Miscellaneous

Some codes written by Min Xu

I spent a lot of time in scientific calculation and began to accumulate some useful codes and tools of my own. They are mostly aimed to solve the problem occurred in my research, but also reflects my taste of a good software.

Version Control

The too many version of one project around is very confusing and version control is necessary. RCS and CVS are the de facto version control system in the free software community. CVS is powerful and complex and it is a overkill for a basically personal version control system. So I choose the much more simpler and coherent version control system PRCS designed primarily by Paul Hilfinger, with input and modifications by Luigi Semenzato and Josh MacDonald. A lot of information about CVS, RCS and related can be found in CVS Bubbles.

PRCS is strong in handling a package of files (a project) and straightforward to resolve conflicts and merge. But I also need a history of each file itself. So I made the following patch one prcs-1.2.14 which makes prcs asks for logs for individual files. The patch is produced by "`prcs -NP -r`*`rev1`* `-r`*`rev2`*" and you can apply this patch by "`patch -p 1 < prcs-1.2.14-patch`".

I have also written several scripts for prcs in perl. You can find prcs-check, prcs-info, prcs-history, prcs-rdiff and a prcs parsing library here.

Recently, to enable remote access of prcs repositories, I wrote python-prcs based on pdist distributed with Python which is able to do remote check, checkout, checkin, diff, log, clean, tar and other stuffs. It also comes with an interactive directory mirroring tool to help synchronize working files spanned over different machines.

Ratfor/C/C++

A code for image visualization is here. It needs the dislin library.

Tela

Tela is a small and nice tensor language. You can find my posts (post1, post2, post3) about tela. My addons for tela are here.

Python

Python is my favorite scripting language. I have accumulated a bunch of modules and scripts for my own convenience. You can use and distribute the codes freely but the original disclaimer should be kept.

- An interface to octave pyoct, the framework is rather general and I have also extended it to support scilab and tela. This code is inspired by pymat of Andrew Sterian. Since octave does not have an engine library, I wrote an engine library emulating the one coming with Matlab4 and the communication between python and octave is made by using named *fifo* when python forks and execls octave.

  The purpose of this code is to complement some shortcoming of python by octave, esp, the graphics at that moment.

- FFT is one of the most useful algorithms around. My image reconstruction research uses it a lot. Travis Oliphant provides a FFTW wrapper, but unfortunately, that wrapper doesnot utilize the most desirable feature of FFTW, i.e., multi-array concurrent Fourier transform and leads to excessive swapping when a big array is to be transformed and unusable in my calculation.

So I wrote my own wrapper of FFTW, the benchmark for a Pentium 200Mhz with 64M RAM running Debian Linux 2.2.10 are following:

```
        ************************************
        *************  Benchmark  ***********
        ************************************

        ===== 1-D comparison =====

        for an array 128x128, transform on axis=1
        FFT1: 0.3135 seconds
        FFT2: 0.2544 seconds
        FFT3: 2.5019 seconds
        FFT4: 0.2489 seconds

        for an array 128x128, transform on axis=0
        FFT1: 0.4289 seconds
        FFT2: 0.2904 seconds
        FFT3: 2.6386 seconds
        FFT4: 0.2796 seconds

        =====2-D comparison=====

        for an array 30x30x30, transform on axis=(0,1)
        FFT1: 1.2863 seconds
        FFT2: 0.6514 seconds
        FFT3: 0.3971 seconds
        FFT4: 0.6426 seconds

        for an array 30x30x30, transform on axis=(1,2)
        FFT1: 1.0924 seconds
        FFT2: 0.6129 seconds
        FFT3: 0.3649 seconds
        FFT4: 0.6095 seconds
```

The FFT1 is fft (1D) and fft2d (2D)from the original FFT.py coming with NumPy, FFT2 is FFTW.fft (1D) and FFTW.fftnd (2D), FFT3 is FFTW.cost (1D) and FFTW.rfftnd (2D), and FFT4 is the inplace version of FFT2.

It is clear that FFTW.fft, FFTW.fftnd are usually faster than the version from FFT for about 30% (1D) and takes about half time in 2D case. The provided real FFT routine reduced the time to only a third of original time in 2D case.

You can download the code here.

- Callback is useful in integration, optimization and so on. When an alien software is wrapped using SWIG, there is no good (?) way to enable the wrapper to call a python callback function. Based on drhfn by Doug Heisterkamp, a new module callback is written. Several examples including a global optimization code (GLOBAL), quanc8, zeroin, fmin of the famous FMM library, and SIGMA, another global optimization code.

  A callback function in the alien language is wrapped as:

```
   %typemap(python,in) U_fp f {
               $target = (U_fp)((callback_getCallbackFn)($source));
               if ($target==NULL) return NULL;
       }
       %init %{
               import_callback();
       %}
```

  and in python wrapper,

```
import callback, _fmm
xa = 0; xb = 1; tol = 1e-8
def fun(x):
    ...
    return val
cb = callback.callback(fun, 'd_dp')
print _fmm.fmin(xa, xb, cb, tol)
```

The original callback function of form 'd_dp', or double fun(double *), is transformed to call a python function fun which accepts one value and return its result.

You can download the callback package with examples here.

- A surface fitting wrapper of surfit from dierckx.