ISAT-CK

(Version 3.0)

User's Guide and Reference Manual

S. B. Pope

October 29, 2000

Contents

Part I

ISAT-CK USER'S GUIDE

1 Introduction

1.1 Overview

ISAT-CK is a Fortran 90 subroutine library that enables gas-phase chemical kinetics to be implemented efficiently in CFD codes. The efficiency stems from the use of the ISAT (*in situ adaptive tabulation*) algorithm (Pope, 1997).

The fluid composition in reactive flows is affected by several processes, e.g., convection, diffusion, radiation and reaction. When CFD is applied to reactive flows, splitting techniques can be used so that, over small time steps δt, the different process can be treated in separate fractional steps. In the fractional step for reaction, for each computational element (e.g., grid node or particle), the problem to be solved is:

> given the thermochemical composition of the fluid element at time t, determine the composition at time $t + \delta t$ resulting from chemical reaction (at constant pressure and enthalpy).

This problem has to be solved repetitively for every computational element on every time step — typically of order 10^9 times.

The straightforward (but expensive) way to solve this problem is through *direct integration* (DI): the chemical kinetics equations are integrated using an ordinary differential equation (ODE) solver. ISAT-CK performs the same function, but it does so much more efficiently by using the ISAT algorithm (Pope, 1997).

1.2 Components of ISAT-CK

The principal components of ISAT-CK are shown in Fig.1; and the paths of the files involved in the standard installation are shown in Fig.2.

The user's code calls one or more of the subroutines contained in the ISAT-CK library `isat-ck.a`. These subroutines are described in Sections 2 and 3.

ISAT-CK works in conjunction with the Chemkin II library `cklib.a`. Chemkin is used to determine the required thermodynamic properties and reaction rates based on specified elementary reaction mechanisms. (It is also possible for the reaction rates to be defined by a user-defined subroutine `usrate`, see Section 3.7.)

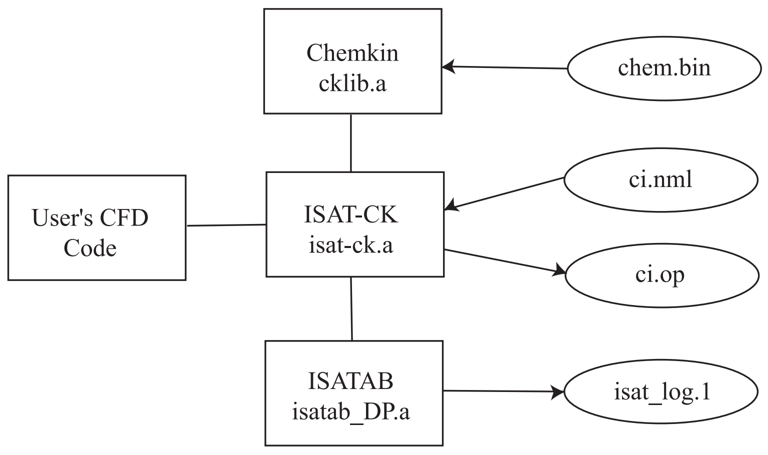

Figure 1: The principal components of ISAT-CK.

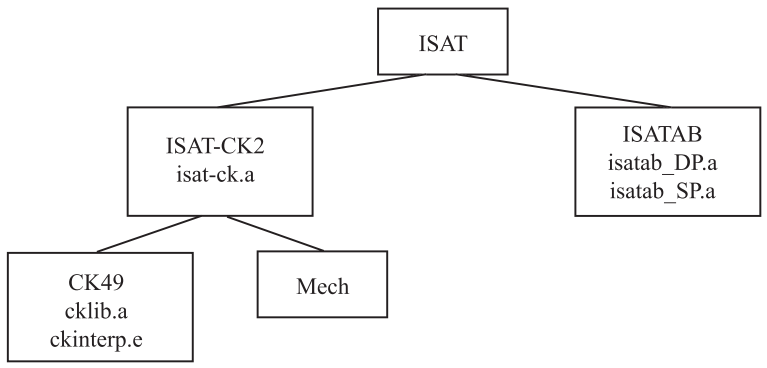

Figure 2: The directory structure of the standard installation of ISAT-CK.

Contained within the ISAT-CK library is an ODE integrator based on DDASAC (Caracotsios and Stewart, 1985), which is used to integrate the chemical kinetics equations.

The ISAT tabulation is performed by the ISATAB library `isatab_DP.a`.

In the standard installation, the principal directory ISAT-CK2 also contains: several demo programs; scripts for pre-processing, post-processing and for running the demos; and a Makefile for the demos which can be adapted to the user's needs.

(Figure 3 on page 18 provides an expanded version of Fig. 1, showing all of the libraries, subroutines and files involved.)

1.3 Reaction-Step and Chemistry-Interface Modes

ISAT-CK can be used in two different modes — *reaction-step mode*, and *chemistry-interface (CI) mode* — which are described in Sections 2 and 3. In reaction-step mode, the composition after a reaction fractional step is determined by a single subroutine call (to `sdtchem` [single precision] or to `dtchem` [double precision]). This may be the simplest way to implement ISAT-CK in an existing CFD code. In CI mode, more functionality, versatility and efficiency is provided by a small set of subroutines. (In fact, `dtchem` and `sdtchem` are implemented through calls to CI routines.)

1.4 Thermochemical Variables and Units

The purpose of this Section is to introduce the notation used, and to define the units of the thermochemical variables. Chemkin uses CGS units. For the most part ISAT-CK follows Chemkin in using CGS units; although, for pressure, both CGS units and standard atmospheres are used.

The variables used and their CGS units are as follows.

c_{pi}	constant-pressure specific heat of species i, ergs/(g K)
h	specific enthalpy of the mixture, ergs/g
h_i	specific enthalpy of species i, ergs/g
h_s	specific sensible enthalpy of the mixture, ergs/g
T	temperature, K
t	time, sec
P	pressure, dynes/cm^2
W_i	molecular weight of species i, g/mole
X_i	mole fraction of species i
Y_i	mass fraction of species i
Z_i	specific mole number of species i, mole/g
ρ	density, g/cm^3

The specific mole number is defined by

$$Z_i \equiv Y_i/W_i, \tag{1}$$

for each of the n_s species. The sensible enthalpy is defined by

$$h_s(\mathbf{Y},T) = \sum_{i=1}^{n_s} Y_i[h_i(t) - h_i(T_{ref}) + c_{pi}(T_{ref})T_{ref}], \tag{2}$$

where $T_{ref} = 298.15$ K is the standard-state temperature.

1.5 Fortran Considerations

Both ISAT-CK and ISATAB are written in Fortran 90, although the calling sequences for user-callable routines generally conform to the Fortran 77 standard. The user's program can be either Fortran 77 or Fortran 90, although it should be compiled and linked using a Fortran 90 compiler. Some Fortran 95 library functions (e.g. `cpu_time`) are also used.

Internally, ISAT-CK uses double precision for real variables, and hence the real arguments to the primary subroutines (e.g., `dtchem` and `cirxn`) are double precision. However, subroutines with single precision real arguments (e.g., `scirxn`) are provided for the convenience of users whose CFD codes use single precision variables.

There is only one interface to ISATAB which uses double precision for real variables. (The ISAT-CK user does not access ISATAB directly, only indirectly through ISAT-CK.) Two ISATAB libraries are provided `isatab_DP.a` and `isatab_SP.a`, which use double precision and single precision, respectively, for *internal* storage. The former is obviously more accurate, but the latter requires about half the storage for a table with a given number of entries. It is recommended to use `isatab_DP.a` initially. Once the user's code is working satisfactorily in conjunction with ISAT-CK, then `isatab_SP.a` can be tested to see if it yields essentially the same results.

1.6 License Key

Prior to using ISAT-CK, the Unix environment variable `ISATKEY` must be set to the full path of the license-key file, which, in the standard installation is `.../ISAT/isat.key`. This is best done in the user's login script. If the full pathname is `user/ISAT/isat.key` then using `csh` the appropriate command is:

```
setenv ISATKEY user/ISAT/isat.key
```

Using `bash` it is:

```
export ISATKEY=user/ISAT/isat.key
```

1.7 Comparison to Earlier Versions

ISAT-CK Version 3.0 is a completely new code, written in Fortran 90, in which tabulation (ISATAB) is separate from the chemistry (ISAT-CK). Compared to earlier versions, the improvements and enhancements are:

1. the restriction of constant time step δt is removed
2. the restriction of constant pressure is removed
3. radiative heat loss is added (as an option)
4. the tabulation employs multiple binary trees (which leads to substantial performance gains)

Because of the scope of these changes, it has not been possible to maintain complete backwards compatibility. However, the changes to the calling sequences have been minimized. For the user upgrading from earlier versions, the principal changes to be aware of are:

1. the pressure `p` needs to be passed to subroutines `cirxn/scirxn` and `cicomp/scicomp`, in the array elements dpt(2) and comp(1), respectively
2. the definition of the tabulation error has been changed (because the normalization of variables has been changed), so that different error tolerances may be needed to achieve comparable results
3. the "compact representation" is no longer used
4. the units of pressure have been changed in subroutines `dtchem/sdtchem`
5. subroutines `ciconv/sciconv` and `cisave` have been added
6. several input files have been renamed and redefined (e.g., `chem.bin` in place of `cklink` and `ci.nml` in place of `opt.in`).

1.8 Outline of this Manual

This manual is in four parts: the ISAT-CK User's Guide, the ISAT-CK Reference Manual, the ISATAB User's Guide, and the ISATAB Reference Manual. The usual ISAT-CK user should not need to study the ISATAB parts extensively, although that is where details on error tolerances and control of the tabulation can be found. On the other hand, ISATAB is a general tabulation library, which can be used in other applications, separate from ISAT-CK.

In the next Section, the basic usage of ISAT-CK in reaction-step mode is described. This serves as an introduction to some of the concepts used in ISAT-CK. Many of the details are deferred to Section 3 where the preferred chemistry-interface (CI) mode is more fully described.

2 Reaction-Step Mode

2.1 Subroutine `dtchem/sdtchem`

`dtchem` is the double-precision version of the reaction-step subroutine and `sdtchem` is the single-precision version. Both subroutines are called with the same arguments except that the floating point arguments should be declared as double precision in `dtchem` and single precision in `sdtchem`.

A description of `dtchem` is shown below:

```
      subroutine dtchem( t, kspec, nspec, spec0, press, kht, ht,
     1                  modecp, modeit, spect, temp, dens )

!  ISAT-CK routine to determine thermochemical composition after
!  isobaric, reaction for a specified time interval.

!  input:
!      t       - time interval (seconds)
!      kspec   - representation of species
!              = 1 - mole fractions
!              = 2 - mass fractions
!              = 3 - specific mole numbers
!      nspec   - number of species
!      spec0   - initial species vector (length nspec)
!      press   - pressure (Chemkin units)
!      kht     - type of second thermodynamic variable
!              = 1 - enthalpy (Chemkin units)
!              = 2 - temperature (K)
!      ht      - initial value of second thermodynamic variable
!      modecp  - not used (retained for backward compatibility)
!      modeit  = 6 - direct integration
!              = 7 - ISAT
!  output:
!      ht      - final value of second thermodynamic variable
!      spect   - final species vector (length nspec)
!      temp    - final temperature
!      dens    - final density (Chemkin units)
!  notes:
!   1/ All reals are double precision.
!   2/ Units are those used in Chemkin.
!   3/ If this routine is called, then ciinit should not be called.
!   4/ The file  streams.in  is used if it exists, but the value of
!      modeci  specified in  streams.in is ignored: the value is taken
!      from the argument modeit.
!   5/ The enthalpy changes solely due to radiative heat loss (if at all).
```

As shown above, the initial values (before the fractional reaction step) of the chemical composition are passed in `spec0` (with the order specified in the Chemkin mechanism file); and the thermodynamic variables are passed in `press` (pressure) and `ht` (enthalpy or temperature). `dtchem` then invokes the CI subroutines to compute the final composition after reaction for time `t` and returns the values in `spect`. Also returned are the final temperature and fluid density.

Setting `modeit`= 6 will activate the direct integration (DI) subroutines at every time step and may increase the computational time substantially. For standard ISAT operations, `modeit`= 7 should be used.

In the following example, `spec0(k,i)` is the initial mass fraction of the kth species at the ith grid node, and `temp0(i)` is the initial temperature. The composition (`spect, tempt, dens`) after reaction for a time interval dt is obtained by:

```
      double precision dt, spec0(nspec, nodes), press, temp0(nodes),
     1                 spect(nspec, nodes), tempt(nodes), dens(nodes)

........

      do i = 1, nodes
         call dtchem( dt, 2, nspec, spec0(:,i), press, 2, temp0(i),
     1             0, 7, spect(:,i), tempt(i), dens(i) )
      end do

........
```

The demo program `cktest` illustrates the use of `dtchem`, see Section 5.1.

2.2 Files

Figure 1 on page 2 gives an overview of the libraries and the basic files used in a simple application of ISAT-CK in reaction-step mode.

2.2.1 The Chemkin Input File `chem.bin`

The input file `chem.bin` is generated by the Chemkin interpretor program (`ckinterp.e`). In order to generate the `chem.bin` file, the user needs to prepare his or her chemical kinetics and thermodynamic data in the Chemkin format. The elements involved are specified by a list of their chemical symbols, e.g., H, O, N, C, etc. Similarly the species are specified, e.g., CO2, H2O, CH4, etc. The reactions are specified, e.g., CO + OH = CO2 + H, and quantitative information about the reaction rates are given. The precise format of this file is described in the Chemkin manual.

For convenience, ISAT-CK provides several commonly used mechanism files, including CH_4/air (`CH4.mech`), H_2/air (`H2.mech`) and CO/H_2/air (`COH2.mech`). The ISAT-CK user can either select from these mechanism files, or else supply his or her own file.

Further information on the execution of the Chemkin interpreter `ckinterp.e` is given in Section 4.1.

2.2.2 The ISAT-CK Input File `ci.nml`

The optional namelist file `ci.nml` allows the user to change certain default settings in ISAT-CK. A version of `ci.nml` containing the default settings (and hence having no effect) is:

```
&cinml
const_pr  = .false.
const_dt  = .false.
user_rate = .false.
radiation = .false.
ichout    = 0
ichin     = 0
ntree     = 0
errtol    = 1.d-3
stomby    = 50.
/
```

The variables have the following meanings:

`const_pr` (logical) If the pressure is the same on every call to ISAT-CK, then set `const_pr=.true.`. When it is applicable, this leads to a saving of computer time and memory.

`const_dt` (logical) If the time step is the same on every call to ISAT-CK, then set `const_dt=.true.`. When it is applicable, this leads to a saving of computer time and memory.

`user_rate` (logical) If `user_rate=.true.`, then the user must specify the reaction rates through the user-supplied subroutine `usrate` (see Section 3.7). A sample program `usrate.f` is included in the standard installation (see Section 9).

`radiation` (logical) For the default `radiation=.false.`, the reaction occurs adiabatically (so that the enthalpy remains constant). For `radiation=.true.`, the enthalpy decreases due to radiative heat loss. The treatment of radiation is described in Section 3.8.

`ichout` (integer). If `ichout=1` is specified, then the ISAT table is periodically checkpointed to the file `isat_dat.1`.

`ichin` (integer). If `ichin=1` is specified, then the ISAT table is initialized from the file `isat_dat.1` generated on a previous run. This saves the computer time needed to generate the table. This option should be used only if the conditions of the run are identical to those of the run that generated the file `isat_dat.1`. In particular, the `ci.nml` file (apart from `ichin` and `ichout`) must be identical.

`ntree` (integer). ISATAB stores the table in n_t binary trees, where n_t is a strictly positive integer. For `ntree` positive, n_t is set to `ntree`. For the default `ntree=0`, n_t is set to 4. See the ISATAB User's Guide (Section 14.6) for a discussion on the setting of n_t.

`errtol` (real) — this parameter controls the tabulation error in ISAT. Let $\boldsymbol{\phi}_0$ be the composition before reaction, let $\boldsymbol{\phi}_E$ be the exact value after reaction (obtained by direct integration), and let $\boldsymbol{\phi}_T$ be the approximate value obtained from the table. Then the tabulation error ε is defined as the two-norm of the difference $(\boldsymbol{\phi}_T - \boldsymbol{\phi}_E)$, scaled by appropriate reference values. This error is deemed to be acceptable if the error ε is less than ε_{tol} =`errtol`. The appropriate choice of `errtol` is problem dependent. If too large a value is specified, then a small table is generated that yields unacceptably large tabulation errors. If too small a value is specified, an unnecessarily large table is generated. Hence the specification of `errtol` is crucial to the effective use of ISAT-CK: this is discussed further in Section 5. (See also Sections 14.1 and 14.3.)

`stomby` (real) specifies the maximum amount of storage (in megabytes) allowed for the ISAT table.

2.2.3 The ISAT-CK Output File `ci.op`

This is the output file generated when ISAT-CK is initialized, which should be checked to ensure that there are no errors. It contains information on: the values of the variables set in `ci.nml`; the species and reactions specified in the Chemkin reaction mechanism; information about the streams (see Section 3.2); and the reference values used for scaling (see Section 14.1).

2.2.4 The ISATAB Log File `isat_log.1`

This is the output file generated when ISATAB is initialized, and should also be checked to ensure that there are no errors. It is described in more detail in the ISATAB Reference Manual, Section 15.2.

2.3 Libraries

The user's code calls subroutines in the ISAT-CK library `isat-ck.a`. These routines call Chemkin (`cklib.a`) and ISATAB (`isatab_DP.a` or `isatab_SP.a`). ISATAB calls some LAPACK routines, which in turn call BLAS routines. The necessary LAPACK and BLAS routines are provided in the library `blapack.a`; but it is preferable to use instead installations of these libraries on the user's machine. (This can be done simply by redefining the variable `LAPACK` in the Makefile.)

In summary, and as illustrated in the Makefile, the modules and libraries to be linked (in order) are: the user's code, `usrate.o` (if necessary), `isat-ck.a`, `cklib.a`, `isatab_DP.a` (or `isatab_SP.a`), `blapack.a` (or other installation of LAPACK and BLAS).

The user wishing to use his or her own version of Chemkin should read Section 4.

2.4 Checklist

In summary, the following steps are needed to use ISAT-CK in reaction-step mode.

1. In the user's CFD code, add the appropriate calls to `dtchem` or `sdtchem`.
2. Use the Chemkin interpreter to generate the `chem.bin` file from the appropriate Chemkin mechanism and thermodynamic data files (see also Section 4.1).
3. Edit `ci.nml` as necessary.
4. If `user_rate=.true.`, supply the user-defined reaction rates in the subroutine `usrate.f`.

5. As illustrated in the Makefile, using a Fortran 90 compiler, compile the user's CFD code and `usrate.f` (if necessary), and link with the necessary libraries to produce the executable module.
6. Set the license key as described in Section 1.6.

3 Chemistry-Interface (CI) Mode

3.1 Overview

The preferred way to use ISAT-CK is in CI mode. The user's CFD code interacts with CI (primarily) through an input file and six subroutines. These are:

`streams.in` An input file that specifies the composition of a number of "streams". Streams are reference compositions which are described further in the next subsection.

`ciinit` A subroutine that initializes CI and Chemkin.

`cistrm` A subroutine that can be called to obtain information about a stream.

`cicomp` A subroutine that can be used to determine different representations of the composition (e.g., mass fractions or mole fractions).

`cirxn` A subroutine that performs reaction (similar in function to `dtchem`).

`ciconv` A subroutine to convert between different representations of the thermochemical state.

`cisave` A subroutine that checkpoints the ISAT table.

Before describing these subroutines and files it is necessary to introduce the concept of "streams".

3.2 Streams

Many reactive flow problems consist of one or more streams flowing into a vessel, each stream with a different but uniform composition. To perform reactive flow calculations, these stream composition must, of course, be known. Based on this simple idea, ISAT-CK (in CI mode) requires the user to input the compositions of one or more streams. In the CFD code, when the composition at a node or for a particle is initialized (as an initial condition or a boundary condition) then it is set to the composition of one of the streams (or to a linear combination of them). Hence the streams must be specified such that the appropriate composition initializations can be performed — this is usually a simple matter.

3.3 The File `streams.in`

The file `streams.in` specifies information about the streams: a sample file is shown below.

```
7
3 5
2 1. 1113. 0. 0. 79. 21. 10.5
1 1.  300. 0. 0. 79. 21.  0.0
1 1.  300. 0. 0.  0.  0. 10.5
```

The first line is an integer, which is the value of the variable `modeci` which determines the mode of thermochemistry to be used. The values `modeci=6` and `modeci=7` correspond to DI and ISAT, respectively: details for other values of `modeci` can be found in the ISAT-CK Reference Manual (Section 8) under the subroutine `ciinit`.

The second line of this file consists of two integers: the number of streams n_{strm}, and the index n_l of the last non-zero stream species. This is now explained. The species are numbered $1, 2, 3, ..., n_s$ according to the order in which they appear in the Chemkin mechanism file. In practice, the number of species n_s may be of order 50, and yet the number of non-zero species in the streams may be less than 5. The index n_l is such that the species with indices $n_l + 1, n_l + 2, ..., n_s$ do not appear in the streams: n_l can always be set to n_s, but it is a convenience to order the species so that n_l is smaller.

The subsequent lines of the files – one for each stream – consist of:

1. an integer k_{strm} described below
2. the pressure P (in atmospheres)
3. the temperature T (K)
4. the chemical composition expressed as n_l numbers, $X'_1, X'_2, ..., X'_{n_l}$: X'_i gives the relative amount of species i in the stream in volume units (i.e., X'_i is a constant times the mole fraction X_i of species i).

Valid values of k_{strm} are 1 and 2. For $k_{strm} = 1$, the stream composition is simply that specified by P, T and $\mathbf{X}'$. For $k_{strm} = 2$, the stream is the thermochemical *equilibrium composition* of the mixture with the same pressure, enthalpy and elemental composition as that specified by (2)-(4).

The streams are numbered sequentially $(1, 2, \ldots, n_{strm})$, in the order in which they appear in the file `streams.in`.

The sample file `streams.in` shown above describes a three-stream methane-air combustion process. In this example $n_l = 5$, and the first 5 species are H_2O, CO_2, N_2, O_2 and CH_4. Thus the third stream (the last line) is pure methane at 300K at 1 atmosphere. The second stream is an approximation to air, 79% N_2, 21% O_2 at 300K and 1 atmosphere. The first stream is stoichiometric methane/air in equilibrium with the same enthalpy as an unburnt mixture at 1113K.

3.4 Representation of the Thermochemistry

Internally, ISAT-CK represents the thermochemical composition by the pressure, P, and by the $n_c = n_s + 1$ *composition variables*, $\boldsymbol{\phi}$. These are the species specific mole numbers ($\phi_i = Z_i, i = 1, 2, \ldots, n_s$) and the sensible enthalpy ($\phi_{n_c} = h_s$).

Additionally, the *full composition* is defined as the $n_f = n_s + 4$ variables corresponding to the species (in some units), the density, the temperature, the pressure, and the enthalpy, e.g., $\{\mathbf{Y}, \rho, T, P, h\}$.

3.5 CI Subroutines

The basic operation of the CI subroutines is now described. A complete specification of the calling sequences is given in Section 8. For routines named `ci...`, real arguments are double precision; whereas in the corresponding routines named `sci...` the calling sequence is the same, but with real arguments being single precision.

3.5.1 Subroutine `ciinit`

This routine initializes ISAT-CK and must be called before any other CI routine. It has no input arguments. Output arguments are the number composition variables, n_c, the number of variables in the full representation, n_f, and the number of streams, n_{nstrm}. This subroutine generates the output file `ci.op` which includes the full compositions of the streams, and the reference values used to scale the variables in ISATAB (see Section 14.1).

3.5.2 Subroutine `cistrm/scistrm`

This routine returns information about a specified stream. The input arguments are the index of the stream and the number of composition variables, n_c. The output arguments are the composition variables $\boldsymbol{\phi}$ for the stream, and its density, pressure and temperature (returned in the array `dpt`).

3.5.3 Subroutine `cicomp/scicomp`

Given the pressure P and the composition variables $\boldsymbol{\phi}$, this routine returns the full thermochemical composition in terms of familiar variables. An input argument (`krep`) specifies whether the species concentration are to be expressed as:

1. mole fractions
2. mass fractions
3. specific mole numbers.

The outputs are: the species concentration, the density, the temperature, the pressure and the enthalpy.

3.5.4 Subroutine `cirxn/scirxn`

This routine determines the composition and density after reaction for a time interval δt, i.e., it is similar in function to `dtchem`. The input arguments are the time interval δt, the number of composition variables, n_c, their values before reaction, and the pressure. The output arguments are the composition variables, and the density, pressure and temperature after reaction.

3.5.5 Subroutine `ciconv/sciconv`

The purpose of `ciconv` is convert from one representation of the thermochemistry to another. Inputs and outputs are density, pressure, an energy variable, and species. The density can be in CGS or SI units. The pressure can be in atmospheres, CGS units or SI units. The energy variable can be temperature (K), the sensible enthalpy (CGS) or the enthalpy (CGS or SI). The species can be mole fractions, mass fractions or specific mole numbers.

3.5.6 Subroutine `cisave`

As described in Section 2.2.2, setting `ichout=1` in the file `ci.nml` causes the ISAT table to be checkpointed periodically, so that it can be used on subsequent runs. Calling `cisave` at the end of a run (or at any other time) causes the table to be checkpointed immediately.

3.6 Using the Chemistry Interface

The philosophy behind the Chemistry Interface is to allow the CFD code to be independent of the details of the thermochemistry. In this way, different descriptions of the chemistry can be used – e.g., detailed chemistry using DI, reduced chemistry using ISAT-CK, or a very simple mixture-fraction or progress variable formulation – without any changes to the CFD code.

In an implementation of the CFD code that fully exploits this philosophy, the only thermochemical quantities represented in the CFD code are: pressure, density, temperature, and "composition variables." In some approaches (e.g., PDF methods), it is not necessary to know the definition of the composition variables, as long as they are linear combinations of species mass fractions and enthalpy.

The thermochemistry to be used is specified by the Chemkin file `chem.bin`, by the subroutine `usrate` (if it is used), and by the CI input file `streams.in`. In order to initialize the thermochemical variables, the CFD code makes appropriate calls to `cistrm`. Ideally, during the main part of the computation, the only CI routine that is called is `cirxn`. In addition to advancing the composition due to reaction, this routine also returns the density and temperature. The subroutine `cicomp` is designed to return quantities for output, not to be used as part of the computation. Again, note that in this ideal application, the definition of the "composition variables" does not need to be known in the CFD code.

The CI routines can be used (differently) even if the CI philosophy is not used in the CFD code. In particular, the subroutine `ciconv` can be used to convert between the different representation of the thermochemistry used in the CFD code and that used on ISAT-CK. For example, if the CFD code represents the thermochemistry by ρ(SI), P(SI), T(K) and mass fractions $\mathbf{Y}$, then the composition can be advanced by the calls:

```
      double precision Y(ns), Z(ns), phi0(ns+1), phit(ns+1), dpt(3)

.........................

!   convert from { rho(SI), P(SI),  T(K),    Y }
!             to { rho(CGS) P(CGS), hs(CGS), Z }

      call ciconv( 2, 3, 1, 2, rho,    P,    T,  Y,
     1             1, 2, 2, 3, rhocgs, pcgs, hs, Z )

!   advance composition

      nc          = ns + 1
      phi0(1:ns) = Z(1:ns)
      phi0(nc)   = hs
      dpt(2)     = pcgs
      call cirxn( dt, nc, phi0, phit, dpt )

!   convert back to obtain  { rho(SI), P(SI), T(K), Y }  after reaction

      call ciconv( 1, 2, 1, 3, dpt(1), dpt(2), dpt(3), phit(1:ns),
     1             2, 3, 1, 2, rho,    P,      T,      Y )
```

There is, of course, some computational overhead in performing these conversions. Hence, if possible, it is best to use the same representation in the CFD code as is used in ISAT-CK. (Note that the conversion from enthalpy to temperature involves iteration: this is avoided in the above code fragment.)

An explicit routine to determine density is not provided: the density returned by `cirxn` (i.e., `dpt(1)`) should be used if possible. Otherwise, the density can be obtained from `cicomp`.

3.7 User-Supplied Reaction Rates

In normal usage of ISAT-CK (in which all of the thermochemistry is provided by Chemkin), the reaction rates are obtained by ISAT-CK through a call to the Chemkin subroutine `ckwyp`. If the user wishes instead to specify the reaction rates, then it is necessary to:

1. set `user_rate=.true.` in the file `ci.nml`
2. provide the subroutine `usrate`

A trivial version of `usrate` is given in the ISAT-CK Reference Manual (Section 9), which shows the required calling sequence. Setting `user_rate=.true.` causes ISAT-CK to call `usrate` in place of `ckwyp`, but with the same arguments. Hence, with the trivial version of `usrate` – which simply calls `ckwyp` – the result is the same whether `user_rate` is `.true.` or `.false.`.

The trivial version of `usrate` is included in the ISAT-CK library `isat-ck.a`. On most systems, if the user's version `usrate.o` is linked before `isat-ck.a`, then the user's version will be used. On some systems it may be necessary to remove `usrate.o` from `isat-ck.a`.

In the Makefile provided, the user's routine `usrate_example.f` will be used if the definition of `USRATE` (in the Makefile) is changed to `USRATE = usrate_example.o`.

Several versions of `usrate`, corresponding to different reduced mechanisms, are provided in the directory `Mech`.

3.8 Radiation

By default, radiation is not included, in which case reaction takes place adiabatically (i.e., with no heat loss, and with constant enthalpy).

Heat loss by radiant emission from the gas mixture is accounted for by the setting `radiation=.true.` in the file `ci.nml`. The radiant emission is calculated based on the absorption coefficients specified in the file `rad.in`. The file provided contains these coefficients (obtained from Radcal, Grosshandler, 1993) for the species CO, CO_2, H_2O and CH_4. (More details are given in Section 10.4 and by Tang and Pope, 1999.)

The radiative heat loss rate is determined from the expression

$$\dot{q}''' = 4\sigma \sum_i p_i a_i(T)\gamma_i(T^4 - T_b^4), \tag{3}$$

where: $\dot{q}'''$ is the net energy emitted per unit volume per unit time (W/m^3, in SI units); $\sigma = 5.669 \times 10^{-8}$ (W/m^2/K^4) is the Stefan-Boltzman constant; p_i is the partial pressure of species i (kPa); $a_i(T)$ is the absorption coefficient of species i (1/(m kPa)); $\gamma_i = 1$ is the *augmentation factor*; T is the temperature (K); and T_b is the background temperature, e.g., 300 K; and summation is over all of the radiating species. (The augmentation factors γ_i are included to facilitate the investigation of radiation from individual species, e. g., radiation from species i is neglected if γ_i is set to zero.)

This treatment of radiation is complete only if: only those species mentioned generate significant radiation; the system is free of soot and other non-gaseous matter; and absorption is negligible (i.e., the optically-thin limit). If other effects are significant – e.g., absorption – then, these effects need to be treated in a different fractional step.

The file `rad.out` summarizes the initialization of the radiation routine in ISAT-CK.

3.9 CI Files

Figure 3 shows all of the input and output files involved in ISAT-CK. Many of these files are descibed in other sections:

- the Chemkin files `chem.inp`, `therm.dat`, `chem.out` and `chem.bin` in Sections 4
- `streams.in` in Sections 3.3 and 10.3
- `ci.nml` in Sections 2.2.2 and 6.1

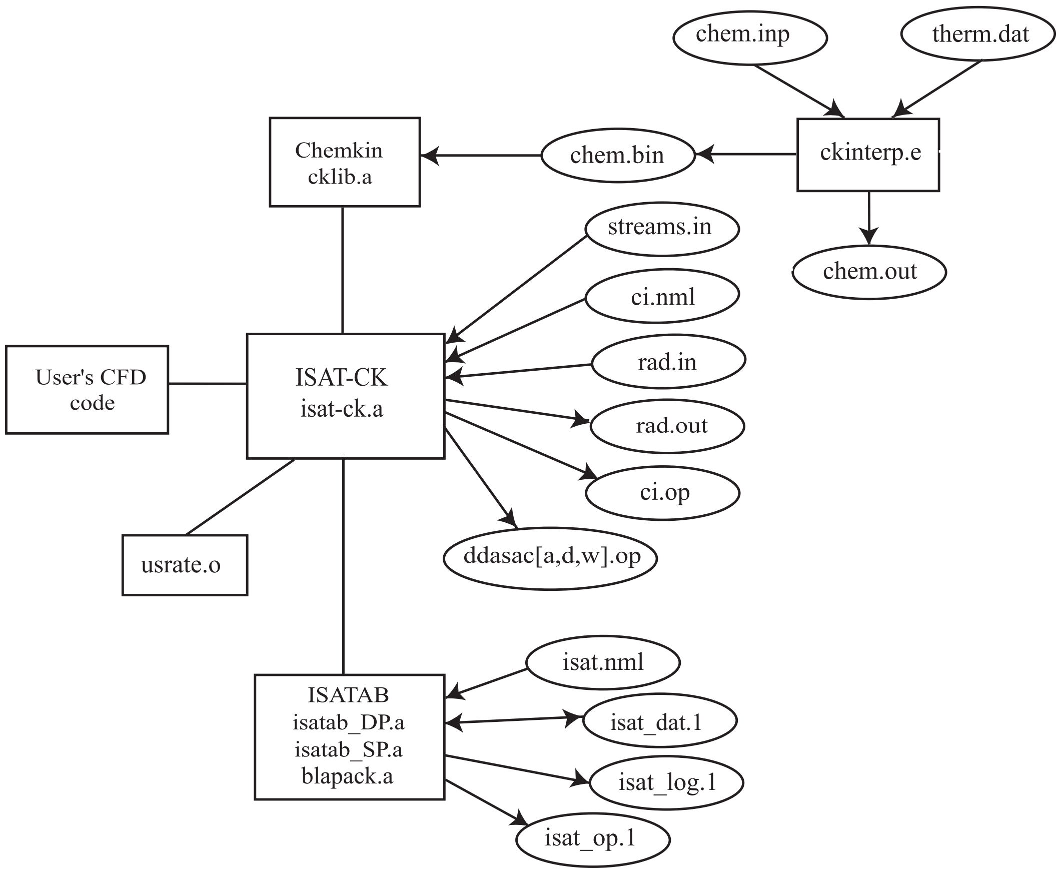

Figure 3: The components of ISAT-CK showing all of the input and output files.

- `rad.in` and `rad.out` in Section 3.8
- `ci.op` in Section 2.2.3
- `isat_dat.1` in Section 2.2.2.

The remaining files are now described.

The file `isat.nml` is a namelist file that can be used to change some of the default setting in ISATAB (see Section 14). In normal usage of ISAT-CK, this file is not needed.

The file `isat_op.1` is an output file containing information on the performance of ISATAB, and which is described in Section 15.3 and 16.5. This can be post-processed using the Matlab script `isat.m`.

The three files `ddasac[a,d,w].op` are output files associated with the DDASAC ODE integrator. The ODE's arising in chemical kinetics can be challenging and ill-conditioned; and, unfortunately, DDASAC is not always capable of integrating them reliably and accurately.

The file `ddasaca.op` reports on the accuracy of the ODE integration. Every time an ISAT table entry is added, the ODE integration is performed twice, with different DDASAC error tolerances, so that the integration error can be measured directly. Two errors are computed: the maximum absolute error is the species specific mole numbers; and the absolute error in temperature. If either of these errors is a record (i.e., greater than any previous error), then the errors are written to `ddasaca.op`. Thus, each line of `ddasaca.op` corresonds to a record error. The five entries on each line are:

1. the number of ODE integrations performed so far
2. the maximum absolute species error (on this integration)
3. the absolute temperate error
4. the initial temperature
5. the time step δt

The files `ddasacw.op` and `ddasacd.op` give warnings and diagnostics, respectively, in case the ODE integration is not accurate or successful.

3.10 Checklist

In summary, the following steps (which are illustrated in `demo_pasr` and in the Makefile) are needed to use ISAT-CK in CI mode:

1. Edit the `streams.in` and `ci.nml` files as necessary.
2. Use the Chemkin interpreter to generate the `chem.bin` file from the appropriate Chemkin mechanism and thermodynamic data files (see also Section 4.1).
3. If `user_rate=.true.`, supply the user-defined reaction rates in the subroutine `usrate.f`.
4. In the user's CFD code:
   (a) initialize ISAT-CK by calling `ciinit`
   (b) obtain the stream information and initialize the composition variables using `cistrm`.
   (c) for the reaction sub-step, obtain the composition and density after the reaction using `cirxn` (and `ciconv` if necessary)
   (d) use `cicomp` (or `ciconv`) to obtain the thermochemical composition in terms of familiar variables for output

5. As illustrated in the Makefile, using a Fortran 90 compiler, compile the user's CFD code and `usrate.f` (if necessary), and link with the necessary libraries to produce the executable module.

6. Set the license key as described in Section 1.6.

4 Chemkin Considerations

4.1 Chemkin Interpreter

Prior to running a program that involves Chemkin, it is necessary to run the Chemkin interpreter. In the standard installation, the interpreter is the executable `ISAT-CK2/CK49/ckinterp.e`. The two required input files are: the Chemkin mechanism file `chem.inp`; and the thermodynamic data file `therm.dat`. In the directory `ISAT-CK2`, execution of `CK49/ckinterp.e` produces the two files: `chem.bin`, which is the required input file for Chemkin; and the diagnostic output file `chem.out`. After execution of `CK49/ckinterp.e`, the file `chem.out` should be examined to verify correct and successful execution.

Note that different versions of Chemkin and its interpreter use different names for the input and output files. ISAT-CK requires that the Chemkin input file be named `chem.bin`.

The script file `ck_setup` is provided to run `ckinterp.e`. For example, given the mechanism file `CH4.mech` and the thermodynamic data file `therm.dat`, then executing:

```
ck_setup CH4.mech
```

executes `ckinterp.e` to generate the corresponding Chemkin input file `chem.bin`. The diagnostic output file (`chem.out`) is renamed `CH4.mech.op`.

4.2 Different Versions

The user may wish to use his or her own version of Chemkin. A problem that may arise in this event is caused by the fact that the calling sequences to the Chemkin routines `ckinit` and `cklen` are different in different versions. Specifically, later versions include `iflag` as the final argument, and ISAT-CK assumes this to be the case.

If the version of Chemkin being used does not contain `iflag`, then the subroutine `ci_ck_noflag.f` should be compiled and linked (before `isat_ck.a`). (On some systems it may also be necessary to remove the file `ci_ck_flag.o` from the library.)

4.3 Subroutine `ck_corr`

In Chemkin, thermodynamic quantities (e.g., enthalpy) are represented as polynomials in two temperature ranges. The temperature ranges and the polynomial coefficients are specified in the file `therm.dat`. At the intersection of the two temperature ranges (which is usually 1,000K), all the properties should be the same whether they are evaluated based on the low-temperature region or on the high-temperature region. For some thermodynamic data bases (i.e., versions of `therm.dat`) this condition is not exactly satisfied; and this can lead to difficulties in the iterative procedure used to determine temperature from the enthalpy.

The subroutine `ck_corr` remedies this problem by making small adjustments to the polynomial coefficients so as to ensure continuity of properties between the temperature ranges. By default, in ISAT-CK this routine is *not* used, because it is not portable (when different versions of Chemkin are used). If it is found necessary, the routine can be added by compiling and linking it (before `isat_ck.a`), but first making sure that the common blocks in `ck_corr.f` are consistent with those in Chemkin. In order for this subroutine to be called, in the file `ci.nml`, it is necessary to set `ickcorr=1`.

5 The Demo Programs

Several "demo" programs are provided. After the initial installation of ISAT-CK, these should be run to verify the correct operation of ISAT-CK. The source code and Makefile also illustrate the use of ISAT-CK.

5.1 The Demo Program `cktest`

The main purpose of this demo is to test the correct operation of Chemkin coupled to ISAT-CK. Given an initial composition, `cktest` determines the composition after reaction for a specified time. It does this twice: first using `dtchem`, and then using `cirxn` (demonstrating the use of `ciconv` discussed in Section 3.6). The source code `cktest.f` can be edited to change (among other things): the initial composition and temperature, the pressure, and the time increment `t`.

The standard demo is performed by executing the script `demo_cktest`. This generates the following output files which can be examined to verify correct operation:

`CH4.mech.op` - the output from the Chemkin interpreter.

`ci.op` - the log file produced by ISAT-CK giving information about its initialization.

`cktest.op` - the output from `cktest` giving the initial and final compositions. This can be compared to the standard result `cktest.op_ref`: the only differences should be in format and round-off.

Note that `cktest` uses direct integration rather than ISAT. To use ISAT instead, edit `cktest.f` to change the value of `modeci` to `7`. To re-run the demo, either execute `demo_cktest`, or execute:

```
make cktest
cktest
```

Whenever new thermochemistry is used, it is recommended to test it using `cktest`, first with DI (i.e., `modeci=6`), and then with ISAT (i.e., `modeci=7`).

ISAT-CK will use the file `streams.in` if it exists (but note that `demo_cktest` removes this file). Hence, by appropriately specifying the file `streams.in` and then executing `cktest`, the streams (including equilibrium compositions) are output to `ci.op`. This is a convenient way to calculate equilibrium properties.

5.2 The Demo Program pasr

The program `pasr` performs a calculation for a *Partially-Stirred Reactor* (PASR), using the pairwise mixing model. This is an excellent test-bed in which to examine many aspects of ISAT-CK's performance.

The partially-stirred reactor used previously to test implementations of combustion chemistry (Correa & Braaten 1993) has the undesirable property that (in the steady state), the accessed region is a one-dimensional manifold: the composition of each particle is a unique function of its residence time. The PASR using the pairwise mixing model, on the other hand, is designed to yield a much larger accessed region, and hence provides a more stringent test.

At any time t, the PASR consists of an even number N of particles, the ith particle having composition $\boldsymbol{\phi}^{(i)}(t)$. With Δt being the specified time step, at the discrete times $k\Delta t$ (k integer) events occur corresponding *outflow*, *inflow* and *pairing*, which can cause $\boldsymbol{\phi}^{(i)}(t)$ to change discontinuously. Between these discrete times, the composition evolves by a *mixing* fractional step and a *reaction* fractional step.

The particles are arranged in pairs: particles 1 and 2, 3 and 4, ..., $N-1$ and N are partners. The mixing fractional step consists of pairs (p and q, say) evolving by

$$\frac{d\boldsymbol{\phi}^{(p)}}{dt} = -(\boldsymbol{\phi}^{(p)} - \boldsymbol{\phi}^{(q)})/\tau_{\mathrm{mix}}, \tag{4}$$

$$\frac{d\boldsymbol{\phi}^{(q)}}{dt} = -(\boldsymbol{\phi}^{(q)} - \boldsymbol{\phi}^{(p)})/\tau_{\text{mix}}, \tag{5}$$

where τ_{mix} is the specified mixing timescale.

In the reaction fraction step, each particle evolves by the reaction equation

$$\frac{d\boldsymbol{\phi}^{(i)}}{dt} = \mathbf{S}(\boldsymbol{\phi}^{(i)}). \tag{6}$$

With τ_{res} being the specified residence time, outflow and inflow consists of selecting $\frac{1}{2}N\Delta t/\tau_{\text{res}}$ pairs at random and replacing their compositions with inflow compositions, which are drawn from a specified distribution.

With τ_{pair} being the specified pairing timescale, $\frac{1}{2}N\Delta t/\tau_{\text{pair}}$ pairs of particles (other than the inflowing particles) are randomly selected for pairing. Then these particles and the inflowing particles are randomly shuffled so that (most likely) they change partners.

In the program `pasr`, the residence time is `tres`, these are `nstr` inflowing streams, with relative mass flow rates `flstrm`. The initial condition for all particles is set to the composition of the first stream. The particles react; and they mix on a timescale `tmix` with partners which are randomly re-assigned on a timescale `tpair`. The pressure varies sinusoidally in time between `prmin` and `prmax` with a period of `prtime`.

The largest time step `dtmax` is determined by the specified constant `cdtrp`. The smallest time step `dtmin` is smaller than `dtmax` by the factor `dtfac` (i.e., `dtmin = dtmax * dtfac`). The time step used is random, uniformly distributed between `dtmin` and `dtmax`. The largest time step `dtsmax` for mixing and reaction substeps is determined by the specified constant `cdtmix`.

Typically this test program is used in two ways: (1) short runs to test the accuracy of ISAT-CK for given error tolerances, etc., and (2) very long runs to test the storage requirements and efficiency of ISAT-CK. The duration of the run is specified as `anres` residence times.

Default values of all of the parameters mentioned above are set (in data statements) in `pasr.f`. They can be changed through the namelist file `pasr.nml`.

There are three PASR demos which are now described.

5.2.1 `demo_pasr`

The standard case is run by executing the script `demo_pasr`. This uses a skeletal mechanism for methane (`CH4.mech`), and three streams which are defined in `streams.pasr`.

The program `pasr` produces three output files:

`pasrm.op` contains ensemble mean properties of the reactor as functions of time. This file can be post-processed using the Matlab script `pasr_m.m`.

`pasrs.op` contains properties of a single particle which can be displayed using the Matlab script `pasr_s.m`.

`pasrp.op` contains a full dump of particle properties. This file is used in the demo `demo_pasr_err` described below.

In addition to these three output files from `pasr`, ISATAB generates the output file `isat_op.1` which can be post-processed using the Matlab script `isat.m`; and ISAT-CK generates the file `ci.op` which provides a log of the initialization of Chemkin and ISAT-CK.

After it has run `pasr`, the script `demo_pasr` then executes the program `psrtest`. This compares the output file `pasrm.op` to the reference `pasrm.op_ref` to check the correct operation of ISAT-CK. The error — due to round-off — should not be greater than a percent or two.

5.2.2 `demo_pasr_err`

An important use of the PASR test case is to measure the global error ε_g that results when ISAT-CK is used with a specified error tolerance ε_{tol}. The script `demo_pasr_err` demonstrates this use.

Two PASR runs are performed (denoted by A and B) with identical conditions, except for the error tolerances which are $\varepsilon_A = 10^{-4}$ and $\varepsilon_B = 10^{-5}$, respectively. The output file `pasrp.op` from run B is renamed `pasrp.op_ref`. The program `pasr_err` then measures the error between `pasrp.op` (from run A) and `pasrp.op_ref` (from run B) and writes the results to the file `pasrerr.op`. This file can be post-processed using the Matlab script `pasr_err.m`.

Figure 4 shows the absolute and relative error in the species specific mole numbers obtained from `pasr_err.m`. The relative error is less than 5% for all species. Figure 5 shows the same errors, but plotted against the average value of the species' specific mole number.

(In general, it is best to measure the error in ISAT-CK relative to the result obtained by direct integration, which is effected by setting `modeci=6`. However, such a run is quite time-consuming. Instead, a good estimate of the global error in ISAT-CK using the given error tolerance ε_A can be obtained by comparing the results to a second run with $\varepsilon_B = \frac{1}{10}\varepsilon_A$. This is the method used in `demo_pasr_err`.)

It is emphasized that `demo_pasr_err` is just a demo. The global error should be measured under conditions that are as close as possible to the intended application.

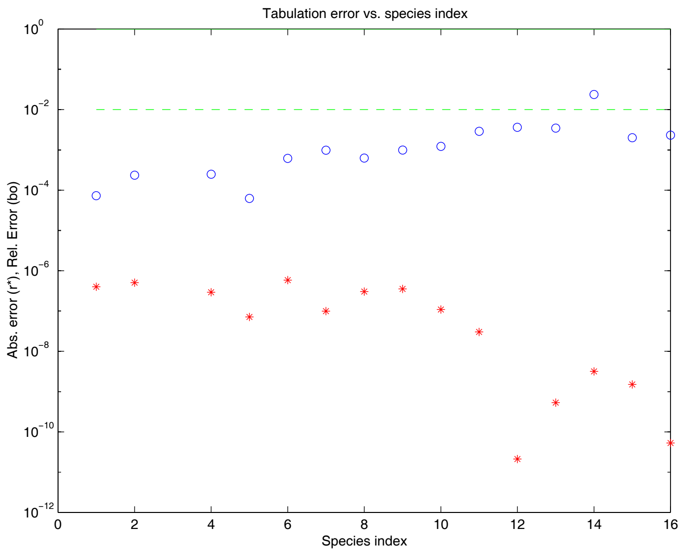

Figure 4: Absolute (*) and relative (o) errors in the species specific mole numbers from `demo_pasr_err`. The numbering of the species is the same as in the Chemkin mechanism file `CH4.mech`. The horizontal lines show 1% and 5% error levels.

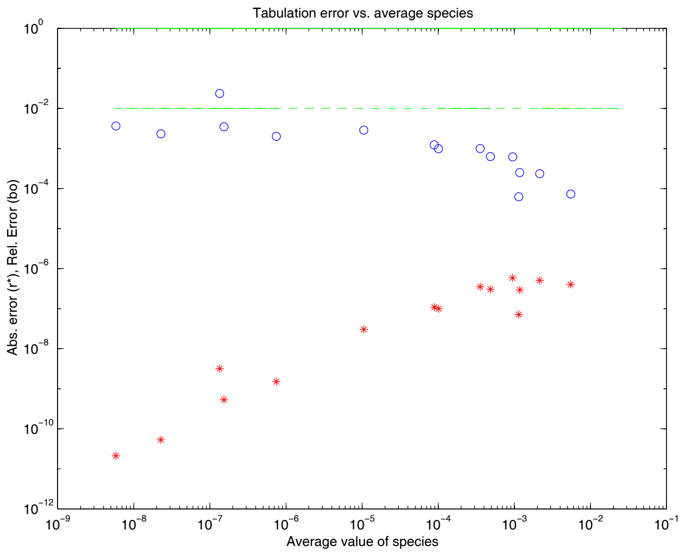

Figure 5: Absolute (*) and relative (o) errors in the species specific mole numbers from `demo_pasr_err` plotted against the average value of the species' specific mole number. The horizontal lines show 1% and 5% error levels.

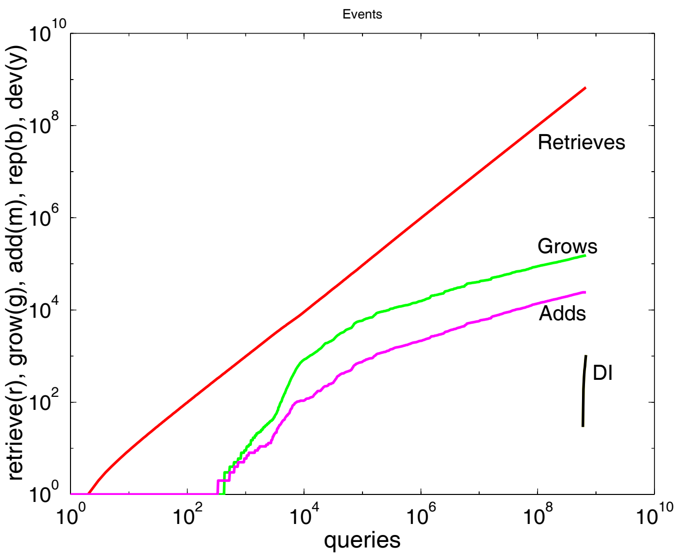

Figure 6: Number of retrieves, grows and adds against the total number of queries for a long PASR calculation similar to `demo_pasr_long`.

The test is also more reliable if it is performed after the bulk of the ISAT table has been generated.

5.2.3 `demo_pasr_long`

The standard case of `pasr` runs the PASR for one residence time, which results in too small a computation for ISAT-CK to show a dramatic speed-up relative to direct integration of the chemical kinetic equations. A longer computation (100 residence times) is provided by the script `demo_pasr_long`. (This may take several hours to run.) The output file `isat_op.1` can be examined using the Matlab script `isat.m`.

Figures 6 and 7 show results for a test similar to `demo_pasr_long`, but with the number of residence times increased to over 50,000. As a function of the number of *queries* (i.e., calls to ISATAB), Fig. 6 shows: the number of *retrieves* from the ISAT table; the number of *adds* to the table; the number of *grows*; and the number of direct integrations (see Section 13 for explanations). After about 6 x 10^8 queries, the table becomes "full" as the number of entries reaches the allowed maximum (of about 20,000) for the given amount of storage. Thereafter if a query cannot be resolved by a retrieve or a grow, then it is resolved by direct integration (DI). For this long run, over 99.97% of the queries are resolved by retrieves.

Figure 7 shows the average CPU time (on a 450 MHz PC) per query, and for retieves, grows and adds. The CPU time for a grow is dominantly the time taken to perform a direct integration (DI), which may be seen to be about 3 x 10^{-2}s. In

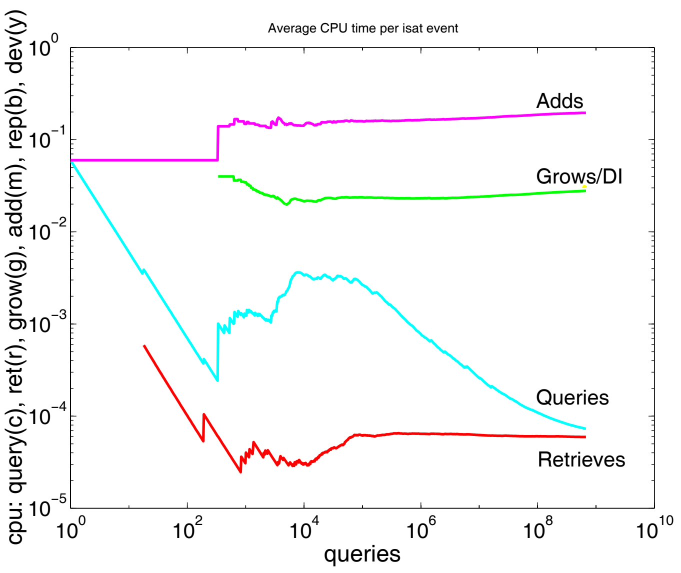

Figure 7: Average CPU time per query, retrieve, grow and add against the total number of queries for a long PASR calculation similar to `demo_pasr_long`.

comparison, an add takes about 7 times as long, whereas a retieve takes 6 x 10^{-5}s, i.e., less that a DI by a factor of 500. After 10^5 queries, the average time per query (i.e., the total time divided by the number of queries) decreases steadily, as most queries result in retrieves. Eventually the average time per query approaches that for a retrieve, leading to a speed-up factor (compared to DI) of 380.

5.2.4 Specification of the Number of Trees

For a given problem, and a given value of ε_{tol}, the primary parameter controlling the efficiency and memory demands of ISAT-CK is the number of binary trees n_t = `ntree`. The influence of n_t is illustrated in Figs. 8 and 9 for a demanding case similar to that in `demo_pasr_long`, but with $\varepsilon_{tol} = 10^{-4}$ and the time step varying between 10^{-5} and 10^{-4} (i.e., `dtfac=0.1` in `pasr`). The results of 8 runs are shown, for $n_t = 1$, 4, 16, 32, 64, 128, 256, 1024, with each run being for 8 hours on a 500 MHz Intel processor.

Figure 8 shows the number of leaves added (i.e., the ISAT table size) at the end of each run. As may be seen, the table size decreases substantially with increasing n_t — by about a factor of 5 over the range shown.

Figure 9 shows the average CPU time per query and per retrieve. The first observation is that the time to retrieve increases with n_t (because more trees are being traversed and more EOA's are being considered). For $n_t = 1$, the time per query is just over half of that for a direct integration, and over 200 times that for a retrieve. This is because a significant fraction of queries result in grows and adds. For $n_t = 1024$, on

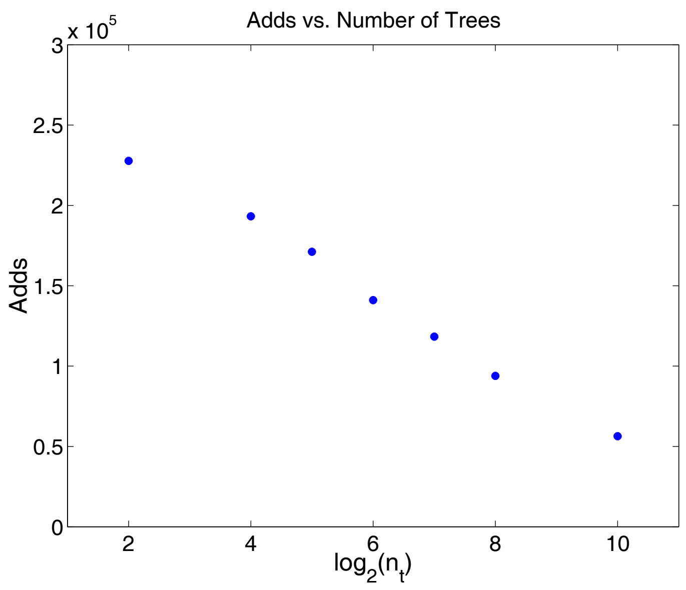

Figure 8: Table size (number of leaves added) as a function of the number of binary trees n_t. PASR test case, $\varepsilon_{tol} = 10^{-4}$, `dtfac=0.1`, after 8 hours CPU time.

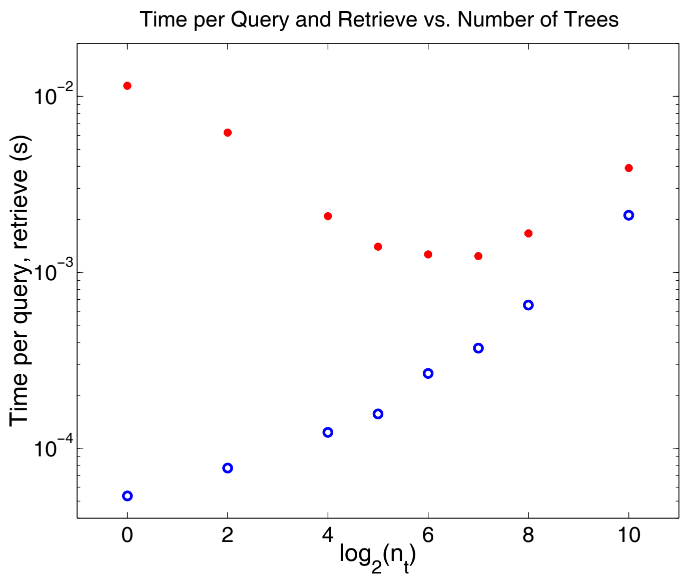

Figure 9: Average time per query (solid symbols), and average time per retrieve (open symbols), as functions of the number of binary trees n_t. PASR test case, $\varepsilon_{tol} = 10^{-4}$, `dtfac=0.1`, after 8 hours CPU time.

the other hand, the query time is just twice the retrieve time, since, by the end of the run, nearly all queries result in retrieves. As n_t increases (and as the duration of the run increases) the query time approaches the retrieve time.

It may be seen from Fig. 9 that the time per query has a minimum for $n_t = 128$, which is therefore the optimal value (for this case). In general, the optimal value of n_t depends on: the storage available; the number of queries required; the error tolerance; and the complexity of the problem (in terms of the thermochemistry and the ranges of pressure and time step). For demanding problems it may be easiest initially to use a large value of n_t (e.g., $n_t = 128$), since (compared to the default $n_t = 4$) less storage is required, and the asymptotic retrieve-dominated state is reached more quickly.

6 Advanced Features

6.1 Changing Default Settings

Various default settings can be changed through the namelist file `ci.nml`. It is strongly recommended that these settings (except for those discussed above) not be changed except by expert users, and when there is a need to do so.

6.1.1 Parameters Passed to ISATAB

The following parameters are passed to ISATAB: see Sections 16.3 and 16.4 for a detailed description.

```
ichin  = 0      ! =1 to read checkpointed table
ichout = 0      ! =1 to write checkpointed table
ntree  = 0      ! number of ISAT trees
noisat = 0      ! =1 to suppress ISAT and use DI
idites = 0      ! =1 to perform DI for checking inside EOA
ifull  = 0      ! =1 for leaf substitution when table full
kcpv   = 0      ! kind of cutting-plane vector

errtol  = 1.d-3 ! ISAT absolute error tolerance
elpmax  = 0.1   ! maximum allowed size of EOA
elp0max = 0.    ! controls size of initial EOA
stomby  = 50.   ! storage allowed for ISAT table
outinc  = 1.02  ! controls amount of ISAT output
```

6.1.2 DDASAC Error Tolerances

The ODE integrator DDASAC has absolute and relative error tolerances for the ODE integration and for the calculation of sensitivity coefficients. These are set by:

```
atolc = 1.d-6   ! absolute error tolerance for ddasac
rtolc = 1.d-9   ! relative error tolerance for ddasac
atols = 1.d-2   ! absolute error tolerance for sensitivities
rtols = 1.d-2   ! relative error tolerance for sensitivities
```

6.1.3 DDASAC Accuracy Testing

The accuracy of DDASAC can be monitored on ISATAB adds (`ichdas=1`, the default), on ISATAB grows and adds (`ichdas=2`), or not at all (`ichdas`=0). A warning message is issued (on the file `ddasacw.op`) if the species error exceeds `pewarn`, or if the temperature error exceeds `tewarn`. Only one warning is issued.

6.1.4 Bounds on Temperature

Errors in the thermochemical inputs and elsewhere can cause spuriously high or low temperatures. ISAT-CK terminates if a temperature is encountered that is less than `tbadlo` (default 250K), or greter than `tbadhi` (default 3000K). For problems in which the temperature exceeds these bounds, then the bounds should be changed. It is emphasized that a spuriously high or low temperature is usually an indication of a gross error in the input. It can also occur if `errtol` is too large, leading to unrealistic thermochemical compositions.

6.2 Simpler CI Thermochemistry

The Chemistry Interface is designed to allow CFD calculations to be performed with different thermochemistry with little or no modification to the flow code. The different chemistry is specified through the files `streams.in` and `chem.bin`. The first line in the file `streams.in` defines the integer variable `modeci`, which has the following effect:

`modeci=1` — inert, constant-density flow

`modeci=2` — mixture fraction formulation

`modeci=3` — reaction progress variable formulation

`modeci=4` — reserved for future use

`modeci=5` — reserved for future use

`modeci=6` — direct integration of Chemkin kinetics

`modeci=7` — Chemkin kinetics using ISAT.

The specification of the remainder of the file `streams.in` depends on the value of `modeci`. For `modeci`= 6 or 7, it is described in Section 3.3. For all values it is given in the ISAT-CK Reference Manual (in Section 10).

7 References

Caracotsios, M. and Stewart, W. E. (1985) "Sensitivity analysis of initial value problems with mixed ODEs and algebraic equations," Computers and Chemical Engineering **9**, 359–365.

Correa, S. M. and Braaten, M. E. (1993) Combust. Flame **94**, 469.

Pope, S. B. (1997) "Computationally Efficient Implementation of Combustion Chemistry using In Situ Adaptive Tabulation," Combustion Theory and Modeling, **1**, 41.

Tang, Q and Pope, S. B. (1999) "Implementation of Radiation in ISAT," Cornell University report, FDA 99-05.

Part II

ISAT-CK REFERENCE MANUAL

8 ISAT-CK Subroutines

This section gives a list of the ISAT-CK subroutines and their arguments, including `dtchem`. The double-precision versions of the subroutines are shown. For the single-precision versions, the calling sequence is the same, but with all reals being single-precision.

```
---------------------------------------------------------------------------
! The calling sequences for the double-precision Chemistry Interface
! subroutines are given in the following order.  The names of the
! single-precision versions are shown to the right.

!   double precision                              single precision

!   ciinit( ncv, nfullv, nstrms )                           ciinit

!   cistrm( istr, ncv, c, dpt )                             scistrm

!   cicomp( ncv, c, krep, nfullv, comp, cname )             scicomp

!   cirxn( t, ncv, c0, ct, dpt )                            scirxn

!   ciconv( ncv, jd,   jp,   je,   js,                      sciconv
!                din,  pin,  ein,  spin,
!                kd,   kp,   ke,   ks,
!                dout, pout, eout, spout )

!   cisave( nrec )                                          cisave

!   dtchem( t, kspec, nspec, spec0, press, kht, ht,
!          modecp, modeci, spect, temp, dens )              sdtchem

---------------------------------------------------------------------------

        subroutine ciinit( ncv, nfullv, nstrms )

!  chemistry interface initialization routine.
!  input:      - none
!  output:
!      ncv     - number of composition variables (integer)
!      nfullv  - number of items in the full representation (integer)
!      nstrms  - number of streams (integer)

!  files:
!      streams.in - input file of stream information.
!              the first record of streams.in contains the value of the
!              integer variable  modeci  which determines the mode of
!              thermochemistry to be used.  The specification of the
!              subsequent records of  streams.in  depends on  modeci.
!        modeci = 1 - inert, constant-density flow
!        modeci = 2 - mixture fraction formulation (f)
!        modeci = 3 - reaction progress variable formulation (c)
!        modeci =[4]- (f,c) formulation
!        modeci =[5]- general user-supplied thermochemistry
!        modeci = 6 - Chemkin/direct integration
!        modeci = 7 - Chemkin/ISAT
!      ci.op - output file

----------------------------------------------------------------------------

       subroutine cistrm( istr, ncv, c, dpt )

!  chemistry interface routine to return composition of specified stream.

!  input:
!      istrm - index of stream ( 1 <= istrm <= nstr ) (integer)
!      ncv   - number of composition variables (integer)
!  output:
!      c     - composition vector for stream istrm (length ncv, double)
!      dpt   - density, pressure and temperature of stream istrm
!        (length 3, double)

----------------------------------------------------------------------------

        subroutine cicomp( ncv, c, krep, nfullv, comp, cname )

! chemistry interface routine to return full composition comp,
! corresponding to composition c.
! input:
!      ncv     - number of composition variables (integer)
!              - If ncv=0, then only cname is returned: other input ignored.
!      c       - composition vector (length ncv, double)
!      krep    - type of representation required
!              - for modeci = 6 or 7, then
!      krep    = 1 - express species as mole fractions
!      krep    = 2 - express species as mass fractions
!      krep    = 3 - express species as specific mole numbers
!      nfullv  - number of full composition variables (integer)
!      comp(1) - pressure in Chemkin units (modeci = 6 or 7, only)
! output:
!      comp    - full composition vector (length nfullv, double)
!      cname   - names of composition variables (character*10)

!  for  modeci = 6 or 7, nfull = ns + 4 ( ns = number of species)
! comp(i)    = species(i), i=1, ns
!       comp(ns+1) = density (Chemkin units)
!       comp(ns+2) = temperature (K)
!       comp(ns+3) = pressure (Chemkin units)
!       comp(ns+4) = enthalpy (Chemkin units)
```

```
subroutine cirxn( t, ncv, c0, ct, dpt )

!  chemistry interface routine to return composition c(t) resulting from
!      reaction for a time t from the initial composition c(0).  Also
!      returned are density, pressure and temperature.

!  input:
!      t     - time (seconds), duration of reaction (double)
!      ncv   - number of composition variables (integer)
!      c0    - initial composition vector (length ncv, double)
!      dpt(2)- pressure in Chemkin units (required for modeci=6,7)
!  output:
!      ct    - final composition vector (length ncv, double)
!      dpt   - density, pressure and temperature (length 3, double)
```

```
      subroutine ciconv( ncv, jd,   jp,   je,   js,
   1                       din,  pin,  ein,  spin,
   2                       kd,   kp,   ke,   ks,
   3                       dout, pout, eout, spout )

!  chemistry interface routine to perform conversion of thermochemical
!     variables for modeci = 6 or 7.

!  input:
! ncv  - number of composition variables (=ns+1) (integer)
! jd   - units of input density (integer):
!      =1, CGS (g/cm^3)
!      =2, SI (kg/m^3)
! jp   - units of input pressure (integer):
!      =1, standard atmospheres
!      =2, CGS (dyne/cm^2)
!      =3, SI (Pa)
! je   - type of input energy variable (integer):
!      =1, temperature, T (K)
!      =2, sensible enthalpy, h_s (CGS) ergs/g
!      =3, enthalpy, h (CGS) ergs/g
!      =4, enthalpy, h (SI) J/kg
! js   - type of species variable (integer):
!      =1, mole fraction, X
!      =2, mass fraction, Y
!      =3, specific mole number, Z (mole/g)
! din  - input density (double)
! pin  - input pressure (double)
! ein  - input energy variable (double)
! spin - input species variables (length ns=ncv-1, double)
! kd   - units of output density (integer) (=1 or 2, as jd)
! kp   - units of output pressure (integer) (=1,2 or 3 as jp)
! ke   - type of output energy variable (integer) (=1,2,3 or 4, as je)
! ks   - type of species variable (integer) (=1,2, or 3, as js)
! dout - output density (double)
! pout - output pressure (double)
! eout - output energy variable (double)
! spout- output species variables (length ns, double)

---------------------------------------------------------------------------

subroutine cisave( nrec )
```

```
!   Force checkpointing of ISATAB table

! inout:  none

! output:
! nrec - number of records in table (integer)

----------------------------------------------------------------------------

       subroutine dtchem( t, kspec, nspec, spec0, press, kht, ht,
     1                    modecp, modeit, spect, temp, dens )

!  ISAT-CK routine to determine thermochemical composition after
!  isobaric, reaction for a specified time interval.

!  input:
!      t       - time interval (seconds)
!      kspec   - representation of species
!              = 1 - mole fractions
!              = 2 - mass fractions
!              = 3 - specific mole numbers
!      nspec   - number of species
!      spec0   - initial species vector (length nspec)
!      press   - pressure (Chemkin units)
!      kht     - type of second thermodynamic variable
!              = 1 - enthalpy (Chemkin units)
!              = 2 - temperature (K)
!      ht      - initial value of second thermodynamic variable
!      modecp  - not used (retained for backward compatibility)
!      modeit  = 6 - direct integration
!              = 7 - ISAT
!  output:
!      ht      - final value of second thermodynamic variable
!      spect   - final species vector (length nspec)
!      temp    - final temperature
!      dens    - final density (Chemkin units)
!  notes:
!       1/ All reals are double precision.
!       2/ Units are those used in Chemkin.
!       3/ If this routine is called, then ciinit should not be called.
!       4/ The file  streams.in  is used if it exists, but the value of
!          modeci  specified in  streams.in is ignored: the value is taken
!          from the argument modeit.
!       5/ The enthalpy changes solely due to radiative heat loss (if at all).
```

9 Subroutine `usrate`

The trivial version of the subroutine `usrate` is:

```
---------------------------------------------------------------------------

       subroutine usrate( ns, p, t, y, liwk, lrwk,
     1    ickwrk, rckwrk, wdot )

!  User routine to specify reaction rates.  This version calls (and is
!  functionally equivalent to) the Chemkin routine ckwyp.

!  input:
!      ns      - number of species
!      p       - pressure (Chemkin units)
!      t       - temperature (K)
!      y       - species mass fractions
!      liwk    - dimension for Chemkin integer array ickwrk
!      lrwk    - dimension for Chemkin double-precision array rckwrk
!      ickwrk  - Chemkin integer array
!      rckwrk  - Chemkin double-precision array
!  output:
!      wdot    - reaction rates in molar units

integer ns, liwk, lrwk, ickwrk(liwk)
double precision p, t, y(ns), rckwrk(lrwk), wdot(ns)

call ckwyp( p, t, y, ickwrk, rckwrk, wdot )

return
end
```

10 Input Files

10.1 The File `streams.in` for `modeci=1`

```
--------------------------------------------------------------------------

!      Specification of the file  streams.in  for modeci = 1

!   1st record: modeci
!   2nd record: density
```

10.2 The File `streams.in` for `modeci=2`

```
--------------------------------------------------------------------------

!      Specification of the file  streams.in  for modeci = 2 and 3

!   1st record: modeci
!   2nd record: nstr  - number of streams
!   3rd record: nfull - number of variables in full representation
! modeci=3 only:
!   4th record: trxn3 - reaction time scale
!   next nstr records:  f, dens, p, T, c(1), c(2),...,c(nfull)
!   optionally, next nfull records:
!           names of variables in full representation (character*10)
!  notes:
! 1/ f denotes mixture fraction (modeci=2) and reaction progress
!    variable (modeci=3).
! 2/ the values of f of the streams must be strictly increasing.
! 3/ properties are linearly interpolated in f.
! 4/ the specific volume (1/dens) is linearly interpolated, not dens.
! 5/ for modeci=3 the reaction is:
! db/dt = -b/trxn3,  where b = ( fmax - f ) / ( fmax - fmin )
```

10.3 The File `streams.in` for `modeci=6 or 7`

```
--------------------------------------------------------------------------

!      Specification of the file  streams.in  for modeci = 6 and 7
!        1st record - modeci
!        2nd record - nstr, nsin
!                     nstr - number of streams (integer)
!                     nsin - number of non-zero species (integer)
!        subsequent - k, p, T, c(1), c(2),...,c(nsin)
!                     k = 1 if stream is as stated (integer)
!                     k = 2 if stream is an equilibrium mixture with the
!                           same elemental composition, pressure and
!                           enthalpy as that stated.
!                     p    - pressure in atm  (real)
!                     T    - temperature in K (real)
!                     c(i) - composition in relative volume/mole units (real)
```

10.4 The File `rad.in`

```
---------------------------------------------------------------------------

!   Specification of the file rad.in

!  1st line: nrad number of species tabulated
!  2nd line: tback background temperature (K)
!  next  nrad  lines: symrad(j), augrad(j)   (a16, 4x, 1pe13.4)
!                     symrad - Chemkin symbol of radiating species
!                     augrad - augmentation factor (=1.0 for normal usage)
!                     If augrad(j)=0, then the correspond species j
!                     will not be considered in the radiation calculation
!  next ntemp lines: t(i), (apc(i,j),j=1,nrad)
!                     t(i) -- temperature (K)
!                     apc(i,j) -- Plank Mean Absorption Coeff.  1/(m kPa)

!  (Radiation can be supressed by setting nrad=0)
```

Part III

ISATAB USER'S GUIDE

11 Introduction

11.1 Overview

ISATAB is a Fortran library which implements the ISAT algorithm (*In Situ* Adaptive Tabulation[1]) for a general function $\mathbf{f}(\mathbf{x})$. The subroutine `isatab` is called many times by the user's program with different values of $\mathbf{x}$, and the routine returns (a good approximation to) $\mathbf{f}(\mathbf{x})$. The user must provide a subroutine to evaluate $\mathbf{f}(\mathbf{x})$ and its derivatives. The name of this subroutine is arbitrary, but for definiteness we call it `usrfgh`.

11.2 Simple Usage

The independent variable $\mathbf{x}$ has n_x components, i.e., $\mathbf{x} = x_1, x_2, \ldots, x_{n_x}$; and similarly the dependent variable $\mathbf{f}$ has n_f components, i.e., $\mathbf{f} = f_1, f_2, \ldots, f_{n_f}$. The gradient of $\mathbf{f}$ at $\mathbf{x}$ is denoted by $\mathbf{g}(\mathbf{x})$ which has components

$$g_{ij} \equiv \frac{\partial f_i}{\partial x_j}. \tag{7}$$

For given $\mathbf{x}$, the user-provided subroutine `usrfgh` returns the value of $\mathbf{f}(\mathbf{x})$ and, if so required, the value of $\mathbf{g}(\mathbf{x})$. The complete specification of `usrfgh` is given in Section 17.

In the user's program, $\mathbf{f}(\mathbf{x})$ is to be evaluated many times for different values of $\mathbf{x}$. This could be accomplished by calling `usrfgh` — which is referred to as *direct evaluation*. The objective of ISATAB is to produce (almost) the same effect, but at a substantially reduced computational cost (measured in CPU time). Thus a call to `isatab` yields an approximate value of $\mathbf{f}(\mathbf{x})$. The error in this approximation can be controlled through the specification of an error tolerance (see Section 14.3 for details). The complete calling arguments to `isatab` are described in Section 16.

[1]see S.B. Pope, "Computationally efficient implementation of combustion chemistry using *in situ* adaptive tabulation", *Combustion Theory and Modelling*, 1:41, 1997

11.3 Ellipsoids of Accuracy

To understand the general usage of ISATAB it is necessary to understand the *ellipsoids of accuracy* (EOA) used in the ISAT algorithm. The basic tabulation element is a hyper-ellipsoid, whose center is denoted by $\mathbf{x}^0$. The information stored for the element consists of: $\mathbf{x}^0$; $\mathbf{f}^0 \equiv \mathbf{f}(\mathbf{x}^0)$; $\mathbf{g}^0 \equiv \mathbf{g}(\mathbf{x}^0)$; and the principal axis of the EOA. Based on this information, a linear approximation to $\mathbf{f}(\mathbf{x})$ is

$$f_i^\ell \equiv f_i^0 + g_{ij}^0(x_j - x_j^0). \tag{8}$$

The EOA is constructed so that, within the EOA, $\mathbf{f}^\ell$ is an adequate approximation to $\mathbf{f}(\mathbf{x})$. Thus ISATAB uses this piecewise-linear approximation to $\mathbf{f}(\mathbf{x})$.

11.4 General Usage

In addition to returning $\mathbf{f}^\ell$, `isatab` (on request) returns $\mathbf{g}^0$ – which is a piecewise-constant approximation to $\mathbf{g}(\mathbf{x})$ within the EOA.

With $\mathbf{h} \equiv \{h_1, h_2, \ldots, h_{n_h}\}$ being a second user-specified function, `isatab` (on request) also returns $\mathbf{h}^0 \equiv \mathbf{h}(\mathbf{x}^0)$. Again, this is a piecewise-constant approximation to $\mathbf{h}(\mathbf{x})$ within the EOA.

Note that $\mathbf{f}^\ell$ is a piecewise-linear approximation, the error in which is controlled. In contrast $\mathbf{g}^0$ and $\mathbf{h}^0$ are piecewise-constant approximations without (direct) error control.

ISATAB is capable of tabulating any number of different functions by constructing a separate table for each function. In general we can consider $\mathbf{f}^{(k)}$ to be the kth function, consisting of $n_f^{(k)}$ components and depending on $n_x^{(k)}$ independent variables. The first argument of `isatab` is the positive integer `idtab` which is the identifier of the table (i.e., k). For each different table, a different user-defined function `usrfgh` can be used.

11.5 Fortran Considerations

ISATAB is written in Fortran 90, and the call to `isatab` conforms to the Fortran 77 standard.

All real variables are double precision.

Most names (of files and routines) have `isat` as the first four characters. Hence such names should be avoided in the user's program to avoid name conflicts.

ISATAB uses logical units (which are not already in use) in the range `60 <= lu <= 99`. A conflict can arise if the user's program uses a logical unit that is being used by ISATAB. The range used by ISATAB can be changed to, for example, `71 <= lu <= 88` by setting the file `isat.nml` to: `&isat_nml`

```
luf=71
lul=88
/
```

ISATAB includes an optional MPI implementation. However, this is not described here, nor is it supported.

12 Installation and Testing

12.1 The ISATAB Directory

The result of performing the installation is to produce a directory `ISATAB` which contains (at least) the following files:

`isatab_ser.a`	the ISATAB Fortran library
`isat.key`	the license key
`isatab_ug.ps`	the postscript file of this User Guide
`isatab.txt`	a text file containing a summary of ISATAB operation and arguments
`testser.f90`	a test program
`fghex.f90`	the subroutine `usrfgh` used by the test program
`isat_rnu.f90`	a subroutine called by `testser.f90`
`Makefile`	the makefile to generate the executable `testser`
`testser.op`	the correct output from `testser`
`isat.m`	a Matlab script that can be used to examine `isat_op.1`
`dispr.m`	a Matlab script called by `isat.m`

12.2 License Key

The Unix variable `ISATKEY` must be set to the location of the license key, e.g., `ISAT/ISATAB/isat.key`. This is best done in the user's login script. Using `csh` the appropriate command is:

```
setenv ISATKEY ISAT/ISATAB/isat.key
```

Using `bash` it is:

```
export ISATKEY=ISAT/ISATAB/isat.key
```

12.3 Testing

The test program `testser` is provided to test the correct operation of ISATAB. To create the executable, on the directory `ISAT/ISATAB` type `make`. Then, execute `testser > t.op` (which should take about 10 seconds). The output file `t.op` can be compared to the reference output file `testser.op`. The only differences should be in format and due to round-off error differences.

12.4 Linking ISATAB with User's Code

The makefile `ISAT/ISATAB/Makefile` provides a model for linking ISATAB with a user program. In addition to the ISATAB library `isatab_ser.a` it is necessary also to link the lapack library and any libraries (e.g. blas) required by lapack.

13 Overview of the ISAT Algorithm

In order to use ISATAB effectively, it is useful to have an understanding of the ISAT algorithm, so that various parameters can be set appropriately.

The EOA's are stored in a specified number (n_t =`ntree`) of binary trees. Each leaf of a tree is an EOA: each node corresponds to a cutting plane in $\mathbf{x}$-space.

Each call to `isatab` is referred to as a *query*. On each query, ISATAB attempts to find an EOA that contains the query point $\mathbf{x}$. If such an EOA is found, then the linear approximation is returned. This outcome is designated a *retrieve*.

If the query is not fulfilled by a retrieve, then a direct evaluation of $\mathbf{f}(\mathbf{x})$ is performed. Based on this value, some EOA's are examined to determine if the linear approximation based on $\mathbf{f}^0$ and $\mathbf{g}^0$ is sufficiently accurate. If it is, then that EOA is "grown" to include the query point. This outcome is referred to as a *grow*.

If the query is not satisfied by a grow then a new EOA is added; and the outcome is referred to as an *add*.

14 Control of ISATAB

The operation of ISATAB is controlled primarily through the integer array `info` and the real array `rinfo` that are arguments in the call to `isatab`. For example, `info(5)` specifies the storage (in Mbytes) to be allowed for the table. A full specification of the elements of these arrays is given in Section 16. Expert users can also alter some internal ISATAB settings through entries in the optional namelist file `isat.nml` (as illustrated in Section 11.5).

14.1 Scaling

ISATAB works with scaled variables $\tilde{x}_j \equiv x_j/x^s_j$ and $\tilde{f}_i \equiv f_i/f^s_i$ where x^s_j and f^s_i are specified, strictly positive, scale factors. By default (`info(1) = 0`) these scale factors are unity.

For best performance of ISATAB, the scale factors should be chosen so that differences in the scaled variables are of order unity. In particular, the elements of the scaled gradient matrix

$$g^s_{ij} \equiv \frac{\partial f^s_i}{\partial x^s_j} \tag{9}$$

should not be large compared to unity.

The scale factors can be set (different from unity) by setting `info(1)=1`, `rinfo(20+i)` $= x^s_i$ and `rinfo(20+nx+j)`$=f^s_j$.

14.2 Transformation of Variables

Because of the piecewise-linear approximation used, ISATAB works best if $\mathbf{f}$ is approximately linear in $\mathbf{x}$. The user should consider whether a transformation of variables (prior to calling `isatab`) can be used to reduce the level of non-linearity.

14.3 Error Tolerance

The error in the (scaled) linear approximation is defined to be

$$\varepsilon \equiv |\tilde{\mathbf{f}}^\ell - \tilde{\mathbf{f}}^0|, \tag{10}$$

where, on the right-hand side, the scaled values of $\mathbf{f}^\ell$ and $\mathbf{f}^0$ are defined in an obvious way. This error is deemed to be acceptable if it is less than the tolerance

$$\varepsilon_{tol} \equiv \varepsilon_a + \varepsilon_r|\tilde{\mathbf{f}}|, \tag{11}$$

where ε_a and ε_r are the specified absolute and relative error tolerances. These are set in `rinfo(1)` and `rinfo(2)`, respectively.

The specification of the error tolerance is crucial to the accurate and efficient use of ISATAB. The following should be considered in every new application.

1. Obviously, the smaller the error tolerance, the more accurately $\mathbf{f}^\ell$ approximates $\mathbf{f}$.

2. Specifying too small an error tolerance degrades performance because:

   (a) a larger table is needed to achieve peak performance (i.e., nearly all retrieves)

   (b) more time is needed to build the larger table

   (c) for given storage (and therefore maximum allowed table size) the table becomes full more quickly, resulting in more queries that cannot be fulfilled by retrieves.

3. The acceptable error depends on the application. Tests should be performed to determine the largest acceptable error tolerance.

14.4 Checkpointing

Setting `info(3)=1` causes the ISAT table to be checkpointed periodically to the file `isat_dat.1` (for table number 1). In a subsequent execution of the user's program, ISATAB starts from the previously-generated table (read from `isat_dat.1`) if `info(2)` is set to `1`. At any time, the checkpoint file can be written by calling `istab` with `info(9)=1`.

In general, ISATAB should be started from a checkpoint file only if the conditions of the two runs are identical.

14.5 Treatment of a Full Table

It is generally found that the ISAT table grows continually, even after a huge number of queries, albeit at a decreasing rate. Consequently, a table with a maximum allowed amount of storage will eventually become full. That is, additional EOA's cannot be added.

Two options are provided for this eventuality.

1. For `info(11)=0`, no new EOA's are added, and a query that cannot be fulfilled by a retrieve is fulfilled by a direct evaluation.

2. For `info(11)=1`, ISATAB continues to add new EOA's, by first removing existing EOA's. (This outcome is referred to as a *replace*.)

Which option is preferable depends on the nature of the problem at hand.

14.6 Number of Trees

The EOA's are stored in a specified number (n_t) of binary trees. In attempting to retrieve, ISATAB examines up to n_t EOA's to see if the query point is contained within it.

The value of n_t can be set in `info(5)`. (For the default setting `info(5)=0`, n_t is set to 4.) The optimal value of n_t is problem dependent. For simple problems n_t=1 may be optimal, since it yields the minimum retrieve time. Larger values of n_t yield slower retrieve times, but lead to smaller tables. Consequently less storage is required, and less time is needed to build the table.

14.7 Approximation of Derivatives

Large errors can result if values of $\mathbf{f}(\mathbf{x})$ are used in finite-difference approximations to derivatives such as (in the simplest scalar case)

$$f'(x) \approx [f(x+h) - f(x-h)]/(2h). \tag{12}$$

This is because in ISATAB $f(x+h)$ and $f(x-h)$ could be obtained from different EOA's, and could differ by ε_{tol}. Hence an error of order ε_{tol}/h in the approximation of $f'(x)$ can arise. Instead, for accuracy and efficiency, the piecewise-constant approximation $\mathbf{g}^0(\mathbf{x})$ should be used. The error in this approximation (in the simplest case) can be estimated to be of order $[\varepsilon_{tol} f''(x)]^{\frac{1}{2}}$.

14.8 The Namelist File `isat.nml`

Expert users can change some default settings through the namelist file `isat.nml`.

The real variable `chk_inc` (default value 1.2) controls the frequency of the table checkpointing. For `info(3)=1`, the table is checkpointed when the number of leaves is `chk_inc` times the number of leaves on the previous checkpointing.

Setting `if_flush=1` (default value 0) causes the output files to be flushed.

The logical variable `grow_linear` (default `.false.`) controls the returned value `fa` on a grow. The usual behavior (`grow_linear = .false.`) is to return the value of $\mathbf{f}$ obtained from direct evaluation. But for (`grow_linear = .true.`) the linear approximation $\mathbf{f}^\ell$ is returned instead.

15 Output from ISATAB

15.1 Overview of Files

When ISATAB is used for a single table (with `idtab=1`), the following input and output files may be used.

1. `isat_dat.1` — input/output file used (optionally) for checkpointing: see Section 14.4
2. `isat_log.1` — output log file
3. `isat_op.1` — output file containing statistics of ISATAB operation
4. `isat.nml` — optional namelist input file (for expert users only)

For each table considered (i.e., for each value of `idtab`) there may be corresponding output files. For example, the log file corresponding to `idtab=7` is `isat_log.7`.

15.2 Log File `isat_log.1`

This file should be examined when ISATAB is first applied to a new problem, or when there is an error condition. The file lists:

1. the principal input arguments to `isatab`, including the arrays `info` and `rinfo`
2. the array `rinfoi` which contains the values specified in `rinfo` but with default values set
3. the maximum number of table entries (i.e., EOA's) allowed given the specified amount of storage.

15.3 ISATAB Statistics

For `idtab=1`, the output file `isat_op.1` is produced if `info(1)` is set to `1`. Setting `rinfo(6)=1.0` – which is not recommended — produces one line of output for each query (i.e., each call to `isatab`). The output frequency can be reduced by increasing the value of `rinfo(6)`: the default is `rinfo(6)=1.02`.

Each line of `isat_op.1` consists of statistics of ISATAB's performance for the query (that generates the line of output), and also cumulative statistics. For example, the first four entries are the total numbers of queries, retrieves, grows and adds, respectively. The full details of the file `isat_op.1` are given in Section 16.5.

The Matlab script file `isat.m` is provided to process `isat_op.1`. This yields graphical output, and also the file `isat_stats` which summarizes ISATAB's performance.

Part IV
ISATAB REFERENCE MANUAL

16 Arguments of `isatab`

The call to `isatab` is:

```
  call isatab( idtab, nx, x, nf, nh, nhd, usrfgh,
               iusr, rusr, info, rinfo, fa, ga, ha, stats)
```

All `real` variables are double precision.

16.1 Input Arguments

`idtab`	integer: positive integer that identifies the function being tabulated. For a single table it is recommended to use `idtab=1`.
`nx`	integer: n_x, number of components of $\mathbf{x}$
`x`	real(nx): components of $\mathbf{x}$
`nf`	integer: n_f, number of components of $\mathbf{f}$
`nh`	integer: n_h, number of components of $\mathbf{h}$
`nhd`	integer: dimension of array `ha (nhd >= max(1,nh))`
`usrfgh`	name of the user-supplied subroutine that, given $\mathbf{x}$, returns values of $\mathbf{f}(\mathbf{x})$, $\mathbf{g}(\mathbf{x})$, and $\mathbf{h}(\mathbf{x})$
`iusr`	integer(*), optional input: user-defined integer array passed to `usrfgh`
`rusr`	real(*), optional input: user-defined real array passed to `usrfgh`
`info`	integer(20): integer array controlling ISATAB performance (see Section 16.3 for details)
`rinfo`	real(20+nx+nf): real array controlling ISATAB performance (see Section 16.4 for details)

16.2 Output Arguments

`fa`	real(nf): piecewise-linear approximation to $\mathbf{f}(\mathbf{x})$

`ga`	real(nf,nx): piecewise-constant approximation to $\mathbf{g}(\mathbf{x})$, which is returned only for `info(7)=1`. `ga(i,j)` is the approximation to $\partial f_i/\partial x_j$.
`ha`	real(nhd): piecewise-constant approximation to $\mathbf{h}(\mathbf{x})$
`stats`	real(50): statistics of ISATAB performance, which are returned only for `info(10)>0` (see Section 16.5 for details)

16.3 Input Array `info`

Shown below for each component of `info` is the corresponding local variable name (e.g., `info(1) = iscale`) and its definition. All default values of `info` are zero. Items marked `*` cannot be changed after the first call for a given table (i.e., for a given value of `idtab`), but they may be different for different tables. The value of `idtab` is denoted by #.

`*info(1)=iscale`	For `iscale=0`, $x_i^s =$ `xscale(i)` and $f_j^s =$ `fscale(j)` are taken to be unity; for `iscale=1`, `xscale(i)` and `fscale(j)` are taken from `rinfo`.
`*info(2)=ichin`	For `ichin=0`, the table is created from scratch; for `ichin=1`, the initial table is read from the file `isat_dat.#`.
`*info(3)=ichout`	Set `ichout=1` to checkpoint the table occasionally.
`*info(4)=isatop`	Set `isatop=1` for generate ISATAB performance output on the file `isat_op.#`.
`*info(5)=ntree`	`ntree` $= n_t$ is the number of trees to be used. If ntree is set to zero, then the default `ntree=4` is used.
`info(6)=noisat`	Set `noisat=1` to suppress ISATAB operation, in which case `fa` is obtained by direct evaluation from `usrfgh`. (This option is for testing only.)
`info(7)=ifdfdx`	Set `ifdfdx=1` if $\mathbf{g}^0$ (the approximation to $\mathbf{g}(\mathbf{x})$) is to be returned in `ga`.
`info(8)=idites`	Set `idites=1` to perform accuracy testing even if the query point lies inside an EOA. (This option is for testing only.)
`info(9)=ichfrc`	Set `ichfrc=1` to force checkpointing, and return immediately. On a call with `ichfrc=1`, the value of `fa` is not returned.

`info(10)=istats`	Set `istats=1` to return statistics in array `stats`, as well as performing normal ISATAB operations: set `istats=2` to return statistics in array `stats` without performing normal ISATAB operations.
`*info(11)=ifull`	Determines the action taken when the table becomes full and an "add" is indicated. For `ifull=0`, no new EOA's are added, and the query is resolved by direct evaluation: for `ifull=1`, an old EOA is deleted, and the new one is added.
`*info(12)=kcpv`	The value of `kcpv` determines how the cutting planes in the binary tree are generated. Valid values are 0, 1, 2 and 3, with 0 being the default.
`info(13)=kill`	Set `kill=1` to delete the table (to free memory for other purposes).
`info(14-16)`	These values should be set to zero.
`info(17=noadd)`	Set `noadd`=1 to prevent adding to the table. If the query cannot be resolved by a retrieve, then it is resolved by direct evaluation.
`info(18-20)`	These values should be set to zero.

16.4 Input Array `rinfo`

Shown below for each component of `rinfo` is the corresponding local variable name (e.g., `rinfo(1)=etola`) and its definition. A strictly positive value of `rinfo(1)` must be supplied. For other components, if zero is specified, then a default value is used. Items marked `*` cannot be changed after the first call for a given table (i.e., for a given value of `idtab`), but they may be different for different tables.

`*rinfo(1)=etola`	The specified absolute error tolerance, ε_a, which must be strictly positive. A value must be specified: there is no default.
`*rinfo(2)=etolr`	The specified relative error tolerance, ε_r. The default value is `etolr=0`.
`*rinfo(3)=dxsmax`	The maximum allowed size of an EOA. In the scaled **x**-space, this is the greatest allowed value of a principal axis of an EOA. This can be specified to prevent spurious EOA growth beyond a reasonable limit. The default value is `dxsmax=0.1`.

`*rinfo(4)=dxs0mx`	The maximum allowed initial size of an EOA. In the scaled **x**-space, this is the greatest allowed value of an initial principal axis of an EOA. The default value is `dxs0mx=0.1`. If `dxs0mx=0.0` is specified, then a different default value is used based on `etola` and `dxsmax`.
`*rinfo(5)=stomby`	The maximum storage (in megabytes) allowed for the ISATAB table. For best performance, specify the largest value that the computer system allows without paging. The default value is `stomby=50.`
`*rinfo(6)=outinc`	This controls the frequency of output onto the file `isat_op.#`. For `outinc=1.0`, there is one line of output for each query—which is likely to generate a huge file. To decrease the output frequency, increase `outinc`. The default is `outinc=1.02`.
`rinfo(7:20)`	These are not used, but are reserved for future use.
`*rinfo(20+i)`	The scale factors x_i^s for $i = 1, 2, \ldots, n_x$. These are required only for `info(1)=1`, and must be strictly positive.
`*rinfo(20+nx+j)`	The scale factors f_j^s for $j = 1, 2, \ldots, n_f$. These are required only for `info(1)=1`, and must be strictly positive.

16.5 Output Array `stats`

On "regular" calls to `isatab` (and for `info(10)>0`), the array `stats` is returned, and it is written to the output file `isat_op.#` (for `info(4)=1`). (It is not returned on "special" calls, namely when ISAT is by-passed (`info(6)=1`), or when the table is checkpointed (`info(9)=1`) or killed (`info(13)=1`).) The following shows the corresponding local variable names (e.g., `stats(1)=queries`) and their significance.

`stats(1)=queries`	total number of queries
`stats(2)=retrieves`	total number of queries resulting in retrieves
`stats(3)=grows`	total number of queries resulting in grows
`stats(4)=adds`	total number of queries resulting in adds
`stats(5)=replaces`	total number of queries resulting in replaces (because the table is full and `ifull=1`)

`stats(6)=dir_eval`	total number of queries resulting in direct evaluation (because the table is full and `ifull=0`)
`stats(7)=last`	action on last query: 2=retrieve, 3=grow, 4=add, 5=replace, 6=direct evaluation
`stats(8)=trees`	number of non-empty trees
`stats(9)=ntrees`	number of trees specified, n_t
`stats(10)=leaves`	number of leaves (and EOA's) in the table
`stats(11)=nleaves`	maximum number of leaves (and EOA's) allowed in the table
`stats(12)=min_leaves`	minimum (over trees) number of leaves
`stats(13)=max_leaves`	maximum (over trees) number of leaves
`stats(14)=max_depth`	maximum (over trees) of tree depth
`stats(15)=traverses`	total number of traverses performed
`stats(16)=trav_nodes`	total number of nodes encountered in traverses
`stats(17)=ngrows`	total number of EOA's that have been grown
`stats(18)=errsq`	the square of the error on the last EOA accuracy test
`stats(19)=tolsq`	the tolerance (squared) on the last EOA accuracy test
`stats(20)=sincef`	number of queries since `usrfgh` was last called to evaluate $\mathbf{f}(\mathbf{x})$
`stats(21)=sinceg`	number of queries since `usrfgh` was last called to evaluate $\mathbf{g}(\mathbf{x})$
`stats(22)=cpu_sec`	cpu seconds for this call
`stats(23)=cpu_cum`	cumulative cpu seconds in `isatab` (for this table)
`stats(24)=cpu_out`	cumulative cpu seconds outside `isatab` (for this table)
`stats(25)=cpu_ret`	cumulative cpu seconds spent on retrieves
`stats(26)=cpu_grow`	cumulative cpu seconds spent on grows
`stats(27)=cpu_add`	cumulative cpu seconds spent on adds
`stats(28)=cpu_rep`	cumulative cpu seconds spent on replaces

`stats(29)=cpu_dev`	cumulative cpu seconds spent on direct evaluations
`stats(30:46)`	reserved
`stats(47)=errsqmax`	square of the maximum (over retrieves) error on EOA accuracy tests: significant only for info(8)=1
`stats(48:50)`	reserved

17 Arguments of `usrfgh`

A subroutine (referred to here as **`usrfgh`**) must be provided to evaluate (on request) $\mathbf{f}(\mathbf{x})$, $\mathbf{g}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$. The calling sequence is:

```
    call usrfgh( need, nx, x, nf, nh, iusr, rusr, fa, ga, ha )
```

All real arguments are double precision. The arguments are as follows:

`need`	integer(3), input: the values of `need` indicate whether or not $\mathbf{f}(\mathbf{x})$, $\mathbf{g}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$ need to be evaluated. For `need(1)=0`, $\mathbf{f}(\mathbf{x})$ does not need to be evaluated: for `need(1)=1`, $\mathbf{f}(\mathbf{x})$ needs to be evaluated. Similarly, `need(2)=1` and `need(3)=1` indicate that $\mathbf{g}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$, respectively, need to be evaluated.
`nx`	integer, input: n_x, number of components of $\mathbf{x}$
`x`	real(nx), input: components of $\mathbf{x}$
`nf`	integer, input: n_f, number of components of $\mathbf{f}$
`nh`	integer, input: n_h, number of components of $\mathbf{h}$
`iusr`	integer(*), optional input: user-defined integer array passed through `isatab`
`rusr`	real(*), optional input: user-defined real array passed through `isatab`
`fa`	real(nf), output: components of $\mathbf{f}(\mathbf{x})$ which must be returned if `need(1)=1`
`ga`	real(nf,nx), output: components of $\mathbf{g}(\mathbf{x})$ which must be returned if `need(2)=1`; `ga(i,j)`$=\partial f_i/\partial x_j$
`ha`	real(nhd), output: components of $\mathbf{h}(\mathbf{x})$ which must be returned if `need(3)=1`