Perl and Inline Octave

(or IPC with an interactive application)

Andy Adler

School of Information Technology and Engineering
University of Ottawa
Rating:

*Viewer discretion advised*

Warning, this talk contains

- Conspicuous use of windows OS
- Occasional insulting comments about Perl and/or Perl people
Background

- I use Perl to manage files, and Octave to crunch numbers.
- Recently, I worked on a project that generated enormous data files, which needed to be processed and then analysed - a perfect task for my two favourite languages.
- Since I'd just heard a mighty cool talk on Inline (YAPC::NA 2001), it seemed clear to me that I needed to write Inline::Octave.
Sample Problem

- Is the temperature rising?

- Let's suppose we've decided that we don't trust those pundits, and we'd like to calculate for ourselves whether the earth is getting warmer.
Sample Problem

- However, for some crazy reason, we do trust random stuff published on the internet

- Type “daily temperature data” into google
This looks interesting
This looks interesting
Step 1: download file

```
$ wget ftp://ftp.engr.udayton.edu/jkissock/gsod/allsites.zip
   `allsites.zip'
Resolving ftp.engr.udayton.edu... 131.238.32.52
Connecting to ftp.engr.udayton.edu[131.238.32.52]:21... connected.
Logging in as anonymous ... Logged in!
  ==> SYST ... done.
  ==> PWD ... done.
  ==> TYPE I ... done.
  ==> CWD /jkissock/gsod ... done.
  ==> PORT ... done.
  ==> RETR allsites.zip ... done.
Length: 5,693,459 (unauthoritative)
8% [==>] 1506622 3.30K/s ETA 20:37
```

Step 2: look at contents

```
Archive: allsites.zip
Length Date Time Name
---------- ----- ---- ----------------
163954 05-17-04 16:06 ALALGIER.txt
163953 05-17-04 16:06 AGBUENOS.txt
163815 05-17-04 16:06 AKANCHOR.txt
163912 05-17-04 16:06 AKFAIRBA.txt
163930 05-17-04 16:06 AKJUNEAU.txt
162385 05-17-04 16:06 ABTIRANA.txt
163957 05-17-04 16:06 ALBIRMIN.txt
163957 05-17-04 16:06 ALHUNTSU.txt
163956 05-17-04 16:06 ALMOBILE.txt
163945 05-17-04 16:06 ALMONTGO.txt
163947 05-17-04 16:06 ARFTSMIT.txt
```
Step 3: file contents

```shell
$ unzip -p allsites.zip ALALGIER.txt
```

<table>
<thead>
<tr>
<th>Month</th>
<th>Day</th>
<th>Year</th>
<th>Temperature (°F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1995</td>
<td>64.2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1995</td>
<td>49.4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1995</td>
<td>48.8</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1995</td>
<td>46.4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1995</td>
<td>47.9</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>1995</td>
<td>48.7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>1995</td>
<td>48.9</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>1995</td>
<td>49.1</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>1995</td>
<td>49.0</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>1995</td>
<td>51.9</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>1995</td>
<td>51.7</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>1995</td>
<td>51.3</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>1995</td>
<td>47.0</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>1995</td>
<td>46.9</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>1995</td>
<td>47.5</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 4: parse in Perl

```perl
if (/\s* (\d+) \s+ (\d+) \s+ (\d+) \s+ ([\-\d.]+)/x) {
    next if $4 == -99;
    push @time, timegm( 0, 0, 12, $2, $1-1, $3);
    push @temp, ($4-32)/1.8;
}  # Code for no data
``` Convert to °C
What does data look like?

Many ways to plot data

- Load into spreadsheet (OOcalc, Excel)
- Octave (uses gnuplot)
- Perl
  - Lots of ways
  - Try GD::Graph
Plot data with Perl?

- CPAN: Get GD::Graph
- Requires: GD
- Requires: libgd, libjpeg, libpng, libX11 ...
- Requires: GD::Text

This road leads to “DLL hell”. Time to try a detour
Plot data via Octave via Gnuplot

Octave plot syntax

Xdata = [x1, x2, x3];
Ydata = [y1, y2, y3];
plot(Xdata, Ydata);

Usage:

unzip –p allsites.zip CNOTTOWA.txt | \ 
perl showplot.pl | \
octave -q

Note misspelling: I wonder how reliable these data are?
Plot code

```perl
#!/usr/bin/perl -w
use Time::Local;
my $y2k = timegm(0, 0, 0, 1, 0, 2000);
while (<>) {
    if (m/^\s+ (\d+) \s+ (\d+) \s+ (\d+) \s+ ([\-\d.]+)\s+/x) {
        next if $4 == -99;
        my $sec_y2k = timegm(0, 0, 12, $2, $1-1, $3) - $y2k;
        push @time, 2000 + $sec_y2k / 365.2422/24/60/60;
        push @temp, ($4-32)/1.8;
    }
}
print "Xdata=[",join("",@time),"];\n",
"Ydata=[",join("",@temp),"];
",
"plot(Xdata, Ydata);\n";
```
Temp (°C): Ottawa for 1999-2004

I went skiing on my birthday here
How to calculate trend

- **Problem:**
  Intra-year variations are much larger than year to year variations

- **Approach:**
  extract and remove signal component in phase with the year. Remainder is the trend

- **Notes:**
  - This can be done in perl – but I think it’s easier to use a mathematical language
  - This is not the correct way to calculate temperature trends. Please refer to the scientific literature
Analysis: remove year harmonics

Red: Temp (°C) vs. time (years) in Ottawa.
Green: Components in phase with the year removed
Blue: Best fit line
Perl code: process input / output

use Time::Local; my ($city, @time, @temp, %rates);
open F, "unzip -c $ARGV[0] |" or die ...
while (<F>) {
    if (/^\s+ (\d+) \s+ (\d+) \s+ (\d+) \s+ ([\-\d\.]+)/x) {
        push @time, ... push @temp, ...
    }
    if (/^\s+ inflating: \s+ (\w+) \s+ .txt/x) {
        process(@time, @temp) if $city;
        $city= $1; @time= (); @temp= ();
    }
} process(@time, @temp); #last one
close F;
printf "Rate: \%1.4f±\%1.4f (°C/year) for \%d cities\n",
calc_stats()->as_list;
Octave code: `calc_stats`

Perl

```perl
printf "Rate: %1.4f±%1.4f (°C/year) for %d cities\n",
calc_stats()->as_list;
```

Octave

```octave
function stat_data= calc_stats()
    global sites;
    n_cities= length(cities);
    stat_data= [  mean(cities),
                  std(cities)/sqrt(n_cities),
                  n_cities];
```
Octave code: `process`

Perl

```perl
process(@time, @temp);
```

Octave

```octave
function TperYr = process( time, temp );
    global sites=[]; static site_no=1;

    time_step = [0;diff(time)/2] + [diff(time)/2;0];
    harmonics = 2*pi*(1:3) / ( 365.2422*24*60*60 );
    year Osc = [ sin(time * harmonics), cos(time * harmonics) ] .* (time_step * ones(1,2*length(harmonics)));
    component = (temp' * year Osc) ./ sumsq( year Osc );
    temp_clean = temp - year Osc * component'; # harmonics

    fit = polyfit( time, temp_clean, 1); # fit to line
    cities( site_no++ ) = fit(1) * ( 365.2422*24*60*60 ); # deg/year
```
Linking Octave code to perl

```perl
use Time::Local;
open F, "unzip -c $ARGV[0] |" or die ...
while (<F>) {
    # stuff
}  # stuff
process(@time, @temp);
close F;
printf "Rate:%1.4f±%1.4f (°C/year) for %d cities\n",
calc_stats()->as_list;

use Inline Octave => q{
    # Octave code (previous slide)
};
```
Results

Yes, the average temperature is increasing but not everywhere.

Time: (using P4 2.4Ghz machine)

Cygwin / WinXP: 192.5 sec
Linux (Knoppix 3.3): 75.5 sec
A closer view of the glue

Inline

- Infrastructure to write perl in other languages
- Many Inline modules

inline

CPAN

Results 1 - 10 of 532 Found

1 2 3 4 5 6 Next >>

Page Size: 10 20 50 100

Inline::CPP
Write Perl subroutines and classes in C++.
Inline-CPP-0.25 - 12 Aug 2003 - Neil Watkiss

Inline::Java
Write Perl classes in Java.
Inline-Java-0.47 - 09 Mar 2004 - Patrick LeBoutillier

Inline
Write Perl subroutines in other programming languages.
Inline-0.44 - 04 Nov 2002 - Brian Ingerson
Inline::C example

use Inline C;
print "9 + 16 = ", add(9, 16), "\n";
___END___
___C___
int add(int x, int y) {
    return x + y;
}

- C code is extracted and compiled to a dynamic library (or shared object)
- At run-time, perl is linked to dll (or so)
Inline Java example

```perl
use Inline Java => q[ class Pod_alu {
    public Pod_alu() { }
    public int add(int i, int j){ return i + j; }
}
];
my $alu = new Pod_alu();
print $alu->add(9, 16) . "\n";
```

- Java code is extracted and compiled
- Two possibilities
  - Run code in JVM and interface via sockets
  - Link perl to JVM dll and make calls into it
Octave

- Octave is an interpreted language
  - Syntax is like Matlab
- Specializes in mathematical functions
- Why not use perl (ie. PDL)
  - Efficiency
  - Lots of code Octave/Matlab available
  - There’s more than one way to do it
Warning:

The following slide makes an unfair jab at Perl saints.

Viewer discretion is advised
Math languages take correctness seriously

For example, in the (excellent) Perl Cookbook (Christiansen & Torkington):

> These lines do not catch the IEEE notations of “Infinity” and “NaN”, but unless you are worried that the IEEE committee members will stop by your workplace and beat you over the head with copies of the relevant standards, you can probably forget about these strange numbers

> Serious math people just can’t make jokes about things like that
Interfacing with an interpreter

- An interpreter may be controlled by linking to its `stdin`, `stdout`, and `stderr`.
- Perl module: IPC::Open3
- Documentation says:
  - If you try to read from the child's stdout writer and their stderr writer, you'll have problems with blocking, ...
- I did try, and I did have problems.
Example.pm

- **Usage:**
  - `Example::interpret`: send code to interpreter, and capture `stdout` and `stderr`

$ perl -MExample -e 'print Example::interpret("1/2")'
ans = 0.50000

$ perl -MExample -e 'print Example::interpret("1/0")'
warning: division by zero (in octave code) at -e line 1
ans = Inf
Example.pm: setup

```perl
package Example;
use strict;
use Carp;
use IO::Handle;
my $Oerr = new IO::Handle;
use IPC::Open3;
open3( my $Oin, my $Oout, $Oerr, "octave -qH");
setpriority 0,0, (getpriority 0,0)+4; #lower priority slightly
use IO::Select;
my $select = IO::Select->new($Oerr, $Oout);
```

Stderr handle must be preinitialized
Example.pm: *interpret*

```perl
my $marker = "-9ABa8l_8Onq,zU9-"; # random string
my $marker_len = length($marker) + 1;

sub interpret {
  my $cmd = shift;
  print $Oin \n\n$cmd\ndisp('$marker');fflush(stdout);\n";
  my $input;
  while (1) {
    sysread $Oout, (my $line), 16384;
    $input.=$line;
    last if substr($input, -$marker_len, -1) eq $marker;
  }
  return substr($input, 0, -$marker_len);
}
```

Make interpreter give back marker when finished.
Example.pm: **handling stderr**

**Concept**
- *stdout* is an arbitrary stream of data
- *stderr* will consist of bursts of error data

**Implementation**
- When we detect stderr data, switch to \textit{process\_errors} until finished
Example.pm: error handler

```perl
sub process_errors {
    my $select = IO::Select->new( $Oerr );
    my $input;
    while ( $select->can_read(0.1) ) {
        sysread $Oerr, (my $line), 1024;
        last unless $line;
        $input.= $line;
    }
    croak "$input (in octave code)" if $input =~ /error:/;
    carp "$input (in octave code)" if $input;
}
```

Timeout for stderr data stream

Simple test to detect warnings and errors
Example.pm: handling stderr

```perl
sub interpret {
    my $cmd = shift;
    my $marker = "-9Ahv87uhBa8l_8Onq,zU9-"; # random string
    my $marker_len = length($marker) + 1;
    print $Oin \n\n$cmd\n$marker\n"\n";
    my $input;
    while (1) {
        for my $fh ( $select->can_read() ) {
            if ($fh eq $Oerr) {
                process_errors();
            } else {
                sysread $fh, (my $line), 16384;
                $input .= $line;
            }
        }
        last if substr( $input, -$marker_len, -1) eq $marker;
    }
    return substr($input, 0, -$marker_len);
}
```
Example.pm

Usage:

- `Example::interpret`: send code to interpreter, and capture `stdout` and `stderr`

```
$ perl -MExample -e 'print Example::interpret("1/2")'
ans = 0.50000

$ perl -MExample -e 'print Example::interpret("1/0")'
warning: division by zero (in octave code) at -e line 1
ans = Inf
```
Conclusion

- Inline is a great way to glue different languages together
- Inline::Octave is one option to do mathematical work in Perl
- Controlling an interpreted language is tricky. However perl allows this with IPC::Open3