

# The MCTDH Program: Structure and Development.

Graham Worth

King's College London

1. Requirements of nuclear dynamics
2. Program structure
  - passing and storage of information
3. Analyse programs

## Aims

- General dynamics program based on MCTDH  
– complex but powerful
- User friendly
- Efficient
- Easy to develop
- Modular (independent pieces), with library routines for frequent operations

## Nuclear dynamics

In a basis set (DVR, HO, ...)

$$\Psi(\mathbf{q}, t) = \sum_{j_1 \dots j_f}^N A_{j_1 \dots j_f}(t) \varphi_{j_1}^{(\kappa)}(q_1) \cdots \varphi_{j_f}^{(\kappa)}(q_f)$$

$$\begin{array}{ccc} & \mathbf{H}\Psi = i\dot{\Psi} & \\ & \swarrow \quad \nwarrow & \\ \text{matrix} & & \text{vector} \end{array}$$

1. Set up basis
2. Set up operator
3. Set up initial  $\Psi(t_0)$
4. Propagate  $\Psi(t_0) \rightarrow \Psi(t)$   
– evaluate information
5. Analyse information

```
#####  
###          Photodissociation on NOCl S1 surface          ###  
#####
```

RUN-SECTION

```
name = nocl1      propagation  
tinit = 0.0      tfinal = 25.0  tout   = 1.0   tpsi   = 1.0  
output  auto    timing  steps  update  speed  gridpop  psi  
end-run-section
```

OPERATOR-SECTION

```
opname = nocl1  
end-operator-section
```

SPF-BASIS-SECTION

```
rd   = 5      rv   = 5      theta = 5  
end-spf-basis-section
```

PRIMITIVE-BASIS-SECTION

```
#Label   DVR   N   Parameter  
rd       sin  36  3.800   5.600           # N, xi, xf  
rv       HO   24  2.136   0.272,eV  7.4667,AMU     # N, x-eq, freq., mass  
theta    Leg  60  0       all            # N, l_z, sym  
end-primitive-basis-section
```

INTEGRATOR-SECTION

```
CMF/var = 0.5, 1.0d-5      # initial step size (fs), CMF-accuracy  
BS/spf  = 8 , 1.0d-6      # order, accuracy, [initial step size]  
SIL/A   = 15 , 1.0d-6, standard # order, accuracy, error-estimate  
end-integrator-section
```

INIT\_WF-SECTION

```
file = nocl0  
end-init_wf-section
```

end-input

## Operation of Hamiltonian on $\Psi$ in DVR

$$\mathbf{H} = \mathbf{T}_1 \oplus \mathbf{T}_2 \oplus \cdots \mathbf{T}_f \oplus \mathbf{V}$$

The diagram illustrates the decomposition of the Hamiltonian  $\mathbf{H}$ . It is shown as a direct sum of terms:  $\mathbf{T}_1 \oplus \mathbf{T}_2 \oplus \cdots \mathbf{T}_f \oplus \mathbf{V}$ . Arrows indicate the dimensions of the components:  $N^f \times N^f$  points to  $\mathbf{H}$ ;  $N \times N$  points to  $\mathbf{T}_1$  and  $\mathbf{T}_2$ ; and  $N^f$  points to  $\mathbf{V}$ .

$$\begin{aligned} \Psi_I &= \Psi(i_1, i_2, \dots, i_\kappa \dots, i_f) \\ &= \Psi(I_\nu, i_\kappa, I_n) \end{aligned}$$

$$(\mathbf{V}\Psi)_{i_1, \dots, i_f} = V(i_1, \dots, i_f) \Psi_{i_1, \dots, i_f}$$

“vector”  $N^f$  operations

$$(\mathbf{T}\Psi)_{I_\nu, j_\kappa, I_n} = \sum_{i_\kappa} T_{j_\kappa, i_\kappa} \Psi_{I_\nu, i_\kappa, I_n}$$

matrix  $\times$  tensor  $N^{f+1}$  operations

## The MCTDH EOMs

$$\begin{aligned}\Psi &= \sum_{j_1 \dots j_f}^n A_{j_1 \dots j_f}(t) \varphi_{j_1}^{(1)}(t) \dots \varphi_{j_f}^{(f)}(t) \\ &= \sum_J A_J \Phi_J\end{aligned}$$

Basic Form:

$$\begin{aligned}i\dot{\mathbf{A}} &= \mathbf{H}\mathbf{A} \\ i\dot{\varphi} &= (1 - P)\rho^{-1}\mathcal{H}\varphi\end{aligned}$$

where

$$H_{IJ} = \langle \Phi_I | H | \Phi_J \rangle$$

Using

$$\hat{H} = \sum_n c_n h_n^{(1)} \dots h_n^{(f)}$$

SPF operator matrices

$$H_{IJ,n} = \langle \varphi_{i_1} | h_n^{(1)} | \varphi_{j_1} \rangle \dots \langle \varphi_{i_f} | h_n^{(f)} | \varphi_{j_f} \rangle$$

so  $n \times n$  matrix  $\times$  tensor operations

$$\mathbf{H}\mathbf{A} = \sum_n \mathbf{c}_n \mathbf{h}_n^{(1)} \dots \mathbf{h}_n^{(f)} \mathbf{A}$$

Similar operation used to build mean-fields

NB  $\varphi$  is a matrix

$$\varphi_{i\alpha} = \langle \chi_\alpha | \varphi_i \rangle = \varphi_i(x_\alpha)$$

so  $\mathcal{H}\varphi$  is ( $n \times n$  matrix)  $\times$  ( $n \times N$  matrix)

$h_n\varphi$  is ( $N \times N$  matrix)  $\times$  ( $N$  vector)

Form 2:  $H = \sum_{\kappa} h_{\kappa} + H_R$

$$i\dot{\mathbf{A}} = \mathbf{H}_R \mathbf{A}$$

$$i\dot{\varphi} = (h_{\kappa} + (1 - P)\rho^{-1}\mathcal{H}_R) \varphi$$

Form 3:  $\langle \varphi_i | \dot{\varphi}_j \rangle = -i \langle \varphi_i | g_{\kappa} | \varphi_j \rangle$

$$i\dot{\mathbf{A}} = \left( \mathbf{H}_R - \sum g_{\kappa} \right) \mathbf{A}$$

$$i\dot{\varphi} = (h_{\kappa} + g_{\kappa} + (1 - P)\rho^{-1}\mathcal{H}_R) \varphi$$

Form 4. Multiset

$$\Psi = \sum_J^{(\alpha)} A_J^{(\alpha)} \Phi_J^{(\alpha)} \quad (1)$$

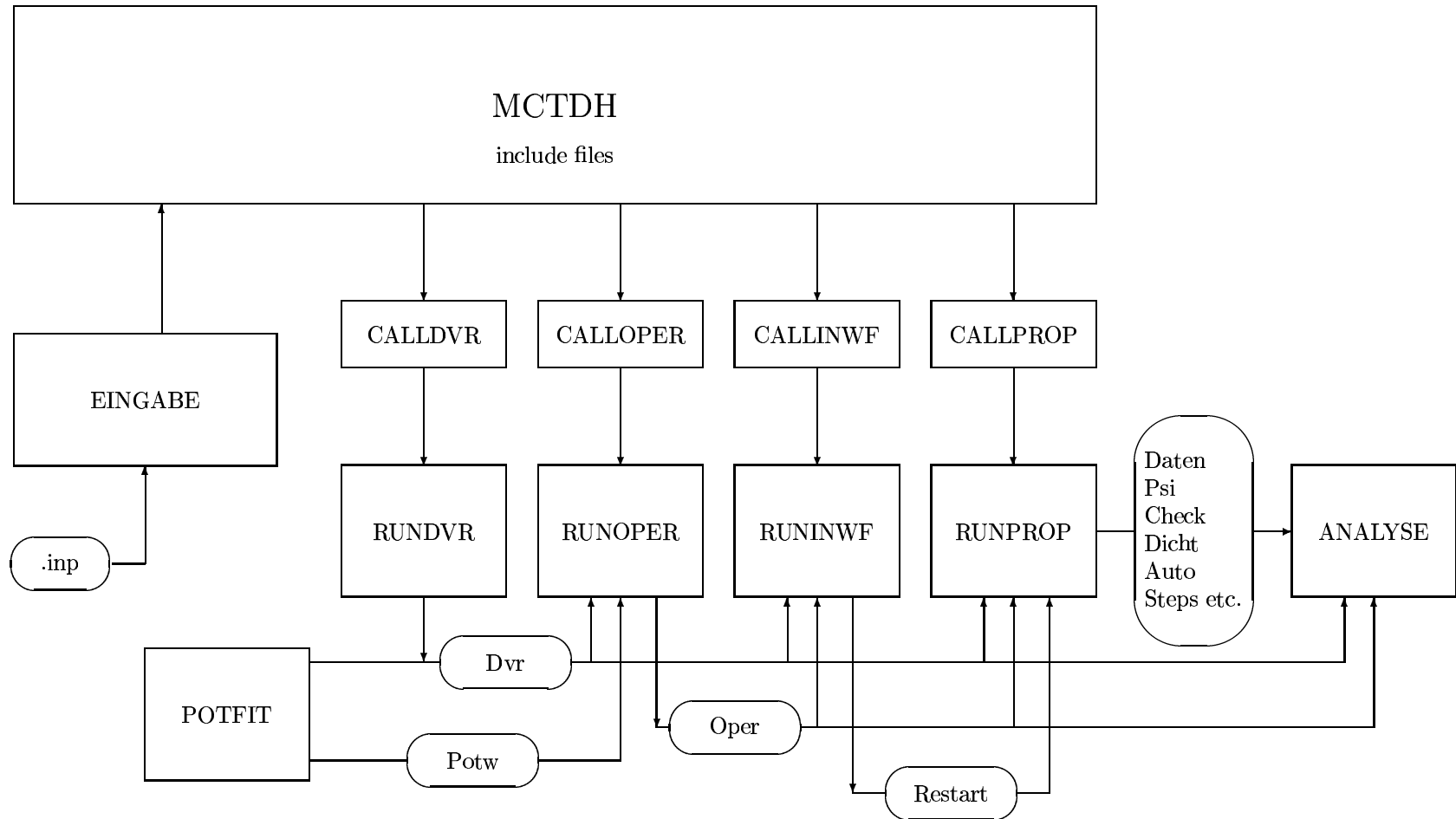
$$i\dot{\mathbf{A}}^{(\alpha)} = \sum_{\beta} \mathbf{H}^{(\alpha\beta)} \mathbf{A}^{(\beta)}$$

$$i\dot{\varphi}^{(\alpha)} = (1 - P^{(\alpha)})\rho^{(\alpha)-1} \sum_{\beta} \mathcal{H}^{(\alpha\beta)} \varphi^{(\beta)}$$

Form 5. Combined modes.  $\varphi(q_i, \dots, q_j)$

SPFs now also use tensor structure to access grid points for one DOF  $\varphi(q_v, q_{\kappa}, q_n)$

# The MCTDH Program Structure



## Include files

global.inc	general information, e.g. max sizes
paths.inc	paths where text files are stored
timing.inc	timing information
versions.inc	version information
maxdim.inc	maximum no. of degrees of freedom
maxkoe.inc	maximum no. of Hamiltonian terms
maxsta.inc	maximum no. of states
griddat.inc	Information on primitive basis (grid)
operdef.inc	Information on operator
psidef.inc	Information on wavefunction
runX.inc	Information needed only in part X
genX.inc	list of include files needed in part X
hpsi.inc	Information required for $H \times \varphi$

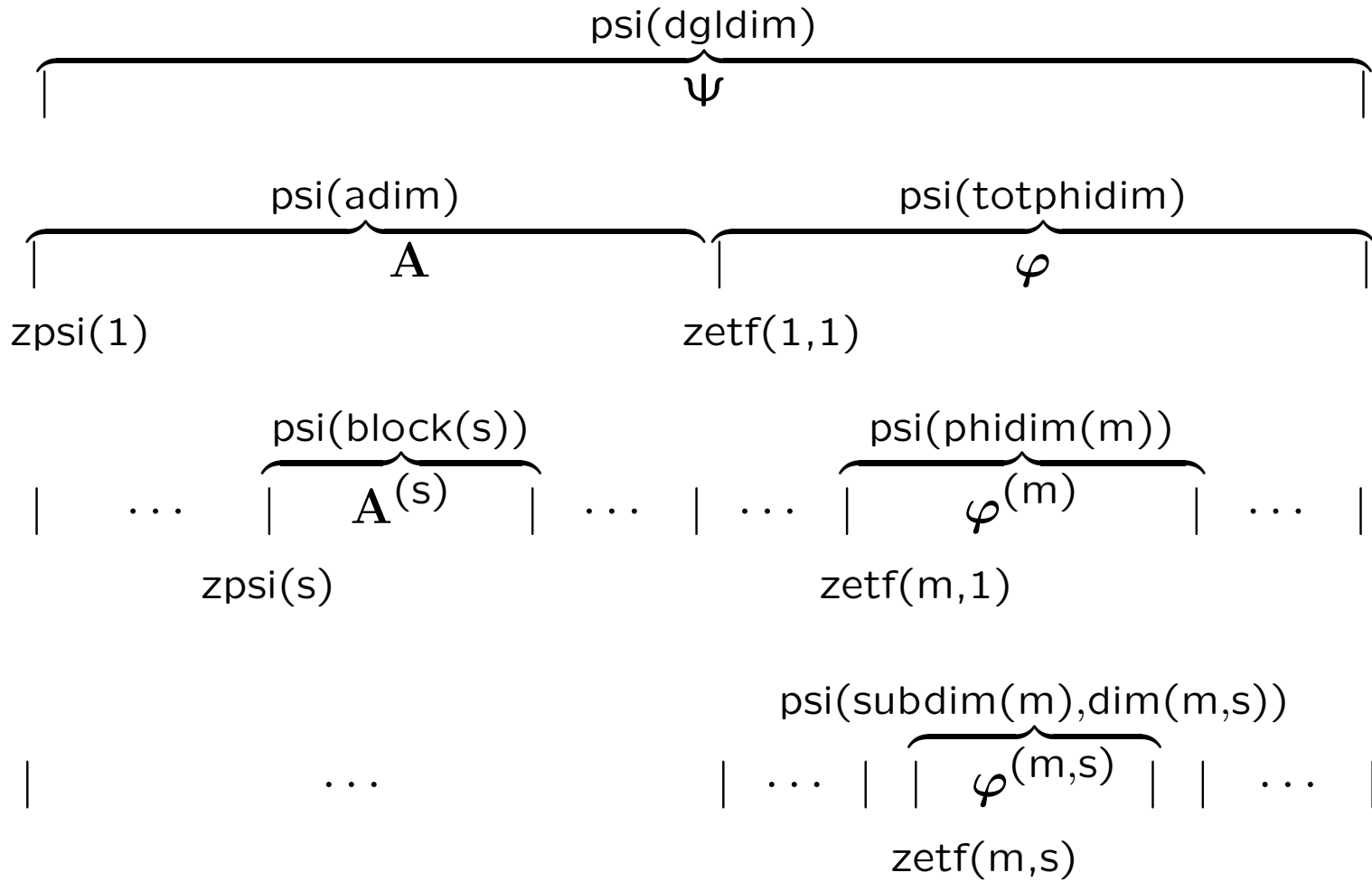
## Memory

main memory arrays:

psi, dtpsi	Wavefunction, psi time derivative
mc, mr, mi, ml	complex*16, real*8, integer, logical arrays
ms, mf	complex*8, real*4 in ANALYSE

- Pointers allow arrays to be “unfolded” into entities, e.g. hloch (mean fields), dicht2 (reduced density matrices) etc.
- Sizes and pointers for entity arrays calculated in zeigX.F
- Pointers for memory arrays allocated in memX.F
- Memory arrays dynamically allocated for each part using callX C-routines

X = dvr, inwf, oper, prop, ....



from subroutine funkr (propwf/function.F):

---

```
C-----  
C Loop over correlated Hamiltonian terms  
C 1. compute hpsi array  
C 2. compute hteil matrices  
C 3. compute hloch matrices and dtpsi for A coefficients  
C 4. multiply hloch and hpsi and store in dtpsi  
C-----  
    ktype=0  
    call hphi(time,psi,mc(mchpsi),  
+          mc(mchc),mr(mrhr),mi(mihi),ml(mlhl),ktype,1)  
  
    call phihphi(psi,mc(mchpsi),mc(mchteil),1,usehsym)  
  
    if (dentype .eq. 0) then  
        call mfields(psi,dtpsi,mc,mr,mi,ml,1,psi,0,lconst,usehsym,  
+          time)  
    else if (dentype .eq. 2) then  
        call d2mfields(psi,dtpsi,mc,mr,mi,ml,time)  
    endif  
  
    call hlochphi(dtpsi,mc(mchpsi),mc(mchloch),1,0)
```

from subroutine hphi (propwf/hphi.F):

---

```
subroutine hphi(time,psi,hpsi,hc,hr,hi,h1,ktype,nham)

implicit none

#include "global.inc"
#include "propwf.inc"

integer    m,ktype,nham
real*8     time
complex*16 psi(dgldim),hpsi(hpsidim)
complex*16 hc(*)
real*8     hr(*)
integer    hi(*)
logical    h1(*)

C --- LOOP OVER EACH MODE AND STATE ---

do m = 1,nmode
  call hphi1m(time,psi(zetf(m,1)),hpsi,m,zetf(m,1),
+          phidim(m),hc,hr,hi,h1,ktype,nham)
enddo

return
end
```

from subroutine hphi1m (propwf/hphi.F):

---

```
subroutine hphi1m (time,psi,hpsi,m,zetf1,phidim1,
+                hc,hr,hi,hl,ktype,nham)
.
.
complex*16 psi(zetf1:zetf1+phidim1-1),hpsi(hpsidim)
.
.
C --- LOOP OVER EACH HAMILTONIAN TERM ---

do k=k1,k2
  s1=ki(k)
  if (kcalc(k)) then
    call hphi1mk(psi(zetf(m,s1)),hpsi(zhpsi(m,k)),
+              m,k,dim(m,s1),subdim(m),hc,hr,hi,hl)
  endif
enddo
```

from subroutine hphi1mk (propwf/hphi.F):

---

```
subroutine hphi1mk (psi,hpsi,m,k,dim,subdim,hc,hr,hi,hl)
.
.
complex*16 psi(subdim,dim),hpsi(subdim,dim)
.
.
do e = 1,dim
  call hcall(m,k,hpsi(1,e),psi(1,e),subdim,hc,hr,hi,hl)
enddo

return
end
```

from subroutine hcall (propwf/hphi.F):

---

```
subroutine hcall(m,k,hpsi,psi,subdim1,hc,hr,hi,hl)
.
.
complex*16 psi(subdim1),hpsi(subdim1)
.
.
call hop(f,k,hpsi,psi,hr(hrhops),subdim1,
+      hi(hifftfak),hc(hchin),hc(hcrueck),
+      hc(zeig),hc(zeig1))
```

## Tensor structures

In A-vector,

$$\text{block}(s) = \text{vdim}(m,s) \times \text{dim}(m,s) \times \text{ndim}(m,s)$$

In SPF,

$$\text{subdim}(m) = \text{vgdim}(f) \times \text{gdim}(f) \times \text{ngdim}(f)$$

## Modes and DOFs

In general  $\varphi^{(\kappa)}(Q_\kappa)$  where  $Q_\kappa = \{q_i, q_j, \dots\}$  (combined modes)

Differentiate between “modes” and “dofs” (degrees of freedom).

DOF info corresponds to PBASIS-SECTION

f = 1,ndof

modelabel(f)	assigned label
gdim(f)	no. of primitive basis functions
pbasis(f)	type of DVR

MODE info is “coordinates” in propagation

m=1,nmode

nspfdo(m)	no. of DOFs in mode
spfdof(n,m)	which DOF is nth mode coordinate
dofspf(f)	in which mode is DOF f
subdim(m)	no. of mode grid points
dim(m,s)	no. of SPFs for mode (and state)

from subroutine zeigdvr (gendvr/zeigdvr.F):

---

```
C-----  
C pointers for ort  
C-----  
    zeig=1  
    do f=1,ndof  
        zort(f)=zeig  
        zeig=zeig+gdim(f)  
    enddo  
    ortdim=zeig-1
```

from subroutine genpsi (geninwf/genpsi.F):

---

```
C-----  
C move initially populated spf to be 1st in list  
C-----  
    do s=1,nstate  
        do m=1,nmode  
            do n=1,nspfdof(m)  
                f=spfdof(n,m)  
                pop=isbaspar(4,f,s)  
                if (pop.ne.1 .and. dimf(f,s).gt.0 .and. sbasis(f).ne.3)  
+                   then  
+                   call swaspsf(spfl1d(zspfl1d(f,s)),workc,pop,gdim(f),  
+                   dimf(f,s))  
                endif  
            enddo  
        enddo  
    enddo
```

## Read-Write files

3 files:

dvr        DVR (primitive basis set)  
oper      Operators  
restart    Wavefunction and propagation

File Structure:

File version number  
System Information (dvrdef, grddef, psidef)  
RW File Information  
RW Data

Interface routines:

gendvr/wrdvr.F        mctdh/rddvr.F  
genoper/wroper.F     mctdh/rdoper.F  
mctdh/iorst.F (wrrst, rdrst)  
Can read information separately from data.

System information interface routines:

mctdh/iodvrdef.F     mctdh/iogrddef.F  
mctdh/iopsidef.F

from subroutine runinwf (geninwf/runinwf.F):

---

```
C-----  
C Read DVR data needed to build wavefunction (see rddvr.f for details)  
C-----  
    call zerovxl(dvrdata,nrwdata)  
    dvrdata(1)=.true.  
    dvrdata(2)=.true.  
    dvrdata(5)=.true.  
    dvrdata(6)=.true.  
    dvrdata(7)=.true.  
    dvrdata(8)=.true.  
    dvrdata(9)=.true.  
    chkdvr=2  
    call rddvr(mr(mrort),mr(mrtrafo),rdum,rdum,mc(mchin),mc(mcrueck),  
+    mi(mifftfak),mi(mijsph),mi(mimsph),rdum,chkdvr)  
  
C-----  
C Read data needed by the operator  
C-----  
    chkdvr=2  
    chkgrd=2  
    call rdoper(mr(mrhops),chkdvr,chkgrd)
```

chkdvr, chkgrd, chkpsi flags:

- 0 read and do nothing
- 1 read and store info
- 2 read and check against stored info
- 3 read and check as subset of system (dvrdef only)

## Updating RW files: Version numbers

-ver keyword produces information on program version

```
linux1: 80 > mctdh82 -ver
```

```
-----  
***** Source code version *****
```

```
Program Version :      8
```

```
Release          :      2
```

```
Revision         :      2
```

```
Heidelberg PRCS repository no. : $ProjectVersion$
```

```
Compiled: Mon Jul 16 11:41:54 BST 2001 ; Host: linux1
```

```
Included surfaces: none
```

```
Default operator-path: /home/graham/mctdh82.2/operators  
-----
```

In include/versions.inc is the version number

```
parameter (projver = 8.2002d0)
```

This is written at the beginning of every RW (and data) file to identify the format.

If changes are made to RW files (or data files), e.g. new information output, this number must be increased. Then, to maintain backward compatibility edit routine that reads edited file using the version number as control structure.

e.g. in subroutine dvrinfo (mctdh/rddvr.F)

---

```
rewind(idvr)
read(idvr) filever(idvr)

file='Dvr'
call zerovxi(chkerr,nerr)
call rddvrdef(idvr,chkdvr,ndof1,fdvr)
call chksyserr(file,lerr)
if (lerr) return

read(idvr) buffer

if (filever(idvr) .ge. 8.2002d0) then
  read(idvr) ortdim,dvrdim,fftdim,expdim,sphdim
else
  read(idvr) ortdim,dvrdim,fftdim,sphdim
endif
```

Files are identified by a channel, idvr, ioper, ichk, etc. The file versions are stored in filever(idvr), etc.

## Modules and workspace

Subroutines grouped together in files as modules, e.g. in propwf

hphi.F	Operation of H on spfs.
mfields.F	Building the mean fields
output.F	Controls the output of information
function.F	Calculating $\dot{A}, \dot{\varphi}$
daten.F	Calculates data such as energies and state populations for output

Work space may be used in a module:

mc(mcworkc) → workc(workcdim)  
mr(mcworkr) → workr(workrdim)

This is assigned in e.g.

```
subroutine wkhphi

implicit none

#include "global.inc"
#include "propwf.inc"

C-----
C assign workspace used in module
C-----

workcdim=max(workcdim,totphidim)
workrdim=workrdim
workidim=workidim
workldim=workldim
```

In zeigX calls to each module work allocation routine (via subroutine wkX) are made. Maximum workspace workcdim etc. stored in include files.

## Adding keywords

Include files allow easy addition of new keywords. Each section in .inp file has a separate IO routine,

RUN-SECTION	mctdh/einrun.F
SBASIS-SECTION	mctdh/einsbas.F
PBASIS-SECTION	gendvr/einpbas.F
OPERATOR-SECTION	genoper/einoper.F
INIT_WF-SECTION	geninwf/eininwf.F
INTEGRATOR-SECTION	propwf/einint.F

In mctdh/einrun.F, loop starts

```
C-----  
C READ KEYWORDS:  
C-----  
 10 continue  
    call rdinpf(iin,keyword,keyorig,inptit,lc,ic,iz,ierr,maxkey)  
    message = inptit  
    call cntrl(keyword(1),lc(1),ierr)  
  
    i=1  
 15 continue  
    if (keyword(i).eq.'end-run-section') then  
        goto 20
```

and continues with entries such as

```
else if( keyword(i) .eq. 'geninwf' ) then
  lruninwf=.true.

else if (keyword(i) .eq. 'readdvr') then
  lrddvr=.true.
  if (keyword(i+1) .eq. '=') then
    i=i+2
    dname=keyorig(i)
    dlaenge = lc(i)
    call abspath(dname,dlaenge)
  endif
endif
```

Add new keyword to list with options, add new flags to relevant include files. Edit code locally.

## Analysing the data files

Stand alone files.

psi	Wavefunction
gridpop	one-dimensional densities
check	information on propagation (state populations, natural orbital populations, expectation values, etc.
auto	autocorrelation file
....	
pes	PES file generated by -pes
...	

Similar in structure to RW files:

File version number

System Information (dvrdef, grddef, psidef)

Data File Information

Data

Either

- General program, e.g. to plot natural orbitals or
- Specific program for a particular property / system

ANALYSE programs provide interfaces / memory handling routines.

Updating of files can take place without destroying programs.

## The ANALYSIS program

The standard programs can be used directly, or via plot scripts. Even easier is the ANALYSIS interface.

Uses interactive menus and GNUPLOT to visualise data.

```
linux1: 77 > analysis82
```

```
*****
```

### THE HEIDELBERG MCTDH PROGRAM ANALYSIS PACKAGE

```
Program Version :    8
Release         :    2
```

```
*****
```

```
Present directory is: /home/graham/runs/allenex/allx2d
```

- 0 = stop
- 1 = list / change directory
- 2 = analyse convergence
- 3 = analyse integrator
- 4 = analyse results
- 5 = analyse system evolution
- 6 = analyse potential surface
- 7 = compare calculations

Option 1 allows browsing of directories. Simply type name of directory using relative or absolute path .

Select an option, e.g.

2

\*\*\* Analyse convergence \*\*\*

0 = return to main menu

1 = check orthonormality of spfs in psi file

2 = check orthonormality of spfs in restart file

3 = plot populations of natural orbitals

4 = plot populations of grid edges

5 = plot time-evolution of norm of wavefunction

6 = norm of wavefunction on restart file

99 = print last screen output

100 = change printer (lpr)

Again select an option or return to main menu.

Select again from main menu

4

\*\*\* Plot Results \*\*\*

0 = return to main menu

1 = Plot autocorrelation function

2 = Plot Fourier Transform of autocorrelation function

3 = Plot spectrum from autocorrelation function

4 = Plot eigenvalues from matrix diagonalisation

select to plot the autocorrelation function

1

0 = stop  
1 = plot to screen  
2 = print plot  
3 = save plot to a postscript file  
4 = save plot to a gnuplot file (use after 1 or 2)  
5 = save data to an xyz file

10 = change plot task (plot Absolute values)  
30 = change time bounds  
40 = show time info  
70 = set Y-range for plot (auto)  
80 = change coordinate units  
90 = change Y-axis units (au)

150 = toggle grid (off)  
240 = toggle key (on)  
250 = toggle show points (off)  
260 = toggle stick spectrum (off)  
270 = toggle smooth curve (off)

900 = toggle gnuplot output format (on)  
910 = change printer (lpr)  
920 = change GNUplot command

10

1 = plot Real values  
2 = plot Imaginary values  
3 = plot Absolute values

etc....

from subroutine analres (analyse/analysis.F):

---

```
write(6,*)
write(6,'(a)') ' *** Plot Results ***'
write(6,'(a)') ' 0 = return to main menu '
write(6,'(a)') ' 1 = Plot autocorrelation function'

read(5,*) task

if (task .eq. 0) then
  go to 999
else if (task .eq. 1) then
  inquire(file=name(1:laenge)//'/auto',exist=lexist)
  if (lexist) then
    line='rdauto'//aversion
    s=slen(line)
    line=line(1:s)//' -inter -i '//name(1:laenge)
    s=slen(line)
    line = line(1:s)//char(zeroi)
    call excmd(line)
  else
    write(6,'(/,a)') 'Needs an auto file from a propagation run'
  endif
```

Effectively calls program  
rdauto -inter  
using data in directory "name"

Easy to add new items to menus

## Writing a new program

aglobal.inc usually needed

analyse/adefault.F sets defaults

analyse/plotutils.F (+ plot.inc) contains interactive plotting routines.

See e.g. analyse/rdauto.F or analyse/rdeigval.F for simple programs that read / plot data from a file.

analyse/rdfiles.F contains IO routines for data files.

analyse/zeigausw.F and analyse/memausw.F control memory allocation

analyse/template.F provides template for writing new program.

```
C-----  
C!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
C substitute "template" with program name and comment in  
C-----  
    program statepop
```

Setting flags enables program to set up memory requirements / pointers

```
C-----  
C!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
C Declare files to be read  
C set flag to .true. if file is to be read  
C  
C   lrddvr    DVR file  
C   lrdoper   OPER file  
C-----  
C  
C   lrddvr=.false.  
C   lrdoper=.false.
```

Which MCTDH arrays are to be used?

```
C-----  
C!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
C Declare resources required (other than those needed to read a file)  
C  
C ncomp  : no. of comparison data sets to be used.  
C         See auswutil.F for details of use of comparison sets  
C lpsi   : wavefunction  
C lpsi1  : second wavefunction  
C lpsigrd : wavefunction to be used in grid representation (rather  
C         than spf)  
C ldmat  : density matrices for psi  
C-----  
C  
C   ncomp=0  
C   lpsi=.false.  
C   lpsi1=.false.  
C   lpsigrd=.false.  
C   ldmat=.true.
```

Select file to be read

```
C-----  
C!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
C open input and output data files  
C  
C use standard channels if possible  
C irho : gridpop file  
C ipsi : psi file  
C iaut : auto file  
C ichk : check file  
C iaus : output file  
C-----  
      filename=filein(1:laein)  
      open(ichk,file=filename,form='unformatted',status='old',err=1000)
```

Due to selections made, pointers for dicht3 are set, so  
in subroutine receiver

```
C-----  
C!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
C provide arguments of pointers to arrays needed for analysis  
C e.g. mc(mcpsi) for psi array  
C-----  
      call substpop(mr(mrdicht3))  
  
.....  
.....  
C*****
```

```
      subroutine substpop (dicht3)  
  
      implicit none  
  
#include "aglobal.inc"  
#include "analyse.inc"  
#include "plot.inc"
```

Read check file in a loop, and store in array for interactive plotting.

```
C-----  
C read check file  
C-----  
    time=tinit  
    ndata=0  
100 continue  
    lrdchk(1)=.true.  
    lrdchk(2)=.true.  
    lrdchk(3)=.false.  
    call rdchk(ichk,ecorr,etot,spop,dicht3,r dum)  
    if (lend) go to 200
```

In different context, rdum can be replaced by an array, and lrdprop(n)=.true. can be set to read different properties.

```
    if (linter) then  
        ndata=ndata+1  
        if (ndata .le. maxdata) then  
            pltime(ndata)=time/fs  
            do n=1,nstate1  
                pldata(ndata,n)=spop(n)  
            enddo  
        endif  
    else  
        write(6,'(f8.3,3e20.8)') time,(spop(n),n=1,nstate1)  
    endif
```

Basic interactive plotting loop is then:

```
250 continue
    call plotmenu(task,plottask,plotdim)
```

```
C-----
C do task
C TASK=0 STOP  else return to menu
C-----
    if (task .gt. 0) then
        if (plottask .eq. 1) then
            nsets=nstate1
            plttitle='State population'
            axislab(1)='Time [fs]'
            axislab(3)='Population'
        endif
        if (lrebound) then
            call databnds(pldata(1,plottask),dimx,dimy,nsets,nplots,
+                maxdata,1,maxsta,1)
        endif
        call dotask(task,plottask,plotdim,pltime,ydum,
+            pldata(1,plottask),dimx,dimy,nsets,nplots,
+            maxdata,1,maxsta,1,xdum,1,pltime,ndata)

        goto 250
    endif
```