

INTRODUCTION TO HIGH-LEVEL-SIMULATION LANGUAGE XMDS 2

Sebastian Wüster



HIGH-LEVEL APPROACH

```
<![CDATA[
  double dens=phi.re*phi.re + phi.im*phi.im;

  dphi_dt = L[phi] + x*Gy[phi] -y*Gx[phi]- i*(U*dens + trap )*phi;
</operators>
</integrate>
```

```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="x" lattice="&Npts;" domain="(-&L;, &L;)" />
    <dimension name="y" lattice="&Npts;" domain="(-&L;, &L;)" />
  </transverse_dimensions>
</geometry>
```

```
<driver name="distributed-mpi" />
```

```
<driver name="multi-path" paths="1000" />
```

```
<noise_vector name="drivingNoise" kind="wiener" type="real" method="dsfmt">
  <components>dW</components>
</noise_vector>
```

- Human-readable equations
- No hassle with loops, array indices, libraries etc.
- Easy addition of more PDE dimensions
- Inbuilt parallel processing (MPI) support
- Inbuilt stochastic (P)DE solving and averaging

COURSE OUTLINE

- 1) Intro to numerical solutions of PDEs (the ultra compact version)
 - 2) XMDS basics (simple example, XML script, main tags)
 - 3) Exercise one (xmDS basics, 1D Gross-Pitaevskii equation)
 - 4) XMDS advanced (advanced examples, parallel code, stochastic DEs, ...)
-
- 5) Exercise two (stochastic sims, kink-bearing ϕ^4 theory)
 - 6) Hacking XMDS

LUNCH

- 7) Exercise three (Code your own problem)

DISCLAIMER

Course targets:

- Enable you to straightaway start working with XMDS
- Enable you to assess whether XMDS suits your problem
- Provide you with some small bits-and-pieces of: XML, C, numerical algorithms, cold-atomic-physics, PDEs to illustrate things.

Not course targets:

- Enable you to use XML or C or C++ beyond what is needed for XMDS.
- Teach you the numerical propagation algorithms employed by XMDS in any detail.
- Teach the physics / science of all the numerical examples

SETUP

- On your computer, right-click desktop “open terminal/shell”
- type “mkdir xmds_course” “cd xmds_course”
- Go to <http://home.iiserb.ac.in/~sebastian> => teaching
- Download course material (examples and these slides) into directory ~/xmds_course
- Check arrival with “ls -lart” (in shell, in that directory)
- Open course notes with “evince xmds2_intro.pdf”

(I) INTRO TO NUMERICAL SOLUTIONS OF PDES

- e.g. Nonlinear Schrödinger equation (NLSE)

$$i \frac{d\psi}{dt} = -\frac{1}{2} \frac{\partial^2}{\partial x^2} \psi - |\psi|^2 \psi$$

- Solution is unknown function of space and time

$$\psi(x, t)$$

- PDEs can come as boundary value problems or initial value problems. XMDS does the latter.

$$\psi(x, t = 0) = \frac{N}{\cosh x}$$

Example is 1st order in time: Initial function specifies all its future.

NUMERICAL DISCRETISATION

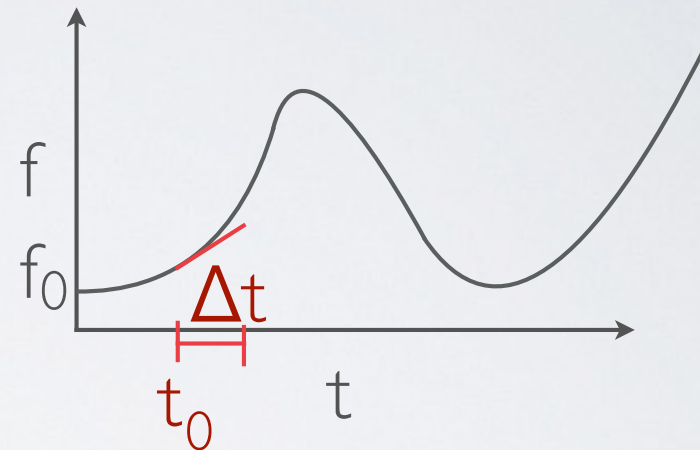
- On computer discretise time and space. Let's look at **time** first (ODE)

- Instead of continuous time, use small discrete time steps Δt

- For each step devise algorithm to find new function. Simple Euler method, see right.

- Time stepper can be much more sophisticated. Order α method:
Error $\sim \Delta t^\alpha$

$$\frac{\partial f(t)}{\partial t} = g(t)f(t)$$
$$f(0) = f_0$$



$$\frac{\Delta f}{\Delta t} = g(t)f(t)$$

$$\frac{f(t + \Delta t) - f(t)}{\Delta t} = g(t)f(t)$$

$$f(t_0 + \Delta t) = f(t_0) + g(t_0)f(t_0)\Delta t$$

NUMERICAL DISCRETISATION

- In a PDE we also must discretise **space**.

$$\frac{\partial f(x, t)}{\partial t} = -\frac{\partial^2}{\partial x^2} f(x, t) + g(x, t)f(x, t)$$

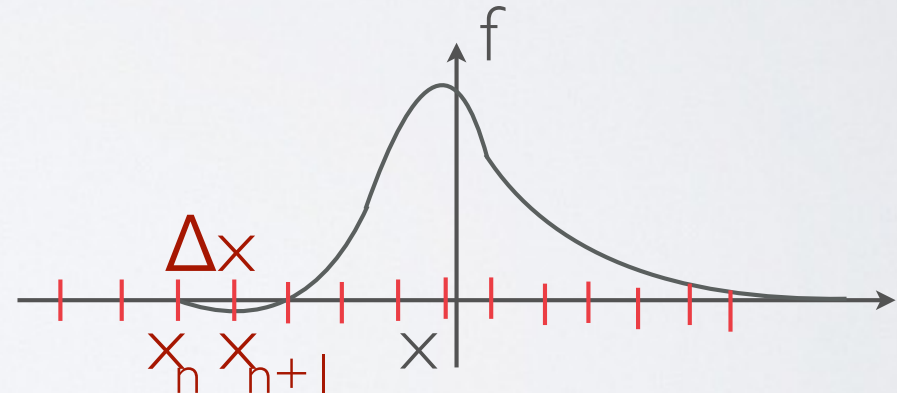
$$f(x, 0) = f_0(x)$$

- Can use same time stepping algorithm as for ODE (e.g. Euler)

$$f(x, t_0 + \Delta t) = f(x, t_0) + \left(\frac{\partial^2}{\partial x^2} f(x, t_0) + g(x, t_0)f(x, t_0) \right) \Delta t$$

- Now also discretise space, to evaluate derivatives

$$\frac{\partial}{\partial x} f(x, t_0) \approx \frac{f(x + \Delta x, t_0) - f(x, t_0)}{\Delta x}$$



- Often better: Fourier transform

$$\frac{\partial}{\partial x} f(x, t_0) = \mathcal{F}^{-1}[ik\mathcal{F}[f](k)]$$

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} dx \tilde{f}(k)e^{ikx}$$

$$\frac{\partial}{\partial x} f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} dx [\tilde{f}(k)(ik)]e^{ikx}$$

(2) XMDS BASICS

simple example (exercise zero)

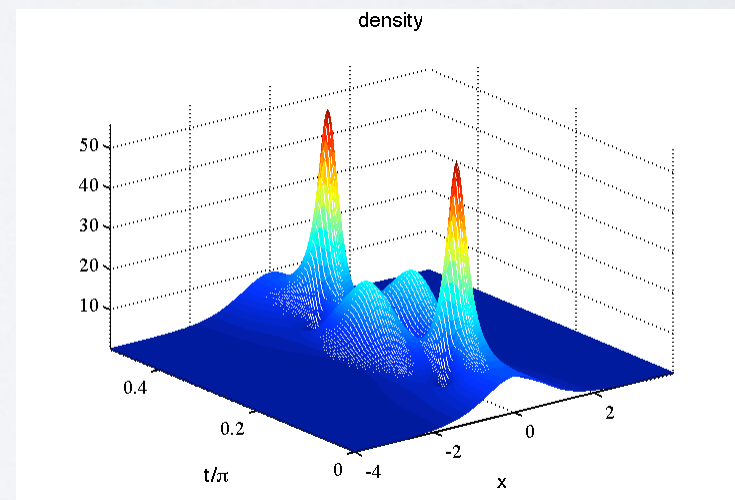
file derived from: [examples/sech_soliton.xmds](#)

- Nonlinear Schrödinger equation (NLSE)
- Solution
- Breathing soliton reforms precisely after period $\pi/2$ (for $N > 1$, $N = 1$ stationary)

$$\psi(x, t = 0) = \frac{N}{\cosh x}$$

$$i \frac{d\psi}{dt} = -\frac{1}{2} \frac{\partial^2}{\partial x^2} \psi - |\psi|^2 \psi$$

$$\psi(x, t)$$



HEADER

```
<simulation xmds-version="2">
  <name>sech_soliton</name>

  <author>Sebastian Wuester / Graham Dennis</author>
  <description>
    Nonlinear Schrodinger equation with attractive interactions.
    This equation has an analytic solution of the form of a breathing soliton.
  </description>

  <geometry>
    <propagation_dimension> t </propagation_dimension>
    <transverse_dimensions>
      <dimension name="x" lattice="4096" domain="(-4.0, 4.0)" />
    </transverse_dimensions>
  </geometry>

  <features>
    <auto_vectorise />
    <fftw />
    <globals>
      <![CDATA[
        const double N = 3.0;
      ]]>
    </globals>
  </features>

  </simulation>
```




$$\psi(x, t = 0) = \frac{N}{\cosh x}$$

- Edit existing XML script
- Header of script:
Name
Geometry (PDE or ODE ?)
- Steering (features, parameters)

INITIALISATION



```
</features>  
  
      dimensions="x"  
<vector name="wavefunction" initial_space="x" type="complex">  
  <components>psi</components>  
  <initialisation>  
    <![CDATA[  
      psi = N/cosh(x);  
    ]]>  
  </initialisation>  
</vector>
```

$$\psi(x, t = 0) = \frac{N}{\cosh x}$$
An arrow originates from the variable 'N' in the equation above and points to the 'psi = N/cosh(x);' line in the code block above.

- `<vector>` tag, defines:
- solution function
- multi-component solution vectors
- parameter type functions appearing in PDE (e.g. potential)

SOLVER

●●●●●
</vector>

```
<sequence>
  <integrate algorithm="ARK45" interval="1.570796327" tolerance="1e-6">
    <samples>200</samples>
    <operators>
      <operator kind="ip">
        <operator_names>L</operator_names>
        <![CDATA[
          L = -0.5*i*kx*kx;
        ]]>
      </operator>
      <integration_vectors>wavefunction</integration_vectors>
      <![CDATA[
        dpsi_dt = L[psi] + i*mod2(psi)*psi;
      ]]>
    </operators>
  </integrate>
</sequence>
```

- All calculations contained in top-level <sequence>
- <integrate> is main calculation/ DE solver block
- <sequence> can contain sub-sequences, these are then loops.
- Top-level sequence can contain also, e.g. <filter>

OUTPUT



```
</sequence>
<output format="hdf5">
  <group>
    <sampling basis="x(512)" initial_sample="yes">
      <moments>density</moments>
      <dependencies>wavefunction</dependencies>
      <![CDATA[
        density = mod2(psi);
      ]]>
    </sampling>
  </group>
  <group>
    <sampling basis="kx(512)" initial_sample="yes">
      <moments>fspec</moments>
      <dependencies>wavefunction</dependencies>
      <![CDATA[
        fspec = mod2(psi);
      ]]>
    </sampling>
  </group>
</output>
</simulation>
```

- write binary, ascii or **hdf5**
- export that data to e.g. matlab
- sample all or some (for large simulations) of accumulated data
- do some processing on output already at code level (e.g. sample Fourier spectrum)
- takes *real part* only

RUNNING

- invoke code generator: `xmds2 sech_soliton.xmds`
- writes and compiles c-code,
filenames taken from <name>
 - `sech_soliton.cc`
 - `sech_soliton`
- run or submit code as usual `./sech_soliton`

READING OUTPUT

```
-rw-r--r-- | sew654 mks    1895 2011-12-27 22:44 sech_soliton_course.xmids
-rw-r--r-- | sew654 mks   66301 2011-12-27 22:44 sech_soliton.cc
-rwxr-xr-x | sew654 mks    81886 2011-12-27 22:44 sech_soliton
-rw-r--r-- | sew654 mks 1673792 2011-12-27 22:44 sech_soliton.h5
-rw-r--r-- | sew654 mks    2853 2011-12-27 22:44 sech_soliton.xsil
```

- After execution we have files:

- h5 contains data.

- xsil contains metadata AND also includes the XMDS script for reference

`xsil2graphics2 sech_soliton.xsil`

→ `sech_soliton.m`

```
-rw-r--r-- | sew654 mks    384 2011-12-27 22:44 sech_soliton.m
```

- generate matlab /octave import script for output

`matlab -nodesktop`
`sech_soliton`

- import data into e.g. matlab /octave

`octave`
`sech_soliton`

MORE ON GEOMETRY

```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="z" lattice="1024" domain="(-100,100)" />
    <dimension name="y" lattice="256" domain="(-10,10)" />
    <dimension name="x" lattice="256" domain="(-10,10)" />
    <dimension name="j" type="integer" lattice="8" domain="(0,7)" aliases="k"/>
  </transverse_dimensions>
</geometry>
```

- Multi-dimensional simulations \Rightarrow more transverse dimensions
- Dimension with most grid-points first, case we ever need MPI
- Can also have discrete dimensions (e.g. internal states of atom)
- Example above:
$$i \frac{d\psi_j(x, y, z)}{dt} = F[\psi_k(x, y, z)]$$
- *(More about integer dimensions in advanced part)*

MORE ON VECTORS

```
</features>
```

```
<vector name="wavefunction" dimensions="x" type="complex">  
  <components>psi</components>  
  <initialisation>  
    <![CDATA[  
      psi = N/cosh(x);  
    ]]>  
  </initialisation>  
</vector>
```

vector with two
different components

```
<vector name="wavefunction" initial_space="kx" type="complex">  
  <components>phi psi</components>  
  <initialisation>  
    <![CDATA[  
      const double delkx = kx - kx0;  
      phi = norm*exp(-0.5*delkx*delkx/sigmak/sigmak);  
      psi = 0.0;  
    ]]>  
  </initialisation>  
</vector>
```

define initial-state
in Fourier-space

dimensions="x"

like, this, compute only
once at beginning

```
<vector name="potentials" dimensions="x" type="real">  
  <components> barrier </components>  
  <evaluation>  
    <![CDATA[  
      barrier = 0.0;  
      if( (x>0)&&(x<1) )  
        barrier = V0;  
    ]]>  
  </evaluation>  
</vector>
```

save memory and
computation time if
data is real

```
<sequence>  
<integrate algorithm="rk4" interval="300" steps="10000">  
  <integration_vectors> wavefunction </integration_vectors>  
  <dependencies> potentials </dependencies>
```

MORE ON VECTORS II

```
</features>
```

```
<vector name="wavefunction" dimensions="x" type="complex">  
  <components>psi</components>  
  <initialisation>  
    <![CDATA[  
      psi = N/cosh(x);  
    ]]>  
  </initialisation>  
</vector>
```

```
<computed_vector name="moments" dimensions="" type="real">  
  <components> norm meanx </components>  
  <evaluation>  
    <dependencies basis="x"> wavefunction </dependencies>  
    <![CDATA[  
      norm = mod2(phi);  
      meanx = x*mod2(phi);  
    ]]>  
  </evaluation>  
</computed_vector>
```

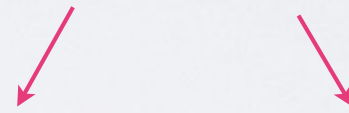
Computed vector:
- NOT stored,
- only calculated when
needed

If #dim dependencies > #dim vector
surplus dimensions are integrated out

MORE ON INTEGRATE

```
</vector>
<sequence>
  <integrate algorithm="ARK45" interval="1.570796327" tolerance="1e-6">
    <samples>200</samples>
    <operators>
      <operator kind="ip">
        <operator_names>L</operator_names>
        <![CDATA[
          L = -0.5*i*kx*kx;
        ]]>
      </operator>
    </operators>
    <integration_vectors>wavefunction</integration_vectors>
    <![CDATA[
      dps_i_dt = L[psi] + i*mod2(psi)*psi;
    ]]>
  </integrate>
</sequence>
```

fixed step algorithm



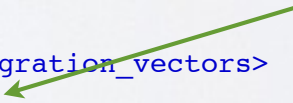
samples for each
output group

explicit derivatives,
MUST use for
products of co-
ordinates and
derivatives

```
<integrate algorithm="rk4" interval="300" steps="10000">
  <samples>200 200 200 200 200</samples>
  <operators>
    <operator kind="ex">
      <operator_names>L Gy Gx</operator_names>
      <![CDATA[
        L = rcomplex(0,-kx*kx/2.0 -ky*ky/2.0);
        Gy = rcomplex(0,Omega*ky);
        Gx = rcomplex(0,Omega*kx);
      ]]>
    </operator>
  </operators>
  <integration_vectors>wavefunction</integration_vectors>
  <dependencies>potentials</dependencies>
  <![CDATA[
    double dens=phi.re*phi.re + phi.im*phi.im;

    dphi_dt = L[phi] + x*Gy[phi] -y*Gx[phi]- i*(U*dens + barrier )*phi - i*kappa*psi;
    dps_i_dt = L[psi] + x*Gy[psi] -y*Gx[psi]- i*(U*dens)*psi - i*kappa*phi;
  ]]>
</integrate>
```

use constant vector



MORE ON ALGORITHMS

- available time-stepping algorithms:

“ark89” adaptive stepsize, 8th/ 9th order Runge-Kutta,

“ark45” adaptive stepsize, 4th/ 5th order Runge-Kutta,

“rk4” fixed stepsize, 4th order Runge-Kutta,

“si” semi-implicit algorithm for stochastic problems

- For PDEs, all these come in the

“ip” (interaction picture) or “ex”

(split-step, for Runge-Kutta)

(explicit) version

$$\frac{\partial}{\partial x} f \rightarrow \mathcal{F}^{-1} [ik\mathcal{F}[f]]$$

$$i \frac{d\psi}{dt} = -\frac{1}{2} \frac{\partial^2}{\partial x^2} \psi + V(x) + U|\psi|^2 \psi$$

(split-step, for Euler scheme)

$$\psi(x, t + \Delta t) \xleftarrow{\mathcal{F}^{-1}} e^{[-\frac{k^2 \Delta t}{4}] \psi(k, t)}$$

$\xleftarrow{\mathcal{F}}$

$$[\psi(x, t) + (V(x) + U|\psi(x, t)|^2 \psi(x, t)) \Delta t]$$

$\xleftarrow{\mathcal{F}^{-1}}$

$$e^{[-\frac{k^2 \Delta t}{4}] \psi(k, t) \xleftarrow{\mathcal{F}} \psi(x, t)}$$

ERROR CHECKING

- Numerical solution, always have to check for convergence

```
<features>  
  <error_check />  
</features>
```

```
.....  
Generating output for GPE_ID_course  
Maximum step error in moment group 1 was 8.905735e-04  
Maximum step error in moment group 2 was 4.287776e-15  
Maximum step error in moment group 3 was 3.622347e-11  
Time elapsed for simulation is: 0.28 seconds
```

```
density_1  
fspec_2  
atomnumber_3  
  
error_density_1  
error_fspec_2  
error_atomnumber_3
```

- Simulation will run once with stepsize $dt = T_{\max}/N_{\text{steps}}$ and once with $dt' = dt/2$. (or for adaptive step-size once with tolerance tol , and once with tolerance $tol/16$ (for fourth order method).

- Little aside: this is an XML comment:

```
<!-- this line is a comment -->
```

(3) EXERCISE ONE

- 1D Gross-Pitaevskii equation (BEC in harmonic trap)

$$i\hbar \frac{d\psi}{dt} = \left[-\frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} + \frac{1}{2} m \omega^2 x^2 + U_{1D} |\psi|^2 \right] \psi$$

- Parameters $\hbar = 1.05457266 \times 10^{-34}$ $U = \frac{4\pi\hbar^2 a_s}{m}$
 $m = 1.4432 \times 10^{-25}$ $U_{1D} = U / (2\pi\sigma_{\perp}^2)$
 $a_s = 5.5 \text{ e-}9$ $\omega_{\perp} = 200 \times (2\pi)$
 $\omega = 10 \times (2\pi)$

- Initial state $\mathcal{N} \exp[-(x - x_0)^2 / 2\sigma^2]$ $\sigma = \sqrt{\frac{\hbar}{m\omega}}$
 $\mathcal{N} = \sqrt{N} / (\pi\sigma^2)^{1/4}$ $N = 100$

- Simulated domain and time $X_{max} = 30\mu m$
 $T_{max} = 0.4s$

EXERCISE ONE

List of tasks, start with
excercise_one_base.xmnds

- Edit simulation: add harmonic trap, use SI units
- Run XMDS, run code, import data into matlab
- Plot some output
- Verify normalisation by sampling $N = \int dx |\psi|^2$
- Determine breathing frequency by sampling

$$\Delta x = \sqrt{\langle x^2 \rangle - \langle x \rangle^2} = \sqrt{\int dx x^2 |\Psi|^2 / N - \left(\int dx x |\Psi|^2 / N \right)^2}$$

(4) XMDS ADVANCED

- XML-shortcuts, MPI-parallelisation, filter, breakpoints, load from file
- Random noise, multi-paths, averaged output, mpi-multi-paths, SDEs
- Command line parameters, discrete dimensions, basis change, convolution
- Exercise II: Kink-bearing ϕ^4 field theory, stochastic simulation

XML SHORTCUTS

from solution to exercise one: GPE_1D_course.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE simulation [
<!ENTITY Npts      "64">
<!ENTITY Nsamples  "64">
<!ENTITY L        "3e-5">
]>
<simulation xmds-version="2">
  <name>GPE_1D_course</name>
```

define XML
"variables"



```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="x" lattice="&Npts;" domain="(-&L;, &L;)" />
  </transverse_dimensions>
</geometry>
```

ensure symmetric
grid



```
<group>
  <sampling basis="x(&Nsamples;)" initial_sample="yes">
    <moments>density phire phiim trappotential interaction_term</moments>
    <dependencies>wavefunction potentials</dependencies>
    <![CDATA[
      density = mod2(phi);
      phire = phi.Re();
      phiim = phi.Im();
      trappotential = trap;
      interaction_term = Uld*mod2(phi);
    ]]>
  </sampling>
</group>
<group>
  <sampling basis="kx(&Npts;)" initial_sample="yes">
    <moments>fspec</moments>
    <dependencies>wavefunction</dependencies>
    <![CDATA[
      fspec = mod2(phi);
    ]]>
  </sampling>
```

this example makes sure we
always sample full Fourier space



PARALLELISATION

```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="z" lattice="1024" domain="(-100,100)" />
    <dimension name="y" lattice="256" domain="(-10,10)" />
    <dimension name="x" lattice="256" domain="(-10,10)" />
    <dimension name="j" type="integer" lattice="8" domain="(0,7)" aliases="k"/>
  </transverse_dimensions>
</geometry>
```

- 2D/3D-multicomponent simulations can benefit from parallelisation

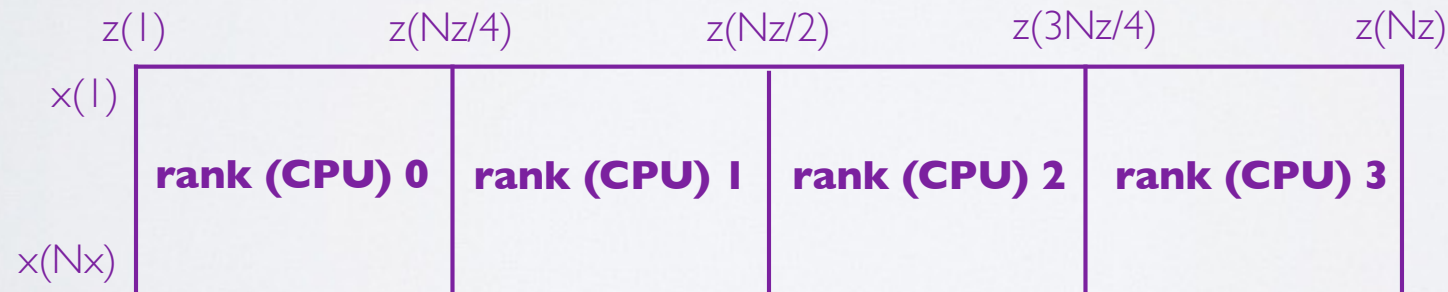
- XMDS parallelises with one line:

```
<driver name="distributed-mpi" />
```

or

```
<features>
  <openmp />
  <globals>
    <![CDATA[
```

- MPI: Slab decomposition along first dimension (that should be the largest)

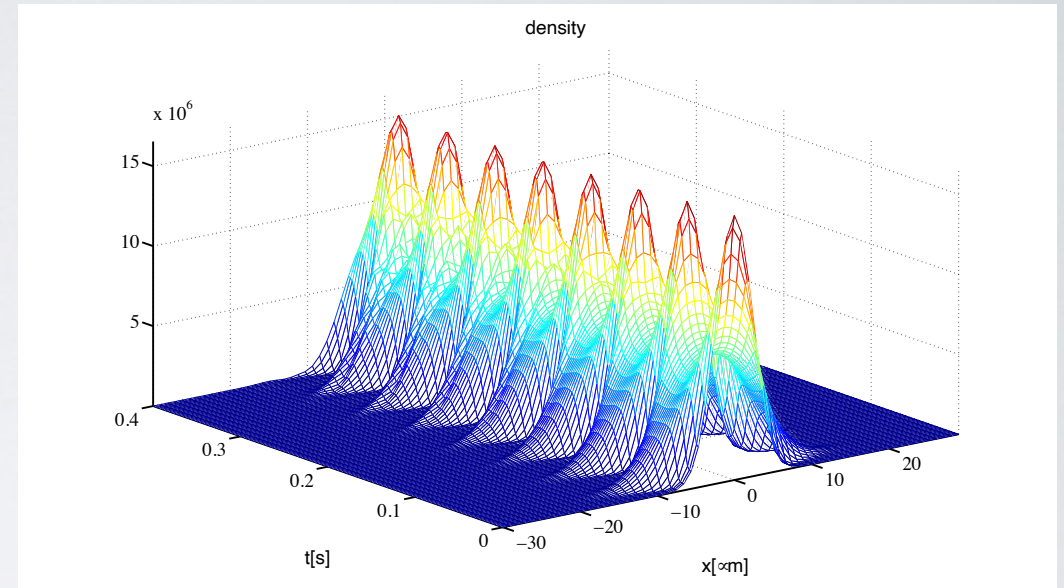


- Will then need mpi jobscripts (see example)

see also http://xmds2.readthedocs.org/en/latest/worked_examples.html#wignerarguments

FILTERS

- Seen breathing oscillations in first example:



- Find Groundstate of: 1D Gross-Pitaevskii equation using **imaginary time evolution**

$$\hbar \frac{d\psi}{d\tau} = - \left[-\frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} + \frac{1}{2} m \omega^2 x^2 + U |\psi|^2 \right] \psi$$

- Need to fix norm of wave function

$$\int dx |\psi|^2 = N$$

FILTERS

see: [GPE_1D_groundstate_course.xmnds](#)

```
<computed_vector name="moments" dimensions="" type="real">  
  <components> norm expecxx expecx </components>  
  <evaluation>  
    <dependencies basis="x"> wavefunction </dependencies>  
    <![CDATA[  
      norm = mod2(psi);  
      expecxx = x*x*mod2(psi);  
      expecx = x*mod2(psi);  
    ]]>  
  </evaluation>  
</computed_vector>
```

filters sub-element
of <integrate>

```
<sequence>  
  <integrate algorithm="RK4" interval="5.0" steps="1000000">  
    <samples>200 200 200</samples>  
    <!-- -->  
    <filters where="step end">  
      <filter>  
        <dependencies>wavefunction moments</dependencies>  
        <![CDATA[  
          // Correct normalisation of the wavefunction  
          psi *= sqrt(Natoms/norm);  
        ]]>  
      </filter>  
    </filters>  
    <!-- -->  
    <operators>  
      <operator constant="yes" kind="ip">  
        <operator_names>L</operator_names>  
        <![CDATA[  
          L = -0.5*hbar*kx*kx/mass;  
        ]]>  
      </operator>  
      <integration_vectors>wavefunction</integration_vectors>  
      <dependencies>potentials</dependencies>  
      <![CDATA[  
        double dens=psi.Re()*psi.Re() + psi.Im()*psi.Im();  
  
        dpsi_dt = L[psi] - (Uld*dens + trap )*psi/hbar;  
      ]]>  
    </operators>  
  </integrate>  
  <!-- -->  
</sequence>
```

calculate norm
using computed
vector

without filter,
exponential decay
to zero

see also http://xmnds2.readthedocs.org/en/latest/worked_examples.html#groundstatebec

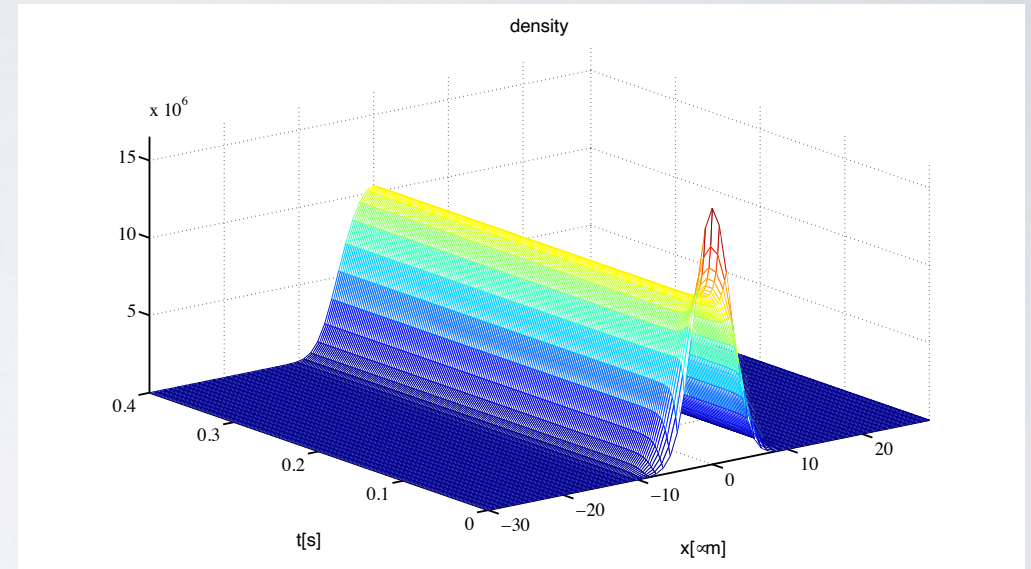
BREAKPOINTS



```
<sequence>
  <integrate algorithm="RK4" interval="5.0" steps="1000000">
    <samples>200 200 200</samples>
    <!-- -->
    <filters where="step end">
      <filter>
        <dependencies>wavefunction moments</dependencies>
        <![CDATA[
          // Correct normalisation of the wavefunction
          psi *= sqrt(Natoms/norm);
        ]]>
      </filter>
    </filters>
    <!-- -->
    <operators>
      <operator constant="yes" kind="ip">
        <operator_names>L</operator_names>
        <![CDATA[
          L = -0.5*hbar*kx*kx/mass;
        ]]>
      </operator>
      <integration_vectors>wavefunction</integration_vectors>
      <dependencies>potentials</dependencies>
      <![CDATA[
        double dens=psi.Re()*psi.Re() + psi.Im()*psi.Im();

        dps_i_dt = L[psi] - (Uld*dens + trap )*psi/hbar;
      ]]>
    </operators>
  </integrate>
  <!-- -->
  <breakpoint filename="groundstate_break" format="hdf5">
    <dependencies basis="x"> wavefunction </dependencies>
  </breakpoint>
  <!-- -->
</sequence>
```

Writes file: groundstate_break.h5



Write chosen
vector to file, for
continuation of
simulation

see also http://xmds2.readthedocs.org/en/latest/worked_examples.html#groundstatebec

LOAD FROM FILE

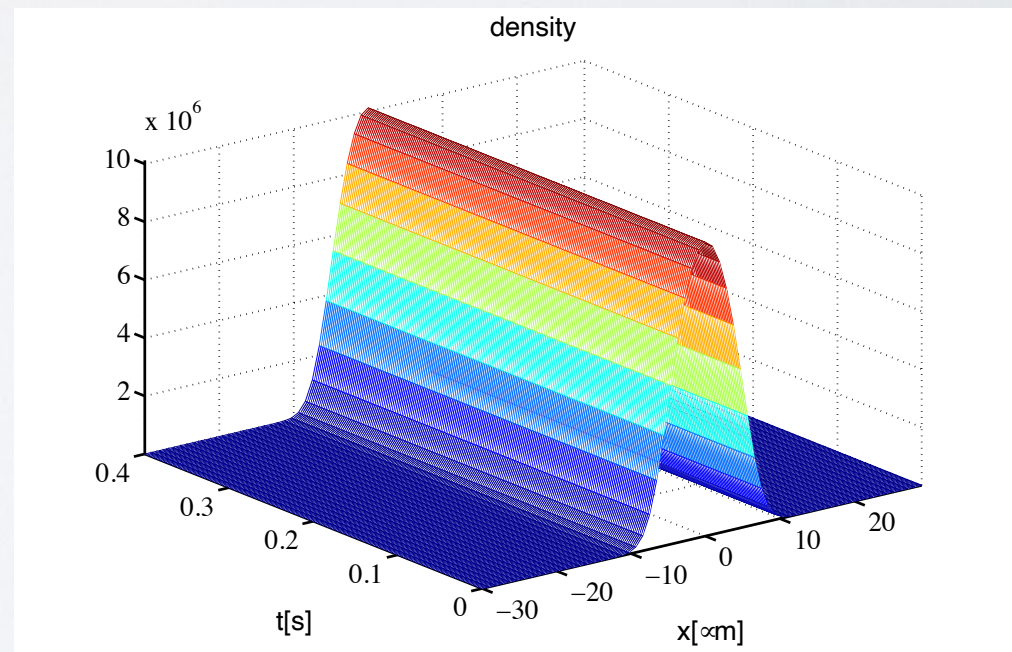
see: GPE_1D_stationary_course.xmds

see: GPE_1D_allinone_course.xmds

- Load into dynamical (real time) simulation
- Confirm stationarity
- Can also load data generated in other ways (matlab, mathematica, text editor)
- Formats: ascii, hdf5, xsil



```
<vector name="wavefunction" initial_space="x" type="complex">  
  <components>psi</components>  
  <initialisation kind="hdf5">  
    <filename> groundstate_break.h5</filename>  
  </initialisation>  
</vector>
```



RANDOM NOISE

see: [GPE_1D_noisy_course.xmls](#)



```
<noise_vector name="thermalNoise" dimensions="x" kind="Gaussian"
  type="complex" method="dsfmt" seed="314 159 276">
  <components>Eta</components>
</noise_vector>
```

Dynamic: Wiener, Jump
Static: Uniform, Gaussian, Poissonian

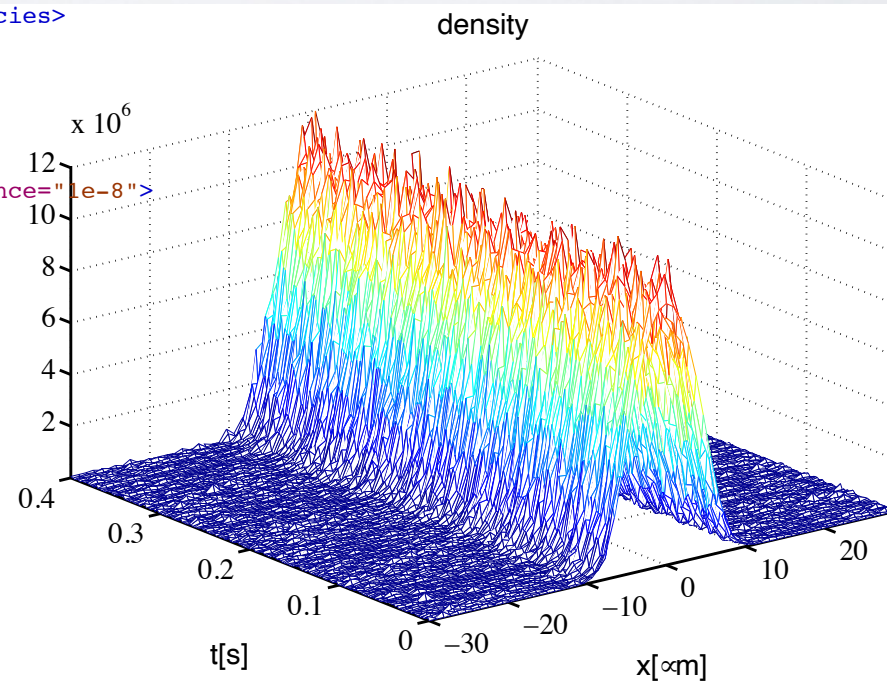
noise generation
algorithm: mkl, dsfmt,
posix(default), solirte

seed: Reproducible
stochastic simulations,
leave empty to have
seed scrambled at
compile time



```
<sequence>
  <!-- -->
  <filter>
    <dependencies>wavefunction thermalNoise</dependencies>
    <![CDATA[
      psi += noiseamp*Eta;
    ]]>
  </filter>
  <!-- -->
  <integrate algorithm="ARK89" interval="0.4" tolerance="1e-8">
    <samples>200 200 200</samples>
    <operators>
```

filter can also be
sub-element of
<sequence>
(only done once)



MULTI-PATH/ AVERAGING

see: GPE_1D_noisy_course_MT.xmds



```
]]>
</globals>
</features>
```

```
<driver name="multi-path" paths="100" />
```

```
<vector name="wavefunction" initial_space="x" type="complex">
  <components>psi</components>
</initialisation kind="hdf5">
```

Run simulation 100 times, automatically calculate averages and stderr



```
<noise_vector name="thermalNoise" dimensions="x" kind="Gaussian"
  type="complex" method="dsfmt" seed="314 159 276">
  <components>Eta</components>
</noise_vector>
```

Seed now valid only initially for all 100 runs



```
</sequence>
```

```
<output format="hdf5">
  <group>
    <sampling basis="x(512)" initial_sample="yes">
      <moments>density</moments>
      <dependencies>wavefunction</dependencies>
      <![CDATA[
        density = mod2(psi);
```

variable A

Output variables: density_l ==> mean_density_l,
stderr_density_l

definition: <A>

$$\sqrt{\frac{\langle AA \rangle - \langle A \rangle \langle A \rangle}{N_{trajs}}}$$

MULTI-PATH/ AVERAGING



```
]]>
</globals>
</features>

<driver name="multi-path" paths="100" />

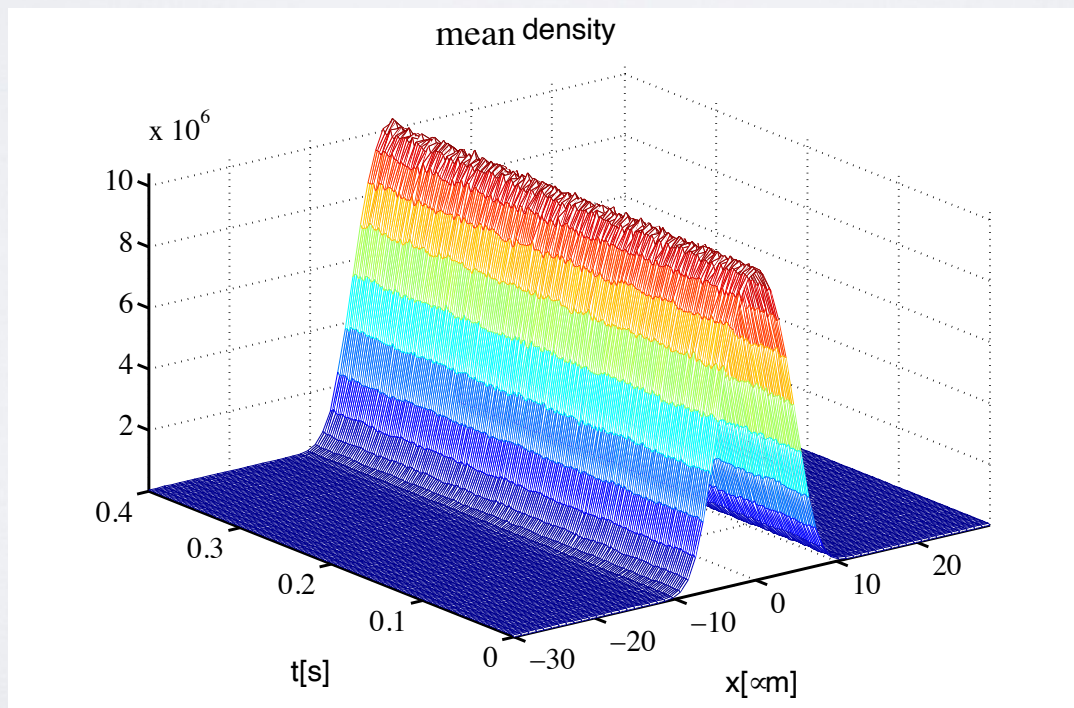
<vector name="wavefunction" initial_space="x" type="complex">
  <components>psi</components>
</initialisation kind="hdf5">
```

Run simulation 100 times, automatically calculate averages and stderr



Output variables: **A** density_1 ==> mean_density_1, stderr_density_1

$$\langle A \rangle$$
$$\text{Sqrt}[(\langle AA \rangle - \langle A \rangle^2) / N_{\text{paths}}]$$



PARALLEL MULTI-PATH

```
<driver name="multi-path" paths="1000" />
```

Only small addition: Go from small sample to large sample, parallel processed

```
<driver name="mpi-multi-path" paths="640" />
```

●●●●●

```
Rank[44]: Starting path 109  
Rank[46]: Starting path 111  
Rank[35]: Starting path 100  
Rank[59]: Starting path 124  
Rank[36]: Starting path 101  
Rank[1]: Starting path 66  
Rank[30]: Starting path 95  
Rank[50]: Starting path 115  
Rank[15]: Starting path 80  
Rank[57]: Starting path 122  
Rank[45]: Starting path 110  
Rank[51]: Starting path 116  
Rank[40]: Starting path 105  
Rank[23]: Starting path 88
```

●●●●●

Npaths, should be multiple of Ncpus

STOCHASTIC DE

- Langevin equation for brownian motion: [see: brownian_motion_course.xmds](#)

$$m \frac{\partial^2 x}{\partial t^2} = -\lambda \frac{\partial x}{\partial t} + \eta(t)$$

$$\langle \eta(t) \eta(t') \rangle = 2\lambda T \delta(t - t')$$

```
const double noiseamp = sqrt(2.0*damping*Temperature);
```

```
.....
```

```
<driver name="multi-path" paths="10000" />
```

```
.....
```

```
<noise_vector name="drivingNoise" kind="Wiener" type="real" method="dsfmt" seed="23 42 1"
```

```
<components>dw</components>
```

```
</noise_vector>
```

```
.....
```

```
<sequence>
```

```
<integrate algorithm="SI" interval="3" steps="10000">
```

```
<samples>200 200 200</samples>
```

```
<operators>
```

```
<integration_vectors>variables </integration_vectors>
```

```
<dependencies>drivingNoise</dependencies>
```

```
<![CDATA[
```

```
    dpos_dt = vel;
```

```
    dvel_dt = -damping/mass*vel + noiseamp/mass*dW;
```

```
]]>
```

```
</operators>
```

```
</integrate>
```

```
</sequence>
```

Dynamical noise on each timestep

Algorithms converge against Stratonovich integral, for solving Ito SDE have to convert!!!

see also http://xmds2.readthedocs.org/en/latest/worked_examples.html#kubo

DISTRIBUTION SAMPLING

see: [brownian_motion_course_MT.xmls](#)

- Suppose we want to sample the position and velocity distribution functions $p(x)$, $p(v)$ [1D functions] of the Langevin equation [0D]:

$$m \frac{\partial^2 x}{\partial t^2} = -\lambda \frac{\partial x}{\partial t} + \eta(t) \quad \langle \eta(t)\eta(t') \rangle = 2\lambda T \delta(t - t')$$



```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="xbin" lattice="&NptsBin;" domain="(-&Xmax;,&Xmax;)" />
    <dimension name="vbin" lattice="&NptsBin;" domain="(-&Vmax;,&Vmax;)" />
  </transverse_dimensions>
</geometry>
```



```
<vector name="variables" type="real" dimensions="">
  <components> pos vel </components>
  <initialisation>
    <![CDATA[
      pos = 0.0;
      vel = 0.0;
    ]]>
  </initialisation>
</vector>
```

New: tell vectors in which dimensions they live (defaults to ALL dimensions)

```
<group>
  <sampling basis="" initial_sample="yes">
    <moments>position velocity</moments>
    <dependencies>variables</dependencies>
    <![CDATA[
      position = pos;
      velocity = vel;
    ]]>
  </sampling>
</group>
```

Dimensions for sampling only

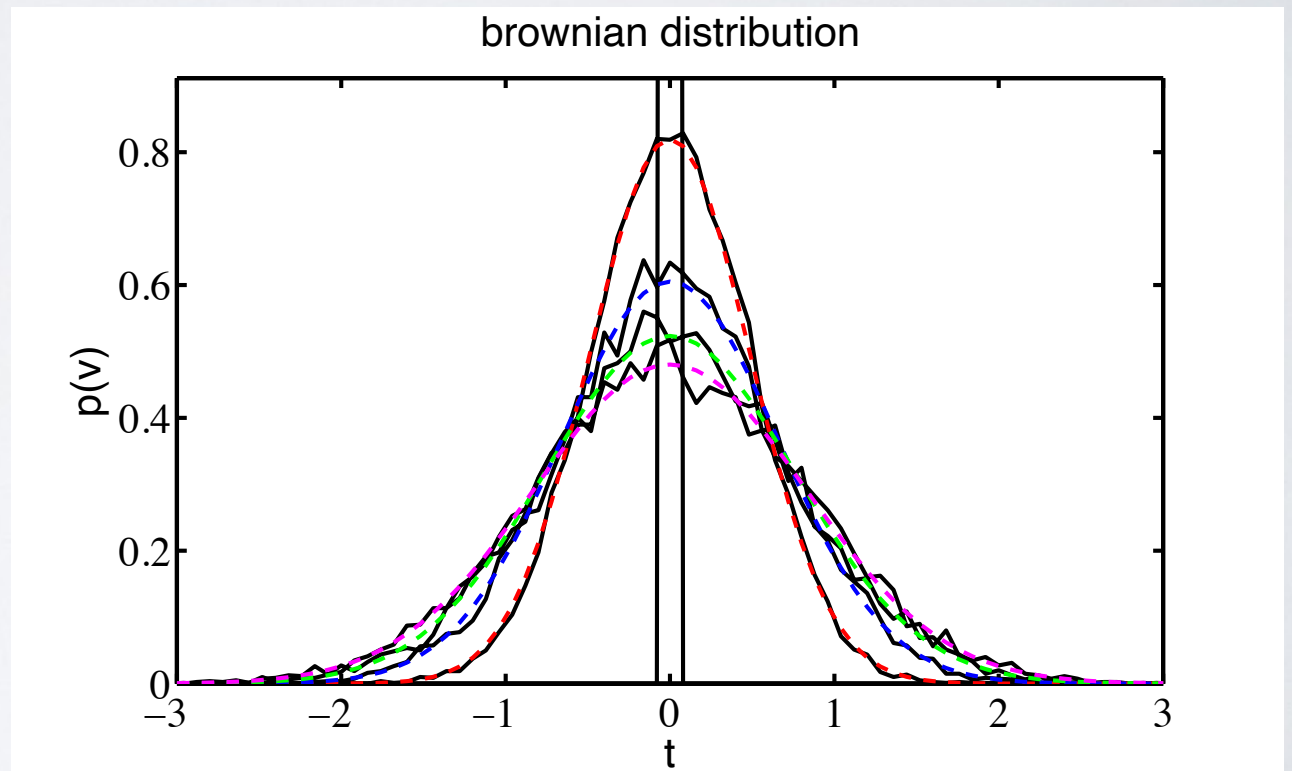
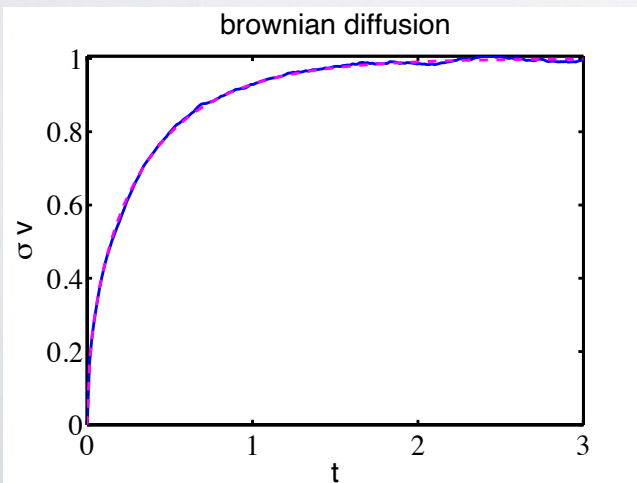
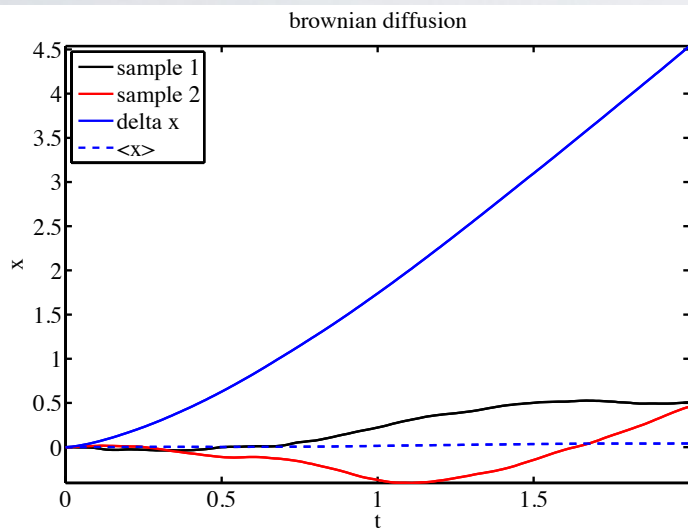
hacking, see last section

```
<group>
  <sampling basis="xbin" initial_sample="yes">
    <moments> pos_distribution </moments>
    <dependencies>variables</dependencies>
    <![CDATA[
      if(_index_xbin==0 && pos < (xbin+dxbin_halves))
        pos_distribution +=1.0;
      if( pos > (xbin-dxbin_halves) && pos < (xbin+dxbin_halves) )
        pos_distribution +=1.0;
      if(_index_xbin==_lattice_xbin-1 && pos > (xbin-dxbin_halves) )
        pos_distribution +=1.0;
    ]]>
  </sampling>
</group>
```

STOCHASTIC DE

- Langevin equation for brownian motion:

$$m \frac{\partial^2 x}{\partial t^2} = -\lambda \frac{\partial x}{\partial t} + \eta(t) \quad \langle \eta(t)\eta(t') \rangle = 2\lambda T \delta(t - t')$$



COMMAND LINE PARAMETERS

see: [GPE_1D_commandline_course.xml](#)

```
<features>
  <benchmark />
  <auto_vectorise />
  <fftw />
  <globals>
    <![CDATA[
      const double hbar = 1.05457266e-34;
      double omega = 10.0*(2.0*M_PI); // to be modified by command line parameters
      const double omega_perp = 200.0*(2.0*M_PI);
      const double x0 = 0.0e-5;

      // Rb 87
      const double mass = 1.4432e-25;
      const double as = 5.5e-9;

      //derived quantities
      const double Natoms = 100.0;

      const double U = 4.0*M_PI*hbar*hbar*as/mass;
      const double sigma_perp = sqrt(hbar/mass/omega_perp);
      const double U1d = U/(2.0*M_PI*sigma_perp*sigma_perp);

      double normfact = 0.0; // get set after command line parameters are known
      double sigma = 0.0;
    ]]>
  </globals>
  <arguments>
    <argument name="trap_scale" type="real" default_value="1.0"/>
    <argument name="width_scale" type="real" default_value="1.0"/>
    <![CDATA[
      omega *= sqrt(trap_scale);
      sigma = width_scale*sqrt(hbar/mass/omega);
      normfact = pow(M_PI*sigma*sigma,-0.25);
    ]]>
  </arguments>
</features>
```

Variables affected by command line parameters must be defined modifyable

Globals block evaluated BEFORE querying run time parameters

to pass on runtime

actions done AFTER querying run time parameters

Execution: `./GPE_1D_commandline_course -t 15.0 -w 0.5`

Execution: `./GPE_1D_commandline_course -trap_scale 15.0`

see also http://xmds2.readthedocs.org/en/latest/worked_examples.html#wignerarguments

DISCRETE DIMENSIONS

- 1D Gross-Pitaevskii equation for **spinor BEC** (spin-1 atoms)

$$i\hbar\partial_t\Psi_{\pm 1} = [\mathcal{H}_s + c_2(n_{\pm 1} + n_0 - n_{\mp 1})]\Psi_{\pm 1} + c_2\Psi_0^2\Psi_{\mp 1}^*$$

$$i\hbar\partial_t\Psi_0 = [\mathcal{H}_s + c_2(n_1 + n_{-1})]\Psi_0 + c_22\Psi_1\Psi_0^*\Psi_{-1}, \quad (1)$$

taken from B. Julia-Diaz *et al.*
PRA **80** 043622 (2009).

where, $\mathcal{H}_s = -\frac{\hbar^2}{2M}\nabla^2 + V + c_0n$, $n_m(\vec{r}, t) = |\Psi_m(\vec{r}, t)|^2$, $n(\vec{r}, t) = \sum_m n_m(\vec{r}, t)$, and $m=0, \pm 1$. The population of each hyper-

- Wave function has subscript for spin state (mF=-1,0,1)

```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="x" lattice="&Npts;" domain="(-&L;, &L;)" />
    <dimension name="n" type="integer" lattice="3" domain="(0,2)" aliases="k"/>
    <!-- Assignment of components
      0: mF= -1
      1: mF=  0
      2: mF= +1 -->
  </transverse_dimensions>
</geometry>
```

integer dimension for spin state

XML comment

for constructions like:

$$\Psi_n = \sum_k C_{nk} \Psi_k$$

see also http://xmds2.readthedocs.org/en/latest/worked_examples.html#integerdimensionexample

DISCRETE DIMENSIONS

see: [GPE_1D_spinor_course.xmnds](#)

```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="x" lattice="&Npts;" domain="(-&L;, &L;)" />
    <dimension name="n" type="integer" lattice="3" domain="(0,2)" aliases="k"/>
    <!-- Assignment of components
      0: mF= -1
      1: mF=  0
      2: mF= +1 -->
  </transverse_dimensions>
</geometry>
```



```
<vector name="wavefunction" initial_space="x n" type="complex">
  <components>psi</components>
  <initialisation>
    <![CDATA[
      double coeff = 0.0;

      if(n == 0)
        coeff = sqrt(0.1);
      if(n == 1)
        coeff = sqrt(0.8);
      if(n == 2)
        coeff = sqrt(0.1);

      psi = normfact*coeff*sqrt(Natoms)*exp(-0.5*x*x/sigma/sigma);
    ]]>
  </initialisation>
</vector>
```

space and spin dependent Psi

see also http://xmnds2.readthedocs.org/en/latest/worked_examples.html#wignerarguments

DISCRETE DIMENSIONS

```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="x" lattice="&Npts;" domain="(-&L;, &L;)" />
    <dimension name="n" type="integer" lattice="3" domain="(0,2)" aliases="k"/>
    <!-- Assignment of components
      0: mF= -1
      1: mF=  0
      2: mF= +1 -->
  </transverse_dimensions>
</geometry>
```



```
<computed_vector name="auxiliary" dimensions="x" type="real">
  <components> totdens </components>
  <evaluation>
    <dependencies basis="x n"> wavefunction </dependencies>
    <![CDATA[
      totdens = mod2(psi);
    ]]>
  </evaluation>
</computed_vector>
```

again, excess dimensions are integrated (here summed) over

- calculation of: $n(\vec{r}, t) = \sum_m n_m(\vec{r}, t)$ $n_m(\vec{r}, t) = |\Psi_m(\vec{r}, t)|^2$

DISCRETE DIMENSIONS

```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="x" lattice="&Npts;" domain="(-&L;, &L;)" />
    <dimension name="n" type="integer" lattice="3" domain="(0,2)" aliases="k"/>
    <!-- Assignment of components
      0: mF= -1
      1: mF=  0
      2: mF= +1 -->
  </transverse_dimensions>
</geometry>
```

matrix in spin space

```
<vector name="coupling" type="real" dimensions="n k">
  <components> F1 </components>
  <initialisation>
    <![CDATA[
      F1 = 0.0;

      // dpsi(-1)/dt
      if(n==0 && (k==0 || k==1))
        F1 = 1.0;
      if(n==0 && k ==2)
        F1 = -1.0;

      // dpsi(0)/dt
      if(n==1 && (k==0 || k==2))
        F1 = 1.0;

      // dpsi(+1)/dt
      if(n==2 && (k==1 || k==2))
        F1 = 1.0;
      if(n==2 && k ==0)
        F1 = -1.0;

    ]]>
  </initialisation>
</vector>
```

```
<computed_vector name="auxiliary2" dimensions="x n" type="real">
  <components> asymmdens </components>
  <evaluation>
    <dependencies basis="x n k"> wavefunction coupling</dependencies>
    <![CDATA[
      asymmdens = F1*mod2(psi( n=>k ));
    ]]>
  </evaluation>
</computed_vector>
```

again, excess dimensions are
integrated (here summed) over;
note use of alias

- calculation of:
$$i\hbar\partial_t\Psi_{\pm 1} = \dots (n_{\pm 1} + n_0 - n_{\mp 1})$$

$$i\hbar\partial_t\Psi_0 = \dots (n_1 + n_{-1})$$

DISCRETE DIMENSIONS

```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="x" lattice="&Npts;" domain="(-&L;, &L;)" />
    <dimension name="n" type="integer" lattice="3" domain="(0,2)" aliases="k"/>
    <!-- Assignment of components
      0: mF= -1
      1: mF=  0
      2: mF= +1 -->
  </transverse_dimensions>
</geometry>
```

```
<sequence>
  <integrate algorithm="ARK89" interval="4" tolerance="1e-8">
    <samples>200 200 200</samples>
    <operators>
      <operator kind="ip">
        <operator_names>L</operator_names>
        <![CDATA[
          L = -i*0.5*hbar*kx*kx/mass;
        ]]>
      </operator>
      <integration_vectors>wavefunction</integration_vectors>
      <dependencies>potentials auxiliary auxiliary2</dependencies>
      <![CDATA[
        complex conversion = 0.0;

        if(n == 0)
          conversion = conj(psi(n=>2))*psi(n=>1)*psi(n=>1);
        if(n == 1)
          conversion = 2.0*conj(psi)*psi(n=>n+1)*psi(n=>n-1);
        if(n == 2)
          conversion = conj(psi(n=>0))*psi(n=>1)*psi(n=>1);

        dpsi_dt = L[psi] - i*( (Ulds*totdens + trap )*psi
          + Ulda*(asymmtdens*psi + conversion) )/hbar;

      ]]>
    </operators>
  </integrate>
</sequence>
```

Final assembly

Non local addressing,
would have been clearer for this
example, but not for cases with
LARGE number of n

DISCRETE DIMENSIONS

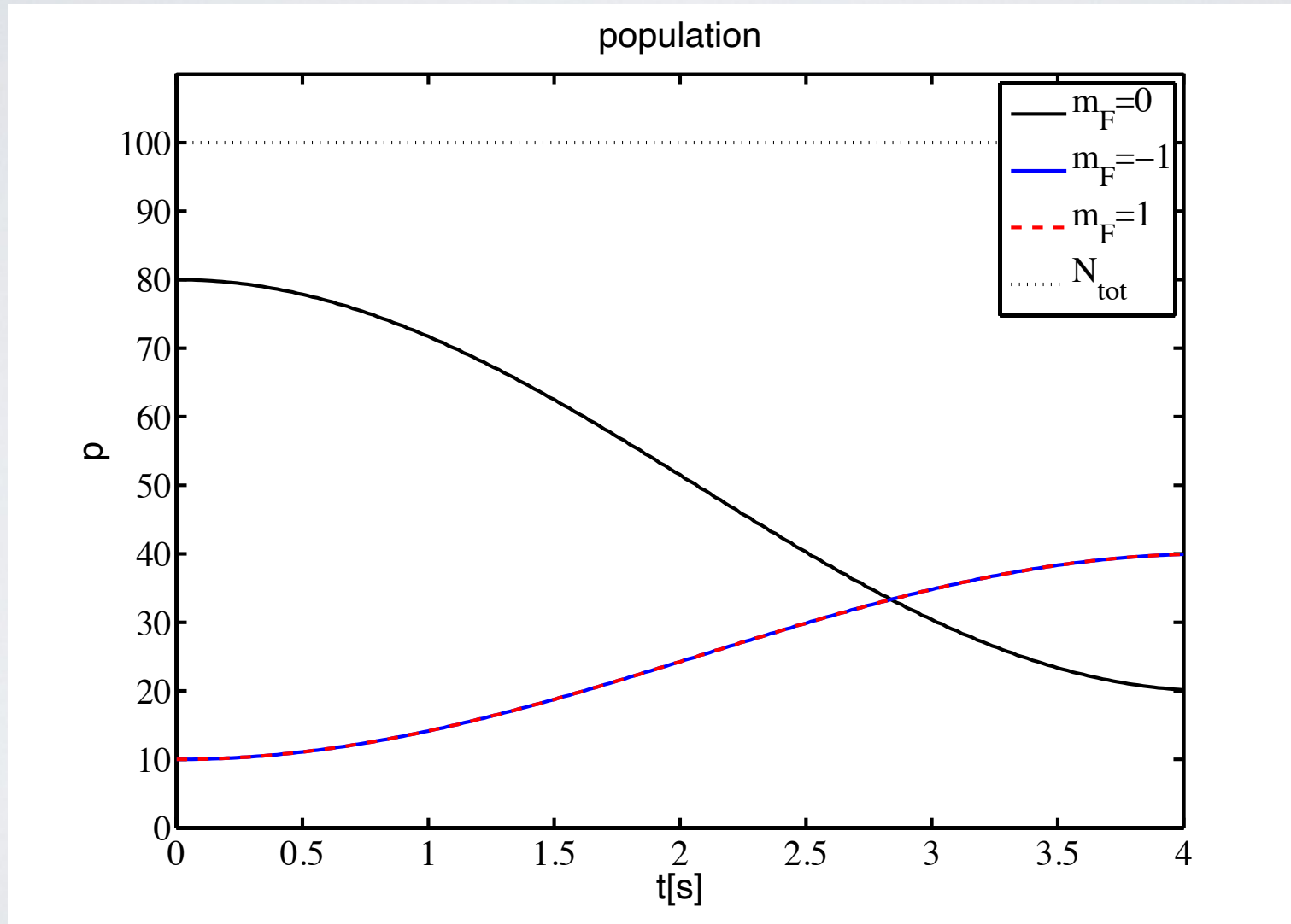
```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="x" lattice="&Npts;" domain="(-&L;, &L;)" />
    <dimension name="n" type="integer" lattice="3" domain="(0,2)" aliases="k"/>
    <!-- Assignment of components
      0: mF= -1
      1: mF=  0
      2: mF= +1 -->
  </transverse_dimensions>
</geometry>
```

```
<output format="hdf5">
  <group>
    <sampling basis="x(&Nsamples;) n" initial_sample="yes">
      <moments>density psire psiim </moments>
      <dependencies>wavefunction </dependencies>
      <![CDATA[
        density = mod2(psi);
        psire = psi.Re();
        psiim = psi.Im();
      ]]>
    </sampling>
  </group>
```

Output now samples separately
each spin component

Name	Size	Bytes	Class
density_l	201x64x3	308736	double

SPIN JOSEPHSON OSCILLATIONS



OSCILLATOR BASIS

- Harmonic oscillator Schroedinger equation, spectral solution

$$i\hbar \frac{d\psi}{dt} = \left[-\frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} + \frac{1}{2} m\omega^2 x^2 \right] \psi$$

- Eigenstates of TISE

$$E_n \phi_n = \left[-\frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} + \frac{1}{2} m\omega^2 x^2 \right] \phi_n$$

- Spectral decomposition

$$\psi(x, t) = \sum_n c_n(t) \phi_n(x)$$

- TDSE in oscillator basis

$$i\hbar \frac{dc_n}{dt} = \hbar\omega \left(n + \frac{1}{2} \right) c_n$$

OSCILLATOR BASIS

see: [GPE_1D_hermite_course.xmfs](#)

- GPE in oscillator basis/ pos basis, split step

$$i\hbar \frac{d\psi}{dt} = \left[-\frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} + \frac{1}{2} m \omega^2 x^2 + U |\psi|^2 \right] \psi$$

$$i\hbar \frac{dc_n}{dt} = \hbar \omega \left(n + \frac{1}{2} \right) c_n$$

$$U |\psi(x)|^2 \psi(x)$$

```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="x" lattice="&Nmodes;"
      length_scale="sqrt(hbar/(mass*omega))" transform="hermite-gauss" />
  </transverse_dimensions>
</geometry>
```



```
<features>
  <benchmark />
  <auto_vectorise />
  <validation kind="run-time" />
```

we explicitly give transform tag,
would default to "fourier"

Length scale: to get to
dimensionless hermites

Need this to use variables in
definition of length scale

see also http://xmfs2.readthedocs.org/en/latest/worked_examples.html#hermitegaussgroundstatebec

OSCILLATOR BASIS

```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="x" lattice="&Nmodes;"
      length_scale="sqrt(hbar/(mass*omega))" transform="hermite-gauss" />
  </transverse_dimensions>
</geometry>
```



```
<vector name="wavefunction" initial_space="x" type="complex">
  <components>phi</components>
  <initialisation>
    <![CDATA[
      const double delx = x - x0;
      phi = normfact*sqrt(Natoms)*exp(-0.5*delx*delx/sigma/sigma);
    ]]>
  </initialisation>
</vector>
```

```
<vector name="wavefunction" initial_space="nx" type="complex">
  <components>psi</components>
  <initialisation>
    <![CDATA[
      psi = 0.0;
      if(nx==0)
        psi = sqrt(Natoms);
    ]]>
  </initialisation>
</vector>
```



Now we can initialise in the energy basis. These two are equivalent for $x_0=0$.

OSCILLATOR BASIS

```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="x" lattice="&Nmodes;"
      length_scale="sqrt(hbar/(mass*omega))" transform="hermite-gauss" />
  </transverse_dimensions>
</geometry>
```



```
<sequence>
  <integrate algorithm="ARK89" interval="0.4" tolerance="1e-8">
    <samples>200 200 200 200</samples>
    <operators>
      <operator kind="ip" constant="yes">
        <operator_names>L</operator_names>
        <![CDATA[
          L = -i*(nx + 0.5)*omega;
        ]]>
      </operator>
      <integration_vectors>wavefunction</integration_vectors>
      <![CDATA[
        double dens=psi.Re()*psi.Re() + psi.Im()*psi.Im();

        dpsi_dt = L[psi] - i*Uld*dens*psi/hbar;
      ]]>
    </operators>
  </integrate>
</sequence>
```

Transform for application of IP operator is hermite-gauss

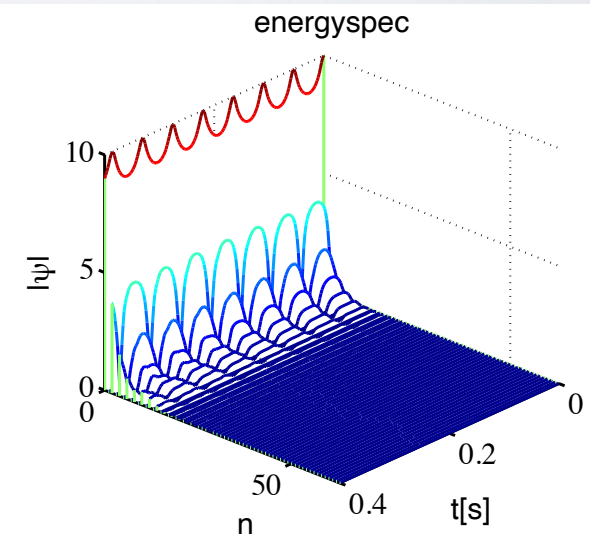
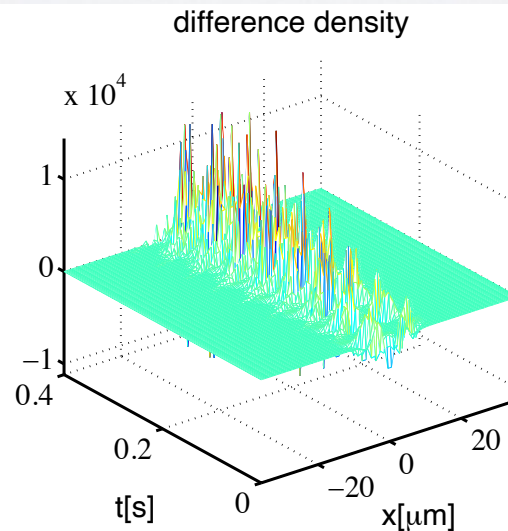
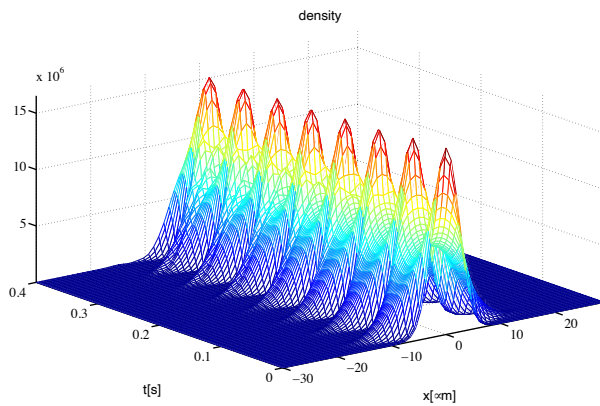
Nonlinear term evaluated in pos-space, in energy space would be nasty triple sum

BASIS CHANGE

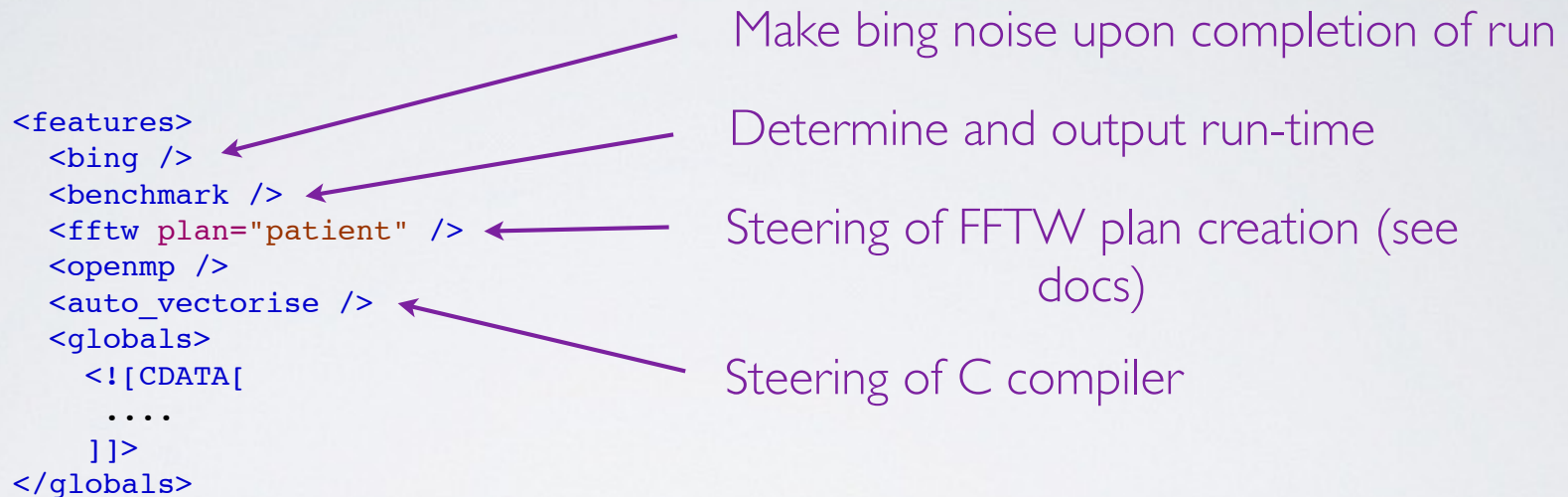
```
<group>
  <sampling basis="nx(&Nmodes;)" initial_sample="yes">
    <moments>energyspec</moments>
    <dependencies>wavefunction</dependencies>
    <![CDATA[
      energyspec = mod2(psi);
    ]]>
  </sampling>
</group>
```

- Can sample in oscillator basis

- Compare breathing in oscillator and position basises



ASSORTED FURTHER TAGS



`<diagnostics />` `<!-- This will make sure that all nonlocal accesses of dimensions are safe -->`

ASSORTED WARNINGS



- If a tag has a non-existent name, it is just ignored. If in doubt, verify C code for existence of functionality.
- Fourier transforms have non-standard phases, see doc: http://xmds2.readthedocs.org/en/latest/advanced_topics.html#convolutions
- ...

(5) EXERCISE TWO

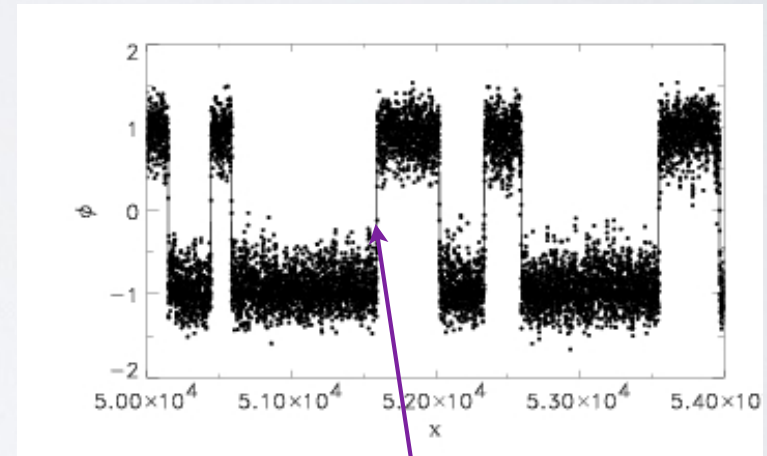
- Kink bearing ϕ^4 classical field theory with noise

- taken from: S. Habib and G. Lythe, Phys. Rev. Lett. **84** 1070 (2000)
<http://www.l.maths.leeds.ac.uk/~grant/research.html>

- Equation of motion

$$\partial_{tt}^2 \phi = \partial_{xx}^2 \phi + \phi(1 - \phi^2) - \eta \partial_t \phi + \xi(x, t)$$

$$\langle \xi(x, t) \xi(x', t') \rangle = 2\eta\beta^{-1} \delta(x - x') \delta(t - t')$$



- Simulation parameters:

$\eta = 1$, $\beta = 7$, $N_{pts} = 256$, $X_{max} = 100$, $T_{max} = 5e4$.

(6) XMDS HACKS

```
<geometry>
  <propagation_dimension> t </propagation_dimension>
  <transverse_dimensions>
    <dimension name="x" lattice="4096" domain="(-4.0, 4.0)" />
  </transverse_dimensions>
</geometry>
```

- Useful internal variables:

```
_max_x=4,
_min_x=-4,
_lattice_x=4096
```

```
<group>
  <sampling basis="x(512)" initial_sample="yes">
    <moments>density</moments>
    <dependencies>wavefunction</dependencies>
    <![CDATA[
      density = mod2(psi);
    ]>
    _mg0_output_lattice_t
    _mg0_output_index_t
```

```
</features>
```

```
<vector name="wavefunction" initial_space="x" type="complex">
  <components>psi</components>
  <initialisation>
    <![CDATA[
      psi = N/cosh(x);
    ]>
  </initialisation>
</vector>
```

```
_index_x
```

```
_x_wavefunction_ncomponents
```

```
_active_x_wavefunction[_lattice_x]
```

- If you know SOME C, don't hesitate to look at the written .cc file. Search for your xmids CDATA text....

XMDS HACKS

see: [GPE_1D_hacks_course.xmids](#)

- Knowing the internal structure of the .cc code, we can add most things we would want to our XMDS code:

```
<features>
  <benchmark />
  <auto_vectorise />
  <fftw />
  <globals>
    <![CDATA[
      #include "gsl_sf_hyperg.h"

      ●●●●●
      //%%%%%%%%%%%%%%%%%%%%%%%%
      // FUNCTION LIBRARY

      void calculate_useless_hypergeometric_functions(){

          gsl_sf_result result;

          int status=gsl_sf_hyperg_1F1_e(t,1.5,0.3,&result);

          hypergeos[counter] = result.val;
          counter++;

      }
      //%%%%%%%%%%%%%%%%%%%%%%%%

      void display_useless_hypergeometric_functions(){

          for(int nn=0;nn<_mg0_output_lattice_t;nn++)
              printf("Hyper[%i] = %e\n",nn,hypergeos[nn]);

      }

    ]]>
  </globals>
</features>
```

Bind in extra libraries

xmids2 --configure --include-path \$GSL_HOME/
include/gsl --lib-path \$GSL_HOME/lib

xmids2 GPE_1D_hacks_course.xmids -g

debug switch, outputs compiler
statement

XMDS HACKS

- Knowing the internal structure of the .cc code, we can add most things we would want to our XMDS code:

```
<sequence>
  <integrate algorithm="ARK89" interval="0.4" tolerance="1e-8">
    ●●●●●
  </integrate>
  <filter>
    <![CDATA[
      display_useless_hypergeometric_functions();
      write_extra_output();
    ]]>
  </filter>
</sequence>
```

Write non-standard sampling at end

```
●●●●●

<group>
  <sampling basis="x(&Nsamples;)" initial_sample="yes">
    <moments>density psire psiim </moments>
    <dependencies>wavefunction </dependencies>
    <![CDATA[
      density = mod2(psi);
      psire = psi.Re();
      psiim = psi.Im();

      if(_index_x==0)
        calculate_useless_hypergeometric_functions();
    ]]>
  </sampling>
</group>
```

Sample whenever we sample `_mg0`

XMDS HACKS

- Knowing the internal structure of the .cc code, we can add most things we would want to our XMDS code:

```
<globals>
<![CDATA[

    ●●●●●

    //%%%%%%%%%%
    //% output
    unsigned long counter=0;
    double hypergeos[_mg0_output_lattice_t];

    const char *outfilename="GPE_1D_hacks_course_additional_output.data";
    fstream outfile;

    ●●●●●

    //%%%%%%%%%%

    void write_extra_output(){
        printf("Writing additional output to: %s \n",outfilename);
        outfile.open(outfilename,ios::out|ios::binary);
        outfile.write((char*)hypergeos,sizeof(double)*_mg0_output_lattice_t);

        if(outfile.bad()){
            printf("Error writing MGs to file, wrote %i bytes.\n",outfile.gcount());
            exit(42);
        }

        outfile.close();

    }

    ●●●●●

]]>
</globals>
```

Perform file I/O
as in usual C



ACKNOWLEDGE XMDS

- If you publish something done mainly with XMDS, please place a reference:

were $\Delta\tau=10^{-5}$ s. The simulations were done with the aid of the high level programming language XMDS [29][57]

[29] Online at www.xmids.org.

[57] G. R. Dennis, J. J. Hope, and M. T. Johnsson, *Comput. Phys. Comm.* **184**, 201 (2013).

- Maybe at some point submit your code as example to the XMDS webpage

IMPROVE XMDS

- If you need a new feature (basis, sampling etc.) consider adding it to XMDS itself, sign up at:

<http://sourceforge.net/projects/xmids/>

XMDS2 BROUGHT TO YOU BY:

Graham Dennis

Joe Hope

Mattias Johnsson

Michael Hush

BASED ON XMDS1 BY:

Greg Collecot

Joe Hope

Clinton Roy

Paul Cochrane

Michael East

et al.

Thanks for your attention