LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-TR-230194

TNT Prout-Tompkins Kinetics Calibration with PSUADE

A. P. Wemhoff, H. Hsieh

April 23, 2007

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Abstract

We used the code PSUADE to calibrate Prout-Tompkins kinetic parameters for pure recrystallized TNT. The calibration was based on ALE3D simulations of a series of One Dimensional Time to Explosion (ODTX) experiments. The resultant kinetic parameters differed from TNT data points with an average error of 28%, which is slightly higher than the value of 23% previously calculated using a two-point optimization. The methodology described here provides a basis for future calibration studies using PSUADE. The files used in the procedure are listed in the Appendix.

Table of Contents

Introduction 5

Methodology 7
- Step 1: Problem Setup 7
- Step 2: Discretization Method 8
- Step 3: Response Surface Generation 8
- Steps 4 and 5: Response Surface Examination 9
- Step 6: Parameter Optimization 11

Results and Conclusions 12

References 14

Appendix A: Contents of PSUADE Input File `odtxrun` 15
Appendix B: Contents of PSUADE Driver File `driver.py` 16
Appendix C: Contents of Batch Script Template `b0_odtx.Tmplt` 23
Appendix D: Contents of `runodtx.pl` 24
Appendix E: Contents of `getresults.pl` 29
Appendix F: Contents of PSUADE Input File `odtxoptMinpack` 33
Appendix G: Contents of PSUADE Driver File `driveropt.py` 34

Introduction

A large number of engineering problems involve calibration, which is the adjustment of a set of parameters in a computational model to maximize model agreement with experimental data. These problems are especially relevant in cookoff modeling of of high explosives (HE), for several aspects of these materials are based on empirical models (e.g. material properties and chemical kinetics). Until recently, few options were available for this calibration procedure, although the LLNL code GLO [1] had previously been used to fine-tune HE kinetic parameters [2-4].

Model calibration is different from model validation in that the latter quantifies the belief/confidence in the predictive capability of a computational code through comparison with a set of experimental measurements. One can see that both calibration and validation are often tedious even the problem is well posed. In this study, we only consider calibration. If one defines the discrepancy between model prediction and experimental measurement as an objective function, then the calibration can be achieved simply by minimizing the objective function. In this report, two approaches were used to calibrate the model by minimizing the model discrepancy. The code PSUADE (Problem Solving Environment for Uncertainty Analysis and Design Exploration) provides a useful tool to facilitate this calibration process in that it provides the following advantages:

- Discretization of the *n*-dimensional input surface according to a variety of design schemes (or design of experiments).
- Sensitivity analysis of each of the input parameters on any output parameter
- Multiple output parameters
- Multiple output parameter optimization using the MinPack toolkit.
- Visualization of the response surface
- Batch submission for long-running and multi-processor analyses.

PSUADE has previously been used in complex analyses [5], and this study makes use of only a small portion of the available features in the code. Here, we provide a methodology by which PSUADE was used to adjust the parameters associated with the Prout-Tompkins chemical kinetic model [6],

$$-\frac{dx}{dt} = A\exp\left(-\frac{E}{RT}\right)x^n(1-qx)^m \tag{1}$$

where x is the mass fraction of unreacted HE. For simplicity, we only use a single reaction cookoff model here. The parameter q is very close to one and is therefore commonly written in terms of a parameter p for clarity,

$$p = -\log_{10}(1-q) \tag{2}$$

This reaction model states that the rate of reaction is related to the amount of product formed, which would be true if the reaction products attack the unreacted material at a rate faster than it decomposes by itself.

Calibration of the Prout-Tompkins parameters *A*, *E*, *m*, *n*, and *p* were performed based on One Dimensional Time to Explosion (ODTX) experimental data [7, 8]. In an ODTX apparatus, the explosive is first pressed into a 1/2"-diameter sphere, and then the explosive is heated isothermally until ignition. Experimental values were run at 23 temperatures as shown in Figure 1. For the optimization, 7 representative data points were chosen (although all 23 points could be chosen instead if desired).

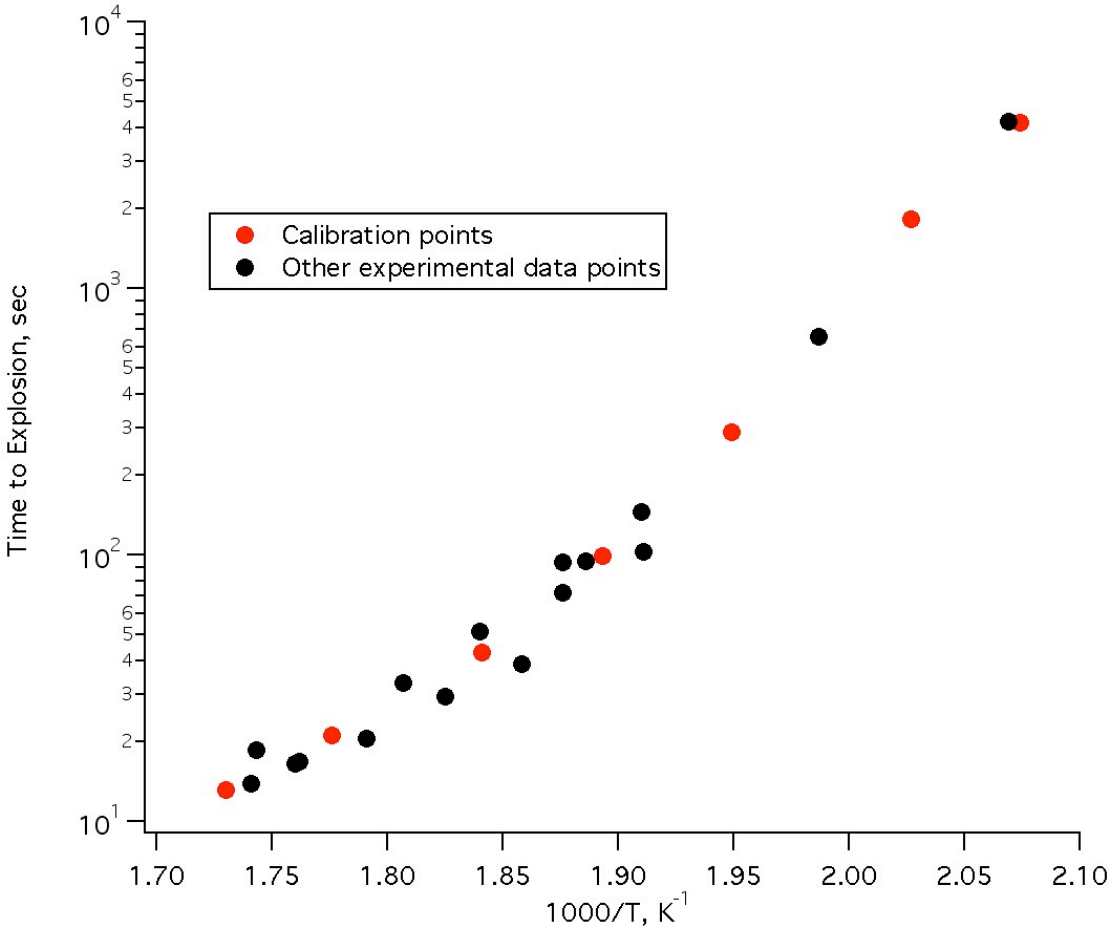

Figure 1. ODTX experimental data for pure recrystallized TNT. Red data points were used for Prout-Tompkins parameter calibration.

The material properties used for pure TNT are provided in Table 1 below. These properties were taken from a previous report [3]. Note that melting and other phase transitions were ignored in the cookoff simulations.

Table 1. TNT properties used in simulations.

Property	Value
Density	1.654 g/cc
Energy of reaction	950 cal/g
Solid heat capacity	1.37 J/(g-K)
Solid thermal conductivity	0.26 W/(m-K)
Products heat capacity	1.0 J/(g-K)*
Products thermal conductivity	0.05 W/(m-K)*
* Estimated value.	

Methodology

To calibrate parameters using PSUADE, the following steps are used:

1. Determine what output is to be optimized, which input parameters should be varied, and what the ranges of the input parameters should be.
2. Choose an appropriate design of experiments for creating the response surface.
3. Run PSUADE to generate the response surface
4. Examine the response surface to ensure that a global minimum lay on the interior of the response surface as opposed to the edges.
5. Examine the sensitivity of the output on each of the input parameters.
6. Run PSUADE with MinPack to optimize the parameters around the minimum on the response surface.

We will now go through each of the above steps to show how the parameters were calibrated.

Step 1: Problem Setup

The input parameters to be varied are already known in the Prout-Tompkins model: A, E, m, n, and p. PSUADE needs two sets of output parameters to be known: one set with multiple output parameters for MinPack optimization, and one set of a single condensed Figure of Merit (FOM). Here, the FOM is chosen as

$$FOM = \frac{1}{7}\sum_{i=1}^{7}\left(\ln\left(t_{sim,i} \middle/ t_{expt,i}\right)\right)^2 \tag{3}$$

where t_{sim} and t_{exp} are the simulated and experimental times to explosion, respectively. The difference between $t_{sim,i}$ and $t_{expt,i}$ for all values of i was chosen as the set with multiple output parameters. Note that the FOM calculated here is slightly different than the one calculated by MinPack, but they are proportional so the trends are identical.

The ranges for the input parameters are shown in Table 2. These values encompass the values generally encountered when applied for a variety of explosives [3].

Table 2. PSUADE input parameter ranges.

Parameter	Units	Minimum Value	Maximum Value
E	cal/mol	10000	70000
$\ln(A)$	$\ln(us^{-1})$	1	40
m	-	0	2
n	-	0	2
p	-	2	9

Step 2: Design of Experiments

Several design schemes are available in PSUADE. In this study, we chose METIS since the computational cost of an individual run was modest (generally only a few minutes). We originally used 100 runs with METIS to obtain a coarse evenly-spaced response surface, but would later add an additional 100 runs with the `randomize` and `randomize_more` options in the PSUADE input file to "fill-in" holes between these discrete points. The perlscript `combresults.pl` was created to combine output data from multiple PSUADE runs into a single output file.

Appendix A lists the contents of the PSUADE input file `odtxrun` featuring the input variables and the discretization method.

Step 3: Response Surface Generation

The run sequence of PSUADE is as follows:

1. PSUADE provides a set of input parameters.
2. The Python script `driver.py` (Appendix B) reads in the parameters and does the following for the first run (similar approaches apply for other runs):
    a. It creates a working directory `workdir.1`
    b. It copies the files `b0_odtx.Tmplt` (Appendix C), `runodtx.pl` (Appendix D), `data.txt`, and the input parameters file (`psuadeApps_ct.in.1`) into the newly created working directory. The file `b0_odtx.Tmplt` is a batch script template file, `runodtx.pl` is the main perlscript, and `data.txt` is a listing of ODTX data points for analysis.
    c. The batch script `b0_odtx` is created by copying `b0_odtx.Tmplt` and replacing strings with the input parameter names. The input variables are fed as command-line arguments to `runodtx.pl`.
    d. The batch script `b0_odtx` is submitted to the batch system. If there are more than 8 such submissions in the system already, then dependencies are used. This approach is done by the command `pstat` > `stat`, opening the file `stat`, and counting the number of times `b0_odtx` appears.
3. When the batch process begins, the perlscript `runodtx.pl` performs the following:

a. It reads in the ODTX data points from `data.txt` and calculates the appropriate time step size based on the experimental time-to-explosion.
b. It copies the ALE3D input decks and SAMI file from the base directory into the working directory.
c. For each of the 7 data points, it runs ALE3D using the `–def` command-line argument to place the input parameters into the simulation. The simulation is run until the simulated time exceeds double the experimental time, the number of time steps exceeds 100,000, or thermal runaway forces the simulation to terminate.
d. It reads the simulated explosion time from the ALE3D log file and stores it in the output files `odtx-lx10.res` (multiple output) and `fom` (single output).

4. After all PSUADE data points have been run, the perlscript `getresults.pl` (Appendix E) is used to create the PSUADE output files `psuadeOut` (multiple output) and `psuadeFom` (single output).

Step 4 and 5: Response Surface Examination

PSUADE has a handy built-in surface plotter feature that allows for examination of the response surface using MATLAB. To use this feature, the two most sensitive input parameters should be used as the axes. The following approach did the sensitivity analysis:

1. Open PSUADE.
5. Type load `psuadeFom`
2. Type `printlevel 3`
3. Type `rscheck`, `0`, `0`
4. PSUADE provides a MARS (Multi-variate Adaptive Regression Spline) importance measure for each input variable (normalized to 100). The two most sensitive parameters are E (100) and ln(A) (77), while the remaining parameters m (38), n (12), and p (8) are less sensitive. The dominance of E and ln(A) is not surprising since they are independent of mass fraction, and this feature has been applied in previous studies to "pin" the simulated curve to three points [2-4].

The response surface may then be created using E and ln(A) as the input parameter axes. PSUADE creates this surface with the following commands:

1. Type `rs`
2. Use input parameters 1 and 2 for the axes since E is 1 and ln(A) is 2 here.
3. Let PSUADE set the other nominal values automatically and choose the threshold values.
4. Close PSUADE, open MATLAB, and type `matlabrs` to create the response surface.

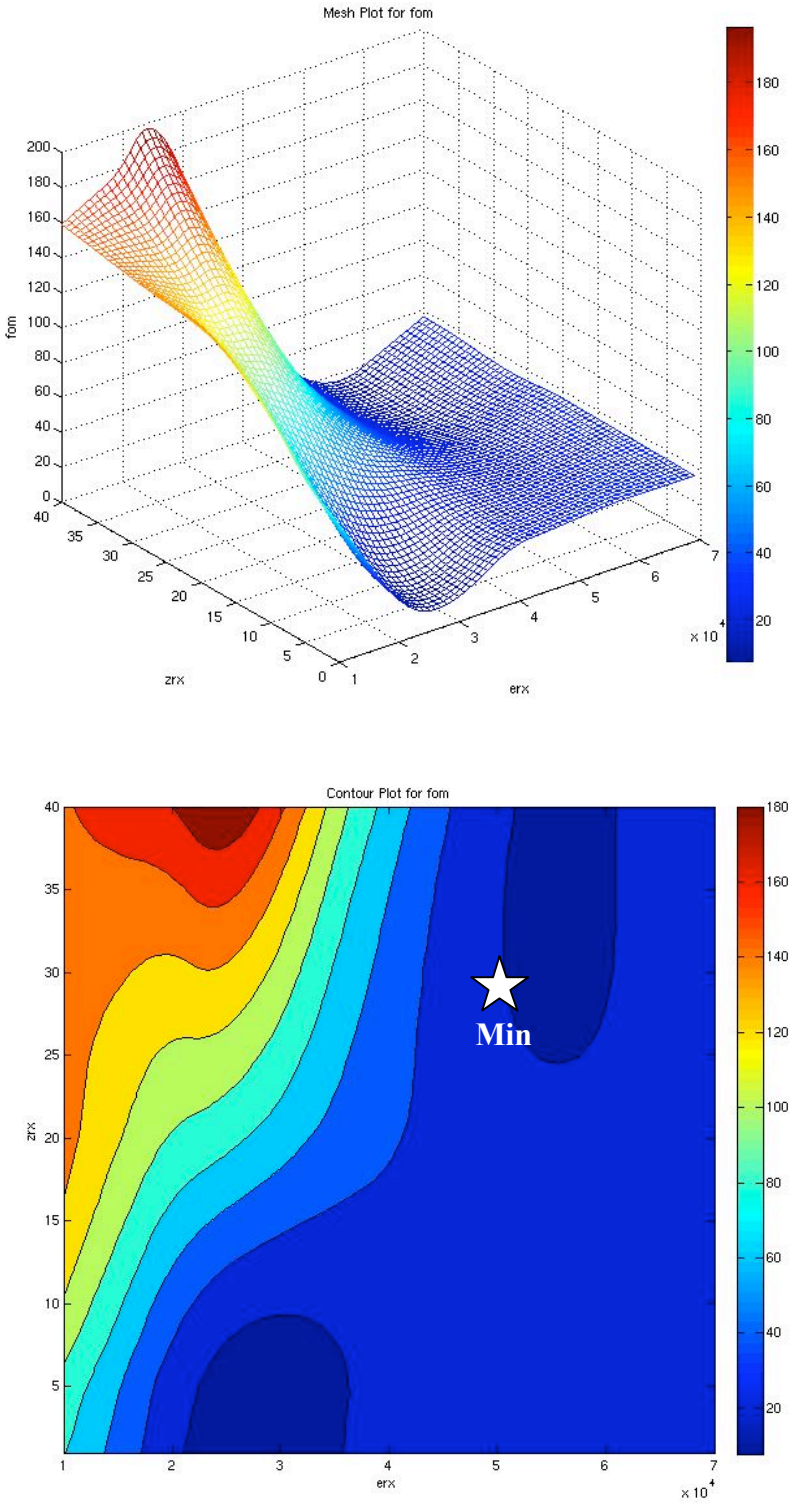

Figure 2. Calculated response surface for *E* (*erx*) and ln(*A*) (*zrx*). The minimum data point lies at 50,372 and 29.38.

The minimum FOM value can be chosen by running PSUADE and typing the command `min`. The response surface for this example, along with the minimum value, is shown below in Figure 2. The primary purpose of these plots is to show that the minimum point falls within the range of input values. The plots shown in this figure are merely spline curve fits to the FOM values and therefore do not capture the FOM values themselves. The resultant plots do not contain enough accuracy to allow for visually choosing a minimum value, but rather they provide an overall qualitative depiction of the response surface. For this reason, the actual minimum data point for this example problem does not appear to be located at the minimum location suggested graphically in the figure, although this point is close to the minimum contour in the figure.

Step 6: Parameter Optimization

The parameters of the Prout-Tompkins cookoff chemical kinetic model are adjusted in two ways. The first approach relies on the response surface that is created from previous steps. The model discrepancy is minimized using the response surface. The second approach involves a point-by-point search. The first approach is a very quick and simple technique but did not provide good results in this case. Therefore, the point-by-point search approach was used. PSUADE performs this optimization in the following manner:

1. The minimum data point on the calculated response surface was used as the starting point for the optimization.
2. The parameter ranges were set as +/- 10% of the starting point values.
3. Monte Carlo was used as the sampling technique.
4. For each run, the following occurred:
    a. PSUADE checked the system every 10 seconds to determine if there are any running jobs. If not, then it grabbed the next data point based on the results of the previous job.
    b. PSUADE provided a sample data point, created a working directory, copied files into this directory, and created the batch script `b0_odtx` in the same manner as for creating the response surface.
    c. The batch script was run and the output values were stored.

Appendices F and G provide the PSUADE input file `odtxoptMinpack` and driver file `driveropt.py` used in the optimization routine.

The perlscript `getresults.pl` was used to observe how the FOM was changed with each successive run. The lowest value of the FOM occurred in run 64 of 75 as shown in Figure 3. Note that the optimization routine reduced the FOM by a factor of five from the initial starting value.

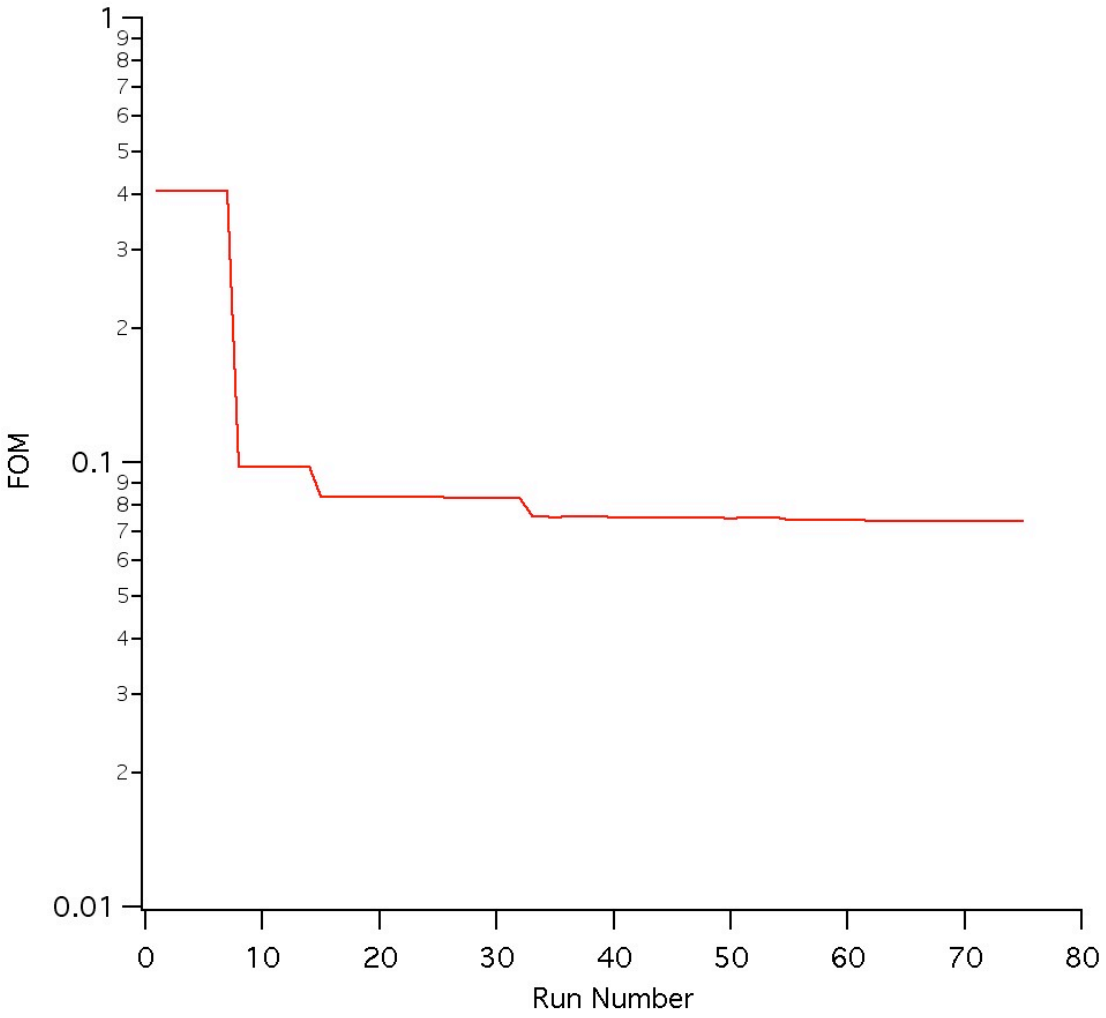

Figure 3. Adjustment of the FOM using MinPack optimization.

Results and Conclusions

The calibrated parameters for TNT are shown in Table 3. Figure 4 provides a plot comparing the simulated and experimental ODTX data for all data points. The average error is 28% for the data, which corresponds to a calculated FOM of 0.093 per Eq. (3) except the averaging is done over all points. For comparison, Figure 4 also provides results for the application of the two-point calibration technique [3], which gives an average error of 23% and a calculated FOM of 0.088. Therefore, the minimum value on the response surface found by PSUADE is not the global minimum. It is anticipated that further discretization of the response surface would allow for determination of the global minimum, although the amount of discretization required is uncertain due to the coupling of Prout-Tompkins parameters. Nevertheless, we have shown that using PSUADE in the described manner does allow for determination of a local minimum that is close to the previously calibrated value.

Table 3. Calibrated Prout-Tompkins parameters for TNT.

Parameter	Value (PSUADE)	Value (2pt calib [3])
E/R	21494 K	19024 K
A	4.48×10^{15} s^{-1}	7.89×10^{14} s^{-1}
m	0.646056	1.0
n	1.94178	0.0
p	4.79118	9.0

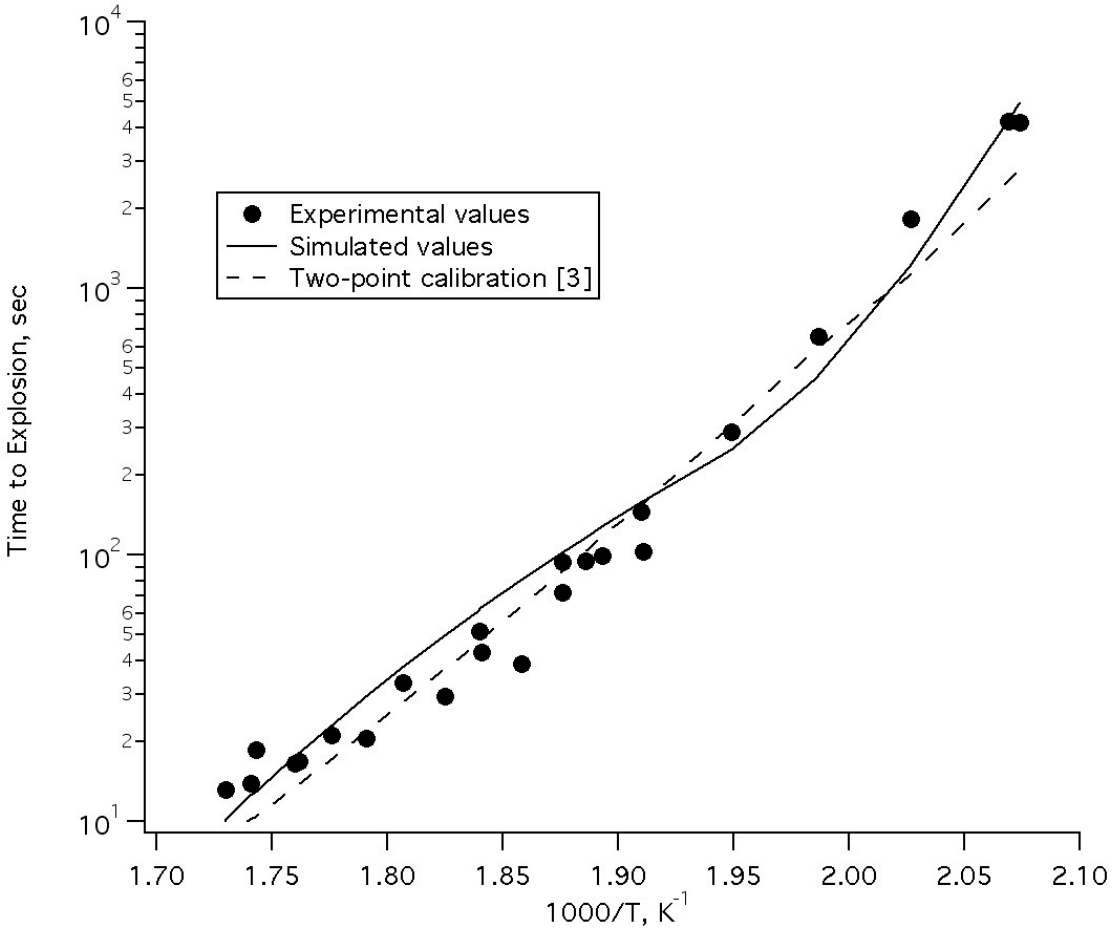

Figure 4. Simulated and experimental ODTX values.

We have shown that PSUADE can be a useful tool to calibrate parameters for in empirical models. In order for this optimization to be useful, however, the following needs to be true:

- The simulation code (such as ALE3D) used must have a text input deck and can run without a GUI.
- The users must have knowledge of Perl or Python (or any general regular expression language) to create the appropriate script to run a series of codes.

- The code should be available to be run on OCF since batch submission is used. The simulations shown here were run on yana.llnl.gov.

References

1. Murphy, M. J., 1999, GLO - Global Local Optimizer User's Manual, Report UCRL-MA-133858, *Lawrence Livermore National Laboratory*.

2. Wemhoff, A. P., Burnham, A. K., and Nichols, A. L., 2007, The Application of Global Kinetic Models to HMX Beta-Delta Transition and Cookoff Processes, *J. Phys. Chem. A*., Vol. 111, pp. 1575-1584.

3. Wemhoff, A. P., and Burnham, A. K., 2006, Calibration Methods for ODTX Chemical Kinetics for Various Explosives, Report UCRL-TR-222032, *Lawrence Livermore National Laboratory*.

4. Wemhoff, A. P., Burnham, A. K., Nichols III, A. L., Knap, J., 2006, Calibration Methods for the Extended Prout-Tompkins Chemical Kinetics Model and Derived Cookoff Parameters for RDX, HMX, LX-10 and PBXN-109, Report UCRL-CONF-226426, *Lawrence Livermore National Laboratory*.

5. Hsieh, H., 2006, Application of the PSUADE Tool for Sensitivity Analysis of an Engineering Simulation, Report CODTU-2006-0029, *Lawrence Livermore National Laboratory*.

6. Burnham, A. K., Weese, R. K., and Weeks, B. L., 2004, A Distributed Activation Energy Model of Thermodynamically Inhibited Nucleation and Growth Reactions and Its Application to the b-d Phase Transition of HMX, *J. Phys. Chem. B*, Vol. 108, pp. 19432-19441.

7. Catalano, E., McGruire, R., Lee, E., Wrenn, E., Ornellas, D., and Walton, J., 1976, The Thermal Decomposition and Reaction of Confined Explosives, presented at *the Sixth Symposium on Detonation*.

8. Tran, T. D., Simpson, L. R., Maienschein, J., and Tarver, C., 2001, Thermal Decomposition of Trinitrotoluene (TNT) with a New One-Dimensional Time to Explosion (ODTX) Apparatus, presented at *the 32nd International Annual Conference of Fraunhofer-Institut fur Chemische Technologie (ICT)*.

Appendix A: Contents of PSUADE Input File ***odtxrun***

```
PSUADE
INPUT
   dimension = 5
   variable   1 erx =   1.00000000e+04   7.00000000e+04
   variable   2 zrx =   1.00000000e+00   4.00000000e+01
   variable   3 m   =   0.00000000e+00   2.00000000e+00
   variable   4 n   =   0.00000000e+00   2.00000000e+00
   variable   5 p   =   2.00000000e+00   9.00000000e+00
END
OUTPUT
   dimension = 7
   variable   1 fom1
   variable   2 fom2
   variable   3 fom3
   variable   4 fom4
   variable   5 fom5
   variable   6 fom6
   variable   7 fom7
END
METHOD
   sampling = METIS
   num_samples = 100
END
APPLICATION
   driver = driver.py
END
ANALYSIS
   diagnostics 2
END
END
```

Appendix B: Contents of PSUADE Driver File driver.py

```
#!/usr/local/bin/python
# sys.argv[1] - input file
# sys.argv[2] - output file

import os
import sys
import string
import shutil
import glob

#############################################################################
###
#############################################################################
###
# Function to get input data from PSUADE input file
#----------------------------------------------------------------------------
---
def getInputData(inFileName, nInputs):
   if nInputs <= 0:
      inputData = []
      return inputData
   inFile = open(inFileName, "r")
   inputData = range(nInputs)
   lineIn = inFile.readline()
   count = 0
   while 1:
      lineIn = inFile.readline()
      ncols = string.split(lineIn)
      inputData[count] = eval(ncols[0])
      count = count + 1
      if count >= nInputs:
         break
   return inputData

#############################################################################
###
# Function to generate input file for the application
#----------------------------------------------------------------------------
---
def
genApplicationInputFile(inputData,appTmpltFile,appInputFile,nInputs,
                        inputNames):
   infile = open(appTmpltFile, "r")
   outfile = open(appInputFile, "w")

# added to provide appropriate directory
   searchString = "PSUADE_COUNTER"
   while 1:
      lineIn  = infile.readline()
      if lineIn == "":
         break
      lineLen = len(lineIn)
      newLine = lineIn
      if nInputs > 0:
         for fInd in range(nInputs):
```

```
            strLen = len(inputNames[fInd])
            sInd = string.find(newLine, inputNames[fInd])
            if sInd >= 0:
               strdata = str(inputData[fInd])
               next = sInd + strLen
               lineTemp = newLine[0:sInd] + strdata +
newLine[next:lineLen]
               newLine = lineTemp

# added to provide appropriate directory
      sInd2    = string.find(newLine, searchString)
      strLen = len(searchString)
      if sInd2 >= 0:
         next = sInd2 + strLen
         strdata = str(fileTag)
         lineTemp = newLine[0:sInd2] + strdata + newLine[next:lineLen]
         newLine = lineTemp
      outfile.write(newLine)
   infile.close()
   outfile.close()
   return

#########################################################################
###
# Function to generate the batch file
#-------------------------------------------------------------------------
---
def genBatchFile(batchTmpltFileName, batchFileName):
   infile = open(batchTmpltFileName, "r")
   outfile = open(batchFileName, "w")
   searchString = "PSUADE_COUNTER"
   strLen = len(searchString)
   while 1:
      lineIn  = infile.readline()
      if lineIn == "":
         break
      lineLen = len(lineIn)
      newLine = lineIn
      sInd    = string.find(newLine, searchString)
      if sInd >= 0:
         next = sInd + strLen
         strdata = str(fileTag)
         lineTemp = newLine[0:sInd] + strdata + newLine[next:lineLen]
         newLine = lineTemp
      outfile.write(newLine)
   infile.close()
   outfile.close()
   return

#########################################################################
###
# Function to run batch file
#-------------------------------------------------------------------------
---
def runApplication(batchFileName):
#  sysCommand = "/usr/local/bin/psub " + batchFileName
#  os.system(sysCommand)
#  statCommand = "/usr/local/bin/pstat | /bin/grep " + batchFileName
```

```
#  status = os.system(statCommand)
#  while status == 0:
#     os.system("sleep 60")
#     status = os.system(statCommand)
   command = "echo Exact_Tran 10.0 > ssdns.info"
   os.system(command)
   command = "echo Exact_Refl 20.0 >> ssdns.info"
   os.system(command)
   return

#########################################################################
###
# Function to run using srun
#-------------------------------------------------------------------------
---
def runApplicationSrun(fileTag,batchFile):
   sysCommand = "psub "+batchFile
   os.system(sysCommand)
   return

#########################################################################
###
# Alternate function to submit batch job (uses dependencies to limit
hogging the processors)
#-------------------------------------------------------------------------
---
def runApplication(executable):
   dependString = ""
   tagString = os.path.splitext(executable)[1]
   stringLen = len(tagString)
   runNumber = eval(tagString[1:stringLen])
#   lastNo    = runNumber - 10
   lastNo    = runNumber - 2
   prefix    = os.path.splitext(executable)[0];
   lastJob   = prefix + "." + str(lastNo) + " "
   statComm  = "/usr/local/bin/pstat -D | /bin/grep "
   statComm  = statComm + lastJob + " > stat"
   status = os.system(statComm);
   dependString = ""
   if status == 0:
      statFile = open("stat", "r")
      lineIn = statFile.readline()
      sIndex = string.find(lineIn, usrName)
      if sIndex > 0:
         sIndex = string.find(lineIn, lastJob)
         if sIndex > 0:
            dependString = "-d " + lineIn[0:sIndex-1]
      statFile.close()
   sysComm = "/usr/local/bin/psub " + dependString + executable
   os.chmod(executable,448)
   os.system(sysComm)
   return

#########################################################################
###
# Alternate function to submit batch job (uses dependencies to limit
hogging the processors) - APW
```

```
#-----------------------------------------------------------------------
---
#def runApplicationAPW(batchFile,fileTag):
#   CRuns = 8;
#   if (fileTag <= CRuns):
#      sysCommand = "psub "+ batchFile
#      print str(fileTag) + ":" +  sysCommand
#      os.system(sysCommand)
#   else:
#      lastNo = fileTag - CRuns;
#      currNo = 0;
#      sysCommand = "pstat -D > stat";
#      os.system(sysCommand)
#      statFile = open("stat", "r")
#      lineIn = statFile.readline()
#      while 1:
#         lineIn = statFile.readline()
#         sIndex = string.find(lineIn, batchFile)
#         if sIndex > 0:
#            currNo = currNo + 1;
#            runNo = eval(lineIn[0:sIndex-1]);
#            if currNo == lastNo:
#               statFile.close()
#               sysCommand = "psub -d "+ str(runNo) + " " + batchFile;
#            print str(fileTag) + ":" +  sysCommand
#               os.system(sysCommand)
#               break
#   return

def runApplicationAPW(batchFile,fileTag):
   CRuns = 8;
# count up the number of previously submitted batch jobs
   sysCommand = "pstat -D > stat";
   os.system(sysCommand)
   statFile = open("stat", "r")
   currNo = 0;
   runNo = range(99)
   lineIn = statFile.readline()
   while 1:
      lineIn = statFile.readline()
      if lineIn == "":
         break
      sIndex = string.find(lineIn, batchFile)
#      print str(fileTag) + ": sIndex = " + str(sIndex) + ", lineIn = "
+ lineIn
      if sIndex > 0:
         currNo = currNo + 1;
         runNo[currNo] = eval(lineIn[0:sIndex-1]);
#         print "runNo[" + str(currNo) + "] = " + str(runNo[currNo])
   statFile.close()
#   print str(fileTag) + ":" + str(currNo) + " instances."

   if currNo <= CRuns:
      sysCommand = "psub "+ batchFile
      print str(fileTag) + ":" +  sysCommand
      os.system(sysCommand)
   else:
      numUse = currNo - CRuns
```

```
#      print "numUse = " + str(numUse)
       sysCommand = "psub -d "+ str(runNo[numUse]) + " " + batchFile;
       print str(fileTag) + ":" +  sysCommand
       os.system(sysCommand)
   return


#########################################################################
###
# Function to get output file from the application
#------------------------------------------------------------------------
---
def getApplicationOutputData():
   searchString = ""
   infile = open("odtx-lx10.res", "r")
   outData = range(1)
   while 1:
      lineIn = infile.readline()
      if lineIn == "":
         break
      sInd = string.find(lineIn, searchString)
      if sInd >= 0:
         ncols = string.split(lineIn)
         outData[0] = eval(ncols[1])
   infile.close()
   return outData

#########################################################################
###
# Function to get data from ultra file
#------------------------------------------------------------------------
---
def getUltraOutputData(ultraFileName, searchString):
   endString = "end"
   infile = open(ultraFileName, "r")
   nCount = 0
   while 1:
      lineIn = infile.readline()
      if lineIn == "":
         break
      sInd = string.find(lineIn, searchString)
      if sInd >= 0:
         while 1:
            lineIn = infile.readline()
            sInd = string.find(lineIn, endString)
            if sInd < 0:
               break;
            nCount = nCount + 1
   infile.close()
   outData = range(nCount)
   infile = open(ultraFileName, "r")
   while 1:
      lineIn = infile.readline()
      if lineIn == "":
         break
      sInd = string.find(lineIn, searchString)
      if sInd >= 0:
         while 1:
```

```
            lineIn = infile.readline()
            sInd = string.find(lineIn, endString)
            if sInd < 0:
               break;
            ncols = string.split(lineIn)
            outData[nCount] = eval(ncols[1])
            nCount = nCount + 1
   infile.close()
   return outData

##########################################################################
###
# Function to generate output file from the application
#--------------------------------------------------------------------------
---
def genOutputFile(outFileName, outData):
   nLeng = len(outData)
   outfile = open(outFileName, "w")
   for ind in range(nLeng):
      outfile.write("%e \n" % outData[ind])
   outfile.close()
   return

##########################################################################
###
## Main program
##########################################################################
###

# --------------------------------------------------------------------------
---
# fetch PSUADE input and output file names
# --------------------------------------------------------------------------
---
inFileName  = sys.argv[1]
outFileName = sys.argv[2]
# --------------------------------------------------------------------------
---
# if output file already exists, do not perform test
# --------------------------------------------------------------------------
---
testFlag = 1
if os.path.isfile(outFileName) != 0:
   testFlag = 0
   exit
# --------------------------------------------------------------------------
---
# extract the sample number -> fileTag
# --------------------------------------------------------------------------
---
tagString = os.path.splitext(inFileName)[1]
stringLen = len(tagString)
fileTag   = eval(tagString[1:stringLen])

# --------------------------------------------------------------------------
---
# application specific settings
```

```
# ------------------------------------------------------------------------
---
#appInputTmpltFile = "b0_odtx_Tmplt"
#appInputFile = "b0_odtx"
batchTmpltFile = "b0_odtx.Tmplt"
batchFile = "b0_odtx"
nInputs = 5
inputNames = ["eeeeeeeeeeeee", "zzzzzzzzzzzz", "mmmmmmmmmmmmm",
"nnnnnnnnnnnn", "pppppppppppp"]
copyList = [batchTmpltFile, "runodtx.pl", inFileName, "data.txt"]

# ------------------------------------------------------------------------
---
# If working directory already exists, do not perform test. Otherwise,
# create the working directory
# ------------------------------------------------------------------------
---
dirname = "./workdir." + str(fileTag)
if os.path.isdir(dirname):
   testFlag = 0
else:
   os.mkdir(dirname)

# ------------------------------------------------------------------------
---
# copy the needed files to the working directory and enter into it
# ------------------------------------------------------------------------
---
if testFlag == 1:
   for file in copyList:
      shutil.copy(file, dirname)
   os.chdir(dirname)

# ------------------------------------------------------------------------
---
# generate input file, run test and gather results
# ------------------------------------------------------------------------
---
if testFlag == 1:
   inputData = getInputData(inFileName, nInputs)
   genApplicationInputFile(inputData,batchTmpltFile,batchFile,nInputs,
                           inputNames)
#   genBatchFile(batchTmpltFile, batchFile)

#   runApplicationSrun(fileTag,batchFile)
#   runApplication(batchFile)
   runApplicationAPW(batchFile,fileTag)

#   outData = getApplicationOutputData()
   outData=range(1)
   outData[0]=1.0E+35
   genOutputFile(outFileName, outData)
   shutil.copy(outFileName, "..")
#   for file in glob.glob('*'):
#      os.remove(file)
#   os.chdir("..")
#  os.rmdir(dirname)
```

Appendix C: Contents of Batch Script Template b0_odtx.Tmplt

```
#!/bin/csh
#  psub script to run jobs on DEC cluster
#
#  Execute this script by typing "psub scriptname" without quotes and
#     where scriptname is your modified version of this file
#
#PSUB -tM 8h       # time limit in minutes
#PSUB -b me      # default bank from which to draw time
#PSUB -np 1      # run using 1 processor
#PSUB -nr        # no restart
#
# ouptut below commands to log file
set echo
date
cd /p/lscratchb/wemhoff2/psuade/TNT/opt/workdir.PSUADE_COUNTER/
#
perl runodtx.pl -list data.txt -erx eeeeeeeeeeeee -zrx zzzzzzzzzzzzz -m
mmmmmmmmmmmmm -n nnnnnnnnnnnn -p pppppppppppp> out.txt

#
# we're done
#
#end
```

Appendix D: Contents of *runodtx.pl*

```
# PERL script that automatically varies odtx temperaure and stores
output data
#########################################################################
###
# - 2/11/07: based on autovary_odtx.pl; this is a simplified version
that
#            takes new values of erx and zrx from input arguments.  For
the
#            first time I use the -def option on the ale3d command line
#            instead of replacing text in the ale3d input file
# - 3/07/07: output run differences instead of calculating FOM
# - 3/12/07: expand for all PT parameters
# - 3/13/07: add a separate FOM file creation
#
#########################################################################
###

# a 'use' command to allow for parsing of the input line
#-------------------------------------------------------
use Getopt::Long;

# Process command line options
#-----------------------------
my $erxval = 0.;
my $zrxval = 0.;
my $nval = 0.;
my $mval = 0.;
my $pval = 0.;
my $varname = "temp";
my $infile = "odtx-lx10-100.in";
my $help;
my $imax = 1;
my $init = 1.0;
my $delta = 1.0;
my $geom = "0.5sph";
my $stepsize = 1.e5;
my $add;
my $tkelvexpl = 1000.;
my $listfile = "abc123xyz";
my $samifile = "default";
my $time_limit = 0.;
my $gen3dexe = "/usr/apps/ale3d/bin/gen3d";
my $ale3dexe = "/usr/apps/ale3d/bin/ale3d";
my $use_srun;
my $noclean;
my $ignorewarn;
GetOptions('h|help'=>\$help,
         'erx=f'=>\$erxval,
           'zrx=f'=>\$zrxval,
           'm=f'=>\$mval,
           'n=f'=>\$nval,
           'p=f'=>\$pval,
         'list=s'=>\$listfile,
         'in=s'=>\$infile);
```

```
if ($help) {
  print "\n";
  print "use -erx to specify value of ERX in ALE3D input [0.]\n";
  print "use -zrx to specify value of ZRX in ALE3D input [0.]\n";
  print "use -list to specify list of run temperatures and goal values
[off]\n";
  print "use -in to specify input file [odtx-lx10-100.in]\n";
  print "\n";
  exit(0);
}


# copy necessary files into local directory
select(stdout);
`cp ../$infile .`;
`cp ../lx10*.in .`;
`cp ../sphh.sami .`;

# read temperatures from list file
$uselist = 0;
if ($listfile ne "abc123xyz"){
  $uselist = 1;
  open(INFILE,"<$listfile") or die ("Could not open data points file:
$listfile\n");
  my $linect = 0;
  $comment = qr/^\*/;
  $imax = 0;
  while (<INFILE>){
    $linect++;
    if (/$comment/){          # commented line
    } else {
      ($value1,$value2,$value3) = split(/\s+/,$_);
      if (length($value1) == 0){
         $value1 = $value2;
         $value2 = $value3;
      }
      if ((length($value1) == 0)&&(length($value2) == 0)){  #blank line
      } elsif (length($value2) == 0){
         print "** ERROR ** only one value read in line $linect of file
$datfile!\n";
         exit 0;
      } else {
         $imax++;
       $p_use[$imax] = sprintf "%5.1f", $value1;
       $goal[$imax] = sprintf "%10.4e", $value2;
         $stepsz[$imax] = sprintf "%10.4e", getsize($goal[$imax]);
      }
    }
  }
} else {
  print "listfile $listfile not found.\n";
  exit 0;
}
close(INFILE);

if ($uselist == 1){
  print "\nData points read in from file $listfile:\n";
  print "    i   temp(i)        goal(i)    stepsize(i)\n";
```

```
  for ($i = 1; $i <= $imax; $i++){
    @linevals = ($i, $p_use[$i], $goal[$i], $stepsz[$i]);
    print (sprintf("%5d%10.1f%15.3e%15.3e\n",@linevals));
  }
}

foreach $i (1..$imax) {

#   Setup ale3d files and execute ale3d
    select(stdout);
    $f_index[$i] = 100+$i;
    $fdat= "odtx-lx10-$f_index[$i].in";
    $fgen= "odtx-lx10-$f_index[$i]-gen*";
    $frin= "odtx-lx10-$f_index[$i].rin*";
    $fpar= "odtx-lx10-$f_index[$i].params*";
    $frf=  "odtx-lx10-$f_index[$i]_001_*";
    $fplf= "odtx-lx10-$f_index[$i]_001.*";
    $fprob="odtx-lx10-$f_index[$i]";
    $flogw= "odtx-lx10-$f_index[$i].log*";
    $flogaw="odtx-lx10-$f_index[$i].loga*";
    $flog="odtx-lx10-$f_index[$i].log";
    $floga="odtx-lx10-$f_index[$i].loga";
    $frf0= "odtx-lx10-$f_index[$i]_001_00000";
    $fabo= "odtx-lx10-$f_index[$i]_abort*";
    $fthistdir = "timehist.odtx-lx10-$f_index[$i].001";
    $fthist="timehist.odtx-lx10-$f_index[$i].001/tkelv-he-max";
    $fthist2="timehist.odtx-lx10-$f_index[$i].001/nodet.a.0000.102";
    `cp odtx-lx10-100.in $fdat`;
    `rm $fgen`;
    `rm $frin`;
#    `rm $fpar`;
    `rm $frf`;
    `rm $fplf`;
    `rm $fthist`;
     $command = "$gen3dexe -in $fdat -prob $fprob -sami sphh.sami -def
sym_temp $p_use[$i] -def sym_step $stepsz[$i] -def sym_erx $erxval -def
sym_zrx $zrxval -def sym_mval $mval -def sym_nval $nval -def sym_pval
$pval";
     print "command = $command\n";
     `$command`;
    `rm $flog`;
    `rm $flogw`;
    `rm $fabo`;
    if ($linux){`rm *.core`;}
    $command = "$ale3dexe -in $fdat -rf $frf0 -def sym_temp $p_use[$i]
-def sym_step $stepsz[$i] -def sym_erx $erxval -def sym_zrx $zrxval -
def sym_mval $mval -def sym_nval $nval -def sym_pval $pval> $floga";
     print "command = $command\n";
     `$command`;

#   Extract time to explosion
    open(F1,"<$flog");
    $change = 0;
    $expcause = "tstep";
    while (<F1>) {
        if ($change == 0) {
           ($dum, $num, $time, $time0[$i]) = split(/\s+/,$_) if
(/dt:/);
```

```
         if ($num == 999) { $change = 1;}
      } else {
         ($num, $time, $time0[$i]) = split(/\s+/,$_) if (/dt:/);
      }
    }
    close(F1);

    open(F1,"<$fthist");
    $reading = 1;
    while (<F1>) {
      if ($reading == 1){
        ($dum, $time, $tkelvmax) = split(/\s+/,$_) if (/./);
        if ($tkelvmax > $tkelvexpl){
        $time0[$i] = $time;
        $reading = 0;
          $expcause = "thist";
      }
      }
    }
    close(F1);

# added 9/11/06 to extract surface temperature at explosion
    if (($varname eq "stex_dual_ramp")||($varname eq "ramp")){
      open(INFILE,"<$fthist2");
      while (<INFILE>) {
        ($dum, $time, $surf_temp) = split(/\s+/,$_) if (/./);
      }
      close(INFILE);

      $stemp[$i] = sprintf "%10.4e", $surf_temp - 273.15;
    }

    $time0[$i] = sprintf "%10.4e", $time0[$i]/1.e6;

    if (($varname eq "temp")||($varname eq "convtemp")){
      print " i=$i, $varname=$p_use[$i], time=$time0[$i] sec,
cause=$expcause\n";
    } else {
      print " i=$i, $varname=$p_use[$i], time=$time0[$i] sec,
stemp=$stemp[$i] deg C, cause=$expcause\n";
    }
    # Check for warnings
#    if ($ignorewarn){
#      $dum = 1;  # dummy command
#    } else {
#      `rm -f warnings.out`;
#      `grep Warning *.log > warnings.out`;
#      open(FWARN,"<warnings.out");
#      while (<FWARN>) {
#        if (/$rcallWarn/) {
#       print;
#       $exit_sim = 1;
#     }
#      }
#    }
#    close(FWARN);

    # added 6/29/06 to remove unwanted ALE3D files
```

```
    if ($noclean){
      $dum = 1;  # dummy command
    } else {
      `rm $fgen`;
      `rm $frin`;
#      `rm $fpar`;
      `rm $frf`;
      `rm $fplf`;
      `rm -r $fthistdir`;
#      `rm $flog`;
#      `rm $flogw`;
      `rm $fabo`;
      `rm *.xml`;
      if ($linux){`rm *.core`;}
      print "Unwanted ALE3D files removed.\n";
    }
    if ($exit_sim == 1){exit(0);}
    $i++;
}

# Setup data file for explosion times
open(PBXNI_T,">odtx-lx10.res");
open(FOMFILE,">fom");

# calculate FOM from the runs
$calcFOM = 0.;
for ($i = 1; $i <= $imax; $i++){
  $calcFOM = $calcFOM + (log($time0[$i]/$goal[$i]))**2;
  $FOM = log($time0[$i]/$goal[$i]);
  print PBXNI_T "$FOM\n";
}
#print PBXNI_T "$FOM\n";
$calcFOM = $calcFOM / $imax;
print FOMFILE "$calcFOM\n";

close(FOMFILE);
close(PBXNI_T);

exit;

sub getsize {
  my ($goal) = $_[0];
  $p1 = int($goal);
  $p2 = log($p1)/log(10);
  $p3 = int($p2) + 4;
  $p4 = 10.0**$p3;
  return $p4;
}
```

Appendix E: Contents of *getresults.pl*

```
# PERL script that gathers results from each working directory and puts
it in the
# appropriate output file
#########################################################################
###
# - 2/26/07: original based in part from auto_cleanup.pl
# - 3/09/07: output variables usage, direct read from file, added
progress bar, fomfile
# - 3/26/07: added optimized run flag
#
#########################################################################
###

# a 'use' command to allow for parsing of the input line
#-------------------------------------------------------
use Getopt::Long;

# Process command line options
#-----------------------------
my $inputfile = "psuadeData";
my $outputfile = "psuadeOut";
my $dirfile = "odtx-lx10.res";
my $fomfile = "psuadeFom";
my $opt = 0;

GetOptions('h|help'=>\$help,
           'i=s'=>\$inputfile,
           'o=s'=>\$outputfile,
           'd=s'=>\$dirfile,
           'f=s'=>\$fomfile,
           'opt=i'=>\$opt);

if ($help) {
  print "\n";
  print "use -i to specify input file [psuadeData]\n";
  print "use -o to specify output file [psuadeOut]\n";
  print "use -d to specify data file in directories [odtx-lx10.res]\n";
  print "use -f to specify calculated FOM file [psuadeFom]\n";
  print "use -opt to specify number of working directories if results
are from optimized run\n";
  print "\n";
  exit(0);
}

# optimized run only
if ($opt > 0){
  $fomfile = "psuadeOptFom";
  open (FOMFILE,">$fomfile");
  for ($i = 1; $i <= $opt; $i++){
    $curdir = "workdir.$i";
    open (INPUTFILE,"<$curdir/fom");
    while (<INPUTFILE>){
      print FOMFILE "$i $_";
    }
    close(INPUTFILE);
```

```
  }
  close(FOMFILE);
  print "file $fomfile created.\n";
  exit 0;
}

# open input file and get directory info
open (INPUTFILE,"<$inputfile") or die ("Could not open input file:
$inputfile\n");
$lineCounter = 0;
while (<INPUTFILE>){
  $lineCounter++;
  if ($lineCounter == 2){
    ($value1,$value2,$value3,$value4) = split(/\s+/,$_);
    if (length($value1) == 0){
       $value1 = $value2;
       $value2 = $value3;
       $value3 = $value4;
    }
    if (length($value1) > 0){
      $ninputs = $value1;
      $noutputs = $value2;
      $imax = $value3;
      print "Number of input variables = $ninputs\n";
      print "Number of output variables = $noutputs\n";
      print "Number of working directories = $imax\n";
    } else {
      die ("Problem reading in directory info from file $inputfile\n");
    }
  }
}
close (INPUTFILE);

# go through each working directory and acquire output info
 for ($i = 1; $i <= $imax; $i++){
   $j = 0;
   $curdir = "workdir.$i";
   chdir "./$curdir";
   open(DIRFILE,"<$dirfile") or die ("Could not open input file
$dirfile in directory $curdir\n");
   while (<DIRFILE>){
     ($value1,$value2) = split(/\s+/,$_);
     if (length($value1) == 0){
       $value1 = $value2;
     }
     if (length($value1) > 0){
       $j++;
       $fomvals[$i][$j] = sprintf "%12.4e", $value1;
     }
   }
   chdir "..";
   close (DIRFILE);
   progressbar("Reading data from directories",$i/$imax*100);
 }
 $jmax = $j;

# calculate and output a single FOM based on least squares
 for ($i = 1; $i <= $imax; $i++){
```

```
   $calcfom[$i] = 0;
   for ($j = 1; $j <= $jmax; $j++){
     $calcfom[$i] = $calcfom[$i] + ($fomvals[$i][$j]*$fomvals[$i][$j]);
   }
   $calcfom[$i] = $calcfom[$i]/$jmax;
 }

# prep and change the psuadeData file
 `cd ..`;
 open(INPUTFILE,"<$inputfile");
 open(OUTPUTFILE,">$outputfile");
 open(FOMFILE,">$fomfile");
 $lineCounter = 0;
 $blCounter = 0;
 $blockCounter = 1;
 $outputCounter = 0;
 progressbar("Writing data from directories",$blockCounter/$imax*100);
 while (<INPUTFILE>){
   $lineCounter++;
   if ($lineCounter > 2){
     $blCounter++;
     if ($blCounter == (2+$ninputs+$noutputs)){
       $blCounter = 1;
       $blockCounter++;
       if ($blockCounter <= $imax){progressbar("Writing data from
directories",$blockCounter/$imax*100);}
     }
     if ($blockCounter <= $imax){
       $adjblCounter = $blCounter - $ninputs - 1;
       if ($adjblCounter > 0){
         print (OUTPUTFILE "$fomvals[$blockCounter][$adjblCounter]\n");
       if ($adjblCounter == 1){
         print (FOMFILE "$calcfom[$blockCounter]\n");
       }
       } else {
         print (OUTPUTFILE "$_");
         print (FOMFILE "$_");
       }
     } else {
       print (OUTPUTFILE "$_");
       if ($outputCounter == 0){
         if (/OUTPUT/){
           print (FOMFILE "OUTPUT\n   dimension = 1\n   variable 1
fom\n");
           $outputCounter = 1;
       } else {
           print (FOMFILE "$_");
         }
       } else {
         $outputCounter++;
         if ($outputCounter > ($noutputs+2)){
           print (FOMFILE "$_");
         }
       }
     }
   } elsif ($lineCounter == 2){
     print (OUTPUTFILE "$_");
     print (FOMFILE "$ninputs 1 $imax\n");
```

```
    } else {
      print (OUTPUTFILE "$_");
      print (FOMFILE "$_");
    }
  }
  close (INPUTFILE);
  close (OUTPUTFILE);
  print "done!\n";
  exit 0;

sub progressbar {
    my ($string) = $_[0];
    my ($percent) = $_[1];
    my $progtext = "\r$string: [";

    for (my $i=1; $i <= 50; $i++) {
        if ($i*2 <= $percent) {
            $progtext = $progtext."#";
        } else {
            $progtext = $progtext." ";
        }
    }
    $progtext = $progtext.sprintf("] %u%%",$percent);
    if ($percent == 100) {
        $progtext = $progtext."\n";
    }
    print $progtext;
}
```

Appendix F: Contents of PSUADE Input File *odtxoptMinpack*

```
PSUADE_IO (Note : inputs not true inputs if pdf ~=U)
5 1 1
1 0
 5.03720682e+04
 2.93816728e+01
 5.08082254e-01
 1.87947652e+00
 2.94693822e+00
 4.08016791e-01
PSUADE_IO
PSUADE
INPUT
   dimension = 5
   variable   1 erx =   4.41E+04        5.39E+04
   variable   2 zrx =   2.72E+01        3.33E+01
   variable   3 m   =   9.90E-01        1.21E+00
   variable   4 n   =   1.35E+00        1.65E+00
   variable   5 p   =   2.12E+00        2.59E+00
END
OUTPUT
   dimension = 7
   variable   1 fom1
   variable   2 fom2
   variable   3 fom3
   variable   4 fom4
   variable   5 fom5
   variable   6 fom6
   variable   7 fom7
END
METHOD
   sampling = MC
   num_samples = 1
   num_replications = 1
   num_refinements = 0
   randomize
END
APPLICATION
   driver = NONE
   opt_driver = driveropt.py
END
ANALYSIS
   optimization method = minpack
   optimization num_local_minima = 1
   optimization fmin = 0.000000e+00
   optimization threshold = 0.000000e+00
   optimization fmin = 1.0e-5
   optimization num_fmin = 1
   optimization print_level = 3
   diagnostics 2
END
END
```

Appendix G: Contents of PSUADE Driver File driveropt.py

```
#!/usr/local/bin/python
# sys.argv[1] - input file
# sys.argv[2] - output file

import os
import sys
import string
import shutil
import glob

##########################################################################
###
##########################################################################
###
# Function to get input data from PSUADE input file
#--------------------------------------------------------------------------
---
def getInputData(inFileName, nInputs):
   if nInputs <= 0:
      inputData = []
      return inputData
   inFile = open(inFileName, "r")
   inputData = range(nInputs)
   lineIn = inFile.readline()
   count = 0
   while 1:
      lineIn = inFile.readline()
      ncols = string.split(lineIn)
      inputData[count] = eval(ncols[0])
      count = count + 1
      if count >= nInputs:
         break
   return inputData

##########################################################################
###
# Function to generate input file for the application
#--------------------------------------------------------------------------
---
def
genApplicationInputFile(inputData,appTmpltFile,appInputFile,nInputs,
                           inputNames):
   infile = open(appTmpltFile, "r")
   outfile = open(appInputFile, "w")

# added to provide appropriate directory
   searchString = "PSUADE_COUNTER"
   while 1:
      lineIn  = infile.readline()
      if lineIn == "":
         break
      lineLen = len(lineIn)
      newLine = lineIn
      if nInputs > 0:
```

```
         for fInd in range(nInputs):
            strLen = len(inputNames[fInd])
            sInd = string.find(newLine, inputNames[fInd])
            if sInd >= 0:
               strdata = str(inputData[fInd])
               next = sInd + strLen
               lineTemp = newLine[0:sInd] + strdata +
newLine[next:lineLen]
               newLine = lineTemp

# added to provide appropriate directory
      sInd2    = string.find(newLine, searchString)
      strLen = len(searchString)
      if sInd2 >= 0:
         next = sInd2 + strLen
         strdata = str(fileTag)
         lineTemp = newLine[0:sInd2] + strdata + newLine[next:lineLen]
         newLine = lineTemp
      outfile.write(newLine)
   infile.close()
   outfile.close()
   return

#########################################################################
###
# Function to generate the batch file
#-------------------------------------------------------------------------
---
def genBatchFile(batchTmpltFileName, batchFileName):
   infile = open(batchTmpltFileName, "r")
   outfile = open(batchFileName, "w")
   searchString = "PSUADE_COUNTER"
   strLen = len(searchString)
   while 1:
      lineIn  = infile.readline()
      if lineIn == "":
         break
      lineLen = len(lineIn)
      newLine = lineIn
      sInd    = string.find(newLine, searchString)
      if sInd >= 0:
         next = sInd + strLen
         strdata = str(fileTag)
         lineTemp = newLine[0:sInd] + strdata + newLine[next:lineLen]
         newLine = lineTemp
      outfile.write(newLine)
   infile.close()
   outfile.close()
   return

#########################################################################
###
# Function to run batch file
#-------------------------------------------------------------------------
---
def runApplication(batchFileName):
#  sysCommand = "/usr/local/bin/psub " + batchFileName
#  os.system(sysCommand)
```

```
#  statCommand = "/usr/local/bin/pstat | /bin/grep " + batchFileName
#  status = os.system(statCommand)
#  while status == 0:
#     os.system("sleep 60")
#     status = os.system(statCommand)
   command = "echo Exact_Tran 10.0 > ssdns.info"
   os.system(command)
   command = "echo Exact_Refl 20.0 >> ssdns.info"
   os.system(command)
   return

#########################################################################
###
# Function to run using srun
#-------------------------------------------------------------------------
---
def runApplicationSrun(fileTag,batchFile):
   sysCommand = "psub "+batchFile
   os.system(sysCommand)
   return

#########################################################################
###
# Alternate function to submit batch job (uses dependencies to limit
hogging the processors)
#-------------------------------------------------------------------------
---
def runApplication(executable):
   dependString = ""
   tagString = os.path.splitext(executable)[1]
   stringLen = len(tagString)
   runNumber = eval(tagString[1:stringLen])
#   lastNo    = runNumber - 10
   lastNo    = runNumber - 2
   prefix    = os.path.splitext(executable)[0];
   lastJob   = prefix + "." + str(lastNo) + " "
   statComm  = "/usr/local/bin/pstat -D | /bin/grep "
   statComm  = statComm + lastJob + " > stat"
   status = os.system(statComm);
   dependString = ""
   if status == 0:
      statFile = open("stat", "r")
      lineIn = statFile.readline()
      sIndex = string.find(lineIn, usrName)
      if sIndex > 0:
         sIndex = string.find(lineIn, lastJob)
         if sIndex > 0:
            dependString = "-d " + lineIn[0:sIndex-1]
      statFile.close()
   sysComm = "/usr/local/bin/psub " + dependString + executable
   os.chmod(executable,448)
   os.system(sysComm)
   return

#########################################################################
###
# Alternate function to submit batch job (uses dependencies to limit
hogging the processors) - APW
```

```
#-----------------------------------------------------------------------
---
def runApplicationAPW(batchFile,fileTag):
   sysCommand = "psub "+ batchFile
   print str(fileTag) + ":" +  sysCommand
   os.system(sysCommand)
   statCommand = "/usr/local/bin/pstat | /bin/grep " + batchFile
   status = os.system(statCommand)
   while status == 0:
      os.system("sleep 10")
      status = os.system(statCommand)
   return


#######################################################################
###
# Function to get output file from the application
#-----------------------------------------------------------------------
---
def getApplicationOutputData():
   searchString = ""
   infile = open("odtx-lx10.res", "r")
   outData = range(7)
   outDataCounter = 0;
   while 1:
      lineIn = infile.readline()
      if lineIn == "":
         break
      sInd = string.find(lineIn, searchString)
      if sInd >= 0:
         ncols = string.split(lineIn)
#         outData[0] = eval(ncols[1])
         outData[outDataCounter] = eval(ncols[0])
         outDataCounter = outDataCounter + 1
   infile.close()
   return outData

#######################################################################
###
# Function to get data from ultra file
#-----------------------------------------------------------------------
---
def getUltraOutputData(ultraFileName, searchString):
   endString = "end"
   infile = open(ultraFileName, "r")
   nCount = 0
   while 1:
      lineIn = infile.readline()
      if lineIn == "":
         break
      sInd = string.find(lineIn, searchString)
      if sInd >= 0:
         while 1:
            lineIn = infile.readline()
            sInd = string.find(lineIn, endString)
            if sInd < 0:
               break;
            nCount = nCount + 1
```

```
   infile.close()
   outData = range(nCount)
   infile = open(ultraFileName, "r")
   while 1:
      lineIn = infile.readline()
      if lineIn == "":
         break
      sInd = string.find(lineIn, searchString)
      if sInd >= 0:
         while 1:
            lineIn = infile.readline()
            sInd = string.find(lineIn, endString)
            if sInd < 0:
               break;
            ncols = string.split(lineIn)
            outData[nCount] = eval(ncols[1])
            nCount = nCount + 1
   infile.close()
   return outData

##########################################################################
###
# Function to generate output file from the application
#--------------------------------------------------------------------------
---
def genOutputFile(outFileName, outData):
   nLeng = len(outData)
   outfile = open(outFileName, "w")
   for ind in range(nLeng):
      outfile.write("%e \n" % outData[ind])
   outfile.close()
   return

##########################################################################
###
## Main program
##########################################################################
###

# ------------------------------------------------------------------------
---
# fetch PSUADE input and output file names
# ------------------------------------------------------------------------
---
inFileName  = sys.argv[1]
outFileName = sys.argv[2]
# ------------------------------------------------------------------------
---
# if output file already exists, do not perform test
# ------------------------------------------------------------------------
---
testFlag = 1
if os.path.isfile(outFileName) != 0:
   testFlag = 0
   exit
# ------------------------------------------------------------------------
---
# extract the sample number -> fileTag
```

```
# ------------------------------------------------------------------------
---
tagString = os.path.splitext(inFileName)[1]
stringLen = len(tagString)
fileTag   = eval(tagString[1:stringLen])

# ------------------------------------------------------------------------
---
# application specific settings
# ------------------------------------------------------------------------
---
#appInputTmpltFile = "b0_odtx_Tmplt"
#appInputFile = "b0_odtx"
batchTmpltFile = "b0_odtx.Tmplt"
batchFile = "b0_odtx"
nInputs = 5
inputNames = ["eeeeeeeeeeeee", "zzzzzzzzzzzzz", "mmmmmmmmmmmmm",
"nnnnnnnnnnnnn", "ppppppppppppp"]
copyList = [batchTmpltFile, "runodtx.pl", inFileName, "data.txt"]

# ------------------------------------------------------------------------
---
# If working directory already exists, do not perform test. Otherwise,
# create the working directory
# ------------------------------------------------------------------------
---
dirname = "./workdir." + str(fileTag)
if os.path.isdir(dirname):
   testFlag = 0
else:
   os.mkdir(dirname)

# ------------------------------------------------------------------------
---
# copy the needed files to the working directory and enter into it
# ------------------------------------------------------------------------
---
if testFlag == 1:
   for file in copyList:
      shutil.copy(file, dirname)
   os.chdir(dirname)

# ------------------------------------------------------------------------
---
# generate input file, run test and gather results
# ------------------------------------------------------------------------
---
if testFlag == 1:
   inputData = getInputData(inFileName, nInputs)
   genApplicationInputFile(inputData,batchTmpltFile,batchFile,nInputs,
                           inputNames)
#   genBatchFile(batchTmpltFile, batchFile)

#   runApplicationSrun(fileTag,batchFile)
#   runApplication(batchFile)
   runApplicationAPW(batchFile,fileTag)

   outData = getApplicationOutputData()
```

```
#   outData=range(1)
#   outData[0]=1.0E+35
   genOutputFile(outFileName, outData)
   shutil.copy(outFileName, "..")
#   for file in glob.glob('*'):
#      os.remove(file)
#   os.chdir("..")
#  os.rmdir(dirname)
```