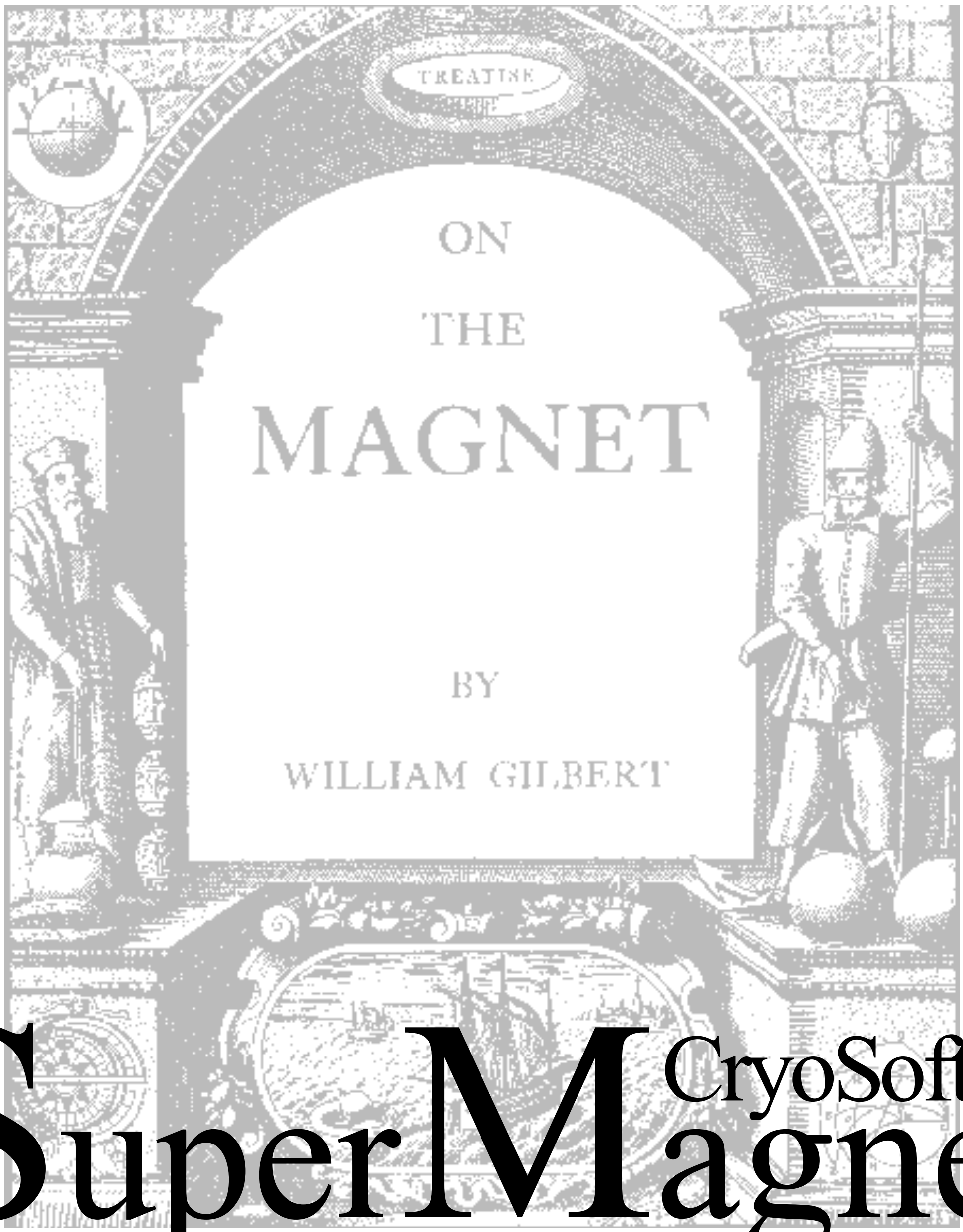
User’s Guide

Version 2.1a
November 2021



CryoSoft

SuperMagnet

Multitasking code manager

Contents

Roadmap

Before you start

This manual is the reference user's guide for `SUPERMAGNET`. It gives a user-end explanation on the installation and working of the code, case studies to familiarize with the preparation of the input, a complete list of input commands and parameters specifications, and a list of error messages. `SUPERMAGNET` is a manager application that launches and administers communication among other CryoSoft programs. We assume throughout the manual that you are familiar with the specific applications coupled by `SUPERMAGNET`, and in particular `THEA`, `FLOWER`, `POWER,` `HEATER`, and `MrX` for which we provide separate user's manuals.

How to use this manual

This manual is structured as follows:

- Chapter 1 gives a short description of the manager and communication functions of `SUPERMAGNET` and how the codes are coupled in practice;
- Chapter 2 gives basic information on the installation, explains how to start a `SUPERMAGNET` run on a UNIX workstation.
- Chapter 3 contains case studies that the reader should use to familiarize with the operation and features of the program.
- Chapter 4 contains additional information on the preparation of the input and the meaning of the input variables;
- Chapter 5 deals with troubleshooting and error messages;
- Chapter 6 gives the references and a general bibliography for documentation.

Beginners to `SUPERMAGNET` should read chapters 1, 2 and 3 in sequence. They will make occasional cross reference to chapters 4 and 5 for detailed information. Experienced users will use chapter 4 for daily operation.

CHAPTER 1

Introduction

What is SUPERMAGNET

SUPERMAGNET provides an answer to a basic need of superconducting magnet designers, namely to consider in his daily work various and different configurations of superconductors in a magnet, with disparate cooling modes and power supply connection, and for each of them a spectrum of operating conditions. Each of these situations is analyzed using basic tools and methods that are common to most configurations, but are sequenced differently according to the analysis to be performed in a specific magnet system, and for a specific operating condition.

The idea behind SUPERMAGNET reflects the designer's approach, namely to use existing tools, and join them in a customizable, flexible, and powerful environment for the analysis of thermo-hydraulic and electrical transients in superconducting magnetic systems, including its proximity cryogenics and power supply.

The CryoSoft suite of codes THEA (Thermal, Hydraulic and Electric Analysis of Superconducting Cables) [1], FLOWER (Hydraulic Network Simulation) [2], POWER (Electric Network Simulation of Magnetic Systems) [3], HEATER (Simulation of Heat Conduction) [4], and MrX[1] (Generic Data Exchange) provides a basis for well optimized and flexible tools for the analysis of specific issues in superconducting magnet systems. SUPERMAGNET is the manager application that launches two or more of the above codes, schedules their communication and terminates execution as appropriate[2]. The codes communicate through a data exchange mechanism (described later) that achieves the desired physical coupling and makes it possible to describe a series of processes such as:

- effect of helium expulsion during thermal transients on the proximity cryogenics;
- regulation of cryogen flow and valving conditions, subject to transient response in superconducting cables;
- evolution of the coil current during quench, including the effect of quench resistance, and coupling within segments of the same magnetic system;
- cooling of a coil with thermally coupled parallel channels;

and others.

[1] MrX is a generic data exchange application presently in conceptual design.
[2] In the process of programming the modifications necessary for this functionality in each of the basis codes, we have also minimized the impact on each of them, thus maintaining the full stand-alone capability.

To achieve the above result, SUPERMAGNET makes use of UNIX specific features, namely the possibility to launch and identify *child* processes from a *father* process that is running in the OS, and to launch and retrieve *signals* through the OS. Because of the technology chosen, SUPERMAGNET is a UNIX or UNIX-like specific applications, and only operates under UNIX, UNIX emulators, LINUX and Mac-OSX (native).

The mechanism by which SUPERMAGNET couples processes is described schematically in Fig. 1. SUPERMAGNET is the *father* process of an arbitrary number of *children* processes. This *family* of processes (father and children) shares direct read/write access of a series of files constituting the *shared family memory*.

At start-up, the father reads from a user defined input file the sequence of coupled codes, the children. For all children processes to be launched, the father starts the child process and provides the following information:

- the father process-ID (PID);
- the input file name for the child process;
- the file name for the shared family memory.

After being started, the child confirms that the process is launched and identifies itself, providing the child PID. At this point the child process starts the desired simulation based on the child input file provided, and stores the results as in a stand-alone run.

The child process is in general a time stepping algorithm, progressing a simulation from one time step to the next. At each time step, the child writes results to the shared family memory file, to be used by the father, and, eventually, by other children processes. It then notifies the father that results are available and waits for a receipt from the father.

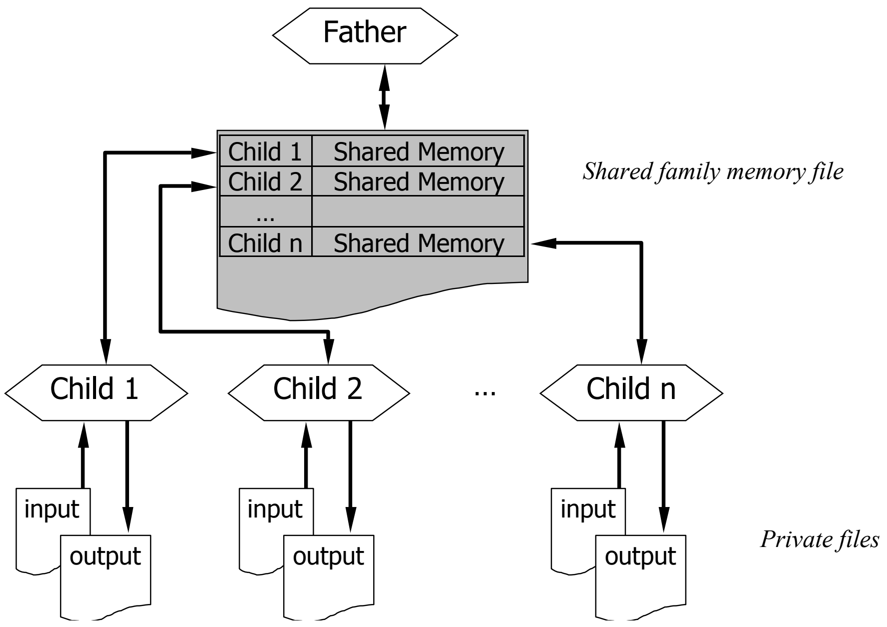


Figure 1. Schematic view of the family formed by SUPERMAGNET (father process) and an arbitrary number of children processes. Communication between processes takes place through a file containing the shared family memory. Each process accesses a single file record in read/write mode, and the father manages the exchange of variables by reading/writing multiple records. Input and output of each child process is on dedicated, private files to that process.

To proceed with the time stepping, in most common cases the child has to receive results from other processes. The results to be exchanged are written on the shared family memory file, upon which the father provides a ready status. The child then reads the results, notifies the father of the receipt and proceeds with the time stepping.

The handling of signals and the exchange of results is a bit more involved on the side of the father. At each notification of available results, the father first identifies the child providing results, and then reads and stores results to respond to subsequent requests from other children. At each demand of results, the father identifies the demanding child and interpolates the results at the demanded time. The results requested are then written on the shared family memory file for the child process to retrieve and proceed.

At the end of the simulation, as specified in the child input file, each child notifies father that the process is completed and waits for a receipt from the father. The father, in turn, at the receipt of a notification of end of simulation notifies the receipt to the child, and responds to all other processes until the maximum simulation time reached by the child who completed its run is reached. At this point all other processes are killed.

The mechanism described above is fully parametric, meaning that the number and nature of child processes managed by `SUPERMAGNET` is defined by the user. At the same time, however, the user should take care that there is no conflict in the use of files, and that the single simulation inputs are coherent and consistent for the desired coupled system simulation.

Details of the coupling procedure

The processes and coupling possibilities implemented in the present version of `SUPERMAGNET` are shown in the following table. The entries are intended as the quantities provided by a given code (row) to a coupled code (column). Future versions of `SUPERMAGNET` are planned to complete and extend the above table, and in particular to make available the coupling through the code `MrX` that is presently not available to the end user.

		THEA		FLOWER		POWER	HEATER	MrX
		Thermal	Hydraulic	Junction	Volume			
THEA	Thermal					Branch resistance	Line temperature	
	Hydraulic			Junction B.C.			Line temperature (and HTC)	
FLOWER	Junction		Hydraulic B.C.				Line temperature (and HTC)	
	Volume						Point temperature (and HTC)	
POWER		Cable current						
HEATER		Thermal heat flux	Hydraulic heat flux and wall temperature	Junction heat flux and wall temperature	Volume heat flux and wall temperature			
MrX								

The following sections contain a detailed description of the couplings implemented.

THEA-FLOWER coupling

Hydraulic - junction boundary conditions. A `THEA hydraulic` component can be coupled to a `FLOWER` hydraulic network, making the `THEA hydraulic` a part of the `FLOWER` network. This is achieved by setting:

- boundary conditions of type `external` in the `THEA hydraulic` (see the reference manual [1]). An `external` boundary conditions in `THEA` signals the need for coupling;
- a `junction` of type `external` in `FLOWER` (see the reference manual [2]) which provides a *phantom representation* of the `THEA hydraulic` component in the `FLOWER` network, and signals to `FLOWER` the presence of a coupled component . As for any other `junction`, this `external junction` connects two `volume`’s in the hydraulic network of `FLOWER`.

The coupling algorithm is the following:

- the boundary conditions at the two ends of a `THEA hydraulic` component is set equal to the pressure and temperature of two `volume`’s in the `external junction` in `FLOWER`;
- the mass and enthalpy fluxes in or out of the two same `volume`’s in `FLOWER` are augmented by those entering and exiting the coupled `hydraulic` in `THEA`.

The equations in `FLOWER` for the volume conservation of mass and energy, written in terms of the volume pressure and temperature, are modified as follows:

$$V\frac{\partial p}{\partial t}+\sum \dot{m}_i\left[c^2+\phi\left(h_i+\frac{v_i^2}{2}-h\right)\right]+\underline{\dot{m}_{THEA}\left[c^2+\phi\left(h_{THEA}+\frac{v_{THEA}^2}{2}-h\right)\right]}=\phi\dot{q} \qquad (1)$$

$$V\rho C_v\frac{\partial T}{\partial t}+\sum \dot{m}_i\left(\phi C_v T+h_i+\frac{v_i^2}{2}-h\right)+\underline{\dot{m}_{THEA}\left(\phi C_v T+h_{THEA}+\frac{v_{THEA}^2}{2}-h\right)}=\dot{q} \qquad (2)$$

where the modifications are the terms underlined in the above equations, and variables with the subscript *THEA* indicate the inlet or outlet conditions from the `THEA hydraulic`[3]. In `THEA` the modification corresponds to using the volume pressure and temperature to impose the `hydraulic` boundary conditions:

$$p_{boundary}=p_{FLOWER} \qquad (3)$$

$$T_{boundary}=T_{FLOWER} \qquad (4)$$

where the variable with subscript *boundary* indicates inlet or outlet conditions, and the variable with subscript *FLOWER* stands for the conditions in the volume connected to the specific hydraulic boundary considered. Note that the treatment of boundary conditions in `THEA` is otherwise standard, and depending on the direction of the flow either both *p* and *T*, or only *p* can be imposed.

[3] Note that the massflow is intended as positive when entering the `FLOWER` volume in question, i.e. when exiting the corresponding pipe end in `THEA`

THEA-POWER coupling

Cable - branch current. The total cable current used for simulation by THEA can be obtained by a simulation of an electric network with POWER. This is achieved by setting:

- the model for the current in the THEA simulation to `external` (see the reference manual [1]), which signals the external coupling;
- a `resistance branch` of type `external` in the input file of POWER . (see the reference manual [3]) that represents the THEA cable. This branch provides a *phantom representation* of the THEA cable in the POWER network, and signals to POWER the presence of a coupled component. As for any other `branch`, this `external branch` is listed in the connectivity of the network of POWER.

The coupling algorithm is the following:

- the cable current in THEA equal to the current in the `resistance branch` in POWER;
- the resistance of the same `branch` in POWER equals to the total resistance of the cable length in THEA.

To achieve the coupling, the total cable current in THEA is set to:

$$I_{cable} = I_{POWER} \tag{5}$$

where the variable with subscript *POWER* stands for the result of the POWER simulation in the branch identified. In POWER, the resistance of the branch is set to

$$R_{branch} = R_{THEA} \tag{6}$$

where the variable with subscript *THEA* stands for the total cable resistance computed by THEA.

THEA-HEATER coupling

Thermal heat flux - line temperature. Thermal processes in THEA can be coupled to a solution of heat conduction in HEATER. More specifically, it is possible to couple the temperature of a THEA `thermal` component into a HEATER mesh through the boundary condition along a `line` of the mesh. This is achieved by setting:

- the model for the heat flux in the coupled THEA component (`thermal`) to `external` (see the reference manual [1]), which signals the external coupling;
- a `line` with a `heat` boundary condition of type `external` in the mesh used by HEATER (see the reference manual [4]). This `line` represents the THEA component in the HEATER mesh, and for consistency reasons should be of identical length as the length of the component itself. Such a line signals to HEATER the presence of a coupling.

The coupling algorithm is the following:

- the linear heat flux in the nodes of the THEA `thermal` component is equal to the heat load in the nodes of the `line` in the HEATER mesh;
- the temperature in the nodes of the `line` in the HEATER mesh is set equal to the temperature of the THEA `thermal` component.

Although the lengths of the `HEATER line` and of the `THEA` component should be equal, the nodes in the two objects do not need to be coincident. At each coupling step the values are interpolated in space to provide the desired values at the desire locations. The interpolation is performed using piecewise linear functions.

In practice, the coupling is achieved as follows. The linear heat flux in node *i* of the `THEA` component, indicated as $\dot{q}'_{THEA}\left(x_i\right)$ is set to:

$$\dot{q}'_{THEA}\left(x_i\right)=\dot{q}'_{HEATER}\left(s_{j-1}\right)+\frac{\dot{q}'_{HEATER}\left(s_j\right)-\dot{q}'_{HEATER}\left(s_{j-1}\right)}{s_j-s_{j-1}}\left(x_i-s_{j-1}\right) \tag{7}$$

where *s* indicates the curvilinear coordinate along the `HEATER line` at the points *j*-1 and *j* bracketing x_i (i.e. $s_{j-1} \le x_i \le s_j$). The heat flux along the `HEATER line` $\dot{q}'_{HEATER}\left(s_j\right)$ is obtained by a projection of the nodal heat resultant at each node of the line. In `HEATER`, the temperature at each node of a `line`, indicated as $T_{HEATER}\left(s_j\right)$ is obtained by interpolation of the `THEA` temperatures:

$$T_{HEATER}\left(s_j\right)=T_{THEA}\left(x_{i-1}\right)+\frac{T_{THEA}\left(x_i\right)-T_{THEA}\left(x_{i-1}\right)}{x_i-x_{i-1}}\left(s_j-x_{i-1}\right) \tag{8}.$$

Hydraulic heat flux - line temperature. Hydraulic processes in `THEA` can be coupled to a solution of heat conduction in `HEATER`. There are two ways that a `THEA hydraulic` component can couple into a `HEATER` mesh, namely through direct coupling of temperature and heat flux, as described earlier in the case of a `THEA thermal` component, or through convective coupling. The first possibility is in essence identical to the procedure described earlier. For the second one, convective coupling, this achieved by setting:

- the model for the convective heat flux in the coupled `THEA` component (`hydraulic`) to `external` (see the reference manual [1]), which signals the external coupling;
- a `line` with a `convection` boundary condition of type `external` in the mesh used by `HEATER` (see the reference manual [4]). This `line` represents the `THEA` component in the `HEATER` mesh, and for consistency reasons should be of identical length as the length of the component itself. Such a line signals to `HEATER` the presence of a coupling.

The coupling algorithm is the following:

- the wall temperature (heat convection) in the nodes of the `THEA hydraulic` component is equal to the temperature in the nodes of the `line` in the `HEATER` mesh;
- the temperature and heat transfer coefficient in the nodes of the `line` in the `HEATER` mesh is set equal to those of the `THEA hydraulic` component.

Although the lengths of the `HEATER line` and of the `THEA` component should be equal, the nodes in the two objects do not need to be coincident. At each coupling step the values are interpolated in space to provide the desired values at the desire locations. The interpolation is performed using piecewise linear functions.

In practice, the coupling is achieved as follows. The linear heat flux in node *i* of the `THEA` component, indicated as $\dot{q}'_{THEA}\left(x_i\right)$ is computed using the wall convection temperature

$T_{HEATER}(s)$ and the heat transfer coefficient of the flow $HTC_{THEA}(x_i)$. This requires the calculation of the wall temperature $T_{THEA}^{wall}(x_i)$ for THEA at the location x_i:

$$T_{THEA}^{wall}(x_i) = T_{HEATER}(s_{j-1}) + \frac{T_{HEATER}(s_j) - T_{HEATER}(s_{j-1})}{s_j - s_{j-1}}(x_i - s_{j-1}) \quad (9)$$

where s indicates the curvilinear coordinate along the HEATER line at the points j-1 and j bracketing x_i (i.e. $s_{j-1} \le x_i \le s_j$). The heat flux in node i of the THEA component is then obtained as follows:

$$\dot{q}'_{THEA}(x_i) = HTC_{THEA}(x_i)\left(T_{THEA}^{wall}(x_i) - T_{THEA}(x_i)\right) \quad (10)$$

The heat flux along the HEATER line $\dot{q}'_{HEATER}(s_j)$ is obtained in a similar way, whereby the heat transfer coefficient is the one computed in the THEA hydraulic component. In this case wall temperature $T_{HEATER}^{wall}(s_j)$ and the heat transfer coefficient $HTC_{HEATER}(s_j)$ for HEATER at the location s_j are obtained by interpolation of the THEA temperatures and heat transfer coefficient:

$$T_{HEATER}^{wall}(s_j) = T_{THEA}(x_{i-1}) + \frac{T_{THEA}(x_i) - T_{THEA}(x_{i-1})}{x_i - x_{i-1}}(s_j - x_{i-1}) \quad (11)$$

$$HTC_{HEATER}(s_j) = HTC_{THEA}(x_{i-1}) + \frac{HTC_{THEA}(x_i) - HTC_{THEA}(x_{i-1})}{x_i - x_{i-1}}(s_j - x_{i-1}) \quad (12).$$

FLOWER-HEATER coupling

Junction heat flux - line temperature. This is a way to couple flow processes in FLOWER to a solution of heat conduction in HEATER. As for THEA (see earlier), there are two ways that a FLOWER junction component can couple into a HEATER mesh, namely through direct coupling of temperature and heat flux, or through convective coupling. Both possibilities are obtained by coupling the heat source in the FLOWER junction to the boundary conditions in a HEATER line.

Direct coupling of temperature and heat flux is achieved by setting:

- the model for the heat flux in the coupled FLOWER junction to external (see the reference manual [2]), which signals the external coupling;
- a line with a heat boundary condition of type external in the mesh used by HEATER (see the reference manual [4]). This line represents the FLOWER junction in the HEATER mesh, and for consistency reasons should be of identical length as the length of the component itself. Such a line signals to HEATER the presence of a coupling.

The coupling algorithm is the following:

- the heat source in the nodes of the FLOWER junction is equal to the interpolated values along the line in HEATER;

- the temperature in the nodes of the `line` in `HEATER` is equal to the interpolated values along the `FLOWER junction`.

Convection coupling is achieved by setting:

- the model for the convective heat flux in the coupled `FLOWER junction` to `external` (see the reference manual [2]), which signals the external coupling;
- a `line` with a `convection` boundary condition of type `external` in the mesh used by `HEATER` (see the reference manual [4]). This `line` represents the `FLOWER` component in the `HEATER` mesh, and for consistency reasons should be of identical length as the length of the component itself. Such a line signals to `HEATER` the presence of a coupling.

The coupling algorithm is the following:

- the wall temperature (heat convection) in the nodes of the `FLOWER junction` component is equal to the temperature in the nodes of the `line` in the `HEATER` mesh;
- the temperature and heat transfer coefficient in the nodes of the `line` in the `HEATER` mesh is set equal to those of the `FLOWER junction` component.

The lengths of the `HEATER line` and of the `FLOWER` junction should be equal, but the nodes in the two objects do not need to be coincident. At each coupling step the values are interpolated in space to provide the desired values at the desire locations. The interpolation is performed using piecewise linear functions. (see earlier description of the `THEA - HEATER` coupling for details).

Volume heat - point temperature. A second way to couple thermal processes in `FLOWER` to a solution of heat conduction in `HEATER` is to link local heating from a `HEATER point` to a `FLOWER volume`. Also in this case there are two ways that a `FLOWER volume` component can couple into a `HEATER` mesh, namely through direct coupling of temperature and heat flux, or through convective coupling.

Direct coupling of temperature and heat flux is achieved by setting:

- the model for the heat flux in the coupled `FLOWER` volume to `external` (see the reference manual [2]), which signals the external coupling;
- a `point` of type `external` in the mesh used by `HEATER` (see the reference manual [4]). This `point` represents the `FLOWER` volume in the `HEATER` mesh, and signals to `HEATER` the presence of a coupling.

The coupling algorithm is the following:

- the heat source in the `FLOWER volume` is equal to the heat flux in the `point` in `HEATER`;
- the temperature in the node identified as the `point` in `HEATER` is equal to the value in the `FLOWER volume`.

Convection coupling is achieved by setting:

- the model for the convective heat flux in the coupled `FLOWER volume` to `external` (see the reference manual [2]), which signals the external coupling;
- a `point` with a `convection` boundary condition of type `external` in the mesh used by `HEATER` (see the reference manual [4]). This `point` represents the `FLOWER` component in the `HEATER` mesh, and signals to `HEATER` the presence of a coupling.

The coupling algorithm is the following:

- the wall temperature (heat convection) in the `FLOWER volume` is equal to the temperature in the `point` in the `HEATER` mesh;
- the temperature and heat transfer coefficient in the `point` in the `HEATER` mesh is set equal to those of the `FLOWER volume`.

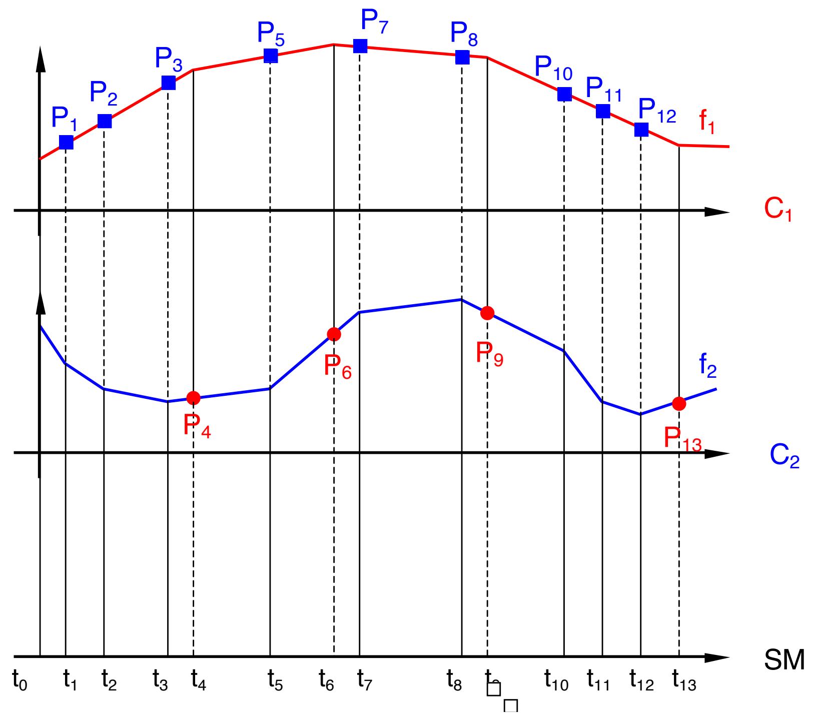
MrX coupling

`MrX` provides a mechanism for generic data exchange *from* or *to* one of the codes coupled by `SUPERMAGNET`. `MrX` does not perform calculations, but can substitute to any of the coupling ends detailed earlier, to interface a data exchange to other codes that are running concurrently to `SUPERMAGNET`. Launched by `SUPERMAGNET`, `MrX` receives data from one of the CryoSoft codes, running a coupled simulation, and reformats it into an ASCII file with a given protocol that can be read and used by a concurrent simulation code. The concurrent simulation is then expected to produce coupling results that `MrX` reads and converts as appropriate to provide it to the coupled CryoSoft code.

Note At the time of writing of this manual, `MrX` is in conceptual design and testing, and is not yet available to the end user.

Staggered time steps and synchronization

In the general case, each child process of a global run advances the time integration on its own time step, possibly adapted following the evolution of the transient. The exchange of information between children processes needs hence to be synchronized so that each process receives data to advance its time integration at the required time. This is done by `SUPERMAGNET` interpolating linearly between the data made available by each child process. We show schematically below how this is achieved by taking the case of two children processes, C_1 and C_2, starting at the same initial time t_0 and advancing from there on their independent, and variable time steps.



The two processes produce data that needs to be exchanged for the coupling, i.e. the two functions drawn by points f_1 and f_2. At each time reached by process C_1, the process itself needs the value of f_2 to continue the time stepping. Similarly, process C_2 requires the value of f_1 at each time reached to continue with the stepping algorithm.

With reference to the scheme above, this is achieved by the following sequence of operations:

- process C_1 performs a time step from time t_0 to time t_4 and requests the value of f_2 at time t_4;
- process C_2 performs a time step from time t_0 to time t_1 and requests the value of f_1 at time t_1;
- `SUPERMAGNET` interpolates (linearly) the value of f_1 at time t_1 (point P_1);
- process C_2 performs a time step from time t_1 to time t_2 and requests the value of f_1 at time t_2;
- `SUPERMAGNET` interpolates (linearly) the value of f_1 at time t_2 (point P_2);
- process C_2 performs a time step from time t_2 to time t_3 and requests the value of f_1 at time t_3;
- `SUPERMAGNET` interpolates (linearly) the value of f_1 at time t_3 (point P_3);
- process C_2 performs a time step from time t_3 to time t_5 and requests the value of f_1 at time t_5;
- `SUPERMAGNET` interpolates (linearly) the value of f_2 at time t_4 (point P_4);
- process C_1 performs a time step from time t_4 to time t_6 and requests the value of f_2 at time t_6;
- `SUPERMAGNET` interpolates (linearly) the value of f_1 at time t_5 (point P_5);
- process C_2 performs a time step from time t_5 to time t_7 and requests the value of f_1 at time t_7;
- …

In practice, each process is advancing by one step at a time, provide the required update on the data for coupling and wait for new data. `SUPERMAGNET` takes care of putting the processes on hold, interpolating results, and releasing data to processes to make them advance. We finally remark that the scheme above does not require iterations, which simplifies greatly the interactions among codes, as well as the structure of each coupled code. As remarked in the next section, however, this demands care on the side of the user to avoid coupling instabilities.

Coupling instabilities

In accordance with the description above, the physical coupling of the parallel simulations takes place at each time step, and is fully explicit. This is most practical as it reduces the modifications of the single coupled codes to a minimum, and corresponds to the simplest staggered partitioned field analysis described by Park and Felippa [5]. The drawback is that this procedure can become numerically unstable if the exchange of information between the domains takes place on a time scale much longer than the characteristic times of each domain. Instability of this type, presently not detected, can be cured by the user decreasing the time step of the single simulations coupled.

CHAPTER 2

Installing and Running SUPERMAGNET

Platforms

SUPERMAGNET is provided as a package developed for running under UNIX or UNIX-like (e.g. Linux) operating system. The reason is that it requires messaging among processes as standard in the above environment. At the time when this manual is written, the platform where SUPERMAGNET is developed is

- Macintosh running MacOS-X (10.10.5 and higher) under XQuartz,(2.7.8) gcc (5.1) with gfortran.

At different time of the development and production, the code has been installed and tested on the following platforms:

- Mac-OS X (10.2 and higher) operating system;
- GNU/Linux operating system (most distributions).
- INTEL PC's running RedHat Linux OS;
- IBM-RISC workstations running the AIX-V4 operating system and later;
- SUN-SPARC workstation running the Solaris OS operating system;
- DEC-ALPHA workstation running the OSF-1 operating system;
- HP workstations running HP-UX OS;
- Windows-2000 and Windows-XP operating system, with an installed CYGWIN environment (the reference version tested is CYGWIN 1.5.24-2).

Although UNIX obeys strict standards, the architecture of the operating and file system may vary from vendor to vendor. It is therefore possible that porting may require minor adaption of code and libraries. Contact us for advice.

In the following sections we assume here that you are running under a UNIX or UNIX-like operating system, and that you are familiar with UNIX commands, directory and file handling. Contact your system administrator for matters regarding UNIX commands and file system.

Although versions of SUPERMAGNET have been ported to PC's running the Windows OS, at the time when this manual is written this is not a platform directly supported and part of the instructions provided below (i.e. how to run and post-process a case) may not be directly applicable.

Installation

SUPERMAGNET is one of the CryoSoft family of programs. You will have therefore received the CryoSoft package containing SUPERMAGNET either as a tar-ball or in pre-installed form. Verify in the CryoSoft installation manual [6] the procedure to be followed for the proper installation of the complete package. The executable code is in the directory `~/CryoSoft/bin/`. You will find the example inputs command files in the directory `~/CryoSoft/xample/supermagnet/code_x.x/` (the symbol `~/` stands for your home directory, `x.x` for the version you received).

How to run SUPERMAGNET

Start-up To run SUPERMAGNET you will need to launch the executable code. In the standard installation on a UNIX system described above SUPERMAGNET is launched typing the command:

```
~/CryoSoft/bin/supermagnet [-i InputFile] [-v/-s] [-h]
```

Note that command line options are not mandatory (enclosed in brackets, following UNIX documentation standard). The meaning of the options is the following:

`-i, --input`	use `InputFile` to parse the input for the run
`-v, --verbose`	print simulation progress on stdout (default)
`-s, --silent`	no output to stdout
`-h, --help`	print a help message

Once launched, the program decodes the options, if any are given, and checks for the specific operation mode requested. If no input file is provided as an option, then the program prompts the user for the input file name. SUPERMAGNET reads the run definition from an ASCII file whose structure and content are described in detail in Chapter 4 of this manual. Examples of input files are given in Chapter 3. At this time you will enter the name of a file containing the input for the case to be run (e.g. `file.input`):

```
Enter input file name
file.input
```

SUPERMAGNET then parses the input file, checks consistency, configures the case and starts execution. This begins by launching all children processes, in the same sequence as they are defined in the input file. Note that the children processes are usually distributed among the available processors in a multi-core CPU, which takes natural advantage of parallelism.

Each child process is assigned an input file (also defined in the SUPERMAGNET input), which is parsed to define the specific simulation. Upon successful data input, all children start execution, set initial conditions and begin the time integration. This is visible through the usual status message as time advances. Because all children processes run concurrently, the messages are interleaved, e.g. in the case of a `THEA - HEATER` coupled simulation:

```
....
HEATER Time :    5.819E-04   Step :  1.946E-04   Time/Tend :   0.00006
THEA   Time :    7.589E-04   Step :  2.563E-04   Time/Tend :   0.00008
HEATER Time :    8.738E-04   Step :  2.919E-04   Time/Tend :   0.00009
THEA   Time :    1.143E-03   Step :  3.844E-04   Time/Tend :   0.00011
....
```

As should be clear from the introduction, SUPERMAGNET only performs supervision functions, and has, as such, no direct output or result. As a consequence, SUPERMAGNET does not monitor advancement in time. The processes continue to execute until the simulation is completed by each process, or a `TERMINATE` signal is launched by SUPERMAGNET in case of an error. Messages of this type will be printed in case of normal end:

```
....
HEATER Time :    1.000E+01   Step :  1.888E-02   Time/Tend :   1.00000
HEATER Total Cpu [s]:   0.301997006
THEA   Time :    1.000E+01   Step :  3.137E-03   Time/Tend :   1.00000
THEA Total Cpu [s]:    4.78279400
```

In case of abort from one of the processes (e.g. input file parsing error), and with the present version of the CryoSoft codes, there may be situations where one or more processes remain hanging. This is evident as a stall of the messages from time advancement. In this case the user can kill the SUPERMAGNET run with a `control-C` (UNIX kill command) which in most systems will kill all processes, and remove the temporary files created for data exchange. It is good practice to check that no hanging processes are left (using the UNIX ps command), and if required remove the temporary files from the directory (files with ending `.PName`, where `PName` is the name of the process as defined in the input file of SUPERMAGNET).

Finally, results are provided by each of the simulation codes running under the management of SUPERMAGNET, and are accessed using the standard post-processing facilities of each code.

Restart It is possible to profit from the restart features in the CryoSoft codes, launching a simulation that restarts from the last time stored by each single code. This process requires that all processes managed by SUPERMAGNET have stored their end time, and that the end time is the same. A restart procedure is seen by SUPERMAGNET as a normal run (previous section), but the input files provided to each single process trigger a restart.

CHAPTER 3

Case Studies

As discussed in Chapter 2, `SUPERMAGNET` requires an input file that defines the family of processes to start and couple. We refer to this file as the *input file*. Each child process belonging to the family, in turn, also requires an appropriate *input file* to define the model, the boundary and operating conditions, and the simulation parameters. Indeed, each child process runs as if it performed a stand-alone simulation, and we refer to the manuals of each of the codes coupled by `SUPERMAGNET` for details on these input files. It is the responsibility of the user to insure that the input files of each child process are coherent.

Post-processing of the results from each child process is performed using the specific post-processor of the code coupled (e.g., `THEAPOST` for `THEA`, `FLOWERPOST` for `FLOWER`, etc.). Post processing the results of a simulation also requires an input file with a sequence of commands that select results, print and plot them. We refer to this file as the *post-processing command file*.

In this Chapter we give examples of input files and post-processing command files to deal with practical modeling situations. The case studies given here are intended to guide the user from the formulation of a problem to its modeling, the creation of the input file for the case, running the case, and finally the generation of the results. For obvious reasons, they are of limited complexity and are intended as examples to illustrate minimum capability of the program. More complex situations can obviously be modeled, taking the following case studies as starting points and evolving or combining them. Refer to Chapter 2 on how to run the examples described here with `SUPERMAGNET` and to the specific manuals of the coupled codes on how to generate results and plots with the appropriate post-processor.

Note All input files for the case studies discussed in this manual are provided with the standard installation. They are located in the directory:

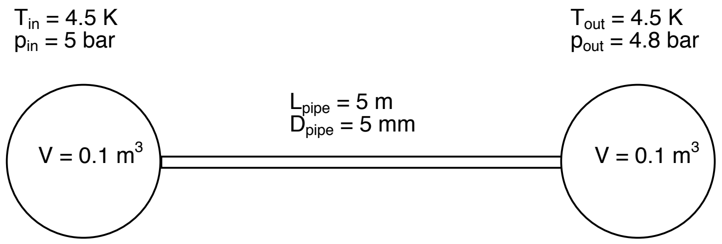
```
~/CryoSoft/xample/supermagnet/code_x.x
```

where `x.x` stands for the version you received. In the following sections we use the `Courier` font to reproduce the content of those input files, while text in *italic* indicates our comments to the input.

Pressure oscillations in a pipe connecting two volumes

Physical definition of the problem

We consider a pipe of 5 mm diameter and 5 m length filled with stagnant helium, initially at 4.5 k and 5 bar. The pipe is connected at the two extremities to two large volumes, each 0.1 m^3. The initial temperature of the volume is 4.5 K, while the pressure is different, 5 bar at one end and 4.8 bar at the other end. The helium is expected to flows through the pipe from the high pressure volume to the low pressure volume. We assume for this demonstration that the friction factor of the pipe is very small (inviscid flow). The situation is shown schematically below.



The pipe flow is simulated using an hydraulic component in THEA, while the volumes are simulated using FLOWER. The coupling takes place between the volumes and the inlet/oulet sections of the pipe.

Input file for the THEA simulation

The model requires the definition of a single hydraulic component, with inlet and outlet boundary conditions set to `external` to achieve the coupling. The input file for the THEA run is shown below. Refer to [1] for details on the input syntax.

thea.input

```
Begin Model

  ModelName             'test of coupling thea-flower'

  Length                5.0
  CurrentModel          none
  MagneticFieldModel    none
  StrainModel           none

end

Begin Hydraulics

   Components           1
   Fluid                Helium
   Model                constant
   Area                 19.6e-6
   Dh                   5.0e-3

   fModel               constant
   frictionfactor      1.0e-9
   hModel               DB

   InitialCondition     constant
   TInitial             4.5
   pInitial             5.0e5
   mdotInitial          0.0

   QModel               none
```

```
;
; the boundary conditions for the hydraulic are of type reservoir, as the connection
; to FLOWER happens through two volumes. The boundary conditions are imposed as
; external, triggering the connection to a coupled simulation.
;
   BoundaryType         reservoir reservoir
   BoundaryConditions   external  external

end

Begin Simulation

  MeshType              uniform
  NrElements            100
  ElementOrder          1
  ElementNodes          2

  StartTime             0.0
  EndTime               1.0
  OutputStep            0.01

  TimeMethod            EulerBackward
  MinimumStep           1.0e-6
;
; Note that the maximum step should be kept sufficiently small to avoid coupling
; instabilities and achieve accurate simulation results. The time steps of the THEA
; and FLOWER simulations do not need to be the same.
;
  MaximumStep           5.0e-3
  StepEstimate          smooth
  ErrorEstimate         change
  ErrorControl          on
  Tolerance             1.0e-3

  LogFile               thea.log
  StorageFile           thea.store

end
```

Input file for the FLOWER simulation The model for `FLOWER` consists of the two large inlet and outlet volumes, and a junction that phantoms the `THEA hydraulic` and specifies the link between the two volumes. The input file for the `FLOWER` run is shown below. Refer to [2] for details on the input syntax.

`flower.input`

```
Begin Simulation
   title          'test of coupling flower-thea'

   Volumes        2
   Junctions      1

   StartTime      0.0
   EndTime        1.0
   OutputStep     5.e-2
;
; Note that the maximum step should be kept sufficiently small to avoid coupling
; instabilities and achieve accurate simulation results. The time steps of the THEA
; and FLOWER simulations do not need to be the same.
;
```

```
   TimeMethod     EulerBackward
   MinimumStep    1.0e-6
   MaximumStep    2.0e-3
   StepEstimate   smooth
   ErrorEstimate  change
   ErrorControl   none
   Tolerance      1.0e-3

   StorageFile    flower.store
   LogFile        flower.log

End

Begin Volume 1  ; inlet volume node
   V 1.0e-1  P 5e5   T 4.5
End

Begin Volume 2  ; outlet volume node
   V 1.0e-1  P 4.8e5   T 4.5
End

;
; This is the phantom junction, of type external, that mimics the THEA hydraulic
; and achieves the coupling between inlet and outlet of the pipe and the two volumes
; defined above
;

Begin Junction 1
   type external
   connection 1 2
End
```

Input file for SUPERMAGNET The two simulations above are launched and supervised by `SUPERMAGNET`., whose input is reported below. The input file is based on a standard installation, and the use of the standard version of `THEA` and `FLOWER`. Two child processes are defined, and they are linked by associating the first `hydraulic` of the `THEA` model to the first `junction` of the `FLOWER` model.

`SM.input`

```
; SUPERMAGNET input file for the THEA-FLOWER demo coupling of inviscid pipe flow
; between two large volumes (oscillating system)

Begin Child

; The first child defined is the FLOWER run. Below is the name of the child, used later
; to define the coupling indices between children processes
   Name         F1

; The type of process is "flower" (a FLOWER run)
   ProcessType  flower

; The input for this run is in the file flower.input
   Inputfile    flower.input

End

Begin Child
```

; The second child defined is the THEA run. Below is the name of the child, used later
; to define the coupling indices between children processes
```
   Name         T1
```

; The type of process is "thea" (a THEA run)
```
   ProcessType  thea
```

; The input for this run is in the file thea.input
```
   Inputfile    thea.input

End
```

; Below is the definition of which specific part of the two children processes are
; coupled. In particular, the first hydraulic component of the THEA model corresponds
; to the first junction in the FLOWER model. The numbering refers to the order of
; definition in the models for each child (can be implicit)

```
Begin Connection
;                THEA            FLOWER
      Children   T1              F1           ; children ID's
      Link       hboundary 1     jboundary 1  ; component ID's
End
```

Post-processing command files Post-processing of the results from the coupled simulation is done using the specific post-processors of each child process, in this case `THEAPOST` and `FLOWERPOST`. Following is an example of the sequence of commands necessary to generate plots using the post-processor `THEAPOST`. Refer to [1] for details on the input syntax.

thea.post

```
StorageFile    thea.store
PostScriptFile thea.ps
OutputFile     thea.out

set plotsperpage 4

select x 0 2.5 5.0
plot pressure hydraulic 1
plot temperature hydraulic 1
plot velocity hydraulic 1
plot massflow hydraulic 1

select time 0.00 0.75 1.50 2.25 3.00
plot pressure hydraulic 1
plot temperature hydraulic 1
plot velocity hydraulic 1
plot massflow hydraulic 1

select time 6.25 7.00 7.75 8.50 9.25
plot pressure hydraulic 1
plot temperature hydraulic 1
plot velocity hydraulic 1
plot massflow hydraulic 1

stop
```

Similar to above, we report below an example of the sequence of commands necessary to generate plots using the post-processor FLOWERPOST. Refer to [2] for details on the input syntax.

flower.post

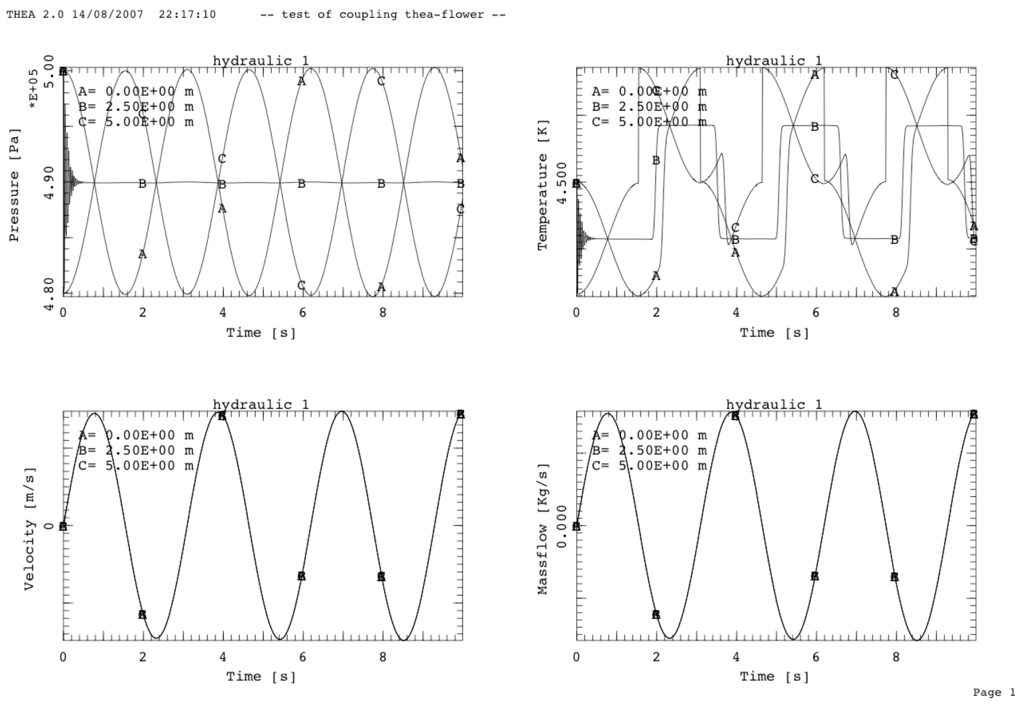
```
StorageFile    flower.store
PostScriptFile flower.ps
OutputFile     flower.out

set plotsperpage 4

plot temperature volume 1
plot temperature volume 2

plot pressure volume 1
plot pressure volume 2

stop
```

Results Two files are generated running the post-processors THEAPOST and FLOWERPOST with the commands described above: the PostScript output `thea.ps`, containing the plot for the THEA run, and PostScript output `flower.ps`, for the FLOWER run.

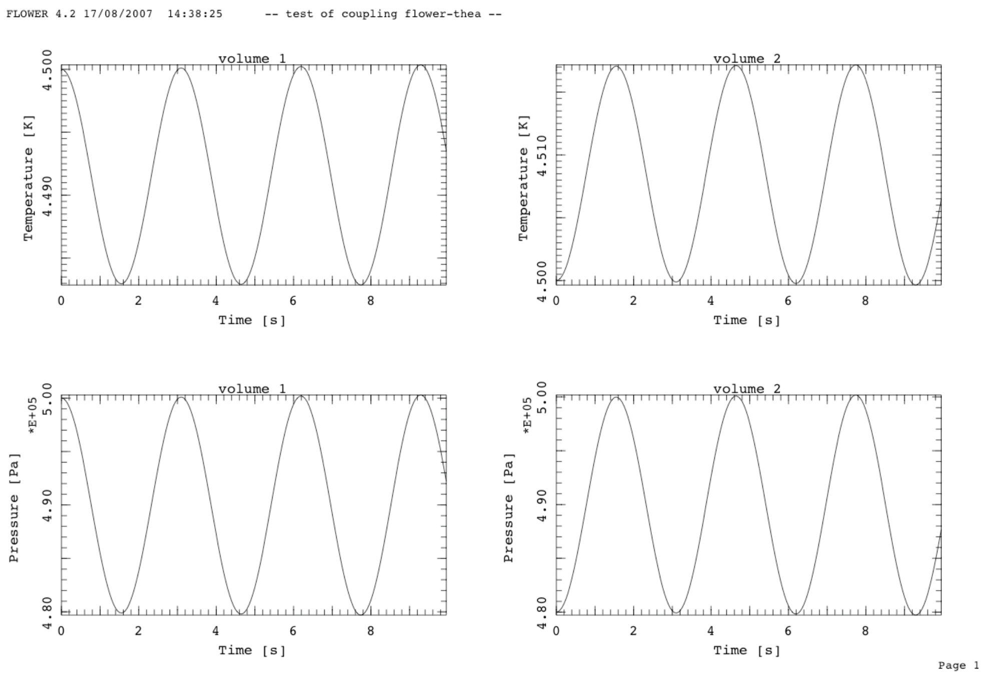
Note You will need a PostScript viewer to look at the plots in the PostScript file. The standard viewer, usually installed on UNIX systems, is `gs`. Try to launch the viewer with the commands:

```
gs thea.ps
gs flower.ps
```

The plots below show the first page in the PostScript output `thea.ps`. As requested in the commands file, the first four plots are the pressure, temperature, velocity and massflow waveforms at three selected positions along the pipe, inlet, middle and outlet. Note the oscillation of the pressure at the two ends, corresponding to an oscillating flow in the pipe. With negligible friction, as considered here, the system behaves as an harmonic oscillator.

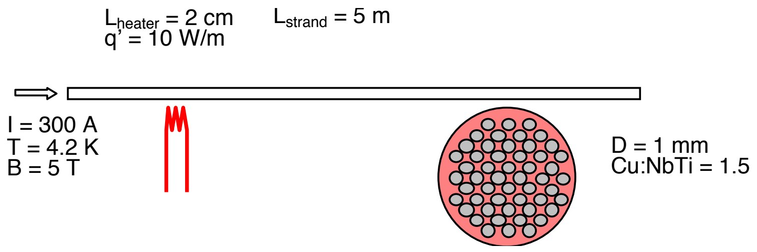


The corresponding output from the post-processing of the `FLOWER` run is shown below, which is the PostScript output `flower.ps`. In this case we requested the temperature and pressure of the two volumes. These oscillate, as observed from the previous plot. In particular the pressures in the two volumes 1 and 2 are equal to the inlet and outlet conditions of the pipe, as desired.

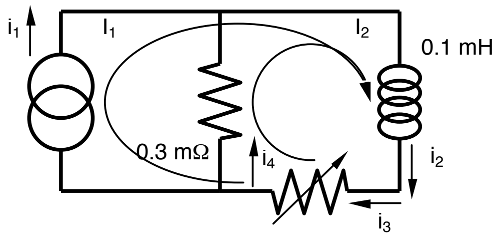


Coil quench simulation

Physical definition of the problem This case study deals with the calculation of the evolution of temperature and current in a quenching coil wound with a NbTi strand that we assume to be perfectly adiabatic. The total strand length in the coil is 5 m and is operated at 4.2 K, 300 A in a background field of 5 T. The strand itself has a diameter of 1 mm, is composed of Copper, with RRR of 100, and standard NbTi in a Cu:NbTi ratio of 1.5:1. The V-I resistive transition at the critical current is modeled using the power law approximation with a reference electric field of 1 µV/cm (10^{-4} V/m) and an exponent n of 20. The quench is initiated by a short heating pulse of 10 W/m with 1 ms duration and deposited over 2 cm length located at 1 m from one coil end,corresponding to a total energy deposited of 0.2 mJ. Below is a schematic of the strand developed length.



The coil is connected to a power supply that provides a constant current, and is protected by a resistance of 0.3 mH connected in parallel to the coil. The coil itself is assumed to have an inductance of 0.1 mH, while the resistance follows from the temperature and normal zone evolution during the quench. A schematic representation of the electrical circuit is shown below. The variable resistance branch represents the coupling to the resistance obtained from the solution of the coil quench.



Below we demonstrate how to run a coupled simulation followed by a restart run.

Input file for THEA For this case we define a single thermal component, consisting of the two materials of the strand: Cu and NbTi. The total current in the cable is set to `external` to achieve the coupling. The input file for the `THEA` run is shown below. The simulation runs up to a time of 100 ms. Refer to [1] for details on the input syntax.

`thea.input`

```
Begin Model

  ModelName             'test coupling thea-power'

  Length                5.0
;
; the current of the cable modelled by THEA is imposed as being external, triggering
; the connection to a coupled simulation, with the cable current computed by POWER
;
  CurrentModel          external
  InitialCurrent        300.0

  MagneticFieldModel    constant
  MagneticFieldSS       5.0 5.0
  StrainModel           none

end

Begin Thermals

;
; The thermal models a quenching NbTi-Cu strands
;
   Components                 1

   Model                      constant

   NrMaterials                2
   Materials                  Cu            Nb-Ti

   Area                       0.4712e-6     0.3142e-6
   RRR                        100.0         0.0
   E0                         1.0e-4
   nPower                     20

   QModel                     window
   Q                          100.0
   Q_Tau                      1.0e-3
   Q_XBegin                   0.9
   Q_XEnd                     1.1

   InitialCondition           constant
   TInitial                   4.2

   BoundaryType               heat          heat
   BoundaryConditions         constant      constant
   qBoundary                  0.0           0.0

end

Begin Simulation

  MeshType                    uniform
  NrElements                  200
  ElementOrder                1
  ElementNodes                2

  StartTime                   0.0
  EndTime                   100.0e-3
  OutputStep                  1.0e-3

;
; THEA will adapt the time step to the evolution of the temperature in the quenching
; strand between the minimum and maximum allowed steps. Note that the maximum step
; should be kept sufficiently small to avoid coupling instabilities and achieve
; accurate simulation results. The time steps of the THEA  and POWER simulations
; do not need to be the same.
;
  TimeMethod                  CrankNicolson
  MinimumStep                 1.0e-6
  MaximumStep               100.0e-3
  StepEstimate              smooth
  ErrorEstimate             change
  ErrorControl              on
  Tolerance                 1.0e-2

  LogFile                   thea.log
  StorageFile               thea.store

end
```

Input file for the POWER simulation The model for `POWER` consists of the four current branches connected in two meshes. In particular, the third branch phantoms the `THEA thermal` and provides the coupling to the electrical circuit. Also in this case the simulation runs up to a time of 100 ms, consistent with the input of `THEA` (see above). The input file for the `POWER` run is shown below. Refer to [3] for details on the input syntax.

`power.input`

```
Begin Simulation

   Title       'test coupling power-thea'

   StartTime     0.0
   EndTime     100.0e-3
;
; POWER runs a simulation with fixed time step, that should be kept sufficiently
; small to avoid coupling instabilities and achieve accurate simulation results.
; The time steps of the THEA  and POWER simulations do not need to be the same.
;
   TimeStep      0.1e-3
   OutputStep    1.0e-3

   LogFile     power.log
   Storagefile power.store

   Branches    4
   Meshes      2

End

Begin Branch 1
   Type  CurrentSupply
   Model constant
   Current 300.0
End

Begin Branch 2
   Type Inductance
   Inductance  0.0 0.1e-3 0.0 0.0
End

;
; The following branch mimics the THEA cable, and triggers the connection to a coupled
; simulation, with the branch resistance computed by THEA
;
Begin Branch 3
   Type Resistance
   Model external
End
```

```
Begin Branch 4
   Type Resistance
   Model constant
   Resistance  0.3e-3
End

;
; The first current mesh is the normal powering connection to the power supply
;
Begin Mesh 1
   Current  300.0
   connectivity  1  1  1  0
End

;
; The second current mesh is the connection to the protection resistor
;
Begin Mesh 2
   Current  0.0
   connectivity  0  1  1  1
End
```

Input file for SUPERMAGNET The two simulations above are launched and supervised by `SUPERMAGNET`., whose input is reported below. The input file is based on a standard installation, and the use of the standard version of `THEA` and `POWER`. Two child processes are defined, and they are linked by associating the cable current of the `THEA` model to the third `branch` of the `POWER` model.

`SM.input`

```
; SUPERMAGNET input file for the THEA-POWER demo coupling of a strand quench
; to an external power supply discharge on a protection resistance

Begin Child

; The first child defined is the POWER run. Below is the name of the child, used later
; to define the coupling indices between children processes
   Name         P1

; The type of process is "power" (a POWER run)
   ProcessType  power

; The input for this run is in the file power.input
   Inputfile    power.input
End

Begin Child

; The second child defined is the THEA run. Below is the name of the child, used later
; to define the coupling indices between children processes
   Name         T1

; The type of process is "thea" (a THEA run)
   ProcessType  thea

; The input for this run is in the file thea.input
   Inputfile    thea.input
```

```
End

; Below is the definition of which specific part of the two children processes are
; coupled. In particular, the cable current of the THEA model is obtained from the
; to the third branch in the POWER model. Remember that in general the numbering
; refers to the order of definition in the models for each child

Begin Connection
;              THEA          POWER
      Children   T1            P1           ; children ID's
      Link       cable         branch 3     ; component ID's
End
```

Restart run SUPERMAGNET can manage restart runs of all codes supervised, once more provided that the user has taken care of maintaining consistency among the coupled runs. It is possible to run restarts, a restart file is needed for each of the coupled processes. Below is the example of the restart files for THEA and POWER.

thea.restart

```
Begin Simulation

  restart

  EndTime                  2.0
  OutputStep              10.0e-3

  LogFile                  thea.log
  StorageFile              thea.store

end
```

power.restart

```
Begin Simulation

   restart

   EndTime       2.0
   TimeStep      1.0e-3
   OutputStep   10.0e-3

   LogFile       power.log
   Storagefile   power.store

End
```

The input file of SUPERMAGNET that launches the restart run is nearly identical to that of a normal run, reflecting the fact that SUPERMAGNET only supervises applications. To launch the restart, the user runs SUPERMAGNET in the same way as detailed in Chapter 2.

SM.restart

```
; SUPERMAGNET input file for the THEA-POWER demo coupling of a strand quench
; to an external power supply discharge on a protection resistance. This demo
; launches a restrat run
```

```
Begin Child
   Name         P1
   ProcessType  power
   Inputfile    power.restart
End

Begin Child
   Name         T1
   ProcessType  thea
   Inputfile    thea.restart
End

Begin Connection
;               THEA           POWER
      Children  T1             P1          ; children ID's
      Link      cable          branch 3    ; component ID's
End
```

Post-processing command files Post-processing of the results from the coupled simulation is done using the specific post-processors of each child process, in this case `THEAPOST` and `POWERPOST`. Following is an example of the sequence of commands necessary to generate plots using the post-processor `THEAPOST`. Refer to [1] for details on the input syntax.

thea.post

```
StorageFile    thea.store
PostScriptFile thea.ps

set plotsperpage 6

plot Current
plot Resistance
plot TotalQJoule

select time 0.0 0.001 0.002 0.005 0.010 0.020 0.050 0.100
plot temperature thermal 1

select time 0.1 0.2 0.5 1.0 2.0
plot temperature thermal 1

stop
```

Similar to above, we report below an example of the sequence of commands necessary to generate plots using the post-processor `POWERPOST`. Refer to [3] for details on the input syntax.

power.post

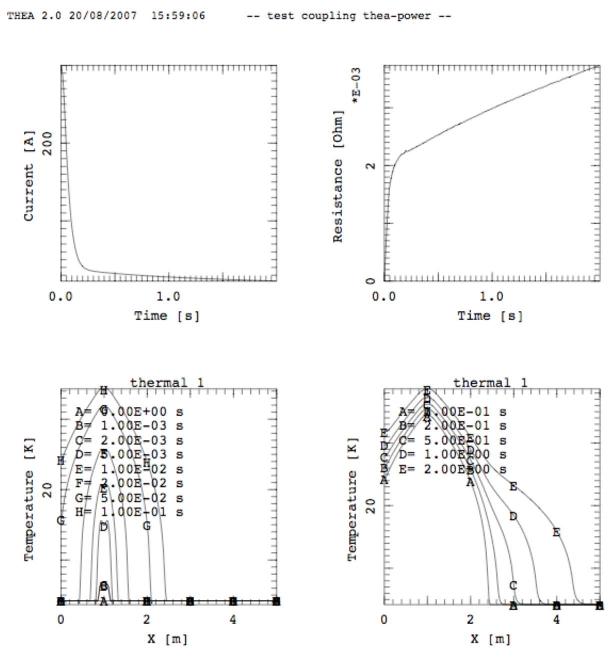
```
StorageFile    power.store
PostScriptFile power.ps

set plotsperpage 4

plot current branch 1
plot current branch 2
plot current branch 3
plot current branch 4

plot voltage branch 1
plot voltage branch 2
plot voltage branch 3
plot voltage branch 4

plot currentderivative branch 1
plot currentderivative branch 2
plot currentderivative branch 3
plot currentderivative branch 4

plot voltagederivative branch 1
plot voltagederivative branch 2
plot voltagederivative branch 3
plot voltagederivative branch 4

newpage

set plotsperpage 2

plot current mesh 1
plot current mesh 2

plot currentderivative mesh 1
plot currentderivative mesh 2
```

Results Two files are generated running the post-processors `THEAPOST` and `POWERPOST` with the commands described above: the PostScript output `thea.ps`, containing the plot for the `THEA` run, and PostScript output `power.ps`, for the `POWER` run.

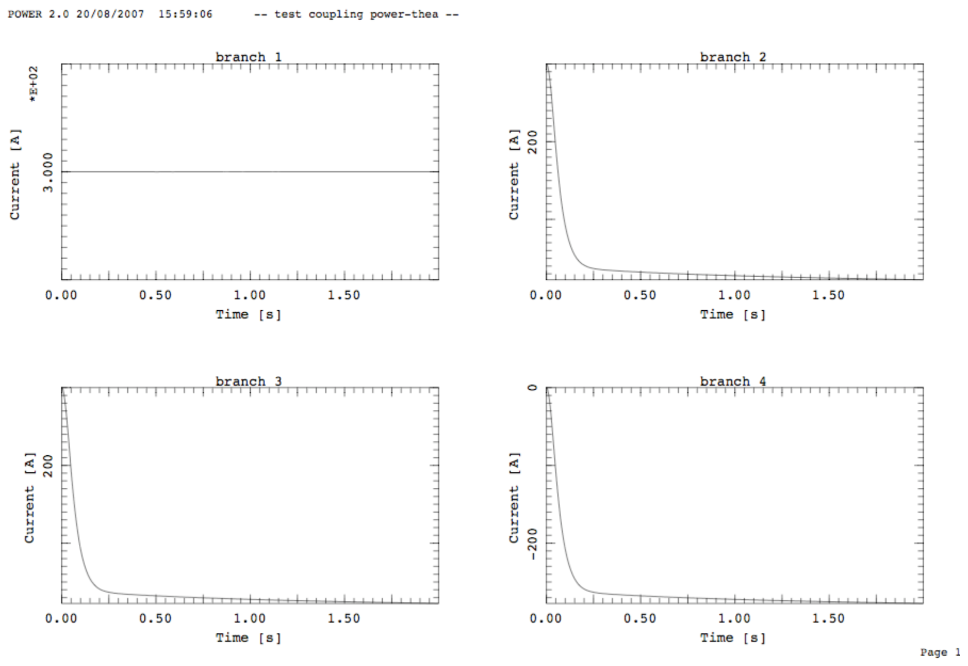
Note You will need a PostScript viewer to look at the plots in the PostScript file. The standard viewer, usually installed on UNIX systems, is `gs`. Try to launch the viewer with the commands:

```
gs thea.ps
gs power.ps
```

The plots below show the first page in the PostScript output `thea.ps`. The first three plots are the cable current, the resistance and the total Joule heat, while the following two are temperature profiles at selected times during the simulation. We note the current decay in the coil, initially very fast, with a quasi-exponential decay, followed by a slower discharge due to the steady rise of the cable resistance. The maximum temperature reached in the strand is of the order of 40 K after 2 s, which is fairly safe.



The first page of the corresponding output from the post-processing of the POWER run is shown below, from the PostScript output `flower.ps`. In this case we see the current in the four circuit branches. The power supply (branch 1) delivers a constant current. The current in branches 2 and 3 is identical and decays (this is the cable current in the previous output from THEA). At the same time the current in the protection resistance, branch 4, rises in amplitude from an initial value of 0, corresponding to the power supply current diverted from the coil.



CHAPTER 4

Input Reference

Structure and syntax

The input file is read by the input interpreter that parses and analyzes the syntax and the grammar of the various entries. In general the file contains a series of blocks that are structured as follows:

```
Begin BlockName
        VariableName    value(s)
        VariableName    value(s)
        ………………..

        VariableName    value(s)
End
```

where *BlockName* is a keyword indicating the block type, and must be one of the following valid choices:

`Child`	define the general properties of a child process
`Connection`	define the links between two children

The content of a block is a series of assignations of a set of values to a generic variable *VariableName*. *VariableName* must be chosen among the set of keywords described in the following sections.

The structure and content of the input file must comply with the following rules and conventions:

- the identifier of a variable and the corresponding value(s) can appear at any position on the line, they can carry on to several lines and must be separated by blanks or tabs;
- the interpretation is case insensitive;
- abbreviations of the keys are not allowed;
- a character ';' in any position of the command line indicates that the remainder of the line must be considered as a comment. If the ';' is the first character in a line, then the whole line is ignored;
- the variables in the block are read sequentially and are checked at read-in time. For this reason the order of precedence of the variables must be respected whenever a value is needed to proceed with the interpretation of a block (i.e. a child must be available before reading its connections) . The same BlockName can appear more than once in a file;
- repeated variable assignation overrides previous values and is not checked at read-in time;

Parsing of the input file is finished as soon as an end-of-file is found. At this point the execution control is passed to the main program that executes checks on data consistency, configures the run and launches the simulation. For sample input files see Chapter 3.

Input variables reference

The following table contains, in alphabetical order, the keywords defining the input variables and meanings for each block type. Predefined possible values are reported in `Courier`. The default value is indicated in the table and underlined.

Note In the tables below we use the following convention for the type of variables:

C	character (a string delimited by blanks, tabs or apices)
R	real (a number in floating point or engineering notation)
I	integer (an integer number)

Typing must be respect in the input file to avoid errors or mis-interpretation by the parser.

Child

The `Child` block defines a child process.

Variable	Type	Units	Meaning
`Executable`	C	(-)	Name of the executable to run. If missing the standard Cryosoft executable for `ProcessType` is taken, as located in `~/CryoSoft/bin/`.
`InputFile`	C	(-)	Name of the input file to be passed to the child. See the manuals of the corresponding programs for details.
`Name`	C	(-)	Nickname of the process, used in the block `Connection` to identify a child uniquelly in case of more processes of the same type running at the same time.
`ProcessType`	C	(-)	This variable defines the kind of the child process. Possible values: `Flower` version 4.2 or higher `Power` version 2.0 or higher `Thea` version 1.5 or higher

Connection

The `Connection` block defines the interconnection of two children.

Variable	Type	Units	Meaning
`Children`	C	(-)	Array of `2` elements containing the nicknames of the children to be connected.

`Link`	C/I	(-)	Array of (`2,2`) elements specifying the type (`LinkType`) and ID (`LinkID`) of the components that are linked between children processes and produce the desired coupling. The entries must be in the following order:
			`LinkType(1)` `LinkID(1)`
			`LinkType(2)` `LinkID(2)`
			The order of the indices is important, `LinkType`(1) and `LinkID`(1) is a valid component type and ID in the process `Children`(1), while `LinkType`(1) and `LinkID`(1) is a valid component type and ID in the process `Children`(2). The allowable values of `LinkType`(1) and `LinkID`(1) depend on the processes linked as follows:

Child	LinkType	
`THEA`	`Cable`	links to `POWER`
	`Hydraulic`	links to `FLOWER/HEATER`
	`Thermal`	links to `HEATER`
`FLOWER`	`Volume`	links to `HEATER`
	`Junction`	links to `THEA/HEATER`
`POWER`	`Branch`	links to `THEA`
`HEATER`	`Point`	links to `FLOWER`
	`Line`	links to `THEA/FLOWER`

The links allowed are listed in the table below that contains for any type of component of the first child the allowed coupling to components in the second child

		Child(2)				
Child(1)	Component	`THEA`	`FLOWER`	`POWER`	`HEATER`	`MrX`
`THEA`	`cable`			`branch`		
	`thermal`				`line`	
	`hydraulic`		`junction`		`line`	
`FLOWER`	`volume`				`point`	
	`junction`	`hydraulic`			`line`	
`POWER`	`branch`	`cable`				
`HEATER`	`point`		`volume`			
	`line`	`thermal/ hydraulic`	`junction`			
`MrX`						

CHAPTER 5

Troubleshooting and Errors

Error messages are reported to the output ASCII log file and to the standard output. The form of a typical error report is the following

```
ERROR in procedure <procedure name>: <error message>
called by <calling procure> at position <n>
called by <calling procure> at position <m>
......
```

where <*procedure name*> is the name of the routine where the error occurred and <*error message*> reports a short description of the error situation. This line is followed by the trace of the <*calling procedure*> up to the main program. In case of queries about error conditions, please take care to report error messages completely, including the calling trace.

Errors can be generated at four different levels in the code, and/or in any of the coupled children processes:

- input parsing and syntax errors;
- data consistency errors;
- runtime errors;
- internal consistency errors.

Input parsing errors

Input parsing and syntax errors are detected during the interpretation of the input file. They indicate that the variable naming, the command syntax or the type and number of numerical data in the input file are incorrect. Verify syntax in the input file in this case.

Data consistency errors

Data consistency errors are detected when input data are not coherent among themselves and would result in a model that cannot be analyzed. Typical cases are selection of incompatible options, or input data out-of-range. Verify the consistency of the input data in this case. While data consistency is easily verified for each separate child process, this is not possible at the level of coupled processes (under the control of the user). For this reason `SUPERMAGNET` demands much care in the preparation of a run, especially to ensure the internal consistency of the various input files. Indeed error messages during a `SUPERMAGNET` run may point to a data consistency problem among input files.

Runtime errors

Runtime errors are detected either when a solver enters a physical or numerical instability, or when the size of the problem exceeds the maximum allowed. Refer to the manual of each of the coupled codes for advice on the treatment of instabilities and how to adapt the solver to the size of the problem.

In addition, in the case of `SUPERMAGNET`, the explicit coupling algorithm can lead to numerical instabilities among codes. The present version of `SUPERMAGNET` does not check for this type of instabilities, which will cause the solution of one or more of the coupled children processes to diverge. The only possible cure to this type of instability is to decrease the time step of the coupled codes.

The maximum size of the coupled system that can be solved is determined by the requested memory allocation in the FORTRAN include files:

```
~/CryoSoft/src/supermagnet/code_x.x/includes/SM.inc
~/CryoSoft/src/library/IC/IC.inc
```

where a number of parameters are set statically. The version of the code you received can be modified by adjusting these parameters as desired. The code then needs to be compiled and link-edited as explained in the installation manual you received [6].

Warning Modifying the code dimensioning parameters requires understanding of the memory allocation for the system variables, and of the internal structure of the code. IN NO EVENT WILL CRYOSOFT BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY AUTHORISED OR UNAUTHORISED USE OF THIS FEATURE, even if advised of the possibility of such damages.

Internal consistency errors

Internal consistency errors indicate corruption of the internal data structure of the program. An internal consistency error cannot be generated using the standard program and reading data from input only. However, they can be detected in case of data inconsistencies among coupled children processes (see above), or they can be triggered in each child process by customized External Routines with improper data handling. Internal consistency errors diagnose a severe fault within the code. Verify data consistency among input files. If you are using External Routines, verify their consistency with the calling protocol. Failing to track internal consistency errors to input or External Routines, report these errors to us.

CHAPTER 6

References

[1] CryoSoft THEA, Thermal, Hydraulic and Electric Analysis of Superconducting Cables, Version 2.4, January 2021.

[2] CryoSoft FLOWER, Hydraulic Network Simulator, Version 4.5, User's Manual, January 2016.

[3] CryoSoft POWER, Electric Network Simulation of Magnetic Systems, Version 2.1, September 2016.

[4] CryoSoft HEATER, Simulation of Heat Conduction, Version 2.1a, April 2021.

[5] K.C. Park, C.A. Felippa, Partitioned Analysis of Coupled Systems, in Computational Methods for Transient Analysis, T. Belytchko and T.J.R. Hughes eds., Elsevier, 157-219, 1983.

[6] CryoSoft Installation Manual, Version 8.2, January 2021.