Data Handling Software

An Experimentalists View

Roman Flesch
Fachbereich Physik, Univ. Osnabrück
Barbarastr. 7, 49069 Osnabrück

What do I do?

- Work in different Labs at different sites
- Obtaining Data in a Variety of Experiments: one-dimensional data (spectra), multi-dimensional data (coincidence exp.)
- Raw data rreduction, simple math with data
  - Normalization of Data
  - Subtraction of Data from other Data
- Comparison of Data with model functions
- Fitting my data against model functions, taking instrumental response functions into account
- Calculate comparatively simple things, like FC Factors for inharmonic oscillator analysis and the like.

General Needs

- Fast data import
- Get very good graphics and figures in publication-quality, including:
  - arbitrary complex annotations in the figure
  - free configuration of everything in the figure
  - getting graphix output in EPS and PDF

Programming Needs

- Integrated high-level programming language (do not want to pass pointers to structs to a function in everyday work ==> data structures such as arrays must be a genuine TYPE)

  ```
  dataset_1 = dataset_1^2;
  ```

- Common tasks, operations, and functions, should be predefined

  ```
  VoigtFit  dataset_1;
  Display dataset_1;
  ```

- „Worst case scenario“: ability to link external C code to get efficient procedures (copy Numerical Recipies)

  ```
  #include <MyExternalFunction>;
  ```

Fitting Capabilities

- Fitting to pre-defined, simple functions [line fit, polynomials, exp, sigmoidals etc.] „trivial fit functions“
- Fitting to user-defined analytical functions, including „higher“ functions, by using optimization methods: □^{2}=f(c_1, c_2, (...))
- Fitting to non-analytical stuff, like convolutions [exponentially modified Gaussians, Voigt Functions, and the like]
- Multi-Peak Fitting (fit something to 7 Voigt functions)

More Things I need

- Two-fold approach to commands:
  - Terminal to pass commands efficiently, with command history (like Unix C-Shell)
  - Menu access to less common commands or those that are hard to memorize
- It should fit on my lap-top
- Ability to create a „project“ that includes everything that belongs to a paper (do not want to manage N_A files myself)
- Something that assists me through the complete process from importing data through data analysis to getting figures for publishing…

Can I help You, Master?

Working with Igor

- Starting Igor creates a new project
- Igor open a command window
- Typically, in an empty project, I would start by importing my data. I do this by using menu-driven commands (since it is easier).

Igor Data Structures

- Variables: single or double precision float, int, „word“, ...

  ```
  variable var1, var2, var3 /* variable declaration */
  var1 = 5             /* variables as lvalues */
  var1 * var1 = 5   /* ERROR!*/
  var2 = exp(-3)     /*returning a predefined function*/
  var3 = var1*var2 /* variables as rvalues */
  var1 += var2 * cos(sqrt(var3))
  ```

- Strings:

  ```
  string str1 = "Yes"
  str1 += ", Master?"
  printf "%s", str1  /* prints "Yes, Master?" */
      /* need no function to compare two strings */
  ```

- Waves
  Data structures containing an array of variables, strings, or waves(!)
  plus additional information.
  Waves are the central data structure in IGOR. Specifically, the graphical
  output is related to waves.

Waves

- A wave is an array with a scaling. The concept of a wave assumes that the data are equally spaced. An unequal spacing will get you into trouble!
- A Wave is can be set equal to an rvalue in an assignment. The result of the assignment depends on the rvalue.
- <u>EXAMPLES</u>

```
make/N=256 wave_1
wave_1 = 3
```

[We have now 256 times the „3“ in the array]

```
make/N=256 wave_1
setscale x, 0, 10, wave_1 /* x is a keyword */
wave_1 = 3 * x
```

[refers to the scaling of the wave, and returns three times the xvalue of each point into the point, yielding y = 3x]

- Equivalent to the following plain code: ---> ~/MyProgs/IGOR1.c
- Binary operators can be used to create waves, like in

```
w1 = (x >= 5) * exp(-x)
```

Dependencies

- Objects (Waves , variables, strings) may be linked dynamically to other objects by using the := operator.
- Show simple example with a string

Procedures

- The user can program Procedures, which may be stored as simple text files.
- Procedure files can be #included into each project.
- There are two kinds of procedures, macros and functions.
- Macros are of no interest to the normal user. They are not compiled, only interpreted at run-time, which makes them slow if loops are involved. Also, macros have some strange properties when waves are passed as parameters.
- Functions are the heart of IGOR programming.
  A function may or may nor return something (cf. void MyFunc(int p1, int p2, ...))
  Functions are compiled by IGOR. They are very much faster than macros, but slower than linked plain C functions ("XFuncs").
  Functions are called from the command line; they can return something into a variable or **INTO A WAVE**, like in:

```
function GetMyValue(k)
     variable k
     return k * x^2
end
```

- The call would then be something like
  `MyWave = GetMyValue()`
  which would return the function's value into each point of the wave, corresponding to ist x value.

Properties of Functions

- Passing parameters is similar to K&R C in that the parameters are declared in the function body like this:

```
Function MyFunc(v1, v2)
variable v1, v2
return v1 * x + v2 * x^2
end
```

- Wave can also be passed to a function, like this:

```
Function MyFunc(w)
wave w
return w[0] * x + w[2] * x^2
end
```

- IGOR provides the usual flow control mechanisms, like `if-then-else; switch case break continue; for(x;y;z); do-while`;
- Functions can be arbitrarily complex, like in the following example:
  ---> Niehaus Function, GaussConvolution

```
#include <Niehaus_Function>
#include <GaussConvolve>
```

Fitting against user-defined fit functions

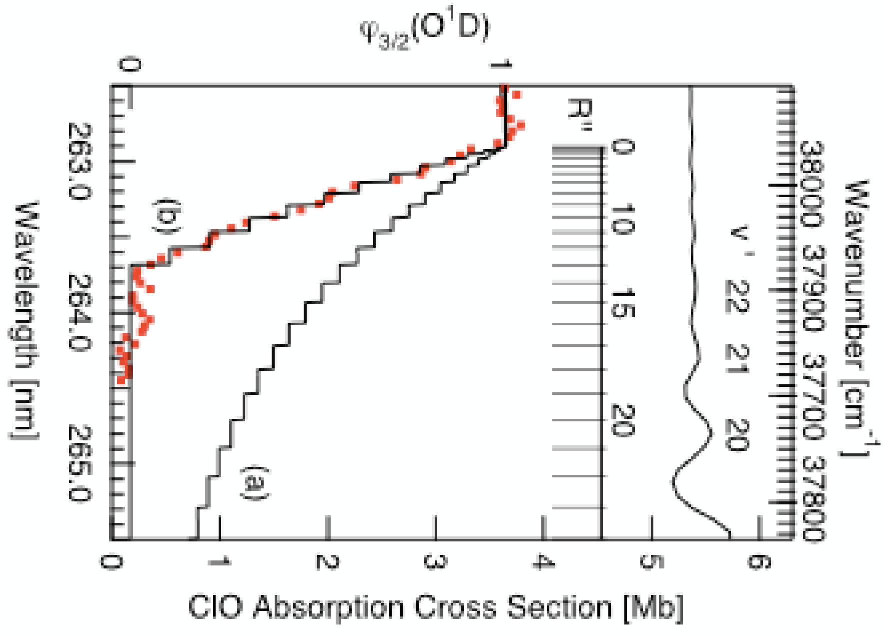
-
  ```
  Function MyFit(w,x): FitFunc       /* FitFunc Keyword */
  wave w
  variable x
  return w[0] * x^2 + w[1] * x + w[2]
  end
  ```
- Elaborate Fitting using user-defined functions: see MultiPeakFitting Panel
- Fitting to non-analytical functions: see Niehauss-Gauss Convolve Fit

Use of cursors

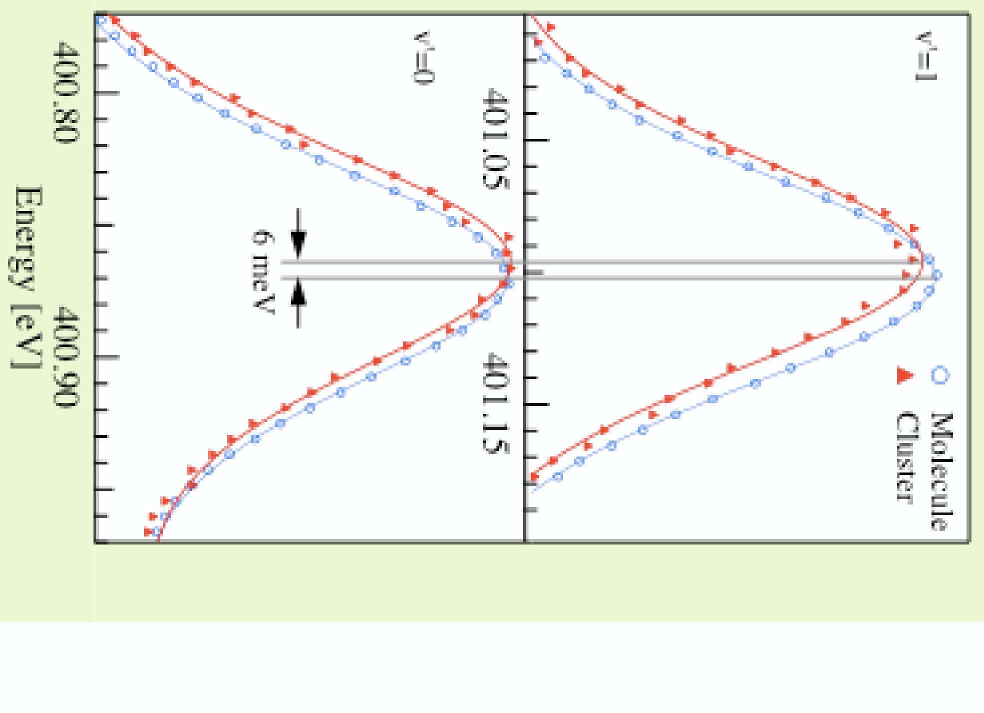
- Using Cursor functions: `xcsr(a), vcsr(a)`
- Returning values into variables:
  `variable xx = vcsr(a)`
- Scaling Waves by user-defined procedures ===> show "cal"

Final Remarks, in no particular order

- Most of the time, I use Igor to look at data, do simple math with them, and convert them into something I can publish (see next page)
- It comes with a very nice help system and good documentation.
- Advanced users make much use of the Xfunctions very much --- they write C programs, compile them using some additional libraries, and let them run within Igor.
- If I would have to fit something against a numerical solution of a partial higher-order differential equation, I would certainly not use Igor.
- Igor can also be used to import data directly from a device (such as a multi-coincidence card).
- Igor is available on different platforms (Windows, MacOS, MacOS-X) and is supposed to be ported to GNU-Linux in the near future.



Wavenumber [cm^{-1}]
38000
37900
37800
37700
v'
22
21
20
R''
0
10
15
20
(a)
(b)
$\varphi_{3/2}(O^1D)$
0
1
263.0
264.0
265.0
Wavelength [nm]
0
1
2
3
4
5
6
ClO Absorption Cross Section [Mb]



v'=1
v'=0
Molecule
Cluster
401.05
401.15
400.80
400.90
6 meV
Energy [eV]