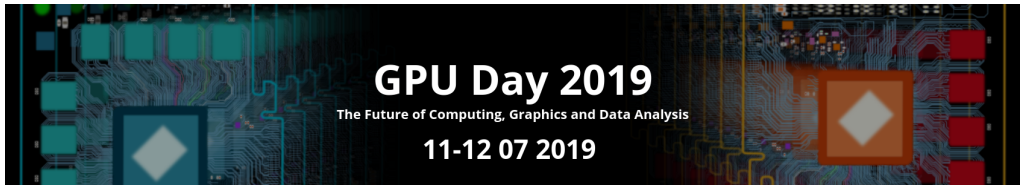
C++QED – a framework for simulating open quantum dynamics

the first ten years

András Vukics

Department of Quantum Optics and Quantum Information
Institute for Solid State Physics and Optics
Wigner Research Centre for Physics of the Hungarian Academy of Sciences



CIRCUIT AND CAVITY QUANTUM ELECTRODYNAMICS

Der Wissenschaftsfonds.

Introduction

Motivation

Building recyclable simulations for open quantum systems

CQED – define elementary physical systems as building blocks of composite systems
+ time-evolution modules

Introduction

Motivation

Building recyclable simulations for open quantum systems

C++qed – define elementary physical systems as building blocks of composite systems
+ time-evolution modules

History

Cavity QED with moving particles ⇒ C++qed

Subsequently: general quantum optics, atomic and molecular(, many-body) physics

Open-source framework `http://cppqed.sf.net/`

2006–2008: version1 partial documentation EPJD44:585(2007)

2008–: version2 online documentation + CPC 183:1381(2012) and 185:2380(2014)

Introduction

Motivation

Building recyclable simulations for open quantum systems

C++qed – define elementary physical systems as building blocks of composite systems
+ time-evolution modules

History

Cavity QED with moving particles ⇒ C++qed

Subsequently: general quantum optics, atomic and molecular(, many-body) physics

Open-source framework `http://cppqed.sf.net/`

2006–2008: version1 partial documentation EPJD44:585(2007)

2008–: version2 online documentation + CPC 183:1381(2012) and 185:2380(2014)

Basic specification: simulate fully quantum open dynamics in general

Schrödinger equation Hamiltonian in finite discrete basis

open dynamics jump (Lindblad) operators in the same basis

Further specification

Basic building blocks

free systems e.g. mode, spin, 1D motional degree of freedom

- or anything describable with 1 quantum number

interactions e.g. x–x, Jaynes-Cummings

Further specification

Basic building blocks

free systems e.g. mode, spin, 1D motional degree of freedom

- or anything describable with 1 quantum number

interactions e.g. x–x, Jaynes-Cummings

Time-evolution modules for generic systems

- Master equation
  - adaptive ODE evolution
- Monte Carlo wave-function trajectory
  - method modified to adaptive time step
- ensemble of trajectories:

$$\rho_{\text{ensemble}}(t) = \frac{1}{\text{number of trajectories}} \sum_{i \in \{\text{set of trajectories}\}} |\Psi_i(t)\rangle \langle \Psi_i(t)|$$

Description of open quantum systems

The Born-Markovian master equation

$$\dot{\rho} = \frac{1}{\hbar}\left[H, \rho\right] + \sum_m \left(J_m \rho J_m^\dagger - \frac{1}{2}\left[J_m^\dagger J_m, \rho\right]_+ \right) \equiv 2\Re\left\{\frac{H_{\mathrm{nH}}}{\hbar}\rho\right\} + \sum_m J_m \left(J_m \rho\right)^\dagger$$

Lindblad operators J_m

non-Hermitian Hamiltonian $H_{\mathrm{nH}} = H - \frac{\hbar}{2}\sum_m J_m^\dagger J_m$

Description of open quantum systems

The Born-Markovian master equation

$$\dot{\rho} = \frac{1}{\hbar}\left[H,\rho\right] + \sum_m \left(J_m \rho J_m^\dagger - \frac{1}{2}\left[J_m^\dagger J_m, \rho\right]_+ \right) \equiv 2\Re\left\{\frac{H_{\mathrm{nH}}}{\hbar}\rho\right\} + \sum_m J_m \left(J_m \rho\right)^\dagger$$

Lindblad operators J_m

non-Hermitian Hamiltonian $H_{\mathrm{nH}} = H - \frac{\hbar}{2}\sum_m J_m^\dagger J_m$

Example: driven mode @ finite temperature

Hamiltonian $H = -\delta a^\dagger a + \left(\eta\, a^\dagger + \text{h.c.}\right)$

Liouville superoperator

$$\mathcal{L}(\rho) = \kappa_- \left(2a\rho a^\dagger - \left[a^\dagger a, \rho\right]_+ \right) + \kappa_+ \left(2a^\dagger \rho a - \left[a\, a^\dagger, \rho\right]_+ \right)$$

$\Rightarrow$ quantum-jump operators

$$J_0 = \sqrt{2\kappa_-}\; a \qquad \text{(photon emission)} \qquad \kappa_- = \kappa\left(n_{\mathrm{Th}} + 1\right)$$

$$J_1 = \sqrt{2\kappa_+}\; a^\dagger \qquad \text{(photon absorption)} \qquad \kappa_+ = \kappa\, n_{\mathrm{Th}}$$

Description of open quantum systems

Unravelling into Monte Carlo wave-function trajectories

Method first published @ around 1990

Description of open quantum systems

Unravelling into Monte Carlo wave-function trajectories

Method first published @ around 1990

Fixed δt

1. Coherent step
   - Non-unitary (norm-decreasing) evolution:
     $$|\Psi_{\mathrm{nH}}(t+\delta t)\rangle = \left(1 - \frac{iH_{\mathrm{nH}}\,\delta t}{\hbar}\right)|\Psi(t)\rangle$$
   - Total jump probability: $\delta p = 1 - \|\Psi_{\mathrm{nH}}(t+\delta t)\|^2 = \delta t \sum_m \|J_m\,|\Psi(t)\rangle\|^2$
   - Probability distribution of jumps: $\delta p_m = \delta t\ \|J_m\,|\Psi(t)\rangle\|^2 / \delta p$

Description of open quantum systems

Unravelling into Monte Carlo wave-function trajectories

Method first published @ around 1990

Fixed δt

1. Coherent step
   - Non-unitary (norm-decreasing) evolution:
     $$|\Psi_{\mathrm{nH}}(t+\delta t)\rangle = \left(1 - \frac{iH_{\mathrm{nH}}\,\delta t}{\hbar}\right)|\Psi(t)\rangle$$
   - Total jump probability: $\delta p = 1 - \|\Psi_{\mathrm{nH}}(t+\delta t)\|^2 = \delta t \sum_m \|J_m\,|\Psi(t)\rangle\|^2$
   - Probability distribution of jumps: $\delta p_m = \delta t\ \|J_m\,|\Psi(t)\rangle\|^2 / \delta p$
2. Probing for quantum jump with probability δp

$$\begin{array}{rl} \text{no} & |\Psi(t+\delta t)\rangle = |\Psi_{\mathrm{nH}}(t+\delta t)\rangle / \sqrt{1-\delta p} \\ \text{yes} & |\Psi(t+\delta t)\rangle = \sqrt{\delta t/(\delta p_m \delta p)}\ J_m\,|\Psi(t)\rangle \text{ distro } \delta p_m \end{array}$$

Open quantum systems in C++QED

Amendment: adaptive MCWF

Upper limit of δt

- validity of ODE stepping
- $\delta p \ll 1 \Rightarrow$ more than 1 jumps per δt with negligible probability

Open quantum systems in C++QED

Amendment: adaptive MCWF

Upper limit of δt

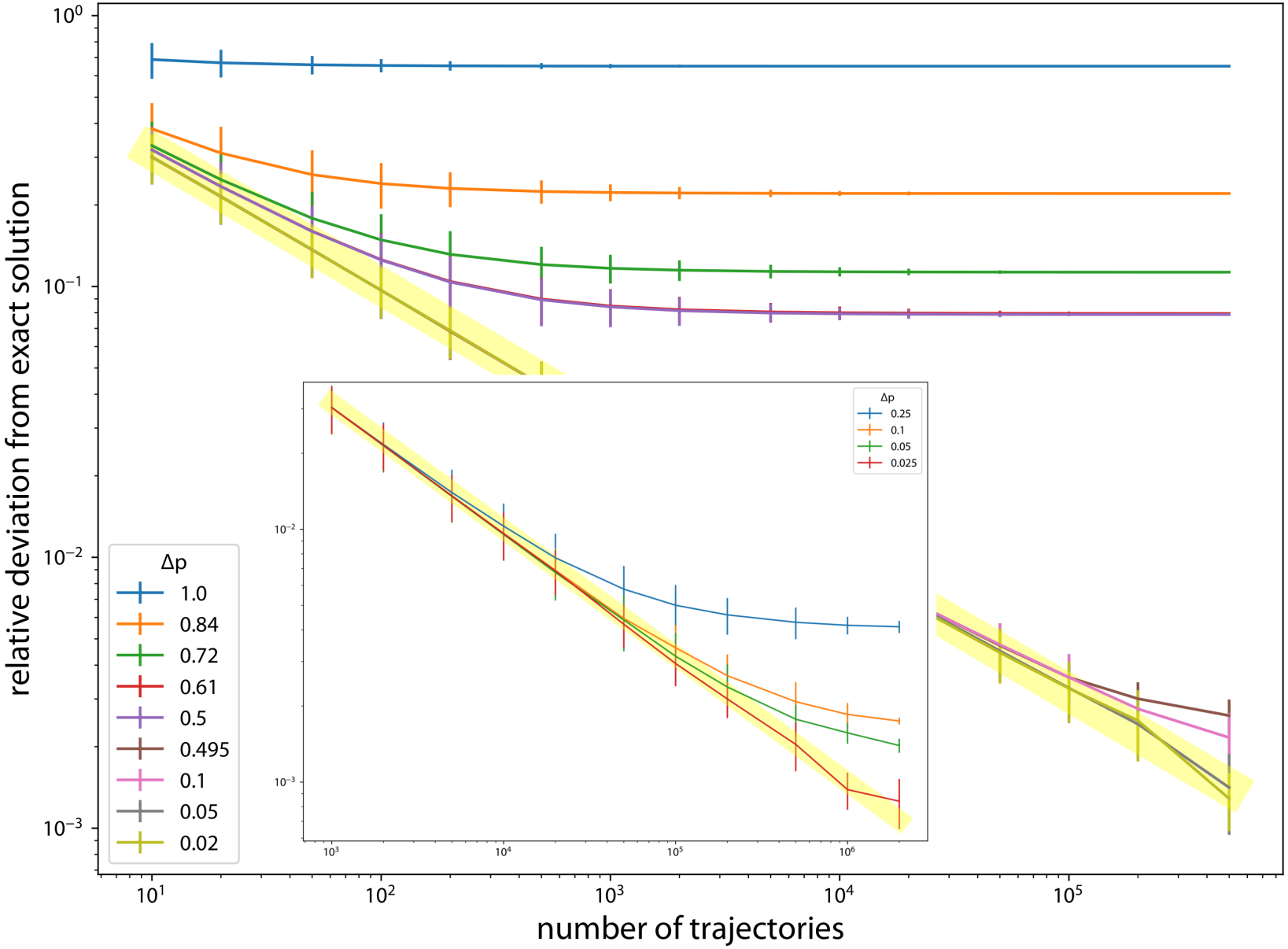
- validity of ODE stepping
- $\delta p \ll 1 \Rightarrow$ more than 1 jumps per δt with negligible probability

Our stepsize control – new parameter Δp introduced

- ODE stepper internal mechanism (absolute and relative precision)
  - 1st order stepping replaced by a higher-order adaptive ODE method (e.g. RKCK)
- $\delta p < \Delta p \ll 1$ two stage control:
  - $\delta p \gtrsim \Delta p \Rightarrow$ decrease δt for the *next* timestep
  - $\delta p \gg \Delta p \Rightarrow$ cancel timestep

MCWF convergence – driven mode @ finite temperature

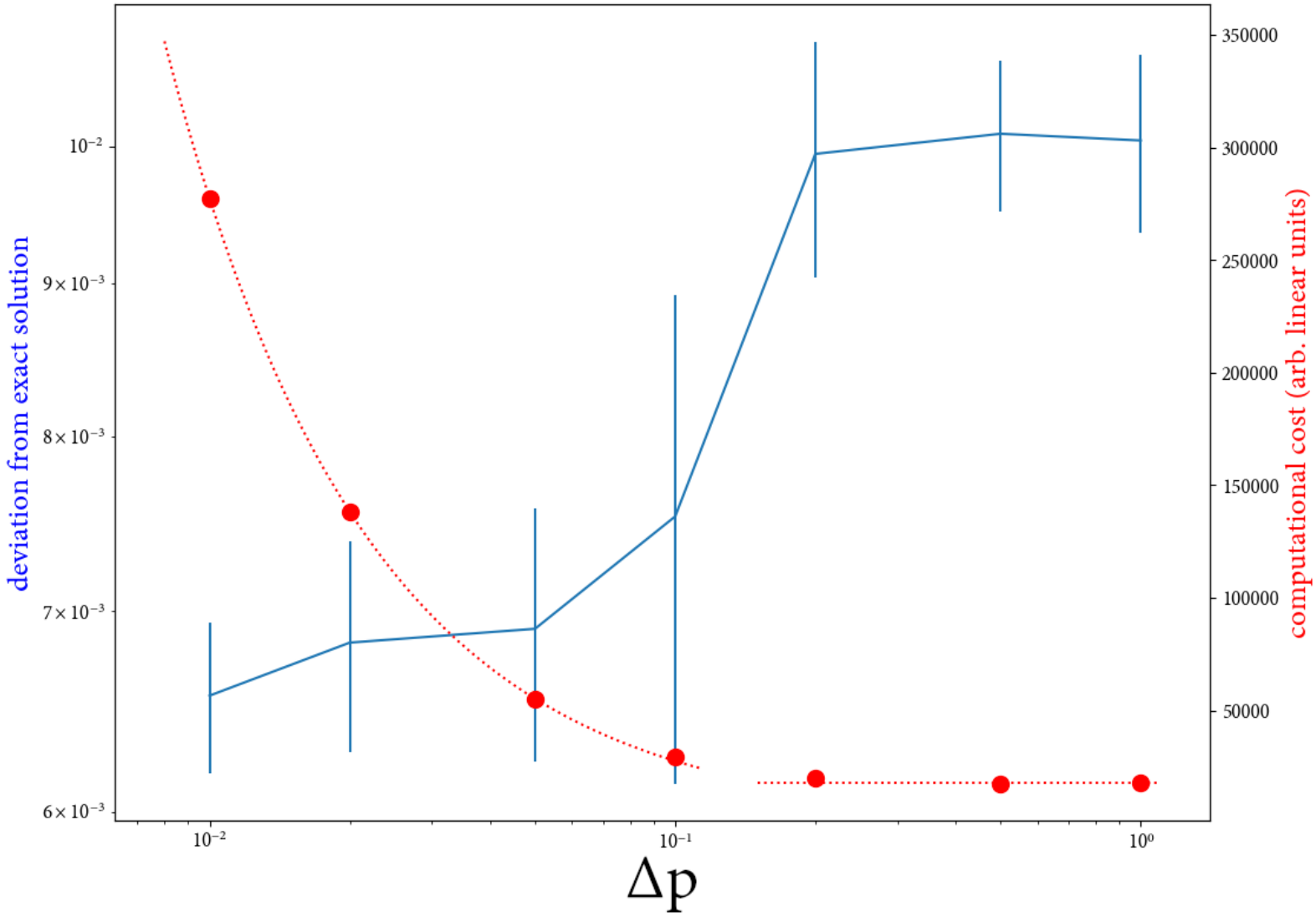
Quantifying the rôle of Δp



[Kornyik & Vukics, Comp. Phys. Comm. **238**:88–101 (2019)]

MCWF convergence – driven mode @ finite temperature

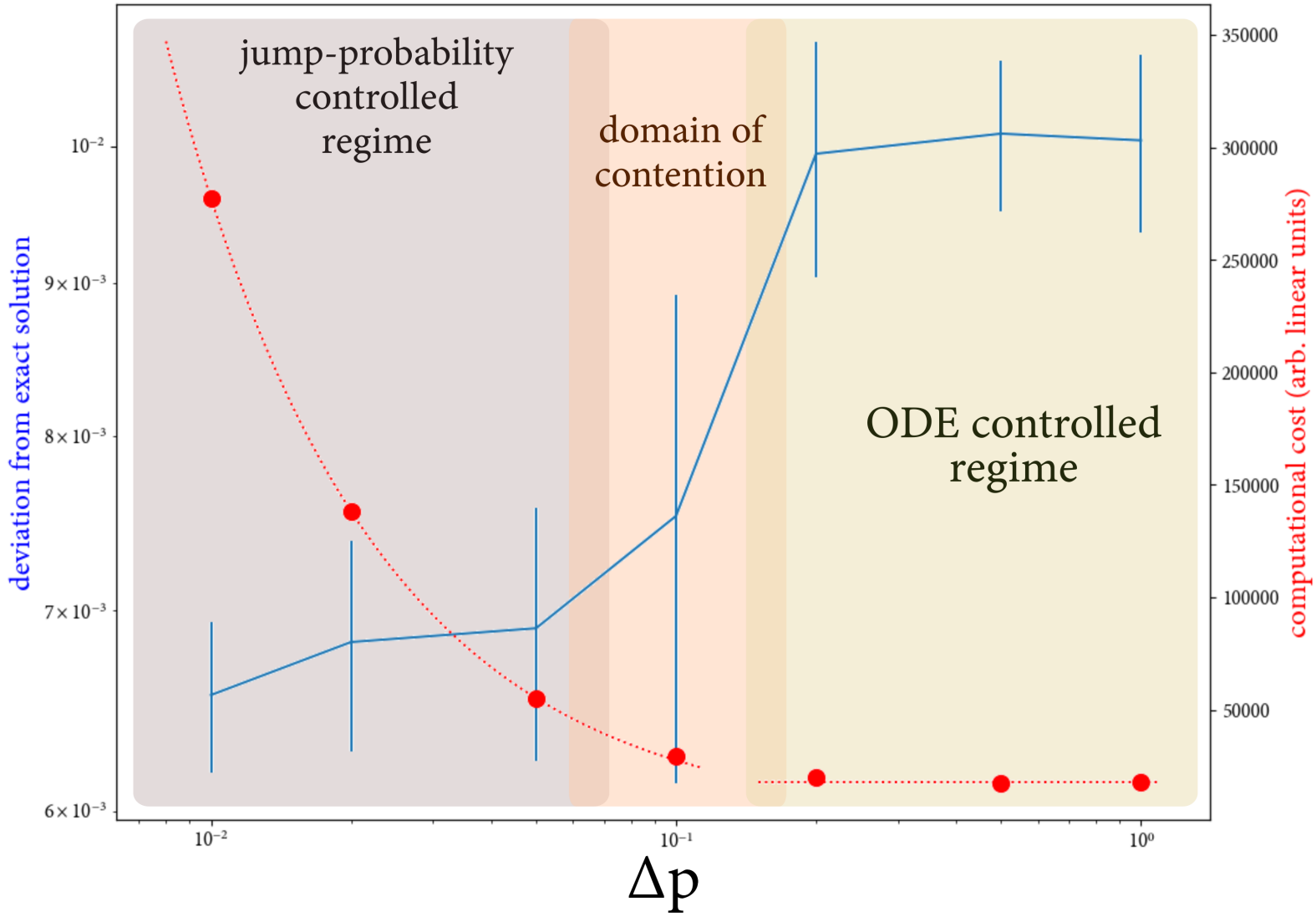
Quantifying the rôle of Δp



[Kornyik & Vukics, Comp. Phys. Comm. **238**:88–101 (2019)]

MCWF convergence – driven mode @ finite temperature

Quantifying the rôle of Δp



[Kornyik & Vukics, Comp. Phys. Comm. **238**:88–101 (2019)]

Compile-time algorithms

C++: compiled language + Turing-complete toolset available @ compile time

Implementation: Boost.MPL

Compile-time algorithms

C++: compiled language + Turing-complete toolset available @ compile time

Implementation: Boost.MPL

fundamental design principle of C++QED

all information available @ compile time should be processed @ compile time using template metaprogramming

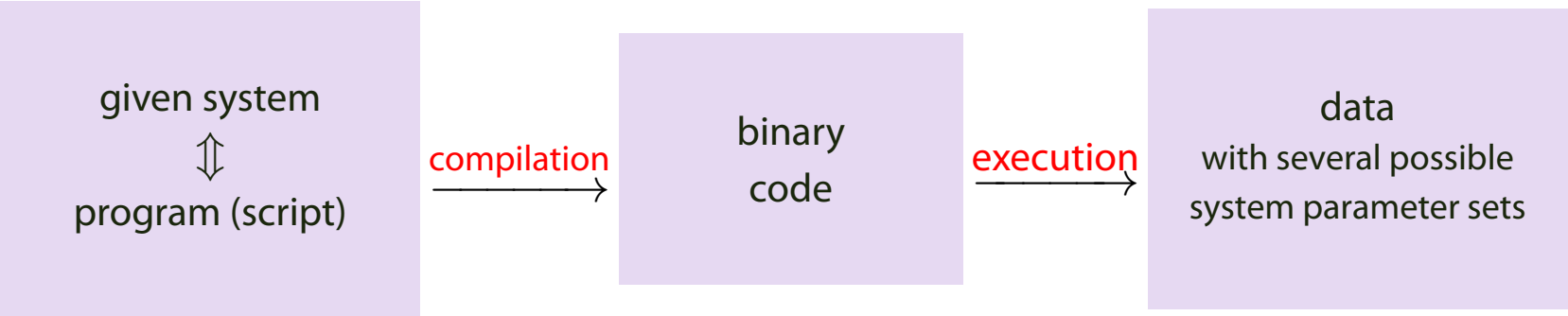
Compile-time algorithms

C++: compiled language + Turing-complete toolset available @ compile time

Implementation: Boost.MPL

fundamental design principle of C++QED

all information available @ compile time should be processed @ compile time using template metaprogramming



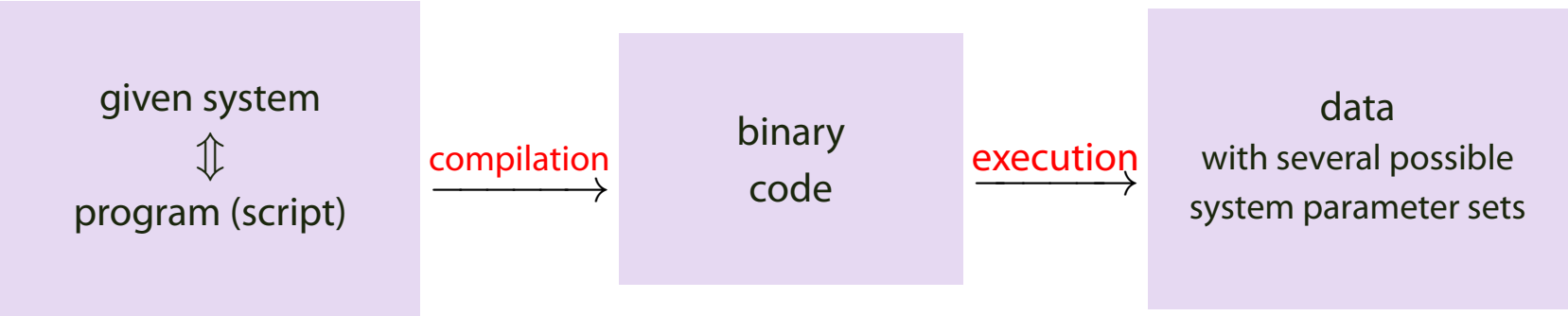
Compile-time algorithms

C++: compiled language + Turing-complete toolset available @ compile time

Implementation: Boost.MPL

fundamental design principle of C++QED

all information available @ compile time should be processed @ compile time using template metaprogramming



For given simulation modules are assembled by high-level C++ program (script)
– defines the physical system and what to do with it

Compile only once ⇒ run several times

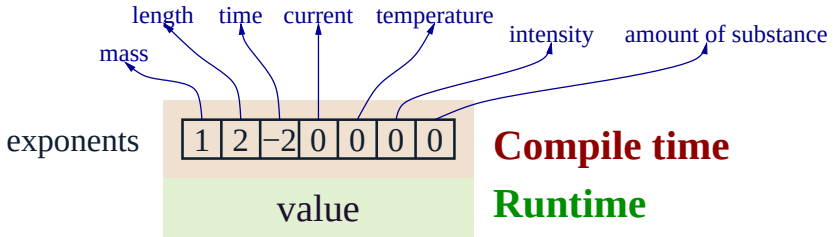
Layout of system known @ compile time ⇒ implies lots of compile-time calculations

Excursus: Template metaprogramming physical application

Dimensional analysis

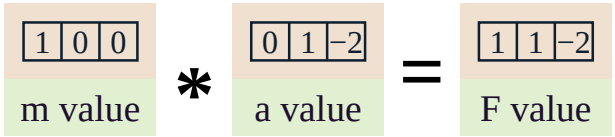
Let's teach C++ dimensions

- bring them into the type system ⇔ "make them part of the grammar"



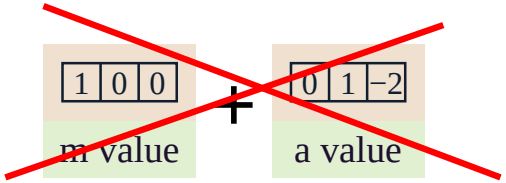
Newton's law

- calculate: $m * a = F$



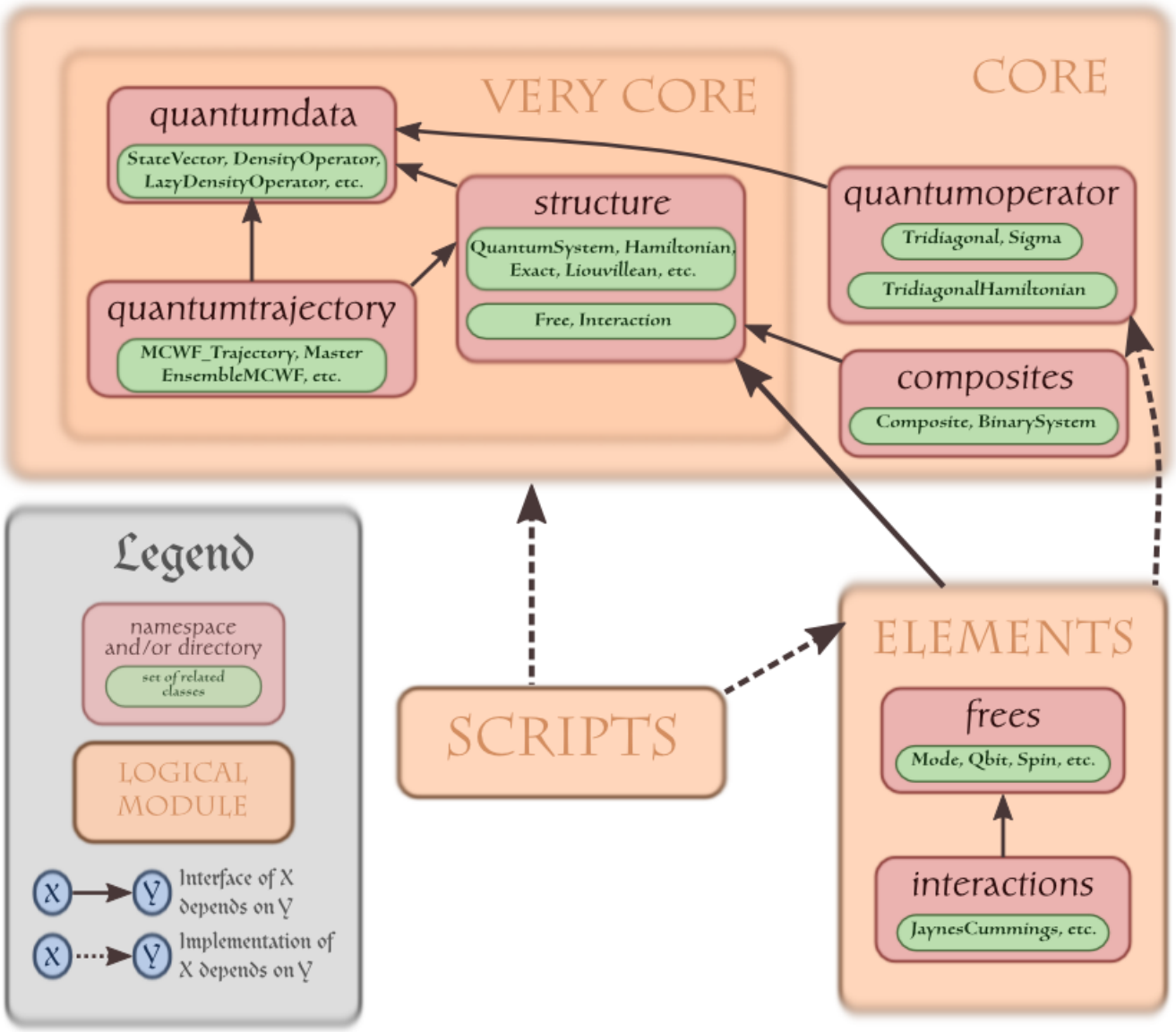
small compile-time algorithm to calculate the resulting dimension

- try to do nonsense: $m + a$



should cause compilation error

Large-scale structure

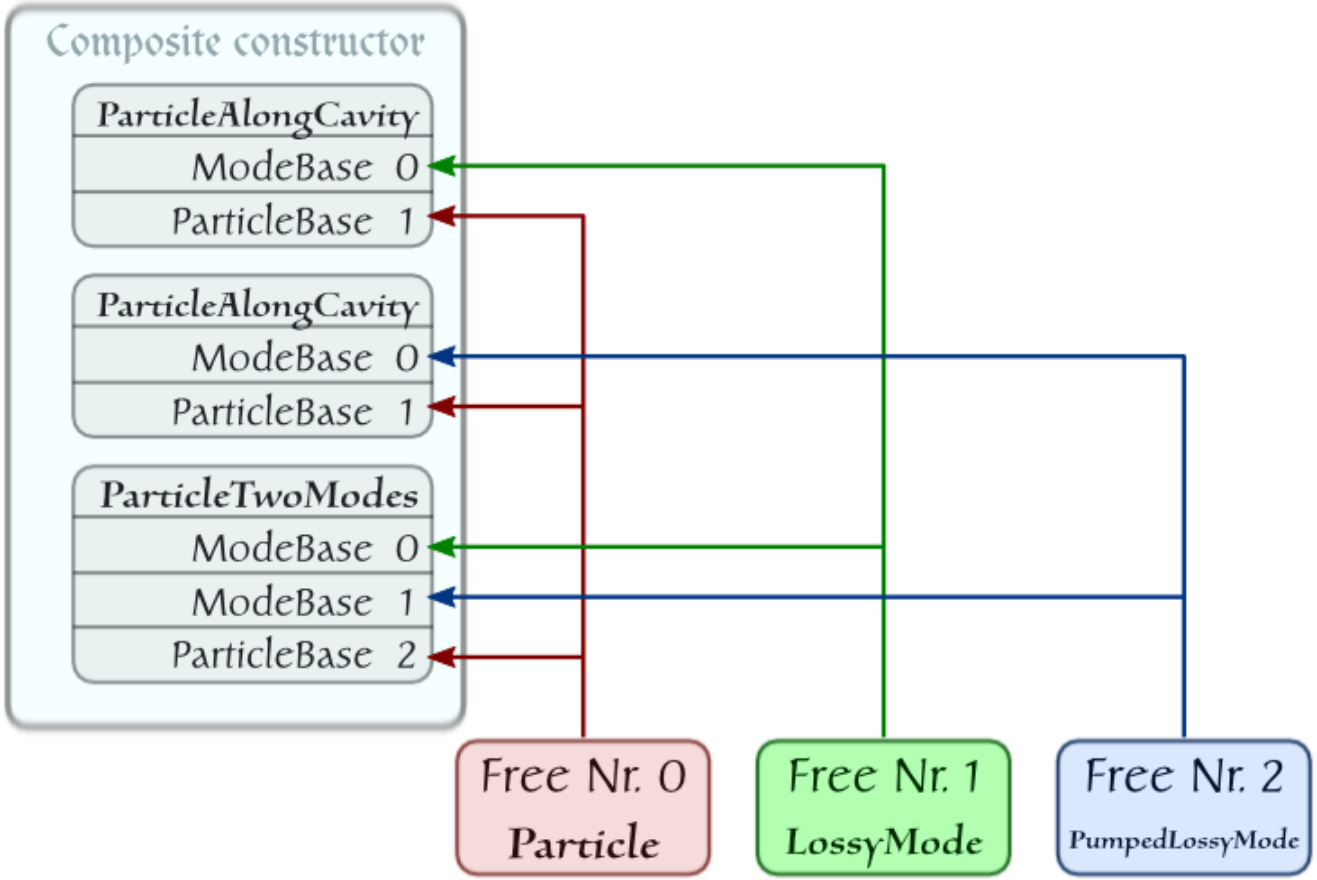


Example script

defining the physical system

System defined as graph of interactions

Particle in ring cavity with two lossy modes, one pumped



interactions mostly binary, but also ternary, quaternary

Building blocks for physical systems

Few are needed for a given problem domain, each having clear physical meaning

Building blocks for physical systems

Few are needed for a given problem domain, each having clear physical meaning

Example

polarizable particles in optical (cavity) fields

Frees	Interactions
(Pumped/Lossy)Mode	ParticleAlongCavity
(Pumped)Particle	ParticleOrthogonalToCavity
	ParticleTwoModes
	(ternary, quaternary)

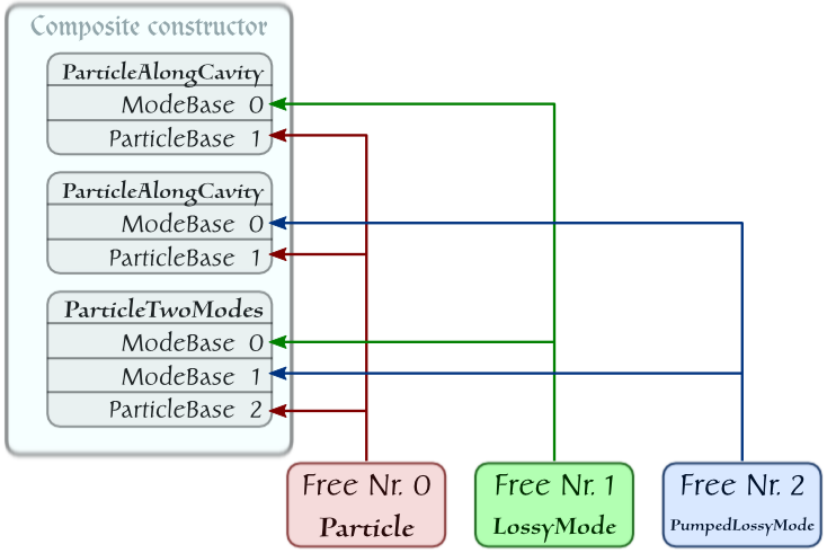
Example script

1-3 Define system part I: free elements

5 Specify initial condition

7-13 Specify & run time evolution

9-11 Define system part II: interactions, layout of full system




```
1   Particle        part (…);
2   LossyMode       plus (…);
3   PumpedLossyMode minus(…);
4
5   StateVector<3> psi(wavePacket(…)*coherent(…)*coherent(…));
6
7   evolve(psi,
8          composite::make(
9              _<1,0>  (ParticleAlongCavity(plus ,part,MFT_PLUS ,…)),
10             _<2,0>  (ParticleAlongCavity(minus,part,MFT_MINUS,…)),
11             _<1,2,0>(ParticleTwoModes(plus,minus,part,…))
12                         ),
13         …);
```

Applications

$\sim$ 20 research papers in 2008–2019

Polarizable particles in optical fields

- particle in the field of a cavity mode
- two particles, two-mode resonator

Applications

$\sim$ 20 research papers in 2008–2019

Polarizable particles in optical fields

- particle in the field of a cavity mode
- two particles, two-mode resonator

Ultracold atoms in optical fields

- optical lattices or BEC in double-well potential coupled to cavity mode

Applications

$\sim$ 20 research papers in 2008–2019

Polarizable particles in optical fields

- particle in the field of a cavity mode
- two particles, two-mode resonator

Ultracold atoms in optical fields

- optical lattices or BEC in double-well potential coupled to cavity mode

(Cavity) optomechanics

Applications

$\sim$ 20 research papers in 2008–2019

Polarizable particles in optical fields

- particle in the field of a cavity mode
- two particles, two-mode resonator

Ultracold atoms in optical fields

- optical lattices or BEC in double-well potential coupled to cavity mode

(Cavity) optomechanics

Complex atoms in electromagnetic fields (with motion)

- Hamiltonian and Liouvillean assembled @ compile time

Implementation

Rely heavily on open-source libraries: Blitz++, Boost, GSL, FLENS

Python interface

Contributed by Raimar Sandner

```cpp
// Copyright András Vukics 2006–2014. Distributed under the Boost Software License
#include "EvolutionComposite.h"

#include "ParticleTwoModes.h"

int main(int argc, char* argv[])
{
  ParameterTable p;

  evolution::Pars pe(p);
  particle::Pars pp(p);
  mode::ParsLossy       pmP(p,"P");
  mode::ParsPumpedLossy pmM(p,"M");
  particlecavity::ParsAlong ppcP(p,"P");
  particlecavity::ParsAlong ppcM(p,"M");

  ppcP.modeCav=MFT_PLUS; ppcM.modeCav=MFT_MINUS;

  auto qmp=updateWithPicture(p,argc,argv,"--");

  particle::Ptr part (make(pp ,qmp));
  mode    ::Ptr plus (make(pmP,qmp));
  mode    ::Ptr minus(make(pmM,qmp));

  quantumdata::StateVector<3> psi(wavePacket(pp)*init(pmP)*init(pmM));

  evolve<0>(psi,
            composite::make(
                            _<1,0>  (ParticleAlongCavity(plus ,part,ppcP)),
                            _<2,0>  (ParticleAlongCavity(minus,part,ppcM)),
                            _<1,2,0>(ParticleTwoModes(plus,minus,part,ppcP,ppcM))
                            ),
            pe);
}
```

```python
import sys
from cpypyqed import *

p=parameters.ParameterTable()

pe=evolution.Pars(p)

pp=particle.Pars(p)

pmP=mode.ParsLossy(p,"P")
pmM=mode.ParsLossy(p,"M")
ppcP=particlecavity.ParsAlong(p,"P")
ppcM=particlecavity.ParsAlong(p,"M")

ppcP.modeCav=ModeFunctionType.PLUS
ppcM.modeCav=ModeFunctionType.MINUS

qmp=updateWithPicture(p,sys.argv,"--")

part=particle.make(pp ,qmp)
plus =   mode.make(pmP,qmp)
minus=   mode.make(pmM,qmp)

evolve(particle.wavePacket(pp)**mode.init(pmP)**mode.init(pmM),
       makeComposite({(1,0):ParticleAlongCavity(plus ,part,ppcP),
                      (2,0):ParticleAlongCavity(minus,part,ppcM),
                      (1,2,0):ParticleTwoModes(plus,minus,part,ppcP,ppcM)}),
       pe)
```

Present problem

Fundamental datastructure (Blitz++ library) outdated

Present problem

Fundamental datastructure (Blitz++ library) outdated

Requirements for replacement

- multiarray interface
- sparse representation capabilities
- efficient operations + linear algebra
- C++11 features, eg.
  - variadic indexing
  - move semantics

Present problem

Fundamental datastructure (Blitz++ library) outdated

Requirements for replacement

- multiarray interface
- sparse representation capabilities
- efficient operations + linear algebra
- C++11 features, eg.
  - variadic indexing
  - move semantics

Most promising candidate: QuantStack/Xtensor library

Thank you for your attention!

http://cppqed.sf.net

CIRCUIT AND CAVITY QUANTUM ELECTRODYNAMICS

Der Wissenschaftsfonds.