Galerkin spectral method with Haskell

Oleksandr Koshkarov
koshmipt@gmail.com

Abstract

This is a short description of Galerkin spectral method which I am using to solve a system of partial differential equations with Haskell+Repa.

1 Notes

This section could be skipped.

1.1 What is in this text?

The system of equations and the method I am using to solve them.

1.2 What is not in this text?

There is no any Haskell or Physics in this text.

1.3 Sorry

This text is a very raw draft, so please be ready for mistakes, typos, inconsistency and other ugliness.

1.4 What is the purpose of this text?

I am learning Haskell, and I have encountered a problem — I am stuck. All my Haskell experience was about solving toy problems and reading different books/tutorials. As for now, I feel comfortable solving those toy problems, but I feel that I cannot make anything significant because I am unfamiliar with most of Haskell ecosystem. Thus, I decided I should start writing Haskell code no matter what. To do so I picked a research project, which has low time priority for me and wrote my first Haskell program. Right now I need a feedback what to do next. So I decided to write this text about mathematical part of my "first Haskell program" to lure curious and experienced Haskell programmers to look at my code.

1.5 Should I have used Haskell?

It seems that I need a more advance time stepper then 4th order Runge Kutta which I use now. Unfortunately, Haskell does not provide one for Repa to my knowladge. So my options are

- Write my own. I am not seriously considering this as writing advanced time stepper is very difficult problem.
- Use different languages with nice numerical infrastructure. I can probably avoid my usual choice C/Fortran by learning something like Julia or Ocaml which both have SUNDIALS bindings, but I want to give Haskell one more chance.
- Find another Haskell library. Probably that is what I am going to do. The most appealing library seems to be petsc-hs by Marco Zocca (ocramz).

1.6 Parallelization

The code is written with Repa so one can think it makes it parallel. Unfortunately, the most time consuming part of my code is repa's FFT which is not parallel, thus if one will use multiple cores, the execution time will increase. Probably, I need to rewrite it to compute convolution explicitly.

1.7 About me

I am a PhD student in the University of Saskatchewan, Canada. My major is theoretical plasma physics.

2 Problem formulation

I am trying to solve the system of equations

$$\partial_t n = -v_0 \partial_x n + \theta + n\theta + \nabla n \cdot \nabla \chi \tag{1a}$$
$$\partial_t \theta = -v_0 \partial_x \theta + \eta - n + 0.5 \nabla^2 (\nabla \chi)^2 \tag{1b}$$
$$\partial_t \eta = -u_0 \partial_y \theta - \nu(\eta - n) - \rho_s \partial_y \phi + \{\phi, \eta\} \tag{1c}$$
$$\mu \nabla^2 \phi = \eta - n \tag{1d}$$
$$\nabla^2 \chi = \theta \tag{1e}$$
$$\{\phi, \eta\} = \partial_x \phi \partial_y \eta - \partial_y \phi \partial_x \eta \tag{1f}$$

where $n, \theta, \eta, \phi, \chi$ are two + one dimensional functions ($f = f(t, x, y)$), and $\nu, \mu, v_0, u_0, \rho_s$ are constants.
Considered spacial domain is

$$x \in [0, L_x] \tag{2}$$
$$y \in [0, L_y] \tag{3}$$

Initial conditions are some fixed functions

$$n(0,x,y) = n_0(x,y) \tag{4}$$
$$\theta(0,x,y) = \theta_0(x,y) \tag{5}$$
$$\eta(0,x,y) = \eta_0(x,y) \tag{6}$$
$$\phi(0,x,y) = \phi_0(x,y) \tag{7}$$
$$\chi(0,x,y) = \chi_0(x,y) \tag{8}$$

and boundary conditions are periodic

$$n(t,0,y) = n(t,L_x,y) \tag{9}$$
$$n(t,x,0) = n(t,x,L_y) \tag{10}$$
$$\text{Same for } \theta,\eta,\phi,\chi \tag{11}$$

3 Spectral method based on Fourier expansion

To solve out system (1), we approximate our functions

$$n(t,x,y) \approx \sum_{k_x=-N_{k_x}}^{N_{k_x}} \sum_{k_y=-N_{k_y}}^{N_{k_y}} \hat{n}_{k_x,k_y}(t) \exp\left(2\pi i \left(\frac{k_x x}{L_x} + \frac{k_y y}{L_y}\right)\right) \tag{12}$$

same for θ, η, χ, ϕ.

Comment here we used Fourier expansion. One can use different set of expansion functions, thus using "different" type of spectral method.

Now we can plug (12) into (1), then multiply by $\exp\left(-2\pi i \left(\frac{k_x x}{L_x} + \frac{k_y y}{L_y}\right)\right)$ and then integrate over the period $\int_0^{L_x}\int_0^{L_y}(***)dxdy$. This will give us the equation for $\hat{n}, \hat{\theta}\ldots$

Comment here we used Fourier functions again, to "project equation", but we again could have used different functions.

I could do what I said, with integration. Then nonlinear terms like $n\theta$ would be the convolution of two matrices $\hat{n}$ and $\hat{\theta}$. The usual way to compute convolution is via fast Fourier transform (FFT), thus I will make a little bit different derivation which is simpler but a little bit more ugly.

Comment if we would derive with integration (for continuous functions), we would need convolution theorem to transform FFT.

Let say we know our functions only on the mesh

$$n(m\Delta x, l\Delta y) = n_{m,l} \tag{13}$$

where $m = 0,1,\ldots,N_x-1$ and $l = 0,1,\ldots,N_y-1$, $N_x = 2N_{k_x}+1$, $N_y = 2N_{k_y}+1$, $\Delta x = L_x/N_x$, $\Delta y = L_y/N_y$. Here and later if I state something general for one function (n) it is true for other functions ($\theta, \eta, \ldots$). I also do not include the last point

because it would be the same as first point due to periodic boundary conditions.
Now we can rewrite equation (12) for discrete quantities

$$n_{m,l} = \mathcal{F}^{-1}\left[\hat{n}_{k_x,k_y}\right]_{m,l} = \frac{1}{N_x N_y}\sum_{k_x=-N_{k_x}}^{N_{k_x}}\sum_{k_y=-N_{k_y}}^{N_{k_y}} \hat{n}_{k_x,k_y}(t)\exp\left(2\pi i\left(\frac{k_x m}{N_x}+\frac{k_y l}{N_y}\right)\right) \tag{14}$$

here I constructed expansion a little bit differently — I multiplied it by $1/(N_x N_y)$. I also notice that this expansion is actually two dimensional reverse FFT. So it has the property

$$\mathcal{F}\circ\mathcal{F}^{-1} = \mathcal{F}^{-1}\circ\mathcal{F} = \mathbf{1} \tag{15}$$

where $\circ$ is function composition, $\mathbf{1}$ is identity operator and

$$\mathcal{F}\left[n_{m,l}\right]_{k_x,k_y} = \sum_{m=0}^{N_x-1}\sum_{l=0}^{N_y-1} n_{m,l}\exp\left(-2\pi i\left(\frac{k_x m}{N_x}+\frac{k_y l}{N_y}\right)\right) \tag{16}$$

Ok, so here is the trick, we cannot plug our discrete transformation into our main system (1), so we plug the transformation (12), differentiate the exponent and only them make it discrete. Then we apply FFT one more time and use the (15) property. For linear terms it is simple, we just substitute

$$\partial_x \to iK_x \tag{17}$$
$$\partial_y \to iK_y \tag{18}$$
$$\tag{19}$$

where $K_\alpha = 2\pi k_\alpha / L_\alpha$ (for $\alpha = x, y$). For nonlinear terms, well, we do nothing. So we have

$$\partial_t\hat{n} = -v_0 iK_x\hat{n} + \hat{\theta} + NL_1 \tag{20a}$$
$$\partial_t\hat{\theta} = -v_0 iK_x\hat{\theta} + \hat{\eta} - \hat{n} + NL_2 \tag{20b}$$
$$\partial_t\hat{\eta} = -u_0 iK_y\hat{\theta} - \nu(\hat{\eta}-\hat{n}) - \rho_s iK_y\hat{\phi} + NL_3 \tag{20c}$$
$$-K^2\mu\phi = \eta - n \tag{20d}$$
$$-K^2\chi = \theta \tag{20e}$$
$$K^2 = K_x^2 + K_y^2 \tag{20f}$$
$$NL_1 = \mathcal{F}\left[\mathcal{F}^{-1}\left[\hat{n}\right]\mathcal{F}^{-1}\left[\hat{\theta}\right] + \mathcal{F}^{-1}\left[iK_x\hat{n}\right]\mathcal{F}^{-1}\left[iK_x\hat{\chi}\right] + \mathcal{F}^{-1}\left[iK_y\hat{n}\right]\mathcal{F}^{-1}\left[iK_y\hat{\chi}\right]\right] \tag{20g}$$
$$NL_2 = \text{similar to \#1} \tag{20h}$$
$$NL_3 = \text{similar to \#1} \tag{20i}$$

So now I just solve the system (20) in time with a time stepper algorithm.

4 Time integration

For time being I am using Runge-Kutta 4. But I need something better. Currently I am considering two options: move to different language, use petsc-hs by Marco Zocca (ocramz).

5 What else?

The report is not finished. But hopefully I will improve it, as I go. So what I have not covered

- deallising
- hyperviscosity
- comparison with theory for linear regime