
```

```
&INWIGERR
gammar0 = 28077.
zmxmeter = 120.00
period = 0.03
nw_avg = 6
aw0 = 2.622
aw_rms_err = 0.001
iseed_aw(1) = 135598
bpm_rms_offset = 50.e-6
iseed_bpm(1) = 1114011712
quad_lattice_rms_offset = 50.e-6
iseed_quad(1) = 123721
quad_lattice_nmag = 2
quad_lattice_zperiod = 4.310
quad_lattice_zstart = 0.
quad_lattice_zend = 500.
quad_lattice_mag_zstart = 1.9775 4.1325
quad_lattice_mag_zend = 2.0975 4.2525
quad_lattice_mag_gradient = 4440.0 -4413.0
mag_units = 'G/cm'
DL_zperiod = 2.155
DL_zdrift_beg_end = 1.920 2.155
zsteer0 = 0.
zsteer_period = 4.31
zsteer_period = 2.155
zsteer_beg_end(1) = 2.04
outfile = 'lcls_bpm-50off.lat'
description = '6 period avg w; quad + pole errors + steering'
/END
```

3.14.5 Example of a LCLS-parameter Template File Use: `inlcls-errA2`

This input file relies upon the use of **`inlcls_fodoSb`** as a "template" file and extends it by using the wiggler error file produced in the previous example. By setting **`dgamma`**, the input file **`inlcls-errA2`** also lowers the energy spread to 3.0 from the template's 6.0, the normalized emittance to 1.0 mm-mrad, increases the number of macroparticle's to 2048, resets the random number seeds for the energy spread and shot noise, and adds an x-offset of 30 microns to the initial e-beam centroid (remember, GINGER nominally uses units of centimeters for transverse quantities such as **`xoff`** and **`yoff`**). To run this example on a single processor, the user would type:

```
xginger r=lcls-errA2 t=inlcls_fodoSb
```

```
LCLS 1.5-A FEL ;
relies on template file inlcls_fodoSb
emit -> 1 mm-mrad; 30 micron x-offset; dgamma -> 3.0 ;
```

```
2048 particles;
uses lattice file lcls_bpm-50off.lat
inlcls-errA2 $
&in
dgamma = 3.0
ntestp = 2048
iseedp(1:2) = 2001 1345
iseeds(1:2) = 1492 2001
xoff = 30.e-4
emit_mks = 1.0e-6
lattice_file = 'lcls_bpm-50off.lat'
/END
```

3.14.6 Example of a NERSC Batch Script to run an LCLS Case

The following script can be submitted to the NERSC batch system for the IBM-SP with the **llsubmit** command to run the previous example (*lcls-errA2*) under batch. Four multiprocessor "nodes" are requested, each with 16 processors, for a grand total of 64 processors. NERSC strongly suggests that batch runs be done in the user's "scratch" directory, which thus requires that the necessary executables, input, lattice, and template files all be copied to that space. If the MPP-ready GINGER executable exists in a directory which lies in the user's "path", it is not necessary to copy the executable. After completing the GINGER simulaton, the 64 resultant *pltfiles* will be concatenated into one big file which then will both be stored in HPSS archive space and written to the user's local disk space.

```
#COMMENT: sample script for a GINGER LCLS batch run
#
#@ class = regular
#NOT USED: class = debug
#NOT USED: class = premium
#
#@ shell = /usr/bin/csh
#@ node = 4
#@ tasks_per_node = 16
#@ network.MPI = csss,not_shared,us
#@ wall_clock_limit = 1:55:00
#@ notification = always
#@ job_type = parallel
#@ output = $(jobid).$(stepid).out
#@ error = $(jobid).$(stepid).err
#COMMENT: following ''mpXXX'' should be replaced with
# user's NERSC account #
#@ account_no = mpXXX
#@ environment = COPY_ALL
```

```
#@ queue
set echo
echo "IBM-SP seaborg batch run"
alias cd cd
alias cp cp
alias rm rm -rf
#---copy GINGER executable and needed files to scratch space:
cd $SCRATCH
set IDIR = $HOME/FEL/LCLS
cp $HOME/FEL/MPP-OBJ/xginger ./xginger-mpp
set name = lcls-errA2
set infile = in$name
cp $IDIR/$infile .
set template = inlcls_fodoSb
cp $IDIR/$template .
latfile = lcls_bpm-50off.lat
cp $IDIR/$latfile .
#---------------------- execute instructions:
echo "beginning actual GINGER execution"
xginger-mpp r=$name t=$template
#------concatenate multiple pltfiles to one big one:
rm plt$name
cat plt0*$name > plt$name
#------save pltfile, fldfile to local disk space:
cp plt$name $HOME/FEL/BATCH
cp fld$name* $HOME/FEL/BATCH
#-------------------- HPSS storage instructions:
echo "beginning HPSS storage sequence"
hsi << EOR
cd BATCH
put plt$name
quit
EOR
echo "completed BATCH run"
exit
```

3.14.7 LCLS-parameter Tail Segment Run: `ins2e-tail-1nC-A`

This input file relies upon the use of an LCLS-parameter "template" file (*e.g.* **`inlcls_fodoSb`**) for undulator and radiation input and extends it by giving the appropriate input describing the tail segment of a multi-segment run. In this particular class ELEGANT tracking particle information will be used to reconstruct a detailed 6D phase space via use of a summary file (*i.e.* **`s2e-1nC-CSR-tail-slice-sum.time`**). This summary file also contains the name of

the ELEGANT macroparticle file to be used in the phase space reconstruction. A temporal slice spacing of 12 attoseconds will be used here (and for all following segment runs in this series). The tail slice of this run is located at $t = 105$ fs (with absolute time defined in the ELEGANT macroparticle file). The head slice (for this particular run) will be at a point 191×12 as earlier in time. This particular example is a SASE run and diagnostics from only every 4th slippage-advance location in z will be saved to the *pltfile*. A $z-$dependent field file will be written to disk to permit proper initialization of the next electron beam segment in this run series. To run this example on a single processor, the user would type: **`xginger r=s2e-tail-1nC-A t=inlcls_fodoSb`**

```
Sample of input file for first (tail) segment run
data from emma's standard ELEGANT S2E-ICFA03 output;
current appropriately adjusted; full SASE run
ins2e-tail-1nC-A $

&in
tracksumfile = 's2e-1nC-CSR-tail-slice-sum.time'
current = 300.
nfold_sym = 2
lfred = .f
iseeds(1:2) = 142312 131203
lshot = .t
t_start_end_ebeam(2) = 105.e-15
l_write_fld_z = .t
l_segment_mode = .t
plaser = 0.01
wattpbin = 1.
nside = 192
dt_slice = 1.2e-17
n_diag_mod = 4
/END
```

3.14.8 "Mid" Segment Run: `ins2e-tail-1nC-B`

This input file continues the multisegment run begun with the input file of the previous example. It both reads in the $z-$dependent field file of the previous run (and will write a new one following its own completion) and the tracking code summary file. Note that is not necessary to set either **`dt_slice`** or **`t_start_end_ebeam(2)`** as they are effectively contained within the field file. Furthermore, the **`nside`** in this run is quite different from that of the previous, tail segment run. To run this example on a single processor, the user would type: **`xginger r=s2e-tail-1nC-B t=inlcls_fodoSb f=flds2e-tail-1nC-A.sdds`**

```
runs to look at mid pulse region in LCLS for ICFA03-S2E
data from emma's standard elegant output
full SASE run using summary file using fldfile from run A
```

```
ins2e-tail-1nC-B $
&in
tracksumfile = 's2e-1nC-CSR-tail-slice-sum.time'
nfold_sym = 2
lfred = .f
iseeds(1:2) = 7321122 131203
lshot = .t
l_write_fld_z = .t
l_read_fld_z = .t
plaser = 0.
wattpbin = 1.
nside = 1024
n_diag_mod = 8
/END
```

3.14.9 Initial Modulator in Harmonic Cascade: `inlux-500A-240nm-mod`

This input file corresponds to the first modulator stage of a harmonic cascade with parameters similar to the LUX project being studied at LBNL. The input external laser has a 240-nm wavelength and 1 GW power which is focused to a 300-micron waist 1.25 m into the undulator. A dispersive section at $z = 3.6\,\mathrm{m}$ is used to improve the microbunching to $b_1 \approx 0.5$ before the electron beam is inserted into the next undulator. A 16-fold symmetric quiet start is employed (such a high number is neeeded because the radiator will utilize the 5th harmonic of 240 nm). Bunching harmonics 1 through 6 will be diagnosed via the **nhar_io** input variable. A macroparticle restart file will be written at the end of the dispersive section. This is a single-slice, FRED-mode run and scatterplot phase space diagnostics are requested every 1.2 meters in the "normal" undulator and also at the end of the dispersive section at $z = 4.0\,\mathrm{m}$. This run took 27.1 seconds on a Sunblade150 (equivalent to a 2-GHZ PC). To run this example, the user would type: **xginger r=lux-500A-240nm-mod**

```
Hypothetical LUX laser modulator at 240 nm ;
500 Amps; 2 pi mm-mrad ; delta gamma = 1.38
1 GW input laser focused at z=1.25 m;
dispersive section with R56=12 microns (d_colson = 0.75)
inlux-500A-240nm-mod $

&in
jmg = 2
current = 500.
emit_mks = 2.0e-6
gammar0 = 4894.
dgamma = 1.38
gam_load = 'gaussian'
ntestp = 16384
```

```
lshot=.f
nfold_sym = 16
nhar_io = 1 2 3 4 5 6
wavels = 240.0e-9
plaser = 1.0e9
zfcmeter = 1.25
omg0 = 300.e-4
wavelw = .12
llinear = .t
idesign = 1
awdmult = 0.995
rkxkw = 0.70711
nside=1
l_write_rst = .t
lfred = .t
ncurve = 40
z_scatterplot = 1.2 2.4 3.6 4.0
rmaxsim = 0.25
rlinear = 120.e-4
nnd = 55
zmxmeter = 4.0
dt=1.e-5
hlb = 1.e-8
eps = 4.e-5
maxstep = 10000
l_extra = .t
/END
&in_extra
z_OptKly = 3.6
d_colson = 0.75
OptKly_zlen = 0.39
/END
```

3.14.10 Radiator in Harmonic Cascade: `inlux-500A-48nm-rad`

This input file corresponds to the 48-nm wavelength radiator which follows the 240-nm modulator of the previous example. For these parameters, ≈ 185 MW of power is radiated by the end of the undulator. This example took 160 seconds on a SunBlade150. To run this example, the user would type: **`xginger -r lux-500A-48nm-rad -rs rstlux-500A-240nm-mod`**

```
48-nm radiator; 4.0-m long undulator;
bunched beam from 240-nm modulator rstfile
inlux-500A-48nm-rad $

&in
```

```
wavels = 48.0e-9
nhar_mult = 5
plaser = 100.
zfocus = 1.0
omg0 = 100.e-4
wavelw = .075
lcnstwgl = .t
llinear = .t
idesign = 1
awdmult = 0.9975
rkxkw = 0.70711
nside=1
lshot=.f
l_read_rst = .t
l_write_fld = .t
lfred = .t
ncurve = 50
rmaxsim = 0.12
rlinear = 120.e-4
nnd = 55
zmxmeter = 4.0
z_scatterplot = 0.0 2.0 4.0
dt=1.e-5
hlb = 1.e-8
eps = 4.e-5
maxstep = 10000
/END
```

3.14.11 Second Modulator in Harmonic Cascade: `inlux-500A-48nm-mod`

This input file corresponds to the second modulator stage of the LUX-like harmonic cascade corresponding to the previous 2 expamples. Since we are presuming we are starting with a "fresh" electron beam, we will use the first modulator input file as our template and only change those items which are unique to this stage (*e.g.* wavelength, radiation from upstream radiator rather than external laser, *etc.*).

This simulation took 25.5 seconds on a SunBlade 150. To run this example, the user would type:

```
xginger r=lux-500A-48nm-mod t=inlux-500A-240nm-mod f=fldlux-500A-48nm-rad
```

```
500-amp 48-nm modulator to bunch beam; 5.0-m long undulator
fld from 48-nm radiator:  fldlux-500A-48nm-radA
aw chosen for peak bunching;
disp section:  R56=8.48 microns (equiv.  d_colson = 1.25)
inlux-500A-48nm-modA $

&in
```

```
wavels = 48.0e-9
plaser = 1.
wavelw = .075
aw0 = 5.4385
awdmult = 1.0
l_write_rst = .t
zmxmeter = 5.3
z_scatterplot = 3.0 4.0 5.0 5.3
l_extra = .t
/END
&in_extra
z_OptKly = 5.0
R56 = 8.48e-6
OptKly_zlen = 0.29
/END
```

3.14.12 Short pulse, UCLA single-pass SASE expt.: `inUCLAt2`

This input file is used to simulate the behavior of the mid-1990's UCLA SASE IRFEL which operated at a central wavelength of 10.6 microns. This is a "short-pulse" simulation as set by **`ltransit=.t`**. With a period of 15 mm and 0.6-m length, the wiggler is 40 periods long and, choosing **`nsidep=20`**, the photon slices advance forward one electron beam slice every two wiggle periods. With **`nside=60`**, the full width of the electron beam is 3 slippage distances long, which is equivalent to 1272 microns or 4.24 ps in time. The electron beam has a parabolic current profile (via **`pulse_shape`**) in time which leads to a FWHM pulse of 3.0 ps. **`nphoton=96`** is greater than **`nside+nsidep+1`** as required. Shot noise is initialized via **`lshot=.t`** and the initial radiation is composed a flat noise spectrum with exactly 100 nanowatts per bin. There is no external focusing in the wiggler plane, but there is presumed to be a simple lens at $z = 0$ whose focal length in the wiggle plane ($\hat{x}$) is set to 0.35 meters via **`xfocus_mtr`**. The initial beam size in x is 0.03 cm (=300 microns); the y-size is set to be in equilibrium with natural wiggler focusing. This is a relatively fast running problem and took ∼3400 CPU-seconds on the NERSC IBM-SP. The execute line to run this problem is: **`xginger r=UCLAt2`**

```
test input file for UCLA IR expt ;
4 ps beam with a parabolic longitudinal profile (== 3.0 ps FHM)
gaussian profile transversely;
I_b=200 A ; 0.1% energy spread ;
linear wiggler with no external focusing ;
shot noise input source
inUCLAt2 $
 &in
 jmg = 2
```

```
 current = 200.
 emit_mks = 5.e-6
 gammar0 = 33.5
 dgamma = 0.0335
 gam_load = 'gaussian'
 ntestp = 2048
 lshot = .t
 iseeds(1:2) = 22813 23785
 wavels = 10.6e-6
 plaser = 0.
 wattpbin = 1.e-7
 iseed(1:2) = 43242 97543
 omg0 = 0.04
 zfocus = 0.5
 ltransit = .t
 pulse_shape = 'parabolic'
 nsidep = 20
 nside = 60
 nphoton = 96
 zmxmeter = 0.6
 rmaxsim = 0.75
 rlinear = 0.05
 nnd = 47
 wavelw = .015
 llinear = .t
 lcnstwgl = .t
 idesign = 1
 xfocus_mtr = 0.35
 omgjx = 0.03
 dt = 1.e-5
 hlb = 1.e-8
 eps = 1.0e-04
 /END
```

3.15 Names and Default Values for GINGER Input Namelist Parameters

```
!..RADIATION FIELD Parameters:
!
  nmg         = 2        ! radiation field radial profile index
  omg0fac     = 0.8      ! scale factor for input radiation field mode size
  omg0        = -1.      ! if omg0<0, then omg0=omg0fac*omgj
  wavels      = 1.06e-5  ! radiation wavelength (m)
  plaser      = 2.4e9    ! initial radiation power (W)
  zfocus      = 0.       ! laser focus position in Rayleigh ranges
```

```
  zfcmeter    = -1.      ! zfocus in meters
  dgamdz0     = 0.       ! constant electrostatic accelerating field
  ghz         = -1.      ! input radiation signal frequency in GHz
  lfixfld     = .false.  ! switch to force non z-evolving radiation field
  lvacfld     = .false.  ! radiation field evolves in an empty ‘‘vacuum’’
                         !  (no gain or refraction)
  l_nodiffrac = .false.  ! switch to turn off diffraction
  l1d         = .false.  ! switch for 1-D field and particle interaction
  l1d_par     = .false.  ! switch for 1-D particle interaction
  l_centroid_axis = .true. ! if .true., radiation centroid remains on axis
                         ! if .false., radiation centroid follows e-beam centroid
!
!..ELECTRON BEAM Parameters:
!
  current   = 2.e3     ! initial beam current (Amps)
  omgj      = -1.      ! e-beam gaussian half-width (cm)
  omgjx     = -1.      ! e-beam radius >  x-plane
  omgjy     = -1.      ! e-beam radius >  y-plane
  omgjfac   =  1.      ! multiplier of initial beam for equilibrium in r
  jmg       =  2       ! electron beam profile index (Gaussian default)
  rmaxcur   = -1.      ! radius at which gaussian e-beam is truncated

  betax_twiss  = 0.    ! Twiss beta parameter (m) in x plane
  alphax_twiss = 0.    ! Twiss alpha parameter in x plane
  betay_twiss  = 0.    ! Twiss beta parameter (m) in y plane
  alphay_twiss = 0.    ! Twiss alpha parameter in yplace
  xoff         = 0.    ! initial electron beam offset in x (cm)
  yoff         = 0.    ! initial electron beam offset in y (cm)
  xprime       = 0.    ! initial beam tilt in x (dx/dz)
  yprime       = 0.    ! initial beam tilt in y (dy/dz)

  bright    = -1.      ! electron beam brightness (A/cm**2) [LLNL definition]
  emit0     = -1.      ! beam normalized edge emittance (rad-cm)
  emit_mks  = -1.      ! beam normalized edge emittance (rad-m)

  gammar0   = -1.      ! initial electron gamma
  energy    = -1.      ! beam energy in MeV
  dgamma    = 0.0      ! half or RMS width of initial electron energy distrib.
  gam_load  = ’uniform’! type of electron energy distribution
  ntestp    = 1024     ! total number of simulation macroparticles
```

```
  nfold_sym = 8        ! symmetry in theta-load (>= 8 needed for 3rd harmonic)
  dpsir0    = -1.      ! theta part. loader: half width of psi around psir0

  exp_data_file = 'NOT SET' ! input data file name containing arrays
                            !  of empirical beam parameter info
  t_start_end_ebeam = (/ -1.e6, 1.e6 /)  ! start & end position (sec) of
                           ! electron beam segment in data file or analytic profile

  lmovxyt   = .true.   ! switch controlling macroparticle betatron motion
  lbetapar  = .false.  ! exact beta-parallel used in d theta/dz eqn.
  luncoupl  = .false.  ! uncouples particles from field for debugging
  lxvxfix   = .false.  ! constant transverse particle positions & velocities
  loval     = .true.   ! see settestp: oval-shaped particle beam
  lquiet    = .true.   ! switch to give ''quiet-start'', symmetric theta load
  lshot     = .true.   ! switch for shot noise in e-beam
  lrandpsi  = .false.  ! switch for random rather than bit-reversed psi
  lprobe    = .false.  ! switch to use probe particles
  lnouperp  = .false.  ! leaves out gam**2 betaperp**2 term in dpsidz eq

!..Wakefield energy loss input parameters:
  l_wakefield  = .false.   ! switch to set wakefield energy loss/gain
  wakefile     = 'NOT_SET' ! input file for wakefield description
  wake_profile = 'NOT_SET' ! hard-wired wake (linear, sinusoid, constant)
!
!..FOCUSING Parameters:
!
  quad0    = 0.0       ! strength of external quadrupole focusing (G/cm)
  wavelx   = -1.       ! x-plane betatron wavelength (m) - waveguide mode
  rkxkw    = -1.       ! curved poleface focusing parameter
  focusfac = 1.        ! tapered quadrupole focusing (if idesign >= 1)
  zlion    = 0.0       ! betatron wavelength (m) of axisymmetric strong focus.
  xfocus_mtr = -1.e3   ! e-beam focus point (m) in x-plane
  yfocus_mtr = -1.e3   ! e-beam focus point (m) in y-plane
  xbscale  = 1.0       ! multiplier of initial x-beam size vs. equilibrium
  ybscale  = 1.0       ! multiplier of initial y-beam size vs. equilibrium
  lequil   = .true.    ! switch to load e-beam in equilibrium with focusing

  lattice_file   = 'NOT SET' ! ''lattice'' file with a_w, quad, bpm errors
                             !   and dipole steering corrections
```

```
  quad_lattice_cst_gradient =  0.0 ! gradient for AG quad focusing (G/cm)
  quad_lattice_mag_gradient =  0.0 ! gradient for quad(G/cm)
                                   !   (positive value for focusing in x)
  quad_lattice_zperiod      = 20.  ! full AG quad lattice period (2L)
  quad_lattice_zstart       =  0.0 ! z-beginning of periodic quad lattice (m)
  quad_lattice_zend         = 1.e6 ! z-end of periodic quad lattice (m)
  quad_lattice_nmag         =  0   ! number of distinct quads perlattice period
  quad_lattice_mag_zstart   =  0.0 ! beginning z of individual quad in lat. pd.
  quad_lattice_mag_zend     =  0.0 ! end z of individual quad in lattice period

  DL_zperiod = -1.0  ! length of full drift section "module"
  DL_zstart  = -1.0  ! point at which drift modules begin
  DL_zend    = 1.e6  ! point at which drift modules begin
  DL_zdrift_beg_end = (/ -1., -1. /)  ! beginning and end of actual
                                      !  drift section
!
!..WIGGLER Parameters:
!
  wavelw    = 0.08     ! wiggler wavelength (m)
  llinear   = .false.  ! wiggler polarization; .false. => helical
  lcnstwgl  = .true.   ! forces constant wiggler strength aw when true
  aw0       = 0.0      ! input constant RMS aw strength
  awdmult   = 1.00     ! dimensionless multiplier for aw
  lawaxis   = .false.  ! if true, ignore radial variation of a_w strength
  bw0       = 0.0      ! peak wiggler magnetic field (Tesla)
  nhar      = 1        ! harmonic number for linear wiggler
  nhar_mult = 1        ! harmonic # to multiply input particle phase

  idesign   = 1        ! wiggler self design flag
                       ! = 0    -->  input bwfile determines a_w
                       ! = 1    -->  code computes constant wiggler a_w
                       ! = 2    -->  self-design of tapered wiggler
  l_bw_linearfit = .false. ! switch for linear a_w interpolation in taper file

  rdesign   = -1.      ! design particle radius (cm) in taper ''self-design''
  rdesfac   = 0.707107 ! rdesign as a fraction of beam radius
  lincrdes  = .false.  ! switch for rdesign to increase adiabatically
  gammad0   = 0.       ! design particle initial gamma
  psir0     = 0.4      ! design particle resonant psi (radians)
  zstm_taper= -1.e6    ! z position (m) to start tapering
```

```
  z1psi     = 0.       !    :
  z2psi     = 0.       !    : more elaborate self design: see setaw
  psirinit  = 0.       !    :
  lphase10  = .false.  ! .t limits d psi / dz term from radiation d phi/dz
                       ! to being (relatively) <= 10% in wiggler self-design
  nside_sim =  1       ! osc. slice chosen to design taper
!
!..Simulation dimensions, random seeds, restart switches
!
  zmxmeter  =  -1.     ! run length in meters
  zmaxsim   =   2.     ! run length in Rayleigh ranges
  zstmeter  =   0.     ! zstart in meters (for restart)
  zstart    =   0.     ! zstart in in Rayleigh ranges
  rmaxsim   =   3.     ! max radius (cm) in simulation radial grid
  rlinear   =  -1.     ! radius to which radial grid is roughly linear
  nnd       =  40      ! number of radial grid points

  iseed     =  0 0     ! random number seed for radiation field fluctuations
  iseedp    =  0 0     ! random number seed for d gamma energy spread loader
  iseeds    =  0 0     ! random number seed for shot noise fluctuations

  l_write_rst  = .false.   ! flag to write a restart particle file
  l_read_rst   = .false.   ! flag to read a restart particle file

  l_trackcode  = .false.   ! flag to read and use input phase space info
  trackparfile = 'NOT SET' ! name of file containing input phase space
  tracksumfile = 'NOT_SET' ! file containing ELEGANT-derived time-dependent
                           ! envelope info, lookup table for particles, etc.

  l_write_fld  = .false.   ! flag to write E-field at wiggler end

  l_write_fld_z= .false.   ! flag to write z-dependent E-field of head e- slice
                           ! for purposes of a later, pulse-extending restart
                           ! or single slice FRED-mode run

  l_read_fld_z = .false.   ! flag to input E-field along wiggler from
                           !  previous polychromatic run
  lrestart     = .false.   ! ** DEPRECATED** switch for restart files
!
!..Numerical Integrator Control Parameters:
```

```
!
  l_debug  = .false.   ! switch to turn on debugging diagnostics mode
  mstep     = 250      ! step interval DRIVIB for integrator diags
  maxstep   = 10000    ! maximum number of steps in trying to converge
  eps       = 1.e-5    ! max permitted weighted error for Gear scheme integrator
  em8       = 0.04     ! set to 0.1 for 1D runs
  dt        = 1.e-8    ! initial step size  -----  Rayleigh lengths
  hlb       = 1.e-11   ! lower bound on step size    "        "
  hub       =  -1.     ! upper bound on step size    "        "
  hub_m     =  -1.     ! upper bound on step size   in >> meters <<
  ncpu      =   1      ! number of CPUS to be used (SMP architecture)
                       ! note: for MPP runs, set # CPU's on execute line
!
!..Space-Charge and Waveguide Parameters:
!
  xwidth   = 9.834    ! full width of waveguide parallel to E-field (cm)
  ywidth   = 2.908    !  "                  "   in direction of variation
  lwavegd  = .false.  ! flag for waveguide(.t) or free space(.f)
  ampmode  = 0.       ! init. rel. amplitude of TEM00-TEM05 modes
  ampmode(1) = 1.0
  te0m     = 0.0      ! fraction of input laser power not in TE01 mode
  mode     = 1        ! mode number of te0m power dependent on lte21
  phite0m  = 0.0      ! phase of TE0m mode in waveguide
  lte21    = .false.  ! switch to signal TE21,TE41,TE61, etc. modes
  lspacech = .false.  ! switch to include longitudinal space charge
!
!..Polychromatic, Multislice Input Parameters:
!
  lfred      = .false.  ! switch to run in "FRED" (monochromatic) mode
  l_segment_mode  = .false.  ! switch for non-periodic, "segment" run mode
  wavelsin   = -1.      ! wavelength of injected signal

  nside      = -1       ! # electron beam slices
  nsidep     = -1       ! # e-slices overtaken in wiggler by each photon slice
  nphoton    = -1       ! total # photon slices >= nside + nsidep ---
                        !    needed for non-periodic BC (ltransit=.t)
  dt_slice   = -1.      ! temporal spacing between slices
  window     = -1.      ! window width (m) for periodic BC in time
  del_wavels = -1.      ! wavelength resolution in meters
```

```
  wattpghz   =  0.      ! amplitude of broadband flat noise in watts/GHz
  wattpbin   =  0.      ! amplitude of broadband flat noise in watts/bin
  ampside    =  0.      ! amplitude of initial sideband amp. rel. to fund
  pwrnoise   =  1.      ! POWER amplitude of shot noise/theoretical value
  field_noise=  0.      ! element 1 = fractional RMS amplitude fluctuation
                        ! element 2 = RMS phase jitter (radians)
  nfreqbin   =  0       ! #freq. or wavelength bins containing laser input
                        ! if = 0, just 1 bin; if < 0, set to nside

  ampchirp   =  0.      ! radiation wavelength chirp fraction (of bandwidth)
  gamchirp   =  0.      ! chirp in avg. macroparticle gamma from
                        !   pulse TAIL to pulse HEAD
  chirp_type = 'sinusoid' ! type of chirp (in gamma or wavelength)
  sidefreq   = -1.      ! freq. of initial sideband excitation (Hz)
  sidewave   = -1.      ! wavelength of "             " (meters)
!
!..Short Pulse Mode (non-periodic BC) Input Parameters:
!
  ltransit    = .false. ! switch to put in non-periodic BC on rad. field
  pulse_shape = 'tophat'! longitudinal e-beam pulse shape
  trise       = -1.     ! current rise and fall time in sec
  tbody       = -1.     ! main current body duration in sec.: RMS pulse width
                        ! for gaussian profile; separation between 1/e points
                        ! for tanh profile
  laser_shape = 'tophat'! longitudinal pulse shape of laser
  laser_body  =  -1.    ! laser RMS pulse duration (sec) for
                        ! gaussian profile, basewidth for tanh,parabolic profiles
  laser_rise  =  -1.    ! rise, falltime in tanh laser profiles
  laser_timing_rel = 0. ! laser timing relative to e-beam pulse center
!
!..Oscillator and Optical Klystron Parameters:
!
  l_extra    = .false.  ! switch to read "inextra" input namelist
  losc       = .false.  ! switch to tell code if an oscillator run
  npass      =  1       ! # passes in oscillator
  npass_out  =  1       ! interval in pass # for writing output diags
  r_cavity   =  1.      ! reflection coefficient for loss in cavity
  d_synch    =  0.      ! relative (to slippage) cavity detuning length
  z_m1_m2    = 12.5     ! cavity length in meters
  rad_mr     =  0.01    ! radius of mirrors in meters
```

```
  rc_m1      =  7.5     ! radius of curvature of first mirror in meters
  rc_m2      =  7.5     ! radius of curvature of second mirror in meters

  fldfac      = -1.     ! factor to multiply input radiation field power
  del_j1strt  =  0      ! offset to move photons in restart

  nu0_colson =  0.      ! norm. offset of particle energy a la Colson
  d_colson   =  0.      ! OK drift dispersion parameter norm. to undulator length
  R56        =  0.      ! dispersion parameter in meters
  OptKly_zlen = 0.5     ! physical length (m) of OK drift for opt. propag.
  OptKly_zwgl_equiv= -1.! dispersion in terms of equivalent wiggler length (m)
  t_OKdrift  =  -1.     ! dispersive length in Rayleigh ranges
  zslip_equiv_OptKly =-1. ! equivalent undulator length (m) corresponding to
                          ! slippage produced by dispersive section

  z_OptKly   = -1.      ! location (m) of drift section for OK
  t_OptKly   = -1.      ! location of drift section for OK in Rayl.
  theta_drift = 0.      ! distance of drift space in radians
!
!..Multislice, FRED-Mode Parameter-Scanning Variables:
!
  param_name  = 'NOT SET' ! name of physical parameter to vary in scan
  l_log_param = .false.   ! flag to make geometrically-increasing
                          ! (vs. linearly-increasing) set on parameter values
  param_min   = 1.        ! initial (and lowest) parameter set value
  param_max   = 0.        ! final (and highest) parameter set value
!
!..Diagnostic Output Control Variables
!
  ncurve         = 50   ! number-1 of z locations in pltfile for FRED mode
  n_diag_mod     =  1   ! slippage-advance z-interval for output to pltfile

  scalar_diags   = ' '  ! string variable to force output of various diagnostics
                        !   in pltfile
  n_scalar_out   =  0   ! variable in postproc namelist which keeps
                        ! track of number of special scalars in pltfile

  nhar_io        = -1   ! bunching harmonic numbers to include in the
  nhar_io(1)     =  3   !   output pltfile (1st and 3rd by default)
```

```
nspec          =  0   ! total number of macroparticle scatterplot z-locations
z_scatterplot  = -1.  ! pre-set z-locations for scatterplots
nz_scatterplot = -1   ! # equally-spaced z-locations for scatterplots
dz_scatterplot = -1.  ! z-interval for outputting scatterplots
nt_scatterplot = -1   ! number of e-beam slices in scatterplots
nt_scatterplot_mod = -1 ! e-beam slice index frequency of scatterplots
spcfile_fmt = 'BINARY'  ! format of macroparticle scatterplot dump files

l_datfile  = .false.  ! flag in FRED mode to write simple ASCII output file
                      !  with variables such as radiation P, bunching, etc.
```

4 "Preferences" File for the *XPLOTGIN* Post-Processor

4.1 General Information

The GINGER postprocessor XPLOTGIN employs an *optional* ASCII "preferences" file to set various plotting options, including producing output in forms (*i.e.* ASCII, SDDS, HDF datafiles) other than graphical. The default name for the preferences file is **`xplotgin.pref`**. Beginning in fall 2001, however, alternate files may be specified by typing the XPLOTGIN execute line option **`pref=''mypfile''`** where **`''mypfile''`** represents the wanted preferences file to be read by XPLOTGIN. The preferences file must exist (in actuality, or via a hard or "soft" UNIX link) in the *same* directory from which XPLOTGIN is being run. The preference file should contain a Fortran90 namelist **`&inpref`** whose variables can control much of the behavior of XPLOTGIN. Some options have effects on all types of GINGER runs, others only for polychromatic runs, and a few apply only to monochromatic (FRED-mode) runs. Recent work on preference-file namelist options has concentrated extending the output data file generation capability to allow users to use other visualization tools (*e.g.* **`gnuplot`**, **`sddsplot`**) to analyze and display GINGER simulation data, and to control various details of the diagnostic plots.

The user should attempt to always use the most recent version of the postprocessor (which also avoids possible compatibility issues when using a very recent version of GINGER). Beginning in mid-2003, the GINGER *pltfile* contains information indicating the "minimum" version of XPLOTGIN necessary for proper analysis. If an earlier version of XPLOTGIN is used, an error message will be sent to the user's terminal window.

4.2 Color, Logo, and Graphical Output Suppression Control Variables

Setting **`l_color=.false`** creates monochrome (B&W) graphical output; one might use this option when needing Postcript output for a scientific paper and/or for later conversion to PDF format. Similarly **`l_logo=.false.`** prevents the "logo" text (which contains the GINGER run name, date, and machine name) from being placed in the upper right-hand corner of the individual plot frames. To minimize the total number of plots generated, keep **`l_fullplots=.f`**, the normal default. This will skip various plots most users generally do not use, *i.e.* 3-D contour plots of $I(r,z)$, $\phi(r,z)$, and obscure plots like the synchrotron wavenumber, e-beam power loss, *etc.* . Many of these plots can be turned on individually via other preference file input variables.

To *completely* suppress the creation of the output graphics file (be it X11, CGM or Postscript output), set **`lnoplot=.true.`**. One normally does this only if, rather than graphical output, you want XPLOTGIN to create only (via the **`l_datfile`**, **`l_sdds_output`** or **`l_hdf`** switches) ASCII (§4.7), SDDS (§4.8), or HDF (§4.10) file output.

4.3 Data File Input Read Control Variables

One seldom needs to use any of these variables as they were developed mainly to deal with "faulty" *pltfiles*, *e.g.* one generated by a GINGER run that did not finish to completion. For example, if an oscillator run did not reach the final pass specified by the value **npass** in the original GINGER input file (and which is echoed in the beginning of the *pltfile*), the preferences variable **npass_i** will override this value. Similarly, **nsidep_i** and **nphoton_i** override the *pltfile* values of **nsidep** and **nphoton**, respectively. However, it remains somewhat hit-or-miss whether the latter two variables will allow the postprocessor to analyze a faulty *pltfile* properly. The variable **l_read_3rd_har=.f** instructs XPLOTGIN not to to read 3rd harmonic bunching data. This capability exists to ensure compatibility with quite old *pltfiles* which did not write such bunching data; most users will never use it.

4.4 Radiation Power and Bunching Plot Control Variables

Setting **l_plot_engbal=.t** generates plots of total energy balance versus z. To force the ordinate scale range for the microbunching snapshots to be [0.0, 1.0] rather than zero to the local (in z) maximum, set **l_bunch_max=.t**. One may also choose a ordinate maximum other than 1.0 by giving a positive value for the variable **bunch_max**. Setting **l_plot_3rd_har=.f** suppresses plots of 3rd (and higher if they exist) harmonic bunching fraction. To suppress plots of the instantaneous energy spread, set the switch **l_plot_delta_gamma=.f**. To set the specific plot locations in z for the bunching and radiation power snapshots versus t, use the real 1D array **zsnap_plot**; the "deprecated" variable **z_snap** previously controlled these locations. Presently, up to 9 values are allowed with default locations being evenly spaced in z through the GINGER run. One can change the variable **ifirst_testp** from its default value of **1** to specify the index of the first snapshot plot z location; this is generally appropriate for shot-noise-initiated runs in which the initial radiation field is essentially zero (the preferences switch **l_sase=.t** also does this).

To generate radial profile plots of the radiation intensity, set **l_plot_radial_profile=.t**. Currently, these plots will be at the same $z-$locations as the power spectrum plots (see the next section). By default, these intensity profile plots are accompanied by plots of the "slowly-varying" (*i.e.* eikonal) radiation phase $\phi(r)$. The latter can be separately turned off by setting **l_plot_phase_r=.f**. To generate contour plots of the intensity, set **l_3Dplots=.t**. Section 4.9 discusses some additional preference file variables which control generation of SDDS files with intensity and electric field information.

The switch **l_plot_farfield** controls the calculation of and plotting of the far field radiation pattern. As of October 2001, it has a new default value of **.false.**.

4.5 Spectrum Plot Control Variables

The switch **l_plot_spec**, normally true, can be used to suppress all plots involving spectra. A number of variables can modify both the number and $z-$locations of the power spectrum plots $P(\lambda)$. Normally there are 5 such plots evenly spaced in z; the preferences variable **n_spec_plot** can override this. Alternatively, the real 1D array **zspec_plot** can override even spacing and set the actual z-locations. Up to 16 values may be input but remember there are only **nsidep+1** available in the simulation. The equivalent array to control z locations in ASCII/SDDS output files is **zspec_print**. The real array **zgain_plot** will set z-locations at which the "instantaneous" gain (db/m) is plotted versus wavelength. By default such gain is averaged over a $z-$interval of one-tenth the simulation length; this averaging length can be changed by giving a value in meters to the preferences variable **dz_gain**.

Normally, the power spectrum plots are produced in a semi-log format. This can be changed to a linear scale by setting **l_linear_specplot=.t**. The variable **spec_pwr_plot_limit** can set the range in watts of the spectral plot ordinate. The default in semi-log format is to allow up to 8-orders of magnitude in the range with the max and min values being exact powers of ten. To change the wavelength interval in the power spectrum plots from the default value of the full simulation bandpass, give the wanted spectral range in meters to the two-element array **wspec_plot_limit**. Setting the variable **n_avg_spec > 1** instructs the postprocessor to average the spectral intensity over multiple neighboring frequency bins for smoothing purposes.

Beginning in fall 2001, XPLOTGIN now has the capability to generate power spectrum plots of the calculated electron beam microbunching in the longitudinal direction. These spectra can be compared with experimental measurements of coherent transition radiation. They will be generated if either **l_plot_bunchspec=.t** (in which case they will be plotted at the same locations as the radiation spectra) or if the user gives $z-$location values for the real array **zbunchspec_plot**.

For time-dependent runs starting from noise, setting **l_sase=.t** will start many plots versus z at the *second* output z-location where true incoherent noise effects should dominate over the initial (coherent) radiation power (which the user presumably set to a very low value). This switch also suppresses plots of the upper and lower sideband growth versus z. One should not use this switch in a MOPA configuration where there is a strong input source. When **l_plot_fund =.f**, the power at the fundamental (*i.e.* central) wavelength is "suppressed" in spectrum plots; this option is useful if one wants to examine sideband development in a single pass amplifier with a strong input signal strongly dominating the sideband power.

To generate plots of the temporal autocorrelation function for the radiation electric field at various z-locations, set **l_autocorr=.t**. As of spring 2001, the on-axis, far field complex amplitude is used in the autocorrelation calculation. The number of such plots is controlled by **nplt_autocorr** (note: incorrectly given as "nplt_corr" in previous versons of the manual) with a default of value of 5; Alternatively, the user use the array **zautocorr_plot**. to input the z-

locations of the autocorrelation plots. Similar plots can be obtained for the microbunching autocorrelation function at the fundamental AND harmonics wavelengths previously specified by **nhar_io** in the GINGER run by specifying the individual harmonics wanted in the preference file namelist variable **nhar_bunch_corr**. For example, if **nhar_io = 1 3 5 7 9** and **nhar_bunch_corr = 1 3 4 7**, microbunching correlation plots will be produced at harmonics 1, 3, and 7 (but not 4 because it was not specified in the GINGER run).

4.6 Generating Plots of Time-Resolved Phase

XPLOTGIN now has a new diagnostic to generate various plots associated with the instantaneous phase $\theta(t)$ and its temporal derivative. For the radiation field, the phase refers to that of the on-axis far field. For the electron beam microbunching, it refers to the phase of the complex bunching $b(t)$. The instaneous derivatives are useful to indicate the presence of "chirps" in the instantaneous frequency $\omega(t)$ (*i.e.* a $\partial\theta/\partial t$ which is linear indicates a constant chirp). The generation of these plots is controlled by giving appropriate numerical values to one of the following preference file namelist variables: **zphase_t_plot** or **nz_phase_t_plot**. Alternatively, setting **l_plot_phase_t =.t** will set **nz_phase_t_plot=4** which produces output at 4 evenly-spaced locations in z. Currently, the instantaneous derivative plot is normalized to the equivalent change in phase over one wavelength; *i.e.* $\dot{\theta}_{plot} \equiv (\lambda_s/c) \times d\theta/dt$. The normalized values of the radiation electric field and microbunching are also displayed on the bottom of the plots — this is particularly useful in short-pulse mode to determine chirps over a short e-beam or radiation pulse. To reduces sensitivity to noise fluctuations where either b or E is small, the preference variables **dphase_dt_bun_floor** and **dphase_dt_pwr_floor** can be used to zero out the derivatives of the normalized $b(t)$ and $P(t)$, respectively, drop below the input values. Presently, the default values for the "floors" are set to 0.04 and 4.E-4 for non-SASE runs and 0.1 and 0.025 for SASE runs). Finally, if **l_plot_dphase_axis=.t**, a plot of the RMS value of the nonlinear (in time) component of $\theta(t)$ is plotted versus z.

4.7 Generating ASCII Output Tabular Data Files

Several users have requested the ability to output simple ASCII files of variables such as $< P(z) >$ for input to other visualization and/or analysis codes. This capability has been significantly extended in XPLOTGIN over the last few years. To enable either simple tabular output (or SDDS-formatted file generation as described in §4.8), set the variable **l_datfile = .t** in the preferences file.

Non-SDDS output files are given names that begin with the GINGER run_name followed by a mnemonic string indicating the physics type of data included therein. For example, if the GINGER ***run_name*** is **palac**, the files **palac_data.zhist, palac_data.time,**

palac_data.spec will be output. At present depending upon the type of GINGER run, 3 types of simple ASCII tabular files can be generated:

'*_data.zhist' - this file contains separate columns of the output z-location, time-averaged radiation power, bunching at both the fundamental FEL wavelength and the third harmonic, rms radiation beam size, far field radiation mode size, and (in polychromatic mode) inverse rms spectral bandwidth.

'*_data.time' - this file contains time-resolved data at the wiggler end and is thus only generated for GINGER runs done in polychromatic mode or monochromatic, parameter scanning FRED-mode, (see section 3.11) where the scanning variable (*e.g.* a_w) replaces time as the independent variable. The first column contains the independent variable value, followed by separate columns for radiation power, electron beam power loss, and bunching at the fundamental FEL wavelength.

'*_data.spec' - this file contains wavelength-resolved radiation power spectra data at different individual $z-$locations and will be generated only for polychromatic GINGER runs. The locations can be set by the preferences variable array **zspec_print**, which may contain up to 8 values. If **zspec_print** is not input, there will be only one location, at the wiggler exit. The first column will contain the wavelength while the successive columns contain the power per wavelength bin at the different z values.

4.8 Generating SDDS Format Output Files

The Self-Describing Data System (SDDS) is a data format developed by M. Borland and is presently in wide use at the APS facility at Argonne National Laboratory. An important and attractive feature of the SDDS package is the existence of numerous processing and plotting tools which can be used directly to analyze data written in SDDS format; see the URL **http://www.aps.anl.gov/asd/oag/oaghome.shtml** and associated links for documentation and download information concerning SDDS software. These tools exist for major UNIX platforms and Windows. In late 2000, SDDS file generation capabilites were first implemented into XPLOTGIN and have been significantly extended since. When *both* **l_datfile=.t** and **l_sdds_output=.t** in the preference file namelist, output data of the type described in section 4.7 will be written into SDDS files. The SDDS output files are in ASCII and contain information essentially identical to that stored in ASCII format. Each will have a suffix of ***.sdds*** (*i.e.* **palac-time.sdds**, **palac-spec.sdds**, **palac-zhist.sdds**). When **l_sdds_output=.t**, *only* SDDS format files (and *not* the ASCII-style files described in the previous section) will be output.

4.9 Generating "Special Purpose" SDDS Output Files

In addition to the "general" SDDS files mentioned in the previous section, one can also generate "special"-purpose SDDS output files. For these files, it is not necessary that **l_sdds_output=.t**. Setting the input variable **pwr_lambda_print** to an array of radiation wavelengths λ_i (units in meters) will cause the generation of an SDDS file with the name ***run_name*-pwr_lambda.sdds** containing separate columns of $P(\lambda_i)$ versus z. If **l_write_inten_RZ =.t**, an SDDS output file containing the time-averaged values of intensity $I(r,z)$ will be created. When **zefld_print** is set equal to an array z_i of z locations, the slowly-varying, complex, time-dependent electric field $\tilde{E}(r,t)$ (see §5.1) will be written out as a series of arrays to a file named ***run_name*-efld.sdds**. A final option **l_write_fld_z** controls generation of single slice radiation fields and is described in §4.12.

4.10 Generating HDF Output Data Files

Hierarchical Data Format (HDF) is a reasonably widespread format used to create portable data files. The HDF system originated at and is currently supported by NCSA in Illinois. When **l_hdf=.t**, XPLOTGIN will write the following 3D data sets in HDF format: instantaneous $P(z,t)$ and normalized ($\equiv P(z,t)/ < P(z) >$) radiation power, instantaneous $b(z,t)$ and normalized e-beam bunching data, and instantaneous $P(\lambda,z)$ and normalized radiation power spectra ($\equiv P(\lambda,z)/ < P(z) >$). Each of the aforementioned power diagnostics has units of watts/bin. HDF files may be generated in addition to either ASCII- or SDDS-format data files mentioned in the previous two sections. The postprocessor HDF capability is targeted toward version 3.4 of the library and is thus VERY seriously out of date as of early 2004!

4.11 Generating Wiggler Exit, Radiation Field Dump Files

For various reasons it can be useful to generate an output file containing the time-dependent radiation fields at the wiggler exit. There are two methods to do this at present. For downstream diagnostic (and *not* restart) purposes, **l_write_outfld=.t** will generate an ASCII file containing the complex $\hat{E}(r,t)$ named ***run_name*.fld** (*e.g.* **palac.fld**). The file will also contain locations of the radial grid points and a normalization value for the electric fields (GINGER internally scales electric field values). One can use this file to propagate the time-dependent radiation field to the far field and/or an intermediate position. One can override the default $z-$location of the wiggler exit by also inputting a value in meters for the parameter **z_write_outfld**.

A second possibility is applicable to oscillator mode runs only and was developed to allow one to extend the number of passes from one GINGER run to the next. If **l_write_oscfld=.t**, the postprocessor dumps the radiation electric field information at the wiggler exit for oscillator

pass number **npass_w**. If not input in the preferences file, **npass_w** defaults to **npass**. This should work both in FRED- and polychromatic mode. The format at present is archaic, possibly buggy, and relevant *only* to oscillator runs and should not be used to create a *fldfile* for a multi-stage radiator/modulator configuration.

4.12 Generating z-dependent, Single-Slice, Radiation Field Dump Files

Knowledge of the z-dependent radiation field advancing (due to slippage) beyond a given electron-beam slice permits one to propagate a subsequent (actually preceding in time) slice. As explained in §3.4.5, one can use the field information from a polychromatic run to initialize a single-slice Fred-mode run with detailed output macroparticle diagnostics. To create such a *field_file* from a *pltfile*, in the preferences file namelist one should set the input variable **l_write_fld_z=.t**. If output from other than the "head" electron beam slice is wanted, the variable **nslice_fld_z** should be set to an integer value in the interval 1 to **nside** with 1 corresponding to the tail slice and **nside** to the head slice. Note, if the *pltfile* had **n_diag_mod**$\neq 1$, this feature will not work properly in a restart run (due to missing z-location field information). The output field file will be in SDDS format and named ***run_name*-fld_z.sdds** (*e.g.* **palac-fld_z.sdds**).

4.13 Macroparticle Phase Space Plot Control Variables

At present, macroparticle scatterplots are only available only for single-slice FRED-mode runs whose GINGER input file had **nspec** $\geq$ **1**. The scatterplots will automatically be produced by XPLOTGIN unless the variable **l_scatterplots=.f**. The 2-element array **gamma_limit** specifies the lower and upper limits in γ for the $\gamma - \theta$ scatterplots — this option is useful if one is attempting to generate a movie. **l_plotMeV = .t** sets the ordinate scale in the $\gamma - \theta$ plots to be megavolts rather than the default of Lorentz factor. **l_plot_gammap=.t** substitutes the computed variable $\gamma_{\|} \equiv (1 - \beta_z^2)^{1/2}$ for γ in the scatterplots. This option can be useful when examining the performance of a highly tapered wiggler. Setting **l_plot_xy=.t** will generate $x - y$, $x - p_x$, *etc.* scatterplots, permitting the user to check the $z-$dependent evolution of the e-beam transverse profile in the focusing lattice.

If color output is produced (the default), by default the macroparticles are color coded by their longitudinal phase in the $z-$location corresponding to the first scatterplot. This coding remains fixed for all the following plots. By setting the namelist variable **var_colorsort** to **'gamma'** or **'r'**, the user can set the coding to be by particle energy or radial position, respectively.

4.14 Default Values for Preference File Namelist Variables

In general, logical variables in the preference file namelist are normally of the form `l_`*name* where *name* represents some mnemonic for a plot function or whatever, integer variables start with the letters `i-n`, and real variables start with the remaining letters of the alphabet. The names and default values for **`&inpref`** namelist variables as of post-processor source version **`01-DECEMBER-2003-a`** are as follows:

```
lnoplot     = .false.  !* switch to turn off output graphics; normally used
                       !  only when writing data files
l_logo      = .true.   !* put plot runID, date, time, etc. in upper right corner
l_color     = .true.   !* switch for color plots; .f --> b&w plots

l_fullplots = .false.  !* switch to produce "full" ensemble of plots
l_3Dplots   = .false.  !* switch to force 3D contour plot output

l_sase      = .false.  !* switch indicating growth from noise; this
                       !  starts many plots in z at iz=2 rather than 1
                       ! it also eliminates ‘‘sideband’’-related plots
l_plot_fund  = .true.  !* switch to plot fundamental wavelength power in
                       !  spectra;  set to .f when looking at sidebands
                       !  in a MOPA configuration

l_plot_engbal =.false. !* switch to control plotting of energy balance
l_plot_3rd_har=.false. !* switch to plot third harmonic bunching
l_read_3rd_har=.true.  !* switch to read 3rd harmonic

l_plot_delta_gamma    = .true.  !* switch for delta-gamma plots
l_plot_farfield       = .false. !* switch for plots of far field diags
l_plot_radial_profile = .false. !* switch for radial profile plots
l_plot_phase_r        = .false. !* switch for plotting radiation phase vs. r

l_plot_phase_t        = .false. !* switch to plot d(phase(t)/dt
zphase_t_plot         = -1.e6   !* z-locations for d(phase(t)/dt plots
nz_phase_t_plot       = -1      !* #evenly-spaced z-locations for phase plots
dphase_dt_pwr_floor   = -1.e6   !* ‘‘floor’’ value (rel to max) of pwr or
dphase_dt_bun_floor   = -1.e6   !* bunching to suppress d phi/dt plot values

l_plot_dphase_axis    = .false. !* switch for plotting RMS far field phase noise
```

```
zgain_plot            = -1.e6   !* z-locations to plot instantaneous gain
gain_logfrac_limit = (/0.25, 0.75/) !* lower & upper fraction limits
                                  !  of the z-interval in log(Power)
                                  !  to calculate gain length


!** the following control (normally 3x3) snapshots of
!     intensity, bunching, and delta-gamma vs time:
zsnap_plot   = -1.     !* z locations (m) of snapshots
iz_snap      = -1      !* # of z locations of snapshots (must lay in
                       !  range from 1 to nsidep)
nz_snap       = -1     !* number of snapshots (equal intervals in z)
ifirst_testp = 1       !* location index of 1st time-dependent snapshot
                       !    (overridden by zsnap_plot or iz_snap)

l_bunch_max  = .true.  !* plot microbunching on interval [0.0, 1.0]
bunch_max    = -1.     !* when >0, defines max ordinate on bunching snapshots

!** the following control locations of spectral plots:
l_plot_spec  = .true.  !* switch to plot intensity vs. wavelength spectra
zspec_plot   = -1.     !* z positions at which spectra desired
n_spec_plot  = -1      !* #z positions (uniformly sep.) for spectra plots
n_avg_spec   =  1      !* # freq pts to smooth power spectra
zspec_print  = -1.     !* z positions (m) to print radiation spectra
wspec_plot_limit = 0.  !* wavelength (or freq) limits on spectrum plots
spec_pwr_plot_limit = -1.e6 !* to set ordinate range on plots of
                            !  radiation power spectrum vs. wavelength
l_linear_specplot = .false. !* switch to have linear P(omega) plots
l_write_spec_WZ   = .false. !* switch to output P(omega,z)

l_plot_bunchspec = .false. !* switch to plot microbunching power spectra
zbunchspec_plot  = -1.e6   !* individual z-locations for microbunching spectra

pwr_lambda_print = -1.e6   !* wavelengths for which to print P(lambda) versus z
pwr_lambda_plot  = -1.e6   !* wavelengths for which to plot P(lambda) versus z
zefld_print      = -1.e6   !* z-locations to print complex E-field (r,t)

l_autocorr       = .false. !* switch to control plotting of autocorr.
zautocorr_plot   = -1.e6   !* z locations for autocorrelation plots
```

```
nplt_autocorr    = 5       !* # of z-locations for autocorrelation plots
nhar_bunch_corr  = -1      !* harmonic #'s to plot bunching autocorrelation


!** the following apply to single-slice FRED-mode,
!** macroparticle scatterplots only:
l_plot_xy    = .false. !* switch for x-y, x'-y' phase space plots
l_scatterplots =.true. !* if .false., suppress all scatter plots
l_plot_prof  = .false. !* switch for radial intensity profile plots
l_plotMeV    = .false. !* switch phase plot ordinate to be MeV, not gamma
gamma_limit(1)  = -1.  !* lower ordinate limit in theta-gamma scatterplots
gamma_limit(2)  =  0.  !* upper ordinate limit in    ''           ''
l_plot_gammap= .false. !* switch to plot gamma_parallel rather than
                       !  gamma in theta-gamma scatterplots
var_colorsort = 'NOT_SET' !* particle variable for scatterplot color coding
                       !* allowable values: "gamma", "phase" (default), "r"
                       !* coding done at first z-location


!** following control generation of output data files:
l_datfile    = .false. !* switch to write out ASCII data files containing
                       !  laser power, bunching, output spectrum, etc.

l_sdds_output= .false. !* switch to write data files in SDDS format
l_hdf        = .false. !* switch to write output HDF files containing
                       !  intensity, bunching data
l_write_fld_z =.false. !* switch to write indiv. slice rad. field vs. z
nslice_fld_z  = -1     !* e-beam slice for radiation field vs. z
l_write_inten_RZ =.false.  !* switch to write time-averaged
                              radiation intensity(r,z)

l_write_outfld=.false. !* switch to write ASCII file containing
                       !  time-dependent radiation field at wiggler output
z_write_outfld= -1.e6  !* z-location to write radiation field file
                       !   (default location == wiggler exit)
l_write_fld  = .false. !* switch to write file with output electric field
                       !  for restart purposes (osc. mode only)
npass_w      = -1      !* pass number to write fld info if l_write_fld=.t

!** the following preference variables are used to override values in a
```

```
!   GINGER-produced pltfile; they are only needed if the GINGER run
!   did not properly finish for some reason; >>> use with extreme caution! <<<

nphoton_i    = -1      !* value to override pltfile value of nphoton
nsidep_i     = -1      !* value to override pltfile value of nsidep
npass_i      = -1      !* value to override pltfile value of npass
```

5 The Physics Model of GINGER

GINGER uses various approximations, assumptions and physics models when simulating the behavior of free-electron lasers. This section discusses these issues and some others related to numerical accuracy.

5.1 Application of the Paraxial Wave Equation

GINGER adopts the slowly-varying envelope approximation (SVEA), which is also widely known as either the paraxial wave equation or the eikonal approximation. GINGER applies this approximation by separating the local radiation electric field $E(r,z,t)$ into a product of a "fast" modulation $\exp\left[i(k_s z - \omega_o t)\right]$ times a "slow" envelope modulation $\tilde{E}(r,z,t)$, whose r and z spatial derivative scale lengths are much greater than λ_s and whose time derivative is much slower than ω_o.

The slow modulation $\tilde{E}$ is a complex quantity whose spatial and temporal phase $\phi(r,z,t)$ is measured relative to that of hypothetical plane wave whose phase varies exactly as $(k_s z - \omega_o t)$. In GINGER, an individual macroparticle's longitudinal phase θ is also measured relative to this hypothetical plane wave. The "FEL" ponderomotive phase ψ of a macroparticle is then $\psi \equiv \theta + \phi$. Effects of diffraction, gain, and refraction all lead to ϕ becoming non-zero, while a non-resonant macroparticle energy leads to a monotonic increase or decrease of θ with z.

Neglecting the second $z-$ and $t-$ derivatives of $\tilde{E}$ relative to $-k_s^2\tilde{E}$ and $-\omega_o^2\tilde{E}$ respectively, the full wave equation may be approximated as

$$2i\left(k_s\frac{\partial\tilde{E}}{\partial z} + \frac{\omega_o}{c^2}\frac{\partial\tilde{E}}{\partial t}\right) + \nabla_\perp^2\tilde{E} = -\frac{4\pi i}{c^2}\,\omega_o\,\tilde{J}_\perp + \left(\frac{\omega_o^2}{c^2} - k_s^2\right)\tilde{E}$$

Here $\tilde{J}_\perp$ is the bunched component of the transverse e-beam current and, for non-waveguide cases, $\omega_o = ck_s$ and $\partial\omega/\partial k \equiv c$. The above equation may be rearranged to produce

$$\frac{1}{c}\frac{\partial\tilde{E}}{\partial T} = \frac{i}{2k_s}\nabla_\perp^2\tilde{E} - \frac{2\pi}{c}\tilde{J}_\perp$$

The operator $c^{-1}\,\partial/\partial T$ represents the co-moving derivative in the forward direction in the wiggler.

For nearly all "normal" FEL problems, the scaling $\nabla_\perp^2,\ \partial^2/\partial z^2 \ll k_s^2$ is well satisfied since Rayleigh ranges and gain lengths are much longer than λ_s. The temporal part of the SVEA, which asserts $|\partial^2\tilde{E}/\partial t^2| \ll \omega_o^2\,\tilde{E}$, is similarly justified for FEL amplifiers whose gain bandpass is small compared to ω_o. The approximation begins to fails if one is interested in a device whose spectral width is comparable to ω_o, as might be true if one is examining incoherent spontaneous emission over an extremely broad bandpass.

5.2 Application of the KMR Equations

In order to determine the value of the bunched transverse electron current and to advance the macroparticles's energies and longitudinal phases, GINGER adopts the Kroll-Morton-Rosenbluth (KMR) (*IEEE J. Quantum Elec.*, **QE-17**, pp. 1436-1468, 1981) wiggle-period-averaging approximation. In this approximation, details of the so-called quiver (or "figure 8") motion and the $v_\perp \cdot \tilde{E}_\perp$ wave-macroparticle interaction are averaged over one or more wiggler wavelengths λ_w in z. The latter averaging requires that γ, a_w, λ_w, and $\tilde{E}_\perp$ be constant or slowly varying over a wiggler wavelength. Furthermore, FEL radiation is presumed to be emitted *exactly* in the forward direction. Nearly all FEL's have gain lengths and Rayleigh ranges much longer than λ_w and have wiggler errors small compared to a locally averaged a_w. Hence, the KMR approximation is expected to be excellent. Numerically, GINGER evaluates values for macroparticles, the radiation field, and the wiggler field locally in space as opposed, for example, of trying to use some averaged value over an interval $\Delta z = \lambda_w$. Note that the KMR equations generally assert a much slower variation in z than does the SVEA for the radiation field (although in the *e-beam* frame the approximations are equivalent).

5.3 GINGER's Transverse Macroparticle Mover

GINGER's transverse particle mover is that of its monochromatic predecessor FRED and is fully three-dimensional and relativistic. The mover uses a 4th-order Runge-Kutta algorithm and follows the wiggler-period-averaged betatron motion only, and not the "fast" wiggle motion (as is appropriate for GINGER's adoption of the KMR equations). At present, the particles are advanced at the same z-locations as the radiation field, which typically involves a step size generally smaller than λ_w which itself is much smaller than the betatron wavelength in either plane. Hence, the numerical error in betatron motion is expected to be extremely small. As of late 2001, transverse drift terms due to wiggler errors can be included via *lattice_file*'s generated by the XWIGERR program (see §3.5.9). Longitudinal space charge forces are also evaluated over each field advance and are applied within the KMR γ advance of the individual macroparticles.

5.4 Temporal Structure of GINGER

Following the approach pioneered by the work of W. Colson and D. Quimby, GINGER models the "slow" temporal modulation $\tilde{E}$ of the "complete" transverse electromagnetic field and particle transverse current source terms by resolving the radiation field and the particle beam into discrete, equally spaced transverse slices. In GINGER, the temporal separations between adjoining slices is normally many times λ_s/c, especially for FEL's whose Pierce parameter $\rho \ll 10^{-2}$. Thus, the macroparticles and radiation field which belong to a given transverse slice should be thought of

those of a typical, "sample" ponderomotive well in the temporal center of a the slice. Following this ansatz, dynamically, when a given macroparticle's longitudinal phase crosses one of the ponderomotive well boundaries $\theta = \pm\pi$, the particle instantly reappears at the other boundary (as opposed to jumping into a well of an adjacent slice). This treatment should work well for spectral bandpasses small compared to the central wavelength λ_s (*i.e.* each slice is many λ_s/c wide temporally) but will break down as full bandpass $\Delta\lambda$ becomes comparable to the central wavelength λ_s.

Although some workers conceive of the electron beam and radiation slices in a polychromatic FEL code as being separated in space (as in a hypothetical snapshot in time), the correct picture (for GINGER at least) is a separation in time (as in an oscilloscope trace). Otherwise, one could not properly model very high gain FEL's whose gain length is less than the equivalent length of the electron beam modeled in the code. Within GINGER itself, only the temporal modulation of the field is followed; decomposition into frequency components is done only by the postprocessor which uses FFT's to determine the frequency content of the electromagnetic field at any position in (r, z).

5.5 Discrete Slippage Model

In optical FEL's, light moves forward with a phase and group velocity c whereas the electron beam particles move forward with a longitudinal velocity $v_\parallel \approx k_s/(k_s + k_w) < c$. This velocity difference introduces a "slippage" between the light and resonant electrons of one radiation wavelength λ_s for every wiggler period of propagation. In GINGER, this slippage is approximated in the following way: A given electron beam slice interacts with a single radiation slice for a discrete distance $\Delta z_{interact} \equiv L_w/$`nsidep`. At the end of this interaction distance, the given electron beam slice abruptly "falls back" in time to begin interacting with a new radiation slice (here time is measured back from the head of the radiation pulse).

Due to considerations of computer memory management on non-MPP platforms (a problem far more pressing in 1985 when GINGER was first written than it is today), the code actually follows a given slice of beam particles through the entire wiggler, slipping it back in time relative to the radiation slices at the appropriate discrete positions in z. With this choice, it was possible to structure the code such that one and only only one set of macroparticles need be in memory at any given instant. Apart from some initial disk storage of particle phase space coordinates both at the beginning and end of the simulation run necessary when periodic boundary conditions are used, no disk swapping of particle quantities is required during the main body of the run. When using multiple processors in SMP mode, there is some disk/memory swapping throughout the run. On MPP's when using multiple processors, *all* the particles are simultaneously in memory and all the slices are advanced simultaneously, lockstep in z.

As mentioned above, at any given position z the beam slice interacts with one and only *one* radiation slice. This ensures that information can move only from the front of the radiation beam

to the back and not visa versa (excepting the effects of the periodic boundary conditions). This is a noteworthy difference from some other polychromatic FEL codes where the field quantities are interpolated in time to the particle positions and, similarly, the particle source terms are interpolated to the field positions. This alternative choice, in principle at least, can allow some information to move unphysically (*i.e.* faster than c) from the tail of the radiation field toward the head.

5.6 Temporal/Frequency Window Duration and Resolution Considerations

There are two ways that one may then set the duration W of the simulation window in time: The first is to specify the equivalent length cW using the input parameter **window** in meters. The second and more common way is to specify the parameter **nsidep** which is defined as the number of photon slices that will slip over a given e-beam slice over the duration of the wiggler. One should also note that as **nsidep** approaches N_w, the polychromatic bandpass begins to include "slow" time variations whose equivalent frequencies are becoming an appreciable fraction of ω_o. This violates the neglect of the second time derivatives of $\tilde{E}$ which underlies the slowly-varying envelope approximation (SVEA) used in the field equations. Moreover, the KMR wiggle-period-averaged equations also require **nsidep** $\leq N_w$ to be true.

There are a number of physics issues to consider when attempting to choose "good" values of **nsidep** and **nphoton**. First, for a single pass amplifier, one wants the interaction distance

$$\Delta z_{interact} \equiv L_w/\texttt{nsidep} \equiv \lambda_w \times [\,\texttt{window}/(\texttt{nphoton} \times \lambda_s)\,]$$

between a given optical and e-beam slice to be a relatively small fraction (*e.g.* $\leq 25\%$) of the electric field gain length, which is approximately $\lambda_w/4\pi\rho$. Equivalently then,

$$\texttt{nsidep} \geq 16\pi\rho N_w$$

At equality, the full normalized frequency bandpass,

$$(\omega_{max} - \omega_{min})/\omega_o \equiv \texttt{nsidep}/N_w = 16\pi\rho$$

should be more than sufficient to enclose the normalized gain bandpass whose full width $\sim 2\rho$. Since ρ typically lies in the range 10^{-2} to 10^{-4}, **nsidep** is almost always significantly less than N_w. For cases beyond saturation where sidebands become important and/or where for one reason or another the peak of the gain curve has a frequency chirp with z or time, one must ensure **nsidep** is sufficiently large to include all frequencies of physics interest.

On the other hand, one probably wants reasonably good frequency resolution $\Delta\omega$ within the gain curve. Since

$$\Delta\omega/\omega_o = \lambda_s/\texttt{window} = \texttt{nsidep}/(N_w \times \texttt{nphoton})$$

choosing $\Delta\omega/\omega_o \leq \rho/2$ is equivalent to

$$\texttt{nphoton} \geq 32\pi\,[\,\texttt{nsidep}\,/\,(16\pi\rho N_w)\,]$$

Consequently, requiring a good z resolution of the field gain length together with reasonable frequency resolution of the gain bandpass will probably require $\texttt{nside} = \texttt{nphoton} \geq 96$. For a "standard" LCLS-like run where there are ~ 15 power gain lengths in the wiggler, one will generally pick $\texttt{nsidep} \geq 64$ and $\texttt{nphoton} = \texttt{nside} \geq 128$. The CPU run time of the simulation will increase linearly with **nside** but have a much slower than linear increase with increasing **nsidep**.

To summarize the above discussion, to increase the overall frequency bandpass, increase **nsidep**. For more frequency resolution within a given bandpass, increase **nphoton** (while keeping **nsidep** or **dt_slice** constant).

Another consideration which tends to make one increase both **nsidep** and **nside** is the accuracy of GINGER's discrete slippage algorithm for frequencies which lie a significant portion of the frequency bandpass away from the central frequency ω_o. For **nsidep** sufficiently large that $\Delta z_{interact} \leq L_{gain}/8$ (where L_{gain} is the power gain length), the frequency ω_{peak} corresponding to the peak of the exponential gain curve can *purposefully* (by the user) be offset from the central frequency ω_o of the simulation as much as $\pm 15\%$ of full simulation spectral bandpass without appreciable (*e.g.* $\geq 10\%$ increase in gain length) from unphysical numerical effects. The useful spectral bandpass increases linearly with **nsidep**. When modeling high gain SASE devices, one must be careful in one's choice of λ_s for a given a_w or *visa versa* if there is not good resolution in z of an exponential gain length (*i.e.* a relatively small **nsidep**). Otherwise, the predicted gain will be unphysically suppressed because the wavelength of peak gain becomes too close to the edges of the bandpass.