Setting up your computing environment

1 Installation

1.1 Operating system

- Linux is the best operating system for our purpose: scientific computing.
- Mac is fine.
- Windows can be used, but you have to make sure that the most important programs/compilers work under your system: plotting program such as gnuplot, Python, Python-Scipy module, ANSI-ISO C++ compiler, BLAS&LAPACK, (fortran compiler).

1.2 Linux/Ubuntu

Currently most popular brand of linux is Ubuntu.

Very brief instructions for installing Ubuntu:

- Goto `http://www.ubuntu.com/getubuntu/download`
- Download the current version with long support (8.04 LTS). Save the ISO image.
- Burn ISO image to CD. Then instert CD into your computer and boot computer from CD.
- Follow the instructions to install Ubuntu.
- Some helpful instructions can be found at `https://help.ubuntu.com/community/GraphicalInstall`
- **Be very carefull:** If you want to keep your windows, you need to make a backup of your windows system first. You have to choose to ”shrink” the existing partition and use free space on your hard drive. Do not ”erase entire disc”.
  If you have experience, you can manually edit partition table: resize the windows partition to smaller size, add linux partition (one big ext3 partition and one small swap partition).

1.3 More installations:

Install the following software:

- Install most common development tools: gcc, gdb,...
  In Ubuntu, you can do the following:
  - goto: System/Administration/Synaptic Package Manager
  - Search for ”build-essential”, and ”Mark for installation”
  - ”Apply”
- Installed ”gnuplot” (using Synaptic Package Manager)
- Install ”emacs” (using Synaptic Package Manager)
- Install ”Intel math kernel library”, which includes blas and lapack (see instructions below)
- Install ”Intel Fortran compiler” (see instructions below). Alternatively, you can also install gfortran using Synamptic Manager.
- Check if Python is installed (name of the package ”python” and ”python-dev” in Synaptic Package Manager)

- Install ”python-numpy” using Synaptic Package Manager (numeric python)
- Install ”python-scipy” using Synaptic Package Manager (scientific python)
- Install ”python-matplotlib” using Synaptic Package Manager (for plotting).
- Install ”ipython” using Synaptic Package Manager (nice frontend).

1.4 Why SciPy?

Because it comes with great packages (reasonable well tested). Most of them written in Fortran and wrapped into Python. Available subpackages:

- fft, fft2, fftn – discrete Fourier transform
- fftpack
- integrate - Numeric Integration
- interpolate - Interpolation, including multidimensional splines!
- linalg - All routines of linear algebra from LAPACK and BLAS
- optimize - Great minimization package

- sparse - suport for sparse matrices
- special - special functions like Bessel, Error and Air functions,...
- stats - statistics
- io - fast input, output

1.5 Instalation of Intel Fortran Compiler:

To install intel fortran compiler, do the following:

- google ”Intel fortran compiler”
- On intel webpage select ”Free Non-Commercial Donwload”
- Under ”Compilers” select ”Intel Fortran Compiler Professional Edition for Linux” and download. You will get `l_cprof_p_11.0.069_ia32.tgz`
- remember serial number which will be needed at install.
- type:
  - `tar xzvf l_cprof_p_11.0.069_ia32.tgz`
  - `cd l_cprof_p_11.0.069_ia32`
  - `./install.sh`

1.6 Instalation of Math Kernel Library

- google ”Intel fortran compiler”
- On intel webpage select ”Free Non-Commercial Donwload”/”Free evaluation”
- Under Performance Libraries select ”Intel Math Kernel Library (Intel MKL) for Linux” and download. (It is best to download `"l_mkl_p_10.0.011.tgz"`version, which is compatible with current scipy)
- remember serial number which will be needed at install.
- type:
  - `tar xzvf l_mkl_p_10.0.011.tgz`
  - `cd l_mkl_p_10.0.011`
  - `./install.sh`

- To test math library do
  - `cd /opt/intel/mkl/10.0.011/tests/blas`
  - `sudo -i`
  - `make lib32 name=zblat1`
  - Check if the tests passed

    `_results/intel_parallel_32_lib/*.res`
  - Good luck!

Introduction to computing in high level languages

We will use the following programming languages:

- Fotran

  We will learn how to use existing fortran subroutine and packages in modern languages.

  ↑ A lot of scientfic code written in fortran ⟶ we need to use it.

  ↓ For todays standards, it is obsolete.

  – Developed by IBM in the 1950s (John W. Backus 1953).

  ↓ Many releases (Fortran, Fortran II, Fortran III, Fortran 66, Fortran 77, Fortran 90, Fortran 95, Fortran 2003, Fortran 2008).

  ↓ The language keeps changing substatially (Problems with maintainability).

  ↓ Many vedures, but no standard compiler: Intel fortran, PGI fortran,...

- C++

  Nowadays, most common language for numerical algorithms (when speed is crucial). (NRC do not give C++ book for free).

  ↑ Execution nowadays as fast as fortran (or C). It was developed for system programming (unix kernel): very fast, general purpose, great for very large scale problems.

  ↑ Very powerful: object oriented, supports templates, exceptions (C with classes).

  – Middle-level language: Not for writting front-end or web-page.

  – Developed by Bjarne Stroustrup in 1979 at Bell Labs.

  ↑ Mature and solid language: ANSI-ISO standard implemented some years ago. First C++ standard ratified in 1998, few corrections were made in 2003 due to ”defect reports”.

  ↑ Very popular implementation of ANSI standard under gnu licence: **gcc**.

  ↓ More complex than most of other modern languages. Very hard to fully master it.

- Perl
  - $\uparrow$ Very popular scripting language (mainly due to web scripting ”cgi-scripts”).
  - $\uparrow$ Open source, developed by Larry Wall from NASA, in 1987.
  - $\uparrow$ Comprehensive Perl Archive Network (CPAN) repository contains more than 13,500 modules developed by more than 6,500 authors.
  - $\downarrow$ Experst converted from Perl to Python: ”... He replied that Perl was fine up to a hundred lines or so, but beyond that he sort of hit a wall, and it got hard to manage the complexity. He now thought Python was a much better language for readability, maintainability, and modifiability.”
  - $\downarrow$ Loosing agains python lately.

- Python

  Recently it is gaining popularity in the world of scripting.

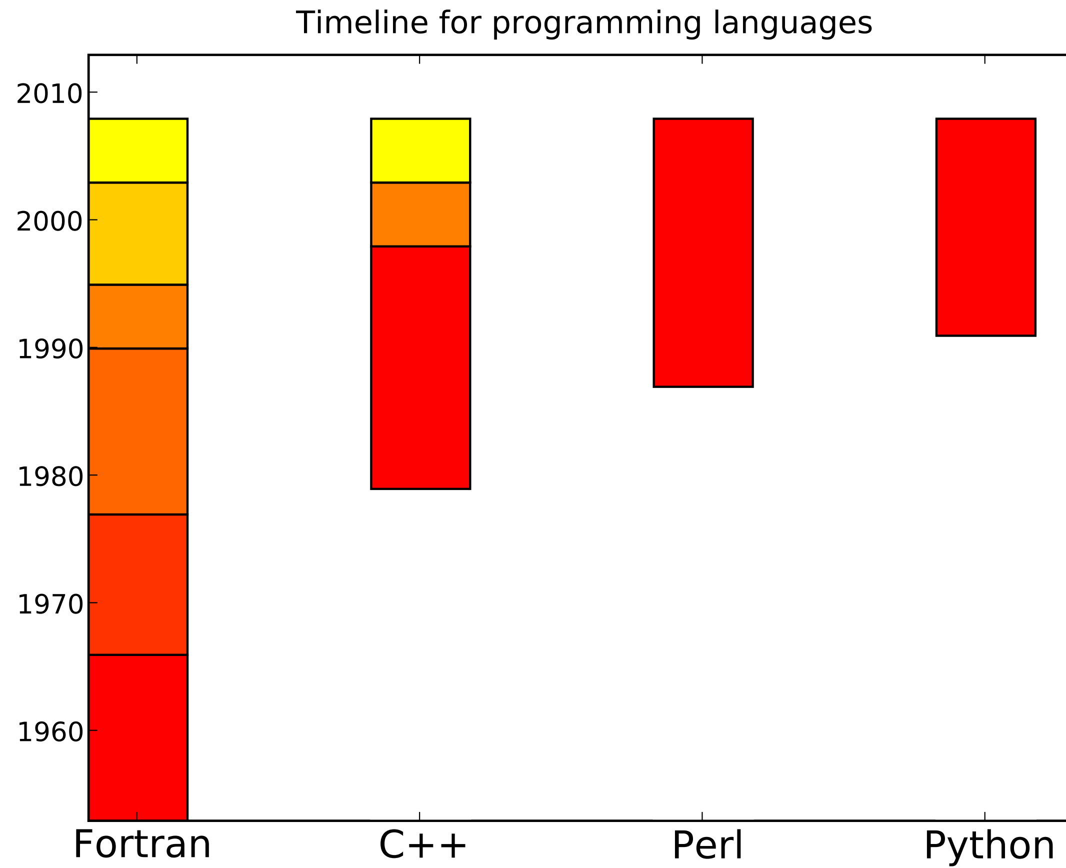
  ↑ Easier to mantain and read than perl code.

  ↑ Open source.

  ↑ Excellent tools for web development (cherryPy, Cheetah).

  ↑ Relatively young: first released by Guido van Rossum in 1991.

  ↓ Very alive: version 3.0 released December 3, 2008. Unfortunately many modifications in 3.0!

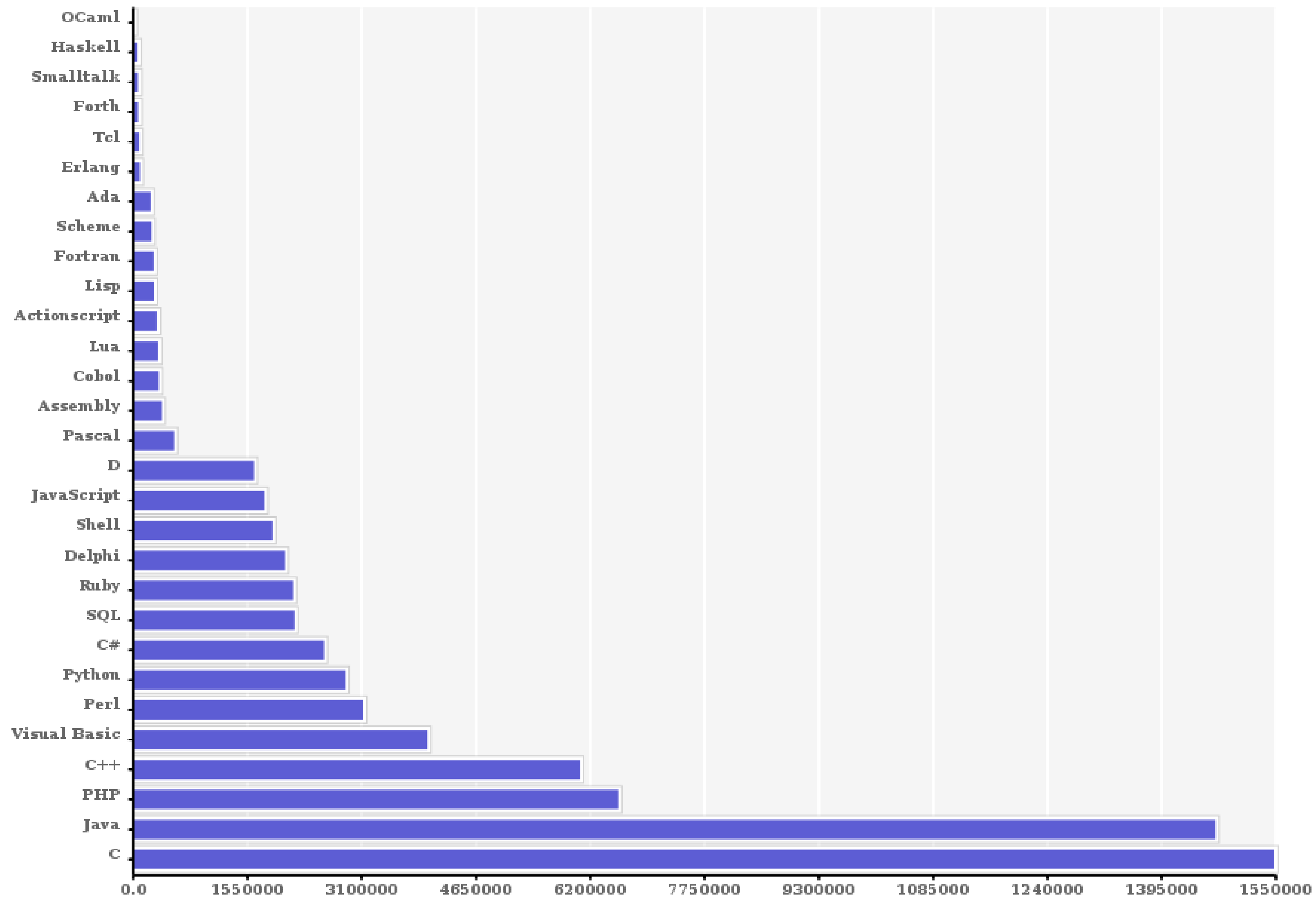


2 Which language is most popular (in 2008)

Research conducted at `http://www.langpop.com/`

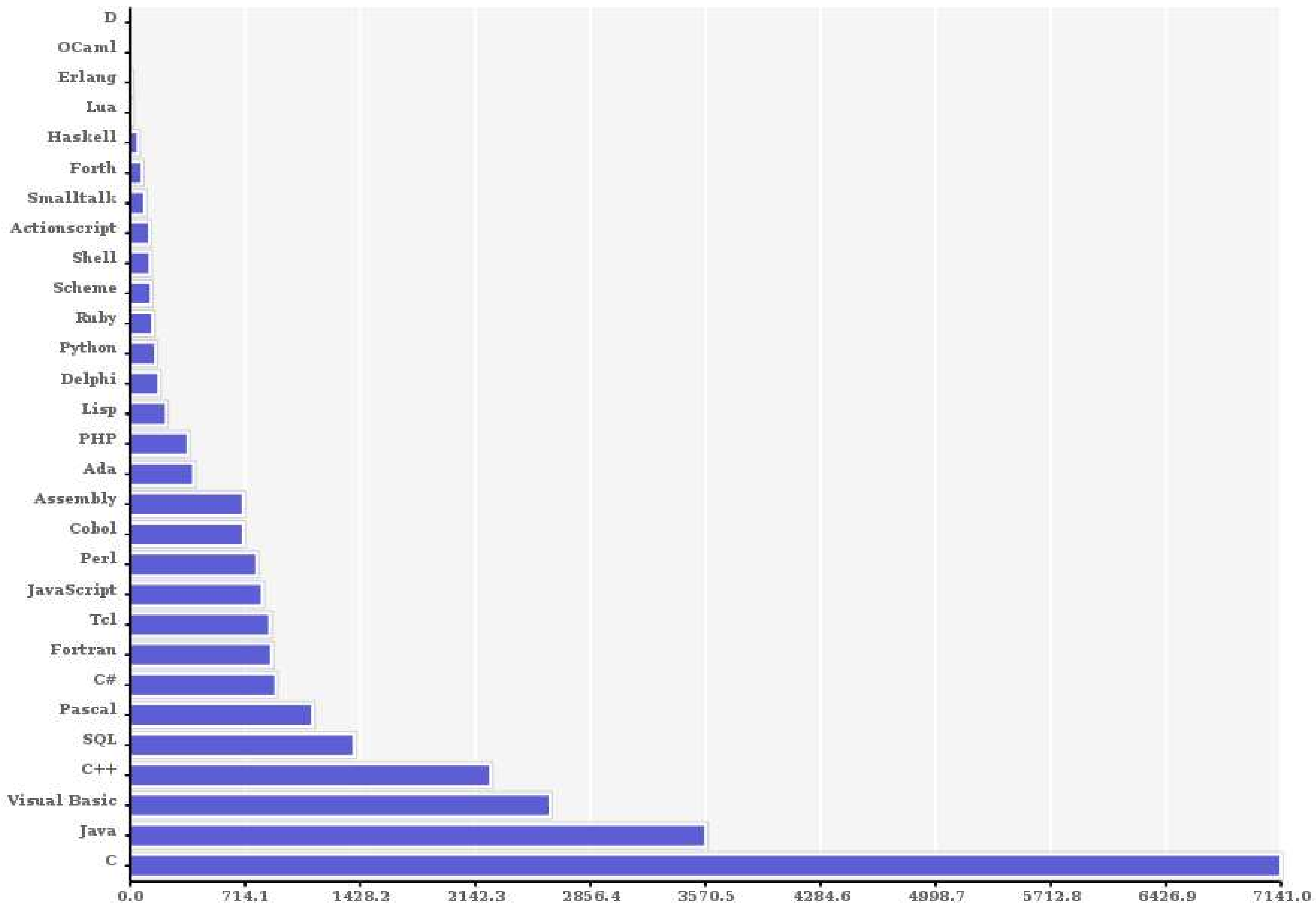
2.1 Yahoo.com

"language programming"



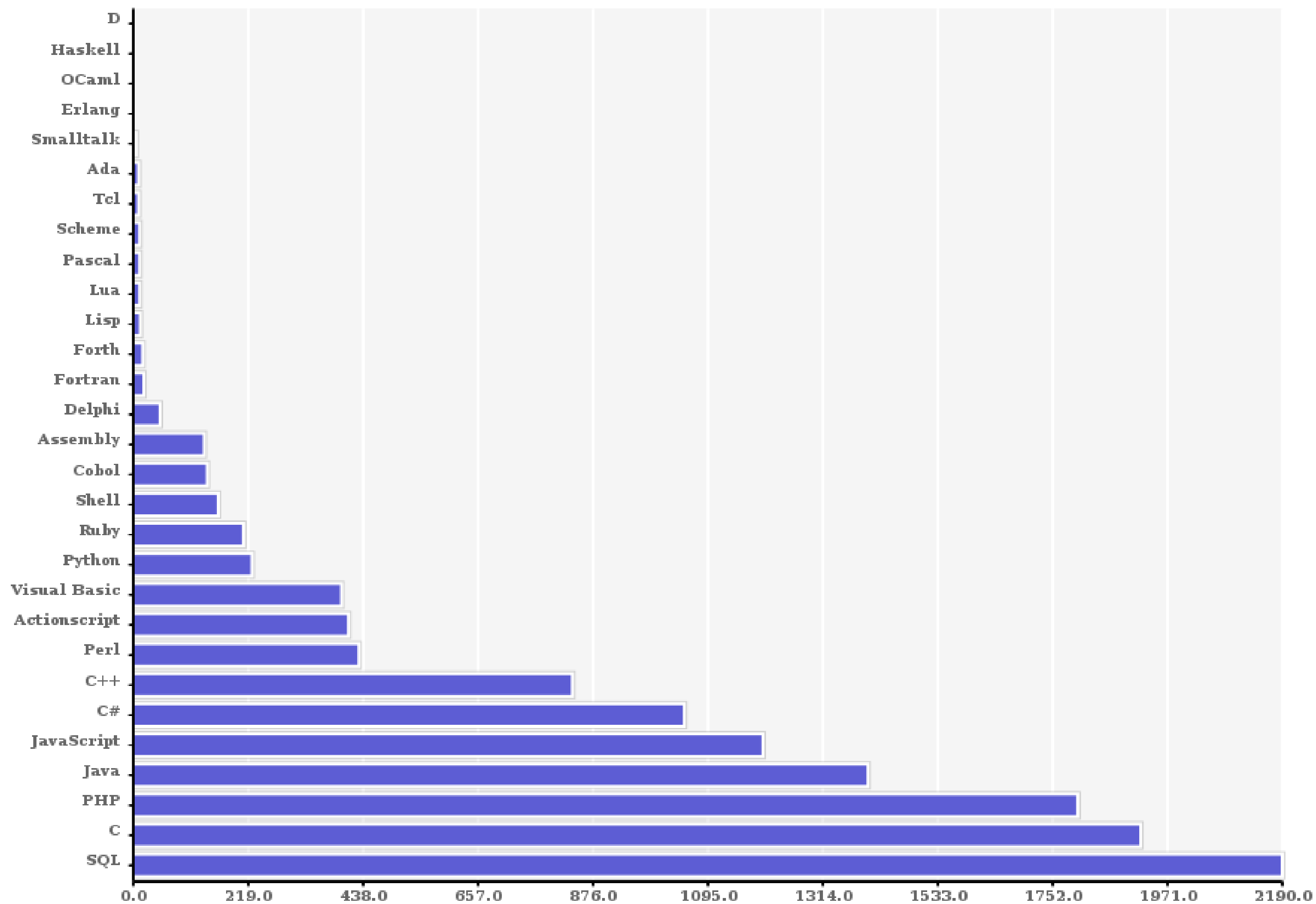
2.2 Amazon.com

language programming in the books index.

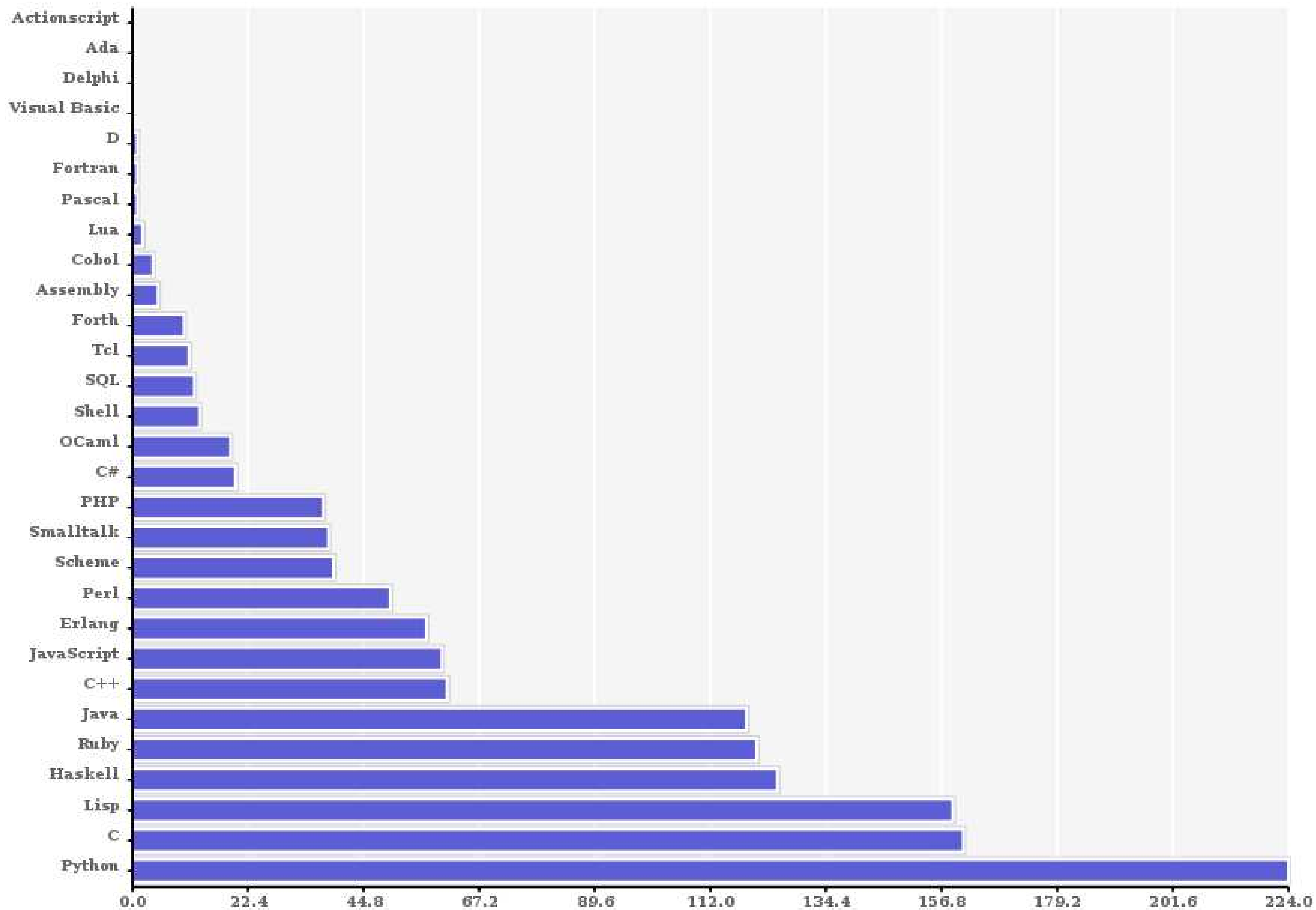


2.3 Craigs List

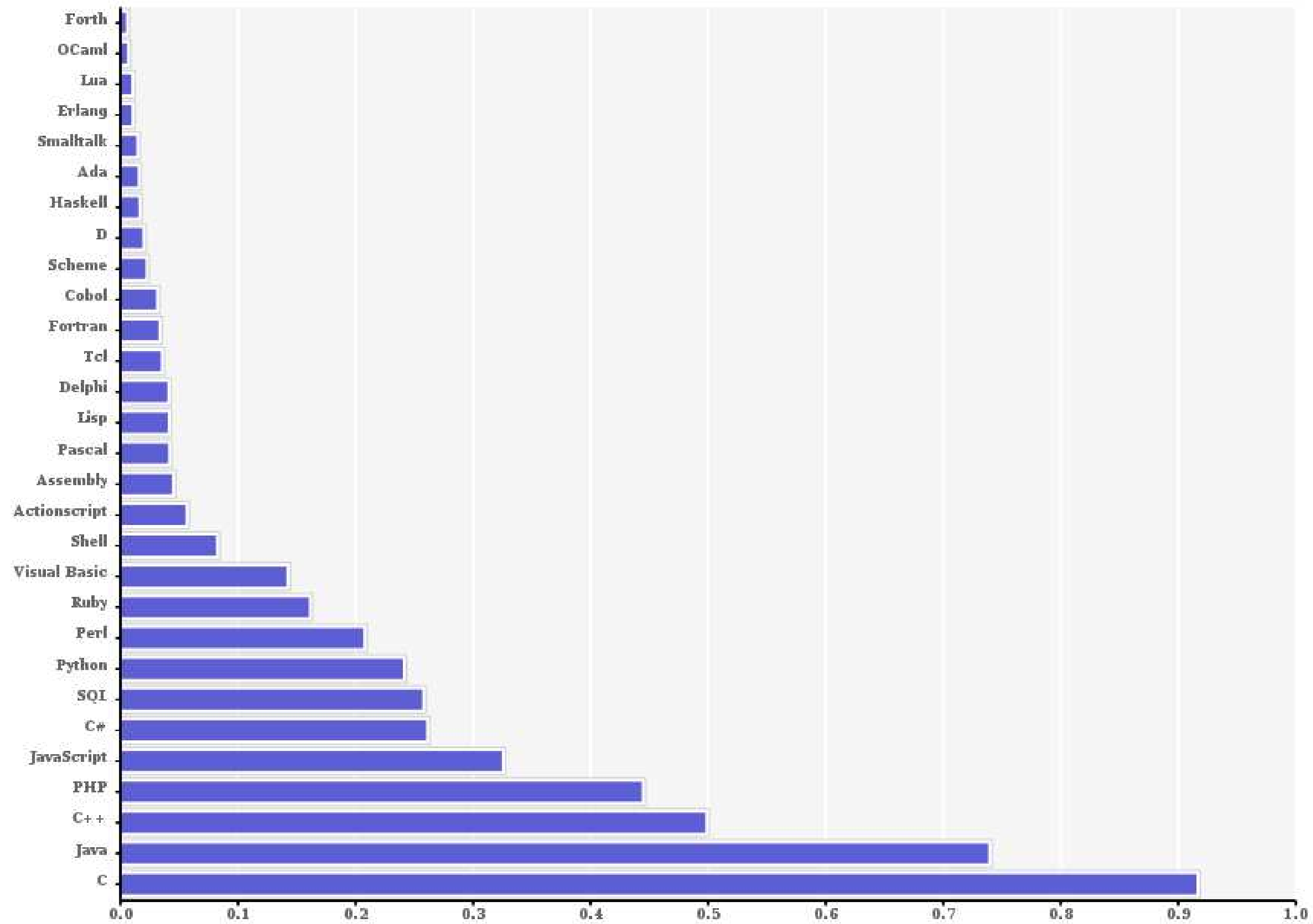
language programmer -”job wanted”



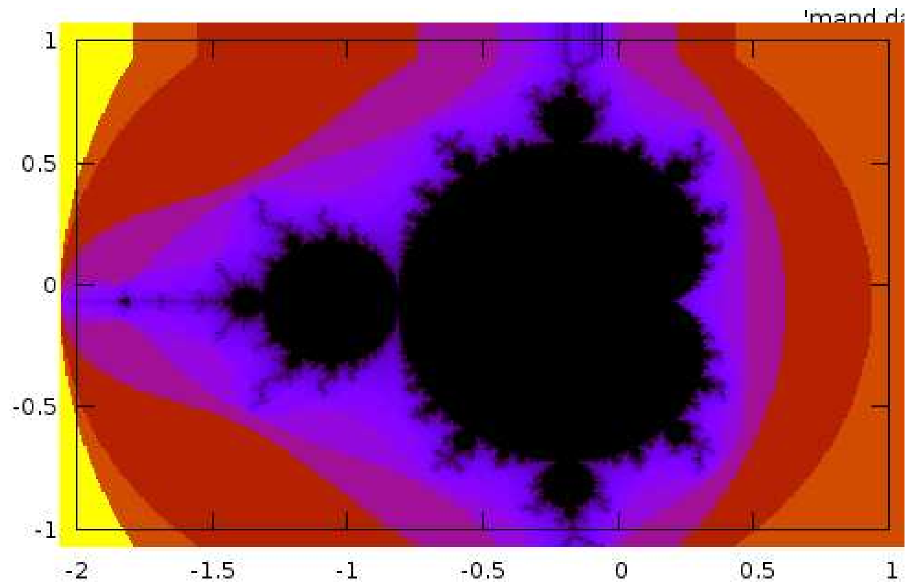
2.4 Discussion on reddit.com



2.5 Total



3 Comparison of languages by generating Mandelbrot set:



Wikipedia: The Mandelbrot set M is defined by a family of complex quadratic polynomials $f(z) = z^2 + z_0$ where z_0 is a complex parameter. For each z_0, one considers the behaviour of the sequence $(0, f(0), f(f(0)), f(f(f(0))), \cdots)$ obtained by iterating $f(z)$ starting at $z = 0$, which either escapes to infinity or stays within a disk of some finite radius. The Mandelbrot set is defined as the set of all points z_0 such that the above sequence does not escape to infinity.

Implementation in Fortran:

```
REAL*8 FUNCTION Mandelb(z0, max_iterations)
  IMPLICIT NONE ! Don't use any implicit names of variables!
  ! Function arguments
  COMPLEX*16 :: z0    ! Need to declare all variables first
  INTEGER    :: max_iterations
  ! Local variables
  INTEGER    :: i
  COMPLEX*16 :: z
  z=0                  ! Implementation after declarations
  DO i=1,max_iterations
     IF (abs(z)>2.) THEN
        Mandelb = i           ! result is number of iterations
        RETURN
     ENDIF
     z = z**2 + z0           ! f(z) = z**2+z0 -> z
  ENDDO
  Mandelb = max_iterations*1e3 ! choose some very large number
  RETURN
END FUNCTION Mandelb
```

```fortran
PROGRAM mand        ! Main part of the program
  IMPLICIT NONE    ! Don't use any implicit names of variables!
  REAL*8       :: Mandelb
  INTEGER      :: Nx, Ny, i, j
  REAL*8       :: x, y
  COMPLEX*16:: z0
  INTEGER      :: max_iterations
  Nx = 400                     ! the mesh in complex plane
  Ny = 400
  max_iterations = 1000

  DO i=1,Nx
     DO j=1,Ny
        x = -2 + 3.*(i-1)/(Nx-1.)
        y = -1 + 2.*(j-1)/(Ny-1.)
        z0 = dcmplx(x,y)
        print *, x, y, 1/Mandelb(z0,max_iterations) ! call to the function
     ENDDO
  ENDDO
END PROGRAM mand
```

Implementation in C++:

```
double Mandelb(complex<double>& z0, int max_iterations=1000) // default value
{
  complex<double> z=0; // initial value of z
  for (int i=0; i<max_iterations; i++){
    if (norm(z)>4.) return i;
    z = z*z + z0;      // if |z|>2 the point is not part of mandelbrot set
  }
  return max_iterations*1e3;
}
```

```
int main()    // the main program should always return int: 0 - if no error occured,
              //                                      non-zero  if an error occured.
{
  int Nx=400;
  int Ny=400;
  for (int i=0; i<Nx; i++){
    for (int j=0; j<Ny; j++){
      double x = -2. + 3*i/(Nx-1.);
      double y = -1. + 2*j/(Ny-1.);
      complex<double> z0(x, y);
      cout<<x<<" "<<y<<" "<<1/Mandelb(z0)<<endl;
      cout<<x<<" "<<y<<" "<<1/Mandelb_optimized(x,y)<<endl;
    }
  }
  return 0; // no error
```

Both C++ and fortran code needs to be compiled (compilation allows optimization):

```
C++ = g++
F90 = ifort

all : mandc mandf

mandc : mandelb.cc
	$(C++) -O3 -o mandc mandelb.cc

mandf : mandelb.f90
	$(F90) -O3 -o mandf mandelb.f90

clean :
	rm -f mandf mandc
```

Perl code does not need to be compiled. It is interpreter.

The call to subroutine is skipped due to optimization:

```
#!/usr/bin/perl
use Math::Complex;

$Nx = 100;          # all variables need $ or @.
$Ny = 100;          # no need to declare variables, only initialize them.
$max_steps =  30; # no typechecking


for ($i=0; $i<$Nx; $i++){
    for ($j=0; $j<$Ny; $j++){
        $x = -2. + 3.*$i/($Nx-1);
        $y = -1. + 2.*$j/($Ny-1);
        $z0 = $x + $y*i;
        $z=0;
        for ($itr=0; $itr<$max_steps; $itr++){
            if (abs($z)>2.) {last;}
            $z = $z*$z + $z0
        }
        print "$x $y ", 1/$itr, "\n";
    }
}
```

Python is interpreter as well.

The call to subroutine is again skipped for optimization:

```
from pylab import *
from scipy import *

# Python example for mandelbroat
Nx = 400          # No declerations necessary
Ny = 400          # Just start using variables
max_steps=100     # No typechecking

# We will store values, rather than print
# We will display plot below using Python matplotlib.
data = zeros((Nx,Ny), dtype=float) # creates numpy array. Requires numpy package!

for i in range(Nx):         # range always starts at 0....<Nx
    for j in range(Ny):     # 0...<Ny
        x = -2. + 3.*i/(Nx-1.)
        y = -1. + 2.*j/(Ny-1.)
        z0 = complex(x,y)
        z = 0
        for itr in range(max_steps):
            if (abs(z)>2.) : break
            z = z*z + z0
        data[j,i] = 1./itr


# Using python's pylab, we display pixels to the screen!
# Requires matplotlib package installed.
imshow(data, interpolation='bilinear', cmap=cm.hot, origin='lower', extent=[-2,1,-1,1], aspect=1.)
colorbar()
show()
```

Testing examples

Type: `make`

the following compilation is executed:

```
g++ -O3 -o mandc mandc.cc
ifort -O3 -o mandf mandf.f90
```

Execute and check the time:

```
time mandf > mand.dat                   | user    0m2.652s
time mandc > mand.dat                   | user    0m0.816s
time perl mandp.pl > mand.dat           | user    69m14.017s
time python mandp.py > mand.dat         | user    1m02.124s
```

- Both interpreters are substantially slower than comilers.
- Python is substantially faster than perl (surprise).
- C++ code was slightly optimized for performance and bits fortran90. In case of good optimization in both languages, they should be comparable.

The codes produce three column output $x, y, color$ and need a plotting program to display results. In `gnuplot` the following command plots the output:

```
set view map
splot 'mand.dat' with p ps 3 pt 5 palette
```

or call the script by

```
gnuplot gnu.sh
```

4 Plotting

Many plotting programs are available and everybody is free to chose his favorite. Some of most common programs and packages

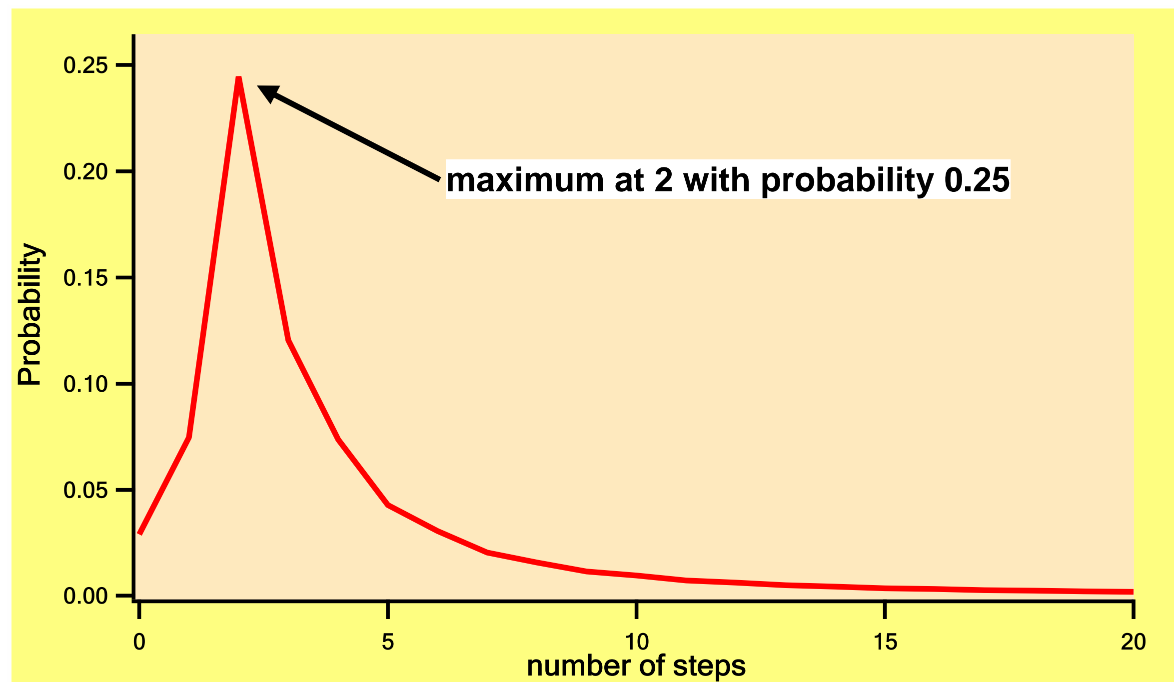
- **gnuplot** `http://www.gnuplot.info/`
  - Freely available
  - Works under linux and cigwin
  - Plots 2D and 3D
  - Quite powerful but somewhat clumsy to make publication quality figures
  - It is very fast and probably the best for ”preview”.
- **xmgrace** `http://plasma-gate.weizmann.ac.il/Grace/`
  - Freely available
  - Works under linux and cigwin
  - plots 2D only
  - Quite powerful for *publication quality* plots.
  - Somewhat old-fashioned with very small actively developing community.

- **Origin**, http://www.microcal.com/index.php?id=380and SigmaPlot, http://www.systat.com/products/sigmaplot/
  - Not free, expensive!
  - One of the most popular products for scientific and engineering plotting
  - works under windows only
  - Quite powerful for *publication quality* plots.
- **Igor**, http://www.wavemetrics.com/
  - Not free, but we **got licence for this course**.
  - Quite popular products for scientific plotting
  - works under windows and macintosh
  - Quite powerful for *publication quality* plots.
- **Excel** from Microsoft
  - Not free.
  - Not very popular for scientific and engineering plotting.
  - works under windows only

- **Mathematica** and **Matlab**
  - Not free
  - Not very popular for scientific and engineering plotting.
- **python-matplotlib**
  - Free, part of Python library
  - requires some coding but can produce publication quality figures
  - many examples available at
    `http://matplotlib.sourceforge.net/gallery.html`

I recommend to download and install at least one of the programs. One can download ”Igor” from the course website `http://www.physics.rutgers.edu/~haule/509/`at the bottom of the page. You need the following information (username:cmp, passwd:cmp123).

Plot produced by Igor:



5 Homework:

- Set up your environment! C++, Python, fortran, BLAS&LAPACK.
- If you are familiar with coding, write your own mandelbrot version of the code.
  If not, download Mandelbrot code written in fortran, C++, perl and python. Execute them and check that they work properly.
- Test your gnuplot by plotting mandelbrot set from generated file `mand.dat`.
- Execute the C++ executavble `mandh` to get distribution of iterations (number of steps required to escape Mandelbrot set) and plot 2D graph using your favorite plotting program. Produce a *nice* plot of the distribution.

Some free books on Python:

- How to Think Like a Computer Scientist: Learning with Python `http://www.ibiblio.org/obp/thinkCSpy/`
- Dive Into Python: `http://www.diveintopython.org/`
- Python for beginners `http://wiki.python.org/moin/BeginnersGuide`
- Gread documentation at `http://docs.python.org/`