University at Buffalo
The State University of New York

Overview of the Libra code

Department of Chemistry, University at Buffalo, The State University of New York, Buffalo, NY 14260-3000

Alexey Akimov

"Excited States and Nonadiabatic Dynamics CyberTraining Workshop"
June 14, 2021

Libra

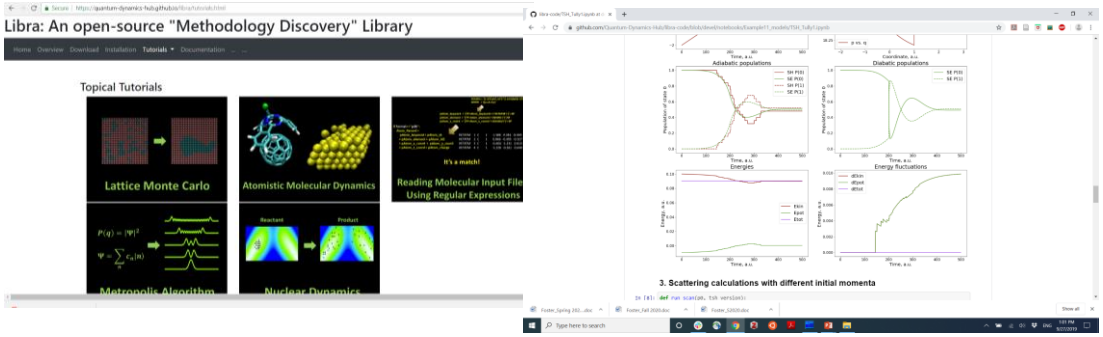
Alexey Akimov

University at Buffalo
The State University of New York

https://github.com/Quantum-Dynamics-Hub/libra-code

Akimov *JCC,* **2016**, 37, 1626
Pradhan et al. *JPCM,* **2018**, 30, 484002
Sato et al. *PCCP,* **2018**, 20, 25275.

- Focus on understanding/assessing methods – methodology prototyping
- A range of methods: fully quantum (grid), TSH, Ehrenfest, wavepackets, decoherence schemes, individual vs. coupled trajectories, etc.
- Thin boundary between C++ and Python, high modularity, model problems database,
- Applications: molecular, condensed matter, NBRA and beyond, model and atomistic
- Interfaces with: QE, ErgoSCF, DFTB+, GAMESS, Gaussian, built-in ES
- **Additional functionality**: versatile analysis and auxiliary tools



- Languages: C++, Python
- Code documentation: extensive
- Tutorials and user documentation: extensive
- Testing: some
- License: GNU GPL 3.0

- Numerous Tutorials and Examples (e.g. Jupyter Notebooks)
- Forum https://groups.google.com/forum/#!forum/quantum-dynamics-hub
- MolSSI workshop materials

Libra philosophy

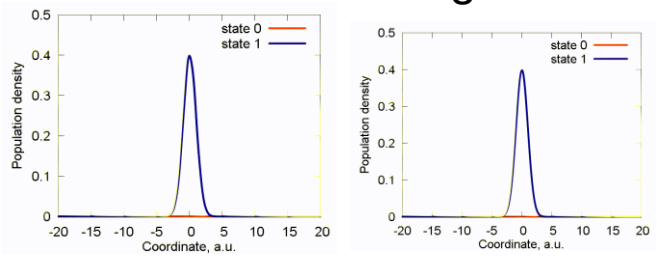
University at Buffalo
The State University of New York

modular

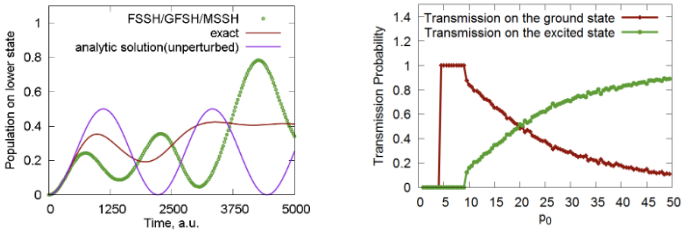
- Maximize and simplify the re-use, OOP
- A variety of methods

versatile

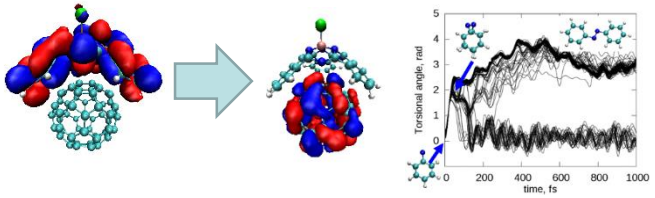
Exact TD-SE integration



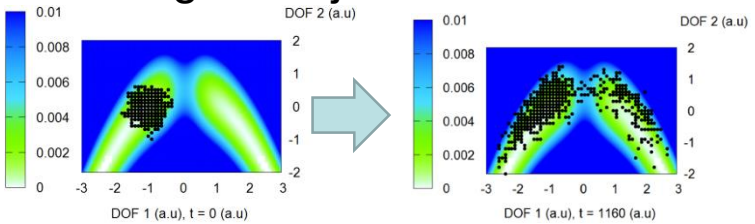
Ehrenfest & TSH methods



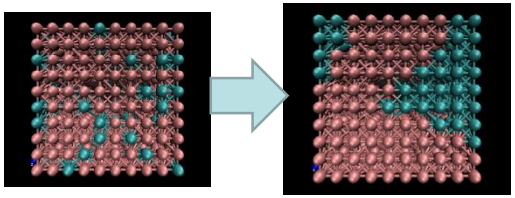
TSH with atomistic systems



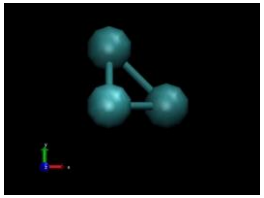
Entangled trajectories methods



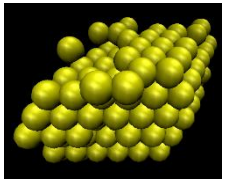
Lattice Monte Carlo

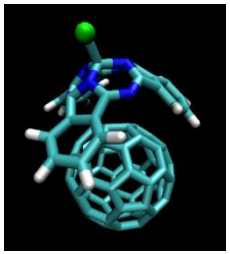


Rigid body



Classical MD





"methodology prototyping"

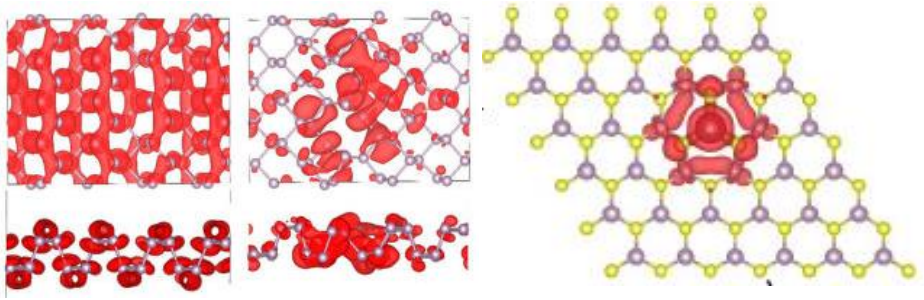
- Use with model problems and atomistic simulations
- Python – for convenience, C++ - for efficiency

Libra & Pyxaid for Energy Materials

University at Buffalo
The State University of New York

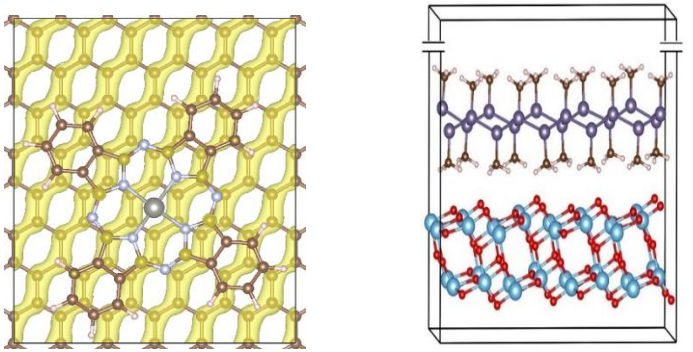
2D systems

Long el al. *JPCL* **2016**, *7*, 653.



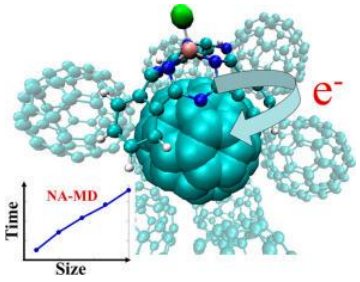
2D heterojunctions

Nijamudheen, A.; AVA *JPCC*, **2017**, *121,* 6520



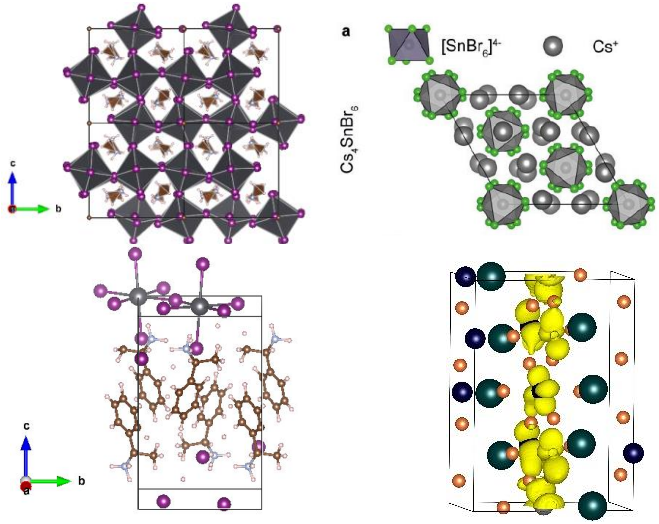
Organic heterojunctions

Sato et al. *PCCP,* **2018**, 20, 25275.



Perovskites

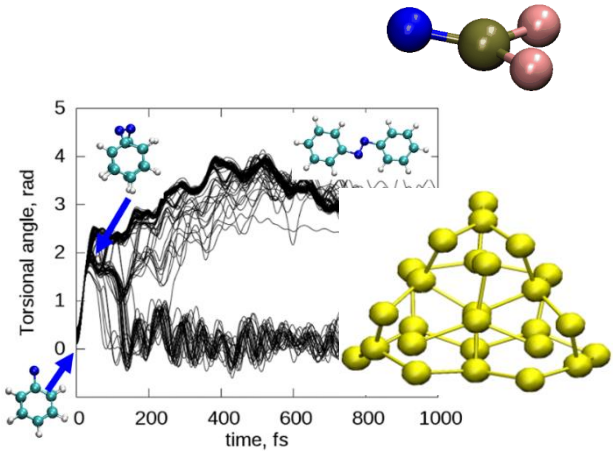
Nijamudheen, A.; AVA *JPCL* **2018**, *9*, 248



Quantum Dots & Molecules

Lin, Y.; AVA *JPCA.* **2016**, 120, 9028

Pradhan et al. *JPCM,* **2018**, 30, 484002



C++/Python Interoperability

University at Buffalo
The State University of New York

C++ layer

Python layer

boost C++ LIBRARIES

User-Customized & Library Python functions

Call by C++

Eigen 3

libra_py

Prototype and Test (Discover)

Translate code

Use in C++ codes

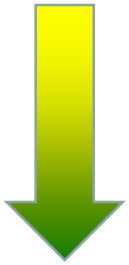
Use in Python codes

API Diversity

University at Buffalo
The State University of New York

- The goal is to suite the **use cases at various levels of complexity**
- Find a balance between **simplicity** and **flexibility**
- Mix of **function-oriented** and **object-oriented** functionality

Developer/Efficiency



User/Convenience

```
double gaussian_overlap( AO* AOa, AO* AOb,int is_normalize, int is_derivs,
VECTOR& dIdA, VECTOR& dIdB, vector<double*>& auxd,int n_aux);

double gaussian_overlap( AO* AOa, AO* AOb,int is_normalize, int is_derivs,
VECTOR& dIdA, VECTOR& dIdB );

double gaussian_overlap(AO* AOa, AO* AOb,int is_normalize);

double gaussian_overlap(AO* AOa, AO* AOb);
```

Computing kinetic energy between Gaussians

```
g1 = PrimitiveG()
g2 = PrimitiveG()
g1.init(n1,m1,k1, a1, VECTOR(x1, y1, z1))
g2.init(n2,m2,k2, a2, VECTOR(x2, y2, z1))

kin = kinetic_integral(g1,g2)
```

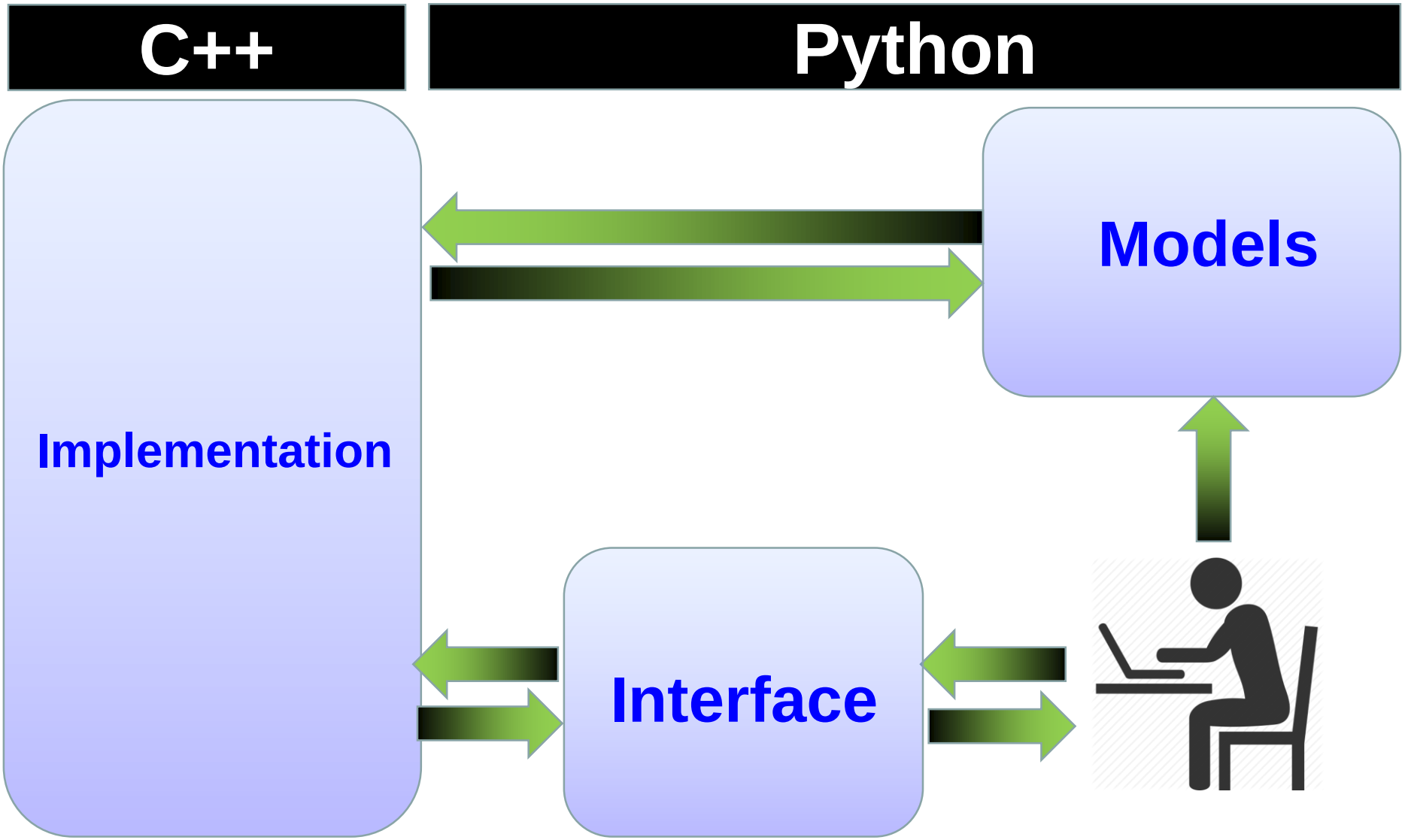
Benchmarked against PyQuante

```
p1 = PyQuante.PGBF.PGBF(a1,(R1.x,R1.y,R1.z),(n1,m1,k1))
p2 = PyQuante.PGBF.PGBF(a2,(R2.x,R2.y,R2.z),(n2,m2,k2))

val_ref = p1.kinetic(p2)
```

Passing Python functions

University at Buffalo
The State University of New York



Example: sampling

University at Buffalo
The State University of New York

```
vector<MATRIX>
metropolis_gau
(Random& rnd,
bp::object target_distribution,
MATRIX& dof,
bp::object distribution_params,
int sample_size, int
start_sampling,
double gau_var){
```

Metropolis Algorithm

```
double p_old =
bp::extract<double>(
target_distribution(s_old,
distribution_params) );

...
}
```

C++

```
def test():
  q  = MATRIX(ndof, 1)
  output = metropolis_gau( piab, q, params, ...)
```

User calls the sampling

Output

```
def piab(q, params):
```

User defines the probability density

Python

Example

University at Buffalo
The State University of New York

User defines how to run the MC sampling
(Interface)

```
q = MATRIX(1,1);  q.set(0, 0.5)
params = {"k":1.0, "m":2000.0, "states":[0], "coeffs":[1.0]}
Nsamp = 1000000; Nstart = 50000
sampling = metropolis_gau(rnd, HO_sup, q, params, Nsamp,Nstart, 0.05)
bin(sampling, -1.5, 2.0, 0.01, 0, 0, "_distrib-1.txt")
```

User defines what probability distribution function is to be sampled
(Model)

```
def HO_sup(q, params):

    k = params["k"];    m = params["m"];
    states = params["states"];    coeffs = params["coeffs"]

    x = q.get(0)
    sz = len(states)
    p = 0.0
    for n in xrange(sz):
        p = p + coeffs[n] * ket_n(x, states[n], k, m)
    p = p * p
    return p
```

The dynamical algorithm is in C++, but…
Don’t need to implement the model in C++

Initial conditions: Metropolis Sampling

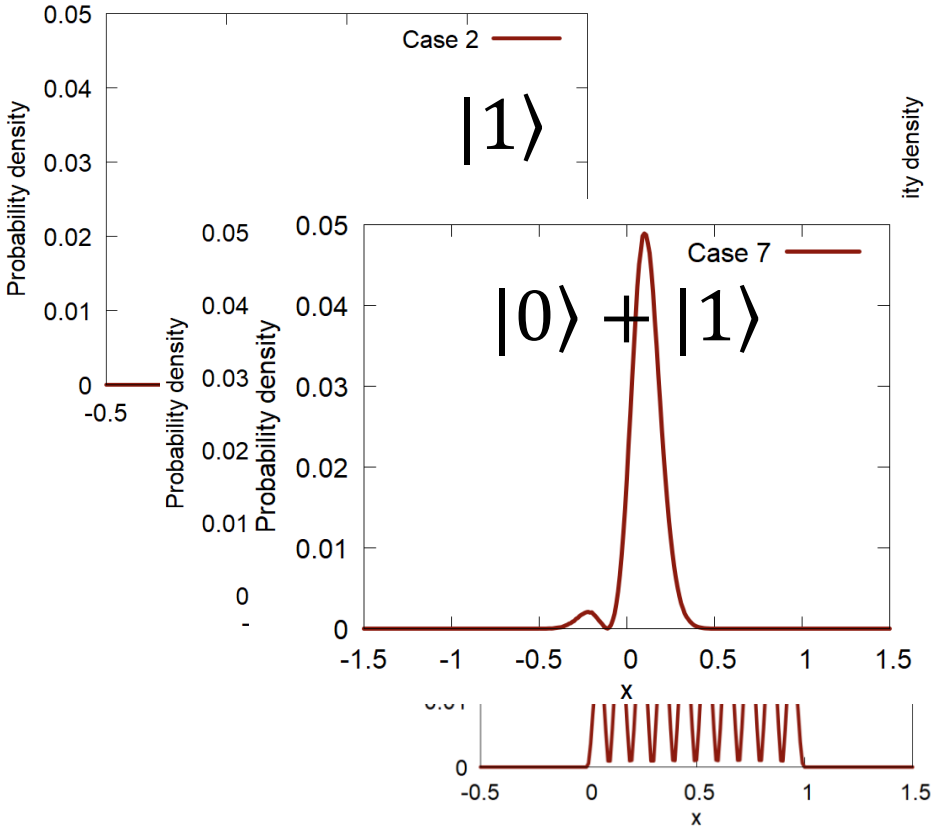
University at Buffalo
The State University of New York

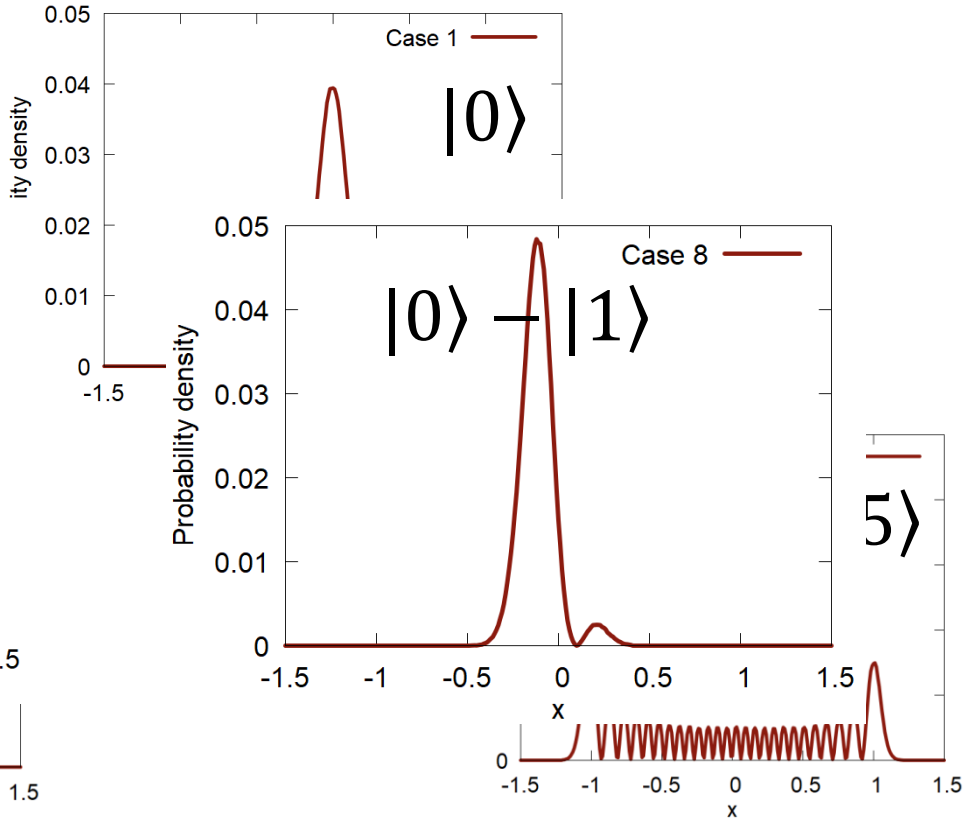
Particle in a box

$$\psi_n(q) \sim sin\left(\frac{\pi n q}{L}\right)$$

Harmonic oscillator

$$\psi_n(q) \sim H_n(q\sqrt{\alpha}) \exp\left(-\frac{\alpha q^2}{2}\right)$$





Some other ideas

University at Buffalo
The State University of New York

Default & Critical Parameters

```
# Parameters and dimensions
critical_params = [  ]
default_params = { "rep_tdse":1, "rep_ham":0, "rep_sh":1, "rep_lz":0, "tsh_method":-1,
                   "force_method":1, "nac_update_method":1, "rep_force":1,
                   "use_boltz_factor":0, "Temperature":300.0, "do_reverse":1, "vel_rescale_opt":-1,
                   "do_phase_correction":1, "tol":1e-3,
                   "state_tracking_algo":2, "MK_alpha":0.0, "MK_verbosity":0,
                   "entanglement_opt":0, "ETHD3_alpha":0.0, "ETHD3_beta":0.0,
                   "decoherence_algo":-1, "decoherence_rates":DR,
                   "decoherence_times_type":0, "decoherence_C_param":1.0,
                   "decoherence_eps_param":0.1, "dephasing_informed":0,
                   "ave_gaps":AG, "instantaneous_decoherence_variant":1, "collapse_option":0,
                   "ensemble":0, "thermostat_params":{},
                   "dt":1.0*units.fs2au, "nsteps":1,
                   "output_level":-1, "file_output_level":-1, "prefix":"tmp"
                 }

comn.check_input(dyn_params, default_params, critical_params)
```

Type & Amount of Output

```
# Memory output
if output_level >= 1:
    obs_T.append(i*dt)
    obs_Ekin.append(Ekin)
    obs_Epot.append(Epot)
    obs_Etot.append(Etot)
    obs_dEkin.append(dEkin)
    obs_dEpot.append(dEpot)
    obs_dEtot.append(dEtot)
    obs_dm_adi.append(CMATRIX(dm_adi))
    obs_dm_dia.append(CMATRIX(dm_dia))
    obs_pop.append(MATRIX(pops))

# Memory output
if output_level >= 2:
    obs_q.append(MATRIX(q))
    obs_p.append(MATRIX(p))
    obs_Cadi.append(CMATRIX(Cadi))
    obs_Cdia.append(CMATRIX(Cdia))
    obs_states.append(list(states))

# File output
res12 = q, p, Ekin, Epot, Etot, dEkin, dEpot, dEtot, Cadi, Cdia, dm_adi, dm_dia, pops, states
dynamics_io.print_results12(i, dt, res12, prefix, file_output_level)
```

Acknowledgements

Funds:

UB startup

Computing resources:

OAC-NSF

Thank you! Questions?