cuPSS: a package for the pseudo-spectral integration of stochastic PDEs

Fernando Caballero[1⋆]

1 Department of Physics, Brandeis University, Waltham, Massachusetts 02453, USA

⋆ fcaballero@brandeis.edu

Abstract

This manuscript introduces cuPSS, a package to numerically integrate arbitrary systems of stochastic partial differential equations (SPDEs). SPDEs are a powerful mathematical tool to describe a large class of physical systems. Their numerical integration usually relies eiher on in-house made codes or external packages like MATLAB, Mathematica, Fenicsx, OpenFOAM, Dedalus, and others. These packages rarely offer a good combination of speed, generality, and the option to easily add stochasticity to the system, while in-house codes depend on certain expertise to obtain good performance, and are usually written for each specific use-case, sacrificing modularity and reusability. cuPSS, by contrast, can deal with the integration of an arbitrary number of stochastic fields and differential equations, and is written in CUDA C++, thus enabling by default GPU acceleration. The numerical integration is done pseudo-spectrally in flat lattices in one, two and three dimensions. This manuscript describes the basic functionality of cuPSS, with an example and benchmarking, showing that cuPSS offers a considerable improvement in speed over other popular finite-difference and spectral solvers.

Received Date
Accepted Date
Published Date

Contents

1 Introduction

Stochastic partial differential equations are a powerful tool to describe a plethora of interesting phenomena in mesoscopic length and time scales in which some fast degrees of freedom can be approximated by a noise term. These include systems in fields as varied as fluid mechanics [1], ferromagnetism [2], phase transitions and critical phenomena [3,4], interface dynamics [5–8], soft and active matter [9–14], systems biology [15, 16], finance [17–19], and many others. These equations can be generally written as a function of a field $\phi(\mathbf{x}, t)$, as

$$d\phi = F[\phi]dt + \sqrt{2D}dW, \tag{1}$$

where $F[\phi]$ is a functional describing the deterministic evolution of ϕ, while dW is a Wiener process. I assume that all SPDEs are written calculated in the Ito formulation when writing any discretization for them, which allows for a more natural description of the noise in discrete time [20]. I also allow the strength of the noise D to be a differential operator, which will be useful for describing different types of uncorrelated noises [3].

Despite the rich theoretical machinery to analyze the properties of this type of equations, their stochastic nature, and potentially complex form of their deterministic evolution, makes numerical methods a powerful tool in investigating them. These methods offer a quick insight into their behavior, as well as a way to test hypotheses about a model prior to investing more effort into either analytical approaches or expensive systematic parameter exploration.

1.1 Pseudo-spectral integration

Of the plethora of methods available to numerically integrate an equation of the type of Eq. 1, spectral methods offer the advantage of higher numerical stability to space discretization, especially in the case where the deterministic part of the evolution $F[\phi]$ has terms with many gradients. Writing the Fourier transform of $\phi(\mathbf{x}, t)$ in space as $\tilde{\phi}(\mathbf{q}, t) = (2\pi)^{-d/2}\int d\mathbf{q}e^{-i\mathbf{q}\cdot\mathbf{x}}\phi(\mathbf{x}, t)$, Eq. 1 as can be rewritten as

$$\partial_t\tilde{\phi}(\mathbf{q}, t) = \tilde{F}[\tilde{\phi}] + \sqrt{2\tilde{D}}\tilde{\eta}, \tag{2}$$

where I have dropped the differential form of the starting equation, and rewritten the noise as $\tilde{\eta}$, now defined with an average of 0 and a variance

$$\langle\eta(\mathbf{q}, t)\eta(\mathbf{q}', t')\rangle = (2\pi)^d\delta^{(d)}(\mathbf{q} + \mathbf{q}')\delta(t - t'). \tag{3}$$

The noise strength is written with a tilde because, as mentioned above, it is not necessarily a constant. For instance, a common form of Eq. 2 is a continuity equation for a mass conserving field, $\partial_t\phi = -\nabla\cdot\vec{J}$, for some current $\vec{J}$. A noise term to this system is therefore applied to the current, so that $\vec{J} = \vec{J}_d + \vec{\Lambda}$, where $\vec{J}_d$ is the deterministic part of the current, and $\vec{\Lambda}$ is a vectorial noise. A simple calculation shows that, in Fourier space, this noise can be written as in Eq. 2, where $\tilde{D} = D_0 q^2$ for a constant parameter D_0 [3].

A drawback of spectral methods is their inefficiency when it comes to computing nonlinear terms in $F[\phi]$, since polynomial nonlinearities of order n will involve convolutions that take $O(N^n)$ steps to compute, where N is the number of steps in which space is discretizes. A solution for dealing with these types of nonlinearities in a more efficient way is to use the so-called pseudo-spectral methods [21], where the nonlinearities are instead calculated in real space, while gradients are calculated in Fourier space. This offers a considerable speedup, despite the need to transform fields back and forth between real and Fourier spaces, since there are fast Fourier transform algorithms (commonly known as FFT algorithms [22]) that work at a speed of $O(N \log N)$.

For instance, let's consider a simple example of a deterministic evolution of a field $\phi(\mathbf{x}, t)$ with cubic decay

$$\partial_t \phi(\mathbf{x}, t) = -\phi(\mathbf{x}, t)^3. \tag{4}$$

This equation can be written for the Fourier transform of $\phi(\mathbf{x}, t)$, $\tilde{\phi}(\mathbf{q}, t)$, with wavevector $\mathbf{q}$,

$$\partial_t \tilde{\phi}(\mathbf{q}, t) = -\iint \frac{d\mathbf{k}d\mathbf{l}}{(2\pi)^{2d}} \tilde{\phi}(\mathbf{k}, t)\tilde{\phi}(\mathbf{l}, t)\tilde{\phi}(\mathbf{q}-\mathbf{k}-\mathbf{l}, t). \tag{5}$$

Computing the right-hand side numerically in a system discretized in a lattice of size N would thus require $O(N^3)$ calculations; $O(N^2)$ calculations for the convolution at each wavevector $\mathbf{q}$. A pseudo-spectral method consists of viewing the previous equation in the following equivalent way

$$\partial_t \tilde{\phi}(\mathbf{q}, t) = \mathcal{F}\left[\mathcal{F}^{-1}\left[\tilde{\phi}(\mathbf{q}, t)\right]^3\right], \tag{6}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the Fourier transform and inverse Fourier transform operators respectively. This method will require only $O(N^2 \log N)$ steps, since multiplication in real space is linear in the number of lattice points, and Fourier transforms can be computed at a faster speed than a naive convolution.

A pseudo-spectral method thus consists of numerically integrating Eq. 2 in Fourier space, but calculating nonlinear terms by transforming each factor to real space, calculating the product in real space, and transforming the result back to Fourier space.

1.2 De-aliasing

One last basic detail must be mentioned when talking about pseudo-spectral methods, for which we need to introduce an explicit discretization of space. Consider a field $\phi(x, t)$, defined in a one-dimensional space of size N, discretized in $N/\Delta x$ points, each separated from the next at a distance of Δx. This induces a natural discretization of Fourier space in N frequencies q_i with a spacing of $\Delta q = 2\pi/(N\Delta x)$, where it is common to choose a frequency interval symmetric around $q = 0$, such that $q_i \in (-\pi/\Delta x, \pi/\Delta x)$. Let us assume for simplicity, and without loss of generality, that $\Delta x = 1$. We then have a lattice of N frequencies going from $-\pi$ to π. Let us now consider the discrete Fourier series of the field $\phi(x)$, which can be written $\phi(x) = \sum_{q_i} \tilde{\phi}(q_i) \exp(iq_i x)$. Consider a quadratic nonlinearity derived from this field

$$\phi(x)^2 = \sum_{q_i, q_j} \tilde{\phi}(q_i)\tilde{\phi}(q_j) \exp(i(q_i + q_j)x). \tag{7}$$

Notice that for high frequencies, it is possible that $q_i + q_j > \pi$, and due to the periodic nature of Fourier frequencies, the resulting equivalent mode in the interval $(-\pi, \pi)$ is $q_i + q_j - 2\pi$, which might describe a much lower wavevector, so that a mode that is describing the amplitude of a low length-scale feature will contribute to a mode describing a large length-scale. This artifact, which results from discretizing space, is known as aliasing, and the need to mitigate aliasing has been known in the context of spectral integration for decades [23–25].

There are several methods to avoid these artifacts, all under the common name of de-aliasing [26]. A simple one, which is implemented here, is creating a de-aliased version of the fields that form part of nonlinearities, by setting to 0 the amplitude of their Fourier modes that create aliasing artifacts, and using these de-aliased fields to compute nonlinearities.

More specifically, if a field is part of a nonlinearity of order n, we find the values of q such that $nq > 2\pi/\Delta x - q$, and set those modes to 0. This must also be done for negative frequencies, that is, for every $q < 0$ such that $nq < -2\pi/\Delta x + q$. This condition trivially becomes $|q| > 2\pi/[(n+1)\Delta x]$. For instance, for a quadratic nonlinearity, where $n = 2$, we set every mode of a field to 0 for every frequency q such that $|q| > 2\pi/(3\Delta x)$. This becomes the commonly known two thirds rule for de-aliasing quadratic nonlinearities.

In the rest of the paper, it is assumed that nonlinear terms in any equation are written in terms of de-aliased fields, for instance, Eq. 6 should be rewritten as

$$\partial_t \tilde{\phi}(\mathbf{q}, t) = \mathcal{F}\left[\mathcal{F}^{-1}\left[\tilde{\phi}^{d_3}(\mathbf{q}, t)\right]^3\right], \tag{8}$$

where the field has been de-aliased before being transformed to real space, such that

$$\tilde{\phi}^{d_3}(\mathbf{q}, t) = \begin{cases} \tilde{\phi}(\mathbf{q}, t) & \text{if } |q| < \frac{2\pi}{4\Delta x} \\ 0 & \text{if } |q| > \frac{2\pi}{4\Delta x} \end{cases} \tag{9}$$

1.3 Manuscript structure

This manuscript offers a detailed description of cuPSS. Sections 1 & 2 describe the basic mathematical background for numerically integrating SDEs that cuPSS is built on. The rest of the manuscript, from Section 3 onward, describes how to write models in cuPSS, together with examples and benchmarkings against other similar packages. All the examples in this document can be found in full compilable form in the codebase, in the `examples` directory. Some basic knowledge of C++ is required, and the codebase `README` file also contains detailed installation instructions. Given the CUDA functionality of cuPSS, an appropriate CUDA toolkit must be installed. Likewise the FFTW3 library must be installed to perform Fourier transforms on CPU, and the Google Test suit to run the unit tests. The `README` points to resources to do this.

2 Models that cuPSS can solve

cuPSS integrates in time a system of first-order stochastic PDEs for a set of fields $\{\phi_i(\mathbf{x}, t)\}$ defined on a discrete lattice. Thus, each field follows an equation.

$$\partial_t \phi_i = \mathcal{L}_i \phi_i + \sum_j \mathcal{N}_{ij}[\{\phi_k\}] + \sqrt{2\tilde{D}}\eta_i \tag{10}$$

where $\mathcal{L}_i$ is a linear operator, and $\mathcal{N}_{ij}[\{\phi_k\}]$ is a set of terms, linear or nonlinear on the full set of fields. The nonlinearities are taken to be polynomials of the fields, each of which can be acted upon by a differential operator, so that $\mathcal{N}_{ij}[\{\phi_k\}]$ can be written as

$$\mathcal{N}_{ij}[\{\phi_j\}] = M_{ij} \prod_k \psi_k^{n_k}. \tag{11}$$

where M_{ij} are differential operators, and the fields ψ_k are functions of the fields ϕ_i, also operated upon by arbitrary differential operators. This form of writing the nonlinearities, although cumbersome at first, will facilitate solving these equations pseudo-spectrally.

Let us consider a simple example; a simple equation to write in the form of Eq. 10 is the KPZ equation [5], describing nonlinear growth of interfaces,

$$\partial_t \phi = \sigma \nabla^2 \phi + \lambda (\nabla \phi)^2 + \sqrt{2D}\eta. \tag{12}$$

In this case, there is one linear term, with $mathcal L = \sigma\nabla^2$, and one nonlinearity $\mathcal{N} = \lambda \psi^2$, where $\psi = \nabla\phi$. The noise variance is a simple constant D.

In order to solve Eq. 10 in a pseudo-spectral manner, first it must be written in Fourier space,

$$\partial_t \tilde{\phi}_i = \tilde{\mathcal{L}}_i \tilde{\phi}_i + \sum_j \tilde{\mathcal{N}}_{ij}[\{\phi_k\}] + \tilde{\eta}_i, \tag{13}$$

where

$$\tilde{\phi}_i(\mathbf{q}, t) = (2\pi)^{-d/2} \int d\mathbf{x} e^{-i\mathbf{q}\cdot\mathbf{x}} \phi_i(\mathbf{x}, t), \tag{14}$$

and where the nonlinearities are written as the general form of Eq. 6

$$\tilde{\mathcal{N}}_{ij}[\{\phi_j\}] = \tilde{M}_{ij} \mathcal{F}\left[\prod_k \mathcal{F}^{-1}[\tilde{\psi}_k]^{n_k}.\right] \tag{15}$$

The amplitude of the noise has been absorbed in the definition of $\tilde{\eta}_i$, so that, in Fourier space, the noise has the following variance

$$\langle \eta(\mathbf{q}, t)\eta(\mathbf{q}', t')\rangle = 2\tilde{D}(\mathbf{q})(2\pi)^d \delta^{(d)}(\mathbf{q}+\mathbf{q}')\delta(t-t'). \tag{16}$$

cuPSS will discretize each field ϕ in space, in a lattice with spacing Δx (and $\Delta y, \Delta z$ if the system is 2-dimensional or 3-dimensional). Given all fields at a time t, time is discretized with a timestep size of Δt, and the next timestep is calculated in a simple finite difference scheme. For simplicity and stability, cuPSS uses an implicit form of the Euler-Maruyama method, where the linear terms are treated in a strong implicit way, equivalent to a Taylor method [20]. Usually, one would discretize Eq. 1 in an implicit way by introducing a parameter α such that the discretization in time looks as follows

$$\frac{\phi(t+\Delta t) - \phi(t)}{\Delta t} = \alpha F[\phi(t+\Delta t)] + (1-\alpha)F[\phi(t)] + \frac{\sqrt{2D}}{\Delta t}\Delta W_t, \tag{17}$$

where α measures the amount of “implicitness”, and ΔW_t is a Wiener process with variance Δt. Choosing the Ito representation of the initial stochastic process allows produces a more natural time discretization of the noise term, so that, in practice, ΔW_t is calculated as a set of independent noise terms every timestep [20].

Setting $\alpha = 1$ corresponds to a fully implicit method for the deterministic part of the equation, sometimes called Milstein scheme [20,27,28], while $\alpha = 0$ corresponds to the Euler-Maruyama scheme. The Milstein scheme involves having to solve an algebraic equation, which would give greater numerical stability at the sacrifice of speed. What cuPSS does —which is used in some other widespread packages for the numerical integration of PDEs [1]— is to use a Milstein scheme for the linear terms of each field only, for which the algebraic equation to solve is trivial, and an Euler-Maruyama step for terms that are either linear on different fields, or nonlinear. This offers a way that sacrifices some numerical stability, and might require smaller timestepping, but offers a higher speed of integration. Specifically, a single time update step for a field ϕ_i is

$$\phi_i(t+\Delta t) = \frac{\phi_i(t) + \Delta t \sum_j \mathcal{N}_{ij} + \sqrt{\Delta t}\tilde{\eta}_i}{1 - \Delta t \tilde{\mathcal{L}}_i}. \tag{18}$$

[1]See for example the CFD package OpenFOAM, which allows the user to describe each term in a dynamic equation as explicit or implicit: https://www.openfoam.org

cuPSS will, every timestep, perform the following steps on each field, first on all static fields, and then on all dynamics fields,

1. If the field has a noise term, generate it and apply the appropriate differential prefactor.
2. Set the value of the field according to its equation
3. If the field has a Fourier callback function, apply it.
4. If the field needs dealiasing (i.e. if it is part of a nonlinearity in any equation in the system), dealias it according to the highest order nonlinearity that it is a part of.
5. Convert the field to real space.
6. If the field has a real space callback function, apply it.
7. Convert the field to Fourier space.

cuPSS will perform these steps using a GPU if set up to do so, in which case it will use CUDA and some of its packages to considerably speed up the process. cuFFT is used to perform all Fourier transforms, and cuRAND is used to calculate noise realizations if any of the fields are stochastic. If the solver is set to run on CPU, then fftw3 will be used to calculate Fourier transforms, and the standard C++ random library will be used to calculate noise terms.

3 Writing models

cuPSS offers functions to make it possible to write models in natural language by encoding equations as strings. The steps to coding a model in cuPSS are (i) creating a system, which holds information about the system size, space and time discretization and data output; (ii) defining the fields that make up the system and the equations that describe their time evolution, and the numerical parameters they depend on; (iii) setting an initial condition and callback functions (if any); (iv) evolving the system in time.

Defining a system for a single field and a single equation can be as simple as

```
system.createField("phi", true);
system.addParameter("D", 1.0);
system.addEquation("dt phi + D*q^2*phi = 0");
system.addNoise("phi", "D");
```

where it is hopefully natural enough to read the equation line as a noisy diffusion equation for a field ϕ, with diffusion constant D, i.e. $\partial_t\tilde{\phi} + Dq^2\tilde{\phi} = \sqrt{2D}\tilde{\eta}$. In this case we create a single field ϕ, with the `true` argument indicating that it is a dynamic field (i.e. it is described by an equation with a time derivative), we added a single parameter `D` and a noise term with variance `D`.

Writing equations is done through a parser that follows rules similar to other available numerical solvers; when writing an equation for a dynamic field, named for instance `phi`, the string corresponding to it must look as follows

```
dt phi + lhs = rhs
```

where `lhs` represents terms that are treated implicitly in time (and thus must be linear in `phi`), while `rhs` represents terms that are treated explicitly, as described above. If there are no implicit terms, the string becomes `dt phi = rhs`; and if there are only implicit terms, it becomes `dt phi + lhs = 0`.

Implicit terms must be linear in the field for which we are writing an equation, so each term must be `op * phi`, where `op` can be a combination of numbers or differential operators that form its prefactor.

Explicit terms can be any combination of differential operators and fields. It is important to have in mind that, since the equation is written in Fourier space, differential operators will be applied on the whole term, and not on any of the factors specifically. For instance, if we write `dt phi = -q^2*phi*psi`, where `psi` is some other field, this equation correspondg to $\partial_t \phi = \nabla^2(\phi\psi)$.

If we wanted the Laplacian to act on a single field, $\partial_t \phi = \psi \nabla^2 \phi$, we would need to define an intermediate field for $\nabla^2\phi$, cuPSS will then appropriately de-alias both this field and ψ, and compute their product, we would then need two equations:

```
dt phi = psi*lapphi
lapphi = -q^2*phi
```

Equations for intermediate fields ψ_i are written similarly, except there is no `dt` operator. Again, the left hand side must only have terms linear in the field the equation is describing, and terms in the right hand side can be general products of the rest of the fields. For instance, we can write, for a field `psi`

```
psi + psi * q^2 = -q^2 * phi
```

which is equivalent to a field $\psi[\phi]$, such that $\tilde{\psi}(q,t) + q^2\tilde{\psi}(q,t) = -q^2\tilde{\phi}(q,t)$, or alternatively, $\tilde{\psi}(q,t) = -q^2(1+q^2)^{-1}\tilde{\phi}(q,t)$.

Reserved keywords

The equation parser has certain keywords that must not be used as parameter or field names. These keywords are the following

`dt`. This keyword should be read as ∂_t, and indicates we are writing the equation of motion for a field. It should be the first two characters of a dynamical equation, and the parser will interpret that whatever comes after it it the name of the field we are writing an equation for, thus if the parser is given `dt phi ...`, it will interpret that string as the dynamic equation for a field with the name `phi`.

`q`. This keyword is interpreted as a derivative operator, it must be accompanied by a caret and an even number, and should be interpreted as powers of the laplacian, for example `q^2` should be read as a $-\nabla^2$ operator.

`iqx`, `iqy` and `iqz`. These represent derivatives in the x, y and z directions respectively. For example `dt phi = iqx^2*phi` will be interpreted as the equation $\partial_t \phi = \partial_x^2 \phi$. Notice there must be an asterisk between every pair of factors.

`1/q`. This keyword represents a division over the absolute value of the frequency in Fourier space. It can be useful to write certain nonlocal kernels [6, 8].

3.1 Adding noise

Stochasticity can be added to a field through the function `evolver::addNoise`, which will turn a field's internal boolean `isNoisy` to true and add the noise's appropriate prefactor describing $\tilde{D}$, so that noise correlations are written as in Eq. 16. This prefactor can be a product of a real number with a power of `q^2` and `1/q`. For instance, the noise corresponding to a mass conserving continuity equation, with $\tilde{D} = Dq^2$ would be added to a field `phi` with these two lines

```
system.addParameter("D", 0.01);
system.addNoise("phi", "D*q^2");
```

4 Examples

Since spectral methods usually overperform real space methods when integrating equations with high order gradient terms, I provide two examples for two such models, which can be found in full in the `examples` directory of the codebase. Commonly known in the literature as model B or Cahn-Hilliard model and model H, they describe the dynamics of a conserved density field, and are often used as basic models of phase separation. Model B will showcase the basic implementation of a single field dynamics, while model H will add a layer of complexity by coupling this density field to a Stokes' flow.

4.1 Model B (Cahn-Hilliard)

The Cahn-Hilliard model describes binary mixtures and their transition between uniform and phase separated regimes [3]. A phase field $\phi(r,t)$ represents local relative concentration of the two species of the binary mixture, and follows a continuity equation $\partial_t \phi + \nabla \cdot J = 0$, where the current follows the gradient of a chemical potential $J = -\nabla\mu$. The chemical potential is derived from a free energy $\mathcal{G}[\phi]$, thus $\mu = \delta\mathcal{G}/\delta\phi$, which is taken to be a Landau-Ginzburg expansion on the field ϕ and its gradients.

$$\mathcal{G}[\phi] = \int d^2 r \frac{a}{2}\phi^2 + \frac{b}{4}\phi^4 + \frac{k}{2}(\nabla\phi)^2 \tag{19}$$

The full equation of motion for $\tilde{\phi}$ can thus be written, in this case ignoring stochasticity, as

$$\partial_t \tilde{\phi} + q^2(a + kq^2)\tilde{\phi} = -bq^2 \mathcal{F}\left[(\mathcal{F}^{-1}[\tilde{\phi}])^3\right]. \tag{20}$$

A solver for this equation involves three parameters, a, b and k, and a single field ϕ. The following short code snippet creates such a system, in this case by creating a system object that describes a discretized two-dimensional lattice of size `Nx`×`Ny`, with lattice sites of size `dx`=`dy`, and with a timestep `dt`, that outputs data to disk every `output` timesteps:

```
evolver system(1, Nx, Ny, dx, dy, dt, output);

system.addParameter("a", -1.0);
system.addParameter("b", 1.0);
system.addParameter("k", 4.0);

system.createField("phi", 1);

system.addEquation(
"dt phi + (a*q^2 + k*q^4)*phi = -b*q^2*phi^3");
```

Since the chosen parameters are $-a = b = 1$ and $k = 4$, if the system is initialized with a small noise, it should display an initial instability that leads the system into a spinodal decomposition into two regions with $\phi = \pm 1$, separated by interfaces of thickness $\ell \approx \sqrt{-k/a} = 2$. Fig. 1 shows some snapshots of the output created by the solver above.

4.2 Model H

Model H is a basic model to describe phase separation of binary fluid mixtures [3]. It introduces a fluid velocity v_i, which we will take to obey a Stokes' flow equation; and a density field ϕ, which obeys relaxational dynamics equal to model B, but is also advected by the fluid velocity. The equations can be written as

$$\partial_t \phi + v_i \partial_i \phi = \nabla^2 \mu, \tag{21}$$

$$\eta \nabla^2 v_i - \partial_i P + \partial_j \sigma_{ij} = 0, \tag{22}$$

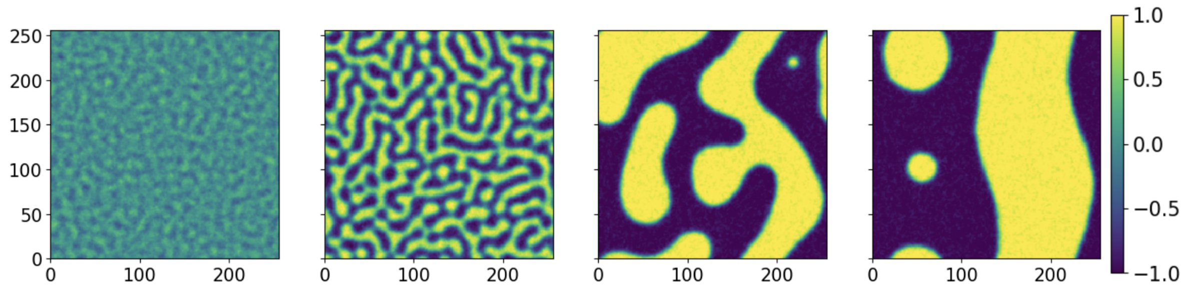

Figure 1: Snapshots at different times of the phase field ϕ obeying Cahn-Hilliard dynamics, as calculated by the example solver of the main text, showing standard spinodal decomposition. The parameters are $-a = b = 1, k = 4$ for a system of size 256×256, a timestep of $\Delta t = 0.1$. The initial condition is a small random perturbation over a uniform state, and the snapshots are taken at times $t =$???.

where the chemical potential is derived from the free energy in Eq. 19, η is the viscosity, P is a pressure term, that we take to be determined by an incompressibility condition ($\partial_i v_i = 0$), and σ_{ij} is a stress tensor. This stress can be derived analytically from the free energy cost of creating density gradients, and gives rise to the interfacial tension [29]; it can be written as $\sigma_{ij} = -k(\partial_i \phi \partial_j \phi - \delta_{ij}(\nabla\phi)^2/d)$.

Equations 21 & 22 (plus incompressibility) can be written in Fourier space in two spatial dimensions as follows

$$\partial_t \tilde{\phi} + q^2(a + kq^2)\phi = -bq^2\mathcal{F}\left[(\mathcal{F}^{-1}[\tilde{\phi}])^3\right] - \mathcal{F}\left[(\mathcal{F}^{-1}[\tilde{v}_i])\mathcal{F}^{-1}[iq_i\tilde{\phi}])\right], \tag{23}$$

$$-\eta q^2 v_i = -iq_i P + iq_j\sigma_{ij}, \tag{24}$$

$$-q^2 P = iq_i iq_j \sigma_{ij}. \tag{25}$$

To translate this into cuPSS fields, we will create the two flow components and the pressure as static fields, as well as separate fields for gradients of ϕ and the two independent components of the stress.

```
system.createField("phi", true);      // density field
system.createField("iqxphi", false);  // x-derivative of phi
system.createField("iqyphi", false);  // y-derivative of phi
system.createField("sigxx", false);   // xx-component of the stress
system.createField("sigxy", false);   // xy-component of the stress
system.createField("vx", false);      // x-component of the flow
system.createField("vy", false);      // y-component of the flow
system.createField("P", false);       // pressure

system.addParameter("a", -1.0);
system.addParameter("b", 1.0);
system.addParameter("k", 4.0);
system.addParameter("eta", 1.0);

system.addEquation("dt phi + q^2*(a + k*q^2)*phi= - b*q^2*phi^3 -vx*
    iqxphi - vy*iqyphi");
system.addEquation("iqxphi = iqx*phi");
system.addEquation("iqyphi = iqy*phi");
system.addEquation("sigxx = - 0.5*k*(iqxphi^2 - iqyphi^2)");
system.addEquation("sigxy = - k*iqxphi*iqyphi");

system.addEquation("-q^2*P = (iqx^2-iqy^2)*sigxx + 2.0 * iqx*iqy*sigxy
    ");

system.addEquation("vx * eta*q^2 = -iqx*P + iqx*sigxx + iqy*sigxy");
system.addEquation("vy * eta*q^2 = -iqy*P + iqx*sigxy - iqy*sigxx");
```

This model shows the importance of writing the right derivative operators in Fourier form by using intermediate fields; for instance, consider the equation for the xy-component of the stress, $\sigma_{xy} = -k(\partial_x\phi)(\partial_y\phi)$. This is coded in cuPSS using two intermediate fields that represent the derivatives of phi, `iqxphi` and `iqyphi`. Using the derivative operators directly would produce a wrong term, specifically, writing `sigxy = -k*iqx*phi*iqy*phi` represents the equation in Fourier space $\tilde{\sigma}_{xy} = -kiq_x\tilde{\phi}iq_j\tilde{\phi}$, which in real space reads $\sigma_{xy} = -k\partial_x\partial_y(\phi^2)$.

5 Callback functions

cuPSS offers direct dynamic access to all fields and terms, so that the user can access and modify the value of all fields during the progress of a simulation. It offers an easy way to do so through callback functions; functions that are called on the Fourier or real components of a field every timestep. There are many applications for callback functions, such as applying boundary conditions. Since a Fourier spectral decomposition assumes periodic boundary conditions, applying other types of boundaries through callback functions should be done with caution. Setting Dirichlet boundary conditions, for instance, which would be properly done spectrally through the choice of a different spectral basis [26], is also usually done by setting a buffer zone of infinite stiffness in the boundaries of the lattice [30,31], i.e. by setting a field to a certain value at a range of positions close to the boundary, making sure this range is bigger than all lengthscales of the problem.

We illustrate the issue of setting boundary conditions this way by solving a heat diffusion equation in one dimension between two heat baths at two different temperatures T_1 and T_2. The temperature $T(x,t)$ obeys $\partial_t T(x,t) = \partial_x^2 T(x,t)$, with the boundary conditions $T(0,t) = T_1$ and $T(L,t) = T_2$. The steady state solution is $T(x,t) = T_1 + (T_2 - T_1)x/L$. To implement this boundary condition we first need to define a callback function which must take five arguments, a pointer to the evolver itself, giving access to the full system, a pointer to the array storing the real space value of the field we are setting a boundary for, and the three dimensional size of the system (the last two parameters will be 1 for a one-dimensional system)

```
#define T1 10.0
#define T2 0.0
void boundary_diffusion(evolver *sys, float2 * array, int Nx, int Ny,
    int Nz) {
     array[0] = T1;
     array[Nx-1] = T2;
}
```

We then assign the boundary condition, by setting the `hasCB` flag on that field to `true`, and assigning a pointer to the callback function to that field's `callback` component

```
system.fieldsMap["phi"]->hasCB = true;
system.fieldsMap["phi"]->callback = boundary_diffusion;
```

Running a diffusion solver with the previous boundary condition would result in a numerical artifact that emerges from creating a high gradient at $x = 0 = L$, created by the periodic nature of the problem, as seen in Fig 2. Alternatively, by creating a frame of width f around the boundaries, and setting value of the Dirichlet boundary condition within this frame, we can effectively remove the artifact. Specifically, if we changed the boudary callback function as follows, where we have fixed $f = 10$,

```
#define T1 10.0
#define T2 0.0
```

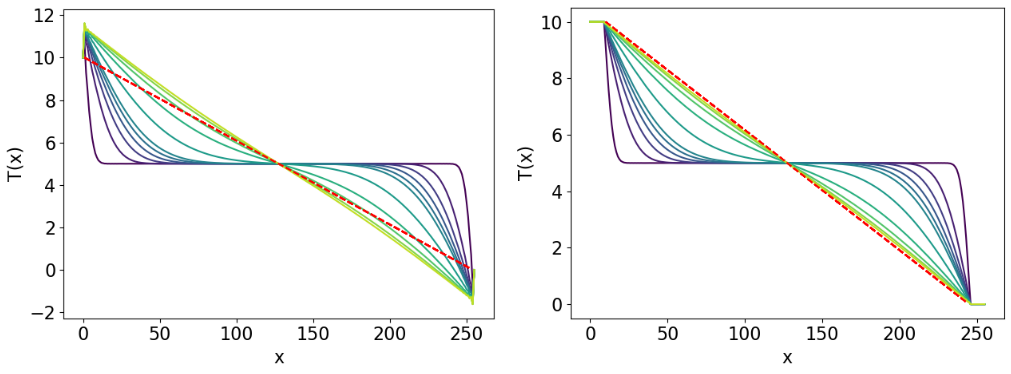

Figure 2: Solution of a diffusion heat diffusion equation ($\partial_t T(x) = \nabla^2 T(x)$) between two heat baths at two different temperatures in discretized space with 256 lattice sites. The dashed red line is the analytical solution, while the other lines represent the numerical solution starting from a constant value $T(x,0) = 5$, and where lighter colors represent later times. The plot on the left shows the solution in which the heat baths are applied by setting $T(0,t) = 10$ and $T(256,t) = 0$ every timestep, showing the numerical artifact that results from setting Dirichlet boundary conditions on an intrinsically periodic field. The plot on the right shows the solution to the same equation, where the heat baths are applied in a frame of width 10 around the boundary, i.e. $T(x < 10,t) = 10$ and $T(x > 246,t) = 0$. This eliminates the numerical artifact, allowing for smoother Fourier representations of the field at any time, and giving a steady state that converges to the analytical solution.

```
void boundary_diffusion(evolver *sys, float2 * array, int Nx, int Ny,
    int Nz) {
     for (int i = 0; i < 10; i++) {
         array[i] = T1;
         array[Nx-1-i] = T2;
     }
}
```

we would obtain a solution that now converges to the analytical solution for the steady state $T(x, t \to \infty)$, which is a straight line between the temperatures of both heat baths, as shown also in Fig 2.

6 Benchmarking

This section shows benchmarking of cuPSS in different systems; a consumer level system with a Intel Core i7-13700K CPU and a NVIDIA RTX 3060 Ti GPU, and a high-performance computing cluster (hpcc), running in several GPUs. For reference, some benchmarks are compared to the same systems integrated in two different packages, FEniCSx [32], a popular finite element solver for PDEs written in weak form, and Dedalus [33], another spectral solver that implements the same techniques as cuPSS, together with extra features such as non-flat spaces and other spectral basis. This section shows the main advantage of cuPSS, which achieves much higher speeds compared to these other packages through the direct implementation of the library on CUDA C++, and having a simplified codebase that relies more on speed than depth and breadth of features.

The benchmarks have been run with the two example solvers of Section 4, for models B and H [3], as these two solvers implement both a simple example with a single field and non-

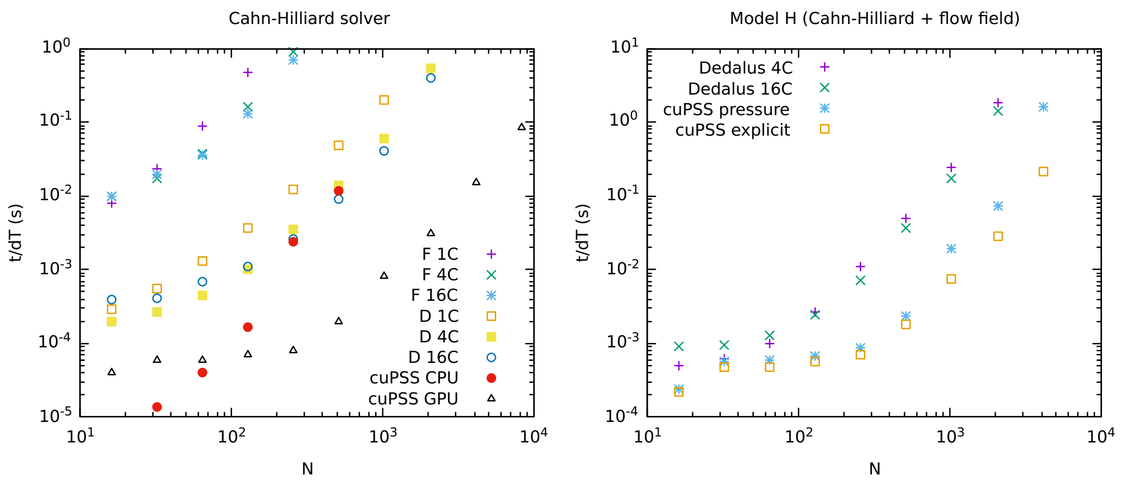

Figure 3: Time t it takes on average to integrate one timestep dT, in seconds, as a function of the size of a 2-dimensional system, N being its side length. On the left, results for model B, and on the right for model H. Each of these runs is done over 10.000 timesteps in a system of size $N \times N$. cuPSS in this case is running on a single core on an i7-13700K, or on a 3060 Ti in the case of the GPU runs. In the legend, F and D stand for FEniCSx and Dedalus respectively, and the number is the number of cores they run on, on the same system.

linear term, and a more complex example with several fields and constraints. The benchmark results are given in terms of average time taken to integrate one timestep, calculated as an average over a number of timesteps that ranges from 100 to 10000 depending on the problem and system size. If there is any overhead in any of these solvers, associated to preparing the problem, reserving memory and so on, this time has not been taken into account in the time measurements. This overhead was negligible compared to the integration time in all runs.

Figure 3 shows the time per timestep to integrate both models and Figure 4 shows again Cahn-Hilliard dynamics benchmarks in six different GPUs, from consumer to enterprise level.

There is one series of benchmarking done on CPU, while all others run on GPUs. CPUs are only more efficient when integrating small systems, in this case when the system size is less than 128×128. Benchmarks have also been run only on system sizes that are powers of 2. These system sizes result in the best speeds given the nature of both the way CUDA launches kernels on GPUs, and the way FFT algorithms work. This means running on system sizes different than powers of 2 might show a loss of speed compared to the system size. In the worst case scenario, a system size different than 2^n might run at the same speed as a system size that is equal to the next power of 2, specially when running on GPUs.

The main result of this section is that cuPSS offers a very significant improvement in speed, of up to a few orders of magnitude, specially in bigger lattice sizes, when compared to other popular finite-difference solvers. This is thanks to several factors, such as relying on GPUs and sacrificing some breadth of application by restricting systems to Fourier representations, as well as keeping a small codebase that does not introduce unnecessary features for simple applications.

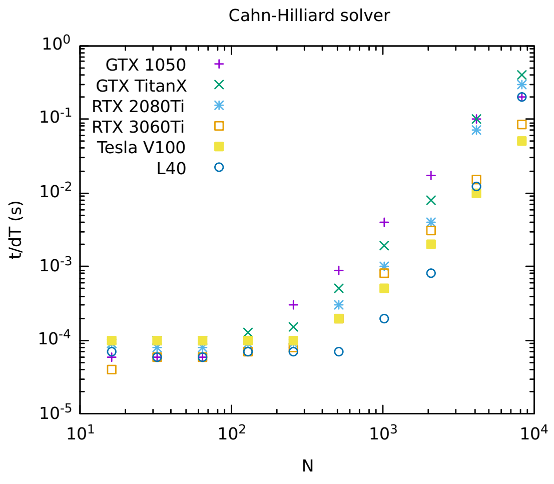

Figure 4: Comparison of speed in several different graphics cards for a Cahn-Hilliard solver on cuPSS. There is no appreciable difference in speed until the system size is large enough, at which point the faster graphics cards are able to process the bigger textures associated to each field in less time.

7 Future development

cuPSS is offered as open source with several examples, and its source code can be found on github [2] under an MIT license, and is thus open to other people to further develop or contribute to. Additionally, cuPSS has a relatively small codebase, making it in principle easy for developers to change and modify to meet their needs. This small codebase is at the same time one of its biggest strengths when compared to other similar packages, making the code very accessible to new users. There is ample space for cuPSS to grow and incorporate new features, such as higher dimensionality spaces, curved spaces, and so on, without much sacrifice to its speed.

In conclusion, cuPSS offers a quick way to prototype and test continuum theories for a variety of different physical systems. It eliminates the need to know details about particular algorithmic implementations of numerical methods, and provides a simple way to write theories in a quasi-natural language, without deep knowledge of either C++ or CUDA.

With continuous development and optimization, cuPSS has the potential to become a useful tool for researchers in very different areas, as it is already being used, specially in its purely deterministic form, some of which has already been published [30, 34–36]. Written in the CUDA C++ standard, it should be accessible to most research groups to, not only use, but easily modify and adapt to their needs.

Acknowledgements

I thank Paarth Gulati for help testing during the early parts of the development of cuPSS, and for suggesting its name, and Daniel Hellstein for guidance in the development of some of the solver components. This work was done during stays at the University of California, Santa Barbara and Brandeis University, and was partially funded by NSF grants DMR-2041459 & DMR-2011846. This work was also supported by the NSF through DMR 2309635 and the Brandeis Center for Bioinspired Soft Materials, an NSF MRSEC (DMR-2011846). We also

[2]https://github.com/fcaballerop/cuPSS

acknowledge computational support from NSF XSEDE computing resources allocation TG-MCB090163, National Energy Research Scientific Computing Center (NERSC), a Department of Energy Office of Science User Facility using NERSC award BES-ERCAP0026774, and the Brandeis HPCC which is partially supported by the NSF through DMR-MRSEC 2011846 and OAC-1920147.

References

[1] D. Forster, D. R. Nelson and M. J. Stephen, *Large-distance and long-time properties of a randomly stirred fluid*, Physical Review A **16**(2), 732 (1977), doi:10.1103/PhysRevA.16.732.

[2] S.-k. Ma and G. F. Mazenko, *Critical dynamics of ferromagnets in 6-ε dimensions: General discussion and detailed calculation*, Physical Review B **11**(11), 4077 (1975), doi:10.1103/PhysRevB.11.4077.

[3] P. C. Hohenberg and B. I. Halperin, *Theory of dynamic critical phenomena*, Reviews of Modern Physics **49**(3), 435 (1977), doi:10.1103/RevModPhys.49.435.

[4] E. Tjhung, C. Nardini and M. E. Cates, *Cluster phases and bubbly phase separation in active fluids: reversal of the ostwald process*, Physical Review X **8**(3), 031080 (2018), doi:10.1103/PhysRevX.8.031080.

[5] M. Kardar, G. Parisi and Y.-C. Zhang, *Dynamic scaling of growing interfaces*, Physical Review Letters **56**(9), 889 (1986), doi:10.1103/PhysRevLett.56.889.

[6] A. J. Bray, A. Cavagna and R. D. Travasso, *Interface fluctuations, Burgers equations, and coarsening under shear*, Physical Review E **65**(1), 016104 (2001), doi:10.1103/PhysRevE.65.016104.

[7] F. Caballero, C. Nardini, F. van Wijland and M. E. Cates, *Strong coupling in conserved surface roughening: a new universality class?*, Physical review letters **121**(2), 020601 (2018), doi:10.1103/PhysRevLett.121.020601.

[8] M. Besse, G. Fausti, M. E. Cates, B. Delamotte and C. Nardini, *Interface roughening in nonequilibrium phase-separated systems*, Physical Review Letters **130**(18), 187102 (2023), doi:10.1103/PhysRevLett.130.187102.

[9] R. Wittkowski, A. Tiribocchi, J. Stenhammar, R. J. Allen, D. Marenduzzo and M. E. Cates, *Scalar φ 4 field theory for active-particle phase separation*, Nature communications **5**(1), 4351 (2014), doi:10.1038/ncomms5351.

[10] A. Tiribocchi, R. Wittkowski, D. Marenduzzo and M. E. Cates, *Active model h: scalar active matter in a momentum-conserving fluid*, Physical review letters **115**(18), 188302 (2015), doi:10.1103/PhysRevLett.115.188302.

[11] F. Caballero, C. Nardini and M. E. Cates, *From bulk to microphase separation in scalar active matter: a perturbative renormalization group analysis*, Journal of Statistical Mechanics: Theory and Experiment **2018**(12), 123208 (2018), doi:10.1088/1742-5468/aaf321.

[12] F. Caballero and M. C. Marchetti, *Activity-suppressed phase separation*, Physical Review Letters **129**(26), 268002 (2022), doi:10.1103/PhysRevLett.129.268002.

[13] H. Chaté and A. Solon, *Dynamic scaling of two-dimensional polar flocks*, arXiv preprint arXiv:2403.03804 (2024), doi:10.1103/PhysRevLett.132.268302.

[14] M. Besse, H. Chaté and A. Solon, *Metastability of constant-density flocks*, Physical Review Letters **129**(26), 268003 (2022), doi:10.1103/PhysRevLett.129.268003.

[15] R. Lande, S. Engen and B.-E. Saether, *Stochastic population dynamics in ecology and conservation*, Oxford University Press, USA, doi:10.1093/acprof:oso/9780198525257.001.0001 (2003).

[16] D. J. Wilkinson, *Stochastic modelling for systems biology*, Chapman and Hall/CRC, doi:10.1201/9781351000918 (2018).

[17] I. Mastromatteo, B. Toth and J.-P. Bouchaud, *Anomalous impact in reaction-diffusion financial models*, Physical Review Letters **113**(26), 268701 (2014), doi:10.1103/PhysRevLett.113.268701.

[18] B. Tóth, Y. Lemperiere, C. Deremble, J. De Lataillade, J. Kockelkoren and J.-P. Bouchaud, *Anomalous price impact and the critical nature of liquidity in financial markets*, Physical Review X **1**(2), 021006 (2011), doi:10.1103/PhysRevX.1.021006.

[19] J.-P. Bouchaud and M. Potters, *Theory of Financial Risk and Derivative Pricing: From Statistical Physics to Risk Management*, Cambridge University Press, 2 edn., doi:10.1017/CBO9780511753893 (2003).

[20] P. E. Kloeden and E. Platen, *Numerical Solution of Stochastic Differential Equations*, Springer Berlin, Heidelberg, doi:10.1007/978-3-662-12616-5 (1992).

[21] B. Fornberg, *A practical guide to pseudospectral methods*, Cambridge University Press, doi:10.1017/CBO9780511626357 (1998).

[22] M. Frigo and S. G. Johnson, *The design and implementation of FFTW3*, Proceedings of the IEEE **93**(2), 216 (2005), doi:10.1109/JPROC.2004.840301, Special issue on “Program Generation, Optimization, and Platform Adaptation”.

[23] S. A. Orszag, *On the elimination of aliasing in finite-difference schemes by filtering high-wavenumber components*, Journal of Atmospheric Sciences **28**(6), 1074 (1971), doi:10.1175/1520-0469(1971)028<1074:OTEOAI>2.0.CO;2.

[24] S. A. Orszag, *Comparison of pseudospectral and spectral approximation*, Studies in Applied Mathematics **51**(3), 253 (1972), doi:10.1002/sapm1972513253.

[25] G. Patterson and S. A. Orszag, *Spectral calculations of isotropic turbulence: Efficient removal of aliasing interactions*, Physics of Fluids **14**(11), 2538 (1971), doi:10.1063/1.1693365.

[26] J. P. Boyd, *Chebyshev and Fourier spectral methods*, Courier Corporation (2001).

[27] R. Mannella, *Numerical integration of stochastic differential equations*, Proc. Euroconf. on Supercomputation in Nonlinear and Disordered Systems pp. 100–30 (1997), doi:10.48550/arXiv.cond-mat/9709326.

[28] R. Mannella, *A gentle introduction to the integration of stochastic differential equations*, In *Stochastic Processes in Physics, Chemistry, and Biology*, pp. 353–364. Springer Berlin Heidelberg, ISBN 978-3-540-45396-3, doi:10.1007/3-540-45396-2_32 (2000).

[29] M. E. Cates and E. Tjhung, *Theories of binary fluid mixtures: from phase-separation kinetics to active emulsions*, Journal of Fluid Mechanics **836**, P1 (2018), doi:10.1017/jfm.2017.832.

[30] L. Zhao, P. Gulati, F. Caballero, I. Kolvin, R. Adkins, M. C. Marchetti and Z. Dogic, *Asymmetric fluctuations and self-folding of active interfaces*, Proceedings of the National Academy of Sciences **121**(51), e2410345121 (2024), doi:10.1073/pnas.2410345121.

[31] L. Risthaus and M. Schneider, *Imposing dirichlet boundary conditions directly for fft-based computational micromechanics*, Computational Mechanics pp. 1–25 (2024), doi:10.1007/s00466-024-02469-1.

[32] I. A. Barrata, J. P. Dean, J. S. Dokken, M. Habera, J. Hale, C. Richardson, M. E. Rognes, M. W. Scroggs, N. Sime and G. N. Wells, *Dolfinx: The next generation fenics problem solving environment* (2023), doi:10.5281/zenodo.10447666.

[33] K. J. Burns, G. M. Vasil, J. S. Oishi, D. Lecoanet and B. P. Brown, *Dedalus: A flexible framework for numerical simulations with spectral methods*, Physical Review Research **2**(2), 023068 (2020), doi:10.1103/PhysRevResearch.2.023068.

[34] F. Caballero, Z. You and M. C. Marchetti, *Vorticity phase separation and defect lattices in the isotropic phase of active liquid crystals*, Soft Matter **19**(40), 7828 (2023), doi:10.1039/D3SM00744H.

[35] P. Gulati, F. Caballero, I. Kolvin, Z. You and M. C. Marchetti, *Traveling waves at the surface of active liquid crystals*, Soft Matter **20**, 7703 (2024), doi:10.1039/D4SM00822G.

[36] P. Gulati, F. Caballero and M. C. Marchetti, *Active fluids form system-spanning filamentary networks*, Physical Review Letters p. Accepted (2025).