

How do develop a numerical project

Morten Hjorth-Jensen, National Superconducting Cyclotron
Laboratory and Department of Physics and Astronomy,
Michigan State University, East Lansing, MI 48824, USA &
Department of Physics, University of Oslo, Oslo, Norway

May 18-22 2015

Some basic ingredients for a successful numerical project

In when building up a numerical project there are several elements you should think of

- 1 How to structure a code in terms of functions
- 2 How to make a module
- 3 How to read input data flexibly from the command line
- 4 How to create graphical/web user interfaces
- 5 How to write unit tests (test functions or doctests)
- 6 How to refactor code in terms of classes (instead of functions only)
- 7 How to conduct and automate large-scale numerical experiments
- 8 How to write scientific reports in various formats (L^AT_EX, HTML)

Additional benefits: A structure approach to solving problems

The conventions and techniques outlined here will save you a lot of time when you incrementally extend software over time from simpler to more complicated problems. In particular, you will benefit from many good habits:

- 1 New code is added in a modular fashion to a library (modules)
- 2 Programs are run through convenient user interfaces
- 3 It takes one quick command to let all your code undergo heavy testing
- 4 Tedious manual work with running programs is automated,
- 5 Your scientific investigations are reproducible, scientific reports with top quality typesetting are produced both for paper and electronic devices.

Analysis of project, Configuration Interaction theory

```
from numpy import *
from sympy import *
from matplotlib.pyplot import *

g_array = linspace(-1, 1, 1001)
e1_array = []
e2_array = []

for g in g_array:
    H1 = matrix([[2-g, -g/2., -g/2., -g/2., -g/2., 0],
                [-g/2., 4-g, -g/2., -g/2., 0, -g/2.],
                [-g/2., -g/2., 6-g, 0, -g/2., -g/2.],
                [-g/2., 0, 6-g, -g/2., -g/2.],
                [-g/2., 0, -g/2., -g/2., 8-g, g/2.],
                [0, -g/2., -g/2., -g/2., -g/2., 10-g]])

    H2 = matrix([[2-g, -g/2., -g/2., -g/2., -g/2.],
                [-g/2., 4-g, -g/2., -g/2., 0],
                [-g/2., -g/2., 6-g, 0, -g/2.],
                [-g/2., -g/2., 0, 6-g, -g/2.],
                [-g/2., 0, -g/2., -g/2., 8-g]])

    u1, v1 = linalg.eig(H1)
    u2, v2 = linalg.eig(H2)
```

Analysis of project, Many-body perturbation theory

```
from sympy import *
from pylab import *

below_fermi = (0,1,2,3)
above_fermi = (4,5,6,7)

states = [(1,1),(1,-1),(2,1),(2,-1),(3,1),(3,-1),(4,1),(4,-1)]
N = 8
g = Symbol('g')

def h0(p,q):
    if p==q:
        p1, s1 = states[p]
        return (p1 - 1)
    else:
        return 0

def f(p,q):
    if p==q:
        return 0

    s = h0(p,q)
    for i in below_fermi:
        s += assym(p,i,q,i)
    return s
```

Analysis of project, Many-body perturbation theory

```
from sympy import *
from pylab import *

below_fermi = (0,1,2,3)
above_fermi = (4,5,6,7)

states = [(1,1),(1,-1),(2,1),(2,-1),(3,1),(3,-1),(4,1),(4,-1)]
N = 8
g = Symbol('g')

def h0(p,q):
    if p==q:
        p1, s1 = states[p]
        return (p1 - 1)
    else:
        return 0

def f(p,q):
    if p==q:
        return 0

    s = h0(p,q)
    for i in below_fermi:
        s += assym(p,i,q,i)
    return s
```