Computer Physics Communications 184 (2013) 201–208

Contents lists available at SciVerse ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

XMDS2: Fast, scalable simulation of coupled stochastic partial differential equations☆

Graham R. Dennis *, Joseph J. Hope, Mattias T. Johnsson

Research School of Physics and Engineering, The Australian National University, Canberra, ACT 0200, Australia

ARTICLE INFO

Article history:
Received 19 April 2012
Received in revised form
17 July 2012
Accepted 21 August 2012
Available online 25 August 2012

Keywords:
Initial value problems
Differential equations
Numerical integration
Stochastic partial differential equations

ABSTRACT

XMDS2 is a cross-platform, GPL-licensed, open source package for numerically integrating initial value problems that range from a single ordinary differential equation up to systems of coupled stochastic partial differential equations. The equations are described in a high-level XML-based script, and the package generates low-level optionally parallelised C++ code for the efficient solution of those equations. It combines the advantages of high-level simulations, namely fast and low-error development, with the speed, portability and scalability of hand-written code. XMDS2 is a complete redesign of the XMDS package, and features support for a much wider problem space while also producing faster code.

Program summary

Program title: XMDS2
Catalogue identifier: AENK_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AENK_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: GNU General Public License, version 2
No. of lines in distributed program, including test data, etc.: 872490
No. of bytes in distributed program, including test data, etc.: 45522370
Distribution format: tar.gz
Programming language: Python and C++.
Computer: Any computer with a Unix-like system, a C++ compiler and Python.
Operating system: Any Unix-like system; developed under Mac OS X and GNU/Linux.
RAM: Problem dependent (roughly 50 bytes per grid point)
Classification: 4.3, 6.5.

External routines: The external libraries required are problem-dependent. Uses FFTW3 Fourier transforms (used only for FFT-based spectral methods), dSFMT random number generation (used only for stochastic problems), MPI message-passing interface (used only for distributed problems), HDF5, GNU Scientific Library (used only for Bessel-based spectral methods) and a BLAS implementation (used only for non-FFT-based spectral methods).

Nature of problem: General coupled initial-value stochastic partial differential equations.

Solution method: Spectral method with method-of-lines integration

Running time: Determined by the size of the problem

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: graham.dennis@anu.edu.au (G.R. Dennis).

0010-4655/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2012.08.016

1. Introduction

The integration of a system of variables from a set of initial conditions is one of the most widely performed tasks in quantitative simulation. Numerical integration is typically performed in one of two different styles: high-level methods using general software tools, or low-level methods using bespoke hand-tuned source code. The high-level approach requires much less code, and is therefore fast to develop and comparatively free of coding errors. However, the low-level approach can provide dramatic and necessary performance improvements, can utilise the full capacity of the computing platform for which it is developed, and is more customisable. XMDS2 is a software package whose aim is to provide the key benefits of both approaches [1].

The purpose of XMDS2 is to simplify the process of creating simulations that solve systems of initial-value partial and ordinary differential equations. Instead of going through the error-prone process of hand-writing thousands of lines of code, XMDS2 enables problems to be described in a simple XML format. From this XML description XMDS2 generates a C++ simulation that solves the problem using fast algorithms. The code generated by XMDS2 is typically as fast as, or faster than, hand-written code, but by using XMDS2 the time taken to produce the simulation is significantly reduced.

XMDS2 can be used to simulate almost any set of (coupled) (partial) (stochastic) differential equations in any number of dimensions. It can input and output data in a range of data formats, produce programs that can take command-line arguments, and produce parallelised code suitable for either modern computer architectures or distributed clusters.

Aside from innumerable low-level libraries and high-level packages for numerical integration, there have also been multiple previous attempts to automate or semi-automate the process of coding low-level numerical simulations, such as [2,3]. Rather than provide 'shell code' that can be edited, an XMDS2 script is effectively a self-contained language written in XML which is used to generate a fast C++ simulation.

The first version of XMDS was released in 1997 as an open-source software package written in C++ which could simulate a class of stochastic partial differential equations [4]. Over the next decade, it was then expanded in scope and features by a growing group of developers. While most of these developers came from the fields of quantum optics and atom optics, where its ability to integrate stochastic equations is particularly pertinent, the package slowly gained wider popularity. In 2008, the decision was taken to completely rewrite the package in Python (although still generating low-level C++ code), with a re-engineered structure that would allow it to address a much wider problem space. XMDS2 was released in 2010, and has recently received its first major update, along with extensive documentation, installers, and an examples library.

Citing only a few examples, XMDS and increasingly XMDS2 have been used in the fields of quantum atom optics [5,6], quantum optics [7,8], quantum control [9], predator–prey dynamics [10] and ecology [11–13].

2. Problem class

XMDS2 solves systems of initial-value differential equations. Each differential equation can:

1. Have an arbitrary number of dimensions, which may differ from that of other differential equations in the system.
2. Involve integrals of quantities in the differential system, or
3. Include stochastic elements either in initial conditions and filters, or in the dynamical equations themselves.

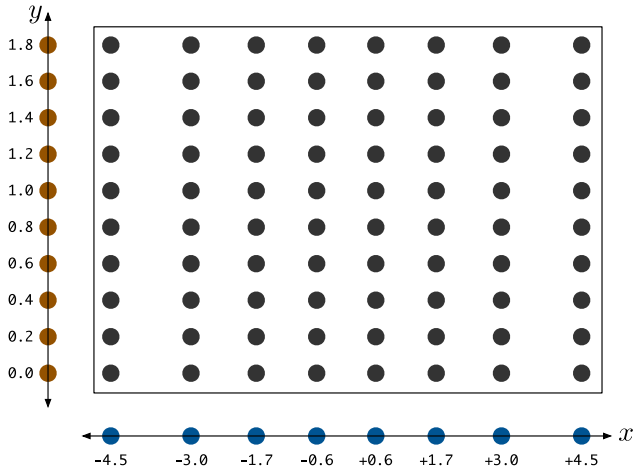


Fig. 1. An example of a tensor product lattice. The specific unequal lattice spacing in the x direction is due to the fact that the basis functions in that dimension have been chosen to be Hermite–Gauss.

As an example, property 1 means that XMDS2 can solve systems in which a partial differential equation is coupled to an ordinary differential equation. Property 2 allows the evolution of the ordinary differential equation to depend upon moments of the partial differential equation. Property 2 also allows partial differential equations to depend non-locally on system quantities. Property 3 permits the integration of systems of stochastic (partial) differential equations, which are typically written using either Gaussian noise (via a Wiener process), or a Poissonian noise in which the system changes state in a discontinuous way.

XMDS2 uses spectral methods [14], which induce two restrictions on the problem space. The first is that the geometry of the simulation domain must be a tensor product of lattices in each dimension (see Fig. 1). The second restriction is that the boundary conditions must be compatible with the spectral method used. XMDS2 currently supports periodic, even, odd and zero boundary conditions. Spectral methods allow the approximation of spatial derivatives with 'exponential' accuracy (see Section 3.1). In addition, the restriction to tensor product lattices affords significant computational savings which will be discussed later. The disadvantage is that XMDS2 cannot be used to solve problems on arbitrary-shaped domains as is possible using finite-element methods. This is not a significant limitation for a wide class of problems, as the system is often constrained to evolve within a finite domain. Quantum atom optics problems, for example, have trapping terms in the differential equation that cause the solution to be non-negligible over a finite domain.

The use of spectral methods means that XMDS2 represents the solution as a linear combination of *global* basis functions that extend over the entire domain. This is an accurate representation for solutions which are smooth. Problems which contain shocks or other spatial discontinuities (including discontinuous derivatives) are better served by *local* methods such as finite difference or finite element methods.

Subject to these caveats, XMDS2 is applicable to a broad problem class and employs efficient and accurate algorithms for the solution of these problems.

3. Algorithms employed

XMDS2 employs efficient algorithms in its generated simulations. These include:

1. Spectral methods for computing spatial derivatives.
2. Fast spatial-to-spectral transforms, including FFTs and parity-exploiting matrix transforms.

3. Distributed memory parallelism.
4. Gaussian quadrature for spatial integration.
5. Method-of-lines explicit temporal integration schemes, and
6. Interaction picture methods for exactly solving linear parts of the problem.

3.1. The spectral method

XMDS2 spatially discretises the problem by applying the spectral method [14]. This method decomposes the solution as a weighted sum of a finite set of orthonormal basis functions. For example, the quantity $f(x, y)$ is represented as

$$f(x, y) = \sum_{n,m} F_{n,m} X_n(x) Y_m(y), \tag{1}$$

where $F_{n,m}$ is a matrix of coefficients, $X_n(x)$ is the nth basis function for the x dimension, and $Y_m(y)$ is the mth basis function for the y dimension. The coefficients $F_{n,m}$ fully describe the solution and are the *spectral* representation of the solution. Typically in XMDS2, the number of basis functions is equal to the number of grid points in each dimension. In this case, the *spectral* representation is equivalent to the *spatial* representation $f(x_i, y_j)$, the values of the solution at the grid points. The two representations are linked by the linear transformation (1) and its inverse.

Spectral methods approximate spatial derivatives using the decomposition (1) and using analytic expressions for the derivatives of the basis functions,

$$\frac{\partial^p}{\partial x^p} \frac{\partial^q}{\partial y^q} f(x, y) = \sum_{n,m} F_{n,m} \frac{d^p X_n(x)}{dx^p} \frac{d^q Y_m(y)}{dy^q}. \tag{2}$$

Spatial derivatives approximated in this manner are 'exponentially' accurate. In general, an optimal M-point method to calculate a k-order derivative of a function will have error $\mathcal{O}(h^{M-k})$, where h is the grid-point spacing. As $h \propto 1/N$, where N is the number of grid points, such a method will converge like $\mathcal{O}(1/N^{M-k})$ for a k-order derivative. In spectral methods the value of the solution at all grid points is used when computing spatial derivatives, hence $M = N$. In this case M increases as the number of grid points N increases, resulting in a method whose order effectively *increases* as the number of grid points increases. The asymptotic error of a spectral method is $\mathcal{O}(1/N^{N-k})$, which converges *exponentially*.

The basis functions are typically chosen to make part of the differential equation diagonal in the spectral basis. XMDS2 supports the following spectral methods for each dimension:

- Fourier modes (complex exponentials),

  $X_n(x) = e^{ik_n x}$.

  This method imposes periodic boundary conditions. The basis functions are eigenfunctions of the Cartesian spatial derivative operator. This is a general purpose method.
- Cosine/sine functions,

  $X_n(x) = \cos(k_n x) \quad \text{or} \quad X_n = \sin(k_n x)$.

  These methods impose even and odd boundary conditions respectively at the ends of the domain. The basis functions are eigenfunctions of the Laplacian in Cartesian coordinates. This method is useful when the problem has even or odd reflection symmetry about a plane.
- 'Cylindrical' Bessel functions,

  $R_n(r) = J_m(k_n r)$,

  where $J_m(r)$ is the order-m Bessel function of the first kind. This method imposes analytic boundary conditions at the origin and zero Dirichlet boundary conditions at the outer boundary. The basis functions are eigenfunctions of the radial component of the Laplacian in cylindrical coordinates. This method is useful for problems with rotational symmetry. See [15] for more details.
- 'Spherical' Bessel functions,

  $R_n(r) = \sqrt{\frac{\pi}{2r}} J_{l+\frac{1}{2}}(k_n r)$.

  This method imposes analytic boundary conditions at the origin and zero Dirichlet boundary conditions at the outer boundary. The basis functions are eigenfunctions of the radial component of the Laplacian in spherical coordinates. This method is useful for problems with spherical symmetry.
- Hermite–Gauss functions,

  $\psi_n(x) = (2^n n! \sigma \sqrt{\pi})^{-1/2} e^{-x^2/2\sigma^2} H_n(\sigma x), \quad \text{where :}$

  $H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} \left(e^{-x^2}\right)$.

  This method requires that the solution decay as $e^{-x^2/2\sigma^2}$ in the limit $x \to \pm\infty$. The basis functions are eigenfunctions of the Schrödinger equation for the harmonic oscillator:

  $$-\frac{\hbar^2}{2m} \frac{\partial^2 \psi_n}{\partial x^2} + \frac{1}{2} m\omega^2 x^2 \psi_n(x) = \hbar\omega \left(n + \frac{1}{2}\right) \psi_n(x), \tag{3}$$

  with $\sigma = \sqrt{\hbar/(m\omega)}$. This method is useful for solving problems similar to (3) with nonlinear terms.

XMDS2 permits the use of different spectral methods in each dimension. Fig. 1 is an example of a lattice using a Hermite–Gauss decomposition in the x dimension and a Fourier decomposition in the y dimension. As discussed in Section 3.4, the grid spacing is determined by the choice of spectral method. Full documentation of the spectral methods supported by XMDS2 and their uses is available from the XMDS2 website [1].

3.2. Fast spatial-to-spectral transforms

In any nonlinear simulation, both the spatial and spectral representations of the solution will be required, as the problem will not be diagonal in either representation. Typically, the spatial representation is used for calculating the nonlinear terms, and the spectral representation for calculating derivatives. The two are linked by a linear transformation, which can in general be performed with a matrix multiplication. The computational complexity of this operation is $\mathcal{O}(N^2)$ for a single dimension. In higher dimensions, the use of a tensor product lattice (see Fig. 1) enables the matrix multiplication to be factorised for each dimension. In two dimensions for example, the computational complexity of a general spatial-to-spectral transformation is $\mathcal{O}(N_1^2 N_2 + N_1 N_2^2)$. Without the use of a tensor product lattice, this cost would be $\mathcal{O}(N_1^2 N_2^2)$.

There are two cases in which we can reduce this computational cost: when we can use the Fast Fourier Transform (FFT) algorithm, or when the basis functions alternate in parity.

Spectral methods using complex exponentials, cosines or sines enable the use of the FFT algorithm and its variants for transformations between spatial and spectral representations. These cost only $\mathcal{O}(N \log N)$ in one dimension or $\mathcal{O}(N_2 N_1 \log N_1 + N_1 N_2 \log N_2)$ in two dimensions.

If the basis functions have explicit, alternating parity $X_n(-x) = (-1)^n X_n(x)$ like the Hermite–Gauss functions, the Parity Matrix Multiplication Transform (PMMT) [14] can be used, which is faster than a direct matrix multiplication in each dimension. The idea is to separately transform the even and odd components of the solution, each of which costs $\mathcal{O}\left((N/2)^2\right)$, giving a total cost of $\mathcal{O}(N^2/2)$. This factor of two reduction does not improve the overall scaling but can be a significant improvement for simulations dominated by the cost of the spatial-to-spectral transforms.

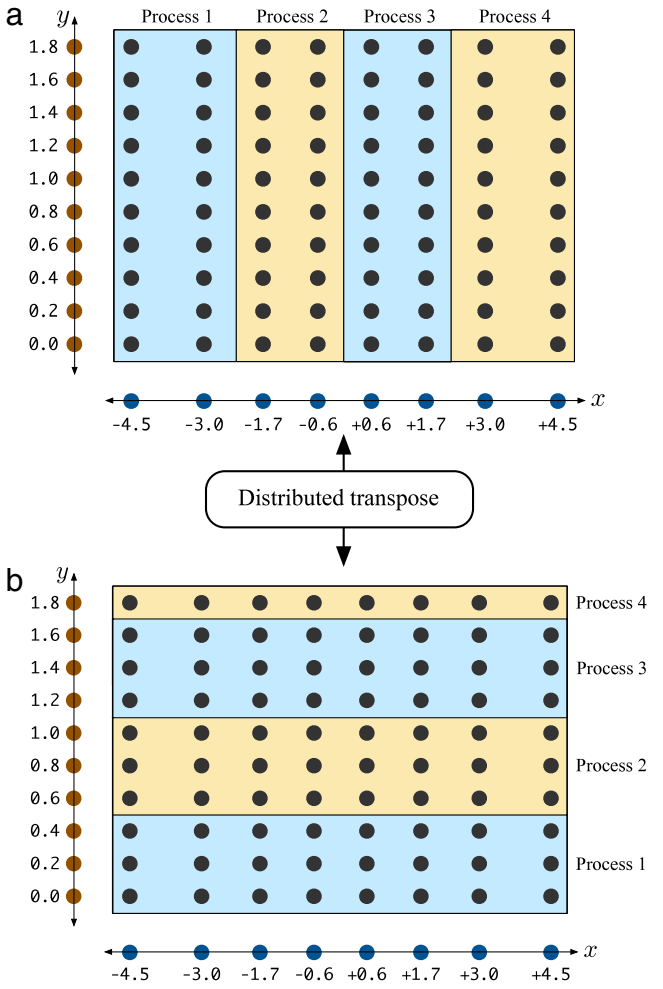


Fig. 2. An example of problem parallelisation on a tensor product lattice. The problem is distributed across the (a) x or (b) y dimensions. These two problem decompositions are linked by a distributed transpose operation.

3.3. Distributed memory parallelism

The use of a tensor product lattice permits the problem to be parallelised by distributing a single dimension across the available processes (see Fig. 2). The advantage of this method is that as the spatial-to-spectral transform can be factorised across different dimensions, when the problem is decomposed across the x dimension (as in Fig. 2(a)), the transform over the y dimension can be performed as a purely *local* operation to each process.

To perform the spatial-to-spectral transform over the x dimension, the problem must instead be decomposed across in the y dimension (as in Fig. 2(b)). As simulations typically require spatial-to-spectral transforms to be performed over all dimensions, a distributed transpose operation is used to link different problem decompositions (see Fig. 2). This enables transforms to be performed over any dimension in a distributed simulation.

3.4. Gaussian quadrature

Gaussian quadrature is an exponentially accurate method for integrating functions. The key idea is to approximate

$$\int f(x)\,dx \approx \sum_i f(x_i)\,w_i, \tag{4}$$

where x_i are the interpolation points and w_i are weight factors. Gaussian quadrature takes advantage of the fact that the interpolation points x_i do not need to be equally spaced. This means the $2N$ degrees of freedom $\{x_i, w_i\}$ can be chosen to exactly integrate $2N$ functions $f(x)$, while it would only be possible to exactly integrate N functions if the w_i were the only degrees of freedom. Further details about Gaussian quadrature are available from [14,16].

3.5. Method-of-lines explicit temporal integration

In method-of-lines integration, each grid point is considered to have its own ODE and the problem is integrated as a system of coupled ODEs. XMDS2 employs a range of explicit integration schemes for deterministic and stochastic problems:

- semi-implicit method (deterministic order 2, stochastic order 1) [17],
- fourth-order Runge–Kutta (deterministic order 4, stochastic order 1/2) [18, Section 3.7(v)],
- ninth-order Runge–Kutta (deterministic order 9, stochastic order 1/2) [19],
- adaptive fourth–fifth order Runge–Kutta (deterministic only), [20] and
- adaptive eighth-ninth order Runge–Kutta (deterministic only) [19].

XMDS2's fixed-step method-of-lines integration methods support integrating stochastic differential equations that depend on Wiener (Gaussian) or jump (counting) processes. These stochastic differential equations must be entered in Stratonovich, not Itô form [21].

Although the fourth-order Runge–Kutta and ninth-order Runge–Kutta algorithms have lower order stochastic convergence than the semi-implicit method, we find that they can be useful for problems where the noise terms are a perturbation on the 'deterministic' dynamics.

XMDS2 can run multiple paths (possibly distributed across multiple processors) to compute moments of the stochastic process. XMDS2 can also test the effect on the strong convergence [22] of the discretisation error of the propagation dimension. This requires sampling the *same stochastic trajectory* with timesteps of multiple sizes.

3.6. Interaction picture method

The method-of-lines integration schemes are supported by the interaction picture method [23,24], which exactly solves a linear part of the differential equation.

The idea is very similar to the interaction picture in quantum mechanics. The differential equation is split into two parts: a linear, exactly solvable component, and the remaining possibly nonlinear components. The differential equation is then transformed to remove the exactly solvable component.

For a PDE of the form

$$\frac{\partial f}{\partial t} = \mathcal{L}[f] + g(x, y, f), \tag{5}$$

where $\mathcal{L}$ is a linear operator that does not depend on time, the differential equation is transformed by defining the new quantity $\tilde{f} = e^{-\mathcal{L}t} f$, which evolves as

$$\frac{\partial \tilde{f}}{\partial t} = e^{-\mathcal{L}t} g(x, y, e^{\mathcal{L}t}\tilde{f}). \tag{6}$$

The new quantity $\tilde{f}$ essentially has the simple dynamics due to $\mathcal{L}$ removed.

The interaction picture method is advantageous when the linear operator $\mathcal{L}$ has a faster characteristic timescale than the remainder of the differential system, which means that the function $\tilde{f}$ varies more slowly in time than the original f. This means that by solving

the faster component separately, and exactly, larger time-steps may be used on the remaining part of the differential equation while achieving the same solution accuracy.

For example, for the nonlinear Schrödinger equation,

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\psi}{\partial x^2} + V(x)\psi + U|\psi|^2\psi, \tag{7}$$

the spatial derivative term can have a faster characteristic timescale than the remainder of the system for high spatial resolutions, corresponding to high-momentum components. In the Fourier basis, the spatial derivative term in Eq. (7) becomes

$$\frac{\hbar^2 k_x^2}{2m}\psi(k_x, t). \tag{8}$$

If we do not use the interaction picture, the maximum value of k_x increases linearly with the number of grid points, and the time step used must decrease as $\Delta t \propto 1/N^2$ in order to be able to resolve the evolution of those terms. The interaction picture method alleviates this problem by solving the spatial derivative term *exactly*. Using the interaction picture method to solve for the evolution of the spatial derivative term enables (7) to be solved with a time-step which is *independent* of the spatial resolution.

The effect of the interaction picture method can be seen by solving (7) with an adaptive temporal integration method and comparing the number of time steps needed to achieve a given accuracy to that needed when calculating the derivatives explicitly (but still using a spectral method). The results in Fig. 3 demonstrate that the number of steps needed to solve the PDE using the interaction picture method is essentially independent of the spatial resolution, while for the explicit method, the number of steps needed increases quadratically.

The computational cost of the interaction picture method is small if the linear operator $\mathcal{L}$ is local in either the spatial or spectral basis. In this case, the application of $e^{\pm\mathcal{L}t}$ to the quantity f can be calculated by transforming f to the appropriate basis (spatial or spectral), performing a local multiplication, and transforming back to the original basis. For fixed time-step algorithms, calculating the exponential function at every time step can be avoided by essentially redefining $\tilde{f}$ at each time step so that only the quantities $e^{\pm\mathcal{L}\Delta t}$ are needed.

To make best use of the interaction picture method, the basis functions should be chosen to make all derivative terms local in the spectral basis. This ensures optimal scaling of the computational effort with spatial resolution.

Although the interaction picture method can be used with any integration software by applying the transformation manually, XMDS2 makes its use particularly easy by allowing the differential equation to be entered in a form equivalent to (5) with XMDS2 automatically making the transformation to (6). This also enables easy comparisons to be made between the interaction picture and explicit methods.

4. Examples

In order to show the syntax of an XMDS2 script, as well as to demonstrate the ease with which simulations can be extended, we consider the behaviour of a Bose–Einstein condensate (BEC) in a harmonic magnetic trap.

4.1. Example 1: nonlinear Schrödinger equation (`examples/cpc_example1.xmds`)

Under a semiclassical approximation, the dynamics of the BEC will be governed by the nonlinear Schrödinger equation with a harmonic trapping potential. In dimensionless units this equation is written

$$i\frac{\partial\psi}{\partial\bar{t}} = -\frac{1}{2}\frac{\partial^2\psi}{\partial\bar{x}^2} + \frac{1}{2}\bar{x}^2\psi + U|\psi|^2\psi, \tag{9}$$

where $\bar{x} = \sqrt{m\omega/\hbar}\,x$, $\bar{t} = \omega t$, m is the atomic mass, ω is the trapping frequency and U is the nonlinear energy in units of $\hbar\omega$. XMDS2 is capable of solving much more complicated (sets) of PDEs, but this serves as a illustrative example.

Our initial condition will specify the wavefunction at $t = 0$, and we choose

$$\psi(\bar{x}, 0) = \sqrt{N}\,\pi^{-1/4}\exp(-\bar{x}^2/2) \tag{10}$$

which is the ground state solution to Eq. (9) in the absence of the nonlinearity, normalized to N atoms in total.

We initially solve in one dimension, using a fourth–fifth order adaptive Runge–Kutta algorithm, evolving the system for a time $\bar{t} = 2\pi/\omega$ (one trap period), sampling 50 times and outputting the real and imaginary parts of the wavefunction in position space at every grid point. An XMDS2 script to solve this problem is shown in Fig. 4.

When XMDS2 is run on this script, it produces an optimized binary `nonlinear_SE` which is run to carry out the simulation. The result is shown in Fig. 6.

Changing parameters such as the domain or number of grid points, the number of sample points, integration interval, output moments, algorithm and precision used and so on is simply a matter of changing the contents of an XML tag, then re-running XMDS2 on the script to produce the new executable. While it is trivial to change such parameters, it is also easy to extend the simulation in more complex ways.

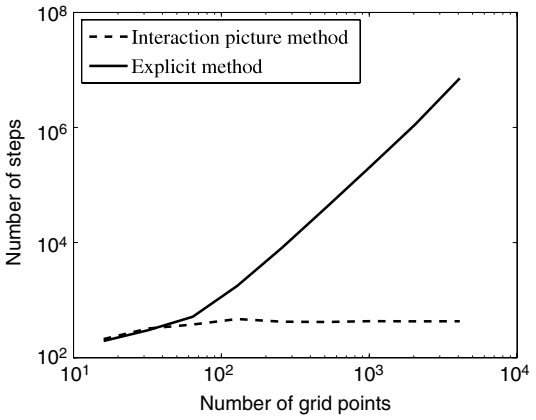
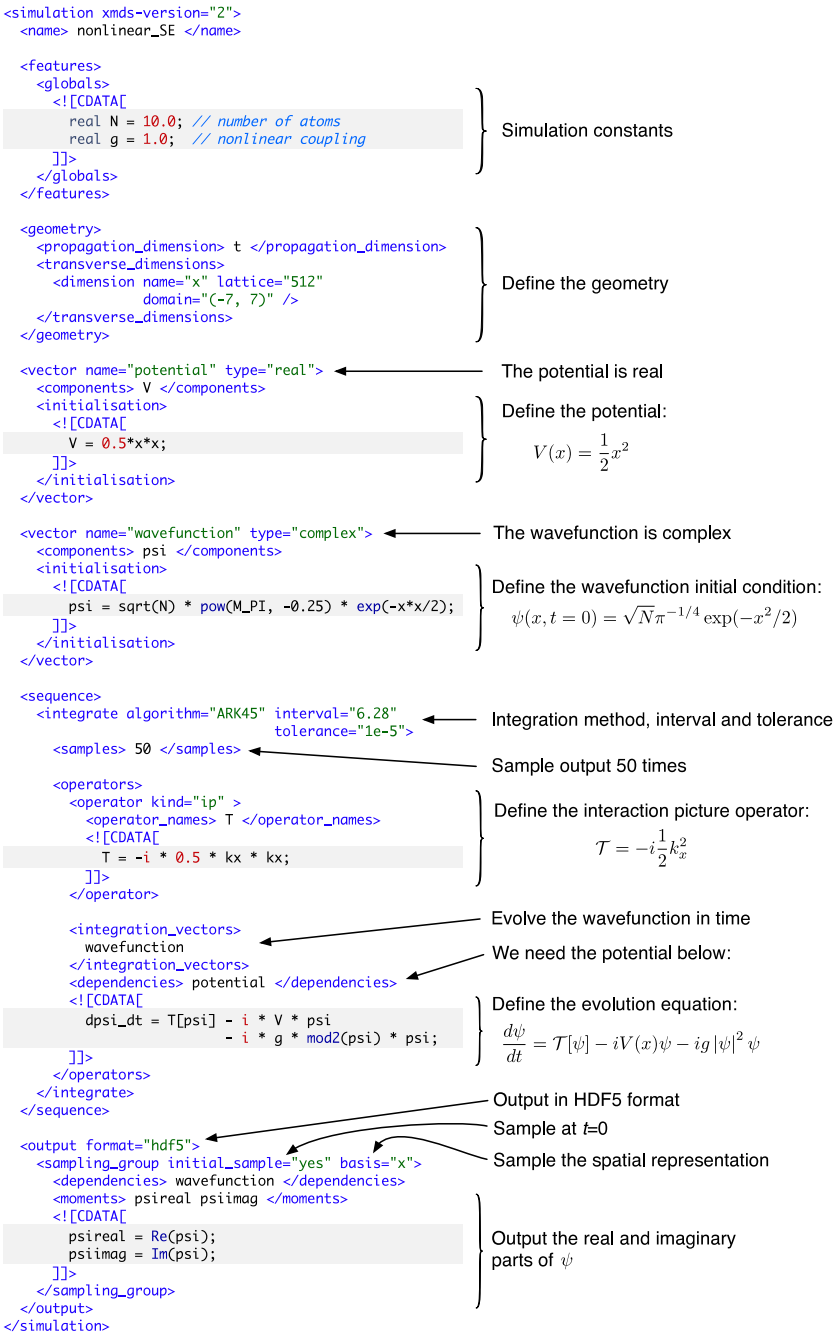


Fig. 3. Comparison of the scaling of the interaction picture and 'explicit' methods with grid resolution for computing the evolution due to spatial derivative terms. Both methods were used to integrate the PDE (9) with a fixed accuracy using an adaptive integrator. As the resolution is increased, the number of steps remains approximately steady for the interaction picture method, while it increases quadratically (due to the second order spatial derivatives in (9)) for the 'explicit' method. As the computational cost of each time step for both methods increases with resolution as $\mathcal{O}(N\log N)$, due to the use of Fourier transforms, the overall running time for the interaction picture method scales as $\mathcal{O}(N\log N)$ compared to $\mathcal{O}(N^3\log N)$ for the 'explicit' method. The XMDS2 scripts used can be found in `examples/cpc_ip_scaling.xmds` and `example/cpc_ex_scaling.xmds` in the XMDS2 distribution.

4.2. Example 2: higher dimensions (`examples/cpc_example2.xmds`)

If one wished to run the simulation in two dimensions rather than one, all that is required is adding the element

```
<dimension name="y" lattice="512"
                    domain="(-7, 7)" />
```




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

  <features>
    <globals>
      <![CDATA[
        real N = 10.0; // number of atoms
        real g = 1.0;  // nonlinear coupling
      ]]>
    </globals>
  </features>

  <geometry>
    <propagation_dimension> t </propagation_dimension>
    <transverse_dimensions>
      <dimension name="x" lattice="512"
                 domain="(-7, 7)" />
    </transverse_dimensions>
  </geometry>

  <vector name="potential" type="real">
    <components> V </components>
    <initialisation>
      <![CDATA[
        V = 0.5*x*x;
      ]]>
    </initialisation>
  </vector>

  <vector name="wavefunction" type="complex">
    <components> psi </components>
    <initialisation>
      <![CDATA[
        psi = sqrt(N) * pow(M_PI, -0.25) * exp(-x*x/2);
      ]]>
    </initialisation>
  </vector>

  <sequence>
    <integrate algorithm="ARK45" interval="6.28"
                                 tolerance="1e-5">
      <samples> 50 </samples>

      <operators>
        <operator kind="ip" >
          <operator_names> T </operator_names>
          <![CDATA[
            T = -i * 0.5 * kx * kx;
          ]]>
        </operator>

        <integration_vectors>
          wavefunction
        </integration_vectors>
        <dependencies> potential </dependencies>
        <![CDATA[
          dpsi_dt = T[psi] - i * V * psi
                           - i * g * mod2(psi) * psi;
        ]]>
      </operators>
    </integrate>
  </sequence>

  <output format="hdf5">
    <sampling_group initial_sample="yes" basis="x">
      <dependencies> wavefunction </dependencies>
      <moments> psireal psiimag </moments>
      <![CDATA[
        psireal = Re(psi);
        psiimag = Im(psi);
      ]]>
    </sampling_group>
  </output>
</simulation>
```

Fig. 4. Annotated example XMDS2 script for integrating Eq. (9). This script can be found in `examples/cpc_example1.xmds` in the XMDS2 distribution.

to the `transverse_dimensions` element, changing the `basis="x"` attribute of the `sampling_group` element to `basis="x y"`, adding a `"0.5*y*y"` term to the potential `"V"` and initial condition, and adding a `"-i*0.5*ky*ky"` term to the kinetic energy operator `"T"`.

4.3. Example 3: different transforms (*examples/cpc_example3.xmds*)

This problem is obviously symmetric about $x = 0$, so it is a waste of computational resources to simulate the problem on both sides of the origin. Since the differential equation and the boundary conditions are symmetric, by using the discrete cosine transform rather than the default exponential Fourier transform, we need only carry out the simulation on half the interval, and use only half the number of grid points for the same accuracy. This is accomplished simply by changing the content of the `transverse_dimensions` element to be

```
<dimension name="x" lattice="256"
           domain="(0, 7)" transform = "dct"/>
```

4.4. Example 4: easy parallelization with MPI (*examples/cpc_example4.xmds*)

As this is a deterministic simulation, if it has two or more dimensions, one can parallelize the simulation simply by adding the `<driver name="distributed-mpi"/>` tag to the script. This would result in a binary that could be run across, for example,

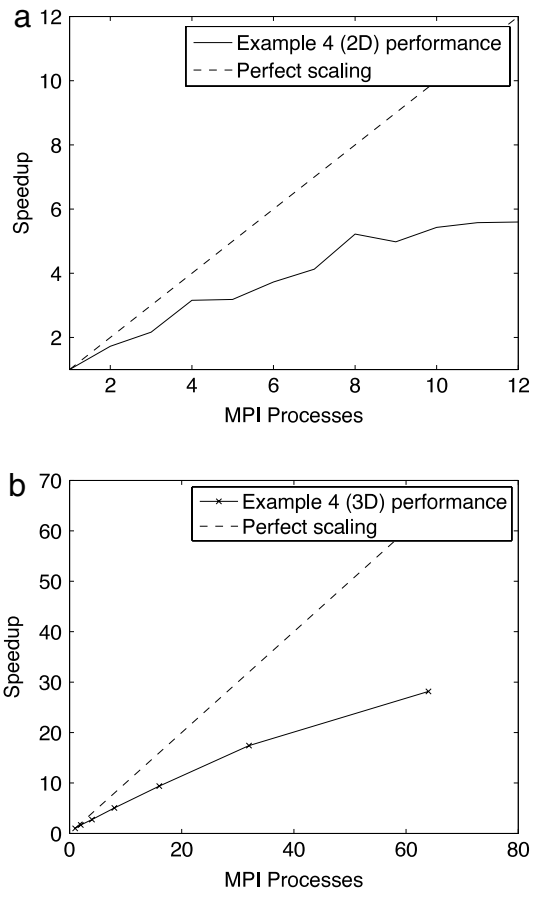


Fig. 5. Example runtime scaling using MPI on (a) a single computer, and (b) a supercomputer. Figure (a) demonstrates the simulation `examples/cpc_example4.xmds` run on a Linux computer with two Xeon 5675 CPUs running at 3.07 GHz. Each CPU has 6 execution cores. Figure (b) demonstrates the simulation `examples/cpc_example4_3D.xmds` run on the NCI National Facility supercomputer, 'vayu'. The modified simulation is extended to three dimensions with 256 points in each to demonstrate performance on larger problems. Note that optimal parallelisation for these problems is achieved when the number of grid points in the first dimension (256) is divisible by the number of processes.

four CPUs with the command

```
mpirun -n 4 nonlinear_SE
```

The fact that two- or higher-dimensional deterministic simulations, as well as stochastic simulations of any dimension, can be trivially parallelized using a single line in a script, without spending days (or weeks) writing and debugging bespoke code, is one of XMDS2's most powerful features. The runtime scaling of this simulation with the number of processes is illustrated in Fig. 5.

4.5. Example 5: non-local terms (`examples/cpc_example5.xmds`)

Many problems will involve non-local interactions that occur in the form of a convolution $\int f(r-r')g(r')\,dr'$. For example, within the context of the current problem, if the BEC were charged there would be an additional potential of the form

$$V(x) = \frac{e^2Z^2}{4\pi\epsilon_0}\int \frac{1}{|x-x'|}|\psi(x')|^2\,dx' \tag{11}$$

where eZ is the charge associated with each particle. While this term could be explicitly integrated within XMDS2 it is more efficient to make use of convolutions and the speed of fast Fourier transforms. This is done using `<computed_vector>` elements, which are described in detail on our website [1].

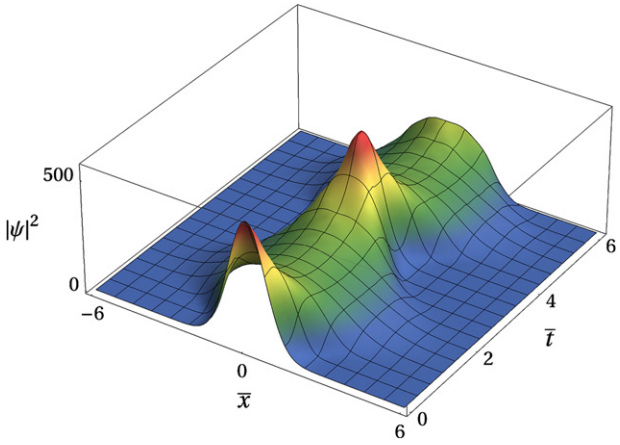


Fig. 6. Solution to the nonlinear Schrödinger equation given by Eq. (9). The density $|\psi(x,t)|^2$ is shown evolving over one trap period.

4.6. Example 6: stochastics (`examples/cpc_example6.xmds`)

As a final tweak to this example, we will make use of XMDS2's stochastic features to add noise. XMDS2 has a number of fast random number generators built in, which are capable of producing Gaussian, Poissonian and uniform probability distributions, and applying them as Wiener or jump processes during stochastic integration. This enables the simulation of stochastic differential equations, which are useful in fields such quantum field theory, mathematical finance, and many others. For this example we will simply use noise to model perturbations of the magnetic trap—that is, the trapping potential will be slightly noisy, due to it moving around. To do this we define a noise vector

```
<noise_vector name="trapNoise" kind="wiener"
              type="real" method="dsfmt">
  <components> noise_x </components>
</noise_vector>
```

change the potential term in the equation of motion in the script to

```
- i * (V + g * mod2(psi) + alpha*noise_x) * psi
```

where `alpha` is a constant governing the magnitude of the noise, and add a `<dependencies>trapNoise< /dependencies>` tag to the `<initialisation>` block of the `potential` vector. This would add a time-dependent Gaussian-distributed noise to the potential. If we wished to average over many different realisations of this noise, we could add the tag

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

to the script, which would run the simulation 100 times, and average over whichever results were requested in `<output>` section. Such a simulation could be trivially run over any number of CPU cores with near perfect scaling.

Note that the `mpi-multi-path` driver should only be used for stochastic simulations where individual realisations are independent, in contrast, the `distributed-mpi` driver parallelises a single deterministic simulation. As the different components of a deterministic simulation will in general be coupled, the `distributed-mpi` driver necessarily incurs a larger communication overhead than the `mpi-multi-path` driver. In general, the `distributed-mpi` driver can be used to parallelise a single realisation of a stochastic simulation, but if many paths are needed, the `mpi-multi-path` driver will be preferable.

5. Software used

XMDS2 makes use of the following external libraries in its generated C++ simulations:

- FFTW3 [25] for FFTs and MPI distributed transpose operations,
- dSFMT [26] for random number generation,
- MPI for inter-process communication,
- HDF5 for data input and output, and
- GNU Scientific Library for special function evaluation.

XMDS2 itself also uses the following Python libraries when generating simulations: Cheetah, pyparsing, lxml, h5py, mpmath, and numpy.

6. Conclusion

Using XMDS2 for simulations accelerates development time, produces code that executes extremely quickly, and also produces a self-documenting workflow, as output data is wrapped with the compact XML code used to produce it.

XMDS2 particularly excels at providing a smooth transition from a low-dimensional simulation to a higher-dimensional one, from a deterministic simulation to a stochastic one, or from a single-processor simulation to a distributed simulation running in parallel across multiple computers (or on a supercomputer). In hand-written codes, unless they were initially written with such a potential future extension in mind, each such change would require significant effort in rewriting the code. In XMDS2 such changes require only minimal change to the input script. This encourages users to create test simulations of a simpler system (e.g. reduced dimensionality), which makes the code run faster, allowing problems in the input script to be found and fixed more quickly. Later, the simulation can be scaled up to the full problem. Fundamentally, the ease with which codes can be generated encourages experimentation with different types of simulations, as the time taken to create the code is no longer the rate-limiting factor.

The installers, documentation and examples for XMDS2 can be found at the website [1]. This same documentation is available in the `documentation/` directory of the XMDS2 distribution.

Acknowledgments

We would like to thank B. Blakie for assistance with the Hermite–Gauss basis, M. Hush, R. Stevenson, and S. Szigeti for testing early versions of XMDS2, and the XMDS2 community for testing and ideas. We also acknowledge support from the NCI National Supercomputing Facility.

References

[1] J.J. Hope, G.R. Dennis, M.T. Johnsson, XMDS website and documentation. http://www.xmds.org.
[2] L. DeRose, K. Gallivan, E. Gallopoulos, B. Marsolf, D. Padua, An environment for the rapid prototyping and development of numerical programs and libraries for scientific computation, in: Proc. DAGS'94 Symposium on Parallel Computing and Problem Solving Environments, 1994, pp. 11–25.
[3] R. Marsa, The RNPL User's guide and Reference are available online at: http://godel.ph.utexas.edu/Members/marsa/rnpl/.
[4] G. Collecutt, P.D. Drummond, XMDS: extensible multi-dimensional simulator, Comput. Phys. Comm. 142 (1–3) (2001) 219–223.
[5] R.G. Dall, L.J. Byron, A.G. Truscott, G.R. Dennis, M. Jeppesen, M.T. Johnsson, J.J. Hope, Observation of interference fringes on an atom laser beam, Opt. Express 15 (26) (2007) 17673.
[6] R.G. Dall, L.J. Byron, A.G. Truscott, G.R. Dennis, M.T. Johnsson, J.J. Hope, Paired-atom laser beams created via four-wave mixing, Phys. Rev. A 79 (2009) 011601(R).
[7] M.T.L. Hsu, G. Hétet, A. Peng, C.C. Harb, H.A. Bachor, M.T. Johnsson, J.J. Hope, P.K. Lam, A. Dantan, J. Cviklinski, A. Bramati, M. Pinard, Effect of atomic noise on optical squeezing via polarization self-rotation in a thermal vapor cell, Phys. Rev. A 73 (2006) 023806.
[8] G. Hétet, A. Peng, M.T. Johnsson, J.J. Hope, P.K. Lam, Characterization of electromagnetically-induced-transparency-based continuous-variable quantum memories, Phys. Rev. A 77 (2008) 012323.
[9] S.S. Szigeti, M.R. Hush, J.J. Carvalho, A.R.R. Hope, Feedback control of an interacting Bose–Einstein condensate using phase-contrast imaging, Phys. Rev. A 82 (2010) 043632.
[10] L. Li, Z. Jin, Pattern dynamics of a spatial predator–prey model with noise, Nonlinear Dynam. 67 (3) (2012) 1737.
[11] T. Reichenbach, M. Mobilia, E. Frey, Noise and correlations in a spatial population model with cyclic competition, Phys. Rev. Lett. 99 (2007) 238105.
[12] T. Reichenbach, M. Mobilia, E. Frey, Mobility promotes and jeopardizes biodiversity in rock-paper-scissors games, Nature 448 (2007) 06095.
[13] T. Reichenbach, E. Frey, Instability of spatial patterns and its ambiguous impact on species diversity, Phys. Rev. Lett. 101 (2008) 058102.
[14] J.P. Boyd, Chebyshev and Fourier Spectral Methods, second ed., Dover, 2000.
[15] S. Ronen, D.C.E. Bortolotti, J.L. Bohn, Bogoliubov modes of a dipolar condensate in a cylindrical trap, Phys. Rev. A 74 (1) (2006) 013623.
[16] A.H. Stroud, Numerical Quadrature and Solution of Ordinary Differential Equations, Springer-Verlag, 1974.
[17] M.J. Werner, P.D. Drummond, Robust algorithms for solving stochastic partial differential equations, J. Comput. Phys. 132 (2) (1997) 312–326.
[18] M. Abramowitz, I.A. Stegun (Eds.), Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, tenth ed., Dover Publications, 1972.
[19] C. Tsitouras, Optimized explicit Runge–Kutta pair of orders 9 (8), Appl. Numer. Math. 38 (1–2) (2001) 123–134.
[20] J.R. Cash, A.H. Karp, A variable order Runge–Kutta method for initial value problems with rapidly varying right-hand sides, ACM Trans. Math. Software 16 (3) (1990) 201–222.
[21] C.W. Gardiner, Handbook of Stochastic Methods, third ed., in: Springer Series in Synergetics, Springer, Berlin, 2004.
[22] P.E. Kloeden, E. Platen, Numerical Solution of Stochastic Differential Equations, Springer, 1992.
[23] R.J. Ballagh, Partial differential equation algorithm: conceptual, Unpublished Personal Papers, 1995.
[24] B.M. Caradoc-Davies, Vortex dynamics in Bose–Einstein condensates, Ph.D. Thesis, University of Otago, July 2000.
[25] M. Frigo, S.G. Johnson, The design and implementation of FFTW3, Proc. IEEE 93 (2) (2005) 216–231.
[26] M. Saito, M. Matsumoto, SIMD-oriented fast Mersenne twister: a 128-bit pseudorandom number generator, in: A. Keller, S. Heinrich, H. Niederreiter (Eds.), Monte Carlo and Quasi-Monte Carlo Methods 2006, Springer, Berlin, Heidelberg, 2008, pp. 607–622.