LA-UR-05-6225

Approved for public release;
distribution is unlimited.

Title: TEMPERATURE CORRECTIONS IN MCNP FOR CALCULATING THE DOPPLER DEFECT

Author(s): JEREMY L. CONLIN, FORREST B. BROWN, & RUSSELL D. MOSTELLER

Submitted to: X-5 Report, 12 August 2005

Temperature Corrections in MCNP for Calculating the Doppler Defect

Jeremy L. Conlin, Forrest B. Brown Russell D. Mosteller

August 12, 2005

Abstract

The current version of MCNP allows the use of physical data libraries generated at temperatures other than the temperature of the simulation; these simulations must be questioned as to their validity. The standard MCNP distribution of data libraries is restricted to a few temperatures because of the size of the data files. Creating new data libraries at discrete temperatures used to be cumbersome and had to be done manually. A new code has been developed at LANL to generate new data libraries at arbitrary temperatures. A Python script to automate the use of this code was developed; a feature of this script allows the user to obtain equivalent physics at any temperature with existing data libraries and a Pseudo Material construct. The construct allows the user to avoid the computational expense and storage space of generating new data libraries. The Python script was used to facilitate the study of the Doppler defect in light water reactors.

I Introduction

Using computer codes to simulate physics is commonplace. Extensive research has been performed developing complex equations to be implemented in codes. Another method for including physics in computer codes is using data libraries. The use of data libraries gives computer codes accurate physics without requiring complex and computationally intensive calculations. Nearly all of these calculations can be performed only once and used in a variety of problems.

One example of data libraries is the use of nuclear cross sections. The cross section for a particular reaction and particle quantifies the probability of that reaction occurring with that particle and another. Cross sections are dependent upon many parameters including the particles involved, energy, and temperature. Cross sections are *not* dependent on geometry and therefore identical libraries may be used in many different problems.

MCNP[1] is a general-purpose Monte Carlo N-Particle transport code. MCNP allows a user to specify an arbitrary 3-D geometry made up of many different materials. MCNP

uses cross section libraries to provide physics to the transport calculation. The extent of the applications of MCNP includes radiation shielding, medical physics, and nuclear criticality safety. All of the different geometries and possible applications use the same set of data libraries; the cross section tables describing the interactions.

A limitation of using pre-computed data libraries is their dependence on variables such as energy, temperature, particles involved, etc. Achieving accurate results requires the use of data libraries generated at the temperature, energy, etc. of interest. With a code like MCNP, distribution of the code and the accompanying data libraries is limited by the storage capacity of the distribution media (i.e. CDs, DVDs) and/or the capacity of the hard drive on the computer that will run the code. The current version of MCNP, version 5, includes data libraries for neutrons, photons and electrons with a few specific reactions that are important to the applications of MCNP. The biggest limitation is the temperatures at which these libraries are created. Most nuclides have libraries corresponding to room-temperature only, while a few important elements/isotopes (e.g. ^{235}U, ^{238}U, ^{239}Pu) have libraries corresponding to higher temperatures.

This paper reports on two solutions that exist for ensuring correct cross section libraries are used when running MCNP: the DOPPLER code[2] and the Pseudo Material[3] construct. The DOPPLER code takes existing MCNP-ready cross section libraries and creates new libraries at higher temperatures. The Pseudo Material construct uses two existing libraries to obtain the effect of a library at the temperature of interest. Both of these techniques are described in detail in this report.

A Python script was developed to facilitate the use of the DOPPLER code and Pseudo Materials. The DOPLR card is used in an MCNP input file and the Python script, DoBruP, detects its existence and the relevant inputs. This new card and the DoBruP script will be described in Section IV; the DOPPLER code is explained in Section II.

The DoBruP script, along with the DOPPLER code and Pseudo Materials were used to study the Doppler defect in Light Water Reactors. This is explained in section V.

II DOPPLER code

DOPPLER is a code that prepares customized temperature dependent nuclear data files for use in MCNP. In the past it was necessary to reprocess the cross section libraries using the NJOY code[4] which is difficult and too cumbersome for most users. The DOPPLER code was written to simplify the process of generating cross section libraries at higher temperatures.

In order to generate new temperature dependent cross section libraries, DOPPLER only requires an existing MCNP cross section library in ACE format for the nuclide of interest. DOPPLER creates a new cross section library by Doppler broadening the resolved resonance data from the existing library to the new temperature of interest. DOPPLER uses the same broadening technique used in NJOY, maintaining the same level of accuracy as would be

obtained by using NJOY directly. For unresolved resonance (probability tables) and S(α, β) thermal scattering data, DOPPLER will interpolate the data between two existing data libraries. If only one data library is available, DOPPLER simply copies it to the new data set.

DOPPLER requires the user to create a simple input file. There are a few required cards of the DOPPLER input file; the first card indicating the path where the original `xsdir` file is found, entry card(s) indicating which libraries are to be broadened, and a final card consisting solely of the word "end".

The entry cards contain the important information for DOPPLER. The form of the entry card is:

```
ZAID1  ZAID2  TEMP  NewZAID.
```

`ZAID1` indicates the MCNP data library the user wants broadened. The temperature of `ZAID1` must be less than `TEMP`, the evaluated temperature of the new library. `TEMP` can be in units of MeV or kelvin. The value of `ZAID2` depends on the form of the entry card. If a cross section library exists for the particular nuclide above the temperature of interest (`TEMP`), the user has the option of interpolating between two data libraries. In this case, the value of `ZAID2` indicates an MCNP data library for the particular nuclide whose temperature is above `TEMP`. If a data library does not exist for the particular nuclide at a temperature above `TEMP` or if the user wants to extrapolate from `ZAID1` to get the new library, `ZAID2` is just the number '`0`'. `NewZAID` is the new ZAID name for the new data library. An example input file for DOPPLER is shown below.

```
path=/home/data/nuclear/mc/type1/
 lwtr.61t lwtr.62t  567   lwtr.01t
 1001.66c 0         567   1001.01c
 8016.66c 0         567   8016.01c
13027.66c 0         866  13027.01c
92235.66c 92235.65c 977  92235.01c
92235.66c 92235.65c 1133 92235.02c
92238.66c 92238.65c 977  92238.01c
92238.66c 92238.65c 1133 92238.02c
end
```

The input file described above is the Normal Mode for DOPPLER. An Advanced Mode exists, but will not be discussed in this report as it isn't used in the DoBruP script. For information on this mode, consult the DOPPLER user manual.[2]

Care must be taken when creating new cross section libraries using DOPPLER as its abilities are limited as compared to NJOY. DOPPLER cannot extrapolate unresolved resonance

data (probability tables) and therefore must interpolate between two libraries. DOPPLER also cannot extrapolate S(α, β) data. In both cases, if `ZAID2` is not specified, the data will be copied without modification. When libraries with unresolved resonance data or S(α, β) data are used, the user *must* choose to interpolate between two libraries in order to achieve correct conversion to a new temperature.

III Pseudo Material Construct

The Pseudo Material construct is another way to apply correct physics to a problem at a specific temperature if cross section libraries at that temperature don't exist. The Pseudo Material construct requires no additional storage space for new cross section libraries. While no additional preprocessing is required, the use of Pseudo Materials in an MCNP input file will increase the time of the processing run.

Materials in MCNP are defined by declaring which element(s)/isotope(s) are to be included in the associated geometry region and the number density for the element/isotope. Any number of isotopes and number densities may be included in a single material definition. Two (or more) different cross section libraries can be declared for the same isotope regardless of the temperatures or other independent variables of the libraries. An example of a material definition in MCNP is:

```
92238.13c   1.27570E-3    $ 500K
92235.13c   2.36990E-4    $ 500K
```

This material is 15.6% enriched uranium at 500K.

Pseudo Materials take advantage of the fact that more than one library for a particular nuclide may be used in a material definition. Instead of using a single library to define the physics and temperature of a nuclide, the Pseudo Material construct interpolates the number density for a library between two libraries whose temperatures bracket the temperature of interest. A Pseudo Material identical to the "normal" MCNP material shown above is listed below; note the differences.

```
92238.14c   6.69986E-04     $ 52.52% @ 600K
92238.12c   6.05714E-04     $ 47.48% @ 400K
92235.14c   1.24465E-04     $ 52.52% @ 600K
92235.12c   1.12525E-04     $ 47.48% @ 400K
```

Pseudo Materials are linearly interpolated with the square root of the temperature, each cross section library getting a fraction of the total number density for that nuclide. The

total cross section for the nuclide is

$$\Sigma(T) = \Sigma(T_L)f_L + \Sigma(T_H)(1 - f_L), \tag{1}$$

where T_H is the temperature of the cross section library with a higher temperature, T_L is the temperature of the cross section library with a lower temperature, and T is the temperature of interest. The temperatures of the two libraries must bracket the temperature of interest, $T_L < T < T_H$. The fraction (f_L) of the total number density associated with the cross section library with the lower temperature is

$$f_L = \frac{\sqrt{T_H} - \sqrt{T}}{\sqrt{T_H} - \sqrt{T_L}}. \tag{2}$$

The fraction(f_H) of the total number density associated with the cross section library with the higher temperature is then clearly

$$f_H = 1 - f_L. \tag{3}$$

The libraries used in a Pseudo Material are not required to evenly bracket the temperature of interest for the nuclide. Another example of a Pseudo Material is shown whose library temperatures are not evenly spaced with respect to the temperature of interest which is 1000K.

```
92238.16c   8.29639E-04     $ 65.03% @  900K
92238.17c   4.46061E-04     $ 34.97% @ 1200K
92235.16c   1.54124E-04     $ 65.03% @  900K
92235.17c   8.28659E-05     $ 34.97% @ 1200K
```

The Pseudo Material construct was verified[3] by running a series of MCNP kcode simulations at different temperatures using both normal materials and Pseudo Materials. The simulation was the Very High Temperature Reactor, a Gen. IV nuclear reactor. The values of k_{eff} from the MCNP simulations using normal materials were assumed to be the correct answer and the values from the the simulations using Pseudo Materials were compared with the "correct" values. Both the normal material and Pseudo Material sets of values are shown in Figure 1. While this verification does not include all possible cases where Pseudo Materials may be used, it gives a good indication of its potential application.

IV The DOPLR Card and DoBruP

To facilitate the use of the DOPPLER code and Pseudo Materials, an unofficial MCNP data card has been introduced called `DOPLR`. The `DOPLR` card is similar in construct to the input file for DOPPLER.

An example of the `DOPLR` card is shown below. The structure of the card is the same as for the input to DOPPLER with a few exceptions.

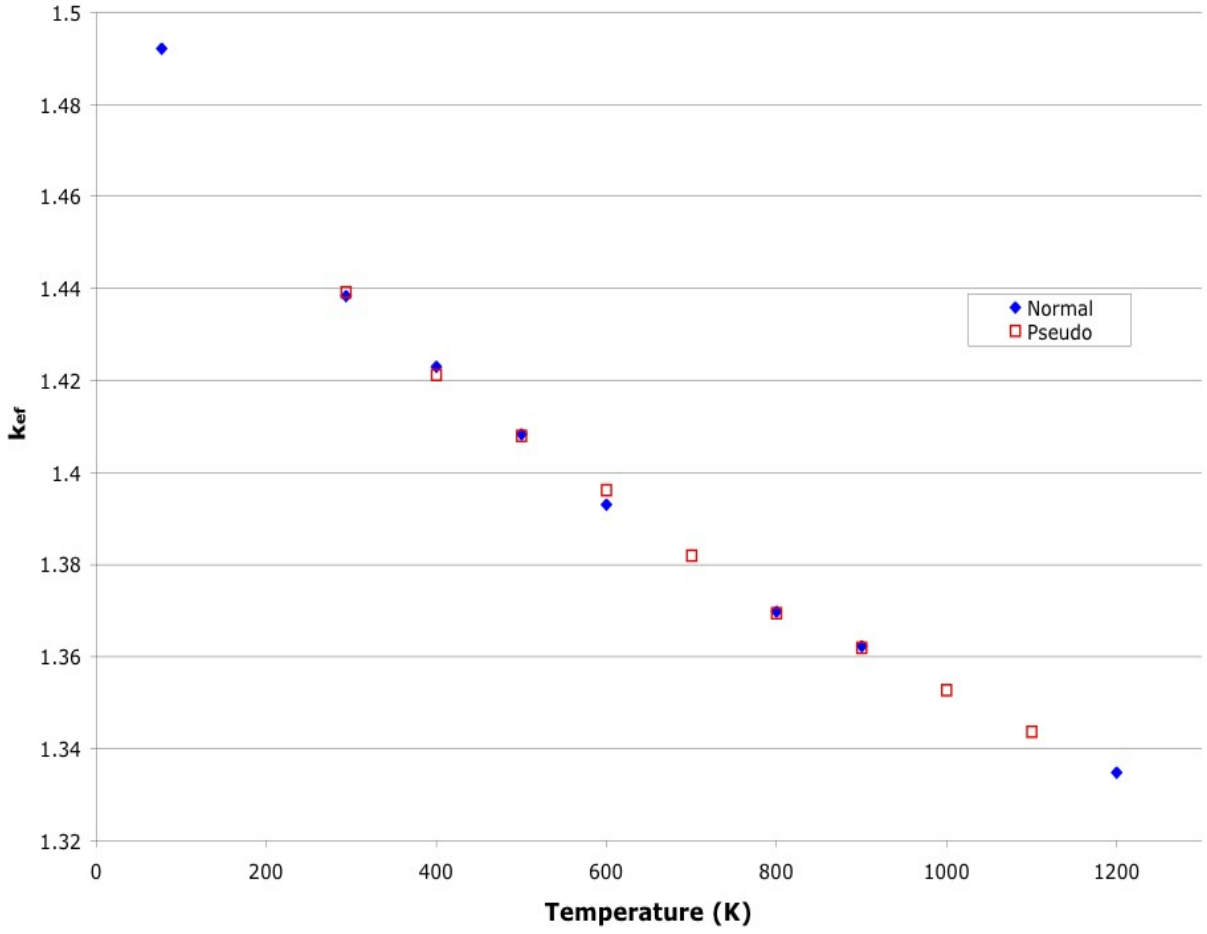

Figure 1: Comparison of k_{eff} between normal and pseudo materials.

```
DOPLR  path=/home/data/nuclear/mc/type1/
       lwtr.61t   lwtr.62t  4.8858E-8    lwtr.01t
       1001.66c             4.8858E-8    1001.01c
       8016.66c             4.8858E-8    8016.01c
      13027.66c             7.4623E-8   13027.01c
      92235.66c  92235.65c  8.4188E-8  *92235.01c
      92235.66c  92235.65c  9.7631E-8  *92235.02c
      92238.66c  92238.65c  8.4188E-8   92238.01c
      92238.66c  92238.65c  9.7631E-8   92238.02c
```

First note there is no line containing the word "end". The temperature definition in the DOPLR card *must* be in units of MeV. If the units are in kelvin, it will be interpreted as a *very* high temperature and the results will be wrong. There currently is no check on the range of this number and the user's error will be assumed correct. Also note the absence of a value for `ZAID2` when the cross section library is to be extrapolated rather than interpolated.

The most important difference between entries in the `DOPLR` card and the entries in the DOPPLER input file is the optional asterisk as the first character of the `NewZAID` value. The asterisk indicates the entry defines a Pseudo Material. For example in the example shown above, both entries for ^{235}U indicate Pseudo Materials are to be used. The implementation of the Pseudo Material will be discussed later.

In view of the fact the DOPLR card is an unofficial MCNP data card, an MCNP input file which includes the DOPLR card and its associated entries will cause MCNP to fail. A Python script was created to examine the input file to determine the existence of the DOPLR card and make changes to the input file if it exists. The Python script, DoBruP, also indicates to the user if a material is used for two (or more) cells having different temperatures.

When the Python script, DoBruP, finds the DOPLR card and its entries, it determines whether the DOPPLER code needs to be executed and prepares an input file appropriately. If there are any Pseudo Materials indicated in an entry of the DOPLR card, DoBruP will find `NewZAID` in the material cards of the MCNP input file and create Pseudo Materials in their place. The materials in the MCNP input deck which contain Pseudo Materials must be in column format or DoBruP will be unable to find `NewZAID`. An example of a material card changed with DoBruP is shown below.

```
Before DoBruP
   M1   8016.99c   4.61174E-02
        92235.99c  1.66030E-04
        92238.99c  2.28927E-02

After DoBruP makes Pseudo Materials
   M1   8016.99c   4.61174E-02        $ Fuel, 0.00711 enrichment
  c     92235.99c  1.66030E-04        $
        92235.66c  1.33565E-04        $ 80.45% @ 2.530E-08 MeV
        92235.65c  3.24650E-05        $ 19.55% @ 2.585E-07 MeV
  c     92238.99c  2.28927E-02
        92238.66c  1.84163E-02        $ 80.45% @ 2.530E-08 MeV
        92238.65c  4.47637E-03        $ 19.55% @ 2.585E-07 MeV
```

In the above example, both `92235.99c` and `92238.99c` were specified as `NewZAID`s for Pseudo Materials. In the material cards these were entered as any other ZAID/density pair. DoBruP found `92235.99c` and `92238.99c` and changed them into Pseudo Materials and commented out the original entry making the material valid for MCNP.

V DoBruP Application—Calculating the Doppler Defect

DoBruP was applied to simplify the determination of the Doppler defect or the Doppler coefficient of reactivity. The Doppler defect is a decrease in the reactivity of a reactor between Hot-Zero Power (HZP) and Hot-Full Power (HFP). For this study, HZP is defined to be 600K and HFP is 900K. This study is similar to that performed by Mosteller, et al.[5]

The Doppler defect was studied by creating an infinite lattice of LWR fuel elements in an MCNP simulation. In both the HZP and HFP cases the coolant, cladding, and gap are at 600K while the fuel pin is at 600K or 900K, respectively.

We studied three different types of fuel each with varying enrichments; UO_2, and both Reactor and Weapons Grade MOX fuel. The enrichment range for the UO_2 fuel is 0.7 weight percent (natural uranium) to 5 weight percent. In the MOX instances PuO_2 is mixed with natural uranium at 1–8 weight percent of PuO_2 for Reactor Grade fuel and 1–6 weight percent of PuO_2 for Weapons Grade fuel. The enrichments for each plutonium isotope in both types of MOX fuel are shown in Table 1.

The geometry for all of the simulations is an infinite lattice of fuel elements with coolant. The cladding is smeared across the gap to simplify the geometry. The dimensions for the fuel, cladding and pitch are shown in Table 2. The temperature of the fuel pins are 600K or 900K, while the cladding and coolant are kept at 600K in all cases.

	^{239}Pu (at %)	^{240}Pu (at %)	^{241}Pu (at %)	^{242}Pu (at %)	Pu Atomic Weight
Reactor Grade	45	30	15	10	239.954006
Weapons Grade	93.6	5.9	0.4	0.1	239.122279

Table 1: PuO_2 Enrichments in MOX Fuel.

Parameter	600K (HZP)	900K (HFP)
Fuel Outer Radius (cm)	0.39398	0.39433
Clad Outer Radius (cm)	0.45972	0.45972
Pitch (cm)	1.26678	1.26678

Table 2: Fuel cell dimensions at 600K and 900K.

Figures 2–4 in Appendix A show the effective multiplication factor (k_{eff}) as a function of fuel enrichment at HZP and HFP for each fuel type in order to see the Doppler defect. Figures 5–7 show the Doppler defect ($\Delta\rho_{Dopp}$) as a function of fuel enrichment.

VI Conclusions

The only way to ensure correct physics in MCNP is to use the correct cross section libraries for the temperature of interest. While the standard MCNP5 distribution is limited in the number of cross section libraries included, the DoBruP script along with the DOPPLER code and Pseudo Material construct provides a means whereby a user can have appropriate temperatures for their simulation.

DoBruP was applied to the problem of calculating the Doppler defect. These calculations were performed as a computational benchmark. In lieu of the fact that they are benchmark calculations, there is nothing with which a direct comparison can be made. We do notice some expected trends which give confidence to the results. The Doppler defect must be negative, and it is always negative. The best fit curve through the plot of the change in reactivity as a function of fuel enrichment (see Figures: 5, 6, 7) increases asymptotically as expected in all cases. The best fit lines through the MOX fuel plots are shown as linear plots, the Doppler defect with this fuel is small and higher order fits are not needed.

References

[1] F.B. Brown, "MCNP—A General Monte Carlo N-Particle Transport Code, Version 5", LA-UR-03-1987, Los Alamos National Laboratory (2003).

[2] R.E. MacFarlane and Patrick Talou, "DOPPLER: A Utility Code for Preparing Customized Temperature-Dependent Data Libraries for the MCNP Monte Carlo Transport Code", Los Alamos National Laboratory (2003).

[3] J.L. Conlin, W. Ji, J.C. Lee, W.R. Martin "Pseudo-Material Construct for Coupled Neutronic-Thermal-Hydraulic Analysis of VHTGR", Trans. ANS 91 (2005).

[4] R.E. MacFarlane and D. W. Muir, "The NJOY Nuclear Data Processing System, Version 91", LA-12740-M Los Alamos National Laboratory (1994).

[5] Mosteller, Russell D.; Eisenhart, Laurence D.; Little, Robert C.; and Chao, Jason. "Benchmark calculations for the Doppler coefficient of reactivity", *Nuclear Science and Engineering*, 107, 3, (1991).

Appendix A Doppler Defect Plots

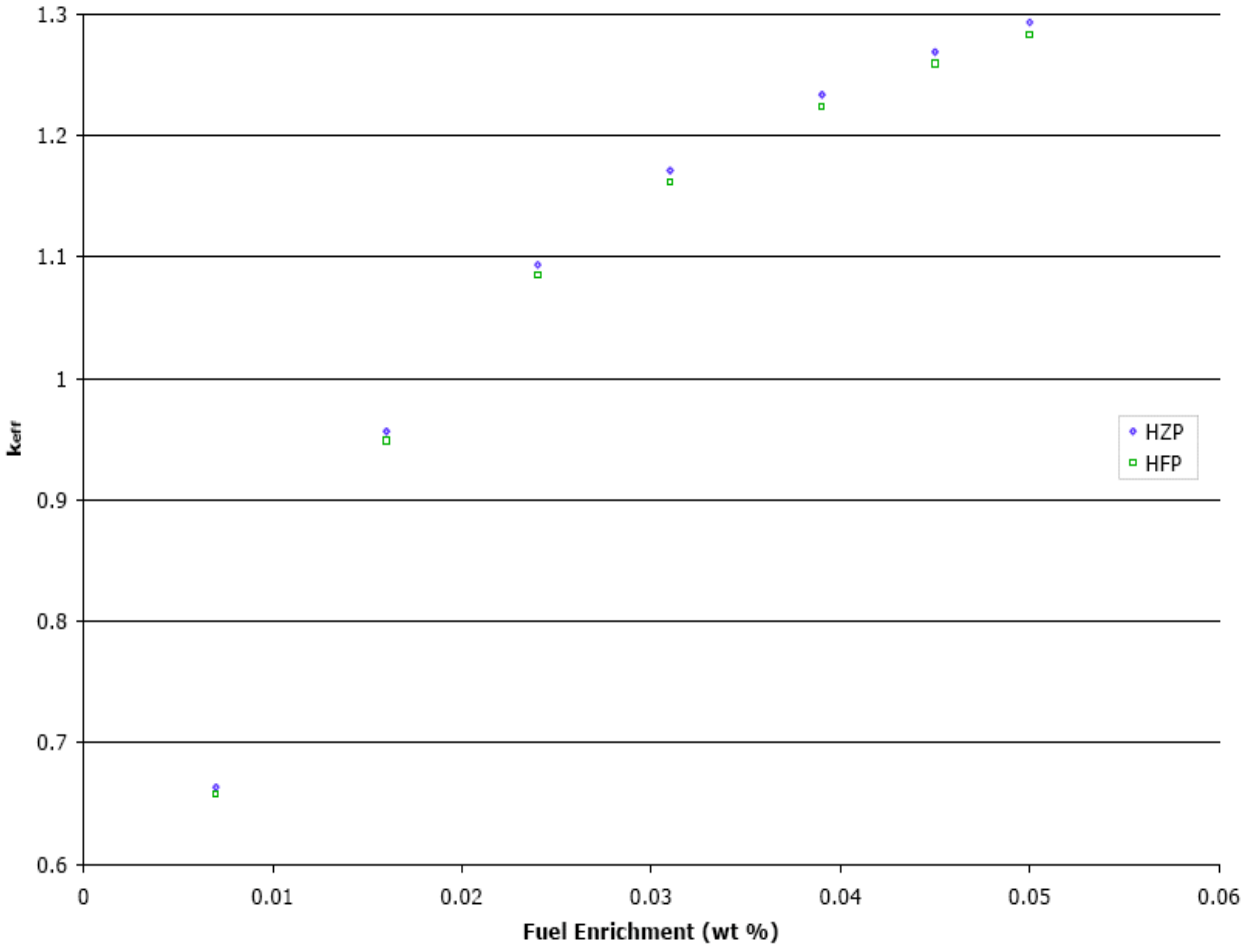

Figure 2: k_{eff} as a function of uranium enrichment in UO_2 fuel for HZP and HFP.

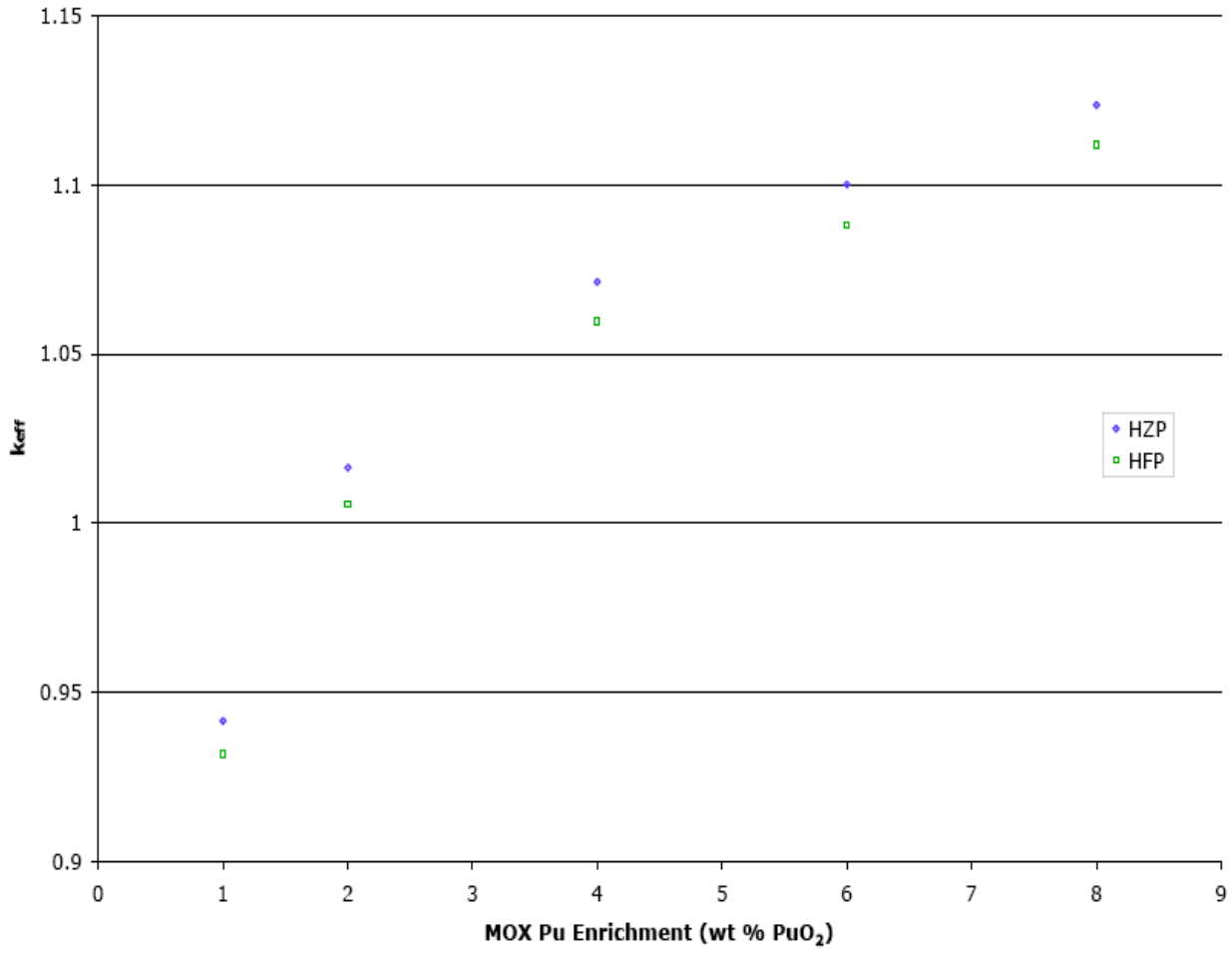

Figure 3: k_{eff} as a function of PuO_2 enrichment in Reactor Grade MOX fuel for HZP and HFP.

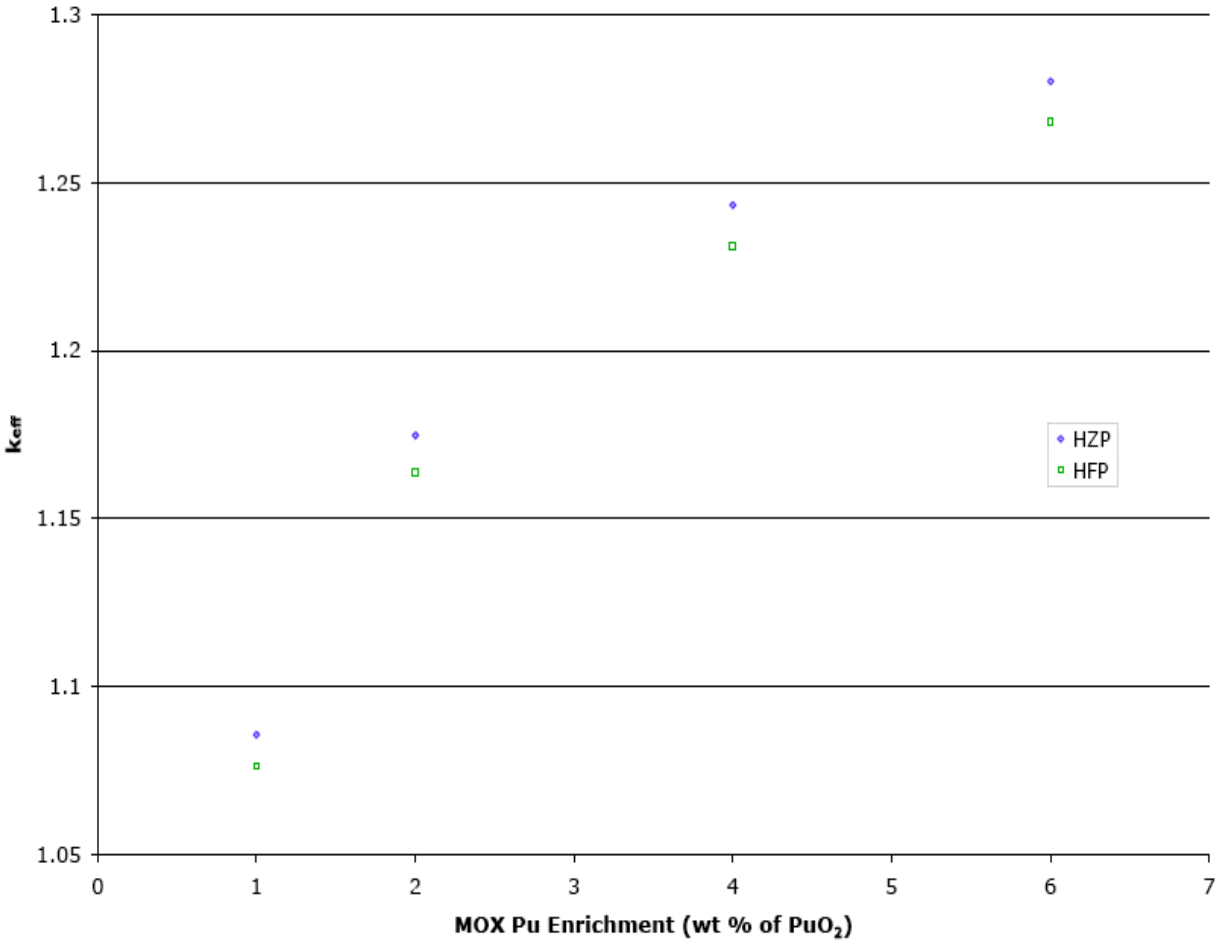

Figure 4: k_{eff} as a function of PuO_2 enrichment in Weapons Grade MOX fuel for HZP and HFP.

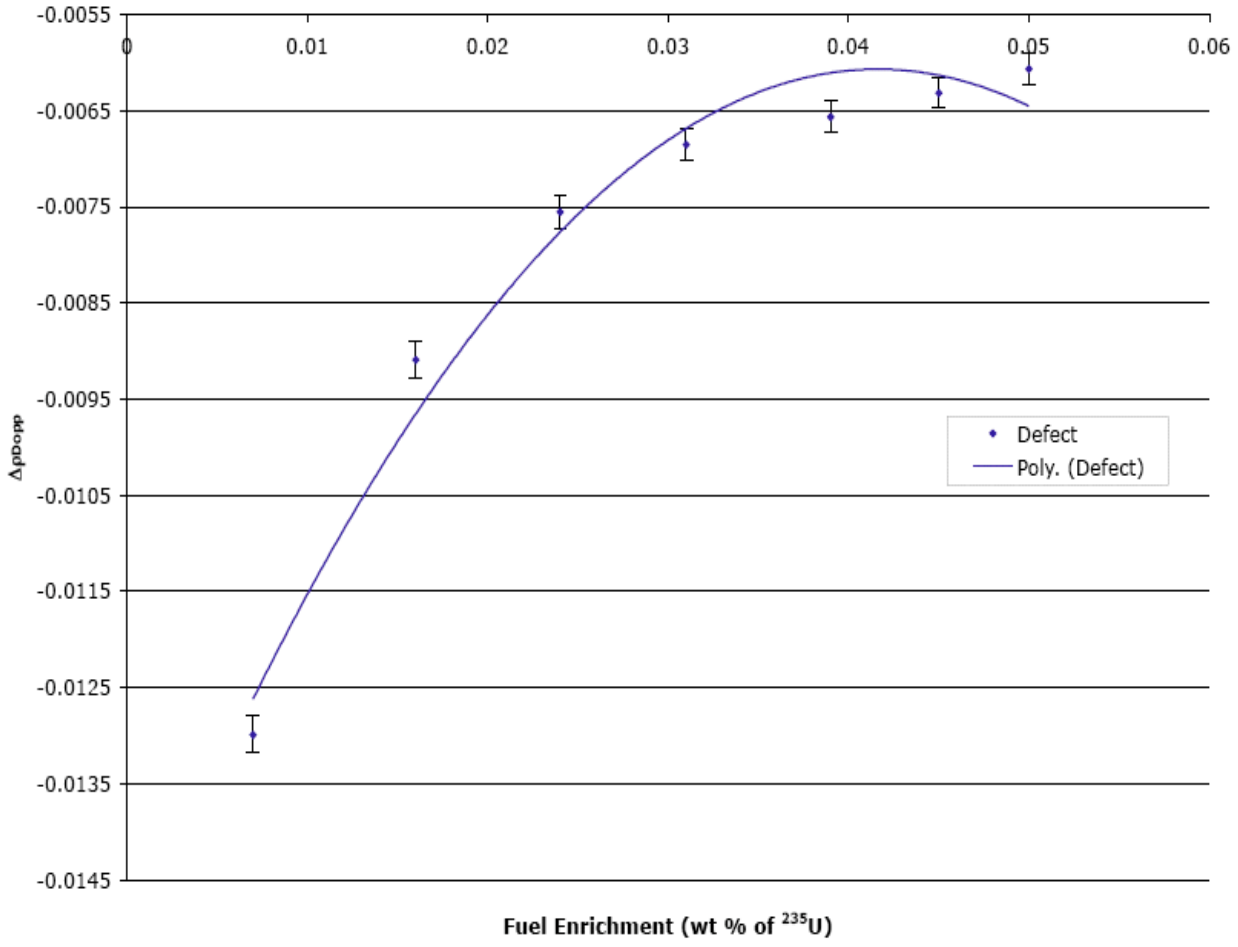

Figure 5: Doppler defect as a function of uranium enrichment in UO_2 fuel.

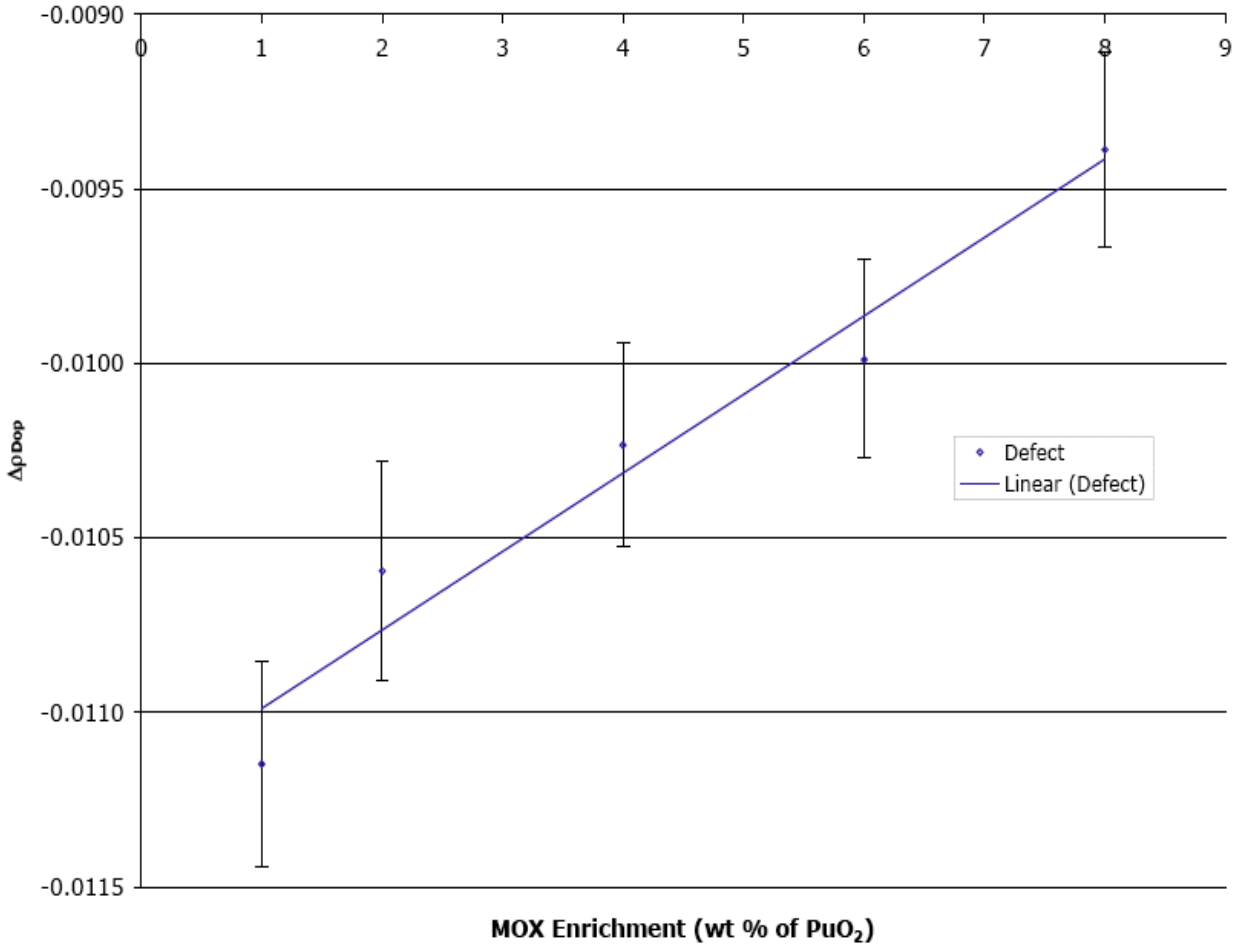

Figure 6: Doppler defect as a function of PuO_2 enrichment in Reactor Grade MOX fuel.

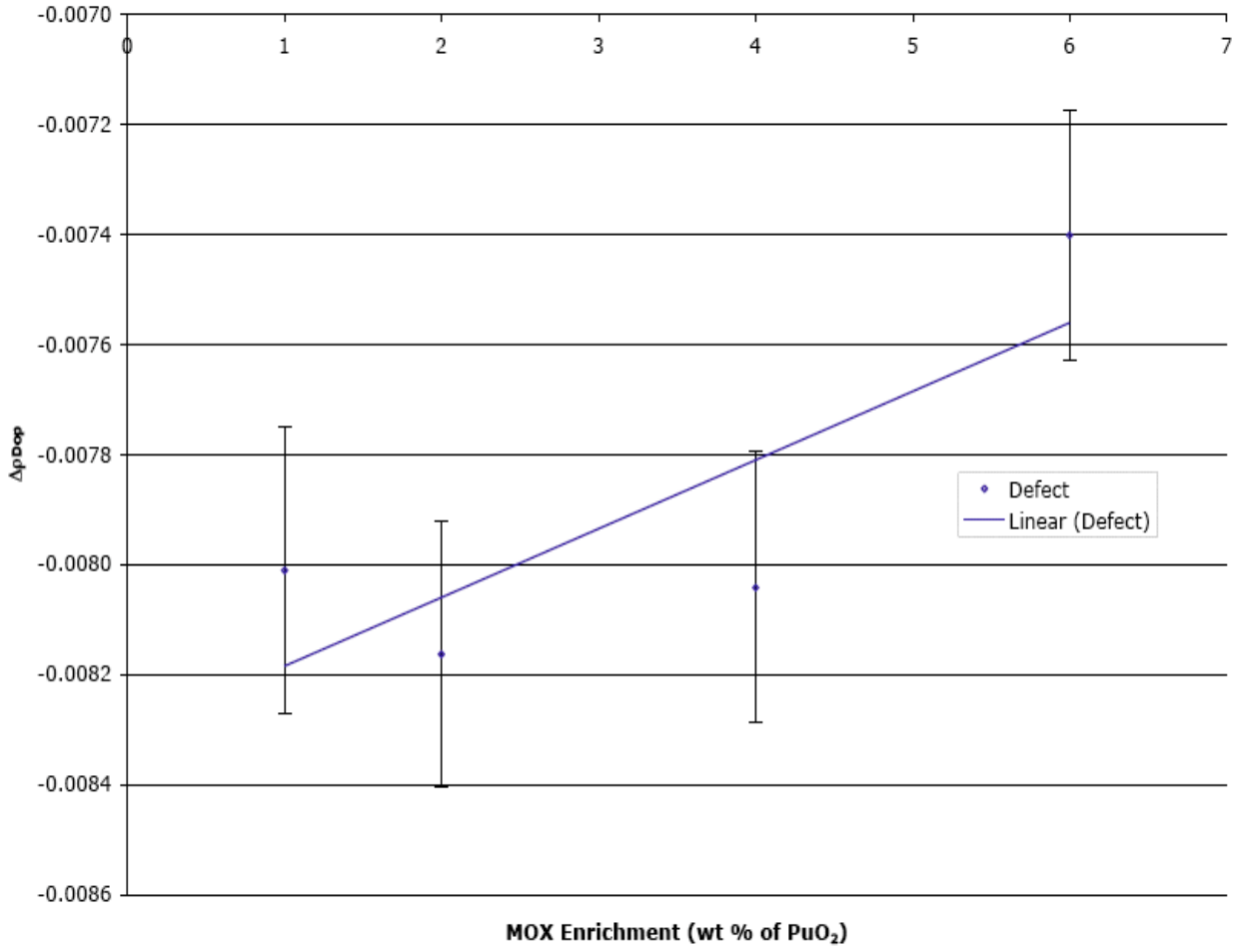

Figure 7: Doppler defect as a function of PuO_2 enrichment in Weapons Grade MOX fuel.

Appendix B DoBruP—Input file

This is an example of an input file for DoBruP/MCNP.

```
EDB.pstudy -- Extended Doppler Benchmark
c
c
c    This input deck investigates the Doppler Coefficient of Reactivity.
c    This file makes use of the pstudy utility to analyze many different
c    parameters in the pin cell.  This input deck uses cross section
c    libraries generated by the DOPPLER code at 600K and 900K, Hot Zero
c    Power and Hot Full Power respectively.  Uranium enrichments ranging
c    from 0.7% to 5% are examined.
c    The number densities were calculated in the file:
c        NumberDensities.DOPPLER.xls
c
c --- Temperature options ---
c - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
c --- Fuel Density ---
c --- Fuel Outer Radius ---
c - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
c --- Oxygen number densities ---
c --- Uranium number densities ---
c - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
c
c    Created by:    Jeremy Conlin     jlconlin@lanl.gov
c    Last Modified: June 29, 2005
c ----------------------------------------------------------------------
c  Cell Cards
c ------------
 101  1  -10.339    -1               TMP=5.1702e-08  IMP:n=1      $ Fuel
 102  2  -6.3902    -2 1             TMP=5.1702e-08  IMP:n=1      $ Clad
 103  3  -0.66163  -10 11 -12 13 2   TMP=5.1702e-08  IMP:n=1      $ Mod
 100  0            10:-11:12:-13                    IMP:n=0

c  Surface Cards
c ---------------
   1   CZ  0.39398     $ Fuel Outer radius
   2   CZ  0.45802     $ Clad Outer Radius
 *10   PX  0.63104
 *11   PX -0.63104
 *12   PY  0.63104
 *13   PY -0.63104

c  Data Cards
```

```
c ------------
kcode 20000 1.0 20 520
ksrc 0 0 0
c
doplr path=/home/data/nuclear/mc/type1
      1001.66c              5.1702E-8   1001.00c
      5010.66c              5.1702E-8   5010.00c
      5011.66c              5.1702E-8   5011.00c
      8016.66c              5.1702E-8   8016.00c
      40000.66c             5.1702E-8   40000.00c
      8016.66c              5.1702e-08   8016.99c
      92234.66c  92234.65c  5.1702e-08  *92234.99c
      92235.66c  92235.65c  5.1702e-08  *92235.99c
      92238.66c  92238.65c  5.1702e-08  *92238.99c
c
 M1   8016.99c   4.61174E-02          $ Fuel, 0.00711 enrichment
      92234.99c  0.00000E+00
      92235.99c  1.66030E-04
      92238.99c  2.28927E-02
c
 M2   40000.00c  1       $ Cladding
c
 M3   1001.00c   4.42326E-2
      5010.00c   1.02133E-5
      5011.00c   4.11098E-5
      8016.00c   2.21163E-2
MT3   lwtr.62t           $ S(a,B) treatment for Hydrogen
```

Appendix C DoBruP—Modified input file for MCNP

This is an example of the DoBruP input file shown in Appendix B as modified by DoBruP. This file is a valid MCNP input file.

```
EDB.pstudy -- Extended Doppler Benchmark
c
c
c    This input deck investigates the Doppler Coefficient of Reactivity.
c    This file makes use of the pstudy utility to analyze many different
c    parameters in the pin cell.  This input deck uses cross section
c    libraries generated by the DOPPLER code at 600K and 900K, Hot Zero
c    Power and Hot Full Power respectively.  Uranium enrichments ranging
c    from 0.7% to 5% are examined.
c    The number densities were calculated in the file:
c        NumberDensities.DOPPLER.xls
c
c --- Temperature options ---
c - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
c --- Fuel Density ---
c --- Fuel Outer Radius ---
c - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
c --- Oxygen number densities ---
c --- Uranium number densities ---
c - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
c
c    Created by:    Jeremy Conlin     jlconlin@lanl.gov
c    Last Modified: June 29, 2005
c ----------------------------------------------------------------------
c  Cell Cards
c ------------
 101  1  -10.339    -1               TMP=5.1702e-08  IMP:n=1      $ Fuel
 102  2  -6.3902    -2 1             TMP=5.1702e-08  IMP:n=1      $ Clad
 103  3  -0.66163  -10 11 -12 13 2   TMP=5.1702e-08  IMP:n=1      $ Mod
 100  0            10:-11:12:-13                    IMP:n=0

c  Surface Cards
c ---------------
   1   CZ  0.39398     $ Fuel Outer radius
   2   CZ  0.45802     $ Clad Outer Radius
 *10   PX  0.63104
 *11   PX -0.63104
 *12   PY  0.63104
 *13   PY -0.63104
```

```
c  Data Cards
c ------------
kcode 20000 1.0 20 520
ksrc 0 0 0
c
c doplr path=/home/data/nuclear/mc/type1
c       1001.66c              5.1702E-8   1001.00c
c       5010.66c              5.1702E-8   5010.00c
c       5011.66c              5.1702E-8   5011.00c
c       8016.66c              5.1702E-8   8016.00c
c       40000.66c             5.1702E-8   40000.00c
c       8016.66c              5.1702e-08   8016.99c
c       92234.66c  92234.65c  5.1702e-08  *92234.99c
c       92235.66c  92235.65c  5.1702e-08  *92235.99c
c       92238.66c  92238.65c  5.1702e-08  *92238.99c
c
 M1   8016.99c   4.61174E-02          $ Fuel, 0.00711 enrichment
c     92234.99c  0.00000E+00
c       92235.99c  1.66030E-04
       92235.66c  1.33565E-04        $ 80.45% @ 2.530E-08 MeV
       92235.65c  3.24650E-05        $ 19.55% @ 2.585E-07 MeV
c       92238.99c  2.28927E-02
       92238.66c  1.84163E-02        $ 80.45% @ 2.530E-08 MeV
       92238.65c  4.47637E-03        $ 19.55% @ 2.585E-07 MeV
c
 M2   40000.00c  1       $ Cladding
c
 M3   1001.00c   4.42326E-2
      5010.00c   1.02133E-5
      5011.00c   4.11098E-5
      8016.00c   2.21163E-2
MT3   lwtr.62t           $ S(a,B) treatment for Hydrogen
```

Appendix D DoBruP—Python code

```
#! /usr/bin/python
"""This python script will read an MCNP input deck and determine if the correct cross-
section libraries are loaded for the temperature indicated on the cell card."""

import os, sys, math, string, re
def main():     #{{{1
    from optparse import OptionParser

    # Version number and last modification
    Version = '1.0'
    Modified = 'August 4, 2005'

    # Prepare parser
    usage = 'usage: %prog [options] mcnp.in'
    parser = OptionParser(usage=usage, version='Version #: %s' %Version)
    parser.add_option("-D", "--Doppler", action="store_true", default=False,
                      help="Run the DOPPLER code.")
    parser.add_option("-o", "--out", action="store", type="string",
                      dest="outfile", default="mcnp.in",  help="Run the DOPPLER code.")
    parser.add_option("-d", "--debug", action="store_true", default=False,
                      help="Debugging options")

    # Parse output to see what the user wants to do
    (options, args) = parser.parse_args()

    # Process the MCNP input file and see if the doplr card exists
    McnpIn = Input(args[0])
    # Find all the cell cards that have the TMP keyword in it.
    (TMPCells, TMPConflict) = McnpIn.TMPCells()
    if not TMPConflict:
        DoplrCard = McnpIn.DoplrCard()
        if options.debug:
            print "Debug ---> DoplrCard: ", DoplrCard
            print "Debug ---> DoplrCard[0]: ", DoplrCard[0]

        if DoplrCard: # Continue processing
            for entry in xrange(DoplrCard[0]):
                # Determine if new xsdir file loaction exists
                if options.debug:
                    print "Debug ---> DoplrCard[1][entry]: ", DoplrCard[1][entry]
                if DoplrCard[1][entry][0] == '~':
                    if  DoplrCard[1][entry][1] == '/':
                        # Path is relative to user's home directory
                        DoplrCard[1][entry] = os.path.join(os.path.expanduser('~'), \
                            DoplrCard[1][entry][2:])
                    else:
                        # Path is relative to some arbitrary user's home directory
                        DoplrCard[1][entry] = os.path.join(os.path.expanduser('~'), \
                            DoplrCard[1][entry][1:])

                # Determine of Doppler Path exists.  If not, create new directory
                if not os.path.exists(DoplrCard[1][entry]):
                    try:
```

```
                        os.makedirs(DoplrCard[1][entry])
                    except OSError:
                        print 'Cannot create directory in path: %s' %DoplrCard[0]

                # Get information from the xsdir file about the libraries and temperatures
                # for each nuclei
                XS = xsdir(DoplrCard[1][entry] + '/xsdir')

                # Select only the doplr entries that apply to the DOPPLER code.  These are
                # entries with Action equal to "Interpolate" or "Extrapolate" but not "Pseudo"
                DoplIn = []
                for item in DoplrCard[2:]:
                    if item['Action'] != 'Pseudo':
                        DoplIn.append(item)

                # Read Material cards
                PseudoIn = []
                for item in DoplrCard[2:]:
                    if item['Action'] == 'Pseudo':
                        PseudoIn.append(item)
                # Write Pseudo Materials
                if options.Doppler:
                    Doppler(DoplIn, DoplrCard[1][entry], XS.libs)
                    # Dictionary of new libraries
                    NewXS = xsdir(os.getcwd() + '/xsdir', options.debug)
                    McnpIn.MatPseudo(PseudoIn, NewXS, options.debug)
                else:
                    McnpIn.MatPseudo(PseudoIn, XS, options.debug)

            # Write new input deck
            McnpIn.Write(options.outfile, options.debug)

        else:       # Doppler card doesn't exist do nothing
            print "doplr card doesn't exist.  This script will not modify the input file."
    else:
        print "There is a conflict between the material temperature and the"
        print "temperature on the Cell card.  There must be unique pairs of"
        print "materials and temperatures. This script will not modify the "
        print "input file."

    # Close input file
    McnpIn.infile.close()

# --------------- End of main() -------------------------------------------}}}1

class Input(object):            # {{{1
    """This class handles all the operations for reading and writing to the MCNP
    input file."""

    def __init__(self, infile, debug = False):  #{{{2
#       if debug:
#           print 'DEBUG ---> I am inside Input.__init__'
        self.infile = file(infile, 'r+')
        self.lines = self.infile.readlines()
        self.CellsEnd = self.lines.index('\n')
        self.SurfacesEnd = self.lines.index('\n', self.CellsEnd+1)
```

```
# --------------- End of function __init__() -------------------------}}}2

def Write(self, outfile, debug = False):  #{{{2
    """This function will write a new MCNP input file using self.lines as the
    lines of the file."""
    if not os.path.exists(outfile):
        output = file(outfile, 'w')
        output.writelines(self.lines)
        output.close()
    else:
        print "Output filename: %s already exists.  I give up." %outfile
# --------------- End of function Write() -------------------------}}}2

def DoplrCard(self, debug=False):           # {{{2
    '''This function determines if there is a doplr card among the Data cards.
    If there is a card, it will return all the information from the doplr card
    necessary to continue processing the input deck and running DOPPLER.  If
    the card does not exist, this function will return "None". If the card does
    exist then the first entry in the list returned will be the path that
    DOPPLER will use to write new files.  The remaining entries are the
    cards explaining what is to be done for each nuclide.
    The action associated with each doplr card is returned with the card.  The
    possible actions are: Doppler, Extrapolate, and Pseudo.'''

    DoplrCards = 0  # The number of doplr cards
    Cards = []      # The paths for the doplr cards
    DoplrData = [DoplrCards, Cards]
    # DoplrSearch is a regular expression search string to discover the existance
    # of the doplr card and its associated path
    DoplrSearch = re.compile(r'^ {0,4}doplr +path *[=] *(\S*)', re.IGNORECASE)

    # Search for the "doplr" card and the path for the dopper code
    DoplrLine = False
    Data = self.Data()
    for card in xrange(len(Data)):
        Doplr = DoplrSearch.search(Data[card])
        if Doplr:
            DoplrLine = Data.index(Data[card])
            LineIndex = self.lines.index(Data[card])
            self.lines[LineIndex] = 'c ' + self.lines[LineIndex]
            DoplrCards +=1
            DoplrData[0] = DoplrCards
            Cards.append(Doplr.group(1))
            if debug:
                print "Debug ---> DoplrData: ", DoplrData

            # Get the ZAIDs to be broadened or to be used for pseudo materials
            # and the associated temperatures.  zaidsearch is a regular expression
            # search string to find all the entries for the doplr card.
            zaidsearch = r'''^\ {5,}((?: \d{4,5} | [\w/]*)  # Look for ZAID1
                          \.\d{2}[ctmpuy]{1})\ *            # ZAID1 Extension
                          ((?: \d{4,5} | [\w/]*)         # Look for ZAID2
                          \.\d{2}[ctmpuy]{1} | \ *)         # ZAID1 Extension
                          \ *([\deEd.+-]+)+              # Temperature
                          \ *(\*?(?: \d{4,5} | [\w/]*)   # Newzaid
                          \.\d{2}[ctmpuy]{1})?\ *           # Newzaid extension'''
```

```
                ZAIDSearch = re.compile(zaidsearch, re.VERBOSE | re.IGNORECASE)

                for entry in Data[DoplrLine+1:]:
                    ZAID = ZAIDSearch.search(entry)
                    if ZAID: # Pattern was found, process card
                        if debug:
                            print "Debug ---> entry : ", entry,
                        LineIndex = self.lines.index(entry)
                        self.lines[LineIndex] = 'c ' + self.lines[LineIndex]

                        carddata = {}           # Dictionary to hold information

                        # Determine what action needs to be done with doplr card(s)
                        if ZAID.group(4)[0] == '*':     # Pseudo
                            try:
                                float(ZAID.group(1)[0])
                                carddata = {'ZAID1':ZAID.group(1), 'ZAID2':ZAID.group(2),
                                            'Temperature':float(ZAID.group(3)),
                                            'NewZAID':ZAID.group(4)[1:],
                                            'Action':'Pseudo', 'Card':DoplrCards}
                                DoplrData.append(carddata)
                                if debug:
                                    print 'Debug ---> carddata: ', carddata
                            except ValueError:
                                # Tried Pseudo material with SaB data library
                                print "I cannot use Pseudo materials with SaB data libraries."
                                print "Try again."
                        elif ZAID.group(2) == '':       # Extrapolate
                            carddata = {'ZAID1':ZAID.group(1), 'Temperature':float(ZAID.group(3)),
                                        'NewZAID':ZAID.group(4),
                                        'Action':'Extrapolate', 'Card':DoplrCards}
                            DoplrData.append(carddata)
                            if debug:
                                print 'Debug ---> carddata: ', carddata
                        else:           # Interpolate
                            carddata = {'ZAID1':ZAID.group(1), 'ZAID2':ZAID.group(2),
                                        'Temperature':float(ZAID.group(3)),
                                        'NewZAID':ZAID.group(4),
                                        'Action':'Interpolate', 'Card':DoplrCards}
                            DoplrData.append(carddata)
                            if debug:
                                print 'Debug ---> carddata: ', carddata
                    else:
#                       if debug:
#                           print "Debug ---> I'm done with this doplr card."
#                           print "Debug ---> DoplrData: ", DoplrData, "\n"
                        break

#               else:
#                   break       # Done with doplr card

        return DoplrData

    # --------------- End of function Doplr() ----------------------------}}}2

    def TMPCells(self, debug=False):            # {{{2
```

```
    """This function will return all the cells that use the TMP keyword.
    It also checks to make sure that there isn't a material that is used at
    two different temperatures.  Different Temperatures are defined as being
    more than 1% different.  It returns a list of the cells, materials and
    Temperatures as well as a boolean (TMPConflict) indicating whether or not
    there is conflicting temperatures for a material.  TMPConflict is False
    when everything as it should be"""
    tmpsearch = r'''^\ {0,4}(\d){1,5}               # Cell #
                    \ +(\d+)\ *                     # Mat #
                    .+                              # Lots of other things...
                    TMP\ *[=]\ *([\deEd.+-]+)+      # TMP and (Temperature)'''
    TMPSearch = re.compile(tmpsearch, re.VERBOSE)
    # Dictionary to hold material numbers and Temperatures
    # The keys are the material numbers and the values are the Temperatures
    TMPCells = {}
    # Cell.group(1) is the cell number
    # Cell.group(2) is the material number
    # Cell.group(3) is the temperature if specified

    TMPConflict = False
    for card in self.Cells():
        if debug:
            print 'Debug ---> card: ', card,
        Cell = TMPSearch.search(card)
        if Cell:
            if TMPCells.has_key(Cell.group(2)):
                if float(min(TMPCells[Cell.group(2)], Cell.group(3))) \
                    /max(TMPCells[Cell.group(2)], Cell.group(3)) > 0.01:
                    TMPConflict = True
                    print 'Materials used in different cells, must have the same temperature'
            else:
                TMPCells[Cell.group(2)] = Cell.group(3)
            if debug:
                print 'Debug ---> groups: ', Cell.groups()
        else:       # TMP card not specified, assume room temperature
            words = string.split(card)
            if TMPCells.has_key(words[1]):
                if debug:
                    print "Debug ---> TMPCells[words[1]] = ", TMPCells[words[1]]
                if float(min(TMPCells[words[1]], 2.52995E-8)) \
                    /float(max(TMPCells[words[1]], 2.52995E-8)) > 0.01:
                    TMPConflict = True
                    print 'Materials used in different cells, must have the same temperature'
            else:
                TMPCells[words[2]] = 2.52995E-8     # Room Temperature

            if debug:
                print "Debug ---> words: ", words[1]

    return TMPCells, TMPConflict
# --------------- End of function TMPCells() ------------------------------}}}2

def MatPseudo(self, PseudoCards, xsdir,  debug=False):            # {{{2
    """This function reads the material cards from the input deck
    to find the nuclides that need expanding via Pseudo Materials.
    It will replace entries in material cards with two entries for
```

```
the Pseudo Material.  It does not return any data to main().
PseudoCards = a list from the doplr card indicating which
              ZAID is specified for pseudo material expansion
xsdir = dictionary with data about all available ZAIDs

Current limitations:  The first line of the material card must
have one ZAID and its number density.  Each additional entry
must be on its own line.  Since it only deals with Pseudo
Materials, it does not look at the MT card."""
Mats = {}  # Dictionary to hold the data for the material cards
# MatSearch looks for the beginning of a material card and the first
# ZAID and density which must be on the same line.
matsearch = r''' \ {0,4}(m\d+)\ *           # Material number
                ([\w/]*                         # ZAID
                (?:\.\d{2}[ctmpuy]{1})?)\ *     # ZAID Extension
                ([\ded.+-]+)?\ *                # Number Density
                '''
MatSearch = re.compile(matsearch, re.VERBOSE | re.IGNORECASE)
# NuclDensSearch looks for ZAIDs and their number densities.
# Only one entry per line is allowed with this script
nucldenssearch = r'''^\ {5,}([\w/]*                          # ZAID
                (?:\.\d{2}[ctmpuy]{1})?)\ *     # ZAID Extension
                ([\ded.+-]+)?\ *                # Number Density
                '''
NuclDensSearch = re.compile(nucldenssearch, re.VERBOSE | re.IGNORECASE)
if debug:
    print "Debug ---> self.Data()[0]: ", self.Data()[0]
Data = self.Data()
for card in xrange(len(Data)):
    Mat = MatSearch.search(Data[card])
    NuclDens = NuclDensSearch.search(Data[card])
    if Mat:
        if debug:
            print "Debug ---> Mat.groups(): ", Mat.groups()
        for entry in PseudoCards:
            if Mat.group(2) == entry['NewZAID']:
                LineIndex = self.lines.index(Data[card])
                PseudoMat =  self.Pseudo(float(Mat.group(3)), entry['Temperature'], \
                    entry['ZAID1'], entry['ZAID2'], xsdir)
                self.lines[LineIndex] = 'c ' + \
                    self.lines[LineIndex] + Mat.group(1) + PseudoMat

                if debug:
                    print "Debug ---> card['NewZAID']: ", entry['NewZAID']
    elif NuclDens:
        for entry in PseudoCards:
            if NuclDens.group(1) == entry['NewZAID']:
                LineIndex = self.lines.index(Data[card])
                PseudoMat =  self.Pseudo(float(NuclDens.group(2)), entry['Temperature'], \
                    entry['ZAID1'], entry['ZAID2'], xsdir)
                self.lines[LineIndex] = 'c ' + \
                    self.lines[LineIndex] + PseudoMat

                if debug:
                    print "Debug ---> card['NewZAID']: ", entry['NewZAID']
```

```
    # --------------- End of function TMPCells() -----------------------------}}}2

    def Pseudo(self, Density, Temperature, ZAIDL, ZAIDH, xsdir, debug=False):       # {{{2
        """This function will calculate the appropriate number densities for
        a Pseudo Material.  It returns a string suitable for entering into a
        material card in MCNP.  It requires as input:
        Density = total number density for nuclide
        Temperature = the temperature of interest for the material
        ZAIDL = ZAID library with temperature smaller than Temperature
        ZAIDH = ZAID library with temperature larger than Temperature
        xsdir = dictionary with all the information about available ZAIDs
        """
        (NuclH, ExtH) = string.split(ZAIDH, '.')
        (NuclL, ExtL) = string.split(ZAIDL, '.')
        Thigh = float(xsdir.libs[NuclH][ExtH])
        Tlow = float(xsdir.libs[NuclL][ExtL])

#       if debug:
#           print "Debug ---> Thigh = ", Thigh
#           print "Debug ---> Tlow = ", Tlow
#           print "Debug ---> NuclH = ", NuclH
#           print "Debug ---> xsdir.libs[NuclH][ExtH]: ", xsdir.libs[NuclH][ExtH]
#           print "Debug ---> NuclL = ", NuclL
#           print "Debug ---> xsdir.libs[NuclL][ExtL]: ", xsdir.libs[NuclL][ExtL]
        Plow = (math.sqrt(Thigh)-math.sqrt(Temperature))/(math.sqrt(Thigh)-math.sqrt(Tlow))
        Phigh = 1 - Plow

        return "     %s  %.5E $%.2f%% @ %.3E MeV\n      %s  %.5E $%.2f%% @ %.3E MeV\n" \
                %(ZAIDL, Density*Plow, 100*Plow, Tlow, ZAIDH, Density*Phigh, 100*Phigh, Thigh)
    # --------------- End of function Pseudo() -----------------------------}}}2

    def Cells(self, debug=False):       # {{{2
        if debug:
            print 'Debug ---> I am in Input.Cells'
        cells = []
        for entry in self.lines[1:self.CellsEnd]:
            words = string.split(entry)
            # Exclude comment lines
            if words[0] != 'c' and words[0] != 'C' and words[0] != '$':
                cells.append(entry)
        return cells
    # --------------- End of function Cells() -----------------------------}}}2

    def Surfaces(self, debug=False):            # {{{2
        if debug:
            print 'Debug ---> I am in Input.Cells'
        surfaces = []
        for entry in self.lines[self.CellsEnd+1:self.SurfacesEnd]:
            words = string.split(entry)
            # Exclude comment lines
            if words[0] != 'c' and words[0] != 'C' and words[0] != '$':
                surfaces.append(entry)
        return surfaces
    # --------------- End of function Surfaces() -----------------------------}}}2
```

```
    def Data(self, debug=False):        # {{{2
        if debug:
            print 'Debug ---> I am in Input.Cells'
        data = []
        for entry in self.lines[self.SurfacesEnd+1:len(self.lines)]:
            words = string.split(entry)
            # Exclude comment lines
            if words[0] != 'c' and words[0] != 'C' and words[0] != '$':
                data.append(entry)
        return data
    # --------------- End of function Surfaces() --------------------------}}}2

# --------------- End of class Input() -------------------------------}}}1

class xsdir(object):    #{{{1
    """This class handles all of the data and methods for reading
    the xsdir file."""

    def __init__(self, Datapath, debug=False):
        self.xsdir = file(Datapath, 'r')        # File object
        self.lines = self.xsdir.readlines()
        self.directory = self.lines.index('directory\n')
        self.libs = {}          # Dictionary to hold available librairies and temperatures
        if debug:
            print "Debug ---> xsdir path: ", Datapath

        # Parse the xsdir file to find the cross-section libraries that are available for use
        for line in self.lines[self.directory+1:]:
            words = string.split(line)
            (Nuclide, Extension) = string.split(words[0],'.')
            if (Extension[-1] != 'c' and Extension[-1] != 'd' and Extension[-1] != 't'):
                continue
            Temperature = float(words[9])
            if not self.libs.has_key(Nuclide):
                self.libs[Nuclide] = {}
            self.libs[Nuclide][Extension] = Temperature
            self.libs[Nuclide][Temperature] = Extension

# --------------- End of class xsdir() ----------------------------------}}}1

def Doppler(Card, XSPath, XS, debug=False):      # {{{1
    """This function will run the DOPPLER code.  It requires as input the values
    from the MCNP doplr card with Action equal to "Doppler" or "Extrapolate",
    the location of the original cross sectin files, and information about the
    contents of the original xsdir file.
    Card = dictionary of the lines from the MCNP doplr card
    XSPath = path to original xsdir file and cross sections
    XS = dictionary of all information from original xsdir file."""

    # Create  dopl.in, the file for input to DOPPLER
    DoplrIn = file('dopl.in', 'w+')
    DoplrIn.write('path=%s/\n' %XSPath)            # Write path

    if debug:
        print "path=%s/\n" %XSPath            # Write path
        print "Debug ---> Path: ", Path
```

```
    print "Debug ---> XSPath: ", XSPath
    print "Debug ---> Doppler(Card, ...) Card: ", Card
for entry in Card:
    ZAID1 = False
    ZAID2 = False
    if debug:
        print 'Entry: ', entry
    # Check to see if the ZAIDs exist in the original xsdir file
    if entry['Action'] == 'Interpolate':
        # Check ZAID1
        (Nucl1, Ext1) = string.split(entry['ZAID1'], '.')
        if XS[Nucl1].has_key(Ext1):
            if debug:
                print 'XS %s has_key(%s)' %(Nucl1, Ext1)
            ZAID1 = True
        else:
            if debug:
                print 'XS %s does not have key(%s)' %(Nucl1, Ext1)

        # Check ZAID2
        (Nucl2, Ext2) = string.split(entry['ZAID2'], '.')
        if XS[Nucl2].has_key(Ext2):
            if debug:
                print 'XS %s has_key(%s)' %(Nucl2, Ext2)
            ZAID2 = True
        else:
            if debug:
                print 'XS %s does not have key(%s)' %(Nucl2, Ext2)
        if ZAID1 and ZAID2:
            # Make sure the temperature of ZAID1 is smaller than that of ZAID2
            # and their temperatures bracket the temperature of NewZAID
            if XS[Nucl1][Ext1] < XS[Nucl2][Ext2]:
                if debug:
                    print 'Debug ---> XS[Nucl1][Ext1] < XS[Nucl2][Ext2]'
                    print 'Debug ---> %10.5E < %10.5E' \
                        %(XS[Nucl1][Ext1],XS[Nucl2][Ext2])
                if entry['Temperature'] < XS[Nucl2][Ext2] and XS[Nucl1][Ext1] < \
                    entry['Temperature'] :
                    DoplrIn.write('%s  %s  %10.5E  %s\n' %(entry['ZAID1'], \
                        entry['ZAID2'], entry['Temperature'], entry['NewZAID']))

                    if debug:
                        print "Debug ---> entry['Temperature'] < XS[Nucl2][Ext2] "
                        print "           and XS[Nucl1][Ext1] < entry['Temperature']"
                        print "Debug ---> %10.5E < %10.5E" \
                            %(entry['Temperature'],XS[Nucl2][Ext2])
                else:
                    print "\nTemperature of interest is not bracketed properly."
                    print "Entry: ", entry
                    print "Temp[ZAID1]: %10.5E,  Temp[ZAID2]: %10.5E" \
                        %(XS[Nucl1][Ext1],XS[Nucl2][Ext2])
                    if debug:
                        print "Debug ---> entry['Temperature'] < XS[Nucl2][Ext2]"
                        print "Debug ---> %10.5E < %10.5E" \
                            %(entry['Temperature'],XS[Nucl2][Ext2])
            else:
```

```
                    print "ZAID1's temperature must be less than ZAID2's temperature."
                    if debug:
                        print "Temperature = ", entry['Temperature']
                        print "XS[Nucl2][Ext2]: ", XS[Nucl2][Ext2]
            else:
                print """I can't Doppler Broaden with these libraries because
                      one of them doesn't exist"""

        else:           # entry['Action'] == 'Extrapolate'
            # Check ZAID1
            (Nucl1, Ext1) = string.split(entry['ZAID1'], '.')
            if XS[Nucl1].has_key(Ext1):
                if debug:
                    print 'XS %s has_key(%s)' %(Nucl1, Ext1)
                ZAID1 = True
            else:
                if debug:
                    print 'XS %s does not have key(%s)' %(Nucl1, Ext1)

            # Check Temperature of ZAID1 as compared to Temperature of interest
            if XS[Nucl1][Ext1] < entry['Temperature']:
                DoplrIn.write('%s  0  %10.5E  %s\n' \
                    %(entry['ZAID1'], entry['Temperature'], entry['NewZAID']))
                if debug:
                    print "I can Extrapolate."
            else:
                print "I can't Extrapolate because the temperature of ZAID1 is"
                print "less than the temperature of interest."

    DoplrIn.write('end\n')                      # Finish the file
    DoplrIn.close()
    os.system('doppler < dopl.in')

# --------------- End of function Doppler() ----------------------------------}}}1

main()
```