Development of a real space grid based PAW-DFT Python code

CENTER FOR ATOMIC-SCALE MATERIALS PHYSICS
DEPARTMENT OF PHYSICS
TECHNICAL UNIVERSITY OF DENMARK

Jens Jørgen Mortensen,
Lars B. Hansen,
Karsten W. Jacobsen,
Jens K. Nørskov

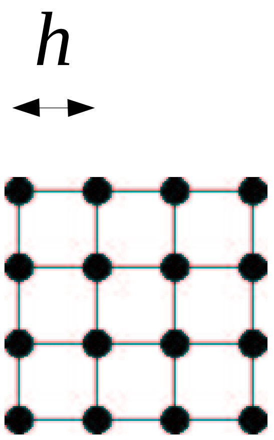
Outline of talk

- Motivation
- Algorithm
- Projector Augmented Wave method
- Double grid technique
- Results
- Python
- Conclusions

Motivation for using real space grids:

- Simple: Only one parameter (grid spacing)
- Flexible boundary conditions
  1) Cluster
  2) Wire
  3) Surface
  4) Bulk
- First step towards order-N
- No Fourier transforms

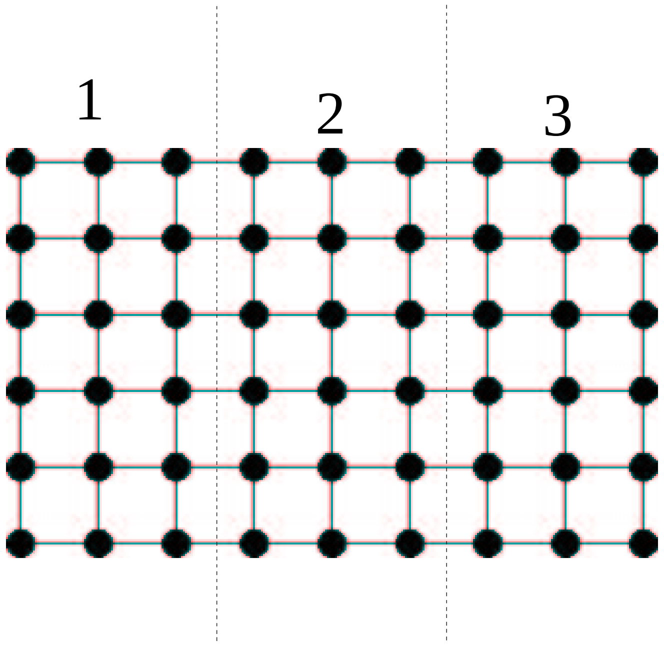
Grids



Finite Difference operators:

$$h^2\nabla^2 = \begin{array}{ccc} & \textcircled{1} & \\ \textcircled{1} & \textcircled{-4} & \textcircled{1} \\ & \textcircled{1} & \end{array}$$

Parallelization:



Algorithm:

G. Kresse *et al.*, Phys. Rev. B 54, 11169-11186 (1996)

LCAO

Update density

Pulay mixing:

$$\tilde{n}_{i+1} = \sum_i \alpha_i \tilde{n}_i^{out}$$

minimize: $\sum_i \alpha_i (\tilde{n}_i^{in} - \tilde{n}_i^{out})$

Symmetrize

Calculate potentials

Multigrid Poisson solver
using Mehrstellen operator

Apply Hamiltonian

Diagonalize subspace

Find occupation numbers

Calculate residuals

$$R = \hat{H}\tilde{\Psi} - \epsilon\tilde{\Psi}$$

Update wavefunctions

$$\tilde{\Psi}' = \tilde{\Psi} + \lambda \hat{P} R$$

minimize: $R' = \hat{H}\tilde{\Psi}' - \epsilon\tilde{\Psi}'$

$$\tilde{\Psi} \leftarrow \tilde{\Psi}' + \lambda \hat{P} R'$$

Orthogonalize

Preconditioning $(\hat{P})$: one V-cycle with $\hat{H} = -\frac{1}{2}\nabla^2$

Projector Augmented-Wave method

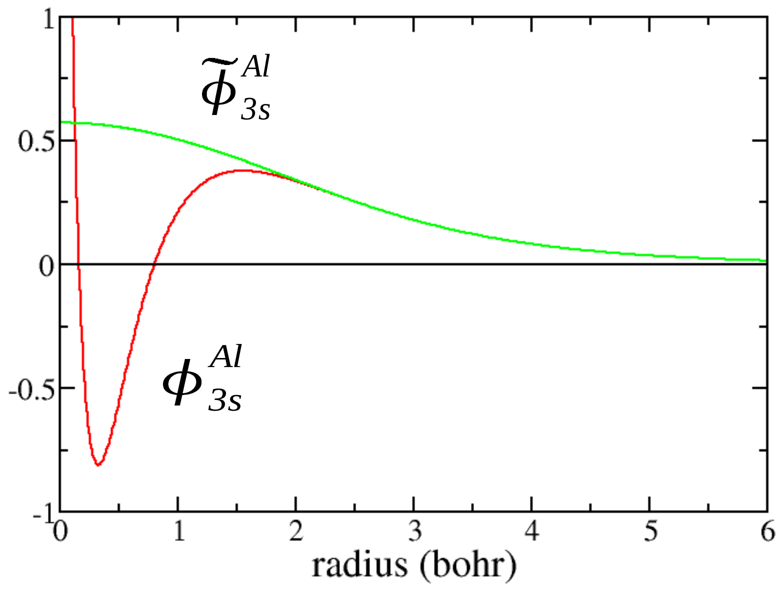
P. E. Blöchl, Phys. Rev. B **50**, 17953 (1994)
P. E. Blöchl, C. J. Först and J. Schimpl, Bull. Mater. Sci, 26, 33 (2003).

Why PAW?

1) Exact all-electron formalism
2) Soft wavefunctions (like USPP)

Approximations:

1) Finite number of projectors
2) Truncated angular momentum expansions
3) Frozen core



Projectors

$$\langle \widetilde{p}^a_i | \widetilde{\phi}^a_i \rangle = \delta_{ij}$$

$$\Psi = \sum_a \sum_{nlm} (\phi^a_{nlm} - \widetilde{\phi}^a_{nlm}) \langle \widetilde{p}^a_{nlm} | \widetilde{\Psi} \rangle + \widetilde{\Psi}$$

All-electron

Soft

$$\widetilde{\rho} = \sum \widetilde{\Psi}^2 + \sum_a \sum_{lm} Q^a_{lm} \widetilde{g}^a_{lm} (\boldsymbol{r} - \boldsymbol{R}^a)$$

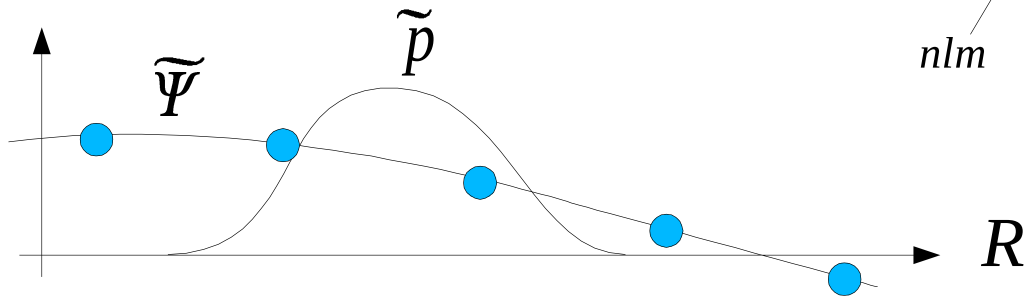
Double grid technique:

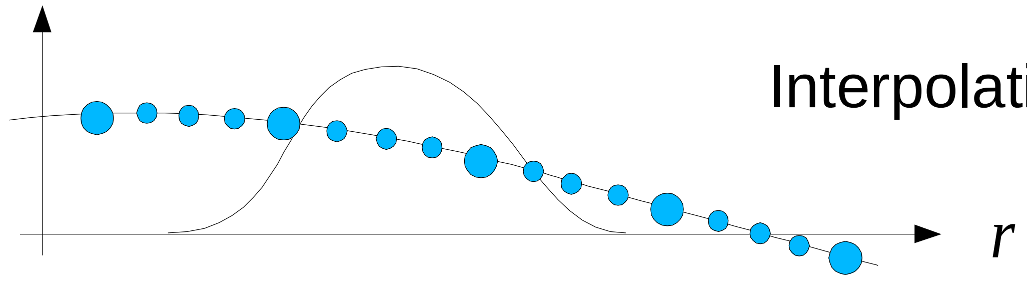
Projections:

Projection: $\langle \tilde{\Psi}_n | \tilde{p}_i^a \rangle$

nlm

$$\boxed{\langle \tilde{\Psi} | \tilde{p} \rangle \simeq \sum_R \tilde{\Psi}(R)\, \tilde{p}(R)\, \Delta V}$$





Interpolation: $\tilde{\Psi}(r) = \sum_R I_{rR} \tilde{\Psi}(R)$

$$\langle \tilde{\Psi} | \tilde{p} \rangle = \sum_r \tilde{\Psi}(r)\, \tilde{p}(r)\, \Delta v = \sum_r \sum_R I_{rR} \tilde{\Psi}(R)\, \tilde{p}(r)\, \Delta v = \sum_R \tilde{\Psi}(R) \underbrace{\sum_r I_{rR}\, \tilde{p}(r) \frac{\Delta v}{\Delta V}}_{\tilde{p}(R)} \Delta V$$

T. Ono & K. Hirose, *Phys. Rev. Lett.*, ***82**, 5016 (1999)*

Test of accuracy

Nitrogen molecule:

XC	d (bohr)			E (eV)		
	PAW	all-electron	Dacapo	PAW	all-electron	Dacapo
LDA	2.072	2.071		11.59	11.58	10.54
PBE	2.086	2.084	2.116	10.55	10.50	9.68
revPBE	2.091	2.089		10.06	10.05	9.50

Bulk aluminium:

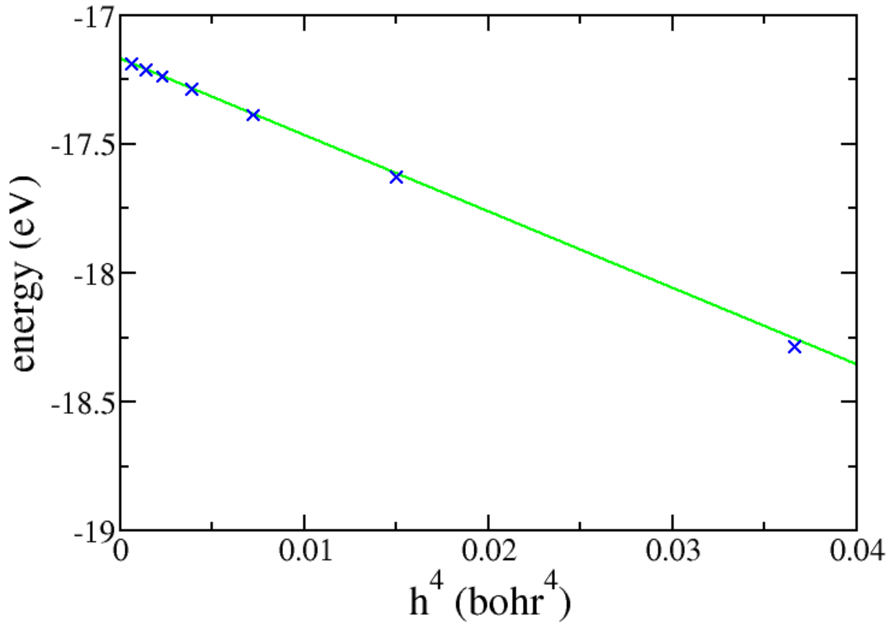
structure	a (Å)		B (GPa)	
	PAW	all-electron	PAW	all-electron
FCC	3.993	3.983	86.7	84.0
BCC	3.201	3.193	77.0	75.0

formation energy	ΔE (eV)	
	PAW	all-electron
BCC – FCC	0.10	0.08

N_2 molecule

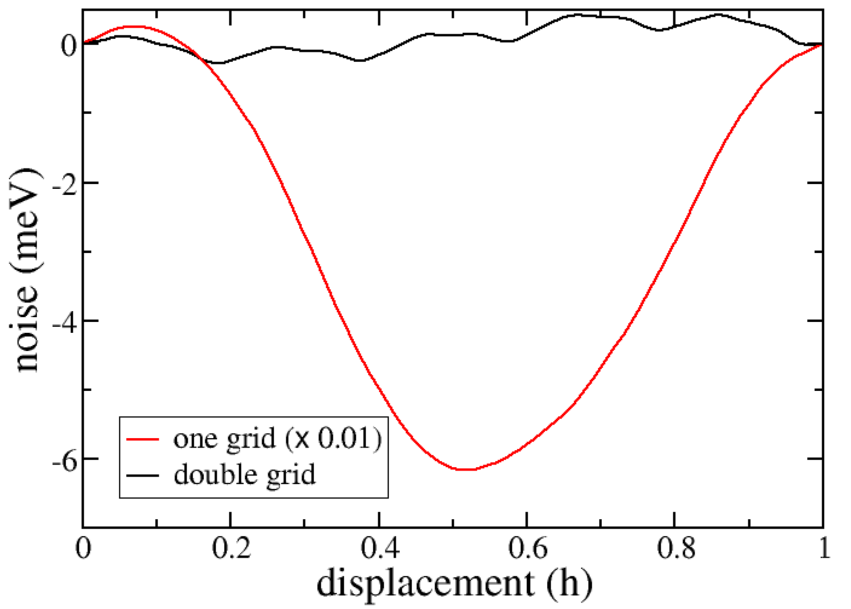
Convergence:

$$\Delta E \sim h^4$$

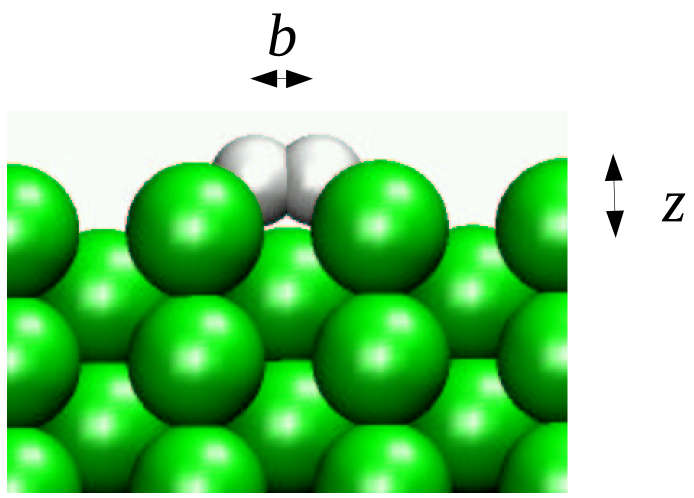


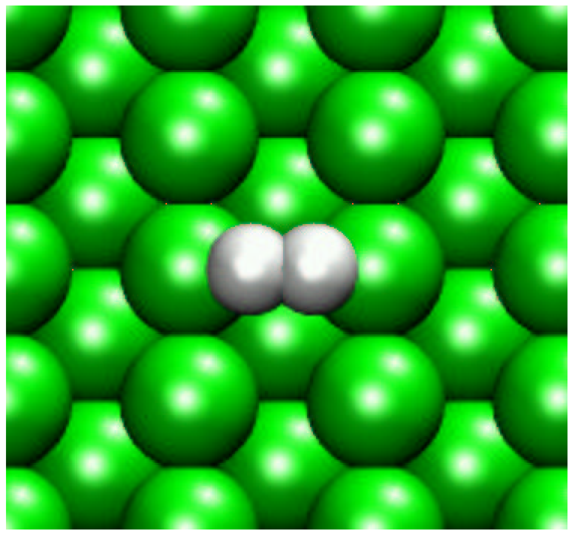
Grid noise:

$h = 0.2$ Å

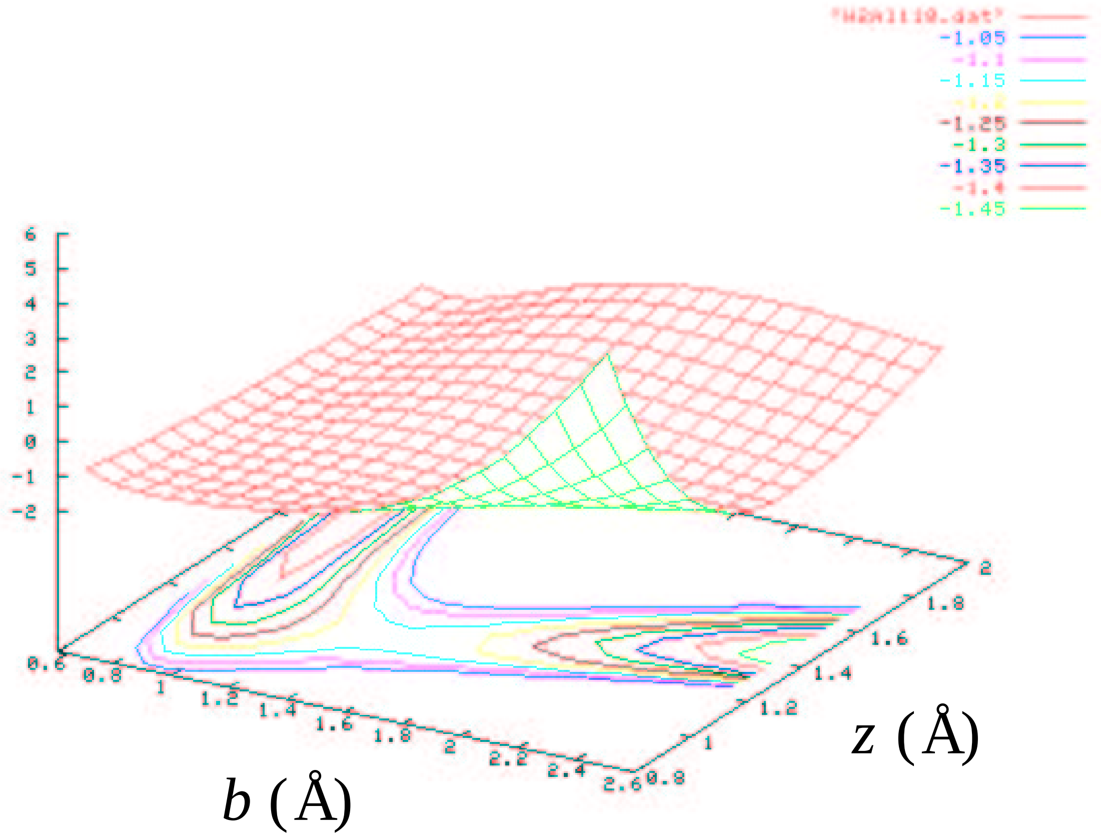


Test: H_2 dissociation on Al(110)





unitcell: 5 layers, (2x1) cell
k-points: (6x2) IBZ: 3
$h = 0.23$ Å



Why use Python?

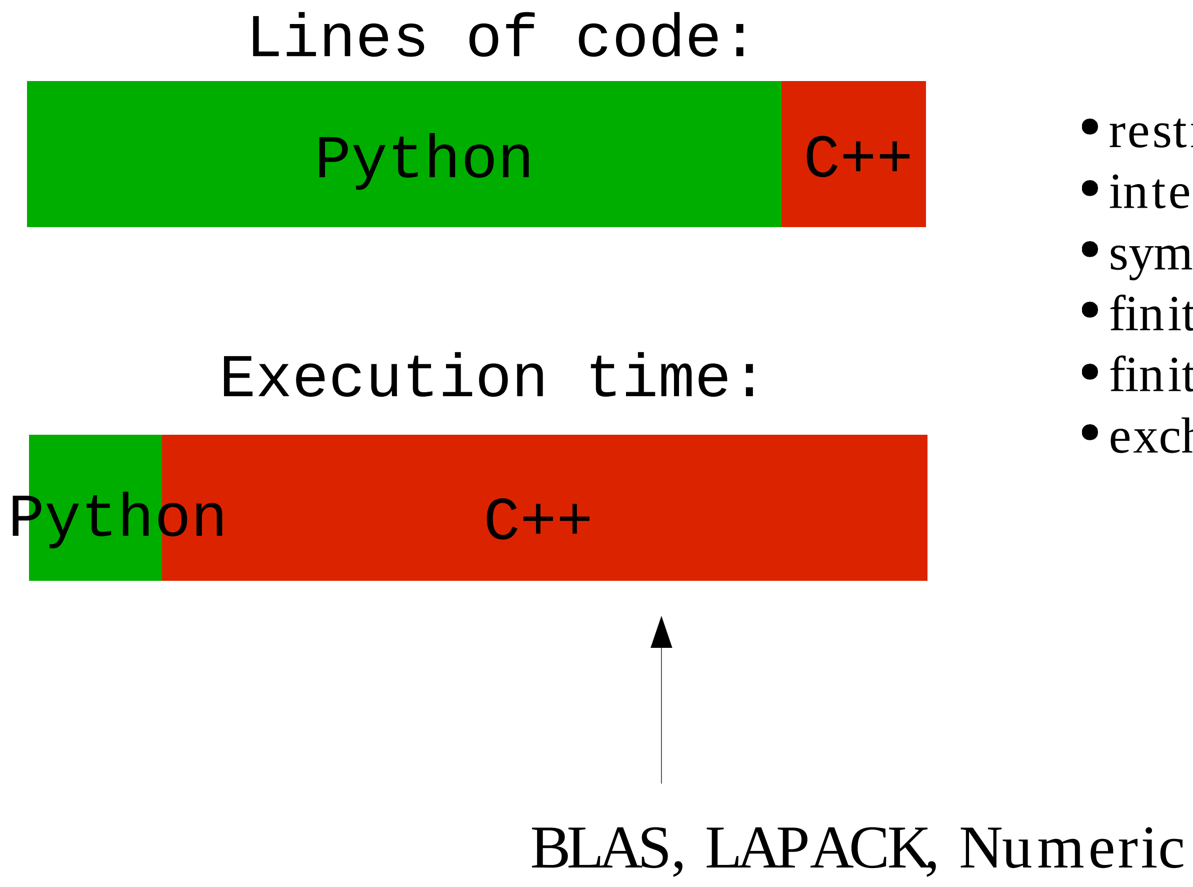
Premature optimization is the root of all evil.

Tony Hoare

By choosing a lower level language, like C+ + , at the start of your project, rather than a higher level one, like Python, you ARE optimizing WAY prematurely.

Alex Martelli

The ultimate goal:

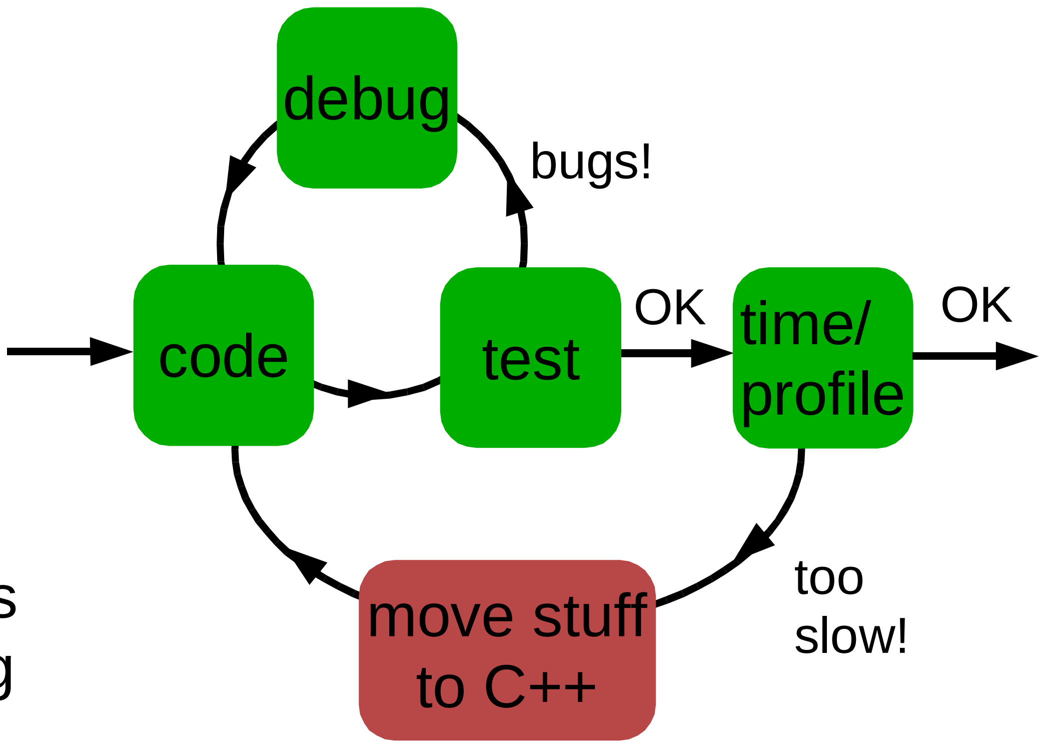


Python:	5600 lines
C++:	1800 lines

- restrictions
- interpolations
- symmetrization
- finite difference Laplacian
- finite difference gradients
- exchange-correlation functionals

Python is Fast!

1) No compilation and linking
2) No declarations and .h files
3) Easy interactive debugging
4) No memory management
5) Many bugs don't exist in Python
6) Object Oriented



Future

- d-projectors
- Non-spherical
  - compensation charges
  - exchange-correlation energy
- Scalar-relativistic data sets
- Parallelization
- ...
- Order-N

Conclusions

- It is possible to do PAW calculations efficiently in real space.
- The Python/C++ combination works well for this type of work.

How fast is Python?

The number *e*:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

```python
e = 0.0
f = 1.0
for n in range(100):
    e += f
    f /= n + 1
print "e =", e
```

```cpp
#include <iostream>
int main()
{
    double e = 0.0;
    double f = 1.0;
    for (int n = 0; n < 100; n++)
      {
        e += f;
        f /= n + 1;
      }
    cout << "e = " << e << endl;
}
```

300 µs

6 µs

```python
print "e =", 1 +  sum(divide.accumulate(arange(1, 100, typecode= Float)))
```