

Fortran Tools

THE FORTRAN COMPANY

www.fortran.com

Library of Congress Catalog Card Number

Copyright © 2005 by The Fortran Company. All rights reserved. Printed in the United States of America. Except as permitted under the United States Copyright Act of 1976, no part of this book may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system without the prior written permission of the authors and the publisher.

9 8 7 6 5 4 3 2 1

ISBN

The Fortran Company
6025 North Wilmot Road
Tucson, Arizona 85750 USA
www.fortran.com

Composition by The Fortran Company

Contents

1	Installation	1-1
1.1	Introduction	1-1
1.2	Implementations Provided	1-1
1.3	Contents	1-1
1.4	Installation	1-1
1.5	Support	1-8
1.6	What is Installed	1-9
1.7	Licenses	1-9
1.8	Source Code	1-10
1.9	Documentation	1-11
2	Command Line Compilation	2-1
2.1	Usage	2-1
2.2	Using g95 with the Fortran Tools	2-2
2.3	Description	2-3
2.4	Some Options	2-3
2.5	Pre-connected Input/Output Information	2-5
2.6	Error Messages	2-5
3	Preprocessors	3-1
3.1	cpp	3-1
3.2	fppr	3-1
3.3	COCO	3-8
4	Photran	4-1
4.1	Introduction to Using Photran	4-1
4.2	Starting Photran	4-1
4.3	Creating a New Project	4-2
4.4	Importing Existing Files	4-3
4.5	Create a New Source File	4-3
4.6	Editing a Source File	4-3

4.7	Building a Project	4-4
4.8	Running a Program	4-5
4.9	Make Files	4-5
4.10	Deleting a Project	4-7
4.11	Debugging	4-7
4.12	Terminating a Program	4-11
4.13	Other Sources of Information	4-11
5	Calling Fortran and C	5-1
5.1	Calling C Functions	5-1
6	The Input/Output Module	6-1
6.1	EOF and EOR Parameters	6-1
6.2	Standard Unit Numbers	6-1
6.3	Getting a New Unit Number	6-2
7	The Math Module	7-1
7.1	Math Constants	7-1
7.2	The gcd Function	7-1
8	The Slatec Library	8-1
8.1	Finding Roots in an Interval	8-1
8.2	Finding Roots of a Polynomial	8-2
8.3	Computing a Definite Integral	8-3
8.4	Special Functions	8-5
8.5	Solving Linear Equations	8-5
8.6	Differential Equations	8-5
9	Defined Data Types	9-1
9.1	Varying Length Strings	9-1
9.2	Big Integers	9-2
9.3	High Precision Reals	9-3
9.4	Rationals	9-4
9.5	Quaternions	9-5
9.6	Roman Numerals	9-6
10	Matrix Operations	10-1

10.1	MATRAN	10-1
10.2	BLAS and LAPACK Libraries	10-7
10.3	Atlas Libraries	10-8
11	Plotting	11-1
11.1	Using Gnuplot	11-2
11.2	Running Gnuplot with a Fortran Program	11-2
11.3	Generating Data with a Fortran Program	11-4
A	Software License Agreement	

Installation 1

This chapter describes how to install the Fortran compiler, Eclipse/Photran visual development environment, and related software.

1.1 Introduction

Before installing the software, please read and abide by the Software Agreement in Appendix A. Please note that most of this software is developed by third parties as open source software and is subject to the licensing agreements for the software.

1.2 Implementations Provided

The implementations provided are for Linux and Windows.

1.3 Contents

The software includes a Fortran 95 compiler, the Eclipse/Photran development environment, other software (including linear algebra and plotting software), a manual (of which this is the first section), several books in PDF format, example programs, appropriate licenses, some source code, and documentation.

The free version has an F compiler and only one version of the Atlas linear algebra libraries. The Photran and JRE software are not in the distribution; they must be downloaded; they are free.

1.4 Installation

Notes:

1. You may need to have administrative privileges or be root to install the software.
2. *Do not use or create a folder (directory) with a space in its name for any purpose related to the Fortran Tools.*

1-2 Installation

3. In order to run Photran, a Java Runtime Environment (JRE) must be installed.
4. On Windows, to run Photran and other Fortran Tools, such as the plotting software, Cygwin, a collection of Unix-like tools, must be installed. Cygwin contains many tools useful to the Fortran programmer.
5. Acrobat Reader should be installed to read the documentation that is in PDF format. It is available free from adobe.com if it is not already installed on your system.

1.4.1 Installing the Fortran Tools Files

1.4.1.1 Installing from a CD

The Fortran Tools CD contains one directory (folder) named `FortranTools`. Copy this into a folder whose name has no spaces (i.e., *not* “Program Files” on Windows). Reasonable choices are the C (or other) drive on Windows and your home directory on Linux. If you have the CD that contains Cygwin, this will take several minutes.

1.4.1.2 Installing the Free Version

Download the Fortran Tools with the F compiler for your system and put it in a location of your choice.

Windows: unzip the file `FortranTools_windows_f.zip` into a folder whose name has no spaces (i.e., *not* “Program Files”). A reasonable choice is the C (or other) drive. This should create a folder `C:\FortranTools` (for example).

Linux: `untar (tar xzf FortranTools_linux_f.tgz)` the file into a folder whose name has no spaces. A reasonable choice is your home directory. This should create a directory `FortranTools`.

1.4.2 Cygwin (Windows Only)

Cygwin must be installed. If it is already installed, skip to Section 1.4.3.

1.4.2.1 Installing from the CD

If you have the CD containing the Cygwin files, follow these directions. If not, go to section 1.4.2.2.

1. Go to `cygwin` in the `FortranTools` folder.
2. Execute `setup.exe`. Select `Next`.
3. On the next page, select `Install` from `Local Directory`, then `Next`.
4. On the page `Select Root Install Directory`, type `C:\Cygwin` for the `Root Directory`. The `Cygwin` folder should be directly under a drive name, such as `C:`, `D:`, etc., so the only alternatives are to pick a disk drive other than `C:`. Defaults for the other selections on the page should be `OK`. Then select `Next`.
5. On the next page, use the current folder for the `Local Package Directory`, e.g., `C:\FortranTools\cygwin`. Select `Next`.
6. After a while, you should see a page titled `Cygwin Setup - Select Packages`. Change the selection after the heading `+All` to `Install` (if it says `Default`, or something else, click on the little icon between `+All` and `Default` until it says `+All Install`). Sometimes this takes a few seconds. This will install all of `Cygwin`. Not everything is needed, but it is not obvious what is needed. Besides, `Cygwin` contains lots of cool stuff, so it is not a bad idea to install everything. Select `Next`. Select `Back` to ensure that each category still indicates `Install`; in some versions of `setup.exe`, each must be selected individually. Then select `Next` again.
7. The files will be installed in `C:\Cygwin` (or wherever you indicated). This can take many minutes. When the installation is finished, you should see a page asking if you want to create icons; do so if you want to. Select `Finish`. You should see a little box indicating that the installation is complete. Click `OK`.

1.4.2.2 Installing from the Internet

Note: this is going to take a long time unless you have a high-speed connection to the internet.

1. Use your browser (e.g., `Mozilla`, `Netscape`, or `Internet Explorer`) to access <http://www.cygwin.com/mirrors.html>.

1-4 Installation

2. Select a mirror site near you for downloading. You should see a page titled *Index of Cygwin*. Select `setup.exe` and save it to your disk in a folder (near to where you copied the Fortran Tools) such as `C:\FortranTools\cygwin`.
3. On your computer, use Explorer (the file manager, not a web browser) to locate and select the file `setup.exe` that you just saved. Select it for execution (usually by double clicking). On the page *Cygwin Net Release Setup Program*, select *Next*. On the next page, select *Download without Installing* and then *Next*.
4. On the page *Select Local Package Directory*, select `C:\FortranTools\cygwin` (or wherever you put the file `setup.exe`). Then select *Next*.
5. Select the internet connection. *Direct Connection* should be appropriate in most cases. *Next*.
6. On the *Choose A Download Site* page, you should be able to pick the same site from which you downloaded the file `setup.exe`. Just select the site and click on *Next*.
7. After some files are downloaded, you should see a page titled *Cygwin Setup - Select*. Follow the instructions in 1.4.2.1 (6) above. Select *Next*.
8. The files should be downloaded. This can take a while—at least a good part of an hour with a typical home-based high-speed broadband connection. When the download is finished, install Cygwin as described in 1.4.2.1.

1.4.2.3 Setting your Path Variable

Make sure that the Cygwin executables are in your *path* variable as follows. On XP, click *Start* → *Control Panel* → *System* → *Advanced* → *Environment Variables* → *System Variables*. Scroll down in the *System Variable* in the top window (note that the bottom window is per-user variables) until you get to *Path*. Click on *Path* and edit the value in the line at the bottom of the screen by adding

```
c:\cygwin\bin;c:\cygwin\usr\local\bin;c:\usr\X11R6\bin
```

(the drive may be something other than C) at the beginning of the path, preceded by a semicolon to separate it from the folders already there. Select OK three times.

1.4.2.4 Your Home Directory

Installation of Cygwin should produce a directory (folder) named `home` and within it a directory that is your login name. E.g., `C:\Cygwin\home\joan`. When you execute the `bash` command, you should be taken to this, your home directory. When running `bash`, `~` is shorthand for your home directory.

1.4.3 Downloading Files for the Free Version

If you are using the free version, you need to download files for Photran and JRE.

1.4.3.1 Photran

Go to <http://www.eclipse.org/photran/> and select downloads in the photran section of the left margin. Click on Full Photran 3.?.? for windows or Full Photran 3.?.? for Linux/GTK. Save the file in the directory `FortranTools/photran` (move it there if the download manager does not give you a choice).

1.4.3.2 JRE

Go to <http://www.java.com/>. Click on Manual Download. Select either windows (Offline Installation) or Linux (self extracting file). Save it in directory `FortranTools/java`.

1.4.4 Fortran Tools

For Windows, open a DOS Command Prompt window: *Start* → *Command Prompt*. For Linux, open a terminal window. Type `bash` to run the Bourne Again Shell.

Go to the `FortranTools` directory. E.g.,

```
cd C:/FortranTools or cd ~/FortranTools
```

One of the files there should be a shell script `install_fortrantools`; execute it by typing

```
./install_fortrantools
```

1-6 Installation

The installed files must go into `/usr/local`. You will be asked to verify that it is OK; if not, the installation will terminate.

Also, unless you have the free version, you will be asked to select a version of the matrix libraries by typing a number. Number 0 indicates that the reference Blas and Lapack libraries are to be loaded and used with Matran. Other options are optimized versions for certain architectures. The PII (Pentium II) version probably will run on any Intel system. If there is a problem later with the one you select, simply copy a different version of the libraries into the file `/usr/local/fortran-tools/lib/libmatrix.a`. The other versions are also in `/usr/local/fortrantools/lib`, with a slightly different name, such as `libmatrixPII.a`.

On Linux, you will be asked to accept a license for the Sun JRE. Hit the space bar to scroll down, then answer `yes`.

Make sure that `/usr/local/bin` is in your `PATH` variable. On Linux, this can be accomplished by adding the following line to `~/.bashrc`, for example.

```
export PATH=/usr/local/bin;/bin;$PATH
```

You can verify this by typing

```
which photran
```

The response should be

```
/usr/local/bin/photran
```

On Windows, this is taken care of by modifying the system path variable (1.4.2.3). Type `exit` to exit bash and type `"echo %path%"` in the Command Prompt window to ensure that the path is properly set even when not running bash.

1.4.5 Creating a Shortcut (Optional)

If you want to be able to start Photran by clicking on an icon on the desktop, follow these steps.

1.4.5.1 Windows

1. Use `Explore` (the file manager) to locate the `FortranTools` directory which is the copy of the contents of the CD. Go into `photran`, then `eclipse`.

2. Right click on `eclipse.exe` and select **Create Shortcut**.
3. Right click on the shortcut and change the name to **Photran** (or whatever you like).
4. Move (drag) the icon to the desktop.
5. Right click on the icon and select **Properties**. Select the **Shortcut** tab. In the **Target** field, add to the end the text:

```
-vmargs -Xmx512M
```

This allows Eclipse/Photran to run with more memory.

1.4.5.2 Linux

This works on Red Hat. Things may be different on other Linux distributions.

1. Right click on the background of the desktop. Select **New Launcher**.
2. Select the **Basic** tab and enter **Photran** (or anything you like) in the **Name** field.
3. Enter **photran** in the **Command** field.
4. Select the box to the right of **Icon**; it contains **No Icon**. In the dialog box that comes up, use the **Browse** tab to find the file `photran/eclipse/icon.xpm` in the `FortranTools` directory. Or just type in the full path of the directory. Select the icon and then **OK**. Select **OK** in the **Create Launcher** box. The icon should appear on the desktop. Selecting it will run `/usr/local/bin/photran`.

1.4.6 Java Runtime Environment (JRE)

A JRE must be installed to run Photran. The JRE is not the same thing as the Java SDK.

1.4.6.1 JRE on Windows

To see whether your browser is configured to use the Java Runtime Environment (JRE) or not, first open the Windows Control Panel. From the **Start** menu button, select **Control Panel** to open the Control Panel (this may be different in different versions of Windows). You should see the **Java Coffee**

1-8 Installation

Cup logo icon in the Control Panel. If you do not see the coffee cup icon in the Windows Control Panel, you do not have the latest version of the Sun JRE installed on your computer. Install it as follows:

1. Go to `java` in the `FortranTools` folder copied from the CD.
2. Execute the file `jre*.exe`.
3. Choose `Typical setup` and click `Accept`.
4. After installation is completed, check that all is OK as described in the first paragraph of this section.

To Java enable your browser (probably not necessary to run the Fortran Tools):

1. Double-click the Java icon in the Control Panel to open the Java Control Panel.
2. In the Java Control Panel, select the `Advanced` tab.
3. Under `Settings`, click on `+` icon against `<Applet> tag support`.
4. Make sure the box next to Internet Explorer, Netscape, or Mozilla is checked.
5. If it is not checked, click the checkbox to enable the JRE for your Web browser.
6. Click `Apply`.

1.4.6.2 JRE on Linux

On Linux, a JRE was installed when you installed the Fortran Tools.

1.5 Support

If you have any difficulties or technical queries, please send email to info@fortran.com.

1.6 What is Installed

A script `photran` is in `/usr/local/bin`. It starts up Photran, the graphical development environment. Photran also can be started in Windows by clicking on the shortcut icon created as described in 1.4.5.

The Fortran compiler also is located in `/usr/local/bin`. It can be run from Photran or executed as a command.

The directory `/usr/local/fortrantools/lib` contains Fortran program libraries and some files needed by Photran.

In the directory where you copied the files from the CD, there is a subdirectory `photran`. It contains a subdirectory `eclipse` created during installation that contains the executable file `eclipse.exe`, which starts Eclipse/Photran. This directory must *not* be removed or Fortran Tools will not work (but the Fortran compiler can still be run from the command line).

Also in the same directory are some example programs (`examples`) and the Fortran Tools documentation (`doc`).

1.7 Licenses

1.7.1 Eclipse and Photran

Eclipse and Photran are licensed under the Eclipse Public License; it is the file `ep1-v10.html` in the `doc` directory. Use is also controlled by the Eclipse Software Agreement in the file `eclipse_software_agreement.htm`.

1.7.2 g95

The Fortran compiler is licensed under the Free Software Foundation General Public License; it is the file `gpl.txt` in the `doc` directory.

1.7.3 Cygwin (Windows)

Most of the Cygwin tools are covered by the GNU General Public License (GPL); it is the file `gpl.txt` in the `doc` directory. However, some are public domain, and others have a X11-style copyright. To cover the GNU GPL requirements, the basic rule is if any binaries are distributed, the source also must be made available.

The Cygwin API library found in the `winsup` subdirectory of the source code is also covered by the GNU GPL. By default, all executables link against this library (and in the process in-

1-10 Installation

clude GPL'd Cygwin glue code). This means that unless you modify the tools so that compiled executables do not make use of the Cygwin library, your compiled programs will also have to be free software distributed under the GPL with source code available to all.

Please also see the `-mno-cygwin` option for `g95`.

1.7.4 Java Runtime Environment

The JRE is subject to the license agreement you were required to read when installing the JRE.

1.7.5 Matran

The copyright and license information for Matran are found in the file `Matran.html` in the `doc` directory of the Fortran Tools distribution.

1.7.6 Blas, Lapack, Atlas, and Slathec

These software libraries are distributed without restrictions by Netlib: <http://www.netlib.org/>.

1.7.7 Other Software

Other software in the Fortran Tools may be controlled by notices in the software or documentation for that software.

1.8 Source Code

1.8.1 Eclipse and Photran

Source files for Eclipse may be found at <http://www.eclipse.org/downloads/>.

1.8.2 g95

Source files for the Fortran compiler may be found at http://www.g95.org/g95_source.tgz.

1.8.3 Cygwin (Windows)

The source files for Cygwin are `cygwin*.bz2` in the `src` directory of the distribution.

1.8.4 Matran

Source code for Matran may be found at <http://www.cs.umd.edu/~stewart/matran/Matran.html>.

1.8.5 Blas, Lapack, Atlas, and Slatenc

Source code for these software libraries are found at <http://www.netlib.org/>

1.8.6 Other Software

Source code for some of the other software is in the `src` directory of the distribution.

1.9 Documentation

The `doc` directory copied from the CD contains the following documentation.

- The complete Fortran Tools manual, of which this is the first section. This manual describes how to run Fortran Tools and contains information about the software provided by The Fortran Company.
- `G95Manual.pdf` and `g95_docs.html` describe the Fortran compiler `g95`. Additional information may be found at the `g95` website <http://www.g95.org>. This site lists lots of programs that work with `g95`. There is also information about how to suspend execution of a program and resume execution at the same place.
- Eclipse documentation may be found at the Eclipse website <http://www.eclipse.org>.
- The file `Cygwin.pdf` in the `doc` directory of the Fortran Tools distribution contains a users manual for Cygwin.
- `Matran` is described in *MatranWriteup* in PDF format in the `doc` directory of the distribution.
- Documentation for `Gnuplot` is in the file `Gnuplot.pdf` in the `doc` directory of the distribution.
- Copies of several books, including *Programmers Guide to Fortran 95 Using F*, *Key Features of Fortran 95*, and *Fortran Array and Pointer Techniques*.
- Documentation of several other programs provided with the Fortran Tools is in the `doc` directory of the distribution.

For additional information about Fortran, visit

1-12 Installation

<http://www.fortran.com/>

or contact

The Fortran Company
6025 North Wilmot Road
Tucson, Arizona 85750 USA
+1-877-355-6640 (voice & fax)
+1-520-760-1397 (outside North America)
info@fortran.com

Command Line Compilation 2

Note: If you are using the F subset compiler in the free distribution, please substitute “F” for “g95” in most places that it occurs in this and later sections.

The F subset is equivalent to invoking the g95 compiler with the `-std=F` option. However, some of the default settings and options may be different. For example, bounds checking is on by default in F and real and complex arrays are initialized to NaN, so that the use of an undefined value can be detected easily.

The syntax of F is described in the file `F_bnf.html` in the `doc` directory of the Fortran Tools distribution.

2.1 Usage

Let us go through the steps to create, compile, and run a simple Fortran program. Suppose we want to find the value of $\sin(0.5)$.

The first step is to use any editor to create a file with the suffix `.f95` that contains the Fortran program to print this value. On Linux or Windows with Cygwin, the editor Emacs or Vi might be used as follows:

```
$ vi print_sin.f95
```

In a Windows command line window, Edit or Notepad might be used to create the file.

Suppose the file contains the following Fortran program:

```
program print_sin
  print *, sin(0.5)
end program print_sin
```

A nice convention is to name the file the same as the name of the program, but with the `.f95` suffix.

The next step is to compile the program. The Fortran command has the following form:

2-2 Command Line Compilation

```
g95 [ option ] ... [ file ] ...
```

so the command for our example is:

```
$ g95 print_sin.f95
```

On a Linux or Unix system, this produces the executable program named `a.out`, which can be executed by entering:

```
$ ./a.out
```

On Windows, the executable file is named `a.exe` and can be run by entering the command `a` or `a.exe`.

2.2 Using g95 with the Fortran Tools

When using the special software that comes with the Fortran Tools, it is often necessary to add information to the command line to tell the compiler where this software is located. For example, to use the `new_unit` subroutine in the input/output module, the compiler command must look something like

```
g95 -I/usr/local/fortrantools/lib \  
the_program.f95 \  
-L/usr/local/fortrantools/lib -lfortrantols
```

which can be typed all on one line without the backslashes.

It might be convenient to create an environment variable with the value `/usr/local/fortrantools/lib` to save some typing. This can be done using the Control Panel in Windows or by a command (possibly placed in a startup file such as `.bashrc`). In `bash`, the command would be

```
export FT_LIB=/usr/local/fortrantools/lib
```

Then the command `aobe` could be written

```
g95 -I$FT_LIB the_program.f95 \  
-L$FT_LIB -lfortrantols
```

Please see the individual sections describing the tools to determine which options are required.

2.3 Description

`g95` is a Fortran 95 compiler. It translates programs written in Fortran into executable programs, relocatable binary modules, assembler source files, or C source files.

The suffix of a filename determines the action `g95` performs upon it. Files with names ending in `.f90` or `.f95` are taken to be Fortran source files. Files ending in `.F90` or `.F95` are taken to be Fortran source files requiring preprocessing (Section 3.2). The file list may contain file names of any form.

Modules and include files are expected to exist in the current working directory or in a directory named by the `-I` option.

Options not recognized by `g95` are passed to the link phase (`gcc`).

2.4 Some Options

`-c`

Compile only (produce `.o` file for each source file); do not link the `.o` files to produce an executable file.

`-Dname`

Defines `name` as a preprocessor variable. This is equivalent to passing the `-D` option directly to the preprocessor.

`-fbounds-check`

Checks array and substring bounds at runtime.

`-ffixed-form`

Assumes the source file is fixed source form.

`-ffree-form`

Assumes the source file is free source form.

`--help`

Display information about the compiler.

`-i8`

Sets the default kind of integers to 8.

2-4 Command Line Compilation

-r8

Sets the default kind of reals to 8.

-d8

Equivalent to both **-i8** and **-r8**.

-I *pathname*

Add *pathname* to the list of directories which are to be searched for module information (**.mod**) files and include files. The current working directory is always searched first, then any directories named in **-I** options.

-lx

Load with library *libx.a*. The loader will search for this library in the directories specified by any **-Ldir** options followed by the normal system directories.

-Ldir

Add *dir* to the list of directories for library files.

-o *output*

Name the output file *output* instead of **a.out** (**a.exe** on Windows). This also may be used to specify the name of the output file produced under the **-c** and **-S** options.

-O

Normal optimization.

--version

Print version information about the compiler.

-std=f95

Check that program conforms to Fortran 95 standard.

-std=F

Check that program conforms to the F subset.

-S

Produce assembler output only. Do not assemble and link.

-wall

Enable most warning messages.

2.5 Pre-connected Input/Output Information

Standard error (stderr) unit number = 0

Default standard input (stdin) unit number = 5

Default standard output (stdout) unit number = 6

2.6 Error Messages

The following table gives the correspondence between runtime error numbers and the cause of the error.

-2	End of record
-1	End of file
0	Successful return
1 - 199	Operating system errno codes
200	Conflicting statement options
201	Bad statement option
202	Missing statement option
203	File already opened in another unit
204	Unattached unit
205	FORMAT error
206	Incorrect ACTION specified
207	Read past ENDFILE record
208	Corrupt unformatted sequential file
209	Bad value during read
210	Numeric overflow on read
211	Out of memory
212	Array already allocated
213	Deallocated a bad pointer
214	Bad record read on input

2-6 Command Line Compilation

Preprocessors 3

Preprocessors are available with the F distribution. `cpp` was written as a C preprocessor, but it works for Fortran, also. `fppr` is a simpler preprocessor written by Michel Olgagnon. `COCO` (conditional compilation) is an ancillary Fortran standard.

3.1 `cpp`

`cpp` is invoked with the `g95 -cpp` option. For a description of `cpp`, consult Linux, Unix, or GNU documentation.

3.2 `fppr`

`fppr` is a preprocessor and “pretty printer”. Here is a simple example.

```
$define WINDOWS 0
$define FPPR_KWD_CASE FPPR_LOWER
$define FPPR_USR_CASE FPPR_LEAVE
$define FPPR_MAX_LINE 132
program test_fppr

  $if WINDOWS
  character(len=*), parameter :: slash = "\"
  $else
  character(len=*), parameter :: slash = "/"
  $endif

  character(len=*), parameter :: file_name = &
    "." // slash // "fppr.f95"
  integer :: ios
  character(len=99) :: line

  open (file=file_name, unit=35, &
        iostat=ios, status="old", &
        action="read", position="rewind")
  if (ios == 0) then
    print *, "Successfully opened ", file_name
```

3-2 Preprocessors

```
        read (unit=35, fmt="(a)") line
        print *, "First line: ", trim(line)
    else
        print *, "Couldn't open ", file_name
        print *, "IOSTAT = ", ios
    end if

end program test_fppr
[walt@localhost Examples]$ fppr < fppr_in.F95 > \
                                fppr.f95
This is f90ppr: @(#) fppridnt.f90
                V-1.3 00/05/09 Michel Ollagnon
( usage: f90ppr < file.F90 > file.f90 )
[walt@localhost Examples]$ F fppr.f95
[walt@localhost Examples]$ ./a.out
Successfully opened ./fppr.f95
First line: program test_fppr
```

Running `fppr` with input from `fppr_in.F95` (shown above) produces an output file `fppr.f95`. `fppr` must be executed explicitly; it is not invoked by the Fortran compiler based on the suffix `.F95`, the way `cpp` is. Because the `fppr` variable `WINDOWS` is not defined to be true, the generated code will include the parameter statement that sets the variable `slash` to the forward slash; if `WINDOWS` were true, it would be the backslash. Here is the output file `fppr.f95`.

```
program test_fppr
!
    character (len=*), parameter :: slash = "/"
!
    character (len=*), parameter :: file_name = &
        "." // slash // "fppr.f95"
    integer :: ios
    character (len=99) :: line
!
    open (file=file_name, unit=35, iostat=ios, &
        status="old", action="read", &
        position="rewind")
    if (ios == 0) then
        print *, "Successfully opened ", file_name
        read (unit=35, fmt="(a)") line
        print *, "First line: ", trim (line)
```

```

else
    print *, "Couldn't open ", file_name
    print *, "IOSTAT = ", ios
end if
!
end program test_fppr

```

fppr does not make use of any command line argument, and the input and output files need thus to be specified with redirection, (they default to the standard input and the standard output).

3.2.1 Options

All options have to be specified through the use of directives.

3.2.2 Directives

All fppr directives start with a dollar symbol (\$) as the first nonblank character in an instruction. The dollar sign was chosen because it is an element of the Fortran character set, but has no special meaning or use. The question mark, which is also an element of the Fortran character set with no special meaning, is used as a “vanishing” separator (see \$define below)

```
$define name token-string
```

Replace subsequent instances of *name* with *token-string*. *name* must be identified as a token. In order to enable replacement of sub-strings embedded within tokens, ? is a special “vanishing” separator that is removed by the pre-processor.

```
$define name $"token-string"
```

Replace subsequent instances of *name* with *token-string* where *token-string* must not be analyzed since it may consist of multiple instructions, for instance.

```
$eval name expression
```

Replace subsequent instances of *name* with *value* where *value* is the result, presently of default real or integer kind, of the evaluation of *expression*.

```
$undef name
```

3-4 Preprocessors

Remove any definition for the symbol name.

```
$include "filename"
```

Read in the contents of *filename* at this location. This data is processed by `fppr` as if it were part of the current file.

```
$if constant-expression
```

Subsequent lines up to the matching `$else`, `$elif`, or `$endif` directive, appear in the output only if *constant-expression* yields a nonzero value. All non-assignment Fortran operators, including logical ones, are legal in *constant-expression*. The logical constants are taken as 0 when false, and as 1 when true. Many intrinsic functions are also legal in *constant-expression*. The precedence of the operators is the same as that for F. Logical, integer, real constants and `$defined` identifiers for such constants are allowed in *constant-expression*.

```
$ifdef name
```

Subsequent lines up to the matching `$else`, `$elif`, or `$endif` appear in the output only if *name* has been defined.

```
$ifndef name
```

Subsequent lines up to the matching `$else`, `$elif`, or `$endif` appear in the output only if *name* has not been defined, or if its definition has been removed with an `$undef` directive.

```
$elif constant-expression
```

Any number of `$elif` directives may appear between an `$if`, `$ifdef`, or `$ifndef` directive and a matching `$else` or `$endif` directive. The lines following the `$elif` directive appear in the output only if all of the following conditions hold:

- The *constant-expression* in the preceding `$if` directive evaluated to zero, the name in the preceding `$ifdef` is not defined, or the name in the preceding `$ifndef` directive was defined.
- The constant-expression in all intervening `$elif` directives evaluated to zero.
- The current constant-expression evaluates to non-zero.

If the constant-expression evaluates to non-zero, subsequent `$elif` and `$else` directives are ignored up to the matching `$endif`. Any constant-expression allowed in an `$if` directive is allowed in an `$elif` directive.

```
$else
```

This inverts the sense of the conditional directive otherwise in effect. If the preceding conditional would indicate that lines are to be included, then lines between the `$else` and the matching `$endif` are ignored. If the preceding conditional indicates that lines would be ignored, subsequent lines are included in the output. Conditional directives and corresponding `$else` directives can be nested.

```
$endif
```

End a section of lines begun by one of the conditional directives `$if`, `$ifdef`, or `$ifndef`. Each such directive must have a matching `$endif`.

```
$macro name ( argument [ , argument ] ... ) token-string
```

Replace subsequent instances of *name*, followed by a parenthesized list of arguments, with *token-string*, where each occurrence of an argument in *token-string* is replaced by the corresponding token in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, `fppr` does not re-start its scan for names to expand at the beginning of the newly created *token-string*, the opposite of the C preprocessor.

3.2.3 Macros and Defines

Macro names are not recognized within character strings during the regular scan. Thus:

```
$define abc xyz  
print *, "abc"
```

does not expand `abc` in the second line, since it is inside a quoted string.

Macros are not expanded while processing a `$define` or `$undef`. Thus:

3-6 Preprocessors

```
$define abc zingo
$define xyz abc
$undef abc
xyz
```

produces abc. The token appearing immediately after an `$if-def` or `$ifndef` is not expanded.

Macros are not expanded during the scan which determines the actual parameters to another macro call. Thus:

```
$macro reverse(first,second) second first
$define greeting hello
reverse(greeting,      &
$define greeting goodbye &
)
```

produces

```
$define greeting goodbye greeting.
```

3.2.4 Options

A few pre-defined keywords are provided to control some features of the output code:

```
FPPR_FALSE_CMT !string
```

Lines beginning with *!string* should not be considered as comments, but processed. For instance, one may define:

```
$define FPPR_FALSE_CMT !HPF$
```

in order to use HPF directives in one's code.

```
FPPR_MAX_LINE expression
```

The current desirable maximum line length for deciding about splitting to a continuation line is set to the value resulting of evaluation of *expression*. If the value is out of the range 2-132, the directive has no effect.

```
FPPR_STP_INDENT expression
```

The current indentation step is set to the value resulting of evaluation of *expression*. If the value is out of a reasonable range (0-60), the directive has no effect. Note that it is recommended to use this directive when current indentation is zero, otherwise unsymmetrical back-indentations would occur.

FPPR_NMBR_LINES *expression*

If *expression* evaluates to true, or non-zero, or is omitted, line numbering information is output in the same form as with `cpp`. If *expression* evaluates to 0, line numbering information is no longer output.

FPPR_FXD_IN *expression*

If *expression* evaluates to true, or non-zero, or is omitted, the input treated as fixed-form. If *expression* evaluates to 0, the input reverts to free-form.

FPPR_USE_SHARP *expression*

If *expression* evaluates to true, or non-zero, or is omitted, the sharp sign (#) may be used as well as the dollar sign as the first character of pre-processing commands. If *expression* evaluates to 0, only commands starting with dollar are processed.

FPPR_FXD_OUT *expression*

If *expression* evaluates to true, or non-zero, or is omitted, the output code is intended to be fixed-form compatible. If *expression* evaluates to 0, the output code reverts to free-form.

FPPR_KWD_CASE *expression*

If *expression* evaluates to 1, or is the keyword `FPPR_UPPER`, F keywords are output in upper case. If *expression* evaluates to 0, or is the keyword `FPPR_LEAVE`, Fortran keywords are output in mixed case. If *expression* evaluates to -1, or is the keyword `FPPR_LOWER`, F keywords are output in lower case.

FPPR_USR_CASE *expression*

If *expression* evaluates to 1, or is the keyword `FPPR_UPPER`, user-defined Fortran identifiers are output in upper case. If *expression* evaluates to 0, or is the keyword `FPPR_LEAVE`, user-defined Fortran identifiers are output in the same case as they

3-8 Preprocessors

were input. If *expression* evaluates to -1 , or is the keyword `FPPR_LOWER`, user-defined Fortran identifiers are output in lower case.

3.2.5 Output

Output consists of a copy of the input file, with modifications, formatted with indentation, and possibly changes in the case of the identifiers according to the current active options.

3.2.6 Diagnostics

The error messages produced by `fppr` are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

3.2.7 Source Code

The source code is available in the `src` directory of the F distribution. It is provided by Michel Olagnon and more information about this program and others provided by Michel may be found at

<http://www.ifremer.fr/ifremer/ditigo/molagnon/>

3.3 COCO

The program `coco` provides preprocessing as per Part 3 of the Fortran Standard (`coco` stands for “conditional compilation”). It implements the auxiliary third part of ISO/IEC 1539-1:1997 (better known Fortran 95). (Part 2 is the `ISO_VARYING_STRINGS` standard, which is sometimes implemented as a module.) A restore program, similar to that described in the standard, is also available for download.

The Fortran source code for `coco` may be found at <http://users.erols.com/dnagle..>

Generally, `coco` programs are interpreted line by line. A line is either a `coco` directive or a source line. The `coco` directives start with the characters `??` in columns 1 and 2. Lines are continued by placing `&` as the last character of the line to be continued. Except for the `??` characters in columns 1 and 2, `coco` lines follow the same rules as free format lines in Fortran source code. A `coco` comment is any text following a `!` following the `??` characters. A `coco` comment may not follow the `&`.

A description of coco may be found in the file `coco.html` in the Docs directory. Here is a simple example.

Statement of the problem to be solved: A Fortran program needs to use full path names for file names. The separator in the file names should be `/` if the system is not Windows and `\` if it is Windows. A file `slash.inc` contains the following, which indicates whether the system is Windows or not.

```
?? logical, parameter :: windows = .false.
```

Then the following program will produce the correct character.

```
module slash

?? include "slash.inc"

    character, parameter, public :: &
?? if (windows) then
    slash = "\"
?? else
    slash = "/"
?? end if

end module slash

program p

    use slash
    print *, "Path is usr" // slash // "local"

end program p
```

The COCO preprocessor is run with

```
coco < slash.f90 > new_slash.f95
```

which produces the file `new_slash.f95`:

```
module slash

!>?? include "slash.inc"
!>??! INCLUDE slash.inc
!>?? logical, parameter :: WINDOWS = .false.
!>??! END INCLUDE slash.inc
```

3-10 Preprocessors

```
character, parameter, public :: &
!>?? if (windows) then
!>    slash = "\"
!>?? else
    slash = "/"
!>?? end if

end module slash

program p

    use slash
    print *, "Path is usr" // slash // "local"

end program p
```

Compiling and running the program produces the output:

```
Path is usr/local
```

Photran is a graphical interface for Fortran. It may be used to edit, compile, run, and debug Fortran programs. It is based on the Eclipse open source software (<http://www.eclipse.org>). Photran itself consists of plugins for eclipse developed at the University of Illinois (<http://www.photran.org>). Photran also is open source software. In addition, some enhancements have been provided by The Fortran Company.

4.1 Introduction to Using Photran

You can always run Fortran programs from the command line, but if you want to use the Photran graphical interface to edit, compile, run, and debug Fortran programs, follow the instructions in this section. But first a little jargon so you can read additional documentation about Eclipse and Photran.

When using Photran, your code is organized into *workspaces* and *projects*. A project usually will contain the code for one complete program, consisting of a main program and possibly some modules. These files, and others used by the Photran system, usually are stored in one directory whose name is the name of the project. A workspace consists of projects; it uses a directory whose name is the name of the workspace to store the project directories. A workspace might contain only one project.

To use Photran for Fortran programs, you create a project, which is part of some workspace. You then add source code to the project, either by copying existing files into the project or by creating new source files and typing in the code. Then, using Photran, the project can be built, run, and debugged.

4.2 Starting Photran

1. On Windows, if you created a shortcut to Photran, select it. Otherwise, type `photran`. If this does not work, in Windows, you can use `Explore` to find `eclipse.exe` and select

4-2 Photran

it for execution. In Linux, type its full path name or go to its directory and type `./eclipse`. The Photran logo should appear and after a while Photran should be running.

2. If the screen contains only some logos, select the curved arrow labelled `workbench` to start running some Fortran programs.
3. You will be asked to select a workspace. It is probably a good idea to select a directory different from the location of the software installed from the CD.

4.3 Creating a New Project

Here are the steps to create a new project. As with most other Photran operations, there are several ways to do it. Here is one.

1. Select the `File` tab at the upper left corner of the screen, then `New`, then `Standard Make Project`. If you see a screen asking you to select a wizard, expand the `Make` option and select `Standard Make Project`.
2. On the next screen, pick a name for the project, such as `a_simple_project`. The default workspace directory should be a good choice for this example. Select `Next` (not `Finish`).
3. On the next screen, select the `Error Parsers` tab. Check the following parsers and uncheck the rest:

```
CDT GNU Make Error Parser
CDT GNU C/C++ Error Parser
CDT GNU Assembler Error Parser
CDT GNU Linker Error Parser
Photran Error Parser for G95 Fortran
```

4. Select the `Binary Parsers` tab on the same screen. If you are using Windows, select `PE windows Parser`. The binary parser for Linux should be `Elf Parser`. Now select `Finish`.
5. Select the `+` symbol to the left of the project name in the `Navigator` view and you will see files that have been put there by Photran. They contain information about the project.

If projects have been created previously, some or all of these settings may already be in place.

4.4 Importing Existing Files

Right click on the project name and select **Import** from the list of options. Select **File System** from the next screen and then **Next**. On the **File System** screen, use the **Browse** button to find the directory `/usr/local/fortrantools/lib`. Select **Makefile**. Then **Finish** to copy it to the current project. This file needs to be present in every project unless you create your own Makefile.

Or you can simply copy **Makefile** into the directory containing your project files.

4.5 Create a New Source File

To create a new source file, select **File** in the upper left corner of your screen, then **New**, then **Source File**. There are also icons to do this; determine which ones by putting your cursor over the icons to see what they do. Enter a name (use `sine.f95` for example) for the source file and **Finish**.

4.6 Editing a Source File

1. Double click on the file. This will display the contents of that file in the editor view, which occupies the upper central portion of the screen.
2. If you make a change to the file (this will be indicated by an asterisk by the file name just above the edit view), save the file by selecting the save (floppy disk) icon near the upper left corner of your screen.

To provide a simple example, enter the following program:

```
program sine
  print *, "The sine of 0.5 is", sin(0.5)
end program sine
```

Note the syntax highlighting of Fortran code by Photran. Comments, character strings, and keywords appear in different colors so that they may be identified readily.

4-4 Photran

Line numbers do not appear in the edit window, but the line number and character position within the line of the cursor are displayed below the edit window.

Here is another nice feature of Photran. If you want to comment a whole block of statements, it is necessary to put the comment symbol (!) at the beginning of each statement. To do this using Photran, select the lines to be commented (or uncommented), right click in any open space in the Edit view and select Comment (or Uncomment).

4.7 Building a Project

To build the program, select the *project name* in the Navigator view and select Project. Then select Build Project from the pulldown menu. If the Build Project option cannot be selected, uncheck the Build Automatically option and try again. For `a_simple_project`, something like the following should appear in the Console view near the bottom of your screen (make sure the Console tab is highlighted).

```
make -k clean all
rm -f *.mod *.o RUN* f_Makefile
perl /usr/local/fortrantools/lib/mkmf.pl -t
/usr/local/fortrantools/lib/mkmf_args \
  -p RUN -m f_Makefile -x
make[1]: Entering directory \
  `/home/walt/FortranTools/workspace/test'
g95 -g -wall -fbounds-check \
  -I/usr/local/fortrantools/lib \
  ./sine.f95
g95 trig_plot.o -o RUN \
  -L/usr/local/fortrantools/lib -lfortrantools \
  -lslatec -lmatrix -lg2c
make[1]: Leaving directory \
  `/home/walt/FortranTools/workspace/test'
```

If nothing happens, select the project name again in the Navigator view, select the Project pulldown menu, and select Clean. Select the Clean selected projects button and OK.

The important steps are those that begin with g95. The first of these compiles the program `sine.f95` and the second creates the executable file `RUN`. You will notice some new files ap-

pearing in the project in the Navigator view, including the file RUN.

4.8 Running a Program

A run configuration must be established before any program can be run from Photran. To check if this has been done, select the **Run As** tab above the edit window. If a tab showing **Local Fortran Application** appears, then simply click it and the program should begin execution.

If only a **Run** tab appears, then select it. A window in which a run configuration can be established should appear. Fortran Local Application should appear in the window to the left. Select **New** (lower left). Enter any name; the name of the project might be a good choice.

Select the **Main** tab. Enter the name of the project and enter **RUN.exe (Windows)** or **RUN (Linux)** as the Fortran Application.

On Windows, select the **Environment** tab and then **New**. Enter the name **LD_LIBRARY_PATH** and the value **C:\Cygwin\bin**.

Next select the **Debugger** tab and select **GDB Debugger**.

Then select **Apply** and either **Run** or **Close**.

Once the run configuration has been set up, instead of selecting **Run** tab and **Run As**, click on the **Run** button (the little green arrow). After the program has been run successfully, selecting the **Run** button may cause it to be recompiled if the source code has been changed.

The **Console** view is used for **read *** and **print ***; make sure that view is selected when typing.

On Linux, before each **read** operation from the Console view, it is necessary to force any previous output (such as a prompt) to be displayed; this is done with **call flush(6)**—unit 6 is the standard output unit for **g95**.

4.9 Make Files

When a Fortran program is complicated, it may be necessary to write your own **Makefile**. This can be done by simply editing the file named **Makefile** in your project. The **Makefile** that is provided executes a Perl script (**mkmf.pl**) which builds another make file (**f_Makefile**) based on the organization of modules and **use** statements in the project. Then that file is used to build the executable program. This is the **Makefile** provided.

4-6 Photran

```
FT_LIB = /usr/local/fortrantools/lib
all:
    perl $(FT_LIB)/mkmf.pl \
        -t $(FT_LIB)/mkmf_args -p RUN \
        -m f_Makefile -x
clean:
    rm -f *.mod *.o RUN* f_Makefile
```

Note that the lines executing `perl` and `rm` begin with a tab character, not spaces.

This process can be modified in a few simple ways by editing the file `mkmf_args` (make makefile arguments) located in `/usr/local/fortrantools/lib`. The one provided is:

```
FC = g95
FFLAGS = -g -wall -fbounds-check \
        -I/usr/local/fortrantools/lib

LD = g95

LDFLAGS = -L/usr/local/fortrantools/lib \
        -lfortrantools -lslatec -lmatrix -lg2c
```

The first and third lines indicate that `g95` is to be used to compile and load the program. The second line provides options to the compiler. `-g` is used for debugging, `-wall` says to check for as many errors as possible (subscripts out of bounds, for example), and `-I` tells the compiler where to find some modules provided with Fortran Tools. When the program is ready for production use, this line might be changed to

```
FFLAGS = -O -I/usr/local/fortrantools/lib
```

to turn off error checking and turn on optimization. The `-I` option can be deleted from `FFLAGS` and the `-L` and `-l` options can be deleted from `LDFLAGS` if no Fortran Tools modules are being used.

Documentation for `mkmf.pl` is in the `doc` directory of the Fortran Tools distribution.

To see how the make file system works, let's go through an example provided by the `test_make` project.

1. Create a new project named `test_make` and import the file `Makefile` in the directory `/usr/local/fortrantools/lib`.

2. Import the files `m1.f95`, `m2.f95`, `m3.f95`, and `p.f95` from the `examples` directory of the distribution.
3. Look at the source files. There are three modules and a main program. Module `m1` contains declarations of the parameters `pi` and `e`. `m2` contains a subroutine `s` that uses module `m1` and prints the value of `pi`. `m3` uses `m1` and `m2` and contains a subroutine `s3` that calls `s` to print `pi` and also prints `e`. The main program `p` uses `m1` so it can print the values of `e` and `pi`. It contains a subroutine `ss` that uses `m3` and calls its module procedure `s3`.
4. Build the program. Note the compile commands that are executed and the order in which they are executed.
5. Experiment by changing one of the source files and then rebuilding the program. For example, change `m2` so that the subroutine prints `2*pi`. Don't forget to save the changed file and select the project before selecting **Build Project**. Note the compile commands when the project is rebuilt. Change the subroutine in `m2` and rebuild. Then change the value of `e` in `m1` and rebuild. In all cases, only the files that need to be recompiled are recompiled. This is not important for such a small program, but is for a big complicated one.

4.10 Deleting a Project

To delete a project, right click on the project name in the Navigator view and select **Delete**. The next screen gives you the option of keeping or deleting the contents of the project directory when the project is deleted.

4.11 Debugging

Programs can be debugged using the same Photran interface that is used to edit, build, and run the programs. The debugger has a lot of features, some of which take some effort to learn, but if all you use it for is a replacement for debugging by inserting `print` statements, learning just the simplest features to do that will be well worth the effort.

Let's learn about some of the features with an example.

1. Create a new project named `buggy` in your workspace.

2. Import the file `buggy.f95` in the `examples` directory of the Fortran Tools distribution. Take a look at it if you like.
3. If you are on Windows, select the project, then select the **Project** tab and select **Properties** from the pulldown menu. From the list on the left, select **Make Project**. Select the **Binary Parser** tab and check **PE Windows Parser**, if it is not already selected. Select **Apply** and then **OK**. On Linux, it should be the **Elf Parser**.
4. Build the project.
5. Run the program. There appears to be a problem; if you can figure it out, great, but if not we need to do some debugging.
6. Select the project name; select the **RUN** tab near the top of the screen; then **Debug**. If you see a window with **Create, manage, and run configurations**. Select the **Debugger** tab. Then uncheck the box labelled **Stop at main(0) on start-up**.
7. Set a breakpoint: with the source file `buggy.f95` in the edit window, place the cursor in the left margin of the edit window to the left of the statement

```
j = 1
```

Right click and select **Toggle Breakpoint**. Notice that a small blue circle appears in the margin to indicate the presence of the breakpoint. If you don't set a breakpoint, execution of the program may hang and you will have to terminate the program `gdb` by other means (see 4.12).

8. Select the project name; select the **RUN** tab near the top of the screen; then **Debug As**; then **Debug Local Fortran Application**. If a list pops up, select **GDB Debugger**. The arrangement of views changes significantly.
9. In the upper right corner of the screen, there is a little window that says **Debug**. This used to say **Make**. With the little icon to the left of this window you can change the perspective (the arrangement of the views) to **Debug** or **Make**. Try it.

10. With the perspective set at **Debug**, the program appears in a view near the center of the screen. The program is suspended at the breakpoint as indicated by the little arrow in the left margin pointing to the program statement.
11. To determine the problem, we want to execute a few statements and then see how things look. One way to do this is to use the icons above the **Debug** view. Move your cursor over them to see what they do. **Restart** begins execution of the program from the beginning. **Resume** continues execution from the current place in the program until it hits a breakpoint. **Terminate** (the red square) stops the program. **Step Into** executes one Fortran statement; if it involves a function evaluation or a subroutine call, it stops at the beginning of the procedure invoked. **Step Over** executes one statement, but does not stop inside a procedure that is invoked. Another similar operation is **Run to Line**; there is no icon for this, but can be performed by right clicking in open space in the **Debug** view and selecting it; it causes the program to run to the point where a line is selected with the cursor.
12. Use **Step Over** or **Step Into** to run to the first **if** statement. check the value of the variables **i** and **j** by examining the **variables** view in the upper right portion of the screen.
13. Perform **Step Over** several times to watch the loop get executed three or four times. Look at the **variables** view and notice that each time **j** changes, it turns red. In fact, since the loop exits only when $i > n$, and **i** never changes during the loop, that explains the problem. Fix it by changing the test to use **j** instead of **i**. Probably the easiest way to do this is to terminate the program by selecting the red square, edit the source file, and rebuild the project.
14. We have fixed the bug, but let's try a few more things with the debugger to see how they work. After rebuilding the project, set a breakpoint at the first **print** statement and start the debugger again
15. When the program stops at the breakpoint, look in the **variables** view. **j** is 11, as it should be. To see the values of **my_array**, select the **+** symbol to its left. Note that the el-

ements of the array are numbered from 0. This is because the debugger is derived from a C debugger. To see the value of the character string `c`, do the same thing. It is treated as an array by the debugger because in C, a character string is treated as an array of characters.

16. Use **Step Into** until you get to the `call` statement. Be sure to use **Step Into** again (maybe a couple of times) to enter the subroutine `SubA`.
17. Place the cursor on the line

```
zed(i) = FuncB(y)
```

right click in the open space in the window and select **Run To Line**. Note that the variables local to the subroutine have been added to the *bottom* of the list. Also, there are variables `i` with two different values; one is the `i` declared local to the subroutine and the other is the `i` in the main program.

18. **Step Into FuncB**. Notice that the variables local to the function (e.g., `xx` and `B_result`) have been added to the **variables** view.
19. Suppose we think all is OK in `FuncB`. **Step Return** to complete execution of `FuncB` and go back to `SubA`.
20. Now suppose we suspect that something goes wrong during the last iteration or two of the `do` loop in `SubA`. It would be tedious step through the loop more than 300 times. Instead we can set a conditional breakpoint. First, set a breakpoint at the line

```
y = i
```

Then right click on the blue circle and select **Breakpoint Properties**. Alternatively, select the **Breakpoints** tab near the upper right corner of the screen, right click on the break point just created and select **Properties**. In the **Condition** field, type

```
i > 357
```

Another option would be to type something like `355` in the **Ignore** field so that the breakpoint would be passed 355

times before the program is stopped. Select OK. Note the ? over the blue circle representing the breakpoint.

Resume to run to the breakpoint just set. Look at the Variables window and check to be sure that the loop was executed until the breakpoint condition was met. If this doesn't work, remove the breakpoint at the print statement and start the debugging process over again

21. Select the array `zed` to look at some of its values. Note that you can select portions of the array, which is very convenient if the array is large.
22. Select the red square to terminate the program. Return to the Fortran perspective.

4.12 Terminating a Program

Usually, an executing program can be stopped by clicking on the red square.

Sometimes, an instance of `RUN` or `gdb` may be left running when you thought everything was terminated. This happens especially during debugging. For example, the compiler may not be able to create a new version of `RUN` if the program is running. If this appears to be a problem, terminate all instances of the programs `RUN` and `gdb`. In Windows, `Ctrl-Alt-Del` to get the task manager; in Linux, use `ps` and `kill`.

4.13 Other Sources of Information

1. With Photran running, select the `Help` tab and `Help Contents`. This leads you to the *Workbench Users Guide*.
2. The *Workbench Users Guide* is also available at

<http://www.eclipse.org/documentation/main.html>

Unfortunately, at this time, there is no additional documentation specific to Photran.

3. In the `doc` directory of the distribution, the file `cdt.pdf` contains the *C/C++ Development Toolkit User Guide*. Because some of the Photran software was developed from this toolkit, there is a lot of information that is applicable to Photran, particularly the debugger information.

Calling Fortran and C 5

Fortran programs may call C programs compiled with gcc.

5.1 Calling C Functions

Calling a C function is a little complicated because of the difference in data types, calling conventions, and other things. Fortran 2003 will make this much easier. In the meantime, here is simple example.

```
typedef struct { float r, i;} Complex;

void csub_ (i, d, a, s, c, slen)
int *i;
double *d;
float a[];
char *s;
Complex *c;
int slen;

{
printf ("The value of i is %d\n", *i);
printf ("The value of d is %f\n", *d);
printf ("The value of a[3] is %f\n", a[3]);
printf ("The value of s is %s\n", s);
printf ("The value of slen is %d\n", slen);
printf ("The value of c is (%f, %f)\n",
        c->r, c->i);
}
```

This can be compiled with the command

```
gcc -c csub.c
```

A Fortran program that calls csub is

```
program f_calls_c
integer, parameter :: n = 4
```

5-2 Calling Fortran and C

```
integer, parameter :: double = &
    selected_real_kind(9)
real(kind=double), pointer :: dp
integer :: i
real, dimension(0:9) :: ra = &
    ( (/ (1.1*i, i=0,9) /) )
character(len=3) :: s = "abc"
complex :: c = (1.1, 2.2)

allocate (dp)
dp = 4.2_double
call csub (n, dp, ra, s, c)
end program f_calls_c
```

The program can be compiled and linked by the command

```
g95 csub.o f_calls_c.f95
```

Executing the program produces the output

```
The value of i is 42
The value of d is 4.200000
The value of a[3] is 3.300000
The value of s is abc
The value of slen is 3
The value of c is (1.100000, 2.200000)
```

Note that the name of the C function has an underscore (`_`) appended. Also, the real and complex dummy arguments are pointers to correspond to the addresses passed for the actual argument.

These programs are in the `examples` directory of the Fortran Tools distribution.

5.1.1 Data Types

The following table shows the correspondence between Fortran and C data types.

where the form of `Complex`, `DComplex`, and `QComplex` are given by

```
typedef struct { float re, im; } Complex;
typedef struct { double re, im; } DComplex;
typedef struct { long double re, im; } QComplex;
```


F data type	C data type
integer (8 bits)	signed char
integer (16 bits)	short
default integer (32bits)	int
integer (64 bits)	long long
logical (8 bits)	char
logical (16 bits)	short
default logical (32 bits)	int
logical (64 bits)	long long
real (single)	float
real (double)	double
real (quadruple)	long double
complex (single)	Complex
complex (double)	DComplex
complex (quadruple)	QComplex
character	***

***For Fortran character actual arguments, there must be two C dummy arguments: `char *` for the string and `int` for the length. The length arguments must be at the end of the dummy argument list in the correct order.

The Input/Output Module 6

The input/output module contains a few useful parameters and a subroutine that returns a unit number that exists, but is not connected.

When compiling using the command line, the following compiler options must be used.

```
-I/usr/local/fortrantools/lib prog.f95\  
-L/usr/local/fortrantools/lib -lfortrantools
```

6.1 EOF and EOR Parameters

The I/O module contains parameters with the values returned by `iostat=` when encountering an end of record or end of file, respectively. The parameter names are `end_of_record` and `end_of_file`.

```
use io_module  
  . . .  
do  
  read (iostat = ios, unit = 17) x  
  if (ios == end_of_file) exit ! End of file  
  . . .
```

6.2 Standard Unit Numbers

The I/O module contains parameters with the values of the standard I/O units. The names of the parameters and their values are

```
standard_input_unit = 5  
standard_output_unit = 6  
standard_error_unit = 0
```

Example:

```
use io_module  
  . . .  
read (unit=standard_input_unit, fmt=*, . . .
```

6.3 Getting a New Unit Number

The I/O module provides a function `new_unit` that is a unit number that exists, but is not open (connected). The value `-1` (an illegal unit number) is returned if none is available.

```
use io_module
integer :: unit_a, unit_b
. . .
call new_unit(unit_a)
open (unit=unit_a, file= . . .)
call new_unit(unit_b)
open (unit=unit_b, file= . . .)
```

Without the first `open` statement, `new_unit` would return the same value both times, finding the same unit number that exists but is not open.

The Math Module 7

When using these features and compiling from the command line the following options must be used.

```
-I/usr/local/fortrantools/lib prog.f95\  
-L/usr/local/fortrantools/lib -lfortrantools
```

7.1 Math Constants

The module `math_module` contains definitions of parameters for the constants π , e , ϕ , and γ . The names of the constants are `pi`, `e`, `phi`, `gamma`, `pi_double`, `e_double`, `phi_double`, and `gamma_double`. An example of its use is

```
program print_pi  
  
    use math_module  
  
    print *, pi_double  
  
end program print_pi
```

7.2 The gcd Function

Also in the `math` module is the elemental function `gcd` that computes the greatest common divisor of two integers or two integer arrays.

```
program test_gcd  
    use math_module  
    print *, gcd((/432,16/), (/796,48/))  
end program test_gcd
```

which prints
4 16

The Slatec Library 8

The Slatec library is a collection of mathematical routines developed jointly by Sandia National Laboratories, Los Alamos National Laboratory, and the Air Force Phillips Laboratory, all in New Mexico.

With Fortran Tools, they may be used in a Fortran program as a “built-in” module. Invoke any of the procedures described below from any Fortran program containing the following statement:

```
use slatec_module
```

When compiling using the command line, the compiler options

```
-I/usr/local/fortrantools/lib prog.f95\  
-L/usr/local/fortrantools/lib -lslatec
```

must be used.

8.1 Finding Roots in an Interval

```
find_root_in_interval(f, a, b, root, indicator)
```

is a subroutine that searches for a zero of a function $f(x)$ between the given values a and b .

f is a function of one variable. a and b specify the interval in which to find a root of f . $root$ is the computed root of f in the interval a to b . These are all type default real.

$indicator$ is an optional default integer argument—if it is zero, the answer should be reliable; if it is negative, it is not.

Here is an example using the subroutine `find_root_in_interval`.

```
module function_module
```

```
public :: f
```

```
contains
```

```
function f(x) result(r)

    real, intent(in) :: x
    real :: r

    r = x**2 - 2.0

end function f

end module function_module

program find_root

    use function_module
    use slatec_module
    real :: root
    integer :: indicator

    call find_root_in_interval&
        (f, 0.0, 2.0, root, indicator)

    if (indicator == 0) then
        print *, "A root is", root
    else
        print *, "Root not found"
    end if

end program find_root
```

Running this program produces

```
A root is 1.4142114
```

8.2 Finding Roots of a Polynomial

The subroutine

```
find_roots_of_polynomial &
    (coefficients, roots, indicator)
```

accepts the coefficients of a polynomial and finds its roots (values where the polynomial is zero).

`coefficients` is a default real array; the element with the smallest subscript is the constant term, followed by the first degree term, etc. Thus, a reasonable choice is to make the lower bound of `coefficients` 0 so that the subscript matches the power of the coefficient.

`roots` is a complex array with at least as many elements as the degree of the polynomial. The roots of the polynomial will be found in this array after calling `find_roots_of_polynomial`.

`indicator` is a default integer optional argument; if it is negative, the solution is not reliable. In particular, if `indicator` is `-1`, a solution was not found in 30 iterations, if it is `-2`, the high-order coefficient is 0, if it is `-3` or `-4`, the argument array sizes are not appropriate; if it is `-5`, allocation of a work array was not successful.

Here is a simple example that computes the roots of $x^2 - 3x + 2 = 0$.

```

program poly_roots

    use slatec_module

    complex, dimension(2) :: roots
    integer :: ind

    call find_roots_of_polynomial &
         ( (/ 2.0, -3.0, 1.0 /), roots, ind)
    print *, "Indicator", ind
    print *, "Roots", roots

end program poly_roots

```

Running the program finds the roots 1 and 2.

```

Indicator 0
Roots (2.00000,0.00000E+00)
(1.00000,0.00000E+00)

```

8.3 Computing a Definite Integral

```
integrate(f, a, b, value, tolerance, indicator)
```

is a general purpose subroutine for evaluation of one-dimensional integrals of user defined functions. `integrate` will pick its own points for evaluation of the integrand and these will vary from problem to problem. Thus, it is not designed to integrate over data sets.

`f` must be a function with a single argument. `a` and `b` are the limits of integration. `tolerance` is an optional requested error tolerance; if it is not present, 10^{-3} is used. `value` is the calculated integral. These are all type default real.

If the returned value of the optional default integer argument `indicator` is negative, the result is probably not correct. A positive value of `indicator` represents the number of integrand evaluations needed.

```
module sine_module

public :: sine

contains

function sine (x) result (sine_result)

    intrinsic :: sin
    real, intent (in) :: x
    real :: sine_result

    sine_result = sin (x)

end function sine

end module sine_module

program integration

use sine_module
use slatec_module
real :: answer
integer :: indicator

call integrate(sine, a=0.0, b=3.14159, &
               value=answer, tolerance=1.0e-5, &
               indicator=indicator)
```

```

print *, "Indicator is", indicator
print *, "Value of integral is", answer

end program integration

```

Running this program produces

```

Indicator is 25
Value of integral is 2.0000000

```

8.4 Special Functions

$\ln_gamma(x)$ is a function that returns the natural logarithm of the gamma function for positive real values of x . $\operatorname{asinh}(x)$, $\operatorname{acosh}(x)$, and $\operatorname{atanh}(x)$ return the inverse hyperbolic function values. The program

```

program test_gamma
use slatec_module
print *, "4! = ", exp(ln_gamma (5.0))
end program test_gamma

```

produces

```

4! = 24.0000000

```

8.5 Solving Linear Equations

The Slatec linear equation solving program has bugs. Use the Matran solver or call the Lapack routines directly (see the Matrix chapter).

8.6 Differential Equations

```

solve_ode &
(f, x0, xf, y0, yf, tolerance, indicator)

```

is a subroutine that solves an ordinary differential equation

$$\frac{du}{dx} = f(x, u)$$

using a fifth-order Runge-Kutta method.

f must be a function of two variables. x_0 is the initial value of x . y_0 is the initial value of y . x_f is the final value of x . y_f is the final solution value of y . $tolerance$ is an optional request-

8-6 The Slatec Library

ed tolerance; if not present 10^{-3} is used. All of these are type default real.

indicator is an optional default integer value—if it is negative, the solution is not reliable; a value of 2 indicates success.

Here is a simple example with $f(x, u) = -0.01y$, $x_0 = 0$, $y_0 = 100$, and $x_f = 100$.

```
module f_module

    public :: f

contains

function f(x, y) result(r)

    real, intent(in) :: x, y
    real :: r

    r = -0.01 * y

end function f

end module f_module

program test_ode

use slatec_module
use f_module

real :: x0 = 0.0, xf = 100.0, &
        y0 = 100.0, yf

call solve_ode (f, x0, xf, y0, yf)

print *, "Answer is", yf

end program test_ode
```

Running the program produces

```
Answer is 36.7878761
```

Defined Data Types 9

There are several modules available to the Fortran programmer that define new data types and a selection of operations on those types. The code for varying strings, big integers, rationals, quaternions, and Roman numerals all conform to the F subset; the source for each of these modules is available in the `src` directory to provide information about the modules and examples of how to build these abstract data types.

When compiling using the command line, the compiler options

```
-I/usr/local/fortrantools/lib prog.f95\  
-L/usr/local/fortrantools/lib -lfortrantools
```

must be used.

9.1 Varying Length Strings

The ISO varying string module provides the type `iso_varying_string` with the operations you would expect to have for character string manipulations (concatenation, input/output, character intrinsic functions). Unlike Fortran character variables, a varying string variable has a length that changes as different values are assigned to the variable. Here is a simple program illustrating these features.

```
program string  
  use iso_varying_string  
  type(varying_string) :: s  
  call get(string=s)  
  s = s // s  
  call put(string=s)  
  print *, len(s)  
end program string
```

The following lines show what happens when the program is compiled and run.

9-2 Defined Data Types

```
$ F string.f95
$./a.exe
A nice string.
A nice string.A nice string. 28
```

The current version of the source code is from Rich Townsend and has been modified slightly so that we have an F conformant version. This program is in the source code directory.

9.2 Big Integers

The `big_integer` data type can represent very large nonnegative integers. The representation of a big integer is a structure with one component that is an array of ordinary Fortran integers. In this version, the largest integer that can be represented is fixed, but the size is specified by a parameter that can be changed. The module may then be recompiled. The source for this module is in the `examples` directory of the distribution. All of the intrinsic operations and functions for intrinsic Fortran integers are available for big integers.

```
program factors
  use big_integer_module
  type(big_integer) :: b, n, s

  b = "9876543456789"

  n = 2
  call check_factor()
  s = sqrt(b)
  n = 3
  do
    if (n > s) exit
    call check_factor()
    n = n + 2
  end do
  if (b /= 1) then
    call print_big(b)
    print *
  end if
```

contains

```
subroutine check_factor()
  do
    if (modulo(b, n) == 0) then
      call print_big(n)
      print *
      b = b / n
      s = sqrt(b)
    else
      exit
    end if
  end do
end subroutine check_factor

end program factors
```

Running the program produces

```
3
3
3
3
17
97
1697
43573
```

9.3 High Precision Reals

9.3.1 The MP Module

This module provides the capability of computing with large precision real values. It was written by David Bailey of Lawrence Berkeley National Laboratory. A description of the module is in the files `mp.ps` and `mp.pdf` in the `doc` directory. More information may be found at <http://www.nersc.gov/~dhbailey/mpdist/mpdist.html>. Here is a simple example of its use.

```
program mp
  use mp_module
  type(mp_real) :: pi

  call mpinit()
```

9-4 Defined Data Types

```
pi = 4.0 * atan(mpreal(1.0))
call mpwrite(6, pi)
end program mp
```

The result printed consists of quite a few digits of π .

```
10 ^      0 x
3.1415926535897932384626433832795028841971693993
7510582097,
```

9.3.2 The XP Module

This module also provides the capability of computing with large precision real values. It was written by David Smith. A description of the module is in the file `xp.txt` in the `doc` directory. Here is a simple example of its use.

```
program test_xp
  use xp_real_module
  type (xp_real) :: x, y
  x = 1.0
  y = 4.0
  call xp_print(y*atan(x))
end program test_xp

3.141592653589793238462643383279502884197E+0
```

9.4 Rationals

A module to compute with rational numbers is provided by Dan Nagle of Purple Sage Computing Solutions, Inc. Some details are provided in the file `rationals.txt` in the `doc` directory. Here is a simple example.

```
program test_rationals
  use rationals_module
  type(rational) :: r1, r2

  r1 = (/3, 4/)
  r2 = (/5, 6/)
  r1 = r1 + r2
  print *, real(r1)
end program test_rationals

1.5833333333333333
```


9.5 Quaternions

The quaternions module was written by David Arnold of the College of the Redwoods. The only documentation is the source file `quaternions_module.f95` in the `src` directory. There is some information about quaternions in the file `quaternions.pdf` in the `doc` directory and the original article about quaternions presented by William Hamilton in 1843 can be found at <http://www.maths.tcd.ie/pub/HistMath/People/Hamilton/Quatern2/Quatern2.html>. Here is an example.

```

program Quaternions
  use Quaternions_module
  type(quaternion) :: u, v
  u=quaternion(1,2,3,4)
  v=quaternion(5,6,7,8)
  call quaternion_print(u+v)
  print *, 3+4
  print *
  call quaternion_print(u-v)
  print *, 3-4
  print *
  call quaternion_print(3.0*u)
  call quaternion_print(u*v)
  print *, 3*4
  print *
  call quaternion_print(conjg(u))
  print *, conjg((3,4))
  print *
  print *, (abs(u))
  print *, abs((3,4))
end program Quaternions

( 6.000000  8.000000  10.000000  12.000000)
7

( -4.000000 -4.000000 -4.000000 -4.000000)
-1

( 3.000000  6.000000  9.000000  12.000000)
( -60.000000  12.000000  38.000000  24.000000)
12

```

9-6 Defined Data Types

```
( 1.000000 -2.000000 -3.000000 -4.000000)
(3.0000000,-4.0000000)

5.4772258
5.0000000
```

9.6 Roman Numerals

This module to compute with Roman numbers was written by Jeanne Martin, former convenor of the international Fortran standards committee and an author of *The Fortran 95 Handbook*. The only documentation available is in the source file in the src directory.

```
program test_roman
use roman_numerals_module
implicit none

type(roman) :: r
integer :: i

write (unit=*, fmt="(a)") "Integer Roman Number"
do i = 1900, 2000
  r = i
  write (unit=*, fmt="(/, tr4, i4, tr2)", &
        advance = "NO") i
  call print_roman (r)
end do
write (unit=*, fmt="(/)")

end program test_roman
```

Here is the result of running the program.

Integer	Roman Number
1900	MCM
1901	MCM I
1902	MCM II
1903	MCM III
1904	MCM IV
1905	MCM V
1906	MCM VI

· · ·
1998 MCMXCVIII
1999 MCMXCIX
2000 MM

9-8 Defined Data Types

Matrix Operations 10

Fortran has extensive built-in operations on arrays, which may be used to do matrix manipulations when the matrices are represented as ordinary Fortran arrays. For example, two matrices may be added by writing

```
A + B
```

and their matrix product may be formed as

```
matmul(A, B)
```

because `matmul` is a standard Fortran intrinsic function.

However, more complicated operations require sophisticated programs to do the calculations effectively and efficiently. Fortunately, a lot of work has been done in this area and the results are included in the Fortran Tools. The BLAS and LAPACK libraries have been used widely for years; in addition, MATRAN (may-tran) provides a higher level interface to these routines.

When compiling using the command line, the compiler options

```
-I/usr/local/fortrantools/lib prog.f95\  
-L/usr/local/fortrantools/lib -lmatrix -lg2c
```

must be used.

10.1 MATRAN

MATRAN is a collections of modules containing procedures that may be used to perform a variety of matrix operations, such as solving linear equations and computing eigenvalues. These procedures call BLAS and LAPACK routines.

All computations are performed with double precision real values.

MATRAN was developed by G. W. (Pete) Stewart, Department of Computer Science, Institute for Advanced Computer Studies, University of Maryland. The web site is:

10-2 Matrix Operations

<http://www.cs.umd.edu/~stewart/>

A few of the features of MATRAN are described here. More complete documentation may be found in *MatranWriteup* in PDF format in the doc directory of the Fortran Tools distribution.

10.1.1 The Rmat and Rdiag Derived Types

Most of the matrix computations in MATRAN are performed on objects of type `Rmat` (real matrix) and `Rdiag` (real diagonal matrix). These are derived types provided with MATRAN. For example, when solving a system of linear equations, objects of type `Rmat` are passed to the solver, not plain Fortran arrays. Here is a partial description of the `Rmat` type:

```
type :: Rmat
  real(wp), pointer :: a(:, :) => null()
  integer :: nrow = 0, ncol = 0
  . . .
end type Rmat
```

The first component `a` is a real array pointer. `wp` is the kind of the working precision, which is default real for the libraries provided with the Fortran Tools. The component `a` is default initialized to null, which means that it will be initialized to the null pointer for each `Rmat` object created. `nrow` and `ncol` are the number of rows and columns of the matrix, respectively.

For example if `x` is declared to be type `Rmat`

```
x % ncol
```

is the number of columns in matrix `x` and

```
x % a(nrow, :)
```

is the last row of the matrix.

Thus, `Rmat` objects may be manipulated directly (their components are not private) as well as with the procedures provided by MATRAN.

The type `Rdiag` (real diagonal matrix) represents a diagonal matrix as a one-dimensional array, consisting of the diagonal elements, and other components, such as the size of the matrix.

10.1.2 Example: Linear Equations

Let us look at some of the MATRAN operations used to construct a program to solve a set of linear equations.

Here is the program; it solves $Ax = b$.

```

program matran_linear_equations

use MatranRealCore_m

integer, parameter :: n = 3
integer :: i, j

type(Rmat) :: A, b, x

! Put some values in the matrix A
A = reshape( (/ ((real(i+j),i=1,n),j=1,n) /), &
            shape = (/ n,n /) )
A%a(n,n) = -A%a(n,n) ! Make sure A is not singular

! Put some values in the vector b
b = reshape( (/ (real(i), i = 1, n)/), (/n,1/))
b = A * b

! Solve the linear equations
x = A .xiy. b

call Print(A, 15, n, "Array A", e=1)
call Print(b, 15, n, "vector B", e=1)
print *
print *
print "(a, 3f7.4)", &
      "The solution to Ax = B is", x%a(1:n, 1)
print *

call Clean(A)
call Clean(b)
call Clean(x)

end program matran_linear_equations

```

The program could be run using Photran or from the command line by entering (on one line, if desired)

10-4 Matrix Operations

```
g95 matran_linear_equations.f95 \  
-I/usr/local/bin/FortranTools \  
-L/usr/local/bin/FortranTools -lmatrix
```

A , b , and x are type `Rmat` objects. All three are, in effect, representations of matrices, even though in this program we think of b and x as vectors to hold the constants of the equations and the solution to the equations, respectively. Remember that Fortran is case insensitive, but case is used in the program in the traditional way: uppercase for matrices and lowercase for vectors.

The `use` statement in the subroutine accesses a module containing many of the MATRAN features. To use other MATRAN features, additional `use` statements may be needed; see the eigenvalues example in 10.1.3.

A and b are given values using an extended assignment statement. In the statement

```
b = A * b
```

the operation is matrix multiplication, provided by MATRAN. This assigns values to b that will produce a solution we will recognize.

The statement

```
x = A .xiy. b
```

assigns the solution to x by computing $A^{-1}b$ using the operator `.xiy.`, which is intended to suggest “ x inverse times y ”. (Of course, the inverse of A is not actually calculated in order to solve the equations.)

The MATRAN subroutine `Print` is used to verify the values of A and b used for the equations.

The subroutine `Clean` releases allocatable storage. Note that there are other MATRAN subroutines that deal with the deallocation of dummy arguments of type `Rmat`.

Here is the output from the program.

```
Array A  
3 3 3 3 GE T 0  
1 1 2 3  
1 2.000E+0 3.000E+0 4.000E+0  
1 1 2 3  
2 3.000E+0 4.000E+0 5.000E+0
```


	1	2	3
3	4.000E+0	5.000E+0	-6.000E+0

Vector B

3 1 3 1 GE T 0	1
1	2.000E+1
2	2.600E+1
3	-4.000E+0

The solution to $Ax = B$ is 1.0000 2.0000 3.0000

10.1.3 Example: Eigenvalues

For this example, we assume that the main program uses ordinary Fortran arrays to store matrices. To compute the eigenvalues of a matrix, we want to call the subroutine `eigenvalues`, passing such an array. This subroutine will use MATRAN objects and the routine `Eig` to compute the eigenvalues.

The program could be run using Photran or from the command line by entering (on one line, if desired)

```
g95 matran_eigenvalues.f95 \
-I/usr/local/bin/FortranTools \
-L/usr/local/bin/FortranTools -lmatrix
```

Here is the program.

```
program matran_eigenvalues

integer, parameter :: n = 5
real, dimension(n,n) :: A
complex, dimension(n) :: e
integer :: i, j

! Put some values in the matrix A
call random_seed()
call random_number(A)
A = 10.0 * A - 5.0

call eigen_values(A, e)
```

10-6 Matrix Operations

```
print *
print *, "The eigenvalues of A are:"
print *
do i = 1, n
    print "(i3, 2(f7.2, a))", i, &
        real(e(i)), " +", &
        imag(e(i)), "i"
end do
print *
print *

contains

subroutine eigen_values (D, e)

    use MatranRealCore_m
    use RmatEig_m

    real, dimension(:, :), intent(in) :: D
    complex, dimension(:), intent(out) :: e
    type(Rmat) :: RD
    type(RmatEig) :: eigD

    RD = D
    call Print(RD, 9, 2, "Array D", e = 1)

    call Eig(eigD, RD)

    ! D is the eigenvalue component of eigD
    e = eigD % D

    call Clean(RD)
    call Clean(eigD)

end subroutine eigen_values

end program matran_eigenvalues
```

The Fortran intrinsic subroutines `random_seed` and `random_number` are used to put some values in the array `A`. The statement

$$A = 10.0 * A - 5.0$$

causes the numbers to be between -5 and $+5$ instead of between 0 and 1 . The array A is passed to the subroutine `Eig` with a vector e to hold the eigenvalues. The dummy argument D is assigned to the variable `RD` of type `Rmat`. The array is printed.

The call to subroutine `Eig` computes the eigenvalues of `RD` and stores the results in `eigD`, a MATRAN object of type `RmatEig`, defined in the module `RmatEig_m`. The component `D` (not the same as the dummy array `D`) contains the eigenvalues and this is assigned to the dummy array e , which is returned as the value of the actual argument e . The eigenvalues stored in e are then printed.

Here is the result of one execution of the program.

Array D

5 5 5 5 GE T 0

	1	2	3	4	5
1	-3.54E+0	-1.88E+0	3.01E+0	2.51E+0	3.33E+0
	1	2	3	4	5
2	4.19E+0	2.04E+0	-4.80E+0	-3.19E+0	1.02E+0
	1	2	3	4	5
3	-3.99E+0	7.37E-1	-3.98E+0	-1.34E+0	-7.92E-1
	1	2	3	4	5
4	-3.44E+0	3.74E+0	1.49E-1	-3.59E+0	-1.79E+0
	1	2	3	4	5
5	-4.16E+0	-1.70E-1	1.48E+0	4.89E+0	4.46E+0

The eigenvalues of A are:

1	-1.31 +	7.13i
2	-1.31 +	-7.13i
3	-3.02 +	1.91i
4	-3.02 +	-1.91i
5	4.05 +	0.00i

10.2 BLAS and LAPACK Libraries

It is possible to invoke any of the BLAS or LAPACK procedures directly in a Phortran project or from the command line by using the `-lmatrix` option on the compile command.

An extensive description of LAPACK is found at

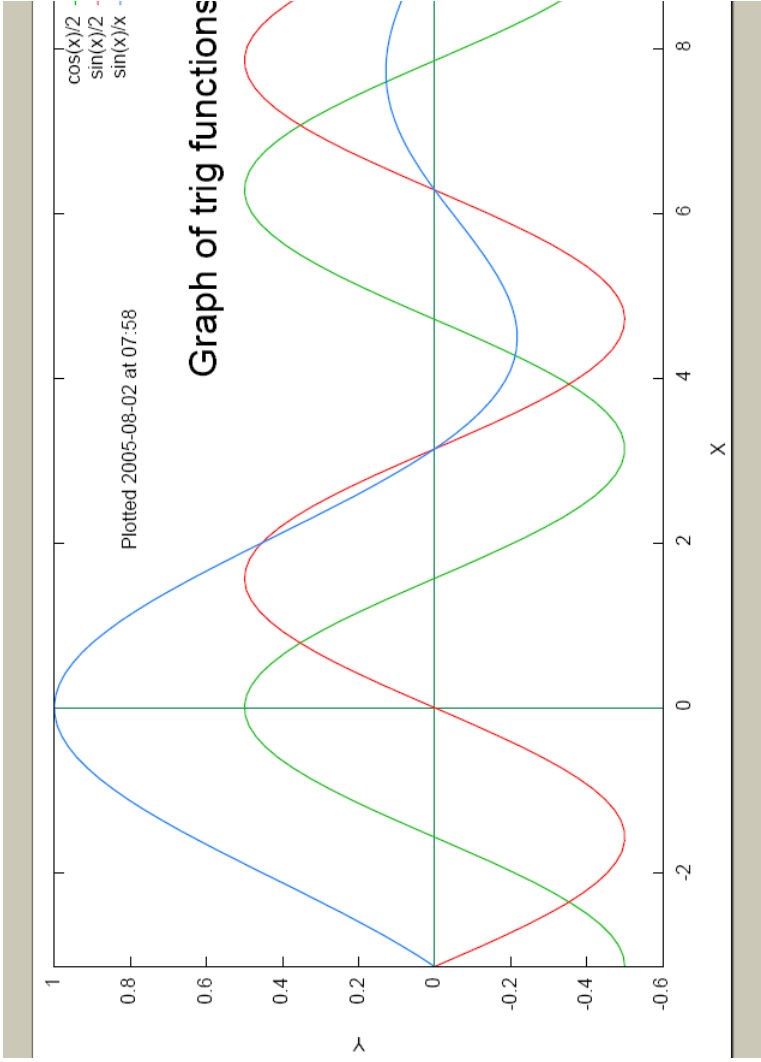
<http://www.netlib.org/lapack/lug/index.html/>

10.3 Atlas Libraries

The Atlas libraries are optimized versions of the Blas and Lapack routines. They must be built for each architecture and operating system. If, when installing, you specified the reference libraries by entering the number 0, you have not installed the Atlas libraries; you have installed the reference libraries. If you specified a different matrix library, the MATRAN programs should run considerably faster. However, if there is some problem, you can always use the reference libraries by doing one of the following:

- Use `-lmatrix0` instead of using `-lmatrix` on the command line when compiling.
- In the file `/usr/local/fortrantools/lib`, edit the file named `mkmf_args` to use `-lmatrix0` instead of `-lmatrix`.
- Reinstall the Fortran Tools and specify a different matrix library.

Conversely, if the reference libraries were installed, you can use the Atlas library by specifying `-lmatrixPII` (or the appropriate library name).



11-2 Plotting

The Fortran Tools use Gnuplot to do plotting. It is part of the Linux distribution and Cygwin.

11.1 Using Gnuplot

One way to use Gnuplot is to type the commands into the `gnuplot` program. On Windows, be in a Cygwin window and type `startx` to produce an X window; or in a DOS window type `bash<C:\cygwin\usr\x11r6\bin\startx`. The default file format is `x11`, so this is necessary only if the default output file format is not changed. On either system, type `gnuplot`. Enter Gnuplot commands, such as

```
$ gnuplot
> plot sin(x)
> exit
```

11.2 Running Gnuplot with a Fortran Program

To run Gnuplot with a Fortran program, create a project and copy Makefile as usual. Then create a new file called, for example, `trig_plot.gp` containing the following Gnuplot commands, which produce the plot on page 11-1.

```
# Plot some trig functions with Gnuplot

# Draw the axes
set xzeroaxis lt 22
set yzeroaxis lt 22
set xlabel "x"
set ylabel "y"

# Establish the time stamp
set time "Plotted 20%y-%m-%d at %H:%M" \
    45,27 "Helvetica,6"

# Provide identifying label
set label "Graph of trig functions" at 4,0.6 \
    font "Helvetica,12"

# Set plot type and output file
set terminal pdf
set output "trig_plot.pdf"
```

```
# Create the plot
plot [-pi:3*pi] cos(x)/2 lt 2 lw 2, \
             sin(x)/2 lt 1 lw 2, \
             sin(x)/x lt 3 lw 2
```

Create a Fortran program that runs `gnuplot` as follows:

```
program trig_plot

    call system ("gnuplot ""trig_plot.gp""")

end program trig_plot
```

Note that the double quotes are there so that the value of the command will be

```
gnuplot "trig_plot.gp"
```

The plot is placed in the output file `trig_plot.pdf`, which may be viewed with Acrobat Reader.

In this case, another way to run the plot is to change `Makefile` to

```
all:
    gnuplot "trig_plot.gp"
clean:
    rm -f *.mod *.o RUN* f_Makefile
```

and simply build the project, which will run `gnuplot`. Remember that the first character in the lines after `all:` and `clean:` must be a tab, not spaces.

11.2.1 Some Gnuplot Commands

We can use the file named `plot_heat.gp` to briefly examine some of the common Gnuplot commands. Complete descriptions of all the commands may be found in `Gnuplot.pdf` in the `docs` directory of the Fortran Tools distribution.

```
set xzeroaxis lt 22
```

indicates that the x axis is to be drawn on the plot with line type (`lt`) 22. To see the line types and point types, execute `gnuplot` and type `test`. The next three commands then do what is expected.

11-4 Plotting

```
set time "Plotted 20%y-%m-%d at %H:%M" \  
45,27 "Helvetica,6"
```

puts a time stamp on the graph at a position indicated by character offsets 45 (x) and 27 (y). The format is given by the first character string and the font and size by the second.

```
set label "Graph of trig functions" at 4,0.6 \  
font "Helvetica,12"
```

places a label at coordinate position (4, 0.6) in Helvetica 12-point font.

```
set terminal pdf  
set output "trig_plot.pdf"
```

indicate the format of the output file is PDF and the plot will be put in file `trig_plot.pdf`.

```
plot [-pi:3*pi] cos(x)/2 lt 2 lw 2, \  
sin(x)/2 lt 1 lw 2, \  
sin(x)/x lt 3 lw 2
```

This command generates the plot, with three curves, the label, the axes, the axis labels, and the time stamp. The notation in brackets indicates the range of x values to include in the plot. `pi` is a built-in variable. `lt` indicates the line type and `lw` means the line width is to be two times the normal size. Line types 1, 2, and 3 produce lines with color red, green, and blue, respectively.

11.3 Generating Data with a Fortran Program

The program `heat.f95` illustrates how a Fortran program can generate the data for a plot. This program generates data for many plots, in fact, and displays them sequentially. To run this program with Photran, be sure to start Photran in an X window; how to do this is explained in 11.1. If not using Photran, execute the program in an X window. Here is the program:

```
! A simple solution to the heat equation using  
! pointers. Results are plotted with Gnuplot.
```

```
program heat
```



```

implicit none

integer, parameter    :: nn = 20, & ! Size of grid
                        plot_frequency = 10, &
                        pause_time = 1
real, dimension(nn,nn), target :: plate
real, dimension(nn-2,nn-2)    :: temp
real, pointer, dimension(:, :) :: n,e,s,w, inside
real, parameter          :: tolerance = 1.0e-3
character(len=20)        :: filename = "heat_data."

real    :: diff
integer :: i,j, niter, ios

open(unit=11, file="heat.gp", status="replace",&
      action="write", iostat=ios)
if (ios > 0) then
  print *, "Couldn't open heat.gp"
  stop
end if

call system("rm -f heat_data.*")

write(unit=11, fmt="(a)") &
  "set terminal x11", &
  "set pm3d", &
  "set palette", &
  "set ticslevel 0", &
  "set view 0,0"
write(unit=11, fmt="(a, i0)") &
  "pause_seconds=", pause_time

! Set up initial conditions
plate = 0
plate(1:nn,nn) = 1.0 ! boundary values
plate(1,1:nn) = (/ ( 1.0/nn*j, j = 1, nn ) /)

! Point to parts of the plate
inside => plate(2:nn-1,2:nn-1)
n => plate(1:nn-2,2:nn-1)
s => plate(3:nn,2:nn-1)
e => plate(2:nn-1,3:nn)

```

11-6 Plotting

```
w => plate(2:nn-1,1:nn-2)

! Iterate
niter = 0
do
  niter = niter + 1
  temp = (n + e + s + w)/4.0
  diff = maxval(abs(temp-inside))
  inside = temp
  if (modulo(niter, plot_frequency) == 0) then
    call print_data()
  end if
  if (diff < tolerance) exit
end do

if (modulo(niter, plot_frequency) /= 0) then
  call print_data()
end if

close (unit=11, status="keep")

call system("gnuplot -persist ""heat.gp""")

contains

subroutine print_data()
  write(unit=filename(11:), fmt="(i0)") niter
  open(file=trim(filename), &
        unit=22, status="replace", &
        action="write", iostat=ios)
  if (ios>0) then
    print *, "File open failed:", niter
    stop
  end if
  do i = 1,nn
    write (unit=22, fmt="(999f7.3)") &
      plate(1:nn,i)
  enddo
  close(unit=22, status="keep")
  write (unit=11, fmt="(a)") &
    "splot "" // trim(filename) // &
    "" matrix pt 0", "pause pause_seconds"
```

```
end subroutine print_data  
  
end program heat
```

A constant heat source is placed along one edge of an $nn \times nn$ grid and a linearly diminishing heat source is placed along an adjacent edge. The steady state condition of each point in the grid is found by iteratively replacing each value in the interior of the grid (the pointer variable `inside` is an alias of this portion of the plate) by the average of the points (n , e , s , w) around it. If the iteration is a multiple of `plot_frequency`, the values in the grid are placed in a file for plotting. `pause_seconds` indicates the number of seconds delay between plot displays. If the largest difference between an old value and a new value in the grid is less than the parameter `tolerance`, no more iterations are performed and the data for the last plot is generated, if it has not been generated already.

The system routine is called to execute the Gnuplot command file `heat.gp` generated by the program. Here is the content of the file generated in this case.

```
set terminal x11  
set pm3d  
set palette  
set ticslevel 0  
set view 0,0  
pause_seconds=1  
splot "heat_data.10" matrix pt 0  
pause pause_seconds  
splot "heat_data.20" matrix pt 0  
pause pause_seconds  
.  
.  
.  
splot "heat_data.150" matrix pt 0  
pause pause_seconds  
splot "heat_data.157" matrix pt 0  
pause pause_seconds
```

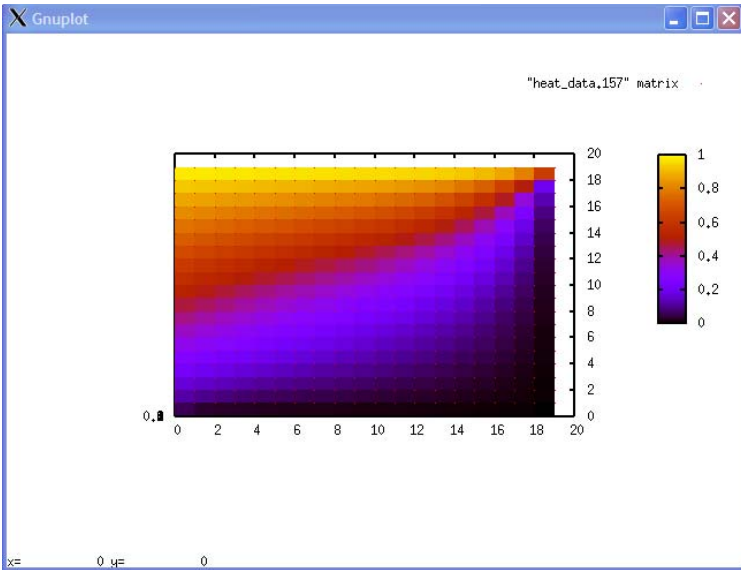
This sets the file format to `x11` and the style to a special three-dimensional format (`pm3d`). The view is from straight above the plot ($0,0$). It draws three-dimensional (projected on the two-dimensional screen, of course) plots (`splot`) for each of the data files generated by the Fortran program. `matrix` indi-

11-8 Plotting

cates that the data in the file is in matrix (two-dimensional array) format. `pt 0` indicates that the point style is 0. Because the command that executes this file is

```
gnuplot -persist heat.gp
```

the last plot remains displayed until closed. Here is the final plot.



Software License Agreement **A**

Read the terms and conditions of this license agreement carefully before installing the Software on your system.

By installing the Software you are accepting the terms of this Agreement between you and The Fortran Company. If you do not agree to these terms, promptly destroy all files and other materials related to the Software.

“Software” means the programs developed by The Fortran Company. The Fortran compiler, Eclipse, Photran, and other programs not developed by The Fortran Company are licensed under agreements with their developers.

The Fortran Company grants to you a nonexclusive license to use the Software with the following terms and conditions:

The Fortran Company retains title and ownership of the Software. This Agreement is a license only and is not a transfer of ownership of the Software.

The Software is copyrighted. You may copy the software provided that you include a copy of this license.

You may adapt and modify any source programs, but may not reverse engineer any object or executable files.

You may not sell, rent, or lease the software.

This license is effective until terminated by The Fortran Company. It will terminate automatically without notice if you fail to comply with any provision of this license. Upon termination, you must destroy all copies of the Software.

The failure of either party to enforce any rights granted under this Agreement or to take action against the other party in the event of any breach of this Agreement will not be deemed a waiver by that party as to subsequent enforcement of rights or subsequent action in the event of future breaches. If applicable statute or rule of law invalidates any provision of this Agreement, the remainder of the Agreement will remain in binding effect.

THE FORTRAN COMPANY MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A

PARTICULAR PURPOSE OR PERFORMANCE OR ACCURACY. THE FORTRAN COMPANY SHALL IN NO EVENT BE LIABLE TO THE LICENSEE FOR ANY DAMAGES (EITHER INCIDENTAL OR CONSEQUENTIAL), EXPENSE, CLAIM, LIABILITY, OR LOSS, WHETHER DIRECT OR INDIRECT, ARISING FROM THE USE OF THE SOFTWARE.